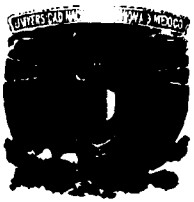


UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS
PROFESIONALES "ACATLÁN"

" LENGUAJE ENSAMBLADOR, HERRAMIENTA PRÁCTICA PARA LOS PROCESOS INDUSTRIALES "

OPCIÓN TESINA

PARA OBTENER EL TÍTULO DE

LIC. MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

ELOINA RODRÍGUEZ GONZÁLEZ

Asesor: LIC. OSCAR G. CABALLERO MARTÍNEZ

Fecha: Julio / 2003

" LENGUAJE ENSAMBLADOR, HERRAMIENTA PRÁCTICA PARA LOS PROCESOS INDUSTRIALES "

OBJETIVO DEL TRABAJO.

"Proporcionar información acerca del lenguaje ensamblador para Motorola, enfatizando la utilización del software para aplicaciones en los procesos industriales, además de mostrar que para actualizar un sistema industrial, solo basta con realizar modificaciones en el software, repercutiendo en beneficios de simplicidad y costo."

ÍNDICE

CAPITULO III. "LENGUAJE ENSAMBLADOR PARA INTEL"

CAPITULO IV. "APLICACIONES EN PROCESOS INDUSTRIALES"

INTRODUCCIÓN

A lo largo de siglo XX, la tecnología ha ido avanzando a pasos agigantados, y la programación pero sobre todo el desarrollo de sistemas basados en la computación permiten gran factibilidad técnica y financiera.

Sus inicios fueron en 1971 cuando Hoof y Fagin trabajaban 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 el man-power disponible (solo 4 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 operaciones, exhibir resultados, etc. con solo modificar su programa.

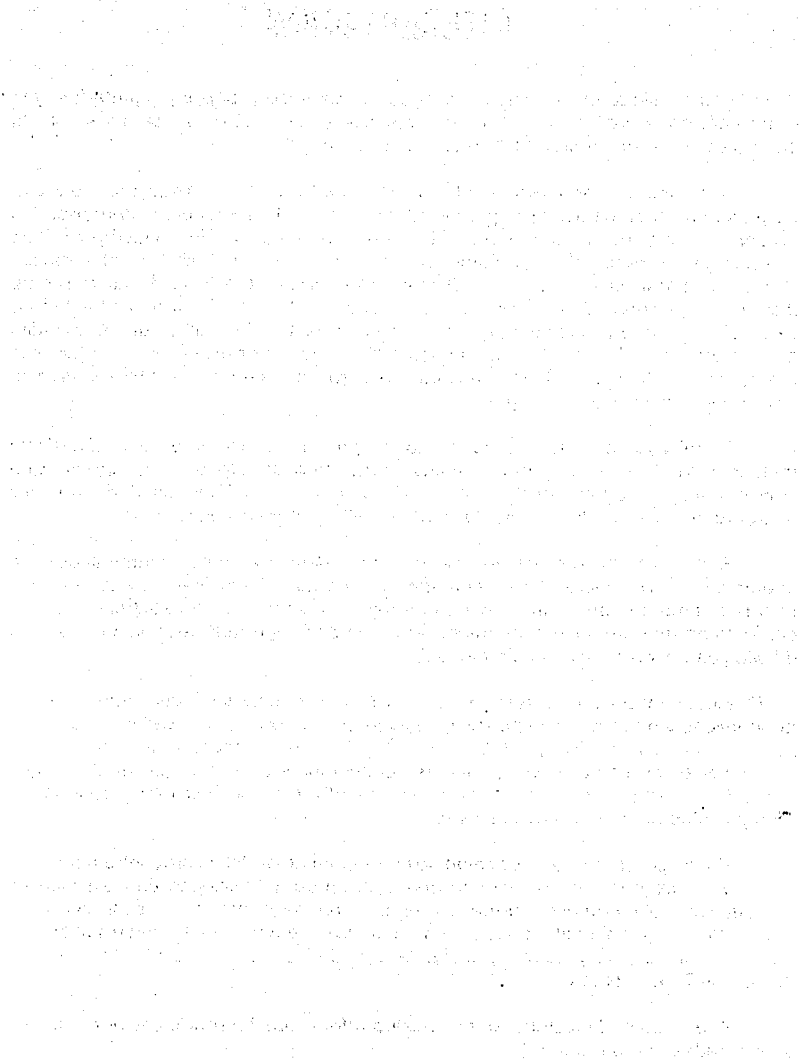
El microprocesador surge como el primer circuito integrado altamente complejo, totalmente estándar y relativamente fácil de utilizar. La combinación complejidad – programabilidad se ha extendido a dos famosos derivados del microprocesador : el Procesador Digital de Señal y el Microcontrolador.

El microcontrolador es uno de los componentes actuales económicos y de mayor campo de aplicación. Teniendo como ventaja principal su manipulación casi básicamente con la programación y algunos componentes electrónicos, y no como anteriormente se implementaba solo con componentes electrónicos que son difíciles de encontrar y de costo elevado.

El microcontrolador es uno de los componente actuales más "entretenidos", económicos y de mayor campo de aplicación. Por un precio cercano a $250.00, se puede conseguir un *chip* que integra una CPU, varios temporizadores, puertos de E/S, bloques de comunicación e incluso conversores analógico-digital. En lo que respecta al *software* de programación, cuenta con instrucciones sencillas y manipulables en diversas aplicaciones.

En el proyecto se presentan los componentes del microcontrolador y la manera de programarlo, incluyendo una sección de aplicaciones en los procesos industriales. El proyecto fuerte es el de una termómetro, el cual censa la temperatura (ambiental o de un líquido), envía esta información al microcontrolador, se procesa y emite la cantidad en grados centígrados en un display de 7 segmentos.

A lo largo de la lectura, se encontrara referencias bibliográficas por capítulo, simbolizadas como [número].



CAPÍTULO I.

"CARACTERÍSTICAS DE HARDWARE DEL MICROCONTROLADOR 68HC11 DE MOTOROLA".

1.1. ¿Qué es un microprocesador?

Un microprocesador es un circuito integrado capaz de realizar operaciones aritméticas, lógicas y de control.

Para funcionar requiere de otros dispositivos tales como memorias, puertos de entrada / salida y otros. Se comunica con éstos a través de buses y hay tres tipos de buses: datos, direcciones y control.[1]

1.2. ¿Qué es un microcontrolador?

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

Se puede decir que es una evolución del microprocesador, al añadirle a este último las funciones que antes era necesario situar externamente con otros circuitos. El ejemplo típico esta 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 porque lo lleva todo incorporado, además en el caso de tener que ampliar el sistema ya ofrece recursos que facilitan esto. En resumen, un microcontrolador es un circuito integrado independiente, que no necesita memoria ni puertos externos pues los lleva en su interior, que facilita la tarea de diseño y reduce el espacio, traduciéndose todo a una aplicación final más económica y fiable.

El microcontrolador 68HC11 destaca por sus recursos, simplicidad y facilidad de manejo. Motorola describe al 68HC11como un microcontrolador de 8 bits fabricado con tecnología HCMOS, con una frecuencia de bus de 2 Mhz y con una amplia lista de recursos internos.

Los recursos internos disponibles son:

- 512 bytes de memoria RAM.
- 12 Kbytes de memoria EPROM.
- 512 bytes de memoria EEPROM.
- 5 puertos de 8 bits, con pines de entrada, salida y de entrada/salida.
- Conversor analógico-digital de 8 canales y 8 bits de resolución.
- Una UART para comunicaciones serie asíncronas (SCI).
- Un módulo de comunicaciones serie síncronas (SPI).
- 5 comparadores con salida hardware.
- 3 capturadores de entrada.
- Un acumulador de pulsos externos de 8 bits.
- Temporizador principal de 16 bits.
- Interrupciones en tiempo real.
- 2 entradas de interrupciones externas.
- Software en ROM para cargar un programa externo en la RAM interna.

Diagrama de bloques del 68HC11

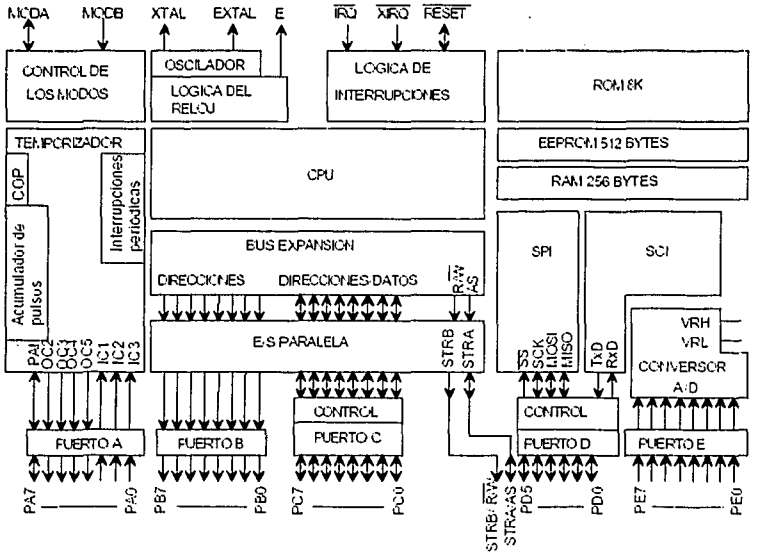


Figura 1. Diagrama a bloques del 68HC11

Muchos de los recursos no son accesibles simultáneamente. Por ejemplo, si se quiere utilizar memoria externa los puertos B y C se deben utilizar como bus de datos y direcciones. Por ello el 68hc11 se suele utilizar sin memoria externa, en caso de necesitar mayor memoria se suele recurrir a modelos de la familia que incorporan un mayor tamaño. Existe la tarjeta CT6811 para realizar pruebas y lograr la comunicación del microcontrolador con la PC para programarse.[2]

1.3. La estructura de hardware que contiene el microcontrolador 68HC11 de Motorola.

El MC68HC11 (MCU), dispone de 98 funciones de entrada y/o salida, las cuales se ven representadas por 52 pines con un encapsulado del tipo PLCC.

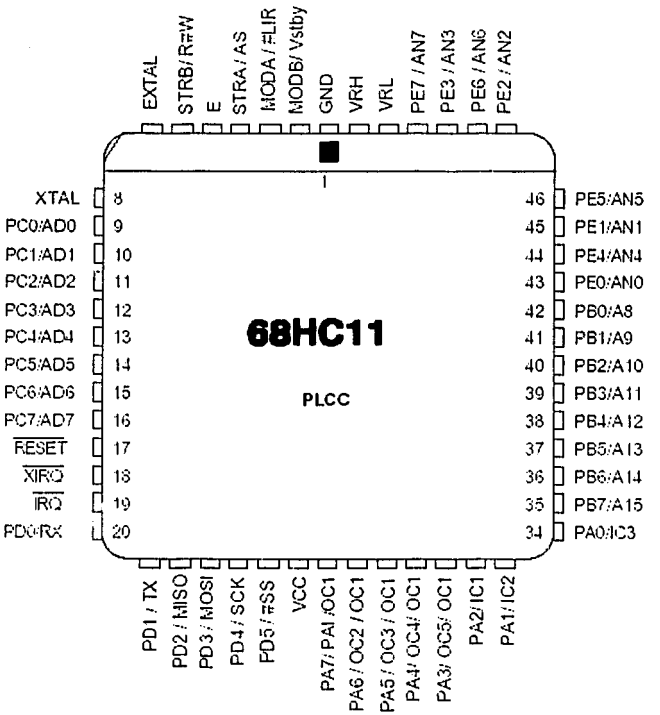


Figura 2. Pines del encapsulado PLCC para el 68HC11

Cuando se monta un sistema digital basado en microcontrolador, existe siempre el peligro de que un mal diseño provoque no solo un mal funcionamiento sino un daño irreparable de los circuitos. Esto se debe a que a diferencia de la lógica digital habitual, los microcontroladores, en general, trabajan con varios tipos de señales, conversores A/D, salidas PWM, líneas de transmisión, etcétera, lo que provoca que una mala conexión pueda tener graves consecuencias.

Un mecanismo de protección frente a este tipo de conflictos es el adoptado por el 68HC11 donde varios de sus pines se encuentran dotados de circuitos internos de protección. Igualmente este tipo de soluciones tienen sus propias limitaciones por lo que nunca se debe bajar la guardia. Para saber más sobre estas protecciones acudir al manual de Referencia Técnica de Motorola.

Para facilitar la comprensión se clasifican todos los pines del microcontrolador en grupos de acuerdo a las funciones de los mismos, siendo estas agrupaciones las siguientes:

1. Alimentación: VDD, VSS.

2. Reloj: EXTAL, XTAL, E.

3. Reset: RESET.

4. Transmisión serie asíncrona: TxD, RxD.

5. Petición de interrupciones hardware: IRQ, XIRQ, IC1-3, PAI, STRA.

6. Modos de arranque: MODA, MODB.

7. Comparadores: OC1-5.

8. Capturadores: IC1-3, PAI.

9. Transmisión serie síncrona: SCK, MISO, MOSI, SS.

10. Puertos: PA0-7, PB0-7, PC0-7, PD0-3, PE0-3.

11. Conversores: AN0-7.

12. Buses: AD0-7, A8-15, AS, R/W.

Con esta clasificación, se intenta dar una vista general de todos los subsistemas de hardware que conforman el microcontrolador y que tienen salida directa al exterior a través del encapsulado. [3]

1.4. Diseño de la tarjeta entrenadora CT6811 para desarrollar las aplicaciones con el microcontrolador 68HC11 de Motorola.

Para armar la tarjeta entrenadora CT6811 se presentan a continuación los subsistemas de la misma, así como los componentes requeridos para su buen funcionamiento y protección del microcontrolador.

1.4.1. Pines de alimentación.

·**VDD**: Es el pin de alimentación positiva, la cual debe ser el valor estándar de cinco voltios (el margen aceptado es de 4.5 a 5.5 voltios)

·**VSS**: Es la masa del MCU.

Para asegurar una buena robustez contra el ruido, es conveniente la conexión de un par de condensadores en paralelo entre VDD y VSS, con el fin de que estos anulen los posibles rizados provenientes de la fuente de alimentación y provocados por las conmutaciones internas del microcontrolador. Dichos condensadores deben estar físicamente lo más cerca posible, el fabricante para dichos componentes son de 1 y 0.01 microfaradios.

1.4.2. Pines de reset.

·**RESET.** Está señal, activa a nivel bajo, es bidireccional. El 68HC11 está preparado no sólo para recibir señales de "reset" por este pin sino que es el propio dispositivo el que es capaz de generar dicha señal para todos los periféricos que conformen el sistema digital. De esta manera es posible que el diseñador trate a los subsistemas internos del microcontrolador como un conversor A/D, del mismo modo que a cualquier dispositivo externo como un display, ya que al recibir la señal de "reset, el MCU la transmitirá internamente al A/D, y generará la misma señal para el display.

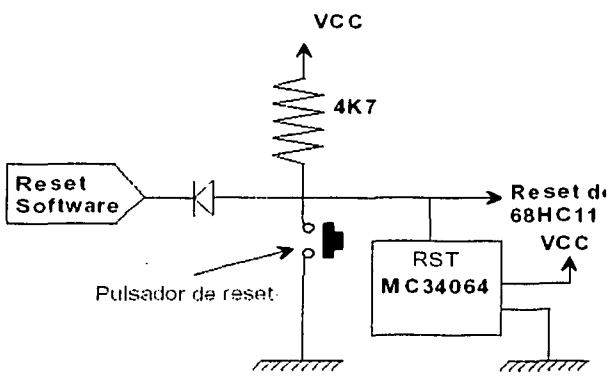


Figura 3. Circuito de reset empleado en la tarjeta CT6811

Por último cabe destacar que este pin es de colector abierto y para deshabilitarlo (que no sea un "reset" constante) debe tener una resistencia de pull-up de 4K7Ω.

1.4.3. Pines de transmisión serie asíncrona

·**TxD** y **RxD**: Son los pines de transmisión y recepción de datos serie asíncronos.

El 68HC11 trabaja con niveles TTL, por lo que será necesario un circuito de adaptación si se quiere comunicar con un dispositivo que utilice otro tipo de norma como por ejemplo un PC (norma RS-232C). Un integrado adaptación es el MAX232.

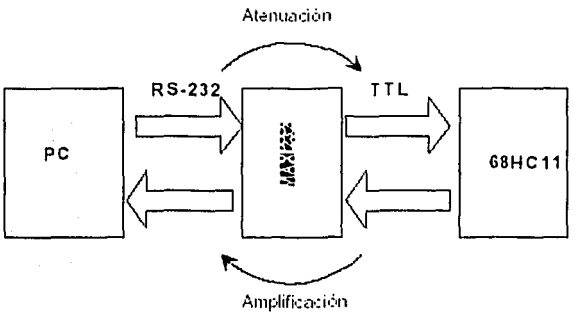


Figura 4. Funcionamiento del integrado MAX232

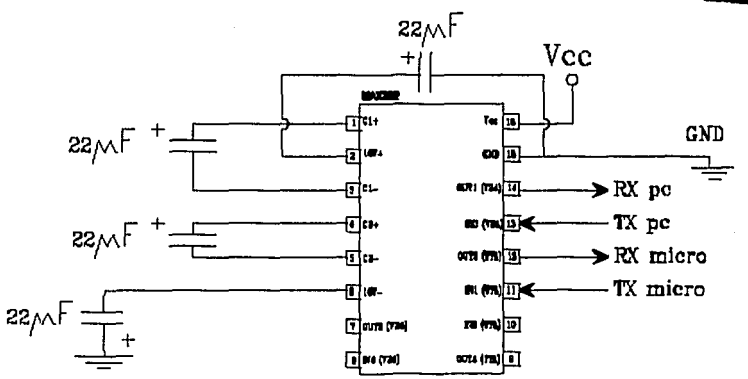


Figura 5. Configuración del MAX232 para conectar el 68HC11 a un PC (norma RS-232C)

Una de las ventajas de este dispositivo o similar es que no hacen uso de alimentaciones externas diferentes a las habituales en estos sistemas (5V), evitando de esta manera aumentar la complejidad de los sistemas de alimentación.

1.4.4. Pines de petición de interrupciones externas.

·**IRQ**: Este pin provee al microcontrolador de una entrada de interrupción enmascarable, activa a nivel bajo, y de colector abierto lo que permite conectar varios dispositivos a la vez. Debe permanecer conectada a una resistencia de pull-up (4K7Ω).

·**XIRQ**: Del mismo modo que el pin anterior, este permite producir todas las peticiones de interrupción con la salvedad de que estas son no enmascarables. Uno de los usos más comunes de esta entrada es para activar alguna rutina de servicio referente a caída de la alimentación del sistema, o cualquier otra función carácter prioritario como esta última.

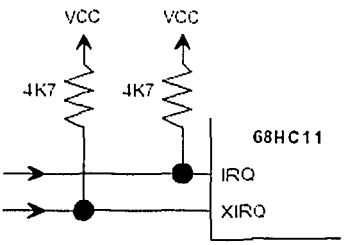


Figura 6. Peticiones de interrupción externas

1.4.5. Pines de reloj

·**EXTAL** y **XTAL**: Son las conexiones de entrada para la introducción de una señal de reloj. El microcontrolador está diseñado para trabajar con osciladores de cristal de la forma que muestra la figura 7. Esta frecuencia de reloj que es introducida en el MCU, es la encargada de regir el funcionamiento interno de los subsistemas que lo componen, por lo que no hay que confundirla con la señal de reloj de sus buses de datos ya sea internos o externos. La velocidad máxima aconsejable por el fabricante está en torno a los 8Mhz.

Es muy recomendable trabajar a esta frecuencia ya que de esta manera se consigue que el chip disponga de valores de velocidades para las transmisiones asíncronas compatibles con el estándar RS232c como los típicos 9600 baudios. Para frecuencias altas, mayores de 1 MHz el circuito es el mostrado en la figura 7.

Para trabajar a frecuencias más bajas es necesaria la inclusión de una resistencia más para lograr que la impedancia de salida aumente y no afecte mucho al MCU.

Los valores que da el fabricante para los componentes del circuito de reloj son:

R = 1 - 10MΩ y C1 = C2= 5 - 25pF. En el circuito de reloj de la tarjeta CT6811 los valores empleados son C1 = C2 = 22pF y R = 10MΩ. El valor del cristal es de 8 MHZ.

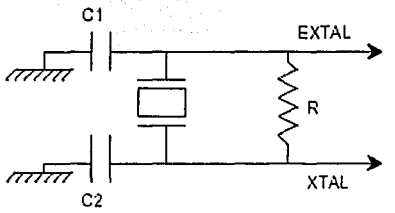


Figura 7. Circuito de reloj para frec. mayores de 1MHZ

1.4.6. Pines de configuración de los modos de arranque.

·**MODA** y **MODB**: Estas dos señales son tenidas en cuenta por el microcontrolador únicamente en el momento del arranque del sistema. Según el nivel al que se encuentren, el 68HC11 se configurará en alguno de los 4 modos del HPRIO:

MODA	MODB	Modo de arranque
0	0	Especial Bootstrap
0	1	Normal
1	0	Especial extendido (Test)
1	1	Normal extendido

1.4.7. Pines de transmisiones serie síncrona.

·**SCK**: Este pin se refiere a la señal de reloj que comanda cualquier transmisión síncrona. Puede ser tanto de salida como de entrada, según que el microcontrolador trabaje como maestro o esclavo respectivamente.

·**MISO y MOSI**: (Master In Slave Out) (Master Out Slave In), estos dos pines son las dos vías por donde van a fluir los datos. Para el caso en que se configure el 68HC11 para trabajar como Maestro frente a otro dispositivo, el cual hace de esclavo, el pin MOSI es el que funciona como salida de datos desde el maestro al esclavo y el pin MISO es la vía contraria, es decir por donde el maestro recibe los datos del esclavo.

68HC11 como esclavo: Cuando SS = 0 se activa el sistema de transferencias síncronas, en éste caso, la señal SCK no es tenida en cuenta y MISO está en estado de alta impedancia.

68HC11 como maestro: SS no repercute dentro de la transmisión propiamente dicha y puede ser utilizada como detección de errores, o de propósito general.

Normalmente se utiliza para activar al esclavo, aunque cualquier bit de cualquier otro puerto sirve para esta función.

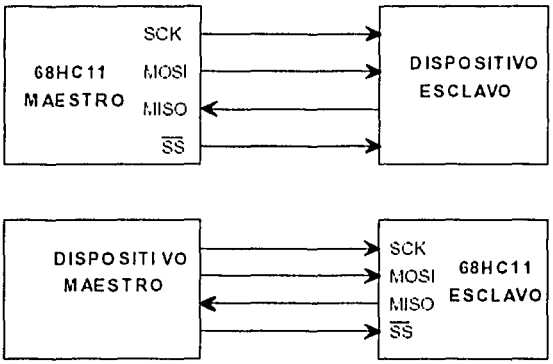


Figura 8. El 68HC11 funcionando como maestro y como esclavo

1.4.8. Pines de los puertos de E/S.

La integración y la potencia de este microcontrolador obliga a un alto grado de multiplexación de las funciones que soporta, por tanto, si bien la cantidad de puertos existentes en el chip es grande y variada, estos se ven a veces desactivados ya que son utilizados para otros fines como comparadores, capturadores, control de transmisiones asíncronas, etc; de tal manera que el número de puertos netos, puede hasta resultar escaso para algunas aplicaciones.

Esto, que siempre es dependiente del sistema en cuestión, tiene una sencilla solución ya que el 68HC11 está preparado para soportar diferentes tipos de expansiones.

Estructura general de los puertos:

Puerto	A	B	C	D	E
Bit 0	entrada	salida	bidireccional	entrada	entrada
Bit 1	entrada	salida	bidireccional	salida	entrada
Bit 2	entrada	salida	bidireccional	bidireccional	entrada
Bit 3	salida	salida	bidireccional	bidireccional	entrada
Bit 4	salida	salida	bidireccional	bidireccional	entrada
Bit 5	salida	salida	bidireccional	bidireccional	entrada
Bit 6	salida	salida	bidireccional	-	entrada
Bit 7	bidireccional	salida	bidireccional	-	entrada

1.4.9. Pines de los buses.

Cuando el microcontrolador se configura para funcionar en el *modo extendido*, es decir, con la capacidad de redirecciónar 64Kbytes de memoria, se ve obligado a generar un bus de direcciones, uno de datos y uno de control. De esta manera, el 68HC11 deja que su CPU interna tenga acceso al exterior, por lo que a partir de aquí, de alguna manera, el microcontrolador comienza a funcionar como un microprocesador . Dicho "nuevo" microprocesador tiene una potencia razonable regida por un bus de datos de 8 bits, y un bus de direcciones de 16 bits, por lo que su espacio de direccionamiento es de 64Kbytes de memoria plana. Este mapa de memoria es plano y comparte espacio con los puertos que se incorporen a nuestro sistema y todos los registros internos de MCU.

En caso de que se superpongan registros internos del microcontrolador con dispositivos externos como pueden ser secciones de memoria; el gestionador de bus, da prioridad a los internos, dejando de lado los restantes.

Mediante la utilización de la totalidad del PUERTO B, el microcontrolador lleva al exterior la PARTE ALTA DEL BUS DE DIRECCIONES, y por medio del PUERTO C se presenta al exterior de forma multiplexada la PARTE BAJA DEL BUS DE DIRECCIONES, y el BUS DE DATOS. Esta multiplexación obliga a tener que agregar un hardware adicional que separare ambos buses. Para esto se puede utilizar un registro tipo latch triestado (para no cargar el circuito) ya sea por flanco (74374) o por nivel (74373, recomendado por el fabricante). Con este latch se intercepta la salida del puerto C de tal forma que se capture la parte baja del bus de direcciones, y no se solape con el dato.

Ahora bien, esta multiplexación viene regida por una señal perteneciente al bus de control llamada E . Por este pin se va a obtener una señal de reloj que es la que se entrega a los periféricos y que por tanto comanda el **ciclo de bus** del sistema, de tal forma que cuando dicha señal E se encuentra a nivel bajo, por el puerto B se direcciona la parte alta del bus A, y por el puerto C la parte baja, los cuales serán capturados en el latch. Finalmente, cuando E pase, en la segunda parte del ciclo del bus, al nivel alto, el puerto C presenta a su salida al bus de datos completando de esta forma la demultiplexación.

En la figura 9 se muestra cómo realizar esta demultiplexación.

·**AD0-AD7**: Señales de los buses (A y D) multiplexadas.

·**A8-A15**: Señales de la parte alta del bus de direcciones (A).

·**AS**: Señal de validación de la dirección puesta en el bus A, la cual es muy útil para validar la captura del latch, ya que al activarse informa que el bus de direcciones está completo.(AS pertenece al bus de control)

·**R/W**: Señal de Lectura (nivel alto) y Escritura (nivel bajo), la cual se comporta de manera idéntica a cualquier microprocesador. (R/W pertenece al bus de control).

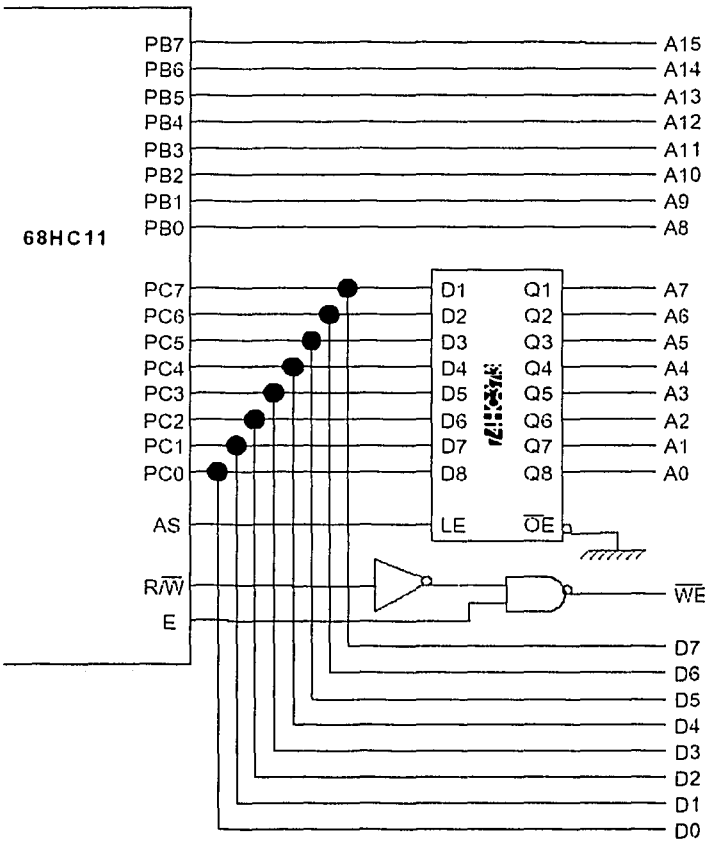


Figura 9. Demultiplexación del bus de datos y direcciones.

1.4.10. Pines de los conversores A/D.

·**AN0-7**: Este sistema es uno de los más populares y además delicado. Se basa en un conversor Analógico Digital de 8 bits que cuenta con 8 canales a los cuales puede acceder de 4 en 4. Las velocidades de muestreo están sujetas a la velocidad de reloj del microcontrolador, a la vez que las señales de referencia se pueden fijar externamente, es decir, disponer al microcontrolador para que convierta a digital señales analógicas que varíen entre 0 y +6 voltios. Es recomendable la utilización precavida del sistema ya que una entrada de tensión fuera de los niveles de referencia prefijados provoca un corto interno y la alta probabilidad de que se destruya, al menos, el canal en cuestión. Por último cabe destacar, que dependiendo del rango de tipo de señales analógicas que se esperan recibir, es siempre aconsejable el uso de filtros que acondicionen las mismas para una mejor conversión.[4]

1.5. Modos de funcionamiento.

El 68HC11 puede funcionar en 4 modos diferentes: Single chip, expanded, bootstrap y special test. En cada modo se dispone de un mapa de memoria diferente, como se muestra en la figura 10.

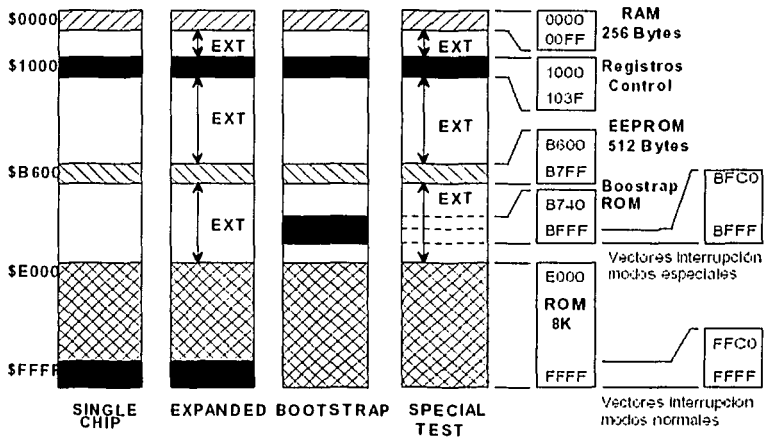


Figura 10: Configuración del mapa de memoria para los diferentes modos de funcionamiento del 68HC11.

· **Single chip:** En este modo de funcionamiento, el mapa de memoria del 68HC11 está constituido por la memoria RAM, la memoria EEPROM, los registros de control y la memoria ROM. Este modo está pensado para funcionar cuando existe un programa grabado en la ROM, de tal manera que al arrancar se comience a ejecutar el programa indicado por los vectores de interrupción que se encuentran en ROM.

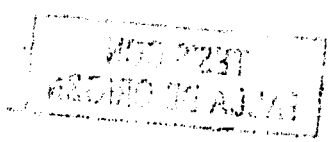
· **Expanded**: Además del mapa de memoria del modo single chip, es posible acceder al resto de las posiciones de memoria conectando memorias externas. El precio a pagar es que se pierden dos puertos de E/S, el puerto B y C, que se utilizarán como bus de datos y direcciones. En este modo se puede utilizar 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.

· **Bootstrap:** Este modo difiere del single chip en que los vectores de interrupción no se encuentran en la memoria ROM de 8Kbytes sino que se encuentran en otra memoria ROM, llamada ROM de arranque. Al arrancar en este modo, automáticamente comienza a ejecutarse el programa BOOTSTRAP que se encuentra en ROM.

· **Special test**: Igual que el 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 otros modos están protegidos.

Los modos de funcionamiento se pueden configurar de dos formas diferentes: configuración hardware y configuración software. La configuración hardware consiste en colocar unos determinados niveles lógicos en las patas **moda y modb** (sección 1.4.6.).

Al realizar un reset del 68HC11 el micro arrancará en el modo especificado por las tensiones de los pines moda y modb. La configuración software del modo de funcionamiento se basa en la modificación del registro HPRIO situado en la dirección $103C.[5]



CAPÍTULO II.
"CARACTERÍSTICAS DE SOFTWARE DEL MICROCONTROLADOR 68HC11 DE MOTOROLA".

2.1. Registros de la CPU.

La CPU del MCU dispone de 2 registros acumuladores de 8 bits, que se unen para formar el registro D de 16 bits, siendo el acumulador A la parte alta y el acumulador B la parte baja. Además dispone de 2 registros para direccionamiento indexado X, Y ambos de 16 bits.

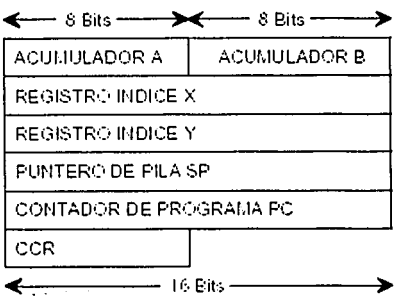


Figura 11. Registros del 68HC11

El puntero de pila y el contador de programa son también de 16 bits, lo que permite que la longitud máxima de un programa sea de 64Kbytes, que es el espacio máximo direccionable por el MCU. El registro CCR es el llamado registro de estado, que contiene unos bits de especial importancia que reflejan el estado de la CPU.

El **puntero de pila** debe ser inicializado por el usuario. La pila "crece" desde direcciones altas a direcciones bajas, por lo que al introducir un elemento en la pila, SP se decrementa en 1 ó 2 bytes dependiendo del tamaño del dato metido en la pila. Al sacar un elemento de la pila, SP se incrementa.

El contador de programa **PC** se va incrementado 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 a SP un valor "seguro" de tal manera que la pila no se solape con el código, si es que el código se encuentra en RAM.

El registro **CCR** es de 8 bits. Cada bit tiene una letra asignada y representa una situación diferente del estado de la CPU. [1]

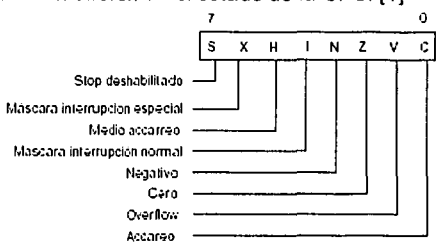


Figura 12: Registro CCR

2.2. MODOS DE DIRECCIONAMIENTO.

Existen 6 modos de direccionamiento distintos. Los modos de direccionamientos son distintas formas que tiene la CPU de acceder a los datos que están en memoria.

2.2.1. Direccionamiento inmediato.

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 ó 2 bytes. Este modo de direccionamiento se indica mediante el signo #.

Ejemplo: **LDAA #08**

Esta instrucción carga el valor decimal 8 en el acumulador A. El valor 08 se encuentra a continuación del código de instrucción de LDAA. La instrucción LDAA tiene el código $86, por tanto, en memoria esta instrucción queda representada mediante los valores: (Se supone que esta instrucción comienza en la dirección $0000).

Dir. memoria	contenido	nemónico
0000	86	LDAA
0001	08	8

2.1.1. Direccionamiento extendido.

El dato se encuentra en la dirección de memoria especificada. El dato puede estar en cualquier posición de la memoria dentro del límite de las 64Kb, por lo que la dirección ocupa 2 bytes.

Ejemplo: **LDAA $FC00**

Esta instrucción carga en el acumulador el contenido de la dirección $FC00. La dirección del dato se almacena después del código de la instrucción y ocupa 2 bytes. Las instrucciones con este modo de direccionamiento ocupan 3 bytes (1 byte para el código de la instrucción y 2 bytes para la dirección).

2.2.3. Direccionamiento directo.

Este modo de direccionamiento es similar que el anterior con la diferencia de que sólo actúa con direcciones comprendidas entre $00-$FF (256 primeros 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.

Ejemplo: **(1) LDAB $FC00** **(2) LDAB $05**

Ambas instrucciones cargan en el acumulador B el contenido de la dirección especificada. En el caso de la instrucción 1, se utiliza direccionamiento extendido y la instrucción ocupa 3 bytes. En el caso de la instrucción 2, como la dirección es menor que $FF se utiliza direccionamiento directo y la instrucción ocupa 2 bytes.

Por lo anterior, al comparar ambos modos de direccionamiento, siempre que sea posible conviene usar el direccionamiento directo, es decir, **"SITUAR LAS VARIABLES EN LAS DIRECCIONES BAJAS DE LA MEMORIA, EN EL ESPACIO $00-$FF".** De esta manera, todas las instrucciones que hagan referencia a variables, utilizarán direccionamiento directo y se ahorrarán muchos bytes de memoria.

¿Quién decide el modo de direccionamiento?. Es el propio ensamblador. Al encontrarse el ensamblador con la instrucción 1 (LDAB $FC00) sabe que la dirección es superior a $FF y que debe utilizar direccionamiento extendido. Al encontrarse con la instrucción 2, la dirección es menor que $FF y por tanto utiliza direccionamiento directo ahorrando un byte.

2.2.4. Indexado.

Este modo de direccionamiento se utiliza para acceder a TABLAS (Arrays o cadenas) en la memoria. El dato se busca de la siguiente forma: Se toma la dirección del registro índice (X ó Y), se le suma un desplazamiento (offset) de 8 bits y el contenido de esa dirección es el dato buscado. Este modo de direccionamiento se especifica colocando como argumentos en la instrucción un offset, una coma y el registro índice.

Ejemplo: **LDAB 5,X**

Esta instrucción carga en el acumulador B el contenido de la dirección especificada por X más un offset de 5. Se simboliza de la siguiente manera:

dir = (X) + 5 , siendo dir la dirección que contiene el dato. Para el caso particular de tener un offset de 0, es decir, que se quiera acceder a la dirección contenido en X, se puede especificar de las siguientes maneras:

LDAB 0,X
LDAB ,X
LDAB X

Las 3 instrucciones son equivalentes. Este modo de direccionamiento es muy interesante porque permite acceder a cualquier dirección de memoria ($0000-$FFFF) como en el direccionamiento extendido pero las instrucciones sólo ocupan 2 bytes, como en el direccionamiento directo. Además, el registro X se puede variar (incrementar, decrementar) con lo que se obtiene una gran flexibilidad a la hora de acceder a tablas de datos.

Para acceder a los **registros de configuración** del microcontrolador, que se encuentran en las direcciones $1000-$103F, es conveniente utilizar este modo de direccionamiento porque así las instrucciones ocupan menos bytes. En el registro índice se introduce la dirección $1000 correspondiente al comienzo de los registros de configuración del microcontrolador, y sólo es necesario especificar el desplazamiento:

Instrucciones	**Comentarios**	**Tamaño instrucción**
LDX $1000	; X = $1000	(3 Bytes)
STAA 1,X	; Meter A en dir. $1001	(2 Bytes)
STAB $2C,X	; Meter B en dir. $102C	(2 Bytes)
STAA $10,X	; Meter A en dir. $1010	(2 Bytes)

Para realizar esto mismo con direccionamiento extendido sería:

```
STAA $1001      ; Meter A en dir. $1001      (3 Bytes)
STAB $102C      ; Meter B en dir. $102C      (3 Bytes)
STAA $1010      ; Meter A en dir. $1010      (3 Bytes)
```

Para acceder a 3 registros de control distintos, ambos trozos de código ocupan 9 bytes de memoria. Pero si se pretende acceder a más de 3 registros de control, que suele ser lo más habitual, se ahorra memoria utilizando direccionamiento indexado. El offset aplicado es de 8 bits y **sin signo** por lo que el offset máximo es de 256 bytes.

2.2.5. Direccionamiento relativo.

Este modo de direccionamiento sólo se utiliza con las instrucciones de **bifurcación.** Estas indican a la CPU que realice un salto de tantos bytes hacia adelante 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 adelante:

Ejemplo:

```
Bucle
        ....
        ....
BNE bucle
```

El programador no necesita calcular el salto a efectuar, lo realiza automáticamente el ensamblador. Sin embargo, es importante saber que los saltos con instrucciones BR sólo se pueden realizar hacia posiciones de memoria que estén a menos de 128 bytes por debajo y a menos de 127 bytes por arriba. Si se sobrepasa el límite el ensamblador dará un mensaje de error.

Ejemplo:

```
Bucle Inst1 ; Dirección 0
        ......
        ......
BNE bucle ; Dirección 200
```

Este programa daría error puesto que el salto que la bifurcación realiza es de más de 128 bytes hacia arriba. La ventaja es que todas las instrucciones BR (BRANCH) ocupan 2 bytes. La desventaja es que sólo se pueden hacer bifurcaciones relativamente cortas. Para realizar bifurcaciones a cualquier posición de la memoria se utilizan las instrucciones JUMP (JMP y JSR) , que ocupan 3 bytes.

2.2.6. Direccionamiento inherente.

Los operandos se encuentran en registros de la CPU. Por el código de la instrucción la CPU sabe de qué registros se trata.

Ejemplo:	**CÓDIGO**	**MNEMÓNICO**	
	$1B	ABA	; A:=A+B
	$5C	INCB	; B:=B+1
	$08	INX	; X:=X+1

La primera instrucción, de código $1B, suma el contenido del acumulador A y B y el resultado lo introduce en el acumulador A. La segunda incrementa el acumulador B y la tercera el registro de índice X. Estas instrucciones sólo ocupan un byte y por ello conviene abusar de ellas.[2]

2.3. JUEGO DE INSTRUCCIONES.

Las instrucciones se dividen en distintos grupos. Todas las instrucciones tienen dos campos: uno es mnemónico y el otro es el dato o la dirección a la que hace referencia la instrucción. Este campo es opcional.

2.3.1. Instrucciones de carga, almacenamiento y transferencia.

· **CARGA:** Estas instrucciones permiten introducir un nuevo valor en los registros, leer una posición de memoria, un puerto, etc.

LDAA : Introducir un dato de 8 bits en el acumulador A

```
Ej.   LDAA #30        ; A:=30 (Direccionamiento inmediato)
      LDAA $1000      ; Introducir en A el contenido de la dirección $1000.
```

LDAB : Introducir un dato de 8bits en el acumulador B

LDD : Introducir un dato de 8 ó 16 bits en el doble acumulador D (Formado por A y B yuxtapuestos)

```
Ej.   LDD #$FFCC      ; D:=$FFFF --> A:=$FF; B:=$CC
      LDD #$10        ; D:=$0010 --> A:=$00; B:=$10
```

LDX : Introducir un dato de 16 bits en el registro índice X

```
Ej.   LDX #$1000      ; X:=$1000
      LDX 3,Y         ; Meter en X el contenido de la dirección Y+3
      LDX 5,X         ; Meter en X el contenido de la dirección X+3
```

LDY : Introducir un dato de 16 bits en el registro de índice Y.

LDS : Introducir un dato de 16 bits en el SP (puntero de pila). Esta instrucción hay que utilizarla al menos una vez en nuestros programas para inicializar la pila.

Ej. LDS #$FC00 ;Inicializar la pila a partir de la dirección $FC00 hacia ;abajo

CLRA : Borrar el contenido del acumulador A. Esta instrucción hace lo mismo que LDAA #0, con la diferencia de que el direccionamiento es inherente y sólo ocupa 1 byte, mientras que LDAA #0 ocupa 2 bytes.

CLRB : Borrar el contenido del acumulador B.

· **ALMACENAMIENTO**: Estas instrucciones permiten alterar una posición de memoria, un puerto, registros internos, etc.

STAA : Almacenar el acumulador A en una dirección de memoria.

Ej. STAA $1000 ; Mandar el acumulador por el puerto A

STAB : Almacenar el acumulador B.

STD : Almacenar el doble acumulador D. (16 bits).

STX : Almacenar el registro de índice X(16 bits).

STY : Almacenar el registro de índice Y (16 bits).

STS : Almacenar el puntero de pila SP.

CLR : Poner a cero el contenido de una dirección de memoria.

Ej. CLR $1000 ; Mandar un 0 por el puerto A

· **TRANSFERENCIAS**: Permiten transferir datos entre registros y registros y memoria. El direccionamiento es inherente por lo que no es necesario especificar dirección. Con el mnemónico basta.

PSHA : Introducir el acumulador A en la pila. Se introduce A en la dirección especificada por SP. SP se decrementa en 1
PSHB : Introducir el acumulador B en la pila.

PSHX : Introducir el registro de índice X en la pila. Se introduce X en la pila. SP se decrementa en 2 unidades puesto que X es de 16 bits.

PSHY : Introducir el registro de índice Y en la pila.

PULA : Sacar A de la pila. Se decrementa SP en una unidad y se introduce en A el dato contenido en la posición apuntada por SP.

PULB : Sacar B de la pila.

PULX : Sacar X de la pila. SP se incrementa en 2 unidades ya que X es de 16 bits.

PULY : Sacar Y de la pila.

TAB : Introducir el valor de A en B.

TBA : Introducir el valor de B en A.

TSX : Introducir el valor de SP en X.

TSY : Introducir el valor de SP en Y.

TXS : Introducir el valor de X en SP.

TYS : Introducir el valor de Y en SP.

XGDX : Intercambiar el registro D con el X.

XGDY : Intercambiar el registro D con el Y.

2.3.2. Instrucciones aritméticas.

· **SUMAR**

ADDA : Añadir un dato al acumulador A

Ej. ADDA #5 ; Sumar 5 al acumulador.
ADDA $C000 ;Sumar el contenido de la dir. $C000 al acumulador.

ADDB : Añadir un dato al acumulador B.

ADDD : Añadir un dato al doble acumulador D.

ADCA : Añadir al acumulador A un dato y el contenido del acarreo.

ADCB : Añadir al acumulador B un dato y el contenido del acarreo.

ABA : Sumar el acumulador A y B y poner el resultado en A.

ABX : Sumar B y X y poner resultado en X.

ABY : Sumar B y Y y poner resultado en Y.

INCA : Incrementar el acumulador A.

INCB : Incrementar el acumulador B.

INC : Incrementar el contenido de una dirección de memoria.

Ej. INC 2,Y ; Sumar una unidad al byte que se encuentra en la dir. Y+2.

INX : Incrementar registro X.

INY : Incrementar registro Y.

INCS : Incrementar puntero de pila SP.

· **RESTAR**

SUBA : Restar un dato al acumulador A.

Ej. SUBA #$2C ; Restar $2C al acumulador.

SUBB : Restar un dato al acumulador B.

SUBD : Restar un dato al doble acumulador D.

SBCA : Restar al acumulador A un dato y el contenido del acarreo.

SBCB : Restar al acumulador B un dato y el contenido del acarreo.

DECA : Decrementar acumulador A.

DECB : Decrementar acumulador B.

DEC : Decrementar byte de una dirección de memoria.

DEX : Decrementar registro X.

DEY : Decrementar registro Y.

DES : Decrementar puntero de pila SP.

· **COMPARACIONES**

CMPA : Comparar acumulador A con un dato. Se activan los bits correspondientes del registro de status. Los bits que se activan son el Z(Cero) y el N (negativo).

Ej. CMPA #10 ; Comparar Acumulador A con 10.

CMPB : Comparar Acumulador B con un dato.

CPD : Comparar doble acumulador D con un dato.

CPX : Comparar registro X con un dato.

CPY : Comparar registro Y con un dato.

CBA : Comparar A con B.

· **COMPLEMENTO A DOS.**

Estas instrucciones permiten obtener números negativos en formato de complemento a dos.

NEG : Complementar a dos un byte de la memoria.

NEGA : Complementar a dos el acumulador A.

NEGB : Complementar a dos el acumulador B.

· **MULTIPLICACIONES Y DIVISIONES**

MUL : Se multiplican A y B y el resultado se introduce en el doble acumulador D.

IDIV : Se divide D entre X y el resultado se guarda en X. El resto se guarda en D.

2.3.3. Operaciones lógicas y de manipulación de bits.

ANDA : Se realiza una operación AND lógica entre el registro A y la memoria. El resultado se almacena en el acumulador A.

ANDB : Idem pero con el acumulador B.

ORAA : Se realiza una operación OR lógica entre el acumulador A y la memoria. El resultado se almacena en el acumulador A.

ORAB : Idem pero con el acumulador B.

EORA : Realizar un or-exclusivo (XOR) entre el registro A y memoria y almacenar resultado en acumulador A.

EORB : Idem pero con el acumulador B.

COMA : Se realiza el complemente a uno de A.

COMB : Se realiza el complemento a uno de B.

BITA : Esta instrucción sirve para comprobar si determinados bits de una posición de memoria están activados o no. Se realiza un AND lógico entre el acumulador y la posición de memoria pero no se altera ninguna de las dos. El resultado queda reflejado en el bit Z del registro CCR.

Ejemplo: Queremos comprobar si los bits 0 y 1 del puerto A están ambos activados:

```
LDAA #$03       ; Meter el valor $03 (00000011 en binario) en A
BITA PORTA      ; Comprobar bits 0 y 1.
BEQ subrutina   ; Saltar si ambos bits son cero.
```

BITB : Idem que BITA pero con el acumulador B.

BCLR : Poner a cero los bits especificados de una posición de memoria. La sintaxis es: **BCLR** *operando máscara*. El operando es una posición de memoria a la que se puede acceder mediante cualquiera de los modos de direccionamiento. Máscara es un byte, cuyos bits a uno se corresponden con los bits del operando que se quieren poner a cero. Por ejemplo, se quiere poner a cero los bits 0 y 1 del puerto A:

```
BCLR PORTA,X $03
```

¡ Sólo está permitido el direccionamiento indexado!.

BSET : Lo mismo que BCLR pero los bits en vez de ponerse a cero se ponen a 1.

BRCLR : Esta instrucción es muy útil y un poco diferente del resto porque tiene 3 parámetros. Se bifurca a la dirección especificada si unos bits determinados están a cero. La sintaxis es: **BRCLR** *operando máscara dirección*. Se realiza un AND lógico entre el operando y la máscara y se bifurca si la operación da como resultado cero, es decir, si todos los bits indicados estaban a cero.

Un ejemplo muy típico es un bucle de espera hasta que se active un bit de una posición de memoria:

```
      LDX #$1000
wait BRCLR 0,X $80 wait
```

El bucle se repite mientras el bit 7 del puerto A sea distinto de cero. En cuanto se ponga a cero se sale del bucle. **¡Esta instrucción sólo permite direccionamiento indexado!.**

BRSET :Igual que BRCLR pero se salta cuando los bits indicados se ponen a 1.

2.3.4. Desplazamientos y rotaciones.

· **Desplazamientos aritméticos:**

ASL : Desplazamiento aritmético a la izquierda de un operando en memoria.

ASLA : Desplazamiento aritmético a la izquierda del acumulador A.

ASLB : Idem pero con el acumulador B.

ASLD : Idem pero con el doble acumulador D.

ASR : Desplazamiento aritmético a la derecha de un operando en memoria.

ASRA : Desplazamiento aritmético a la derecha del acumulador A.

ASRB : Idem con el acumulador B.

· **Desplazamientos lógicos:**

LSR : Desplazamiento lógico a la derecha de un operando en memoria.

LSRA : Desplazamiento lógico a la derecha del acumulador A.

LSRB : Idem con el acumulador B.

LSRD : Idem con el acumulador D.

· **Rotaciones.**

ROL : Rotación a la izquierda de un operando en memoria.

ROLA :Rotación a la izquierda del acumulador A.

ROLB : Rotación a la izquierda del acumulador B.

ROR : Rotación a la derecha de un operando en memoria.

RORA : Rotación a la derecha del acumulador A.

RORB : Rotación a la izquierda del acumulador B.

2.3.5. Bifurcaciones y saltos.

· **Bifurcaciones.** Las bifurcaciones (instrucciones BRANCH) se diferencian de los saltos en que se realizan mediante direccionamiento relativo por lo que sólo se pueden utilizar para saltar 128 bytes hacia atrás o 127 bytes adelante. Las bifurcaciones condicionales bifurcan a la dirección especificada cuando se da una determinada condición en el registro de estado CCR.

BCC : Bifurcación si acarreo está a cero.

BCS : Bifurcación si acarreo está a uno.

BEQ : Bifurcar si el resultado a sido cero (Z=1).

BGE : Bifurcar si mayor o igual (Signo).

BGT : Bifurcar si mayor que (Signo).

BHI : Bifurcar si mayor que (Sin signo).

BHS : Bifurcar si mayor o igual (Sin signo).

BLE : Bifurcar si menor o igual (Signo).

BLO : Bifurcar si menor (Sin Signo).

BLS : Bifurcar si menor o igual (Sin signo).

BLT : Bifurcar si menor (Signo).

BMI : Bifurcar si negativo (N = 1).

BNE : Bifurcar si no igual (Z = 0).

BPL : Bifurcar si positivo (N = 0).

BVC : Bifurcar si overflow está a cero (V = 0).

BVS : Bifurcar si overflow está a uno (V = 1).

BRA : Bifurcar (Salto incondicional).

BSR : Llamar a una subrutina (incondicional).

· **Saltos.** Los saltos se pueden realizar a cualquier dirección de memoria.

JMP : Salto incondicional.

JSR : Salto incondicional a una subrutina.

2.3.6. Instrucciones de modificación del CCR.

Estas instrucciones alteran los bits del registro de estado CCR.

CLC : Poner a cero el bit de acarreo.

SEC : Poner el bit de acarreo a uno.

CLI : Poner el bit de interrupciones a cero. Las interrupciones se permiten.

SEI : Poner el bit de interrupciones a uno. Las interrupciones se inhiben.

CLV : Poner el bit de overflow a cero.

SEV : Poner el bit de overflow a uno.

TAP : Mover el Acumulador A al registro CCR.

TPA : Mover el CCR al acumulador A.

2.3.7. Otras instrucciones.

RTS : Retornar de una subrutina.

RTI : Retornar de una interrupción.

SWI : Interrupción Software.

WAI : Esperar hasta que ocurra una interrupción.

NOP : No operación. No hace nada salvo consumir un ciclo de reloj.

STOP : Parar el reloj. [3]

2.4. INTERRUPCIONES

Las interrupciones son señales generadas interna o externamente al microcontrolador que provocan que la CPU deje de ejecutar el programa en curso y ejecute una rutina específica para atender a la interrupción. Una vez ejecutada la rutina de servicio de la interrupción, la CPU continúa con el programa que estaba ejecutando antes de producirse la interrupción.

Las direcciones de las rutinas de servicio de las interrupciones se encuentran en una tabla en memoria, denominada tabla de vectores de interrupción. Existen dos tablas de vectores de interrupción según el modo de funcionamiento del MCU. Si el modo de funcionamiento es el especial de arranque, la tabla se encuentra en memoria ROM en las direcciones $BFD6-$BFFE. Si funciona en modo normal o extendido, la tabla se encuentra en las direcciones $FFC0-$FFFE. En la tabla hay 21 vectores, y cada vector contiene una dirección de memoria que ocupa 2 bytes, por tanto las tablas de vectores ocupan 21*2 = 42 bytes. En la figura 13 se muestra la tabla de vectores de interrupción.

2.4.1. Interrupción de RESET.

La interrupción de RESET es una interrupción especial. Ocurre cada vez que se recibe un nivel bajo en el pin de RESET del MCU (cada vez que se pulsa el botón de reset). Al producirse esta interrupción, la CPU toma de la tabla de vectores de interrupción la dirección de la rutina que tiene que empezar a ejecutar. Si el MCU funciona en modo especial de arranque, se empezará a ejecutar un programa en ROM, llamado BOOTSTRAP, que permite cargar un programa cualquiera procedente del exterior en la memoria RAM. Si el MCU funciona en modo normal o expandido, se ejecuta la rutina indicada por su correspondiente vector de interrupción.

Al producirse el reset, las interrupciones se inhiben y el contenido de los registros queda indeterminado.

La memoria RAM se mapea en las direcciones $0000-$00FF del mapa de memoria, y los registros de control se sitúan en las direcciones $1000-$103F.

Las memorias ROM y EEPROM quedan configuradas de la misma manera que lo estaban antes del reset puesto que su registro de configuración no se borra al eliminar la alimentación (es un registro tipo EEPROM). La mayoría de los periféricos del MCU (SCI, SPI, Puertos, temporizadores, ...) sufren algún cambio en sus registros.

2.4.2. Tipos de interrupciones.

Existen 3 tipos de interrupciones: las interrupciones enmascarables, las no enmascarables y las interrupciones software.

· **Las interrupciones enmascarables.** Como su nombre indica, se pueden enmascarar, es decir, inhibir sin más que actuar sobre el bit I del CCR. (I = 0 ⇒ se permiten interrupciones; I = 1⇒ Interrupciones inhibidas). Con la instrucción CLI, el bit I del CCR se pone a cero y se permiten las interrupciones. Con SEI, I se pone a 1 y se inhiben las interrupciones.

· **Las interrupciones no enmascarables** no se pueden inhibir. Son las interrupciones correspondientes a un fallo en el MCU, instrucción ilegal, RESET y la interrupción externa XIRQ.

· **Las interrupciones software** son las producidas por el propio programador en unos instantes totalmente conocidos. Sólo existe una interrupción software que se produce con la instrucción SWI. La CPU al leer esta instrucción actúa como si de una interrupción normal se tratase. Estas interrupciones son no enmascarables porque de lo contrario la CPU se colgaría:

```
SEI     ; Inhibir interrupciones
SWI     ; ¡ Nunca se ejecutaría !!
```

Si se pudiesen enmascarar, la CPU nunca ejecutaría la instrucción SWI y quedaría colgada hasta que se "resetease".

También es posible clasificar las interrupciones en internas y externas.

· **Las interrupciones internas** son las producidas por circuitos o periféricos integrados dentro del propio microcontrolador.

· **Las interrupciones externas** son las producidas por circuitos o periféricos externos al MCU. Existen 2 entradas de interrupciones externas:

IRQ es una entrada de interrupciones enmascarables.

XIRQ es la entrada de interrupciones no enmascarables.

IC1, IC2, IC3, PAI y STRA pueden considerarse como interrupciones externas especiales.

2.4.3. Prioridad de las interrupciones.

No todas las interrupciones tienen la misma prioridad. Si se producen dos interrupciones a la vez, primero se ejecuta la de mayor prioridad y después la de menor. Por ejemplo, si se producen a la vez la interrupción de RESET y una procedente del puerto de comunicaciones SERIE (SCI), ¿a qué interrupción se atiende antes?. Lógicamente la de RESET tiene una prioridad mayor y se procedería a la reinicialización del sistema. Las interrupciones no enmascarables (RESET, XIRQ) tienen la máxima prioridad.

Las demás interrupciones tienen una prioridad determinada por el hardware, pero es posible hacer que la prioridad de una determinada interrupción sea máxima. Para ello basta con escribir un cierto valor en los bits 0-3 del registro HPRIO ($103C). **Sólo se puede cambiar la prioridad estando las interrupciones inhibidas.**

2.4.4. Proceso de interrupción.

La CPU del microcontrolador está ejecutando un programa. De repente aparece una interrupción. Si las interrupciones están permitidas (Bits X e I del CCR) se introducen todos los registros en la pila, se inhiben las interrupciones y se obtiene de la tabla de vectores de interrupción la dirección a la que tiene que bifurcar la CPU. Las interrupciones se inhiben para que no se produzcan anidamientos de interrupciones, es decir, que mientras se está atendiendo a una interrupción, por defecto no se atienda a otra. El usuario por supuesto puede activar las interrupciones en una rutina de servicio de una interrupción, con lo que provocaría un anidamiento de interrupciones (No es recomendable anidar interrupciones).

Si se está ejecutando la rutina de servicio de una interrupción enmascarable y se produce una interrupción no enmascarable, como esta última tiene una prioridad mayor, la CPU procede a atenderla dejando a la anterior "congelada". Cuando termina con la no enmascarable vuelve con la enmascarable.

Todas las rutinas de servicio de interrupción deben acabar con **la instrucción RTI** que indica a la CPU que la interrupción ha sido atendida y que se puede retornar al programa anterior: La CPU toma de la pila todos los registros que previamente había guardado y continúa ejecutando el programa que había sido interrumpido.

En el microcontrolador existen muchos periféricos integrados que pueden producir interrupciones en cualquier momento. Para generar una interrupción activan un bit de un determinado registro del periférico. Este bit indica a la CPU que el periférico en cuestión ha solicitado interrupción. Cuando la CPU pasa a ejecutar la rutina de servicio del periférico, lo primero que debe hacer es DESACTIVAR ESTE BIT. Si no se hace, al retornar de la interrupción con RTI, el bit seguirá activo y la CPU lo interpretará como una nueva interrupción, con lo que se vuelve a ejecutar la rutina de servicio. Así permanecería la CPU en un bucle

infinito. En algunos periféricos este bit se desactiva automáticamente al ser ejecutada su rutina de interrupción, pero en otros no, como por ejemplo la unidad de comunicaciones serie asíncronas (SCI). Por tanto, **lo primero que hay que hacer en una rutina de servicio de interrupciones es desactivar el bit de interrupción correspondiente**. Según el periférico de que se trate, este bit se desactivará de una forma u otra. En algunos casos se puede desactivar escribiendo directamente un valor en el registro en que está contenido. En otros casos es necesario ejecutar una serie de instrucciones para que se borre. Por ejemplo, suponiendo que se esta diseñando la rutina de servicio del SCI, ésta rutina se ejecuta cada vez que se recibe un carácter por este puerto. Lo primero que hay que hacer es leer el registro de estado y a continuación el dato recibido.

De esta manera el bit de interrupción se desactiva y la rutina puede terminar normalmente con RTI.

2.4.5. Vectores de interrupción.

En la figura 13 se muestran las tablas de vectores de interrupción para los modos single chip, expanded y bootstrap.

En los modos single chip y expanded, los vectores de interrupción están situados a partir de la dirección FFD6. En el modo bootstrap los vectores se encuentran en memoria ROM a partir de la dirección BFC0. Puesto que estos vectores están en ROM su valor no se puede cambiar. Estos vectores apuntan a diferentes zonas de la RAM. Es allí donde se debe realizar el salto a la subrutina de atención a la interrupción.

En la tabla de la figura 13 puede verse a la izquierda la causa de la interrupción. A la derecha están las direcciones de los vectores de interrupción para los distintos modos de funcionamiento.

La columna de la derecha del todo, indica el contenido del vector de interrupción del modo bootstrap, es decir, indica la dirección en RAM a la que se bifurca cuando ocurre una determinada interrupción.

Por ejemplo, si se produce la interrupción correspondiente al SCI, la CPU pasa a ejecutar el código que se encuentra a partir de la dirección $00C4.

Para el caso del modo extendido no se especifica el contenido del vector porque se puede cambiar por software, según los valores colocados en esas posiciones de memoria.[4]

FUENTE DE INTERRUPCION	DIRECCION DEL VECTOR		
	SINGLE CHIP/ EXPANDED	BOOTSTRAP	
		Direccion ROM	Saltar a la direccion
Interrupción del SCI	FFD6	BFD6	00C4
Interrupción del SPI	FFD8	BFD8	00C7
Acumulador de pulsos	FFDA	BFDA	00CA
Overflow en acumulador de pulsos	FFDC	BFDC	00CD
Overflow del temporizador	FFDE	BFDE	00D0
Comparador de salida 5	FFE0	BFE0	00D3
Comparador de salida 4	FFE2	BFE2	00D6
Comparador de salida 3	FFE4	BFE4	00D9
Comparador de salida 2	FFE6	BFE6	00DC
Comparador de salida 1	FFE8	BFE8	00DF
Capturador de entrada 3	FFEA	BFEA	00E2
Capturador de entrada 2	FFEC	BFEC	00E5
Capturador de entrada 1	FFEE	BFEE	00E8
Interrupción de tiempo real	FFF0	BFF0	00EB
Puerto paralelo:IRQ	FFF2	BFF2	00EE
XIRQ	FFF4	BFF4	00F1
Interrupción software	FFF6	BFF6	00F4
Código de instrucción ilegal	FFF8	BFF8	00F7
Fallo del sistema	FFFA	BFFA	00FA
Fallo en el monitor del reloj	FFFC	BFFC	00FD
Interrupción de RESET	FFFE	BFFE	BF40

Figura 13: Tabla con los vectores de interrupción para los diferentes modos.

2.5. MAPA DE MEMORIA

El MCU direcciona 64Kbytes de memoria, parte de esta memoria se encuentra dentro del MCU y el resto se puede implementar mediante chips de memoria externos al MCU. Según el modelo de microcontrolador empleado (A8, A0, A2, E9,...) se dispondrá de más o menos recursos de memoria. La memoria RAM se sitúa por defecto a partir de la dirección $0000 hasta la $00FF (256 bytes de RAM).

De la dirección $1000 hasta la $103F se encuentran situados los registros de control del MCU. Estos registros son células de memoria ROM o EEPROM (no volátiles). Las células EEPROM sólo se pueden escribir bajo unas circunstancias especiales, y siempre que el MCU esté en modo especial.

Entre las direcciones $B600-$B7FF se sitúan 512 bytes de memoria EEPROM que no están disponibles en el modelo A0. En los modos especiales se activa una memoria ROM en las direcciones $B740-$BFFF que contiene el programa BOOTSTRAP usado para cargar programas externos y una tabla con los vectores de interrupción.

Entre las direcciones $E000-$FFFF se encuentra la memoria ROM, que salvo ocasiones muy especiales siempre está desconectada. En esta ROM el fabricante puede grabar cualquier aplicación que el usuario quiera, en este caso, se grabo un sistema operativo llamado BUFFALO, el cual permite la comunicación (vía software) de la tarjeta CT6811 y la PC.

Desde la dirección $FFC0 hasta el final se encuentra la tabla de vectores de interrupción del modo normal.

En la figura 14 se muestra el **mapa de memoria** por defecto. Configurando algunos registros de control adecuadamente se puede "remapear" el sistema para ajustarlo a las necesidades de la aplicación.

Por ejemplo, la memoria RAM se puede situar en cualquier otra parte (con alguna restricción) dentro de los 64Kbytes del mapa de memoria. Lo mismo ocurre con los registros de control

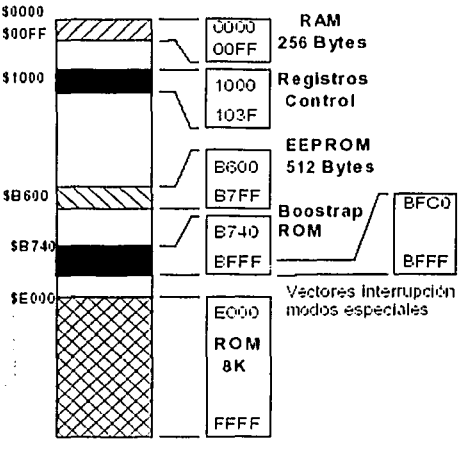


Figura 14: Mapa de memoria en el modo BOOTSTRAP

El registro que permite cambiar las direcciones de la memoria RAM y de los registros de control se denomina INIT y se encuentra en la dirección $103D.

Registro INIT:

Bit	7	6	5	4	3	2	1	0
	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0

Las direcciones de memoria están formadas por 2 bytes, es decir, direcciones de 16 bits. Los bits del 7 al 4 del registro INIT permiten establecer los 4 bits de mayor peso de la dirección en la que se va a mapear la memoria RAM. Los 12 bits restantes no se pueden fijar. Los bits del 3 al 0 hacen lo mismo pero con la dirección de comienzo de los registros de control. El mapa de memoria queda dividido en 16 páginas de 4KBytes cada página (16*4KB = 64Kb). Tanto la RAM interna como los registros de control se pueden situar en cualquiera de estas 16 páginas:

$0000 (Situación de la RAM por defecto),

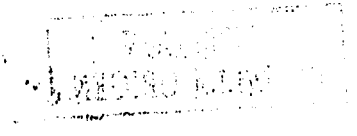
$1000 (Situación de los registros de control por defecto),

$2000, $3000, $4000, $5000 ... ... $D000, $E000 y $F000.

Debido a la posibilidad de remapear, pueden surgir conflictos al encontrarse 2 recursos internos o externos en las mismas posiciones de memoria. El MCU resuelve esto adjudicando unas prioridades.

Si se mapea la RAM, los registros de control y un dispositivo externo en la zona de la ROM (A partir de la dirección $E000 en adelante) el MCU asigna la máxima prioridad a los registros de control, después a la RAM, después la ROM y finalmente el recurso exterior.

En este caso, si se intenta acceder a las direcciones $E000-$E03F se seleccionarían los registros de control. A partir de la $E03F ya no existirían los registros de control y se activaría la memoria RAM. A partir de la dirección en la que la RAM se acaba (sólo son 256 bytes de RAM) se accedería a la ROM. Al dispositivo exterior nunca se podría acceder a no ser que se mapeara fuera de la zona destinada a la ROM o que se desactivase la ROM.[5]



2.6. PUERTOS DE ENTRADA / SALIDA

Existen 5 puertos de 8 bits disponibles: Puerto A, B, C, D y E. Además de comportarse como puertos normales, sus pines están compartidos con alguno de los recursos internos.

2.6.1. PUERTO A

El Puerto A dispone de 3 pines de entrada, 4 pines de salida y uno configurable como entrada o como salida. Se encuentra mapeado en memoria en la dirección $1000, como se muestra en la figura 15.

DIRECCIÓN $1000

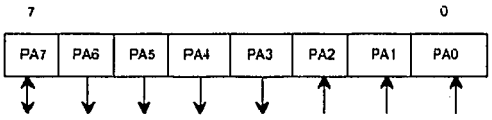


Figura 15: Bits del puerto A.

Los pines del Puerto A están compartidos por otros recursos: comparadores, acumulador de pulsos y capturadores; a continuación se muestra en la figura 16, todas las funciones que están asignadas a cada pin del puerto A.

DIRECCIÓN $1000

7							0
PA7 PAI OC1	PA6 OC2 OC1	PA5 OC3 OC1	PA4 OC4 OC1	PA3 OC5 OC1	PA2 IC1	PA1 IC2	PA0 IC3

Figura 16: Recursos que utilizan cada bit del puerto A

PA0-PA2	Bits de entrada
PA3-PA6	Bits de salida
PA7	Bidireccional
OCx	Salidas de los comparadores
ICx	Capturadores de entrada
PAI	Acumulador de pulsos

Por defecto los recursos internos asociados a los pines del Puerto A están "desconectados". El 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.

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.

2.6.2. PUERTO B

La figura 17 muestra los 8 bits del Puerto B que 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.

DIRECCIÓN $1004

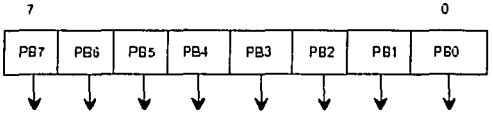


Figura 17: El puerto B

La figura 18 muestra todos los usos de los bits del Puerto B.

DIRECCIÓN $1004

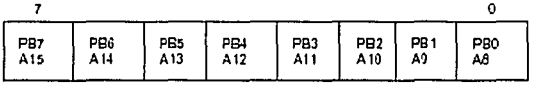


Figura 18: Todos los usos de los bits del puerto B

PB0-PB7 Bits de salida.

A8-A15 Señal de la parte alta del bus de direcciones (A).

2.6.3. PUERTO C

Es un puerto de entrada / salida. En el modo no expandido sus 8 bits pueden actuar como entradas o salidas independientes, según cómo se configuren los bits en el registro **DDRC** ($1007).

Un cero en un bit del registro DDRC configura el pin correspondiente para entrada. Un uno lo hace para salida.

La dirección del Puerto C es $1003, ver figura 19.

DDRC
DIRECCIÓN
$1007

0 = Entrada
1 = Salida

PUERTO C
DIRECCIÓN
$1003

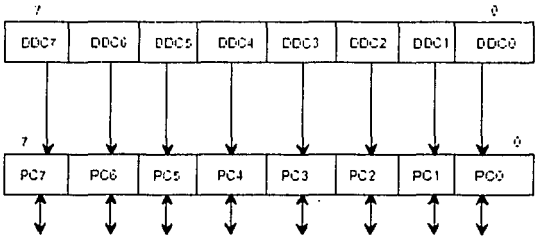


Figura 19: Puerto C y registros de configuración del puerto C (DDRC)

Si el microcontrolador está funcionando en el modo expandido, el Puerto C (PCx) actúa como parte bus de direcciones (Ax) multiplexado con el bus de datos (Dx).

La figura 20 muestra todos los usos de los pines del Puerto C.

DIRECCIÓN
$1003

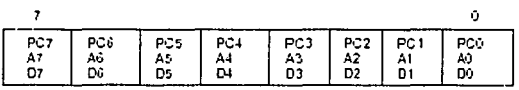


Figura 20: Usos de los bits del puerto C

AD0-AD7 Señales de los buses (A y D) multiplexados.

2.6.4. PUERTO D

DDRD
DIRECCIÓN
$1009
0 = Entrada
1 = Salida

PUERTO D
DIRECCIÓN $1008

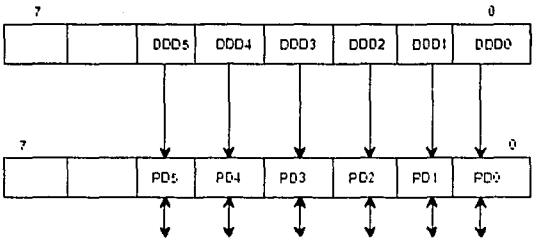


Figura 21: Puerto D y su configuración

Igual que el Puerto C, es un puerto de entrada / salida en el que se pueden configurar sus bits independientemente para entrada o salida. El registro de configuración **DDRD** se encuentra en la dirección $1009. Unos se corresponden con salidas y ceros con entradas. El puerto está mapeado en la dirección $1008. Este puerto es de sólo 6 bits. Está compartido con el SPI y el SCI, ver la figura 21.

DIRECCIÓN
PUERTO D
$1008

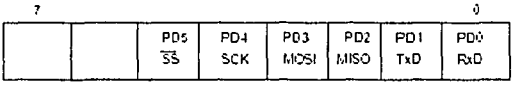


Figura 22: Otros usos de los bits del puerto D

Los bits 0 y 1 están compartidos con el puerto serie. Por ello, para poder ser utilizados es preciso SCI esté desactivado. Los bits 2,3,4 y 5 están compartidos por el SPI. Para que funcionen es necesario asegurarse que el SPI está desactivado.

Una de las características del Puerto D es que sus salidas pueden ser normales o de **colector abierto.** Esto se configura con el bit 5 del registro **SPCR** ($1028).

Si este bit está a 0, las salidas serán normales. Si está a 1, las salidas estarán en colector abierto. Por defecto están en colector abierto.

En la tarjeta CT6811 se debe seleccionar el modo con salidas en colector abierto puesto que existen resistencias de pull-up colocadas en algunas salidas. No se deben configurar como salidas normales.

2.6.5. PUERTO E

Es un puerto de 8 bits de entrada (PEx). Está situado en la dirección $100A. Ver figura 23.

DIRECCIÓN PUERTO E
$100A

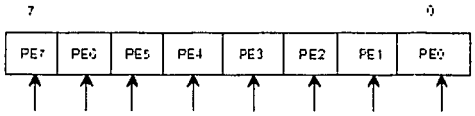


Figura 23: PUERTO E

La figura 24 muestra otros usos de los bits del Puerto E.

DIRECCIÓN PUERTO E
$100A

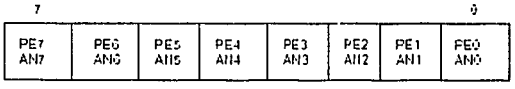


Figura 24: Otros usos de los bits del puerto E

AN0-AN7 Comparte pines con los 8 canales del conversor Analógico / Digital a los cuales puede acceder de 4 en 4.

Para poder utilizar el Puerto E como un puerto normal es preciso que el conversor Analógico / Digital interno del 68HC11 esté desconectado. Por defecto lo está.[6]

CAPÍTULO III.

"LENGUAJE ENSAMBLADOR PARA INTEL"

3.1. Información en las computadoras.

3.1.1 Unidades de información.

Para que la PC pueda procesar la información es necesario que ésta se encuentre en celdas especiales llamadas registros.

Los registros son conjuntos de 8 o 16 flip-flops (basculadores o biestables).

Un flip-flop es un dispositivo capaz de almacenar dos niveles de voltaje, uno bajo, regularmente de 0.5 volts y otro alto comúnmente de 5 volts. El nivel bajo de energía en el flip-flop se interpreta como apagado ó 0, y el nivel alto como prendido ó 1. A estos estados se les conoce usualmente como bits, que son la unidad mas pequeña de información en una computadora.

A un grupo de 16 bits se le conoce como palabra, una palabra puede ser dividida en grupos de 8 bits llamados bytes, y a los grupos de 4 bits les llamamos nibbles. [1]

3.1.2. Sistemas numéricos.

El sistema numérico que utilizamos a diario es el sistema decimal, pero este sistema no es conveniente para las máquinas debido a que la información se maneja codificada en forma de bits prendidos o apagados; esta forma de codificación nos lleva a la necesidad de conocer el cálculo posicional que nos permita expresar un número en cualquier base que lo necesitemos.

Convertir números binarios a decimales.

Trabajando en el lenguaje ensamblador nos encontramos con la necesidad de convertir números del sistema binario, que es el empleado por las computadoras, al sistema decimal utilizado por las personas.

El sistema binario está basado en únicamente dos condiciones o estados, ya sea encendido (1) o apagado (0), por lo tanto su base es dos.

Para la conversión podemos utilizar la formula de valor posicional:

Por ejemplo, si tenemos el numero binario 10011, tomamos de derecha a izquierda cada dígito y lo multiplicamos por la base elevada a la nueva posición que ocupan:

Binario:	**1**		**0**		**0**		**1**		**1**
Decimal:	**1*2^4**	**+**	**1*2^3**	**+**	**0*2^2**	**+**	**0*2^1**	**+**	**1*2^0**
=	**16**	**+**	**0**	**+**	**0**	**+**	**2**	**+**	**1 = 19 decimal.**

Convertir números decimales a binarios.

El método que se explicará utiliza la división sucesiva entre dos, guardando el residuo como dígito binario y el resultado como la siguiente cantidad a dividir.

Tomemos como ejemplo el número 43 decimal.

43/2 = 21 y su residuo es 1

21/2 = 10 y su residuo es 1

10/2 = 5 y su residuo es 0

5/2 = 2 y su residuo es 1

2/2 = 1 y su residuo es 0

1/2 = 0 y su residuo es 1

Armando el número de abajo hacia arriba tenemos que el resultado en binario es 101011. [2]

Sistema hexadecimal.

En la base hexadecimal tenemos 16 dígitos que van del 0 al 9 y de la letra A hasta la F (estas letras representan los números del 10 al 15).

Por lo tanto, contamos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

La conversión entre numeración binaria y hexadecimal es sencilla. Lo primero que se hace para una conversión de un número binario a hexadecimal es dividirlo en grupos de 4 bits, empezando de derecha a izquierda.

En caso de que el último grupo (el que quede más a la izquierda) sea menor de 4 bits se rellenan los faltantes con ceros.

Tomando como ejemplo el número binario 101011 lo dividimos en grupos de 4 bits y nos queda:

10 1011

Rellenando con ceros el último grupo (el de la izquierda):

0010 1011

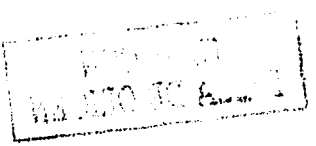
Después tomamos cada grupo como un número independiente y consideramos su valor en decimal:

0010 = 2 1011 = 11 = B

Con lo que obtenemos:

2BH (Donde la H representa la base hexadecimal), esto es de suma importancia debido a que el ensamblador para INTEL utilizan esta nomenclatura y para MOTOROLA se utiliza $.

Para convertir un número de hexadecimal a binario solo es necesario invertir estos pasos: se toma el primer dígito hexadecimal y se convierte a binario, y luego el segundo, y así sucesivamente hasta completar el número. [3]



3.2. Memoria Interna.

La C. P. U. esta compuesta por 2 tipos de memoria: RAM y ROM. Los bytes en memoria se numeran en forma consecutiva iniciando con 0000; de modo que cada dirección es única para cada localidad de memoria teniendo la siguiente distribución:

- De la dirección 0000 a la dirección A0000 se encuentra la memoria convencional con longitud de 640 kb.
- De la dirección A0001 a la dirección C0000 se encuentra la memoria de video con una longitud de 128 kb.
- De la dirección C0001 a la dirección F0000 se encuentra la memoria de expansión ROM. [4]

3.3. Segmentos de la Memoria.

Al iniciar el sistema operativo divide la memoria convencional en 3 segmentos:

1) Segmento de Código (CS)

Contiene las instrucciones de máquina que son ejecutadas por el sistema operativo o en algunas ocasiones por el BIOS (Basic Input Output System). El CS con ayuda del registro IP (Pointer Instruction) ejecuta cada una de las instrucciones contenidas en el segmento de código.

2) Segmento de Datos (DS)

El propósito del DS es almacenar los datos, constantes y áreas de trabajo definidos por el programa, en algunas casos requiere de 1 o más registro para realizar su labor de almacenamiento de datos, en su mayoría utiliza el registro DX o ES.

3) Segmento de Pila (PS)

Tiene el objetivo de almacenar los datos y direcciones que se necesitan guardar temporalmente como resultado de la manipulación o alguna operación de memoria o para uso de algunas llamadas a subrutinas.

3.4. Registros

Los registros en el microprocesador se emplean para controlar instrucciones en ejecución, manejar direccionamientos de memoria y proporcionar capacidad aritmética. Los registros son direccionables por medio de nombres.

La CPU tiene 14 registros internos, cada uno de 16 bits. Los primeros cuatro: AX, BX, CX, y DX son registros de uso general y también pueden ser utilizados como registros de 8 bits, para utilizarlos como tales es necesario referirse a ellos como por ejemplo: AH y AL, que son los bytes alto (high) y bajo (low) del registro AX. Esta nomenclatura es aplicable también a los registros BX, CX y DX.

REGISTRO CS: Tiene longitud de 16 bits y facilita un área de memoria para direccionamiento conocida como el segmento actual. El sistema operativo almacena la dirección inicial de CS de un programa en el registro CS. Ésta dirección de registro más un valor de desplazamiento nos proporciona la dirección actual de la instrucción que se ejecutará, almacenada ésta en el registro IP.

REGISTRO DS: La dirección inicial de un segmento de datos es almacenada en el registro DS, ésta dirección más un desplazamiento dará como resultado una localidad de memoria de un dato específico.

REGISTRO SS: Permite la colocación en memoria de una pila para almacenamiento temporal de direcciones y datos. El sistema operativo almacena la dirección de inicio del segmento de pila de un programa en el registro SS, ésta dirección más un valor de desplazamiento dado en el registro SP nos proporciona el dato o palabra actual que esta siendo direccionada.

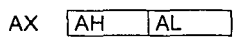
REGISTRO ES: En algunos casos las operaciones con caracteres se pueden realizar con dicho registro, el cual coordinado con el segmento de pila realiza la función de extracción de datos.

REGISTRO SP: Éste registro esta asociado con el registro SS y proporciona un valor de desplazamiento que se refiere al carácter o dato que será accesado en la pila.

REGISTRO BP: Éste registro se utiliza como un apuntador a los datos en relación a las localidades almacenadas ubicadas como parámetros. Comúnmente se utiliza cuando se realiza una interfaz con algún otro lenguaje de programación, el cual almacena los parámetros de entrada en el pila.

REGISTROS DE PROPÓSITO GENERAL: Son aquellos registros que a diferencia de los anteriores pueden estar formados por dos partes cada una de 8 bits, el cual conforma una parte alta y una parte baja, creados con la finalidad de almacenar datos de tipo byte. Éstos registros son: **AX, BX, CX, DX.**

REGISTRO AX: Es el registro llamado acumulador, es utilizado en operaciones que implican entrada y salida de datos y en instrucciones que realizan operaciones aritméticas como los son multiplicar y dividir, además tiene la ventaja de almacenar direcciones de 16 bits.



AH: Acumulator High (Parte alta)
AL: Acumulator Low (Parte baja.)

REGISTRO BX: El BX es conocido como el registro base ya que es el único registro de propósitos generales que pueden ser unos índices para direccionamiento indexado. También es común emplear al BX para cálculos.

REGISTRO CX: El CX es conocido como el registro contador. Puede contener un valor para controlar el número de veces que un ciclo se repite o un valor para corrimiento de bits, hacia la derecha o hacia la izquierda. El CX también es usado para muchos cálculos.

REGISTRO DX: El DX es conocido como el registro de datos. Algunas operaciones de entrada / salida requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen al DX y al AX trabajando juntos. Puede usar los registros de propósitos para suma y resta de cifras de 8, 16, 32 bits. [5]

3.5. Proceso de creación de un programa.

Para la creación de un programa es necesario seguir cinco pasos: Diseño del algoritmo, codificación del mismo, su traducción a lenguaje máquina, la prueba del programa y la depuración.

En la etapa de diseño se plantea el problema a resolver y se propone la mejor solución, creando diagramas esquemáticos utilizados para el mejor planteamiento de la solución.

La codificación del programa consiste en escribir el programa en algún lenguaje de programación; en este caso específico en ensamblador, tomando como base la solución propuesta en el paso anterior.

La traducción al lenguaje máquina es la creación del programa objeto, esto es, el programa escrito como una secuencia de ceros y unos que pueda ser interpretado por el procesador.

La prueba del programa consiste en verificar que el programa funcione sin errores, o sea, que haga lo que tiene que hacer.

La última etapa es la eliminación de las fallas detectadas en el programa durante la fase de prueba. La corrección de una falla normalmente requiere la repetición de los pasos comenzando desde el primero o el segundo.

Para crear un programa en ensamblador existen varias opciones, la primera es utilizar el MASM (Macro Assembler, de Microsoft), la segunda es utilizar el debugger, la tercera es utilizar el C++, el cual permite utilizar el editor de texto para escribir el programa y ensamblarlo en el mismo, y la cuarta es utilizar IASM11 que es el editor de textos de HC11 (Motorola).

Debug solo puede crear archivos con extensión .COM, y por las características de este tipo de programas no pueden ser mayores de 64 kb, además deben comenzar en el desplazamiento, offset, o dirección de memoria 0100H dentro del segmento específico.

Es posible visualizar los valores de los registros internos de la CPU utilizando el programa Debug. [6]

3.6. Las instrucciones.

En el lenguaje ensamblador las líneas de código constan de dos partes, la primera es el nombre de la instrucción que se va a ejecutar y la segunda son los parámetros del comando u operandos. Por ejemplo:

add ah, bh

Aquí "add" es el comando a ejecutar (en este caso una adición) y tanto "ah" como "bh" son los parámetros.

3.6.1. Movimiento de datos

En todo programa es necesario mover datos en la memoria y en los registros de la CPU; existen diversas formas de hacer esto: puede copiar datos de la memoria a algún registro, de registro a registro, de un registro a una pila, de la pila a un registro, transmitir datos hacia dispositivos externos así como recibir datos de dichos dispositivos.

Este movimiento de datos está sujeto a reglas y restricciones. Algunas de ellas son las que se citan a continuación.

No es posible mover datos de una localidad de memoria a otra directamente, es necesario primero mover los datos de la localidad origen hacia un registro y luego del registro a la localidad destino.

No se puede mover una constante directamente a un registro de segmentos, primero se debe mover a un registro de la CPU.

Es posible mover bloques de datos por medio de las instrucciones **movs**, que copia una cadena de bytes o palabras; **movsb** que copia n bytes de una localidad a otra; y **movsw** copia n palabras de una localidad a otra. Las dos últimas instrucciones toman los valores de las direcciones definidas por DS:SI como grupo de datos a mover y ES:DI como nueva localización de los datos.

Para mover los datos también existen las estructuras llamadas pilas, en este tipo de estructuras los datos se introducen con la instrucción **push** y se extraen con la instrucción **pop**

En una pila el primer dato introducido es el último que podemos sacar, esto es, si en el programa se utilizan las instrucciones:

PUSH AX
PUSH BX
PUSH CX

Para devolver los valores correctos a cada registro al momento de sacarlos de la pila es necesario hacerlo en el siguiente orden:

POP CX
POP BX
POP AX

Entrada / Salida de datos.

Para la comunicación con dispositivos externos se utilizan el comando **out** para mandar información a un puerto y el comando **in** para leer información recibida desde algún puerto.

Ej. **OUT DX,AX**

Donde DX contiene el valor del puerto que se utilizará para la comunicación y AX contiene la información que se mandará.

Ej. **IN AX,DX**

Donde AX es el registro donde se guardará la información que llegue y DX contiene la dirección del puerto por donde llegará la información. [7]

3.6.2. Operaciones lógicas.

Las instrucciones de las operaciones lógicas son: **and**, **not**, **or** y **xor**, éstas trabajan sobre los bits de sus operandos. Para verificar el resultado de operaciones recurrimos a las instrucciones **cmp** y **test**.

TABLAS DE OPERACIONES LÓGICAS

AND

Fuente	Destino	Destino
1	1	1
1	0	0
0	1	0
0	0	0

Fuente	Destino	Destino
1	1	1
1	0	1
0	1	1
0	0	0

XOR

Fuente	Destino	Destino
1	1	0
1	0	1
0	1	1
0	0	0

AND: Realiza la conjunción de los operandos bit por bit.

Ej. **AND destino, fuente**

Con esta instrucción se lleva a cabo la operación "y" lógica de los dos operandos:

El resultado de la operación se almacena en el operando destino.

NEG: Genera el complemento a 2

Ej. **NEG destino**

Esta instrucción genera el complemento a 2 del operando destino y lo almacena en este mismo operando. Por ejemplo, si AX guarda el valor de 1234H, entonces:

NEG AX ;Nos dejaría almacenado en el registro AX el valor EDCCH.

NOT: Lleva a cabo la negación bit por bit del operando destino.

Ej. **NOT destino** ;El resultado se guarda en el mismo operando destino.

OR : OR inclusivo lógico

Ej. **OR destino, fuente**

TEST: Comparar lógicamente los operandos. Realiza una conjunción, bit por bit, de los operandos, pero a diferencia de AND esta instrucción no coloca el resultado en el operando destino, solo tiene efecto sobre el estado de las banderas.

Ej. **TEST destino, fuente**

XOR: OR exclusivo

Ej. **XOR destino, fuente** [8]

3.6.3. Operaciones aritméticas.

Las instrucciones utilizadas para las operaciones algebraicas son: para sumar **add**, para restar **sub**, para multiplicar **mul** y para dividir **div**.

ADD : Suma los dos operandos y guarda el resultado en el operando destino.

Ej. **ADD destino, fuente**

ADC : Adición con acarreo. Lleva a cabo la suma de dos operandos y suma uno al resultado en caso de que la bandera CF esté activada, esto es, en caso de que exista acarreo. El resultado se guarda en el operando destino.

Ej. **ADC destino, fuente**

DIV: División sin signo. El divisor puede ser un byte o palabra y es el operando que se le da a la instrucción.

Ej. **DIV fuente**

IDIV: División con signo

Ej. **IDIV fuente**

MUL: Multiplicación sin signo

Ej. **MUL fuente**

IMUL: Multiplicación de dos enteros con signo.

Ej. **IMUL fuente**

SBB: Substracción con acarreo. Esta instrucción resta los operandos y resta uno al resultado si CF está activada. El operando fuente siempre se resta del destino. Este tipo de substracción se utiliza cuando se trabaja con cantidades de 32 bits.

Ej. **SBB destino, fuente**

SUB: Substracción. Resta el operando fuente del destino.

Ej. **SUB destino, fuente** [9]

3.6.4. Saltos, ciclos y procedimientos.

Los saltos incondicionales en un programa escrito en lenguaje ensamblador están dados por la instrucción **jmp**, un salto es alterar el flujo de la ejecución de un programa enviando el control a la dirección indicada.

Un ciclo, conocido también como iteración, es la repetición de un proceso un cierto número de veces hasta que alguna condición se cumpla. En estos ciclos se utilizan los brincos "condicionales" basados en el estado de las banderas. Por ejemplo la instrucción **jnz** que salta solamente si el resultado de una operación es diferente de cero y la instrucción **jz** que salta si el resultado de la operación es cero.

Por último tenemos los procedimientos o rutinas, que son una serie de pasos que se usarán repetidamente en el programa y en lugar de escribir todo el conjunto de pasos únicamente se les llama por medio de la instrucción **call**.

Un procedimiento en ensamblador es aquel que inicie con la palabra **Proc** y termine con la palabra **ret**.

Existen 2 tipos de procedimientos:

Proc Far : procedimiento lejano, generalmente utilizado para el programa principal.

Proc Near : procedimiento cercano, generalmente utilizado para realizar subrutinas.

Realmente lo que sucede con el uso de la instrucción **call** es que se guarda en la pila el registro **IP** y se carga la dirección del procedimiento en el mismo registro, conociendo que IP contiene la localización de la siguiente instrucción que ejecutara la CPU, entonces podemos darnos cuenta que se desvía el flujo del programa hacia la dirección especificada en este registro. Al momento en que se llega a la palabra **ret** se saca de la pila el valor de IP con lo que se devuelve el control al punto del programa donde se invocó al procedimiento.

JMP: Salto incondicional. Esta instrucción se utiliza para desviar el flujo de un programa sin tomar en cuenta las condiciones actuales de las banderas ni de los datos.

Ej. **JMP destino**

JA (JNBE): Brinco condicional. Después de una comparación este comando salta si está arriba o salta si no está abajo o si no es igual.

Ej. **JA Etiqueta**

JAE (JNB): Salto condicional. Salta si está arriba o si es igual o salta si no está abajo.

Ej. **JAE etiqueta**

JB (JNAE): Salto condicional. Salta si está abajo o salta si no está arriba o si no es igual.

Ej. **JB etiqueta**

JBE (JNA): Salto condicional. Salta si está abajo o si es igual o salta si no está arriba.

Ej. **JBE etiqueta**

Instrucción JE (JZ)

JE(JZ): Salto condicional. Salta si es igual o salta si es cero.

Ej. **JE etiqueta**

JNE (JNZ): Salto condicional. Salta si no es igual o salta si no es cero.

Ej. **JNE etiqueta**

JG (JNLE): Salto condicional, se toma en cuenta el signo. Salta si es más grande o salta si no es menor o igual.

Ej. **JG etiqueta**

JGE (JNL): Salto condicional, se toma en cuenta el signo. Salta si es más grande o igual o salta si no es menor que.

Ej. **JGE etiqueta**

JL (JNGE): Salto condicional, se toma en cuenta el signo. Salta si es menor que o salta si no es mayor o igual.

Ej. **JL etiqueta**

JLE (JNG): Salto condicional, se toma en cuenta el signo.

Ej. **JLE etiqueta**

Salta si es menor o igual o salta si no es más grande.

JC: Salto condicional, se toman en cuenta las banderas. Salta si hay acarreo.

Ej. **JC etiqueta**

JNC: Salto condicional, se toma en cuenta el estado de las banderas. Salta si no hay acarreo.

Ej. **JNC etiqueta**

JNO: Salto condicional, se toma en cuenta el estado de las banderas. Salta si no hay desbordamiento.

Ej. **JNO etiqueta**

LOOP: Generar un ciclo en el programa. La instrucción loop decrementa CX en 1, y transfiere el flujo del programa a la etiqueta dada como operando si CX es diferente a 1.

Ej. **LOOP etiqueta**

DEC: Decrementar el operando. Esta operación resta 1 al operando destino y almacena el nuevo valor en el mismo operando.

Ej. **DEC destino**

INC: Incrementar el operando. La instrucción suma 1 al operando destino y guarda el resultado en el mismo operando destino.

Ej. **INC destino**

CMP: Comparar los operandos.

Ej. **CMP destino, fuente**

CLC: Limpiar bandera de acarreo. Esta instrucción apaga el bit correspondiente a la bandera de acarreo, o sea, lo pone en cero.

Ej. **CLC** [10]

3.7. Interrupciones.

Definición de interrupción:

> Una interrupción es una instrucción que detiene la ejecución de un programa para permitir el uso de la CPU a un proceso prioritario. Una vez concluido este último proceso se devuelve el control a la aplicación anterior.

Las interrupciones ocurren muy seguido, sencillamente la interrupción que actualiza la hora del día ocurre aproximadamente 18 veces por segundo. Para lograr administrar todas estas interrupciones, la computadora cuenta con un espacio de memoria, llamado memoria baja, donde se almacenan las direcciones de cierta localidad de memoria donde se encuentran un juego de instrucciones que la CPU ejecutará para después regresar a la aplicación en proceso.

Interrupciones internas de hardware.

Las interrupciones internas son generadas por ciertos eventos que surgen durante la ejecución de un programa.

Este tipo de interrupciones son manejadas en su totalidad por el hardware y no es posible modificarlas.

Un ejemplo claro de este tipo de interrupciones es la que actualiza el contador del reloj interno de la computadora, el hardware hace el llamado a esta interrupción varias veces durante un segundo para mantener la hora actualizada.

Interrupciones externas de hardware.

Las interrupciones externas las generan los dispositivos periféricos, como pueden ser: teclado, impresoras, tarjetas de comunicaciones, etc. También son generadas por los coprocesadores.

Interrupciones de software.

Las interrupciones de software pueden ser activadas directamente por el ensamblador invocando al número de interrupción deseada con la instrucción INT.

El uso de las interrupciones nos ayuda en la creación de programas, utilizándolas nuestros programas son más cortos, es más fácil entenderlos y usualmente tienen un mejor desempeño debido en gran parte a su menor tamaño.

Este tipo de interrupciones podemos separarlas en dos categorías: las interrupciones del sistema operativo DOS y las interrupciones del BIOS.

La diferencia entre ambas es que las interrupciones del sistema operativo son más fáciles de usar pero también son más lentas ya que estas interrupciones hacen uso del BIOS para lograr su cometido, en cambio las interrupciones del BIOS son mucho más rápidas pero tienen la desventaja que, como son parte del hardware son muy específicas y pueden variar dependiendo incluso de la marca del fabricante del circuito.

La elección del tipo de interrupción a utilizar dependerá únicamente de las características que le quiera dar a su programa: velocidad (utilizando las del BIOS) o portabilidad (utilizando las del DOS).

Interrupción 21H : Llamar a diversas funciones del DOS.

Ej. **Int 21H**

Esta interrupción tiene varias funciones, para accesar a cada una de ellas es necesario que en el registro AH se encuentre el número de función que se requiera al momento de llamar a la interrupción.

Funciones para desplegar información al video.

02H Exhibe salida
09H Impresión de cadena (video)
40H Escritura en dispositivo / Archivo

Funciones para leer información del teclado.

01H Entrada desde teclado
0AH Entrada desde teclado usando buffer
3FH Lectura desde dispositivo / archivo

Interrupción 10H: Llamar a diversas funciones de video del BIOS.

Ej. **Int 10H**

Esta interrupción tiene diversas funciones, todas ellas nos sirven para controlar la entrada y salida de video, la forma de acceso a cada una de las opciones es por medio del registro AH.

Funciones comunes de la interrupción 10H.

02H Selección de posición del cursor
09H Escribe atributo y carácter en el cursor
0AH Escribe carácter en la posición del cursor
0EH Escritura de caracteres en modo alfanumérico

Interrupción 16H: Manejar la entrada / salida del teclado.

Ej. **Int 16H**

Opciones de la interrupción 16H, las cuales son llamadas utilizando el registro AH.

Funciones de la interrupción 16H

00H Lee un carácter de teclado
01H Lee estado del teclado

Interrupción 17H: Manejar la entrada / salida de la impresora.

Ej. **Int 17H**

Esta interrupción es utilizada para escribir caracteres a la impresora, inicializarla y leer su estado.

Funciones de la interrupción 16H

00H Imprime un carácter ASCII
01H Inicializa la impresora
02H Proporciona el estado de la impresora. [11]

3.8. Software necesario.

Para poder crear un programa se requieren varias herramientas:

1) Un editor para crear el programa fuente (Bloc de notas, Edit).
2) Un compilador que no es mas que un programa que "traduce" el programa fuente a un programa objeto (MASM,TASM).
3) Un enlazador o linker, que genere el programa ejecutable a partir del programa objeto (LINK, TLINK).

La extensión usada para que MASM reconozca los programas fuente en ensamblador es .ASM ; una vez traducido el programa fuente, el MASM crea un archivo con la extensión .OBJ, este archivo contiene un "formato intermedio" del programa, llamado así porque aún no es ejecutable pero tampoco es ya un programa en lenguaje fuente. El enlazador genera, a partir de un archivo .OBJ o la combinación de varios de estos archivos, un programa ejecutable, cuya extensión es usualmente .EXE aunque también puede ser .COM, dependiendo de la forma en que se ensambló.

Utilización del MASM

Una vez que se creó el programa objeto se debe pasar al MASM para crear el código intermedio, el cual queda guardado en un archivo con extensión .OBJ. El comando para realizar esto es:

MASM Nombre_Archivo.asm; [Enter]

Donde Nombre_Archivo es el nombre del programa fuente con extensión .ASM que se va a traducir. El punto y coma utilizados después del nombre del archivo le indican al macro ensamblador que genere directamente el código intermedio, de omitirse este carácter el MASM pedirá el nombre del archivo a traducir, el nombre del archivo que se generará así como opciones de listado de información que puede proporcionar el traductor.

Uso del enlazador (linker)

El MASM únicamente puede crear programas en formato .OBJ, los cuales no son ejecutables por si solos, es necesario un enlazador que genere el código ejecutable. La utilización del enlazador es muy parecida a la del MASM.

LINK *Nombre_Archivo*.obj ;

Donde Nombre_Archivo es el nombre del programa intermedio (OBJ). Esto generara directamente un archivo con el nombre del programa intermedio y la extensión .EXE [12]

3.9. Formato interno de un programa.

Para poder comunicarnos en cualquier lenguaje, incluyendo los lenguajes de programación, es necesario seguir un conjunto de reglas, de lo contrario no podríamos expresar lo que deseamos.

A continuación se presentan algunas de las reglas que debemos seguir para escribir un programa en lenguaje ensamblador, enfocándonos a la forma de escribir las instrucciones para que el ensamblador sea capaz de interpretarlas.

Básicamente el formato de una línea de código en lenguaje ensamblador consta de cuatro partes:

Etiqueta, variable o constante: No siempre es definida, si se define es necesario utilizar separadores para diferenciarla de las otras partes, usualmente espacios, o algún símbolo especial.

Directiva o instrucción: es el nombre con el que se conoce a la instrucción que queremos que se ejecute.

Operando(s): la mayoría de las instrucciones en ensamblador trabajan con dos operandos, aunque hay instrucciones que funcionan solo con uno. El primero normalmente es el operando destino, que es el depósito del resultado de alguna operación; y el segundo es el operando fuente, que lleva el dato que será procesado. Los operandos se separan uno del otro por medio de una coma ",".

Comentario: como su nombre lo indica es tan solo un escrito informativo, usado principalmente para explicar que está haciendo el programa en determinada línea; se separa de las otras partes por medio de un punto y coma ";". Esta parte no es necesaria en el programa, pero nos ayuda a depurar el programa en caso de errores o modificaciones.

Como ejemplo podemos ver una línea de un programa escrito en ensamblador:

MOV AX,001AH ; Inicializa AX con el valor 001A

La instrucción "MOV", y los operandos "AX" como destino y "001A" como fuente, además del comentario que sigue después del " ; ". [13]

3.10. Formato Externo de un programa.

Además de definir ciertas reglas para que el ensamblador pueda entender una instrucción es necesario darle cierta información de los recursos que se van a utilizar, como por ejemplo los segmentos de memoria que se van a utilizar, datos iniciales del programa y también donde inicia y donde termina nuestro código.

Un programa sencillo puede ser el siguiente:

```
.MODEL SMALL
.STACK
.CODE
Programa:
MOV AX,4C00H
INT 21H
END Programa
```

El programa realmente no hace nada, únicamente coloca el valor 4C00H en el registro AX, para que la interrupción 21H termine el programa, pero nos da una idea del formato externo en un programa de ensamblador.

La directiva .MODEL define el tipo de memoria que se utilizará; la directiva .STACK le pide al ensamblador que reserve un espacio de memoria para las operaciones de la pila; la directiva .CODE nos indica que lo que esta a continuación es nuestro programa; la etiqueta Programa indica al ensamblador el inicio del programa; la instrucción END Programa marca el final del programa.

Ejemplo práctico de un programa

Aquí se ejemplificará un programa que escriba una cadena en pantalla:

```
.MODEL SMALL
.STACK 100h
.CODE
Programa:
MOV AX, @DATA
MOV DS, AX
MOV DX, Offset Texto
MOV AH, 9
INT 21H
MOV AX,4C00H
INT 21H
.DATA
Texto DB 'Mensaje en pantalla.$'
END Programa
```

Se define el modelo de memoria, se indica donde inicia el código del programa y en donde comienzan las instrucciones.

A continuación se coloca @DATA en el registro AX para después pasarlo al registro DS ya que no se puede copiar directamente una constante a un registro de segmento. El contenido de @DATA es el número del segmento que será

utilizado para los datos. Luego se guarda en el registro DX un valor dado por "Offset Texto" que nos da la dirección donde se encuentra la cadena de caracteres en el segmento de datos. Luego utiliza la opción 9 (Dada por el valor de AH) de la interrupción 21H para desplegar la cadena posicionada en la dirección que contiene DX. Por último utiliza la opción 4CH de la interrupción 21H para terminar la ejecución del programa (aunque cargamos al registro AX el valor 4C00H la interrupción 21H solo toma como opción el contenido del registro AH).

La directiva .DATA le indica al ensamblador que lo que está escrito a continuación debe almacenarlo en el segmento de memoria destinado a los datos.

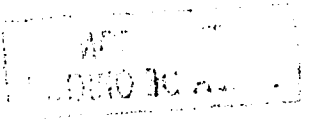
La directiva DB es utilizada para Definir Bytes, esto es, asignar a cierto identificador (en este caso "Texto") un valor, ya sea una constante o una cadena de caracteres, en este último caso deberá estar entre comillas sencillas ' y terminar con el símbolo "$". [14]

CAPÍTULO IV.

"APLICACIONES EN PROCESOS INDUSTRIALES"

4.1. Control de procesos.

El microcontrolador y el microprocesador procesan señales binarias de entrada y las convierte en señales de salida; con estas señales se pueden controlar directamente secuencias mecánicas, proceso fabriles totales o parciales, etcétera.



Los posibles campos de aplicación de un microprocesador o microcontrolador son casi innumerables, pero se utilizan principalmente para las siguientes funciones:

a) **Control de procesos.** En ésta función, el microprocesador o microcontrolador se encarga de que cada paso o fase del proceso sea efectuado en el orden cronológico correcto y sincronizado.

Un buen ejemplo para la función señalada es un sistema transportador en una cadena de producción automatizada. En este caso, los pasos equivalen a los correspondientes recorridos o desplazamientos parciales de la pieza, de una fase de manipulación a otra. Aquí, el microprocesador o microcontrolador se ocupa de controlar todos los electromotores (por ejemplo: la velocidad de la cadena) y todos los elementos hidráulicos o neumáticos (por ejemplo: desviador pivotante) de la instalación.

El microprocesador o microcontrolador vela porque las piezas sean conducidas debidamente a través del taller, acatándose con precisión las fases de su elaboración fabril, ver figura 25.

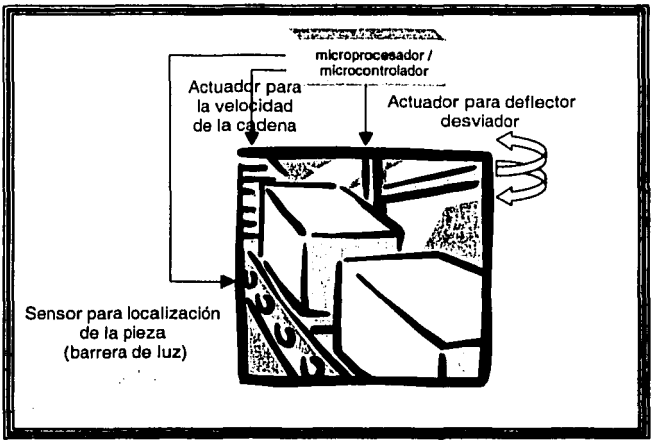


Figura 25. Producción Automatizada.

b) **Visualización de instalaciones.** En este caso el microprocesador o microcontrolador verifica automáticamente ciertas condiciones de la instalación (por ejemplo: temperaturas, presiones, niveles). Cuando en su comprobación, el control registra un exceso en sus coeficientes máximos o mínimos de los parámetros, actúa en dos formas: adopta las medidas necesarias para evitar deterioros o desperfectos, o emite señales de aviso para el personal de servicio.

Un ejemplo para esta función es la depuradora, como se ilustra en la figura 26. Aquí, tanto en los tanques de depuración como en las tuberías se han incorporado sensores para tomar ciertos parámetros (por ejemplo: niveles de agua). El microprocesador o microcontrolador verifica constante y automáticamente los estados reales que registra, los compara frente a los parámetros memorizados y controla los correspondientes caudales en las tuberías. En función de los valores que registre en lugares precisos y bajo las condiciones establecidas, avisa correspondientemente al operario de servicio.

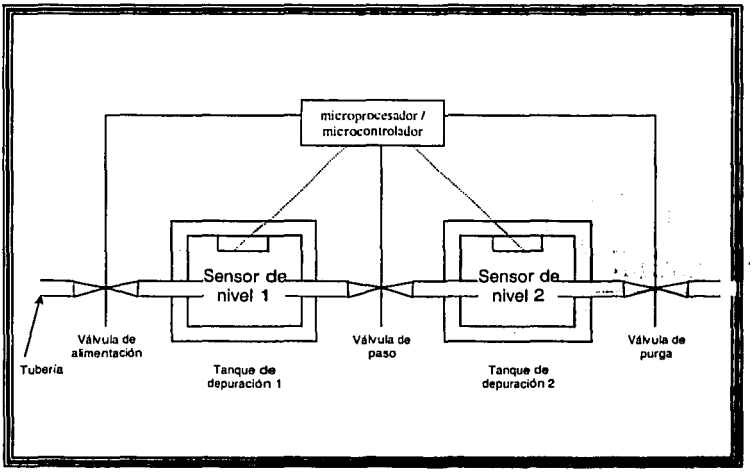


Figura 26. Verificación automatizada de condiciones de la instalación.

c) **Control de puesta a punto para máquinas CNC.** Las máquinas herramientas modernas casi siempre están dotadas de un control numérico computarizado (CNC). El tornero o fresador ya no pone a punto su máquina ajustando manivelas y tornillos. En lugar de ello, programa un control numérico computarizado. Este se encarga entonces de realizar automáticamente los ajustes precisos para trabajar la pieza correspondiente. Pero para que el CNC y la máquina herramienta se "enciendan", es preciso integrar un microprocesador o microcontrolador, que se encarga de la conmutación entre ambos equipos.

Ejemplo: En el programa de un control numérico programable figura la instrucción "activar agente refrigerante".

Cuando durante un ciclo del programa, el control llega a esta instrucción, emite cierta señal al microprocesador o microcontrolador; éste activa, por una parte, todos los grupos de refrigeración y, por otra, se encarga de que todas las demás funciones secundarias (por ejemplo: activación de indicaciones) sean efectuadas debidamente y de ser posibles fallos o averías sean detectados y visualizados inmediatamente. Ver figura 27.

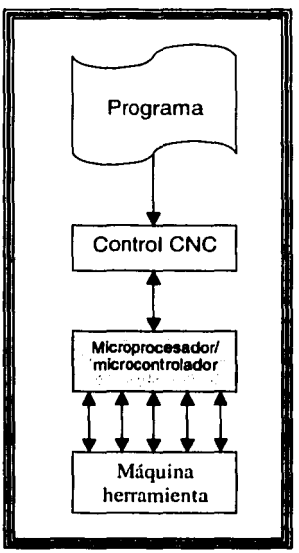


Figura 27. Control de puesta a punto para máquinas CNC.

4.2. Componentes de un sistema industrial.

Según el problema técnico que se tenga que resolver con un microprocesador o microcontrolador, la configuración de éste puede ser más o menos compleja. Independientemente del grado de complejidad de la aplicación, el equipo consta de los siguientes componentes esenciales:

HARDWARE.

Por hardware se entienden los grupos electrónicos. Estos se encargan de activar o desactivar las funciones controlables de la instalación o maquinaria en función de una secuencia lógica determinada.

Los datos que se procesan y memoriza la CPU son señales binarias. Estas se componen respectivamente siempre de un bit (estado cero (inactivo) o estado 1 (activo))o una palabra llamada byte.

Los puertos de entradas y salidas establecen la comunicación entre el microprocesador o el microcontrolador y los sensores / actuadores. Cada uno de estos módulos esta dotado de un número determinado de entradas y/o salidas.

SOFTWARE.

Por software se entienden los programas. Estos determinan los enlaces lógicos y, por consiguiente, la activación o desactivación, o sea el mando, de los grupos controlables en la instalación o maquinaria. El software, están almacenados en una memoria (hardware) propia y especial (parte del microcontrolador en el caso de Motorola), de la cual pueden ser recuperados y, en su caso, modificados en cualquier momento dado (configuración Especial Test y posteriormente grabar el software con las modificaciones en la EEPROM para Motorola, en el caso de Intel, basta con realizar las modificaciones en el código fuente y ensamblar). Al modificar el programa se altera también la secuencia del mando. Una modificación o cambio de software no implica un cambio del hardware, lo que repercute en reducción e incluso eliminación de costos.

Las características de software se explicaron en los capítulos II y III respectivamente.

SENSORES.

La técnica de los sensores abarca todos los grupos o dispositivos sobre la instalación o maquinaria controlable, que se encargan de comunicar al microcontrolador o microprocesador la información sobre estados de máquina.

Los sensores son transmisores de señales. El microcontrolador o el microprocesador utiliza los sensores para consultar estados en la instalación o en los equipos controlables. Éstos trabajan con electricidad; por ello las señales no

eléctricas tienen que ser convertidas (por los sensores) en señales eléctricas. De lo contrario, los puertos de entradas no sabrían interpretarlas.

Ejemplos de sensores:

Detectores de proximidad.

- Interruptores, pulsadores, conmutadores.

Iniciadores.

- Detectores con o sin contacto, que emiten señal 1 o señal 0, cuando se les aproxima un objeto. Transmisores inductivos reaccionan a piezas metálicas; los transmisores capacitivos reaccionan también a otros materiales.

Barreras fotoeléctricas.

- Detectores con o sin contacto, que emiten respectivamente señal 0 ó señal 1 cuando se interrumpe una barrera de luz (emisores / receptores infrarrojo).

Sensores térmicos.

- Detectores con o sin contacto, que emiten respectivamente señal 0 ó señal 1 cuando se llega a la temperatura ajustada.

ACTUADORES

Los actuadores abarcan todos los grupos sobre la instalación o maquinaria controlable, cuya actuación modifica los estados del software y/o hardware, es decir, modifica los procesos o indica alteración de los estados.

Los actuadores son elementos ejecutivos. Estos toman señales binarias de los puertos de salida o las convierten en señales para otras formas de energía.

Se distingue actuadores eléctricos, electrónicos, electrohidráulicos y electroneumáticos. Estos elementos pueden generar conmutaciones así como desplazamientos lineales o rotaciones.

Ejemplos de actuadores:

Dispositivos de indicación.

- Lámparas piloto, zumbadores, timbres.

Cilindros neumáticos (con sistema de válvulas).

- Cilindros de simple o doble efecto, cilindros con vástago doble, cilindros tandem, cilindros multiposición.

Electromotores.

- Motores de corriente continua, motores (lentos) de posicionamiento, sincromotores de corriente alterna, motores aletas.

Actuadores hidráulicos (con sistema de válvulas).

- Cilindros de simple o doble efecto, válvulas reguladoras de caudal, motores hidráulicos.

Motores electro hidráulicos.

- Motores lentos, servo accionamientos.

EQUIPO PROGRAMADOR.

Con éste se elabora el software y se carga a la memoria. En la mayoría de los casos sirve para comprobación de los programas. En el caso de Motorola, se auxilia además de dos tipos de software, el primero, llamado **IASM11**, fue creado para editar, ensamblar y establecer comunicación con el microcontrolador 68HC11; el segundo, denominado **BUFFALO**, fue realizado para tener comunicación el programador con el microcontrolador cuando se esta en modo SPECIAL TEST.

En el caso de Intel, el código fuente se ensambla y enlaza para generar el programa ejecutable que se puede cargar en disco duro o en disco flexible para conectarlo al proceso por medio de sus puertos y tener la puesta en marcha.

4.3. Semáforo en un crucero (INTEL).

OBJETIVO. Controlar la duración de las luces de un semáforo en un crucero.

Suponga que se tiene 2 avenidas dispuestas como se muestra en la figura 28.

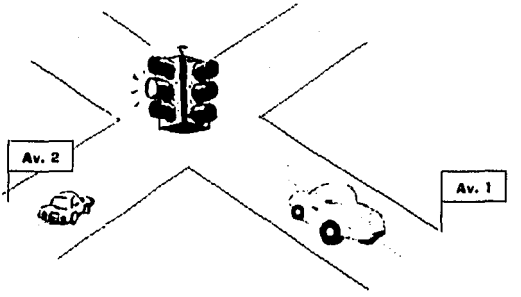


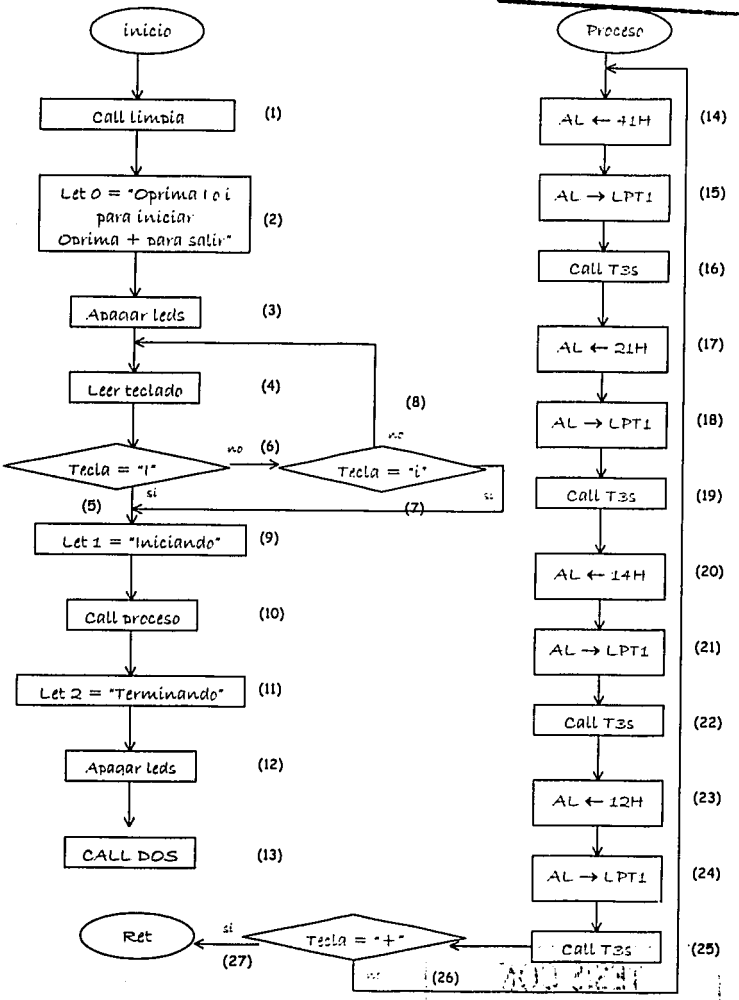
Figura 28. Semáforo en un crucero.

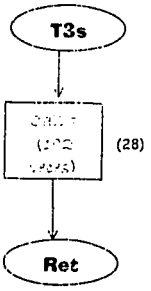
Se requiere programar la secuencia debida para el semáforo de tal forma que los autos tengan el tiempo suficiente para circular sin tener un accidente.

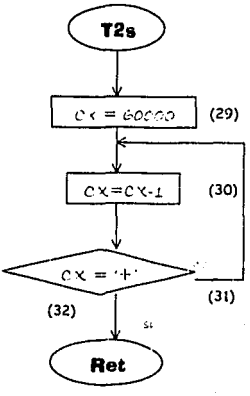
Entonces, se puede generar la siguiente tabla de tiempos, asignando a su vez, el número en Hexadecimal debido a la posición luminosa:

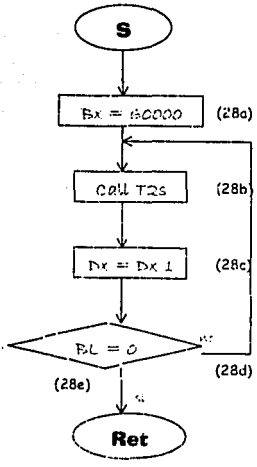
80	40	20	10	8	4	2	1	
Semáforo de la Av 2.				Semáforo de la Av 1.				
X	Verde	Amarillo	Rojo	X	Verde	Amarillo	Rojo	Hexadecimal
0	1	0	0	0	0	0	1	41h
0	0	1	0	0	0	0	1	21h
0	0	0	1	0	1	0	0	14h
0	0	0	1	0	0	1	0	12h

4.3.1. Diagrama de flujo.









4.3.2. Código fuente.

```
.model small
.STACK 100H
.data
LET0 db 'Oprima I o i para iniciar y oprima + para salir',13,10,'$'
LET1 db 'Iniciando ...',13,10,'$'
LET2 db 'Terminando ...',13,10,'$'

.code
        MOV AX,@DATA
        MOV DS,AX
        CALL LIMPIAR            (1)
        LEA DX,LETO             }
        CALL LETRERO            } (2)
        CALL APAGAR             (3)
NIVEL4:  CALL LEER              (4)
        CMP AL,'I'
        JE NIVEL7               (5)
        CMP AL,'i'              (6)
        JE NIVEL7               (7)
        JMP NIVEL4              (8)
NIVEL7:  LEA DX,LET1            (9)
        CALL PROCESO            } (10)
        LEA DX,LET2             }
        CALL LETRERO            } (11)
        CALL APAGAR             (12)
        CALL DOS                (13)

PROCESO PROC NEAR
NIVEL8: MOV AL,41H              (14)
        MOV DX,0378h            } (15)
        OUT DX,AL               }
        CALL T3S                (16)
        MOV AL,21H              (17)
        MOV DX,0378h            } (18)
        OUT DX,AL               }
        CALL T3S                (19)
        MOV AL,14H              (20)
        MOV DX,0378h            } (21)
        OUT DX,AL               }
        CALL T3S                (22)
        MOV AL,12H              (23)
        MOV DX,0378h            } (24)
        OUT DX,AL               }
        CALL T3S                (25)
        CALL LEER               }
        CMP AL,'+'              } (26)
        JNE NIVEL8              }
        RET                     (27)
PROCESO ENDP

T3S PROC NEAR
        CALL S                  }
        CALL S                  }
        CALL S                  } (28)
                                }
```

```
        CALL S
        CALL S
        ret
T3S ENDP

S PROC NEAR
        MOV BX,60000        (28a)
        CALL T2S            (28b)
X:      DEC BX              (28c)
        CMP BL,0            }
        JNE X               } (28d)
        RET                 (28e)
S ENDP

T2S PROC NEAR
        MOV CX,60000        (29)
N1:     DEC CX              (30)
        CMP CX,'+'          }
        JNE N1              } (31)
        RET                 (32)
T2S ENDP

LETRERO PROC NEAR
        MOV AH,09
        INT 21H
        RET
LETRERO ENDP

LEER PROC NEAR
        MOV AH,06
        MOV DL,0FFH
        INT 21H
        RET
LEER ENDP

APAGAR PROC NEAR
        MOV DX,0378H
        MOV AL,00H
        OUT DX,AL
        RET
APAGAR ENDP

DOS PROC NEAR
        MOV AH,4CH
        INT 21H
        RET
DOS ENDP

LIMPIAR PROC NEAR
        MOV AX,0600H
        MOV BH,14
        MOV CX,0000
        MOV DX,184FH
        INT 10H
        RET
LIMPIAR ENDP
END
```

4.3.3. Circuito eléctrico.

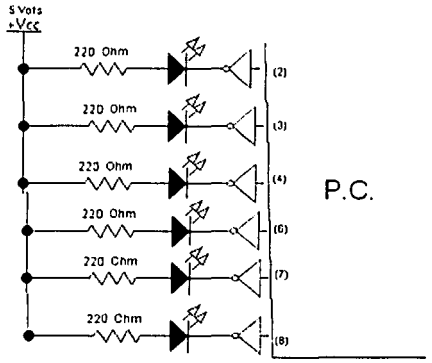


Figura 29. Circuito Eléctrico para diseñar el semáforo de un crucero.

La figura 29 muestra que la comunicación se da por el puerto paralelo de la PC hacia el circuito que emite la respuesta, se conectan las terminales del cable que tiene un conector DB25, siendo los pines 2,3,4,6,7 y 8 conectados a la compuerta lógica NOT, las salidas de la compuerta va hacia los leds y a las resistencias que se conectan al voltaje de 5V de corriente continua. La figura 30 muestra el circuito eléctrico físicamente.

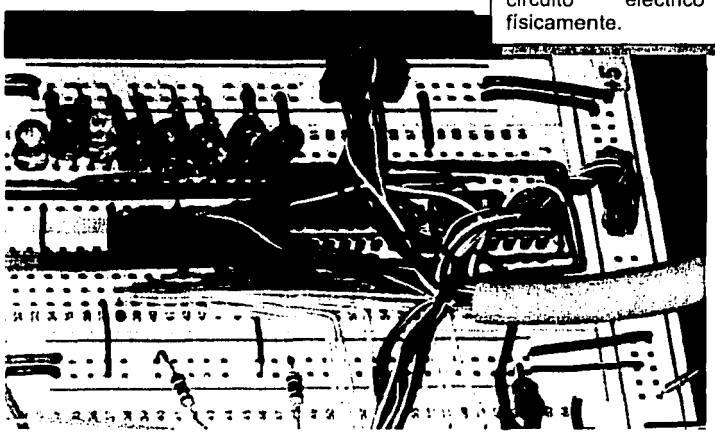


Figura 30. Circuito Eléctrico Físico para el semáforo de un crucero.

4.4. Semáforo en un crucero (68HC11).

OBJETIVO. Controlar la duración de las luces de un semáforo en un crucero.

Suponga que se tiene 2 avenidas dispuestas en la siguiente forma:

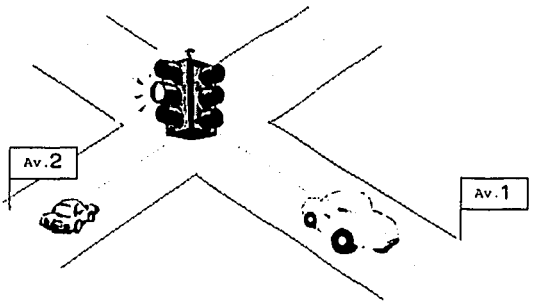


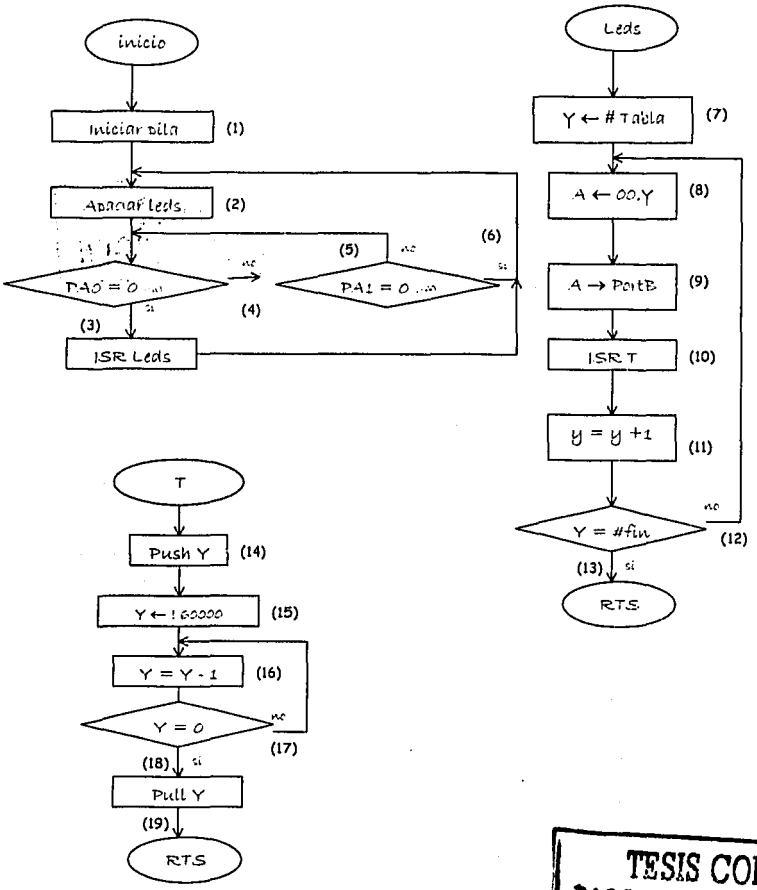
Figura 31. Semáforo de un crucero.

Se requiere programar la secuencia debida para el semáforo de tal forma que los autos tengan el tiempo suficiente para circular sin tener un accidente.

Entonces, se puede generar la siguiente tabla de tiempos, con la asignación respectiva en Hexadecimal:

80	40	20	10	8	4	2	1	
Semáforo de la Av 2.				Semáforo de la Av 1.				
X	Verde	Amarillo	Rojo	X	Verde	Amarillo	Rojo	Hexadecimal
0	1	0	0	0	0	0	1	41h
0	0	1	0	0	0	0	1	21h
0	0	0	1	0	1	0	0	14h
0	0	0	1	0	0	1	0	12h

4.4.1. Diagrama de flujo.



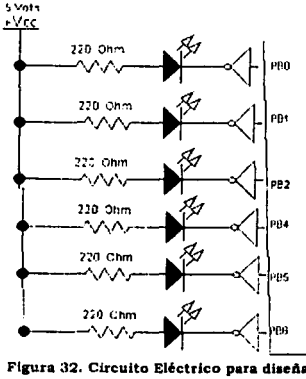
4.4.2. Código fuente.

```
        ORG $100
PORTB   EQU $04
PORTA   EQU $00

        LDS #$0047                  (1)
        LDX #$1000
ES      CLR PORTB,X                 (2)
N4      BRCLR PORTA,X,$01,LED       (3)
        JMP N4                      (4)
LED
        LDY #TABLA                  (7)
N6      BRCLR PORTA,X,$02,ES        (6)
        LDAA 00,Y                   (8)
        STAA PORTB,X                (9)
        JSR T                       (10)
        INY                         (11)
        CPY #FIN  }
        BNE  N6   }                 (12)
        RTS                         (13)

T       PSHY                        (14)
        LDY #!60000                 (15)
SEGUIR  DEY                         (16)
        CPY #$00  }
        BNE SEGUIR}                 (17)
        PULY                        (18)
        RTS                         (19)
TABLA DB $41,$21,$14,$12
FIN DB $00
```


4.4.3. Circuito eléctrico.



La figura 32 muestra que la comunicación se da por el Puerto B del microcontrolador 68HC11, de las terminales 0 – 6 y son conectados a la compuerta lógica NOT, las salidas de la compuerta van hacia los leds y a las resistencias que se conectan al voltaje de 5V de corriente continua. La figura 33 muestra el circuito eléctrico dispuesto físicamente.

Figura 32. Circuito Eléctrico para diseñar el semáforo de un crucero conectado para diseñado para Motorola.

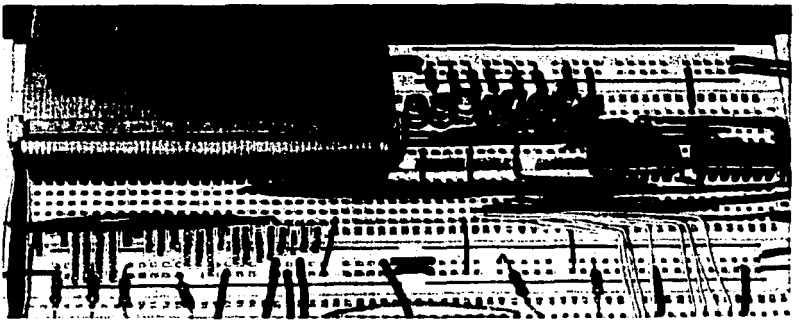


Figura 33. Circuito Eléctrico Físico para el semáforo de un crucero con el microcontrolador de Motorola.

4.4.4. Comparación Intel vs Motorola.

En los Temas 4.1. y 4.2. , se presenta el mismo problema resuelto con dos sistemas de lenguaje ensamblador diferentes, a continuación se presentan las características de utilizar uno u otro para la aplicación de controlar la duración de las luces de un semáforo en un crucero.

Características Intel:

- ✶ Es utilizado por la mayoría de los programadores.
- ✶ No permite indexar sobre los registros, a menos que se utilice push y pull de los valores de los registros al entrar y salir de un procedimiento.
- ✶ El programa esta dividido por segmentos: pila, datos y código.
- ✶ Su programación modular requiere utilizar módulos para limpiar la pantalla, regresar a DOS, leer del teclado, escribir.
- ✶ En éste caso se requieren módulos para regular el tiempo de encendido de cada led, apagar (inicialmente) los leds, y mandar el valor (en Hexadecimal) para asignar el cambio de luces por el puerto paralelo.
- ✶ Permite comparar el teclado solo con la instrucción CMP y destinar a diversas etiquetas (que dirigen a un procedimiento) como lo haría un IF.. THEN...ELSE...ENDIF.
- ✶ La entrada de datos es por el teclado, se visualiza el avance del programa en pantalla y la salida es por el puerto LPT1 (usualmente para impresora), con un cable con conector DB25 (paralelo).
- ✶ El código es muy largo, si se requiere otro valor (Hexadecimal) para la asignación de luces a prender, se necesita seguir paso a paso el código para cambiar los valores, si se requiere aumentar / disminuir dichos valores requiere modificar líneas completas.

Características Motorola:

- La asignación del valor (Hexadecimal) se da por medio de una tabla, siendo flexible, debido a que si se requiere otro valor basta con modificar la tabla.
- El tamaño del código es pequeño.
- La comunicación usuario – programa se da por componentes electrónicos básicos, la entrada se da por push bottom (on/off), la respuesta se da en los leds y para seleccionar el puerto de comunicaciones, se selecciona los requeridos de los disponibles por el microcontrolador.
- Se tiene un puerto de comunicaciones (puerto serie COM1) para realizar pruebas (modo special test).
- Proporciona un software "BUFFALO" para comunicar al usuario para cargar el programa a RAM, y realizar las pruebas.
- Básicamente se utilizan el acumulador A y B (como parte alta y baja del acumulador D).
- Tiene una estructura modular para aplicaciones de tiempo.

4.4.5. Aplicaciones Industriales para un semáforo.

1. La utilización del semáforo puede ser en un principio, en un crucero, pero suponga que tiene una empacadora, en la cual se tienen 2 bandas que transportan botellas con cierta etiqueta y botellas con otro tipo de etiquetas, los operadores de las bandas activan o permiten la circulación de los productos por medio de indicadores (un semáforo) el cuál es un indicador visual para el operador además de realizar de forma automática el cambio de bandas, de modo que las botellas no se colapsen o se mezclen, es decir, corresponda la botella con su etiqueta.

2. Suponga que se tiene una industria cementera, en el proceso de llenado hacia los camiones de carga de grava (arena, cemento, etc), por medio de tres bandas transportadoras y con ayuda de la gravedad, la carga cae hacia el camión; se debe cumplir que el proceso sea puesto en marcha si y sólo si hay carga en el contenedor principal y que el camión este en posición de llenado. El proceso se ilustra a continuación:

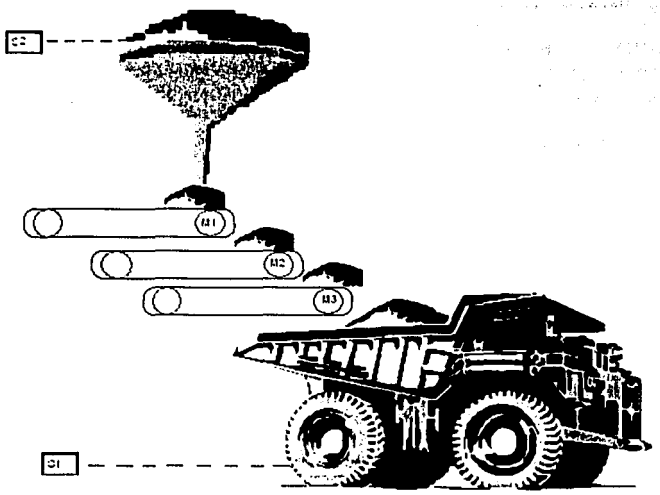


Figura 34. Carga de material hacia un camión con sensores de presencia.

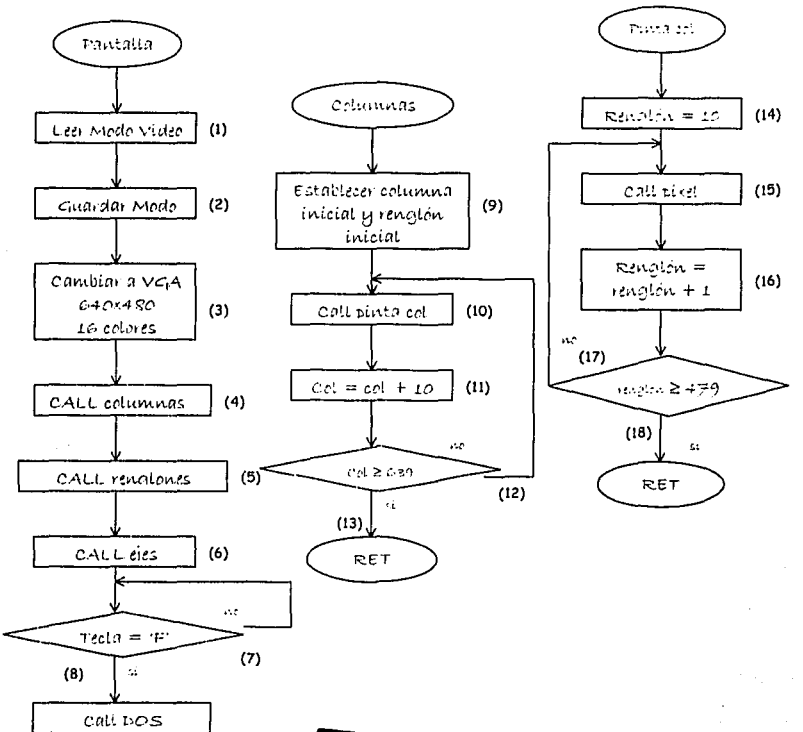
En la figura 34, la aplicación funciona de la siguiente manera: al detectar el S1 (sensor 1) la presencia del camión, el proceso deberá verificar la segunda condición que es la presencia de carga en el S2 (sensor 2), el programa deberá realizar la secuencia de encender el M1 (motor 1), M2 (motor 2) y M3 (motor 3) que hacen girar las bandas para que la carga se deslice hasta la caja del camión, en el momento de detectar la ausencia de cualquiera de los sensores o bien de ambos, entonces, la secuencia de apagado será el M3, M2 y M1, la activación / desactivación de los motores se controlan en el programa por medio de impulsos eléctricos conectados en el motor correspondiente, haciendo las veces de leds para el caso del programa mostrado.

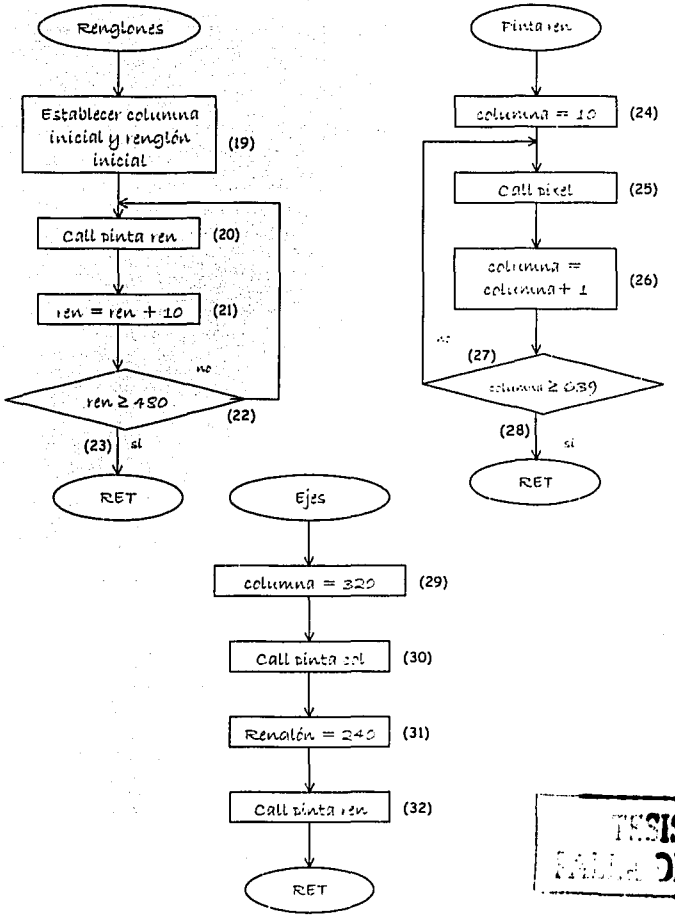
Como se muestra, las aplicaciones para el programa mostrado, es cuestión de resolver alguna necesidad industrial y así como se pueden activar leds, lo mismo seria para una lámpara piloto, zumbadores, timbres, cilindros de simple o doble efecto, cilindros con vástago doble, cilindros tandem, cilindros multiposición, motores de corriente continua, motores (lentos) de posicionamiento, sincromotores de corriente alterna, motores aletas, válvulas reguladoras de caudal, motores hidráulicos, motores electrohidráulicos, motores lentos, servoaccionamientos, etc.

4.5. Generador de funciones (Intel).

OBJETIVO. Realizar un simulador (generador) de funciones, tal que proporcione las siguientes señales: seno, diente de sierra, sierra invertido, triángulo y escalón.

4.5.1. Diagrama de Flujo.





En el diagrama de flujo se muesta solo la parte de generar la pantalla con las divisiones de los ejes y los ejes coordenados, a ésto se le debe de agregar cada función a generar.

4.5.2. Código Fuente.

```
PILA    SEGMENT PARA STACK 'STACK'
        DB   64 DUP('FFFF ')
PILA    ENDS

DATOS   SEGMENT PARA PUBLIC 'DATA'
LETRERO1  DB  'M E N U ',0AH,0DH,0AH,'$'
LETRERO2  DB  'PROPORCIONA LA OPCION QUE DESEES',0AH,0DH,'$'
LETRERO3  DB  'SENO.....A ',0AH,0DH,'$'
LETRERO4  DB  'TRIANGULO.....B ',0AH,0DH,'$'
LETRERO5  DB  'CUADRADA.....C ',0AH,0DH,'$'
LETRERO6  DB  'SIERRA.....D ',0AH,0DH,'$'
LETRERO7  DB  'SIERRA INVERSA.....E ',0AH,0DH,'$'
LETRERO8  DB  'SALIR..... F ',0AH,0DH,'$'

GRAFICA  DB       'VELOCIDAD1=TECLA 1    VELOCIDAD3=TECLA 3'
      DB    13,10,'VELOCIDAD2=TECLA 2    MENU= M          '



TIMER    DW (65535)
FACTOR   DW ?
SEPCOL   DW 10
SEPREN   DW 10
VAR1     DB ?
VAR2     DB ?
TABLA    DW 240,240,235,231,226,222,217,213,209,204,201,196,192,188,184,180
         DW 176,172,168,165,161,158,155,151,148,145,142,139,137,134,132,130
         DW 127,125,124,122,120,119,117,116,115,114,114,113,113,113,112,112
         DW 113,113,114,114,115,116,117,119,120,122,124,125,127,129,132,134
         DW 137,140,142,145,148,151,155,158,162,165,169,172,176,180,184,188
         DW 192,196,201,205,209,214,218,222,227,231,236,240,242,247,251,256
         DW 260,264,269,273,277,282,286,290,294,298,302,306,309,313,317,320
         DW 324,327,330,333,336,339,342,344,347,349,352,354,356,357,359,361
         DW 362,363,364,365,366,367,367,367,367,367,367,367,366,365,364
         DW 363,362,361,359,357,356,354,352,349,347,344,342,339,336,333,330
         DW 327,324,320,317,313,309,306,302,298,294,290,286,282,277,273,268
         DW 264,260,256,251,247,242

FIN      DW 1
DATOS    ENDS

CODIGO   SEGMENT PARA PUBLIC 'CODEC'
         PUBLIC PRINCIPAL

PRINCIPAL  PROC FAR
         ASSUME CS:CODIGO,DS:DATOS,SS:PILA
         PUSH DS
         SUB AX,AX
         PUSH AX
         MOV AX,SEG DATOS
         MOV DS,AX
           CALL   VIDEO
           MOV    WORD PTR [VAR1],AX
           MOV    WORD PTR [VAR2],BX
LEY:       CALL   LEYVA
         JMP    LEY
         RET
PRINCIPAL  ENDP

LEYVA  PROC   NEAR
       CALL   LIMPIAR
       CALL   CURSOR
       LEA    DX,LETRERO1
       CALL   LETRERO
       LEA    DX,LETRERO2
       CALL   LETRERO
       LEA    DX,LETRERO3
       CALL   LETRERO
```

```
LEA   DX,LETRERO4
      CALL  LETRERO
      LEA   DX,LETRERO5
      CALL  LETRERO
      LEA   DX,LETRERO6
      CALL  LETRERO
      LEA   DX,LETRERO7
      CALL  LETRERO
      LEA   DX,LETRERO8
      CALL  LETRERO

      MOV   AH,1
      INT   21H
      CMP   AL,'A'
      JE    SE1
      CMP   AL,'B'
      JE    SE2
      CMP   AL,'C'
      JE    SE3
      CMP   AL,'D'
      JE    SE4
      CMP   AL,'E'
      JE    SE5
      CMP   AL,'F'
      JE    MENY
      JNE   MENX
SE1:      CALL  FONCUA
SE1A:     INC   AX
          PUSH  AX
          CALL  SENO
          CALL  VELO
          CALL  PUERTO
          CALL  SALIR
          POP   AX
          JMP   SE1A
SE2:      CALL  FONCUA

SE2A:     INC   AX
          PUSH  AX
          CALL  TRIANGULO
          CALL  VELO
          CALL  PUERTO
          CALL  SALIR
          POP   AX
          JMP   SE2A
SE3:      CALL  FONCUA
SE3A:     INC   AX
          PUSH  AX
          CALL  CUADRADA
          CALL  VELO
          CALL  PUERTO
          CALL  SALIR
          POP   AX
          JMP   SE3A
SE4:      CALL  FONCUA
SE4A:     INC   AX
          PUSH  AX
          CALL  SIERRA
          CALL  VELO
          CALL  PUERTO
          CALL  SALIR
          POP   AX
          JMP   SE4A
SE5:      CALL  FONCUA
SE5A:     INC   AX
          PUSH  AX
          CALL  SIERRAINV
          CALL  VELO
          CALL  PUERTO
          CALL  SALIR
```

```
        POP   AX
        JMP   SE5A

MENY:  CALL   DOSE
MENX:  RET
LEYVA  ENDP

LETRERO   PROC NEAR
      MOV   AH,9
      INT   21H
      RET
LETRERO   ENDP


;         DESPEJAR PANTALLA

LIMPIAR  PROC     NEAR
     MOV      AX,0600H      ;RECORRER TODA LA PANTALLA
     MOV      BH,15         ;ROJO SOBRE AZUL
     MOV      CX,0000       ;POSICION IZQUIERDA SUPERIOR
     MOV      DX,184FH      ;POSICION DERECHA INFERIOR
     INT      10H
     RET
LIMPIAR  ENDP

FONCUA   PROC   NEAR
     MOV   AL,12H
     CALL  MODO
     CALL  FONDO
     CALL  CUADRICULA
     CALL  GRAF
     RET
FONCUA   ENDP

GRAF    PROC  NEAR
     MOV   AH,40H
     MOV   BX,01
     MOV   CX,80
     LEA   DX,GRAFICA
     INT   21H
     RET
GRAF    ENDP


CUADRICULA   PROC  NEAR
        MOV   AL,08     ;*** Color cuadricula
        CALL  COLUMNAS
        CALL  RENGLONES
        MOV   AL,4      ;****Color ejes
        CALL  EJES
        RET
CUADRICULA   ENDP

PUERTO PROC NEAR
    PUSH AX
    PUSH DX
    MOV  AL,00H      ;enviar
    MOV  DX,0378H     ; al
    OUT  DX,AL       ::puerto
    INC  AL
    POP  DX
    POP  AX
    RET
PUERTO ENDP

SENO    PROC  NEAR

     MOV  CX,130
     LEA  SI,TABLA         ;******cargando tabla
L1:    MOV  DX,WORD PTR[SI]        ;******mueve el valor a DX segun la posicion de SI
     CMP  SI,OFFSET FIN
```

```
        JE  L2
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
        CALL PIXEL
        PUSH AX
        PUSH DX
        CALL VELO
        CALL SALIR
        POP  DX
        POP  AX
        ADD  CX,2
        CMP  CX,639
        JAE  L2
        ADD  SI,2
        JMP  L1
L2:     RET
SENO    ENDP

TRIANGULO   PROC  NEAR
        MOV  CX,00
TRIAX:  MOV  DX,368
TRIA1:  CALL PIXEL
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
        DEC  DX
        ADD  CX,1
        CMP  DX,113
        JBE  TRIA2
        CMP  CX,511
        JAE  TRIAS
        JMP  TRIA1
TRIA2:  CALL PIXEL
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
        INC  DX
        ADD  CX,1
        CMP  DX,368
        JAE  TRIAX
        CMP  CX,511
        JAE  TRIAS
        JMP  TRIA2
TRIAS:  RET
TRIANGULO   ENDP

SIERRA  PROC  NEAR
        MOV  CX,00
DIE1:   MOV  DX,368
DIE2:   CALL PIXEL
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
```

```
        DEC  DX
        ADD  CX,1
        CMP  DX,113
        JBE  DIE1
        CMP  CX,509
        JAE  DIE3
        JMP  DIE2
DIE3:   RET
SIERRA  ENDP

SIERRAINV  PROC  NEAR
        MOV  CX,00
INV1:   MOV  DX,113
INV2:   CALL PIXEL
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
        INC  DX
        ADD  CX,1
        CMP  DX,368
        JAE  INV1
        CMP  CX,509
        JAE  INV3
        JMP  INV2
INV3:   RET
SIERRAINV  ENDP

CUADRADA  PROC  NEAR
        MOV  CX,00
CUAT.   MOV  DX,113
        CALL LINEA
        CMP  CX,600
        JAE  CUAX
        MOV  DX,368
        CALL LINEA
        CMP  CX,600
        JAE  CUAX
        JMP  CUAT
CUAX:   RET
CUADRADA  ENDP

LINEA   PROC  NEAR
        SUB  BX,BX
LIN1:   CALL PIXEL
        PUSH AX
        PUSH BX
        PUSH CX
        CALL DELAY
        POP  CX
        POP  BX
        POP  AX
        ADD  CX,1
        CMP  CX,600
        JAE  LIN2
        ADD  BX,1
        CMP  BX,50
        JNE  LIN1
LIN2:   RET
LINEA   ENDP

DELAY   PROC NEAR
        MOV  CX,3
DDT2:   MOV  BX,CX
        MOV  CX,[TIMER]
DDT     DEC  CX
        CMP  CX,0
        JNE  DDT
```

```
      MOV  CX,BX
      DEC  CX
      CMP  CX,0
      JNE  DDT2
      RET                ;******regresa a proc far
DELAY  ENDP

VIDEO  PROC  NEAR
      MOV  AH,0FH
      INT  10H
      RET
VIDEO  ENDP

MODO   PROC NEAR
      MOV AH,0
      INT  10H
      RET
MODO   ENDP

PIXEL  PROC  NEAR
      CALL PUERTO
      MOV  AH,0CH
      MOV  BH,0
      INT  10H
      RET
PIXEL  ENDP

COLUMNAS  PROC  NEAR
      MOV   CX,10
VER:    CALL  VERTICAL
      ADD   CX,SEPCOL
      CMP   CX,640
      JNE   VER
      RET
COLUMNAS  ENDP

VERTICAL PROC  NEAR
      XOR  DX,DX
OTRO:   CALL  PIXEL
      INC   DX
      CMP   DX,479
      JNE   OTRO
      RET
VERTICAL ENDP

RENGLONES PROC  NEAR
      MOV   DX,10
VER2:   CALL  HORIZONTAL
      ADD   DX,SEPREN
      CMP   DX,480
      JNE   VER2
      RET
RENGLONES ENDP

HORIZONTAL  PROC  NEAR
       XOR  CX,CX
OTRO2:    CALL  PIXEL
       INC  CX
       CMP  CX,639
       JNE  OTRO2
       RET
HORIZONTAL  ENDP

EJES   PROC  NEAR
      MOV  CX,320
      CALL VERTICAL
MOV   DX,240
      CALL HORIZONTAL
      RET
EJES   ENDP
```

```
FONDO   PROC NEAR
        MOV  AX,0600H
        MOV  BH,0B9H        ;*****BH17H
        MOV  CX,0000H
        MOV  DL,79
        MOV  DH,29
        INT  10H
        RET
FONDO   ENDP

SALIR   PROC NEAR
        MOV  DX,00FFH
        MOV  AH,06
        INT  21H
        CMP  AL,'F'         ;*****compara S
        JE   DOSES
        CMP  AL,'f'         ;****compara s
        JE   DOSES
        CMP  AL,'M'
        JE   TWEL
        JMP  REGR
DOSES:  CALL DOSE
TWEL:   CALL PRINCIPAL
REGR:   RET
SALIR   ENDP

VELO    PROC NEAR
        MOV  DX,00FFH
        MOV  AH,06
        INT  21H
        CMP  AL,'1'         ;********compara S
        JE   VEL1
        CMP  AL,'2'         ;*******compara s
        JE   VEL2
        CMP  AL,'3'         ;*******compara s
        JE   VEL3
        JMP  VELX
VEL1:   MOV  [TIMER],60000
        JMP  VELX
VEL2:   MOV  [TIMER],40000
        JMP  VELX
VEL3:   MOV  [TIMER],30000
VELX:   RET
VELO    ENDP

DOSE    PROC NEAR
        MOV  AX,WORD PTR [VAR1]
        MOV  BX,WORD PTR [VAR2]
        CALL MODO
        MOV  AX,4C00H
        INT  21H
        RET
DOSE    ENDP

CURSOR  PROC   NEAR
        MOV    AH,02H
        MOV    BH,00
        MOV    DH,2
        MOV    DL,2
        INT    10H
        RET
CURSOR  ENDP

CODIGO  ENDS
        END    PRINCIPAL
```

Para generar las funciones hacia un osciloscopio se tiene:

Señal Cuadrada:

```
.model small
.STACK 100H
.code
      CALL APAGAR
RE:   MOV DX,0378H
      MOV AL,00  ;MANDE CERO
      OUT DX,AL
      INC AH
      CMP AH,0FH
      JNE RE
EL:   MOV AL,0FFH  ;MANDE UNO
      OUT DX,AL
      INC AH
      CMP AH,0FH
      JNE EL
      JMP RE
APAGAR PROC NEAR
      MOV DX,0378H
      MOV AL,00H
      OUT DX,AL
      RET
APAGAR ENDP

END
```

Señal Diente de Sierra:

```
.model small
.STACK 100H
.code
      CALL APAGAR
      MOV DX,0378H
      MOV AL,00
RE:   OUT DX,AL
      INC AL
      JMP RE

APAGAR PROC NEAR
      MOV DX,0378H
      MOV AL,00H
      OUT DX,AL
      RET
APAGAR ENDP

END
```


Señal Diente de Sierra Invertido:

```
.model small
.STACK 100H
.code
      CALL APAGAR
      MOV DX,0378H
      MOV AL,00
RE:   OUT DX,AL
      DEC AL
      JMP RE
```

```
APAGAR PROC NEAR
     MOV DX,0378H
     MOV AL,00H
     OUT DX,AL
     RET
APAGAR ENDP

END
```

Señal Triángulo

```
.model small
.STACK 100H
.code
     CALL APAGAR
RE:    MOV DX,0378H
     MOV AL,00  ;MANDE CERO
     OUT DX,AL
     INC AL
     CMP AL,0FH
     JNE RE
EL:    MOV AL,0FH  ;MANDE UNO
     OUT DX,AL
     DEC AL
     CMP AL,00H
     JNE EL
     JMP RE

APAGAR PROC NEAR
     MOV DX,0378H
     MOV AL,00H
     OUT DX,AL
     RET
APAGAR ENDP

END
```

El código que se presenta se diseñó para generar la señal en un osciloscopio, así al visualizar en pantalla, también la señal requerida será parte de algún proceso industrial; se menciona como "señal hacia osciloscopio" porque el osciloscopio es el instrumento de medición de las señales, el cual proporciona información de amplitud (Voltaje) y frecuencia (Hertz) de las señales medidas.

4.5.3. Circuito eléctrico.

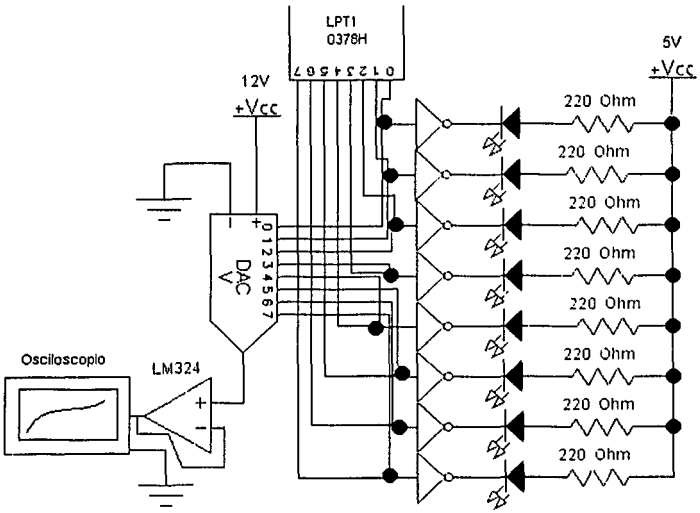


Figura 35. Circuito Eléctrico para el Generador de Funciones con INTEL.

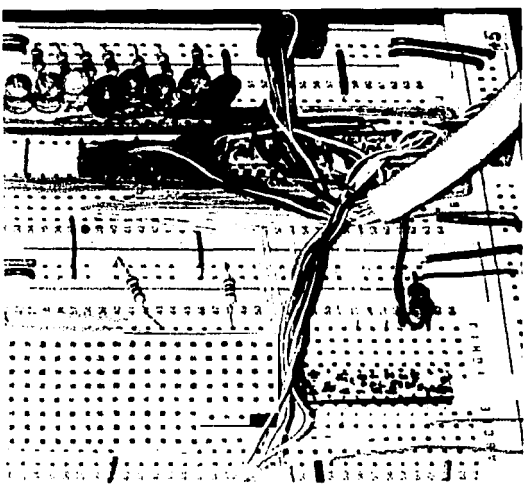


Figura 36. Circuito Eléctrico Físico para el Generador de Funciones para INTEL

La figura 35 muestra que la comunicación se da por el puerto paralelo de la PC hacia el circuito que emite la respuesta, se conectan las terminales del cable que tiene un conector DB25, siendo los pines 2,3,4,6,7 y 8 conectados a la compuerta lógica NOT, las salidas de la compuerta va hacia los leds y a las resistencias que se conectan al voltaje de 5V de corriente continua; también se manda la información al DAC para enviar la señal al osciloscopio. La figura 36 muestra físicamente el circuito

4.5.4. Aplicaciones industriales para un generador de funciones.

Existen señales que colaboran con otros procesos para generar uno final, principalmente constituyen parámetros para otras funciones.

En el mundo de las telecomunicaciones se tiene que una pequeña red de comunicación en su forma simple, esta compuesta como se muestra en la figura 36.

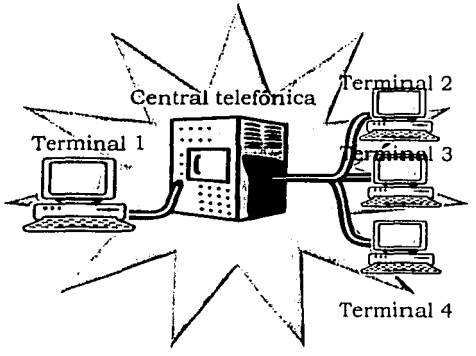


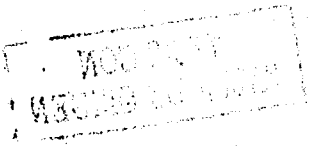
Figura 37. Comunicación Vía Telefónica de PC por medio de la Central Telefónica.

Para lograr la comunicación entre la terminal 1 y la terminal 2, es necesario, como sabemos, la terminal 1 genera datos digitales, por lo que esta conectado a un MODEM (modulador / demodulador) que tiene como objetivo, convertir la señal digital de la terminal 1 a señal analógica para ser enviada por el cable telefónico; en este proceso, lo que pasa dentro del MODEM es que, la señal es multiplicada por una portadora (generalmente **señal seno**) y entonces se puede mandar la señal, al llegar ésta señal a la central telefónica, se direcciona hacia su destino (terminal 2), en la terminal 2, MODEM (de la terminal 2) recibe la señal analógica y la decodifica (convierte en señal digital) para que la terminal 2 pueda entender la información, como se muestra en la figura 37. De tal suerte, que se puede observar que la participación de la señal seno en las telecomunicaciones es trascendental.

La respuesta a los impulsos de un amplificador determina su capacidad de reproducir un pulso de entrada de **onda cuadrada** (un tipo de señal eléctrica regular) de forma rápida y precisa; las entradas de ondas cuadradas son dirigidas hacia un amplificador para su recuento o cronometraje. La respuesta a los impulsos es importante en los circuitos informáticos digitales, la modulación por impulsos codificados y los instrumentos de radar y nucleares, es decir, dondequiera que se procesen pulsos de onda cuadrada de alta frecuencia. Por ejemplo, en un radiómetro, la frecuencia con que las partículas de radiación golpean contra un elemento sensible, como el diodo de unión de un semiconductor, es una medida de la intensidad o de la concentración de partículas. La emisión de diodo resultante puede ser una serie de impulsos que a continuación son amplificados y dirigidos hacia un transductor para poder ser vistos.

Los amplificadores con características de bajo nivel de ruido son esenciales para los satélites de comunicaciones. Las señales electromagnéticas de microondas (de frecuencia extremadamente alta) son amplificadas por dispositivos máser (amplificación de microondas por emisión estimulada de radiación). En lugar de amplificar corriente eléctrica el máser amplifica directamente las señales electromagnéticas.

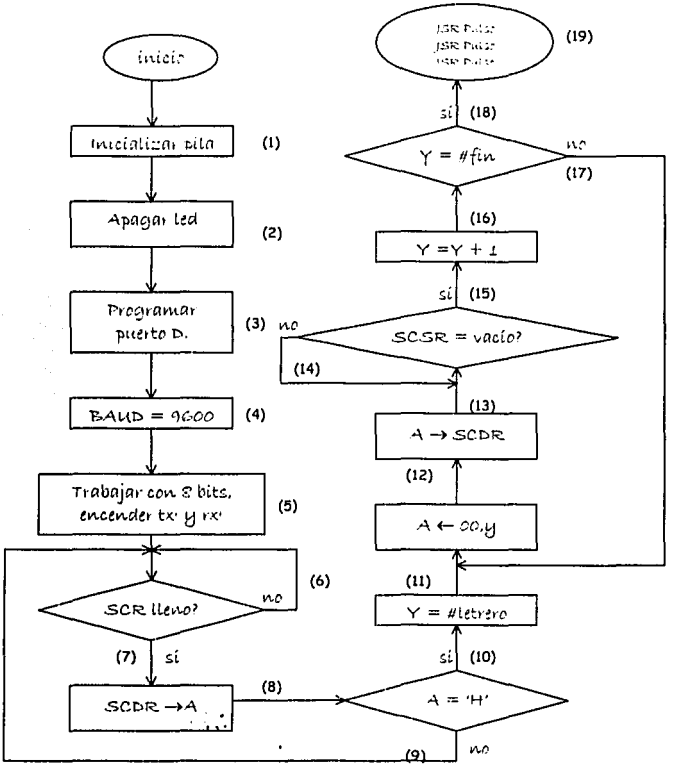
En acústica, la percepción auditiva de cada frecuencia puede verse afectada por diversos factores, como, por ejemplo, la forma o amplitud de las ondas (**ondas de diente de sierra**, **cuadradas**, más o menos ricas en armónicos), el lugar del receptor respecto al foco emisor o las características acústicas del medio en el que se emite el sonido.

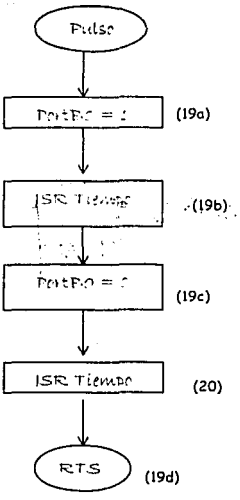


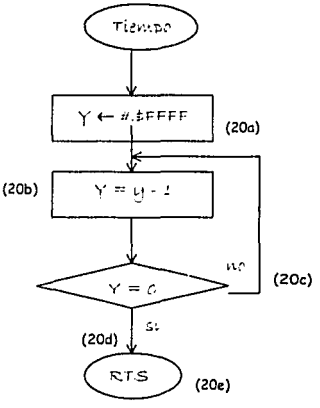
4.6. Mensaje (Motorola).

OBJETIVO. Realizar un programa que al oprimir la tecla 'H' salga el letrero "ENLACE COMPLETADO" a una velocidad de 9600 baudios y encienda y apague un led 3 veces.

4.6.1. Diagrama de flujo.







4.6.2. Código Fuente.

```
      ORG $100
PORTA EQU $1000
PORTB EQU $04
SCDR  EQU $2F
PORTD EQU $08
SCCR1 EQU $2C
SCCR2 EQU $2D
BAUD EQU $2B
DDRD EQU $09
SCSR EQU $2E

      LDS #$0047                (1)
N2    LDX #PORTA                (2)
      BCLR PORTB,X,$01
      LDAA #$02                 (3)
      STAA DDRD,X
      LDAB #$30                 (4)
      STAB BAUD,X
      CLR SCCR1,X               (5)
      LDAB #$0C
      STAB SCCR2,X
ESP   BRCLR SCSR,X,$20,ESP      (6)
      LDAA SCDR,X               (7)
      CMPA #'H'                 (8)
```

```
    BNE ESP                    (9)
    LDY #TABLA             (10), (11)
MAS   LDAA 00,Y                (12)
    STAA SCDR,X                (13)
ESP2  BRCLR SCSR,X,$80,ESP2    (15)
    INY                  (14),(16)
    CPY #FIN          }
    BNE MAS           } (17)
    JSR PULSO         }
    JSR PULSO         } (18), (19)
    JSR PULSO         }
    JMP N2
PULSO  BSET PORTB,X,$01        (19a)
    JSR TIEMPO                 (19b)
    BCLR PORTB,X,$01           (19c)
    JSR TIEMPO                 (20)
    RTS                        (19d)
TIEMPO  LDY #$FFFF             (20a)
SIGUE  DEY                     (20b)
    BNE SIGUE                  (20c)
    RTS                        (20d),(20e)
TABLA DB 'ENLACE COMPLETADO'
   DB 113,110
FIN  DB $00
```


4.6.3. Circuito eléctrico.

La figura 38 muestra que la comunicación esta dada por el puerto serie que tiene una conexión hacia la tarjeta CT68HC11 y así el microcontrolador procesar la información, recibiendo y transmitiendo información entre los dos procesadores, la información la recibe / transmite el microcontrolador por el Puerto D en su bit 0 y 1, al Puerto B en su bit 0 se encuentra conectado un negador, led y a 5V de corriente continua para indicar al usuario que se establece la comunicación.

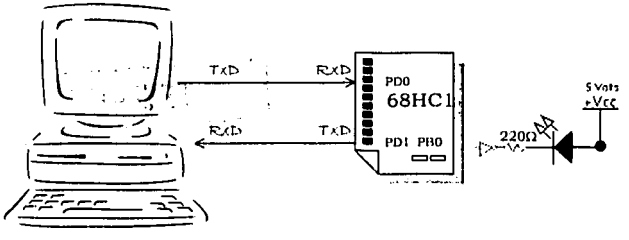


Figura 38. Diseño eléctrico para comunicar la PC con el microcontrolador de MOTOROLA.

4.6.4. Aplicaciones industriales para transmisión/recepción.

En esta aplicación se hicieron pruebas con la interfaz de comunicación serie, en la que se designaron características del transmisor y del receptor, la programación del registro BAUD para sincronizar la velocidad de transmisión y recepción a través del puerto D y verificando el mensaje recibido en el monitor de la PC y por el puerto B al encender un led, también se probó con la manipulación de tablas para enviar un mensaje, lo que repercute en flexibilidad de cambiar o modificar si se requiere el mensaje en la tabla, sin tener que cambiar una gran cantidad de instrucciones del código fuente.

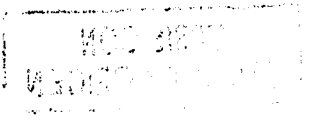
En ocasiones en la industria, los procesos productivos no están aislados, es decir, se requiere un determinado orden y secuencia de éstos procesos, por lo que la comunicación entre cada proceso es indispensable ya sea para efectuar una producción, e incluso por seguridad en caso de ocurrir algún error. Es por ello, que en éste tema se muestra como se puede realizar la comunicación vía puerto SERIE, la cual puede ser como en éste caso PC – microcontrolador, o bien, microcontrolador – microcontrolador, o incluso tener una pequeña red LAN, para tener la mayor eficacia de las comunicaciones se deben de tener en consideración:

1) la distancia entre cada terminal, debido a que mayor distancia entonces tendrá mayor atenuación de la señal y disminución en el tiempo de respuesta.

2) El material conductor del cable, por el momento, se presenta cable de cobre, que es susceptible al ruido.

3) Las características ambientales, debido a que el cable de cobre es susceptible a captar frecuencias electromagnéticas, lo que provocan ruido.

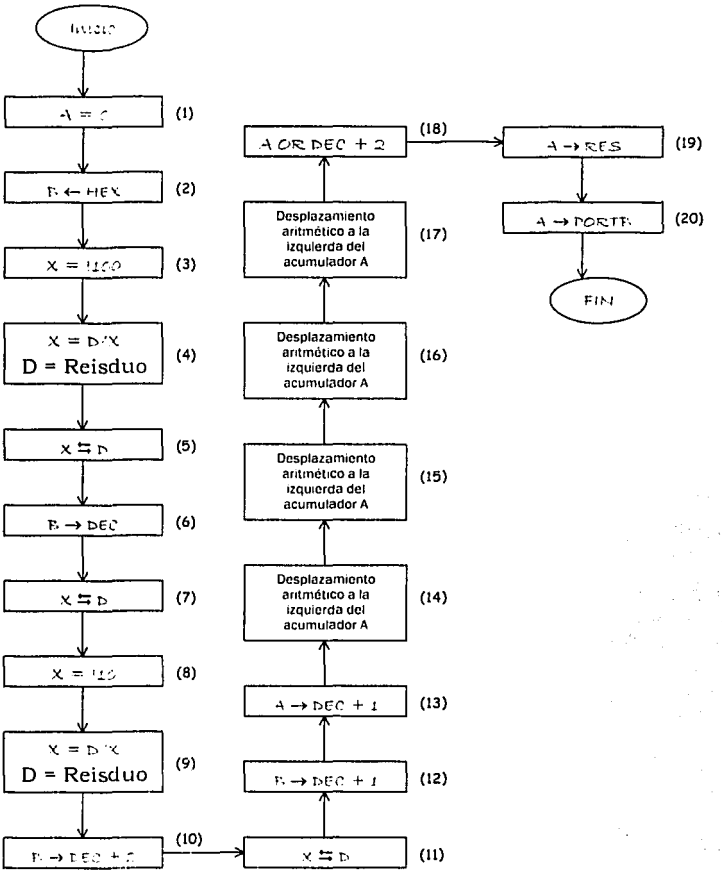
4) Contar con un cableado estructurado que proporcione facilidad para identificar el origen de cada cableado, seguridad del cableado y las terminales.



4.7. Conversor Hexadecimal a Decimal (Motorola).

OBJETIVO. Realizar un convertidor que recibe una señal en el sistema Hexadecimal y convierte ésta señal al sistema decimal.

4.7.1. Diagrama de flujo.



4.7.2. Código Fuente.

```
      ORG $100
HEX   EQU $00
DEC   EQU $01
RES   EQU $04
      LDS #$0047
      CLRA              (1)
      LDAB HEX          (2)
      LDX #!100         (3)
      IDIV              (4)
      XGDX              (5)
      STAB DEC          (6)
      XGDX              (7)
      LDX #!10          (8)
      IDIV              (9)
      STAB DEC+2        (10)
      XGDX              (11)
      STAB DEC+1        (12)
      LDAA DEC+1        (13)
      ASLA              (14)
      ASLA              (15)
      ASLA              (16)
      ASLA              (17)
      ORAA DEC+2        (18)
      STAA RES          (19)
      LDX #$1000        (20)
      STAA 04,X
```


4.7.3. Aplicaciones industriales para un conversor Hexadecimal/decimal.

Es importante automatizar el proceso de conversión numérica de Hexadecimal a decimal, debido en que en la mayoría de los procesos existen dos formas de recibir señales de los elementos sensores:

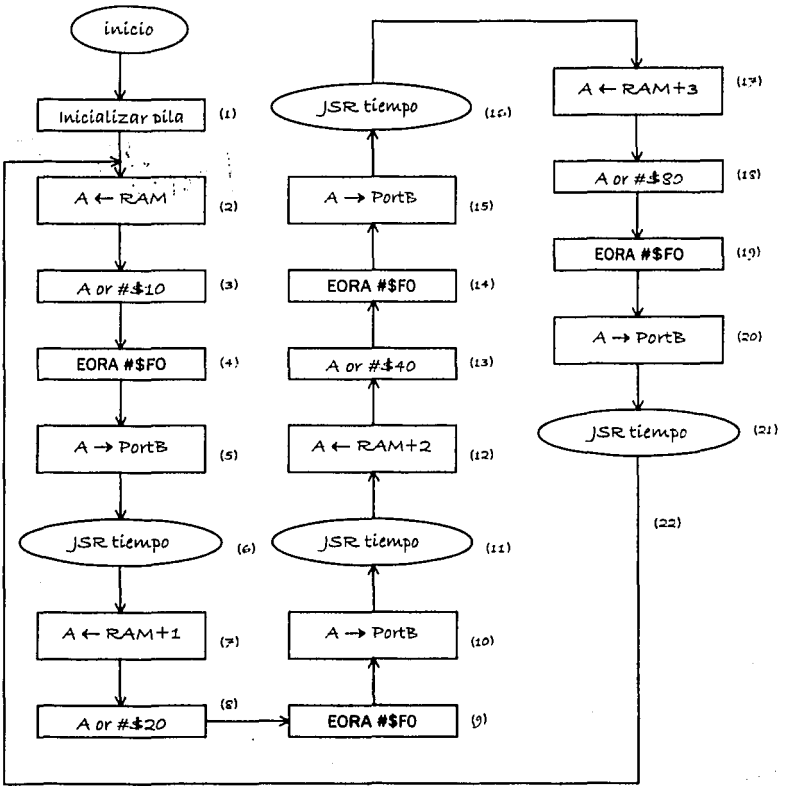
1) Valores de voltaje que asignen valores lógicos, por ejemplo: +5 volts interpretado a 1 lógico, o bien, 0 volts interpretado a 0 lógico. Esto quiere decir que, los elementos sensores tienen valores digitales constantes y su valor no oscila demasiado a no ser por el efecto de ruido, pero en general, se podrán interpretar éstos valores como constantes.

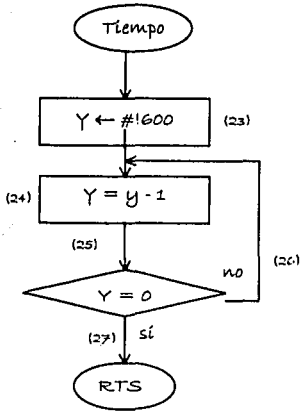
2) Valores de voltaje variantes en el tiempo, esto es, el sensor proporciona diferentes niveles de voltaje dentro de un intervalo determinado (0 a + 5 volts), y por lo general, esta información se interpreta en una Hexadecimal, por lo que para la interpretación y manipulación del dato recibido se requiere convertir el dato Hexadecimal a decimal, desde luego, se pueden controlar los datos en un solo sistema, pero si por ejemplo, se sensa cualquier proceso (temperatura, presión, velocidad, flujo, peso, etcétera) la forma adecuada es de presentar al usuario aquella medición en un display, monitor o pantalla en general en un sistema que la mayoría de las personas utilicen como lo es el sistema decimal.

4.8. Lectura de RAM y display.

Objetivo: Realizar un programa que lea el contenido en RAM y lo despliegue en 4 display.

4.8.1. Diagrama de flujo.





4.8.2. Código fuente.

```
     ORG $100
PORTA   EQU $1000
PORTB   EQU $04
RAM     EQU $00

        LDS #$0047
        LDX #PORTA          (1)
INI     LDAA #RAM           (2)
        ORAA #$10           (3)
        EORA #$F0           (4)
        STAA PORTB,X        (5)
        JSR TIEMPO          (6)
        LDAA #RAM+1         (7)
        ORAA #$20           (8)
        EORA #$F0           (9)
        STAA PORTB,X        (10)
        JSR TIEMPO          (11)
        LDAA #RAM+2         (12)
        ORAA #$40           (13)
        EORA #$F0           (14)
        STAA PORTB,X        (15)
        JSR TIEMPO          (16)
        LDAA #RAM+3         (17)
        ORAA #$80           (18)
        EORA #$F0           (19)
        STAA PORTB,X        (20)
        JSR TIEMPO          (21)
        JMP INI             (22)
TIEMPO  LDY #!600           (23)
SEGUIR  DEY                 (24)
        CPY #$00            (25)
        BNE SEGUIR          (26)
        RTS                 (27)
```

4.8.3. Circuito Eléctrico.

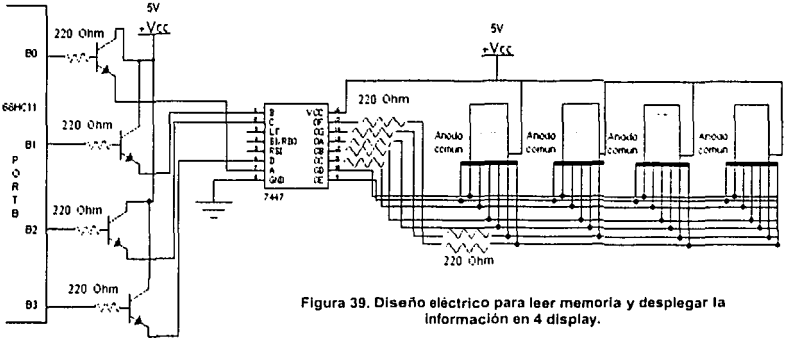
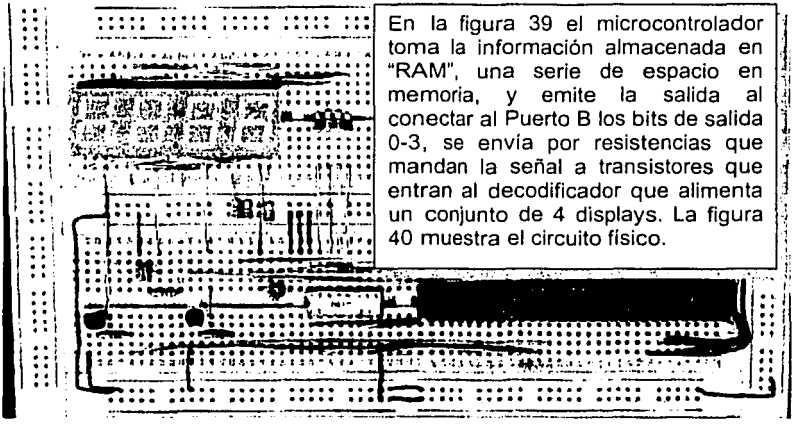


Figura 39. Diseño eléctrico para leer memoria y desplegar la información en 4 display.



En la figura 39 el microcontrolador toma la información almacenada en "RAM", una serie de espacio en memoria, y emite la salida al conectar al Puerto B los bits de salida 0-3, se envía por resistencias que mandan la señal a transistores que entran al decodificador que alimenta un conjunto de 4 displays. La figura 40 muestra el circuito físico.

Figura 40. Circuito eléctrico para leer memoria y desplegar la información en 4 display.

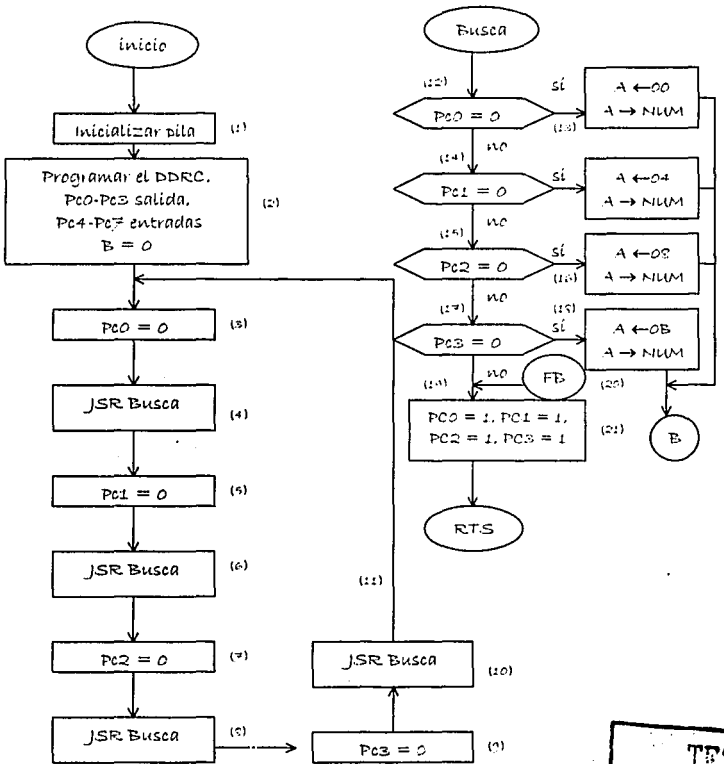
4.8.4. Aplicaciones industriales para lectura de datos de RAM.

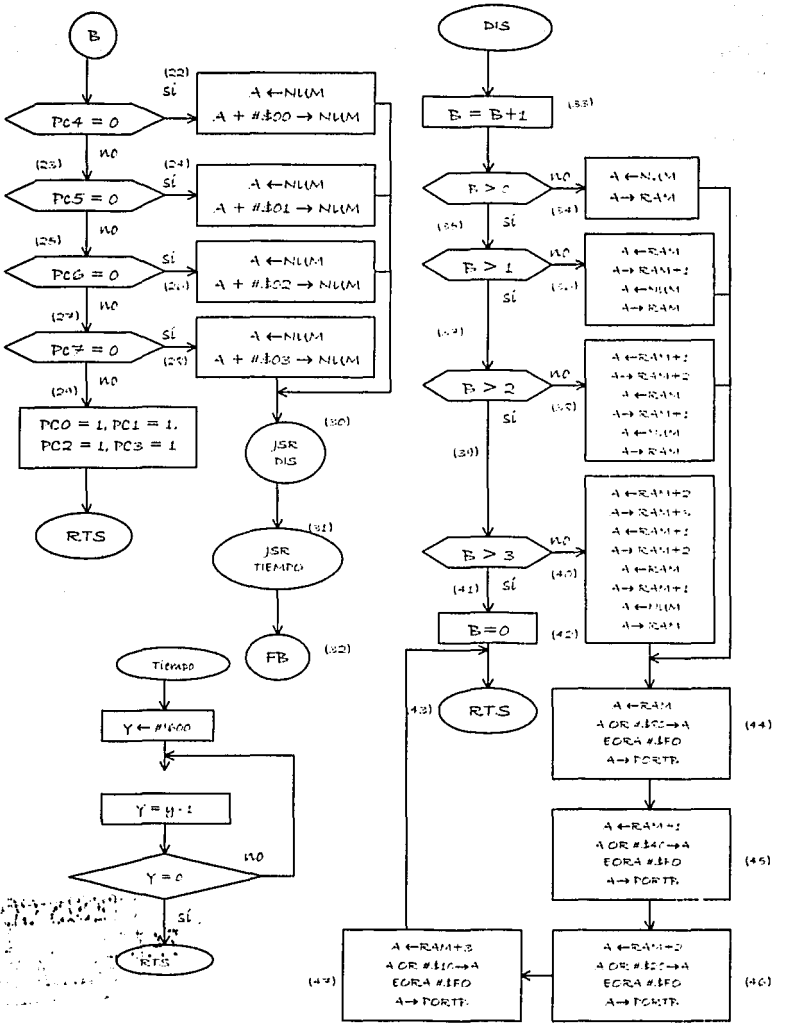
La manipulación de datos tomados de RAM, se colocaron digito a digito en 4 direcciones de memoria desde la 00 a la 03, en la que para desplegar a los transistores, se les sumo $10, $20, $40, $80 dependiendo de la posición para activar los transistores y la parte baja del acumulador se manda al display, es decir, los transistores son la posición (de los 4 display) y el contenido de la dirección (RAM, RAM+1, RAM+2, RAM+3) es el número que se despliega en su correspondiente display. Este proceso constituye una interfaz, es decir, si se tiene un sensor, este proporciona información que se almacena en RAM, en esta aplicación se muestra como teniendo los datos en RAM se puede manipular y controlar algún proceso.

4.9. Detector de teclado y display.

Objetivo: Realizar un programa que lea un teclado de 16 elementos y lo despliegue en 4 display (4 dígitos admisibles), de forma FIFO.

4.9.1. Diagrama de flujo.





4.9.2. Código fuente.

```
  ORG $100
PORTA EQU $1000
PORTB EQU $04
RAM   EQU $00
PORTC EQU $03
DDRC  EQU $07
NUM   EQU $05

      LDS #$0047
      LDX #PORTA
      LDAA #$0F
      STAA DDRC,X
      LDAA #$04
      STAA CONT
INI
      BCLR PORTC,X,$01
      JSR BUSCA
      BCLR PORTC,X,$02
      JSR BUSCA
      BCLR PORTC,X,$04
      JSR BUSCA
      BCLR PORTC,X,$08
      JSR BUSCA
      JMP INI


BUSCA  BRCLR PORTC,X,$01,L0
      JMP PC1
L0    LDAA #$00
      STAA NUM
      JMP B
PC1   BRCLR PORTC,X,$02,L1
      JMP PC2
L1    LDAA #$04
      STAA NUM
      JMP B
PC2   BRCLR PORTC,X,$04,L2
      JMP PC3
L2    LDAA #$08
      STAA NUM
      JMP B
PC3   BRCLR PORTC,X,$08,L3
      JMP FB
L3    LDAA #$B
      STAA NUM
B     BRCLR PORTC,X,$10,PC4S
      JMP PC5
PC4S  LDAA NUM
      ADDA #$00
      STAA NUM
      JSR DIS
```

```
PC5   BRCLR PORTC,X,$20,PC5S
      JMP PC6
PC5S  LDAA NUM
      ADDA #$01
      STAA NUM
      JSR DIS
PC6   BRCLR PORTC,X,$20,PC6S
      JMP PC7
PC6S  LDAA NUM
      ADDA #$02
      STAA NUM
      JSR DIS
PC7   BRCLR PORTC,X,$20,PC7S
      JMP FB
PC7S  LDAA NUM
      ADDA #$03
      STAA NUM
      JSR DIS


      JSR TIEMPO
FB    BSET PORTC,X,$01
      BSET PORTC,X,$02
      BSET PORTC,X,$04
      BSET PORTC,X,$08
      RTS

DIS   INCB
      CMPB #$00
      BHI D2
D1    LDAA NUM
      STAA RAM
      JMP D
D2    CMPB #$01
      BHI D3
      LDAA RAM
      STAA RAM+1
      LDAA NUM
      STAA RAM
      JMP D
D3    CMPB #$02
      BHI D4
      LDAA RAM+1
      STAA RAM+2
      LDAA RAM
      STAA RAM+1
      LDAA NUM
      STAA RAM
      JMP D
D4    CMPB #$03
      BHI D5
      LDAA RAM+2
      STAA RAM+3
      LDAA RAM+1
      STAA RAM+2
      LDAA RAM
      STAA RAM+1
```

```
      LDAA NUM
      STAA RAM

D     LDAA RAM
      ORAA #$80
      EORA #$F0
      STAA PORTB,X
      LDAA RAM+1
      ORAA #$40
      EORA #$F0
      STAA PORTB,X
      LDAA RAM+2
      ORAA #$20
      EORA #$F0
      STAA PORTB,X
      LDAA RAM+3
      ORAA #$10
      EORA #$F0
      STAA PORTB,X
      JMP FDIS
D5    LDAB #$00
FDIS  RTS

TIEMPO  LDY #1600
SEGUIR  DEY
      CPY #$00
      BNE SEGUIR
      RTS
```

4.9.3. Circuito eléctrico.

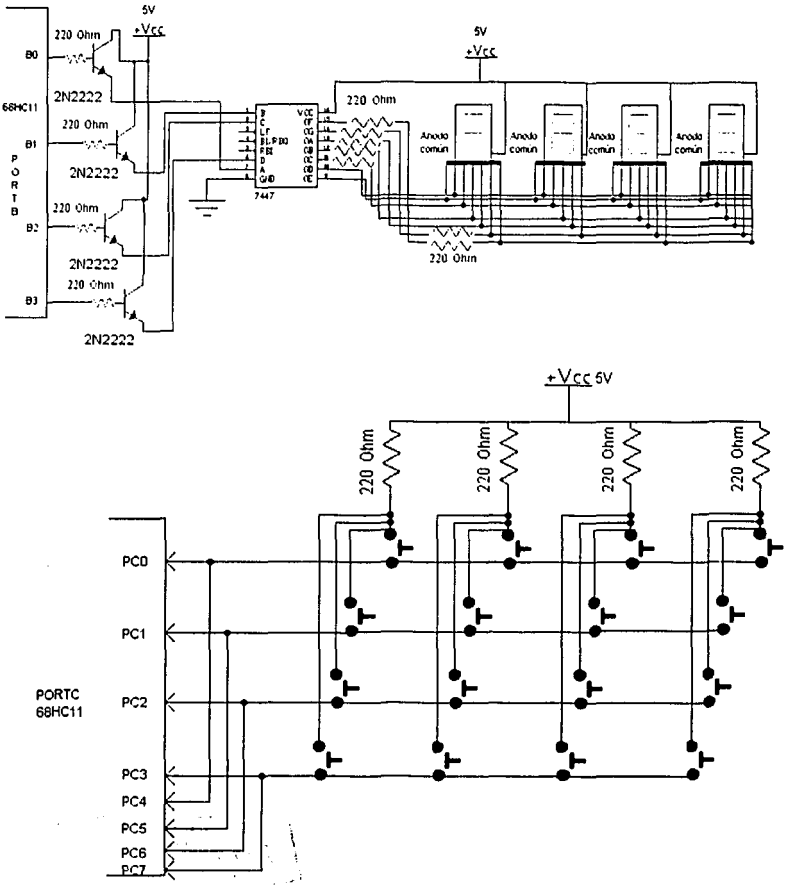


Figura 41. Circuito Eléctrico para lectura de información mediante teclado y desplegar dicha información en 4 display.

La figura 41 muestra la entrada que esta diseñada por una matriz de push bottom, los cuales tienen asignados dígitos del 0-F, se procesan en el microcontrolador y la salida la emite al conectar al Puerto B los bits de salida 0-3, se envía por resistencias que mandan la señal a transistores que entran al decodificador que alimenta un conjunto de 4 display. La figura 42 muestra el circuito físico.

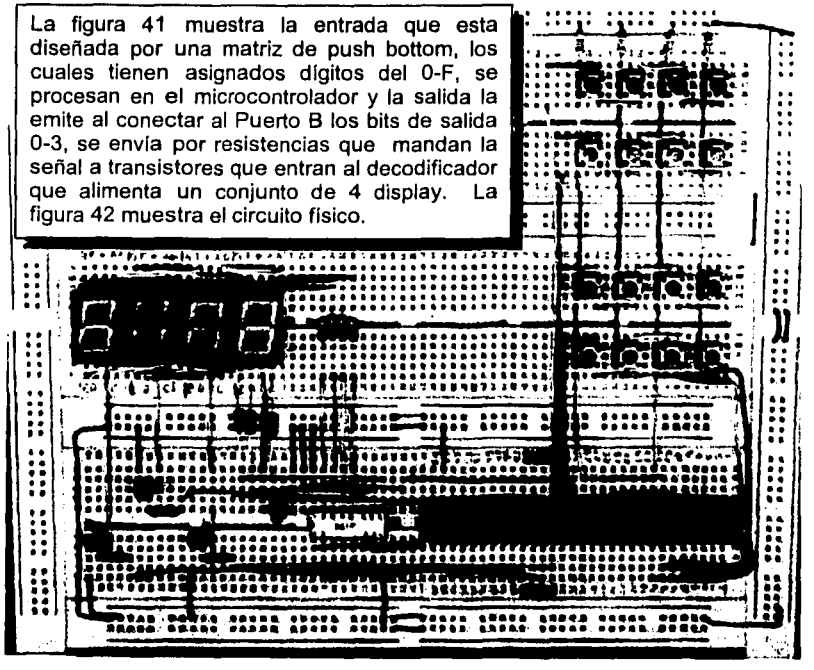


Figura 42. Circuito Eléctrico Físico para lectura de información mediante teclado y desplegar dicha información en 4 display.

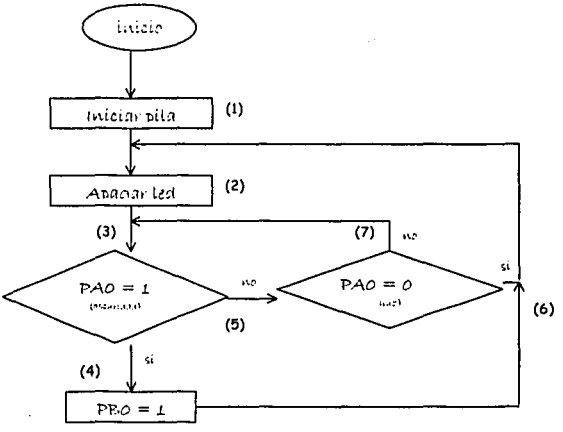
4.9.4. Aplicaciones industriales para lectura de datos de teclado.

Se identifican los datos tomados del teclado y almacenados en RAM, colocándose digito a digito en 4 direcciones de memoria desde la 00 a la 03, en la que para desplegar a los transistores, se les sumo $10, $20, $40, $80 dependiendo de la posición para activar los transistores y la parte baja del acumulador se manda al display, es decir, los transistores son la posición (de los 4 display) y el contenido de la dirección (RAM, RAM+1, RAM+2, RAM+3) es el número que se despliega en su correspondiente display. El principal obstáculo fue la ordenación de los números y regular el tiempo para que el usuario visualice los números y el desplazamiento en FIFO.

4.10. Termómetro I.

Objetivo: Diseñar un sensor de temperatura que identifique oscuridad o luz para encender o apagar una lámpara.

4.10.1. Diagrama de flujo.



4.10.2. Código Fuente.

```
ORG $100
PORTB  EQU $04
PORTA  EQU $00

       LDS #$0047              (1)
       LDX #$1000
INI    CLR PORTB,X             (2)
FO     BRSET PORTA,X,$01,FOCO  (3)
       BRCLR PORTA,X,$01,INI   (5)(6)
       JMP FO                  (7)
FOCO
       LDAA $01
       STAA PORTB,X            (4)
       JSR INI
```

4.7.1. Circuito Eléctrico.

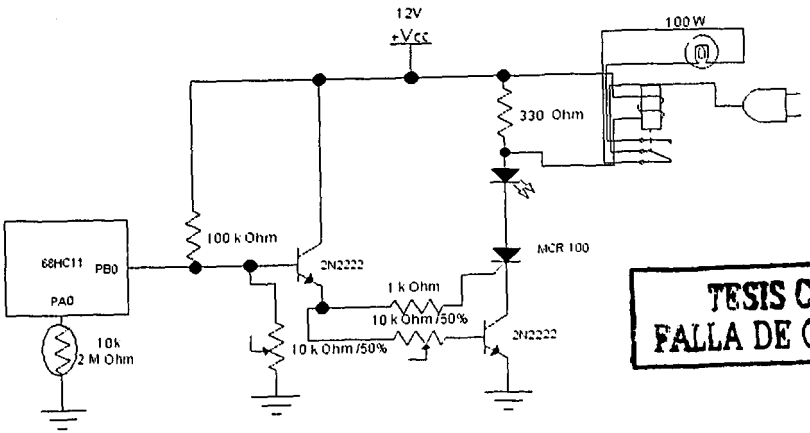
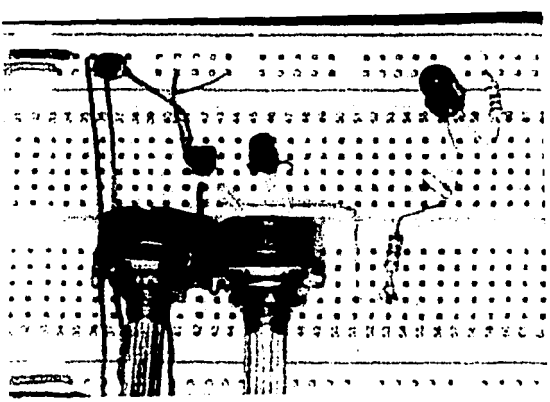


Figura 43. Circuito Eléctrico para sensar oscuridad / luz y activar / desactivar un foco de 100w .



La figura 43 muestra que se conecta al Puerto A, en el bit 0 una fotorresistencia que cambia su nivel de resistencia de acuerdo con la luz recibida, se procesa la información sensante y emite la respuesta por el Puerto B, bit 0; el . cual activa un sistema que además de prender un led, si se le conecta un relevador, entonces activará un foco de 100w. La figura 44 muestra el circuito eléctrico físico.

Figura 44. Circuito Eléctrico Físico para sensar oscuridad / luz y activar / desactivar un foco de 100w .

4.10.4. Aplicaciones industriales para el termómetro.

La utilización de sensores es trascendental en mediciones de ciertos fenómenos ambientales, en éste caso si se detecta luminosidad en la región que abarca la fotorresistencia asociada al puerto A, la lámpara permanecerá sin energizarse, pero si se detecta oscuridad, entonces se energizará la lámpara asociada al puerto B.

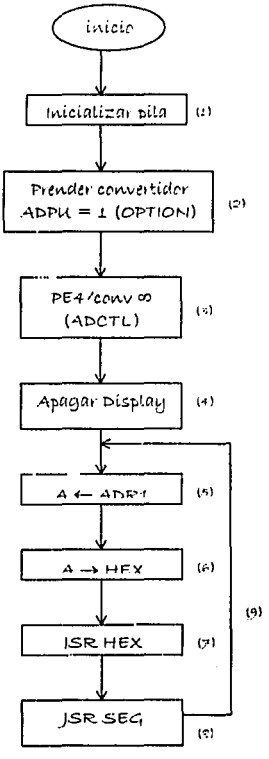
Lo anterior, también es utilizado en exteriores de casa u oficina para ahorrar energía cuando alguna persona no este presente y accione el interruptor de luz, con el mismo objetivo, se puede utilizar otro tipo de sensores como lo son los sensores de presencia, es decir, interrumpir un haz luminoso cuando existe presencia.

En cuestión de seguridad no permite a personas extrañas al inmueble saber si éste esta habitado o no.

4.11. Termómetro II.

Objetivo: Diseñar un sensor de temperatura ambiental y que despliegue en 2 display la temperatura ambiente en grados centígrados, en un rango de 0°C a 99°C.

4.11.1. Diagrama de flujo.



4.11.2. Código fuente.

```
        ORG $100
HEX     EQU $00
DEC     EQU $01
RES     EQU $04
PORTA   EQU $1000
OPTION  EQU $39
ADCTL   EQU $30
ADR1    EQU $31

        LDS #$0047
        LDX #PORTA
        LDAA #$90
        STAA OPTION,X
        LDAB #$24
        STAB ADCTL,X
        CLR 04,X
ESP     BRCLR ADCTL,X,$80,ESP
OTROV   LDAA ADR1,X
        STAA HEX
        JSR HEXDEC
        JSR SEG
        JMP OTROV
HEXDEC
        CLRA
        LDAB HEX
        LDX #!100
        IDIV
        XGDX
        STAB DEC
        XGDX
        LDX #!10
        IDIV
        STAB DEC+2
        XGDX
        STAB DEC+1
        LDAA DEC+1
        ASLA
        ASLA
        ASLA
        ASLA
        ORAA DEC+2
        STAA RES
        LDX #$1000
        STAA 04,X
        RTS
SEG
        LDY #!28560
SIGUE   DEY
        CPY #$00
        BNE SIGUE
        RTS
```

4.11.3. Circuito eléctrico.

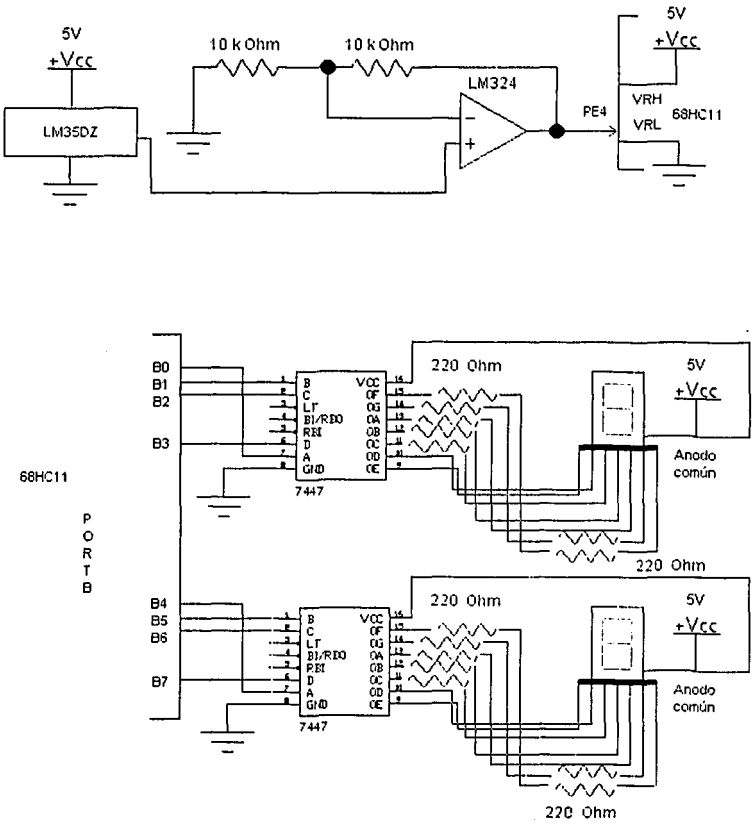


Figura 45. Circuito Eléctrico para sensar temperatura y desplegar la lectura en 2 display que indican un rango de 0°C a 99°C.

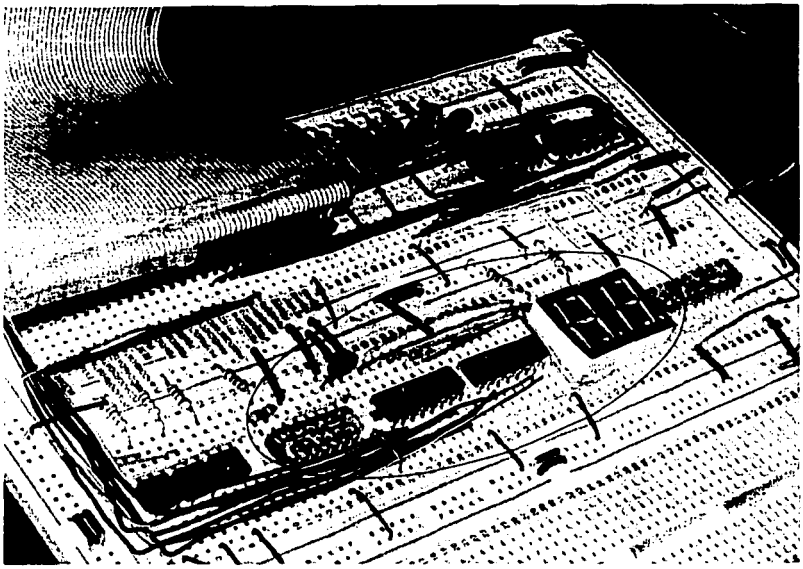


Figura 46. Circuito Eléctrico Físico para sensar temperatura y desplegar la lectura en 2 display que indican un rango de 0°C a 99°C.

La figura 45 muestra que se tiene un sensor de temperatura conectado a un amplificador, el cual envía la señal de entrada al microcontrolador por el Puerto E, bit 4, el cual permite realizar constantes verificaciones de la señal, así se procesa y se da la salida por el Puerto B, de los bits 0-3 para el digito menos significativo y de los bits 4-7 para el dígito más significativo decimal. La figura 46 muestra encerrado en un círculo la etapa del sensor, amplificador hacia la entrada del microcontrolador, como salida después de procesar la información hacia los 2 display.

4.11.4. Aplicaciones industriales para el sensor de temperatura.

El sensor que se utiliza (LM35DZ) proporciona la equivalencia de 20mV por cada grado Centígrado, este voltaje es leído por el PE4, en el cual se tienen lecturas infinitas cada 64μseg y que nunca para. Esta lectura en voltaje el microcontrolador lo convierte en una lectura Hexadecimal, por lo que se realiza la conversión a numeración decimal para enviar la respuesta al decodificador y así la información avance hacia los display.

En algunos procesos industriales es necesario la lectura de la temperatura ambiental como las del propio proceso, como los son fundiciones (en las que se requiere de tecnología que soporte las altas temperaturas), fusión, ebullición o simplemente para verificar la constancia de la temperatura de algún proceso.

Es de suma importancia conocer la temperatura de los procesos para tener control y seguridad sobre el mismo, es por ello que la lectura se presenta al usuario en una escala especifica como los son los grados Centígrados y en forma decimal.

CONCLUSIONES

La utilización de un microprocesador / microcontrolador tiene diferentes ventajas, entre las que destacan:

- ✓ **Aumento de productividad.** La precisión en los procesos y tiempo destinado genera el control en la producción, teniendo un conocimiento exacto de la cantidad de producto a producir y reducción de tiempo para realizar mantenimiento a la maquinaria, ya que la misma se ve reducida en dispositivos, debido a que mayoritariamente es programación, la que controla el proceso productivo .

- ✓ **Optimización de máquinas herramientas.** Cuando el diseño de la máquina – herramienta contiene una gran cantidad de dispositivos electrónicos, se enfrenta el problema de tiempo de vida de los mismos debido a las condiciones físicas en las que se encuentra la maquinaria y para realizar la reparación se requiere del análisis por fases del proceso, hasta detectar el dispositivo que tiene la falla, repercutiendo en tiempo, costo por el dispositivo, además de tener en existencia dicho dispositivo; al emplear la programación en la parte medular del proceso, la falla se limita a revisar el programa y cuando se requiere cambiar el proceso, de igual forma solo se necesita analizar el programa y rediseñar, reduciendo asi el tiempo y evitando reestructuración en diseño lo que resultaría costoso y requeriría tiempo, que al no trabajar la maquinaria genera pérdidas a la empresa.

- ✓ **Mejora de calidad.** El microcontrolador / microrpocesador tiene tiempos de exactitud en μseg, lo cual en el proceso, facilita el cálculo del tiempo que requiere en ejecutarse el proceso, es decir, saber específicamente la cuantificación de verificación de las entradas, el tiempo de proceso de información, y tiempo de respuesta hacia las salidas, además del control de las salidas de acuerdo al comportamiento de las entradas; a lo que repercute en calidad en el proceso.

- ✓ **Disminución de stocks de producto terminado.** Al analizar el comportamiento de ventas en inventario, por medio del microcontrolador / microprocesador, se puede modificar el software para producir solamente lo requerido, evitando exceso de producción.

- ✓ **Disminución en costos de diseño e implementación.** Si se cuenta con un proceso mayoritariamente con dispositivos electrónicos o equipos de control, se tiene la desventaja de que al ocurrir una falla, hay que analizar sistemáticamente cada dispositivo o etapa del proceso, lo que repercute en costo, tiempo, adquisición de nuevos dispositivos (que pueden no estar disponibles en el mercado y recurrir a reemplazos), diseñar para la nueva aplicación y detener el proceso productivo, es por ello, que se muestra la alternativa de utilizar un microprocesador / microcontrolador, que para realizar alguna modificación en el diseño o la implementación del proceso, solo es necesario analizar el software y el problema se limita a la programación y visualizar la secuencia del programa, por lo que se recomienda, documentar el diagrama de flujo y relacionarlo (mediante numeración) con el código fuente, para su manipulación eficiente, reduciendo tiempo y costos puesto que no requiere adquirir nuevos dispositivos.

- ✓ **Realizar trabajos (en especial mediciones) en condiciones peligrosas.** Existen aplicaciones en las que el control sobre la presión, temperatura, peso son significativos para el proceso, por ejemplo, si requiere medir la temperatura de un proceso que oscila entre los –1000°C y 1000°C, representa condiciones peligrosas para los humanos, es por ello, que se recomienda usar un sensor que transmita la información hacia el microprocesador / microcontrolador y éste la procese y emita una respuesta, que como se muestra en la sección 4.10 y 4.11 una verificación continua de 0.25μseg y visualizar en display para la toma de decisiones para el usuario así como puede estar conectado a su vez a alguna válvula si es un líquido o gas que permita mantener la temperatura en optimas condiciones para el proceso productivo. También es aplicable cuando se requiere sistemas inteligentes de ahorro de energía, aprovechando dispositivos electrónicos como lo son los sensores de presencia que informa al microcontrolador / microprocesador y si no hay personal, entonces deshabilitar el sistema de iluminación de la planta.

- ✓ **Producción de diferentes productos sobre una misma línea de producción.** Debido a la precisión que ofrece el microcontrolador / microprocesador, la calidad de la producción aumenta en gran manera, y es relativamente fácil cambiar de tipo de producto o seleccionar sobre varios productos, con creatividad en la programación, proporcionando al usuario alternativas dependiendo de las necesidades productivas.

- ✓ **Accesibilidad.** El microcontrolador / microprocesador proporciona accesibilidad a condiciones a las que el ser humano no puede llegar debido a su peligrosidad, pero también, es accesible, debido a que la programación es modular y sencillo su análisis del código si se tiene un diagrama de flujo claro y relacionado con el código, lo que proporciona accesibilidad para realizar modificaciones o actualizaciones.

- ✓ **Rapidez de respuesta.** El microcontrolador cuenta con capacidad de monitorear sus entradas cada 0.25 μseg y cada instrucción se realiza de 1 ciclo (0.5 μseg) a 3 ciclos (1.5 μseg) dependiendo del modo de direccionamiento con el que se programe; pero relativamente el procesamiento cuenta con la rapidez necesaria para generar salidas hacia diferentes actuadores o monitores de control.

- ✓ **Precisión (resolución, repetitibilidad).** Debido a que el microcontrolador cuenta con velocidad de procesamiento, monitoreo de sus entradas cada 0.25 μseg y con capacidad de respuesta rápida, entonces, genera precisión en el proceso productivo y puede verificar continuamente si existe alguna alteración o condición que afecte el proceso; por lo que se requiere al programar, tomar en consideración los posibles factores afectantes y sus respectivas medidas correctivas, aprovechando así, la precisión que ofrece el microcontrolador.

- ✓ **Interacción con el entorno.** El microcontrolador / microprocesador por los puertos que cuenta permite interactuar con su entorno, verificando su comportamiento y tomando decisiones estipuladas dentro del programa.

- ✓ **Factibilidad económica y técnica.** Como se mencionó anteriormente, el microprocesador / microcontrolador al diseñar, corregir fallas, implementar solo requiere del análisis de la programación, lo que resulta económico y fácil de implementar técnicamente debido a que no se requiere adquirir dispositivos electrónicos mayoritariamente.

- ✓ **Autocontrol en los procesos.** El microcontrolador debido a la capacidad de respuesta y monitoreo constante permite trabajar en forma modular, considerando diferentes condiciones del proceso y generando salidas hacia todas y cada una de las entradas, repercutiendo en control y auto verificación.

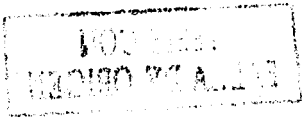
- ✓ **Creación de sistemas semi-inteligentes.** Aprovechando dispositivos electrónicos como lo son los sensores, emiten una señal eléctrica que es transmitida al microcontrolador / microprocesador, éste procesa la información recibida y genera la salida correspondiente, dando la factibilidad de tener dicho sistema en condiciones no aptas para los usuarios o trabajar de forma independiente proporcionándole al sistema "cierta inteligencia" de actuar.

- ✓ **Exactitud en tiempo.** El microcontrolador define exactamente la duración de cada monitoreo de entradas (0.25 μseg), duración de cada instrucción dependiendo del modo de direccionamiento que oscila de 1 a 3 ciclos de reloj (0.5 a 1.5 μseg), de tal suerte, que se puede calcular exactamente el tiempo de duración desde la entrada – proceso – salida para cada aplicación diseñada.

sencillez (display, teclado numérico), velocidad en RAM definida contra precisión y exactitud en tiempo (gran capacidad de respuesta).

Es importante señalar que la creatividad del programador puede ser mucho más poderosa que sistemas complejos electrónicos, además de que para hacer modificaciones solo es necesario actualizar el código fuente y no la circuiteria o incluso realizar un diseño completamente distinto. Así como, el costo en el proyecto disminuye en gran manera y a su vez el tiempo de vida del proyecto aumenta considerablemente porque solo una parte del proyecto esta bajo condiciones de temperatura, humedad, variaciones de corriente que pueden afectar a los dispositivos.

Con las aplicaciones mostradas se puede dar una visión sobre la potencialidad del microprocesador o el microcontrolador (según sea el caso), de tal forma que son sólo el principio de una gran tecnología que ya esta implementando en la industria desde hace algún tiempo y que a los estudiantes permite usar la creatividad para solucionar futuros problemas o crear nuevas tecnologías.



BIBLIOGRAFÍA

1. http://www.dtplan.com/garcia-cuervo/delco.htm
2. http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des
3. MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
4. Lenguaje Ensamblador para PC IBM y Compatible, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995.
5. Intel Microprocessors:Hardware, Software and Aplicattions, Godoy, EU, Mc Graw Hill, 1993.
6. http://members.tripod.com/~MoisesRBB/asm.html (importante)
7. http://www.geocities.com/rafael.sanchez/es/68hc11.html
8. http://www.geocities.com/eidan.rm/conbasicos3.htm
9. http://www.monografias.com/trabajos7/regi/regi.shtml
10. http://www.itlp.edu.mx/publica/tutoriales/ensamblador/portada.htm
11. http://www.topptutoriales.com
12. Manual de Referencia Técnica de Motorola.
13. http://www.geocities.com/albert_m_98/Robot.htm
14. http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
15. http://www.dtplan.com/68HC11/
16. Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991.
17. "Microprocesador." Enciclopedia® Microsoft® Encarta 2001. © 1993-2000 Microsoft Corporation. Reservados todos los derechos.
18. http://members.es.tripod.de/tutorial/index.html/.

Referencias Bibliográficas

Capítulo I. "CARACTERÍSTICAS DE HARDWARE DEL MICROCONTROLADOR 68CH11 DE MOTOROLA".

1.

a) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995.
b) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991.
c) "Microprocesador." Enciclopedia® Microsoft® Encarta 2001. © 1993-2000 Microsoft Corporation. Reservados todos los derechos.

2.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
c) Manual de Referencia Técnica de Motorola.
d) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des
e) "Microcontrolador." Enciclopedia® Microsoft® Encarta 2001. © 1993-2000 Microsoft Corporation. Reservados todos los derechos.

3.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
c) Manual de Referencia Técnica de Motorola.
d) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

4.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.

5.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
c) Manual de Referencia Técnica de Motorola.
d) http://www.geocities.com/albert_m_98/Robot.htm
e) http://www.dtplan.com/68HC11/
f) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

Capítulo II. . "CARACTERÍSTICAS DE SOFTWARE DEL MICROCONTROLADOR 68CH11 DE MOTOROLA".

1.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

2.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

3.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

4.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

5.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/
b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

6.

a) MICROCONTROLADOR 68HC11: Fundamentos, recursos y Programación, Cristina Doblado Alcázar, Juan González Gómez, Andrés Prieto – Moreno, Juan José San Martín, http://www.arrakis.es/~microbot/

b) Manual de Referencia Técnica de Motorola.
c) http://www.unizar.es/euitiz/areas/aretecel/mpeie/kit11/kit11.pdf
d) Manual de Referencia Técnica de Motorola.
e) http://www.geocities.com/albert_m_98/Robot.htm
f) http://www.dtplan.com/68HC11/
g) http://translate.google.com/translate?hl=es&sl=en&u=http://www.hc11.demon.nl/thrsim11/68hc11/&prev=/search%3Fq%3D68hc11%26hl%3Des

Capítulo III. "LENGUAJE ENSAMBLADOR PARA INTEL".

1. Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 2-3.

2.
a) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 3-6.
b) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem1_1_.htm

3.
a) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 6-7.
b) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 8-13.

4.
a) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 9-10.
b) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/unidad2.htm

5.
a) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 10-17.
b) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 23-30.
c) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem2_1_1.htm
d) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem3_1.htm

6.

a) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 31-40.
b) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 29-30, 34-37.
c) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem5_1_.htm

7.

a) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem6_1_.htm
b) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 95-98.
c) Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 41-49.

8.

a) Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 50-57.
b) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 95-97.
c) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem6_3_.htm

9.

a) Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 50-57.
b) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 90-95.
c) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem6_2_.htm

10.

a) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 48, 64-73.
b) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem6_7_.htm
c) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 113-123.

11.

a) Lenguaje Ensamblador para microcomputadoras IBM para principiantes y avanzados, J. Terry Godfrey, ed. Prentice Hall, México, 1991, páginas 48, 143-183.
b) Lenguaje ensamblador y programación para PC IBM y compatibles, Peter Abel, 3ª. Edición, Ed. Prentice Hall, México, 1995, páginas 185-198.
c) Archivo descargado de:

http://members.es.tripod.de/tutorial/index.html/, páginas 74-101.

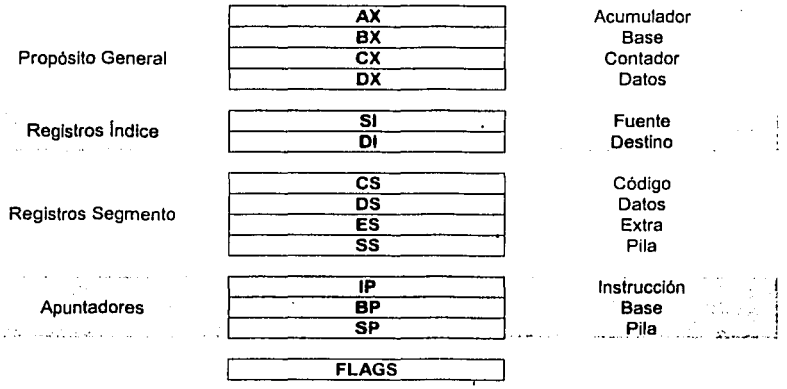
d) http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_1_.htm, http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_2_.htm, http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_3_.htm, http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_4_.htm, http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_5_.htm, http://www.itlp.edu.mx/publica/tutoriales/ensamblador/tem4_6_.htm

12. Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 28-35.

13. Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 114-120.

14. Archivo descargado de: http://members.es.tripod.de/tutorial/index.html/, páginas 120-128.

ANEXO I. REGISTROS DE LA C.P.U. (Intel).



Modos de Direccionamiento.

Son las formas que tiene el microprocesador para obtener datos o guardarlos en la memoria. Mientras más modos de direccionamiento tiene un microprocesador, éste es más versátil porque cada instrucción se puede ejecutar de diferentes maneras.

INMEDIATO	MOV AX, 1000	; mueve el dato 1000 a AX
DIRECTO	MOV AX, [1000]	; mueve la dirección 1000 a AX
INDIRECTO	MOV AX,[SI]	; apunta a una dirección y busca lo que hay dentro de la dirección
REGISTRO	MOV DS,AX	; copia el contenido de un registro a otro
INDEXADO	MOV AX,[SI+5]	; se le agrega una constante
BASE + INDEXADO	MOV AX,[BX,SI]	
BASE + INDEXADO + DESPLAZAMIENTO	MOV AX,[BX,DI+2]	
CADENAS	MOVS, LODS, STOS, SCAS	
INHERENTE O IMPLÍCITO	INC AX	; no requiere operando.

ANEXO II. Resumen de los registros de control del 68HC11

	BIT 7	6	5	4	3	2	1	0	
$1000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PORTA
$1001									sin usar
$1002	STAF	STAI	CWOM	HDNS	OIN	PLS	EGA	INVB	PIOC
$1003	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	PORTC
$1004	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	PORTB
$1005	PCT7	PCT6	PCT5	PCT4	PCT3	PCT2	PCT1	PCT0	PORTCL
$1006									sin usar
$1007	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0	DDRC
$1008	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	PORTD
$1009	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	DDRD
$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	PORTE
$100B	FOC1	FOC2	FOC3	FOC4	FOC5				CFORC
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3				OC1M
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				OC1D
$100E	BIT 15							BIT 8	TCNT high
$100F	BIT 7							BIT 0	TCNT low
$1010	BIT 15							BIT 8	TIC1 high
$1011	BIT 7							BIT 0	TIC1 low
$1012	BIT 15							BIT 8	TIC2 high
$1013	BIT 7							BIT 0	TIC2 low
$1014	BIT 15							BIT 8	TIC3 high
$1015	BIT 7							BIT 0	TIC3 low
$1016	BIT 15							BIT 8	TOC1 high
$1017	BIT 7							BIT 0	TOC1 low
$1018	BIT 15							BIT 8	TOC2 high
$1019	BIT 7							BIT 0	TOC2 low
$101A	BIT 15							BIT 8	TOC3 high

$101C	BIT 15							BIT 8	TOC4 high
$101D	BIT 7							BIT 0	TOC4 low
$101E	BIT 15							BIT 8	TOC5 high
$101F	BIT 7							BIT 0	TOC5 low
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1
$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2
$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I	TMSK1
$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F	TFLG1
$1024	TOI	RTII	PAOVI	PAII			PRI	PR0	TMSK2
$1025	TOF	RTIF	PAOVF	PAIF					TFLG2
$1026	DDRA7	PAEN	PAMOD	PEDGE			RTR1	RTR2	PACTL
$1027	BIT 7							BIT 0	PACNT
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
$1029	SPIF	WCOL		MODF					SPSR
$102A	BIT 7							BIT 0	SPDR
$102B	TCLR		SCP1	SCP0	RCK	SCR2	SCR1	SCR0	BAUD
$102C	R8	T8		M	WAKE				SCCR1
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE		SCSR
$102F	BIT 7							BIT 0	SCDR
$1030	CCF		SCAN	MULT	CD	CC	CB	CA	ADCTL
$1031	BIT 7							BIT 0	ADR1
$1032	BIT 7							BIT 0	ADR2
$1033	BIT 7							BIT 0	ADR3
$1034	BIT 7							BIT 0	ADR4
$1035									sin usar
$1036									sin usar

$1037									sin usar
$1038									sin usar
$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0	OPTION
$103A	BIT 7							BIT 0	COPRST
$103B	ODD	EVEN		BYTE	ROW	ERASE	EELAT	EPGM	PPROG
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT
$103E	TILOP		OCCR	CBYP	DISR	TCMI	TCOP	TCON	TEST1
$103F					NOSEC	NOCOP	ROMON	EEON	CONFIG

Anexo III. Descripción de todos los registros de control del 68HC11.

ADCTL

REGISTRO DE STATUS/CONTROL DE CONVERSOR A/D

$1030	CCF		SCAN	MULT	CD	CC	CB	CA

- CCF - Conversión completa
- SCAN - Control del muestreo continuo
  - 0 - Cuatro conversiones y parar
  - 1 - Realizar conversiones continuamente
- MULT - Control del número de canales de conversión
  - 0 - Conversión de un único canal
  - 1 - Conversión de un grupo de cuatro canales
- CD - CA: Selección del los canales:

CD	CC	CB	CA	Señal del canal	Resultado en ADRx si MULT = 1
0	0	0	0	AD0 por bit 0 puerto E	ADR1
0	0	0	1	AD1 por bit 1 puerto E	ADR2
0	0	1	0	AD2 por bit 2 puerto E	ADR3
0	0	1	1	AD3 por bit 3 puerto E	ADR4
0	1	0	0	AD4 por bit 4 puerto E	ADR1
0	1	0	1	AD5 por bit 5 puerto E	ADR2
0	1	1	0	AD6 por bit 6 puerto E	ADR3
0	1	1	1	AD7 por bit 7 puerto E	ADR4
1	0	0	0	Reservado	ADR1
1	0	0	1	Reservado	ADR2
1	0	1	0	Reservado	ADR3
1	0	1	1	Reservado	ADR4
1	1	0	0	Vref hi	ADR1
1	1	0	1	Vref low	ADR2
1	1	1	0	Vref hi /2	ADR3
1	1	1	1	test	ADR4

ADR1-ADR4

REGISTROS DE RESULTADOS DE CONVERSIÓN A/D

$1031	BIT 7							BIT 0	ADR1
$1032	BIT 7							BIT 0	ADR2
$1033	BIT 7							BIT 0	ADR3
$1034	BIT 7							BIT 0	ADR4

BAUD

Registro de control de los baudios del SCI

$102B	TCLR		SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
RESET:	0	0	0	0	0	U	U	U

- TCLR = Uso exclusivo de test
- RCKB = Uso exclusivo de test
- SCP1,SCP0 = Divisor de velocidad
- SCR2-SCR0 = Selección de velocidad

CFORC

Registro para forzar los comparadores

$100B	FOC1	FOC2	FOC3	FOC4	FOC5			
RESET:	0	0	0	0	0	0	0	0

Escribir un "1" para forzar el comparador correspondiente

CONFIG

Registro de configuración y control

$103F					NOSEC	NOCOP	ROMON	EEON

- NOSEC = Deshabilitación del modo de seguridad
  - 0 = Modo de seguridad
  - 1 = No hay seguridad
- NOCOP = Deshabilitar el sistema COP
  - 0 = COP habilitado
  - 1 = COP deshabilitado
- ROMON = Habilitación de la memoria ROM
  - 0 = La ROM no está en el mapa de memoria
  - 1 = La ROM está en el mapa de memoria
- EEON = habilitación de la memoria EEPROM
  - 0 = La EEPROM no está en el mapa de memoria
  - 1 = La EEPROM está en el mapa de memoria

Este registro está implementado con bits del tipo EEPROM por lo que no será posible escribirlos igual que el resto de los registro. Para modificar los bits habrá que hacerlo de igual forma que para modificar la memoria EEPROM.

COPRST

Temporizador del circuito del COP

$103A	BIT 7							BIT 0

Escribir $55 y $AA para inicializar el temporizador del watchdog

DDRC

Dirección de los datos del puerto C

$1007	DDC7	DDC6	DDC 5	DDC4	DDC3	DDC2	DDC1	DDC0
RESET	0	0	0	0	0	0	0	0

0 = Entradas
1 = Salidas

DDRD

Dirección de los datos del puerto D

$1009			DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
RESET	0	0	0	0	0	0	0	0

0 = Entradas
1 = Salidas

HPRIO

$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0
RESET	----	---	---	---	0	1	0	1

- RBOOT = Habilitación de la rom de arranque (Sólo se puede escribir si SMOD = 1)
  0 = ROM de arranque no se encuentra en el mapa de memoria
  1 = ROM de arranque presente en el mapa de memoria

- SMOD: Indica si se esta en un modo especial o no. En los modos especiales existen privilegios especiales.
  0 : Modo no especial
  1 : Modo especial

- MDA: Selección del modo.

SMOD	MDA	MODO DE FUNCIONAMIENTO
0	0	Single chip
0	1	Expanded
1	0	Special bootstrap
1	1	Special test

- IRV = Internal Read Visibility
  0 = No hay visibilidad de las lecturas internas a través del bus externo
  1 = Los datos de las lecturas internas se dirigen al bus de datos externo

- PSEL3-PSEL0 : Selección de prioridad de las interrupciones. (Sólo se pueden escribir si las interrupciones están deshabilitadas). La interrupción indicada pasará a tener la máxima prioridad sobre las demás

PSEL3	PSEL2	PSEL1	PSEL0	Aumentar prioridad a la fuente de interrupcion:
0	0	0	0	Overflow del temporizador
0	0	0	1	Overflow en el acumulador de pulsos
0	0	1	0	Acumulador de pulsos
0	0	1	1	Transferencia completada en el SPI
0	1	0	0	Sistema SCI
0	1	0	1	Reservado
0	1	1	0	IRQ
0	1	1	1	Interrupción en tiempo real
1	0	0	0	Capturador de entrada 1
1	0	0	1	Capturador de entrada 2
1	0	1	0	Capturador de entrada 3
1	0	1	1	Comparador 1
1	1	0	0	Comparador 2
1	1	0	1	Comparador 3
1	1	1	0	Comparador 4
1	1	1	1	Comparador 5

INIT

Mapeado de la RAM y de los registros de control

$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0
RESET	0	0	0	0	0	0	0	1

- RAM3-RAM0 = Mapeado de la memoria RAM
- REG3-REG0 = Mapeado de los registros de control

OCTD

Registro de datos del comparador 1

$100D	OCTD7	OCTD6	OCTD5	OCTD4	OCTD3			

Si OCTMx esta activo, el dato situado en OCTDx se envía al bit_x del Puerto A cada vez que ocurre una comparación correcta.

OCTM

Registro de mascara del comparador 1

$100C	OCTM7	OCTM6	OCTM5	OCTM4	OCTM3			
RESET	0	0	0	0	0	0	0	0

Activar el bit correspondiente para permitir que el comparador 1 actúe sobre el bit del puerto A indicado.

OPTION

Opciones de configuración del sistema

$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0
RESET	0	0	0	1	0	0	0	0

- ADPU - Alimentación del conversor A/D
  - 0 - Conversor A/D no alimentado
  - 1 - Conversor A/D alimentado
- CSEL - Selección del reloj
  - 0 - Conversor A/D y EEPROM utilizan la señal de reloj E
  - 1 - Conversor A/D y EEPROM utilizan un reloj interno RC
- IRQE - Selección del flanco activo en interrupción IRQ (Time protected)
  - 0 - IRQ activa a nivel bajo
  - 1 - IRQ activa en flancos de baja
- DLY - Retraso en el arranque del oscilador al salir de una situación de STOP
  - 0 - No hay pausa
  - 1 - Pausa oblidada
- CME - Activación del monitor del reloj
  - 0 - No habilitado
  - 1 - Un paro o un descenso de velocidad en la señal de reloj provoca un reset
- CR1,CR0 - Selección del timeout del sistema COP

Para un cristal de 8MHZ:

CR0	CR1	Timeout
0	0	16.384 ms
0	1	65.536 ms
1	0	262.14 ms
1	1	1.049 s

PACNT

Contador del acumulador de pulsos

$1027	PCNT7	PCNT6	PCNT5	PCNT4	PCNT3	PCNT2	PCNT1	PCNT0

Este registro se puede leer y escribir.

PACTL

Control del acumulador de pulsos

$1026	DDRA7	PAEN	PAMOD	PEDGE			RTR1	RTR0
RESET	0	0	0	1	0	0	0	0

- DDRA7 - Dirección de datos del pin 7 del puerto A
  - 0 - Entrada
  - 1 - Salida
- PAEN - Habilitación del contador de pulsos

0 = Desactivado
1 = Activado

- PAMOD = Modo de funcionamiento del acumulador de pulsos
  0 = Modo cuenta de pulsos
  1 = Modo duración
- PEDGE = Flanco de control del acumulador de pulsos
  0 = Flanco de bajada.
  1 = Flanco de subida
- RTR1,RTR0 = Periodo de activación de la interrupción de tiempo real

Para un cristal de 8 MHZ:

RTR1	RTR0	Se produce la interrupción cada:
0	0	4.10 ms
0	1	8.19 ms
1	0	16.38 ms
1	1	32.77 ms

PIOC

Registro de control de la E/S paralela

$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
RESET=	0	0	0	0	0	U	1	1

- STAF = Señal de validación A
  0 = Inactiva
  1 = Se activa con los flancos activos del pin STRA
- STAI = Permiso de interrupción de la señal de validación A
  0 = No se genera interrupción
  1 = Se solicita interrupción cuando STAF = 1
- CWOM = Modo de colector abierto para el puerto C
  0 = Salida normal del puerto C
  1 = Salida en colector abierto
- HNDS = Selección del modo de funcionamiento del protocolo
  0 = Modo de validación simple
  1 = Modo "full handshake"
- OIN = Selección de entrada salida
  0 = Entrada
  1 = Salida
- PLS = Selección del modo de pulsos para la salida STRB
  0 = STRB activo a nivel alto
  1 = Pulsos de STRB
- EGA = Selección del flanco activo para la señal STRA
  0 = Flanco de bajada
  1 = Flanco de subida
- INV = Inversión de la salida STRB
  0 = STRB Activo a nivel bajo
  1 = STRB Activo a nivel alto

PORTA

$1000	PA7 PAI OC1	PA6 OC2 OC1	PA5 OC3 OC1	PA4 OC4 OC1	PA3 OC5 OC1	PA2 IC1	PA1 IC2	PA0 IC3
RESET-	---	0	0	0	0	----	----	---

PORTB

$1004	PB7 A15	PB6 A14	PB5 A13	PB4 A12	PB3 A11	PB2 A10	PB1 A9	PB0 A8
RESET	0	0	0	0	0	0	0	0

PORTC

$1003	PC7 A7 D7	PC6 A6 D6	PC5 A5 D5	PC4 A4 D4	PC3 A3 D3	PC2 A2 D2	PC1 A1 D1	PC0 A0 D0

PORTCL

$1005	PCL7	PCL6	PCL5	PCL4	PCL3	PCL2	PCL1	PCL0

PORTD

$1008			PD5 SS#	PD4 SCK	PD3 MOSI	PD2 MISO	PD1 TxD	PD0 RxD

PORTE

$100A	PE7 AN7	PE6 AN6	PE5 AN5	PE4 AN4	PE3 AN3	PE2 AN2	PE1 AN1	PE0 AN0

PPROG

Programación de la EEPROM

$103B	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
RESET =	0	0	0	0	0	0	0	0

- ODD : programa líneas impares (Solo usado en modo TEST)
- EVEN : programa líneas pares (Solo usado en modo TEST)
- BIT 5 : Siempre es cero
- BYTE : Selección de borrado de un byte (Este bit tiene prioridad sobre el bit ROW)
  - '0' Se ha seleccionado modo de borrado de fila (ROO) o total (BULA)
  - '1' Se ha seleccionado modo de borrado de un byte
- ROO : Selección de borrado ROO (es decir de una fila)
  (Si el bit BYTE está 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 Programación Borrado
- EEPGM: Activación o desactivación del voltaje de programación de la EEPROM
  - '0' Voltaje de programación ON.
  - '1' = Voltaje de programacion OFF.

SCCR1

Registro 1 de control del SCI

$102C	R8	T8		M	WAKE			
RESET	U	U	0	0	0	0	0	0

- R8 = Recibir el bit 8
- T8 = Transmitir el bit 8
- M = Configuración del formato de datos del SCI
  - 0 = 1 bit de start, 8 bits de datos y 1 bit de stop
  - 1 = 1 bit de start, 9 bits de datos y 1 bit de stop
- WAKE = Activación del SCI
  - 0 = Activación por línea libre
  - 1 = Activación por marca

SCCR3

Registro 2 de control del SCI

$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET	0	0	0	0	0	0	0	0

- TIE – Permitir interrupción de Listo para transmitir
- TCIE – Permitir interrupción de transmisión completa
- RIE – Permitir interrupción de carácter recibido
- ILIE – Permitir interrupción por línea libre
  - 0 – No interrupción
  - 1 – Interrupción permitida
- TE – Habilitación del transmisor
- RE – Habilitación del receptor
  - 0 – Off
  - 1 – On
- RWU – Control de la activación del receptor
  - 0 – Normal
  - 1 – Receptor dormido
- SBK – Enviar señal de BREAK

SCDR

Registro de datos del SCI

$102F	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

SCSR

Registro de Status del SCI

$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	
RESET	1	1	0	0	0	0	0	0

- TDRE – Flag de listo para enviar
- TC – Flag de transmisión completa
- RDRF – Flag de carácter recibido
- IDLE – Flag de línea libre
- OR – Error de overrun
- NF – Error por ruido. Dato recibido puede estar mal
- FE – Error en la trama recibida.

SPCR

Registro de control del SPI

$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0
RESET -	0	0	0	0	0	1	U	U

- **SPIE**: Serial Peripheral Interrupt Enable.
  1 Habilita las interrupciones del SPI.
  0 Deshabilita las interrupciones del SPI.
- **SPE**: Serial Peripheral System Enable.
  1 Activa el SPI.
  0 Desactiva el SPI.
- **DWOM**: Port D Wire-Or Mode Option
  1 La totalidad del puerto D actúa con salidas en colector abierto.
  0 La totalidad del puerto D actúa con salidas cmos.
- **MSTR**: Master Mode Select.
  1 Configurado el SPI en modo Maestro.
  0 Configurado el SPI en modo Esclavo.
- **CPOL**: Clock Polarity.
  1 = El reloj se mantiene a nivel alto mientras no existan datos a transmitir.
  0 = El reloj se mantiene a nivel bajo mientras no existan datos a transmitir.
- **CPHA**: Clock Phase.
  Se refiere a los dos formatos de sincronismos
- **SPR1, SPR0**: SPI Clock Rate Select.
  Permiten seleccionar la velocidad de transmisión según la siguiente tabla.

SPR1	SPR0	Reloj interno E dividido por:
0	0	2
0	1	4
1	0	16
1	1	32

SPDR

Registro de datos del SPI

$102A	SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0

SPSR

Registro de status del SPI

$1029	SPIF	WCOL		MODF				
RESET	0	0	0	0	0	0	0	0

- SPIF Flag de listo para enviar
- WCOL Flag de colisión en escritura
- MODF Flag de error

TCNT

Temporizador principal

TCNT (high)

$100E	CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8

TCNT (low)

$100F	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0

Registro de sólo lectura

TCTL1

Registro de control de los comparadores

$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5
RESET	0	0	0	0	0	0	0	0

Omx	OLx	Acción a realizar al producirse una comparación
0	0	Ninguna acción
0	1	Cambiar de estado el pin correspondiente
1	0	Poner a cero el pin correspondiente
1	1	Poner a uno el pin correspondiente

TCTL2

Registro de control de los capturadores

$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A
RESET	0	0	0	0	0	0	0	0

EDGxB	EDGxA	Configuración
0	0	Capturador deshabilitado
0	1	Capturar en flanco de subida
1	0	Capturar en flanco de bajada
1	1	Capturar en flanco de subida y bajada

TEST1

Registro de pruebas de fábrica

$103E	TIEOP		OCCR	CBYP	DISR	FCM	FCOP	TCON
RESET	0	0	0	0	---	0	0	0

- TIEOP Prueba de instrucción ilegal
- OCCR Mandar el registro CCR al puerto del temporizador
- CBYP Timer divider Chain Bypass
- FCM Forzar un error en el monitor del reloj
- FCOP Forzar un fallo en el watchdog
- TCON Configuración del test

TFLG1

Registro 1 de flags del temporizador

$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F
RESET	0	0	0	0	0	0	0	0

- OC1F-OC5F Flag del comparador 'x'
- IC1F-IC3F Flag del capturador 'x'

Al escribir un '1' en los bits, se podrán a cero los correspondientes flags.

TFLG2

$1025	TOF	RTIF	PAOVF	PAIF				
RESET	0	0	0	0	0	0	0	0

- TOF = Flag de overflow en el temporizador
- RTIF = Flag de la interrupción de tiempo real
- PAOVF Flag de overflow en el acumulador de pulsos
- PAIF = Flag de flanco detectado en el acumulador de pulsos

TIC1-TIC3 Registros de datos de los capturadores

TIC1 (high)

$1010	IC115	IC114	IC113	IC112	IC111	IC110	IC19	IC18

TIC1 (low)

$1011	IC17	IC16	IC15	IC14	IC13	IC12	IC11	IC10

TIC2 (high)

$1012	IC215	IC214	IC213	IC212	IC211	IC210	IC29	IC28

TIC2 (low)

$1013	IC27	IC26	IC25	IC24	IC23	IC22	IC21	IC20

TIC3 (high)

$1014	IC315	IC314	IC313	IC312	IC311	IC310	IC39	IC38

TIC3(low)

$1015	IC37	IC36	IC35	IC34	IC33	IC32	IC31	IC30

TMSK1 Registro de máscara de interrupción del temporizador

$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I
RESET	0	0	0	0	0	0	0	0

- OC1I-OC5I Permitir interrupción del comparador 'x'
- IC1I-IC3I Permitir interrupción del capturador 'x'
  - 0 Interrupción inhibida

1 = Interrupción permitida

TMSK2

$1024	TOI	RTII	PAOVI	PAII			PR1	PR0
RESET	0	0	0	0	0	0	0	0

- TOI – Activación de la interrupción de overflow en el temporizador
- RTII – Activación de la interrupción de tiempo real
- PAOVI – Activación de la interrupción de overflow en acumulador de pulsos
- PAII – Activación de la interrupción del acumulador de pulsos
  - 0 – Interrupción inhibida
  - 1 – Interrupción permitida
- PR1,PR0 – Selección del factor de división de la señal de reloj del temporizador

PR1	PR0	Dividir E entre:
0	0	1
0	1	4
1	0	8
1	1	16

TOC1-TOC5

Registros de datos de los comparadores

TOC1 (high)

$1016	OC115	OC114	OC113	OC112	OC111	OC110	OC19	OC18

TOC1 (low)

$1017	OC17	OC16	OC15	OC14	OC13	OC12	OC11	OC10

TOC2 (high)

$1018	OC215	OC214	OC213	OC212	OC211	OC210	OC29	OC28

TOC2 (low)

$1019	OC27	OC26	OC25	OC24	OC23	OC22	OC21	OC20

TOC3 (high)

$101A	OC315	OC314	OC313	OC312	OC311	OC310	OC39	OC38

TOC3(low)

$101B	OC37	OC36	OC35	OC34	OC33	OC32	OC31	OC30

TOC4 (high)

$101C	OC415	OC414	OC413	OC412	OC411	OC410	OC49	OC48

TOC4(low)

$101D	OC47	OC46	OC45	OC44	OC43	OC42	OC41	OC40

TOC5 (high)

$101E	OC515	OC514	OC513	OC512	OC511	OC510	OC59	OC58

TOC5(low)

$101F	OC57	OC56	OC55	OC54	OC53	OC52	OC51	OC50

ANEXO IV. LISTADO DE LOS MNEMÓNICOS DEL 68HC11.

ABA : Añadir el contenido del acumulador B al acumulador A.
ABX: Añadir el contenido del acumulador B (sin signo) al contenido del registro X.
ABY: Añadir el contenido del acumulador B (sin signo) al contenido del registro Y.
ADCA: Añadir al acumulador A un dato y el bit de acarreo.
ADCB: Añadir al acumulador B un dato y el bit de acarreo.
ADDA: Añadir un dato al registro A.
ADDB: Añadir un dato al registro B.
ADDD: Añadir un dato de 16 bits al registro D.
ANDA: Realizar una operación lógica AND entre un dato y el acumulado A, Resultado en A.
ANDB: Realizar una operación lógica AND entre un dato y el acumulador B, Resultado en B.
ASLA: Desplazar un bit a la izquierda el acumulador A.
ASLB: Desplazar un bit a la izquierda el acumulador B.
ASLD: Desplazar un bit a la izquierda el acumulador D.
ASRA: Desplazar un bit a la derecha el acumulador A.
ASRB: Desplazar un bit a la derecha el acumulador B.
BCC: Saltar si no hay acarreo.
BCLR: Poner a cero bits de la memoria.
BCS: Saltar si hay acarreo.
BEQ: Saltar si igual.
BGE: Saltar si mayor que o igual a cero.
BGT: Saltar si mayor que cero.
BHI: Saltar si es mayor.
BHS: Saltar si mayor o igual.
BITA: Comprobar el bit especificado del acumulador A.
BITB: Comprobar el bit especificado del acumulador B.
BLE: Saltar si menor que o igual a cero.
BLO: Saltar si menor (Mismo que BCS).
BLS: Saltar si menor o igual.
BLT: Saltar si menor que cero.
BMI: Saltar si negativo.
BNE. Saltar si no es igual.
BPL: Saltar si es positivo.
BRA: Salto incondicional.
BRCLR: Saltar si los bits especificados están a cero.
BRN: No saltar nunca (Equivalente a una operación NOP de 2 bytes).
BRSET: Saltar si los bits especificados están a uno.
BSET: Poner los bits especificados a uno.
BSR: Saltar a una subrutina.
BVC: Saltar si no ha habido overflow.
BVS: Saltar si ha habido overflow.
CBA: Comparar el acumulador A con el B.

CLC: Poner a cero bit de acarreo.
CLI: Permitir las interrupciones.
CLR: Poner a cero el contenido de memoria especificado.
CLRA: Poner a cero el acumulador A.
CLRB: Poner a cero el acumulador B.
CLV: Poner a cero el bit de overflow.
CMPA: Comparar el acumulador A con un dato.
CMPB: Comparar el acumulador B con un dato.
COMA: Complementar a uno el acumulador A.
COMB: Complementar a uno el acumulador B.
COM: Complementar a uno el contenido de memoria especificado.
CPD: Comparar el registro D con un dato.
CPX: Comparar el registro X con un dato.
CPY: Comparar el registro Y con un dato.
DAA: Ajuste decimal.
DEC: Decrementar una posición de memoria especificada.
DECA: Decrementar el acumulador A.
DECB: Decrementar el acumulador B.
DES: Decrementar el puntero de pila SP.
DEX: Decrementar el registro X.
DEY: Decrementar el registro Y.
EORA: Operación XOR entre un dato y el acumulador A.
EORB: Operación XOR entre un dato y el acumulador B.
FDIV: División.
IDIV: División entera.
INC: Incrementar el contenido de una posición de memoria.
INCA: Incrementar el acumulador A.
INCB: Incrementar el acumulador B.
INS: Incrementar el puntero de pila SP.
INX: Incrementar el registro X.
INY: Incrementar el registro Y.
JMP: Salto incondicional.
JSR: Salto a una subrutina.
LDAA: Cargar un dato en el acumulador A.
LDAB: Cargar un dato en el acumulador B.
LDD: Cargar un dato de 16 bits en el registro D.
LDS: Cargar un dato de 16 bits en el puntero de pila SP.
LDX: Cargar un dato de 16 bits en el registro X.
LDY: Cargar un dato de 16 bits en el registro Y.
LSL: Desplazamiento de un bit hacia la izquierda del contenido de una posición de memoria.
LSLA: Desplazar el acumulador A un bit hacia la izquierda.
LSLB: Desplazar el acumulador B un bit hacia la izquierda.
LSLD: Desplazar el acumulador D un bit hacia la izquierda.
LSR: Desplazar el contenido de una posición de memoria un bit hacia la derecha.
LSRA: Desplazar el acumulador A un bit hacia la derecha.
LSRB: Desplazar el acumulador B un bit hacia la derecha.

LSRD: Desplazar el registro D un bit hacia la derecha.
MUL: Multiplicación sin signo.
NEG: Complementar a dos el contenido de una posición de memoria.
NEGA: Complementar a dos el contenido del acumulador A.
NEGB: Complementar a dos el contenido del acumulador B.
NOP: No operación.
ORAA: Realizar la operación lógica OR entre un dato y el acumulador A.
ORAB: Realizar la operación lógica OR entre un dato y el acumulador B.
PSHA: Meter el acumulador A en la pila.
PSHB: Meter el acumulador B en la pila.
PSHX: Meter el registro X en la pila.
PSHY: Meter el registro Y en la pila.
PULA: Sacar el acumulador A de la pila.
PULB: Sacar el acumulador B de la pila.
PULX: Sacar el registro X de la pila.
PULY: Sacar el registro Y de la pila.
ROL: Rotación a la izquierda del contenido de una posición de memoria.
ROLA: Rotar a la izquierda el acumulador A.
ROLB: Rotar a la izquierda el acumulador B.
ROR: Rotar a la derecha el contenido de una posición de memoria.
RORA: Rotar a la derecha el contenido del acumulador A.
RORB: Rotar a la derecha el contenido del acumulador B.
RTI: Retorno de interrupción.
RTS: Retorno de subrutina.
SBA: Restar un dato al acumulador A.
SBCA: Restar un dato y el bit de acarreo al acumulador A.
SBCB: Restar un dato y el bit de acarreo al acumulador B.
SEC: Poner a uno el bit de acarreo.
SEI: Inhibir las interrupciones.
SEV: Poner a uno el bit de overflow.
STAA: Almacenar el acumulador A en una posición de memoria.
STAB: Almacenar el acumulador B en una posición de memoria.
STD: Almacenar el registro D.
STOP: Para el reloj del sistema.
STS: Almacenar el puntero de pila SP.
STX: Almacenar el registro X.
STY: Almacenar el registro Y.
SUBA: Restar un dato al acumulador A.
SUBB: Restar un dato al acumulador B.
SUBD: Restar un dato al registro D.
SWI: Provocar una interrupción software.
TAB: Transferir el acumulador A al acumulador B.
TAP: Transferir el acumulador A al registro CCR.
TBA: Transferir el acumulador B al acumulador A.
TEST: Instrucción de test. Sólo se puede ejecutar en el modo de test.
TPA: Transferir el registro CCR al acumulador A.
TST: Comprobar si una posición de memoria está a cero.

TSTA: Comprobar si el acumulador A está a cero.
TSTB: Comprobar si el acumulador B está a cero.
TSX: Transferir el puntero de pila al registro X.
TSY: Transferir el puntero de pila al registro Y.
TXS: Transferir el registro X al puntero de pila.
TYS: Transferir el registro Y al puntero de pila.
WAI: Esperar a que se produzca una interrupción.
XGDX: Intercambiar los valores de los registro X y D.
XGDY: Intercambiar los valores de los registros Y y D.

ANEXO V. Instrucciones para bajar un programa a RAM.

1. El programa deberá tener ORG $100.
2. F4 ensamblar.
3. F2 grabar.
4. F7 abrir ventana de comunicaciones.
5. RESET.
6. LOAD T ↵
7. F6 nombre.S19 ↵
8. RESET.
9. R ↵
10. R FFFF 100 ↵
11.
    a. G para correr.
    b. T ↵ para correr a pasos.

COMANDOS DE BUFFALO (EN LA VENTANA DE COMUNICACIONES).

R Cambia un registro.

RA Cambia el contenido del ACC "A".
RP Cambia el contenido del controlador de programa.
P = $100 para RAM.
P = $B600 para EEPROM.

MM $1004 Cambia el contenido de la dirección $1004 (PORTB).

MM $1004 ↵
1004 00 01 Prende el bit menos significativo del puerto B.

MD $100 Despliega la memoria empezando en la dirección 100, se utiliza para ver lo que contiene la memoria.

PARA BAJAR UN PROGRAMA A LA EEPROM.

1) El origen del programa deberá ser $B600.
2) Ensamblar y grabar.
3) Teclear en la ventana de comunicaciones: **MM 102B** ↵ (Aquí se pierde comunicación con el HC11, sin dar RESET, dar **ESC** y dar **2** veces **F10** ir a comunicaciones y en **BAUD** poner **300**.
4) Abrir de nuevo ventana.
5) Bajar como en pruebas.
6) Cambiar el jumper de BUFFALO al lado contrario.
7) RESET. ¡Listo, deberá funcionar la aplicación sin PC!.

ANEXO VI. Glosario.

Amplificador Operacional: También denominado op-amp, es un amplificador diferencial con una ganancia muy alta, con una elevada impedancia de entrada y una impedancia de salida baja. Los usos más típicos del amplificador operacional son proporcionar cambios de amplitud de voltaje (amplitud y polaridad).

Aisladores: Son aquellos materiales a través de los cuales no se puede lograr fácilmente el flujo electrones.

BAUD: Unidad de velocidad de transmisión / recepción.

Capacitor: Dispositivo electrónico que almacena energía y la descarga en un periodo corto de tiempo. Su unidad es el Faradio (F).

Circuito electrónico: Es el camino que sigue una corriente eléctrica que partiendo de su fuente pasa por conductores y componentes y regresa a su punto de partida por lo que se deduce que un circuito electrónico debe ser un camino cerrado para que los electrones que parten de un punto puedan regresar a él completando el circuito.

CNC: Control Numérico Computarizado.

Colector abierto: Transistor PNP (2N2907).

Conductores electrónicos: Son aquellos materiales por los cuales se puede forzar el movimiento de electrones de átomo en átomo cuando se aplica una presión eléctrica ó voltaje.

Conversor A/D: Conversor Analógico/Digital, dispositivo electrónico que recibe una señal analógica y emite una señal digital.

Corriente eléctrica: Flujo ordenado de electrones por un circuito o sección del mismo.

La corriente eléctrica tiene la misma relación con la carga como la tiene una corriente de río con el agua; es la rapidez con que la carga cruza una línea que corta parte del circuito. La regla general de la corriente es que se indica con el símbolo I y se mide en amperios. Un amperio es igual a 1 culombio por segundo. Muy a menudo, tratamos una corriente de miliamperios (mA) y microamperios (μA).

Demultiplexor: Dispositivo electrónico que recibe un canal con varias señales y emite todas las señales por múltiples canales (varias salidas).

Display: Dispositivo electrónico que recibe señales digitales y enciende el led correspondiente.

Display de 7 segmentos: Display que representa dígitos del 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D.

INTEL: Fabricante cuyas siglas significan Electrónica Integrada.

FIFO: First Input First Output. Primer dato en entrar es el primer dato en salir.

Flanco: Cambio de estado en un pulso eléctrico (subir (5V) o bajar (0V)).

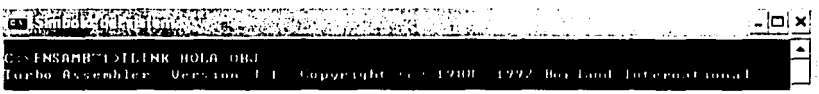
Flip-flop. Interruptor binario.

Flip-flop basculador o biestable. Flip-flop de dos estados o niveles de voltaje, uno bajo, regularmente de 0 a 0.5V y otro alto comúnmente de 4.5 a 5V.

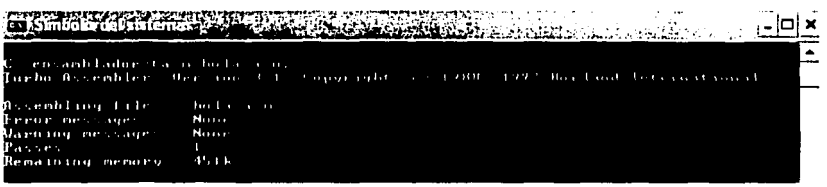
Latch triestado. Selector de tres estados.

Led: Diodo Emisor de Luz. Es un dispositivo en el que intencionalmente se busca el aprovechamiento de luz.

LINK: Software que enlaza el o los programas *.obj y crea el programa *.exe, también se puede utilizar TLINK.EXE que proporciona Borland C.



MASM: Macro Assembled, software que compila el código fuente y crea el programa *.obj, también se puede utilizar TASM que proporciona Borland C.



Multiplexor: Dispositivo electrónico que recibe múltiples entradas y emite todas las señales por un solo canal (una salida).

Nivel: Cambio de estado en un pulso eléctrico (bajo (0V) o alto (5V)).

Norma RS-232c: Estandarización de comunicaciones para PC, consta de 3 etapas: lógica (funcionamiento de cada byte), eléctrica (nivel bajo 0V y nivel alto 5V), mecánica (conector DB9 (puerto serie) y conector DB25 (puerto paralelo)), para garantizar la velocidad de transmisión / recepción entre PC-PC, PC-Microcontrolador, Microcontrolador-PC.

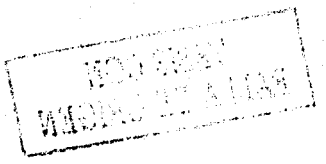
Oscilador de Cristal: Dispositivo electrónico que regula la velocidad de comunicación. Su unidad es el Hertz (Hz).

Resistencia eléctrica: En la oposición o resistencia ofrecida por un material al paso de la corriente eléctrica, su unidad es el Ohm (Ω).

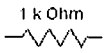
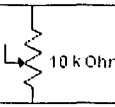
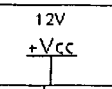
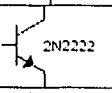
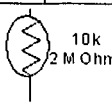
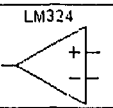
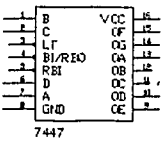
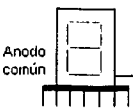
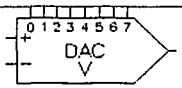
Resistencia de pull-up: Resistencia de protección cuando se tiene conectado de la siguiente forma: Vcc, resistencia pull-up, microcontrolador, así se protege al microcontrolador de las variaciones altas en voltaje.

Voltaje: La atracción de cargas opuestas significa que se requiere energía para apartarlas la que puede recuperarse al juntarse de nuevo. Entre una y otra situación decimos que la energía se conserva como energía potencial. Cuando un sistema tiene energía potencial, tiene el potencial de hacer trabajo.

Trabajo aquí significa una forma más visible de energía, no necesariamente algo que queramos hacer. Su unidad es el Voltio (V).



Símbolos para representar dispositivos electrónicos.

Símbolo	Nombre	Símbolo	Nombre
	Resistencia		GND
	Led		Potenciómetro (Resistencia variable)
	Masa		Push Bottom
	Transistor NPN		Compuerta NOT (LM7408)
	Fotorresistencia		Amplificador Operacional
	Decodificador (LM74347)		Display 7 segmentos Anodo Común
	Convertidor Analógico/Digital (DAC0832)		