Curso introductorio sobre Microcontroladores Familias HC705 y HC908 de Motorola

Parte I

Ing. Daniel Di Lella

D.D.F.A.E For Motorola Products

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Referencias:

- ***Curso sobre Pequeños Microcontroladores – Ing. Juan Cruz***
- ***Understanding Small Microcontrollers – James M. Sibigtroth***
- ***Motorola CPU08 Reference Manual.***
- ***Materiales varios del autor.***

Agradecimientos:

- ***Ing. Juan Cruz, por ser el precursor de este tipo de cursos.***
- ***Sra. Liliana de Di Lella, por organizar y dar forma al presente material.***
- ***A mi familia, por su comprensión y apoyo permanente.***

Descripción del curso.

El curso es de corta duración, intensivo y de carácter formativo con desarrollo teórico practico, proveyendo una sólida base conceptual sobre la arquitectura, dispositivos de memoria y de entrada / salida, lenguaje de programación y sistemas de ayuda para el desarrollo con los modernos microcontroladores (algunas veces utilizaremos la abreviación "MCU" en lugar del termino microcontrolador).

Objetivo.

Durante el curso se capacitara al lector para:

- Evaluar el estado actual de la tecnología.
- Conocer y entender la estructura, funcionamiento e integración de un sistema con un microcontrolador (MCU) moderno.
- Conocer y manejar el repertorio de instrucciones, la memoria y los periféricos de un MCU.
- Conocer y manejar las herramientas de ayuda para el desarrollo con un MCU.
- Evaluar y diseñar aplicaciones básicas con un MCU.

Orientación.

El curso está orientado a ingenieros, técnicos y personal docente con conocimientos o experiencia en técnicas digitales que aún no habiendo recibido una formación específica, deseen extender y actualizar sus conocimientos sobre MCUs o bien desarrollar sencillas aplicaciones con los mismos.

Temario (Parte I).

- Evolución del diseño lógico. El impacto del LSI en el diseño lógico y sus formas de utilización.
- ¿Qué es un microcontrolador?. Revisión de un sistema de computación, subsistemas que lo componen. El microcontrolador. Familias de Microcontroladores y controladores dedicados.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

- Códigos y sistemas de numeración de uso frecuente.
- Dispositivos de memoria y de entrada / salida. Tipos de memoria. Mapa de memoria.
- Arquitectura de una computadora. Registros. Temporización. Estructura de los programas. Operación de una instrucción. Modos de direccionamiento. Reset. Interrupciones.
- Repertorio de instrucciones para MCUs flia. HC705
- Edición, ensamblado, simulación y Emulación de programas.
- El Lazo de Paso (Paced Loop). Ejemplos de uso.

Temario (Parte II).

- Curso MCUs flia. HC908 (Parte II).
- Diferencias entre flias. HC705 y HC908 (Parte II).
- Repertorio de instrucciones para MCUs flia. HC908 (Parte II).
- Distintos periféricos, Ejemplo de Aplicaciones (Parte II).
- Resumen de características de los distintos derivativos HC908 (Parte II).
- Manejo de las herramientas de ayuda para el desarrollo con un MCU OTP / FLASH (Parte II).
- Aplicaciones varias (Parte II).
- Familia de MCUs de 8 y 16 pines HC908Q.

Breve comentario del Autor.

El propósito de este curso es introducir al lector, al fascinante mundo de los microcontroladores. En el mundo de hoy, ya nadie duda de la utilidad y economía de uso de estos pequeños “cerebros” encapsulados. Presentes en casi todas las aplicaciones de la vida diaria, desde el control de pequeños electrodomésticos, máquinas herramientas, hasta como sub sistemas de complejos dispositivos electrónicos, hoy no podríamos imaginar el mundo sin ellos.
En este texto, el lector, encontrará la ayuda necesaria para comprender las tareas internas de estas pequeñas computadoras de propósito general y así ilustrarlo sobre cómo concebir microcontroladores en aplicaciones útiles. En este curso se hace referencia especialmente a los MCUs de las familias HC705 y HC908 FLASH de Motorola. Sin embargo, los conceptos aquí vertidos son aplicables a la totalidad de los microcontroladores.

Se sugiere reforzar los conocimientos impartidos a lo largo del curso, consultando los respectivos manuales de referencia (Reference Manual) de las familias HC705 y HC908, así como los manuales operativos de las herramientas de desarrollo para ambas familias.

Este texto no asume ningún conocimiento previo sobre microprocesadores o técnicas de programación. Los estudiantes secundarios pueden utilizarlo en clases guiadas por un profesor o instructor. Los ingenieros experimentados pueden además utilizarlo para aprender sobre microcontroladores.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Los siguientes párrafos presentan una breve descripción de cada capítulo y apéndices de este curso.

1- ¿Qué es un microcontrolador?

Este capítulo introduce los principales elementos constitutivos de cualquier sistema de computadora. Aquí se presentan las diferentes partes de los sistemas de computadora y los rasgos que distinguen a los microcontroladores de otros tipos de sistemas de computadora.

2- Sistemas de Numeración y Código.

Este capítulo explora los sistemas de numeración y los códigos especiales usados por las computadoras. Las computadoras cuentan en binario (base 2) en lugar de hacerlo en decimal (base 10). El American Standard Code for Information Interchange (ASCII) es otro código que permite a las computadoras trabajar con información alfabética. Por último las computadoras usan instrucciones especialmente codificadas con las que ellas ejecutan sus programas.

3- Memoria y Dispositivos de I/O.

La memoria es un componente básico de cualquier computadora. Este capítulo presenta varios tipos de memoria. Se presenta además el concepto de dispositivos de Entrada / Salida (I/O) paralelo como un tipo de memoria. Así como se explica en detalle el concepto de mapa de memoria de una computadora, Ud. podrá tener un primer panorama sobre las tareas internas de una computadora.

4- Arquitectura de una Computadora.

Este capítulo describe la estructura interna y las operaciones de la unidad central de proceso (CPU). A partir de los conceptos de los primeros tres capítulos se llega a presentar cómo opera una computadora. Esta vista detallada de las operaciones de una computadora provocará la subsecuente presentación de programas de fácil comprensión.

5- El repertorio de Instrucciones del MC68HC05 y el MC68HC08.

Este capítulo comienza con una revisión de la CPU del MC68HC05 y del MC68HC08 tal como los ve un programador. Se explican los modos de direccionamiento a fin de presentar las diferentes formas en que en un programa se puede especificar la localización de un operando. Se presenta al repertorio de instrucciones de dos maneras. Primero, las instrucciones agrupadas por su modo de direccionamiento. Segundo, las instrucciones ordenas por el tipo de función.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

6- Programación.

Las computadoras no son inteligentes. Ellas solo hacen lo que las instrucciones de programa les dicen que deben hacer, las computadoras sólo saben cómo realizar un relativamente pequeño grupo de simples instrucciones. Él sin número de formas en que se pueden combinar las instrucciones es lo que permite a las computadoras realizar gran diversidad de tareas. Este capítulo presenta cómo preparar un grupo de instrucciones para que la computadora las ejecute. Además se presenta a los ensambladores y simuladores.

7- "Paced Loop" o Lazo cíclico o de paso.

Esta estructura de programación puede usarse como una base para muchas aplicaciones de microcontroladores. Las sub tareas que son específicas de una aplicación pueden escribirse en forma independiente. Estas sub tareas pueden agregarse al esqueleto del "lazo de paso" o lazo cíclico.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 1.

¿Qué es un Microcontrolador?

Este capítulo contiene un conjunto de principios esenciales para hacer un detallado repaso de las tareas que se desarrollan dentro de un pequeño microcontrolador. Veremos que el microcontrolador es una de las formas más elementales que puede tener un sistema de computadora. Aún siendo mucho más pequeños que sus primos, las computadoras personales y las computadoras mainframes, los microcontroladores están constituidos por los mismos elementos básicos. En pocas palabras, podemos decir que las computadoras producen un patrón específico de salidas en base al estado actual de sus entradas, y siguiendo estrictamente las instrucciones contenidas en un programa.

Al igual que la mayoría de las computadoras, los microcontroladores son simples ejecutores de instrucciones de propósito general. La verdadera estrella de un sistema de computadora es el programa de instrucciones que son provistas por un programador humano. Este programa instruye a la computadora a realizar largas secuencias de muy simples acciones para efectuar tareas útiles tales como las que se propuso el programador.

Una vista completa de un Sistema de Computadora.

La figura 1-1 es una visión de alto nivel de un **sistema de computadora.** Simplemente cambiando el tipo de dispositivos de entrada y salida ésta puede ser la vista de una **computadora personal (PC),** una **mainframe,** o un simple **microcontrolador (MCU).** Los dispositivos de entrada y salida **(I/O)** presentados en la figura aparecen como típicos dispositivos de I/O encontrados en un sistema de computadora con microcontrolador.

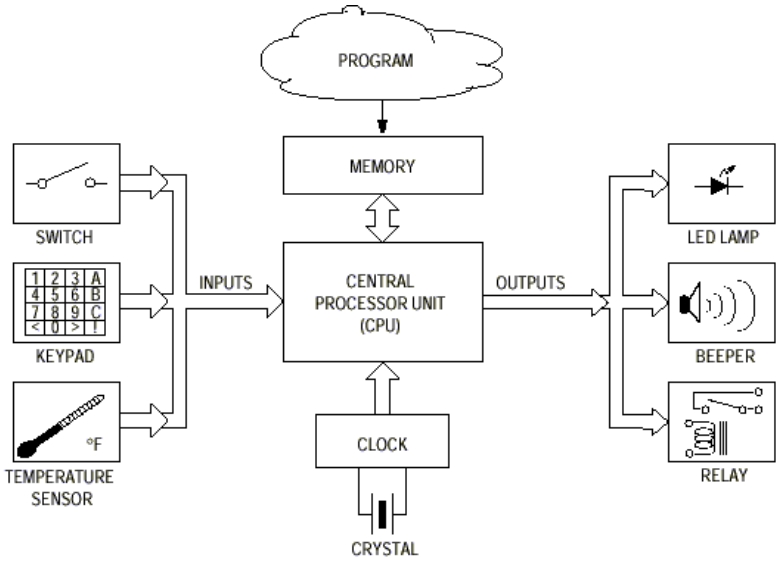


Figura 1-1. – Vista general de un sistema de computación.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Entradas.

Los dispositivos de entrada suministran al sistema de computadora la información proveniente del mundo exterior. En un sistema de PC, el dispositivo de entrada más común es el teclado del tipo máquina de escribir. Las mainframes usan teclados y lectores de tarjetas perforadas como dispositivos de entrada. Los sistemas con microcontrolador normalmente usan dispositivos de entrada mucho más simples tales como interruptores simples o conjuntos de algunas pocas teclas, encontraremos sin embargo, dispositivos de entrada más exóticos en sistemas basados en microcontroladores. Un ejemplo de dispositivo de entrada exótico es el sensor de oxígeno que en un automóvil tiene por objetivo medir la eficiencia de la combustión tomando muestras de los gases expelidos.

La mayoría de las entradas de los microcontroladores pueden únicamente procesar **señales digitales,** de los mismos niveles de tensión que la fuente de alimentación de la lógica principal. El potencial cero o nivel de tierra se denomina **Vss** y el potencial positivo o nivel de alimentación **(Vdd)** es típicamente 5 Volts de C.C. Un nivel de aproximadamente cero Volts es indicativo de un **cero lógico** y una tensión aproximadamente igual que la de la fuente de alimentación positiva es indicativo de un **uno lógico.**

Por supuesto que el mundo real está poblado de señales analógicas, o señales que son de otro niveles de tensión. Algunos dispositivos de entrada trasladan los niveles de tensión de una señal a los niveles Vdd y Vss necesarios para el microcontrolador. Otros dispositivos de entrada convierten **señales analógicas** en **señales digitales** (a valores binarios compuestos por unos y ceros) que la computadora puede reconocer y manipular. Hay microcontroladores que también incluyen tales circuitos conversores de señal analógica a digital en su mismo circuito integrado (es el caso de la familia HC908 FLASH).

Podemos recurrir a **transductores** para trasladar señales del mundo real a señales del nivel lógico que un microcontrolador puede reconocer y manipular. Veremos aplicaciones que incluyen transductores de temperatura, detectores de nivel de iluminación, sensores de presión, etc.. Es posible afirmar que con el transductor adecuado, casi cualquier propiedad física puede ser utilizada como una entrada a un sistema de computadora.

Salidas.

Los dispositivos de salida sirven para que el sistema de computadora suministre información al mundo exterior o bien realice acciones sobre éste. En un sistema de PC, el dispositivo de salida más común es la pantalla de un **tubo de rayos catódicos (TRC).** Los sistemas de computadora con microcontrolador normalmente utilizan dispositivos de salida mucho más simples tales como indicadores lumínicos o sonoros.

Existen circuitos de translación (a veces incluidos en el mismo circuito integrado del microcontrolador) que pueden convertir señales digitales a niveles de tensión analógica.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

De ser necesario, podemos recurrir a un circuito auxiliar para trasladar los niveles naturales de la MCU Vdd y Vss a otros niveles de tensión.

El termino “controlador”de la palabra microcontrolador proviene del hecho que estas pequeñas computadoras usualmente se utilizan para controlar alguna cosa, mientras que, con una PC habitualmente se procesa información. En el caso de una PC, la salida en su mayor parte es información (presentada en la pantalla de un TRC o en su defecto impresa en papel). En un sistema de computadora con microcontrolador la salida en su mayor parte la integran señales de nivel lógico digital que se utilizarán para manejar presentadores de diodos de emisión lumínica **(LED)** o bien dispositivos eléctricos tales como reveladores o motores.

Unidad Central de Proceso (CPU)

La **CPU** es el núcleo de todo sistema de computadora. La tarea de la CPU es ejecutar obedientemente las instrucciones del programa que le fuera suministrado por el programador. Un **programa de computadora** da instrucciones a la CPU para **leer** información de las entradas, leer información de la memoria de trabajo y escribir información en ella, y para **escribir** información sobre ellas. Algunas instrucciones del programa involucran sencillas decisiones que causan que se continúe ejecutando la próxima instrucción o bien que se la saltee pasando a un nuevo punto dentro del programa. En un capítulo posterior veremos más cuidadosamente el conjunto de las instrucciones disponibles para un microcontrolador en particular.

En computadoras mainframes y en PC’s hay actualmente niveles de programas comenzando con los programas internos que controlan las operaciones más básicas de la computadora. Otros niveles incluyen programas de usuario que habiendo sido cargados en la memoria del sistema de computadora estarán disponibles para ser ejecutados desde allí. Esta estructura es muy compleja y no resulta ser un buen ejemplo para presentar a un principiante como trabaja una computadora.

En un microcontrolador hay usualmente sólo un programa, el que atiende una aplicación específica de control. La CPU MC68HC05 (CPU05) y la MC68HC08 (CPU08) reconocen alrededor de **60 y 89 instrucciones** diferentes respectivamente, no obstante, ellas son representativas del conjunto de instrucciones de un sistema de computadora. Este modesto sistema de computadora, resulta ser un buen modelo para aprender los fundamentos de la operación de una computadora puesto que es posible conocer con exactitud qué sucede en cada pequeña etapa en que la CPU ejecuta un programa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Reloj

Con muy pocas excepciones, las computadoras utilizan un pequeño reloj o **circuito oscilador** para forzar a la CPU a moverse en secuencia desde una etapa a la próxima. En el capítulo sobre arquitectura de una computadora veremos que siempre, una instrucción por simple que sea, se puede dividir en una serie de etapas más elementales. Cada una de estas pequeñas etapas en la operación de una computadora, toma un ciclo de reloj de la CPU.

Memoria

Hay varios tipos de memoria que se utilizan para diversos propósitos en los sistemas de computadora. Los principales tipos que se encuentran en sistemas con microcontroladores, son la memoria de la lectura solamente **(ROM)** y la memoria de lectura / escritura de acceso aleatorio **(RAM).** La ROM se utiliza principalmente para el almacenamiento de programas y datos en forma permanente, que permanecerán inalterables aún cuando se apague la fuente de alimentación del sistema con microcontrolador. Mientras que la RAM en cambio sirve para el almacenamiento temporário de datos y resultados intermedios generados durante la operación. Hay microcontroladores que incluyen otro tipo de memoria tales como ROM programable eléctricamente y borrable eléctricamente **(EEPROM)**. Conoceremos más respecto a estos tipos de memoria en un capítulo posterior.

La menor unidad de almacenamiento de memoria en una computadora es el **bit** que puede retener un valor de cero o uno lógico. Estos bits agrupados en conjuntos de a 8 forman lo que se denomina **byte.** Las computadoras más grandes poseen grupos de bits en conjuntos de a 16 a 32 formando una unidad denominada **word.** El tamaño de un word puede variar según la computadora, sin embargo un byte siempre estará formado por **8 bits.**

Las PC's trabajan con programas muy extensos y gran cantidad de datos por lo que recurren a tipos especiales de memoria denominados dispositivos de **almacenamiento masivo.** Son dispositivos de almacenamiento masivo los discos flexibles, rígidos y compactos. Hay en día es usual encontrar en computadoras personales millones de bytes de RAM. Igualmente esto no permite mantener almacenados los extensos programas típicos de las PC's, por lo que además incluyen discos rígidos con una capacidad de almacenamiento de decenas o cientos de millones de bytes. Los discos compactos, muy similares a los usados para grabaciones de audio, por su parte poseen una capacidad de alrededor de 600 millones de bytes de ROM. Los sistemas con pequeños microcontroladores típicamente poseen un total que va de 1 mil a 64 miles de bytes de memoria.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Programa

La figura 1-2 muestra al programa como una nebulosa ya que éste es pergeñado en la imaginación de un programador, analista o ingeniero. Esto es comparable al diseño de un circuito por parte de un ingeniero electrónico o a la formulación de un ensamble por un ingeniero mecánico. Los componentes de un programa son las instrucciones contenidas en el repertorio de instrucciones de la CPU. Del mismo modo que un diseñador de circuitos puede implementar un circuito sumador partiendo de elementos simples como AND, OR y NOT, un programador puede escribir un programa que adiciona números recurriendo a simples instrucciones.

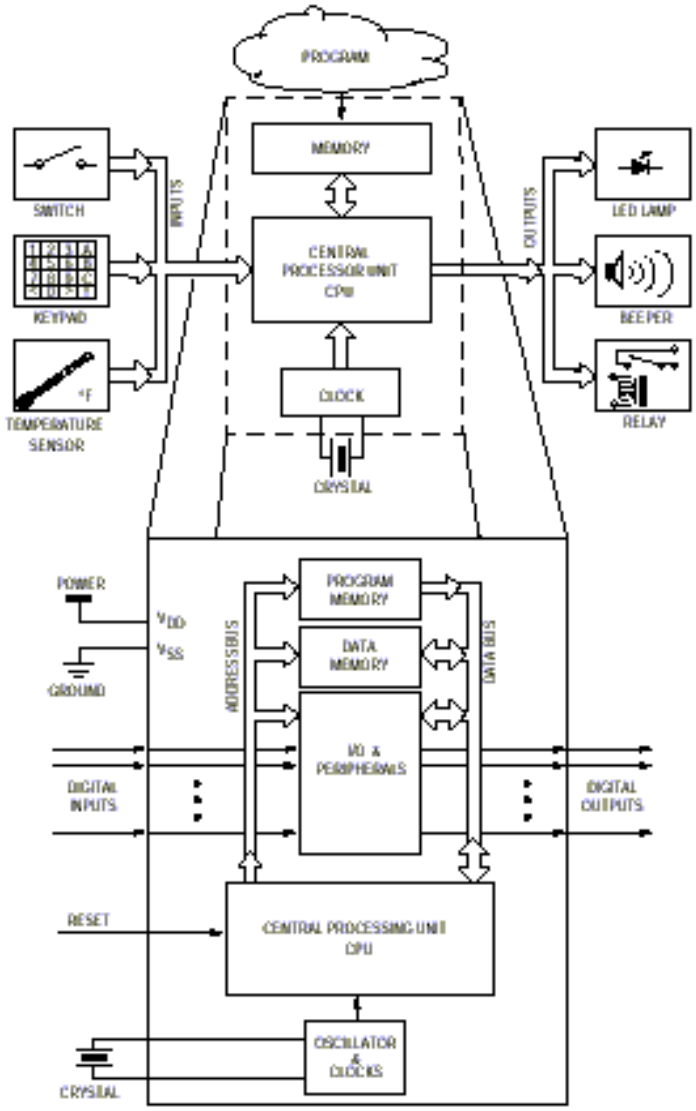


Fig. 1-2 – Esquema básico de un Controlador y su programa

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Microcontrolador

Ahora que hemos discutido sobre las diferentes partes de un sistema de computadora, estamos listos para decir qué es un microcontrolador. La mitad superior de la figura 1-2 nos muestra un sistema de computadora genérico con una parte encerrada por línea de puntos. La porción enmarcada es un microcontrolador y la mitad inferior de la figura es un diagrama de bloques que presenta su estructura interna con un mayor detalle. El cristal no está en el microcontrolador si bien es una parte esencial del circuito oscilador. En algunos casos, se pueden usar componentes más económicos reemplazando al cristal, como resonadores cerámicos o bien circuitos con resistor y capacitor (R-C).

Un **microcontrolador** puede definirse como un sistema de computadora completo incluyendo un CPU, memoria, reloj oscilador y I/O en el mismo circuito integrado. Cuando carece de alguno de estos elementos, ya sea I/O o memoria, el circuito integrado lleva el nombre de **microprocesador.** En una PC la CPU es un microprocesador. En una computadora mainframe la CPU se conforma por varios circuitos integrados.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Revisión del Capítulo 1

Un microcontrolador es un sistema de computadora completo, incluyendo la CPU, la memoria, un reloj oscilador y I/O en un único circuito integrado.

Las partes de cualquier computadora

- Una Unidad Central de Proceso **(CPU)**
- Un **reloj** para secuenciar a la CPU
- **Memoria** para el almacenamiento de instrucciones y datos.
- **Entradas** para ingresar información al sistema de computadora
- **Salidas** para sacar información desde el sistema de computadora
- Un **programa** que le da a la computadora una utilidad

Tipos de Computadoras

Aunque todas las computadoras poseen los mismos principios y elementos básicos, las hay de diferentes tipos, orientadas a satisfacer diversos propósitos. Las mainframes son sistemas de computadora muy grandes que se utilizan para trabajos de procesamiento de grandes volúmenes de información. Las PC's son versiones reducidas de las mainframes, aplicables a tareas de menor envergadura aún, tales como por ejemplo procesadores de texto o gráficos. Los microcontroladores son computadoras en un solo circuito integrado, utilizados para controlar una pequeña aplicación. Los pequeños microcontroladores se usan por ejemplo para convertir el movimiento de un mouse de computadora en una salida serie de datos para ingresarlo a una PC. Muy frecuentemente los microcontroladores se hallan **embebidos** dentro de un producto y quien lo usa no necesariamente sabe que el mismo en su interior alberga una computadora.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 2.

Sistemas de Numeración y Código

Las computadoras trabajan adecuadamente con información en un formato diferente al que la gente está habituada a manejar a diario. Nosotros trabajamos típicamente en el sistema de numeración de base 10 (decimal, con niveles de 0 a 9), probablemente por tener diez dedos en las manos el hombre primitivo adopto este sistema de numeración como el más conveniente. Las computadoras digitales binarias trabajan con el sistema de numeración en base 2 (binario, 2 niveles), puesto que éste permite representar cualquier información mediante un conjunto de dígitos, que solo serán "ceros" (0) ó "Unos" (1).
En su momento, un **uno** o un **cero** puede representar la presencia o ausencia de un nivel lógico de tensión sobre una línea de señal, o bien el estado de **encendido** o **apagado** de una simple llave.
Este capítulo aborda los sistemas de numeración comúnmente utilizados por las computadoras, es decir, **binario**, **hexadecimal**, **octal** y **binario codificado en decimal (BCD).**

Las computadoras además usan códigos especiales para representar información del alfabeto o sus instrucciones. La comprensión de estos códigos nos ayudará a entender cómo una computadora se las ingenia para entender cadenas de dígitos que sólo pueden ser unos o ceros.

Números Binarios y Hexadecimales.

Para un número decimal (base 10) el peso (valor) de un dígito es diez veces mayor que el que se encuentra a su derecha. El dígito del extremo derecho de un número decimal entero, es el de las unidades, el que está a su izquierda es el de las decenas, el que le sigue es el de las centenas, y así sucesivamente. Para un número binario (base 2), el peso de un dígito es dos veces mayor que el que se encuentra a su derecha. El dígito del extremo derecho de un número binario entero, es el de la unidad, el que está a su izquierda es el de los duplos, el que le sigue es el de los cuádruplos, el que le sigue es el de los óctuplos, y así sucesivamente.

Para algunas computadoras es habitual trabajar con números de 8, 16 o bien 32 dígitos binarios, para nosotros resulta demasiado engorroso trabajar con tal cantidad de dígitos. El sistema de numeración de **base 16 (hexadecimal)** resulta ser una práctica solución de compromiso. Con un dígito hexadecimal podemos representar exactamente igual, a cuatro dígitos binarios, de este modo un número binario de 8 dígitos se puede expresar mediante dos dígitos hexadecimales.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

La sencilla correspondencia que existe entre las representaciones de un dígito hexadecimal y la de cuatro dígitos binarios, nos permite realizar mentalmente la conversión entre ambos. Para un número **hexadecimal (base 16),** el peso de un dígito es dieciséis veces mayor que el que se encuentra a su derecha. El dígito del extremo derecho de un número hexadecimal entero, es de las unidades, el que está a su izquierda es el de los décimo séxtuplos, y así sucesivamente.

La tabla 2-1 nos muestra la relación existente, en la representación de valores, en decimal, binario y hexadecimal. Estos tres diferentes sistemas de numeración resultan ser tres modos diferentes de representar físicamente las mismas cantidades. Se utilizan las letras de la A a la F para representar los valores hexadecimales correspondientes que van del 10 al 15, ya que cada dígito hexadecimal puede representar 16 cantidades distintas. Si consideramos que nuestro sistema de representación sólo incluye diez símbolos (del 0 al 9); por lo tanto, debemos recurrir a algún otro símbolo de un solo dígito, para de esta manera, representar los valores hexadecimales que van del 10 al 15.

Para evitar confusiones acerca de si un número es hexadecimal o decimal, pondremos un símbolo $ antecediendo a cada cantidad hexadecimal. Por ejemplo, 64 es el decimal "sesenta y cuatro"; consideremos entonces que $64 es hexadecimal "seis-cuatro", que es equivalente al decimal 100. Algunos fabricantes de computadoras escriben una letra H como terminación de un valor hexadecimal (para el ejemplo anterior sería 64H).

Base 10 Decimal	Base 2 Binary	Base 16 Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
100	0110 0100	64
255	1111 1111	FF
1024	0100 0000 0000	400
65,535	1111 1111 1111 1111	FFFF

Tabla 2-1 – Equivalentes entre Decimal, Binario y Hexadecimal.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

El hexadecimal es el modo adecuado tanto para expresar como para discutir acerca de información numérica procesada por una computadora, ya que a la gente le resulta fácil realizar la conversión entre un dígito hexadecimal y sus 4 bits equivalentes. La notación hexadecimal es mucho más compacta que la binaria mientras seguimos manteniendo las connotaciones binarias.

Código ASCII

Las computadoras deben manejar otros tipos de información además de los números. Tanto los textos (caracteres alfanuméricos) como las instrucciones deben codificarse de tal modo que la computadora interprete esta información. El código más común para la información tipo texto es el American Standard Code for Information Interchange **(ASCII).** El código ASCII es una correlación ampliamente aceptada entre caracteres alfanuméricos y valores binarios específicos. En este código, el número $41 corresponde a una letra A mayúscula, el $20 al carácter espacio, etc. El código ASCII traduce un carácter a un código binario de 7 bits, aunque en la práctica la mayoría de las veces la información es transportada en caracteres de 8 bits con el bit más significativo en cero. Este estándar permite hace posible las comunicaciones entre equipos hechos por diversos fabricantes, puesto que todas las máquinas utilizan el mismo código.

La tabla 2-2 nos muestra la relación existente entre los caracteres ASCII y los valores hexadecimales.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$20	SP space	$40	@	$60	` grave
$01	SOH	$21	!	$41	A	$61	a
$02	STX	$22	"	$42	B	$62	b
$03	ETX	$23	#	$43	C	$63	c
$04	EOT	$24	$	$44	D	$64	d
$05	ENQ	$25	%	$45	E	$65	e
$06	ACK	$26	&	$46	F	$66	f
$07	BEL beep	$27	' apost.	$47	G	$67	g
$08	BS back sp	$28	(	$48	H	$68	h
$09	HT tab	$29	)	$49	I	$69	i
$0A	LF linefeed	$2A	*	$4A	J	$6A	j
$0B	VT	$2B	+	$4B	K	$6B	k
$0C	FF	$2C	, comma	$4C	L	$6C	l
$0D	CR return	$2D	- dash	$4D	M	$6D	m
$0E	SO	$2E	. period	$4E	N	$6E	n
$0F	SI	$2F	/	$4F	O	$6F	o
$10	DLE	$30	0	$50	P	$70	p
$11	DC1	$31	1	$51	Q	$71	q
$12	DC2	$32	2	$52	R	$72	r
$13	DC3	$33	3	$53	S	$73	s
$14	DC4	$34	4	$54	T	$74	t
$15	NAK	$35	5	$55	U	$75	u
$16	SYN	$36	6	$56	V	$76	v
$17	ETB	$37	7	$57	W	$77	w
$18	CAN	$38	8	$58	X	$78	x
$19	EM	$39	9	$59	Y	$79	y
$1A	SUB	$3A	:	$5A	Z	$7A	z
$1B	ESCAPE	$3B	;	$5B	[	$7B	{
$1C	FS	$3C	<	$5C	\	$7C	|
$1D	GS	$3D	=	$5D	]	$7D	}
$1E	RS	$3E	>	$5E	^	$7E	~
$1F	US	$3F	?	$5F	_ under	$7F	DEL delete

Tabla 2-2 – Conversión de ASCII a Hexadecimal.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Códigos de Operación

Las computadoras utilizan otros códigos para darle instrucciones a la CPU. Este código se denomina código de operación **(opcode).** Cada código de operación instruye a la CPU en la ejecución de una muy específica secuencia de etapas que debe seguirse para cumplir con la operación propuesta. Las computadoras de distintos fabricantes usan diferentes repertorios de códigos de operación, previstos en la lógica cableada de la CPU. El **repertorio (set) de instrucciones** para una CPU es el conjunto de instrucciones que ésta es capaz de realizar. Los códigos de operación son una representación del set de instrucciones y los mnemónicos son otra. Aún cuando difieren de una computadora a otra, todas las computadoras digitales binarias realizan el mismo tipo de tareas básicas de modo similar. La CPU en la MCU MC68HC05 puede entender 62 instrucciones básicas. Algunas de éstas presentan mínimas variaciones, cada una de las cuales requiere su propio código de operación. El set de instrucciones del MC68HC05 incluye 210 opcodes distintos.
Discutiremos en un capítulo posterior cómo la CPU ejecuta realmente una instrucción. Antes necesitamos entender unos pocos conceptos básicos más.

Mnemónicos y Ensambladores

Un opcode tal como $4C es interpretado por la CPU, pero no es fácilmente manejable por una persona. Para resolver este problema, se usa un sistema de **mnemónicos** de instrucción equivalentes. El opcode $4C corresponde al mnemónico INCA, que se lee incrementar el acumulador". Aunque contamos con la información impresa que muestra la relación entre el mnemónico de cada instrucción y el opcode que la representa, es rara vez utilizada por el programador puesto que el proceso de transducción lo realiza automáticamente un programa de computadora específico denominado **ensamblador.** Este programa es el que convierte los mnemónicos de las instrucciones de un programa en una lista de **códigos de máquina** (códigos de operación e información adicional), para que puedan ser utilizados por la CPU.

Un ingeniero desarrolla un grupo de instrucciones para una computadora en la forma de mnemónicos y luego utiliza un ensamblador para trasladar estas instrucciones a los opcodes que la CPU pueda entender. En otros capítulos discutiremos respecto a las instrucciones, para una computadora en forma de mnemónicos, mientras que la computadora entiende solamente códigos de operación; por lo tanto, se requerirá una etapa de traducción para cambiar los mnemónicos por opcodes, siendo ésta la función que cumple un ensamblador.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Octal

Antes de comenzar la discusión sobre los sistemas de numeración y códigos, vimos dos códigos más respecto de los cuales podemos oír hablar. La notación **octal (base 8)** fue usada para trabajar con algunas computadoras primitivas, pero rara vez es utilizada en la actualidad. Esta usa los números que van del 0 al 7 para representar un conjunto de tres dígitos binarios de un modo análogo que en la hexadecimal se recurre a un conjunto de cuatro dígitos binarios. El sistema octal tiene la ventaja de no necesitar símbolos no numéricos (como los símbolos hexadecimales ya vistos, que van de la "A" a la "F").

Dos desventajas acarrea cambiar la notación haxadecimal usada hoy en día por la octal. La primera, de todas las computadoras, la mayoría utilizan "words" (palabras) de 4, 8, 16 ó 32 bits, estas "palabras" no resultan fácilmente fraccionables en grupos de tres bits (algunas computadoras muy antiguas usaban palabras de 12 bits siendo divisibles en cuatro grupos de tres bits). La segunda, es el hecho de carecer de la compactabilidad del hexadecimal. Por ejemplo, el valor ASCII de la letra "A" mayúscula es **10000001(2)**, **41(16)** en hexadecimal y **101(8)** en octal.
Cuando una persona menciona el valor ASCII para la "A", es más fácil decir **"cuarto - uno"** que **"uno - cero - uno".**

La tabla 2-3 presenta las correlaciones entre octal y binario. La columna "binario directo" muestra dígito por dígito el pasaje de los dígitos octales a grupos de 3 bits.
El bit del extremo izquierdo (noveno) se ha resaltado. Este cero resaltado no es significativo para formar un resultado de ocho bits. La columna "8 bits binarios" contiene la misma información que la anterior salvo que están ordenados en grupos de 4 bits. Cada grupo de 4 bits se convierte directamente en un dígito hexadecinal.

Octal	Direct Binary	8-Bit Binary	Hexadecimal
000	000 000 000	0000 0000	$00
001	000 000 001	0000 0001	$01
002	000 000 010	0000 0010	$02
003	000 000 011	0000 0011	$03
004	000 000 100	0000 0100	$04
005	000 000 101	0000 0101	$05
006	000 000 110	0000 0110	$06
007	000 000 111	0000 0111	$07
010	000 001 000	0000 1000	$08
011	000 001 001	0000 1001	$09
012	000 001 010	0000 1010	$0A
013	000 001 011	0000 1011	$0B
014	000 001 100	0000 1100	$0C
015	000 001 101	0000 1101	$0D
016	000 001 110	0000 1110	$0E
017	000 001 111	0000 1111	$0F
101	001 000 001	0100 0001	$41
125	001 010 101	0101 0101	$55
252	010 101 010	1010 1010	$AA
377	011 111 111	1111 1111	$FF

Tabla 2-3 – Equivalentes Octal, Binario y Hexadecimal.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Cuando mentalmente convertimos valores octales a valores binarios de un byte, el valor octal resultante se representará mediante 3 dígitos octales. Cada dígito octal representa a su vez 3 bits con lo que resulta un bit extra (3 dígitos x 3 bits = 9 bits). Típicamente la gente opera de izquierda a derecha, lo que hace fácil olvidarse del tratamiento que debe recibir el bit extra del extremo izquierdo (noveno) de un octal. Cuando convertimos de hexadecimal a binario, nos resulta fácil puesto que cada dígito hexadecimal se transforma exactamente en cuatro bits. Dos dígitos hexadecimales coinciden exactamente con los ocho bits de un byte.

Binario Codificado en Decimal.

El sistema **B**inario **C**odificado en **D**ecimal (**BCD**) es una notación híbrida usada para expresar valores decimales en forma binaria. Un BCD utiliza cuatro bits para representar cada dígito decimal.

De esta manera cuatro dígitos binarios pueden expresar 16 diferentes cantidades físicas, habiendo seis combinaciones consideradas no válidas (específicamente, los valores hexadecimales de la “A” a la “F”). Los valores BCD se representan con el signo “$”, pues ellos son números hexadecimales que representan cantidades decimales.

Cuando la computadora hace una operación de suma BCD, ella realiza una suma binaria y luego realiza un ajuste que genera un resultado BCD. Como un simple ejemplo, consideremos la siguiente suma BCD.

9(10) + 1(10) = 10 (10) donde (10), significa Base 10.

La computadora suma.....

0000 1001(2) + 0000 0001(2) = 0000 1010(2)

donde (2), significa Base 2.

Pero 1010(2) es equivalente a “A(16)” que es un código BCD no válido. Cuando la computadora termina el cálculo, realiza un chequeo para ver si el resultado es un código BCD válido. Si hubo un “acarreo” (un desborde) de un dígito BCD a otro o si Hubiese algún código no válido, se desencadena una secuencia de etapas para corregir el resultado y llevarlo al formato BCD apropiado. El número 0000 1010(2) es corregido y se transforma en 0001 0000(2) (BCD 10) en este ejemplo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Decimal	BCD	Binary	Hexadecimal (reference)
0	$0	0000	$0
1	$1	0001	$1
2	$2	0010	$2
3	$3	0011	$3
4	$4	0100	$4
5	$5	0101	$5
6	$6	0110	$6
7	$7	0111	$7
8	$8	1000	$8
9	$9	1001	$9
Invalid BCD Combinations		1010	$A
		1011	$B
		1100	$C
		1101	$D
		1110	$E
		1111	$F
10	$10	0001 0000	$10
99	$99	1001 1001	$99

Tabla 2-4 – Equivalentes Decimal, BCD, y Binario.

En la mayoría de los casos es ineficiente utilizar la notación BCD para los cálculos de la computadora. Es mejor convertir la información de decimal a binario en el momento de su ingreso, realizar todos los cálculos en binario y convertirlos nuevamente a BCD o decimal sólo si es necesario presentarlos en un exhibidor.
No todos los microcontroladores son capaces de realizar cálculos en BCD ya que necesitamos tener la indicación del acarreo dígito a dígito que no está presente en todas las computadoras (tener en cuenta que en los MCUs de Motorola tienen este indicador de semi-acarreo). Forzar a una computadora a comportarse como nosotros resulta menos eficiente que permitirle trabajar en su sistema de numeración natural.

Revisión del capítulo 2

Las computadoras aceptan dos niveles lógicos (0 y 1) con los que trabaja en el sistema de numeración binario. Las personas por tener diez dedos en las manos trabajan con el sistema de numeración decimal.

Los números hexadecimales utilizan dieciséis símbolos del 0 al 9 y de la A a la F. Cada dígito hexadecimal puede representarse mediante cuatro dígitos binarios. La tabla 2-1 nos muestra la equivalencia entre decimal, binario y hexadecimal de varios valores. Para poder distinguir un valor hexadecimal de otro decimal, colocaremos un símbolo $ antecediendo al hexadecimal.

El ASCII es un código ampliamente aceptado y nos permite representar tanto información alfanumérica como valores binarios.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Cada instrucción o variante de una instrucción posee un único código de operación (valor binario) que la CPU reconoce como un pedido para realizar una instrucción específica. Las CPUs de diferentes fabricantes poseen repertorios de códigos de operación distintos.

Los programadores explicitan instrucciones mediante un mnemónico tal como **"INCA"**. Un programa de computadora llamado ensamblador (ASSEMBLER COMPILER), traduce las instrucciones de mnemónicos a códigos de operación que la CPU puede reconocer.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 3.

Memoria y Dispositivos de I/O.

Antes de poder discutir la operación de la CPU en detalle, es preciso conocer algunos conceptos respecto de la memoria de una computadora. En muchos cursos de programación para principiantes, se presenta a la memoria como si fuese similar a una matriz de casillas de correo o casillas de un guardarropa, donde se puede guardar mensajes u otra información. La analogía con casillas de correo o las casillas de un guardarropa, si bien son una buena aproximación a lo que sucede en una memoria, necesitan de alguna explicación adicional para describir exactamente el funcionamiento de la misma.

Analogía de las Casillas de Correo.

La misión final de cualquier arreglo circuital de memoria, es que tenga la capacidad de retener información. Obviamente no se satisface sólo con almacenar la información si no hay manera de retirarla. El arreglo de casillas de correo de un gran edificio de departamentos u oficina postal puede utilizarse como una memoria. Se podrá poner información en la casilla de correo de un número dado de departamento o número de usuario de oficina postal. Cuando se desee recuperar información se podrá ir a la casilla de correo de la dirección especifica (departamento o número de casilla) y retirar la información.

Lo siguiente es trasladar esta analogía para explicar más ampliamente como la computadora ve a la memoria. Confinaremos esta discusión para una computadora de 8 bits a los efectos de poder ser muy específicos.

En una CPU de 8 bits, cada casilla de correo puede pensarse como conteniendo un conjunto de ocho interruptores SI / NO. Al igual que en un casillero de correo, no se podrá colocar más información escribiendo más apretado, si ya se ha colmado su capacidad (ocho interruptores, cada uno de ellos abierto o cerrado). El contenido de una posición de memoria puede ser desconocido o indefinido en un momento dado, igual que el estado de los interruptores en la casilla de correo, pueden estar en un estado desconocido hasta que se los acciona por primera vez. Los ocho interruptores podrían estar en una fila donde cada uno de ellos representa un simple dígito binario (bit). Un “uno” binario corresponde a un interruptor en SI, y un “cero” binario corresponde a un interruptor en NO. Cada casilla de correo (posición de memoria) posee una única dirección tal que la información puede almacenarse y retirarse con confianza (no existen direcciones “gemelas”, hay una y solo una dirección posible para cada casilla).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

En un edificio de departamentos, la dirección de la casilla de correo puede ser de 100 a 175 para el primer piso, de 200 a 275 para el segundo, etc. Estos son números decimales que tienen sentido para la gente. Como se discutió anteriormente, las computadoras trabajan con el sistema de numeración binario.
Una computadora con cuatro cables (líneas) de dirección puede únicamente identificar 16 direcciones, ya que un conjunto de cuatro 1's y 0's pueden formar 16 combinaciones distintas.
Esta computadora podría identificar la dirección de las primeras 16 posiciones de memoria (casillas de correo) con los valores hexadecimales que van del $ 0 al $ F.

En los pequeños microcontroladores como el MC68HC705J1A (familia HC705 OTP de Motorola) hay diez líneas de dirección con las que estos MCUs pueden manejar 1024 (o 2 a la 10) posiciones de memoria únicas. Los microcontroladores de 8 bits de propósitos generales como el MC68HC11 o el MC68HC908 poseen 16 líneas de dirección con las que son capaces de direccionar 65536 (o 2 a la 16) posiciones de memorias únicas.

Elementos básicos de Lógica binaria.

A continuación, mostraremos algunos de los elementos básicos utilizados en el interior de un MCU para implementar la lógica binaria necesarias para muchas de sus funciones. Sugerimos al lector no experimentado con lógica binaria y álgebra de Boole, consultar en textos técnicos específicos sobre lógica binaria y álgebra Boolena. En el presente curso, se dará por sentado los conocimientos mínimos necesarios sobre este tema.

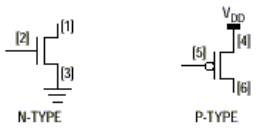
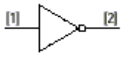


Fig. 3-a - Transistores CMOS Tipo "N" y Tipo "P"



Input [1]	Output [2]
0	1
1	0

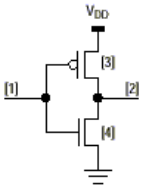
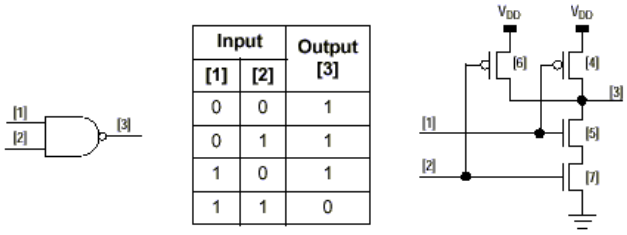


Fig. 3-b – Inversor CMOS.

Input [1]	Transistor		Output [2]
	[3]	[4]	
0	On	Off	Connected to V_{DD} (1)
1	Off	On	Connected to ground (0)

Tabla 3-a – Tabla operación del inversor CMOS.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

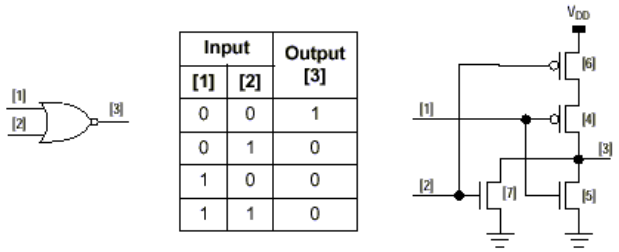


Input		Output
[1]	[2]	[3]
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 3-c – Compuerta NAND CMOS.

Input		Transistor				Output
[1]	[2]	[6]	[4]	[5]	[7]	[3]
0	0	On	On	Off	Off	V_{DD} (1)
0	1	Off	On	Off	On	V_{DD} (1)
1	0	On	Off	On	Off	V_{DD} (1)
1	1	Off	Off	On	On	GND (0)

Tabla 3-b – Tabla de operación Compuerta NAND CMOS.



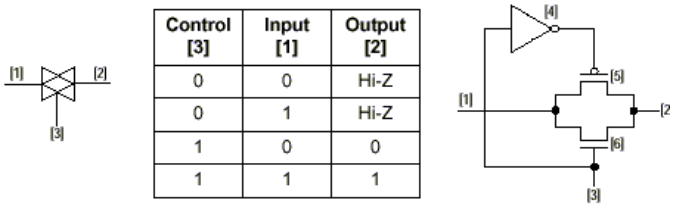
Input		Output
[1]	[2]	[3]
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 3-d – Compuerta NOR CMOS.

Input		Transistor				Output
[1]	[2]	[4]	[5]	[6]	[7]	[3]
0	0	On	Off	On	Off	V_{DD} (1)
0	1	On	Off	Off	On	GND (0)
1	0	Off	On	On	Off	GND (0)
1	1	Off	On	Off	On	GND (0)

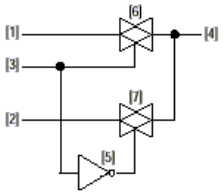
Tabla 3-c – Tabla de verdad para la compuerta NOR CMOS.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar



Control [3]	Input [1]	Output [2]
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

Fig. 3-e – Compuerta de Transmisión CMOS.



Select [3]	Input [1]	Input [2]	Output [4]
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 3-f – Multiplexor de Datos.

Select [3]	Input [1]	Input [2]	Transmission Gate [6]	Transmission Gate [7]	Output [4]
0	0	0	Off	On	0
0	0	1	Off	On	1
0	1	0	Off	On	0
0	1	1	Off	On	1
1	0	0	On	Off	0
1	0	1	On	Off	0
1	1	0	On	Off	1
1	1	1	On	Off	1

Tabla 3-d – Tabla de operación del Multiplexor.

Control [3]	Input [1]	Output [2]
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

Fig. 3-g – Buffer Tree – State CMOS.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Control [3]	Input [1]	Node		Transistor				Output [2]
		[4]	[9]	[5]	[6]	[7]	[8]	
0	0	1	1	Off	Off	Off	On	Hi-Z
0	1	0	1	Off	Off	On	Off	Hi-Z
1	0	1	0	On	On	Off	On	GND (0)
1	1	0	0	On	On	On	Off	V_{DD} (1)

Tabla 3-e – Tabla de operación Buffer Tree – State.

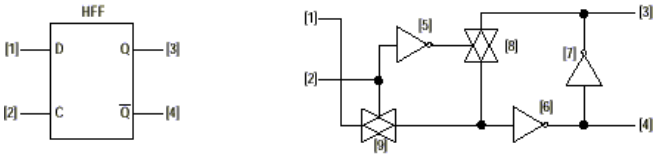


Fig. 3-h – Flip – Flop Tipo "D".

Como ve una Computadora a la memoria.

Una computadora de 8 bits con 10 líneas de dirección (podría ser el caso del MCU MC68HC705J1A) ve a la memoria como una columna continua de 1024 (ó 2 a la 10) valores de 8 bits. La dirección de la primer posición de memoria es 00 0000 0000 (2) y la de la última es 11 1111 1111 (2). La dirección de diez bits se expresa normalmente como dos números de 8 bits que se vuelcan en cuatro dígitos hexadecimales. En notación hexadecimal, el rango de estas direcciones va desde $0000 a $03FF.

La computadora especifica que posición está siendo accedida (lectura desde o escritura sobre) colocando una única combinación de unos y ceros sobre las líneas de dirección. La intención de leer la posición o escribir en ella se señaliza colocando un uno (leer) o un cero (escribir) sobre una línea denominada "leer / escribir" (R/W). La información de o para la posición de memoria es transportada por las ocho líneas de datos.

Para una computadora cualquier posición puede ser leída o escrita. No todos los tipos de memoria soportan la escritura, pero es tarea del programador saberlo, no de la computadora. Si el programador por error instruye a la computadora a escribir en una memoria de lectura solamente, esta tratará de hacerlo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Kilobytes, Megabytes y Gigabytes.

La menor unidad de memoria de una computadora es un simple **bit** que puede almacenar un valor de cero o uno. Estos bits agrupados en conjuntos de 8 bits forman un **Byte.** En computadoras más grandes se adiciona un agrupamiento en conjuntos de 16 o 32 bits formando una unidad denominada "word" (palabra). El tamaño de un word puede ser diferente para diferentes computadoras.

En el mundo decimal a veces expresamos valores muy pequeños o muy grandes recurriendo a prefijos antecediendo a la unidad de medida, tales como "mili", "kilo", etc.. En el mundo binario usaremos prefijos similares para describir grandes cantidades de memoria. En el sistema decimal, el prefijo "kilo"significa 1000 (o 10 a la 3) veces un valor. En el sistema binario, el entero potencia de 2 más próximo a 1000(10) es 2 a la 10 = 1024(10). Decimos "kilobytes"pero queremos decir "K bytes"que son 1024(10) bytes. Aunque ésta sea una terminología poco científica, se ha convertido en un estándar por su uso con el correr de los años.

Un megabyte es 2 a la 20 ó 1048576(10) bytes. Un gigabyte es 2 a la 30 ó 1073741824(10) bytes. Una PC con 32 líneas de dirección puede teóricamente direccionar 4 Gigabytes (4294967296(10)) de memoria.
Los pequeños microcontroladores abordados en este texto poseen sólo entre 512 bytes a 16 K bytes de memoria.

Tipos de Memoria

Las computadoras usan diversos tipos de información que requieren diferentes tipos de memoria. Las instrucciones que controlan la operación del microcontrolador son almacenadas en memoria **no-volátil** así el sistema no debe reprogramarse luego de apagar la fuente de alimentación. Tanto las variables de trabajo como los resultados intermedios necesitan ser almacenados en una memoria que permita una rápida y fácil escritura durante la operación del sistema. Si no es importante retener la información al apagar la alimentación podemos usar una memoria de tipo **volátil.** Este tipo de memoria es cambiada (escrita) y leída sólo por la CPU en la computadora.

Como otra información de memoria, las entradas de datos son leídas y las salidas de datos son escritas por la CPU. Las I/O y los registros de control son también formas de memoria para la computadora, pero difieren de otros tipos de memoria, ya que la información puede ser monitoreada y / o alterada por algún otro además de la CPU.

La RAM es una forma de almacenamiento volátil, que puede ser leída y escrita por la CPU. Como su nombre lo indica, las posiciones de memoria de una RAM pueden ser accedidas en cualquier orden. Este es el tipo más común de memoria en una PC. Una RAM requiere un área de silicio del circuito integrado relativamente grande. Por la importante área que necesitan (y por lo tanto su mayor costo), normalmente sólo pequeñas cantidades de RAM son integradas en los chips de microcontroladores.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Memoria de Lectura solamente

La ROM recibe la información que almacenará durante el proceso de fabricación. La información debe ser provista al fabricante antes que el circuito integrado que la contendrá sea fabricado. Cuando finalmente se use el microcontrolador, esta información podrá ser leída por la CPU pero cambiada. La ROM es considerada una forma de almacenamiento no-volátil puesto que la información no cambia si su fuente de alimentaciones apaga. La ROM es la más simple, más pequeña y más barata de las memoria no-volátiles.

ROM Programable (PROM)

La PROM es similar a una ROM excepto que es posible programarla con posterioridad a la fabricación del circuito integrado. Algunas de sus variantes son la PROM borrable **(EPROM),** PROM programable una única vez **(OTP)**, PROM borrable eléctricamente **(EEPROM)**, y, por último, la de más reciente aparición la memoria del tipo **FLASH.**

EPROM

La EPROM puede ser borrada exponiéndola a la luz ultravioleta. Los microcontroladores de este tipo de memoria, poseen una ventana de cuarzo que permite al exponerlo a la luz ultravioleta (de una longitud de onda determinada) su paso al interior del circuito integrado. El número de veces que una EPROM puede borrarse y reprogramarse está limitado a algunos cientos de ciclos dependiendo del dispositivo en particular y de la duración de la exposición a la luz ultravioleta durante el borrado de la misma.

Se usa un procedimiento especial para grabar información en una EPROM. La mayoría de los microcontroladores con EPROM recurren a una fuente de alimentación adicional tal como +12Vcc o tensiones similares durante la operación de programación de la EPROM.
La CPU no puede escribir información en las posiciones de memoria EPROM tal como lo hace sobre la memoria del tipo RAM.

Hay microcontroladores que contienen un circuito programador de EPROM, de modo que la CPU podrá grabar información en posiciones de memoria EPROM. Tal es el caso de los microcontroladores de Motorola MC68HC705C8A, MC68HC705P6A, etc. Durante la grabación de una EPROM, el chip debe retirarse del circuito eléctrico del que es parte, de modo que el programador maneje las direcciones y datos; y una vez programada debe volver a colocarse. En la analogía de las casillas de correo, se deberían retirar todas las casillas de correo del edificio para llenarlas con información y luego restituirlas. Mientras se procede a llenar las casillas, los habitantes del edificio o los usuarios de las casillas de correo no tienen acceso a ellas.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Algunos microcontroladores con EPROM (no el MC68HC705J1A) poseen un modo especial de programación que los hace aparecer como una memoria EPROM estándar. Estos dispositivos se graban recurriendo a programadores de EPROM comerciales de propósitos generales.

OTP (One Time Programming)

Cuando un microcontrolador con EPROM es encapsulado en una cápsula plástica opaca, se lo denomina microcontrolador programable una única vez u OTP. Ya que la luz ultravioleta no puede pasar a través de la cápsula, la memoria no puede ser borrada. El circuito integrado dentro del chip de un MCU OTP es idéntico al que viene encapsulado con una ventana de cuarzo. El encapsulado de plástico es mucho más barato que el cerámico con ventana de cuarzo. Los MCUs OPT son ideales para prototipos, primeras producciones y aplicaciones de pequeño volumen. En la actualidad este tipo de memoria es una de las más usadas tanto por el hobbista como por el profesional de la industria.

EEPROM (Electrical Ereasable PROM)

La EEPROM puede borrarse eléctricamente por comandos en un microcontrolador. A fin de grabar un nuevo valor en una posición de memoria, debemos primero borrar dicha posición y luego realizar una serie de pasos de programación. Esto es algo más complicado que alterar una posición de RAM que simplemente puede ser escrita con el nuevo valor por la CPU. La ventaja de la EEPROM radica en el hecho de ser una memoria no-volátil. Una EEPROM no pierde su contenido al apagar su fuente de alimentación. A diferencia de la RAM, el número de veces que una posición de memoria EEPROM puede ser borrada y reprogramada es limitado (típicamente 10000 ciclos), mientras que el número de lecturas es ilimitado. Esta facilidad de la no-volatilidad de los datos almacenados en EEPROM, constituyen una ventaja muy interesante a la hora de almacenar “tablas temporales”
cuyos datos Irán variando según una “personalización” para cada usuario en su aplicación.
En Motorola, la línea HC05 dispone de algunos dispositivos con memoria EEPROM, tal como el MC68HC805K3.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Memoria tipo "FLASH".

La memoria de tecnología "FLASH", es una memoria del tipo "no-volátil", similar en algunos aspectos a las de tecnología EEPROM, pero con muchas ventajas a estas últimas, que la han hecho en la práctica una memoria muy utilizada en los modernos microcontroladores de las grandes marcas. La tecnología FLASH, es del tipo borrable y programable eléctricamente al igual que una EEPROM, pero a diferencia de esta se han reducido a la mínima expresión los circuitos auxiliares de acceso orientados al "byte", esto significa que mientras en una memoria EEPROM es posible borrar un solo Byte en una posición determinada a lo largo de toda la memoria (acceso orientado al byte), en una memoria con tecnología FLASH no es posible acceder al borrado de un solo byte en forma arbitraria, ya que no se cuenta con los mecanismos de acceso (circuitos de direccionamiento) que permitan el borrado de un byte independientemente del resto.

En la práctica, este inconveniente en el borrado, se compensa con mayor esfuerzo en "software" por parte del MCU (por lo general, muchos MCUs ya tienen incorporadas rutinas internas que facilitan los procedimientos de borrado y programación de este tipo de memorias), y una mayor velocidad en la programación de la misma.
El ahorro que se produce por no utilizar la circuitería adicional de acceso, se ve reflejado en un menor costo de implementación final de este tipo de memorias en un MCU, además de una mayor densidad de circuitos de memoria versus superficie ocupada por los mismos.
Tal es así, que mientras los MCUs con memoria del tipo EEPROM eran considerados una "rara avis" en la mayoría de las aplicaciones masivas de consumo, por su elevado costo (2 ó 3 veces más caros que las versiones similares en OTP), en los MCUs FLASH, van creciendo día a día las aplicaciones para ellos, ya que su costo no es significativamente mayor a los difundidos OTP (entre un 20 a 30 % superior a estos).

Entre otras de las muchas ventajas que presentan los MCUs FLASH, es la de posibilitar la programabilidad "En – Circuito", esto significa que no es necesaria la remoción del microcontrolador FLASH del circuito en donde esté formando parte, ya que estos modernos MCUs poseen pines especiales para "comunicarlos" con el mundo exterior (vía una PC) y permitir programarlos, borrarlos y actualizarlos de forma sencilla y segura. Además, muchos de ellos poseen circuitería interna que elimina el uso de tensiones elevadas especiales, como las utilizadas en las memorias del tipo EPROM, EEPROM.

La durabilidad (número ciclos de borrado - programación) en las tecnologías FLASH de muchos fabricantes es similar a la de las EEPROM. Por ejemplo, la familia HC908 FLASH de Motorola permite unos 10.000 ciclos de Borrado - Programación a una temperatura de funcionamiento de - 40 ° C o 125 ° C (la peor condición) y de más de 100.000 ciclos a temperaturas de entre 20 ° C y 30 ° C.

En capítulos posteriores, veremos en detalle las ventajas de esta nueva tecnología de memoria.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Dispositivos de I/O como un tipo de memoria.

La información del estado y control de I/O (Input / Output o Entrada / Salida) es un tipo de posición memoria que permite al sistema de computadora intercambiar información desde o hacia su mundo exterior. Este tipo de memoria es poco usual, ya que la información puede ser "monitoreada" y / o alterada por algún otro además de la CPU.

Los tipos más simples de posiciones de memoria de I/O son un simple "port" (puerto) de entrada y un simple port de salida. En un MCU de 8 bits, un simple port de entrada (input) contiene 8 pines (por lo general un port típico tiene 8 pines, pero como veremos más adelante, en algunos MCUs la cantidad puede ser inferior a 8 en algún port en particular) que pueden ser monitoreados (leídos) por la CPU. Un simple port de salida (output) contiene 8 pines (por lo general un port típico tiene 8 pines, pero como veremos más adelante, en algunos MCUs la cantidad puede ser inferior a 8 en algún port en particular) que pueden ser controlados (escritos) por la CPU.

En la práctica, la implementación de un port de salida es un óctuple "data latch" y las realimentaciones que permitan a la CPU leerlo. La figura 3-1 muestra el circuito equivalente para un bit de una RAM, un bit de un port de entrada y un bit de un típico port de salida que puede ser leído.
En un MCU real, estos circuitos deberán repetirse ocho veces para armar una simple posición de memoria RAM de 8 bits, port de entrada o port de salida (o menos según la cantidad de pines del port). En la figura 3-1 los half flip - flops (HFF) son simples flip - flops transparentes. Cuando la señal de reloj (C) está en alto, los datos pasan libremente de la entrada "D" a las salidas Q y Q (Negado). Cuando la entrada de reloj es baja, el dato es sostenido en las salidas Q y Q (Negado).

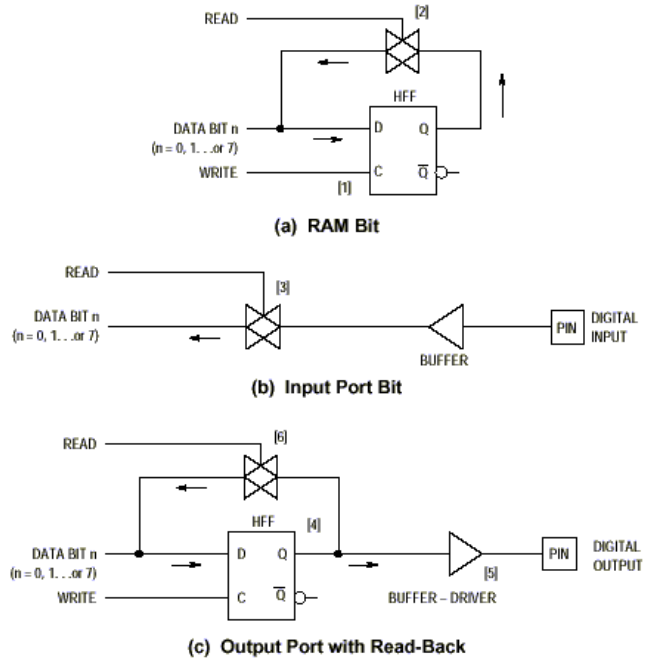


Fig. 3-1 – Circuitos de Memoria e I/O.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Cuando la CPU almacena un valor en la dirección que corresponde al bit de RAM de la figura 3-1 (a), se activa la señal WRITE para almacenar el dato proveniente de las líneas del bus de datos en el flip-flop (1). Este almacenamiento es estático y retendrá el valor escrito hasta que un nuevo valor sea escrito en esta posición (o se apague la fuente de alimentación). Cuando la CPU lee en la dirección de este bit de RAM, se activa la señal READ, que habilita al multiplexor (2). El multiplexor acopla el dato de la salida del flip-flop con la línea del bus de datos. En una MCU real, los bits de RAM son mucho más sencillos que los mostrados, pero su funcionamiento es equivalente al del circuito visto.

Cuando la CPU lee un valor en la dirección que corresponde al bit del port de entrada de la figura 3-1 (b), se activa la señal READ que habilita al multiplexor (3). El multiplexor acopla el dato del buffer con la línea del bus de datos. Una escritura en esta dirección no tiene efecto alguno.

Cuando la CPU almacena un valor en la dirección que corresponde al bit del port de salida de la figura 3-1 (c), se activa la señal WRITE para almacenar el dato proveniente de las líneas del bus de datos en el flip-flop (4). La salida del latch, llega a través de un buffer (5) al pin de salida como una señal READ, que habilita al multiplexor (6). El multiplexor acopla al dato desde la salida del flip-flop con la línea del bus de datos.

Registros Internos de Estado y de Control

Los registros internos de estado y de control son una versión especial de posiciones de memoria de I/O. Con el objeto de monitorear y controlar pines externos, los registros de estado y de control monitorean y controlan los niveles lógicos de las señales internas.

Mire en la figura 3-1 y compare un bit de RAM con uno de port de salida. La única diferencia es que el port de salida posee un buffer que conecta al estado del flip-flop con el pin exterior. En el caso de un bit interno de control el buffer de salida está conectado a alguna línea de control al igual que el pin exterior.

Los microcontroladores M68HC05 incluyen pines de I/O de ports paralelos de propósito general. El sentido de cada pin se puede programar por un bit de control accesible por programa. La figura 3-2 presenta la lógica para un pin de I/O bidireccional incluyendo un latch port de salida y un bit de control de dirección de dato **(DDR bit).**

Un pin de port se configura como salida si en su correspondiente DDR bit es escrito un uno lógico. Un pin de port se configura como entrada si en su correspondiente DDR bit es resetearse, todos los DDR bits son borrados, lo que configura a todos los pines como entrada. Los DDRs pueden ser tanto escritos como leídos por el procesador.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

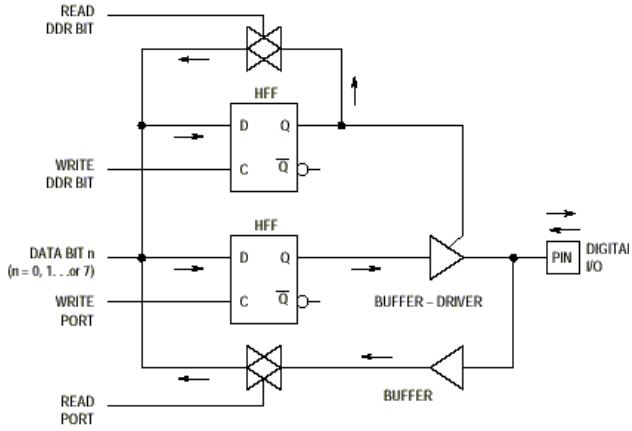


Fig. 3-2 – Puerto de I/O con Data Direction Control.

Mapas de Memoria

A veces hay miles de posiciones de memoria o más en un sistema de MCU, es importante entonces contar con un medio conveniente para no perder de vista su contenido. Un **mapa de memoria** es una representación gráfica de la totalidad de la memoria del MCU. La figura 3-4 es un típico mapa de memoria que nos presenta los recursos de memoria del MC68HC705J1A.

Los valores hexadecimales de cuatro dígitos del lateral izquierdo de la figura 3-4 son direcciones que van comenzando desde $0000 arriba y en forma creciente hasta $07FF abajo. $0000 corresponde a la primera posición de memoria (seleccionada cuando la CPU pone cero lógico en todas las líneas de dirección del bus de direcciones). $07FF corresponde a la última posición de memoria (seleccionada cuando la CPU pone uno lógico en todas las líneas de dirección del bus interno de direcciones). Los textos dentro del rectángulo vertical identifican el tipo de memoria (RAM, EPROM, registros de I/O, etc.), residentes en un área de memoria en particular.

Es necesario ver con mayor detalle a algunas áreas, tales como la de registros de I/O, ya que es importante conocer el nombre de cada posición individual. El rectángulo vertical completo puede interpretarse como una fila de 2048 (ó 2 a la 11) casillas de palomas (posiciones de memoria). Cada una de éstas 2048 (ó 2 a la 11) posiciones de memoria contiene 8 bits de datos como se muestra en la figura 3-3.

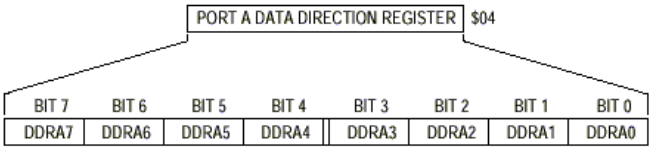


Fig. 3-3 – Detalle expandido de una locación de memoria.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Las primeras 256 posiciones de memoria ($0000-$00FF) pueden ser accedidas por la computadora en un modo de **direccionamiento directo.** Discutiremos los modos de direccionamientos más detalladamente en el capítulo 5. En direccionamiento directo, la CPU asume que los dos dígitos hexadecimales de mayor peso de una dirección son siempre cero; así, sólo son necesarios los dos dígitos de menor peso de la dirección para explicitarla dentro de una instrucción. Los registros de I/O del chip y 64 bytes de RAM se ubican en el área de memoria $0000-$00FF. En el mapa de memoria (figura 3-4) la expansión del área de I/O identifica la posición de cada registro con los dos dígitos de menor peso de su dirección en lugar de usar los cuatro dígitos completos. Por ejemplo, el valor hexadecimal de dos dígitos $00 aparece a la derecha del registro de datos del port A, que realmente está en la posición la dirección $0000 del mapa de memoria.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

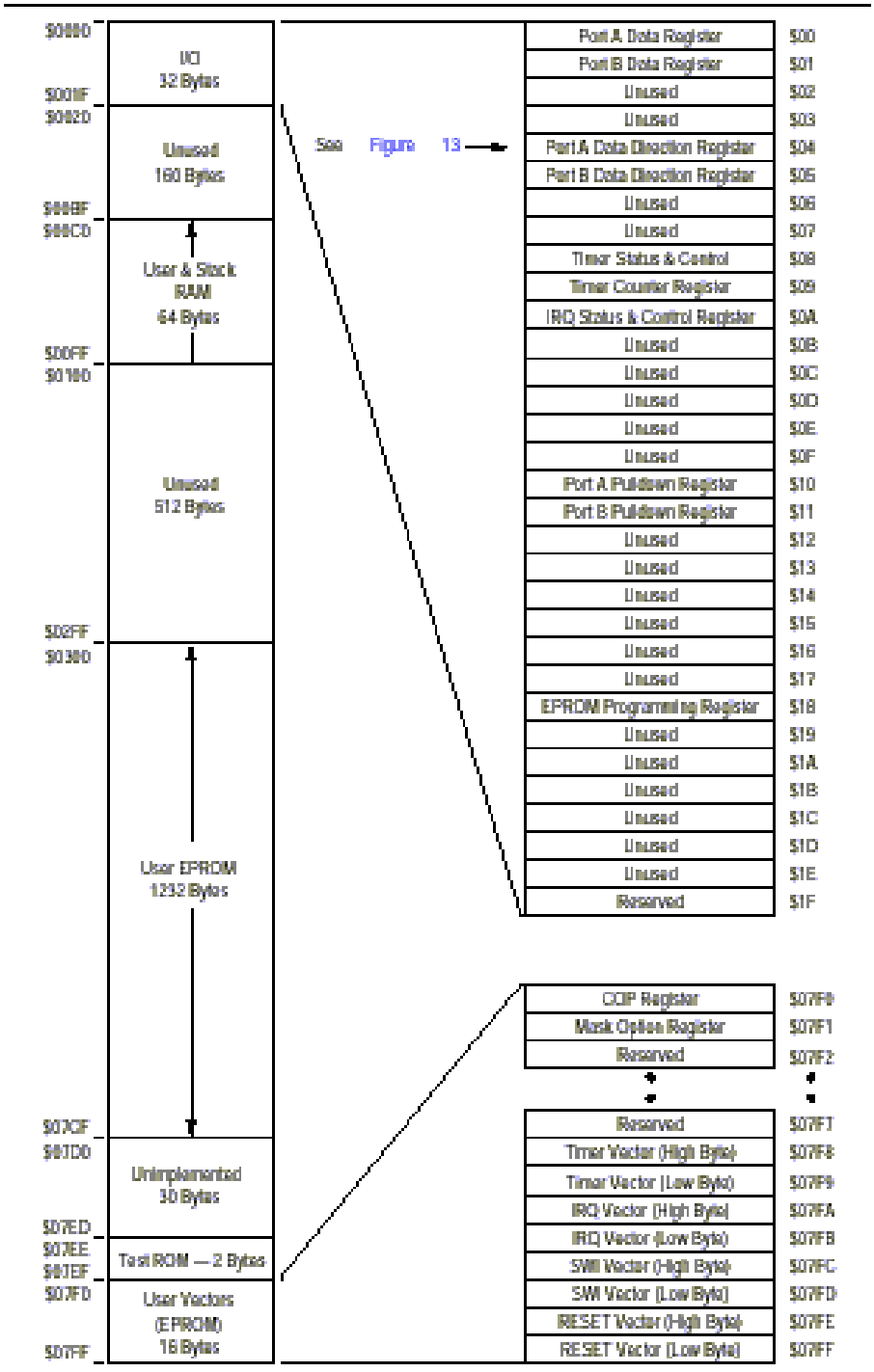


Fig. 3-4 – Mapa de Memoria Típico.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Memorias Periféricas

Las memoria pueden ser una forma de periféricos. Ya hemos discutido el uso de diferentes tipos de memoria, pero sin considerar la lógica requerida para soportarlas. Las ROM y RAM son muy íntegras y no requieren lógica de soporte más que la lógica de selección de direcciones para distinguir una posición de otra. Esta lógica es provista en el mismo chip que aloja a la memoria.

Las memorias EPROM (PROM borrable) y EEPROM (PROM borrable eléctricamente) requieren una lógica de soporte para grabarlas (y borrarlas en las EPROM). La lógica periférica de soporte para el MC68HC705J1A es similar a un grabador de PROM contenido en la misma MCU. Un registro de control incluye bits de control para seleccionar entre los modos de grabación y lectura, y habilitar la fuente de alimentación de programación de alta tensión.

Revisión del Capítulo 3

Nos figuramos a la memoria como un conjunto de casillas de correo. La revisión de una computadora presenta a la memoria como una serie de valores de 8 bits.

Si una computadora tiene **n** líneas de dirección, ella puede únicamente direccionar **2 a la n** posiciones de memoria. Una computadora con once líneas de dirección puede direccionar **2 a la 11** ó **2048 (10)** posiciones.

Un kilobyte (que se escribe 1K byte) es igual a 1024(10) bytes.

Tipos de Memoria

- **RAM.** La memoria de acceso aleatorio puede ser leída y escrita por una CPU. Su contenido se mantiene mientras la fuente de alimentación esté encendida.
- **ROM.** La memoria de lectura solamente puede ser leída más no escrita. El contenido debe establecerse antes de la fabricación del circuito integrado. No necesitan alimentación para mantener su contenido.
- **EPROM.** La memoria ROM programable y borrable pueden cambiar su contenido previo proceso de borrado mediante la exposición a la luz ultravioleta, y entonces programarlas con un nuevo valor. Las operaciones de borrado y grabación pueden realizarse un número limitado de veces desde su fabricación. No necesitan alimentación para mantener su contenido..
- **OTP.** El chip en una EPROM programable una única vez es idéntico al de una EPROM, pero su encapsulado es opaco. Así la luz ultravioleta no puede atravesar el encapsulado, esta memoria no puede ser borrada una vez que ha sido grabada.
- **EEPROM.** La memoria PROM borrable eléctricamente permite que su contenido sea cambiado usando señales eléctricas y reteniendo su contenido aún sin alimentación. Típicamente una posición de memoria EEPROM puede borrarse y grabarse hasta alrededor de 100.000 o 1.000.000 veces.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

- FLASH este tipo de memoria es similar a las EEPROM pero el proceso de borrado no está orientado al byte sino a la "página" que puede ser de 32 / 64 o 128 bytes según la estructura interna de la memoria FLASH. La principal ventaja de este tipo de memorias es su bajo valor en relación a las EEPROM lo que permite MCUs reprogramables y con capacidad de almacenamiento no-volátil a precios comparables a la tecnología OTP.
- **I/O.** Los registros de I/O, de estado y de control, son un tipo especial de memoria puesto que la información puede ser monitoreada y / o alterada por algún otro además de la CPU.

Una memoria no-volátil mantiene su contenido aún cuando se apague su fuente de alimentación.

Una memoria volátil pierde su contenido al apagar su fuente de alimentación.

Un mapa de memoria es una representación gráfica de la totalidad de la memoria de un sistema de computadora.

Las primeras 256 posiciones de memoria en un microcontrolador pueden ser accedidas de un modo especial denominado **modo de direccionamiento directo.** En este modo la CPU asume que el byte de mayor peso de una dirección es $00 no siendo necesario explicitarlo al escribir un programa (ahorrando el espacio que ocuparía, así como los ciclos de reloj que requeriría su búsqueda).

En especial las memorias tales como EPROM y EEPROM pueden considerarse periféricas en un sistema de computadora. Se requiere una lógica de soporte y control de programación para modificar su contenido. Esto difiere de simples memorias tales como las RAM que pueden ser leídas o escritas en simples ciclos de reloj de la CPU.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 4.

Arquitectura de una Computadora.

En este capítulo tomaremos el corazón de una computadora para ver qué hacen sus latidos. Esta será una más detallada revisión de la normalmente necesaria para poder **usar** una MCU pero nos ayudará a entender por qué algunas cosas se hacen de cierta manera.

Toda actividad de la CPU puede dividirse en secuencias de muy simples etapas. Un reloj oscilador genera un reloj de CPU que es usado para mover a la CPU a través de esas secuencias. El reloj de CPU en términos humanos es muy rápido, nos parece que las cosas ocurren casi instantáneamente.
Siguiendo esta secuencias paso a paso, ganaremos un trabajado entendimiento de cómo una computadora ejecuta los programas. Además de ganar un valioso conocimiento de las capacidades y limitaciones de una computadora.

Arquitectura de una Computadora

Los **MCUs de 8 bits** de Motorola M68HC05 tienen una organización específica denominada arquitectura de Von Neumann, en honor a un matemático norteamericano. En esta arquitectura, un CPU y un arreglo de memoria se interconectan mediante un conjunto de líneas **(bus)** de direcciones **(address)** y otro de líneas de datos **(data)**. El **bus de direcciones** es usado para indicar qué posición de memoria está siendo accedida y el **bus de datos** se usa para transportar la información desde el CPU a la **posición de memoria** (casilla para palomas) o bien desde la posición de memoria al CPU.

En la implementación hecha por Motorola de esta arquitectura, hay algunas pocas casillas para palomas especiales (denominadas registros del CPU) en el interior del CPU, que actúan como pequeños anotadores de borrador y tablero de control del CPU. Los registros del CPU son similares a las memorias en que la información puede ser escrita y retenida en su interior. Es importante recordar que ellos están cableados en el CPU **(registros internos)** y no forman parte de la memoria direccionable por la CPU **(registros externos).**

Toda la información (no la de los registros de la CPU) accesible a la CPU es vista (por la CPU) como una única fila de miles de casillas para palomas o más. A veces esta organización es llamada sistema de **I/O mapeado en memoria** dado que la CPU trata a toda posición de memoria por igual aunque ella contenga **instrucciones de programa, variables** de datos o **entradas / salidas (I/O)** de control. Hay otras arquitecturas de computadoras, pero no es el objetivo de este texto explorarlas. Afortunadamente, la arquitectura M68HC05 de Motorola que discutiremos es una de las más fáciles de entender y usar.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Esta arquitectura abarca las más importantes ideas de las computadoras binarias digitales; por lo tanto, la información presentada en este texto será igualmente aplicable si se propone estudiar otras arquitecturas.

El número de líneas del bus de direcciones determina número total de casillas para palomas: el número de líneas del bus de datos determina la cantidad total posible de información que se puede almacenar en cada casilla para palomas. En el M68HC705J1A, el bus de direcciones tiene 11 líneas, que hacen un máximo de 2048 (ó 2 a la 11) casillas para paloma independientes (en la jerga de los MCU diremos que esta CPU puede acceder 2K posiciones de memoria). Como el bus de datos del MC68HC705J1A es de ocho bits, cada casilla para paloma puede almacenar un byte de información un byte es ocho dígitos binarios, o dos dígitos hexadecimales, o un carácter ASCII, o un valor decimal de 0 á 255.

Registros de la CPU

Diferentes CPU poseen diferentes conjuntos de registros internos. Las diferencias son principalmente el número y el tamaño de estos registros. En la figura 4-1 se presentan los registros de la CPU que encontramos en MC68HC05. Este es un relativamente sencillo conjunto de registros de la CPU, que resulta ser representativo de todos los tipos de registros de la CPU y puede usarse para explicar todo sobre los conceptos fundamentales. Este capítulo brinda una breve descripción de los registros del MC68HC05 como una introducción a la arquitectura de la CPU en general. Veremos información más detallada respecto a los registros del MC68HC05 en otro capítulo de este texto denominado repertorio de instrucciones del MC68HC05.

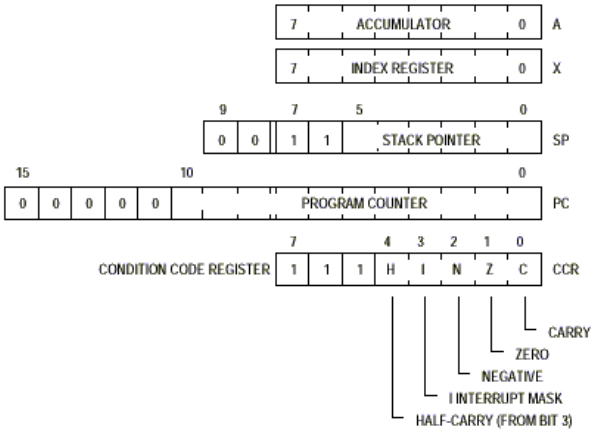


Fig. 4-1 – Registros del CPU MC68HC705

El registro A, es un registro de almacenamiento temporário de 8 bits, se lo llama acumulador ya que se usa a menudo para alojar uno de los **operandos** o el resultado de una operación aritmética.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

El registro X, es un registro índice de 8 bits, que además puede servir como un simple registro de almacenamiento temporario. El principal propósito de un registro índice es apuntar a un área en la memoria desde donde la CPU cargará (leer) o almacenará (escribir) información. Un registro índice suele llamarse **registro puntero.** Entenderemos más respecto a índices cuando discutamos los modos de direccionamiento indexado.

El registro contador de programa **(PC)** es usado por la CPU para no perder de vista la dirección de la próxima instrucción a ejecutar. Al resetear la CPU (encenderla), la PC es cargado con el contenido de un par de posiciones de memoria específico denominado **vector de reset.** Las operaciones del vector de reset contienen la dirección donde está almacenada la lógica interna de la CPU incrementa a la PC de modo tal que siempre apunte a la próxima parte de información que la CPU puede necesitar. El número de bits de la PC coincide exactamente con el número de líneas del bus de direcciones. Esto determina el total espacio de memoria potencialmente disponible que puede ser accedido por la CPU.
En el caso del MC68HC705J1A, la PC es de 11 bits; con el que, su CPU podrá acceder hasta 2K (2048 ó 2 a la 11) bytes de memoria. Los valores para este registro se expresan mediante cuatro dígitos hexadecimales donde los cinco bits más significativos de la correspondiente dirección binaria de 16 bits son siempre ceros.

El registro de código de condición **(CCR)** es de 8 bits, almacena indicadores de estado que reflejan al resultado de alguna operación previa de la CPU. Los tres bits de más peso de este registro no se usan y siempre son iguales a cero lógico. Las **instrucciones de bifurcación** usan a los bits de estado para tomar simples decisiones respecto a su estado.

El puntero a la pila **(stack pointer, SP)** es usado como puntero a la próxima posición disponible de una pila **(stack)** del tipo último en ingresar / primero en salir (LIFO). El stack puede tomarse como una pila de cartas, en la que cada carta almacena un solo byte de información. En el momento que se desee, la CPU puede poner una carta arriba de la pila o retirar una de arriba de la pila. No podemos retirar una carta del interior de la pila antes de haber retirado todas las cartas que tiene encima. La CPU sigue el comportamiento del stack con el SP. El SP apunta a la posición de memoria (casilla para palomas libre), la que se considera es la dirección de la próxima carta disponible. Cuando la CPU agrega una porción de dato en la pila, el dato es escrito en la casilla para palomas apuntada por el SP y el SP luego se decrementa para apuntar a la posición de memoria previa (casilla para palomas anterior libre). Cuando la CPU retira una porción de dato de la pila, el SP se incrementa para apuntar a la casilla para palomas previamente usada y de ella se retira el dato. Al encender la CPU o bien luego de ejecutar la instrucción reset stack pointer (RSP), el SP apunta a una específica posición de memoria en RAM (a cierta casilla para palomas).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Temporización

Una fuente de **reloj** de alta frecuencia (típicamente derivada de un cristal conectado al MCU) se usa para controlar el secuenciamiento de las instrucciones de la CPU. Los MCU típicos dividen la frecuencia base del cristal por dos o más para obtener un reloj de frecuencia de bus. Cada lectura o escritura de memoria toma un período (ciclo) de la frecuencia de bus. En el caso del MC68HC705J1A, un reloj oscilador a cristal de 4 MHz (máximo) dividido por 2 obtiene la frecuencia interna de reloj del procesador de frecuencia de bus (500 nS mínimo). La mayoría de las instrucciones toman de dos a cinco de estos pasos; por lo tanto, la CPU es capaz de ejecutar más de 500000 instrucciones por segundo.

Programa visto por la CPU

El listado 4-1 es de un pequeño programa de ejemplo que usaremos en nuestra discusión de la CPU. El capítulo de programación provee mucha más información que la que necesita ka CPU para que una persona pueda leer y entender programas. En la primera columna se presentan direcciones de cuatro dígitos hexadecimales. Las tres columnas que le siguen, son valores de 8 bits (el contenido de una única posición de memoria). El resto de la información del listado es para ayudar a la persona que necesita leer el listado.
El significado de esta información será discutido más detalladamente en el capítulo de programación.

La figura 4-2 es un mapa de memoria del MC68HC705J1A, presenta cómo el programa de ejemplo se aloja en la memoria del MCU. Esta figura es similar a la figura 3-4 excepto que una diferente porción del espacio de memoria ha sido expandida para presentar el contenido de todas las posiciones de memoria en el programa de ejemplo.

La figura 4-2 muestra que la CPU ve al programa de ejemplo como una secuencia lineal de códigos binarios, incluyendo instrucciones y **operandos** en posiciones de memoria consecutivas. Un operador es cualquier valor, que no es un código de operación, y que la CPU necesita para completar una instrucción. La CPU inicia este programa con su contador de programa (PC) apuntando al primer byte del programa. El código de operación de cada instrucción le dice a la CPU cuántos (si los hay) y de qué tipo son, los operandos que vienen en la instrucción. Así la CPU puede mantenerse en línea con los límites de una instrucción sin mezclar códigos de operación con operandos, lo que se nos presenta confuso.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
                        ********************************************************
                        * Simple 68HC05 Program Example                        *
                        * Read state of switch at port A bit-0; 1=closed       *
                        * When sw. closes, light LED for about 1 sec; LED on   *
                        * when port A bit-7 = 0. Wait for sw release,          *
                        * then repeat. Debounce sw 50mS on & off              *
                        * NOTE: Timing based on instruction execution times   *
                        *  If using a simulator or crystal less than 4MHz,    *
                        *  this routine will run slower than intended         *
                        ********************************************************
                        $BASE   10T                 ;Tell assembler to use decimal
                                                    ;unless $ or %  before value
0000                    PORTA   EQU    $00          ;Direct address of port A
0004                     DDRA    EQU    $04           ;Data direction control, port A
00E0                     TEMP1   EQU    $C0           ;One byte temp storage location

0300                            ORG    $0300         ;Program will start at $0300

0300  A6 80              INIT    LDA    #$80          ;Begin initialization
0302  B7 00                      STA    PORTA         ;So LED will be off
0304  B7 04                      STA    DDRA          ;Set port A bit-7 as output
                        * Rest of port A is configured as inputs

0306  B6 00              TOP     LDA    PORTA         ;Read sw at LSB of Port A
0308  A4 01                      AND    #$01          ;To test bit-0
030A  27 FA                      BEQ    TOP           ;Loop till Bit-0 = 1
030C  CD 03 23                   JSR    DLY50         ;Delay about 50 mS to debounce
030F  1F 00                      BCLR   7,PORTA       ;Turn on LED (bit-7 to zero)
0311  A6 14                      LDA    #20           ;Decimal 20 assembles to $14
0313  CD 03 23  DLYLP   JSR    DLY50         ;Delay 50 ms
0316  4A                         DECA                 ;Loop counter for 20 loops
0317  26 FA                      BNE    DLYLP         ;20 times (20-19,19-18,...1-0)
0319  1E 00                      BSET   7,PORTA       ;Turn LED back off
031B  00 00 FD  OFFLP   BRSET  0,PORTA,OFFLP ;Loop here till sw off
031E  CD 03 23                   JSR    DLY50         ;Debounce release
0321  20 E3                      BRA    TOP           ;Look for next sw closure

                        ***
                        * DLY50 = Subroutine to delay ~50ms
                        * Save original accumulator value
                        * but X will always be zero on return
                        ***

0323  B7 C0              DLY50   STA    TEMP1         ;Save accumulator in RAM
0325  A6 41                      LDA    #65           ;Do outer loop 65 times
0327  5F                OUTLP   CLRX                 ;X used as inner loop count
0328  5A                INNRLP  DECX                 ;0-FF, FF-FE,...1-0 256 loops
0329  26 FD                      BNE    INNRLP        ;6cyc*256*500ns/cyc = 0.768ms
032B  4A                         DECA                 ;65-64, 64-63,...1-0
032C  26 F9                      BNE    OUTLP         ;1545cyc*65*500ns/cyc=50.212ms
032E  B6 C0                      LDA    TEMP1         ;Recover saved Accumulator val
0330  81                         RTS                  ;Return
```

Listado 4-1 – Programa de Ejemplo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

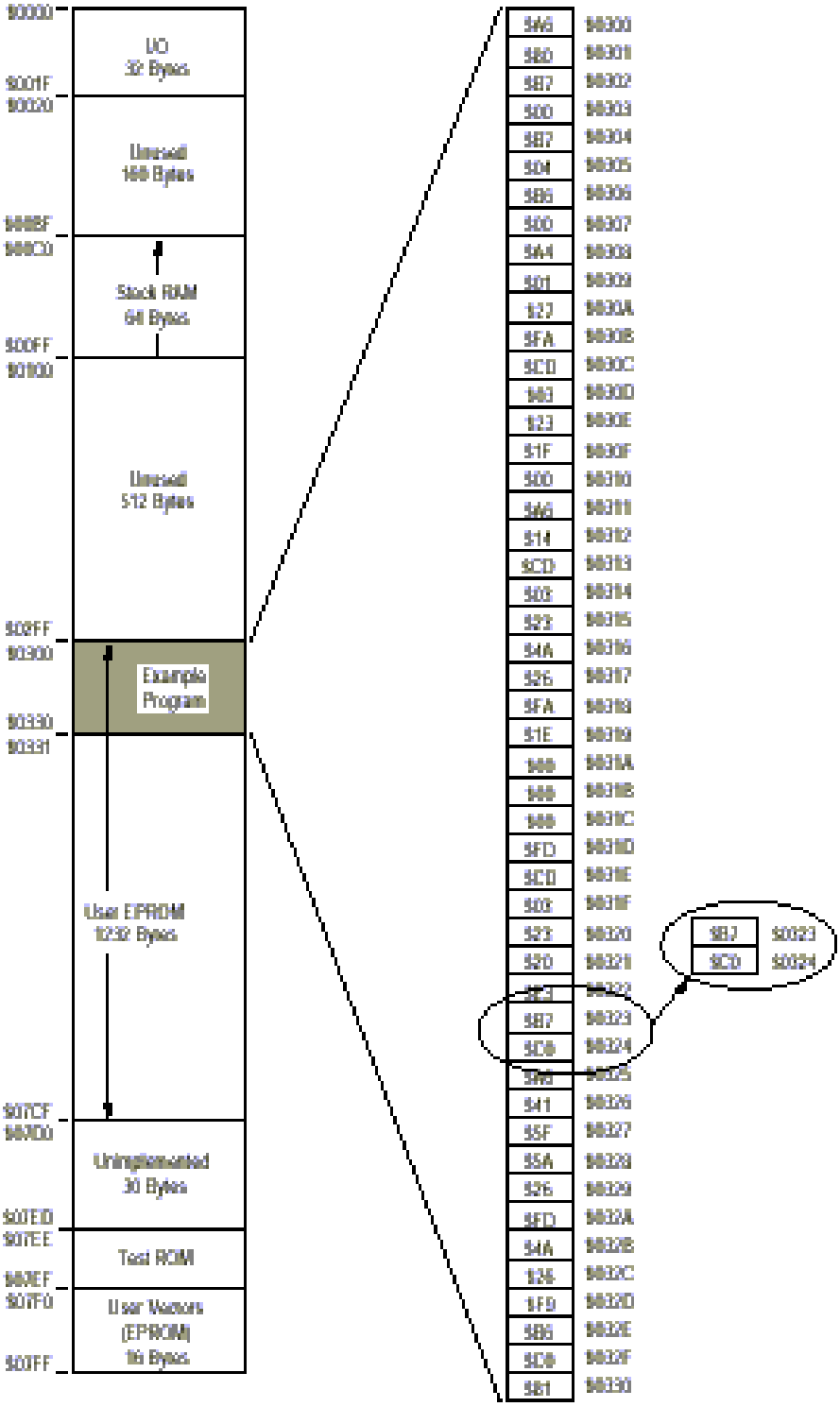


Fig. 4-2 – Mapa de memoria del Programa Ejemplo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

La mayoría de los programas de aplicación podrían alojarse en ROM, EPROM u OTPROM / FLASH, aunque no es un requerimiento especial las instrucciones deberán estar en una memoria tipo ROM para ser ejecutadas. En lo que a la CPU respecta, un programa es sólo una serie de patrones de bits que procesará secuencialmente.

Estudiaremos cuidadosamente el listado 4-1 del programa de ejemplo y el mapa de memoria de la figura 4-2. encontraremos la primera instrucción de la subrutina DLY50 en el listado 4-1 y luego encontraremos los mismos dos bytes en la figura 4-2.

Habremos encontrado una línea de la siguiente forma casi al final del listado 4-1.

0323 B7 EO DLY50 STA TEMP1 ; Save accumulator en RAM

La sección resaltada de memoria a la derecha de la figura 4-2 es el área que habremos identificado.

Operación de la CPU

En esta sección discutiremos primero la operación detallada de las instrucciones de la CPU y luego explicaremos cómo la CPU ejecutaría el programa de ejemplo. Trataremos de hacer una detallada descripción de típicas instrucciones de la CPU para pensar cómo lo hace una CPU. Podemos entonces recorrer el programa de ejemplo usando una técnica de adiestramiento llamada “jugando a la computadora”en la que pretenderemos ser una computadora interpretando y ejecutando las instrucciones de un programa.

Operación detallada de las Instrucciones de la CPU

Antes de ver cómo la CPU ejecuta los programas, nos ayudaría conocer (en detalle) cómo la CPU divide una instrucción en operaciones fundamentales y realiza estas pequeñas etapas para llevar a cabo una determinada instrucción. Como veremos, muchos pequeños pasos se ejecutan rápidamente y con suma precisión dentro de cada instrucción, pero nada de estas pequeñas etapas es demasiado complicada.

El circuito de lógica dentro de la CPU parecería ríspido para un ingeniero habituado a trabajar con lógica TTL o bien con lógica de relevadores. Lo que hace del MCU y su CPU otras formas de lógica digital es la densidad del encapsulado. Las técnicas de muy alta escala de integración (VLSI) hacen posible colocar el equivalente de miles de circuitos integrados TTL en una simple pastilla de silicio. Ordenando las compuertas de lógica para formar una CPU, podemos obtener un ejecutor de instrucciones de propósito general capaz de actuar como una **caja negra** universal. Colocando diferentes combinaciones de instrucciones podría virtualmente realizar cualquier función definible.

Una típica instrucción toma de dos a cinco períodos del reloj interno del procesador. Aunque normalmente no es importante saber con exactitud qué sucede durante cada uno de estos períodos de ejecución, será de ayuda recorrer algunas instrucciones en detalle para entender cómo trabaja internamente la CPU.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Almacenamiento del Acumulador (Modo de Direccionamiento Directo)

Echemos un vistazo a la instrucción **STA** en el apéndice A. En la tabla de la parte inferior de la página, veremos que **$B7** es la versión en modo de direccionamiento directo (DIR) de la instrucción almacenamiento del acumulador. Además veremos que la instrucción requiere dos bytes, el primero para especificar el código de operación ($B7) y el segundo para especificar la **dirección directa** donde será almacenado el acumulador. (Los dos bytes se presentan como "B7 dd"en la columna de código de máquina de la tabla).

Discutiremos el modo de direccionamiento directo más detalladamente en otro capítulo, pero la breve descripción siguiente nos ayudará a entender cómo la CPU ejecuta esta instrucción. En el modo de direccionamiento directo, la CPU asume que la dirección está en el rango de $0000 a $00FF; en consecuencia, no es necesario incluir los dos dígitos de más peso de la dirección del operando en la instrucción (pues es siempre $00).

La tabla de la parte inferior de la página correspondiente a STA nos muestra que la versión de STA en modo de direccionamiento directo toma cuatro períodos de la CPU para su ejecución. Durante el primer ciclo, la CPU vuelca el contenido del contador de programa sobre el bus interno de direcciones y lee el código de operación $B7, que identifica a la instrucción como la versión en modo de direccionamiento directo de la instrucción STA y avanza a la PC hasta la próxima posición de memoria.

Durante el segundo ciclo, la CPU vuelca el contenido de la PC sobre el bus interno de direcciones y lee el byte de menos peso de la dirección directa ($00 por ejemplo). La CPU usa el tercer ciclo de esta instrucción STA para armar en su interior la dirección completa donde se almacenará el acumulador, ya que avanza al PC hasta la próxima posición de memoria (la dirección del código de operación de la próxima instrucción).

En este ejemplo, la CPU une el valor asumido $00 (por ser de modo de direccionamiento directo) con el $00 que fue leído durante el segundo ciclo la instrucción para obtener la dirección completa $0000. durante el cuarto ciclo de esta instrucción la CPU vuelca la dirección construida sobre el bus interno de direcciones, además vuelca el contenido del acumulador sobre el bus interno de datos y acciona la señal de escribir. Esto es, la CPU escribe el contenido del acumulador en la posición de memoria $0000 durante el cuarto ciclo de la instrucción STA.

Mientras el acumulador está siendo almacenado, los bits N y Z del registro de código de condición son modificados de acuerdo al dato almacenado. La fórmula de lógica booleana para estos bits está en la mitad de la página de la instrucción STA. El bit Z irá a uno lógico si el valor almacenado es $00; sino, el bit Z irá a cero lógico. El bit N irá a uno lógico si el bit más significativo del valor almacenado es uno; sino el bit N irá a cero lógico.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Carga del Acumulador (Modo de Direccionamiento Inmediato)

Echemos un vistazo a la instrucción **LDA** en el apéndice A. En la tabla de la parte inferior de la página, veremos que **$A6** es la versión en modo de direccionamiento inmediato (IMM) de la instrucción carga del acumulador. Además veremos que la instrucción requiere dos bytes, el primero para especificar el código de operación ($A6) y el segundo para especificar el **dato inmediato** con que será cargado el acumulador. (Los dos bytes se presentan como "A6 ii"en la columna de código de máquina de la tabla).

La tabla de la parte inferior de la página correspondiente a LDA nos muestra que la versión de LDA en modo de direccionamiento inmediato toma dos períodos de la CPU para su ejecución. Durante el primer ciclo, la CPU vuelca el contenido del contador de programa sobre el bus interno de direcciones y lee el código de operación $A6, que identifica a la instrucción como la versión en modo de direccionamiento inmediato de la instrucción LDA y avanza a la PC hasta la próxima posición de memoria (la dirección del operando inmediato ii).

Durante el segundo ciclo, la CPU vuelca el contenido del PC sobre el bus interno de direcciones, lee el byte del dato inmediato y lo carga en el acumulador, y luego avanza la PC hasta la próxima posición de memoria (la dirección del código de operación de la próxima instrucción).

Mientras el acumulador está siendo cargado, los bits N y Z del registro de código de condición son modificados de acuerdo al dato almacenado. La fórmula de lógica booleana para estos bits está en la mitad de la página de la instrucción LDA. El bit Z irá a uno lógico si el valor cargado es $00; sino el bit Z irá a cero lógico. El bit N irá a uno lógico si el bit más significativo del valor almacenado es uno; sino el bit N irá a cero lógico.

El bit N (negativo) puede servir para detectar el signo de números signados según el convenio de **complemento a dos.** En este convenio el bit más significativo es usado como un bit de signo, un uno indica un valor negativo y un cero uno positivo. El bit N puede además usarse sólo como indicador del estado del bit de más peso de una magnitud binaria.

Bifurcación Condicional

Las instrucciones de bifurcación condicional permiten a la CPU elegir uno de dos posibles caminos a seguir por el programa, dependiendo del estado de un bit en particular de la memoria o bien de varios bits del CCR. Si la condición evaluada por la instrucción de bifurcación es **verdadera,** el programa efectúa la bifurcación a la posición de memoria especificada. Si la condición evaluada es **falsa,** la CPU continúa con la instrucción siguiente a la instrucción de bifurcación. Los bloques de decisión en un diagrama de flujo corresponden a instrucciones de bifurcación condicional en un programa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

La mayoría de las instrucciones de bifurcación contienen dos bytes, uno para el código de operación y otro para un byte de desplazamiento relativo. Las instrucciones bifurcar si el bit es cero (BRCLR) y bifurcar si el bit es uno (BRSET) requieren tres bytes: el código de operación, un byte de una dirección directa de un byte (para especificar la posición de memoria a evaluar) y el byte de desplazamiento relativo.

El byte correspondiente al desplazamiento relativo es interpretado por la CPU como un número signado en convenio de complemente a dos. Si la condición de bifurcación evaluada es verdad, este desplazamiento con signo se suma a la PC y la CPU lee su próxima instrucción desde la nueva dirección calculada. Si la condición de bifurcación evaluada es falsa, la CPU continúa con la instrucción siguiente a la de bifurcación.

Llamado a Subrutinas y Retornos

Las instrucciones salto **(JSR)** y bifurcación **(BSR)** a subrutina automatizan el proceso permitiendo salir del flujo lineal normal de un programa, ejecutar un grupo de instrucciones y luego retornar al punto desde donde se saliera del flujo normal del programa. El grupo de instrucciones externas al programa normal se denomina **subrutina.** Una instrucción JSR o BSR se usa para ir del programa en curso a una subrutina. Una instrucción retorno de subrutina **(RTS)** se usa para completar una subrutina, para retornar al programa desde el que fuera llamada la subrutina.

El listado 4-2 presenta líneas del listado de un programa de ejemplo generado por un ensamblador que usaremos para demostrar cómo la CPU ejecuta un llamado a subrutina. Asumimos que el puntero a la pila (SP) apunta a la dirección $00FF cuando la CPU encuentra la instrucción JSR en la posición de memoria $0302. Este listado se describe con más detalle en el capítulo 6.

```
0300  A6 02     TOP     LDA    #$02     ;Load an immediate value
0302  CD 04 00           JSR    SUBBY    ;Go do a subroutine
0305  B7 E0              STA    $E0      ;Store accumulator to RAM
0400  4A         SUBBY   DECA            ;Decrement accumulator
0401  26 FD              BNE    SUBBY    ;Loop till accumulator=0
0403  81                 RTS             ;Return to main program
```

Listado 4-2 – Ejemplo de llamado a Sub – Rutina.

Haremos referencia a la figura 4-3 durante la siguiente discusión. Partiremos nuestra explicación con la CPU ejecutando la instrucción “LDA #$02”en la dirección $0300. el lado izquierdo de la figura muestra el flujo normal del programa compuesto por TOP LDA #$ 02, JSR SUBBY y STA $EO (en ese orden) ubicados en posiciones de memoria consecutivas. En el lado derecho nos presenta las instrucciones de la subrutina SUBBY DECA, BNE SUBBY y RTS.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

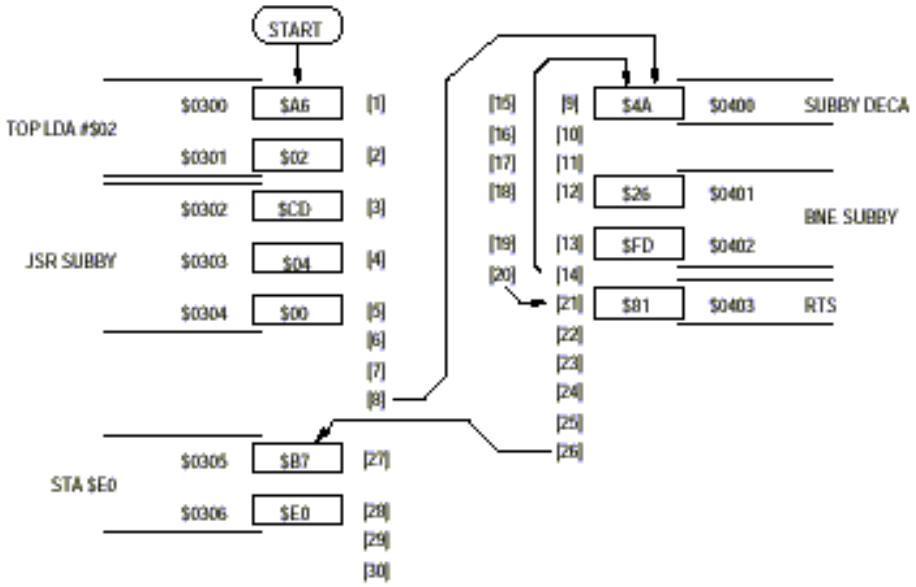


Fig. 4-3 – Secuencia de llamado de una Sub-rutina.

El número de ciclos de reloj de la CPU (entre corchetes) se usará como referencia en la siguiente explicación de esta figura.

[1] La CPU lee el código de operación $A6 de la posición de memoria $0300 (LDA inmediato).

[2] La CPU lee el dato inmediato $02 de la posición de memoria $0301 y lo carga en el acumulador.

[3] La CPU lee el código de operación $CD de la posición de memoria $0302 (JSR extendido)

[4] La CPU lee el byte más significativo de la dirección extendida $04 de la posición de memoria $0303.

[5] La CPU lee el byte menos significativo de la dirección extendida $00 de la posición de memoria $0304.
[6] La CPU arma la dirección completa de la subrutina ($0400).

[7] La CPU escribe $05 en la posición de memoria $00FF y decrementa el SP a $00FE. En otras palabras diremos que “hemos ingresado en la pila el byte menos significativo de la dirección de retorno”.

[8] La CPU escribe $03 en la posición de memoria $00FE y decrementa el SP a $00FD. En otras palabras diremos que “hemos ingresado en la pila el byte más significativo de la dirección de retorno”. La dirección de retorno salvada en la pila es $0305, que es la de la instrucción siguiente al JSR.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

[9] La CPU lee el código de operación $4A de la posición de memoria $0400 (DECA). Esta es la primera instrucción de la subrutina invocada.

[10] La CPU usa su unidad aritmético lógica **(ALU)** para restar uno al contenido del acumulador.

[11] El resultado de la ALU (A – 1) se escribe nuevamente en el acumulador.

[12] La CPU lee el código de operación $26 de la posición de memoria $0401 (BNE relativo).

[13] La CPU lee el byte de **desplazamiento relativo** $FD de la posición de memoria $0402.

[14] Con la instrucción LDA #$02 en [1], el acumulador se cargó con el valor 2; con la instrucción DECA en [9], el acumulador se decrementó a 1 (que es distinto de cero). De este modo, en [14], la condición de bifurcación fue verdad y el complemento a dos del desplazamiento ($FD o –3) se suma al valor contenido por la PC ($0403 en este momento) para obtener el valor $0400.

[15] a [19] Son la repetición de los ciclos [9] a [13] excepto que al ejecutar la instrucción DECA en [15] en esta pasada, el acumulador era de $01 a $00.

[20] Como ahora el acumulador es “igual a cero”la condición de bifurcación de BNE [19] es falsa y no se efectúa la bifurcación.

[21] La CPU lee el código de operación $81 de la posición de memoria $0403 (RTS).

[22] La CPU incrementa el SP a $00FE.

[23] La CPU lee $03 de la posición de memoria $00FE. En otras palabras diremos que “hemos retirado de la pila el byte más significativo de la dirección de retorno”.

[24] La CPU incrementa el SP a $00FF.

[25] La CPU lee $05 de la posición de memoria $00FF. En otras palabras diremos que “hemos retirado de la pila el byte menos significativo de la dirección de retorno”.

[26] la CPU arma la dirección de retorno completa recuperada de la pila ($0305) y con ella carga la PC.

[27] La CPU lee el código de operación $B7 de la posición de memoria $0305 (STA directo).

[28] la CPU lee el byte menos significativo de la dirección directa $E0 de la posición de memoria $0306.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

[29] y [30] La instrucción STA directo toma en total cuatro ciclos. En los últimos dos ciclos de la instrucción, la CPU arma la dirección completa en la que se almacenará el acumulador uniendo $00 (valor asumido para la mitad más significativa de la dirección para el modo de direccionamiento directo) con $E0 leído en [28]. El contenido del acumulador ($00 en este momento) es almacenado entonces en la dirección reconstruida ($00E0).

Jugando a la Computadora

Jugando a la computadora es una técnica de adiestramiento en la que pretenderemos ser una computadora interpretando y ejecutando las instrucciones de un programa. Los programadores a menudo verifican mentalmente un programa jugando a este juego del mismo modo que leen una subrutina. Mientras lo jugamos no es necesario dividir las instrucciones en ciclos individuales del procesador. En cambio, una instrucción es tratada como una única operación completa más que como diversas etapas definidas.

Los siguientes párrafos demuestran el proceso del juego partiendo del ejercicio del llamado a subrutina de la figura 4-3. Una aproximación para analizar esa secuencia es mucho menos detallada que el análisis ciclo por ciclo realizado, pero debemos cumplir el mismo objetivo básico (por ejemplo presentar qué pasa cuando la CPU ejecuta la secuencia). Luego de estudiar el capítulo de programación, podremos intentar lo propio con largos programas.

Debemos iniciar el proceso preparando una planilla de trabajo similar a la presentada en la figura 4-4. Esta planilla incluye los mnemónicos del programa y el código de máquina en que es ensamblado. (Podremos alternativamente optar por el uso de un listado adjunto a la planilla). La planilla además incluye a los nombres de los registros de la CPU a lo largo de su parte superior. Hay un amplio espacio disponible para escribir los nuevos valores con que los registros cambian en el curso del programa.

En esta planilla, hay un área para seguir la evolución de la pila. Una vez habituados al trabajo de la pila, probablemente prescindamos de esta área, aunque la dejaremos por ahora ya que nos sea instructiva.

Cuando un valor es ingresado en la pila, debemos tachar cualquier valor anterior y escribir el nuevo valor exactamente abajo en una fila vertical. Podemos ahora actualizar (decrementar) el valor del SP. Tachamos cualquier valor previo y escribimos el nuevo valor debajo de la columna correspondiente al SP. Cuando un valor es retirado de la pila, debemos actualizar (incrementar) el valor del SP, tachar el viejo valor y escribir el nuevo valor debajo. Podemos entonces leer el valor desde la posición de memoria apuntada por el SP y ponerlo en el lugar de la CPU correspondiente. (Por ejemplo en la mitad de más peso o en la de menos peso de la PC).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

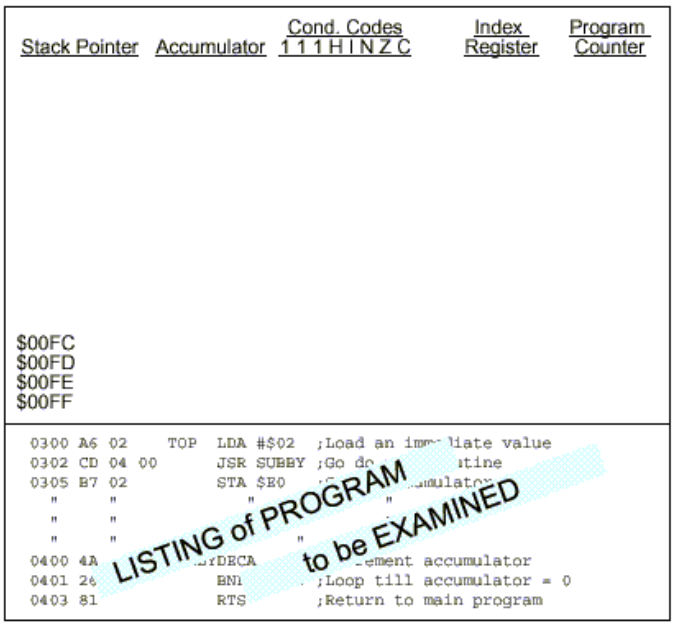


Fig. 4-4 – Hoja de trabajo para "Jugando a la Computadora".

La figura 4-5 presenta cómo luce la planilla luego de actuar sobre la secuencia completa del JSR. Se explica el proceso siguiendo los números entre corchetes [número]. Durante el proceso, se escriben muchos valores y luego se tachan; para marcar una referencia se dibuja una línea desde el número entre corchetes hasta el valor o la tachadura correspondiente.

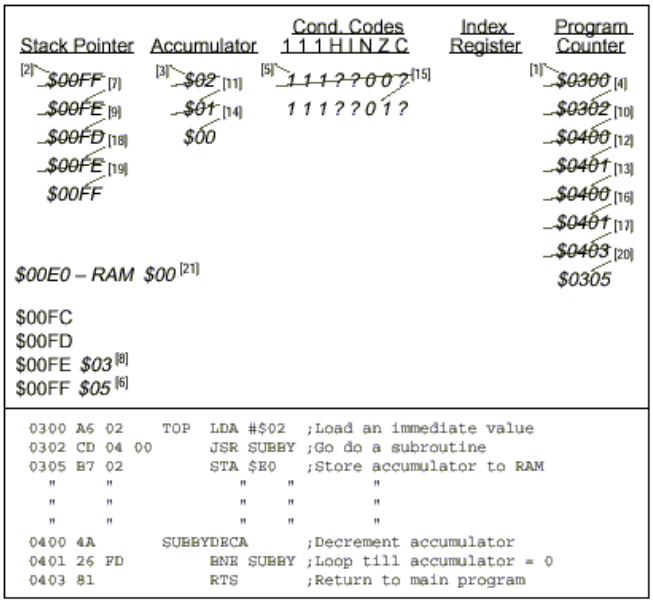


Fig. 4-5 – Hoja de trabajo Completa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Iniciamos la secuencia con la PC apuntando a $0300 [1] y el SP apuntando a $00FF [2] (tal como lo asumimos previamente). La CPU lee y ejecuta la instrucción LDA $02 (carga del acumulador con el valor inmediato $02) por ello, escribimos en la columna del acumulador $02 [3] y reemplazamos el valor de la PC por $0302 [4], que es la dirección de la próxima instrucción. La instrucción carga del acumulador afecta los bits N y Z del CCR. Ya que el valor cargado es $02, los bits N y Z se ponen ambos en cero [5]. Esta información podemos encontrarla en el apéndice A. Como hay otros bits del CCR que no son afectados por la instrucción LDA y además no tenemos forma de saber su estado previo los marcaremos con un signo de interrogación [5].

A continuación la CPU lee la instrucción JSR SUBBY. Retiene temporariamente el valor $0305, que es la dirección a la que la CPU deberá regresar luego de ejecutar el llamado a sub-rutina. La CPU ingresa la mitad de menos peso de la dirección de retorno en la pila; por lo que , escribimos $05 [6] en la posición de memoria apuntada por el SP ($00FF) y decrementamos el SP [7] a $00FE. Luego la CPU ingresa la mitad de más peso de la dirección de retorno en la pila; así, escribimos $03 [8] en $00FE y luego decrementamos el SP [9] (ahora a $00FD). Para finalizar la instrucción JSR, cargamos el PC con $0400 [10], que es la dirección de la sub-rutina invocada.

La CPU busca la próxima instrucción. Como el PC contiene $0400, la CPU ejecuta la instrucción DECA, que es la primera instrucción de la sub-rutina. Tachamos el $02 de la columna del acumulador y escribimos su nuevo valor $01 [11]. Además podemos cambiar el contenido del PC a $0401 [12]. Puesto que la instrucción DECA altera el contenido del acumulador de $02 a $01 (que no es ni cero ni negativo) los bits Z y N siguen en cero. Mientras N yA se mantengan en cero [5] podemos dejarlos tal cual en la planilla.

La CPU ahora ejecuta la instrucción BNE SUBBY. Mientras el bit Z permanezca en cero, se cumple la condición de bifurcación y la CPU la realiza. Tachamos $0401 en la columna del PC y escribimos $0400 [13].

La CPU ejecuta la instrucción DECA otra vez. El acumulador ahora cambia de $01 a $00 [14] (que es cero y no negativo); por ello, el bit de Z va a uno y el bit N sigue en cero.. el PC avanza hasta la próxima instrucción [16].

La CPU ahora ejecuta la instrucción BNE SUBBY, pero esta vez es falsa la condición de bifurcación (Z está en uno), en consecuencia no se realiza la bifurcación. La CPU sigue simplemente con la próxima instrucción (RTS en $0403). Actualizamos el PC con $0403 [17].

La instrucción RTS obliga a la CPU a retirar el PC previamente ingresado a la pila. Retiramos la mitad de más peso del PC de la pila incrementando el SP a $00FE [18] y leyendo $03 de la posición de memoria $00FE. A continuación, retiramos la mitad de menos peso de la dirección de la pila incrementando el SP a $00FF y leyendo $05 de $00FF. La dirección recuperada de la pila reemplaza al valor en el PC [20].

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

La CPU ahora lee la instrucción STA $EO de la posición de memoria $0305. el flujo del programa ha retornado a la secuencia principal de programa desde la que se ha llamado a la subrutina. La instrucción STA (en modo de direccionamiento directo) escribe el valor del acumulador en la dirección directa $00EO, de la RAM del MC68HC705J1A. Podemos ver en la planilla que el valor del acumulador es $00, en consecuencia todos los bits de esa posición de memoria en RAM irán a cero. Si bien en la planilla original no hay un lugar establecido para colocar el valor en RAM, podemos hacerlo y escribir $00 allí [21].

Para un extenso programa, la planilla tendrá muchos más valores tachados, tantos como sean necesarios. Jugando a la computadora sobre la planilla luce como una buena técnica de adiestramiento, pero a medida que el programador gana experiencia, va simplificando el proceso. En el capítulo de programación veremos una herramienta de desarrollo llamada simulador que automatiza el proceso del juego. El simulador es un programa de computadora que corre sobre una PC. El contenido actualizado de los registros y de las posiciones de memoria son presentados en la pantalla de la PC.

Una de las primeras simplificaciones que podemos hacer es eliminar el seguimiento del PC pues comprenderemos cómo lo maneja la CPU y lo tendremos en cuenta. Otra más es eliminar el seguimiento del CCR. Al encontrar una instrucción de bifurcación dependiente de algún bit del CCR elaboraremos mentalmente el estado de los mismos para decidir si realizamos o no la bifurcación.

Luego, podemos obviar el almacenamiento de valores en la pila, además del seguimiento del SP. Un principio fundamental de la operación de la pila es que luego de un período de tiempo, el mismo número de items han sido retirados e ingresados a la pila. Así como en una fórmula matemática ambos miembros de una igualación deben coincidir, los JSRs y BSRs uno a uno deben coincidir con los subsecuentes RTSs en un programa. Los errores de quebrantar esta regla aparecerán como valores erróneos del SP mientras jugamos a la computadora.

A veces un programador experimentado aplica el juego para resolver un problema de cierta dificultad. El procedimiento que un programador experimentado usa es mucho menos formal que el aquí explicado, pero él se coloca en el lugar de la CPU y predice qué sucede al ejecutar el programa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Resets

El reset es usado para forzar al sistema de MCU a ir a un punto de partida conocido (dirección). Los sistemas periféricos y muchos bits de control y estado son también forzados a un estado conocido como resultado del reset.

Las siguientes acciones internas ocurren como resultado de cualquier reset del MCU:

1) Todos los Registros de Dirección de Datos se colocan en cero (como entradas).
2) El puntero a la pila (SP) es forzado a $00FF.
3) El bit I del CCR se pone en uno inhibiendo a las interrupciones enmascarables.
4) El latch de interrupciones externas es borrado.
5) El latch de STOP es borrado.
6) El latch de WAIT es borrado.

Cuando el sistema de computadora sale de reset, el contador de programa se carga con el contenido de las posiciones de memoria más altas ($07FE y $07FF para el MC68HC705J1A). El valor que se encuentra en $07FE se carga en el byte más significativo del PC y el valor que se encuentra en $07FF se carga en el byte menos significativo del PC. Esto se denomina **"búsqueda del vector de reset".** En este punto la CPU comenzará la búsqueda y ejecución de instrucciones, comenzando por la dirección almacenada en el vector de reset.

Las siguientes condiciones pueden causar el reset del MC68HC705J1A:

1) Externamente, una señal de entrada activa baja en el pin / RESET.
2) Internamente, al encender la fuente de alimentación (power on reset POR).
3) Internamente, expiración de tiempo del cronómetro de vigilancia del comportamiento apropiado de la computadora (computer operating properly COP watchdog timed out).
4) Un intento de ejecutar una instrucción desde una dirección ilegal.

Pin de Reset

Una llave o un circuito externo puede conectarse a este pin para permitir el reset manual del sistema.

Reset al encender la Fuente de Alimentación (Power-On Reset)

El reset al encender la fuente de alimentación ocurre al detectarse una transición positiva sobre VDD. Su uso es estrictamente para la condición de encendido y no podrá utilizarse para detectar caídas de la tensión de la fuente de alimentación. Podrá usarse un circuito inhibidor de baja tensión (LVI) para detectar caídas de la fuente.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

El circuito de power-on provee una demora de 4064 ciclos desde el momento en que el oscilador se ha activado. Si el pin de /RESET exterior permanece en bajo al expirar el tiempo de los 4064 ciclos de demora, el procesador permanecerá en la condición de reset hasta que /RESET se coloque en alto.

Reset por Watchdog Timer

El sistema de cronómetro de vigilancia del comportamiento apropiado de la computadora (computing operating properly COP watchdog timer system) se propone detectar errores de programas. Cuando se activa el COP es responsabilidad del programa evitar que un cronómetro de vigilancia que corre libremente llegue al final de su cuenta. Si llega a completar su cuenta, sería una indicación de que el programa no ha sido ejecutado por un largo período de tiempo en la secuencia deseada; entonces se inicia el reset del sistema.

Un bit de control del registro (no volátil) máscara de opciones (MOR) puede usarse para habilitar o deshabilitar el reset del COP. Si el COP es habilitado, la adecuada operación del programa debe periódicamente escribir un cero en el bit COPC del registro de control COPR. Ir a la hoja de datos del MC69HC705J1A por información sobre el ritmo del tiempo de expiración. Algunos miembros de la familia de microcontroladores MC68HC05 tienen distintos sistemas de cronómetro de vigilancia del comportamiento apropiado de la computadora.

Reset por Acceso a Dirección Ilegal

Si el programa es escrito incorrectamente, es posible que la CPU intente saltar o bifurcar a una dirección en la que no haya memoria. Si esto sucede, la CPU podría continuar leyendo datos (resultando ser valores impredecibles) e intentaría actuar en consecuencia si se tratase de programa. Estas instrucciones sin sentido pueden provocar que la CPU escriba datos inesperados en memoria o registros direccionados inesperados. Esta situación se llama desbocamiento.

En el MC68HC705J1A hay un circuito detector de direcciones ilegales para protegernos de la condición de desbocamiento. Si la CPU trata de buscar una instrucción de una dirección que no pertenece a la EPROM ($0300 - $07CF, $07F0 - $07FF), ni a la ROM de prueba interna ($07EE - $07EF), se genera un reset que obliga al programa a comenzar nuevamente.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Interrupciones

Son a veces usadas para interrumpir el procesamiento normal para responder a algún evento inusual. El MC68HC705J1A puede ser interrumpido por las siguientes fuentes de interrupción:

1) Un cero lógico aplicado al pin de interrupción externa (/IRQ).
2) Un cero lógico aplicado a cualesquiera de los pines PA3 - PA0 (si la función de port de interrupción es habilitada).
3) Un pedido de desborde (overflow TOF) o interrupción de tiempo real (RTIF) desde el sistema de temporización por programa (SWI).
4) La instrucción de interrupción por programa (SWI).

Si una interrupción se produce mientras la CPU está ejecutando una instrucción, la instrucción será completada antes que la CPU responda al pedido de interrupción.

Las interrupciones pueden ser inhibidas en conjunto poniendo un uno en el bit I del CCR o bien individualmente, poniendo ceros en los bits de control de habilitación de cada fuente de interrupción. El reset fuerza el bit I a uno y a cero a todos los bits de habilitación de interrupciones locales a fin de prevenir interrupciones durante el proceso de inicialización. Cuando el bit I está en uno, ninguna interrupción (excepto SWI) es reconocida. Aunque pueda registrarse a la fuente de interrupción su pedido no será atendido hasta que el bit I se ponga en cero.

La figura 4-6 presenta cómo las interrupciones afectan el normal flujo de las instrucciones de la CPU. Las interrupciones provocan que los registros del procesador sean salvados en la pila y la máscara de interrupciones (el bit I) se ponga en uno, para prevenir interrupciones adicionales hasta la finalización de la presente interrupción. El vector de interrupción apropiado entonces apuntará a la dirección de inicio de la rutina de atención de interrupción (tabla 4-1). Completada la rutina de atención de interrupción, una instrucción RTI (que normalmente es la última instrucción de una rutina de atención de interrupción) provoca que el contenido de los registros sea recuperado de la pila. De esta forma el contador de programa es cargado con el valor previamente salvado en la pila, continuando con el procesamiento desde donde nos sacó la interrupción. En la figura 4-7 se presenta qué registros son recuperados de la pila en orden inverso al que fueron salvados.

Reset or Interrupt Source	Vector Address
On-Chip Timer	$07F8, $07F9
$\overline{\text{IRQ}}$ or Port A Pins	$07FA, $07FB
SWI Instruction	$07FC, $07FD
Reset (POR, LVI, Pin, COP, or Illegal Address)	$07FE, $07FF

Tabla 4-1 – Direcciones de Vectores para RESETs e Interrupciones (MC68HC705J1A).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Interrupciones Externas

Las interrupciones externas proceden del pin /IRQ o de los bits 3 - 0 del port A, si el port A se ha configurado como port de interrupciones. En el MC68HC705J1A, la sensibilidad del pin /IRQ es programable. Disponemos de disparo sólo sensible a un flanco o bien sensible a flanco descendente y nivel. El MC68HC705J1A usa un bit del registro máscara de opciones (MOR) en la posición de memoria $7F1 para configurar la sensibilidad del pin /IRQ. El pin /IRQ es activo bajo y las interrupciones del port A son activo alto.

Cuando una interrupción es reconocida, el estado actual de la CPU es salvado en la pila y el bit I es puesto en uno. Esto enmascara posteriores interrupciones hasta que la presente sea atendida. La dirección de rutina de atención de la interrupción externa es especificada por el contenido de las posiciones de memoria $07FA y $07FB.

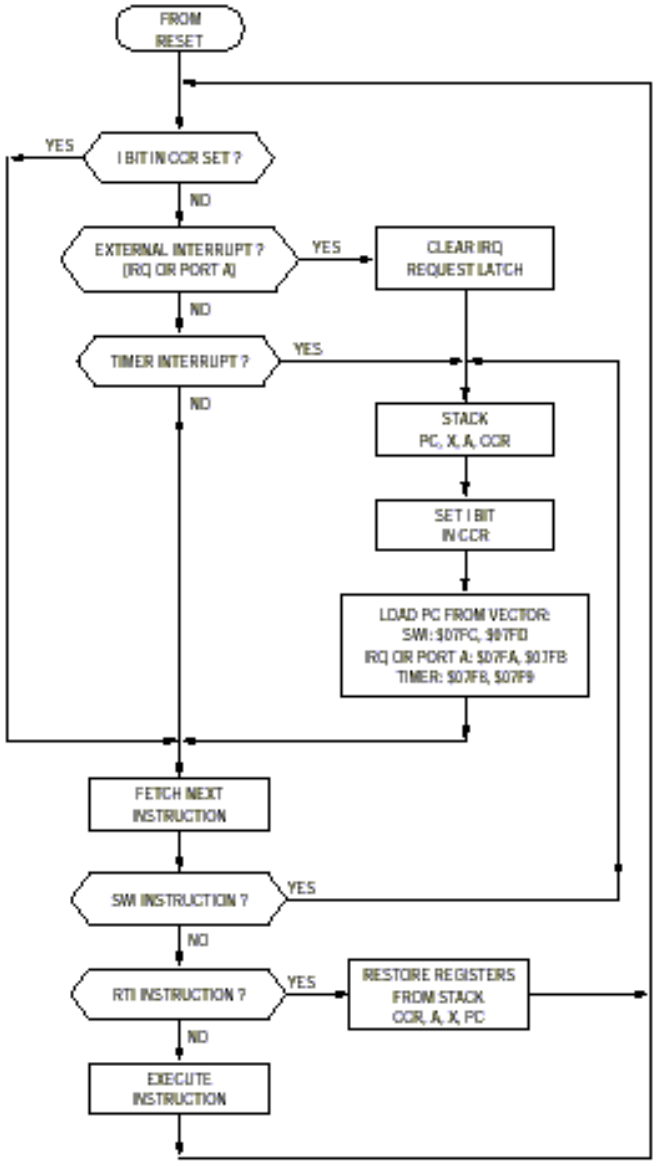


Fig. 4-6 – Diagrama de Flujo de la Interrupción por Hardware.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

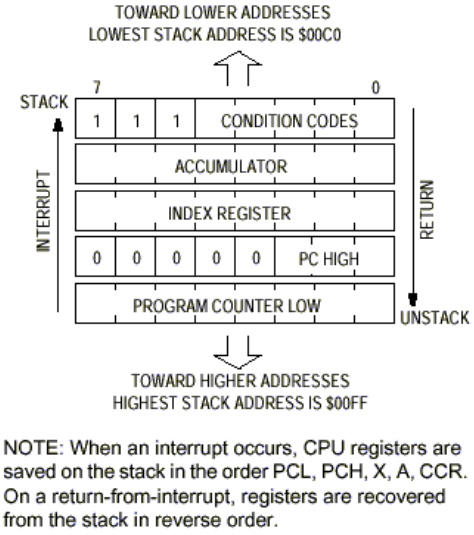


Fig. 4-7 – Orden de Apilamiento (stacking) de una Interrupción.

Interrupciones Internas

Los microcontroladores a veces incluyen sistemas periféricos dentro de su mismo chip que pueden generar interrupciones a la CPU. El sistema de temporización en el MC69HC705J1A es un ejemplo de semejante periférico. Las interrupciones internas trabajan del mismo modo que las externas excepto por que hay vectores de interrupción separados para cada sistema periférico incluido en el circuito integrado.

Interrupciones por Programa (SWI)

La interrupción por programa es una instrucción ejecutable. La acción de la instrucción SWI es similar a la de una interrupción (bit I) del CCR. La dirección de la rutina de atención es especificada por el contenido de las posiciones de memoria $07FC y $07FD (en el MC68HC705J1A).

Revisión del Capítulo 4

En la arquitectura del MC69HC705J1A hay cinco registros de la CPU que están directamente con la CPU y no forman parte del mapa de memoria. Toda otra información disponible por la CPU está dispuesta en una serie de posiciones de memoria de 8 bits.

Un **mapa de memoria** presenta el nombre y el tipo de memoria de todas las posiciones que son accesibles por la CPU. La expresión **I/O mapeado en memoria** dice que la CPU trata a los registros de I/O y control exactamente igual que a cualquier otro tipo de memoria. (En algunas arquitecturas de computadoras se separan los registros de I/O del espacio de memoria de programa y se usan instrucciones separadas para acceder a las posiciones de I/O).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Para poder comenzar de un lugar conocido, se debe hacer el **reset** de la computadora. El reset obliga a los sistemas periféricos incluidos en el chip y a la lógica de I/O a ir a condiciones conocidas y carga al contador de programa con una dirección de inicio conocida. El usuario especifica la posición de memoria de inicio deseada colocando los bytes de mayor y menor peso de esta dirección en las posiciones de memoria del **vector de reset** ($07FE y $07FF en el MC68HC705J1A).

La CPU utiliza al registro **puntero a la pila (SP)** para implementar una pila del tipo último en ingresa / primero en salir (LIFO) en la memoria **RAM.** Esta pila retiene el contenido previo de todos los registros mientras la CPU está ejecutando una secuencia de interrupción. Al recobrar esta información de la pila, la CPU puede proseguir en el punto en que se encontraba antes que la subrutina o la interrupción comenzaran.

Las computadoras usan un reloj de alta velocidad para pasar a través de las sub etapas de cada operación. Ya que cada instrucción toma varios ciclos de este reloj, resulta tan rápido que las operaciones parecen ser instantáneas para una persona. Un MC68HC705J1A puede ejecutar alrededor de 500000 instrucciones por segundo.

Una CPU ve a un programa como una secuencia lineal de números binarios de 8 bits. Los **códigos de operación** y datos de la instrucción están mezclados en esta secuencia pero la CPU se mantiene alineada con los límites de la instrucción ya que los códigos de operación le dicen a la CPU cuántos bytes de datos de **operando** vienen con el código de operación.

Jugando a la computadora es una técnica de adiestramiento con la que tratamos de ser la CPU que ejecuta el programa.

El reset puede producirse por condiciones internas o externas. Un pin de reset permite que una causa externa inicie un reset. Un cronómetro de vigilancia y un sistema detector de acceso a dirección legal pueden producir un reset en el supuesto caso que el programa no esté ejecutándose en la secuencia deseada.

Las interrupciones causan que la CPU detenga temporariamente la ejecución del programa principal para responder a la interrupción. Todos los registros de la CPU son salvados en la pila de modo que la CPU puede regresar al punto del programa principal una vez que la interrupción ha sido atendida.

Las interrupciones pueden ser inhibidas globalmente cargando el bit I del CCR con uno o bien localmente colocando cero en los bits de control de habilitación de cada fuente de interrupción.

Los pedidos podrán ser registrados mientras las interrupciones están inhibidas de modo que la CPU pueda atenderlos tan pronto como las interrupciones se rehabiliten. SWI es una instrucción y no puede ser inhibida.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 5.

El Repertorio de Instrucciones del MC68HC05

El repertorio de instrucciones de una computadora es su vocabulario. Este capítulo describe a la CPU y al repertorio de instrucciones del MC68HC05. El apéndice A contiene la detallada descripción de cada instrucción del MC68HC05 y puede usarse como referencia. En este capítulo se discuten las mismas instrucciones en grupos de operaciones de simular funcionamiento. También se discuten la estructura y los modos de direccionamiento del MC68HC05. Los modos de direccionamiento refieren las variadas maneras en que una CPU puede acceder a los operandos participantes de una instrucción.

Unidad Central de Proceso del MC68HC05 (CPU)

La CPU del MC68HC05 es la responsable de ejecutar todas las instrucciones en la secuencia programada para una aplicación específica. En la figura 5-1 se presenta el diagrama en bloques de una CPU típica del MC68HC05.

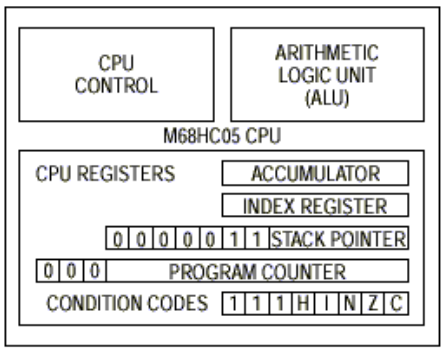


Fig. 5-1 – Diagrama en Bloques del CPU MC68HC05.

Unidad Aritmético / lógica (ALU)

La unidad aritmético / lógica **(ALU)** es usada para realizar las operaciones aritméticas y lógicas definidas por el repertorio de instrucciones.

Varios circuitos de operación aritmética binaria decodifican la instrucción en curso y preparan a la ALU para cumplir con la función deseada. La mayor parte de la aritmética binaria se basa en el algoritmo de suma, y la resta es manejada como una suma de números negativos. La multiplicación no se realiza como una instrucción directa, sino como una cadena de operaciones de suma y de desplazamiento en el interior de la ALU bajo el control de la lógica de control de la CPU. La instrucción multiplicación (MUL) requiere 11 ciclos internos del procesador para completar esta cadena de operaciones.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Lógica de Control de la CPU

El circuito de control de la CPU secuencia los elementos lógicos de la ALU para producir la operación requerida. Un elemento central de la sección de control de la CPU es el **decodificador de instrucciones.** Cada código de operación es decodificado para determinar cuántos operandos se necesitan y qué secuencia de etapas se requiere para completar una instrucción. Al finalizar una instrucción el próximo código de operación es leído y decodificado.

Registros de la CPU
(CPU05)

La CPU contiene cinco registros tal como se lo presenta en la figura 5-2. Los registros de la CPU son registros de memoria que se alojan dentro del microprocesador (no son parte del mapa de memoria). El conjunto de registros de la CPU es a veces denominado el **modelo de programación.** Un programador experimentado puede aventurarnos la suerte de una computadora a partir de su modelo de programación.

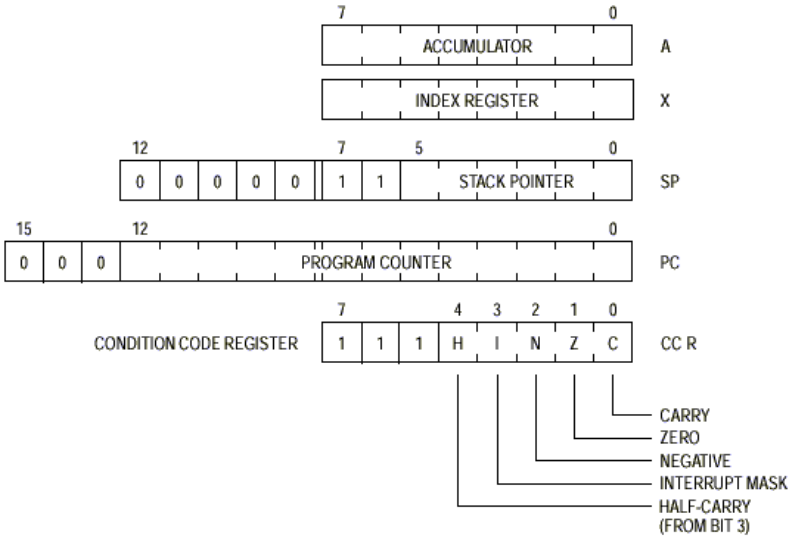


Fig. 5-2 – Modelo de Programación (Registros del CPU).

Acumulador (A). El acumulador es un registro de propósitos generales de 8 bits usado para almacenar operandos, resultados de cálculos aritméticos, y de manipulación de datos. Además él es directamente accesible a la CPU para operaciones no aritméticas. El acumulador es usado durante la ejecución de un programa donde el contenido de alguna posición de memoria es cargado en el acumulador. También, la instrucción almacenar causa que el contenido del acumulador sea almacenado en alguna posición de memoria preestablecida.

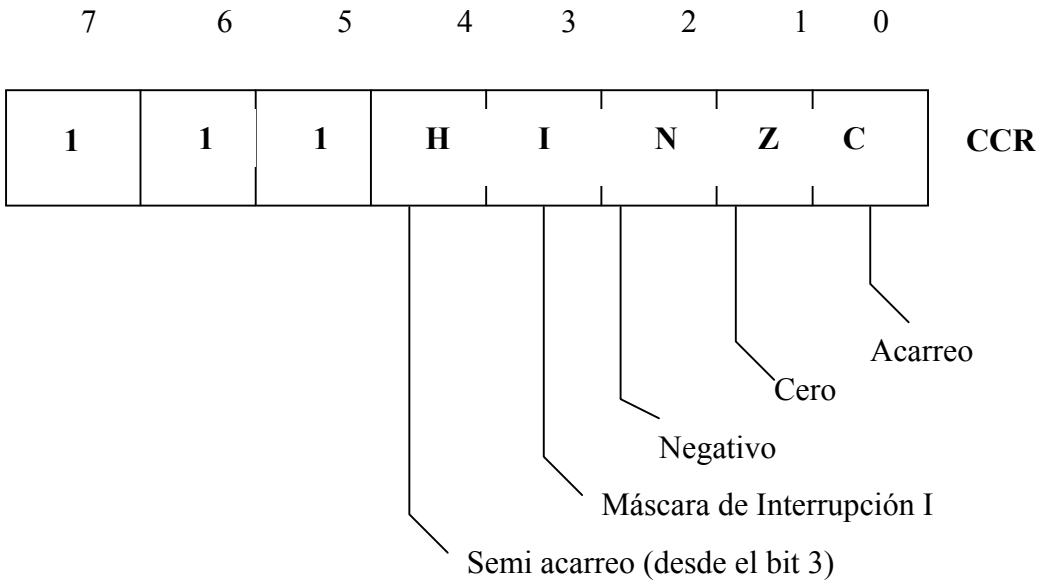
Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Registro Índice (X). El registro índice es usado para los modos de direccionamiento indexados o bien puede ser usado como un acumulador auxiliar. Este registro de 8 bits puede ser cargado directamente o sino desde memoria, siendo su contenido almacenado en memoria, o bien su contenido puede ser comparado con memoria.

En las instrucciones indexadas, el registro X provee un valor de 8 bits que es sumado a la dirección base provista por la instrucción para crear una **dirección afectiva.** El valor provisto por la instrucción puede ser de 0, 1 ó 2 bytes de largo.

Registro de Código de Condición (CCR). El registro de código de condición contiene una máscara de interrupción y cuatro indicadores de estado que reflejan el resultado de operaciones aritméticas y de otro tipo de la CPU. Las cinco banderas son semi acarreo (H), máscara de interrupción (I), negativo (N), cero (Z) y acarreo / préstamo (C).

Registro de Código de Condición



Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Bit de Semiacarreo (H). La bandera de semiacarreo es usada por las operaciones aritméticas en decimal codificado en binario (BCD) y es afectada por las instrucciones de suma ADD o ADC. El bit H se pone en uno cuando se produce un acarreo del dígito hexadecimal de menos peso en los bits 3-0 al dígito de más peso en bits 7-4. Luego de la suma binaria de dos valores de 2 dígitos BCD, este bit de semiacarreo es parte de la información necesaria para volver el resultado a un valor BCD válido.

Bit de Máscara de Interrupción (I). El bit I no es una bandera de estado, es un bit de máscara de interrupción que desabilita todas las fuentes de interrupción enmascarables cuando el bit I está en uno. Las interrupciones están habilitadas cuando este bit está en cero. Cuando cualquier interrupción ocurre, el bit I es automáticamente forzado a uno luego de haber salvado los registros en la pila, pero antes el vector de interrupción es buscado.

Si una interrupción interna ocurre mientras el bit I está en uno, la interrupción es almacenada y procesada luego que el bit I se ponga en cero; de esta manera, no perderemos una interrupción IRQ que se presente cuando el bit está en uno.

Luego de haber atendido a una interrupción, la instrucción retorno desde una interrupción (RTI) provocará que los registros recuperen sus valores previos. Normalmente, el bit I permanecería en cero luego que fuese ejecutada la instrucción RTI. Luego de cualquier reset, el bit I estará en uno y sólo podrá llevárselo a cero por medio de una instrucción.

Bit de Negativo (N). el bit N es forzado a uno cuando el resultado de la última operación aritmética, lógica o de manipulación de datos es negativo. Para los valores signados según el convenio de complemento a dos, se considera que un número es negativo si el bit más significatico es un uno.

El bit N tiene otros usos. Mediante la asignación de un bit de bandera de evaluación frecuente al MSB de un registro o posición de memoria, podemos evaluar este bit simplemente cargando el acumulador con el contenido de esta posición de memoria.

El bit de Cero (Z). El bit Z es forzado a uno cuando el resultado de la última operación aritmética, lógica o de manipulación de datos es cero. Una operación de comparación resta un valor desde la posición de memoria que está siendo evaluada. Si los valores son iguales antes de la comparación, el bit Z se pondrá en uno.

Bit de Acarreo/Préstamo (C). El bi C es usado para indicar si ha habido o no acarreo de una suma o pedido de préstamo como resultado de una resta. Las instrucciones de desplazamiento y rotación operan sobre y a través del bit C para facilitar operaciones de desplazamiento de múltiples bytes. El bit C es además afectado durante las instrucciones de evaluación de bit y de bifurcación.
La figura 5-3 es un ejemplo del modo en que los bits de código de condición son afectados por operaciones aritméticas.
El bit H carece de sentido luego de realizar la operación ya que el acumulador no contenía un valor BCD válido antes de la operación.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

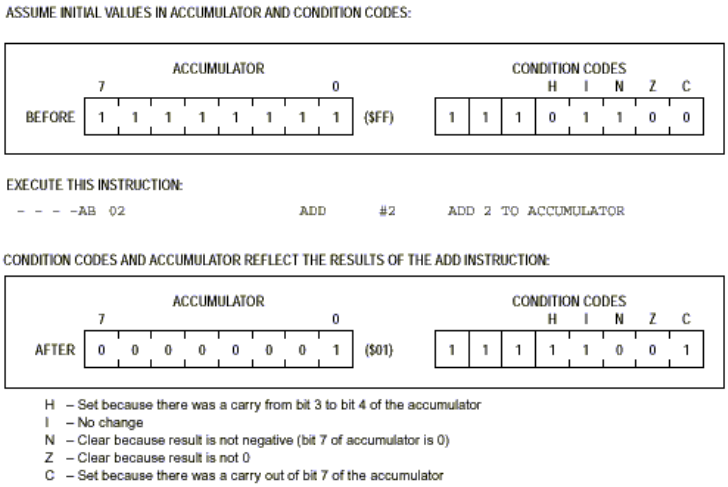
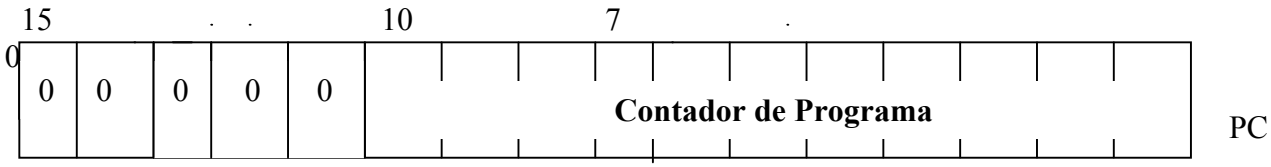


Fig. 5-3 – Como el CCR es afectado por las operaciones Aritméticas.

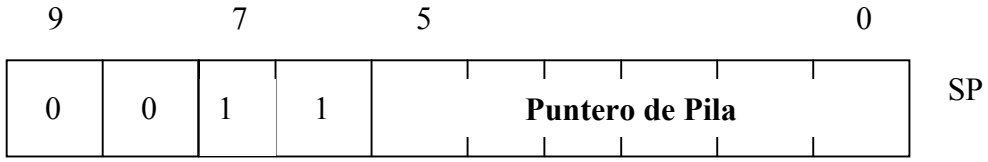
Contador de Programa (PC). El contador de programa es un registro de 16 bits que contiene la dirección de la próxima instrucción u operando de instrucción a ser buscado por el procesador. En muchas de las variantes del MC68HC05, algunos de los bits superiores del contador de programa no son usados y están siempre en cero. El MC68HC705J1A usa sólo 11 bits del contador de programa de modo que los cinco bits superiores están siempre en cero. El número de bits utilizados en el contador de programa coincide con el número de líneas de dirección implementadas en el sistema de computadora.



Normalmente, el contador de programa avanza una posición de memoria a la vez cuando se buscan instrucciones y operandos de instrucciones. Las operaciones de salto, bifurcación e interrupción provocan que el contador de programa sea cargado con la dirección de una posición de memoria diferente que la de la posición de memoria de la próxima secuencia.
Durante el Reset, el contador de programa (PC) se carga con la dirección contenida en el "Reset Vector" que para el MC68HC705J1A se encuentra en la posición $07FE y $07FF.
La dirección contenida en el vector, es la dirección de la primera instrucción a ser ejecutada después de salir del estado de RESET.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Puntero de Pila (SP). El puntero de pila puede tener tantos bits como líneas de dirección, en el MC68HC705J1A se ha pensado al SP como un registro de 10 bits. Durante un reset del MCU o por instrucción reset del puntero a la pila (RSP), el puntero de pila es cargado con la dirección $000FF. El puntero de pila decrementado cuando un dato es almacenado **(push)** dentro de la pila e incrementado cuando un dato es recuperado **(pull/pop)** desde la pila.



Algunas variantes del MC68HC05 permiten a la pila usar más de 64 posiciones (de $00FF a $00C0), pero las versiones más pequeñas permiten sólo 32 bytes de pila (de $00FF a $00E0). En el MC68HC705J1A, los cuatro MSBs del puntero de pila están permanentemente puestos en 0011b. Estos cuatro bits junto con los seis bits menos significativos producen una dirección en el rango que va de $00FF a $00C0. Las subrutinas y las interrupciones pueden utilizar hasta 64 posiciones de memoria.

Si se excede las 64 posiciones, el puntero de pila retorna a $00FF y comienza a escribir sobre la información previamente almacenada. Un llamado a subrutina usa dos posiciones de la pila y una interrupción usa cinco posiciones.

Registros del CPU08

El CPU08 (CPU de la familia HC908) es una versión mejorada del CPU05 de la familia HC705, en el, podemos encontrar los mismos códigos de instrucciones que en el CPU05 más el agregado de nuevas instrucciones y modos de direccionamiento. Por ello, podemos decir que el CPU08 es código compatible (100%) con los del CPU05.
En el CPU08 podemos hallar la misma cantidad de registros que el CPU05, pero con las siguientes diferencias:

Acumulador (Acc, A):

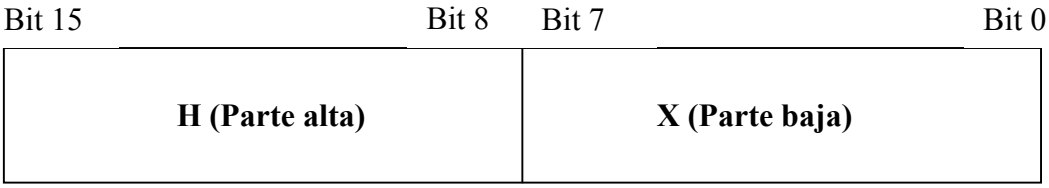
Es idéntico al del CPU05, 8 bits de largo de palabra.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Registro Índice (H:X):

Aquí el registro índice, a diferencia del CPU05, es de 16 bits de longitud, formado por una parte "baja" (el byte de menor peso) denominado "X" y una parte alta (el byte de mayor peso) denominado "H". Estos registros se encuentran concatenados para formar un único registro H:X. Esto permite direccionamientos indexados de **hasta 64 Kbytes** de espacio de memoria.

Para conservar la compatibilidad con la familia HC705, en el registro Índice puede utilizarse la parte baja ("X"), en los distintos modos de direccionamiento, de igual forma que en esta. Solo se debe tener en cuenta que cuando en una instrucción con direccionamiento indexado, se menciona el registro "X", en realidad se está haciendo mención al registro concatenado H:X de 16 bits de largo, por lo que deberá ponerse a cero
(forzar el valor $00) la parte superior del registro índice, o sea "H", para guardar total compatibilidad con la familia HC705. De esta forma, cuando se utilice el registro índice, el contenido del mismo será $00xx, donde "xx" contendrá el valor del registro "X" propiamente dicho.



Registro Índice (H:X)

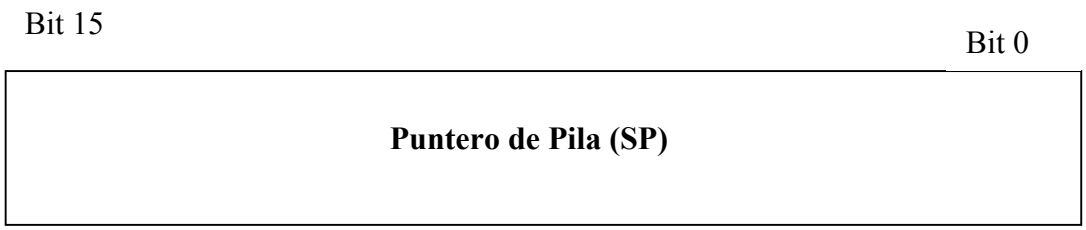
Puntero de Pila (SP) :

El puntero de pila (SP), es un registro de 16 bits que contiene la dirección del próximo Lugar en la pila (Stack). Durante un Reset, el puntero de pila, es preseteado a $00FF. La instrucción Reset Stack Pointer (RSP), setea al byte menos significativo a $FF y no afecta al byte más significativo. Esto se hace para mantener la compatibilidad con el modo de funcionamiento del puntero de pila de la familia HC705.
El puntero de pila es decrementado cuando un dato es almacenado **(Push)** dentro de la pila e incrementado cuando un dato es recuperado **(Pull)** desde la pila.

La localización de la pila es arbitraria, y puede ser "re-ubicada" en cualquier parte de la memoria RAM. Moviendo el puntero fuera de la página 0 ($0000 a $00FF) libera el espacio del direccionamiento directo. Para una operación correcta, el puntero de pila debe apuntar solamente posiciones de RAM, aunque por su longitud, pueda "barrer" todo el espacio de memoria del MCU.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Gracias a esta característica, en los modos de direccionamiento con el SP (Stack Pointer) con 8 bits de offset y 16 bits de offset, el puntero de pila (SP) puede funcionar como un segundo registro índice de 16 bits o bien para acceder a datos en la pila.
El uso del SP como un segundo registro índice, es muy utilizado en los compiladores de lenguaje de alto nivel como los compiladores "C" y otros.



Contador de Programa (PC):

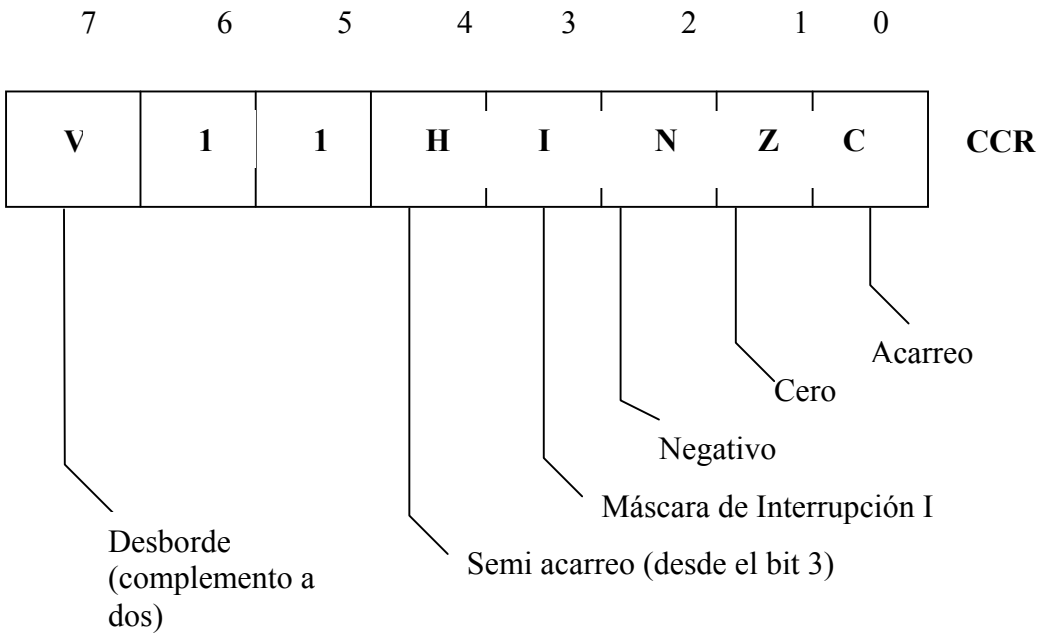
Al igual que en el CPU05, el Contador de Programa (PC), es de 16 bits de longitud, pero a diferencia de este, el PC del CPU08, no tiene bits fijos en algunas de sus posiciones, o sea el PC puede moverse entre $0000 y $FFFF. De esta forma, el PC puede moverse teóricamente (muchos MCUs de la familia HC908, poseen memorias de programas inferiores a los 64Kbytes) por los 64 Kbytes de espacio de memoria.

Durante el Reset, el contador de programa (PC) se carga con la dirección contenida en el "Reset Vector" que para el MC68HC908 se encuentra en la posición $FFFE y $FFFF. La dirección contenida en el vector, es la dirección de la primera instrucción a ser ejecutada después de salir del estado de RESET.

Registro de Código de Condición (CCR). El registro de código de condición contiene una máscara de interrupción y cinco indicadores de estado que reflejan el resultado de operaciones aritméticas y de otro tipo de la CPU. En esencia, es idéntico al del CPU05, pero con el agregado de una nueva bandera, la de “desborde” (Overflow) o V flag.
Las cinco banderas son, Desborde (V), semi acarreo (H), máscara de interrupción (I), negativo (N), cero (Z) y acarreo / préstamo (C).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Registro de Código de Condición



V – Bit de Desborde :

El CPU pone en 1 el bit de desborde cuando ocurre un desborde por complemento a dos.
Las instrucciones de salto condicionales signados como BGT, BGE, BLE, y BLT usan este bit de desborde.

Modos de Direccionamiento

El poder de cualquier computadora radica en la habilidad para acceder a memoria. Los modos de direccionamiento de la CPU proveen esta capacidad. Los **modos de direccionamiento** difieren la manera en que una instrucción obtendrá el dato requerido para su ejecución. Debido a los diferentes modos de direccionamiento, una instrucción puede acceder al operando en una de las diversas maneras. Cada variante de diferente modo de direccionamiento de una instrucción debe tener un único código de operación de instrucción, de tal modo que las 62 instrucciones básicas de la CPU del MC68HC05 requieren 210 códigos de operación de instrucción distintos.

La CPU del MC68HC05 usa seis modos de direccionamiento para hacer referencia a memoria. Los seis modos de direccionamiento son **inherente, inmediato, extendido, directo, indexado (sin desplazamiento, con desplazamiento de 8 bits, o de 16 bits) y relativo.** En los pequeños microcontroladores MC68HC05, todas las variables del programa y los registros de I/O caben en el área de memoria que va de $0000 a $00FF donde el modo de direccionamiento más comúnmente usado es el direccionamiento directo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

En los siguientes párrafos se provee una descripción general y ejemplos de los variados modos de direccionamiento. El término **dirección efectiva** es usado para indicar la dirección de la posición de memoria donde el argumento para una instrucción es buscado o almacenado. En el apéndice A está disponible la descripción de cada instrucción.

La información provista en los listados de los programas de ejemplo, usan diversos símbolos para identificar variados tipos de números que se presentan en un programa. El capítulo 2 incluye la descripción de los sistemas de numeración y códigos de computadora.

1. El espacio en blanco o ningún símbolo indica que el número es decimal. Este número será trasladado a un valor binario antes de ser almacenado en memoria para ser usado por la CPU.
2. El símbolo $ precediendo a un número indica que el número es hexadecimal; por ejemplo $24 es 24(16) en hexadecimal o el equivalente de 36(10).
3. El símbolo # indica un operando y el número es buscado en la posición de memoria siguiente a la del código de operación. Una variedad de símbolos y expresiones pueden ser usadas siguiendo al carácter #. Ya que no todos los ensambladores usan las mismas reglas de sintaxis ni los mismos caracteres especiales, es necesario referirse a la documentación del ensamblador en particular que usemos.

Prefijo	**Indica que el valor siguiente es...**
Nada	Decimal
$	Hexadecimal
@	Octal
%	Binario
' (apóstrofe)	Un simple carácter ASCII

Para cada modo de direccionamiento es explicada en detalle una instrucción. Esta explicación describe qué sucede en la CPU durante cada ciclo de reloj del procesador de la instrucción. En estos ejemplos, los números de secuencia entre corchetes [nro.] hacen referencia a un ciclo de reloj de la CPU específico.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Modo de Direccionamiento Inmediato

En el modo de direccionamiento **inmediato,** el operando está contenido en el byte inmediato siguiente al código de operación. Este modo es usado cuando un valor o constante conocido al momento de escribir el programa que no cambiará durante la ejecución del programa. Esta es una instrucción de dos bytes, uno para el código de operación y otro para el byte de dato inmediato.

Listado del Programa de Ejemplo:

0300 A6 02 **LDA** **#$02** ; Cargar el Acumulador con el valor inmediato

Secuencia de Ejecución:

$0300 $A6 [1]
$0301 $02 [2]

Explicación:

[1] La CPU lee el código de operación $A6 - Carga del acumulador con el valor inmediato siguiente al código de operación.

[2] La CPU lee el dato inmediato $02 de la posición de memoria $0301 y lo carga en el acumulador.

La tabla 5-1 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento inmediato.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Load Accumulator from Memory	LIDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Subtract	SUB

Tabla 5-1 – Instrucciones con Modo de Direccionamiento Inmediato.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Modo de Direccionamiento Inherente

En el modo de direccionamiento **inherente,** toda la información requerida para la operación ya es implícitamente conocida por la CPU y no es necesario recuperar un operando exterior desde la memoria. Los operandos (si los hay) son sólo los registros de la CPU o bien valores de datos almacenados en la pila. Esta es una instrucción de un solo byte.

Listado del Programa de Ejemplo:

```
0300   4C        INCA                ; Incrementar el Acumulador
```

Secuencia de Ejecución:

$0300 $4C [1], [2] y [3]

Explicación:

[1] La CPU lee el código de operación $A6 - Incremento del acumulador.

[2] La CPU le suma uno al valor actual del acumulador.

[3] La CPU almacena el nuevo valor en el acumulador y ajusta las banderas del registro de código de condición de ser necesario.

La tabla 5-2 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento inherente.

Instruction	Mnemonic
Arithmetic Shift Left	ASLA,ASLX
Arithmetic Shift Right	ASRA,ASRX
Clear Carry Bit	CLC
Clear Interrupt Mask Bit	CLI
Clear	CLRA,CLRX
Complement	COMA, COMX
Decrement	DECA,DECX
Increment	INCA, INCX
Logical Shift Left	LSLA,LSLX
Logical Shift Right	LSRA, LSRX
Multiply	MUL
Negate	NEGA,NEGX
No Operation	NOP
Rotate Left thru Carry	ROLA, ROLX
Rotate Right thru Carry	RORA, RORX
Reset Stack Pointer	RSP
Return from Interrupt	RTI
Return from Subroutine	RTS
Set Carry Bit	SEC
Set Interrupt Mask Bit	SEI
Enable IRQ, Stop Oscillator	STOP
Software Interrupt	SWI
Transfer Accumulator to Index Register	TAX
Test for Negative or Zero	TSTA,TSTX
Transfer Index Register to Accumulator	TXA
Enable Interrupt, Halt Processor	WAIT

Tabla 5-2 - Instrucciones con Modo de Direccionamiento Inherente.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Modo de Direccionamiento Extendido

En el modo de direccionamiento **extendido,** la dirección del operando está contenida en los dos bytes siguientes al código de operación. Este modo es usado para hacer referencia a cualquier posición de memoria dentro del espacio de memoria del MCU, incluyendo I/O, RAM, ROM, EPROM, FLASH. Esta es una instrucción de tres bytes, uno para el código de operación y otros dos para la dirección del operando.

Listado del Programa de Ejemplo:

```
0300    C6  03  65      LDA     $0365   ; Cargar el acumulador desde
                                        ; una dirección extendida
```

Secuencia de Ejecución:

```
$0300   $C6     [1]
$0301   $03     [2]
$0302   $65     [3] y [4]
```

Explicación:

[1] La CPU lee el código de operación $C6 - Carga del acumulador usando el modo de direccionamiento extendido.

[2] La CPU lee $03 de la posición de memoria $0301. Este $03 es interpretado como la mitad de mayor peso de una dirección.

[3] La CPU lee $65 de la posición de memoria $0302. Este $65 es interpretado como la mitad de menor peso de una dirección.

[4] La CPU arma la dirección extendida completa $0365 con los dos valores previamente leídos. Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en la posición de memoria $0365 y lo carga en el acumulador.

La tabla 5-3 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento extendido.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB

Tabla 5-3 – Instrucciones con Modo de Direccionamiento Extendido

Modo de Direccionamiento Directo

El modo de direccionamiento **directo,** es similar al extendido excepto que la parte alta de la dirección del operando se asume de valor $00. De tal manera, que sólo es necesario incluir el byte de menos peso de la dirección del operando en la instrucción. Esta área se denomina **página directa** e incluye a los registros de RAM e I/O del interior del chip. Este modo es eficiente tanto en economía de espacio de memoria de programa como en menor tiempo de ejecución. Esta es una instrucción de dos bytes, uno para el código de operación y otro para el byte de menor peso de la dirección del operando.

Listado del Programa de Ejemplo:

```
0300  B6  E0      LDA    $E0     ; Cargar el acumulador desde una dirección de página
                                  ; directa
```

Secuencia de Ejecución:

```
$0300    $B6    [1]
$0301    $E0    [2] y [3]
```

Explicación:

[1] la CPU lee el código de operación $B6 - Carga del acumulador usando el modo de direccionamiento directo.

[2] La CPU lee $E0 de la posición de memoria $0301. Este $E0 es interpretado como la mitad de menor peso de una dirección de página directa (desde $0000 hasta $00FF).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

[3] La CPU arma la dirección directa completa $00E0 asumiendo el valor del byte de mayor peso en $00, con el valor previamente leído del byte de menor peso. Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en la posición de memoria $00E0 y lo carga en el acumulador.

La tabla 5-4 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento directo.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Clear Bit in Memory	BCLR
Bit Test Memory with Accumulator	BIT
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Set Bit in Memory	BSET
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

Tabla 5-4 – Instrucciones con Modo de Direccionamiento Directo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Modo de Direccionamiento Indexado

En el modo de direccionamiento **indexado,** la dirección efectiva del operando es variable y depende de dos factores: 1) el contenido actual del registro índice (X) y 2) el desplazamiento contendido en el / los byte/s siguiente/s al código de operación. La CPU del MC68HC05 soporta tres tipos de direccionamientos indexados: sin desplazamiento, con desplazamiento de 8 bits y con desplazamiento de 16 bits. Un buen ensamblador usará el modo de direccionamiento indexado que requiera el menor número de bytes para expresar el desplazamiento.

Indexado sin desplazamiento. En el modo de direccionamiento indexado sin desplazamiento, la dirección efectiva del operando para la instrucción está contenida en los 8 bits del registro índice. Así, este modo de direccionamiento puede acceder a las primeras 256 posiciones de memoria (desde $0000 hasta $00FF). Esta es una instrucción de un solo byte.

Listado del Programa de Ejemplo:

```
0300    F6      LDA     0,X     ; Cargar el acumulador desde la dirección apuntada
                                ; por X
```

Secuencia de Ejecución:

$0300 $F6 [1], [2] y [3]

Explicación:

[1] La CPU lee el código de operación $F6 - Carga del acumulador usando el modo de direccionamiento indexado sin desplazamiento.

[2] la CPU arma la dirección completa sumando $0000 al contenido del registro índice de 8 bits (X).

[3] Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

La tabla 5-5 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado sin desplazamiento y con desplazamiento de 8 bits.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Arithmetic Shift Left	ASL
Arithmetic Shift Right	ASR
Bit Test Memory with Accumulator	BIT
Clear	CLR
Compare Accumulator with Memory	CMP
Complement	COM
Compare Index Register with Memory	CPX
Decrement	DEC
Exclusive OR Memory with Accumulator	EOR
Increment	INC
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Logical Shift Left	LSL
Logical Shift Right	LSR
Negate	NEG
Inclusive OR	ORA
Rotate Left thru Carry	ROL
Rotate Right thru Carry	ROR
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register in Memory	STX
Subtract	SUB
Test for Negative or Zero	TST

Tabla 5-5 – Instrucciones con Modo de Direccionamiento Indexado (No offset o 8 Bits Offset)

Indexado con desplazamiento de 8 bits. En el modo de direccionamiento indexado con desplazamiento de 8 bits, la dirección efectiva es la suma del contenido del registro índice de 8 bits y el byte de desplazamiento siguiente al código de operación. El byte de desplazamiento suministrado en la instrucción es un número entero no signado de 8 bits. Esta es una instrucción de dos bytes, uno para el código de operación y otro para el byte de desplazamiento. El contenido del registro índice no es alterado.

Listado del Programa de Ejemplo:

```
0300    E6    05      LDA    5,X      ; Cargar el acumulador desde el 6to. ítem de la
                                      ; tabla apuntada por X
```

Secuencia de Ejecución:

```
$0300    $E6    [1]
$0301    $05    [2], [3] y [4]
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Explicación:

[1] La CPU lee el código de operación $E6 - Carga del acumulador usando el modo de direccionamiento indexado con desplazamiento de 8 bits.

[2] La CPU lee $05 de la posición de memoria $0301. Este $05 es interpretado como un desplazamiento de 8 bits.

[3] La CPU arma la dirección completa sumando el valor antes leído ($05) al contenido del registro índice de 8 bits (X).

[4] Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

La tabla 5-5 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado sin desplazamiento y con desplazamiento de 8 bits.

Indexado con desplazamiento de 16 bits. En el modo de direccionamiento indexado con desplazamiento de 16 bits, la dirección efectiva es la suma del contenido del registro índice de 8 bits y los dos bytes de desplazamiento siguientes al código de operación. El byte de desplazamiento suministrado en la instrucción es un número entero no signado de 16 bits. Esta es una instrucción de tres bytes, uno para el código de operación y otros dos para los bytes de desplazamiento. El contenido del registro índice no es alterado.

Listado del Programa de Ejemplo:

```
0300  D6  03  77     LDA     $377,X     ; Cargar el acumulador desde el X+1er.
                                        ;  ítem de la tabla $0377
```

Secuencia de Ejecución:

```
$0300    $D6    [1]
$0301    $03    [2]
$0302    $77    [3], [4] y [5]
```

Explicación:

[1] La CPU lee el código de operación $D6 - Carga del acumulador usando el modo de direccionamiento indexado con desplazamiento de 16 bits.

[2] La CPU lee $03 de la posición de memoria $0301. Este $03 es interpretado como la mitad de mayor peso de una dirección base.

[3] La CPU lee $77 de la posición de memoria $0302. Este $77 es interpretado como la mitad de menor peso de una dirección base.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

[4] La CPU arma la dirección completa sumando la dirección base de 16 bits antes leída ($0377) al contenido del registro índice de 8 bits (X).

[5] Esta dirección es colocada en el bus de direcciones y la CPU lee el valor del dato contenido en esa posición de memoria y lo carga en el acumulador.

La tabla 5-6 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento indexado desplazamiento de 16 bits.

Instruction	Mnemonic
Add with Carry	ADC
Add	ADD
Logical AND	AND
Bit Test Memory with Accumulator	BIT
Compare Accumulator with Memory	CMP
Compare Index Register with Memory	CPX
Exclusive OR Memory with Accumulator	EOR
Jump	JMP
Jump to Subroutine	JSR
Load Accumulator from Memory	LDA
Load Index Register from Memory	LDX
Inclusive OR	ORA
Subtract with Carry	SBC
Store Accumulator in Memory	STA
Store Index Register In Memory	STX
Subtract	SUB

Tabla 5-6 – Instrucciones con Modo de Direccionamiento Indexado (16 bits de Offset)

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Modo de Direccionamiento Relativo

El modo de direccionamiento **relativo** es usado solamente por las instrucciones de bifurcación (saltos condicionados). Las instrucciones de bifurcación, salvo las bifurcaciones en su versión de manipulación de bits generan dos bytes de código de máquina: uno para el código de operación y otro para el desplazamiento relativo. Ya que es deseable bifurcar en cualquier sentido, el byte de desplazamiento es un número signado según el convenio de complemento a dos con un rango que va desde -128 hasta +127 bytes (respecto a la dirección de la instrucción inmediata posterior a la instrucción de bifurcación). Si la condición de salto es verdad, el contenido de los 8 bits del byte con signo siguiente al código de operación (desplazamiento) es sumado al contenido del contador de programa para formar la dirección de bifurcación efectiva; de otro modo, el control continúa bajo la instrucción inmediata posterior a la instrucción de bifurcación.

Un programador especifica el destino de una bifurcación como una dirección absoluta (o rótulo que hace referencia a una dirección absoluta). **El ensamblador calcula el desplazamiento relativo de 8 bits con signo**, que es colocado en memoria luego del código de operación de la bifurcación.

Listado del Programa de Ejemplo:

```
0300    27    rr        BEQ        DEST        ; Bifurcar a DEST si Z = 1 (si es igual o
cero)
```

Secuencia de Ejecución:

```
$0300     $27     [1]
$0301     $       [2] y [3]
```

Explicación:

[1] La CPU lee el código de operación $27 - Bifurcar si Z = 1. El bit Z del registro de código de condición será uno si el resultad de la operación aritmética o lógica previa fue cero.

[2] La CPU lee $rr de la posición de memoria $0301. Este $rr es interpretado como el valor de desplazamiento relativo. Después de este ciclo el contador de programa apunta al primer byte de la próxima instrucción ($0302).

[3] Si el bit Z está en cero, nada sucede en este ciclo y el programa debe continuar con la próxima instrucción. Si el bit Z está en uno, la CPU armará la dirección completa sumando el desplazamiento signado antes leído ($rr) al contenido del registro contador de programa para obtener la dirección destino de la bifurcación. Esto provoca que la ejecución del programa continúe desde una nueva dirección (DEST).

La tabla 5-7 incluye una lista de todas las instrucciones del MC68HC05 que pueden usar el modo de direccionamiento relativo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Instruction	Mnemonic
Branch if Carry Clear	BCC
Branch is Carry Set	BCS
Branch if Equal	BEQ
Branch if Half-Carry Clear	BHCC
Branch if Half-Carry Set	BHCS
Branch if Higher	BHI
Branch if Higher or Same	BHS
Branch if Interrupt Line is High	BIH
Branch if Interrupt Line is Low	BIL
Branch if Lower	BLO
Branch if Lower or Same	BLS
Branch if Interrupt Mask is Clear	BMC
Branch if Minus	BMI
Branch if Interrupt Mask Bit is Set	BMS
Branch if Not Equal	BNE
Branch if Plus	BPL
Branch Always	BRA
Branch if Bit n is Clear	BRCLR
Branch if Bit n is Set	BRSET
Branch Never	BRN
Branch to Subroutine	BSR

Tabla 5-7 – Instrucciones con Modo de Direccionamiento Relativo

Instrucciones de Evaluación de bits y de Bifurcación

Estas instrucciones usan modo de direccionamiento directo para especificar la posición de memoria a evaluar y direccionamiento relativo para especificar la dirección destino de la bifurcación. Este texto trata a estas instrucciones como de direccionamiento directo. En documentación antigua de Motorola se denomina al modo de direccionamiento de estas instrucciones BTB como "bit test and branch".

Instrucciones Ordenadas por su Tipo

Las tablas 5-8 a 5-11 incluyen las listas de todo el repertorio de las instrucciones del MC68HC05 agrupadas por su tipo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

		Addressing Modes																	
		Immediate			Direct			Extended			Indexed (No Offset)			Indexed (8-Bit Offset)			Indexed (16-Bit Offset)		
Function	Mnem.	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles
Load A from Memory	LDA	A6	2	2	B6	2	3	C6	3	4	F6	1	3	E6	2	4	D6	3	5
Load X from Memory	LDX	AE	2	2	BE	2	3	CE	3	4	FE	1	3	EE	2	4	DE	3	5
Store A in Memory	STA	—	—	—	B7	2	4	C7	3	5	F7	1	4	E7	2	5	D7	3	6
Store X in Memory	STX	—	—	—	BF	2	4	CF	3	5	FF	1	4	EF	2	5	DF	3	6
Add Memory to A	ADD	AB		2	BB	2	3	CB	3	4	FB	1	3	EB	2	4	DB	3	5
Add Memory and Carry to A	ADC	A9	2	2	B9	2	3	C9	3	4	F9	1	3	E9	2	4	D9	3	5
Subtract Memory	SUB	A0	2	2	B0	2	3	C0	3	4	F0	1	3	E0	2	4	D0	3	5
Subtract Memory from A with Borrow	SBC	A2	2	2	B2	2	3	C2	3	4	F2	1	3	E2	2	4	D2	3	5
AND Memory to A	AND	A4	2	2	B4	2	3	C4	3	4	F4	1	3	E4	2	4	D4	3	5
OR Memory with A	ORA	AA	2	2	BA	2	3	CA	3	4	FA	1	3	EA	2	4	DA	3	5
Exclusive OR Memory with A	EOR	A8	2	2	B8	2	3	C8	3	4	F8	1	3	E8	2	4	D8	3	5
Arithmetic Compare A with Memory	CMP	A1	2	2	E11	2	3	C1	3	4	F1	1	3	E1	2	4	D1	3	5
Arithmetic Compare X with Memory	CPX	A3	2	2	B3	2	3	C3	3	4	F3	1	3	E3	2	4	D3	3	5
Bit Test Memory with A (Logical Compare)	BIT	A5	2	2	B5	2	3	C5	3	4	F5	1	3	E	2	4	D5	3	5
Jump Unconditional	JMP	—	—	—	BC	2	2	CC	3	3	FC	1	2	EC	2	3	DC	3	4
Jump to Subroutine	JSR	—	—	—	BD	2	5	CD	3	6	FD	1	5	ED	2	6	DD	3	7

Tabla 5-8 – Instrucciones de Registros / Memoria.

		Addressing Modes														
		Inherent (A)			Inherent (X)			Direct			Indexed (No Offset)			Indexed (8-Bit Offset)		
Function	Mnem.	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles	Op-code	# Bytes	# Cycles
Increment	INC	4C	1	3	5C	1	3	3C	2	5	7C	1	5	6C	2	6
Decrement	DEC	4A	1	3	5A	1	3	3A	2	5	7A	1	5	6A	2	6
Clear	CLR	4F	1	3	5F	1	3	3F	2	5	7F	1	5	6F	2	6
Complement	COM	43	1	3	53	1	3	33	2	5	73	1	5	63	2	6
Negate Twos Complement	NEG	40	1	3	50	1	3	30	2	5	70	1	5	60	2	6
Rotate Left Thru Carry	ROL	49	1	3	59	1	3	39	2	5	79	1	5	69	2	6
Rotate Right Thru Carry	ROR	46	1	3	56	1	3	36	2	5	76	1	5	66	2	6
Logical Shift Left	LSL	48	1	3	58	1	3	38	2	5	78	1	5	68	2	6
Logical Shift Right	LSR	44	1	3	54	1	3	34	2	5	74	1	5	64	2	6
Arithmetic Shift Right	ASH	47	1	3	57	1	3	37	2	5	77	1	5	67	2	
Test for Negative or Zero	TST	4D	1	3	5D	1	3	3D	2	4	7D	1	4	6D	2	5
Multiply	MUL	42	1	11	—	—	—	—	—	—	—	—	—	—	—	—
Bit Clear	BCLR	—	—	—	—	—	—	See Note	2	5	—	—	—	—	—	—
Bit Set	BSET	—	—	—	—	—	—	See Note	2	5	—	—	—	—	—	—

NOTE: Unlike other read-modify-write instructions, BCLR and BSET use only direct addressing.

Tabla 5-9 – Instrucciones Lectura / Modificación / Escritura.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Function	Mnemonic	Relative Addressing Mode		
		Opcode	# Bytes	# Cycles
Branch Always	BRA	20	2	3
Branch Never	BRN	21	2	3
Branch if Higher	BH1	22	2	3
Branch if Lower or Same	BLS	23	2	3
Branch if Carry Clear	BCC	24	2	3
Branch if Higher or Same (Same as BCC)	BHS	24	2	3
Branch if Carry Set	BCS	25	2	3
Branch if Lower (Same as BCS)	BLO	25	2	3
Branch if Not Equal	BNE	26	2	3
Branch if Equal	BEQ	27	2	3
Branch if Half-Carry Clear	BHCC	28	2	3
Branch if Half-Carry Set	BHCS	29	2	3
Branch if Plus	BPL	2A	2	3
Branch if Minus	BMI	2B	2	3
Branch if Interrupt Mask Bit is Clear	BMC	2C	2	3
Branch if Interrupt Mask Bit is Set	BMS	2D	2	3
Branch if Interrupt Line is Low	BIL	2E	2	3
Branch if Interrupt Line is High	BIH	2F	2	3
Branch to Subroutine	BSR	AD	2	6

Tabla 5-10 – Instrucciones de Salto.

Function	Mnemonic	Relative Addressing Mode		
		Opcode	# Bytes	# Cycles
Transfer A to X	TAX	97	1	2
Transfer X to A	TXA	9F	1	2
Set Carry Bit	SEC	99	1	2
Clear Carry Bit	CLC	98	1	2
Set Interrupt Mask Bit	SEI	9B	1	2
Clear Interrupt Mask Bit	CLI	9A	1	2
Software Interrupt	SWI	83	1	10
Return from Subroutine	RTS	81	1	6
Return from Interrupt	RTI	80	1	9
Reset Stack Pointer	RSP	9C	1	2
No-Operation	NOP	9D	1	2
Stop	STOP	8E	1	2
Wait	WAIT	8F	1	2

Tabla 5-11 – Instrucciones de Control

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Sumario del Repertorio de Instrucciones

Las computadoras usan códigos de operación u opcodes para darle instrucciones a la CPU. El repertorio de instrucciones para una CPU específica es el conjunto de todas las operaciones que la CPU sabe cómo realizar.

La CPU en un MC68HC705J1A puede reconocer 62 instrucciones básicas, algunas con diversas variaciones que requieren códigos de operación separados. Del repertorio de instrucciones del MC68HC05 están representados sólo 210 códigos de operación de instrucción.

La tabla 5-12 incluye una lista de todas las instrucciones del MC68HC05 en orden alfabético.

Símbolos de Código de Condición

H ..... Semi acarreo (bit 4)
I..... Máscara de Interrupción (bit 3)
N..... Negativo (bit 2)
Z..... Cero (bit 1)
C..... Acarreo / préstamo (bit 0)

0..... En Bajo
1..... En Alto
↑ .... Evalúa y Levanta el bit si es verdad, (sino Baja el bit)
- ..... No Afectado

Símbolos de Expresiones Booleanas

. ..... AND Lógica
; ..... OR Lógica
⊕ .... OR Exclusiva

- ..... Not (inversión)
_ ..... Negativo o Resta
+ ..... Suma Aritmética
x ..... Multiplicación
← ..... es Cargado con ....
( ) ..... el Contenido de ....

A..... Acumulador
X ..... Registro Índice
M ..... Posición de Memoria

CCR.. Registro de Código de Condición
PC ..... Contador de Programa
PCL ... PC (byte de menor peso)
PCH ... PC (byte de mayor peso)
SP ..... Puntero a la Pila
REL .... Desplazamiento Relativo

Modo de Direccionamiento	**Abreviatura**	**Operandos**
Inherente	INH	ninguno
Inmediato	IMM	ii
Directo	DIR	dd
(para evaluación de bits)		dd rr
Extendido	EXT	hh ll
Indexado (sin desplazamiento)	IX	ninguno
Indexado (con desplazamiento de 8 bits)	IX1	ff
Indexado (con desplazamiento de 16 bits)	IX2	ee ff
Relativo	REL	rr

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

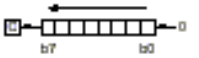
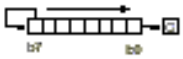
Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
ADC #opr ADC opr ADC opr ADC opr,X ADC opr,X ADC ,X	Add with Carry	A ← (A) + (M) + (C)	↕	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	A9 B9 C9 D9 E9 F9	ii dd hh ll ee ff ff	2 3 4 5 4 3
ADD #opr ADD opr ADD opr ADD opr,X ADD opr,X ADD ,X	Add without Carry	A ← (A) + (M)	↕	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	AB BB CB DB EB FB	ii dd hh ll ee ff ff	2 3 4 5 4 3
AND #opr AND opr AND opr AND opr,X AND opr,X AND ,X	Logical AND	A ← (A) ∧ (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	A4 B4 C4 D4 E4 F4	ii dd hh ll ee ff ff	2 3 4 5 4 3
ASL opr ASLA ASLX ASL opr,X ASL ,X	Arithmetic Shift Left (Same as LSL)		—	—	↕	↕	↕	DIR INH INH IX1 IX	38 48 58 68 78	dd ff 	5 3 3 6 5
ASR opr ASRA ASRX ASR opr,X ASR ,X	Arithmetic Shift Right		—	—	↕	↕	↕	DIR INH INH IX1 IX	37 47 57 67 77	dd ff 	5 3 3 6 5
BCC rel	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? C = 0	—	—	—	—	—	REL	24	rr	3
BCLR n,opr	Clear Bit n	Mn ← 0	—	—	—	—	—	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	5 5 5 5 5 5 5 5
BCS rel	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + rel ? C = 1	—	—	—	—	—	REL	25	rr	3
BEQ rel	Branch if Equal	PC ← (PC) + 2 + rel ? Z = 1	—	—	—	—	—	REL	27	rr	3
BHCC rel	Branch if Half-Carry Bit Clear	PC ← (PC) + 2 + rel ? H = 0	—	—	—	—	—	REL	28	rr	3
BHCS rel	Branch if Half-Carry Bit Set	PC ← (PC) + 2 + rel ? H = 1	—	—	—	—	—	REL	29	rr	3
BHI rel	Branch if Higher	PC ← (PC) + 2 + rel ? C ∨ Z = 0	—	—	—	—	—	REL	22	rr	3
BHS rel	Branch if Higher or Same	PC ← (PC) + 2 + rel ? C = 0	—	—	—	—	—	REL	24	rr	3
BIH rel	Branch if IRQ Pin High	PC ← (PC) + 2 + rel ? IRQ = 1	—	—	—	—	—	REL	2F	rr	3
BIL rel	Branch if IRQ Pin Low	PC ← (PC) + 2 + rel ? IRQ = 0	—	—	—	—	—	REL	2E	rr	3

Tabla 5-12 – Sumario de Instrucciones (1- de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
BIT #opr BIT opr BIT opr BIT opr,X BIT opr,X BIT ,X	Bit Test Accumulator with Memory Byte	(A) ∧ (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	A5 B5 C5 D5 E5 F5	ii dd hh ll ee ff ff	2 3 4 5 4 3
BLO rel	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + rel ? C = 1	—	—	—	—	—	REL	25	rr	3
BLS rel	Branch if Lower or Same	PC ← (PC) + 2 + rel ? C ∨ Z = 1	—	—	—	—	—	REL	23	rr	3
BMC rel	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + rel ? I = 0	—	—	—	—	—	REL	2C	rr	3
BMI rel	Branch if Minus	PC ← (PC) + 2 + rel ? N = 1	—	—	—	—	—	REL	2B	rr	3
BMS rel	Branch if Interrupt Mask Set	PC ← (PC) + 2 + rel ? I = 1	—	—	—	—	—	REL	2D	rr	3
BNE rel	Branch if Not Equal	PC ← (PC) + 2 + rel ? Z = 0	—	—	—	—	—	REL	26	rr	3
BPL rel	Branch if Plus	PC ← (PC) + 2 + rel ? N = 0	—	—	—	—	—	REL	2A	rr	3
BRA rel	Branch Always	PC ← (PC) + 2 + rel ? 1 = 1	—	—	—	—	—	REL	20	rr	3
BRCLR n opr rel	Branch if Bit n Clear	PC ← (PC) + 2 + rel ? Mn = 0	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN rel	Branch Never	PC ← (PC) + 2 + rel ? 1 = 0	—	—	—	—	—	REL	21	rr	3
BRSET n opr rel	Branch if Bit n Set	PC ← (PC) + 2 + rel ? Mn = 1	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET n opr	Set Bit n	Mn ← 1	—	—	—	—	—	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	5 5 5 5 5 5 5 5
BSR rel	Branch to Subroutine	PC ← (PC) + 2; push (PCL) SP ← (SP) – 1; push (PCH) SP ← (SP) – 1 PC ← (PC) + rel	—	—	—	—	—	REL	AD	rr	6
CLC	Clear Carry Bit	C ← 0	—	—	—	—	0	INH	98		2
CLI	Clear Interrupt Mask	I ← 0	—	0	—	—	—	INH	9A		2

Tabla 5-12 – Sumario de Instrucciones (2 - de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
CLR *opr* CLRA CLRX CLR *opr*,X CLR ,X	Clear Byte	M ← $00 A ← $00 X ← $00 M ← $00 M ← $00	—	—	0	1	—	DIR INH INH IX1 IX	3F 4F 5F 6F 7F	dd ff 	5 3 3 6 5
CMP #*opr* CMP *opr* CMP *opr* CMP *opr*,X CMP *opr*,X CMP ,X	Compare Accumulator with Memory Byte	(A) – (M)	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	A1 B1 C1 D1 E1 F1	ii dd hh ll ee ff ff 	2 3 4 5 4 3
COM *opr* COMA COMX COM *opr*,X COM ,X	Complement Byte (One's Complement)	M ← (M) = $FF – (M) A ← (A) = $FF – (A) X ← (X) = $FF – (X) M ← (M) = $FF – (M) M ← (M) = $FF – (M)	—	—	↕	↕	1	DIR INH INH IX1 IX	33 43 53 63 73	dd ff 	5 3 3 6 5
CPX #*opr* CPX *opr* CPX *opr* CPX *opr*,X CPX *opr*,X CPX ,X	Compare Index Register with Memory Byte	(X) – (M)	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	A3 B3 C3 D3 E3 F3	ii dd hh ll ee ff ff 	2 3 4 5 4 3
DEC *opr* DECA DECX DEC *opr*,X DEC ,X	Decrement Byte	M ← (M) – 1 A ← (A) – 1 X ← (X) – 1 M ← (M) – 1 M ← (M) – 1	—	—	↕	↕	—	DIR INH INH IX1 IX	3A 4A 5A 6A 7A	dd ff 	5 3 3 6 5
EOR #*opr* EOR *opr* EOR *opr* EOR *opr*,X EOR *opr*,X EOR ,X	EXCLUSIVE OR Accumulator with Memory Byte	A ← (A) ⊕ (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	A8 B8 C8 D8 E8 F8	ii dd hh ll ee ff ff 	2 3 4 5 4 3
INC *opr* INCA INCX INC *opr*,X INC ,X	Increment Byte	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1	—	—	↕	↕	—	DIR INH INH IX1 IX	3C 4C 5C 6C 7C	dd ff 	5 3 3 6 5
JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Unconditional Jump	PC ← Jump Address	—	—	—	—	—	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff 	2 3 4 3 2

Tabla 5-12 – Sumario de Instrucciones (3 - de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

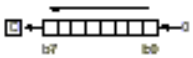
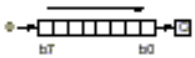
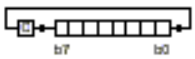
Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
JSR *opr* JSR *opr* JSR *opr,X* JSR *opr,X* JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) − 1 Push (PCH); SP ← (SP) − 1 PC ← Effective Address	—	—	—	—	—	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	5 6 7 6 5
LDA #*opr* LDA *opr* LDA *opr* LDA *opr,X* LDA *opr,X* LDA ,X	Load Accumulator with Memory Byte	A ← (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	A6 B6 C6 D6 E6 F6	ii dd hh ll ee ff ff	2 3 4 5 4 3
LDX #*opr* LDX *opr* LDX *opr* LDX *opr,X* LDX *opr,X* LDX ,X	Load Index Register with Memory Byte	X ← (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	AE BE CE DE EE FE	ii dd hh ll ee ff ff	2 3 4 5 4 3
LSL *opr* LSLA LSLX LSL *opr,X* LSL ,X	Logical Shift Left (Same as ASL)		—	—	↕	↕	↕	DIR INH INH IX1 IX	38 48 58 68 78	dd ff 	5 3 3 6 5
LSR *opr* LSRA LSRX LSR *opr,X* LSR ,X	Logical Shift Right		—	—	0	↕	↕	DIR INH INH IX1 IX	34 44 54 64 74	dd ff 	5 3 3 6 5
MUL	Unsigned Multiply	X : A ← (X) × (A)	0	—	—	—	0	INH	42		11
NEG *opr* NEGA NEGX NEG *opr,X* NEG ,X	Negate Byte (Two's Complement)	M ← −(M) = $00 − (M) A ← −(A) = $00 − (A) X ← −(X) = $00 − (X) M ← −(M) = $00 − (M) M ← −(M) = $00 − (M)	—	—	↕	↕	↕	DIR INH INH IX1 IX	30 40 50 60 70	dd ff 	5 3 3 6 5
NOP	No Operation		—	—	—	—	—	INH	9D		2
ORA #*opr* ORA *opr* ORA *opr* ORA *opr,X* ORA *opr,X* ORA ,X	Logical OR Accumulator with Memory	A ← (A) ∨ (M)	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX	AA BA CA DA EA FA	ii dd hh ll ee ff ff	2 3 4 5 4 3
ROL *opr* ROLA ROLX ROL *opr,X* ROL ,X	Rotate Byte Left through Carry Bit		—	—	↕	↕	↕	DIR INH INH IX1 IX	39 49 59 69 79	dd ff 	5 3 3 6 5

Tabla 5-12 – Sumario de Instrucciones (4 - de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
ROR *opr* RORA RORX ROR *opr*,X ROR ,X	Rotate Byte Right through Carry Bit		—	—	↕	↕	↕	DIR INH INH IX1 IX	36 46 56 66 76	dd ff 	5 3 3 6 5
RSP	Reset Stack Pointer	SP ← $00FF	—	—	—	—	—	INH	9C		2
RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	INH	80		9
RTS	Return from Subroutine	SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	—	—	—	—	—	INH	81		6
SBC #*opr* SBC *opr* SBC *opr* SBC *opr*,X SBC *opr*,X SBC ,X	Subtract Memory Byte and Carry Bit from Accumulator	A ← (A) – (M) – (C)	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	A2 B2 C2 D2 E2 F2	ii dd hh ll ee ff ff 	2 3 4 5 4 3
SEC	Set Carry Bit	C ← 1	—	—	—	—	1	INH	99		2
SEI	Set Interrupt Mask	I ← 1	—	1	—	—	—	INH	9B		2
STA *opr* STA *opr* STA *opr*,X STA *opr*,X STA ,X	Store Accumulator in Memory	M ← (A)	—	—	↕	↕	—	DIR EXT IX2 IX1 IX	B7 C7 D7 E7 F7	dd hh ll ee ff ff 	4 5 6 5 4
STOP	Stop Oscillator and Enable IRQ Pin		—	0	—	—	—	INH	8E		2
STX *opr* STX *opr* STX *opr*,X STX *opr*,X STX ,X	Store Index Register in Memory	M ← (X)	—	—	↕	↕	—	DIR EXT IX2 IX1 IX	BF CF DF EF FF	dd hh ll ee ff ff 	4 5 6 5 4
SUB #*opr* SUB *opr* SUB *opr* SUB *opr*,X SUB *opr*,X SUB ,X	Subtract Memory Byte from Accumulator	A ← (A) – (M)	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX	A0 B0 C0 D0 E0 F0	ii dd hh ll ee ff ff 	2 3 4 5 4 3
SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) – 1; Push (PCH) SP ← (SP) – 1; Push (X) SP ← (SP) – 1; Push (A) SP ← (SP) – 1; Push (CCR) SP ← (SP) – 1; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	—	1	—	—	—	INH	83		10
TAX	Transfer Accumulator to Index Register	X ← (A)	—	—	—	—	—	INH	97		2

Tabla 5-12 – Sumario de Instrucciones (5 - de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Source Form	Operation	Description	Effect on CCR					Address Mode	Opcode	Operand	Cycles
			H	I	N	Z	C				
TST *opr* TSTA TSTX TST *opr*,X TST ,X	Test Memory Byte for Negative or Zero	(M) – $00	—	—	↕	↕	—	DIR INH INH IX1 IX	3D 4D 5D 6D 7D	dd ff 	4 3 3 5 4
TXA	Transfer Index Register to Accumulator	A ← (X)	—	—	—	—	—	INH	9F		2
WAIT	Stop CPU Clock and Enable Interrupts		—	0	—	—	—	INH	8F		2

A	Accumulator	*opr*	Operand (one or two bytes)
C	Carry/borrow flag	PC	Program counter
CCR	Condition code register	PCH	Program counter high byte
dd	Direct address of operand	PCL	Program counter low byte
dd rr	Direct address of operand and relative offset of branch instruction	REL	Relative addressing mode
DIR	Direct addressing mode	*rel*	Relative program counter offset byte
ee ff	High and low bytes of offset in indexed, 16-bit offset addressing	rr	Relative program counter offset byte
EXT	Extended addressing mode	SP	Stack pointer
ff	Offset byte in indexed, 8-bit offset addressing	X	Index register
H	Half-carry flag	Z	Zero flag
hh ll	High and low bytes of operand address in extended addressing	#	Immediate value
I	Interrupt mask	•	Logical AND
ii	Immediate operand byte	+	Logical OR
IMM	Immediate addressing mode	⊕	Logical EXCLUSIVE OR
INH	Inherent addressing mode	( )	Contents of
IX	Indexed, no offset addressing mode	–( )	Negation (twos complement)
IX1	Indexed, 8-bit offset addressing mode	←	Loaded with
IX2	Indexed, 16-bit offset addressing mode	?	If
M	Memory location	:	Concatenated with
N	Negative flag	↕	Set or cleared
n	Any bit	—	Not affected

Tabla 5-12 – Sumario de Instrucciones (6 - de 6).

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Revisión del Capítulo 5

Registros de la CPU

Los cinco registros de la CPU del MC68HC05 no son posiciones de memoria pertenecientes al mapa de memoria. El **modelo de programación** de la CPU presenta los cinco registros de la CPU.

- El **acumulador (A)** es un registro de 8 bits de propósitos generales.
- El **registro índice (X)** es un registro puntero de 8 bits.
- El **puntero a la pila (SP)** es un registro puntero que es decrementado automáticamente al ingresar un dato a la pila y es incrementado al retirar un dato de la pila.
- El **contador de programa (PC)** tiene tantos bits como líneas el bus de direcciones. El contador de programa siempre apunta a la próxima instrucción o porción de dato que la CPU usará.
- El **registro de código de condición (CCR)** contiene cuatro banderas de resultados aritméticos H, N, Z y C y un bit de control I de la máscara de interrupción.

Modos de Direccionamiento

La CPU del MC68HC05 posee seis modos de direccionamiento que determinan cómo la CPU obtendrá el / los operando / s para completar cada instrucción. La CPU del MC68HC05 tiene sólo 62 **mnemónicos** de instrucción. Hay 210 códigos de operación debido a que por cada variación de modos de direccionamiento que una instrucción puede tener debe haber un único código de operación.

- En modo de direccionamiento **inmediato,** el operando para la instrucción es el byte inmediato siguiente al código de operación.
- En modo de direccionamiento **inherente,** la CPU no necesita más operandos de memoria. Los operandos, si los hay, son registros internos o datos almacenados en la pila.
- En modo de direccionamiento **extendido,** la dirección de 16 bits del operando se coloca en los dos bytes siguientes al código de operación de la instrucción.
- En modo de direccionamiento **directo,** los 8 bits de menor peso de la dirección del operando se colocan en el byte siguiente al código de operación de la instrucción y el byte de mayor peso de la dirección se asume que es $00. Este modo es más eficiente que el extendido ya que el byte de mayor orden está implícitamente incluido en el programa.
- En modo de direccionamiento **indexado,** el valor actual del registro índice es sumado a un desplazamiento de 0, 1 ó 2 bytes que vienen en las 0, 1 ó 2 posiciones de memoria siguientes al código de operación, para formar la dirección del operando en memoria.

- En modo de direccionamiento **relativo,** es sumado para instrucciones de bifurcación condicional. El byte siguiente al código de operación es un valor de desplazamiento de 8 bits con signo entre -128 y +127. Si la condición de bifurcación es verdad, el desplazamiento es sumado al valor del contador de programa para obtener la dirección donde la CPU buscará la próxima instrucción del programa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Ejecución de Instrucciones

Cada **código de operación** le dice a la CPU la operación a realizar y el modo de direccionamiento a usar para acceder a cualquier **operando** necesario para completar la instrucción. La explicación ciclo a ciclo de las instrucciones de ejemplo de cada modo de direccionamiento proporcionan una visión de las pequeñas simples etapas que lleva a cabo una instrucción.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 6

Programación.

En este capítulo se discute como planear y escribir un programa de computadora. Se enseñará como preparar un diagrama de flujo y como escribir un programa en lenguaje assembler. Un editor de texto o procesador de palabras se usa para escribir el programa. Luego, una herramienta de programación llamada "ensamblador" se usa para traducir el programa en una forma que la computadora pueda usar. Las herramientas de programación, son programas de PC (computadoras personales) que ayudan en el desarrollo de programas para microcontroladores. Se tratarán los ensambladores, simuladores, emuladores en tiempo real y un conjunto de herramientas muy útiles.

Escribiendo un simple programa.....

Escribiremos un pequeño programa en forma de mnemónico y traduciremos esto en forma de código de máquina. El primer paso será planear el programa y documentar este plan con un diagrama de flujo. Luego, se escribirán las instrucciones en mnemónicos para cada bloque en el diagrama de flujo. Finalmente se usará un Ensamblador para traducir el programa de ejemplo, en los códigos que la computadora (MCU) necesita para ejecutar el programa.

El programa en cuestión, leerá el estado de un switch (pulsador) conectado a un pin de entrada. Cuando el pulsador es accionado (cerrado), el programa causará que un LED conectado a un pin de salida se encienda por aproximadamente 1 segundo y luego salga de ese estado. El LED no encenderá nuevamente hasta que el pulsador no sea liberado y vuelto a pulsar. El tiempo que el pulsador se encuentre presionado, no afectará el tiempo de encendido del LED.

A pesar que el programa es muy sencillo, este demuestra los elementos más comunes en cualquier programa de aplicación de un MCU. Primero, este demuestra como un programa puede sensar entradas como el cierre de un pulsador. Segundo, este es un ejemplo de cómo un programa controla una señal de salida. Tercero, el encendido del LED por un tiempo de cerca de un segundo, demuestra una forma que puede ser usada para medir tiempos reales.
Además, se verá que con la ayuda de los MCUs se puede reemplazar el uso de circuitos discretos complejos.

En la figura 6-1, podemos ver un diagrama de flujo del programa de ejemplo. Los diagramas de flujo, se usan a menudo como una herramienta de planeación para escribir el software, porque ellos muestran las funciones y flujos del programa bajo desarrollo. La importancia de las notas, comentarios, y documentación no pueden ser desestimadas. Un buen diagrama de flujo, ayudará a una correcta compresión del funcionamiento de nuestro programa, no solo para una revisión posterior de nuestra parte, sino para otras personas ajenas al proyecto que deban tomar contacto con el.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

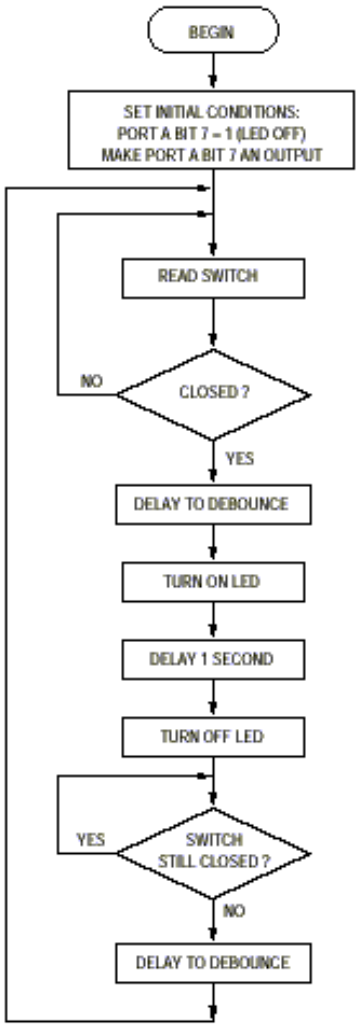


Fig. 6-1. – Diagrama de Flujo de Ejemplo.

Mnemónico del Código fuente :

Una vez que el diagrama de flujo o plan sea completado, el programador desarrollará una serie de instrucciones en lenguaje ensamblador que realizarán las funciones llamadas por cada bloque del diagrama. El programador está limitado a elegir las instrucciones desde un listado de instrucciones (set de instrucciones) correspondientes al CPU utilizado.
El programador escribe las instrucciones en forma de mnemónicos que son fáciles de entender.
La figura 6-2, muestra el Mnemónico del código fuente junto con el diagrama de flujo del programa de ejemplo, donde se puede observar las distintas instrucciones del CPU usadas en cada bloque del diagrama de flujo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

El significado de los Mnemónicos usados en este ejemplo, pueden encontrarse en el Apéndice A.
Durante el desarrollo del programa, se debe hacer notar el uso de una demora de tiempo en tres lugares distintos. Una sub-rutina fue creada para generar una demora de 50 mSeg.
La misma se usa en dos lugares directamente para producir el algoritmo de "antirrebote" de los pulsadores y además participar en la construcción de la demora de 1 segundo.
Para mantener esta figura simple, los comentarios que deberían incluirse en el código fuente, fueron omitidos en forma intencional.
Los comentarios completos se muestran en el listado 6.1.

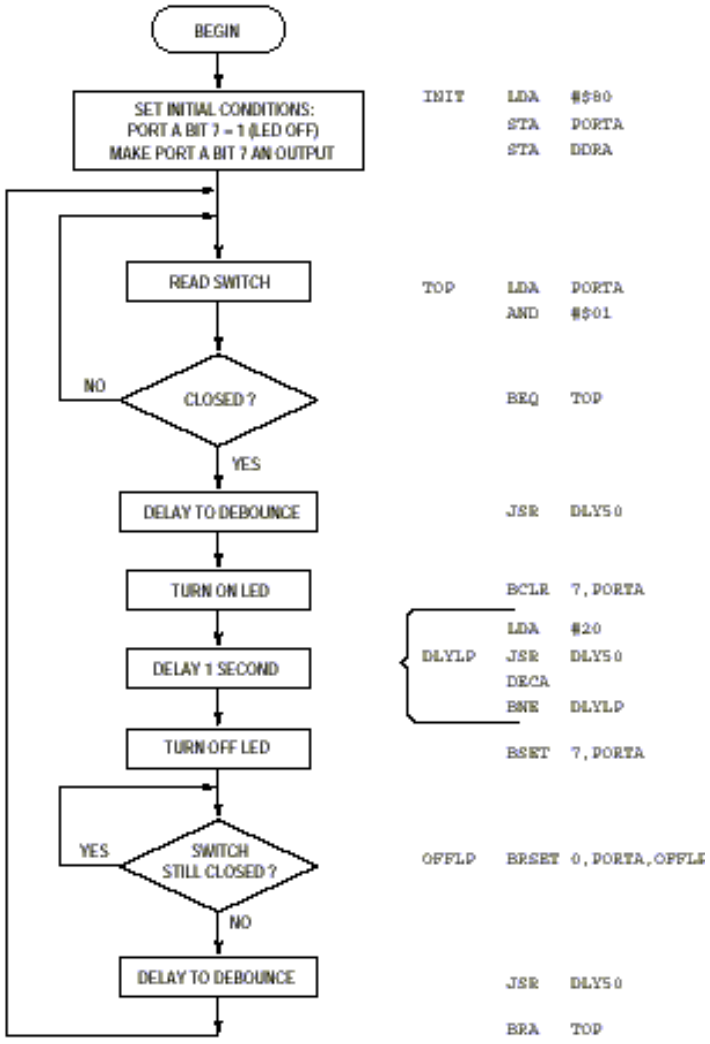


Fig. 6-2 – Diagrama de Flujo y Mnemónicos.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Software de Demora.

En la figura 6-3, se muestra un diagrama de flujo expandido de la sub-rutina de la demora de 50 mSeg.
Una sub-rutina, es un programa relativamente pequeño que realiza algunas funciones comunes requeridas. Siempre que una función, necesita ser realizada muchas veces en el curso de un programa, la sub-rutina solamente es escrita una sola vez. En cada lugar donde se necesita esta función el programador podría utilizarla por medio de instrucciones como "salto corto a sub-rutina" (Branch to Subroutine – BSR) o por medio de un "salto largo a sub-rutina" (Jump to Subroutine – JSR).

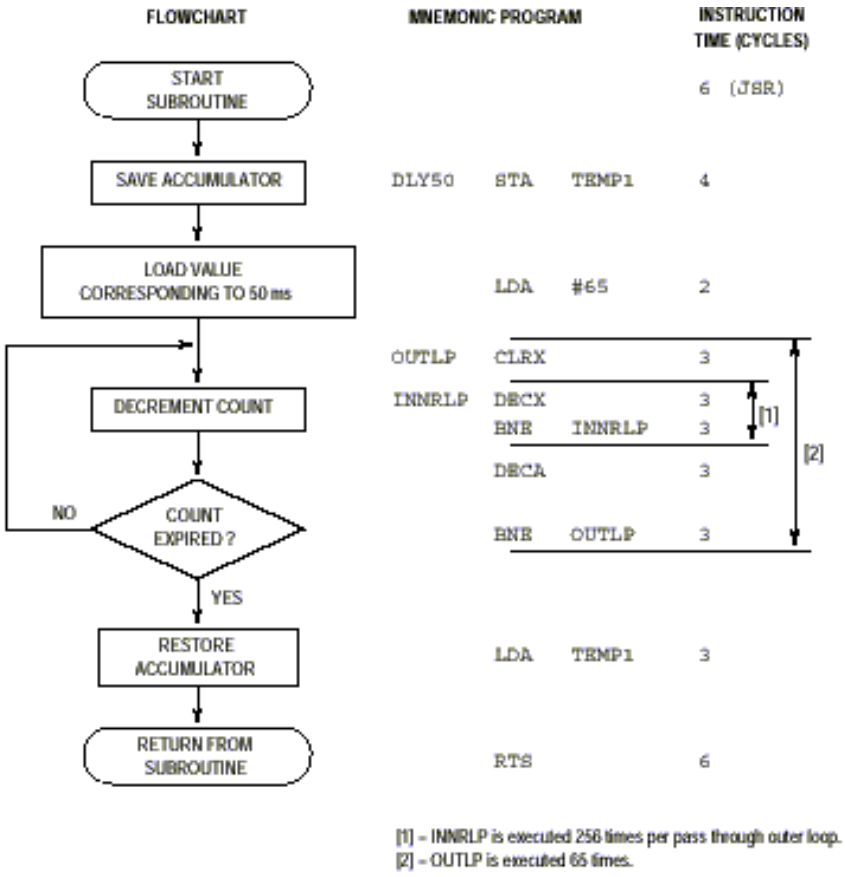


Fig. 6-3 – Diagrama de Flujo de la rutina de Demora y Mnemónicos.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Antes de comenzar a ejecutar la instrucción en la sub-rutina , la dirección de la instrucción que sigue al JSR (o BSR) se almacena automáticamente en el STACK (PILA) que está ubicado dentro del mapa de la memoria RAM temporaria. Cuando el CPU finaliza la ejecución de las instrucciones dentro de la sub-rutina, una instrucción llamada "Retorno desde Sub-rutina" (Return fron Subroutine – RTS) se ejecuta como última instrucción dentro de la sub-rutina. La instrucción RTS, causa que el CPU recobre la dirección de "retorno" previamente salvada (guardada en el STACK o pila), de esta forma el CPU continua el programa con la instrucción siguiente a la instrucción JSR (o BSR) que originalmente llamo a la sub-rutina.
La rutina de demora de la figura 6-3 involucra un "loop interno o inherente" (INNRLP) dentro de otro loop (OUTLP) . El loop interno está formado por dos instrucciones que se ejecutan 256 veces antes que X (puntero índice) alcance el valor $00 y el salto condicional BNE falle (o sea la condición sea por igual). Esto suma la cantidad de 6 ciclos a 500 nS por ciclo ejecutados 256 veces, lo que dá un valor del loop interno de 0,768 mS cada vez que el mismo es ejecutado. El loop externo se ejecuta 65 veces. El tiempo total de ejecución para el loop externo es de 65 x (1536 + 9) o 100.425 ciclos que dan un total de 50,212 mS.

Las instrucciones "misceláneas" en esta sub-rutina de demora suman un total de 21 ciclos, lo que arroja un tiempo total para la rutina DLY50 de 50,223 mS incluyendo el tiempo requerido por la instrucción JSR que llama a DLY50.

El sistema de timer (temporizador) incluido dentro del MCU MC68HC705J1A también se puede usar para medir tiempos. El uso de timers es siempre aconsejable, ya que el CPU puede realizar otras tareas durante la demora o temporización, y el tiempo de la demora no es afectado por el número exacto de instrucciones ejecutadas como lo es en el ejemplo en DLY50.

Listado Assembler.

Después de que un programa completo o sub-rutina se escribe, debe convertirse ello desde los mnemónicos al código binario de máquina que el CPU pueda luego ejecutar. Un sistema de computación separado tal como una PC (Personal Computer o computadora personal), se usa para convertir ello en un lenguaje de máquina. El programa para realizar dicha conversión se denomina "ensamblador o Compilador Assembler". El ensamblador lee la versión en mnemónico del programa (también llamada la versión "fuente" o "Source" del programa) y produce una versión código de máquina del programa en el formato que pueda ser programado dentro de la memoria del MCU.
El ensamblador también produce un listado "compuesto" que muestra al mismo tiempo el programa original en mnemónicos y la traslación en código objeto.
Este listado se usa durante la fase de depuración del programa y es parte de la documentación del programa de software.

El listado 6-1, muestra como resulta el "ensamblado" del programa de ejemplo. Los comentarios fueron agregados antes de ejecutar el programa de ensamblado.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
                        ********************************************************
                        * Simple 68HC05 Program Example                        *
                        * Read state of switch at port A bit-0; 1 = closed     *
                        * When sw. closes, light LED for about 1 sec; LED on   *
                        * when port A bit-7 = 0. Wait for sw release,          *
                        * then repeat. Debounce sw 50 ms on & off             *
                        * NOTE: Timing based on instruction execution times    *
                        *  If using a simulator or crystal less than 4 MHz,    *
                        *  this routine will run slower than intended          *
                        ********************************************************
                        $BASE     10T                   ;Tell assembler to use decimal
                                                        ;unless $ or % before value
0000                    PORTA     EQU     $00           ;Direct address of port A
0004                    DDRA      EQU     $04           ;Data direction control, port A
00E0                    TEMP1     EQU     $C0           ;One byte temp storage location

0300                              ORG     $0300         ;Program will start at $0300

0300  A6 80             INIT      LDA     #$80          ;Begin initialization
0302  B7 00                       STA     PORTA         ;So LED will be off
0304  B7 04                       STA     DDRA          ;Set port A bit-7 as output
                        * Rest of port A is configured as inputs

0306  B6 00             TOP       LDA     PORTA         ;Read sw at LSB of Port A
0308  A4 01                       AND     #$01          ;To test bit-0
030A  27 FA                       BEQ     TOP           ;Loop till Bit-0 = 1
030C  CD 03 23                    JSR     DLY50         ;Delay about 50 ms to debounce
030F  1F 00                       BCLR    7,PORTA       ;Turn on LED (bit-7 to zero)
0311  A6 14                       LDA     #20           ;Decimal 20 assembles to $14
0313  CD 03 23          DLYLP     JSR     DLY50         ;Delay 50 ms
0316  4A                          DECA                  ;Loop counter for 20 loops
0317  26 FA                       BNE     DLYLP         ;20 times (20-19,19-18,...1-0)
0319  1E 00                       BSET    7,PORTA       ;Turn LED back off
031B  00 00 FD          OFFLP     BRSET   0,PORTA,OFFLP ;Loop here till sw off
031E  CD 03 23                    JSR     DLY50         ;Debounce release
0321  20 E3                       BRA     TOP           ;Look for next sw closure

                        ***
                        * DLY50 - Subroutine to delay ~50mS
                        * Save original accumulator value
                        * but X will always be zero on return
                        ***

0323  B7 C0             DLY50     STA     TEMP1         ;Save accumulator in RAM
0325  A6 41                       LDA     #65           ;Do outer loop 65 times
0327  5F                OUTLP     CLRX                  ;X used as inner loop count
0328  5A                INNRLP    DECX                  ;0-FF, FF-FE,...1-0 256 loops
0329  26 FD                       BNE     INNRLP        ;6 cyc*256*500ns/cyc = 0.768 ms
032B  4A                          DECA                  ;65-64, 64-63,...1-0
032C  26 F9                       BNE     OUTLP         ;1545cyc*65*500ns/cyc=50.212ms
032E  B6 C0                       LDA     TEMP1         ;Recover saved Accumulator val
0330  81                          RTS                   ;Return
```

Listado 6-1 – Listado Assembler.

Ahora nos referiremos a la figura 6-4 para la siguiente discusión. Esta figura muestra algunas líneas del listado con números de referencia indicando varias partes de la línea. La primera línea es un ejemplo de una “línea de directiva” del assembler. Esta línea no es realmente parte del programa; sin embargo, esta provee información al ensamblador de tal forma que el programa real pueda ser correctamente convertido en código binario de máquina.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
0000              PORTA    EQU     $00       ;Direct address of port A

0300                       ORG     $0300     ;Program will start at $0200

0306  B6 00       TOP      LDA     PORTA     ;Read sw at LSB of Port A
----  --------- -------  ----    -------- --------------------------------
[1]    [2]       [3]      [4]     [5]       [6]->
```

Fig. 6-4 – Explicación del listado Assembler.

EQU, es la forma corta de escribir EQUATE, se usa para proporcionar una ubicación específica de memoria o a un número binario dotarlo de un nombre que pueda usarse en otra instrucción de programa. En este caso, la directiva EQU está siendo usada para asignarle el nombre PORTA al valor $00, el cuál es la dirección del registro del puerto A en el MC68HC705J1A. De esta forma, es más fácil para el programador, recordar el mnemónico con el nombre PORTA que un valor numérico anónimo de $00. Cuando el ensamblador encuentra uno de estos nombres, el nombre es automáticamente reemplazado por su correspondiente valor binario de la misma forma que un mnemónico es reemplazado por un código de instrucción binario.
La segunda línea mostrada en la figura 6-4 es otra directiva del ensamblador.
El mnemónico ORG, el cuál es la forma corta de escribir ORIGINATE, le dice al ensamblador donde el programa comenzará (la dirección del comienzo de la primera instrucción siguiente a la directiva ORG). Más de una directiva ORG puede usarse en un programa para decirle al ensamblador donde poner diferentes partes de un programa en lugares específicos de la memoria. Referirse al mapa de memoria del MCU elegido para seleccionar un localización apropiada de memoria donde el programa debería comenzar.

En el listado de assembler dado como ejemplo, los dos primeros campos, [1] y [2], son generados por el ensamblador, y los últimos 4 campos, [3], [4],[5], y [6], son del programa fuente original escrito por el programador. El campo [3] es una etiqueta o label (TOP) la cuál puede ser referida por otra instrucción dentro del programa. En nuestro programa de ejemplo, la última instrucción fue “BRA TOP”, la cuál simplemente significa que el CPU continuará ejecutando las instrucciones dentro del loop marcado por esa etiqueta “TOP”.

Cuando el programador está escribiendo un programa, las direcciones donde las instrucciones serán alojadas no son típicamente conocidas. Peor aun, en las instrucciones de salto, raramente se usa la dirección de destino, el CPU usa un “offset” (diferencia) entre el valor corriente del PC (Program Counter) y la dirección de destino. Afortunadamente, el programador no debe preocuparse por esos problemas, porque el ensamblador toma cuenta de esos detalles por medio de un sistema de “etiquetas”.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Este sistema de etiquetas es una manera conveniente para el programador de identificar puntos específicos en el programa (sin conocer las direcciones exactas); el ensamblador puede luego convertir esas etiquetas mnemónicos en direcciones específicas de memoria y siempre calcular los offsets para las instrucciones de salto que el CPU puede utilizar luego.

El campo [4] es una instrucción de campo. El mnemónico LDA es la forma corta de escribir LOAD ACCUMULATOR (carga del acumulador). Como hay 6 variantes (diferentes códigos) de la instrucción "load accumulator", se requiere información adicional antes que el ensamblador pueda elegir el código binario correcto para que el CPU utilice durante la ejecución del programa. El campo [5] es el campo del "operando", que provee información sobre locación específica de memoria o valores a ser operados por la instrucción. El ensamblador usa ambos, el mnemónico de instrucción y el operando especificado en el programa fuente para determinar el código específico para la instrucción.

Las diferentes maneras de especificar el valor a ser operado es llamado "modos de direccionamiento"(una más completa discusión al respecto, fue presentada en el capítulo 5).
La sintaxis del campo de operando es sutilmente diferente para cada modo de direccionamiento de esta forma el ensamblador puede determinar el código correcto para cada modo de direccionamiento desde la sintaxis del operando. En este caso, el operando [5] es PORTA, el cuál el ensamblador convierte en forma automática a $00 (re-llamando a directiva EQU). El ensamblador interpreta $00 como un modo de direccionamiento directo entre $0000 y $00FF, de este modo selecciona el código (opcode) $B6,el cuál es la variación correspondiente al modo de direccionamiento directo de la instrucción LDA.
Si PORTA fuera precedida por el símbolo "#", la sintaxis debería ser interpretada por el ensamblador como un modo de direccionamiento "inmediato", y el código $A6 debería ser elegido en lugar de $B6.

El campo [6] es conocido como el campo de comentarios y no es utilizado por el ensamblador para trasladar el programa en código de máquina. En lugar de ello, el campo de comentarios es usado por el programador para documentar el programa. Aunque el CPU no utilice esta información durante la ejecución del programa, un buen programador sabe que ello es una de las partes más importantes de un buen programa. El comentario [6] para esa línea del programa dice "; Lee sw en el LSB del port A". Este comentario le dice a alguien que está leyendo el listado porque el port A está siendo leído, lo cuál es esencial para entender como trabaja el programa. El punto y coma (;) indica que el resto de la línea debería ser tratado como un comentario (no todos los ensambladores requieren el ";") .
Una línea entera puede ser comentario usando un asterisco "*" como primer carácter de la línea. Además de utilizar buenos comentarios en un programa, es bueno recomendar el uso de diagramas de flujo u otro tipo de anotaciones que sirva para entender el flujo de ejecución de un programa.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Archivo de Código Objeto.

Nosotros aprendimos en el capítulo 4 que la computadora espera que el programa sea una serie de valores de 8 bits en memoria. Nada tan alejado de ello en su aspecto, ya que nuestro programa fue escrito para personas. La versión que la computadora necesita cargar en su memoria es llamada "archivo de código objeto" (en inglés, Object Code File).
Para los microcontroladores Motorola, la forma más común de archivo de código objeto es el archivo "S-Record". El ensamblador puede generar en forma directa un archivo de listado y / o un archivo de código objeto.
Un archivo S-record es un archivo de texto que puede ser leído por un editor de texto o un procesador de palabras. Usted no debería tratar de editar este tipo de archivo, porque la estructura y el contenido de este archivo son críticos para el buen funcionamiento del mismo. Cada línea de un archivo S-record es un registro. Cada registro comienza con una S mayúscula seguida por un número de código desde 0 hasta 9. Los únicos números de código que son importantes para nosotros son S0, S1 y S9. S0 es un registro encabezado opcional (header record) que puede contener el nombre del archivo para beneficio del personal humano que necesita mantener dicho archivo. S1 es un registro que contiene los datos principales del programa. Un registro S9 se usa para marcar el fin de un archivo S-record.
Para el trabajo que estamos haciendo con microcontroladores de 8 bits, la información en un registro S9 no es importante, pero un registro S9 siempre es requerido en el fin de nuestros archivos S-record. La figura 6-5 nos muestra la sintaxis de un registro S1.

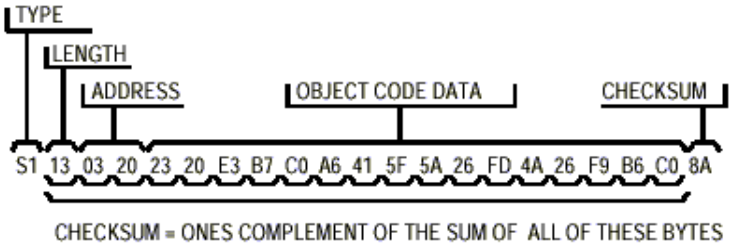


Fig. 6-5 – Sintaxis de un Archivo S1 – Record.

Todos los números en un archivo S-record están en hexadecimal. Nosotros usaremos tipos de campos S0, S1 y S9 para nuestros archivos S-record (1er campo).El largo del campo es el número de pares de dígitos hexadecimales en el registro excluyendo los campos de tipo y longitud. El campo de dirección es una dirección de 16 bits donde el primer byte de dato será almacenado en memoria. Cada par de dígitos hexadecimales en el campo de datos de código de máquina representa un valor de dato de 8 bits a ser almacenado en sucesivas posiciones de memoria. El campo de "Checksum" es un valor de 8 bits que representa el complemento a uno de la suma de todos los bytes en el archivo S-record a excepción de los campos de tipo y checksum respectivamente. Este Checksum se usa durante la carga de un archivo S-record para verificar que los datos están completos y correctos en cada registro.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

La figura 6-6 es el archivo S-record resultante de compilar (pasar por el ensamblador) el programa de ejemplo del listado 6-1. Los dos bytes de dato de código de máquina que están resaltados son los mismos dos bytes que fueron resaltados en la figura 4-2 y el texto que sigue a la figura 4-2. Estos bytes fueron localizados buscando en el listado y viendo que la dirección donde comienza esa instrucción era $0323. En el archivo S-record encontramos el registro S1 con la dirección $0320. Moviéndonos a la derecha encontramos el dato $23 para la dirección $0320, $20 para la dirección $0321, $E3 para $0322, y finalmente el byte que buscamos para la dirección $0323 y $0324.

```
S1130300A680B700B704B600A40127FACD03231FC3
S113031000A614CD03234A26FA1E000000FDCD03D7
S11303202320E3B7C0A6415F5A26FD4A26F9B6C08A
S10403308147
S9030000FC
```

Fig. 6-6 – Archivo S- Record para el Programa de Ejemplo

Directivas del Ensamblador.

En esta sección se discutirán seis de las más importantes directivas del programa ensamblador. Los ensambladores de los diferentes vendedores, difieren en el número y clase de directivas soportadas por cada uno de ellos. Debe siempre referirse al manual de usuario o documentación pertinente para conocer las particularidades del ensamblador en uso.

ORIGINATE (ORG).

Esta directiva se usa para “setear” (fijar, establecer) el contador de locación para el ensamblador. El contador de locación mantiene el seguimiento de la dirección donde el próximo byte de código de máquina será almacenado en memoria. En nuestro programa de ejemplo había una directiva ORG que fijaba el comienzo de nuestro programa en $0300.

Como el ensamblador traslada sentencias de programa en instrucciones código de máquina y dato, el contador de locación es avanzado a apuntar a la próxima locación en memoria disponible.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Todo programa tiene, al menos, una directiva ORG que establece el lugar de comienzo en memoria para el programa. Programas más completos también tendrán una segunda directiva ORG cerca del fin del programa para establecer el contador de locación a la dirección donde están localizados los "vectores de Reset e Interrupciones" ($07F8 - $07FF en el MC68HC705J1A). **El Reset Vector (vector de Reset) debe SIEMPRE especificarse y es una buena práctica también especificar los vectores de interrupciones aún si no se usaran los mismos. Un error muy común en los programadores noveles (recién iniciados) es omitir los vectores de Reset e interrupciones, originando de esta forma que el ensamblador no incluya los mismos en la memoria de programa del MCU. Este error genera que el MCU al ser alimentado y salir de la etapa de Power On Reset (inicialización interna) busque el vector de Reset con un contenido frecuente de $0000 (memoria de programa virgen del MCU tipo OTP ROM) o $FFFF (memoria de programa virgen del MCU tipo HC908 FLASH) que NO SON POSICIONES DE MEMORIA DE PROGRAMA VALIDAS y dan como resultado que el MCU quede en un loop errático sin posibilidad de salir de el.**

EQUATE (EQU).

Esta directiva se usa para asociar un valor binario con una "etiqueta" o "label".
El valor puede ser tanto un valor de 8 bits de longitud como una dirección de 16 bits.
Esta directiva NO genera un código objeto.
Durante el proceso de ensamblado, el ensamblador debe mantener una lista cruzada de referencia donde este almacena el equivalente binario de cada etiqueta. Cuando una etiqueta aparece en el programa fuente, el ensamblador mira en esta tabla cruzada de referencia para encontrar el equivalente binario. Cada directiva EQU genera una entrada en la tabla cruzada de referencia.

Un ensamblador lee el programa fuente dos veces. En la primera pasada, el ensamblador cuenta bytes del código objeto e internamente construye la tabla cruzada de referencia. En la segunda pasada el ensamblador genera un archivo listado y / o el archivo objeto S-record.
Este arreglo de dos pasadas permite al programador generar etiquetas de referencia que son definidas más tarde en el programa.
Las directivas EQU deberían aparecer cerca del comienzo de un programa, antes que las etiquetas allí definidas sean usadas por otros pasos del programa. Si el ensamblador encuentra una etiqueta antes que esta sea definida, este no tiene elección pero asume el peor caso de un valor de 16 bits de dirección. Esto causaría que se use el modo de direccionamiento extendido en lugar de un modo más eficiente como el de direccionamiento directo donde podría ser usado.
En otros casos, el modo de direccionamiento indexado con 16 bits de offset, podría ser usado por el ensamblador, en donde un direccionamiento indexado de 8 bits o un indexado sin offset mucho más eficiente podría usarse.

En el programa de ejemplo donde había dos directivas EQU para definir las etiquetas PORTA y DDRA a sus páginas de direccionamiento directas.
Otro uso de las directivas EQU es el de identificar la posición de un bit con una etiqueta como esta.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
LED     EQU     %10000000   ;LED is connected to bit-7
 "       "       "             "
 "       "       "             "
INIT    LDA     #LED        ;There's a 1 in LED bit position
        STA     PORTA       ;So LED will be off
        STA     DDRA        ;So LED pin is an output
```

El símbolo % (porcentaje) indica que el valor que sigue está expresado en binario. Si movemos el LED a otro pin durante el desarrollo solo necesitaríamos cambiar la sentencia EQU y re-ensamblar el programa.

Formulario Bytes de Constantes (FCB).

Los argumentos para esta directiva son etiquetas o números separados por comas, que pueden ser convertidos en un solo byte de datos. Cada byte especificado en una directiva FCB, genera un byte de código de máquina en el archivo de código objeto. Las directivas FCB se usan para definir constantes en un programa, como por ejemplo una tabla de constantes, etc. .

Formulario de doble bytes (FDB).

Los argumentos para esta directiva son etiquetas o números separados por comas, que pueden ser convertidos en valores de 16 bits de datos. Cada argumento especificado en una directiva FDB, genera dos bytes de código de máquina en el archivo de código objeto.
Las siguientes líneas de un listado assembler demuestran las directivas ORG y FDB.

```
 "     "  "      "       "       "        "
 "     "  "      "       "       "        "
0300                     ORG     $0300    ;Beginning of EPROM in 705J1A

0300  B6 00    START     LDA     PORTA    ;Read sw at LSB of port A
 "     "  "      "       "       "        "
 "     "  "      "       "       "        "
041F  80       UNUSED    RTI              ;Return from unexpected int
 "     "  "      "       "       "        "
 "     "  "      "       "       "        "
07F8                     ORG     $07F8    ;Start of vector area

07F8  04 1F    TIMVEC    FDB     UNUSED   ;An unused vector
07FA  04 1F    IRQVEC    FDB     $041F    ;Argument can be a hex value
07FC  04 1F    SWIVEC    FDB     UNUSED   ;An unused vector
07FE  03 00    RESETV    FDB     START    ;Go to START on reset
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Reserva de Byte de Memoria (RMB).

Esta directiva se usa para reservar un lugar (asignar un espacio) en RAM para las variables de un programa. La directiva RMB no genera código objeto alguno, pero ello genera una entrada en la tabla cruzada interna del ensamblador.
En el programa de ejemplo (listado 6-1), la variable en RAM TEMP1 fue asignada con una directiva EQU. Otra forma de asignar esa variable podría ser de la siguiente forma:

```
00C0                    ORG     $00C0     ;Beginning of RAM in 705J1A

00C0            TEMP1   RMB     1         ;One byte temp storage location
```

Esta forma es preferible a la del uso de EQU en la asignación de espacio en RAM porque es muy común en el desarrollo de un programa borrar o agregar variables sobre la marcha. Si se usara directivas EQU se debe tener en cuenta de cambiar varias sentencias después de remover una sola variable. Con la directiva RMB, el ensamblador asigna direcciones tanto como sean necesarias.

Establecer por Definición Número de Base Decimal.

Algunos ensambladores, tal como el P & E Microcomputers Systems, asumen que cualquier valor que no sea específicamente marcado será interpretado como un número hexadecimal. Le idea es simplificar el ingreso de información numérica por la eliminación de la necesidad del uso del símbolo $ antes de cada valor. Si se quiere que el ensamblador asuma que los valores no marcados sean valores decimales, usar la directiva $BASE.

```
....            $BASE   10T             ;Set default # base to decimal

000A            TEN     EQU     #10     ;Decimal 10 not $10 = 16
```

Esta directiva es un tanto diferente con respecto a otras descriptas en esta capítulo.
La directiva $BASE comienza en la columna extrema izquierda del programa fuente.
Esta directiva está incluida cerca del comienzo de cada programa ejemplo en este curso.
Si se está usando un ensamblador que no requiere esta directiva, se puede borrar la misma o incluir un asterisco (*) al comienzo de la línea para anular la sentencia.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Popurrí de Soluciones.

Como en muchos otros campos de la ingeniería, hay más de una secuencia de instrucciones que pueden realizar una tarea en particular. Una buena forma de aprender una nueva instrucción es ver como en cuantas diferentes maneras se puede resolver un pequeño problema de programación. A esto yo lo llamo “Popurrí de Soluciones”.

La figura 6-7 nos muestra cuatro diferentes métodos para verificar el cierre de un interruptor conectado al puerto A bit 0 (PTA0).
Dos de estos métodos fueron usados en el programa de ejemplo del listado 6-1. Si bien todas las secuencias aquí presentadas realizan la misma tarea básica, existen diferencias entre ellas. Usualmente estás diferencias no son significativas, pero algunas veces ellas pueden ahorrar tiempo de ejecución o espacio de memoria de programa.
En microcontroladores pequeños, el ahorro de espacio, puede ser algo muy importante a tener en cuenta.

```
0000                    PORTA   EQU     $00       ;Direct address of port A

0300                            ORG     $0300     ;Program will start at $0300

0300  B6 00      [ 3]  TOP1    LDA     PORTA     ;Read sw at LSB of Port A
0302  A4 01      [ 2]          AND     #$01      ;To test bit-0
0304  27 FA      [ 3]          BEQ     TOP1      ;Loop till bit-0 = 1

0306  01 00 FD   [ 5]  TOP2    BRCLR   0,PORTA,TOP2  ;Loop here till sw ON

0309  B6 00      [ 3]  TOP3    LDA     PORTA     ;Read sw at LSB of Port A
030B  44         [ 3]          LSRA              ;Bit-0 shifts to carry
030C  24 FB      [ 3]          BCC     TOP3      ;Loop till switch ON

030E  A6 01      [ 2]          LDA     #$01      ;1 in LSB
0310  B5 00      [ 3]  TOP4    BIT     PORTA     ;To test sw at bit-0
0312  27 FC      [ 3]          BEQ     TOP4      ;Loop till switch ON
```

Fig. 6-7 – Cuatro Maneras de Chequear un Interruptor.

Los números entre paréntesis cuadrados son los números de ciclos de CPU que se requieren por instrucción en cada línea del programa. La secuencia TOP1 ocupa 6 bytes de espacio de programa y toma 8 ciclos para su ejecución. El acumulador está en $01 cuando el programa “cae” (sigue de largo) a través de la instrucción BEQ.
La secuencia TOP2 ocupa solamente 3 bytes de memoria y 5 ciclos para su ejecución. Además el acumulador no es alterado (Esta es probablemente la mejor secuencia en la mayoría de los casos).
La secuencia TOP3 ocupa un byte menos que TOP1 pero también insume un ciclo más en la ejecución de la misma. Después de la secuencia TOP3, el acumulador mantendrá los otros 7 bits leídos desde el PORT A, también desplazados en una posición a la derecha.
La última secuencia ocupa 6 bytes en memoria y toma 8 ciclos su ejecución, pero el loop en si mismo solamente toma 6 ciclos. Con ejercicios como el anterior, nuestra habilidad para encontrar mejores soluciones en cantidad de ciclos y espacios de memoria ocupados irá mejorando, lo cuál será muy útil a la hora de necesitar reducir la extensión de nuestros programas o mejorar la velocidad de ejecución del mismo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Sistemas de Desarrollo.

Motorola y fabricantes de terceras fuentes, disponen de una variedad muy extensa de herramientas de desarrollo tanto para la familia HC705 como para la nueva familia HC 908 FLASH. En esta basta propuesta de sistemas, encontraremos desde herramientas profesionales de altas prestaciones y por consiguiente, costos elevados, hasta herramientas de muy bajo costo pero con prestaciones más que interesantes para su categoría.
Para la nueva familia HC908 FLASH, presentaremos más adelante (en el curso complementario (Parte II) "Curso sobre Microcontroladores HC908 FLASH" incluido en el presente curso) un portfolio de herramientas de fabricación nacional que permiten prestaciones similares a equipos de mayores costos.

Pues bien, los simuladores en circuito (ICS) (In-Circuit Simulator) tal el nombre genérico para la familia de MCUs HC705, ofrecidos por Motorola, incluyen simulación pura (software) y simulación en circuito propiamente dicha. Estos simuladores en circuito (ICS05xxx para las distintas versiones de la familia HC705) poseen una placa principal (Board) denominada "POD" que se conecta por medio de un puerto serial (COMxx) a una PC (Computadora Personal). Un conector del tipo DIP y un cable plano permiten al ICS conectarse con la aplicación bajo desarrollo tomando el lugar del microcontrolador que eventualmente esté en uso. Un zócalo especial, del tipo ZIF (Fuerza de Inserción Cero) viene provisto en estos sistemas. Este zócalo permite programar chips del MCU bajo desarrollo, tanto en versiones OTP como en versiones EPROM (borrables por medio de luz ultravioleta) desde la misma PC.

A esta altura, se hace necesario, explicar las diferencias existentes entre **"Simulación Pura", "Simulación En Circuito"** y **"Emulación en Circuito".**

Un **"Simulador Puro"** o simplemente **"Simulador"** es un programa para una computadora personal (PC) que nos ayuda durante el desarrollo y depuración de un programa dado.
Esta herramienta de soft "simula" las acciones de un microcontrolador real, pero tiene algunas ventajas importantes. En un simulador tenemos un completo control de cómo y cuando el CPU debería avanzar a la próxima instrucción. También se puede mirar y cambiar valores de registros internos o de memoria RAM, etc, etc, antes de ir a la próxima instrucción.

Los simuladores NO CORREN A VELOCIDAD DE TIEMPO REAL!! . Mientras la PC está "simulando" las acciones del MCU con programas de software, cada instrucción del MCU tomará mucho más tiempo de ejecución que si estas fueran efectuadas por el MCU REAL. Para muchos programas de MCUs, esta reducción en la velocidad no es algo problemática. Sin embargo, NO puede considerarse como TIEMPO REAL la ejecución de las instrucciones en los simuladores, ya que la velocidad de ejecución de cada instrucción estará fuertemente ligada a la velocidad de ejecución del programa simulador en la PC. Por lo que en PCs con alta perfomance, el programa simulador correrá mucho más rápido que en una PC de baja perfomance, pero nunca podrá igualar a la velocidad real de ejecución de cada instrucción como lo haría el propio MCU.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Un truco para acelerar la ejecución de algunos loops de demora que tomarían algún tiempo su ejecución en simulación, es reemplazar las cuentas a alcanzar por los contadores dentro del lazo por valores más pequeños, de esta forma, se llegará más rápido al valor final de la cuenta. Tener presente que, una vez terminada la simulación, se deberán reemplazar nuevamente los valores por los originales, a fin de no alterar la ejecución del programa en el MCU real.

Un **"Simulador en Circuito"** es un simulador que puede ser conectado al sistema del usuario (sistema bajo desarrollo) en lugar del microcontrolador. Un simulador puro normalmente solo toma información de entrada por medio de la PC y muestra resultados y salidas por medio del display o pantalla de la PC. En un simulador En Circuito, las entradas y las salidas pueden controlarse en forma real como si fuera el propio microcontrolador el que tomará el control de las mismas. Las consideraciones anteriores (simulador puro) siguen siendo válidas para la simulación en circuito. Todo transcurrirá más lentamente que en un MCU real pero la gran diferencia aquí radica en la posibilidad de controlar entradas y salidas del MCU en forma física y NO simulada por software.

Trabajar con este tipo de herramientas (simuladores / simuladores en circuito) es mucho más sencillo que si lo hiciéramos directamente sobre el MCU, volcándole el programa en la memoria del mismo sin posibilidad de "ver" lo que está sucediendo con el programa en cada ciclo de instrucción del mismo. Solo nos quedaría el método de "prueba y error" con múltiples intentos (borrado y re-grabación de un chip versión EPROM) hasta alcanzar el objetivo deseado.
En los Simuladores / Simuladores en Circuito, podemos parar nuestro programa a voluntad, producir puntos de quiebre (paradas) o breakpoints en lugares específicos, modificar / verificar valores de registros, posiciones de memoria RAM, correr paso a paso nuestro programa, etc, etc. Además nos informa cuando una variable no ha sido inicializada al intentar usarla sin haberlo hecho.
Todo ello contribuye a facilitar la tarea del desarrollador, ya que permite observar con suma facilidad los errores cometidos en la confección de un programa y corregirlos en consecuencia.

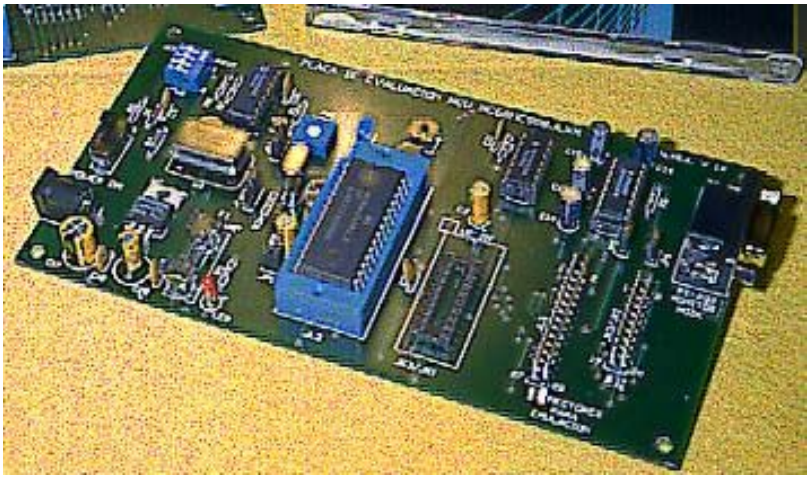
Un **"Emulador En Circuito"** o **ICE (In-Circuit Emulator)** es una herramienta de desarrollo en TIEMPO REAL. El "emulador" está construido alrededor de un MCU REAL de forma tal que este pueda ejecutar instrucciones de programa exactamente como ellas fueran ejecutadas en la aplicación final. Por lo general, un emulador dispone de memoria RAM donde debería haber memoria ROM o EPROM en el MCU final. Esto permite una rápida carga del programa en el emulador y efectuar cambios constantes en el programa durante el proceso de depuración del mismo. Una serie de circuitos extras permiten al emulador establecer "Breakpoints" o puntos de quiebre en el programa bajo desarrollo.
Cuando el programa alcanza una de estas direcciones de breakpoint, el programa bajo desarrollo el programa temporariamente se detiene y un "programa monitor" de dicha herramienta toma control. Este programa monitor permite mirar o cambiar registros del CPU, locaciones de memoria, o registros de control.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Un emulador tiene típicamente menos visibilidad de las acciones internas del MCU que un simulador, pero SOLO el emulador puede correr las instrucciones del programa bajo desarrollo en TIEMPO REAL, o sea a la velocidad REAL del MCU. Esta ventaja es particularmente apreciable, en programas de aplicación donde existan interrupciones u otro tipo de eventos muy rápidos que no puedan "simularse" debido a su velocidad y su no repetibilidad en el tiempo. Además tener la posibilidad de correr el programa bajo desarrollo en tiempo real, permite constatar que los tiempos calculados en los distintos retardos o temporizados que tuviera, sean exactos o bien actuar en función de ello.
Los sistemas emuladores en tiempo real, son por lo general, más costosos que los simuladores en circuito o aún más costosos que un simple simulador puro (software). Sin embargo, como se verá más adelante en el curso complementario sobre HC908 FLASH, la nueva tecnología de esta familia de MCUs, posibilita el acceso a EMULADORES en Tiempo Real (que también funcionan como Simuladores puros / Simuladores En Circuito / Programadores) muy económicos de costos comparables a los Simuladores en Circuito para la familia HC705. Gracias a esta facilidad que presenta la nueva familia HC908 FLASH de Motorola, el diseñador ve facilitado su trabajo de desarrollo, ya que cuenta con las características de una herramienta muy poderosa como un emulador en tiempo real, pero sin afrontar los costos altos que por lo general presentan este tipo de sistemas.

"E-FLASH08" - Sistema de Desarrollo para toda la familia HC908 FLASH

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar



“EVAL08GP /JL” – Sistemas de Desarrollo tipo I.C.E para HC908GP32 HC908JL3 / JK3 / JK1.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Revisión del Capítulo 6.

El proceso de escritura de un programa comienza con un plan. Un "Diagrama de Flujo" puede usarse para documentar el plan. Mnemónicos de sentencias en código fuente para cada block son escritos desde el diagrama de flujo. Las sentencias mnemónico de código fuente pueden incluir cualquier instrucción del repertorio de instrucciones del MCU bajo desarrollo. El próximo paso es combinar todas las instrucciones del programa con las directivas del ensamblador para tener un archivo de texto fuente.

Las directivas del ensamblador son sentencias de programa que le dan instrucciones al ensamblador en lugar del CPU del microcontrolador.
Estas instrucciones le dicen al ensamblador cosas como donde alojar instrucciones en la memoria del MCU. Las directivas assembler también le informan al ensamblador el significado binario una etiqueta en mnemónico.
Seis directivas se han discutido.

ORG – Las directivas ORIGINATE establecen la dirección de comienzo para el código objeto que le sigue.

EQU – Las directivas EQUATE asocian una etiqueta con un número binario o dirección.

FCB – Las directivas de Formulario de constantes de Bytes se usan para introducir valores
constantes de 8 bits de datos en un programa.

FDB – Las directivas de Formulario de doble Bytes se usan para introducir valores constantes de 16 bits de datos o direcciones en un programa.

RMB – Las directivas de Reserva de Bytes de Memoria se usan para asignar espacios en forma de etiquetas en direcciones de RAM.

$BASE 10T – Cambia la base numérica por default a Decimal.

Después que el programa fuente se escribe completamente, se lo procesa por medio de un "ensamblador" para producir un archivo de listado y un archivo objeto S-Record.
El archivo de listado es parte de la documentación del programa. El archivo objeto S-record se puede cargar en un simulador / emulador o puede ser descargado por medio de un programador en la memoria de programa del microcontrolador.

Un lazo condicional (loop condicional) puede producir un tiempo de demora. Ls demora dependerá del tiempo de ejecución de las instrucciones dentro del lazo o loop. Una sub-rutina tal como, ese tipo de rutina de demora, puede usarse muchas veces en un programa llamándolas con las instrucciones JSR o BSR.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

En el popurrí de soluciones, vimos como un mismo problema se puede resolver de muchas formas o sea con diferentes secuencias. Cada secuencia ocupará distintos espacios en memoria y tomará más o menos ciclos de máquina su resolución. La selección de una secuencia dada será la mejor para cada situación.

Un **simulador** o **simulador puro** es una herramienta de desarrollo que corre en una PC y simula las acciones de un microcontrolador. Un simulador solo involucra a la PC o sea Software y no es una herramienta de Tiempo Real (las instrucciones no corren a la velocidad de clock del MCU, sino a la velocidad resultante del software de simulación que está bajo el control de la PC), por lo que no es útil cuando los eventos a depurar son muy rápidos.
Un **Simulador en Circuito (ICS)**, toma la idea del simulador puro, pero le agrega una placa de interfase con la PC y el sistema a depurar del usuario, que permite controlar los puertos I/O del MCU bajo desarrollo. Es un paso adelante en la depuración de un programa, ya que permite controlar el mundo "real" externo al MCU. Al igual que el simulador, no es una herramienta en Tiempo Real.

Un **emulador (I.C.E)** está construido alrededor de un MCU real de tal forma que se pueda correr las instrucciones a la velocidad final del MCU. Los emuladores usan memoria RAM en lugar de memoria EPROM o ROM para el área de programa, ya que de esta forma se hacen más sencillas las reiteradas modificaciones al programa durante la depuración del mismo.
Los simuladores / simuladores en circuito / emuladores hacen más fácil la tarea de depuración de un programa. Estos permites ejecutar las instrucciones paso a paso, modificar valores de registros internos al CPU, valores de memoria RAM o ROM y hacer cambios con gran facilidad en los programas originales bajo desarrollo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Capítulo 7.

El Lazo Cíclico o “lazo de paso” .

Este capítulo presenta una estructura de programa de uso general que puede usarse como un “entorno de trabajo” (framework) para muchas aplicaciones con microcontroladores. La mayoría de las tareas de los sistemas se escriben como sub-rutinas. Estas sub-rutinas están organizadas dentro de un lazo (loop) de forma tal que cada una es llamada una vez por cada pasada por el lazo. En el tope del lazo hay una pequeña rutina que cíclicamente “lanza” al mismo a su ejecución en intervalos regulares. Este tipo de estructuras son conocidas como de “lazo cíclico” o programa tipo “lazo de paso”. Un programa de reloj se mantiene como la primera tarea a ejecutar en el lazo. Este reloj puede usarse como una entrada para otra sub-rutina de tareas que decidirá cuál de las sub-rutinas deberá activarse en cada pasada del lazo principal.

Adicionalmente a la estructura de lazo en si misma, este capítulo discutirá la inicialización de un sistema y las definiciones de software que facilitarán la tarea para dedicar nuestro tiempo a las rutinas específicas de nuestra aplicación.

Equates del Sistema.

Es poco conveniente el uso de patrones binarios y direcciones en las instrucciones de un programa. Las directivas “Equate” (EQU) se usan para asignar nombres Mnemónicos a registros de direcciones y posiciones de bits. Estos nombres pueden usarse en instrucciones de un programa en lugar de los números binarios. Esto hace al programa mucho más fácil de escribir y de leer. Cuando se usa un Simulador / Emulador En Circuito para depurar un programa de aplicación, los nombres mnemónicos pueden usarse en la pantalla de depuración en lugar de los números y direcciones binarias.

Equates para los Registros del MC68HC705J1A.

Los nombres para los registros y bits de control recomendados por el fabricante (Motorola) están incluidos en el programa de lazo cíclico o programa “lazo de paso” en el entorno de trabajo (framework) del listado 7-1. Esto permite escribir instrucciones de programa que tengan sentido para las personas en lugar de oscuros números y direcciones binarias.

Cada registro se iguala (equate) a una dirección de página directa con una directiva EQU.
Cada bit de control se define de dos formas. Primero, una directiva EQU iguala el nombre del bit a un número entre 7 y 0 correspondiente al número de bit donde cada bit se localiza en un registro de control dado. Segundo, la mayoría de los bits de control son igualados a un patrón binario de bits tal como 0010 0000 ($20) el cuál puede usarse como una máscara de bits para identificar la localización del bit en un registro.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Como no se puede igualar (hacer un equate) con el mismo nombre para dos valores binarios diferentes, el segundo equate (EQU) usará un punto "." después del nombre del bit. Para obtener un nombre de bit correspondiente al número de bit (7 – 0) se debe usar el nombre simplemente, para obtener una máscara de indicación de la posición del bit, se debe usar el mismo nombre seguido de un punto. Esta convención se usa en el entorno de trabajo (framework) del lazo cíclico o programa calesita, pero no necesariamente es un estándar recomendado por Motorola o por las companías de ensambladores.

En el set de instrucciones del MC68HC05, las instrucciones de manipulación de bits son de la siguiente forma....

```
xxxx 14 08        ------- BSET bit#,dd   ; Set bit in location dd
```

Bit# es un número entre 7 y 0 que identifica el bit dentro del registro en la localización *dd* a ser cambiado o testeado.
En otros casos, si uno quisiera construir una máscara con varios bits en "1" (seteados) y luego escribir este valor compuesto en la localización de un registro, por ejemplo supongamos querer setear los bits RTIFR, RTIE y RT1 en el Registro TCSR, podríamos usar las siguientes instrucciones.

```
Xxxx A6 16        LDA #{RTIFR.+RTIE.+RT1.}          ;Form Mask
Xxxx B7 08        STA TCSR          ;Write mask to TCSR Register
```

El símbolo # significa modo de direccionamiento inmediato. La expresión (RTIFR.+RTIE.+RT1.) es la operación OR booleana de tres bits de la máscara de posición.
El ensamblador evalúa la expresión booleana durante la compilación (ensamblado) del programa y sustituye la respuesta (un solo valor binario de 8 bits) dentro del programa ensamblado. La siguiente sentencia de programa produciría exactamente el mismo resultado, pero no sería tan fácil de leer como la anterior.

```
xxxx  A6 16          LDA #%00010110      ;Form mask
xxxx  B7 08          STA TSCR            ;Write mask to Register
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Equates del sistema de aplicación.

Será común encontrar directivas Equates específicas en un programa que definen las señales conectadas a los pines de I/O. Estas directivas EQU deberían ubicarse después de las directivas equates estándar del MCU y antes del comienzo del programa propiamente dicho. El entorno de trabajo del tipo lazo cíclico fue desarrollado teniendo en mente una pequeña placa de desarrollo. Este sistema tiene un interruptor (switch) conectado al bit 0 del puerto A (PA0) y un LED conectado al bit 7 del puerto A (PA7) de forma tal que dichas conexiones fueron definidas con directivas EQU.
El interruptor no se usó en el programa de lazo cíclico o calesita del listado 7-1 , pero no sería malo incluir la directiva EQU relativa al mismo. Las directivas EQU no generan código objeto alguno que ocupe lugar de memoria en el sistema microcontrolado final.

Seteo de Vectores.

Todos los programas deben tener SETEADOS (definidos) los vectores de RESET y de interrupciones!!!!. Los vectores especifican la dirección donde el CPU comenzará el procesamiento de la instrucciones cuando un RESET o una INTERRUPCION ocurran.
El RESET y cada fuente de interrupción esperan encontrar sus vectores asociados en un específico par de locaciones de memoria. Por ejemplo, el vector de reset está en las dos más altas posiciones de memoria ($07FE y $07FF en el MC68HC705J1A). Si no se pusieran valores en esas posiciones, el CPU tomaría cualquiera de los valores binarios encontrados en dicha posición y los interpretaría como si fueran la dirección de comienzo del programa o de la rutina correspondiente a dicha interrupción, sin poder distinguir el error cometido.
El Reset Vector (vector de Reset) debe SIEMPRE especificarse y es una buena práctica también especificar los vectores de interrupciones aún si no se usaran los mismos. Un error muy común en los programadores noveles (recién iniciados) es omitir los vectores de Reset e interrupciones, originando de esta forma que el ensamblador no incluya los mismos en la memoria de programa del MCU. Este error genera que el MCU al ser alimentado y salir de la etapa de Power On Reset (inicialización interna) busque el vector de Reset con un contenido frecuente de $0000 (memoria de programa virgen del MCU tipo OTP ROM) o $FFFF (memoria de programa virgen del MCU tipo HC908 FLASH) que NO SON POSICIONES DE MEMORIA DE PROGRAMA VALIDAS y dan como resultado que el MCU quede en un loop errático sin posibilidad de salir de el.

RESET VECTOR (VECTOR DE RESET).

La forma usual para definir un vector es con la directiva FDB.

```
07FE  03  00        RESETV    FDB     START    ;Beginning of program on reset
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Durante el ensamblado, el ensamblador evalúa la etiqueta START en una dirección de dos bytes y almacena esa dirección en los próximos dos lugares disponibles de la memoria del programa. Las columnas a la izquierda de la línea de listado muestran que la dirección $0300 fue almacenada en $07FF. ($03 en $07FE y $00 en $07FF).

RESETV es una etiqueta opcional en esa línea de programa. Vemos también que no es usada como referencia por otra sentencia en este programa en particular. Solo fue incluida para identificar esa línea de directiva FDB como la sentencia que define el vector de reset.
El vector de reset fue seteado para apuntar a la etiqueta START. Los variados sistemas simuladores en circuito y emuladores que Motorola y otros fabricantes de terceras partes ofrecen como herramientas de muy bajo costo, usan esta información para setear la pantalla de simulación. Cuando un programa se carga en el simulador, el simulador busca la dirección en el vector de reset del programa cargado. Si se encuentra, el simulador selecciona la instrucción de programa y muestra ello en la ventana del código fuente del simulador. Si no hubiera un vector de reset, el simulador mostraría un mensaje de atención indicando que el vector de reset no fue inicializado. Aun así se podría seguir con la depuración del programa, pero el mismo no funcionaría si se quisiera programar este en la memoria de un MCU EPROM , porque el programa no comenzaría cuando se produjera el reset.

Interrupciones No Utilizadas.

Para toda interrupción que sea usada, los vectores de las mismas pueden ser definidos tal cuál como fue definido el vector de reset (con una directiva FDB). En el programa de ejemplo, la interrupción del Timer se usa para lograr interrupciones en tiempo real. La interrupción externa y la interrupción SWI no son usadas.

Es una buena idea setear los vectores de interrupción NO UTILIZADOS para el caso que una de esas interrupciones sea inesperadamente requerida. Bien podríamos imaginarnos lo que sucedería en un sistema funcionando si alguna interrupción no utilizada fuera atendida.
Este tipo de errores es muy común en programadores noveles, que podrían ocasionar la habilitación y el disparo de interrupciones no contempladas en el programa original.

Las siguientes líneas del listado muestran como las interrupciones y el vector de reset fueron seteados en el programa de ejemplo.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
                        ******************************************************
                        * RTIF interrupt service routine
                        ******************************************************
0345 3A E0      RTICNT  DEC     RTIFs       ;On each RTIF
 "   "  "       "       "       "            "
 "   "  "       "       "       "            "
0351 80         AnRTI   RTI                 ;Return from RTIF interrupt

0351            UNUSED  EQU     AnRTI       ;Use RTI at AnRTI for unused
                                            ;interrupts to just return

                        ******************************************************
                        * Interrupt & reset vectors
                        ******************************************************
07F8                    ORG     $07F8       ;Start of vector area

07F8 03 45      TIMVEC  FDB     RTICNT      ;Count RTIFs 3/TIC
07FA 03 51      IRQVEC  FDB     UNUSED      ;Change if vector used
07FC 03 51      SWIVEC  FDB     UNUSED      ;Change if vector used
07FE 03 00      RESETV  FDB     START       ;Beginning of program on reset
```

Las primeras líneas en este listado parcial muestran las primeras y las últimas líneas de la rutina de servicio de la interrupción del Timer. La línea.....

0351 80	AnRTI	RTI	;Return from RTIF interrupt

Muestra una instrucción de Retorno desde Interrupción (RTI , Return To Interrupt) con la etiqueta “AnRTI”. La próxima línea iguala la etiqueta “UNUSED” a la dirección de la instrucción RTI en AnRTI. Más abajo en el listado, los vectores de interrupciones no utilizados y la interrupción SWI se setean para apuntar a esa instrucción RTI ($0351).

Si una interrupción SWI fuera encontrada inesperadamente, el CPU salvaría los registros de internos de este en el stack (pila, RAM temporaria) y cargaría el contador de programa con la dirección $0351 el vector de SWI. El CPU cargaría luego la instrucción RTI desde la dirección $0351. La instrucción RTI le diría al CPU que recobre los registros salvados del CPU (incluyendo el contador de programa) desde el stack (pila). El valor del contador de programa recobrado determinaría que haría el CPU como próximo paso.

Una manera alternativa de responder a una interrupción no esperada sería resetear el stack pointer (puntero de pila , SP) (con una instrucción RSP) y luego saltar a la misma dirección como si un reset hubiera ocurrido. Esta aproximación parte de asunción pesimista que si una interrupción no esperada ocurre, puede haber otros problemas serios. Por acción de resetear el stack pointer (SP) y empezar todo nuevamente, podríamos pensar que estaríamos en mejores condiciones para corregir la o las causas que pudieron haber provocado la interrupción no esperada.

Mientras se depura un programa en un simulador, hay otra forma posible de manejar interrupciones no esperadas.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

" " "	" "	"	"	"
0351	BADINT	BRA	BADINT	;Infinite loop to here
" " "	" "	"	"	"
07FA	VECTOR	FDB	BADINT	;Hang on unexpected int

En este esquema, una interrupción no esperada causará que el vector asociada a esta, apunte a BADINT. La instrucción a BADINT es un lazo infinito a la propia instrucción BADINT, de esta forma el sistema queda corriendo dentro de este lazo indefinidamente. Se puede detener el simulador y chequear los valores de los registros del CPU en el STACK (PILA) para poder ver que estaba haciendo el programa cuando este recibe una interrupción no esperada.

Variables en RAM.

Las variables de un programa cambian de valor durante la ejecución del programa en curso. Estos valores no pueden especificarse antes de que el programa sea escrito y grabado en el MCU. El CPU debe usar instrucciones para inicializar y modificar estos valores. Cuando el programa se escribe, se reserva espacio para las variables en la RAM del MCU, usando directivas de reserva de bytes de memoria (RMB).

Primero, se colocará una directiva originate (ORG) para setear el contador del ensamblador a la dirección de comienzo de la RAM en el MCU ($00C0 en el MC68HC705J1A).
Cada variable o grupo de variables serán seteadas con una directiva RMB. La línea RMB se identifica por el nombre de la variable. El ensamblador le asigna el nombre (etiqueta) al próximo espacio de memoria disponible. Después de que cada variable o grupo de variables es asignada, el contador de localización se avanza a la próxima posición libre de memoria.

Como se podrá ver en variados casos, algunos programadores, sostienen que es buena práctica limpiar todas las posiciones de memoria RAM utilizadas, como un de los primeros pasos de inicialización luego de salir de la condición de reset. Esto puede ser útil, cuando en nuestro proceso de depuración de código de programa, queremos comprobar la escritura o no de una posición de memoria RAM por procesos propios del programa, ya que los nuevos valores serán seguramente distintos de cero y por eso es sencillo observar que posiciones de memoria han variado.

Lazo de Paso (Paced Loop).

El "lazo de paso" es una estructura de software de uso general que es utilizable para una gran variedad de aplicaciones con MCUs. La idea principal, es dividir la aplicación en una serie de tareas como por ejemplo, mantener un seguimiento del tiempo, leer entradas al sistema, y actualización de las salidas del mismo. Cada tarea se escribirá como una subrutina. Un lazo principal se construye fuera de las instrucciones a los saltos a subrutina (JSR) para cada tarea. En lo alto del lazo habrá un "derivador" (pacemaker) de software.

Cuando el derivador se dispara, la lista de subrutinas de tareas se ejecuta de a una por vez y una instrucción de derivación nos ubicará en lo alto del lazo a la espera del próximo disparo del derivador.
La figura 7-1 nos muestra un diagrama de flujo para el lazo de paso principal. El bloque superior es un lazo que espera por el disparo del derivador (cada 100 milisegundos). Los siguientes bloques tienen que efectuar el mantenimiento del contador de TIC.
La versión de este programa mostrada en el listado 7-1 tiene dos simples tareas principales, TIME (tiempo) y BLINK (destello). El lector puede remover una o ambas tareas de este lazo de paso y sustituirlas por las propias. La única limitación en el número de tareas principales es que todas deben terminar rápidamente de modo que el derivador no pierda de "disparar" a alguna de ellas. El último bloque en el diagrama de flujo es justamente un salto a la parte superior del lazo donde se esperará un nuevo disparo del derivador.

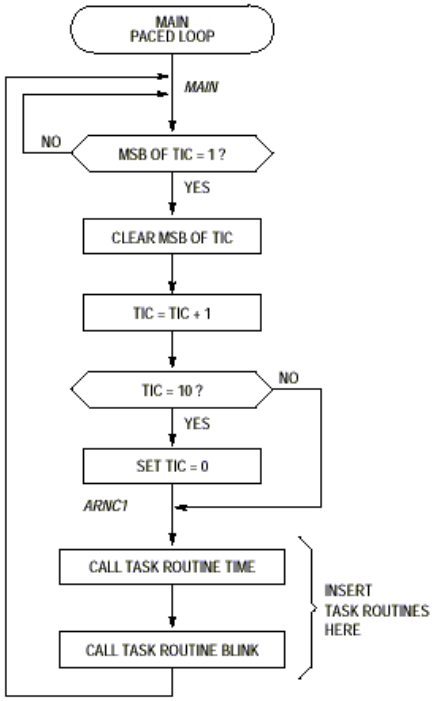


Fig. 7-1. Diagrama de Flujo del Lazo de Paso principal.

Disparador del Lazo.

En el programa de lazo de paso del listado 7-1, el derivador se basa en una interrupción de tiempo real (RTI). Esta RTI está ajustada para generar una interrupción al CPU cada 32,8 milisegundos. El diagrama de flujo de la figura 7-2, muestra que sucede en cada interrupción RTI. Esta actividad de la interrupción puede pensarse como si esta sucediera en forma asincrónica con respecto al programa principal. El bit significativo de la variable se usa como bandera (flag) para decirle al programa principal cuando es tiempo de incrementar la variable TIC y ejecutar una vuelta a través del programa de lazo de paso.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

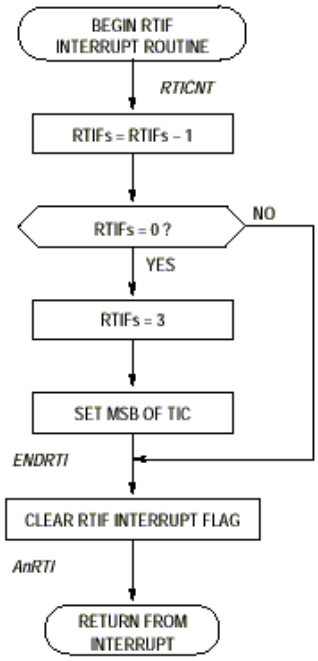


Fig. 7-2. Diagrama de Flujo de la Rutina de Servicio de la Interrupción RTI.

La variable en RAM **RTIFs** se usa para contar 3 interrupciones de tiempo real antes de setear el bit más significativo (MSB) de la variable TIC. El programa principal estará vigilando la variable TIC para ver cuando el MSB se pone en "1".

Cada 32,8 milisegundos el flag (bandera) de RTIF se pondrá en "1", disparando de esta forma un pedido de interrupción. Uno de los usos de una rutina de servicio de interrupción, es limpiar el flag que causa la interrupción antes de que se retorne desde la interrupción.
Si RTIF no es limpiado antes de retornar, un nuevo pedido de interrupción se genera inmediatamente en lugar de esperar los 32,8 milisegundos para el nuevo disparo.

Reloj del sistema de lazo (System Clock).

La variable "TIC" es el reloj más básico para el derivador. TIC cuenta de 0 a 10. Como TIC se incrementa de 9 a 10, el programa reconoce ello y resetea TIC a 0. Excepto dentro del derivador en si mismo, TIC aparece contando desde 0 a 9. TIC es 0 cada décimo pulso de disparo del derivador.

La primera subrutina de tareas en el lazo principal se llama "TIME" .
Esta rutina hace el mantenimiento de un reloj más lento llamado "TOC" . TOC se incrementa cada vez que el lazo de paso se ejecuta y TIC es 0. TOC es seteado como un contador de software que cuenta desde 0 a 59. Las restantes rutinas de tareas después de TIME pueden usar los valores corrientes de TIC y TOC para decidir que se necesita hacer en cada pasada del lazo de paso.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

Vuestro Programa.

Hay muy pocas restricciones en la implementación de subrutinas de tareas. Cada subrutina de tarea deberá realizar todas las cosas que necesite hacer, tan rápidamente como sea posible, y luego ejecutar un Retorno desde Subrutina (RTS). El tiempo total requerido para ejecutar una pasada a través de todas las subrutinas de tareas deberá ser menor que dos disparos del derivador. El punto importante es que la subrutina de tarea no espere por la ocurrencia o no de algún evento externo tal como un interruptor o pulsador que deba ser oprimido. Esto conspiraría con el mantenimiento de los tiempos dentro del lazo de paso.

El lazo de paso puede proporcionar el sensado "anti-rebote" del pulsador. Es importante implementar rutinas de "filtrado" de los estados de un pulsador para evitar el "rebote" (múltiples pasos abierto-cerrado) típico de estos. Para un microcontrolador es sencillo implementar rutinas de supervisión de estados, ya que puede realizar esta tarea rutinaria dentro de otra rutina principal. Existen soluciones de hardware (circuitos anti-rebote) que podrían implementarse, pero requerirían de circuitería adicional y mayor costo en el producto final.

Consideraciones de Tiempo.

Idealmente se deberían terminar todas las rutinas dentro del lazo de paso, antes que el próximo disparo del derivador arribe. Si en una sola pasada por el lazo de paso, alguna rutina toma más tiempo que el conveniente haciendo que el disparo del derivador sea superado, el flag (bandera) que indica que es tiempo para comenzar una nueva pasada por el lazo principal estará continuamente prendido (seteado) cuando se regrese a la cima del lazo. Nada malo pasará si se regresa antes que un nuevo disparo del derivador tenga efecto, ya que se podrá reconocer el pedido anterior siempre y cuando no se supere los dos pedidos de disparo consecutivos dentro del lazo.

Consideraciones de la Pila (Stack).

Pequeños controladores como el MC68HC705J1A tienen poca RAM para la pila (stack) y las variables de programa. Las interrupciones toman 5 Bytes del stack en RAM y cada subrutina toma otros 2 bytes en el stack. Si una subrutina llama a otra subrutina, y una interrupción es requerida antes que la segunda subrutina fuera finalizada, el stack usaría 2+2+5=9 Bytes de RAM de los 64 disponibles en este pequeño controlador.
Si el stack se torna más profundo (más variables de distintas subrutinas e interrupciones son guardadas en el) se corre el riesgo de corromper (sobre-escribir) los datos de otras variables del sistema almacenadas en RAM.
Para evitar este problema, se deberá calcular el peor caso de "profundidad" del stack y la suma de ello más todas las variables utilizadas en el sistema deben ser inferior a la cantidad total de RAM disponible.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

A continuación, se presentará un programa de trabajo que puede utilizarse como plataforma para todos nuestros futuros trabajos de práctica.
En el podremos encontrar:

- Sentencias Equate para todos los registros y nombres de bits del MC68HC705J1A.
- Sentencias Equate específicamente para la aplicación.
- Sección de Variables de Programa.
- Sección de Inicialización (START).
- Derivador para el programa principal basado en interrupciones del tipo RTI.
- Llamados a Subrutinas de tareas.
- Dos ejemplos muy sencillos de subrutinas de tareas (TIME y BLINK).
- Una rutina de Servicio de Interrupción (para interrupciones RTIF).
- Sección de definición de vectores.

Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
$BASE    10T
*********************************************************
* Equates for MC68HC705J1A MCU
* Use bit names without a dot in BSET..BRCLR
* Use bit name preceded by a dot in expressions such as
*  #.ELAT+.EPGM to form a bit mask
*********************************************************
PORTA     EQU     $00         ;I/O port A
PA7       EQU     7           ;Bit #7 of port A
PA6       EQU     6           ;Bit #6 of port A
PA5       EQU     5           ;Bit #5 of port A
PA4       EQU     4           ;Bit #4 of port A
PA3       EQU     3           ;Bit #3 of port A
PA2       EQU     2           ;Bit #2 of port A
PA1       EQU     1           ;Bit #1 of port A
PA0       EQU     0           ;Bit #0 of port A
PA7.      EQU     $80         ;Bit position PA7
PA6.      EQU     $40         ;Bit position PA6
PA5.      EQU     $20         ;Bit position PA5
PA4.      EQU     $10         ;Bit position PA4
PA3.      EQU     $08         ;Bit position PA3
PA2.      EQU     $04         ;Bit position PA2
PA1.      EQU     $02         ;Bit position PA1
PA0.      EQU     $01         ;Bit position PA0

PORTB     EQU     $01         ;I/O port B
PB5       EQU     5           ;Bit #5 of port B
PB4       EQU     4           ;Bit #4 of port B
PB3       EQU     3           ;Bit #3 of port B
PB2       EQU     2           ;Bit #2 of port B
PB1       EQU     1           ;Bit #1 of port B
PB0       EQU     0           ;Bit #0 of port B
PB5.      EQU     $20         ;Bit position PB5
PB4.      EQU     $10         ;Bit position PB4
PB3.      EQU     $08         ;Bit position PB3
PB2.      EQU     $04         ;Bit position PB2
PB1.      EQU     $02         ;Bit position PB1
PB0.      EQU     $01         ;Bit position PB0

DDRA      EQU     $04         ;Data direction for port A
DDRA7     EQU     7           ;Bit #7 of port A DDR
DDRA6     EQU     6           ;Bit #6 of port A DDR
DDRA5     EQU     5           ;Bit #5 of port A DDR
DDRA4     EQU     4           ;Bit #4 of port A DDR
DDRA3     EQU     3           ;Bit #3 of port A DDR
DDRA2     EQU     2           ;Bit #2 of port A DDR
DDRA1     EQU     1           ;Bit #1 of port A DDR
DDRA0     EQU     0           ;Bit #0 of port A DDR
DDRA7.    EQU     $80         ;Bit position DDRA7
DDRA6.    EQU     $40         ;Bit position DDRA6
DDRA5.    EQU     $20         ;Bit position DDRA5
DDRA4.    EQU     $10         ;Bit position DDRA4
DDRA3.    EQU     $08         ;Bit position DDRA3
DDRA2.    EQU     $04         ;Bit position DDRA2
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
DDRA1.    EQU     $02         ;Bit position DDRA1
DDRA0.    EQU     $01         ;Bit position DDRA0

DDRB      EQU     $05         ;Data direction for port B
DDRB5     EQU     5           ;Bit #5 of port B DDR
DDRB4     EQU     4           ;Bit #4 of port B DDR
DDRB3     EQU     3           ;Bit #3 of port B DDR
DDRB2     EQU     2           ;Bit #2 of port B DDR
DDRB1     EQU     1           ;Bit #1 of port B DDR
DDRB0     EQU     0           ;Bit #0 of port B DDR
DDRB5.    EQU     $20         ;Bit position DDRB5
DDRB4.    EQU     $10         ;Bit position DDRB4
DDRB3.    EQU     $08         ;Bit position DDRB3
DDRB2.    EQU     $04         ;Bit position DDRB2
DDRB1.    EQU     $02         ;Bit position DDRB1
DDRB0.    EQU     $01         ;Bit position DDRB0

TSCR      EQU     $08         ;Timer status & control reg
TOF       EQU     7           ;Timer overflow flag
RTIF      EQU     6           ;Real time interrupt flag
TOIE      EQU     5           ;TOF interrupt enable
RTIE      EQU     4           ;RTI interrupt enable
TOFR      EQU     3           ;TOF flag reset
RTIFR     EQU     2           ;RTIF flag reset
RT1       EQU     1           ;RTI rate select bit 1
RT0       EQU     0           ;RTI rate select bit 0
TOF.      EQU     $80         ;Bit position TOF
RTIF.     EQU     $40         ;Bit position RTIF
TOIE.     EQU     $20         ;Bit position TOIE
RTIE.     EQU     $10         ;Bit position RTIE
TOFR.     EQU     $08         ;Bit position TOFR
RTIFR.    EQU     $04         ;Bit position RTIFR
RT1.      EQU     $02         ;Bit position RT1
RT0.      EQU     $01         ;Bit position RT0

TCR       EQU     $09         ;Timer counter register

IRSCR     EQU     $0A         ;IRQ status & control reg
IRQE      EQU     7           ;IRQ edge/edge-level
IRQF      EQU     3           ;External interrupt flag
IRQR      EQU     1           ;IRQF flag reset

PDRA      EQU     $10         ;Pulldown register for port A
PDIA7     EQU     7           ;Pulldown inhibit for PA7
PDIA6     EQU     6           ;Pulldown inhibit for PA6
PDIA5     EQU     5           ;Pulldown inhibit for PA5
PDIA4     EQU     4           ;Pulldown inhibit for PA4
PDIA3     EQU     3           ;Pulldown inhibit for PA3
PDIA2     EQU     2           ;Pulldown inhibit for PA2
PDIA1     EQU     1           ;Pulldown inhibit for PA1
PDIA0     EQU     0           ;Pulldown inhibit for PA0
PDIA7.    EQU     $80         ;Bit position PDIA7
PDIA6.    EQU     $40         ;Bit position PDIA6
PDIA5.    EQU     $20         ;Bit position PDIA5
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
PDIA4.    EQU     $10         ;Bit position PDIA4
PDIA3.    EQU     $08         ;Bit position PDIA3
PDIA2.    EQU     $04         ;Bit position PDIA2
PDIA1.    EQU     $02         ;Bit position PDIA1
PDIA0.    EQU     $01         ;Bit position PDIA0

PDRB      EQU     $11         ;Pulldown register for port B
PDIB5     EQU     5           ;Pulldown inhibit for PB5
PDIB4     EQU     4           ;Pulldown inhibit for PB4
PDIB3     EQU     3           ;Pulldown inhibit for PB3
PDIB2     EQU     2           ;Pulldown inhibit for PB2
PDIB1     EQU     1           ;Pulldown inhibit for PB1
PDIB0     EQU     0           ;Pulldown inhibit for PB0
PDIB5.    EQU     $20         ;Bit position PDIB5
PDIB4.    EQU     $10         ;Bit position PDIB4
PDIB3.    EQU     $08         ;Bit position PDIB3
PDIB2.    EQU     $04         ;Bit position PDIB2
PDIB1.    EQU     $02         ;Bit position PDIB1
PDIB0.    EQU     $01         ;Bit position PDIB0

EPROG     EQU     $18         ;EPROM programming register
ELAT      EQU     2           ;EPROM latch control
MPGM      EQU     1           ;MOR programming control
EPGM      EQU     0           ;EPROM program control
ELAT.     EQU     $04         ;Bit position ELAT
MPGM.     EQU     $02         ;Bit position MPGM
EPGM.     EQU     $01         ;Bit position EPGM

COPR      EQU     $07F0       ;COP watchdog reset register
COPC      EQU     0           ;COP watchdog clear
COPC.     EQU     $01         ;Bit position COPC

MOR       EQU     $07F1       ;Mask option register
SOSCD     EQU     7           ;Short osc delay enable
EPMSEC    EQU     6           ;EPROM security
OSCRES    EQU     5           ;Oscillator parallel resistor
SWAIT     EQU     4           ;STOP instruction mode
PDI       EQU     3           ;Port pulldown inhibit
PIRQ      EQU     2           ;Port A IRQ enable
LEVEL     EQU     1           ;IRQ edge sensitivity
COP       EQU     0           ;COP watchdog enable
SOSCD.    EQU     $80         ;Bit position SOSCD
EPMSEC.   EQU     $40         ;Bit position EPMSEC
OSCRES.   EQU     $20         ;Bit position OSCRES
SWAIT.    EQU     $10         ;Bit position SWAIT
PDI.      EQU     $08         ;Bit position PDI
PIRQ.     EQU     $04         ;Bit position PIRQ
LEVEL.    EQU     $02         ;Bit position LEVEL
COPEN.    EQU     $01         ;Bit position COPEN

* Memory area equates
RAMStart  EQU     $00C0       ;Start of on-chip RAM
ROMStart  EQU     $0300       ;Start of on-chip ROM
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
ROMEnd   EQU     $07CF      ;End of on-chip ROM
Vectors  EQU     $07F8      ;Reset/interrupt vector area

* Application specific equates
LED      EQU     PA7        ;LED ON when PA7 is low (0)
LED.     EQU     PA7.       ;LED bit position
SW       EQU     PA0        ;Switch on PA0, closed=high (1)
SW.      EQU     PA0.       ;Switch bit position

*********************************************************
* Put program variables here (use RMBs)
*********************************************************
         ORG     $00C0      ;Start of 705J1A RAM

RTIFs    RMB     1          ;3 RTIFs/TIC (3-0)
TIC      RMB     1          ;10 TICs make 1 TOC (10-0)
                            ;MSB=1 means RTIFs rolled over
TOC      RMB     1          ;1 TOC=10*96.24ms= about 1 sec

*********************************************************
* Program area starts here
*********************************************************
         ORG     $0300      ;Start of 705J1A EPROM

* First initialize any control registers and variables

START    CLI                ;Clear I bit for interrupts
         LDA     #LED.      ;Configure and turn off LED
         STA     PORTA      ;Turns off LED
         STA     DDRA       ;Makes LED pin an output
         LDA     #(RTIFR.+RTIE.+RT1.)
         STA     TSCR       ;To clear and enable RTIF
                            ;and set RTI rate for 32.8 ms
         LDA     #3         ;RTIFs counts 3->0
         STA     RTIFs      ;Reset TOFS count
         CLR     TIC        ;Initial value for TIC
         CLR     TOC        ;Initial value for TOC
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
*************************************************************
* MAIN - Beginning of main program loop
*        Loop is executed once every 100 ms (98.4 ms)
*        A pass through all major task routines takes
*        less than 100mS and then time is wasted until
*        MSB of TIC set (every 3 RTIFs = 98.4 ms).
*        At each RTIF interrupt, RTIF cleared & RTIFs
*        gets decremented (3->0). When RTIFs = 0, MSB of
*        TIC gets set and RTIFs is set back to 3.
*        (3*32.8/RTIF = 98.4 ms).
*
*        The variable TIC keeps track of 100mS periods
*        When TIC increments from 9 to 10 it is cleared
*        to 0 and TOC is incremented.
*************************************************************
MAIN     CLRA                ;Kick the watch dog
         STA     COPR        ; if enabled
         BRCLR   7,TIC,MAIN  ;Loop here till TIC edge

         LDA     TIC         ;Get current TIC value
         AND     #$0F        ;Clears MSB
         INCA                ;TIC = TIC+1
         STA     TIC         ;Update TIC
         CMP     #10         ;10th TIC ?
         BNE     ARNC1       ;If not, skip next clear
         CLR     TIC         ;Clear TIC on 10th
ARNC1    EQU     *           ;
* End of synchronization to 100 ms TIC; Run main tasks
*  & branch back to MAIN within 100 ms.  Sync OK as long
*  as no 2 consecutive passes take more than 196.8 ms

         JSR     TIME        ;Update TOCs

         JSR     BLINK       ;Blink LED

* Other main tasks would go here

         BRA     MAIN        ;Back to Top for next TIC

** END of Main Loop ************************************

*************************************************************
* TIME - Update TOCs
*  If TIC = 0, increment 0->59
*  If TIC not = 0, just skip whole routine
*************************************************************
TIME     EQU     *           ;Update TOCs
         TST     TIC         ;Check for TIC = zero
         BNE     XTIME       ;If not; just exit
         INC     TOC         ;TOC = TOC+1
         LDA     #60
         CMP     TOC         ;Did TOC -> 60 ?
         BNE     XTIME       ;If not, just exit
         CLR     TOC         ;TOCs rollover
XTIME    RTS                 ;Return from TIME
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar

```
*********************************************************
* BLINK - Update LED
*  If TOC is even, light LED
*     else turn off LED
*********************************************************
BLINK    EQU     *          ;Update LED
         LDA     TOC        ;If even, LSB will be zero
         LSRA               ;Shift LSB to carry
         BCS     LEDOFF     ;If not, turn off LED
         BSET    LED,PORTA  ;Turn on LED
         BRA     XBLINK     ;Then exit
LEDOFF   BCLR    LED,PORTA  ;Turn off LED
XBLINK   RTS                ;Return from BLINK

*********************************************************
* RTIF interrupt service routine
*********************************************************
RTICNT   DEC     RTIPs      ;On each RTIF decrement RTIPs
         BNE     ENDTOF     ;Done if RTIPs not 0
         LDA     #3         ;RTIPs counts 3->0
         STA     RTIPs      ;Reset TOPS count
         BSET    7,TIC      ;Set MSB as a flag to MAIN
ENDTOF   BSET    RTIFR,TSCR  ;Clear RTIF flag
AnRTI    RTI                ;Return from RTIF interrupt


UNUSED   EQU     AnRTI      ;Use RTI at AnRTI for unused
                            ;interrupts to just return

*********************************************************
* Interrupt & reset vectors
*********************************************************
         ORG     $07F8      ;Start of vector area

TIMVEC   FDB     RTICNT     ;Count RTIPs 3/TIC
IRQVEC   FDB     UNUSED     ;Change if vector used
SWIVEC   FDB     UNUSED     ;Change if vector used
RESETV   FDB     START      ;Beginning of program on reset
```


Autor : **Ing. Daniel Di Lella** – Dedicated Distributor Field Application Engineer (D.D.F.A.E.) For Motorola Products – fae@electrocom.com.ar