

# **MICROCONTROLADOR COP8™**

---

Manual de Teoría y Práctica Básica

Literatura Num. XXXXXX-001  
Febrero 2001

## CONTENIDO

---

OBJETIVOS	...	2
CAPITULO 1		
Introducción	...	3
CAPITULO 2		
Microcontrolador COP8	...	17
CAPITULO 3		
Set de Instrucciones	...	25
CAPUTILO 4		
Unidades Básicas del COP8	...	56
CAPITULO 5		
Unidades Especiales	...	81
CAPITULO 6		
Simulador y Emulador	...	97
APENDICE A	...	103
Set de Instrucciones		
APENDICE B	...	104
Configuración de Pines		
APENDICE C	...	107
Esquemáticos		
BIBLIOGRAFÍA	...	108

Al finalizar la lectura y prácticas de esta guía, el lector tendrá la capacidad de identificar la situación donde sea necesaria la implementación de un control con un Microcontrolador.

Así mismo, tendrá la capacidad de acoplar este circuito Microcontrolador a la mayor parte de los periféricos, sensores o actuadores. Como también desarrollar la circuitería necesaria para su correcto desempeño.

Por último, será capaz de crear software en lenguaje ensamblador y con éste, controlar cualquier dispositivo bajo cualquier algoritmo de control que sea requerido implementar.

Se familiarizará con el Microcontrolador COP8 de National Semiconductor y con todas sus herramientas en hardware y software.

## **1.1 MICROCONTROLADORES**

Un Microcontrolador está diseñado para realizar la tarea de muchos circuitos lógicos simplificando el diseño. Su uso es extremadamente popular por su facilidad de implementación y costo. Los pasos necesarios que el usuario tiene que cubrir para desarrollar circuitos con Microcontroladores es determinar las tareas a realizar, escribirlas en un archivo, procesarlo para que después se almacene a la memoria del Microcontrolador.

Un Microcontrolador por definición no tiene una función especial (Como la tendría un amplificador que amplifica una señal, un comparador la compara con otra y un regulador regulará el voltaje.). Es decir, un Microcontrolador es un circuito integrado el cual, de no ser programado no realizará tarea alguna. Requiere de ser programado para que realice desde la tarea mas sencilla hasta el control mas complicado.

La ventaja del uso de los Microcontroladores son muchas y muy diversas. Los circuitos discretos son alambrados permanentemente para realizar una función específica. Si los requerimientos del diseño cambian, es probable que sea necesario rediseñar todo el circuito para ajustar estas nuevas necesidades.

Con un Microcontrolador, la mayoría de los cambios pueden implementarse simplemente reprogramando el dispositivo. Es decir, solo es necesario cambiar un programa y no el circuito lógico.

Las aplicaciones de los microcontroladores son limitadas por la imaginación del usuario, ya que se pueden encontrar en Televisiones, Teclados, Modems, Impresoras, Lavadoras, Teléfonos, Automóviles, Línea Blanca, Unidades de seguridad en la oficina y/o casa, VCRs, juegos de video, etc. Algunas fuentes estadísticas estiman que hoy en día se tienen alrededor de 250 Microcontroladores en una casa típica de EU.

## 1.2 ¿ QUÉ ES UN MICROCONTROLADOR ?

Un Microcontrolador es un circuito integrado de muy alta escala de integración el cual contiene tres unidades básicas que lo identifican como tal y son *CPU* para procesar la información, *Memoria de datos* para guardar información y *Memoria de Programa* para almacenar las instrucciones.

### **CPU**

La Unidad de Procesamiento Central es el corazón del Microcontrolador y es aquí donde todas las operaciones aritméticas y lógicas son realizadas. Es decir, es la unidad que calcula todas las operaciones que son ordenadas por la memoria de programa.

### **Memoria de Programa**

Contiene las intrucciones organizadas en una secuencia particular para realizar una tarea. Típicamente es denominada memoria de sólo lectura (ROM) o también OTP, EPROM o FLASH que son memorias que una vez programadas almacenan la información aunque el sistema no sea energizado. Esto permite que el Microcontrolador ejecute el programa almacenado en Memoria inmediatamente después de ser energizado.

### **Memoria de Datos**

Esta es una memoria que puede ser escrita y leída según sea requerido por el programa. Tiene las funciones de almacenamiento de datos (pila) y como almacenamiento de variables. Este tipo de memoria es usualmente llamada memoria RAM (Memoria de Acceso Aleatorio). Cada localidad de memoria tiene una dirección única con la cual el CPU encuentra la información necesaria.

Los microcontroladores actuales contienen ambas memorias (Datos y Programa) incuídas dentro del circuito integrado. Por otro lado, resulta necesario contar con otras unidades que hacen posible el funcionamiento mínimo de un Microcontrolador que son Circuitería de Temporización y Entradas/Salidas

### **Circuitería de Temporización**

Los Microcontroladores usan señales de temporización llamadas Reloj que proveen una referencia en el tiempo para la ejecución del programa. Esta señal determina en qué momento los datos deben ser escritos o leídos de la memoria. Así mismo, provee la sincronía con los dispositivos conectados al Microcontrolador (Periféricos).

### **Entradas/Salidas**

Los Microcontroladores requieren de una interfase para comunicarse con la circuitería externa. Esta Interfase es denominada comúnmente como Puerto. Existen Puertos de Entrada y Salida los cuales permiten que las señales (o datos) sean leídos del exterior o mandados al exterior del Microcontrolador. Los Puertos estan formados de pines, (terminales del circuito intergrado) los cuales, dependiendo de la aplicacion, son conectados a un sin fin de dispositivos como teclados, interruptores, sensores, relevadores, motores, etc.

### **1.3 MICROCONTROLADOR vs MICROPROCESADORES**

Es importante distinguir las diferencias entre ambos circuitos integrados para poder identificar cuándo es conveniente emplearlos. Los Microcontroladores generalmente tienen una arquitectura con un bus dual (Es decir, un bus dedicado para la memoria de programa y otro para la memoria de datos) Mientras que en los Microprocesadores es común encontrar la arquitectura Von Neumann (un solo bus para la memoria de datos y programa).

Para aplicaciones de control, los Microcontroladores generalmente son más eficientes en el manejo de memoria dado que su set de instrucciones es más pequeño y fácil de manejar que el de un Microprocesador. También es común encontrar soluciones implementadas con Microcontroladores con un solo circuito integrado, mientras que los Microprocesadores requieren de más circuitería (Puentes, Controladores de Memoria, Controlador de Periféricos, Memorias, etc).

Internamente la diferencia es mas notable en velocidad, longitud de datos, unidad de procesamiento lógico, manejo de intrucciones así como memoria de Datos y de Progrmama. Afortunadamente gracias al avance tecnológico de los Microcontroladores estas diferencias se hacen cada vez más pequeñas.

### **1.4 ARQUITECTURA**

Los Microcontroladores pueden identificarse por su arquitectura ya sea arquitectura Vonn Neumann o arquitectura Harvard.

## **Arquitectura Von Neumann**

John Von Neumann fue quien ideó una arquitectura característica con el CPU y la memoria interconectada por un bus de direcciones y datos común. Hay aspectos positivos en esta configuración como los accesos a tablas almacenadas en ROM y un set de instrucciones más ortogonal. El bus de direcciones es usado para identificar qué localidad de memoria está siendo accesada, mientras que el bus de datos es utilizado para trasladar información entre el CPU y alguna localidad de memoria o viceversa.

En las primeras computadoras el almacenamiento del programa era completamente diferente al almacenamiento de los datos. Con un solo bus, la arquitectura Von Neumann es usada secuencialmente para acceder instrucciones de la memoria de programa y ejecutarlas regresando desde/hacia la memoria de datos. Esto significa que el ciclo de instrucción no puede traslaparse con algún acceso a la memoria de datos.

La principal ventaja de la arquitectura Von Neumann es que se tiene un bus de direcciones y de datos uniendo la memoria con el CPU. Una desventaja podría ser que el apuntador de programa o algún otro registro se corrompiera y apuntara a la memoria de datos y se tomara ésta momentáneamente como memoria de programa. Consecuentemente se ejecutaría una instrucción no deseada o un error en la decodificación de la instrucción.

## **Arquitectura Harvard**

La Arquitectura Harvard fue desarrollada en Harvard por Howard Aiken, otro pionero en las computadoras. Esta arquitectura se caracteriza por tener buses separados para la memoria de Programa y la memoria de Datos. Una de las ventajas de la arquitectura Harvard es que la operación del Microcontrolador puede ser controlada más fácilmente si se presentara una anomalía en el apuntador de programa. Existe otra arquitectura que permite accesos a tablas de datos desde la memoria de programa. Esta arquitectura es llamada Arquitectura Harvard Modificada.

Esta última arquitectura es la dominante en los microcontroladores actuales ya que la memoria de programa es usualmente ROM, OTP, EPROM o FLASH mientras que la memoria de datos es usualmente RAM.

Consecuentemente, las tablas de datos pueden estar en la memoria de programa sin que sean perdidas cada vez que el sistema es apagado. Otra ventaja importante en la arquitectura Harvard Modificada es que las transferencias de datos pueden ser translapadas con los ciclos de decodificación de instrucciones. Esto quiere decir que la siguiente instrucción puede ser cargada de la memoria de programa mientras se está ejecutando una instrucción interviniendo la memoria de datos. La desventaja de la arquitectura Harvard Modificada podría ser que se requieren instrucciones especiales para acceder valores en RAM y ROM haciendo la programación un poco complicada.

## 1.5 DIAGRAMA DE UN MICROCONTROLADOR

Para entender la configuración básica del Microcontrolador, describiremos una solicitud del CPU de un dato localizado en Memoria: El CPU solicita información a la memoria (o un Puerto) por un llamado con la dirección correspondiente a la localidad donde se encuentre el dato solicitado. La dirección (de la información solicitada) es almacenada en el CPU como un número binario en un registro temporal. Las salidas de este registro son mandadas por muchas vías (o una vía sencilla) a la memoria del Microcontrolador y periféricos. Al grupo de vías de comunicación que comparten una trayectoria común en forma paralela se le denomina **“BUS”**.

Es entonces cuando el registro de dirección almacena el dato recibido. El número de bits recibidos dependen del tipo de Microcontrolador. El dato o la información buscada es mandada al CPU por el bus de datos. El bus de datos es diferente al bus de direcciones ya que el bus de datos sirve únicamente para recibir o mandar datos a memoria o periféricos.

Las señales del bus de direcciones son controladas solamente por el CPU y la información va siempre del CPU a los bloques de memoria. Por otro lado, la información en el bus de datos puede ser de entrada o salida al CPU por medio del registro de datos.

En otras palabras, el bus de datos es bi-direccional y el bus de direcciones es uni-direccional. El ancho de los buses de datos y de direcciones también

pueden ser diferentes, dependiendo del tipo de microcontrolador y del tamaño de la memoria. A continuación se muestra a bloques el CPU, Memorias y Buses.

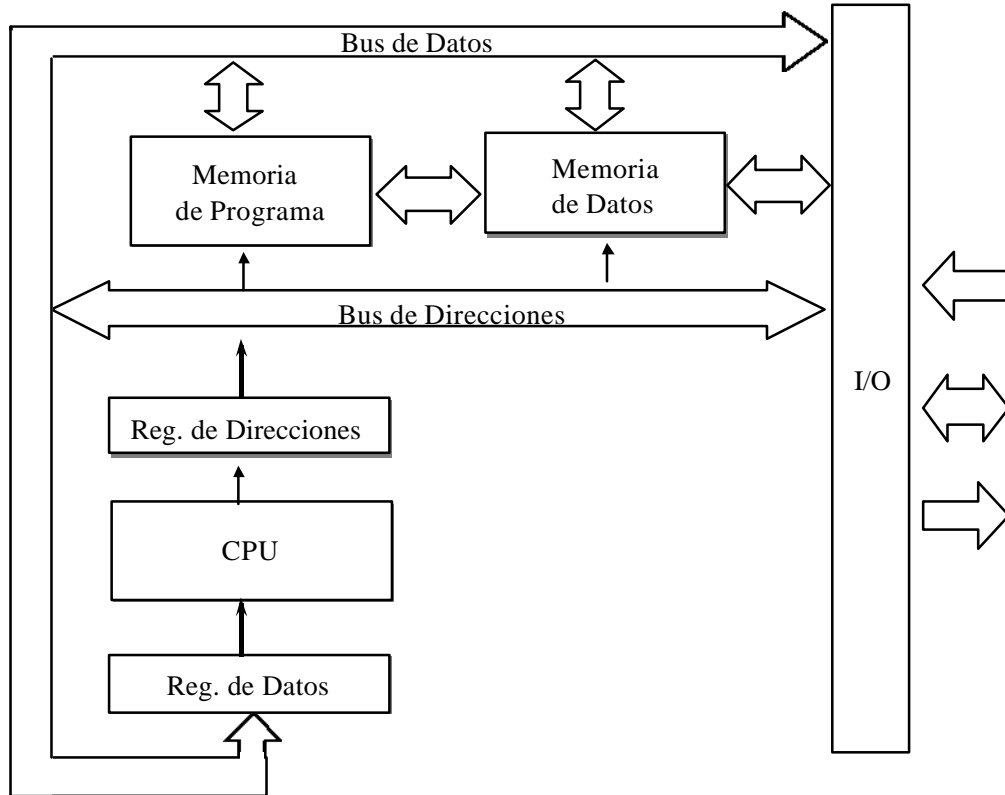


Figura 1. Diagrama a Bloques de un Microcontrolador

## 1.6 MEMORIA DE PROGRAMA

La Memoria de Programa contiene el programa del Microcontrolador. Es decir es aquí donde se almacenan los comandos a ejecutar por el CPU para realizar una tarea de control determinados por el usuario dependiendo de los requerimientos de la aplicación. Existen muchos tipos de memoria de programa ROM, EPROM, OTP, EEPROM, FLASH.

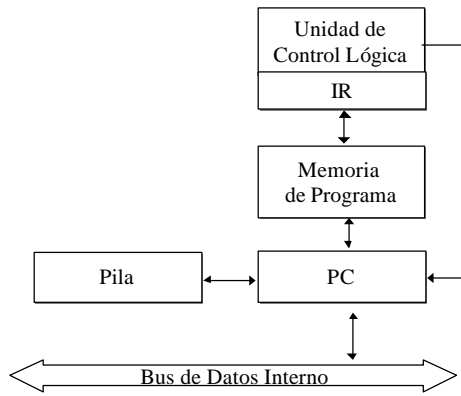


Figura 2. Diagrama de la Memoria de Programa

Existe un registro que se identifica como PC (Program Counter) o Contador de Programa.

El PC es requerido por el CPU para apuntar a la localidad de memoria que contiene las instrucciones del programa.

Cada vez que un Opcode (Representación binaria de una instrucción) es leído de la memoria, el registro PC es incrementado en uno para apuntar a la siguiente instrucción a ser ejecutada. Cada vez que el CPU realiza una operación de decodificación de instrucción el registro PC es incrementado en uno, por lo tanto, el registro PC siempre apunta a la siguiente instrucción por ser ejecutada. La longitud del registro PC es determinante en el tamaño de la Memoria de Programa. Ya que no se podría apuntar con el registro PC de 8 bits una Memoria de Programa de 1Kbytes.

## 1.7 MEMORIA DE DATOS

La Memoria de datos es utilizada para guardar y otorgar información. Típicamente, hay dos tipos de memoria de datos que pueden ser empleadas : RAM y EEPROM (Memoria Electricamente Borrable).

### Estructura

1. Tamaño: La memoria de datos varía en tamaño dependiendo del Microcontrolador y/o la familia. (4-bit/8-bit/16-bit por 16, 32, 64, 128 bytes, etc.).
2. Arquitectura Von Neumann: Con la arquitectura Von Neumann la memoria de programa y de datos comparten el mismo espacio de memoria. Si el código es colocado en un espacio de memoria separado (0000 a la 03FF), entonces los datos deberán recibir más allá de las localidades 03FF Hex.

3. Arquitectura Harvard: Con la arquitectura Harvard la memoria de programa y de datos se encuentran en espacios separados. El Código puede ser colocado en las localidades 0000 a la 03FF Hex y los datos pueden ser también localizados en las localidades 000 a la 03F Hex.

### **Almacenamiento de Datos**

La memoria de datos puede contener valores de resultados provenientes de operaciones matemáticas como sumas, restas, tablas, banderas y pila del sistema.

### **Apuntador**

Un apuntador es un registro que contiene una dirección en la cual se especifica una localidad de memoria. Es decir, un apuntador “apunta” a la localidad de algún dato. El registro apuntador de datos (PTR) es cargado con la dirección donde se encuentre la información de interés. Para acceder un byte de datos, el registro apuntador puede ser usado en lugar de usar la dirección propia.

Esto es particularmente útil, cuando se encuentran localidades consecutivas y todas ellas van a ser accedidas. El apuntador puede ser incrementado o decrementado automáticamente después de un acceso a RAM en lugar de hacer referencia en cada momento a las direcciones de RAM

### **STACK (Pila)**

La sección de pila es simplemente memoria de datos dedicada a almacenar datos y direcciones en forma de pila (consecutivamente). La pila opera de forma LIFO (último en entrar, primero en salir).

Es necesario contar con un apuntador a esta sección de memoria llamado registro apuntador a pila, dado que es este registro el utilizado para mantener la dirección de la última entidad (dato o dirección) almacenada. Usualmente el puntero a pila es inicializado apuntando a una localidad de memoria.

De acuerdo a como se requiera guardar o sacar los datos de la pila, el registro apuntador de pila también se mueve hacia arriba o hacia abajo. El Microcontrolador utiliza la pila par almacenar o recuperar una dirección durante un llamado a una subrutina o una interrupción.

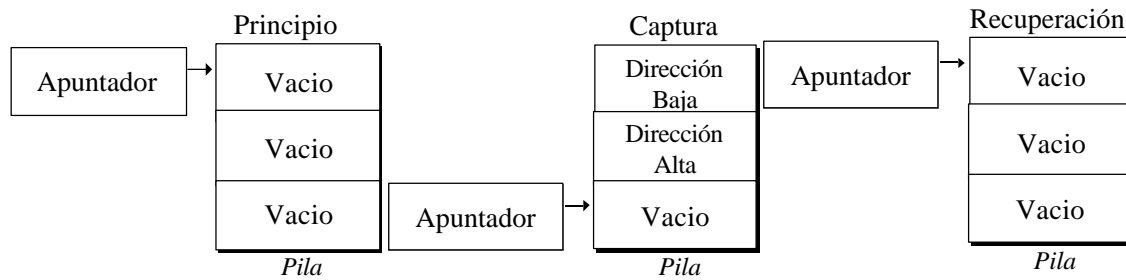


Figura 3. Almacenamiento y Recuperación en Pila de una Dirección de 2 Bytes.

Algunos Microcontroladores también almacenan información de status del controlador en la pila antes de responder a alguna interrupción. El programa puede utilizar la pila para guardar datos temporalmente, especialmente cuando se desea almacenar datos entre subrutinas. No todas las pilas de los Microcontroladores son accesibles al usuario.

## 1.8 EL CPU DEL MICROCONTROLADOR

La función clave del CPU es realizar las tareas de Fetch (cargado), Decode (decodificación), Execute (ejecución).

**Fetch:** La dirección de memoria de programa que se encuentra almacenada en el registro apuntador (PC) es utilizada para capturar la instrucción localizada en ésta dirección. La instrucción es copiada al registro de instrucciones (IR).

El registro PC es incrementado para apuntar a la siguiente instrucción disponible.

**Decode:** La instrucción localizada en el registro IR es decodificada. Es decir la representación en bits en el registro IR especifican determinada acción y es entonces cuando se generan señales de control y ajuste para preparar la ejecución de la instrucción.

**Execute:** Las señales de control se distribuyen por todo el Microcontrolador, causando que la acción deseada se realice.

### ALU; Unidad Aritmética Lógica (Arithmetic Logic Unit)

La Unidad Aritmética Lógica es un sumador binario. Es aquí donde se realizan todas las operaciones lógicas y aritméticas en el Microcontrolador. El resultado de todas estas operaciones es usualmente alojado en un acumulador. En especial el COP8 identifica a este registro de ocho bits como Acumulador (A).

El Acumulador siempre está involucrado en cualquier operación aritmética o lógica. Es decir, cuando se tiene una suma o resta u operación lógica AND, etc., un comando se encontrará necesariamente en (A) y el otro operando podrá ser un dato inmediato o algún dato localizado en Memoria.

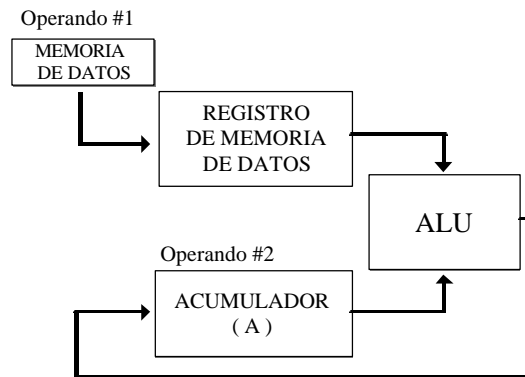


Figura 4. Suma de 2 Operandos

Algunas veces, hay un tercer operando en las operaciones del ALU, este operando es de 1 bit y es llamado Carry (C).

### Registros de Control/Status

Los registros de control y status son registros de propósito especial usados para almacenar el estado actual del Microcontrolador. Los bits de control son manipulados por el programa de usuario para llevar al controlador y sus periféricos a algún estado en particular. Los bits de status son monitoreados por el programa para informar al Microcontrolador/usuario del estado actual. Algunos ejemplos son el bit de Carry, el bit habilitador de interrupciones, el bit de modo de ahorro de energía, etc.

## 1.9 TEMPORIZACIÓN

El Microcontrolador utiliza el tiempo del ciclo de instrucción como referencia de temporización interna. El tiempo de ciclo de instrucción es la cantidad de tiempo que se toma para que una instrucción sea cargada (fetch), decodificada (decode) y ejecutada (executed). Este tiempo difiere entre algunas instrucciones y entre Microcontroladores. Normalmente los fabricantes de Microcontroladores especifican un mínimo de tiempo en el ciclo de instrucción. Por ejemplo, el COP8SA tiene un tiempo de ciclo de instrucción de 1 microsegundo ( $t_c=1\mu s$ ). Esto significa que las instrucciones más rápidas serán ejecutadas en un tiempo de 1 microsegundo con el Microcontrolador operando al máximo de frecuencia. Algunas instrucciones más complicadas usualmente toman un múltiplo del tiempo mínimo de ciclo de instrucción ( $t_c$ ).

El ciclo de instrucción es usualmente resultado de una división de la frecuencia de reloj de entrada al Microcontrolador. Por ejemplo, El COP8SAC7 tiene un divisor de 10. Esto significa que con una frecuencia de entrada de 10MHz se tendrá 1 MHz de reloj de ciclo de instrucción (Un tiempo de ciclo de instrucción de 1 $\mu$ s.) El mínimo del tiempo de ciclo de instrucción puede tomar muchos pulsos de reloj o solamente un pulso, dependiendo del Microcontrolador. Los factores que limitan el tiempo de ciclo de instrucción son :

1. Tiempo de acceso a memoria ( Velocidad de la Memoria).
2. Número de bytes por instrucción.
3. Ancho del bus de datos.
4. Nivel de decodificación requerido por las instrucciones.
5. Tiempo de Ejecución.

### Circuitos Osciladores

Típicamente existen tres opciones de relojes osciladores : Oscilador Externo, Oscilación por Cristal, Oscilación por R/C.

Oscilador Externo: Cualquier señal cuadrada periódica generada externamente puede ser fuente de la unidad de temporización del Microcontrolador. Esta fuente de reloj deberá cumplir con las especificaciones en ciclo de trabajo, tiempos de flancos positivos y negativos y niveles de entrada.

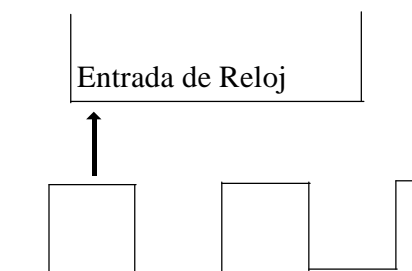


Figura 5. Oscilador Externo

Oscilador R/C: El principio de este oscilador es implementado con una Resistencia y un Capacitor. De tal forma, que se requiere del uso externo de R/C o en su defecto tenerlos integrados dentro del Microcontrolador. La frecuencia de oscilación es una función de la resistencia, la capacitancia y en determinado momento ésta es la oscilación que más se afecta por la temperatura externa.

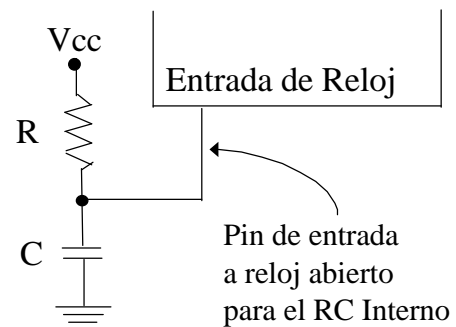


Figura 6. Oscilador R/C

Oscilador por Cristal: Probablemente este sistema de oscilación es el más popular, Tiene una muy buena estabilidad en su señal de reloj, baja impedancia así como muy bajo consumo de potencia. La frecuencia es independiente del voltaje del sistema y temperatura. El circuito mostrado en la Figura 7 debe cumplir con dos criterios.

1. La ganancia en lazo cerrado debe ser  $> 1$ .
2. El margen de fase debe ser aprox. de  $360^\circ$

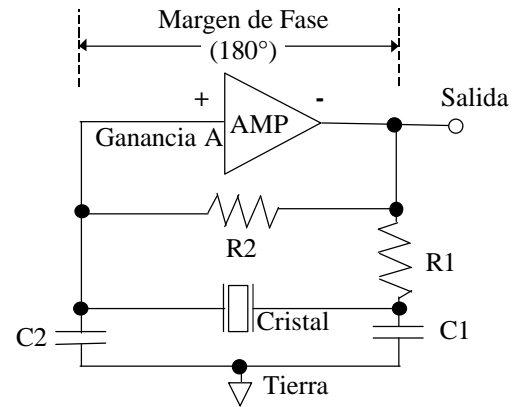


Figura 7. Oscilación por Cristal

El amplificador (integrado en el Chip) provee la ganancia deseada y el desplazamiento de fase de  $180^\circ$ . Los Capacitores externos C1 y C2 proveen los restantes  $180^\circ$  necesarios para el desplazamiento de fase. La resistencia R1 es usada para incertar pérdidas en el lazo y para las oscilaciones en bajas frecuencias (Típicamente menores a 2.5MHz) y previene los armónicos. R2 es utilizado como "bias" del amplificador y para un mejor y más rápido reinicio de las oscilaciones.

## 1.10 ENTRADAS/SALIDAS

Las entradas y salidas permiten que el microcontrolador se comunice con los dispositivos externos al Microcontrolador.

### Entradas

Un dispositivo externo otorga al Microcontrolador una señal en estado alto o bajo. El nivel lógico es leído por el Microcontrolador como un bit sencillo de información de entrada.

### Salidas

El Microcontrolador fuerza uno de sus pines a un estado alto o bajo. El voltaje de salida en el pin corresponde a un bit sencillo de información.

### Puertos

Un puerto es un grupo de pines utilizado para mandar o recibir información. Un puerto puede tener únicamente salidas, entradas o incluso una combinación de pines de entradas y salidas. Actualmente la mayoría de los puertos son bi-

direccionales, es decir pueden ser configurados como pines de entrada o salida dependiendo de los requerimientos del usuario.

Usualmente cada puerto (grupo de pines) tiene asignada una dirección como si fuera un registro en memoria. La escritura a una dirección asignada a un puerto ocasiona que los pines asociados con la dirección del puerto sean forzados a un estado alto o bajo de acuerdo al valor escrito. Si los puertos no son mapeados en memoria, se tendrán instrucciones especiales de Entrada/Salida para accederlos.

## **1.11 SET DE INSTRUCCIONES**

Cada Microcontrolador tiene un set de instrucciones. El usuario puede organizar las instrucciones en un orden lógico para crear un programa. El Microcontrolador ejecuta el programa para realizar una tarea específica. Para comprender completamente un set de instrucciones debemos entender los siguientes conceptos.

### **Opcod**

El Opcode (Operation Code) es un código numérico de la instrucción que representa la operación a ser realizada por el CPU. Es decir, el Opcode es un grupo de bits los cuales le indican al Microcontrolador qué operación en particular hay que realizar. Por ejemplo un 64 podría significar “limpiar el Acumulador”.

### **Mnemonicos**

Los Mnemonicos son nombres asignados a una operación en particular. Cada mnemonico es asociado con un opcode. El usuario puede hacer referencia a una operación por medio de un mnemonico “ADD” en lugar del Opcode “84”. Con el uso de los mnemonicos se hace más fácil escribir un programa. Por ejemplo, el mnemonico “LD” representa la operación “Cargar”.

En conclusión el usuario tendrá que crear una secuencia de tareas para realizar una acción de control. Esta secuencia de tareas será representada por instrucciones (mnemonicos). Luego entonces, es necesario convertir estos mnemonicos a lenguaje “entendible” para el Microcontrolador, por lo tanto, se translada esta secuencia de mnemonicos a una secuencia de Opcodes y datos adicionales.

## **Categorías de Instrucciones**

Podríamos agrupar todo el set de instrucciones en seis grupos que son:

1. Aritméticas/lógicas/recorrimientos
2. Transferencias de control
3. Movimientos en Memoria
4. Manipulación de bits
5. Control de la Pila
6. Condicionales/Prueba

En el capítulo 3 veremos a detalle el set de instrucciones.

## **Mapa de Memoria**

Todo Microcontrolador tiene un “mapa” o lista de correspondencia de las localidades de memoria con los registros de control, puertos, etc. En el caso del COP8 podemos encontrar (a excepción del Acumulador y Program Counter) todos los registros, variables, puertos, etc. en un plano unitario sencillo de la memoria de datos (A esta característica se le llama Mapeo en Memoria). Ver el Mapa de Memoria típico en el Apéndice.

## **Autoevaluación del CAPITULO 1**

1. ¿ Qué ventajas ofrece un Microcontrolador sobre un circuito lógico ?
2. ¿ Qué es un Microcontrolador ?
3. ¿ Cuáles son las unidades básicas de un Microcontrolador y describe c/u ?
4. ¿ Describe la diferencia entre un Microprocesador y un Microcontrolador ?
5. ¿ Qué diferencia existe entre la arquitectura Von Neumann y Harvard ?
6. ¿ Para qué sirve un registro apuntador ?
7. ¿ Qué función tiene el stack (pila) ?
8. ¿ Describe las tres tareas principales del CPU ?
9. ¿ Describe los circuitos osciladores más comunes implementados en los Microcontroladores ?
10. ¿ Qué diferencia hay entre un Opcode y un Mnemonico ?
11. ¿ Define un Bus de datos ?
12. ¿ Explica las ventajas/desventajas de la diferencia en tamaño de los buses de datos y los buses de direcciones ?

## MICROCONTROLADOR COP8

### 2.1 EL MICROCONTROLADOR COP8

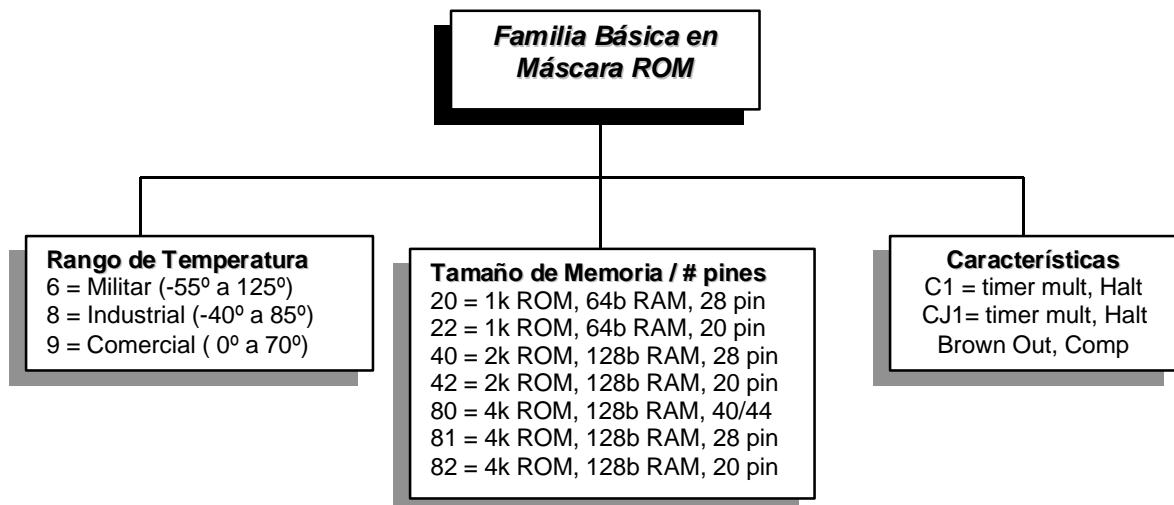
El Microcontrolador COP8 es un circuito integrado fabricado por National Semiconductor. Las siglas COP8 identifican a un **P**rocesador **O**rientado al **C**ontrol de 8 bits.

La familia del COP8 se divide en cuatro grandes grupos que son :

- Familia Básica en Máscara
- Familia Característica en Máscara
- Familia OTP
- Familia S

### 2.2 FAMILIA BÁSICA EN MÁSCARA

Como su nombre lo indica, son Microcontroladores con periféricos integradas de funciones sencillas . Podemos encontrar las siguientes variantes.

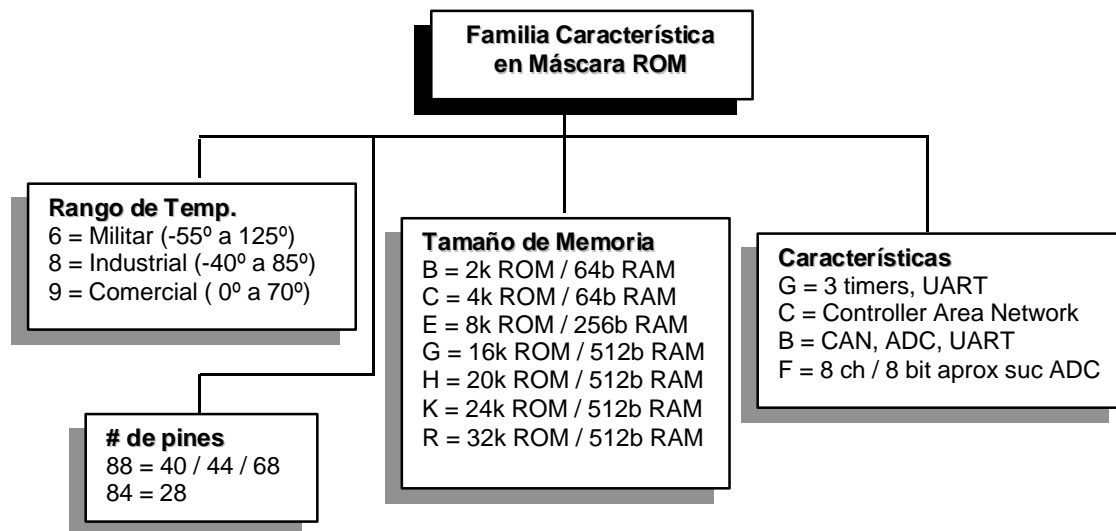


La notación del Microcontrolador se va dando de izquierda a derecha. Por ejemplo, en esta familia tenemos tres características (de Izq a Der) que son Rango de Temperatura, Tamaño de Memoria y # de Pines y por último tenemos la identificación de los periféricos integrados al controlador. Consecuentemente si tenemos un Microcontrolador COP842CJ1 identificaremos de forma inmediata que es un Microcontrolador de rango de operación en temperatura industrial, 2K en memoria ROM, 128 bytes en memoria RAM, 20 pines, un timer multifunciones, modo de ahorro de energía HALT, protección Brown Out y comparadores.

Estos Microcontroladores tienen como principal característica la máscara, es decir, que la compañía fabricante del Microcontrolador aplica en el último proceso de manufactura del integrado una máscara que hace las veces de programación y por lo tanto nos ahorra este último paso. Todo esto es posible, bajo la condicionante de ordenar la manufactura en grandes volúmenes con nuestro programa residente en ROM.

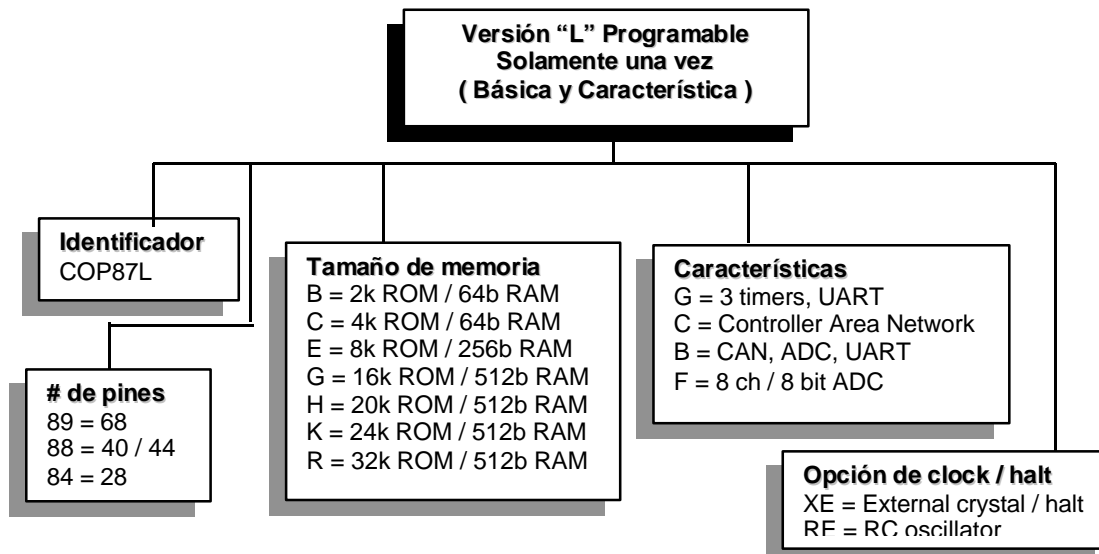
### 2.3 FAMILIA CARACTERÍSTICA EN MÁSCARA

La segunda gran familia de COPs es la Familia Característica o en otras palabras la familia orientada a las Comunicaciones. En esta familia encontramos cuatro variantes de Izq a Der que son Temperatura, Número de Pines, Tamaño de Memoria y por último características en Periféricos. Al igual que la familia básica en máscara, estos Microcontroladores son programados por el fabricante. Aquí podríamos identificar todas las características del COP888RB (Controlador automotriz típico).



## 2.4 FAMILIA OTP

La Familia OTP prácticamente involucra a todos los COP8. Ya que es la versión que se puede programar solamente una vez. Debemos tener especial cuidado con estos Microcontroladores ya que una programación mal realizada o la programación de un software no depurado hacen que el integrado ya no sea útil para realizar la tarea de control deseada. Esta familia tiene cinco variantes de Izq. a Der. Tenemos COP87L88RBXE que es un Microcontrolador de la familia OTP ( identificador COP87L ), puede ser de 40 o 44 pines, 32K en ROM, 512 bytes en RAM, Protocolo CAN y opción de cristal externo. De hecho, el COP87L88RBXE es la versión OTP del COP888RB de la Familia Característica en Máscara.



## 2.5 FAMILIA S

La Familia S contiene los Microcontroladores más recientes y los de más variantes en características eléctricas y físicas (encapsulados). Aquí tenemos cuatro grupos que son :

- Familia COP8SA
- Familia COP8SG
- Familia COP8AC
- Familia COP8SB/Familia COP8CB

## 2.6 FAMILIA COP8SA

El COP8SA es un Microcontrolador de la familia caaracterística de 8 bits y un proceso EPROM de alta densidad, con una gran variedad en encapsulados, variedad en rangos de temperatura y voltaje, etc. Observar las variantes en el diagrama a la izquierda.

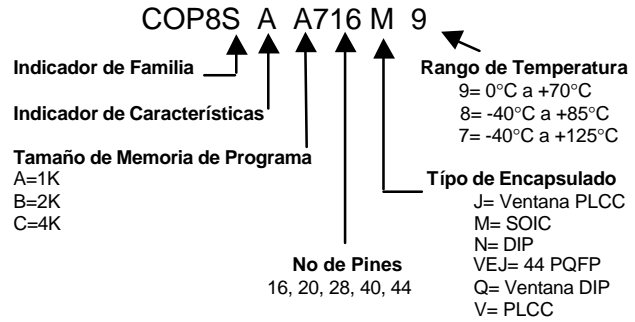


Figura 8. Identificación del COP8SA

La Familia COP8SA se basa en la arquitectura Harvard Modificada, la cual permite que las tablas de datos se accesen directamente desde la memoria de programa. Consecuentemente las tablas de datos pueden estar en ROM o EPROM sin problemas que bajo alguna circunstancia los datos se puedan perder. El Microcontrolador COP8SA es un controlador de 5 puertos de propósito general (D, F, C, G, L), Memorias ROM que van desde 1K hasta los 4K, Memorias RAM desde los 128 Bytes hasta los 64 Bytes, funciones alternativas en algunos pines como “MultiInput Wakeup” orientados a “despertar” al COP8SA de los estados de bajo consumo de potencia. Por otro lado encontramos dos timers T0 y T1. El Timer de 16 bits T1 es capaz de funcionar en varios modos. En la figura se ilustra un diagrama a bloques del COP8SA (Figura 9). De cualquier forma, todas los bloques serán explicados en su funcionamiento a mas detalle en los capítulos siguientes.

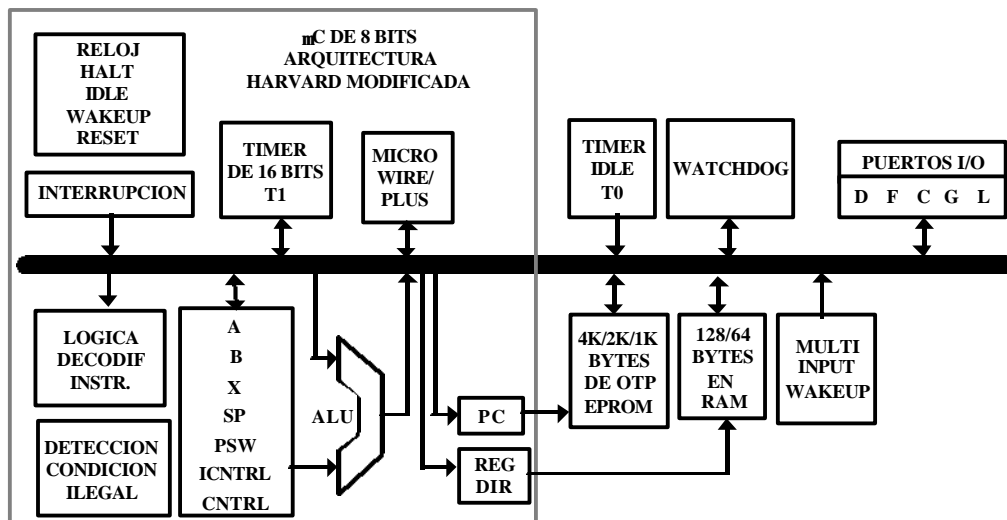


Figura 9. Diagrama del Microcontrolador COP8SA

## 2.6 Familia COP8SG

La Familia COP8SG es muy similar al COP8SA, a diferencia que en esta familia se pueden encontrar memorias ROM de hasta 32K y encapsulados de hasta 44 pines en PQFP. Observar variantes en la Figura 10.

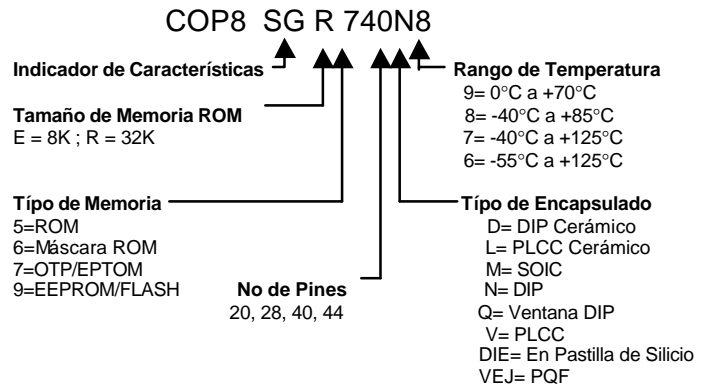


Figura 10. Identificación del COP8SG

La Familia COP8SG es otro miembro de la familia característica de alta integración. Incluye cinco puertos de propósito general en el que algunos pines tienen funciones alternativas. Podemos encontrar hasta un par de comparadores analógicos, una Unidad de comunicación serial USART, Tres Timers de 16 bits totalmente programables con capacidad de operación en tres modos, soporta cristales de hasta 15MHz ( $t_c=0.67\mu s$ ), 14 interrupciones, modos de ahorro de energía y puerto MultiInput Wakeup y encapsulados de 28, 40 y 44 pines. En el diagrama abajo se muestra el COP8SG a bloques.

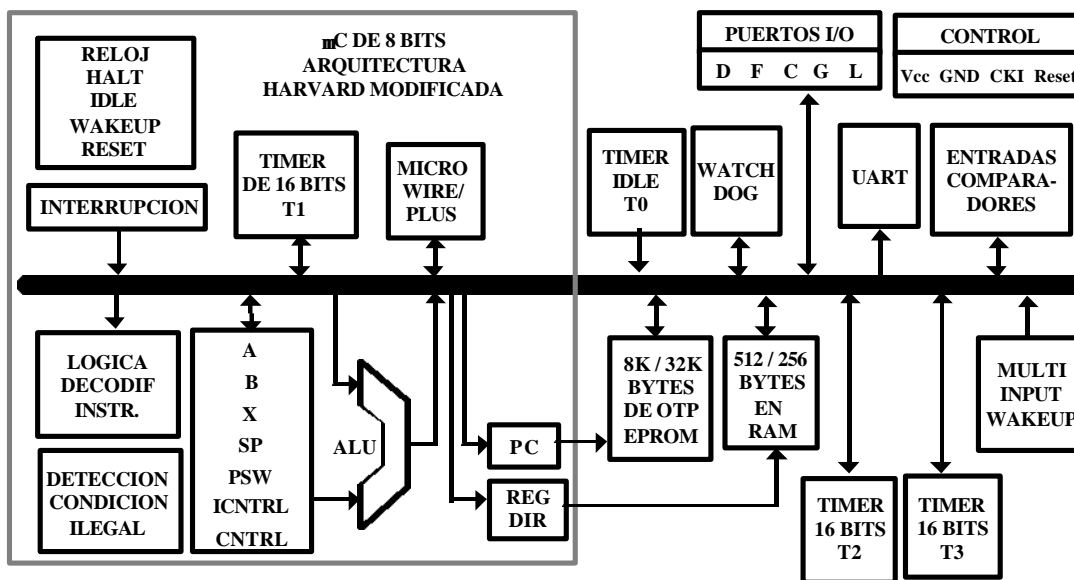


Figura 11. Diagrama del Microcontrolador COP8SG

## 2.7 Familia COP8AC

La Familia del COP8AC identifica a los Microcontroladores más recientes con una unidad de conversión analógica-digital integrada en el Micro. Esta Unidad es de gran versatilidad ya que se puede ajustar en resolución y velocidad.

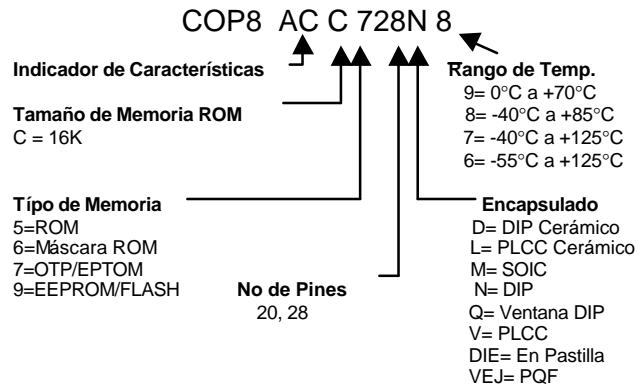


Figura 12. Identificación del COP8AC

La familia COP8AC pertenece a la familia característica de alta densidad, la cual cuenta con una memoria de 16k en ROM, trabaja con cristales externos de 4MHz ( $t_c=2.5\mu s$ ), seis canales de conversión analógica-digital de hasta 12 bits de resolución, un timer de 16 bits multifunciones así como demás unidades comunes como Microwire, modos de ahorro de energía, MultiInput Wakeup, etc. Se tienen versiones de 20 y de 28 pines así como la capacidad de operar en voltajes desde 2.7 hasta 5.5.

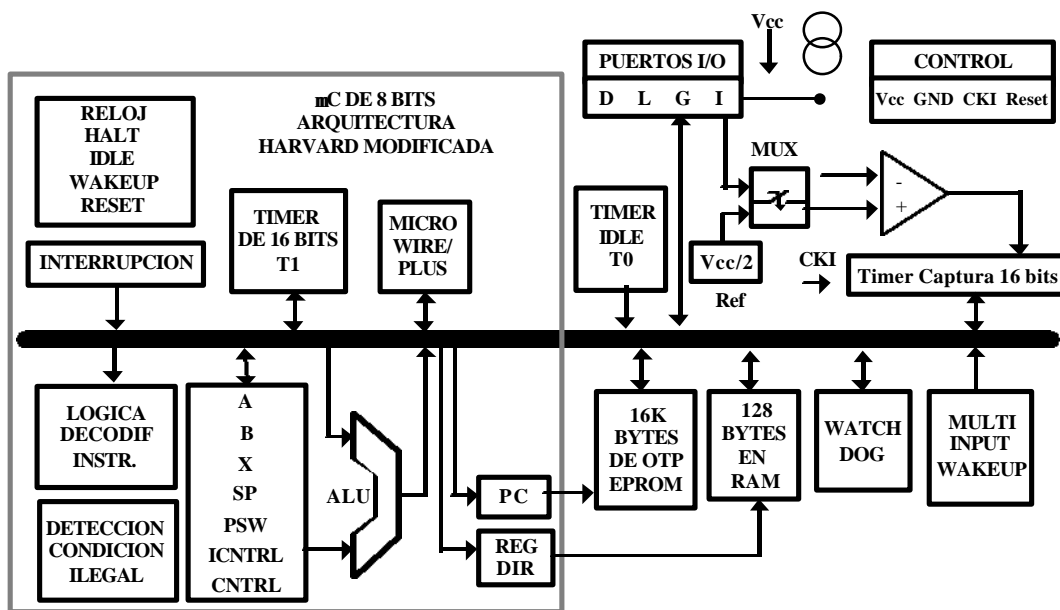


Figura 13. Diagrama del Microcontrolador COP8AC

## 2.8 Familia SB/Familia CB

El Microcontrolador COP8SB o COP8CB es el único con memoria tipo FLASH. Esta familia, al momento, presenta versiones únicamente de 32K en Flash y hasta tres versiones en voltaje de Brownout. Los encapsulados son de 44 y 68 en PLCC.

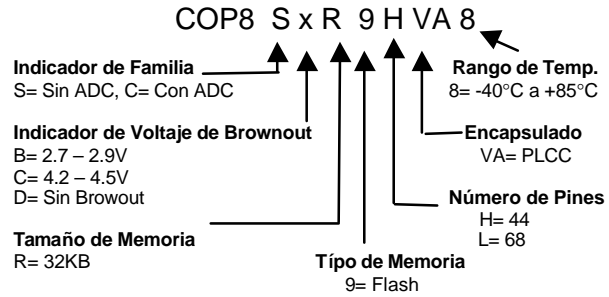


Figura 14. Identificación del COP8SB/COP8CB

Esta familia, al igual que las familias anteriores, pertenece al núcleo de la familia característica de alta escala de integración. La característica particular del COP8CB es el comportamiento de la memoria FLASH, ya que puede actuar como memoria virtual EEPROM (es decir, desde el programa de usuario puede grabar datos en la misma memoria). Aunado a esta característica se tiene una unidad de conversión analógica-digital (COP8CB), timers de alta velocidad, Ocho puertos, Unidad de comunicación serial USART y Reset por Brownout. Se temporiza con cristales de hasta 20Mhz y es ISP (Se puede programar en circuito). Estas y otras más características las veremos a detalle en los capítulos siguientes. A continuación se muestra el diagrama a bloques del COP8CB.

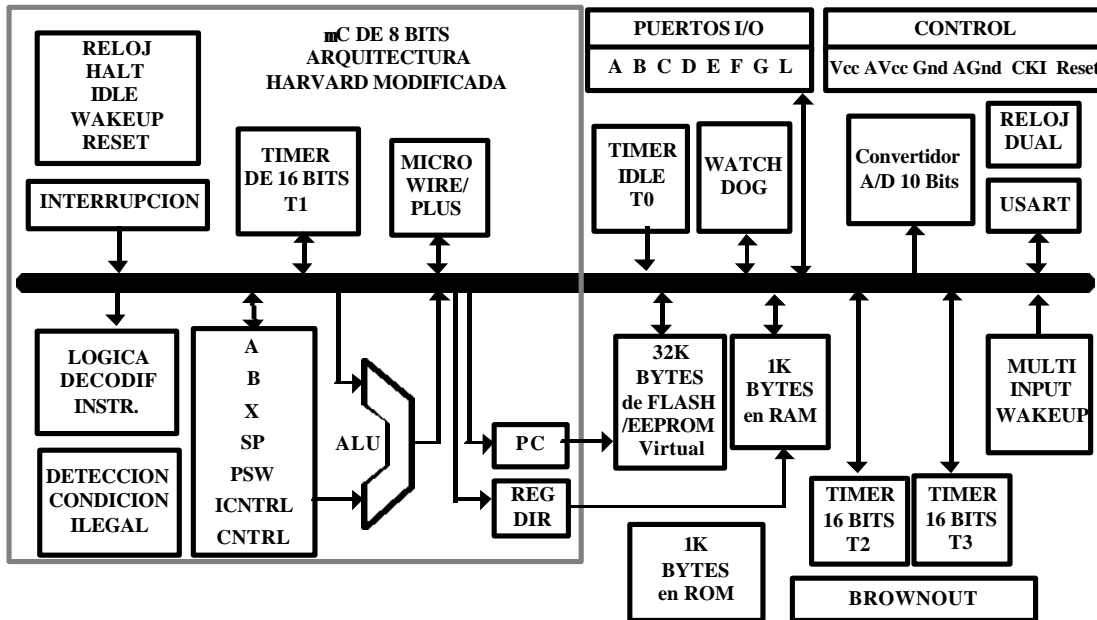


Figura 15. Diagrama del Microcontrolador COP8CB

## **Autoevaluación del CAPITULO 2.**

1. ¿ Qué significan las siglas COP8 ?
2. ¿ Cuántas familias tiene el COP8 ?
3. ¿ Qué significa un Microcontrolador OTP ?
4. ¿ Qué diferencia hay entre un micro OTP y uno de Máscara ?
5. ¿ Cuántos grupos encontramos en la Familia S ?
6. ¿ Que diferencias hay entre la familia COP8SA y COP8SG ?
7. ¿Cuál es la característica principal del COP8AC ?
8. ¿Cuál es la característica principal del COP8CB ?
9. ¿ Qué ventajas tiene la memoria Flash sobre la EEPROM ?
10. ¿ Para que sirve tener un comparador analógico en un Microcontrolador ?
11. ¿ Cuántos puertos tiene el COP8SA ?
12. ¿ Cuántos puertos tiene el COP8CB ?
13. ¿ Menciona las características que son iguales entre el COP8SA, SG y CB ?
14. ¿ Menciona las características diferentes entre el COP8SA y AC ?

### **Para Investigar**

15. ¿ Qué es Brownout ?
16. ¿ Cuántas variantes en encapsulados (en Microcontroladores) se pueden tener ?
17. ¿Cuál es la diferencia entre el COP8CB y COP8SBR ?
18. ¿ Explica brevemente los protocolos CAN, UART, Microwire, ISP, I<sup>2</sup>C ?
19. ¿ Cuáles son las formas de conversión analógica-digital utilizadas en los Microcontroladores actualmente, explica brevemente como funciona cada una. ?

### 3.1 SET DE INTRUCCIONES

En este capítulo describiremos el set de instrucciones del COP8SA (Por ser el más básico) y después haremos referencia a las instrucciones especiales como las instrucciones del UART o las instrucciones de la memoria FLASH.

El Set de intrucciones del COP8 es muy amigable, casi todas las instrucciones son de un solo byte ( 77%) lo que minimiza el tamaño del programa, tambien contiene instrucciones multifunción (realizan más de una tarea como decrementar un valor y luego compararlo y saltar a otra localidad de memoria), Se tienen tres apuntadores a memoria RAM (Dos son apuntadores para accesar datos en forma indexada y otro para accesar la pila), disponibilidad de 12 registros de propósito general para ser utilizados por algunas intrucciones, habilidad de poner, limpiar y probar cualquier bit individualmente localizado en la memoria de datos incluyendo los registros y puertos entrada/salida mapeados en memoria, instrucciones con incrementos y decrementos automáticos, etc. La forma de programación mas popular es vía el lenguaje ensamblador. Existen otras mas complicadas principalmente por su costo como el compilador en C. Consecuentemente, nos enfocaremos al ensamblador y a su Sintaxis. Una línea típica de una instrucción en lenguaje ensamblador está dada de la siguiente forma :

[ mnemónico ]      operando1, operando2

Donde típicamente el resultado de una operación lógica, aritmética o de carga es almacenado en operando1. En los ejemplos siguientes quedará mas clara esta disposición.

### 3.2 MODOS DE DIRECCIONAMIENTO

El set de instrucciones ofrece una amplia variedad de métodos para especificar la dirección de memoria. Cada método es llamado Modo de Direccionamiento. Estos modos de direccionamiento son clasificados en dos categorías: Modos de direccionamiento con Operandos y Modos de direccionamiento de transferencia de control. Los modos de direccionamiento con operandos son

los métodos donde se especifica una dirección para acceder un dato (escribirlo o leerlo). Los modos de direccionamiento de transferencia de control son usados con las instrucciones de saltos para determinar la secuencia en la ejecución del programa.

### 3.2.1 Modo de Direccionamiento con Operandos

El operando de una instrucción determina la localidad de memoria que será afectada por la instrucción. Existen varios modos de direccionamiento de este tipo permitiendo que las localidades de memoria sean determinadas en diferentes formas. Una instrucción puede especificar una dirección de forma directa proporcionando una dirección específica, o indirectamente a través de un apuntador a memoria. En el modo inmediato, el dato a ser utilizado es el que es especificado en la propia instrucción. Cada modo de direccionamiento tiene sus ventajas y desventajas con respecto a la flexibilidad, velocidad de ejecución y lo compacto del programa. No todos los modos están disponibles con todas las instrucciones. Los modos de direccionamiento disponibles son:

- Directo
- Indirecto con el registro B ó X
- Indirecto con el registro B ó X con un post Incremento/Decremento
- Inmediato
- Inmediato Corto
- Indirecto desde la Memoria de Programa.

Por ahora, sin detallar en las instrucciones, entenderemos la lógica de los modos de direccionamiento con un ejemplo.

#### Modo de direccionamiento directo

La dirección de memoria es especificada directamente como un byte en la instrucción. En lenguaje ensamblador, la dirección directa es escrita con un valor numérico (o alguna etiqueta que ha sido definida previamente en el programa con un valor numérico)

Ejemplo: LD A,05

(LD x,y) Carga x el contenido y	Contenido antes	Contenido después
ACUMULADOR ( A )	?	A6 hex
MEMORIA ( 05 )	A6 hex	A6 hex

Esto significa que queremos mover al registro A lo que se encuentre en la localidad de Memoria RAM 05 hexadecimal. El contenido del registro A antes de la ejecución de la instrucción no es de nuestro interés mientras que el contenido de la localidad 05 de memoria RAM es A6 Hex. Por lo tanto, el mnemónico LD obliga a que el dato localizado en 05 se “copie” al registro Acumulador A. Ver el resultado en la tabla anterior.

### **Modo de direccionamiento indirecto con el registro B ó X**

La dirección de memoria es especificada por el contenido del registro B ó el registro X (Son los registros punteros a RAM). En el lenguaje ensamblador, la notación que especifica qué registro es utilizado como apuntador es [B] ó [X].

Ejemplo: LD A,[B]

Carga al Acumulador	Contenido antes	Contenido después
ACUMULADOR ( A )	?	87 hex
MEMORIA ( 05 )	87 hex	87 hex
REGISTRO ( B )	05 hex	05 hex

### **Modo de direccionamiento indirecto con el registro B ó X con post incrementos o decrementos**

La dirección de memoria es especificada por el contenido del registro B ó el registro X con la diferencia que después de tomar el valor apuntado por B ó X el registro apuntador es incrementado ó decrementado en uno automáticamente. Este tipo de direccionamiento es utilizado cuando hacemos referencia a localidades de memoria consecutivas. La notación en lenguaje ensamblador es [B+], [B-], [X+], [X-] donde se especifica que apuntador es utilizado y que operación realizar (incremento o decremento)

Ejemplo: LD A,[B+]

Carga al Acumulador	Contenido antes	Contenido después
ACUMULADOR ( A )	?	87 hex
MEMORIA ( 05 )	87 hex	87 hex
REGISTRO ( B )	05 hex	06 hex

### Modo de direccionamiento inmediato

En este modo el dato de la operación es agregado al Opcode de la instrucción en la memoria de programa. En el lenguaje ensamblador, el signo numérico (#) identifica un operando inmediato. Es decir, cuando antepone el signo # a un número identifica que estamos haciendo referencia a su valor mismo.

Ejemplo: LD A,#05

Carga al Acumulador	Contenido antes	Contenido después
ACUMULADOR ( A )	?	05 hex

### Modo de direccionamiento inmediato corto.

Este es un caso especial de una instrucción inmediata. En la instrucción “carga inmediata al registro B”, el valor de 4 bits inmediato de la instrucción es cargado en el nibble bajo del registro B. El nibble alto del registro B es limpiado al 0000 binario.

Ejemplo: LD B,#07

Carga al Acumulador	Contenido antes	Contenido después
Registro ( B )	12 hex	07 hex

### Modo de direccionamiento indirecto de memoria de programa

Este es un caso especial de una instrucción indirecta que permite el acceso a las tablas de datos guardadas en memoria de programa. En el caso de “Carga al Acumulador Indirecta” que es la instrucción LAID, los bytes altos y bajos del registro Contador de Programa son usados temporalmente como apuntador de memoria de programa con el propósito de acceder la información. El contenido del byte bajo del (PC) identificado como (PCL) es reemplazado por el contenido del registro (A), entonces el dato apuntado por la nueva dirección apuntada por (PCH) y por A es cargado en el acumulador y simultáneamente, el contenido original de (PCL) es recuperado para que el programa pueda continuar con la ejecución normal del programa.

Ejemplo: LAID

Carga al Acumulador	Contenido antes	Contenido después
ACUMULADOR ( A )	1F hex	25 hex
Registro PCH	04 hex	04 hex

Registro PCL	35 hex	36 hex
Localidad de Memoria 04F1 hex	25 hex	25 hex

### 3.2.2 Modo de direccionamiento de transferencia de control

Las instrucciones de un programa son ejecutadas usualmente en orden. Sin embargo, las instrucciones de salto pueden ser utilizadas para cambiar la secuencia de ejecución normal. Un cambio en la secuencia del programa no requiere de un cambio en el contenido del (PC). El bit más significativo del PC no es usado por lo que tenemos 15 bits para direccionar a la memoria de programa.

Existen diferentes modos de direccionamiento de transferencia que especifican una dirección nueva para el (PC). La elección del modo de transferencia depende, en primera instancia, de la distancia del salto. Saltos más lejanos algunas veces requieren más bytes para determinar completamente el contenido de (PC). Los modos de direccionamiento de Transferencia de Control son :

- Salto Relativo
- Salto Absoluto
- Salto Absoluto Largo
- Salto Indirecto

A continuación veremos como es afectado el (PC) con los direccionamientos de transferencia de control.

#### Salto relativo

Esta instrucción es de un byte. Seis bits del Opcode especifican la distancia del salto desde la localidad donde se encuentre el (PC) apuntando en memoria de programa. La distancia del salto puede ser de un rango de  $-32$  a  $+32$ . Un salto de una localidad no es permitido, el programador deberá utilizar una instrucción de No Operación "NOP".

Ejemplo: JP 0A

Salto Relativo	Contenido antes	Contenido después
PCU	02 hex	02 hex
PCL	05 hex	0F hex

### Salto absoluto

Esta instrucción es de dos bytes de longitud, 12 bits del OpCode determinan el nuevo contenido del PC. Los tres bits más significativos del (PC) permanecen sin cambio, restringiendo que el nuevo valor del (PC) se encuentre en un espacio de 4Kbytes de la instrucción actual. (Esta restricción es relevante únicamente en dispositivos de más de 4K de memoria de programa.

Ejemplo: JMP 0125

Salto Relativo	Contenido antes	Contenido después
PCU	0C hex	01 hex
PCL	77 hex	25 hex

### Salto absoluto largo

Esta instrucción es de tres bytes donde 15 bits del Opcode especifican el nuevo contenido del registro (PC).

Ejemplo: JMP 03625

Salto Relativo	Contenido antes	Contenido después
PCU	42 hex	36 hex
PCL	36 hex	25 hex

### Salto indirecto

Esta es una instrucción de un solo byte, el byte bajo de la dirección es obtenido de la tabla almacenada en la memoria de programa, con el Acumulador funcionando como byte bajo apuntando a la memoria de programa. Para propósito de accesos a la memoria de programa, el contenido del Acumulador es escrito en el (PCL) temporalmente. El dato apuntado por el registro Contador de Programa (PCH/PCL) es cargado a (PCL), mientras (PCH) permanece sin cambio.

La instrucción VIS es un caso especial de direccionamiento indirecto, donde el vector de dos bytes asociado con la interrupción es transferido al (PC) para saltar a la rutina de servicio asociada a la interrupción.

Ejemplo: JID

Salto Relativo	Contenido antes	Contenido después
PCU	01 hex	01 hex
PCL	C4 hex	32 hex
Acumulador (A)	26 hex	26 hex
Localidad en Memoria 0126	32 hex	32 hex

### **Tipos de Instrucciones**

El set de instrucciones contiene una amplia variedad de instrucciones. Para su estudio dividiremos todo el set de instrucciones en grupos como Instrucciones Aritméticas, Transferencia de Control, Carga e intercambio, Instrucciones lógicas, Manipulación por bit y por último las Instrucciones Condicionales.

El set está compuesto de 58 diferentes instrucciones (COP8SA) teniendo en consideración todos los modos de direccionamiento tenemos un total de 87 instrucciones. Para poder realizar un programa identificaremos todos los registros a los que haremos referencia. La siguiente lista muestra la correspondencia entre la notación (en ensamblador) con su significado correspondiente.

Notación	Definición/Función	Tamaño
A	Registro Acumulador	8 bits
B	Registro Apuntador B	8 bits
[B]	Contenido en RAM apuntado por el registro B	8 bits
[B+]	Igual que [B] con un post-incremento en el registro B	8 bits
[B-]	Igual que [B] con un post- decremento en el registro B	8 bits
C	Bandera de Acarreo (Carry), bit 6 del registro PSW	1 bit
HC	Bandera de Acarreo Intermedio, bit 7 del reg. PSW	1 bit
MA	Dirección de 8 bits para almacenar datos en RAM	8 bits
MD	Memoria directa (localidad de Memoria RAM)	>=8 bits
PC	Contador de Programa, apuntador a 32768 localidades	15 bits
PCU	Registro superior del Contador de Programa	7 bits
PCL	Registro inferior del Contador de Programa	8 bits
PSW	Registro indicador del estado del Microcontrolador	8 bits
REG	Registro de propósito general (16 localidades en RAM)	8 bits
REG#	#de Registro a ser usado (# = 0-F hex)	8 bits
#	Símbolo utilizado para indicar un valor inmediato	8 bits
[X]	Contenido en RAM apuntado por el registro X	8 bits
[X+]	Igual que [X] con un post-incremento en el registro X	8 bits
[X-]	Igual que [X] con un post-decremento en el registro X	8 bits
SP	Stack Pointer - Registro apuntador a la pila	8 bits

### 3.3 SET DE INSTRUCCIONES

#### ADC— Suma con acarreo

Sintaxis:

- a)ADC A,[B]
- b)ADC A,#
- c)ADC A,MD

Descripción: El contenido de

- a) El dato almacenado en la localidad de memoria apuntada por el puntero **B**
- b) El dato inmediato encontrado en el segundo byte de la instrucción

c) La localidad de memoria en el segundo byte de la instrucción

es sumado al contenido del acumulador y el resultado es simultáneamente incrementado si la bandera de Carry esta en uno. El resultado es colocado en el acumulador

Operación:  $A \leftarrow A + \text{VALOR} + C$   
 $C \leftarrow \text{CARRY}; HC \leftarrow \text{HALF CARRY}$

### **ADD — Suma**

Sintaxis:

- a) ADD A,[B]
- b) ADD A,MD
- c) ADD A,#

Descripción: El contenido de la localidad de memoria referida por:

- a) El apuntador **B**
- b) La localidad de memoria en el segundo byte de la instrucción
- c) El dato inmediato encontrado en el segundo byte de la instrucción

Son sumados al contenido del acumulador, el resultado es colocado en el acumulador. Las banderas de C y HC **NO** son afectadas

Operación:  $A \leftarrow A + \text{VALOR}$

### **AND — Operación lógica AND**

Sintaxis:

- a) AND A,[B]
- b) AND A,#
- c) AND A,MD

Descripción: La operación *AND* es realizada en los bits correspondientes en el acumulador y

- a) El dato almacenado en la localidad de memoria apuntada por el puntero **B**
- b) El dato inmediato encontrado en el segundo byte de la instrucción
- c) La localidad de memoria en el segundo byte de la instrucción

El resultado es colocado en el acumulador

Operación:  $A \leftarrow A \text{ AND } \text{VALOR}$

**ANDSZ — Hacer la operación lógica AND, saltar si el resultado es cero**  
Sintaxis: ANDSZ A,#

Descripción: La operación *AND* es realizada en los bits correspondientes del acumulador con el valor del dato inmediato en el segundo byte de la instrucción. Si el resultado es cero, la siguiente instrucción es ignorada. El acumulador permanece sin cambio. La instrucción puede ser usada para probar la presencia de cualquier bit en el A. La máscara en el segundo byte es usada para seleccionar el bit a ser probado.

Operación: Si  $(A \text{ AND } \#) = 0$ , Entonces salta la siguiente instrucción.

**CLR — Limpiar acumulador**  
Sintaxis: CLR A

Descripción: El acumulador es limpiado (El contenido es llevado a ceros)

Operación:  $A \leftarrow 0$

**DCOR — Corrección decimal**  
Sintaxis: DCOR A

Descripción: Esta instrucción es usada después de una instrucción *ADC* (suma con carry) o *SUBC* (resta con carry) realizando una corrección decimal al resultado. Hay que tomar en cuenta que antes de realizar la operación *ADC* la instrucción debe ser precedida por *ADD A, #066* (suma de un 66 hexadecimal) para la corrección decimal. Esta instrucción asume que ambos operandos se encuentran en formato BCD (Binary Coded Decimal) y produce el resultado en éste mismo formato. Las banderas de C y HC permanecen sin cambio.

Operación:  $A \text{ (Formato BCD)} \leftarrow A \text{ (Formato binario)}$

**DEC — Decrementa acumulador**  
Sintaxis: DECA

Descripción: El contenido del acumulador se decrementa y se almacena el resultado en el mismo acumulador. Las banderas C y HC permanecen sin cambio.

Operación:  $A \leftarrow A - 1$

## **DRSZ REG# — Decrementa registro y salta si el resultado es cero**

Sintaxis: DRSZ REG#

Descripción: Esta instrucción decrementa el contenido del registro localizado en memoria (seleccionado por el signo (#), donde # es igual un valor desde 0 a F) y coloca el resultado en el mismo registro. Si el resultado es cero, la siguiente instrucción es saltada. Esta instrucción es muy útil cuando el usuario requiere de la repetición de una secuencia de instrucciones un número determinado de veces. Por lo tanto, el usuario almacena el número de veces requeridas de repeticiones en el registro. Consecuentemente *DRSZ* es acompañada por la instrucción *JP* (Salto Relativo) que regresa al inicio de la secuencia que esta siendo repetida, de tal forma que cuando el registro llega a ser cero la instrucción *JP* es saltada evitando que se repita la secuencia una vez mas.

Operación:  $REG <- REG - 1,$   
Si  $(REG - 1) = 0,$   
Entonces salta la siguiente instrucción

## **IFBIT — Probar bit**

Sintaxis:

- a)IFBIT #,[B]
- b)IFBIT #,MD
- c)IFBIT #,A

Descripción: El bit seleccionado ( $\# = 0$  a 7, con 7 siendo el bit de más alta prioridad) desde

- a) La localidad de memoria que esta siendo apuntada por el puntero **B** es probado
- b) La localidad de memoria que se encuentra en el segundo byte de la instrucción es probado
- c) El acumulador es probado

Si el bit seleccionado es alto (=1), entonces la siguiente instrucción será ejecutada. De otra forma (si el bit es =0) la instrucción será saltada.

Operación: Si BIT (#) Seleccionado es igual a 0,  
Entonces salta la siguiente instrucción

NOTA: En determinado momento la instrucción *IFBIT #,A* comparte el mismo Opcode que la instrucción *ANDSZ* (Opcode 60). Esta instrucción es entonces equivalente a la instrucción *ANDSZ* de la siguiente forma:

IFBIT 0,A equivale a ANDSZ A,#1  
IFBIT 1,A equivale a ANDSZ A,#2  
IFBIT 2,A equivale a ANDSZ A,#4  
IFBIT 3,A equivale a ANDSZ A,#8

IFBIT 4,A equivale a ANDSZ A,#16  
IFBIT 5,A equivale a ANDSZ A,#32  
IFBIT 6,A equivale a ANDSZ A,#64  
IFBIT 7,A equivale a ANDSZ A,#128

### **IFBNE # — Si el apuntador B no es igual**

Sintaxis: IFBNE #

Descripción: Si el nibble bajo del puntero **B** no es igual a # (donde el signo # es igual a 0 ó hasta F), entonces salta ejecuta la siguiente instrucción. De otra forma, la instrucción siguiente es saltada. Esta instrucción es de utilidad cuando el puntero **B** está siendo recorrido apuntando tablas de datos o en cualquier secuencia en un ciclo. La instrucción *IFBNE* usualmente es seguida de la instrucción *JP* que hace un salto al inicio de la secuencia que se ha estado repitiendo. Este ciclo se repetirá hasta que el puntero **B** sea encontrado igual al valor de #.

### **IFC — Probar si está puesta la bandera de acarreo (carry)**

Sintaxis: IFC

Descripción: La siguiente instrucción será ejecutada si la bandera de Carry es encontrada puesta (es decir C=1). De otra forma, la siguiente instrucción será saltada. La bandera de Carry permanece sin cambio alguno.

### **IFEQ — Probar si es igual**

Sintaxis:

- a)IFEQ A,[B]
- b)IFEQ A,#
- c)IFEQ A,MD
- d)IFEQ MD,#

Descripción:

- a) El contenido de la localidad apuntada por el puntero **B** es comparado (si son iguales) con el contenido del acumulador.
- b) El valor inmediato encontrado en el segundo byte de la instrucción es comparado (si son iguales) con el contenido del acumulador.
- c) El contenido de la localidad de memoria cuya dirección se encuentra en el segundo byte de la instrucción es comparado (si son iguales) con el acumulador.

- d) El contenido de la localidad de memoria cuya dirección se encuentra en el segundo byte de la instrucción es comparado (si son iguales) con el valor inmediato encontrado en el tercer byte de la instrucción.

Una igualdad exitosa en la comparación resulta en la ejecución de la siguiente instrucción. De cualquier otra forma, la siguiente instrucción es saltada.

### **IFGT — Probar si es mayor que**

Sintaxis:

- a)IFGT A,[B]
- b)IFGT A,#
- c)IFGT A,MD

Descripción: El contenido del acumulador es comparado si es mayor que

- a) El contenido de la localidad apuntada por el puntero **B**
- b) El valor inmediato encontrado en el segundo byte de la instrucción
- c) El contenido de la localidad de memoria que se encuentra en el segundo byte de la instrucción

Una comparación exitosa ( $A >$  operando) resulta en que la siguiente instrucción se ejecuta. De cualquier otra forma la siguiente instrucción es ignorada

### **IFNC — Probar si no esta puesta la bandera de acarreo**

Sintaxis: IFNC

Descripción: La siguiente instrucción es ejecutada si la bandera de Carry es encontrada en cero (en reset). De cualquier otra forma la siguiente instrucción será ignorada. La bandera de Carry no es afectada.

### **IFNE — Probar si no es igual**

Sintaxis:

- a)IFNE A,[B]
- b)IFNE A,#
- c)IFNE A,MD

Descripción:

- a) El contenido de la localidad apuntada por el puntero **B** es comparada (si no son iguales) con el contenido del acumulador
- b) El Valor inmediato localizado en el segundo byte de la instrucción es comparado (si no son iguales) con el contenido del acumulador

- c) El dato en memoria referenciado por la dirección encontrada en el segundo byte de la instrucción es comparado (si no son iguales) con el contenido del acumulador.

Una desigualdad exitosa resulta en la ejecución de la siguiente instrucción, de otra forma la siguiente instrucción es saltada.

### **INC — Incrementar acumulador**

Sintaxis: INC A

Descripción: Esta instrucción incrementa el contenido del acumulador y coloca el resultado de regreso en el acumulador. Las banderas de Carry y Half Carry permanecen sin cambio.

Operación:  $A \leftarrow A + 1$

### **INTR — Interrupción (“Software Trap”)**

Sintaxis: INTR

Descripción: Este Opcode de 00 es llamado Software Trap. La primera tarea que realiza es almacenar la dirección en donde se encuentre el (PC) en la memoria de datos del stack. Después hace un salto a la localidad 00FF. Esta localidad es la dirección donde todas las interrupciones del COP888 saltan, ya sea las interrupciones por software y por hardware. La instrucción INTR no significa que sea programada explícitamente, sino que es invocada automáticamente cuando se encuentra una condición de error. La lectura de un Opcode indefinido (inexistente) produce una carga de ceros, lo cual se traduce en la instrucción INTR. Una condición similar puede presentarse si en una subrutina el registro puntero a la pila (SP) Stack Pointer es inicializado en el tope superior del espacio de memoria de datos en RAM. Luego entonces si el software obliga a que la pila sea desbordada (ó overpopped = Más subrutinas o regresos de interrupciones que llamados), entonces se obtendrá un valor no definido de RAM (puros unos). Esto causará que la ejecución del programa salte a la dirección FFFF Hex, donde se identificarán únicamente ceros y otra vez más se invocará a la intrucción software trap.

Operación:  $[SP] \leftarrow PCL ; [SP - 1] \leftarrow PCU$   
 $[SP - 2] : \text{Preparado para la siguiente carga en la Pila}$   
 $PC \leftarrow 0FF$

## **JID — Salto indirecto**

Sintaxis: JID

Descripción: La instrucción *JID* usa el contenido del acumulador para apuntar a un vector indirecto direccionando el programa. El contenido del acumulador es transferido al (PCL), después de que el dato de la memoria de programa es accesado por el (PC) es transferido al (PCL). Entonces el programa salta a la localidad de la memoria de programa . Debe de notarse que el (PCU) no cambia durante la instrucción *JID*, por lo tanto el salto indirecto deberá de ser realizado hacia alguna localidad de programa no mayor a 256 localidades.

Operación: PCL <- A

PCL <- Program Memory (PCU,A)

## **JMP — Salto absoluto**

Sintaxis: JMP ADDR

Descripción: Esta instrucción salta a la dirección de memoria especificada. El valor encontrado en el nibble inferior (4 bits) del primer byte de la instrucción es transferido al nibble inferior de (PCU), después el valor encontrado en el segundo byte de la instrucción es transferido al PCL. El programa entonces salta a la localidad de memoria accesada por PC. El rango de direccionamiento es de 15 bits. La instrucción *JMP* tiene un rango de direccionamiento de 12 bits.

Operación:

PC11-8 <- HIADDR (nibble alto del segundo byte de la instrucción,  
nibble bajo del primer byte de la instrucción)

PC7-0 <- LOADDR (Segundo byte de la instrucción)

## **JMPL — Salto absoluto largo**

Sintaxis: JMPL ADDR

Descripción: La instrucción *JMPL* permite saltos a cualquier localidad en un espacio de memoria de programa de 32 Kbytes. El valor encontrado en el segundo y tercer byte de la instrucción son transferidos al PCU y PCL respectivamente. El programa entonces salta a la localidad de memoria de programa apuntada por PC.

Operación:

PC14-8 <- HIADDR (Segundo byte de la instrucción)

PC7-0 <- LOADDR (Tercer byte de la instrucción)

### **JP — Salto relativo**

Sintaxis: JP DISP

Descripción: El valor encontrado en el desplazamiento relativo (8 bits) es sumado al (PC). Esto permite retroceder de 0 a 31 espacios (Con 0 representando un ciclo infinito) y un desplazamiento hacia delante de 2 a 32 direcciones. Un desplazamiento hacia delante unitario no es permitido.

Operación: PC <- PC + DISP + 1 (DISP no es = 0)

### **JSR — Salto a subrutina**

Sintaxis: JSR ADDR

Descripción: Esta instrucción salva la dirección de retorno en la pila y después realiza un salto a la dirección de la subrutina. La instrucción JSR tiene un espacio de direccionamiento de 12 bits.

Operación:

[SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: Preparado para la siguiente carga en la pila.

PC11-8 <- HIADDR (nibble alto de retorno de dirección, nibble bajo del primer byte de la instrucción)

PC7-0 <- LOADDR (Segundo byte de la instrucción)

### **JSRL — Salto a subrutina largo**

Sintaxis: JSRL ADDR

Descripción: La instrucción *JSRL* permite un salto a una subrutina localizada en cualquier localidad de memoria dentro del espacio de los 32Kbytes de la memoria de programa. La instrucción salva la dirección de retorno en la pila y después realiza un salto a la dirección de la subrutina.

Operación:

[SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: Preparado para la siguiente carga en la pila

PC14-8 <- HIADDR (Segundo byte de la instrucción)

PC7-0 <- LOADDR (Tercer byte de la instrucción)

### **LAID — Carga indirecta del acumulador**

Modo de direccionamiento: INDIRECTO

Descripción: La instrucción LAID utiliza el acumulador para apuntar a un dato almacenado en la memoria de programa. Primeramente el contenido del acumulador es intercambiado con el contenido del (PCL) formando una nueva dirección apuntando a memoria de programa. El dato localizado en tal localidad apuntada por PC es colocado en el acumulador y el contenido previo de (PCL) es regresado a este mismo registro. El contenido de (PCU) no es afectado por esta instrucción, por lo tanto los accesos a memoria de programa pueden ser realizados en un espacio de 256 bytes.

Operación: A <- Program Memory (PCU, A)

### **LD — Carga al acumulador**

Sintaxis:

a)LD A,[B]

b)LD A,[B+]

c)LD A,[B-]

d)LD A,#

e)LD A,MD

f)LD A,[X]

g)LD A,[X+]

h)LD A,[X-]

Descripción:

a) El contenido de la localidad de memoria apuntada por el puntero **B** es cargado en el acumulador

b) El contenido de la localidad de memoria apuntada por el puntero **B** es cargado en el acumulador, inmediatamente después el puntero **B** es incrementado

c) El contenido de la localidad de memoria apuntada por el puntero **B** es cargado en el acumulador, inmediatamente después el puntero **B** es decrementado

- d) El valor inmediato encontrado en el segundo byte de la instrucción es cargado en el acumulador
- e) El contenido de memoria de la localidad referenciada por la dirección en el segundo byte de la instrucción es cargado en A.
- f) El contenido de la localidad de memoria apuntada por el puntero **X** es cargado en el acumulador
- g) El contenido de la localidad de memoria apuntada por el puntero **X** es cargado en el acumulador, inmediatamente después el puntero **X** es incrementado
- h) El contenido de la localidad de memoria apuntada por el puntero **X** es cargado en el acumulador, inmediatamente después el puntero **X** es decrementado

Operación:  $A \leftarrow \text{VALUE}$

### **LD — Carga al apuntador B**

Sintaxis:

- a) LD B,# (# < 16)
- b) LD B,# (# > 15)

Descripción:

- a) El complemento a uno del valor encontrado en el nibble inferior encontrado en la instrucción es transferido al nibble bajo del registro puntero **B**, el nibble superior es puesto en ceros.
- b) El valor inmediato encontrado en el segundo byte de la instrucción es transferido al registro apuntador **B**.

Operación:            a)  $B3-B0 \leftarrow \#$  and  $B7-B4 \leftarrow 0$   
                           b)  $B \leftarrow \#$

### **LD — Carga a memoria**

Sintaxis:

- a) LD [B],#
- b) LD [B+],#
- c) LD [B-],#
- d) LD MD,#

Descripción:

- a) El valor inmediato encontrado en el segundo byte de la instrucción es cargado en la localidad de memoria apuntada por el registro apuntador **B**.

- b) El valor inmediato encontrado en el segundo byte de la instrucción es cargado en la localidad de memoria apuntada por **B**, inmediatamente el valor de **B** es incrementado en uno
- c) El valor inmediato encontrado en el segundo byte de la instrucción es cargado en la localidad de memoria apuntada por el apuntador **B**, inmediatamente el valor de **B** es decrementado en uno
- d) El valor inmediato localizado en el tercer byte de la instrucción es cargado en la localidad de memoria referenciada por la dirección localizada en el segundo byte de la instrucción.

Operación:

- a)  $[B] \leftarrow \#$
- b)  $[B] \leftarrow \#; B \leftarrow B + 1$
- c)  $[B] \leftarrow \#; B \leftarrow B - 1$
- d)  $MD \leftarrow \#$

### **LD — Carga a un registro**

Sintaxis: LD REG,#

Descripción: El valor inmediato del segundo byte de la instrucción es cargado en la memoria de datos referenciada en el nibble bajo del primer byte de la instrucción

Operación:  $REG \leftarrow \#$

### **NOP — No operación**

Sintaxis: NOP

Descripción: No Operación es realizada por esta instrucción. Entonces el resultado neto es un retardo unitario del ciclo de instrucción.

### **OR — Operación lógica OR**

Sintaxis:

- a) OR A,[B]
- b) OR A,#
- c) OR A,MD

Descripción: La operación OR es realizada en los bits correspondientes en el acumulador con

- a) El contenido de la localidad de memoria apuntada por el registro apuntador **B**.
- b) El valor inmediato encontrado en el segundo byte de la instrucción.
- c) El contenido de la localidad de memoria referenciada por la dirección encontrada en el segundo byte de la instrucción.

Operación:  $A \leftarrow A \text{ OR VALOR}$  ; El resultado es colocado en el acumulador

### **POP — Restaurar datos de la pila “Pop Stack”**

Sintaxis: POP A

Descripción: El registro (SP) es incrementado y el contenido de la localidad de memoria referenciada por (SP) es transferido al acumulador.

Operación:

$SP \leftarrow SP + 1$   
 $A \leftarrow [SP]$

### **PUSH — Salvar datos en la pila “Push Stack”**

Sintaxis: PUSH A

Descripción: El contenido del acumulador es transferido a la localidad de memoria referenciada por (SP) y después el registro (SP) es decrementado.

Operación:

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

### **RBIT — Limpiar el bit de memoria**

Sintaxis:

- a) RBIT #,[B]
- b) RBIT #,MD

Descripción: El bit seleccionado (# = 0 hasta 7, con 7 siendo el bit más significativo) del dato en memoria referenciado por

- a) **B** es puesto a cero.
- b) La dirección del segundo byte de la instrucción es puesto en cero.

Operación:  $[Address:\#] \leftarrow 0$

## **RC — Limpiar acarreo**

Sintaxis: RC

Descripción: Ambas banderas Carry y Half Carry son limpiados a cero.

Operación:

$C \leftarrow 0$  ;  $HC \leftarrow 0$

## **RET — Retorno de subrutina**

Sintaxis: RET

Descripción: El registro apuntador a pila (SP) es incrementado. El contenido en la localidad de memoria referenciada por (SP) es transferida al PCU, después el registro (SP) es nuevamente incrementado. Entonces el contenido de la localidad de memoria referenciado por (SP) es transferido a PCL. Es entonces cuando la dirección de retorno es recuperada de la memoria de datos RAM. Por último el programa salta a la memoria de programa apuntada por PC.

Operación:

$PCU \leftarrow [SP + 1]$

$PCL \leftarrow [SP + 2]$

$[SP + 2]$  : Preparado para la siguiente carga en la pila

## **RETI — Retorno de una interrupción**

Sintaxis: RETI

Descripción: El registro (SP) es incrementado. El contenido de la localidad de memoria apuntado por (SP) es transferido al PCU, después el registro (SP) es nuevamente incrementado dando lugar a que el contenido de la localidad de memoria siguiente sea cargado en PCL. La dirección de retorno es entonces recuperada de la memoria de datos RAM.

El programa ahora salta a la localidad de memoria apuntada por PC. La bandera de (GIE) Global Interrup Enable es puesta en uno.

Operación:

$PCU \leftarrow [SP + 1]$

$PCL \leftarrow [SP + 2]$

$[SP + 2]$  : Preparado para la siguiente carga en la pila

$GIE \leftarrow 1$

## **RETSK — Retorno de subrutina y salto**

Sintaxis: RETSK

Descripción: El registro apuntador a pila (SP) es incrementado. El contenido en la localidad de memoria referenciada por (SP) es transferida al PCU, después el registro (SP) es nuevamente incrementado. Entonces el contenido de la localidad de memoria referenciado por (SP) es transferido a PCL. Es entonces cuando la dirección de retorno es recuperada de la memoria de datos RAM. Por último el programa salta a la memoria de programa apuntada por PC y ignora la siguiente instrucción.

Operación:

PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : Preparado para la siguiente carga en la pila

Salto de la siguiente instrucción

## **RLC — Rotar el acumulador hacia la izquierda incluyendo el acarreo**

Modo de direccionamiento: RLC A

Descripción: El contenido del acumulador y la bandera de Carry son rotados la posición de un bit hacia la izquierda con el bit de Carry sirviendo como el noveno bit. El estado previo de la bandera de Carry es transferido al bit menos significativo del Acumulador (A0). El bit más significativo del Acumulador (A7) es transferido a la bandera de Carry. El bit mas significativo del nibble inferior del Acumulador (A3) es transferido a la bandera de Half Carry así como al siguiente bit del Acumulador (A4).

Operación:

C <- A7 <- A6 <- A5 <- A4 <- A3 <- A2 <- A1 <- A0 <- C

HC <- A3

## **RPND — “Reset Pending”; Limpiar bandera de interrupción**

Sintaxis: RPND

Descripción: La instrucción RPND limpia la bandera de “Non-Maskable Interrupt” (NMIPND) afectada si se presenta una interrupción ha sido reconocida y la bandera de software trap no fue encontrada en alto. De la misma forma, RPND limpia incondicionalmente la bandera de Software Trap (STPND).

Operación:

SI la interrupcion NMI es reconocida y STPND = 0

ENTONCES NMPND <- 0 and STPND <- 0

DE OTRA FORMA STPND <- 0

### **RRC — Rotar a la derecha incluyendo el acarreo**

Modo de Direccionamiento: RRC A

Descripción: El contenido del acumulador y la bandera de Carry son rotados la posición de un bit hacia la derecha con el bit de Carry sirviendo como el noveno bit. El estado previo de la bandera de Carry es transferido al bit más significativo del Acumulador (A7). El bit menos significativo del Acumulador (A0) es transferido a la bandera de Carry y a la bandera de Half Carry.

Operación:

C -> A7 -> A6 -> A5 -> A4 -> A3 -> A2 -> A1 -> A0 -> C ; A0 -> HC

### **SBIT — Poner en uno un bit en memoria**

Sintaxis:

a) SBIT #,[B]

b) SBIT #,MD

Descripción: El bit seleccionado (# = 0 hasta 7, con 7 siendo el más significativo) de la localidad de memoria referenciada por el

a) El puntero **B** es puesto en uno.

b) Dirección localizada en el segundo byte de la instrucción es puesto en uno.

Operación: [Dirección:#] <- 1

### **SC — Poner en uno la bandera de acarreo**

Sintaxis: SC

Descripción: Ambas banderas de Carry y Half Carry son puestas en 1.

Operación:

C <- 1; HC <- 1

## **SUBC — Resta incluyendo el accareo**

Sintaxis:

- a) SUBC A,[B]
- b) SUBC A,#
- c) SUBC A,MD

Descripción:

- a) El contenido de la localidad de memoria apuntada por el puntero **B** es restado del contenido del acumulador y el resultado es simultáneamente decrementado si la bandera de Carry es encontrada en cero.
- b) El valor inmediato encontrado en el segundo byte de la instrucción es restado del contenido del Acumulador y el resultado es simultáneamente decrementado si la bandera de Carry es encontrada en cero.
- c) El valor contenido en la localidad de memoria encontrada en el segundo byte de la instrucción es restado del contenido del Acumulador y el resultado es simultáneamente decrementado si la bandera de Carry es encontrada en cero.

El resultado es colocado en el Acumulador y la bandera de Carry es ya sea puesta en uno o limpiada a cero, dependiendo de la presencia o ausencia de la condición de borrow del resultado.

Similarmente, la bandera de Half Carry es ya sea puesta o limpiada dependiendo de la presencia o ausencia de la condición de borrow del nibble inferior.

Esta instrucción es implementada por la suma en complemento a uno del substrayendo al acumulador y después realiza la suma. Consecuentemente, la condición de borrow es equivalente a la ausencia de Carry y viceversa. Similarmente, la bandera de Half Carry es el equivalente de la ausencia de Half borrow y viceversa. Un borrow previo (ausencia de un carry previo) evitará el incremento del resultado.

Operación:

- A <- A + VALOR + C
- C <- AUSENCIA DEL BORROW DEL BYTE
- HC <- AUSENCIA DEL BORROW DEL NIBBLE BAJO

## **SWAP — Intercambio de “nibbles” del acumulador**

Sintaxis: SWAP A

Descripción: El nibble superior y el inferior del Acumulador es intercambiado.

Operación:  $A(7-4) \leftrightarrow A(3 - 0)$

## **VIS — Selector del vector de interrupción**

Sintaxis: VIS

Descripción: El propósito de la instrucción *VIS* es realizar la identificación de la interrupción y salto al vector de interrupción correspondiente con la más alta prioridad que haya sido habilitada y solicitada. La instrucción *VIS* realiza esta operación de vectorización a la rutina de servicio. Todas las interrupciones forzan un salto a la localidad 00FF hex, una vez reconocidas. Consecuentemente el usuario tiene la libertad de decidir el manejo de la situación (como almacenar en la pila el contenido del Acumulador o los apuntadores B ó X, etc.). La instrucción *VIS* puede ser colocada en la localidad de memoria 00FF hex. De ser así, la instrucción *VIS* salta al vector formado por dos bytes en una tabla de memoria de programa localizada en el borde superior de la memoria de programa desde la dirección xyE0 hasta xyFF hex. Notar que xy es el número de bloque (usualmente 01) donde la instrucción *VIS* es localizado (cada bloque de memoria de programa contiene 256 bytes). Este vector de dos bytes es transferido al PC, entonces el programa salta a la rutina de servicio asociada con la interrupción indicada por el vector. Esta rutina de servicio puede localizarse en cualquier lugar dentro del espacio de los 32 Kbytes de memoria de programa. La instrucción *VIS* debe ser programada en el tope de un bloque de memoria (como la dirección 00FF), la tabla asociada de 32 bytes es entonces localizada en el siguiente bloque más alto (01E0 a 01FF hex con la instrucción *VIS* en 00FF hex)

Operación:

PCL <- VA (Vector Arbitro de Interrupción generado por hardware)

PCU <- Memoria de Programa (PCU,VA)

PCL <- Memoria de Programa (PCU,VA+1)

## **X — Intercambio de datos con el acumulador**

Sintaxis:

a) X A,[B]

b) X A,[B+]

c) X A,[B-]

d) X A,MD

- e) X A,[X]
- f) X A,[X+]
- g) X A,[X-]

Descripción:

- a) El contenido de la localidad de memoria apuntada por el puntero **B** es intercambiado con el contenido del acumulador
- b) El contenido de la localidad de memoria apuntada por el puntero **B** es intercambiada con el contenido del acumulador, inmediatamente después el puntero **B** es incrementado
- c) El contenido de la localidad de memoria apuntada por el puntero **B** es intercambiada con el contenido del acumulador, inmediatamente después el puntero **B** es decrementado
- d) El contenido de la localidad de memoria de datos localizado en el segundo byte de la instrucción es intercambiado con el contenido del acumulador.
- e) El contenido de la localidad de memoria apuntada por el puntero **X** es intercambiado con el contenido del acumulador
- f) El contenido de la localidad de memoria apuntada por el puntero **X** es intercambiada con el contenido del acumulador, inmediatamente después el puntero **X** es incrementado
- g) El contenido de la localidad de memoria apuntada por el puntero **X** es intercambiada con el contenido del acumulador, inmediatamente después el puntero **X** es decrementado

Operación:

- a)  $A \leftrightarrow [B]$
- b)  $A \leftrightarrow B; B \leftarrow B + 1$
- c)  $A \leftrightarrow B; B \leftarrow B - 1$
- d)  $A \leftrightarrow MD$
- e)  $A \leftrightarrow X$
- f)  $A \leftrightarrow X; X \leftarrow X + 1$
- g)  $A \leftrightarrow X; X \leftarrow X - 1$

### **XOR — Operación lógica OR exclusiva**

Sintaxis:

- a) XOR A,[B]
- b) XOR A,#
- c) XOR A,MD

Descripción: Una operación XOR (OR Exclusiva) es realizada en los bits correspondientes del acumulador con

- a) El dato almacenado en la localidad de memoria apuntada por el puntero **B**
- b) El dato inmediato encontrado en el segundo byte de la instrucción
- c) La localidad de memoria en el segundo byte de la instrucción

El resultado es colocado en el acumulador  
 Operación: A <- A **XOR** VALOR

### 3.4 PROGRAMACIÓN BÁSICA

En esta sección veremos algunos ejemplos de programación básica.

Ejemplo 1 : Identificar el estado de los registro y localides de memoria en la ejecución de cada instrucción.

	<b>A</b>	<b>B</b>	<b>X</b>	<b>C</b>	<b>01</b>	<b>02</b>	<b>03</b>	<b>04</b>	<b>05</b>	<b>06</b>
CLR A	0	-	-	0	0A	0B	0C	01	02	03
LD A,01	0A	-	-	0	0A	0B	0C	01	02	03
SC	0A	-	-	1	0A	0B	0C	01	02	03
LD B,#03	0A	03	-	1	0A	0B	0C	01	02	03
X A, [B]	0C	03	-	1	0A	0B	0A	01	02	03
ADC A,04	0E	03	-	0	0A	0B	0A	01	02	03
SUBC A,02	02	03	-	1	0A	0B	0A	01	02	03
LD B,#04	02	04	-	1	0A	0B	0A	01	02	03
X A,	01	05	-	1	0A	0B	0A	02	02	03
[B+]	05	01	-	1	0A	0B	0A	02	02	03
X A,B	0	01	05	1	0A	0B	0A	02	02	03
X A,X	02	01	04	1	0A	0B	0A	02	0	03
X A, [X-]	06	01	04	0	0A	0B	0A	02	0	03
ADC A,06	0A	02	04	0	06	0B	0A	02	0	03
X A,[B+]	01	02	04	0	06	0B	0A	02	0	03
XOR A,[B]	02	02	04	0	06	0B	0A	02	0	03
INC A	03	02	04	0	06	0B	0A	02	0	02
X A,06	02	02	05	0	06	0B	0A	03	0	02
X A,[X+]										

Ejercicio 1 : Llenar la tabla con los resultados de las operaciones realizadas en la siguiente secuencia

	<b>A</b>	<b>B</b>	<b>X</b>	<b>C</b>	<b>010</b>	<b>011</b>	<b>012</b>	<b>013</b>	<b>014</b>	<b>015</b>
CLR A	0	-	-	0	02	0A	0B	0F	0A	0A
LD A,#011										
LD B,#010										
LD [B],#012										
LD B,#011										
LD X,#013										
LD 014,#0A										
X A,[B-]										
LD A,B										
SC										
ADD A,010										
LD 015,#0AA										
X A,[X-]										
SWAP A										
ADC A,#010										
SUBC A,10										
INC A										
OR A,13										

Ejercicio 2 : Llenar la tabla con los resultados de las operaciones realizadas en la siguiente secuencia.

	<b>A</b>	<b>B</b>	<b>X</b>	<b>C</b>	<b>HC</b>	<b>SP</b>	<b>01</b>	<b>02</b>	<b>03</b>	<b>04</b>
CLR A	0	01	-	0	0	6F	00	01	02	03
LD A, [B+]										
X A, [B]										
ADC A,#0FF										
X A,02										
IFEQ 02,#03										
SC										
PUSH A										
X A,03										
XOR A,04										
X A,04										
SBIT 4,01										
RBIT 2,02										
DEC A										
ANDSZ										
A,#055										
PUSH A										

Ejercicio 3 : Suponer los siguientes valores en los registros y localidades de memoria.

PCU: 00 hex

PCL: 034 hex

SP: 06F hex

A: 021 hex

Localidad de memoria ROM [0021] : 055 hex

La subrutina siguiente se encuentra en al localidad [01E4]

La instrucción LAID es de un solo byte

La instrucción LD A,#VALOR INMEDIATO es de dos bytes

La instrucción NOP es de un byte

Teniendo en cuenta la información anterior, llenar la siguiente tabla con los resultados de las instrucciones listadas en la columna de la izquierda

	A	SP	PCU	PCL
LAID				
JSR siguiente				
...				
siguiente:				
LD A,#010				
NOP				
NOP				
RETSK				

## Limpiar RAM

La siguiente rutina limpia todas las localidades de memoria RAM. El argumento de la instrucción IFBNE puede necesitar ser ajustado, dependiendo de la cantidad de memoria disponible en el Microcontrolador.

Ejemplo 2 : Programa que limpia la RAM excepto SP

CLRAM:

LD 0FC,#070 ;Define un contador (0FC <- 070)

LD B,#0 ;Inicializa el puntero B

CLRAM2:

LD [B+],#0 ;Cargar la localidad de memoria apuntada por B con cero

DRSZ 0FC ;Decrementa contador

JP CLRAM2 ;Salta a CLRAM2 si el contador > 0

LD B,#0F0 ;Inicializa el puntero B, esta el la parte alta de RAM

CLRAM3:

LD [B+],#0 ;Cargar la localidad de memoria apuntada por B con cero

IFBNE #0D ;Hasta que B apunte a 0FD (=SP)

JP CLRAM3 ;B = 0FD salta esta instrucción, si B<0FD Salta a CLRAM3

LD B,#0 ;Incializa B a 0

## Tabla visual de la ejecución del programa ejemplo 1.

Instrucción	(0FC)	B	(00)	(01)	(02)
LD 0FD, #070	070	-	-	-	-
LD B,#0	070	00	-	-	-
DRSZ 0FC	06F	00	-	-	-
LD [B+],#0	070	1	00	-	-
DRSZ 0FC	06E	1	00	?	?
LD [B+],#0	070	2	00	00	?
DRSZ 0FC	06D	2	00	00	?
LD [B+],#0	070	3	00	00	00
DRSZ 0FC	06C	3	00	00	00
...	...	...	00	00	00

## Ejemplo 3 : Carga y copia de los datos localizados en ROM hacia RAM

```

.sect regs,reg                ; Definición segmento de registros
    cnt1:    .dsb 1           ; cnt1 es el registro 0
.endsect                       ; final del segmento de registros

.sect codigo1, rom           ; Definiendo segmento de código
reset:
    LD        cnt1,#10       ; cnt1 <- 0A en el registro R0
    LD        B,#0           ; limpiar apuntador B
c1_ciclo1:                    ; llena las direcciones 0x0-0xA (RAM) con la tabla 1
    LD        A,B            ; A <- B
    ADD       A,#low(tabla1) ; A <- dirección (byte bajo) donde se encuentra la tabla1
    LAID     ; A obtiene el valor de la tabla (en ROM)
    X        A,[B+]         ; Copiando el valor de ROM a RAM
    DRSZ     cnt1
    JP       c1_ciclo1
    LD        cnt1,#10       ; cnt1 <- 10 decimal
    LD        B,#010        ; B <- 10 hexadecimal
    SC      ; Poner la bandera de Carry
c1_ciclo2:                    ; llena direcciones 0x10 - 0x1A con la tabla 2 (lejana)
    LD        A,#10         ; A <- 10 decimal ó 0A hex
    SUBC     A,cnt1         ; Cálculo del desplazamiento; A <- (0A-cnt1)
    JSR     lee_tabla2
    X        a,[b+]         ; Copiando el valor de ROM a RAM
    DRSZ     cnt1
    JP       c1_ciclo2
fin:
    JP       fin
tabla1:
    .byte    040,041,042,043,044,045,046,047,048,049
.endsect

.sect tabla2,rom, abs=0x400
lee_tabla2:
    ADD       a,#low(tabla2) ; Realiza el desplazamiento de A sobre la tabla 2
    LAID
    RET
tabla2:
    .byte    '0','1','2','3','4','5','6','7','8','9'
.endsect
.end reset

```

Ejercicio 4 : Hacer un programa que realice las siguientes tareas:

1. Limpiar las localidades en RAM de 0 - 020 hex
2. Declarar una tabla en memoria ROM con el siguiente contenido  
“Microcontroladores COP8”
3. Copiar el contenido de la tabla ROM en las localidades 0-020 hex en memoria RAM
4. Encontrar el valor más grande dentro de ésta tabla y almacenarlo en el registro apuntador X

### **Autoevaluación del CAPITULO 3**

1. ¿ Qué significa una instrucción multifunción y dar un ejemplo ?
2. ¿ Describe las ventajas/desventajas del set de instrucciones del COP8 ?
3. ¿ Cuáles son los modos de direccionamiento del COP8 ?
4. ¿ Dar dos ejemplos de los direccionamientos directos, indirecto con un registro apuntador, inmediato y por último indirecto de memoria de programa ( LAID ) ?
5. ¿ Describe brevemente los tipos de saltos existentes en el COP8 ?
6. ¿ Dar un ejemplo de la utilización de la instrucción DCOR?
7. ¿ Dar un ejemplo de la utilización de la instrucción DRSZ ?
8. ¿ Dar un ejemplo de la utilización de la instrucción SUBC ?
9. ¿ Describe brevemente la función de la instrucción VIS ?
10. ¿ Dar un ejemplo de la utilización de la instrucción IFEQ ?

### **Para Investigar**

11. ¿ Cuántas instrucciones tiene el COP8SA vs COP8CB ?
12. ¿ Qué opciones en compilación en C existen para el COP8 ?
13. ¿ Qué ventajas tiene un compilador en C vs ensamblador ?
14. ¿ Qué desventajas tiene un compilador en C vs ensamblador ?
15. ¿ Explica la sintáxis típica de un archivo ensamblador, dar un ejemplo ?

## UNIDADES BÁSICAS DEL MICROCONTROLADOR COP8

---

### 4.1 UNIDADES BÁSICAS

Todos los Microcontroladores COP8 más recientes cuentan con al menos, cuatro unidades básicas que son Timers ( Temporizadores ), Interfase Microwire ( Unidad de Comunicación Serial ), Modos de Ahorro de Energía y por último Interrupciones.

#### 4.1.1 Descripción de pines

La estructura del COP8SAx permite que el usuario defina el comportamiento de las terminales del Microcontrolador (pines), ya sea como entradas (lecturas de las señales externas) o como salidas (establecer el estado de los pines al exterior). Cada pin puede ser configurado independientemente como salidas (niveles lógicos altos o bajos) ó como entradas (alta impedancia o weak pull-up). Los puertos son identificados con letras (Puerto A, Puerto B, Puerto C, etc.) El número de pines de cada puerto varía dependiendo de la versión del Microcontrolador, tamaño del encapsulado, familia, etc. Para hacer referencia a los pines de los puertos se les asocia un número a la letra del puerto al que pertenecen. Por ejemplo, para hacer referencia al primer y tercer pin del puerto G escribiremos G0 y G2 respectivamente.

#### Configuración de entrada

Un pin puede ser configurado como entrada ya sea programandolo en weak pull-up o como estado de alta impedancia. En la configuración en *alta impedancia* el pin no recibirá ni aportará ningún estado lógico al resto del circuito. En configuración *weak pull-up* las líneas de entrada obtendrán una lectura de nivel alto cuando el pin, al no detectar circuitería exterior, es llevado por una resistencia interna a un estado lógico alto.

Los puertos de salida pueden ser programados pin a pin para mantener un estado lógico alto o bajo, según sea el requerimiento del usuario. En la actualidad, casi todos los Microcontroladores tienen puertos bi-direccionales. Es decir, el usuario tiene la plena libertad de decidir que pin funcionará como entrada y cuál como salida. En la familia COP8SA todos los puertos son bi-direccionales (a excepción del puerto D). Por otro lado existen puertos

únicamente de salida, esto es usual cuando el puerto está diseñado para manejar más corriente que los otros puertos (como el puerto D).

**Vcc y GND** son los pines de alimentación. Todos los pines de Vcc y GND deben ser conectados.

**CKI** es la entrada de reloj. Esta puede tomarse de un R/C, de un oscilador externo o un cristal (en conjunción con CKO).

**Reset** en operación normal este pin recibe un estado lógico de uno. Cuando este estado cambia momentáneamente a cero y retorna a uno el dispositivo se reinicializa, es decir PC apuntará a la localidad 00 hex.

El Microcontrolador COP8SA contiene cuatro puertos bi-direccionales de hasta 8 bits (C, G, L y puerto F). Cada puerto tiene asociados tres registros localizados en la memoria de datos y para ser programados se requiere de la intervención a dos de los tres registros (Registro de Configuración y Registro de Datos). El registro restante es la dirección reservada para hacer lecturas de cada uno de los Puertos. La siguiente tabla muestra la forma de configuración de los Puertos.

REGISTRO DE CONFIGURACIÓN	REGISTRO DE DATOS	CONFIGURACIÓN DEL PUERTO
0	0	Entrada de Hi-Z
0	1	Entrada en weak pull-up
1	0	Salida en Cero
1	1	Salida en Uno

#### 4.1.2 Registros de configuración

##### Registro ECON (Configuración EPROM)

El Registro ECON es usado para configurar la opción de reloj, seguridad, tamaño en memoria, power-on reset, Watchdog y opción de Halt. El registro puede ser programado y leído únicamente en el modo de programación de EPROM. Por lo tanto, el registro debe ser programado al mismo tiempo que es programada la memoria de programa. El contenido del registro ECON de fábrica es 00 hex (dispositivo de ventana) y el 080 hex (dispositivo OTP).

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
X	POR	SEC	CKI 2	CKI 1	WD	X	Halt

- Bit 7 = x bit reservado por la fábrica. El estado lógico es “Don’t Care.”
- Bit 6 = 1 habilitación Power-on.  
= 0 deshabilitación de Power-on reset.
- Bit 5 = 1 Habilidad del bit de seguridad. Escrituras o lecturas a EPROM no son permitidas.  
= 0 Deshabilitación del bit de seguridad. Es permitido leer y escribir a EPROM.
- Bits 4,3  
= 0, 0 Opción de CKI externa. G7 está disponible como restaurador de HALT y/o de entrada de propósito. CKI es una entrada de reloj.  
= 0,1 Oscilador R/C. G7 es disponible como restaurador de HALT y/o entrada de propósito general. CKI es la entrada de reloj. Los componentes internos R/C tienen los valores apropiados para generar la frecuencia R/C máxima.  
= 1,0 Oscilador por Cristal con la resistencia interna deshabilitada. G7 (CKO) es la salida del generador de reloj al cristal/resonador.  
= 1, 1 Oscilador por Cristal con la resistencia interna habilitada. G7 (CKO) es la salida del generador de reloj al cristal/resonador.
- Bit 2 = 1 WATCHDOG deshabilitado. G1 es E/S de propósito general.  
= 0 WATCHDOG habilitado. G1 es el pin de salida de WATCHDOG con weak pullup.
- Bit 1 = Reservado.
- Bit 0 = 1 modo de HALT deshabilitado.  
= 0 modo de HALT habilitado.

### **Espacio EPROM reservado al usuario**

Existen 8 bytes en EPROM para almacenar información al usuario, los cuales no son afectados por el bit de seguridad del registro *ECON*. Cuando el bit de seguridad del registro *ECON* esta puesto, este espacio puede ser leído o escrito. Este espacio de memoria no es accesible desde la memoria de programa por lo tanto es común utilizar este espacio para almacenar datos como la versión del programa, número serial del Microcontrolador, versión del software, número de lote de producción, etc. El procedimiento para grabar en este espacio de memoria puede ser uno de los siguientes ejemplos :

Información a grabar (como números): 1,2,3,4,5,6,7,8

Comando:

.USER = 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08

Información a grabar (fecha de ensamblado):

DD, MM, YY, 00, HH, MM, SS

Comando:

.USER = ASS\_DATE

Información a grabar (fecha de programación):

DD, MM, YY, 00, HH, MM, SS

Comando:

.USER = PRG\_DATE

### Registro de Control (0x0EE)

El registro de Control contiene bits para manipular las funciones del Timer T1 y Microwire/plus. Está formado por los siguientes bits:

SL1 & SL0 : Seleccionan el divisor de reloj del MICROWIRE/PLUS

(00 = 2, 01 = 4, 1x = 8)

IEDG : Selector de polaridad del flanco de la Interrupción

(0 = Flanco de subida, 1 = Flanco de bajada)

MSEL : Selecciona G5 y G4 como señales MICROWIRE/PLUS SK y SO  
Respectivamente

T1C0 : Inicio/Paro del timer T1 en modo 1 y 2.

Bandera de Interrupción de T1 de 'Underflow' en modo 3 del timer.

T1C1 : Bit de control del Timer T1

T1C2 : Bit de control del Timer T1

T1C3 : Bit de control del Timer T1

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

### Registro de PSW (0x0EF)

El registro de PSW contiene los siguientes bits :

GIE : Global interrupt enable (Habilita interrupciones)

EXEN : Enable external interrupt (Habilita interrupciones externas)

BUSY : Bandera de corrimiento del MICROWIRE/PLUS en ocupado

EXPND : External interrupt pending (Bandera de interrupción externa)

T1ENA : Habilitación de la interrupción del Timer T1 para Underflow ó flanco de captura de T1A.

T1PNDA : Bandera de Interrupción del Timer T1 (Autorecarga RA en modo 1, Underflow de T1 en Modo 2, Flanco de captura de T1A en modo 3)

C : Bandera de Carry (Acarreo)

HC : Bandera de Half Carry (Acarreo del nibble bajo al alto)

La bandera de Half Carry es afectada en todos los casos, por las instrucciones que afectan la bandera de Carry. Las instrucciones de SC y RC pondrán y limpiarán ambas banderas.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

### Registro ICNTRL (0x0E8)

El registro de ICNTRL contiene los siguientes bits:

T1ENB : Habilitador de la interrupción del Timer T1 Interrupt Enable para el flanco de captura del T1B

T1PNDB : Bandera de la Interrupción del Timer T1 para el flanco de captura del T1B

mWEN : Habilitador de la Interrupción del MICROWIRE/PLUS

mWPND: Bandera de la interrupción MICROWIRE/PLUS

TOEN : Habilitador de la interrupción del Timer T0 (Bit 12 conmuta)

TOPND : Bnadera de interrupción del Timer T0

LPEN : Habilitador de la Interrupcion Multi-Input Wakeup del Puerto L

Bit 7 : Puede ser usada como bandera de propósito general

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Sin Uso	LPEN	TOPND	TOEN	mWPND	mWEN	T1PNDB	T1ENB

## 4.2 TIMERS

El dispositivo cuenta con un set muy versátil de timers (T0, T1). El timer T1 es un registro de 16 bits de autorecarga/captura que decremanta en uno cada ciclo de intrucción (tc) independientemente de lo que suceda en el Microcontrolador. Gracias a estos decrementos podemos realizar varias tareas de control.

### Timer T0 (Timer IDLE)

Con el Modo de IDLE, el dispositivo soporta aplicaciones que requieren mantener el tiempo real y bajo consumo de potencia. Este modo de operación se basa en la temporización del timer T0. El timer T0 decremента continuamente en una determinada velocidad de ciclo de instrucción ( $t_c$ ). El usuario no puede leer ni escribir al timer de IDLE (Timer T0). El timer T0 es afectado por las siguientes instrucciones:

- Salida del modo de Idle
- Lógica WATCHDOG
- Retardo de inicialización en el modo de HALT
- Temporizando la duración del power-on-reset interno.

El timer T0 puede generar una interrupción cuando el 12vo bit conmute de estado. Esta conmutación es capturada por la bandera T0PND, y se presentará cada 4,096 ms en la frecuencia máxima de reloj ( $t_c = 1$  ms). Una bandera de control TOEN permite habilitar o deshabilitar las interrupciones del 12vo bit del timer

### Timer T1

Una de las principales funciones del Microcontrolador es proveer capacidad de temporización y conteo para tareas de control en tiempo real. El COP8 ofrece un timer muy versátil de 16 bits de autocarga/recarga con dos registros asociados (R1A y R1B) optimizados para reducir código. Cada registro asociado tiene la disponibilidad de un pin T1A y T1B.

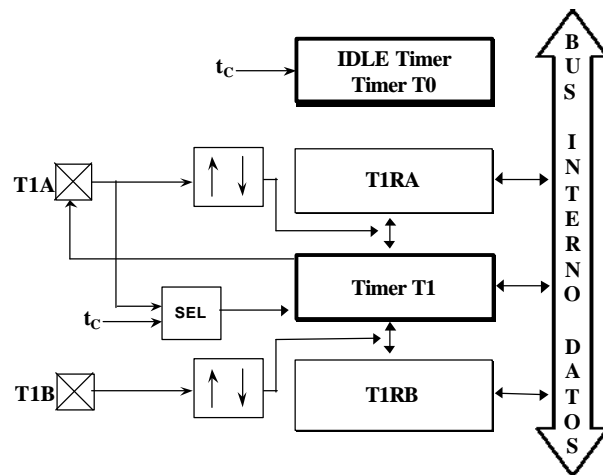


Figura 16 Estructura del Timer T1

El bloque de timer tiene tres modos de operación: Modo PWM, Modo Contador de Eventos Externos y Modo de Captura.

### Mode 1. Modo PWM

En este modo el timer genera una señal PWM “independiente del procesador” dado que una vez que el timer es configurado, no requiere de acción alguna del

CPU. El usuario tiene que intervenir el timer únicamente para cambiar los parámetros de la señal PWM configurados en el bloque de timer. Se tiene esta capacidad gracias a los dos registros de 16 bits asociados al timer. El registro R1A contiene el tiempo de “alto” mientras que el R1B contiene el tiempo “bajo” de la señal PWM. El timer puede generar la salida PWM con la frecuencia y el ciclo de trabajo controlados por los valores almacenados en los registros de recarga. Los registros de recarga controlan el valor de la cuenta regresiva. Estos valores son copiados al registro independiente de 16 bits.

En este modo, el timer T1 cuenta decrementando en uno su valor actual cada vez que hay un ciclo de instrucción ( $t_c$ ). A cada underflow el registro timer carga ya sea el contenido de R1A ó R1B. Los bits de control del Timer son T1C3, T1C2 y T1C1. Los Underflows pueden ser programados para que el pin T1A conmute y así mismo puede ser programado para que genere una Interrupción.

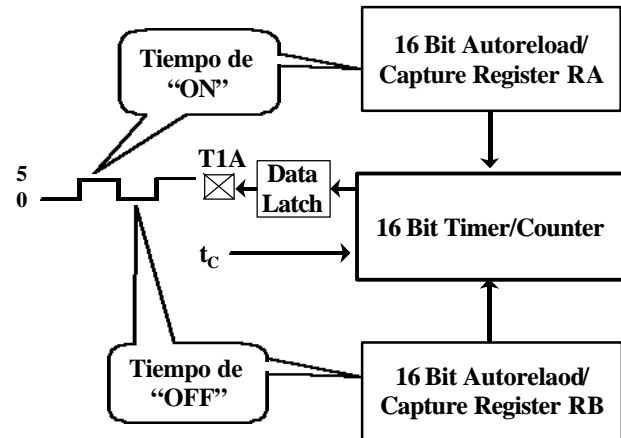


Figura 17 Diagrama del Timer en PWM

Los underflows del timer pueden ser alternativamente detectados con la presencia de las banderas T1PNDA y T1PNDB. El usuario debe limpiar estas banderas por software. Hay que conciderar las banderas de hailitación T1ENA y T1ENB para que permitan que las interrupciones sean atendidas o no. Por ejemplo, si ponemos la bandera de T1ENA, se causará una interrupción cuando el timer “underflows” el registro R1A, lo mismo pasa con T1ENB. Ninguna o ambas interrupciones pueden ser habilitadas. Esto le da al usuario la flexibilidad the recibir una interrupción una vez por periodo PWM en cualquiera de los flancos (positivo o negativo). Alternativamente, el usuario puede escoger la interrupción en ambos flancos de la salida PWM.

## Modo 2. Modo de Contador de Eventos Externos

Este modo de operación es muy similar al modo de PWM. La diferencia principal es que en éste modo el timer T1, es sincronizado por la señal de entrada al pin T1A. Los bits de control T1C3, T1C2 y T1C1 permiten la selección de los flancos a capturar, ya sea flancos positivos o negativos. Los Underflows del timer son capturados por la bandera T1PNDA y en el caso que la bandera de control T1ENA este puesta, el underflow ocasionará una

interrupción. En éste modo el pin de entrada T1B puede ser utilizado como una entrada independiente capaz de detectar flancos positivos (levantando la bandera de T1PNDB) y causar una interrupción en el caso que la bandera T1ENB se encuentre puesta.

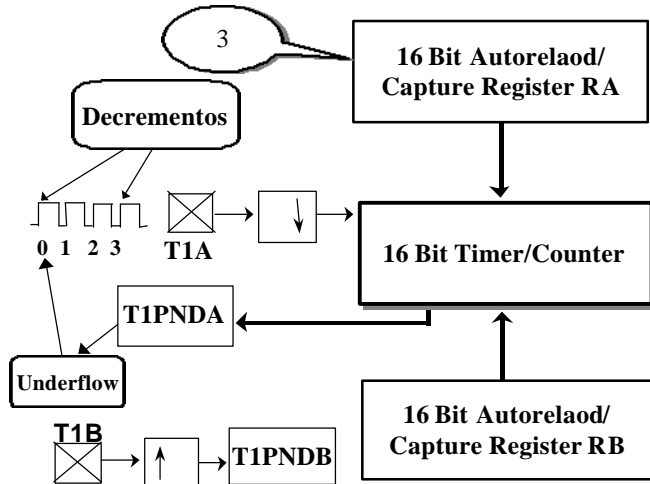


Figura 18 Timer en Contador Eventos Ext.

En la Figura 18 se muestra el modo de operación del timer en contador de eventos externos. En este caso se desea capturar cinco flancos negativos y al término de estos, se presenta una interrupción. El pin T1B se encuentra preparado para detectar cualquier flanco positivo en la terminal.

### Mode 3. Modo de Captura

El modo de captura es utilizado para medir la frecuencia o el tiempo de algún evento externo. Los registros de recarga RA y RB se utilizan como registros independientes de captura del contenido del Timer cuando un evento externo se presenta (transición en el pin de entrada del Timer).

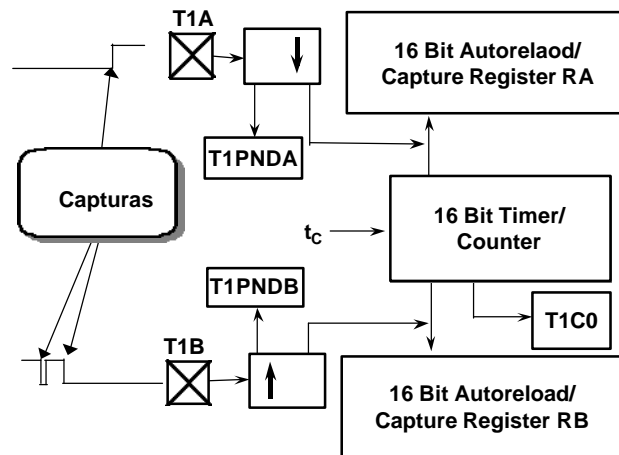


Figura 19 Timer en Modo de Captura

Los registros de captura pueden ser leídos mientras se mantiene la cuenta, una característica que permite al usuario medir el tiempo transcurrido entre eventos. La mayoría de los Microcontroladores tienen un tiempo de retardo porque no se puede determinar el valor del timer cuando un evento externo se presenta. El registro de captura elimina el tiempo de retardo, entonces es posible capturar el valor del timer en el registro de recarga cuando se presenta un evento. El timer T1 está decrementando cada ciclo de instrucción. Los registros R1A y R1B actúan como registros de captura. Cada registro actúa en conjunción con

el pin asociado (R1A con el pin T1A y R1B con el pin T1B). El valor del timer es copiado en el registro de recarga cuyo pin asociado detecte un flanco. Los bits de control T1C3, T1C2 y T1C1 determinarán que flanco será capturado (flanco positivo o negativo). Cada captura hace que las banderas T1PNDA y T1PNDB sean puestas, así mismo se puede generar una interrupción si las banderas correspondientes T1ENA y/o T1ENB se encuentran en uno. Underflows del timer puede ser programado para generar una interrupción levantando la bandera de T1C0. Consecuentemente, el bit de control T1C0 debe ser limpiado cuando se opera en modo de captura.

### Banderas de control del Timer

Los bits de control del timer y sus funciones son los siguientes:

- T1C0: Timer Start/Stop control in Modes 1 and 2 (Processor Independent PWM and External Event Counter), where 1 = Start, 0 = Stop  
Timer Underflow Interrupt Pending Flag in Mode 3 (Input Capture)
- T1PNDA: Timer Interrupt Pending Flag
- T1PNDB: Timer Interrupt Pending Flag
- T1ENA: Timer Interrupt Enable Flag
- T1ENB: Timer Interrupt Enable Flag
  - 1 = Habilitación de la interrupción del Timer
  - 0 = Deshabilitación de la interrupción del Timer
- T1C3: Control del Modo de Operación del Timer
- T1C2: Control del Modo de Operación del Timer
- T1C1: Control del Modo de Operación del Timer

Las opciones de los bits de control (T1C3, T1C2 and T1C1) son detalladas en la siguiente tabla:

### 4.3 MICROWIRE / PLUS

Microwire/Plus es una interface de comunicación síncrona la cual permite establecer comunicación con otros dispositivos periféricos que cuenten con esta misma interfase o la interfase SPI. Ya sea ADCs, PLLs o incluso con otro Microcontrolador

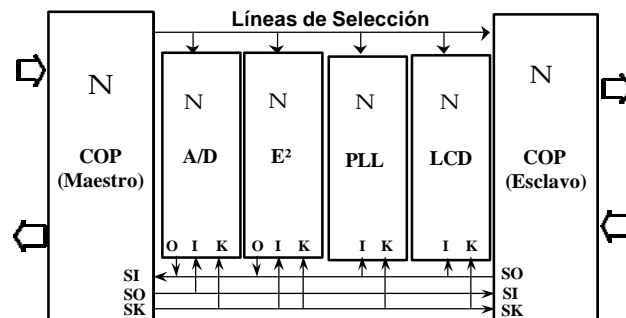


Figura 20 Diagrama en HW del Microwire

El Microwire/Plus consiste en un registro de corrimiento de 8 bits (SIO) con una entrada de datos serial (SI), una salida de datos (SO) y un reloj (SK).

El reloj puede ser tomado de una fuente interior o exterior. Cuando el Microcontrolador toma la señal de reloj del exterior se debe configurar en *modo esclavo*. Cuando la señal de reloj es generada internamente el Microcontrolador trabaja en *modo maestro*. El registro CNTRL es usado para configurar y controlar el modo de operación MICROWIRE/PLUS. El bit MSEL en el registro de CNTRL es el habilitador del MICROWIRE/PLUS. En el modo maestro, la velocidad de reloj es seleccionada por dos bits (SL0 y SL1) en el registro de CNTRL.

SL1	SL0	SK period
0	0	2xTc
0	1	4xTc
1	X	8xTc

### **Operación del MICROWIRE/PLUS.**

Cuando se pone el bit de BUSY en el registro de PSW, la unidad de MICROWIRE/PLUS comienza a recorrer los datos. El usuario puede limpiar el bit de BUSY por software para recorrer un número menor de 8 bits. Si se encuentra habilitada, la interrupción es generada cuando los ocho bits han sido recorridos. El registro SIO debe ser cargado únicamente cuando el reloj SK se encuentra en la fase de inactivo (idle). De otra forma, el valor del dato cargado será indefinido. Por seguridad (en Modo de Esclavo), la bandera de BUSY debe ser puesta únicamente cuando la entrada del reloj SK se encuentra en la fase de inactivo.

### **Operación en modo maestro del MICROWIRE/PLUS**

En el Modo Maestro del MICROWIRE/PLUS el reloj (SK) es generado internamente. El Microcontrolador Maestro siempre inicializa las transferencias de datos. El bit MSEL en el registro de CNTRL debe ser puesto para habilitar las funciones de SO y SK en el puerto G. Los pines SO y SK deben ser configurados como salidas.

### **Operación en modo de esclavo de MICROWIRE/PLUS**

El Modo Esclavo del MICROWIRE/PLUS la señal de SK es generada por una fuente externa. El bit MSEL en el registro de CNTRL debe ser puesto para habilitar las funciones de SO y SK en el puerto G. El pin de SK debe ser

configurado como entrada y el pin de S0 es configurado como pin de salida. La siguiente tabla ilustra las posibilidades de configuración. El usuario debe de poner la bandera de Busy inmediatamente después de entrar al modo de esclavo. Con esto se asegura que los bits mandados por el maestro son recibidos apropiadamente. Después de los 8 pulsos del reloj, la bandera de BUSY es limpiada y los pulsos del reloj ya no se presentan. Esta secuencia se puede repetir las veces que sea necesario.

G4 (SO)	G5 (SK)	G4 Func	G5 Func	Operation
1	1	SO	Int. SK	Maestro MICROWIRE/PLUS
0	1	3er Edo.	Int. SK	Maestro MICROWIRE/PLUS
1	0	SO	Ext. SK	Esclavo MICROWIRE/PLUS
0	0	3er Edo.	Ext. SK	Esclavo MICROWIRE/PLUS

#### 4.4 MODOS DE AHORROS DE ENERGÍA

Hoy en día tenemos un sin fin de dispositivos operados por baterías y consecuentemente demandan un consumo de potencia muy bajo. Estos dispositivos no son los únicos que requieren de bajo consumo de potencia, existen aplicaciones como las industriales y automotrices donde se requiere de un consumo regulado muy bajo para reducir costos. Por consecuencia, los Microcontroladores tienen, en su mayoría, la facultad de entrar a modos de ahorro de energía. El COP8 tiene la característica de poder entrar en modos de operación de bajo voltaje, bajo consumo de corriente y ahorro de potencia como los modos de HALT y de IDLE. En el Modo de Halt, toda actividad interna en el Microcontrolador es detenida. En el Modo de Idle el oscilador interno y el timer T0 permanecen activos, sin embargo, todas las demás actividades internas son detenidas. En cualquiera de ambos modos, el contenido de la RAM, registros, estados de puertos y timers (excepto el T0) son inalterados. Si se encuentra habilitado, el monitor de reloj puede permanecer activo en ambos modos.

##### 4.4.1 Modo de ahorro de energía HALT

El Microcontrolador puede ser forzado a entrar al modo de operación HALT con solo escribir un "1" en la bandera de HALT (bit de datos G7). Todas las actividades internas, incluyendo el reloj y timers son detenidas. La lógica de Watchdog es deshabilitada durante este modo. Sin embargo, el monitor de reloj puede permanecer activo y tendremos el pin de WATCHDOG (WDOUT) un estado lógico bajo.

Si el modo de HALT es utilizado y no se desea que el pin WDOOUT tenga un cero lógico, el monitor de reloj deberá de ser desactivado después de que el Microcontrolador salga de un Reset. En el modo de HALT, los requerimientos de potencia son mínimos. Es decir el voltaje de alimentación ( $V_{cc}$ ) puede bajar hasta ( $V_r = 2.0V$ ) sin alterar el estado del integrado.

El dispositivo puede salir del modo de HALT por cualquiera de tres vías. La primera (más usual) es la utilizada por la característica de Multi-Input Wakeup. La segunda es un cambio de bajo a alto (flanco positivo) en el pin CKO (G7).

Hay que tener en cuenta que esta forma de salida del Halt puede ser aplicada únicamente con una temporización de R/C o un generador externo ya que con un cristal externo el pin CKO se configura como salida. El tercer método para salir del modo de HALT es llevando al pin de RESET a un estado bajo.

Dado que tanto el Cristal como el R/C tienen un tiempo de espera para que la señal de oscilación sea la apropiada en magnitud y estabilidad en frecuencia, el Microcontrolador, al salir del wakeup, no ejecutará acción alguna sino hasta después de un tiempo razonable de espera.

Por otro lado el timer de IDLE es usado para generar un retardo establecido para asegurar que el oscilador se ha estabilizado antes de ejecutar una instrucción. En este caso, en vez de detectar la activación de la señal de Wakeup, únicamente la circuitería del oscilador es habilitada. (Observar Figura del Wakeup del Halt)

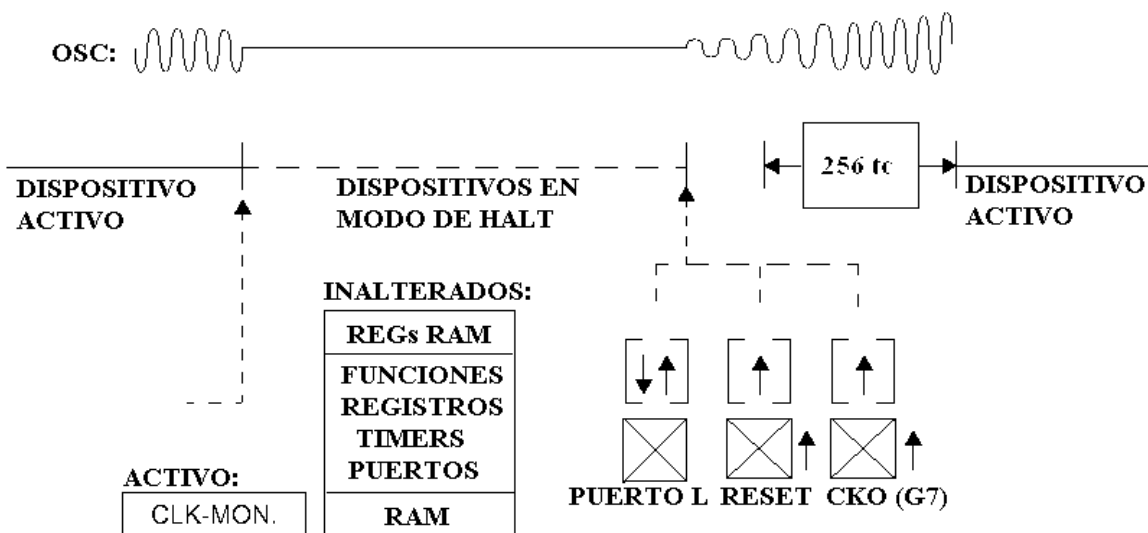


Figura 21 Entrada y Salida del modo de HALT

El bit 0 del registro de ECON contiene la opción de habilitar o deshabilitar el modo de HALT. El detector de Watchdog es inhibido durante el modo de HALT para permitir la operación del monitor de reloj.

#### 4.4.2 Modo de ahorro de energía IDLE

El Microcontrolador es puesto en modo de IDLE cuando se escribe un “1” a la bandera de IDLE (Bit de datos G6). En este modo, toda actividad es detenida internamente, excepto las asociadas con la circuitería del oscilador interno y el Timer de IDLE T0.

Similarmente al modo de HALT, el dispositivo puede retornar a la operación normal con un reset, o con el Puerto Multi-Input Wakeup.

Alternativamente, el Microcontrolador retorna a la operación normal cuando el doceavo bit (representando 4.096ms con una frecuencia de reloj de 10 MHz,  $T_c = 1\mu s$ ) del timer IDLE Timer conmuta. Esta conmutación es capturada por la bandera de T0PND. El usuario tiene la opción de ser interrumpido por una transición del doceavo bit del Timer IDLE siempre y cuando esta interrupción haya sido habilitada por el bit e control T0EN (TOEN=1).

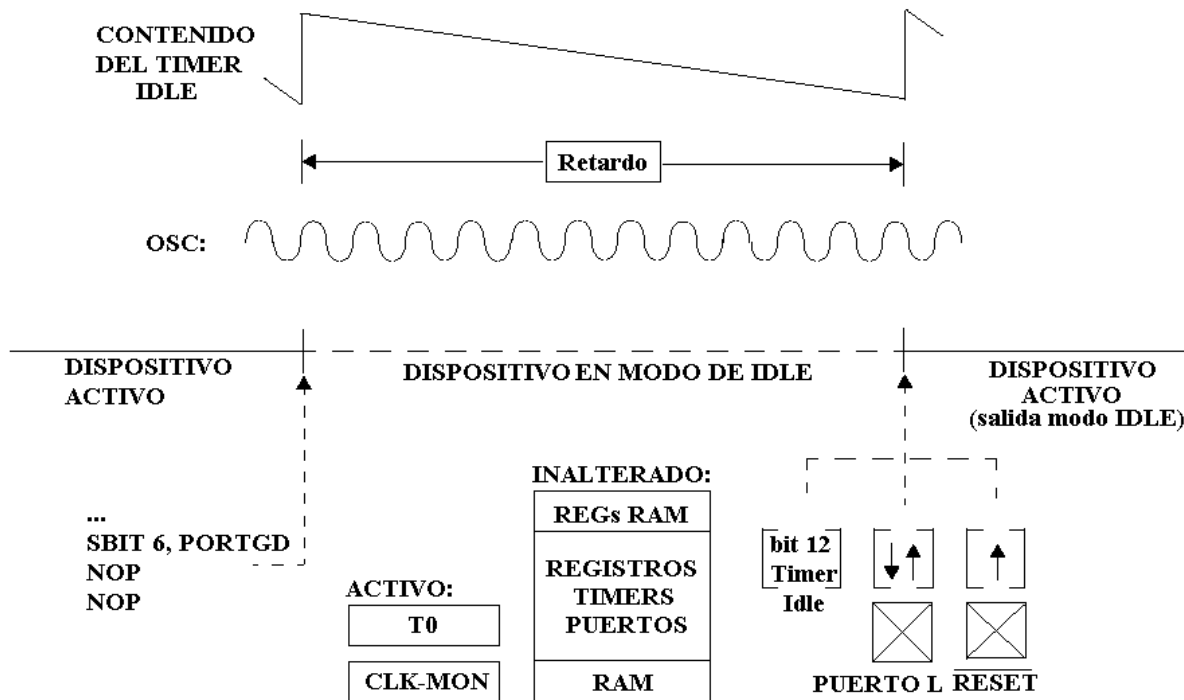


Figura 22 Entrada y Salida del Modo de IDLE

### 4.4.3 Multi-input wakeup

La característica del Multi-Input Wakeup es utilizada para regresar (despertar) al Microcontrolador del modo de operación de HALT o IDLE. Alternativamente la característica Multi-Input Wakeup puede ser empleada para abrir la posibilidad de detectar 8 flancos seleccionables como interrupciones.

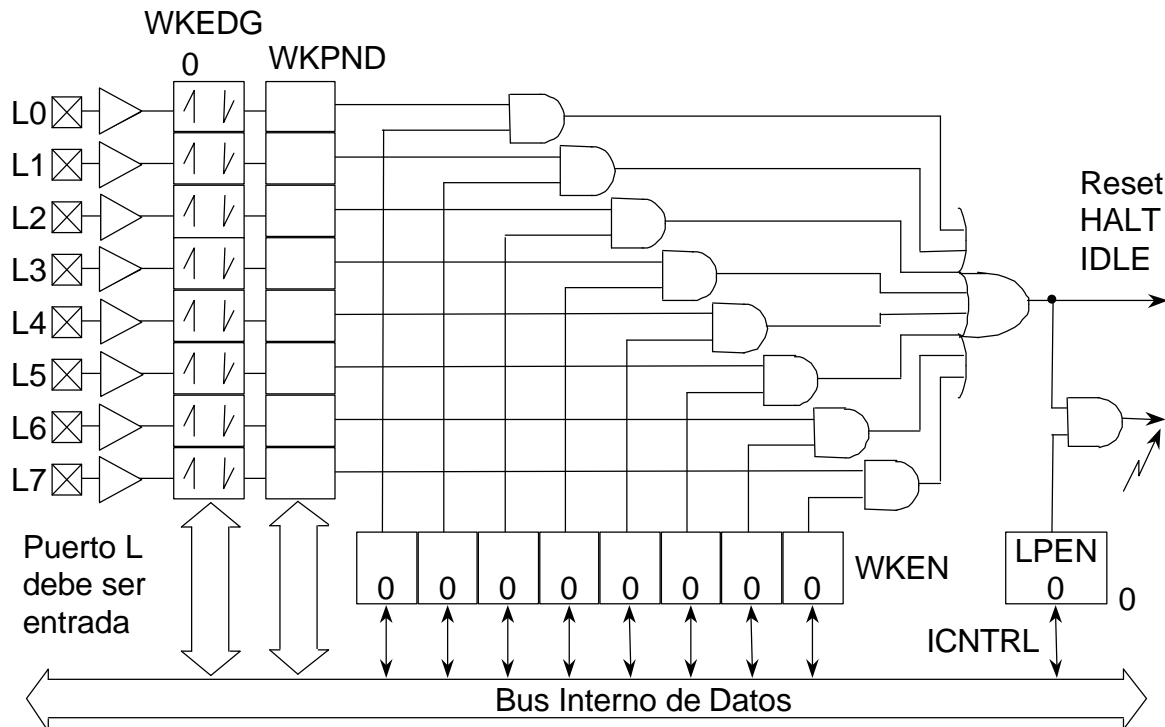


Figura 23 MULTI-INPUT WAKEUP

Multi-Input Wakeup (MIWU) permite que los diseñadores especifiquen cual de los 8 pines del Microcontrolador serán utilizados para “despertar” al COP8 y procesar instrucciones. Esta capacidad es activada durante un cambio de estado lógico en el pin de entrada (flancos positivos o negativos) los cuales son capturados internamente en registros. Por lo tanto, se puede configurar la detección del flanco esperado ya sea positivo, negativo o inclusive ambos flancos en el pin (o pines) elegidos. Sin el Multi-input wakeup, el Microcontrolador tendría que mantener la ejecución de código y por lo tanto un alto consumo de potencia.

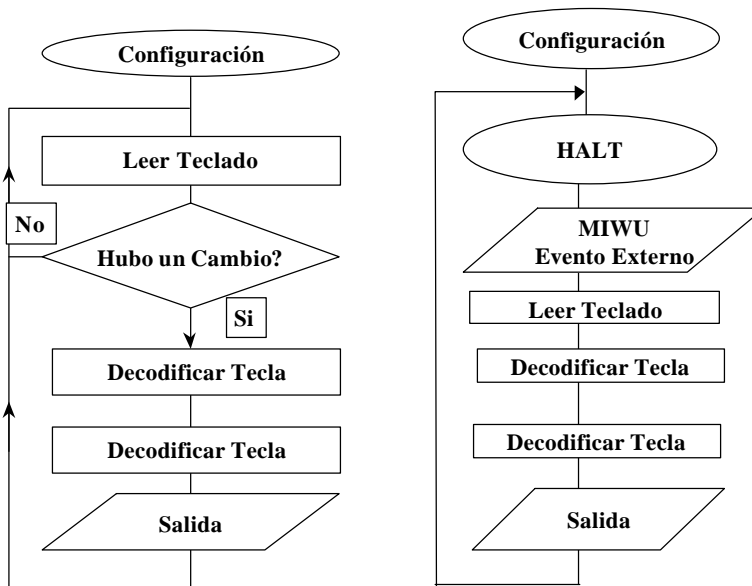


Figura 24 Aplicación con MIWU/Teclado

Típicamente el MIWU reduce el consumo de corriente hasta 4mA en estados de bajo consumo comparados a los 6 mA de consumo si el código se mantuviera corriendo. Por ejemplo en una aplicación de un celular o PC portátil podemos tener 2 formas de decodificación de teclado. Observar figura mostrada a la izquierda (Figura 24).

El puerto L es comúnmente asociado al Multi-input Wakeup. En el ejemplo anterior las líneas del teclado estarían conectadas como entradas al Puerto L. El Microcontrolador se encontrará en modo de HALT hasta que una tecla sea presionada haciendo que el Micro salga del modo HALT y decodifique que tecla fue presionada y haga el proceso correspondiente. Cuando la operación es completada el Microcontrolador regresará al modo de HALT nuevamente. La selección del MIWU es realizada por medio del registro WKEN donde podremos encontrar el bit que habilita la característica de Wakeup.

#### 4.5 INTERRUPCIONES

Una interrupción es aquel evento que, al presentarse, ofrece la alternativa de realizar momentáneamente una tarea en especial y al concluirla, continúa con la ejecución normal de la secuencia del programa. El número de tareas comúnmente son identificadas como número de fuentes de Interrupción. Cada fuente de interrupción tiene un llamado vector de interrupción, el cual contiene la dirección a saltar cuando se presenta la interrupción correspondiente. El Microcontrolador COP8SA maneja hasta 8 vectores de interrupciones. Las fuentes de interrupción incluyen: Timer 1, Timer T0, Puerto L Wakeup, Software Trap, MICROWIRE/PLUS, y Entrada Externa. Todas las interrupciones forzan un salto a la localidad 00FF Hex en Memoria de Programa. La instrucción VIS puede ser utilizada para vectorizar hacia la rutina de servicio apropiada.

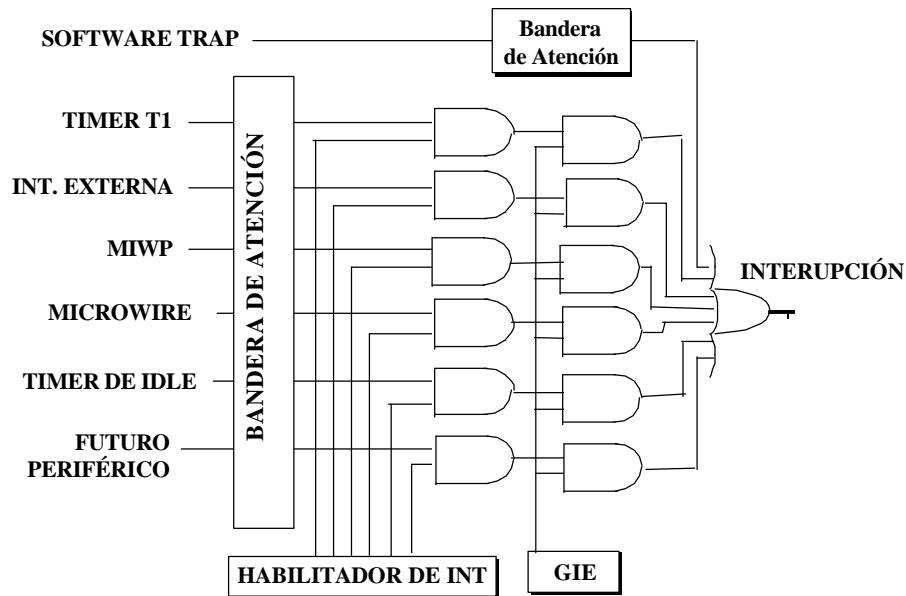


Figura 25 Arquitectura de las Interrupciones

#### 4.5.1 Interrupciones Enmascarables

Todas las interrupciones son enmascarables (excepto Software Trap). Cada interrupción enmascarable tiene un bit habilitador y una bandera de un bit. Todas las interrupciones enmascarables, tanto los bits habilitadores como los bits de bandera, se encuentran localizados en los registros de control en memoria y por lo tanto pueden ser controlados por software.

Una interrupción enmascarable se presenta bajo las siguientes circunstancias:

1. Habilitación del bit asociado con la interrupción
2. Habilitación del bit GIE (El cual habilita todas las interrupciones)
3. El dispositivo no se encuentra procesando una interrupción no enmascarable. (Si una interrupción no enmascarable está siendo atendida, una interrupción enmascarable deberá esperar a que la rutina de servicio sea completada.)

Una interrupción se presenta únicamente bajo estas circunstancias al inicio de una instrucción. Si más de una interrupción enmascarable tiene todas las circunstancias necesarias para ser atendida y se presentan simultáneamente, la interrupción de mayor prioridad será atendida en primer lugar. Las otras interrupciones deberán esperar a que la interrupción de más alta prioridad sea atendida y servida. Después del Reset, todas las banderas y el bit GIE son puestos en cero. Por lo tanto no se puede presentar una interrupción si no es

configurada de forma general (poniendo GIE en uno) y de forma particular (poniendo el bit de interrupción en uno). Al comienzo de una interrupción las siguientes acciones se presentan :

1. El bit GIE es automáticamente puesto en cero, previniendo que alguna otra interrupción interrumpa a la rutina de servicio actual. Esta característica previene que una interrupción enmascarable que está siendo atendida sea interrumpida por alguna otra interrupción.
2. La dirección de la instrucción por ser atendida es salvada en la memoria de datos tipo pila (stack).
3. El registro Contador de Programa (PC) es cargado con el dato 0FF Hex, ocasionando un salto a esa dirección de memoria de programa. El dispositivo requiere siete ciclos de instrucción para realizar las acciones mencionadas.

Si el usuario desea “permitir” que se presente una interrupción mientras se atiende otra interrupción, deberá poner en alto el bit de GIE en el registro PSW teniendo precaución de no sobrepasar la capacidad de almacenamiento de direcciones en la pila (stack), pérdida de información y condiciones no deseadas. Cada vez que se presenta una interrupción específica, la bandera de interrupción deberá ser limpiada (puesta en cero).

Típicamente esto es realizado lo más rápidamente posible en la rutina de servicio para evitar que la siguiente interrupción se pierda. Por ejemplo, si una interrupción de timer se presenta y se limpia la bandera correspondiente del timer en la rutina de servicio y casi al mismo tiempo se vuelve a presentar la misma petición de interrupción del timer por segunda vez (cuando aún no termina de realizarse completamente la primera rutina de servicio) ésta última petición esperará hasta que se termine la ejecución de la subrutina por completo para empezar inmediatamente a realizar la misma subrutina otra vez.

Típicamente una rutina de servicio termina con la instrucción RETI. Esta instrucción pone automáticamente el bit GIE en uno, restaura la dirección guardada en la pila (stack) y almacena esta dirección en el apuntador de Memoria de Programa. La ejecución del programa procede entonces con la ejecución de la siguiente instrucción que hubiera sido ejecutada en la secuencia normal del programa de no presentarse la interrupción. Si alguna otra interrupción tiene una bandera en alto, la interrupción de más alta prioridad será servida inmediatamente después del retorno de la interrupción previa.

## 4.5.2 Interrupción VIS

La rutina de servicio general, la cual comienza en la dirección 0FF Hex, debe ser capaz de manejar todas las interrupciones. La instrucción VIS junto con una tabla vector de interrupciones, dirige al Microcontrolador a la rutina de servicio de la interrupción específica que haya causado la interrupción.

La instrucción de un byte VIS es usada típicamente al inicio de la rutina de servicio general en la dirección 0FF Hex. Es la instrucción VIS la encargada de determinar cual es la interrupción que tiene la bandera en uno, se encuentra habilitada y tiene la más alta prioridad. Una vez identificada se realiza un salto indirecto a la dirección correspondiente a la rutina de servicio de la interrupción.

Las direcciones de salto (vectores) para todas las posibles fuentes de interrupción son almacenadas en una tabla de vectores. La tabla de vector puede ser de un largo de 32 bytes (máximo de 16 vectores) y reside en la parte alta del bloque de 256 bytes incluyendo a la instrucción VIS. Sin embargo, si la instrucción VIS se encuentra al inicio del bloque de 256 bytes (Dirección 00FF Hex) y consecuentemente si la instrucción VIS esta localizada en algún lugar entre 00FF y 01FD (caso usual), la tabla de vectores se encuentra entre las localidades 01E0 y 01FF Hex. Si la instrucción VIS se encuentra entre las direcciones 01FF y 02DF, el vector se encontrará entre las direcciones 02E0 y 02FF Hex, así consecutivamente.

Cada vector es de 15 bits de longitud y apunta al inicio de la rutina de servicio de una interrupción específica. Cada vector ocupa dos bytes de la tabla de vectores (primero el byte de la dirección alta seguido del byte de la dirección baja) y se encuentran ordenados por prioridad (la interrupción de más alta baja prioridad se encuentra en la dirección 0yE0-0yE1 y las siguientes en las localidades siguientes más altas).

La interrupción de Software Trap es la interrupción de más alta prioridad localizada en la dirección 0yFE-0yFF. El tabla de vectores deberá ser llenada por el usuario con las localidades de memoria de la rutina de servicio correspondiente. Corresponde al programador tener el cuidado suficiente para respetar la correspondencia de la dirección-interrupción-rutina de servicio.

<b>Rango de Prioridad de la Interrupción</b>	<b>Fuente</b>	<b>Descripción</b>	<b>Vector Dirección (Byte alto-bajo)</b>
(1) Más Alta	Software Trap	Instrucción INTR	0yFE – 0yFF
(2)	Reservado	Futuro	0yFC – 0yFD
(3)	Externa	G0	0yFA – 0yFB
(4)	Timer T0	Underflow	0yF8 – 0yF9
(5)	Timer T1	T1A/Underflow	0yF6 – 0yF7
(6)	Timer T2	T1B	0yF4 – 0yF5
(7)	Microwire/PLUS	Busy Low	0yF2 – 0yF3
(8)	Reservado	Futuro	0yF0 – 0yF1
(9)	Reservado	Futuro	0yEE – 0yEF
(10)	Reservado	Futuro	0yEC – 0yED
(11)	Reservado	Futuro	0yEA – 0yEB
(12)	Reservado	Futuro	0yE8 – 0yE9
(13)	Reservado	Futuro	0yE6 – 0yE7
(14)	Reservado	Futuro	0yE4 – 0yE5
(15)	Puerto L/Wakeup	Flanco Puerto L	0yE2 – 0yE3
(16) Más Baja	Dafault	Intrucción VIS	0yE0 – 0yE1

Tabla de los Vectores de Interrupción

Por ejemplo si la rutina de servicio de interrupción de Software Trap está localizada el la 0310 Hex (normalmente identificada por una etiqueta), entonces el vector 0yFE-0yFF Hex deberá contener el dato 03-10 Hex respectivamente. Si la instrucción VIS es ejecutada sin existir haber una solicitud de interrupción, el vector de interrupción de más baja prioridad será utilizado. Es por esto, que se recomienda que de no ser utilizada la interrupción de más baja prioridad, su vector apunte a la rutina de servicio del Software Trap.

La instrucción RETI debe de estar presente en cada surrutina para que la pila retorne valores correctos en la lógica el programa. También es muy recomendable utilizar siempre la instrucción VIS, la alternativa de ploeo de las banderas de interrupción es también otro recurso muy valido (para cambiar las prioridades de las interrupciones) pero de esta forma la confiabilidad del programa esta sujeta totalmente al usuario.

## **Bandera de Interrupción no enmascarable**

Existe una bandera asociada con la interrupción no enmascarable llamada STPNP. Esta bandera no se encuentra mapeada en memoria y no puede ser accesada directamente por software. La bandera STPNP es limpiada cuando el dispositivo sale de un Reset. Cuando la interrupción no enmascarable se presenta, la bandera STPNP es puesta en uno. La rutina de servicio correspondiente deberá tener la instrucción STPNP para limpiar bandera.

## **Software Trap**

El Software Trap es una interrupción especial la cual ocurre cuando la instrucción INTR (Usada para generar interrupción) es cargada de la memoria de programa y colocada en el registro de instrucción. Esto puede suceder de varias formas, usualmente por una condición de error. Por ejemplo :

Si el (PC) apunta incorrectamente a una localidad de memoria más allá del espacio disponible en la memoria de programa. Las localidades no usadas o inexistentes retornan ceros, con lo cual es interpretado como una instrucción INTR.

Si la pila retorna datos (popped) más allá del límite (dirección 02F ó en otras versiones 06F Hex), el Software Trap será solicitado.

También la bandera de Software Trap puede ser disparada por una condición temporal de hardware un brownout o una falla en la fuente de alimentación.

Una vez que el dispositivo entra a la rutina de servicio del Software Trap, es recomendable inicializar la pila (stack) y realizar un procedimiento de recuperación que reinicie el software a un punto determinado similar al Reset más no necesariamente igual.

## **Interrupciones en el Puerto L**

El Puerto L provee al usuario ocho interrupciones adicionales capaces de detectar flancos positivos o negativos los cuales son atendidos por la misma subrutina de servicio. El Puerto L comparte las funciones de wake up con estados lógicos entrada/salida de propósito general. El registro WKEN permite que las interrupciones del Puerto L sean individualmente habilitadas o deshabilitadas mientras que el registro WKEDG especifica el flanco a ser

detectado (positivo o negativo). El registro WKPND contiene las banderas de los flancos detectados. La bandera LPEN funciona como un habilitador de interrupciones global para el Puerto L. Dado que el Puerto L es usado también para salir de un modo de operación HALT o IDLE, el usuario puede seleccionar estas salidas con o sin las interrupciones habilitadas. Si se deshabilita la interrupción, el dispositivo reiniciará ejecutando la instrucción inmediata siguiente a la que estableció la entrada al modo del HALT o IDLE. Si la interrupción se encuentra habilitada, el dispositivo ejecutará la rutina de servicio y después retomará la ejecución normal.

### **Sumario de Interrupciones.**

El Microcontrolador tiene las siguientes características en Interrupciones, listado en orden de prioridad:

1. Interrupción no mascarable de Software Trap, ocasionada por la instrucción INTR (Opcode 00). El Software Trap es reconocido inmediatamente y su subrutina solo puede ser interrumpida por la detección de otro Software Trap. El Software Trap debe terminar con la instrucción RPND seguido de un procedimiento de reinicio.
2. Interrupciones Enmascarables, ocasionadas por alguna unidad interna o externa del Microcontrolador. Bajo condiciones ordinarias, una interrupción enmascarable no interrumpirá cualquier otra rutina de interrupción en proceso (a excepción de que se presente una interrupción no enmascarable). Una interrupción enmascarable debe concluir en su rutina de servicio con una instrucción RETI.

## **Autoevaluación del CAPITULO 4**

### **Ejercicio**

1. Si al revisar un programa se encuentra la siguiente secuencia :

```
; los datos son 1 2 3 4 5 6 7 8  
. =0x4F9  
.BYTE 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08  
.USER=0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
```

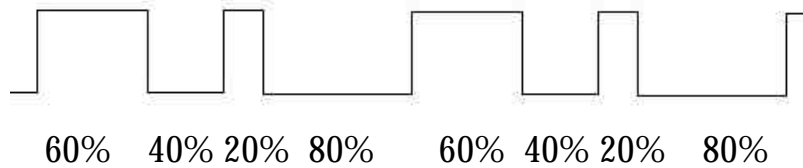
Qué es lo que está buscando el autor del programa realizar ?

## Para investigar

1. ¿ Cómo puede ser configurado un Puerto ?
2. ¿ Explica para que es utilizada la configuración weak pull-up ?
3. ¿ De forma general, describe que se puede configurar en el registro ECON ?
4. ¿ De cuántas formas puede ser configurado un Timer del COP8?
5. ¿ Explica que es una señal PWM y cuáles son sus aplicaciones ?
6. ¿ Describe brevemente el funcionamiento interno del Timer en Modo Contador de Eventos externos ?
7. ¿ Qué utilidad tiene el poder capturar Flancos ?
8. ¿ Qué es el Microwire ?
9. ¿ Qué diferencia hay entre un Microcontrolador Maestro y Esclavo ?
10. ¿ Qué se entiende por un modo de ahorro de energía y cuántos modos de ahorro de energía tiene el COP8 ( explicar brevemente c/u ) ?
11. ¿ Explica que es una interrupción ?
12. ¿ Qué es Power-on Reset ?
13. ¿ Qué diferencia existe entre una interfase serial síncrona y asíncrona ?
14. ¿ Por qué razón cuando el COP8 con un cristal de 10MHz entra al modo de ahorro de energía HALT, no puede retornar al modo de operación normal con una transición con un flanco positivo en G7. ?
15. ¿ Qué consumo en corriente tiene el Microcontrolador en modo de operación normal, en modo de ahorro de energía HALT y en modo de ahorro de energía IDLE ?
16. ¿ Describe brevemente la secuencia de la instrucción VIS ?
17. ¿ Describe y explica las circunstancias necesarias para que una interrupción enmascarable se presente ?
18. ¿ Describe las acciones que realiza el microcontrolador cuando una interrupción comienza ?
19. ¿ Qué es el software trap ?
20. ¿ Describe la característica Multi – input Wake up ?

**Prácticas** \* Para realizar éstas prácticas es necesario cubrir el capítulo 6

1. Desarrollar un programa el cual genere una señal PWM en el pin T1A. La señal deberá tener un ciclo de trabajo de 60us en alto y 40us en bajo ( sin usar interrupciones)
2. Desarrollar un programa que genere una señal PWM y alterne entre los ciclos de trabajo de 60/40 y 20/80. La secuencia deberá presentar una señal en alto de 60us, una señal en bajo de 40us, una señal en alto de 20us y una señal en bajo de 80us sucesivamente.



3. Desarrollar un programa que sea capaz de detectar 10 eventos externos (flancos negativos) y simultáneamente el puerto D despliegue la cuenta del evento actual en formato binario.

1er evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1110 bin (0x0FE hex)
2do evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1101 bin (0x0FD hex)
3er evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1100 bin (0x0FC hex)
4to evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1011 bin (0x0FB hex)
5to evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1010 bin (0x0FA hex)
6to evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1001 bin (0x0F9 hex)
7mo evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 1000 bin (0x0F8 hex)
8vo evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 0111 bin (0x0F7 hex)
9no evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 0110 bin (0x0F6 hex)
10mo evento externo ( cambio de un estado alto a un estado lógico bajo )	Puerto D : 1111 0101 bin (0x0F5 hex)

4. Desarrollar un programa que comunique al Microcontrolador COP8 vía Microwire/Plus a una Memoria EEPROM NM93Cxx. El programa debe tener residente en la memoria ROM una tabla de datos con el siguiente mensaje (Practica 4 de Microcontroladores UNIVA 2000). Esta tabla de datos debe ser copiada a la memoria EEPROM.
5. Optimizar el siguiente programa ( con la instrucción JID y/o utilizar Multi-input Wakeup)

; --- Aplicacion : Lectura de un teclado y desplegado en un display de 7 segmentos ---

```
; ----- Definicion de variables
.sect localdata, ram
digito:      .dsb   1
.endsect
; ----- Programa Principal
.sect  text,rom
main:
    JSR  display          ; subrutina mostrar tecla
    JSR  lee_teclado     ; subrutina leer teclado
    JP   main
; -----
lee_teclado:
    LD   PORTLC, #B'00011111
    LD   PORTLD, #B'11100001
    LD   A,PORTLP
    IFEQ A,#B'11100001
    jp   no_tecla        ; no key pressed
    LD   PORTLC, #B'11100001
    LD   PORTLD, #B'00011111
    OR   A,PORTLP
    IFEQ A,#B'00011111
    jp   no_tecla
decode:
    IFEQ A,#B'10101111   ; Tecla No 1
    LD   digito,#0x1
    IFEQ A,#B'01101111   ; Tecla No 2
    LD   digito,#0x2
    IFEQ A,#B'11001111   ; Tecla No 3
    LD   digito,#0x3
    IFEQ A,#B'10111101   ; Tecla No 4
    LD   digito,#0x4
    IFEQ A,#B'01111101   ; Tecla No 5
    LD   digito,#0x5
    IFEQ A,#B'11011101   ; Tecla No 6
    LD   digito,#0x6
    IFEQ A,#B'10111011   ; Tecla No 7
    LD   digito,#0x7
    IFEQ A,#B'01111011   ; Tecla No 8
    LD   digito,#0x8
    IFEQ A,#B'11011011   ; Tecla No 9
    LD   digito,#0x9
    IFEQ A,#B'01110111   ; Tecla No 0
    LD   digito,#0x0A
    IFEQ A,#B'10110111   ; Tecla No *
```

```

        LD    digito,#0x0B
        IFEQ A,#B'11010111          ; Tecla No #
        LD    digito,#0x0C
fin_lectura:
        ret
no_tecla:
        ld    digito,#0x00
        ret
.endsect
; ----- Subrutina que despliega numero
.sect show, rom
display:
        LD    A,digito
        ADD   A,#(LOW(bar_graph))   ; Cargar direccion de tabla
        LAID          ; Carga indirecta
        XOR   A,#0xFF               ; Display de Anodo comun
(enciende con 0s)
        X    A,PORTD                ; mandar al Puerto D
        ret
bar_graph:          ; Tabla del display
        .db   b'00000000   ; null
        .db   b'00000110   ; 01
        .db   b'01011011   ; 02
        .db   b'01001111   ; 03
        .db   b'01100110   ; 04
        .db   b'01101101   ; 05
        .db   b'01111101   ; 06
        .db   b'00000111   ; 07
        .db   b'01111111   ; 08
        .db   b'01101111   ; 09
        .db   b'00111111   ; 00
.endsect
.end main

```

## 5.1 USART

Algunos COP8 contienen internamente una unidad serial llamada Universal Synchronous/Asynchronous Receiver/Transmitter (USART), la cual puede ser utilizada para transmitir y recibir datos de forma serial ya sea síncronamente o asíncronamente. Esta unidad es full-duplex, con dos buffers totalmente programable y puede ser configurada en una amplia variedad de modos. Las características principales son las siguientes:

- Operación Full-duplex
- Interfase con opciones programables incluyendo el baud rate, bit de comienzo, longitud de datos (7, 8, o 9 bits), bit de paridad (par/impar, marca, espacio, o ninguno), bits de paro (1 o 2)
- Generación de un baud rate exacto usando el reloj del Microcontrolador.
- Reporte completo de estado
- Vectores de interrupción separados para detectar el buffer de transmisión vacío y el buffer de recepción lleno.
- Entradas independientes de reloj para las unidades de transmisión y Recepción que permiten diferentes velocidades en Tx y Rx
- Operación Síncrona y Asíncrona.
- Modo de operación “Receiver Attention” para una mejor capacidad en circuitos tipo red
- Circuito de detección de Error y capacidad de autodiagnóstico.

### 5.1.1 Operación de la unidad USART

En el diagrama siguiente podemos ver a bloques la circuitería del COP8 USART. Tenemos tres grandes secciones en el circuito: el receptor, el transmisor y la sección de control. La sección de recepción es la encargada de recibir los datos seriales por medio del pin RDX. Estos datos son recorridos en el registro Receive Shift, el bit de más baja prioridad es recibido primero. Cuando se recibe el byte por completo es transferido al buffer (RBUF) en este momento la bandera de buffer lleno es puesta en uno (RBFL). Si la interrupción USART es habilitada se genera una interrupción. El software de esta subrutina leerá el byte en RBUF y hará el proceso correspondiente. La sección de

transmisión manda datos seriales por medio del pin TDX. Cuando el registro (TBUF) se encuentra vacío, la bandera de buffer vacío de transmisión es puesta en uno (TBMT). Si la interrupción de la unidad de transimisión es habilitada, una interrupción es tambien generada. El software entonces escribirá el siguiente byte a ser transmitido y mandado al registro (TBUF). En el tiempo apropiado, el byte de datos es transferido del registro TBUF al registro de corrimiento y de esta foma son mandados bit a bit por medio del pin TDX transmitiendo primero al bit de mas baja prioridad primero.

Los registros de control del USART son: El registro de Control y Status (ENU), el registro de control de Recepción y Status (ENUR) y el registro de Interrupción y Fuente de Reloj (ENUI). Algunos bits en estos registros son bits de control, mientras otros son de estado o monitoreo. Dos registros mas son usados para seleccionar la velocidad en Baud Rate que son los registros de Prescaler (PSR) y Baud (BAUD). La fuente de reloj para generar la transición puede ser seleccionada del reloj interno (CKI) o por una señal de reloj externo recibida en el pin CKX.

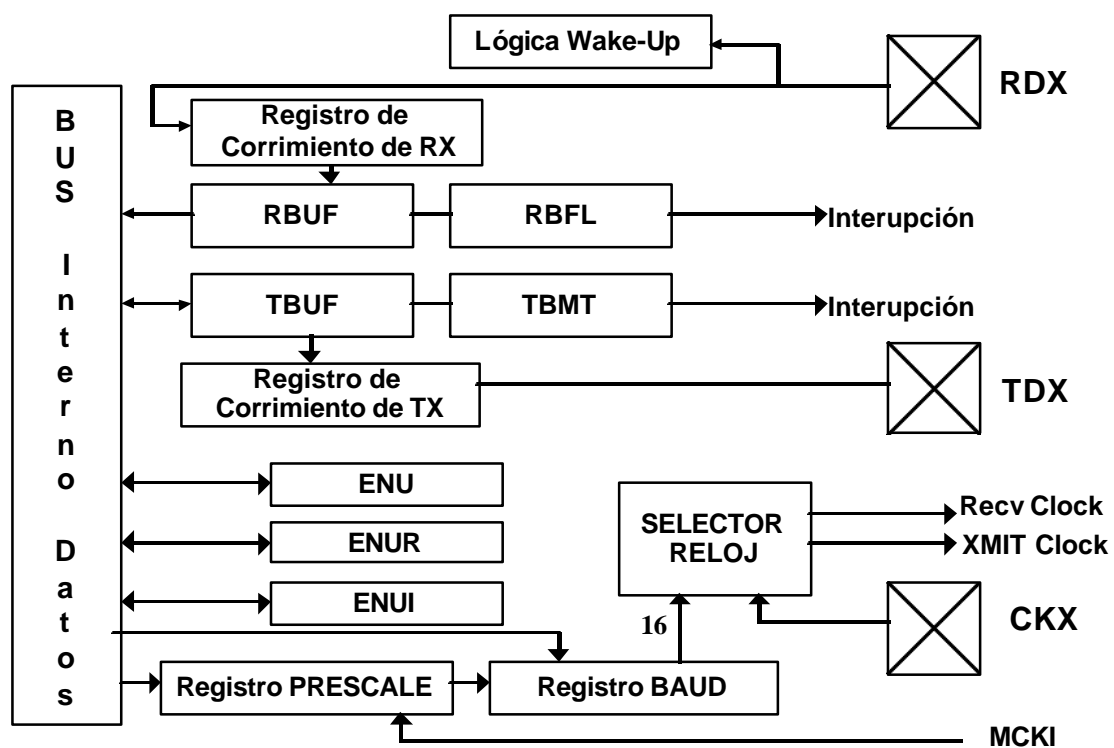


Figura 26 Diagrama a Bloques del USART

### 5.1.2 Interfase USART

Los pines del L1, L2 y L3 del Puerto L son utilizados para la interfase USART como CKX (reloj), TDX (transmisión) y RDX (recepción), respectivamente.

#### Modo asíncrono

El modo asíncrono es seleccionado por el bit SSEL en cero localizado en el registro ENUI. La frecuencia de entrada la USART es 16 veces el baud rate. Cuando el registro de corrimiento se encuentra mandado bit a bit por el pin TDX, el registro TBUF puede ser cargado con el siguiente byte a transmitir. También tenemos el bit de XMTG el cual nos indica cuando la USART está transmitiendo un dato. Por otro lado la USART se encuentra continuamente monitoreando la señal del pin RDX (por algún nivel lógico bajo para detectar el inicio del bit de Start). Los datos seriales recibidos por el pin RDX son mandados al registro de corrimiento Receive Shift. Una vez que el dato fue recibido completamente, el registro RBUF recibe el byte y simultáneamente se levanta la bandera de (RBFL). La fuente de sincronía para la transmisión y recepción puede ser seleccionada por una fuente externa (en el pin CKX) o por el generador interno de baud rate. Si el generador de baud rate interno es utilizado, el reloj interno puede actuar como salida en el pin CKX.

#### Modo síncrono

Al igual que en el modo anterior el modo síncrono es seleccionado por el bit SSEL en uno. En este modo los datos son transmitidos el flanco de subida y recibidos en el flanco de bajada del reloj síncrono en el pin CKX. La frecuencia de entrada al USART es la misma que el baud rate en el pin CKX. El generador de baud rate interno es utilizado para producir la señal de reloj síncrona. La transmisión y recepción son realizadas síncronamente con el reloj. Cuando es seleccionada una sincronía externa en el pin CKX, las tareas de transmisión y recepción son realizadas síncronamente con este reloj por medio de los pines TDX y RDX.

### 5.1.3 Banderas de error del USART

Las condiciones de error pueden ser detectadas por la USART (recepción) haciendo posible que por medio de software se pueda solicitar una retransmisión. Las banderas disponibles son tres y están localizadas en el registro ENUR. La bandera FE (Framing Error) es puesta cuando el receptor falla en la detección de un bit de paro (Stop) al final de la trama. La bandera de DOE (Data Overrun Error) es puesta si un nuevo dato es recibido en RBUF

aún cuando este se encuentra lleno (Antes de que el software haya leído el dato previo del registro RBUF). La paridad seleccionada es también verificada por hardware y la bandera PE (Parity Error) es puesta cuando un error en paridad es detectada. Si es detectada una condición de rompimiento de línea (o desconexión), el bit de BD (Break Detect) es puesto. Una bandera global ERR es puesta si una de las cuatro condiciones de error se presenta (FE, DOE, PE or BD).

## **5.2 WATCHDOG & CLOCK MONITOR**

Algunos COP8 tienen un circuito interno llamado Watchdog y Clock Monitor. Este circuito monitorea la operación del dispositivo y reporta las condiciones anormales con el pin G1 (WDOOUT). El Watchdog es un circuito que monitorea el número de ciclos de instrucción ejecutados y detecta ciertos tipos de errores de ejecución. Es decir, el Watchdog es el encargado de monitorear que la secuencia de ejecución del programa sea la preestablecida por el usuario. El programa debe cargar periódicamente un valor específico al registro de watchdog (Watchdog Service Register) en un intervalo de tiempo específico. En caso de anomalías, el watchdog reporta un estado de alarma por medio de una señal en el pin G1 que normalmente se conecta externamente del chip al pin de RESET. El monitor de reloj es un circuito que detecta la ausencia de la señal de reloj o en su defecto una señal de reloj con una frecuencia demasiado lenta. Un error en la señal de reloj es reportada por el mismo pin del watchdog. El uso del monitor de reloj y watchdog son opcionales. El monitor de reloj puede ser deshabilitado seguido por un Reset, mientras que el watchdog siempre opera y es responsabilidad del usuario si se ignora la señal de salida en el pin WDOOUT.

### **5.2.1 Operación de Watchdog**

El circuito de Watchdog monitorea el contenido del Timer T0 de IDLE para mantener el número de ciclos de instrucción que han sido ejecutados. El software debe escribir un valor específico en intervalos periódicos en el registro WDSVR. Esto deberá suceder dentro del tiempo permitido “Ventana de Tiempo”. Es decir, después de la última recarga no deberá volverse a recargar WDSVR antes de 2,048 ciclos de instrucción, ni después de 65,536. El límite superior puede ser programado ya sea a 8,192, 16,384, 32,768 ó 65,536 ciclos de instrucción.

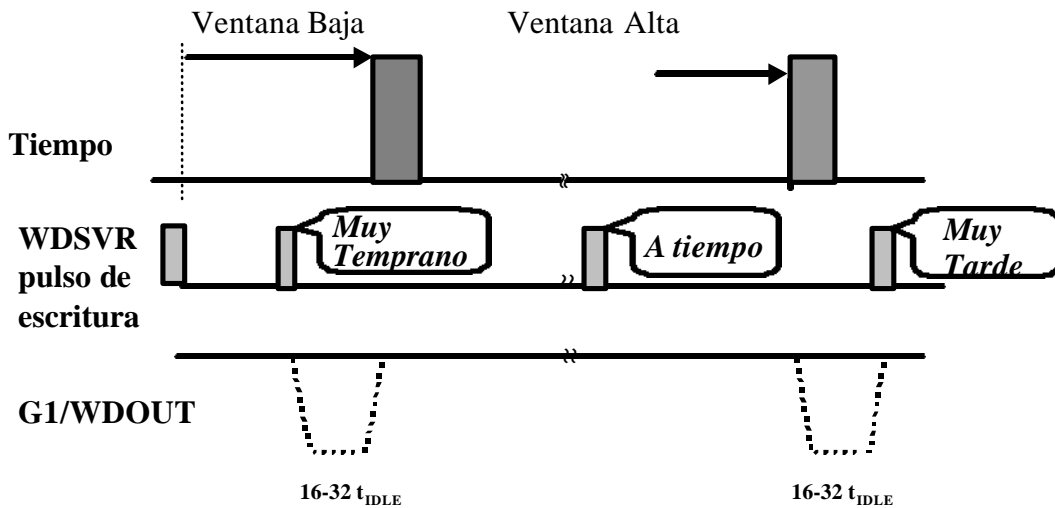
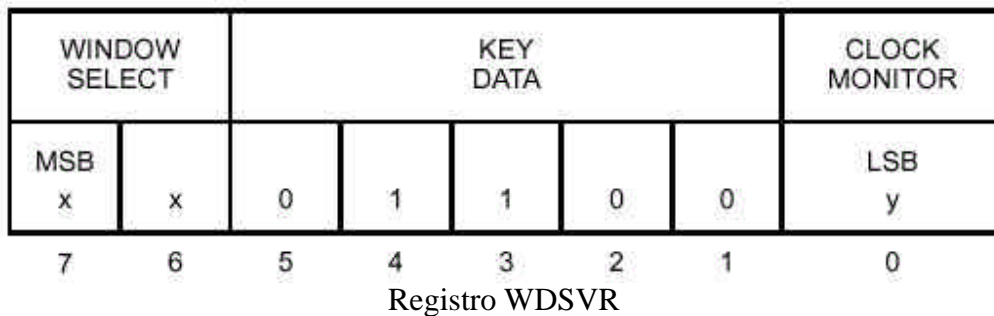


Figura 27. Operación del Timer

El valor específico que debe ser escrito en el registro WDSVR es mostrado en la siguiente figura.



Los dos bits más significativos seleccionan el ancho de la ventana de watchdog en la primera recarga. Los bits 5 al 1 son el patrón que se deberá respetar para identificar la validez de la recarga, el bit menos significativo es el habilitador del monitor de reloj.

### 5.2.2 Operación del monitor de reloj

El Monitor de reloj monitorea constantemente la señal de reloj interno y de existir una anomalía genera una señal de error por medio del pin de WDOOUT. El reloj de instrucción es 1/10 de la frecuencia del reloj del Microcontrolador. Un reloj de instrucción corriendo a 10Khz o a una frecuencia mayor es interpretado como una operación normal. Mientras que un reloj de instrucción a

10Hz o menor es considerado como error. Después del Reset, el monitor de reloj puede ser deshabilitado por software.

### 5.3 Convertidor A/D

La Familia COP8 cuenta en algunas versiones con canales de conversión Analógica-Digital. Por ejemplo el COP8CB cuenta con una unidad de conversión de hasta 16 canales multiplexados con 10 bits de resolución. El convertidor recibe una señal de voltaje analógica (en un pin o par de pines de entrada) y la convierte a un valor digital de 10 bits usando aproximaciones sucesivas. El valor digital es entonces disponible en dos registros mapeados en memoria. El rango del voltaje analógico a convertir es definido por los voltajes AVcc y AGnd. El convertidor A/D es de utilidad cuando se requiere leer un valor de un periférico típico (sensor de temperatura, Convertidor Frec-V, etc.) para procesar la información. Tenemos dos modos de conversión llamados "Single Ended" (con el cual se convierte la señal analógica que se encuentra en un solo pin) y el modo "Differential" (recibe de entrada dos señales en un par de pines). Podemos ver en el diagrama a bloques la estructura de un convertidor A/D por aproximaciones sucesivas.

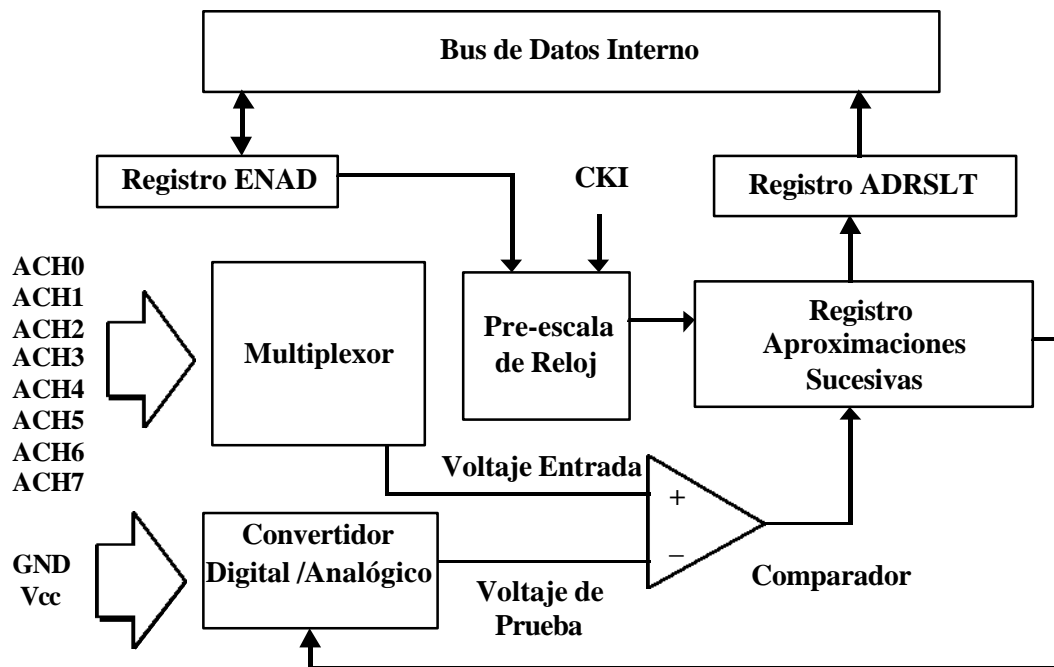


Figura 28. Diagrama a Bloques del Convertidor A/D

Para el COP8CB, la unidad de conversión A/D tiene tres registros en memoria: El registro de control (ENAD), usado para configurar y habilitar el convertidor A/D; Los registros de resultados (ADRSTH y ADRSTL) son un par de registros que almacenan el resultado digital de la conversión analógica ( 000 hex es el valor mínimo y 03FF hex es el resultado máximo). El rango de voltaje de entrada es determinado por AGND (voltaje límite inferior) y AVcc (voltaje límite superior).

Una conversión A/D sencilla toma 15 ciclos de reloj, el tiempo total de conversión depende de la velocidad del reloj del chip y del factor divisor usado para generar sincronización en la unidad A/D. La velocidad de reloj máxima permitida es de 1.67 MHz (Periodo de reloj de 600 ns). El tiempo requerido mínimo es entonces de 15 ciclos de reloj ó 9.0 microsegundos.

Si dos canales van a ser monitoreados simultáneamente, deberán estar multiplexados en el tiempo. Es decir, dedicar una fracción de tiempo para un canal y después otra fracción de tiempo para el otro canal. El convertidor A/D es controlado por el registro ENAD.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
ADCH3	ADCH2	ADCH1	ADCH0	ADMOD	MUX	PSC	ADBSY
ADCH3–ADCH0: Channel Selection bits							
ADMOD:		Mode Selection (0 = single-ended, 1 = differential)					
MUXOUT:		Analog Mux Output and Sample and Hold Input Enable					
PSC:		A/D clock prescaler selection bit					
ADBSY:		A/D enable and Busy bit					

Figura 29. Registro ENAD

En este registro se realiza la selección de canal (Bit 7-5), el modo de conversión (ADMOD), la función de mandar una señal de entrada al exterior del Microcontrolador nuevamente (con el fin de amplificar, filtrar, etc), la selección de la pre-escala del reloj PSC y el habilitador del Convertidor y bandera (ADBSY).

## 5.4 BROWN OUT

La circuitería de protección de “Brown Out” reinicializa el dispositivo después de que los voltajes de operación ( $V_{cc}$ ) se encuentran en niveles por debajo del voltaje de Brown Out ( $V_{bor}$  es predeterminado de Fabrica). El dispositivo es detenido en reset cuando  $V_{cc}$  permanece por debajo del voltaje de Brown Out y el timer IDLE T0 carga un valor 0Fx (240-256 Tc).

Cuando el voltaje  $V_{cc}$  alcanza un valor mayor que  $V_{bor}$ , el timer de Idle comienza una cuenta regresiva. Una vez que se presenta un Underflow del Timer IDLE, el reset interno es liberado y el dispositivo comenzará a ejecutar incrucciones.

Este reset interno realizará la misma función que el reset externo. Una vez que  $V_{cc}$  es mayor que  $V_{bor}$  y que el Timer T0 de IDLE termina la cuenta precargada permitiendo la operación normal.

Una excepción al caso descrito anteriormente se presenta cuando el voltaje  $V_{cc}$  cae por debajo de los 1.8V. De esta forma, el circuito de Brownout insertará un retardo de 3ms aproximadamente cuando se restaura el voltaje  $V_{cc}$ . El Microcontrolador permanecerá en reset interno durante estos 3ms antes de que el Timer de IDLE incerte el retardo de 240 a 256 Tc.

Podemos entonces tener tres casos posibles .

**Caso 1.**  $V_{cc}$  sube de 0V a un valor estable. En el transcurso de este incremento de voltaje, la señal de reset interno del Microcontrolador es indefinida hasta que el voltaje es mayor a 1.0 V aproximadamente. En este momento, el circuito de Browout se hace activo y sostiene al dispositivo en RESET.

Conforme el voltaje de  $V_{cc}$  incrementa y sobrepasa el nivel del 1.8V, un retardo de 3ms es incertado y el timer de IDLE carga un valor entre 00F0 y 00FF hex. Una vez que  $V_{cc}$  es mayor al volaje  $V_{bor}$  y el retardo (3ms) ha expirado, el timer de IDLE empieza a contar regresivamente. Terminada esta cuenta del timer, el dispositivo es liberado del reset interno y comienza la operación normal solo si el voltaje de  $V_{cc}$  es mayor a  $V_{bor}$ .

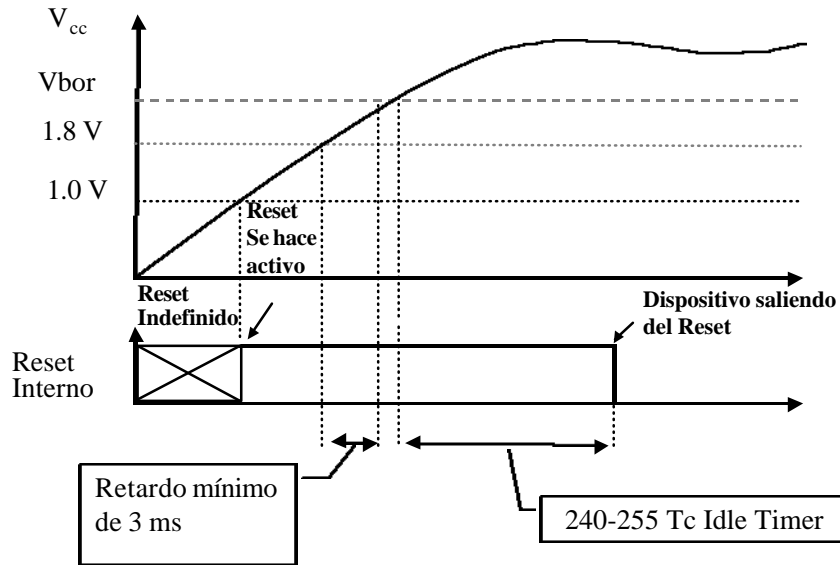


Figura 30. Gráfica Brownout Caso 1.

**Caso 2.** Muestra una caída de  $V_{cc}$  por debajo de los 1.8V. Como  $V_{cc}$  es menor que  $V_{bor}$ , la señal de RESET interna se activa. Cuando  $V_{cc}$  incrementa su valor por arriba de 1.8V, El primer retardo de 3ms es insertado. Dado que  $V_{cc}$  incrementa rápidamente, al término del retardo de 3ms,  $V_{cc}$  ya es mayor a  $V_{bor}$ . Por lo tanto el segundo retardo es insertado inmediatamente (decrementos del timer IDLE). Una vez que los decrementos ocasionan un Underflow el dispositivo comienza a ejecutar código ya que  $V_{cc}$  continúa siendo mayor a  $V_{bor}$ .

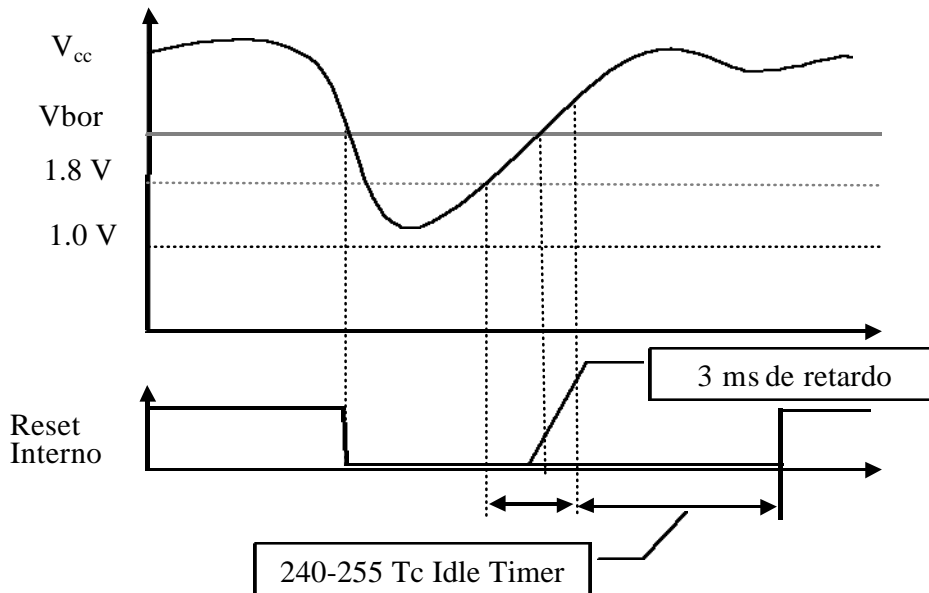


Figura 32. Gráfica Brownout Caso 2

**Caso 3.** Muestra un pequeña caída de voltaje donde  $V_{cc}$  en determinado momento es menor a  $V_{bor}$ , pero siempre se encuentra por arriba de 1.8V. El RESET interno estará activo cuando  $V_{cc}$  es menor a  $V_{bor}$ . El Timer IDLE se encuentra preparado para realizar la cuenta una vez que  $V_{cc}$  sea mayor a  $V_{bor}$ . Cuando esto sucede inmediatamente el timer inserta el retardo (240-256  $T_c$ ) y una vez que es concluido el retardo y  $V_{cc}$  es mayor  $V_{bor}$  la operación es reinicializada.

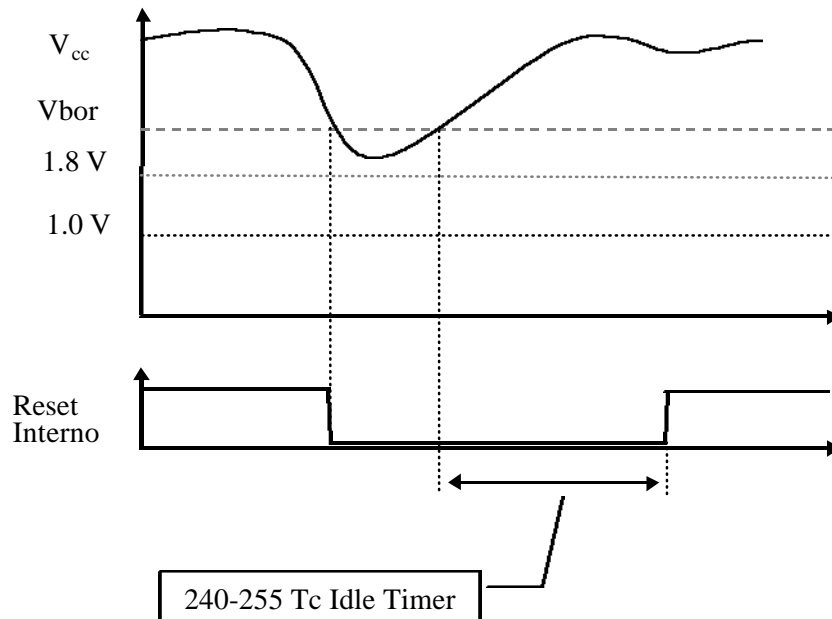


Figura 33. Gráfica Brownout Caso 3

Los voltajes típicos de  $V_{bor}$  son al rededor de 4.5V y 2.7V. Es decir si tenemos una aplicación alimentada con un voltaje de 3.3V, optaríamos por una versión de Microcontrolador con un voltaje de brownout de 2.7. Bajo ninguna circunstancia el pin de RESET deberá estar flotando si el brownout es habilitado (típicamente es conectado a  $V_{cc}$  o a una resistencia de pull-up). Si el Brownout es deshabilitado, no se genera ningún reset interno y el timer de IDLE cargará un valor indeterminado. Es en este caso cuando el RESET externo deberá ser utilizado.

## 5.5 MEMORIA FLASH

Hoy en día existen Microcontroladores con una memoria denominada FLASH. El COP8 tiene dos versiones de Microcontroladores con memoria de FLASH que son el COP8SB y el COP8CB.

La memoria Flash tiene muchas ventajas sobre la memoria EEPROM. La Flash ofrece la característica de poder ser borrada eléctricamente (como la memoria RAM) y simultáneamente la no volatilidad de la memoria ROM para retener datos después de que  $V_{cc}$  es removido. El tiempo de escritura promedio típico es de 17us por byte. La unidad básica de la memoria Flash es un transistor de  $2.0 \mu\text{m}^2$  de tamaño.

Por otro lado la memoria EEPROM ofrece la posibilidad de escribir a memoria (afectando a toda una página). El sistema deberá esperar al menos un tiempo promedio típico de 10ms para una escritura e EEPROM. No requiere que la celda a ser escrita sea previamente borrada (como en la memoria Flash) y por último la unidad básica de la memoria EEPROM se compone de dos transistores de aprox  $4.0 \mu\text{m}^2$

La Memoria Flash es afectada en gran medida por las condiciones externas en temperatura. En el peor de los casos (temperaturas menores a los  $0^\circ\text{C}$ ) conveniente hacer borrados masivos para asegurar su funcionamiento máximo en ciclos de escritura/lectura. Observar gráfica a la Izq.

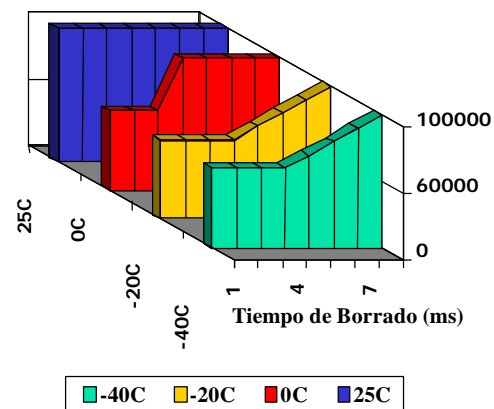


Figura 34. Gráfica Temp vs Núm. de Borrados

En particular los COP8SB y COP8CB tienen la capacidad de ser borrados hasta 10,000 veces con una retención de datos asegurada de hasta 100 años. La memoria Flash se encuentra organizada en 8 bloques de 128 bytes. Para ser escrita requiere de un alto voltaje que es generado internamente en el Microcontrolador, por lo tanto no requiere de hardware externo.

Otra característica muy importante de la memoria Flash del COP8 es que tiene la capacidad de actuar como memoria EEPROM. Es decir, el programa almacenado en memoria Flash puede ejecutar una escritura de datos a la misma memoria. La cantidad de memoria de programa y de datos la determina el usuario, por lo que podríamos tener en un espacio de 32K de Flash, 1k de memoria de programa y 31k de memoria EEPROM y viceversa.

Las escrituras a memoria Flash pueden ser realizadas byte a byte o por bloques de bytes. El procedimiento para salvar datos a memoria Flash es muy sencillo.

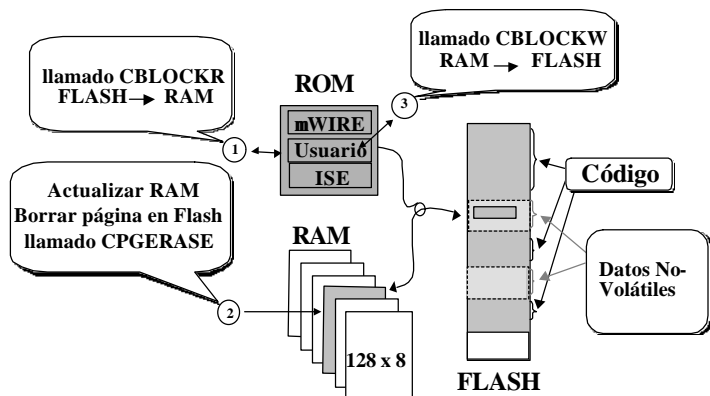


Figura 35. Actualización de una Página en Flash.

El primer paso consiste en hacer una lectura de una página localizada en Flash. Esto es logrado mandando a llamar una subrutina grabada en ROM. Una vez que tenemos los datos en memoria RAM, es posible modificarlos y/o actualizarlos y simultáneamente borramos la página donde se almacenarán los datos.

Por último llamamos a otra subrutina de ROM para que escriba los datos localizados en RAM a la memoria FLASH. De esta forma estamos almacenando datos en memoria no volátil desde la misma memoria FLASH.

Como ya hemos visto, es necesario contar con subrutinas que sirvan como herramienta para el acceso a memoria FLASH. Estas subrutinas están localizadas en una sección de memoria ROM reservada y que todos los COP8CB y SB ya cuentan de fábrica. De hecho, un Microcontrolador COP8 con memoria Flash puede ejecutar código en memoria ROM y en memoria FLASH. El bit FLEX es el responsable de determinar dónde se ejecutará código. Este bit se encuentra localizado en el registro de OPTION y puede ser intervenido por el usuario. Es decir, el usuario puede determinar cuándo quiere que su programa (Residente en Flash) sea ejecutado o cuando quiere ejecutar código en ROM (subrutinas de fábrica).

Cuando el bit de FLEX se encuentra en cero, el Microcontrolador ejecutará código en ROM. Cuando FLEX se encuentre en uno, se ejecutará código en FLASH. Suponiendo que tenemos un COP8 con Flash nuevo, el bit de FLEX se encontrará en cero, en otras palabras si alimentamos este controlador se encontrará preparado para ejecutar código en ROM.

La memoria ROM está dividida en tres secciones las cuales son: Rutinas microwire ISP (In System Programming), Rutinas de soporte para ISP y Rutinas de soporte para la Emulación.

Rutinas Microwire ISP: Estas subrutinas son las encargadas de establecer comunicación vía microwire para poder realizar la programación de la memoria

FLASH. Es la sección que se ejecuta cuando el Microcontrolador no está programado y pone al dispositivo en espera para recibir comandos vía microwire.

Rutinas de soporte ISP: Cuando el usuario decide programar la memoria FLASH, es posible que requiera de tareas especiales como escritura/lectura de memoria FLASH. Las subrutinas localizadas en esta sección son las encargadas de realizar estas tareas. Consecuentemente, el usuario tendrá que hacer su llamado desde el código de programa.

Rutinas de soporte para la Emulación: Esta sección de ROM es la encargada de realizar el soporte para poder visualizar el estado del Microcontrolador con un emulador. Gracias a esta sección ya no es necesario trasladar los estados analógicos o digitales de todos los pines a los cuales se encuentra conectado el Microcontrolador.

### **Programación de la memoria FLASH.**

La memoria FLASH del COP8 puede ser programada por cuatro vías :

1. Vía Microwire, accesible de fábrica utilizando la memoria ROM la cual prepara al Microcontrolador a recibir comandos para programar/leer la memoria FLASH.
2. Utilizando las rutinas ISP (In System Programming) desde el código de programa.
3. Vía externa, es decir por medio de un programador de Memorias
4. Por medio del emulador en modo de emulación.

Las rutinas ISP o funciones disponibles en memoria ROM son : Escritura de Byte, Borrado de Página, Borrado Masivo, Escritura por Bloque, Lectura de Byte, Lectura de Bloque, Salida del ISP, Exec. FLASH.

Por ejemplo, para escribir un byte tendríamos una secuencia en el programa como la siguiente :

```
LD PGMTIM, #Value_From_Table ; Tiempo de programación
LD ISPADHI, #HI_ADDR ; Determinar la dirección parte alta
LD ISPADLO, #LO_ADDR ; Determinar la dirección parte baja
LD ISPWR, Date_To_Write ; Guardar el dato a ser escrito en FLASH
LD ISPKEY, #0x98 ; poner el bit KEY para validar el salto a ROM
JSRB CWRITEBF ; Rutina ISP de escritura
```

La programación de la memoria FLASH por la vía de Microwire ofrece muchas ventajas y es muy sencilla de realizar. Si el Microcontrolador tiene el bit de FLEX en cero, quiere decir que el dispositivo se encuentra esperando comandos vía Microwire. Por lo tanto, desarrollaremos el siguiente circuito para poder hacer la programación ISP.

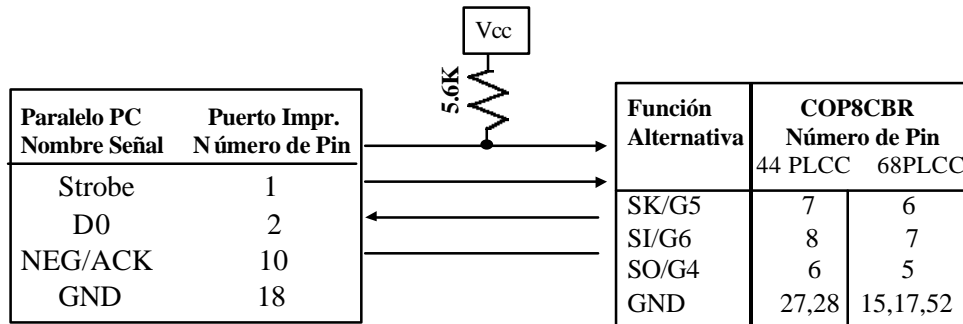


Figura 36. Configuración de Cable ISP

En conclusión, con un Puerto Paralelo de una Computadora, con un software llamado FLASHWIN y un cable sencillo podremos programar un Microcontrolador COP8 con memoria Flash sin importar que éste ya se encuentre soldado en una tarjeta (por consecuencia se le llama programación en circuito).

El software Flashwin (Figura 37) muestra una ventana de diálogo muy sencilla con la cual podemos verificar la programación realizada anteriormente, poner el bit de seguridad, recuperar el código contenido en la memoria del Microcontrolador o incluso grabar o leer bloques de memoria con un solo comando.

Típicamente, el diseñador contempla esta característica de ISP y deja disponibles los pines G4, G5 y G6 así como Tierra en un conector para facilitar posibles reprogrmaciones o actualizaciones del código de programa.



Figura 37. Ventana Flashwin

## Autoevaluación del CAPITULO 5

1. ¿ Qué significa USART y cuales son sus características principales ?
2. ¿ Qué diferencia existe entre el modo asíncrono y síncrono ?
3. ¿ Describe qué es el Watchdog y Clock Monitor ?
4. ¿ Describe el funcionamiento del convertidor Analógico – Digital ?
5. ¿Cuál es la funcionalidad del Brown out y explica un ejemplo ?
6. ¿Qué ventaja tiene el hacer un borrado masivo ?
7. ¿ La Memoria FLASH del COP8 puede funcionar como EEPROM ?
8. ¿ Describe el procedimiento para actualizar la memoria FLASH ?
9. ¿ Para qué se utiliza el bit de FLEX ?
10. ¿ Por cuántas vías puede ser programado un COP8 Flash, describir brevemente cada una ?

### Prácticas \*

1. Programar un COP8CBR que contenga una tabla en Memoria de Programa. Después de que se presente un evento ( con un push button, interrupción externa, una captura en un timer, etc. ) la tabla deberá ser cambiada incrementando el valor ( en uno ) de todas las localidades es decir :

Contenido Original ( ROM )	Contenido después del evento
<i>[ Localidad ] : Valor</i>	<i>[ Localidad ] : Valor</i>
[ 0100 ] : 0x03	[ 0100 ] : 0x04
[ 0101 ] : 0x0F	[ 0101 ] : 0x00
[ 0102 ] : 0x0D	[ 0102 ] : 0x0E
[ 0103 ] : 0x08	[ 0103 ] : 0x09
[ 0104 ] : 0x02	[ 0104 ] : 0x03
[ 0105 ] : 0x00	[ 0105 ] : 0x01
[ 0106 ] : 0x0C	[ 0106 ] : 0x0D
[ 0107 ] : 0x0C	[ 0107 ] : 0x0D
[ 0108 ] : 0x09	[ 0108 ] : 0x0A
[ 0109 ] : 0x01	[ 0109 ] : 0x02

2. Hacer un programa que establezca la comunicación serial de un Microcontrolador a otro Microcontrolador.

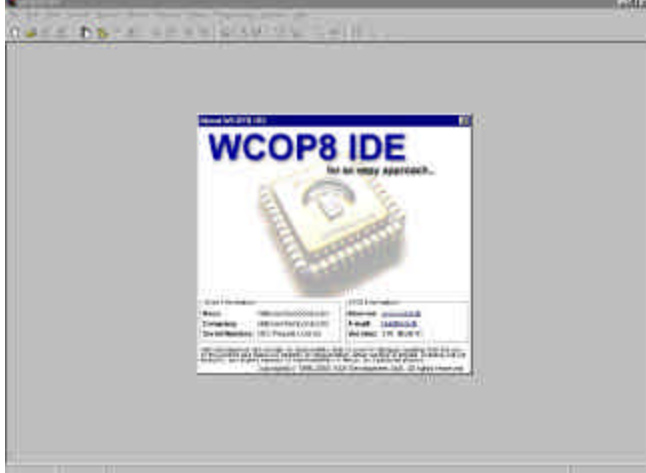
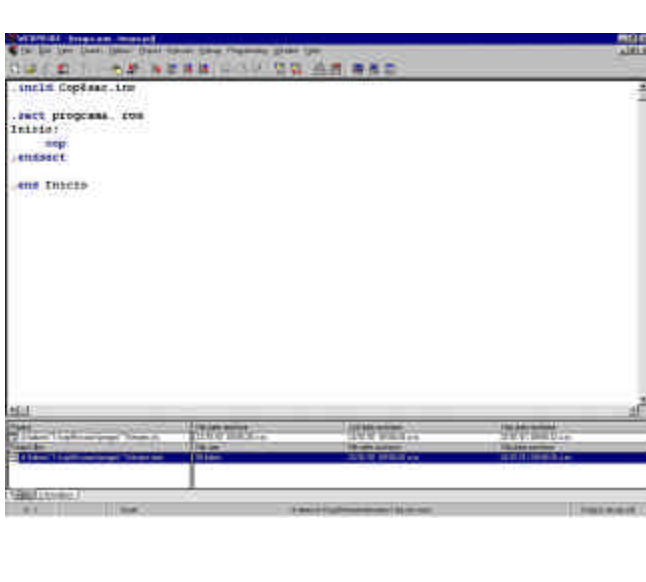
3. Hacer un programa que establezca la comunicación serial con el puerto serial de una PC a 9600 bps.
4. Hacer un programa con el COP8CBR que utilice la unidad de conversión A/D. Es necesario que el valor digital obtenido sea desplegado por algún puerto (Leds, Displays, LCDs, etc. ) de tal forma que se haga visible la conversión A/D.

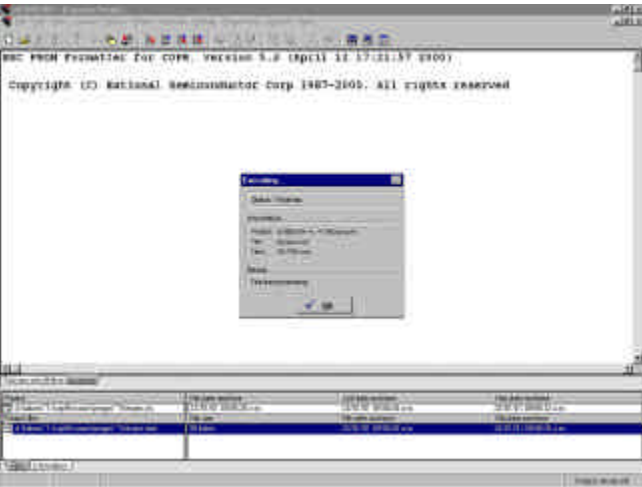
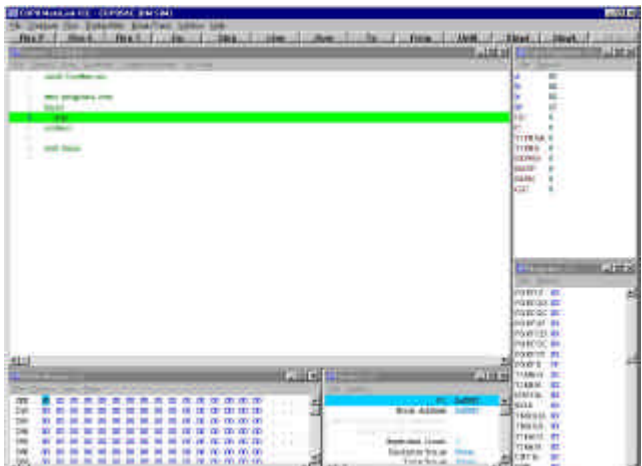
\* Para realizar las prácticas es necesario cubrir el capítulo 6

**SIMULADOR Y EMULADOR**

**Simulador:** Un simulador es una herramienta en software la cual tiene implementadas las funciones que realizaría, ante ciertas situaciones, una entidad. La entidad simulada puede ser cualquier sistema que tenga la capacidad de recibir datos, procesarlos y otorgar un resultado. El COP8 cuenta con un simulador con el cual podemos realizar la ejecución de nuestros programas SIN necesidad de contar con alguna herramienta o incluso sin contar con el Microcontrolador.

Los pasos para editar y simular un programa son los siguientes.

<p>1. Ejecutar el integrador WCOP8 IDE</p>	
<p>2. Crear / Llamar al Proyecto</p>	

<p>3. Ensamblar/Ligar</p>	
<p>4. Llamar al Simulador</p>	

En la secuencia anterior se describen, de forma muy general, dos programas como son el integrador (WCOP8 IDE) y el Simulador de Metalink. El Integrador WCOP8 es una herramienta que nos ayuda a desarrollar código, ensamblar y ligar bajo una sola ventana y sin necesidad de abrir y cerrar programas. En el paso número 2 lo que tenemos es el llamado o la creación de un proyecto. Todos los programas tienen que estar bajo este esquema de proyectos (Observar dibujo del apartado número 2 donde se tienen una ventana dividida en dos. La parte inferior muestra los archivos .ASM asociados al proyecto y en la parte superior se muestra la ventana de edición del archivo .ASM). En el paso número 3 se muestra el resultado del ensamble y ligado del archivo .ASM (en caso de detectar un error, se muestra en ésta ventana la

localización y descripción del mismo). Por último tenemos el llamado al simulador que muestra las ventanas de visualización de Código, Memoria RAM, Estado del Microcontrolador, Registros del Core y Registros de configuración.

Los archivos creados por la realización de este procedimiento son :

**Archivo.asm:** Este archivo contiene el programa fuente que ha sido creado por el usuario.

**Archivo.lis:** Este es uno de los dos archivos creados después de ensamblar el archivo.asm. Contiene información detallada del programa y su información algunas veces es de suma importancia para la depuración.

**Archivo.obj:** El otro archivo creado es el llamado archivo objeto que será el utilizado por el ligador para realizar la corrección lógica del programa.

**Archivo.cof:** Después de realizar el ligado del programa se generan otros dos archivos que son el .COF y el .MAP. Este archivo es el que será programado en la memoria de programa del Microcontrolador COP8

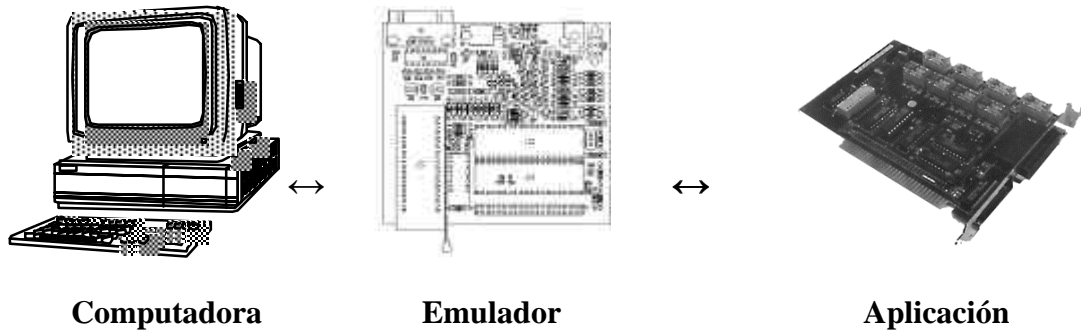
**Archivo.map:** En este archivo se despliega toda la información del mapa de memoria, así como el tamaño total del programa, Checksum, etc.

**Archivo.hex:** De la misma forma que el archivo .COF, este archivo es resultado del ligado del programa fuente pero en formato hexadecimal.

**Archivo.prj:** Normalmente se asocia el mismo nombre del archivo.asm al proyecto al que este asociado. Este archivo lo único que tiene son los parámetros bajo los cuales se encuentra el archivo ensamblador (herramientas a utilizar, trayectoria de localización, ligas a otros archivos, etc. )

**Emulador :** A diferencia del Simulador, el Emulador es una herramienta basada en hardware que permite la prueba de nuestro circuito con la ventaja de poder visualizar la ejecución de nuestro programa y detenerla en el momento deseado así como la modificación de información en ciertas localidades en memoria y muchas otras cosas más. Todo esto puede ser realizado sin contar con el Microcontrolador. Las funciones del Microcontrolador las realizará una tarjeta conectada simultáneamente a una computadora y a un protoboard o tarjeta de aplicación. La secuencia para emular es igual que en la simulación con la diferencia de que el llamado se hace en este caso a la herramienta de

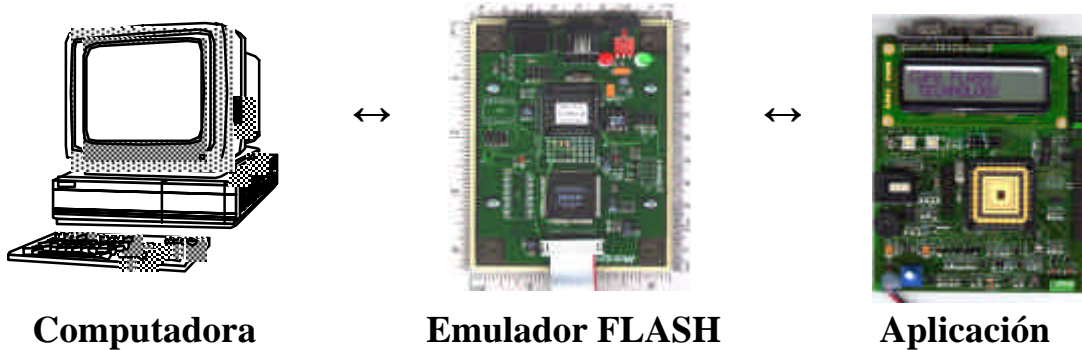
emulación correspondiente. A continuación se muestran la forma de conexión así como las variantes en emuladores con las que cuenta el COP8.



Los emuladores existentes para el COP8 SA, SG y ACC son

<p>EPU: El Emulador and Programming Unit es la herramineta más sencilla que programa todos los encapsulados de cada familia (SA,SG). La emulación no es realizada en tiempo real. Esto significa que la respuesta en tiempo de la emulación no será fiel a la respuesta en tiempo de un Microcontrolador.</p>	
<p>DM: El Debug Module es la herramienta más recomendable para la depuración de programas. Al igual que el EPU, ésta herramienta programa y emula pero en tiempo real. Es decir la respuesta en tiempo es exactamente igual a la que tendría un Microcontrolador. Contiene bases para programar todos los encapsulados en PLCC, SO y DIP</p>	
<p>El ICE Master: Esta herramienta es utilizada únicamente para depurar con una gran resolución (6us). No programa ningún Microcontrolador y por ésta causa no es tan recomendable a nivel diseño.</p>	

Para el Microcontrolador COP8 con Flash (Familias SB y CB) tenemos otras herramientas para la depuración.



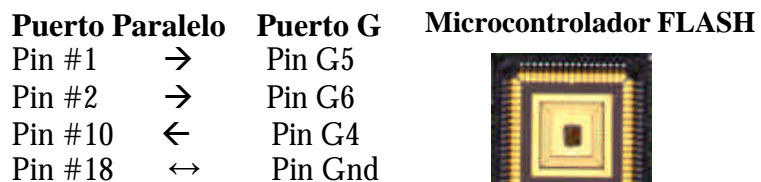
La siguiente tabla ilustra las diferencias entre las versiones de los emuladores Flash (EM, DM e IM).

NSC Product Names	COP8-EMflash	COP8-DMflash	COP8-IMflash
<b>Breakpoints</b>			
Number of SW Breakpoints	16	32K	32K
Number of HW Breakpoints	None	None	32K
Pass Counter	No	No	Yes
<b>Trace</b>			
Real-time Trace	No	Yes	Yes
Trace Memory Size (Frames)	N/A	32K	32K
Adjustable Trace Trigger Points	N/A	No	Yes
Transparent Trace	N/A	No	Yes
Read Data Trace	N/A	No	Yes
Trace Filtering (On/Off) in Real Time	N/A	No	Yes
Trace Search	N/A	Yes	Yes
<b>Accessories</b>			
Enclosed in Case	No	Yes	Yes
Power Supply Included	Yes, 110 or 220	Yes	Yes
Serial Cable Included	Yes	Yes	Yes

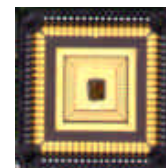
Una de las ventajas de usar un Microcontrolador Flash es que no requiere de una herramienta de programación. Es decir, se puede programar de forma muy sencilla en cualquier tarjeta a través del puerto paralelo. El siguiente diagrama muestra el circuito necesario para programar el COP8SB o CB.



**Computadora**



**Cable 4 vías**



**COP8CB**

## **Autoevaluación del CAPITULO 6**

- ¿ Qué diferencia existe entre un emulador y un simulador ?
- ¿ Enumera los pasos para poder emular un Programa ?
- ¿ Cuántos archivos debo esperar que se generen después de ensamblar y ligar un archivo .asm ?
- ¿ Cuántos emuladores tiene el COP8 y que ventajas ofrece cada uno ?
- ¿ Por qué se dice que para el COP8 Flash se no se requiere programador ?

### **Para investigar**

- ¿ Explica detalladamente qué diferencia existe entre el COP8SA-EPU y el COP8SA-DM ?

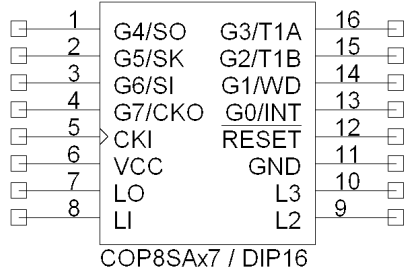
### **Prácticas**

1. Simular el ejemplo 1 del capítulo 3 (Sección Programación Básica 3.4) y obtener los resultados de la tabla.
2. Simular el ejercicio 1 del capítulo 3 (Sección Programación Básica 3.4) y llenar la tabla con los valores correspondientes.
3. Simular el ejercicio 2 del capítulo 3 (Sección Programación Básica 3.4) y llenar la tabla con los valores correspondientes.

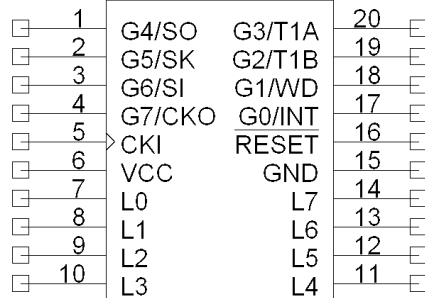
Set de Instrucciones

ADD	A, MemI	Suma	A <- A + MemI
ADC	A, MemI	Suma con Acarreo	A <- A + MemI + C, C <- Carry
SUBC	A, MemI	Resta con Acarreo	A <- A - MemI + C, C <- Carry
AND	A, MemI	Operación Lógica AND	A <- A and MemI
ANDS	A, Imm	Operación AND Imm, Saltar si Cero	Saltar la siguiente si (A and Imm) = 0
Z	A, MemI	Operación Lógica OR	A <- A OR MemI
OR	A, MemI	Operación Lógica OR Exclusiva	A <- A XOR MemI
XOR	A, MemI	SI es Igual	Compara A y MemI, ejecutar siguiente si A=MemI
IFEQ	MD, Imm	SI es Igual	Compara MD y Imm, ejecutar siguiente si MD=Imm
IFEQ	A, MemI	SI NO es Igual	Compara A y MemI, ejecutar siguiente si A≠MemI
IFNE	A, MemI	Si es mayor que	Compara A y MemI, ejecutar siguiente si A>MemI
IFGT	#	Si B NO es igual	Ejecutar siguiente si 4 bits bajos de B No = Imm
IFBNE	Reg	Decrementa Reg, Satar si Cero	Reg <- Reg - 1, saltar siguiente si Reg se hace cero
DRSZ	#, Mem	Poner Bit	1 a bit de Mem (bit = 0 a 7 inmediato)
SBIT	#, Mem	Limpiar Bit	0 a bit de Mem (bit = 0 a 7 inmediato)
RBIT	#, Mem	Si esta puesto un Bit	Si bit Mem es verdadero, ejecutar sig. Instrucción
IFBIT		Limpiar Bandera Pendiente	Limpiar Bandera de Software Interrupt Pending
RPND			
X	A, Mem	Intercambio de A con memoria	A <-> Mem
LD	B, Imm	Carga B con dato Inmediato	B <- Imm
LD	A, MemI	Carga A con memoria	A <- MemI
LD	Mem, Imm	Carga Memoria con dato inmediato	Mem <- Imm
LD	Reg, Imm	Carga Registro con dato inmediato	Reg <- Imm
X	A, [B+]	Intercambio de A con memoria [B]	A <-> [B] (B <- B + 1)
X	A, [X+]	Intercambio de A con memoria [X]	A <-> [X] (X <- X + 1)
LD	A, [B+]	Carga A con memoria [B]	A <- [B] (B <- B + 1)
LD	A, [X+]	Carga A con memoria [X]	A <- [X] (X <- X + 1)
LD	[B+], Imm	Carga memoria [B] con dato inmediato	[B] <- Imm (B <- B + 1)
CLRA		Lipiar A	A <- 0
INC	A	Incrementar A	A <- A + 1
DEC	A	Decrementar A	A <- A - 1
LAID		Carga indirecta en A de ROM	A <- ROM(PU, A)
DCOR	A	Corrección decimal en A	A <- BCD corrección (sigue de ADC, SUBC)
RRC	A	Rotar a la derecha con acarreo	C > A7 -> ... -> A0 > C, HC <- A0
RCL	A	Rotar a la izquierda con acarreo	C <- A7 <- ... <- A0 <- C, HC <- A3
SWAP	A	Intercambio de nibbles de A	A7...A4 <-> A3...A0
SC		Poner en Alto C	C <- 1
RC		Poner en Bajo C	C <- 0
IFC		Si C	Si C es uno, ejecutar siguiente instrucción
IFNC		Si NO C	Si C no es uno, ejecutar siguiente instrucción
POP	A	Salvar A en la Pila	SP <- SP + 1, A <- [SP]
PUSH	A	Restaurar A de la Pila	[SP] <- A, SP <- SP - 1
VIS		Salto a la Rutina de Servicio Corresp.	PCL <- [VL], PCU <- [VU]
JMPL	Addr.	Salto Absoluto Largo	PC <- ii (ii = 15 bits, 0 a 32K)
JMP	Addr.	Salto Absoluto	PC11...PC0 <- i (I = 12 bits)
			PC15...PC12 permanecen sin cambio
JP	Disp.	Salto Relativo Corto	PC <- PC + r (r es -31 a +32, no 1)
JSRL	Addr.	Salto Subrutina largo	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- ii
JSR	Addr.	Salto Subrutina	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC11..PC0 <- ii
JID		Salto Indirecto	PL <- ROM(PU, A)
RET		Regreso de Subrutina	SP + 2, PL <- [SP], PU <- [SP - 1]
RETS		Regreso de Subrutina y Salto	SP + 2, PL <- [SP], PU <- [SP - 1], Saltar siguiente instrucción
K			SP + 2, PL <- [SP], PU <- [SP - 1], GIE <- 1
RETI		Regreso de Interrupción	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- 0FF
INTR		Generar una interrupción	PC <- PC + 1
NOP		Sin Operación	

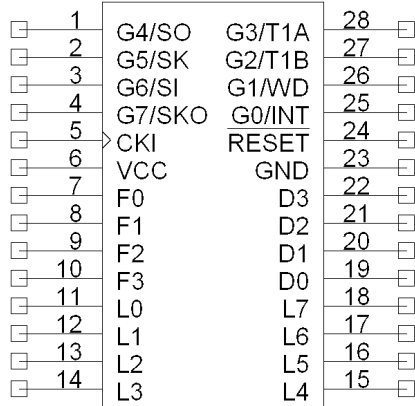
Encapsulados del COP8SA \*



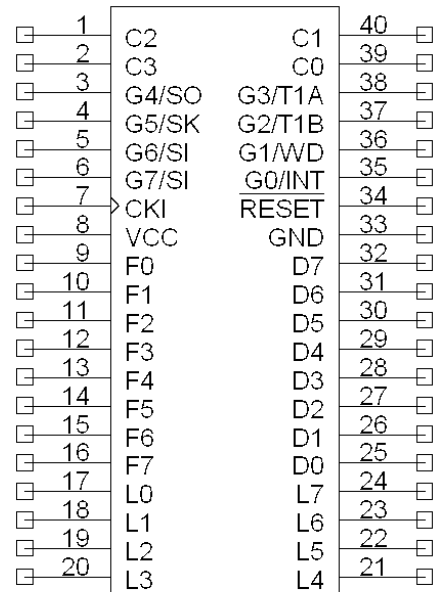
COP8SAx7 / DIP16



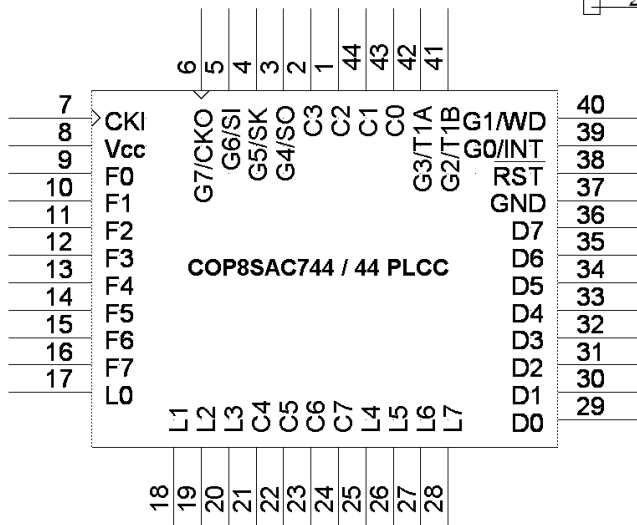
COP8SAx7 / DIP20



COP8SAx7 / DIP28

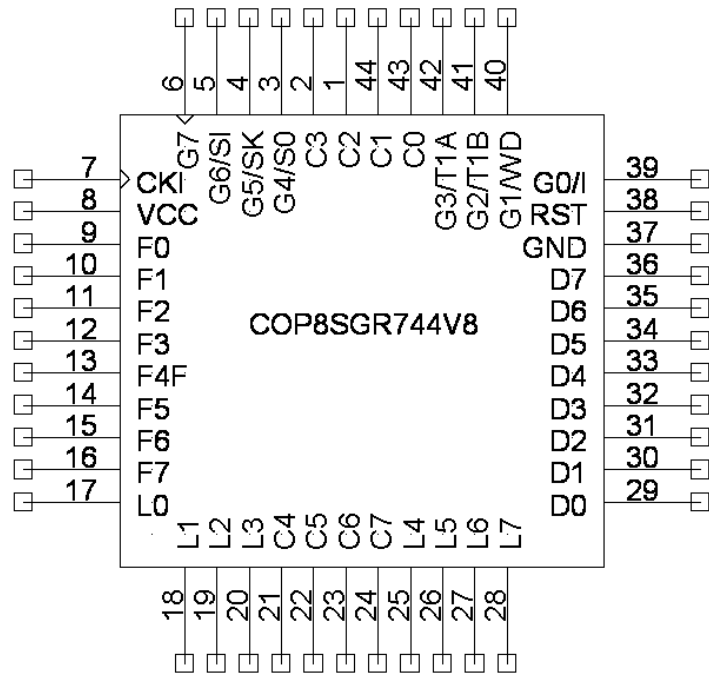
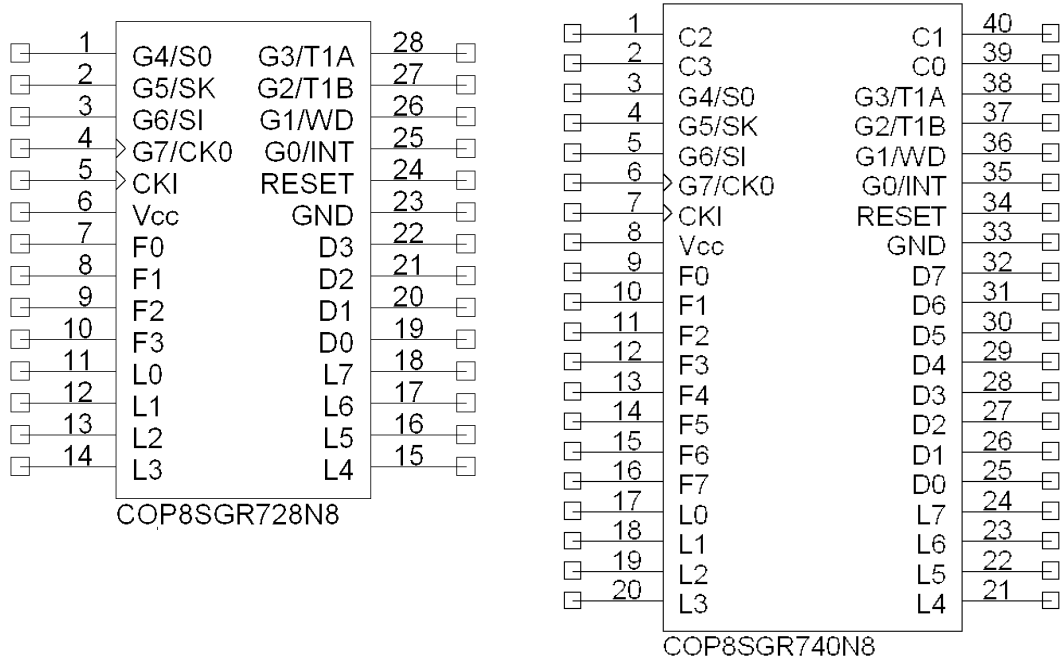


COP8SAC7 / DIP40

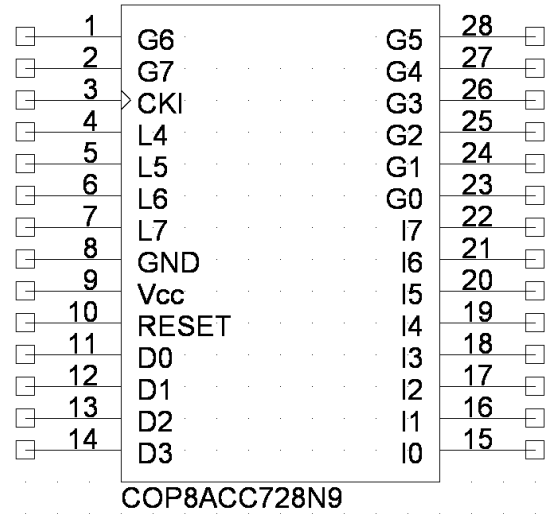
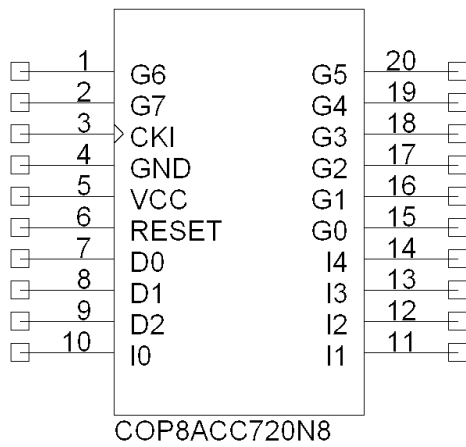


COP8SAC744 / 44 PLCC

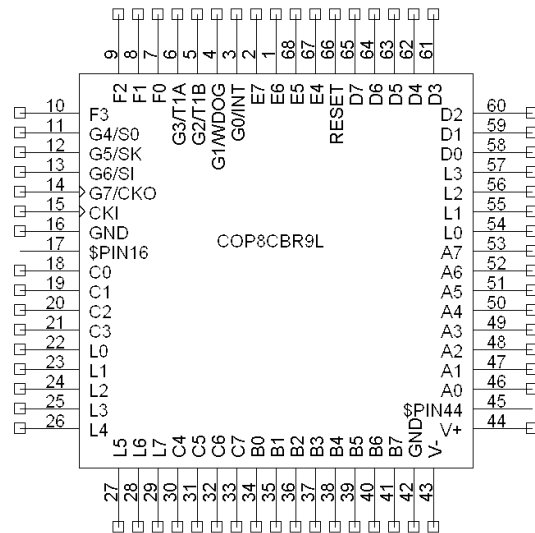
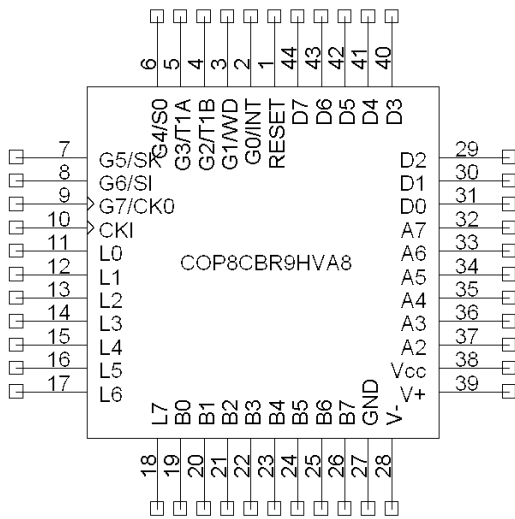
## Encapsulados del COP8SG \*



## Encapsulados del COP8ACC \*



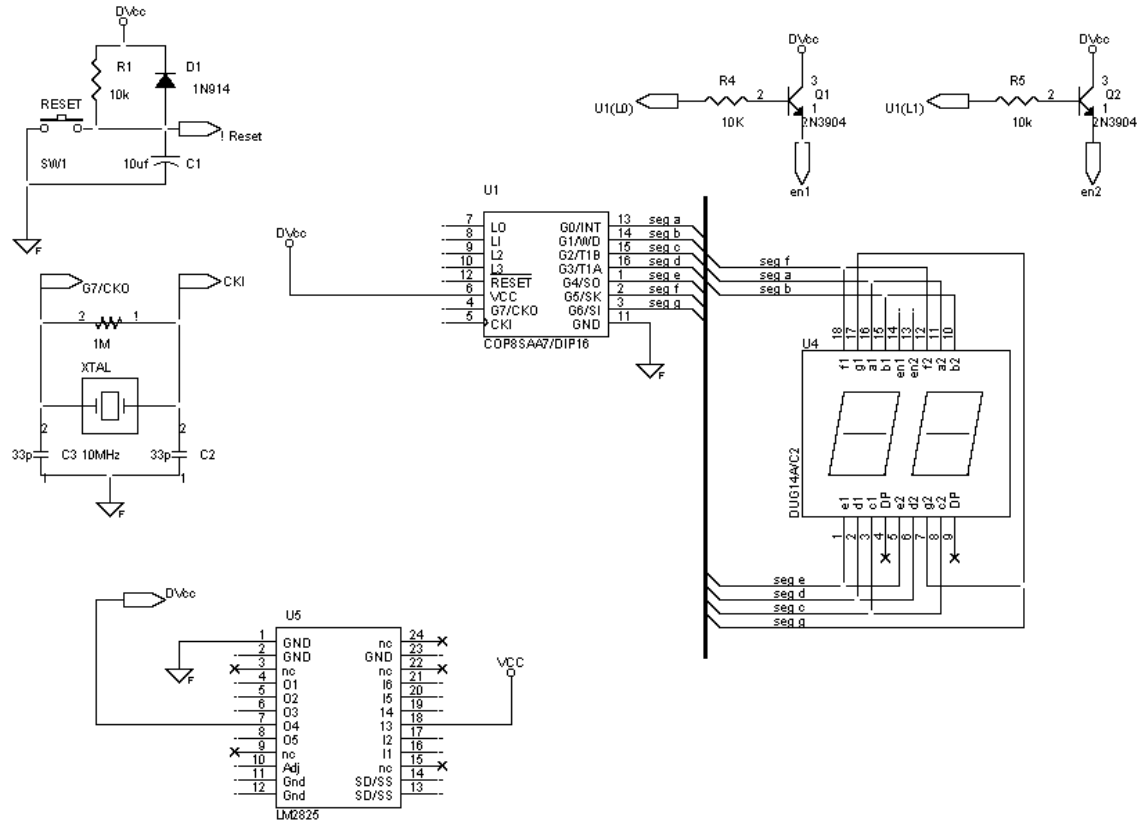
## Encapsulados del COP8CBR \*



\* Algunas de las versiones del COP8 en Montaje de Superficie y PLCC no son mostradas.  
Para más información acceder la página [www.national.com/cop8](http://www.national.com/cop8)

# APENDICE C

C.1 El siguiente esquemático incluye el COP8SA configurado con operación de cristal externo y conectado de forma básica con un display de dos dígitos alimentado por una fuente conmutada LM2825.



## BIBLIOGRAFIA

---

COP8SAx Designer's guide  
Lit 620894-002  
Agosto 1997

Feature Family user's manual  
Lit 620897-003  
Septiembre 1996

Assmbler/Linker/Librarian user's manual  
Lit 620896-004  
Noviembre 1997

COP8 Microcontroller Databook  
Lit 400004  
1996/1997

Flash family user's manual  
Lit 620889-002  
Octubre 2000

---

**Elaborado por :**

Ing. Jesús Flores V.  
Staff Field Applications Engineer  
National Semiconductor Corp.  
Guadalajara, Jal.  
Enero-2001

**Con la cooperación y revisión de :**

Ing. Marco C. Fernandez  
Staff Field Applications Engineer  
National Semiconductor Corp.  
Mexico, DF / Monterrey

Ing. Norbel Navarro  
Staff Field Apps Engineer  
National Semiconductor Corp.  
Scottsdale, AZ. EUA.

---

Todo el material expuesto es propiedad intelectual de National Semiconductor Corp.  
COP8, MICROWIRE/PLUS y WATCHDOG son marcas registradas de National Semiconductor Corp.  
Derechos Reservados de National Semiconductor Corp. 2001

Los productos de National Semiconductor no estan autorizados para ser usados como componentes críticos o sistemas que en los que dependa la vida de alguien sin la autorización, por escrito, del Presidente de National Semiconductor Corporation. National Semiconductor no asume ninguna responsabilidad por uso de cualquier circuitería descrita aún con licencia de patentes de circuitos con productos implicados. National se reserva el derecho, en cualquier momento y si notificación alguna, a cambiar su circuiteria y especificaciones.