65

# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN

"DESCRIPCION DE LA ARQUITECTURA GENERAL (HARDWARE Y SOFTWARE) DEL MICROCONTROLADOR PIC 16F84"

T E S I S

QUE PARA OBTENER EL TITULO DE:

**INGENIERO MECANICO ELECTRICISTA**

P R E S E N T A N :

**ALDO MONTIEL NAVARRO**

**SALVADOR RODRIGUEZ DAZA**

ASESOR: M. EN A.I. PEDRO GUZMAN TINAJERO



CUAUTITLAN IZCALLI, EDO. DE MEX. 2002

# PAGINACION DISCONTINUA

**FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN**
**UNIDAD DE LA ADMINISTRACION ESCOLAR**
**DEPARTAMENTO DE EXAMENES PROFESIONALES**

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES CUAUTITLAN

ASUNTO: VOTOS APROBATORIOS

DEPARTAMENTO DE
EXAMENES PROFE...

**DR. JUAN ANTONIO MONTARAZ CRESPO**
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN: Q. Ma. del Carmen García Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS:
"Descripción de la arquitectura general (hardware y software) del microcontrolador PIC16F84"

que presenta el pasante: Aldo Montiel Navarro
con número de cuenta: 9409921-8 para obtener el título de :
Ingeniero Mecánico Electricista

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E
**"POR MI RAZA HABLARA EL ESPIRITU"**
Cuautitlán Izcalli, Méx. a 10 de Septiembre de 2002

| | | |
|---|---|---|
| PRESIDENTE | Ing. Yolanda Benítez Trejo | |
| VOCAL | Ing. Jorge Buendía Gómez | |
| SECRETARIO | MAI. Pedro Guzmán Tinajero | |
| PRIMER SUPLENTE | Ing. Guillermo Santos Olmos | |
| SEGUNDO SUPLENTE | Lic. Dulce María Ligia Malo Ortega | |

**FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN**
**UNIDAD DE LA ADMINISTRACION ESCOLAR**
**DEPARTAMENTO DE EXAMENES PROFESIONALES**

ASUNTO: VOTOS APROBATORIOS

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES CUAUTITLAN

**DR. JUAN ANTONIO MONTARAZ CRESPO**
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN: Q. Ma. del Carmen García Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS:

"Descripción de la arquitectura general (hardware y software) del microcontrolador PIC16F84"

que presenta el pasante: Salvador Rodríguez Daza
con número de cuenta: 9401452-5 para obtener el título de:
Ingeniero Mecánico Electricista

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E
**"POR MI RAZA HABLARA EL ESPIRITU"**
Cuautitlán Izcalli, Méx. a 10 de Septiembre del 2002

| | | |
|---|---|---|
| PRESIDENTE | Ing. Yolanda Benítez Trejo | |
| VOCAL | Ing. Jorge Buendía Gómez | |
| SECRETARIO | MAI. Pedro Guzmán Tinajero | |
| PRIMER SUPLENTE | Ing. Guillermo Santos Olmos | |
| SEGUNDO SUPLENTE | Lic. Dulce María Ligia Malo Ortega | |

## AGRADECIMIENTOS

Agradecemos a la Universidad Nacional Autónoma de México, en especial a al F.E.S. Cuautitlán por la oportunidad de realizarnos profesionalmente.

Agradecemos al M. en A. I. Pedro Guzmán Tinajero por su apoyo como maestro en el transcurso de nuestra carrera y como asesor de esta tesis.

A los verdaderos maestro de la carrera de I.M.E..

Así mismo a los profesores que dieron un poco de tiempo para la revisión de esta tesis.

Un especial agradecimiento a los compañeros de la generación 97 de la carrera de I.M.E., que nos brindaron su apoyo


***"POR MI RAZA HABLARÁ EL ESPÍRITU"***

***A. Montiel***
***S. Rodríguez***


E AGRADEZO Y TE DEDICO ESTE TRABAJO "CABALLO VIEJO" PORQUE SIEMPRE ESTUVISTE CONMIGO MAS EN LAS MALAS, QUE EN LAS BUENAS, TE QUIERO.


ADRE, GRACIAS POR TU APOYO INCONDICIONAL, POR LAS DESVELADAS, LAS DESMAÑANADAS, POR TODO TU AMOR, GRACIAS, SIN EL DEBERAS QUE NO LLEGO.


LFRED, GRACIAS HERMANO, POR SER UN GRAN GUIA Y AMIGO EN UN CAMINO TAN OBSCURO, A MI GURU SE LO DEBO TODO, TE QUIERO.


ANY, HERMANO, GRACIAS POR TUS APRECIABLES CONSEJOS, CARIÑO, Y LA FUERZA INMENSA QUE ME DAS GRACIAS POR SOPORTAR ESA LUZ DE MADRUGADA, TE QUIERO.


STEDES TAMBIEN TIENEN UN LUGAR MUY ESPECIAL, GRACIAS POR SU INNUMERABLE APOYO, GRACIAS: ALFRED MONTIEL (KIWI), GRACIAS DANY MONTIEL, Y A LOS AMIGOS DEL LADO OBSCURO, MOI ALVAREZ (EL MOI: YES, MASTER) DANY ALVAREZ (EL MAMER, YES MASTER), SALVADOR RODRÍGUEZ (EL DAZA, POR ESTAR SIEMPRE CONMIGO EN TODAS, PERO EN TODAS) JUAN CARLOS ROA (EL JUAN), A EL "PADRINO" (J.L PEREZ) A TI MARY SOTELO POR SER LA MEJOR AMIGA Y POR SOPORTAR A ESTE "TIRANO" QUE TE QUIERE MUCHO.


INDUDABLEMENTE A TI..... QUE SIEMPRE ESTAS EN TODAS PARTES... GRACIAS DIOS.

A. MONTIEL.

DEDICO ESTA TESIS A MARY SOTELO, GRACIAS POR ESTAR SIEMPRE AHÍ

CUANDO TE NECESITE, POR DARME PACIENCIA Y ENSEÑARME

LA TRANQUILIDAD DE LA TORMENTA, TE QUIERO.

".....VIENEN A QUE EL GRAN LOBO SE LOS COMA....."

ADOLF HITLER, 1943.

## *AGRADECIMIENTOS*

*Agradezco en primer lugar a DIOS por darme la oportunidad de vivir hasta concluir un gran proyecto como lo es una carrera universitaria.*

*A mis padres Rosa Daza y Martín Rodríguez, por los años de apoyo, esfuerzo y sacrifico para darme los estudios que ahora concluyo gracias a ellos.*

***Este triunfo también es suyo.***

*A todos mis hermanos por los buenos ratos y su apoyo incondicional.*

*A Ma. de Lourdes Hernández Baez por todo lo que me ha enseñado (aun sin proponérselo), por su compañía, su confianza, por aguantarme y creer en mí. Pero sobre todo, por su amistad.*

***GRACIAS AMIGA MIA.***

*A Aldo Montiel por su apoyo académico, tanto en el desarrollo de la carrera como en la creación de esta tesis. Por escucharme, ayudarme y por lo buenos consejos.*

*Simplemente por ser un* ***BUEN AMIGO.***

*De manera muy especial a E. Fonseca Sánchez para quien no encuentro palabras que describan todo mi agradecimiento.*

*Que* ***DIOS te bendiga*** *y te cuide donde quieres que te encuentres.*

***S. Rodríguez Daza***

## ÍNDICE

**OBJETIVO GENERAL**

Describir la arquitectura, tanto de hardware como de software, del microcontrolador PIC16F84 para utilizarlo de forma práctica en el ámbito didáctico y como auxiliar en el diseño de prototipos de control.

**OBJETIVOS PARTICULARES**

Detallar la arquitectura externa (hardware) del microcontrolador PIC16F84 para utilizarlo en distintas formas y proyectos.

Describir la arquitectura del software (set de instrucciones) de este dispositivo con la finalidad de programarlo correctamente en una aplicación práctica.

Dar a conocer las características del software MPLAB necesarias para escribir, depurar y optimizar los programas diseñados en el uso de los microcontroladores PIC16F84.

Explicar el uso del software y hardware del NOPPP (No Piece Programer Pic), útil en la carga de programas a la memoria del microcontrolador PIC16F84

Realizar una práctica final, como un uso práctico del microcontrolador y así demostrar que la información teórica de este trabajo es suficiente al realizar un proyecto cualquiera que involucre al PIC16F84.

## HIPÓTESIS

El microcontrolador PIC16F84 de Microchip, es un microcontrolador muy adecuado en uso didáctico considerando su versatilidad, la facilidad de programación y su forma de borrado (eléctricamente), así como la amplia información que se puede encontrar referente al mismo.

## ALCANCES.

Esta tesis busca cumplir con los objetivos propuestos, con la meta especifica de verificar si es cierta o no la hipótesis. Solo será tratado un microcontrolador (el PIC16F84) de forma completa, con la finalidad de no hacer confusa la información en torno a estos componentes y compararlo de una manera básica con otros dispositivos de 8 bits semejantes, como es el caso de los microcontroladores de la familia del 83C751 de PHILIPS, el MC68HC705J1 de MOTOROLA y los de la serie COP de NATIONAL. Así mismo, se mencionarán los elementos básicos necesarios para programar el microcontrolador de una manera rápida y económica.

Los primeros temas son importantes para un mejor entendimiento de los capítulos que componen esta tesis. Después se profundizará en la estructura del microcontrolador (Hardware y Software), para comprender su funcionalidad y versatilidad en el ámbito didáctico y como un gran aliado en el diseño de prototipos de control.

Posteriormente se describirá la forma de realizar y depurar los programas para el microcontrolador con auxilio de las herramientas del software MPLAB, que se analiza solo de forma básica pero suficiente para su manejo. Éste se utiliza debido a la amplia información que provee respecto al programa para el controlador.

El uso del cargador NOPPP (No Piece Programer PIC o Programardor de PIC sin Piezas) es muy importante (y muy económico) para grabar la memoria del PIC, por ello se menciona en forma extensa. En su software se tiene la ventaja de realizar un auto diagnóstico para evitar errores de armado del circuito del NOPPP, motivo por el cual se usa éste en lugar de otros que tienen el mismo fin.

La realización del proyecto al final de esta tesis tiene el objeto de mostrar una aplicación del microcontrolador, basándose en la información contenida en este documento.

Será posible seguir los temas paso a paso, para poder llegar a los resultados deseados, o bien, se podrán leer los temas que se consideren convenientes para obtener resultados rápidos en el uso de los microcontroladores.

Si bien los temas se tratarán en forma básica, se debe tener en cuenta que el alcance real de la tesis no solo será para uso didáctico, también contempla la posibilidad de usarla para proyectos de índole industrial (ya que las características del PIC lo permiten).

## INTRODUCCIÓN

Gracias a la integración de miles de circuitos en un solo chip fue posible desarrollar el microprocesador y sus periféricos, como las memorias, los relojes, los puertos de entrada / salida, pudiendo crear las primeras computadoras compactas, pues el requisito previo para la producción de éstas era esa integración minúscula. La misma ha provocado que un chip contenga a un microordenador, o mejor conocido como un microcontrolador.

Pero ¿qué es un microcontrolador? y ¿cuál es la diferencia con un microprocesador?. Estas dos preguntas se responden en los primeros temas de este trabajo, analizando primeramente conceptos básicos y posteriormente a ambos dispositivos para diferenciarlos Y ¿para qué usar un microcontrolador si existe el microprocesador?, una de las grandes ventajas de usar un microcontrolador es que, debido a que contiene en si mismo a los periféricos, todo proyecto a realizar con él reduce su costo y su tamaño, por lo tanto resulta económico y mucho más fácil de usar.

Es muy cierto que la electrónica es muy amplia y habiendo muchos tipos de microcontroladores, ¿cuál será el microcontrolador más apropiado para un proyecto?, ello dependerá de las características del proyecto que desee realizar, el cual puede ser tanto de uso industrial como de uso didáctico. Y es aquí en donde se encuentra la importancia de este estudio en particular, ya que se pretende dar a conocer un componente de este tipo dentro del medio de la enseñanza, principalmente para actualizar el conocimiento de nuevas tecnologías para alumnos interesados en ellas.

El objetivo de esta tesis es ofrecer un entendimiento más a fondo del microcontrolador PIC16F84 con motivo de obtener los conocimientos necesarios para la programación de estos dispositivos y su uso en el diseño de sistemas digitales o híbridos (analógicos - digitales) basados en ellos con fines didácticos.

Se preguntará ¿por qué el PIC es el más apropiado para ese uso?, pues ciertamente no es el "más apropiado" pero, conforme avance en la lectura de este trabajo y vaya conociendo las características de este dispositivo, verá si es o no uno de los que se pueden considerar como muy adecuado dentro de la enseñanza; ¿qué tan sencillo y útil puede resultar el uso de un dispositivo como este?, ¿existirá información suficiente para entenderlo?. Es esa otrarazón para realizar esta tesis, dar a conocer las virtudes de un componente como el PIC, la versatilidad del mismo, su bajo consumo de energía y su precio que es bastante accesible; gozando de abundancia de información tanto en libros completos como en Internet. Estos son los motivos que dan la justificación para realizar una tesis a cerca de un componente bastante interesante como lo es el microcontrolador PIC16F84.

Esta investigación comprende a fondo únicamente el microcontrolador PIC16F84 fabricado por la empresa Microchip Technology, (y en forma básica otros microcontroladores de 8 bits para mostrar las ventajas del primero ante los últimos), además de las herramientas necesarias para la creación, simulación y carga de programas en dicho componente; más esta limitación no se considera importante, ya que si se comprende su funcionamiento básico, los demás microcontroladores pueden aprenderse con facilidad partiendo de la estructura del primero. Además, el estudio particular de éste elimina la posibilidad de una presentación superficial o confusa y permite al lector enfrentarse a problemas reales en la práctica.

Este estudio comenzará desde las raíces del microcontrolador y llegará a un nivel medio-avanzado. Abarcará los conceptos teóricos y prácticos con los que será posible realizar diseños para usar el microcontrolador PIC16F84 con diversos elementos según su aplicación como motores paso a paso, display de siete segmentos, diodos emisores de luz individuales, pulsadores, relays, memorias seriales, comunicación serial, teclados matriciales y mucho más.

El tipo de estudio que se realiza en esta tesis es tanto de investigación documental descriptiva y explicativa como de experimentación, lo cual permite conocer más a fondo este microcontrolador, logrando que cuando se lea el contenido de esta tesis sea posible conocer teóricamente el PIC16F84, y posteriormente conocerlo en la práctica por medio de un proyecto de generación de patrones de video.

La estructura de los temas responde a la necesidad de mostrarle al lector, que por primera ocasión estudia microcontroladores, la forma más sencilla de aprender, primero lo referente a la constitución externa e interna del dispositivo y su arquitectura en general; una vez dominada pasar entonces a los detalles de la programación. Sin embargo, la tesis en si misma no exige que se siga este orden, sino por el contrario tiene la facilidad para que se puedan adoptar otras modalidades en la secuencia de lectura, generando así mayor interés en el lector.

Las extensas áreas de aplicación de estos microcontroladores exigirán un gigantesco trabajo de diseño y fabricación. Aprender a manejar y aplicar estos dispositivos sólo se consigue desarrollando de forma práctica diseños reales.

# CAPITULO 1

# GENERALIDADES

# CAPITULO 1. GENERALIDADES

La necesidad de diferenciar a los microprocesadores de los microcontroladores, así como los elementos que operan en conjunto con los mismos, es esencial para todo aquel que está interesado en el ancho mundo de la electrónica, y más cuando se requiere que alguno sea utilizado con fines prácticos en el ámbito didáctico.

Es por ello que en este capítulo se expondrán algunos conceptos para poder encontrar las diferencias más importantes entre el microprocesador y el microcontrolador, y poder observar las ventajas de éste último frente al primero. Además, encontrará información que puede resultar útil para un mejor análisis y comparación de estas diferencias.

## TEMA 1.1. MEMORIAS

Las memorias son dispositivos de almacenamiento de datos binarios de largo o corto plazo.

Como regla general las memorias almacenan datos en unidades comúnmente de 8 bits (conocidas como bytes). Una unidad completa de información se denomina ***palabra*** y está formada por uno o varios bytes.

Cada elemento de memoria puede almacenar un '1' o un '0' (valores lógicos) y se denomina celda. Las memorias están formadas por matrices de celdas. La situación de cada celda se especifica por una fila y una columna. Una memoria de 64 celdas se puede organizar como una matriz de 8 bytes, como se ilustra en la figura 1.1.

Una memoria se identifica por el número de palabras que puede almacenar multiplicado por el tamaño de la palabra. Por ejemplo una memoria de 16K x 4 puede almacenar 16.384 palabras de 4 bits. Es decir, la memoria se identifica por su ***capacidad***.

La posición de una unidad de datos en una matriz de datos se denomina ***dirección***. La dirección de un bit será la fila y la columna, y la dirección de un byte la fila.

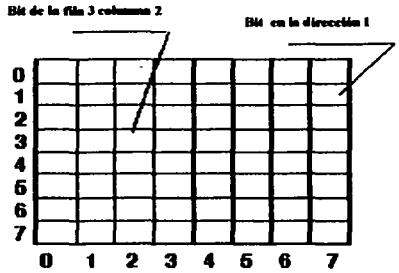



*FIGURA 1.1 ESQUEMA DE UNA MEMORIA DE 8 X 8*

### 1.1.1. OPERACIONES BÁSICAS DE LAS MEMORIAS

Las operaciones básicas de una memoria son las de escritura y lectura. La operación de ***escritura*** coloca los datos en una posición específica de la memoria y la operación de ***lectura*** extrae los datos de una posición específica de la memoria.

Los datos se introducen y se extraen a través de un conjunto de líneas denominado ***bus de datos*** (figura 1.2) Además, en las operaciones de escritura y de lectura se tiene que seleccionar una dirección introduciendo un código binario, que representa la dirección deseada, en un conjunto de líneas denominado ***bus de direcciones***. El código de dirección se decodifica y de esa forma se selecciona la dirección adecuada.

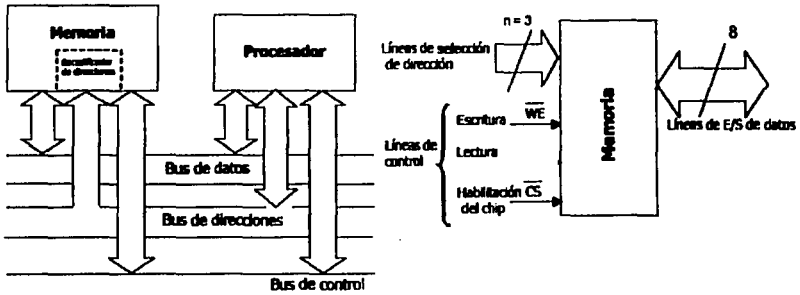



*FIGURA 1.2 DIAGRAMA DE UNA MEMORIA Y LOS BUSES DE COMUNICACIÓN*

### 1.1.1.1. OPERACIÓN DE ESCRITURA

Para almacenar un byte de datos en memoria, se introduce en el bus de direcciones el código binario de la posición de memoria donde se quiere escribir el dato.

Una vez que el código de dirección está ya en el bus, el decodificador de direcciones lo decodifica y selecciona la posición de memoria especificada. La memoria recibe entonces, del bus de control una orden de escritura y los datos almacenados en el registro que los contiene se colocan en la dirección de memoria seleccionada (figura 1.3). Cuando se escribe un nuevo byte de datos en una dirección de memoria se destruye el byte que estaba en esa dirección.

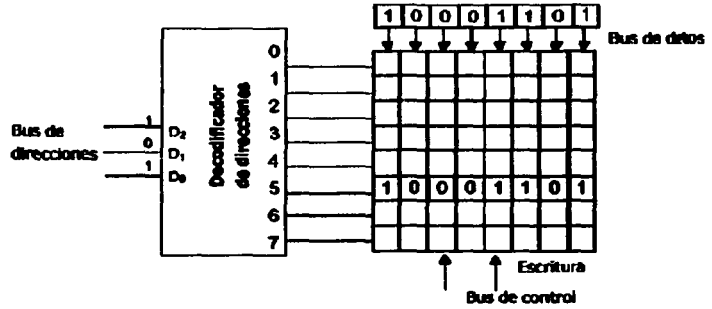



*FIGURA 1.3 OPERACIÓN DE ESCRITURA*

**La figura 1.4 muestra el ciclo para la escritura de algunas memorias.**

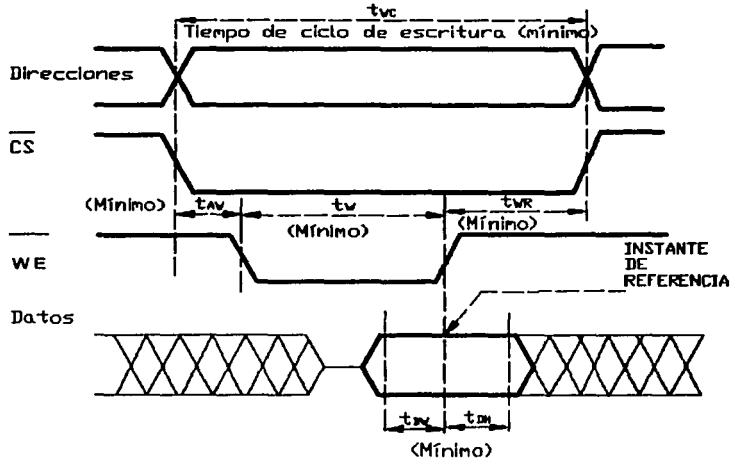
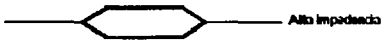
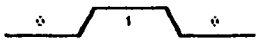
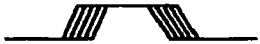



*FIGURA 1.4 CICLO DE ESCRITURA*

Donde:

| | |
|---|---|
|  | Es una señal compuesta de señales de varias líneas |
|   | Es un estado de alta impedancia |
|   | Es la representación del "0" y "1" lógicos |
|  | Es el instante de cambio no determinado |
|  | Es un valor de la señal irrelevante. |

Además:

$t_w$=Mínimo tiempo del impulso de escritura en $\overline{WE}$

$t_{DW}$=Mínimo tiempo anterior a $\overline{WE}$ con dato estable

$t_{DH}$=Mínimo tiempo posterior a $\overline{WE}$ con dato estable

$t_{AW}$=Mínimo tiempo entre dirección y $\overline{WE}$

$t_{WR}$=Mínimo tiempo entre $\overline{WE}$ y nueva dirección

$t_{WC}$=Mínimo tiempo de ciclo de escritura

Básicamente el ciclo de escritura es:

1. Establecer la posición a escribir en las entradas de direcciones
2. Activar el chip de memoria ( $\overline{CS}$ )
3. Establecer el dato a escribir en las entradas de datos
4. Activar la entrada $\overline{WE}$ para escribir.

### *1.1.1.2. OPERACIÓN DE LECTURA*

De nuevo se introduce en el bus de direcciones el código binario de la posición de memoria de donde se quiere leer el dato. El decodificador de direcciones decodifica dicho código y selecciona la posición de memoria especificada. La memoria recibe entonces, del bus de control una orden de lectura y una copia del byte de datos, almacenado en la dirección de memoria seleccionada, se introducen en el bus de datos y se carga en el registro de datos. Cuando se lee un byte de datos en una dirección de memoria éste sigue almacenado en dicha dirección.

El ciclo de lectura se muestra en la figura 1.5

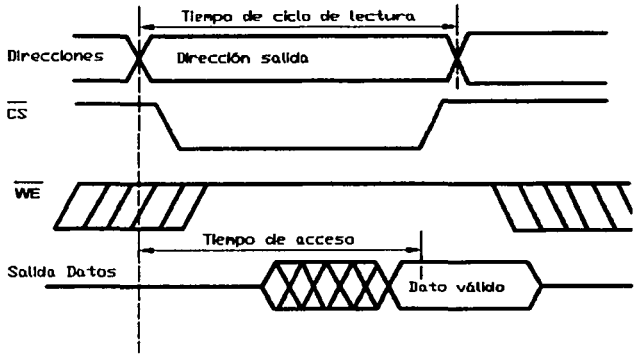



*FIGURA 1.5 CICLO DE LECTURA*

Resumiendo, básicamente para la lectura de una memoria se debe:

1. Establecer la dirección de la posición a leer en las entradas de direcciones
2. Activar el chip de memoria ( $\overline{CS}$ )
3. Activar la entrada $\overline{WE}$ para lectura.

### 1.1.2. TIPOS DE MEMORIA

Existen distintos tipos de memoria que tienen características diferentes. Algunas de ellas son las que se muestran a continuación en la figura 1.6 y que posteriormente se explicarán con más detalle:

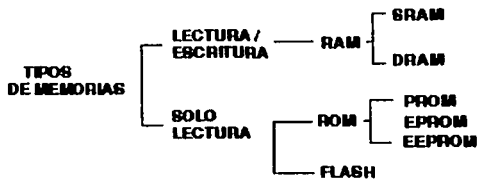



*FIGURA 1.6 ESQUEMA BÁSICO DE DIFERENTES TIPOS DE MEMORIAS*

#### *1.1.2.1. RAM*

La información que debe cambiarse durante el funcionamiento de un programa suele almacenarse en **memoria de acceso aleatorio RAM** (*random access memory*), que es el nombre asignado a la memoria de escritura y lectura rápidas. El nombre se debe al hecho de que en estos dispositivos se puede tener acceso a cualquier byte de información con la misma rapidez (esto no es cierto para dispositivos de almacenamiento como las cintas magnéticas).

Un nombre más apropiado sería memoria de lectura/escritura (RWM: Read-Write Memory), pero la palabra RAM es de uso universal y ha quedado establecida.

La RAM se implanta básicamente mediante una de las dos técnicas siguientes:

**A) RAM ESTÁTICA (SRAM)**

La ***RAM estática (SRAM)*** utiliza un circuito biestable, la información que se escribe en este dispositivo se mantiene indefinidamente siempre y cuando se mantenga la alimentación.

La figura 1.7 presenta un diagrama lógico funcional de una celda SRAM. La celda se selecciona poniendo a nivel alto las líneas de fila y de columna. Cuando la línea $\overline{\text{WRITE}}$ está a nivel bajo (escritura), el bit de datos de entrada se escribe en la celda. Cuando la línea $\overline{\text{WRITE}}$ está a nivel alto (lectura), la celda no se ve afectada, pero el bit de dato almacenado pasa a la línea de salida de dato.

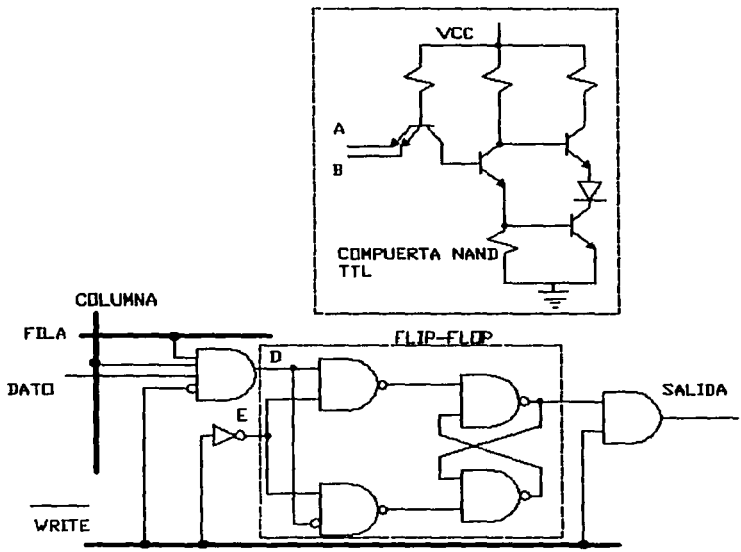



*FIGURA 1.7 CELDA SRAM*

Observe como una celda se compone de compuertas AND y NAND y, a su vez, las compuertas se componen de transistores como se observa en le caso de la NAND. Por esto una memoria SRAM tiene muchos componentes pero poca memoria en comparación con otras y, por lo tanto, la velocidad de acceso se vuelve muy lenta.

### B) RAM DINÁMICA

La **RAM dinámica (DRAM)** almacena información mediante la carga o descarga de una matriz de condensadores (figura 1.8). La RAM dinámica requiere mucho menos componentes por cada bit de información almacenada, lo cual permite que se integren más elementos de almacenamiento dentro de un solo chip. Sin embargo, tiene la desventaja de que las cargas en el condensador tienden a desvanecerse con el tiempo, lo cual hace necesario que se refresquen los dispositivos en forma periódica mediante la aplicación de una secuencia apropiada de señales de control. En este tipo de celda el transistor actúa como interruptor.

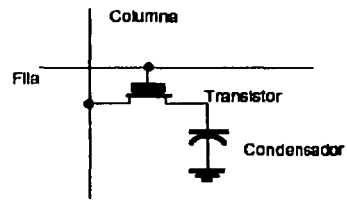



*FIGURA 1.8 CELDA DRAM*

Una de las características de la RAM es que es ***volátil.*** Es decir, pierde su contenido una vez que se ha desconectado la alimentación.

En la actualidad se fabrican ***RAM no volátiles***, aunque en realidad se trata de RAM volátiles con muy bajo consumo (elaboradas por medio de tecnología CMOS) con una batería integrada. Estos dispositivos tienen una vida útil de unos diez años y resultan adecuados para muchas aplicaciones en las que se requiere la retención de los datos.

También es posible encontrar la memoria DRAM sincrónica (SDRAM) que es una nueva tecnología de DRAM que utiliza un reloj para sincronizar la entrada y la salida de señales en un chip de memoria. El reloj está coordinado con el reloj de la CPU, para que la temporización de los chips de la memoria y de la CPU estén sincronizados. La DRAM sincrónica ahorra tiempo al ejecutar los comandos y al transmitir los datos, aumentando de esta manera el rendimiento total del ordenador.

Otro tipo es la Double Data Rate o SDRAM II que es la próxima generación de la SDRAM. La DDR se basa en el diseño de la SDRAM, con mejoras que suponen un aumento de la velocidad de transferencia. Como resultado de esta innovación, la DDR permite la lectura de datos tanto en la fase alta como baja del ciclo del reloj, con lo que se obtiene el doble de ancho de banda que con la SDRAM estándar. La DDR duplica la velocidad respecto a la tecnología SDRAM sin aumentar la frecuencia del reloj.

#### *1.1.2.2. ROM*

La ROM *(read only memory)* es la **memoria de sólo lectura,** es decir, el procesador puede leer de ella pero no puede escribir en ella. Estos dispositivos no son volátiles, y por tanto son adecuados para almacenar programas o cualquier información que no deba cambiar. Hay muchos tipos de ROM como se muestra en la tabla 1.

| Siglas | Descripción |
|---|---|
| ROM | Memoria de sólo lectura |
| PROM | Memoria de sólo lectura programable |
| EPROM | Memoria de sólo lectura programable y borrable |
| EEPROM | Memoria de sólo lectura eléctricamente programable y borrable |

*TABLA 1 SIGLAS PARA DIVERSOS TIPOS DE ROM*

Algunas memorias de sólo lectura son programables por máscara **(ROM),** lo cual significa que el fabricante del chip lo programó en la última etapa de la producción. Ésta es la opción más atractiva para la producción de grandes tiradas, pero no es adecuada para el desarrollo de baja tirada por su elevado costo. Una ROM puede estar fabricada tanto en tecnología bipolar como MOS.

La figura 1.9 muestra celdas ROM bipolares. La presencia de una unión desde una línea de fila a la base de un transistor representa un '1' en esa posición. En las uniones fila/columna en las que no existe conexión de base, las líneas de la columna permanecerán a nivel bajo ('0') cuando se direccione la fila.

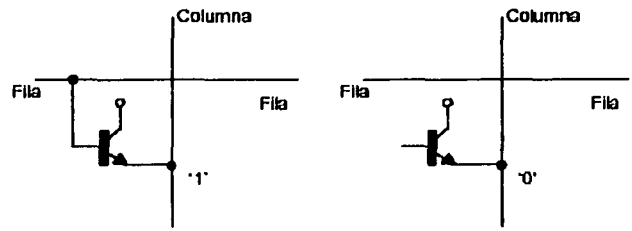



*FIGURA 1.9 CELDAS ROM BIPOLARES*

### A) PROM

Una alternativa para proyectos pequeños es el uso de una de las memorias de sólo lectura programables o **PROM** (*programmable read only memories*). Éstas existen en muchas variantes, pero todas permiten que el usuario programe el dispositivo por si mismo, ahorrándose el alto costo de la producción de la máscara. Una característica de estos dispositivos es que una vez programados no se les puede modificar.

### B) EPROM

Para lograr un desarrollo de sistemas flexible resulta provechoso tener un dispositivo de memoria que se pueda programar y luego reprogramar si fuera necesario.

Estas características son parte de las memorias de sólo lectura programables y borrables o ***EPROM*** (*erasable and programmable read only memories*). Aunque el término se puede aplicar a varios componentes, por lo general esta descripción se aplica a la memoria que se borra por exposición a la luz ultravioleta (UV).

Los chips cuentan con una ventana de mica que permite que la luz ultravioleta llegue a la superficie del silicio. La programación se realiza generalmente por medio de un programador EPROM. Las EPROM son muy utilizadas en el desarrollo de sistemas, en la elaboración de prototipos y en la producción de baja tirada. Sin embargo, tienen la desventaja de que por lo general se les debe retirar del circuito y colocar en un borrador y un programador especiales para que se puedan modificar.

**C) EEPROM**

Otra forma de PROM, la **EEPROM** (*electrically erasable and programmable read only memory*) se puede modificar en forma eléctrica sin necesidad de una fuente de luz ultravioleta, como sucedía con las anteriores. Esto permite modificar o cambiar un programa mientras el chip está colocado en su circuito.

Podría parecer que la EEPROM se debe clasificar como una RAM ya que se puede escribir (programar) así como leer. Sin embargo, debe hacerse notar que una RAM en general se puede escribir y leer en una fracción de microsegundo. Una EEPROM se puede leer a esta velocidad, pero es posible que se necesiten 10 ms para escribir un solo byte. Entonces la EEPROM es un dispositivo de rápida lectura pero lenta escritura, y se le describe mejor como una ROM que como una RAM.

### *1.1.2.3. MEMORIA TIPO FLASH*

Las memorias flash son memorias de lectura/escritura de alta densidad (alta densidad se refiera a gran capacidad de almacenamiento de bits) no son volátiles, lo que significa que los datos se pueden almacenar indefinidamente sin necesidad de alimentación (al igual que la ROM, EPROM o EEPROM).

Al poseer alta densidad puede almacenar en una pequeña superficie de chip gran cantidad de celdas. Esta alta densidad se consigue con celdas formadas por un único transistor MOS. Un bit de datos se almacena con la carga o ausencia de carga en la puerta.

Una memoria flash se puede reprogramar fácilmente dentro del sistema y la densidad de su memoria es comparable a la de la ROM y la EPROM, ya que todas utilizan celdas de un único transistor. La EEPROM utiliza un diseño de celda más complejo.

A modo de comparación observe los siguientes puntos:

Las memorias estáticas SRAM son volátiles, por lo que requiere de alimentación constante para mantener los datos. En muchas aplicaciones se utiliza una batería, sin embargo no se puede garantizar que los datos permanezcan porque siempre existe la posibilidad de que la batería falle. Como las celdas de una SRAM son biestables (formado por varios transistores) la densidad es relativamente baja.

Las memorias DRAM son de alta densidad pero no sólo requieren de alimentación constante para mantener los datos, sino que también los datos almacenados deben refrescarse frecuentemente. Típicamente una memoria flash consume menos potencia que una DRAM equivalente.

La tabla 2 muestra algunas diferencias entre las memorias mencionadas hasta el momento.

| Tipo de Memoria | No volátil | Alta densidad | Celda de un solo transistor | Re-escribible en el sistema |
|---|---|---|---|---|
| Flash | Si | Si | Si | Si |
| SRAM | No | No | No | Si |
| DRAM | No | Si | Si | Si |
| ROM | Si | No | Si | No |
| EPROM | Si | No | Si | No |
| EEPROM | Si | No | No | Si |

*TABLA 2 COMPARACIÓN DE LOS TIPOS DE MEMORIA.*

## TEMA 1.2. MICROPROCESADOR Y MICROCONTROLADOR

Hablando de microprocesadores y microcontroladores, para la mayoría de la gente parecen ser lo mismo. Es por ello que se hace necesario definir a ambos y sus diferencias más marcadas para saber en que lugar del mundo electrónico se encuentra cada uno.

### 1.2.1. MICROPROCESADOR

Es un circuito electrónico que actúa como unidad central de proceso de un sistema digital, proporcionando el control de las operaciones de cálculo. Los microprocesadores también se utilizan en otros sistemas informáticos avanzados, como impresoras, automóviles o aviones.

El microprocesador es un tipo de circuito sumamente integrado y programable. Los microprocesadores modernos incorporan hasta 10 millones de transistores (que actúan como osciladores o, más a menudo, como conmutadores), además de otros componentes como resistencias, diodos, condensadores y conexiones, todo ello en una superficie comparable a la de un sello postal.

Un microprocesador consta de varias secciones diferentes. La unidad aritmético-lógica (ALU por sus siglas en inglés) que efectúa cálculos con números y toma decisiones lógicas; los registros que son zonas de memoria especiales para almacenar información temporalmente; la unidad de control que decodifica los programas; los buses que transportan información digital a través del chip y de la computadora; la memoria local que se emplea para los cómputos realizados en el mismo chip.

#### *1.2.1.1. ACUMULADOR*

En los microprocesadores tradicionales todas las operaciones se realizan sobre el ***acumulador*** (registro de datos de trabajo). La salida del ***acumulador*** está conectada a una de las entradas de la ***ALU***, y por lo tanto éste es siempre uno de los dos operandos de cualquier instrucción.

Por convención, las instrucciones de simple operando (borrar, incrementar, decrementar, complementar), actúan sobre el ***acumulador***. La salida de la ***ALU*** va solamente a la entrada del ***acumulador***, por lo tanto el resultado de cualquier operación siempre quedará en el mismo (figura 1.10). Para operar sobre un dato de memoria, después de realizar la operación tendrá que mover siempre el ***acumulador*** a la memoria con una instrucción adicional

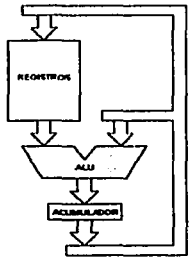



*FIGURA 1.10 ACUMULADOR EN EL MICROPROCESADOR TRADICIONAL*

Los microprocesadores más complejos contienen a menudo otras secciones; por ejemplo, secciones de memoria especializada denominadas memoria *caché*, que sirven para acelerar el acceso a los dispositivos externos de almacenamiento de datos. Los microprocesadores modernos funcionan con una anchura de bus de 64 bits.

Un cristal oscilante situado en el ordenador proporciona una señal de sincronización, o señal de reloj, para coordinar todas las actividades del microprocesador. Mientras que, quien se encarga de llevar la cuenta sobre los pasos que se están ejecutando, es el contador de programa.

También es posible que el microprocesador pueda ser interrumpido y para ello se precisa de un registro especial donde estén definidas las condiciones de operación del microprocesador.

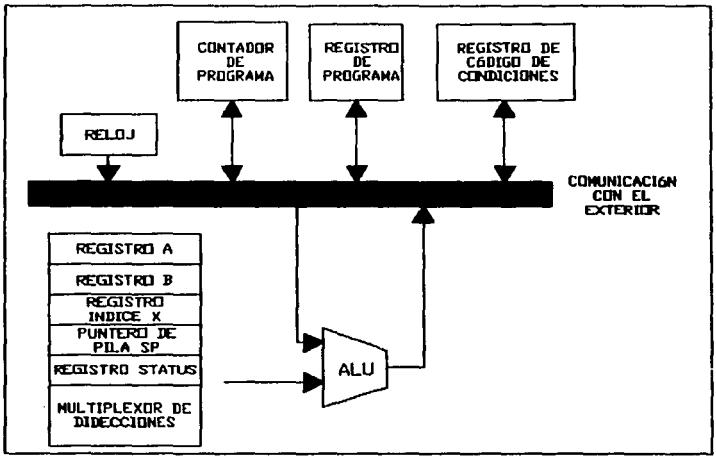



*FIGURA 1.11 BLOQUES INTERNOS DEL MICROPROCESADOR*

Quienes han tenido la oportunidad de realizar un diseño con un microprocesador han observado que dependiendo del circuito se requerían algunos circuitos integrados adicionales además del microprocesador. Para proporcionar la memoria necesaria se emplean otros circuitos integrados de memoria de acceso aleatorio (RAM), que contienen grandes cantidades de transistores.

La memoria RAM estática suele emplearse como memoria *caché* porque funciona a gran velocidad.

La memoria DRAM resulta más económica que la SRAM y se emplea como elemento principal de memoria en la mayoría de las computadoras. También memorias ROM para almacenar el programa que se encargaría del proceso del equipo

Además de las memorias, suele utilizarse un circuito integrado para los puertos de entrada y salida y finalmente un decodificador de direcciones.

Sin embargo, verá a continuación que esto ya no es necesario gracias al microcontrolador.

### 1.2.2. MICROCONTROLADOR

Un microprocesador no es un ordenador completo. No contiene grandes cantidades de memoria en si mismo, ni es capaz de comunicarse con dispositivos de entrada —como un teclado, un joystick o un ratón— o dispositivos de salida como un monitor o una impresora, como se expuso anteriormente.

Un tipo diferente de circuito integrado llamado **microcontrolador** es de hecho una computadora completa situada en un único chip, que contiene todos los elementos del microprocesador básico, además contiene en un solo integrado la Unidad de Proceso, la memoria RAM, memoria ROM, puertos de entrada, salidas y otros periféricos así como otras funciones especializadas. Los microcontroladores se emplean en video juegos, reproductores de vídeo, automóviles y otras máquinas.

## TEMA 1.3. COMPARACIÓN DEL MICROCONTROLADOR Y EL MICROPROCESADOR

Las comparaciones son reconocidas inmediatamente para aquellas personas que han trabajado con los microprocesadores y después pasaron a trabajar con los microcontroladores.

Estas son las diferencias más importantes que dan origen a las ventajas del microcontrolador frente al microprocesador:

Por ejemplo, la configuración mínima básica de un microprocesador estaba constituida por un Micro de 40 Pines, una memoria RAM de 28 Pines, una memoria ROM de 28 Pines y un decodificador de direcciones de 18 Pines; pero un microcontrolador incluye todo estos elementos en un solo circuito integrado por lo que implica una gran ventaja en varios factores: en el circuito impreso por su simplificación de diseño del circuito, el costo para un sistema basado en microcontrolador es mucho menor y, lo mejor de todo, el tiempo de desarrollo de su proyecto electrónico se disminuye considerablemente.

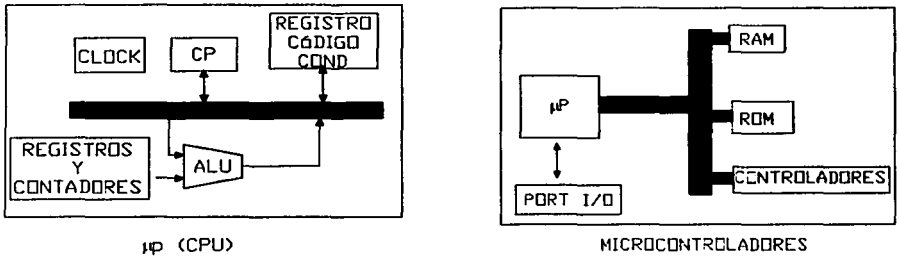



*FIGURA 1.12 MICROPROCESADOR Y MICROCONTROLADOR*

También se encuentran diferencias en cuanto al registro de trabajo de ambos integrados como se expone a continuación.

### 1.3.1. ACUMULADOR VS REGISTRO W[1]

Otra de las diferencias entre el microcontrolador y el microprocesador es la localización del registro de trabajo y el acumulador, respectivamente.

[1] http://www.iespana.es/portosin/ (España, 2002)

En los microcontroladores PIC16F84, la salida de la ***ALU*** va al registro de trabajo ***W*** y también a la memoria de datos, por lo tanto el resultado puede guardarse en cualquiera de los dos destinos. Cosa que no sucede en los microprocesadores tradicionales, en los que el acumulador está directamente en la salida de la ALU, lo cual implica que cualquier resultado de ésta deba pasar primero por el acumulador para posteriormente ser movido a otro registro de memoria (ver figura 1.13). Esta situación del microprocesador provoca que el programa para el mismo deba extenderse un poco más, al ser necesarias instrucciones para desplazar el contenido del acumulador a otro registro.

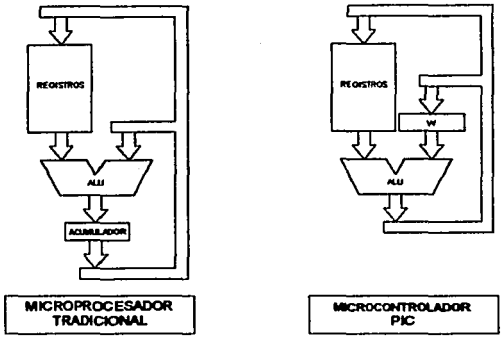



*FIGURA 1.13 ACUMULADOR Y REGISTRO W*

La gran ventaja de la arquitectura del microcontrolador PIC es que permite un gran ahorro de instrucciones ya que el resultado de cualquier instrucción que opere con la memoria (sea de simple o doble operando), puede dejarse en la misma posición de memoria o en el registro ***W***, según se seleccione con un bit de la misma instrucción. Las operaciones con constantes provenientes de la memoria de programa (literales) se realizan sólo sobre el registro ***W***.

Existen unos microcontroladores mas avanzados que otros por los componentes especiales que estos incluyen. Algunos solamente contienen puertos de entrada y de salida, otros incluyen puertos hasta de 12 Bits para conversiones analógicas digitales entre otras cosas. Algunas características especiales que poseen los microcontroladores actuales son: Comunicación Serial Síncrona, Comunicación Serial Asíncrona, Temporizadores, Contadores, etc.

## TEMA 1.4. TIPOS DE ARQUITECTURA INTERNA

Existen principalmente dos tipos de arquitectura interna en los microprocesadores y/o en los microcontroladores, las cuales se describen a continuación.

### 1.4.1. ARQUITECTURA VON NEUMANN[2]

La arquitectura tradicional de computadoras y microprocesadores está basada en la arquitectura tipo Von Neumann, en la cual la unidad central de proceso (CPU), está conectada a una memoria única donde se guardan las instrucciones del programa y los datos como se ve en la figura 1.14.

El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus que comunica la memoria con la CPU. Así un microprocesador de 8 bits (byte) con un bus de 8 bits, tendrá que manejar datos e instrucciones de una o más unidades de 8 bits de longitud. Si tiene que acceder a una instrucción o dato de más de un byte de longitud, tendrá que realizar más de un acceso a la memoria. Y el tener un único bus hace que el microprocesador sea más lento en su respuesta, ya que no puede buscar en memoria una nueva instrucción mientras no finalicen las transferencias de datos de la instrucción anterior.

---

[2] Op cit [1]

Resumiendo lo anterior, las principales limitaciones que se encuentran con la arquitectura Von Neumann son:

- La limitación de la longitud de las instrucciones por el bus de datos, que hace que el microprocesador tenga que realizar varios accesos a memoria para buscar instrucciones complejas.
- La limitación de la velocidad de operación a causa del bus único para datos e instrucciones que no deja acceder simultáneamente a unos y otras, lo cual impide superponer ambos tiempos de acceso.

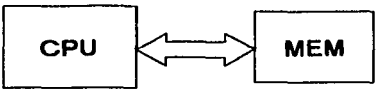



*FIGURA 1.14 ARQUITECTURA VON NEUMANN*

### 1.4.2. ARQUITECTURA HARVARD[3]

En la arquitectura Harvard se tiene la unidad central de proceso (CPU) conectada a dos memorias, una con las instrucciones y otra con los datos, por medio de dos buses diferentes, según se muestra en la figura 1.15.

Una de las memorias contiene solamente las instrucciones del programa (Memoria de Programa), y la otra sólo almacena datos (Memoria de Datos).

---

[3] Op cit [1]

Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC (Reduced Instruction Set Computer), el set de instrucciones y el bus de memoria de programa pueden diseñarse de tal manera que todas las instrucciones tengan una sola posición de memoria de programa de longitud.

Al ser los buses independientes, la CPU puede acceder a los datos para completar la ejecución de una instrucción, y al mismo tiempo leer la siguiente instrucción a ejecutar.

Las ventajas de esta arquitectura son:

- El tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.
- El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad en cada operación.

Una pequeña desventaja de los procesadores y microcontroladores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontraran físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).

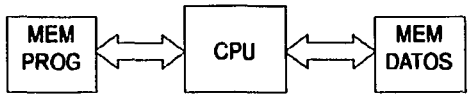



*FIGURA 1.15 ARQUITECTURA HARVARD*

El microcontrolador PIC16F84 posee arquitectura Harvard, con una memoria de datos de 8 bits, y una memoria de programa de 14 bits, por lo tanto su velocidad de operación es comparable con la velocidad que tienen los microprocesadores.

# CAPITULO 2

# EL MICROCONTROLADOR PIC16F84

# CAPITULO 2. EL MICROCONTROLADOR PIC16F84

Ciertamente el microcontrolador PIC no es algo nuevo, y su desarrollo tienen mucha historia en el curso de los avances electrónicos.

## TEMA 2.1. INTRODUCCIÓN AL MICROCONTROLADOR PIC16F84

En 1965, la empresa GI creó una división de microelectrónica, GI Microelectronics Division, que comenzó su andadura fabricando memorias EPROM y EEPROM. A principios de los años 70 diseñó el microprocesador de 16 bits CP1600, razonablemente bueno, pero que no manejaba eficazmente las Entradas y Salidas. Para solventar este problema, en 1975 diseñó un chip destinado a controlar E/S: el PIC (donde PIC significa Programmable Integrated Circuits). Se trataba de un controlador rápido pero limitado y con pocas instrucciones pues iba a trabajar en combinación con el CP1600.

La arquitectura del PIC, se comercializó en 1975. En aquel momento se fabricaba con tecnología NMOS y el producto sólo se ofrecía con memoria ROM y con un pequeño pero robusto microcódigo.

La década de los 80's no fue buena para GI, que tuvo que reestructurar sus negocios, concentrando sus actividades en los semiconductores de potencia. La compañía GI Microelectronics Division se convirtió en una empresa subsidiaria, y orientaron su negocio a los PIC, las memorias EPROM paralelo y las EEPROM serie. Se comenzó rediseñando los PIC, que pasaron a fabricarse con tecnología CMOS, surgiendo la familia de gama baja PIC16CSX, considerada como la "clásica".

Una de las razones del éxito de los PIC's se basa en su utilización. Cuando se aprende a manejar uno de ellos, conociendo su arquitectura y su repertorio de instrucciones, es muy fácil emplear otro modelo, ya que como se mencionó anteriormente el funcionamiento básico de los microcontroladores es el mismo principio para todos los demás PIC's.

Microchip cuenta con su factoría principal en Chandler, Arizona, en donde se fabrican y prueban los chips con los más avanzados recursos técnicos. Ha superado el millón de unidades por semana en productos CMOS de la familia PIC16CSX, lo cual refleja que día con día el uso de la microtecnología va formando parte de las actividades cotidianas.

En la actualidad existen muchos microcontroladores de diferentes empresas reconocidas mundialmente, pero esta tesis sólo trata al microcontrolador de Microchip Technology modelo ***PIC16F84.***

El microcontrolador PIC16F84 forma parte del grupo PIC16F8X, perteneciente a la familia de microcontroladores PIC16CXX, económicos, de alto rendimiento, CMOS, totalmente estáticos y de 8-bits. Además los microcontroladores PIC16F84 emplean una avanzada arquitectura RISC (Reduce Instruction Set Computer).

En esta familia se encuentran los microcontroladores: PIC16F83, PIC16F84, PIC16CR83 y PIC16CR84.

En la tabla 3 se muestra una breve comparación de los diferentes microcontroladores dentro de la familia a la que pertenece el PIC16F84.

| DISPOSITIVO | MEMORIA DE PROGRAMA | DATOS RAM (bytes) | DATOS EEPROM (bytes) | FRECUENCIA MAXIMA (MHz) |
|---|---|---|---|---|
| PIC19F83 | 512 FLASH | 36 | 64 | 10 |
| PIC16F84 | 1K FLASH | 68 | 64 | 10 |
| PIC16CR83 | 512 ROM | 36 | 64 | 10 |
| PIC16CR84 | 1K ROM | 68 | 64 | 10 |

*TABLA 3 COMPARACIÓN DE MICROCONTROLADORES PIC*[4]

[4] http://www.microcihip.com, (U.S.A. 2002)

Como se observa hay varios "tipos" de PIC's indicados en el dispositivo por un número y una letra[5], esta última se refiere a:

- F, como en PIC16F84. Estos dispositivos tienen la memoria de programa tipo FLASH y opera por encima del rango de voltaje normal.
- LF, como en PIC16LF84. Estos dispositivos tienen la memoria de programa tipo FLASH y opera por encima de un extenso rango de voltaje.
- CR, como en PIC16CR83. Estos dispositivos tienen memoria de programa tipo ROM y opera en un rango de voltaje normal.
- LCR, como en PIC16LCR84. Estos dispositivos tienen memoria de programa tipo ROM y opera en un rango de voltaje extenso.

Los dispositivos del grupo PIC16F8X han reforzado o perfeccionado los rasgos característicos de la familia de microcontroladores PIC16CXX, entre las que se encuentran múltiples fuentes de interrupción, tanto internas como externas y en los que la pila o stack tiene una capacidad de ocho niveles.

El bus separado de instrucción y de datos de la arquitectura de Harvard permite un ancho de la palabra de instrucción de 14-bits y 8-bits de ancho de la palabra de datos. Los dos diferentes buses permiten que todas las instrucciones puedan ejecutarse en un solo ciclo de reloj, salvo algunas que requieren de dos ciclos. Un total de 35 instrucciones (RISC) están disponibles, así como un lenguaje de directivas. Adicionalmente, un gran juego de registros logra un nivel de ejecución muy alto.

---

[5] "Microchip PIC Microcontrollers Data Book", Microchip Technology Inc., Microchip, 1997. U.S.A.

Los microcontroladores PIC16F84 logran una condensación de código de 2:1 y una mejora de velocidad de 4:1 (10 MHz) por encima de otros microcontroladores de 8-bits de su clase.

El PIC16F84 tiene 68 bytes de RAM, 64 bytes de memoria EEPROM de datos, y 13 pines de I/O. Un timer/count (cronómetro/contador) también está disponible.

La familia PIC16CXX, a la cual pertenece el PIC16F84, tiene características especiales para reducir los componentes externos, reduciendo así el costo, reforzando la fiabilidad del sistema y reduciendo el consumo de energía con cuatro opciones del oscilador externo. [6]

1. Oscilador tipo "HS" para frecuencias mayores de 4 MHz. En el caso del PIC16F84 podrá instalarse un oscilador hasta 10 MHz.
2. Oscilador tipo "XT" para frecuencias no mayores de 4 MHz.
3. Oscilador tipo "LP" para frecuencias entre 32 y 200 KHz. Minimiza el consumo de energía
4. Oscilador tipo "RC" para frecuencias no mayores de 4 MHz. Proporciona una solución económica

El modo de SLEEP (energía baja) ofrece economizar el consumo de energía, el usuario puede despertar al chip a través de varias interrupciones externas e internas y restablecerlo.

Cronómetro WATCH DOG (Perro guardián) está dentro de su propio chip con oscilador de RC para proporcionar protección contra ciclos de programa repetidos de manera infinita, con lo cual se evita el funcionamiento innecesario del microcontrolador.

---

[6] Ver Capitulo 6 OSCILADOR GENERADOR0 para mayor referencia al respecto.

Al programar el PIC deberá especificar en la palabra de configuración, el tipo de oscilador que utilizará en su proyecto electrónico (Tema 5.6 PALABRA DE CONFIGURACIÓN DEL PIC16f84)

## TEMA 2.2. CONFIGURACIÓN EXTERNA DEL PIC16F84

El microcontrolador PIC16F84 está constituido externamente por 18 terminales o pines, según se muestra en la figura 2.1

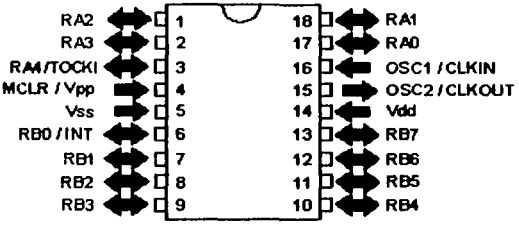



*FIGURA 2.1 TERMINALES DE ENTRADA / SALIDA DEL PIC16F84*

La descripción de cada uno se muestra en la tabla 4

*TABLA 4 CARACTERÍSTICAS DE LOS PINES DEL PIC16F84*[7]

| Nº pin | Nombre | Tipo E/S/A | Tipo de buffer | Descripción |
|---|---|---|---|---|
| 16 | OSC1/CLKIN | E | ST/CMOS (3) | Entrada del cristal oscilador / entrada de reloj |
| 15 | OSC2/CLKOUT | S | - | Salida del cristal oscilador. En el modo de oscilación por cristal se conecta al cristal o resonador. En modo RC OSC2 proporciona CLKOUT (salida de reloj), que posee 1/4 de la frecuencia de OSC1, y representa el ciclo de instrucción. |
| 4 | $\overline{MCLR}$ | E/A | ST | Entrada de reset / entrada de voltaje de programación. Este pin es un reset activo a nivel lógico bajo del dispositivo. |
| 17 | RA0 | E/S | TTL | PORTA es un puerto de E/S bidireccional<br>Puede ser seleccionado también para ser la entrada de reloj al contador/temporizador TMR0. |
| 18 | RA1 | E/S | TTL | |
| 1 | RA2 | E/S | TTL | |
| 2 | RA3 | E/S | TTL | |
| 3 | RA4/T0CKI | E/S | ST | |
| 6 | RB0/INT | E/S | TTL/ST (1) | PORTB es un puerto de E/S bidireccional que puede ser programado para levantar internamente todas las entradas.<br>RB0/INT puede ser seleccionado como un pin de interrupción externa RB4 a RB7 son pines de interrupción por cambio de estado.<br>RB6 es la entrada de reloj de programación y<br>RB7 la entrada de programación serie. |
| 7 | RB1 | E/S | TTL | |
| 8 | RB2 | E/S | TTL | |
| 9 | RB3 | E/S | TTL | |
| 10 | RB4 | E/S | TTL | |
| 11 | RB5 | E/S | TTL | |
| 12 | RB6 | E/S | TTL/ST (2) | |
| 13 | RB7 | E/S | TTL/ST (2) | |
| 5 | Vss | A | - | Referencia tierra (masa) para todos los pines lógicos |
| 14 | Vdd | A | - | Alimentación positiva |

**Leyenda y notas:** E = entrada, S = salida, E/S = entrada/salida, A = alimentación, - = no empleado, TTL = entrada TTL, ST = entrada Schmitt

(1) Este buffer es una entrada Schmitt cuando se configura como interrupción externa

(2) Este buffer es una entrada Schmitt cuando se emplea en el modo de programación serie

(3) Este buffer es una entrada Schmitt cuando se configura en el modo de oscilación RC y como entrada CMOS

[7] "Microchip PIC Microcontrollers Data Book", Microchip Technology Inc., Microchip, 1997. U.S.A. pag 9

### 2.2.1. PUERTOS DE ENTRADA Y SALIDA[6]

En la figura 2.2 se muestran los dos puertos del microcontrolador PIC16F84 denominados "A" y "B". El puerto "A" tiene 5 líneas disponibles **(RA0, RA1, RA2, RA3, RA4)** y el puerto "B" tiene 8 líneas disponibles **(RB0, RB1, RB2, RB3, RB4, RB5, RB6, RB7)**. Ambos Puertos suman un total de 13 líneas que podrán ser programadas mediante software independientemente como entradas o como salidas. Estas son las líneas que estarán destinadas a comunicar el microcontrolador con el mundo exterior.

También observe que el Pin No. 3 perteneciente al puerto "A" = RA4 también tiene otra nomenclatura denominada "TOCKI" lo cual quiere decir que esta línea podrá ser programada como entrada, salida o temporizador/contador.

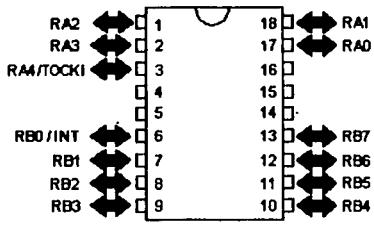



*FIGURA 2.2 TERMINALES DE ENTRADA / SALIDA DEL PIC16F84*

#### *2.2.1.1. LÍMITE DE CORRIENTE PARA LOS PUERTOS "A" Y "B"*

Los puertos "A" y "B" del microcontrolador podrán ser programados como entradas y salidas indistintamente. Para el caso de que sean programados como salida se denominan *"Modo Fuente"* por que suministran corriente y cuando son programados como entrada se denominan *"Modo Sumidero"* por que reciben corriente.

---

[6] http://www.iespana.es/portosin/ (España, 2002)

La máxima corriente que puede suministrar una línea programada como salida es de 20 mA, pero si se utilizan todas las líneas del puerto "A" como salida, no deberá exceder de 50mA para todo el puerto. Para el caso del puerto "B" no deberá exceder de 100 mA.

Si las programa como entradas, la corriente máxima que puede manejar una sola línea es de 25 mA. Para el caso del puerto "A" programado con todas sus líneas como entrada, la máxima es de 80 mA. En el caso del puerto "B" es de 150 mA. En la figura 2.3 se muestran los límites para los puertos A y B.

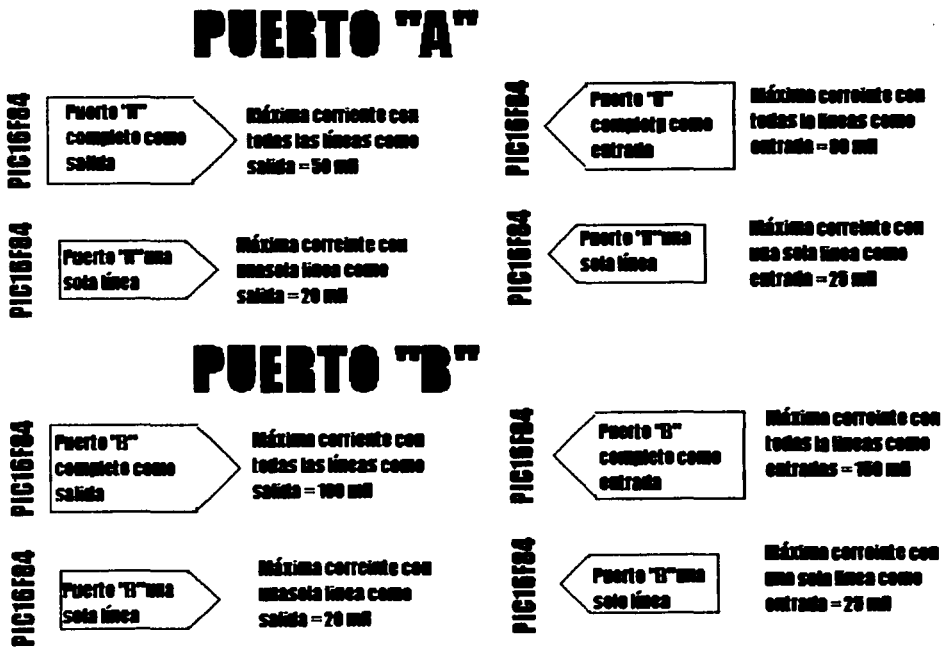



*FIGURA 2.3 MÁXIMA CORRIENTE EN LOS PUERTOS DE ENTRADA/SALIDA*

En caso de querer utilizar periféricos que manejen mayor cantidad de corriente de la especificada, habrá que aplicar un circuito acoplador como por ejemplo los buffers o transistores que se encarguen de controlar la corriente.

#### *2.2.1.2. CONFIGURACIÓN DE LOS PUERTOS DE ENTRADA / SALIDA*

Los bits de cada puerto se configuran mediante los bits correspondientes de un registro de control asociado que recibe el nombre de TRIS (registro de control de puerto). En realidad cada puerto soporta dos registros:

- El registro de datos, al que se denomina Puerto A o B (PortA o PortB).
- El registro de control TRISA o TRISB, con el que se programa el sentido (entrada o salida) de las líneas de cada puerto. [9]

Los Puertos A y B se corresponden con las posiciones 5 y 6 del área de datos. Cada uno de sus bits puede programarse como una línea de entrada o de salida, según se ponga un 1 ó un 0 en el bit del registro de control TRIS correspondiente.

Cualquier línea puede funcionar como entrada o como salida. Sin embargo, si actúa como entrada, la información que se introduce desde el exterior no se memoriza o se graba, pasa simplemente por un dispositivo triestado por lo cual el valor de dicha información debe mantenerse hasta que sea leída, es decir, la lectura se realiza en "tiempo real".

Cuando un pin de un puerto de E/S funciona como salida, el bit que proviene del bus de datos se guarda en un biestable de datos con lo cual la información que ofrece este pin permanece invariable hasta que se reescriba otro bit.

---

[9] Estos temas son tratados con detalle en el capítulo 4.2.1.9

Cuando se produce un RESET[10], todos los bits de los registros TRIS tendrán el valor 1 y todas las líneas de E/S actúan como entrada por motivos de seguridad y para evitar daños irreparables.

Hay que prestar mucha atención a las operaciones que, tras una lectura de un puerto sigue una escritura del mismo. Se debe dejar pasar un tiempo determinado para que se estabilice el voltaje de las patitas. Insertando entre la lectura y la escritura una instrucción NOP[11], o cualquier otra que no implique a los puertos, se eliminan estos errores potenciales.

Otra consideración importante, al momento de utilizar los puertos, es que cualquier operación que se realice sobre ellos implicará antes una lectura de los mismos, es decir, si se desea poner a uno cualquier pin de algún puerto, el microcontrolador primeramente tendrá que leer el estado actual del puerto para posteriormente escribir en él.

Suponga que tiene habilitados los pines 6 y 7 del puerto B como salida y la siguiente secuencia de programa:

```
BCF   PortB, 7
BCF   PortB, 6
```

Debería esperar el resultado 00 en estos bits, pero si por coincidencia tiene como entrada del bit 7 y un estado alto, el resultado no será el esperado al leer el puerto si no que se cambiará por 10, por ello debe tener mucho cuidado al momento de programar los puertos del micro, para evitar este tipo de errores muy comunes.

---

[10] Ver capitulo 4.2.2 referente a las condiciones de RESET

[11] Ver Capitulo 5 referente al SET de instrucciones

Si quiere insertar un dato en algún puerto, también debe considerar el tiempo mínimo que debe permanecer el dato presente en el mismo, dicho tiempo deberá de ser mayor a 15 ns a partir de que el programa ejecuta la instrucción de lectura. De no ser así se puede dar el caso de que no sea leído correctamente y se produzca un error en el resultado final. También se deben considerar los niveles adecuados para que se detecten los estados alto y bajo en los puertos, tanto de entrada como de salida, que para el caso del PIC16F84 son:

- Estado bajo de entrada y con oscilador tipo **XT, HS o LP:** como mínimo el nivel a tierra y como máximo 0.3 de $V_{DD}$ (Voltaje de alimentación del microcontrolador). Para oscilador **RC** el máximo es de 0.1 $V_{DD}$
- Estado alto de entrada y con oscilador tipo **XT, HS o LP:** como mínimo 0.7 $V_{DD}$ y como máximo es $V_{DD}$. Para oscilador **RC** el voltaje mínimo es $V_{DD}$.
- Estado bajo de salida en general para todo tipo de oscilador es de 0.6 V, considerando una alimentación de 4.5 V.
- Estado alto de salida generalmente será de $V_{DD} - 0.7$ V.

### 2.2.2. OTRAS TERMINALES

En lo que se refiere a los pines de alimentación para el funcionamiento del PIC16F84, estos son el pin 14 de entrada positiva que debe ser de 5 V preferentemente[12], y el pin 5 que es la referencia a masa para todos los pines.

[12] El rango del voltaje de alimentación está en función del oscilador externo a utilizar, así por ejemplo cuando se usa un oscilador tipo XT, RC o LP el rango de voltaje va de 4 a 6 volts; pero cuando se usa un oscilador tipo HS el rango es de 4.5 a 5.5 (Anexo 1).

También se observan pines configurados para el funcionamiento propio del PIC como lo son el pin 4 para el reset y el pin 16 para el reloj, además del pin 15 que se usa para poder extraer una señal de reloj del interior del PIC en el caso de que se desee y solo cuando use un reloj externo RC. Así mismo tiene pines que pueden ser utilizados como pines de interrupción externa, como es el caso del pin 6 (RB0) y los pines del 10 al 13 (RB4 a RB7) que se pueden utilizar como pines de interrupción al cambiar alguno de ellos.

Existen funciones especiales de algunos pines para poder realizar la programación del PIC16F84 según se explica en el Capitulo 8.

# CAPITULO 3

# CONFIGURACIÓN INTERNA DEL PIC16F84

## CAPITULO 3. CONFIGURACIÓN INTERNA DEL PIC16F84

En la figura 3.1 se muestra la arquitectura interna del microcontrolador PIC16F84, organizada en bloques interconectados, en donde se incluye la memoria RAM, la memoria EEPROM, la memoria FLASH, los puertos de entrada y salida (I/O) y todo lo que compone al PIC16F84 para su funcionamiento.

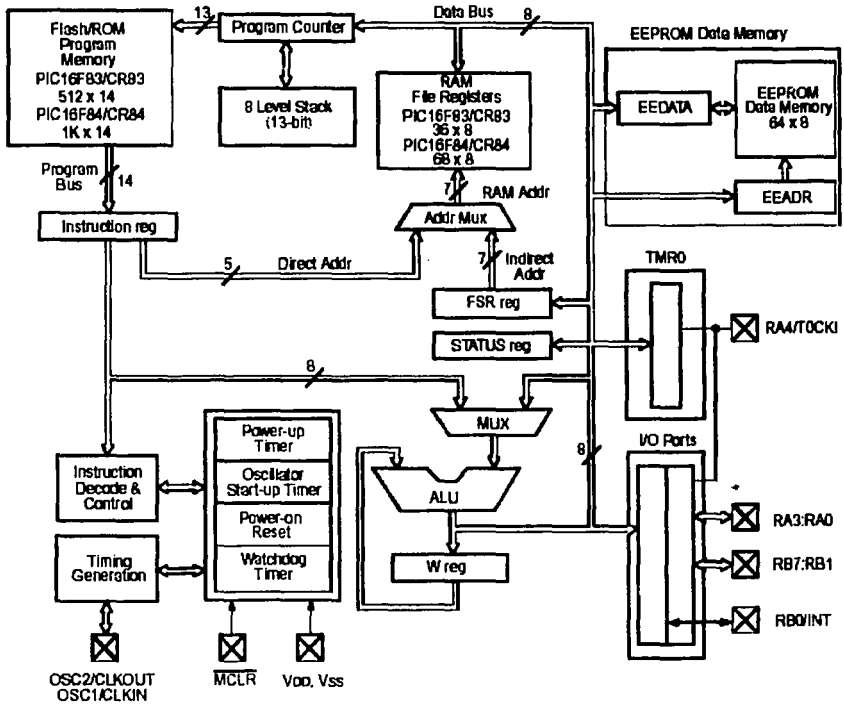



*FIGURA 3.1 DIAGRAMA DE BLOQUES DEL MICROCONTROLADOR PIC16F84*

Se observa en la parte inferior izquierda del diagrama de la figura 3.1 un bloque, externo, que hace referencia a los pines OSC1 /OSC2 en donde se conecta el generador de CLOCK para el sistema. Un poco a la derecha se observa una señal de entrada llamada $\overline{MCLR}$, que es el nombre que se le ha dado al RESET (debido a que ese pin tiene un doble uso). Así mismo, en la parte externa se observan los puertos paralelos de E/s (entrada/salida) llamados puerto A y B[13] y los pines de alimentación.

Sobre el pin de $\overline{MCLR}$ se observa un temporizador de encendido, un temporizador de arranque del oscilador de CLOCK, un circuito de reset y un circuito de vigilancia llamado WATCHDOG. Los dos primeros bloques procuran un arranque ordenado para no producir una carga al mismo tiempo sobre la fuente. La función del WATCHDOG, traducido literalmente como "perro guardián", es la de estar vigilando el máximo de tiempo que tarda el microcontrolador en completar la derivación más larga de su programa, y en caso de superar el tiempo máximo, que es de 3 segundos, provoca un reset automático al considerar que el microcontrolador se quedó trabado en alguna parte de su programa.[14]

Analizando la parte superior izquierda del diagrama de la figura 3.1, se observa la memoria de programa (que para el PIC16F84 es de tipo flash), el contador de programa, el registro de instrucciones y la pila o STACK de 8 niveles. El registro se refiere a pequeñas unidades de memoria transitoria, construida por lo general con un registro de desplazamiento. Son memorias volátiles que se utilizan para guardar información por un tiempo mínimo, con el fin de realizar una operación compleja de varios pasos.[15]

---

[13] Uno de los pines del puerto B puede ser utilizado como entrada de interrupciones. Ver Tema 4.6 referente a las interrupciones.

[14] "Saber electrónica No. 113", Vallejo Horacio D, Televisa S.A. de C.V, 2001, México. Pag 64.

[15] Op cit [14] Pag 64

### 3.1.1. CONTADOR DE PROGRAMA[16]

Este registro, normalmente denominado PC, es equivalente al de todos los microprocesadores y contiene la dirección de la próxima instrucción a ejecutar. Se incrementa automáticamente al ejecutar cada instrucción, de manera que la secuencia natural de ejecución del programa es lineal, una instrucción después de la otra. Algunas instrucciones (llamada de control) cambian el contenido del PC alterando la secuencia lineal de ejecución. Dentro de estas instrucciones se encuentran GOTO y CALL que permiten cargar en forma directa un valor constante en el PC haciendo que el programa salte a cualquier posición de la memoria. Otras instrucciones de control son los SKIP o saltos condicionales, que producen un incremento adicional del PC si se cumple una condición especifica, haciendo que el programa salte sin ejecutar la instrucción siguiente.

El PC es un registro de 13 bits en el PIC16F84, lo que permite direccionar 8.192 posiciones de memoria de programa, pero que internamente solamente podrá direccionar 1.024.

A diferencia de la mayoría de los microprocesadores convencionales, el PC es también accesible al programador como registro de memoria interna de datos, en la posición 02. Es decir que cualquier instrucción común que opere sobre registros puede ser utilizada para alterar el PC y desviar la ejecución del programa.

El contador de programa (PC) es el responsable de que el microprocesador vaya analizando las instrucciones en orden ascendente. Éste guarda el número de instrucción en el STACK y la instrucción misma la pasa al registro de instrucciones desde donde se envía al resto del microcontrolador.

---

[16] http://www.iespana.es/portosin/ (España, 2002)

### 3.1.2. STACK[17]

En los microcontroladores PIC el stack es una memoria interna dedicada, de tamaño limitado, separada de las memorias de datos y de programa, inaccesible al programador, y organizada en forma de pila, que es utilizada solamente, y en forma automática, para guardar las direcciones de retorno de subrutinas e interrupciones. Cada posición es de 11 bits y permite guardar una copia completa del PC. Como en toda memoria tipo pila, a los datos se accede de forma LIFO (Last Input First Output) de manera que el último en entrar es el primero en salir.

El STACK es, en realidad, una pila de registros; que para el caso del microcontrolador PIC16F84 es de 8 niveles, debido a que el programa puede tener derivaciones (LOOPS) o subprogramas. Cuando se termina de ejecutar un loop se puede volver al mismo punto del programa en donde se había producido la bifurcación y eso es posible porque ese número de instrucción quedó guardado en uno de los registros de la pila

El stack y el puntero interno que lo direcciona, son invisibles para el programador, solo se accede a ellos automáticamente para guardar o rescatar las direcciones de programa cuando se ejecutan las instrucciones de llamada o retorno de subrutinas, cuando se produce una interrupción o cuando se ejecuta una instrucción de retorno de ella.

Siguiendo con el diagrama de bloques (figura 3.1), en la parte inferior derecha se ubican los bloques responsables de efectuar operaciones matemáticas y lógicas binarias.

---

[17] Op cit [10]

La ALU (por sus siglas en ingles Aritmetic Logic Unity - unidad aritmética y lógica) es la encargada de las operaciones a partir de un registro en donde se acumula un dato, dicho registro es el registro de trabajo W (Work), y del que se encuentra presente en el instante en el que se invoca la memoria de datos.

Como las operaciones pueden encadenarse, el registro W tiene un retorno a la ALU, como se muestra en diagrama de bloques de la figura 3.1.

Se ve además que la ALU está comandada por un bloque MUX (Multiplexor), ya que la ALU requiere que se le envíen números para procesar que le lleguen desde la memoria de datos, pero antes la debe predisponer para que efectúe la operación requerida. Su salida se encuentra conectada al acumulador ***W***

### 3.1.3. REGISTRO W

En los microcontroladores PIC16F84, la salida de la ***ALU*** va al registro ***W*** y también a la memoria de datos, por lo tanto el resultado puede guardarse en cualquiera de los dos destinos (figura 3.2). En las instrucciones de doble operando, uno de los dos datos siempre debe estar en el registro ***W***, como ocurría en el modelo tradicional con el ***acumulador***. En las instrucciones de simple operando el dato en este caso se toma de la memoria (también por convención).

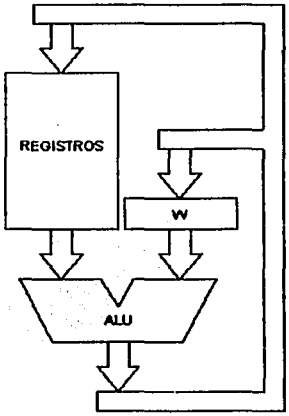



*FIGURA 3.2 REGISTRO W DEL MICROCONTROLADOR*

### 3.1.4. REGISTRO STATUS

El registro de estado o STATUS[18] colabora durante las operaciones matemáticas, pues aunque se trate de una operación entre dos números, su ejecución requiere guardar lo que se llama acarreo en otro registro y este no es otra cosa más que el registro STATUS.

[18] STATUS (ver tema 4.2.1.5) no se refiere a la palabra de configuración del programa para el microcontrolador.

En el diagrama general del microcontrolador PIC16F84 se observa también la existencia de una memoria RAM de registros y la memoria de tipo EEPROM (en la esquina superior derecha) que pueden ser llamadas desde el registro de instrucción de direcciones. Esta última es como una extensión de la memoria RAM solo que los datos no se pierden por falta de energía, como se ha mencionado en capítulos anteriores.

# CAPITULO 4

# MEMORIA DEL PIC16F84

# CAPITULO 4. MEMORIA DEL PIC16F84

La memoria del microcontrolador PIC16F84 se divide en dos partes, cada una con funciones especificas según se detalla a continuación.

## TEMA 4.1. MEMORIA DE DATOS DEL PIC16F84[19]

El microcontrolador **PIC16F84** puede direccionar 128 posiciones diferentes de memoria RAM; pero Microchip Technology solamente ha implementado 80 posiciones para este PIC.

Esta memoria está dividida en dos partes o bancos de registro. Cada banco se divide a su vez en dos áreas. La primera es la de SFR (Special Function Registers) o Registros de Funciones Especiales, que controlan el funcionamiento del dispositivo. Estos se emplean para el control del funcionamiento de la CPU y de los periféricos.

La primera parte consta de 12 registros que serán utilizados por funciones especiales del microcontrolador. Comienza en la dirección 00h y termina en la 0Bh para el banco 0.

La segunda área (68 bytes SRAM) es la de GPR (General Purpose Register) o Registros de Propósito General, y puede accederse a ellos tanto directa como indirectamente haciendo uso del registro FSR (File Select Register). Estos registros de memoria serán utilizados para almacenar datos temporales requeridos por los programas. Comienza en la dirección 0Ch y termina en la posición 4Fh

Para cambiar de página o banco, se utiliza 1 bit del registro STATUS (RP0 – bit de selección de banco de registros).

---

[19] http://www.iespana.es/portosin/ (España, 2002)

Para direccionar la memoria de datos se emplean dos modos de direccionamiento, el directo y el indirecto. En el direccionamiento directo, los 7 bits de menos peso del OPCODE (código de operación) de la instrucción proporcionan la dirección en la posición de la página, mientras que los bits RP1 y RP0 del registro STATUS seleccionan la página o banco.

En el direccionamiento indirecto el operando de la instrucción hace referencia al registro INDF (direccionamiento indirecto), que ocupa la posición 00h del área de datos. Se accede a la posición que apunta el registro FSR. Los 7 bits de menos peso de FSR seleccionan la posición y su bit de más peso selecciona la página.

La memoria **RAM** así como algunos registros especiales son los mismos en los dos bancos del mapa de memoria del PIC.

Este tipo de memoria **(RAM),** se caracteriza por perder los datos si se llegase a desconectar el microcontrolador o la tensión cae por debajo de los límites mínimos. Se genera entonces un tipo de reset para los registros especiales, las otras formas de reset se mencionan a continuación

### 4.1.1. TIPOS DE RESET

El ***PIC 16F84*** contiene diferentes tipos de RESET como se ve a continuación:

- **RESET** de Encendido ( Power On Reset "***POR***")
- **RESET** durante la operación normal utilizando el $\overline{\text{MCLR}}$
- **RESET** durante la instrucción ***SLEEP*** utilizando el $\overline{\text{MCLR}}$
- **RESET** durante la operación normal utilizando ***WDT*** (WatchDog Timer: Temporizador que resetea el sistema cuando se desborda su valor)
- **RESET** durante la instrucción ***SLEEP*** utilizando el ***WDT.***

El tipo de ***RESET*** con $\overline{\textbf{MCLR}}$ tiene filtros de ruido para ignorar los pequeños pulsos, con especificaciones eléctricas que requiere el pin de $\overline{\textbf{MCLR}}$.

Algunos registros no son afectados en ninguna de las condiciones de ***RESET***, así que su estado o "***STATUS***" es desconocido en el caso de un ***RESET*** del tipo POR (Power on Reset), y para otros tipos de ***RESET*** se manejará sin cambios. Pero la mayoría de los registros se encuentran como en un estado de ***RESET*** para el caso de un ***POR***, $\overline{\textbf{MCLR}}$ o ***WDT*** durante una operación normal y en el caso de un reset usando el ***WDT*** durante la instrucción ***SLEEP***, no serán afectados ya que este tipo de RESET es parte de la operación normal

Las condiciones de ***RESET*** se muestran en cada registro especial, y en la tabla 15 (página 62) se podrán analizar las condiciones de ***RESET*** en conjunto para cada registro.

## TEMA 4.2. ORGANIZACIÓN DE LA MEMORIA

La memoria interna de datos, también llamada archivo de registros (register file), está dividida en dos grupos por bloque según se muestra en la figura 4.1.

Los registros especiales que son 8, con algunas diferencias por banco, y se muestran dentro de la línea continua en la figura 4.1.

Los registros de propósito generales son 72 registros y se observan dentro de la línea punteada en la figura 4.1.

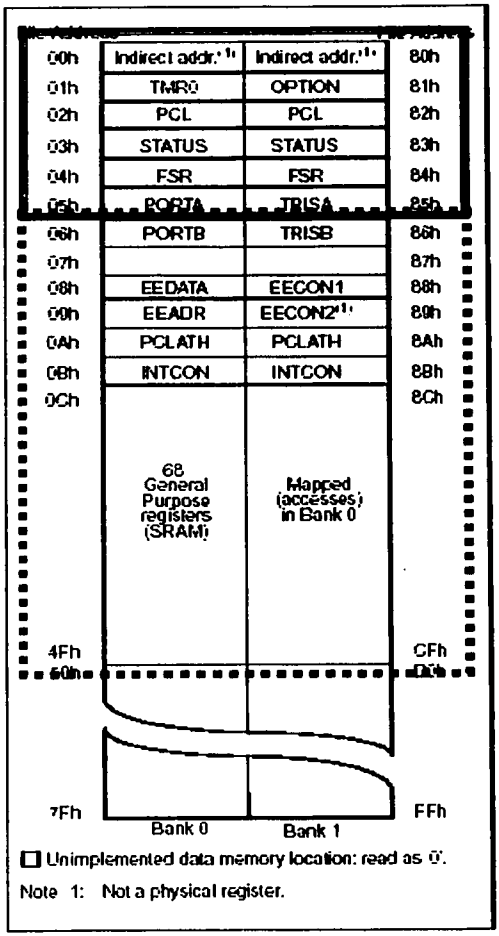



*FIGURA 4.1 ORGANIZACIÓN DE LA MEMORIA DE DATOS*

Los registros de propósito general se dividen a su vez en dos grupos:

- Los registros de posición fija (8 registros en total).
- Los bancos de registros (64 registros).

Los primeros ocupan las 8 posiciones que van de la 08 a la 0B en el banco 0 y de la 88 a la 8C en el banco 1 (figura 4.1).

Los bancos de registros consisten en dos grupos o bancos que se encuentran superpuestos en las direcciones que van de la 0C a la 4F en el banco 0 y desde la 8C a la CF en el banco 1. Se puede operar con un solo banco a la vez, los cuales están mapeados en el área de registros de propósito general (figura 4.1).

### 4.2.1. REGISTROS ESPECIALES

Los registros de posición fija ocupan las 12 primeras posiciones que van desde la 00 a la 07 (figura 4.1), y de la 08 a la 0B.

Entre los registros especiales se encuentra la palabra de estado (STATUS), los registros de datos de los puertos de entrada salida (Puerto A, Puerto B), los 8 bits menos significativos del Program Counter (PC) y un registro puntero llamado File Select Register (FSR).

La posición 00 no contiene ningún registro en especial y es utilizada en el mecanismo de direccionamiento indirecto.

A continuación se da una descripción más completa de cada uno de éstos registros.

#### *4.2.1.1. EL REGISTRO INDF*

El registro INDF (Indirect Address) que ocupa la posición 0 no está implementando físicamente. Dirigirse a INDF en realidad produce la dirección del registro que está contenido en el registro FSR.

Las condiciones de reset para el registro INDF son muy simples ya que siempre se pondrán los bits a 0, en la tabla 5 se aprecia lo anterior:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| INDF | 00h | ---- ---- | ---- ---- | ---- ---- |
| Leyendas: u=sincambios x=desconocido - = bit leído como cero q= el valor depende de la condición | | | | |

*TABLA 5 BITS DEL REGISTRO INDF DESPUÉS DE UN RESET*[20]

#### *4.2.1.2. REGISTRO CONTADOR TMR0*

En la dirección 01H esta el registro TMR0 (Temporizador) que puede ser leído y escrito como cualquier otro registro.

El registro TMR0 es un particular tipo de registro cuyo contenido es incrementado con una cadencia regular y programable directamente por el software del microcontrolador PIC.

Este registro puede usarse para contar eventos externos por medio de un pin de entrada especial (modo contador) o para contar pulsos internos de reloj de frecuencia constante (modo timer). Además, en cualquiera de los dos modos, se puede insertar un preescaler, es decir un divisor de frecuencia programable que puede dividir por 2, 4, 8, 16, 32, 64, 128 o 256. Este divisor puede ser utilizado alternativamente como preescaler del TMR0 o del Watch Dog Timer, según se lo programe.

En la práctica, a diferencia de los otros registros, el TMR0 no mantiene inalterado el valor que tiene memorizado, sino que lo incrementa continuamente.

---

[20] "Microchip PIC Microcontrollers Data Book", Microchip Technology Inc., Microchip, 1997. U.S.A. Pag 43

Si por ejemplo escribe en él el valor 10, después de un tiempo igual a cuatro ciclos de máquina, el contenido del registro comienza a ser incrementado a 11, 12, 13 y así sucesivamente con una cadencia constante y totalmente independiente de la ejecución del resto del programa.

Una vez alcanzado el valor 255, el registro TMR0 es puesto a cero automáticamente comenzando entonces a contar desde cero y no desde el valor originalmente cargado.

La frecuencia de conteo es directamente proporcional a la frecuencia de reloj aplicada al chip y puede ser modificada programando adecuadamente algunos bits de configuración.

En la figura 4.2 está representada la cadena de bloques internos del PIC que determinan el funcionamiento del registro TMR0.

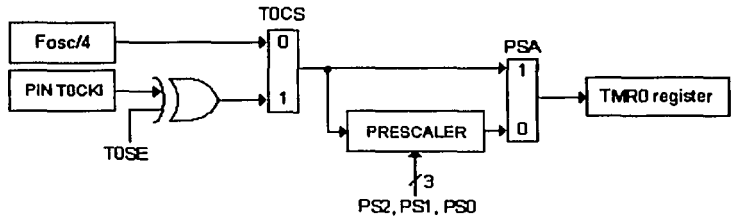



*FIGURA 4.2 BLOQUES DE FUNCIONAMIENTO DEL REGISTRO TMR0*

Los bloques **Fosc/4** y **T0CKI** representan las dos posibles fuentes de señal de reloj, para el contador **TMR0**. **T0CKI** es una señal generada por un posible circuito externo y aplicada al pin **T0CKI** correspondiente al pin 3 del PIC16F84.

**Fosc/4** es una señal generada internamente por el PIC tomada del circuito de reloj y que es igual a la frecuencia del oscilador dividida por cuatro.

Los bloques **T0CS** y **PSA** son conmutadores de señal en cuya salida se presenta una de las dos señales de entrada en función del valor de los bits **T0CS** y **PSA** del registro **OPTION**.

Las condiciones de reset para el registro TMR0, se muestran en la tabla 6:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| TMR0 | 01h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| Leyendas: u=sincambios x=desconocido - = bit leído como cero q= el valor depende de la condición | | | | |

*TABLA 6 BITS DEL REGISTRO TMR0 DESPUÉS DE UN RESET*[21]

El bloque PRESCALER es un divisor programable. En el capítulo 4.2.1.11 se muestra como dividir la frecuencia de conteo, interna ó externa, activando el PRESCALER.

### *4.2.1.3. REGISTRO OPTION_REG*

Este es un registro que se puede leer y en el que se puede escribir, el cual contiene varios bits de control con los que se pueden configurar el preescaler del TMR0 (Timer 0: Contador en Tiempo Real de 8 Bits) o del WDT, la interrupción externa por el pin INT, la del TMR0 y el resistor de polarización interno del puerto B ( Weak Pull-Up ).

La dirección de memoria en la que se encuentra este registro es la 81H, esta se localiza en el banco de memoria 1, por lo que se hace necesario el cambio de banco mediante el bit 5 (RP0) del registro STATUS. En la figura 4.3 se muestra la ubicación y nombre de cada uno de los bits del registro OPTION_REG.

| R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |
|---|---|---|---|---|---|---|---|
| RBPU | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |
| bit7 | | | | | | | bit0 |

*FIGURA 4.3 NOMBRE DE LOS BITS DEL REGISTRO OPTION_REG*

[21] Op cit [20] Pag 43

Las letras que se encuentra encima de los nombres de cada bit se refieren a que: **R.**- es un bit que se puede leer y **W.**- es un bit que se puede escribir.

El número que se encuentra junto a las letras, indica el valor que toma el bit cuando ocurre un reset al microcontrolador.[22]

El Bit 7, denominado $\overline{\text{RBPU}}$ (PORTB Pull-Up Enable Bit), es el bit que habilita el pull-up del PORTB. Si el bit tienen un valor de 1 los pull-ups del puerto son habilitados, y si vale 0 serán deshabilitados.

El Bit 6 (INTEDG: Interruption Edge), cuando vale 1 habilita la interrupción por el pin RB0/INT cuando el flanco de la señal en él es de subida, es decir, el cambio en la señal del pin es de bajo a alto. En tanto que cuando el bit vale 0 el flanco es de bajada en el pin, es decir, la señal de entrada cambia de alto a bajo (1 a 0 lógico)

EL bit 5 de este registro es el TOCS, este el bit que le indica al microcontrolador que reloj debe usar para que funcione el registro TMR0, si debe usar un reloj o señal conectada en el pin RA4/TOCKI su valor debe ser 1, pero si debe usar el reloj interno de ciclos de instrucción su valor deberá ser 0.

Ahora se le debe hacer saber al microcontrolador, en que momento debe de realizar el conteo en el registro TMR0 para una señal externa (en el pin RA4), si durante la transición de alto a bajo o de bajo a alto. Esto se hace mediante el Bit 4 del este registro (TOSE), el cual si vale 1 realiza el conteo en el flanco de bajada de la señal externa, mientras que si vale 0 lo realiza durante el flanco de subida.

---

[22] Podrá ver un resumen de las condiciones de reset para los registros especiales en el tema 4.2.2

El Bit 3 sirve para asignar el preescaler[23], ya sea al WTD que para este caso debe asignarse el valor 1 al bit PSA, o al TMR0 asignándole el valor 0 al bit. Se debe aclarar que solo se le puede asignar el preescaler a uno de los dos registros mencionados.

Los Bits del 0 al 2 (PS0, PS1 y PS2) son los que designan el valor con el que el registro correspondiente se va a incrementar (según la combinación de estos bits), es decir, no siempre serán los incrementos de uno en uno para el TMR0 o el WTD. Esto se ve en el capítulo correspondiente a cada uno de estos registros.

Las condiciones de Reset para el registro Option_Reg se encuentran en la tabla 7

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| OPTION_REG | 81h | 1111 1111 | 1111 1111 | uuuu uuuu |

*TABLA 7 BITS DEL REGISTRO OPTION_REG DESPUÉS DE UN RESET*[24]

### *4.2.1.4. EL CONTADOR DE PROGRAMA PC*

El contador de programa PC del microcontrolador PIC16F84 ocupa la posición 2 del área de datos en donde se halla el registro PCL (byte bajo del PC) junto con el **PCLATH** (byte alto del PC) en la dirección **0A**. El primero es legible y escribible mientras que el segundo no lo es en forma directa.

---

[23] El preescaler sirve para poder realizar variaciones en el conteo del TMR0 o del WTD . Los detalles del mismo se dan a conocer en los capítulos correspondientes a cada uno de los registros mencionados (tema 4.2.1.2 y Tema 4.4), pues para cada uno es diferente el funcionamiento del preescaler.

[24] Op cit [20] Pag 43

El PC tiene una longitud de 13 Bits, considerando que se añaden 3 bits auxiliares y se conectan con los dos niveles de la Pila en las instrucciones CALL y RETLW, por lo que implica que podrá direccionar cualquier posición comprendida en un rango de los 8 K x 14; pero la empresa Microchip Technology solamente ha implementado internamente 1 K que representa desde dirección 000h hasta la 03FFh. En el caso de que se direccione fuera de este rango, automáticamente causará un solapamiento.

La condición de reset del PC se muestra en la tabla 8, en los diferentes tipos de reset.

| CONDICION | Registro STATUS |
|---|---|
| Encendido RESET | 000h |
| RESET Durante operación normal usando MCLR' | 000h |
| RESET Durante la instrucción SLEEP | 000h |
| RESET Durante la operación normal usando WDT | 000h |
| RESET Despertando usando el WDT | PC + 1 |
| Interrupción de la instrucción SLEEP | PC + 1 |

*TABLA 8 BITS DEL REGISTRO STATUS DESPUÉS DE UN RESET*[25]

### *4.2.1.5. PALABRA DE ESTADO DEL PROCESADOR ( REGISTRO STATUS)*

El registro de Estado (STATUS) ocupa la posición 3 y entre sus bits se encuentran los señalizadores C, DC y Z, que muestran el estado de la ALU, y el bit RP0 que selecciona la página en la memoria de programa.

Los bits del registro STATUS son los que se muestran a continuación:

| No. BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Nombre | IRP | RP1 | RP0 | T0 | PD | Z | DC | C |

Los bits IRP y RP1 (Bits de selección de banco de registros) no son usados para el PIC 16F84. Deben mantenerse en cero.

---

[25] Op cit [20] Pag 43

El bit de selección de banco de registros RP0 se utiliza en las instrucciones de salto GOTO y CALL. Manipulando el bit número 5 (RP0) del registro STATUS podrá indicar al microcontrolador si desea trabajar en el banco "0" o en el "1".

La operación normal del microcontrolador se efectúa en el banco "0". Pero cuando se cambia del banco "0" al banco "1" es para efectuar ciertos cambios que definen como estarán configurados los puertos del microcontrolador.

El bit TO (TIME-OUT) sirve para detectar si una condición de reset fue producida por el watch dog timer, está relacionado con los mismos elementos que el bit siguiente.

El bit PD (POWER DOWN)[26] sirve para detectar si la alimentación fue apagada y encendida nuevamente, tiene que ver con la secuencia de inicialización, el watch dog timer y la instrucción sleep.

El bit Z indica si el resultado de la última operación fue CERO o no.

El bit DC (digit carry) indica acarreo del cuarto bit (bit 3) del resultado de la última operación de suma o resta, con un comportamiento análogo al del bit C, y es útil para operar en BCD (para sumar o restar números en código BCD).

El bit C indica acarreo del bit más significativo (bit 7) del resultado de la última operación de suma. En el caso de la resta se comporta a la inversa, C resulta 1 si no hubo pedido de préstamo. Es usado además en las operaciones de rotación derecha o izquierda como un paso intermedio entre el bit 0 y el bit 7.

---

[20] Ver tema 4.2.1.6

Las condiciones de reset para el registro de estado (STATUS) son distintas en las diferentes modalidades de reset que ya se mencionaron anteriormente[27], en esta sección la tabla 9 muestra las condiciones distintas para los bits de este registro.

| CONDICION | Registro STATUS |
|---|---|
| Encendido RESET | 0001 1 xxx |
| RESET Durante operación normal usando MCLR' | 000u uuuu |
| RESET Durante la instrucción SLEEP | 0001 0uuu |
| RESET Durante la operación normal usando WDT | 0000 1uuu |
| RESET Despertando usando el WDT | uuu0 0uuu |
| Interrupción de la instrucción SLEEP | uuu1 0uuu |
| Leyendas u= sin cambios x=desconocido | |

*TABLA 9 BITS DEL REGISTRO STATUS DESPUÉS DE UN RESET*[28]

### *4.2.1.6. FUNCIONAMIENTO DEL POWER DOWN MODE*

El ***Power Down Mode*** o ***Sleep Mode***, es un estado particular de funcionamiento del PIC, utilizado para reducir el consumo de corriente en los momentos que no realiza ninguna tarea o está a la espera de un suceso externo.

Si toma como ejemplo un control remoto para TV, verá que la mayor parte del tiempo el micro permanece a la espera de la presión de alguna tecla. Apenas oprimida, efectúa una breve transmisión y queda nuevamente a la espera de la presión de otra tecla.

El tiempo de uso efectivo de la CPU del microcontrolador está por tanto, limitado a unos pocos milisegundos necesarios para efectuar la transmisión mientras que durante varias horas no efectúa ninguna tarea particular

---

[27] Vease la sección de Condiciones de RESET en el tema 4.2.2

[28] Op cit [20] Pag 43

Para no consumir inútilmente la energía de las baterías, es posible apagar varios de los circuitos del microcontrolador y reencenderlos sólo en correspondencia con algún suceso externo.

La instrucción ***SLEEP*** es utilizada para colocar al PIC en Power Down Mode y reducir el consumo de corriente, que pasará de unos 2 mA (a 5 volts y el clock en 4MHz) a unos 2uA, o sea, unas 1000 veces menos.

Para entrar en Power Down Mode basta insertar la instrucción SLEEP en cualquier parte del programa.

Cualquier instrucción siguiente a ***SLEEP*** no será efectuada por el microcontrolador PIC16F84 el cual finalizará en este punto la ejecución, apagará los circuitos internos, excepto aquellos necesarios para mantener el estado de los puertos de I/O y aquellos que lo sacarán de esa condición.

Para despertar al PIC se utilizan diversas técnicas:

- Reset del PIC llevando a cero el pin 4 ($\overline{MCLR}$).
- Timeout del Watch Dog Timer (si está habilitado[29]).
- Verificación de una interrupción (interrupción desde el pin RB0/INT, cambio de estado en el puerto B, finalización de la escritura sobre la EEPROM).

En los dos primeros casos, el microcontrolador PIC es reseteado y la ejecución es retomada en la dirección 0 de memoria.

[29] Ver el Tema 5.6 referente a la palabra de configuración del microcontrolador

En el tercer caso, el PIC se comporta como en el caso de una interrupción normal, siguiendo primeramente el Interrupt handler, retomando la ejecución después de la instrucción SLEEP. Para que el PIC sea despertado por una interrupción deben ser habilitados los indicadores o bits de control (flags) del registro ***INTCON.***[30]

#### *4.2.1.7. EL REGISTRO FSR*

El registro FSR se ubica en la dirección 4 y en la 84 a la vez, puede usarse para contener la dirección del dato en las instrucciones con direccionamiento indirecto y también para guardar operandos en sus 5 bits de menos peso.

El reset en el registro FSR es similar a las condiciones que se presentan del registro TMR0, esta direccionado en dos localidades distintas según se ve en la tabla 10:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| FSR | 04h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| FSR | 84h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| Leyendas: u=sincambios x=desconocido q= el valor depende de la condición | | | | |

*TABLA 10 BITS DEL REGISTRO FSR DESPUÉS DE UN RESET*[31]

#### *4.2.1.8. REGISTROS PARA LOS PUERTOS A Y B*

Los registros que ocupan las posiciones 5 y 6 soportan los Puertos A y B de E/S. Pueden ser leídos y escritos como cualquier otro registro y manejan los valores de los bits que entran y salen por los pines de E/S del microcontrolador.

---

[30] Ver tema 4.2.1.10 sobre el registro INTCON.

[31] Op cit [20] Pag 43

Las condiciones de Reset para los registros de los puertos A y B son distintos que los de TRIS A y B, se muestran en la tabla 11:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| PORTA | 05h | ---x xxxx | ---u uuuu | ---u uuuu |
| PORTB | 06h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| Leyendas u=sincambios x=desconocido - = bit leído como cero q= el valor depende de la condición | | | | |

*TABLA 11 BITS DE LOS REGISTROS PORTA Y PORTB DESPUÉS DE UN RESET*[32]

### *4.2.1.9. REGISTROS TRIS A Y TRIS B*

Observando de nuevo la figura 4.1 se ve que existen algunas diferencias entre el banco "0" y el banco "1". Los registros denominados OPTION, TRISA, TRISB, EECON1 y EECON2 no existen en el banco "0".

Los registros TRIS A y TRIS B, en las direcciones 85 y 86 respectivamente, sirven para programar a los puertos ya sea como puertos de entrada o como puertos de salida.

Si se necesita acceder al registro TRISA, que solamente se encuentra en el banco "1", obligatoriamente habrá que cambiarse del banco "0" al banco "1" por medio de las instrucciones del microcontrolador (registro Status).

Se debe acceder al banco "1" solamente para utilizar los registros que no se encuentran en el banco "0". Una vez utilizados esos registros (y modificados si fuese necesario), regrese al banco "0" para que el microcontrolador siga con su tarea asignada en la memoria del programa.

---

[32] Op cit [20] Pag 43

En el caso de los registros OPTION y TRISA y B no será necesario realizar el cambio de banco, ya que hay dos instrucciones que se pueden utilizar para hacerlo, a pesar de que Microchip recomienda no usarlas para mantener la compatibilidad con el juego de instrucciones del microcontrolador 16CXX, nada impide que se haga uso de ellas.

Las condiciones de Reset para los registros Tris A y B son distintos a las de los puertos como se muestra en la tabla 12:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| TRISA | 85h | ---1 1111 | ---1 1111 | ---u uuuu |
| TRISB | 86h | 1111 1111 | 1111 1111 | uuuu uuuu |
| Leyendas<br>u=sincambios x=desconocido - = bit leído como cero q= el valor depende de la condición | | | | |

*TABLA 12 BITS DE LOS REGISTROS TRISA Y TRISB DESPUÉS DE UN RESET*[33]

### *4.2.1.10. REGISTRO INTCON*

El registro INTCON (del inglés Interruption Control) se encuentra en la dirección 0Bh del banco cero y está mapeado en la 8Bh del banco 1, es un registro que se puede leer y escribir y contiene varios bits para habilitar los recursos de interrupciones para el PIC16F84.

La ubicación de cada bit se muestra en la figura 4.4:

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-x |
|---|---|---|---|---|---|---|---|
| GIE | EEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |
| bit7 | | | | | | | bit0 |

*FIGURA 4.4 NOMBRE DE LOS BITS DEL REGISTRO INTCON*

[33] Op cit [20] Pag 43

El bit 7 GIE (Global Interrupt Enable Bit: Bit de Habilitación de Interrupción Global) sirve para habilitar todas las interrupciones disponibles cuando su valor es uno y las deshabilita cuando vale 0.

El bit 6 EEIE (EE Write Complete Interrupt Enable Bit: Bit de Habilitación de Interrupción de Escritura Completa en la EEPROM) habilita la interrupción, si vale 1, para cuando la escritura en la memoria EEPROM se completa, y si vale 0 dicha interrupción es deshabilitada.

EL bit 5 TOIE (TMR0 Overflow Interrupt Enable Bit: Bit de Habilitación de Interrupción por Desbordamiento del TMR0) habilita la interrupción, cuando vale 1, para el caso en el que el TMR0 ha llegado a su valor máximo (255) y comienza conteo desde cero. Se deshabilita la interrupción si se le signa el valor 0.

EL bit 4 INTE ( RB0/INT Interrupt Enable Bit: Bit de Habilitación de Interrupción en el Pin RB0/INT) habilita, con el valor 1, la interrupción por el cambio en el flanco de la señal del pin RB0/INT.

El Bit 3 RBIE (RB Port Change Interrupt Enable Bit: Bit de Habilitación de Interrupción por Cambio en el Puerto B) cuando vale 1 habilita la interrupción para el caso de que exista un cambio en los bits del 4 al 7 del puerto B.

El bit 2 TOIF (TMR0 Overflow Interrupt Flag Bit: Bit Indicador de Interrupción por Desbordamiento del TMR0) es un indicador (flag), para cuando el TMR0 ha llegado a su limite máximo o no, uno y cero respectivamente. Es el mismo caso del bit 1 INTF (RB0/INT Interrupt Flag Bit: Bit Indicador de Interrupción por el Bit RB0/INT), y el bit 0 RBIF(RB Port Change Interrupt Flag Bit: Bit Indicador de Interrupción por Cambio en el puerto B) indica el cambio en uno de los pines superiores del puerto B, entonces el bit RBIF valdrá 1.

Para el registro Intcon las condiciones de Reset son las mostradas en la tabla 13:

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| INTCON | 0Bh | 0000 000x | 0000 000u | uuuu uuuu |
| INTCON | 8Bh | 0000 000x | 0000 000u | uuuu uuuu |

*TABLA 13 BITS DEL REGISTRO INTCON DESPUÉS DE UN RESET*[34]

### *4.2.1.11. EL PREESCALER*

El **PREESCALER** consiste en un divisor programable de 8 bits a utilizar en el caso de que la frecuencia de conteo enviada al contador TMR0 sea demasiado elevada para sus propósitos. Se configura a través de los bits PS0, PS1 y PS2 del registro OPTION.

La frecuencia Fosc/4 es una cuarta parte de la frecuencia de reloj. Utilizando un cristal de 4Mhz tendrá una Fosc/4 igual a **1 MHz**. Tal frecuencia es enviada directamente al registro TMR0 sin sufrir ningún cambio. La cadencia de conteo que se obtiene es por lo tanto igual a 1 millón de incrementos por segundo del valor presente en TMR0, que para muchas aplicaciones podría resultar demasiado elevada.

Con el uso del PREESCALER puede dividir posteriormente la frecuencia Fosc/4 configurando oportunamente los bits **PS0**, **PS1** y **PS2** del registro **OPTION** según la tabla 14

[34] Op cit [20] Pag 43

| PS2 | PS1 | PS0 | DIVISOR FREC. SALIDA | PREESCALER (KHz.) |
|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 500 |
| 0 | 0 | 1 | 4 | 250 |
| 0 | 1 | 0 | 8 | 125 |
| 0 | 1 | 1 | 16 | 62 |
| 1 | 0 | 0 | 32 | 31.250 |
| 1 | 0 | 1 | 64 | 15.625 |
| 1 | 1 | 0 | 128 | 7.813 |
| 1 | 1 | 1 | 256 | 3.906 |

*TABLA 14 COMBINACIÓN DE LOS BITS DEL PREESCALER*

### 4.2.2. RESUMEN DE CONDICIONES DE RESET PARA LOS REGISTROS

Esta sección está enfocada a las condiciones de Reset de los registros antes vistos y los que no se nombraron con anterioridad, como son el caso de los registros EECON 1 y 2, en esta pequeña sección están sus condiciones de Reset, también los registros EEDATA y EEADR, que en pocas palabras forman parte complementaria de la memoria EEPROM.

A continuación, a modo de resumen, se muestra la tabla 15 con las condiciones de reset para los registros, así como su dirección dentro de la memoria.

| REGISTRO | DIRECCION | ENCENDIDO RESET (POR) | RESET usando MCLR -Operación Normal -Instrucción SLEEP RESET usando WDT -Operación Normal | Despierta de Instrucción SLEEP -Por medio de Interrupción -Usando el WDT Time Out |
|---|---|---|---|---|
| W | —— | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| INDF | 00h | ---- ---- | ---- ---- | ---- ---- |
| TMR0 | 01h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PCL | 02h | 0000h | 0000h | pc + 1 |
| STATUS | 03h | 0001 1xxx | 000q quuu | uuuq quuu |
| FSR | 04h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PORTA | 05h | ---x xxxx | ---u uuuu | ---u uuuu |
| PORTB | 06h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| EEDATA | 08h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| EEADR | 09h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PCLATH | 0Ah | ---0 0000 | ---0 0000 | ---u uuuu |
| INTCON | 0Bh | 0000 000x | 0000 000u | uuuu uuuu |
| INDF | 80h | ---- ---- | ---- ---- | ---- ---- |
| OPTION_REG | 81h | 1111 1111 | 1111 1111 | 1111 1111 |
| PCL | 82h | 0000h | 0000h | pc + 1 |
| STATUS | 83h | 0001 1xxx | 000q quuu | uuuq quuu |
| FSR | 84h | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TRISA | 85h | ---1 1111 | ---1 1111 | ---u uuuu |
| TRISB | 86h | 1111 1111 | 1111 1111 | uuuu uuuu |
| EECON1 | 88h | ---0 x000 | ---q q000 | ---0 uuuu |
| EECON2 | 89h | ---- ---- | ---- ---- | ---- ---- |
| PCLATH | 8AH | ---0 0000 | ---0 0000 | ---u uuuu |
| INTCON | 8Bh | 0000 000x | 0000 000u | uuuu uuuu |

Leyendas: u=sincambios x=desconocido - = bit leído como cero q= el valor depende de la condición

*TABLA 15 CONDICIONES DE RESET PARA LOS REGISTROS ESPECIALES*[35]

[35] Op cit [20] Pag 43.

## TEMA 4.3. LA MEMORIA EEPROM

Hasta el momento no se han mencionado cuatro registros especiales que son EECON1, EECON2, EEDATA Y EEADR, debido a que estos registros se utilizan para leer y/o escribir en la memoria EEPROM. Esta memoria se considera como una extensión de la memoria de datos, solo que no esta direccionada directamente, sino que se direcciona a través del registro EEADR.

EEDATA contiene los datos de 8 bits para leer y/o escribir, y EEADR contiene la dirección de las localidades de la EEPROM que son accedidas. El PIC16F84 contiene 64 bytes de datos en la EEPROM, iniciando en la 00h y finalizando en la 3Fh.

El tiempo de escritura puede variar según la tensión, la temperatura, etc., Así mismo, según la versión del circuito ya que cada uno contiene un control por medio de un oscilador incorporado en cada chip.

El registro EECON1 se encuentra en la dirección 88h. Los bits del 5 al 7 no están implementados y se leen como cero, el bit 4 EEIF(EEPROM Write Operation Interrupt Flag Bit) es un indicador de fin de escritura en la EEPROM y si se encuentra a 0 significa que no se cumplió con la escritura o esta no ha comenzado.

El bit 3 WRERR (EEPROM Error Flag Bit) es un indicador de error, si vale 1 significa que la escritura en la EEPROM terminó en forma prematura (por un reset $\overline{\text{MCLR}}$ o del WDT).

Al poner el bit 2 WREN (EEPROM Write Enable Bit) a 1 se permite activar la escritura, mientras que el bit 1 WR (Write Control Bit) se debe poner en estado alto para iniciar el ciclo de la misma, este último bit se pone automáticamente a cero mediante el hardware del propio PIC, de tal forma que solo se puede poner a 1 por software.

Por último el bit 0 RD (Read Control Bit) es el que permite se realice la lectura de los datos en la EEPROM al ponerlo a 1 y, al igual que el anterior, se pone a cero por hardware.

### 4.3.1. ESCRITURA EN LA EEPROM[36]

Para escribir en una localidad de la memoria de datos EEPROM, se debe escribir la dirección primero en el registro EEADR y los datos en el registro EEDATA. Después se debe seguir una secuencia específica para iniciar la escritura de cada byte. Dicha secuencia es especificada por el fabricante y es la siguiente:

```
BCF     STATUS,RPO      ;PASA AL BANCO 0
MOVLW   H'18'           ;CUALQUIER DIRECCIÓN
MOVWF   EEADR           ;SE PONE LA DIRECCIÓN DESEADA
MOVLW   H'12'           ;CUALQUIER DATO
MOVWF   EEDATA          ;DATO A GRABAR EN LA EEPROM
BSF     STATUS,RPO      ;BANCO1
BSF     EECON1,WREN     ;SE SOLICITA PERMISO DE ESCRITURA
BCF     INTCON,GIE      ;SE RECOMIENDA PARA DESHABILITAR
                        ;LAS INTERRUPCIONES
;AHORA SE PONE LA SECUENCIA REQUERIDA POR EL FABRICANTE
MOVLW   H'55'
MOVWF   EECON2
MOVLW   H'AA'
MOVWF   EECON2
BSF     EECON1,WR       ;COMIENZA LA ESCRITURA
```

[36] "Microcontroladores PIC", Vallejo Horacio D. Quark S.R.L., 2002, Argentina, Pag 39.

```
WI      BTFSC   EECON1,WR
        GOTO    WI              ;ESPERA A QUE TERMINE LA ESCRITURA
        BCF  STATUS,RPO         ;SALE DEL BANCO 1
```

Se debe respetar la secuencia mencionada, sino, no es posible lograr la escritura del dato. Además, gracias al bit WREN se impide la escritura accidental a la EEPROM, la limpieza de este bit no afecta a la secuencia de escritura y se deberá de mantener limpio en todo momento, salvo al momento de escribir en la EEPROM.

### 4.3.2. LECTURA DE LOS DATOS DE LA EEPROM

A diferencia de la escritura en la EEPROM, la lectura es más fácil. Solo es necesario, desde el banco 0, cargar la dirección que se desea leer en el registro EEADR, posteriormente en el banco uno se pone a 1 el bit RD de EECON1 regresando al banco cero. De esta forma se tiene el dato, de la dirección solicitada con EEADR, en el registro EEDATA desde el cual se puede mover dicho dato.

Cabe mencionar que el registro EECON2 no está implementado físicamente, de tal forma que solo se usa para la rutina de escritura.

## TEMA 4.4. EL WATCH DOG TIMER (WDT)[37]

El Watch Dog Timer (que podría traducirse como temporizador perro guardián) es un oscilador interno del microcontrolador PIC, pero completamente independiente del resto de la circuitería, cuya función es eliminar eventuales bloqueos de la CPU del PIC16F84 y resetearlo para que retome la ejecución normal del programa.

---

[37]Op cit ([19])

Para poder eliminar un eventual bloqueo de la CPU durante la ejecución del programa principal, se inserta en él una instrucción especial:

"CLRWDT (Clear Watch Dog Timer)"

Esta instrucción pone a cero en intervalos regulares el WDT, no permitiéndole llegar al final de su temporización. Si la CPU no realiza esta instrucción antes del término de la temporización, entonces se asume que el programa se ha bloqueado por algún motivo y se efectúa el reset de la CPU.

El periodo mínimo alcanzado en el cual la CPU es reseteada es de unos ***18 ms*** (depende de la temperatura y de la tensión de alimentación). Es posible, sin embargo, asignar el preescaler al WDT para de obtener retardos mayores (hasta unos *2 o 3 segundos*).

Para habilitar el WDT debe, en el programa, habilitar el bit ***WDTE*** de la palabra de configuración[38].

#### *4.4.1.1. ASIGNACIÓN DEL PRESCALER AL WDT*[39]

Actuando sobre el bit ***PSA*** del registro ***OPTION_REG*** es posible asignar el preescaler al WDT para obtener tiempos de intervención mayores. El bit PSA va puesto a uno.

En caso contrario, el preescaler será asignado al ***TIMER 0***. Obviamente, asignando el preescaler al WDT, no será posible usarlo con el TIMER 0 y viceversa.

Según los valores de los bits ***PS0, PS1 y PS2*** del ***OPTION_REG*** podrá obtener distintos intervalos de retardo.

---

[38] Ver Tema 5.6 correspondiente a la palabra de configuración.

[39] Op cit [19]

En la tabla 16 se ven los retardos, según los valores de PS0, PS1 y PS2:

| PS2 | PS1 | PS0 | DIVISOR | Retardo (ms) |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 18 |
| 0 | 0 | 1 | 2 | 36 |
| 0 | 1 | 0 | 4 | 72 |
| 0 | 1 | 1 | 8 | 144 |
| 1 | 0 | 0 | 16 | 288 |
| 1 | 0 | 1 | 32 | 576 |
| 1 | 1 | 0 | 64 | 1.152 |
| 1 | 1 | 1 | 128 | 2.304 |

*TABLA 16 COMBINACIÓN DE LOS BITS DEL PREESCALER PARA USO EN EL WDT*[40]

La elección correcta deberá ser hecha teniendo en cuenta el máximo retardo que se logra obtener en algún programa tras la ejecución de dos instrucciones CLRWDT sucesivas

La mejor forma de evitar un reset a causa del WDT es no habilitar su flag en la fase de programación, evitando de esta forma el uso de la instrucción CLRWDT.

## TEMA 4.5. MEMORIA DE PROGRAMA DEL PIC16F84[41]

La memoria para almacenar el programa en el microcontrolador PIC16F84, tipo FLASH, es una memoria parecida a la ROM que puede ser borrada de manera eléctrica y reescrita, es decir, que basta con introducir datos con cierto nivel de tensión para que estos borren el programa anterior y graben uno nuevo[42]. Resulta perfecta para realizar pruebas y experimentos, además para la programación on-board (actualización del programa interno de chip sin necesidad de retirarlo del circuito de prueba).

---

[40] http://www.microchip.com, USA 2002

[41] Op cit [19]

[42] Para la propgramación de este microcontroladores se propone el software NOPPP (Capitulo 8) y su respectivo hardware.

Esta memoria tiene una longitud de 1 K byte x 14 Bits de memoria tipo Flash. La memoria tipo Flash tiene la característica de poderse borrar en bloques completos y no podrán borrarse posiciones concretas o específicas. Este tipo de memoria no es volátil, es decir, no pierde los datos si se interrumpe la energía.

La memoria del programa comienza en la posición 000h y termina en la posición 03FFh. Siempre está direccionada desde el Contador de Programa (PC), mientras que la memoria de datos puede direccionarse directamente desde parte del OPCODE (Operation Code: Codigo de Operación) de la instrucción o indirectamente a través de un registro denominado FSR (File Select Register).

En la figura 4.5 se muestra como está organizada la memoria de programa dentro del microcontrolador.

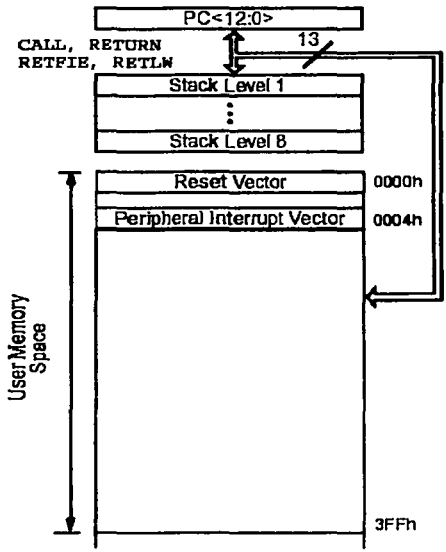



*FIGURA 4.5 MEMORIA DE PROGRAMA*

En la figura anterior puede observarse como dentro de la memoria de programa se encuentra el contador de programa PC, la pila o STACK y los vectores de interrupción y reset del microcontrolador PIC16F84, como parte de los vectores fijos en la memoria. Además se observa el espacio reservado para el programa del usuario a partir de la dirección 005 H hasta la 3FF H. lo cual implica una memoria de usuario de 1018 bytes.

## TEMA 4.6. INTERRUPCIONES PARA EL PIC16F84[43]

El sistema de interrupciones del microcontrolador PIC16F84 consiste en un mecanismo por el cual un evento interno o externo, asíncrono respecto del programa, puede interrumpir la ejecución de éste produciendo automáticamente un salto a una subrutina de atención, de manera que pueda atender inmediatamente el evento, y retomar luego la ejecución del programa exactamente en donde estaba en el momento de ser interrumpido.

Este mecanismo es muy útil por ejemplo para el manejo de timers o rutinas que deben repetirse periódicamente (refresh de display, antirebote de teclado, detección de pulsos externos, recepción de datos, etc.)

### 4.6.1. FUNCIONAMIENTO

Las interrupciones se comportan casi exactamente igual que las subrutinas. Desde el punto de vista del control del programa, al producirse una interrupción se produce el mismo efecto que ocurriría si el programa tuviese una instrucción CALL 0004h en el punto en que se produjo la interrupción.

---

[43] Op cit [19]

En uno de los registros de control del sistema de interrupciones existe un bit de habilitación general de interrupciones GIE[44], que debe ser programado en 1 para que las interrupciones puedan actuar.

Al producirse una interrupción, este bit se borra automáticamente para evitar nuevas interrupciones. La instrucción RETFIE que se utiliza al final de la rutina de interrupción, es idéntica a un retorno de subrutina, salvo que además coloca en uno automáticamente el bit GIE volviendo a habilitar las interrupciones.

Dentro de la rutina de interrupción, el programa deberá probar el estado de los flags de interrupción de cada una de las fuentes habilitadas, para detectar cual fue la que causó la interrupción y así decidir que acción tomar.

A continuación se muestra en la figura 4.6 un esquema lógico del circuito que controla las interrupciones.

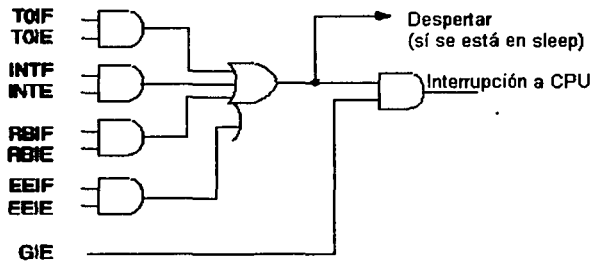



*FIGURA 4.6 DIAGRAMA LÒGICO DE INTERRUPCIONES DE LOS PIC16F8X*

[44] Es parte del registro INTCON y lo puede verificar en el tema 4.2.1.10

### 4.6.2. FUENTES DE INTERRUPCIÓN

La señal que produce la interrupción es en realidad una sola, que resulta de la combinación de todas las fuentes posibles y de los bits de habilitación. Existen dos grupos de fuentes, unas que se habilitan con solo colocar en uno el bit ***GIE***, y otras que además necesitan que este habilitado para cada fuente de interrupciones su respectivo bit de habilitación individual.

Las fuentes más comunes de interrupción varían con cada versión, y pueden ser por ejemplo:

- Interrupción externa por pin ***RB0/INT***: se refiere a que en el puerto B existe el pin RB0/INT el cual puede configurarse como un pin en el cual si recibe una determinada señal, este provocara que se interrumpa la ejecución normal del programa y pase el control a una subrutina del mismo

- Desborde del Timer 0 (***TMR0***): esta interrupción sucede cuando, una vez habilitada, el contador TMR0 llega a su máximo (255), sucede entonces que el programa ejecuta una subrutina en la dirección determinada para la interrupción.

- Cambio en el estado de los bits 4 a 7 del puerto B: si se habilita esta interrupción, la subrutina se ejecuta después de que se presenta un cambio en cualquiera de los bits del 4 al 7 del puerto B.

- Desborde del timer 1.

- Desborde del timer 2.

- Escritura de ***EEPROM*** finalizada: cuando se termina la escritura en la memoria EEPROM se produce una interrupción del programa principal, siempre y cuando se habilite el bit para esta interrupción.

La habilidad de quien realice el programa es fundamental para poder usar correctamente las diversas interrupciones, pues se requiere que estas tengan una secuencia dentro del programa ya que no es posible realizar una interrupción dentro de otra, por lo que ya se mencionó, es por ello que se debe tener cuidado al momento de hacer el programa.

Para evitar errores de algún tipo dentro del programa que se deba cargar al microcontrolador, existen diversos programas con los que se puede auxiliar. Sin embargo, en este trabajo se trata únicamente el que está desarrollado por el fabricante del PIC16F84, es decir el MPLAB.

# CAPITULO 5

# CONJUNTO DE INSTRUCCIONES (SET) DEL PIC16F84

# CAPITULO 5. CONJUNTO DE INSTRUCCIONES (SET) DEL PIC16F84.

El microcontrolador PIC16F84 dispone de un total de 35 instrucciones de una sola palabra que podrán aprenderse muy fácilmente. Este tema va orientado a entender las instrucciones para programar el PIC16F84 de una manera sencilla y rápida por lo que se explicarán a continuación el conjunto de instrucciones, su significado, modo de operación y traducción[45].

Las instrucciones tienen algunas letras que van relacionadas y que tienen cierta interpretación. Por ejemplo la letra "W" es el registro más importante que tiene el microcontrolador y es denominado ACUMULADOR, como ya se había mencionado. La letra *k* hace referencia a que se trata de una literal o un valor cualquiera, expresado en decimal (d), binario (b) o hexadecimal (h) después de que se pone el valor; con él se realizará la operación que indique la instrucción correspondiente en el programa. La letra *b* indica que se trata de un bit del registro presente en la instrucción a ejecutarse. Por último, la letra *f* hace referencia al registro que se encuentra en la instrucción a ejecutarse o a su dirección en memoria.

## TEMA 5.1. INSTRUCCIONES DE CONTROL Y MANEJO DE LITERALES.[46]

Estas instrucciones básicamente realizan operaciones con un valor definido por el usuario "k" y el registro de trabajo "W", aunque no en todos los casos.

---

[45] Si desea, para una referencia más rápida de todas las instrucciones puede ver tabla 17

[46] http://www.iespana.es/portosin/ (España, 2002)

- **INSTRUCCIÓN: ADDLW k**

La función de esta instrucción es la de sumar al acumulador W el valor k, es decir, realiza la operación W = W + k. El resultado de dicha operación lo guarda en el registro "W". Como ejemplo, se muestra lo siguiente:

```
MOVLW   3
ADDLW   1
```

Por lo tanto el acumulador tendrá el valor 4.

En lo que se refiere al registro de estado o STATUS, al aplicar esta instrucción, se modifican los bits Z, DC y C. Si Z vale 1 si el resultado de la operación es 0, que es el caso en el que el valor anterior es 255 decimal, si DC vale 1 el resultado de la operación es un número superior a 15 y por último si C vale 1 es porque el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado (W) vale 0. En caso contrario C vale 0 para indicar el resultado negativo.

- **INSTRUCCIÓN: ANDLW k**

Esta instrucción realiza una operación lógica AND entre el contenido de "W" y k. El resultado se guarda siempre en el acumulador "W", es decir, W = W AND k

Como ejemplo suponga que carga el acumulador con el binario W=10101010B y hace un AND con el binario k=11110000B, quedará el resultado de la operación en el acumulador "W". Es decir:

```
MOVLW 10101010B
ANDLW 11110000B
```

El resultado es "W" = 10100000B.

Solamente en el caso de que ambos bits sean 1, el resultado será 1. Esta instrucción compara dos bytes, bit a bit.

El registro STATUS solo se ve modificado en el bit Z. Si Z vale 1 el resultado de la operación es 0.

- **INSTRUCCIÓN: CALL k**

Esta instrucción llama a un grupo de instrucciones (subrutina) que comienzan en la dirección k, donde k puede ser un valor numérico o una etiqueta. Siempre termina con la instrucción de retorno **RETURN** o **RETLW**.

Su forma de operación es: CALL → k...RETURN → PC+1.

Las **subrutinas** son un grupo de instrucciones que forman un programa dentro del programa principal y que se ejecutan cuando las llama el programa principal.

Las subrutinas sirven para utilizarlas varias veces en cualquier parte del programa, sin necesidad de tener que copiar las mismas instrucciones, con el consiguiente ahorro de memoria.

La manera en que funcionan es cuando un programa ejecuta una instrucción **CALL**, se guarda en el **stack** el valor del registro **PC+1** (PC = Program Counter) de manera que al regresar de la subrutina continúa con la instrucción siguiente recuperándola del **stack**.

Las limitaciones de una subrutina son que en el PIC16F84 **están disponibles 8 niveles de stack**, por lo que el número máximo de **CALL** reentrantes (instrucciones **CALL** que contengan otra instrucción **CALL**) queda limitado a 8.

Como ejemplo suponga que: PRINCIPAL es la etiqueta que identifica una dirección de memoria, RETARDO es la etiqueta que identifica el comienzo de una subrutina, BUCLE es la etiqueta que identifica una dirección de memoria. Ahora suponga la siguiente parte de un programa:

```
PRINCIPAL  CALL    RETARDO
           BTFSC   PORTB,  RB0
           GOTO    PRINCIPAL
```

```
        *
        *
        *
RETARDO CLRF    CONTADOR
BUCLE   DECFSZ  CONTADOR, 1
        GOTO    BUCLE
        RETURN
```

En este listado se ve que la subrutina RETARDO salta a un grupo de instrucciones que forman un bucle y cuando éste termina regresa para seguir con la instrucción siguiente al salto (BTFSC...).

El registro STATUS no resulta modificado en ningún bit de estado.

- **INSTRUCCIÓN: CLRWDT**

Esta instrucción se utiliza cuando programa el PIC con la opción Watch Dog habilitada[47]. Para evitar el reset del PIC, el programa debe contener cíclicamente la instrucción **CLRWDT** para ponerlo a cero. Si no se pone a cero a tiempo, el **WDT** interpretará que se ha bloqueado el programa y ejecutará un reset para desbloquearlo.

El registro STATUS no es modificado en ningún bit de estado.

- **INSTRUCCIÓN: GOTO k**

Esta instrucción ejecuta un salto del programa, sin que exista alguna condición especial para ello, a la dirección k. El parámetro "**k**" puede ser un valor numérico o una etiqueta.

Como ejemplo vea lo siguiente:

---

[47] La forma de habilitar o deshabilitar el WDT se muestra en el Tema 5.6

```
        INSTRUCCIÓN 1
        GOTO            ABAJO : salta hasta la instrucción 6
        INSTRUCCIÓN 3
        INSTRUCCIÓN 4
        INSTRUCCIÓN 5
ABAJO   INSTRUCCIÓN 6
```

Lo que sucede en esta fracción de un programa es que primero se ejecuta la instrucción 1, después **GOTO** y continúa con la instrucción 6 saltándose las instrucciones 3, 4 y 5.

El registro STATUS no se modifica en ningún bit de estado.

- **INSTRUCCIÓN: IORLW k**

Esta instrucción realiza un **OR** entre el contenido del acumulador **"W"** y el literal **k**. El resultado se guarda siempre en el acumulador ( recuerde que **"k"** es un valor, no un registro).

Es decir que si carga el acumulador con el binario 11110000B y k= 00001111B y aplica esta instrucción:

```
MOVLW   11110000B
IORLW   00001111B
```

El resultado de la operación queda **"W"** = 11111111B.

Solamente en el caso de que ambos bits sean 0, el resultado será 0. Esta instrucción compara dos bytes, bit a bit.

Registro STATUS se ve modificado en su bit Z, por lo que vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: MOVLW k**

Esta instrucción asigna al acumulador **"W"** el valor del literal **k** (entre 0 y 255 decimal).

Si tiene el acumulador a cero o con cualquier valor, y quiere que contenga el que le asigne directamente entonces usará esta instrucción como sigue:

```
MOVLW 100
```

Por lo que al final el acumulador valdrá 100 (W = 100), sin importar el valor anterior del registro W.

En el registro STATUS no se modifica ningún bit de estado.

- **INSTRUCCIÓN: OPTION**

Esta instrucción guarda en el registro especial **OPTION** el valor contenido en el acumulador "**W**".

Si aplica la instrucción OPTION, entonces OPTION = W.

Esta instrucción existe para mantener la compatibilidad con los PIC producidos con anterioridad, y como en el futuro podría dejar de implementarse, Microchip aconseja realizar lo anterior de esta otra forma (considerando cargar W=10):

```
BSF     STATUS, RP0  ; activa el banco 1.
MOVLW   10H          ; carga el acumulador con 10H.
MOVWF   OPTION_REG   ; carga OPTION con el acumulador.
```

El registro de estado no tienen ninguna modificación, pues no existe operación alguna.

- **INSTRUCCIÓN: RETFIE**

Esta instrucción devuelve el control al programa principal después de ejecutarse una subrutina de gestión de interrupción.

Observe como ejemplo lo siguiente:

```
        ORG    00H
BUCLE   GOTO   BUCLE    ; bucle infinito.
        ORG    04H      ; vector de interrupción.
```

```
                RETFIE                  ; retorna de la interrupción
```

Este código de programa ejecuta un bucle infinito. Si habilita una de las interrupciones del PIC16F84, en cuanto ésta se produzca pasará el control al programa situado en la dirección 04H y la instrucción RETFIE regresa de la interrupción.

Al ejecutarse una interrupción, el bit GIE del registro INTCON se pone a 0 y así evita que otra interrupción se produzca mientras ya está con una en marcha.

Con la instrucción RETFIE se pone de nuevo el bit GIE a 1 para así atender de nuevo a futuras interrupciones.

No hay modificación de la palabra de estado.

- **INSTRUCCIÓN: RETLW**

Esta instrucción retorna de una subrutina al programa principal, cargando el acumulador "**W**" con el literal "**k**".

Es la última instrucción que forma una subrutina (al igual que **RETURN**).

Pero se preguntará ¿para qué sirve regresar de una subrutina con un determinado literal en el acumulador?. Será muy útil al programar con **TABLAS**.

Ejemplo:

```
                CALL      SUBRUT1 ; llama a Subrut1.
                MOVWF     DATO 1  ; carga W en Dato1.
                CALL      SUBRUT2 ; llama a Subrut2.
                MOVWF     DATO2   ; carga W en Dato2.
                  *
                  *
SUBRUT1   RETLW     0A          ; carga W = 0A y retorna.
SUBRUT2   RETLW     0B          ; carga W = 0B y retorna.
```

Al registro STATUS no se le modifica ningún bit de estado.

- **INSTRUCCIÓN: RETURN**

Esta instrucción retorna de una subrutina al programa principal en la instrucción siguiente a la llamada de la subrutina, tomando el valor almacenado en el **stack** para continuar.

Es la última instrucción que forma una subrutina (al igual que **RETLW**).

Observe la siguiente fracción de un programa como ejemplo:

```
            CALL            COMPARA ; llama a Compara.
            INSTRUCCION1
            INSTRUCCION2
              *
              *
COMPARA     INSTRUCCIÓN R1
            INSTRUCCIÓN R2
            RETURN
```

Se observa como se llama a la subrutina **COMPARA**, se ejecutan las instrucciones R1 y R2 y con el **RETURN** regresa a la instrucción siguiente al CALL y ejecuta las instrucciones 1, 2 y sigue con el programa.

- **INSTRUCCIÓN: SLEEP**

Esta instrucción detiene la ejecución del programa y deja el PIC en modo suspendido. No ejecuta ninguna instrucción hasta que sea nuevamente reinicializado (reset).

Durante este modo, el contador del Watch Dog (**WDT**) sigue trabajando, y si lo tiene activado el PIC se reseteará por este medio. El consumo de energía es mínimo.

- **INSTRUCCIÓN: SUBLW k**

Esta instrucción resta al literal **k** el valor almacenado en el **acumulador "W"** y el resultado se guarda en el **acumulador**.

Ejemplo:

```
MOVLW  10      ; carga el acumulador W con el valor 10.
SUBLW  15      ; resta a 15 el valor del acumulador.
```

Al final el acumulador tendrá el valor W = 5.

El registro STATUS se modifica en los bits Z, DC y C, de tal manera que si Z vale 1 significa que el resultado de la operación es 0. Si DC vale 1 el resultado de la operación es un número superior a 15. Si C vale 1 el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).

- **INSTRUCCIÓN: TRIS f**

Esta instrucción guarda el valor del acumulador "**W**" en uno de los registros especiales de **TRIS** que se indicó en el parámetro "**f**".

Los registros **TRIS** determinan el funcionamiento como entrada y salida de las líneas I/O del PIC.

A modo de ejemplo:

```
MOVLW  16H     ; carga el acumulador W con el valor 16H.
TRIS   PORTA ; carga el registro PORTA con el acumulador.
```

Esta instrucción existe para mantener la compatibilidad con los PIC producidos anteriormente, y como en el futuro podría dejar de implementarse, Microchip aconseja realizar el ejemplo anterior de esta otra forma (aunque ocupa más memoria):

```
BSF    STATUS, RP0 ; activa el banco 1.
MOVLW  16H         ; carga el acumulador con el valor 16H
MOVWF  TRISA       ; carga el registro PORTA con W.
```

- **INSTRUCCIÓN: XORLW k**

Esta instrucción realiza un OR exclusivo entre el contenido del **acumulador "W"** y el valor del literal **k**. El resultado se guarda siempre en el acumulador (**k** es un literal, no un registro).

Si carga el acumulador con el binario 11110000B y hace un XOR con el binario 10101010B, quedará el resultado de la operación en el acumulador **"W"**.

```
MOVLW  11110000B
XORLW  10101010B
```

El resultado de la operación queda como **"W"** = 01011010B.

Si ambos bits son iguales el resultado será 0. Y si ambos son diferentes, el resultado será 1. Por lo tanto, esta instrucción sirve para verificar si dos bytes son iguales o no, ya que compara dos bytes, bit a bit.

La única modificación en le registro de estado es en Z, si vale 1 el resultado de la operación es 0.

## TEMA 5.2. INSTRUCCIONES ORIENTADAS A REGISTROS.[48]

Estas instrucciones realizan operaciones con registros denominados como **"f"** y en ocasiones en combinación con el registro de trabajo.

[48] Op cit [46]

- **INSTRUCCIÓN: ADDWF f,d**

Esta instrucción suma el contenido del **acumulador** con el registro **"f"**, y el resultado se guarda dependiendo del valor de **"d"**; si d = 0, se guarda en **"W"** pero si d = 1, se guarda en **"f"**. Si no se pone nada, el valor por defecto es 1 y se guarda en **"f"**. También se puede indicar directamente **"W"** ó **"f"**.

El registro STATUS se modifica en los bits Z, DC y C.

- **INSTRUCCIÓN: ANDWF f,d**

Esta instrucción realiza una operación lógica **AND** entre el contenido del acumulador **"W"** y el registro **"f"**. El resultado se guarda según sea el valor de **"d"**, como ya se mencionó.

Un uso muy práctico de esta instrucción se ve cuando desea extraer los 4 bits menos significativos de un registro, podrá utilizar una máscara para obtenerlos.

Se usa el valor 00001111B en W para realizar la operación AND y así obtenerlos (para los 4 bits más significativos utilizaría el valor 11110000B).

```
MOVLW   10101010
MOVWF   f
MOVLW   00001111
ANDWF   f, W
```

El resultado queda guardado en el acumulador (W = 00001010B).

Solamente en el caso de que ambos bits sean 1, el resultado será 1. Se comparan dos bytes, bit a bit.

En el registro STATUS Z vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: CLRF f**

Esta instrucción pone a cero el valor contenido en el registro direccionado por el parámetro **"f"**. Puede decirse que borra el registro **"f"**.

Por ejemplo, si desea poner a cero el registro **TMR0**, cuya dirección es 01H, tendría que utilizar

```
CLRF  01H
```

Si ha incluido al inicio del código fuente el fichero **PIC16F84.INC**[49], podrá utilizar el nombre simbólico de dicho registro:

```
CLRF  TMR0
```

El registro STATUS pone a 1 el bit Z (ya que el resultado de la operación es 0).

- **INSTRUCCIÓN: CLRW**

Esta instrucción pone a cero el valor contenido en el registro **"W" (acumulador)**. Al igual que en la instrucción anterior el bit Z del registro de estado se pone a 1.

- **INSTRUCCIÓN: COMF f,d**

Esta instrucción efectúa el complemento del valor contenido en el registro direccionado por el parámetro **"f"**. La operación de **complementar** consiste, básicamente, en invertir los bits: poner los ceros a unos y los unos a ceros.

El parámetro **"d"** determina el destino del valor obtenido. Si d = **"W"**, el resultado se almacena en el **acumulador**, pero si d = **"f"**, el resultado se almacena en el propio registro.

---

[49] En el Tema 5.5 se trata la forma de insertar este tipo de archivos en el programa para el microcontrolador.

A modo de ejemplo observe lo siguiente:

Si tiene en PORTA el valor 00001111B, al ejecutar:

```
COMF   PORTA
```

El resultado será PORTA = 11110000B.

Si aplica para el mismo valor inicial, la instrucción:

```
COMF   PORTA, W
```

El resultado será W = 11110000B y PORTA = 00001111B.

En el registro STATUS, Z vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: DECF f,d**

Esta instrucción decrementa en uno el contenido del registro direccionado por el parámetro **"f"**. El parámetro **"d"** determina el destino del valor obtenido según se ha mencionado.

Si tiene un registro DIA = 7 y aplica la instrucción

```
DECF DIA, 0,
```

tendrá W = 6 y DIA = 7, y si aplica esta otra

```
DECF DIA, 1,
```

tendrá DIA = 6.

En el registro STATUS se modifica el bit Z .

- **INSTRUCCIÓN: DECFSZ f,d**

Esta instrucción decrementa el contenido del registro direccionado por el parámetro **"f"**, y si el resultado es 0 se salta la instrucción siguiente. El parámetro **"d"** determina el destino del valor obtenido. Es decir, la instrucción decrementa f (f-1) y salta si es cero.

Si el resultado guardado **no** es 0, sigue con la instrucción consecutiva sin salto.

- **INSTRUCCIÓN: INCF f,d**

Esta instrucción incrementa en uno el contenido del registro direccionado por el parámetro "**f**". Debe estar consciente de la base numérica que esta manejando para no considerar algún error en la secuencia de los números.

El parámetro "**d**" si d = "**W**", el resultado se almacena en el **acumulador** y si d = "**f**", el resultado se almacena en el propio registro "**f**".

En el registro STATUS, Z vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: INCFSZ f,d**

Esta instrucción incrementa en uno el contenido del registro direccionado por el parámetro "**f**", y si el resultado es 0 (después del 255 se regresa a cero) se salta la instrucción siguiente y como anteriormente el parámetro "**d**" determina el destino del valor obtenido.

Observe el siguiente ejemplo:

```
INCFSZ  VALOR, W
INSTRUCCION 1
INSTRUCCIÓN 2
```

Si el contenido del registro "VALOR" al incrementarlo en 1 es igual a 0, se guarda el resultado en el acumulador y sigue con la INSTRUCCION2, saltándose la INSTRUCCION1.

Si el resultado que guardó en W **no** es 0, sigue con la INSTRUCCION1 y después con la INSTRUCCION2 (no se salta la inmediata siguiente).

- **INSTRUCCIÓN: IORWF f,d**

Con esta instrucción puede realizar una operación lógica **OR** entre el acumulador **"W"** y el registro direccionado por el parámetro **"f"**.

Si no se pone nada en el parámetro "d", el valor por defecto es 1 y se guarda en **"f"**.

En el registro STATUS, Z vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: MOVF f,d**

Esta instrucción copia el contenido del registro direccionado por el parámetro **"f"** en el acumulador **"W"** o en el mismo registro **"f"**[50] según se ponga en el parámetro **"d"** . El valor por defecto es 1 y se guarda en **"f"**.

En el registro STATUS Z vale 1 si el resultado de la operación es 0.

- **INSTRUCCIÓN: MOVWF f**

Esta instrucción copia el contenido del acumulador **"W"** en el registro direccionado por el parámetro **"f"**.

Si desea escribir el valor 10H en el registro **TMR0**, que está situado en la dirección 01H, tendrá que cargar primero el valor en el **acumulador** y después copiarlo al registro, es decir:

```
MOVLW   10H ; cargar el valor 10H en el acumulador.
MOVWF   01H ; copia el acumulador en la dirección 01H.
```

[50] El motivo para copiar el contenido de un registro sobre si mismo, no es otro que poder comprobar en el registro **STATUS** el estado del bit Z.

Con los registros utilizados por el PIC para funciones específicas, es habitual no escribir directamente su dirección, sino el nombre simbólico definido en el fichero **PIC16F84.INC**. El ejemplo anterior quedaría así:

```
MOVLW  10H   ; cargar el valor 10H en el acumulador.
MOVWF  TMR0  ; copia el acumulador en el registro TMR0.
```

Esto será a consideración de quien realice la programación del PIC

- **INSTRUCCIÓN: NOP**

Esta instrucción no realiza ninguna función específica, pero consume **4 ciclos de reloj** completos. Es útil para insertar un retardo igual a un ciclo de máquina.

A modo de ejemplo vea lo siguiente:

Utilizando un cristal de cuarzo de 4 MHz. en el oscilador, podrá obtener un retardo igual a un microsegundo por cada instrucción **NOP** que inserte en el código del programa:

```
RETARDO  NOP
         NOP
         NOP
         RETURN
```

Cada vez que llame a la subrutina RETARDO, obtendrá 3 microsegundos de demora.

- **INSTRUCCIÓN: RLF f,d**

Esta instrucción rota a la izquierda todos los bits del registro direccionado en el parámetro "**f**" pasando por el bit **CARRY** del registro **STATUS** (o si se prefiere, desde los bits menos significativos a los más significativos).

Es como si **multiplicará por dos** el contenido del registro.

Vea el registro "**f**" de forma gráfica en la figura 5.1:

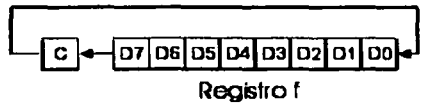



*FIGURA 5.1 ROTACIÓN A LA IZQUIERDA DE LOS BITS DE UN REGISTRO "F"*

El bit D7 pasa al **CARRY** del registro **STATUS**, el contenido del **CARRY** pasa al bit D0, el bit D0 al D1, etc.

Aquí también el parámetro "**d**" determina el destino del resultado.

En el registro STATUS se modifica el bit **C (CARRY)**.

- **INSTRUCCIÓN: RRF f,d**

Esta instrucción rota a la derecha todos los bits del registro direccionado en el parámetro "**f**" pasando por el bit **CARRY** del registro **STATUS** (o si se prefiere, desde los bits más significativos a los menos significativos) semejante como sucedió con la instrucción anterior.

Ahora es como si **dividiera por dos** el contenido del registro.

De forma gráfica se observa en la figura 5.2:


*FIGURA 5.2 ROTACIÓN A LA DERECHA DE LOS BITS DE UN REGISTRO "F"*

El bit **C** del registro **STATUS** pasa al bit D7, el D0 pasa al bit **C**, el D1 al D0, etc.

También se especifica el destino de la operación con el parámetro "**d**" y, lo mismo que para la anterior, el registro STATUS se modifica en el bit **C (CARRY)**.

- **INSTRUCCIÓN: SUBWF f,d**

Esta instrucción resta el valor contenido en el acumulador **"W"** del valor contenido en el registro direccionado por el parámetro **"f"**, determinando el destino mediante el parámetro **"d"**.

Observe el ejemplo:

Según sean los valores de **W** y el registro **DATO**, si aplica

SUBWF DATO, obtendrá diferentes resultados en el bit **CARRY**.

Si DATO = 3 y W = 2; el resultado será DATO = 1 y C = 1.

Si DATO = 2 y W = 2; el resultado será DATO = 0 y C = 1.

Si DATO = 1 y W = 2; el resultado será DATO = FF y C = 0.

Vea que C = 1 porque el resultado es positivo y C = 0 cuando el resultado es negativo. En cuanto al registro STATUS se modifica los bits Z, DC y C como sigue:.

Z vale 1 si el resultado de la operación es 0. DC vale 1 si el resultado de la operación es un número superior a 15. Por último C vale 1 si el resultado de la operación es positivo o el bit 7 del registro que contiene el resultado vale 0. En caso contrario C vale 0 (resultado negativo).

- **INSTRUCCIÓN: SWAPF f,d**

Esta instrucción intercambia el valor de los 4 bits más significativos (D7-D4) contenidos en el registro direccionado por el parámetro **"f"**, con los 4 bits menos significativos (D3-D0) del mismo. El parámetro **"d"** determina el destino.

Vea que si tiene

```
MOVLW 00001111
MOVWF DATO
MOVLW W = 00000000B ;al aplicar
SWAPF DATO ;  el resultado es DATO = 11110000B y W = 00000000B.
```

Y si aplica SWAPF DATO, W; el resultado es DATO = 00001111B y W = 11110000B.

Con la primera instrucción modifica el valor del registro DATO, y en la segunda instrucción modifica el valor del **acumulador** sin que varíe el registro DATO.

- **INSTRUCCIÓN: XORWF f,d**

Esta instrucción efectúa la operación lógica **XOR** (OR exclusivo) entre el valor contenido en el acumulador **"W"** y el valor contenido en el registro direccionado por el parámetro **"f"**.

Si ambos bits son iguales el resultado será 0. Y si ambos son diferentes, el resultado será 1. Esta instrucción compara dos bytes, bit a bit.

Esta instrucción sirve para comparar dos valores y comprobar si son iguales o no.

Suponga que tiene el registro NUMERO y quiere comprobar si es igual a 26H.

Tendría que efectuar las siguientes instrucciones:

```
MOVLW  26H          ; carga el acumulador con el valor a comparar.
XORWF  NUMERO, W    ; compara el acumulador con NUMERO.
BTFSS  STATUS, Z    ; salta la instrucción siguiente si XOR = 0.
GOTO   DISTINTO     ; salta a DISTINTO  si XOR no es 0.
GOTO   IGUAL        ; salta a IGUAL.
```

El registro STATUS modifica el bit Z.

## TEMA 5.3. INSTRUCCIONES ORIENTADAS A BITS.[51]

Básicamente se trata de instrucciones en las que se hace la modificación a un solo bit de algún registro "f".

---

[51] Op cit [46]

- **INSTRUCCIÓN: BCF f,b**

Esta instrucción pone a cero un bit que haya elegido mediante el parámetro "b" de un registro determinado.

Ejemplo:

```
BCF   PORTA,  RA4 ; pone a 0 el bit RA4 del registro PORTA
BCF   PORTA,  4   ; igual, si no conoce el nombre del bit
```

Si en PORTA tiene como valor inicial 11111111B, después de aplicar el ejemplo anterior, PORTA = 11101111B.

- **INSTRUCCIÓN: BSF f,b**

Esta instrucción es semejante a la anterior solo que ahora se pone a uno un bit que se haya elegido de un registro determinado.

Ejemplo:

```
BSF   PORTA,  RA0 ; pone a 1 el bit RA0 del registro PORTA
BSF   PORTA,  0   ; igual, si no conoce el nombre del bit
```

Si en PORTA tiene como valor inicial 00000000B, después de aplicar el ejemplo anterior, PORTA = 00000001B.

- **INSTRUCCIÓN: BTFSC f,b**

Esta instrucción comprueba el valor del bit "b" en el registro "f", y si "b" = 0 entonces se salta la siguiente instrucción. Si "b" = 1 no salta y sigue con su ejecución normal.

Ejemplo:

```
BTFSC   PORTA,  2
INSTRUCCIÓN 1
INSTRUCCIÓN 2
```

Si en PORTA tiene como valor inicial 11111011B, el programa continúa con la instrucción 2, saltándose la instrucción 1

Si en PORTA tiene el valor 00000100B, el programa sigue con la instrucción 1 y después la instrucción 2.

- **INSTRUCCIÓN: BTFSS f,b**

Esta instrucción comprueba el valor del bit "b" en el registro "f", y si "b" = 1 entonces se salta la siguiente instrucción. Si "b" = 0 no salta y sigue con su ejecución normal.

Ejemplo:

```
BTFSS   PORTB,  7
INSTRUCCIÓN 1
INSTRUCCIÓN 2
```

Si en PORTB tiene como valor inicial 10000000B, el programa continúa con la instrucción 2, saltándose la instrucción 1.

Si en PORTB tiene el valor 01111111B, el programa sigue con la instrucción 1 y después la instrucción 2.

## TEMA 5.4. RESUMEN DEL SET DE INSTRUCCIONES

Como observó el set de instrucciones no es muy amplio y se divide para tres distintos casos como son: operaciones orientadas a Bytes, operaciones orientadas a Bits, y por ultimo orientadas a Literales.

Por lo que resta, en esta sección se muestra la tabla 17 para el mejor entendimiento del set de instrucciones. Se hace énfasis al operando o Mnemónico, la descripción o función que realizan, los ciclos utilizados para cada instrucción, el código de operación, así como los bits de la palabra o registro de estado (STATUS) que son modificados al momento de aplicar determinada instrucción.

| Mnemónicos, Operandos | | Descripción | Ciclos | 14-Bit Opcode MSB | | LSB | | Estado "STATUS" | Notas |
|---|---|---|---|---|---|---|---|---|---|
| OPERACIONES ORIENTADAS A BYTES | | | | | | | | | |
| ADDWF | f, d | Add W and f | 1 | 00 | 0111 | dfff | ffff | C, DC, Z | 1,2 |
| ANDWF | f, d | AND W with f | 1 | 00 | 0101 | dfff | ffff | Z | 1,2 |
| CLRF | f | Clear f | 1 | 00 | 0001 | lfff | ffff | Z | 2 |
| CLRW | - | Clear W | 1 | 00 | 0001 | 0xxx | xxxx | Z | |
| COMF | f, d | Complement f | 1 | 00 | 1001 | dfff | ffff | Z | 1,2 |
| DECF | f, d | Decrement f | 1(2) | 00 | 0011 | dfff | ffff | Z | 1,2 |
| DECFSZ | f ,d | Decrement f, Skip if 0 | 1 | 00 | 1011 | dfff | ffff | | 1,2,3 |
| INCF | f, d | Increment f | 1(2) | 00 | 1010 | dfff | ffff | Z | 1,2 |
| INCFSZ | f, d | Increment f, Skip if 0 | 1 | 00 | 1111 | dfff | ffff | | 1,2,3 |
| IORWF | f, d | Inclusive OR W with f | 1 | 00 | 0100 | dfff | ffff | Z | 1,2 |
| MOVF | f, d | Move f | 1 | 00 | 1000 | dfff | ffff | Z | 1,2 |
| MOVWF | f, d | Move W to f | 1 | 00 | 0000 | lfff | ffff | | |
| NOP | - | No Operation | 1 | 00 | 0000 | 0xx0 | 0000 | C | 1,2 |
| RLF | f, d | Rotate Left f through | 1 | 00 | 1101 | dfff | ffff | C | 1,2 |
| RRF | f, d | Rotate Right f through | 1 | 00 | 1100 | dfff | ffff | C,DC,Z | 1,2 |
| SUBWF | f, d | Subtract W from f | 1 | 00 | 0010 | dfff | ffff | | 1,2 |
| SWAP | f, d | Swap nibbles in f | 1 | 00 | 1110 | dfff | ffff | | 1,2 |
| XORWF | f, d | Exclusive OR W with f | 1 | 00 | 0110 | dfff | ffff | Z | 1,2 |
| OPERACIONES ORIENTADAS A BITS | | | | | | | | | |
| BCF | f, d | Bit Clear f | 1 | 01 | 00bb | bfff | ffff | | 1,2 |
| BSF | f, d | Bit Set f | 1 | 01 | 01bb | bfff | ffff | | 1,2 |
| BTFSC | f, b | Bit Test f, Skip if Clear | 1(2) | 01 | 11bb | bfff | ffff | | 3 |
| BTFSS | f, b | Bit Test f, Skip if Set | 1(2) | 01 | 11bb | bfff | ffff | | 3 |
| OPERACIONES ORIENTADAS A LITERALES | | | | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ADDLW | k | Add literal and W | 1 | 11 | 111x | kkkk | kkkk | C,DC,Z | |
| ANDLW | k | AND literal with W | 1 | 11 | 1001 | kkkk | kkkk | Z | |
| CALL | k | Call subroutine | 2 | 10 | 0kkk | kkkk | kkkk | | |
| CLRWDT | k | Clear Watchdog Timer | 1 | 00 | 0000 | 0110 | 0100 | TO, PD | |
| GOTO | k | Go to Address | 2 | 10 | 1kkkk | kkkk | kkkk | | |
| IORLW | k | Inclusive OR literal with W | 1 | 11 | 1000 | kkkk | kkkk | Z | |
| MOVLW | k | Move literal to W | 1 | 11 | 00xx | kkkk | kkkk | | |
| RETFIE | - | Return from interrupt | 2 | 00 | 0000 | 0000 | 1001 | | |
| RETLW | k | Return with literal in W | 2 | 11 | 01xx | kkkk | kkkk | | |
| RETURN | - | Return from Subroutine | 2 | 00 | 0000 | 0000 | 1000 | | |
| SLEEP | - | Go into standby mode | 1 | 00 | 0000 | 0110 | 0011 | TO,PD | |
| SUBLW | k | Subtract W from literal | 1 | 11 | 110x | kkkk | kkkk | C,DC,Z | |
| XORLW | k | Exclusive OR literal with W | 1 | 11 | 1010 | kkkk | kkkk | Z | |

Notas:

1 Cuando un registro de entrada / salida es modificado como función de si mismo (ejem: MOVF PORTB1) el valor usado, será el valor presente en los pines, ejemplo, si el dato de Latch es "1" para la configuración de salida de un pin este será manejado como un dispositivo externo, y el dato será escrito devuelta como "0".

2 La instrucción es ejecutada en el registro TMR0 y es aplicable cuando d=1, el preescaler Será limpiado y asignado como Timer 0 en un modulo.

3 Si el PC es modificado o una prueba de condicional es verdadera, la instrucción requeriría de dos ciclos, si el segundo ciclo es ejecutado como un NOP.

*TABLA 17 SET DE INSTRUCCIONES PARA EL PIC16F84*[52]

## TEMA 5.5. LENGUAJE DE DIRECTIVAS.

Las ***instrucciones*** de los PIC's son únicas y no hay más que las que ha visto anteriormente. No confundir las propias ***instrucciones*** de los PIC's con las ***directivas*** del ensamblador.

[52] "Microchip PIC Microcontrollers Data Book", Microchip Technology Inc., Microchip, 1997. U.S.A, Pag 56.

Las famosas ***directivas*** son solo eso, directivas para el ensamblador, le dicen cómo hacer tal cosa o tal otra, y solo sirven para realizar las cosas más rápido, y así ayudar en la tarea de programación. Hay muchos tipos de ***directivas***, las cuales se escriben dentro del código fuente. Por eso no hay que confundir las ***directivas*** con las ***instrucciones*** del PIC.

Los tipos de directivas son[53]:

- Directivas de control.
- Directivas de datos.
- Directivas de listado.
- Directivas de macros.
- Directivas de ficheros objeto.

Vea las que tienen mayor uso:

**__CONFIG:** Sirve para indicarle al PIC como está configurado para trabajar, es lo que se refiere a la palabra de configuración.

**ORG:** Le indica al ensamblador dónde comienza el programa, y así lo graba a partir de la posición que se especifica aquí.

---

[53] Las directivas de control, permiten un ensamblado CONDICIONAL.

Las directivas de datos, son todas aquellas que permiten la manipulación simbólica y posicionamiento en memoria.

Las directivas de listado permiten todo el control sobre la paginación o listado del programa, así como su formato.

La directivas de macros, permiten todas las gestiones de las macros.

Las directivas de fichero objeto, sólo se utilizan para la creación de ficheros objeto o reubicables/reusables para luego enlazar con el MPLINK.

**#DEFINE:** Muy útil. Define un nombre para algún registro, pin, etc. Por ejemplo, #DEFINE SALIDA PORTA,3 sirve para no tener necesidad de recordar cual era el pin de salida, sino que sólo lo menciona como SALIDA. Puede ponerlo a 0 con la instrucción BCF SALIDA en vez de hacer BCF PORTA,3 o BCF RA3.

**RES:** Se utiliza para reservar espacios en memoria para las variables utilizadas en el programa. Muy útil cuando un registro cambia su valor durante la ejecución del programa.

**END:** Indica al MPASM que ha terminado el programa.

**EQU:** Define una constante con una etiqueta. La constante puede ser un registro o un valor numérico.

**#INCLUDE:** Con esta directiva empiezan casi todos los programas. Como el ensamblador no sabe que microprocesador se está utilizando en el código, Microchip ha creado una serie de archivos con extensión *.INC donde se definen todos los registros y datos respecto del procesador que se esté utilizando. En dicho archivo se define que la dirección 0x05 se llama PORTA, y por eso no hay necesidad de recordar la dirección, simplemente escribir en el código CLRF PORTA directamente. Lógicamente, si en el código está definido que PORTA ahora se llama R2D2, así lo le tendrá que llamar. Por eso al principio del código se pone ***#INCLUDE P16F84.INC*** para trabajar con el PIC16F84. Todos los archivos *.INC se encuentran en el directorio de instalación del MPLAB. También sirve esta directiva para incluir librerías de rutinas. Si no existiera el archivo .inc debe definir cada nobre de dirección mediante la directiva EQU.

**MACRO:** Para crear una macro. Por ejemplo:

```
NUEVO MACRO
        CLRF PORTA
        BSF PORTB,2
        ENDM
```

Los bits del 4 al 13 (llamados *CP* = code protection) sirven para la protección del programa en memoria del microcontrolador, esto sirve para que nadie pueda ver el código del programa, de tal forma que si son puestos a 1 no existe dicha protección y el programa puede ser leído, no así si se ponen a cero. Para no proteger el codigo estos bits generan el número hexadecimal 3FF.

El bit 3 sirve para habilitar el Power Up Timer ( $\overline{\text{PWRTE}}$ ), si vale 1 es deshabilitado y se habilita con un cero. Normalmente se deja habilitado.

El bit 2 (*WDTE*) es el que permite habilitar el Watch Dog mediante un 1 en esta posición de la palabra de configuración.

Por último los bits 0 y 1, *FOSC0 y FOSC1* respectivamente, sirven para seleccionar el tipo de oscilador externo con el cual trabajara el PIC[54]. Las combinaciones de estos bits para el tipo de oscilador son:

11: oscilador tipo RC (Resistor/Capacitor)

10: oscilador tipo HS (High Speed): Cristal / Resonador de Alta Velocidad.

01: oscilador tipo XT (Cristal / Resonador)

00: oscilador tipo LP (Low Power): Cristal de Bajo Consumo.

Dentro de un programa generado para el PIC16F84 esta palabra se debe indicar al comienzo del mismo. Aunque existen varias formas de indicar esta palabra dentro del programa, la más recomendable se basa en lo expuesto anteriormente y mediante la directiva **__CONFIG** (es doble guión bajo), es decir, se debe poner al iniciar del programa

[54] Para mayor detalle del tipo de oscilar diríjase al Tema 6.2

**__CONFIG "X"**

Donde X representa un numero en hexadecimal, el cual se obtiene de los bits seleccionados para la forma de trabajo del PIC.

A modo de ejemplo suponga que desea que el PIC trabaje con el *WatchDog* activado, un oscilador del tipo RC, con temporizador de reset y con código protegido. Con esto se genera el número binario 01111, al que le corresponde en hexadecimal "0F".

También es posible configurar esta palabra de la siguiente forma:

**__CONFIG_CP_ON &_PWRTE_ON &_WDT_OFF &_XT_OSC,**

Esto indica que debe trabajar con código protegido, el power time habilitado, el *WATCHDOG* deshabilitado y un oscilador del tipo XT. Sin embargo, en esta tesis se recomienda usar el método a base de números hexadecimales debido a que la última opción presentada depende de la inclusión del archivo **.inc** en el programa.

# CAPITULO 6

# OSCILADOR GENERADOR

# CAPITULO 6. OSCILADOR GENERADOR

## TEMA 6.1. CICLO DE INSTRUCCIÓN[55]

El reloj es una señal muy importante del microcontrolador, y se obtiene de un componente externo llamado "Oscilador". La entrada del reloj es el pin llamado OSC1.

La señal del oscilador entra en un microcontrolador vía OSC1 donde el circuito interno del microcontrolador divide el reloj en cuatro relojes o semiciclos Q1, Q2, Q3, y Q4 los cuales no se traslapan(figura 6.1). Estos cuatro relojes constituyen un ciclo de la instrucción (también llamado ciclo de la máquina) durante el cual la instrucción se ejecuta.

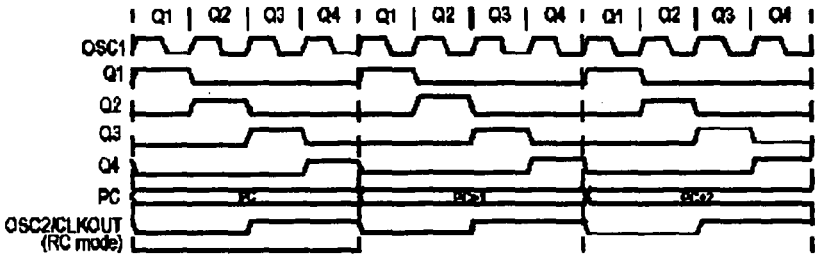



*FIGURA 6.1 SEÑALES O CICLOS DE RELOJ*

[55] "Microcontroladores PIC", Vallejo Horacio D. Quark S.R.L, 2002, Argentina, Pag 13

La ejecución del inicio de la instrucción se hace llamando una instrucción que está en la siguiente línea. La instrucción se llama desde la memoria del programa cada Q1 y es escrita en el registro de la instrucción en Q4. La decodificación y ejecución de la instrucción se hace entre los ciclos Q1 y Q4. En la figura 6.1 puede ver la relación entre el ciclo de la instrucción y reloj del oscilador (OSC1) así como los relojes internos Q1 a Q4. El contador de programa (PC) sostiene la información de la dirección de la próxima instrucción.

Un ciclo de instrucción consiste de cuatro ciclos Q (Q1, Q2, Q3 y Q4). La instrucción normalmente es extraída y su ejecución es conducida de tal manera que la extracción se hace en un ciclo de instrucción mientras se decodifica y se ejecuta en otro ciclo de instrucción. Sin embargo, debido a la conducción (Pipeline), cada instrucción se ejecuta efectivamente en un ciclo. Si la instrucción causa que el contador de programa (PC) cambie, entonces dos ciclos son requeridos para completar la instrucción.

| | TCY0 | TCY1 | TCY2 | TCY3 | TCY4 |
|---|---|---|---|---|---|
| | BUSQUEDA 1 | EJECUTA 1 | | | |
| 1. MOVLW 55 h | | BUSQUEDA 2 | EJECUTA 2 | | |
| 2. MOVWF PORTB | | | BUSQUEDA 3 | EJECUTA 3 | |
| 3. CALL SUB1 | | | | BUSQUEDA 4 | SALIR |
| 4. BSF PORTA BIT 3 | | | | | BUSQUEDA SUB1 | EJECUTA SUB1 |

*FIGURA 6.2 FLUJO DE EJECUCIÓN DE INSTRUCCIONES*

En la figura 6.2 se observa lo siguiente:

En TCY0 se lee la instrucción MOVLW 55h

En TCY1 se ejecuta la instrucción MOVLW 55h y se lee la instrucción MOVWF PORTB.

En TCY2 se ejecuta MOVWF PORTB y lee la instrucción CALL SUB_1.

En TCY3 se ejecuta una llamada a un subprograma CALL SUB_1, y se lee la instrucción BSF PORTA, BIT3. Como esta instrucción no es la que se necesita, o no es la primera instrucción del subprograma SUB_1 cuya ejecución es la siguiente, la instrucción debe leerse de nuevo. Éste es un buen ejemplo de una instrucción que necesita más de un ciclo.

En TCY4 el ciclo de instrucción es totalmente usado a para leer la primera instrucción de un subprograma a la dirección SUB_1.

En TCY5 (que no se muestra) se ejecuta la primera instrucción del subprograma SUB_1 y lee el próximo.

## TEMA 6.2. TIPOS DE OSCILADOR

El oscilador es un circuito externo al PIC que sirve para generar una señal de reloj de trabajo para el microcontrolador. El reloj es necesario para que el microcontrolador pueda ejecutar un programa o instrucciones de programa

El PIC16F84 puede trabajar con cuatro diferentes configuraciones para su oscilador externo. Desde la configuración con osciladores de cristal y los de resistencia capacitancia, serán los osciladores tratados aquí, ya que varios de ellos resultan muy semejante solo se hará mención a algunos usados más frecuentemente. A los osciladores de resistencia capacitancia se les designa como RC y a los de cristal como XT, HS o LP según la frecuencia a la que oscilan.

### 6.2.1. OSCILADOR TIPO HS[56]

En primer lugar se encuentra un Oscilador llamado "HS" basado en un Cristal para frecuencias mayores a 4 MHz. Esta versión es la más costosa; pero representa la forma mas práctica por la cantidad de conexiones y por la precisión en la señal de reloj emitida. En la figura 6.3 se muestra como debe conectarse al microcontrolador.

---

[56] http://club.telepolis.com/fremiro/mplab.htm

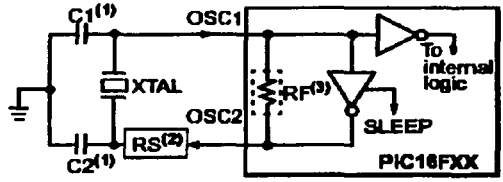



*FIGURA 6.3 CONFIGURACIÓN BÁSICA DEL OSCILADOR TIPO HS*

La resistencia RS se puede o no colocar según las recomendaciones del fabricante del cristal.

Estos tipos de cristales están diseñados especialmente para tecnologías TTL. Las frecuencias (MHz) disponibles para esta versión de cristal son muy amplias y las más usuales son 1, 1.8432, 2, 4, 8, 10, 11.059, 12, 14.31818, 16, 20, 25, 32, 33, 40, 50, 80 y 100 MHz. Se puede observar claramente que usted podrá adquirir este tipo de cristal con frecuencias por debajo de 4 MHz lo cual quiere decir que usted podrá configurar su microcontrolador en "XT" indicándole que se encuentra por debajo de 4 MHz.

### 6.2.2. OSCILADOR TIPO RC[57]

En segundo lugar se muestra el oscilador tipo RC, es el más económico por que tan solo se utiliza un condensador no polarizado y una resistencia. Este tipo de oscilador proporciona una estabilidad mediocre en la frecuencia y podrá ser utilizado para aquellos proyectos en que no se requiera precisión. Observe la figura 6.4 donde se muestra como debe ser conectado al microcontrolador el circuito RC.

---

[57] Op cit [56]

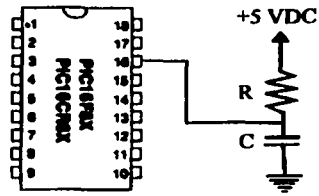



*FIGURA 6.4 CONFIGURACIÒN BÁSICA DEL OSCILADOR TIPO RC*

Es importante saber que para valores de resistencia menor a 4 K, el sistema se hace inestable o se podría detener la frecuencia de oscilación totalmente. Para valores de resistencias mayores a 100 Megas, el oscilador se hace susceptible al ruido, humedad y a la temperatura por lo tanto se recomienda que para este tipo de oscilador se encuentre en el rango de 5K < R < 100 K. Por otro lado se recomienda utilizar un condensador no polarizado de 20 pF por el ruido del medio y la estabilidad del sistema. También es importante saber que la tensión de alimentación que tendrá el microcontrolador influye directamente en la frecuencia final.

En la figura 6.5 se muestran algunas combinaciones de resistencias y condensador, se muestra una frecuencia máxima de 700 KHz basado en un capacitor de 300 pF.

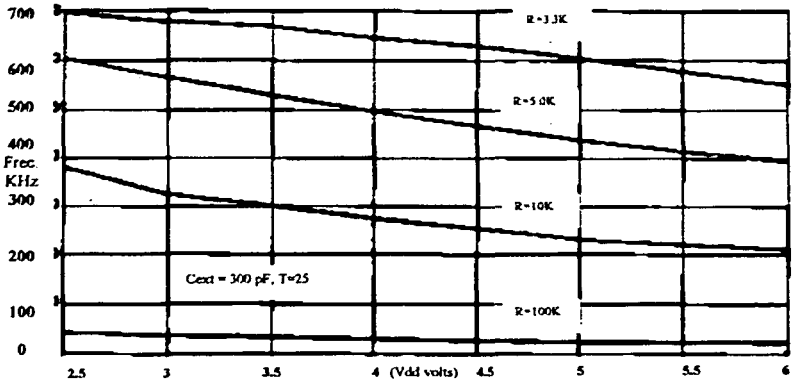



*FIGURA 6.5 RESISTENCIA Y CAPACITANCIA PARA DIFERENTES FRECUENCIAS*

Una consideración importante que se debe tener cuando se utiliza un oscilador tipo RC es que la señal del mismo no es perfectamente cuadrada, sino más bien es como se muestra en la figura 6.6.

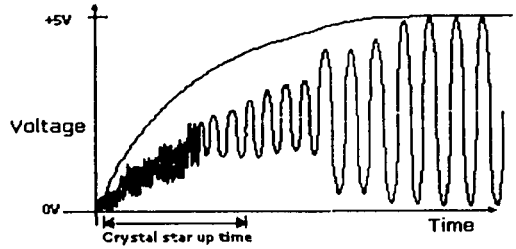



*FIGURA 6.6 DIAGRAMA DE SEÑAL DE UN OSCILADOR TIPO RC*

Para prevenir las variaciones en la señal del microcontrolador necesita mantener en reset al microcontrolador que es un estado de estabilización del reloj de oscilación, por lo que arriba en el diagrama se muestra.

### 6.2.3. OSCILADORES TIPO XT[55]

En tercer lugar está el oscilador tipo "XT" para frecuencias no mayores de 4 MHz. En la figura 6.7 se puede observar la configuración del circuito:

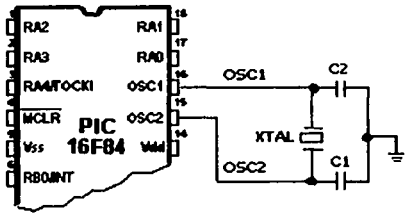



*FIGURA 6.7 CONFIGURACIÓN BASICA PARA EL OSCILADOR TIPO XT*

La condición básica importante para este oscilador es que los condensadores C1 y C2 deberán ser iguales.

Los osciladores y capacitores pueden ser un solo componente alojándose en 3 pines, estos elementos son llamados resonadores (figura 6.8) de cerámica.

[55] Op cit [56]

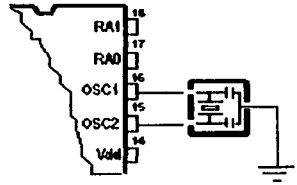



*FIGURA 6.8 RESONADOR CONECTADO A UN MICROCONTROLADOR*

Para lograr la frecuencia de oscilación se debe escoger la capacidad del capacitor, que en ambos caso el valor de c1 será igual al de c2, la recomendación es colocar un oscilador cerca del microcontrolador para evitar interferencia entre las líneas de transmisión de señal de reloj.

Los capacitores recomendables para las diferentes frecuencias son:

| Modo | Freq | OSC1/C1 | OSC2/C2 |
|---|---|---|---|
| LP | 32 kHz | 68 - 100 pF | 68 - 100 pF |
| | 200 kHz | 15 - 33 pF | 15 - 33 pF |
| XT | 100 kHz | 100 - 150 pF | 100 - 150 pF |
| | 455 KHz | 47 - 100 pF. | 47 - 100 pF. |
| | 2 MHz | 15 - 33 pF | 15 - 33 pF |
| | 4 MHz | 15 - 33 pF | 15 - 33 pF |
| HS | 4 MHz | 15 - 33 pF | 15 - 33 pF |
| | 10 MHz | 15 - 33 pF | 15 - 33 pF |

# CAPITULO 7

# SOFTWARE MPLAB

# CAPITULO 7. SOFTWARE MPLAB

EL software MPLAB es un "Entorno de Desarrollo Integrado" (Integrated Development Environment, IDE) que funciona bajo Windows, mediante el cual puede desarrollar programas para los microcontroladores de las familias PIC 16/17 de la empresa Microchip.

EL MPLAB permite escribir, depurar y optimizar los programas diseñados para utilizar con dichos microcontroladores.

De nada serviría saber programar si finalmente no es posible hacerlo por carecer de las herramientas necesarias para ello. Por eso es importante conocer el funcionamiento del MPLAB.[59]

Con el MPLAB es posible:

- Escribir programas fuente y depurarlos
- Detectar errores automáticamente en programas fuente para corregirlos.
- Depurar programas utilizando puntos de corte (breakpoints) mediante valores de los registros internos.
- Observar el flujo del programa con el simulador MPLAB -SIM.
- Realizar medidas de tiempo utilizando un cronómetro.
- Ver variables en las ventanas de observación.

---

[59] http://www.iespana.es/portosin/ (España, 2002)

Este entorno funciona como un contenedor, es decir, las diferentes opciones que contiene están asociadas a programas independientes que se ejecutarán cuando sean seleccionados los diferentes modos dentro del MPLAB.

El MPLAB incluye un organizador de proyectos para mantener el código ordenado, un editor de texto y un simulador para depurar el software que desarrolle.[60]

La ventaja de utilizar este software, frente a otros que permiten la creación, depuración, edición, seguimiento y/o simulación de programas para el microcontrolador PIC16F48, que es proporcionado por el fabricante y se encuentra accesible en forma gratuita en su pagina de Internet. Por otro lado la cantidad de información que se puede obtener sobre las características del programa que desarrolle para el PIC, gracias a este software, es comparable con la que proporcionan otros programas semejantes.

Si bien es posible crear un programa (no confundir con ensamblado) para el PIC en cualquier editor de texto, considerando la estructura adecuada, no es posible analizarlo de manera visible y gráfica, lo cual puede provocar errores al momento de pretender que funcione el microcontrolador.

Por ello es muy importante el uso de un software como el MPLAB, pues éste permite reducir el tiempo de verificación y corrección de errores en el programa y la visualización del funcionamiento del mismo. El ambiente de trabajo, tipo Windows, permite que se pueda interactuar más fácil con las diferentes variables que afectan y son afectadas dentro del programa a ensamblar, logrando así un mejor análisis en cuanto al verdadero comportamiento del mismo una vez cargado dentro del microcontrolador.

---

[60] Op cit [56]

La instalación del MPLAB no implica grandes problemas, solo se deben tener en cuenta las características de la computadora a usar y las que requiere el MPLAB según lo especifica el fabricante, así como las instrucciones que se proporcionan al momento de la instalación.

Considerando que se tiene instalado el software MPLAB, los siguientes temas están enfocados a tratar, de manera rápida y sencilla, su uso para quienes no tienen experiencia con él, de tal forma que sea posible crear pequeños programas didácticos para el PIC16F84. Posteriormente, la experiencia le dará al programador. la oportunidad de crear sus propios proyectos, tan complejos como lo desee.

## TEMA 7.1. ORGANIZADOR DE PROYECTOS

El organizador de proyectos es la parte central del MPLAB. Sin la creación de un proyecto no se puede hacer depuración alguna. A través de él podrá:

- Crear un proyecto.
- Añadir un archivo de código fuente a un proyecto.
- Ensamblar o compilar código fuente.
- Editar código fuente..
- Reconstruir todo el código fuente, o compilar un archivo solo.
- Depurar el código fuente.

## TEMA 7.2. CONFIGURANDO EL MPLAB PARA UN NUEVO PROYECTO

Si ya tiene instalado el MPLAB, lo primero que tiene que hacer es configurarlo para su uso, si es editor solamente o simulador.

Al poner simulador se entiende que es para programar, depurar, etc. Para ello debe accesar al menú ***OPTIONS*** y abrir ***DEVELOPMENT MODE*** en donde aparece la ventana de la figura 7.1 , en la cual se configura solamente la solapa ***TOOLS*** y ***CLOCK*** dejando el resto por defecto. En la ventana se muestra la selección del PIC16F84, y en el modo ***SIMULATOR.***

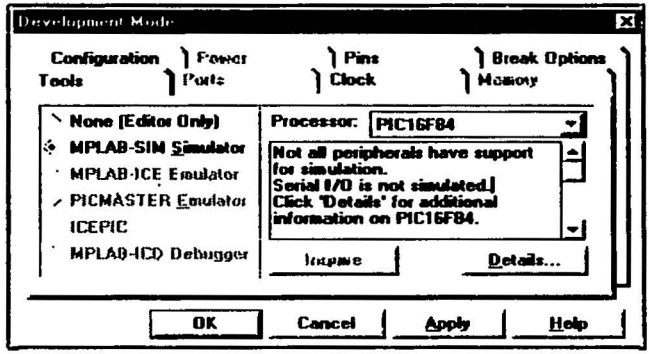



*FIGURA 7.1 MENÚ DE CONFIGURACIÓN DEL MPLAB*

Se debe pulsar en el botón OK o APPLY, y se observará que en la barra de estado o STATUS BAR (en la parte baja de la ventana del MPLAB) cambia el chip para el PIC16F84 y en modo simulador (SIM).

En caso de recibir un mensaje como el de la figura 7.2 es porque ha configurado mal el modo simulador, de un clic en **cancel** y seleccione correctamente el modo **MPLAB-SIM** para poder continuar.

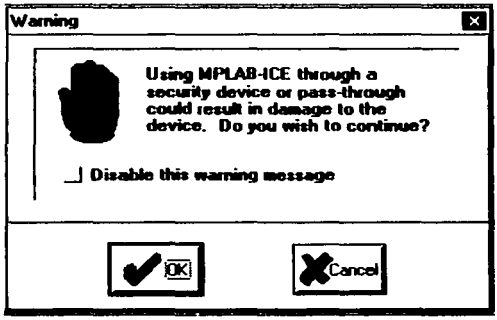



*FIGURA 7.2 MENSAJE DE PRECAUCIÓN EN EL MODO SELECCIONADO*

También se sugiere que en caso de recibir un mensaje como el que se muestra en la figura 7.3, de un clic en OK, ya que este mensaje estará notificándole que está configurando un modo nuevo al MPLAB, y que la memoria de programa del mismo será limpiada para empezar un nuevo proyecto.

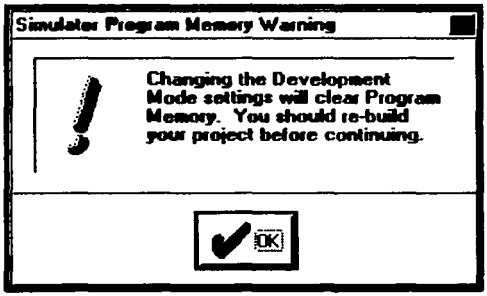



*FIGURA 7.3 MENSAJE DE PRECAUCIÓN POR CAMBIO EN LA CONFIGURACIÓN*

## TEMA 7.3. CREAR UN PROYECTO NUEVO

El siguiente paso es la creación de un nuevo proyecto, aunque esto implica todas las consideraciones de programación, no resulta tan difícil como aparenta. Además, la intención de estos temas es simplificar lo más posible la creación de un proyecto, como se describe a continuación.

### 7.3.1. NUEVO PROYECTO

Para todo programa es preferible que todo lo que desee hacer con él pudiese abrir o cerrar todo al mismo tiempo, pues para ello tiene el PROYECTO. Con un proyecto puede agregar ventanas de registros personales que sólo ese programa necesita o utiliza. Puede simular algunas condiciones que únicamente ese programa usa, y que sería completamente inútil para otro programa.

Dentro de un proyecto puede realizar, guardar y personalizar todo a su gusto y bajo un mismo nombre. Luego, al abrirlo de nuevo, lo hará con toda la personalización y valores previamente guardados. Es algo así como la personalización de Windows con varios usuarios, en el que cada uno tiene un tipo diferente de escritorio, etc.

Para iniciar seleccione ***FILE - NEW*** del menú ***FILE*** y saldrá la figura 7.4 con una pantalla de dialogo:

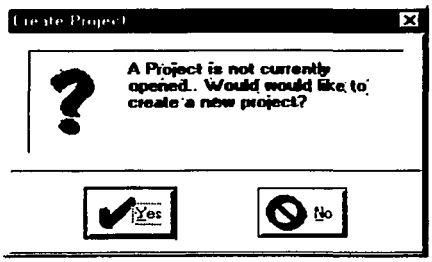



*FIGURA 7.4 VENTANA CREATE PROJECT*

Observe que también se abre una ventana en blanco en el fondo, que es donde se debe introducir el código fuente. En la ventana anterior se da un clic en YES y así comenzará a crear un proyecto.

Posteriormente se abrirá la siguiente ventana para que escriba el nombre del proyecto, y seleccione el directorio donde quedará guardado.

La extensión es por defecto *.PJT que corresponde a proyectos del MPLAB. Dando un nombre y haciendo un clic en el botón OK, automáticamente se abre el siguiente cuadro de EDICIÓN DE PROYECTO según se muestra en la figura 7.5.

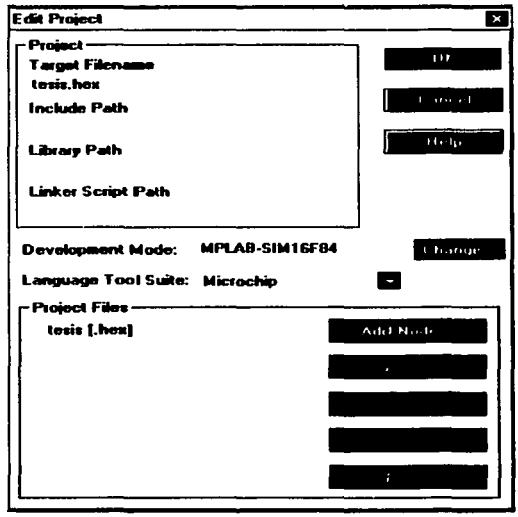



*FIGURA 7.5 VENTANA EDIT PROJECT*

Tanto el simulador como el programador o el emulador trabajan a partir del archivo *.HEX generado por el MPLAB, por el ensamblado y/o compilado del código fuente. Diferentes herramientas del MPLAB generan el archivo *.HEX, y estas herramientas son parte de cada proyecto. El proyecto da la flexibilidad de poder utilizar diversas herramientas para crear el ejecutable *.HEX.

En la pantalla de EDICIÓN DEL PROYECTO se muestra que por defecto, el proyecto asignó el mismo nombre para el fichero ejecutable (en este caso ***tesis.hex)***. Así mismo, también se observa el MPLAB SIM 16F84 que es lo que se había seleccionado anteriormente en el menú OPTION, al igual que la suite de lenguajes (en este caso Microchip).

Si selecciona dentro de la parte Projects Files el titulo (tesis.hex), se habilitará el botón ***NODE PROPERTIES***. Este botón sirve para configurar y decirle al MPLAB IDE cómo crear el fichero *.HEX. Lo pulsa y se abre el cuadro de la figura 7.6:

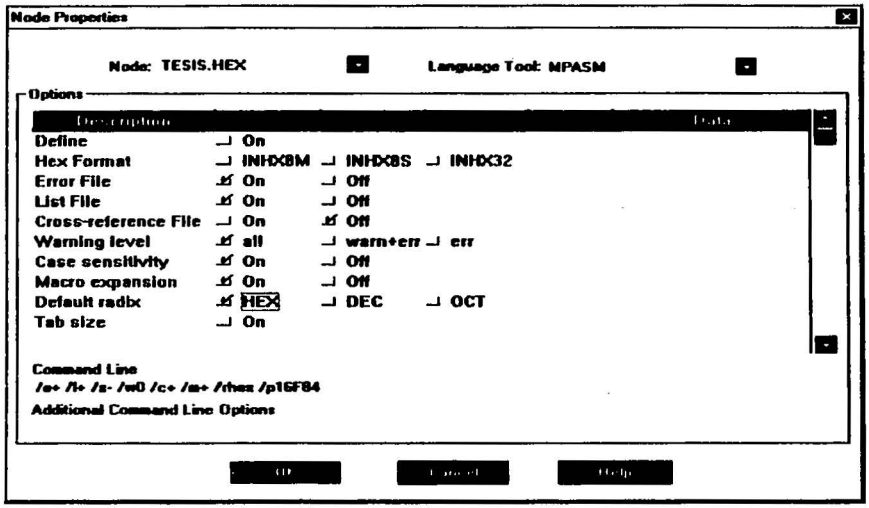



*FIGURA 7.6 CUADRO NODE PROPERTIES*

Como se observa, aparece en la esquina superior derecha el lenguaje utilizado; a la izquierda el nombre del fichero a crear, que por defecto sale, en este caso tesis.hex. Todas las filas y columnas dentro del cuadro, son switchs o llaves en las cuales se marcan las preferencias a utilizar. Por ejemplo la numeración utilizada (HEX = hexadecimal) si quiere utilizar macros, etc. Esto se traduce en una línea de comandos que se visualiza en la parte inferior de la pantalla, en ***Command Line***. Básicamente hay que marcarlo como se ve en la ventana de muestra, aunque no es obligatorio, pero solo se debe cambiar esta configuración una vez que conozca cada una de las funciones que se muestran en esta pantalla.

Ahora al dar clic en el botón OK y se muestra la ventana de edit project, se debe hacer clic en ADD NODE y se mostrará una ventana con la opción de guardar, siempre dentro del mismo directorio del proyecto, el archivo fuente con el mismo nombre del proyecto (en este caso se llama **tesis.*asm)***. Se da un clic en el botón ACEPTAR para volver otra vez al cuadro anterior:

Se ha añadido un nodo (según el nombre que le ponga) bajo el nombre del archivo con extensión .hex (figura 7.7).

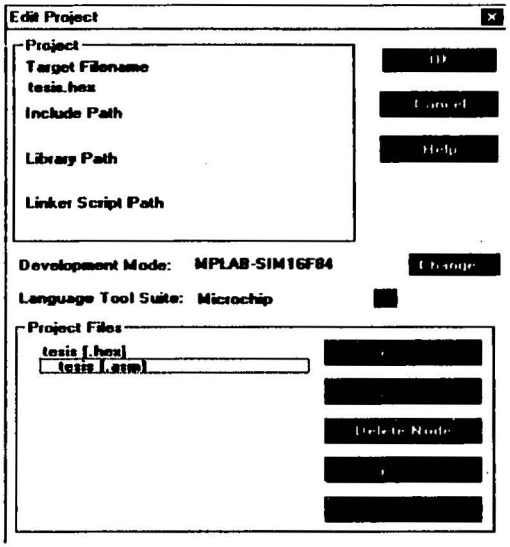



*FIGURA 7.7 CUADRO DE EDIT PROJECT*

Dar un clic en OK para volver al escritorio en donde tendrá abierta una ventana en donde introducirá el código, pero aún sin titulo. Para dar el titulo se selecciona en el menú ***FILE-SAVE AS*** y se le da el nombre del nodo que se va a configurar (para esta caso **tesis.*asm)***.

Se puede apreciar ahora la barra de titulo de la ventana de código a cambiado al nombre que se le asigne con extensión **.asm**. Puede ver que se coloca **el mismo nombre** en el proyecto y en el código fuente. Esto es porque no está trabajando con un linker que permite incorporar o linkear distintos archivos con distintos nombres. Entonces al trabajar con un archivo único debe nombrarlo igual que el proyecto al cual pertenece, además de guardarlo en el mismo directorio.

## TEMA 7.4. EL EDITOR MPLAB

El Editor de MPLAB permite a los programadores escribir y editar código fuente para las familias de microcontroladores PIC16/17, así como otros archivos de texto.

### 7.4.1. ESCRITURA DE PROGRAMAS EN ASM.

El ensamblado de un programa traduce las instrucciones que se han escrito utilizando los mnemónicos del lenguaje maquina (CALL, etc.), a código binario ejecutable por el microcontrolador. La ***secuencia de mnemónicos*** se llama ***listado*** o ***código fuente*** del programa, mientras que el ***código binario*** se llama ***objeto*** o ***ejecutable***.

Una vez que el programa se ha escrito y ensamblado (compilado no existe para el lenguaje ensamblador, solo se compilan los lenguajes de alto nivel como el C) ya tiene un binario ejecutable.

### 7.4.2. ESTRUCTURA DE UN PROGRAMA TÍPICO:[61]

Los programas desarrollados para el microprocesador PIC constan básicamente de la siguiente estructura (figura 7.8):

---

[61] Op cit [50]

- **Definir el Microcontrolador a usar:** En esta directiva se especifica el tipo de microcontrolador que será utilizado; esto se logra utilizando LIST = PIC16F84.

- **Establecer las constantes a utilizar:** Las constantes son útiles para referenciar direcciones de memoria, posiciones de los bits y valores que no cambiarán a lo largo de todo el programa. Esto se logra utilizando la directiva "EQU". Si se desea por ejemplo establecer un valor a una constante, sería CONST EQU 1, con lo cual le asigna el valor de "1" al registro CONST.

- **Reservar los espacios en memoria de las variables:** Se utiliza para reservar espacios en memorias para las variables utilizadas en el programa. Se logra utilizando la directiva denominada "**RES**". Muy útil cuando un registro cambia su valor durante la ejecución del programa.

- **Configurar los puertos:** Es necesario establecer si los bits de cada puerto serán utilizados como entrada o como salida. Para ello se utilizan los registros especiales TRISA y TRISB.

- **Desarrollar el Programa:** Es el verdadero corazón del trabajo que se realiza, puesto que aquí se desarrollan las rutinas que serán ejecutadas por el microcontrolador.


*FIGURA 7.8 EJEMPLO DE PROGRAMA PARA EL EDITOR DEL MPLAB*

En la figura 7.8 se observa un pequeño programa como ejemplo, para el microcontrolador PIC16F84 en la cual se ven zonas diferentes. La primera zona (no se consideran los renglones con el símbolo de ";" ) representa la definición del microcontrolador y la palabra de configuración, la segunda representa el establecimiento de las constantes, la tercera representa la reserva de memoria, la cuarta es donde realmente comienza el programa en sí y representa la configuración de los puertos y el cuerpo del programa.

En lo referentes a las columnas del programa:

- La primera de la izquierda son etiquetas que pueden tener cualquier nombre y sirven para referenciar direcciones a llamar o bifurcar/saltar. Puede cambiarle los nombres por cualquier otro que se le ocurra, siempre que también lo haga en la instrucción que llama a dichas etiquetas.
- En la segunda columna están las instrucciones o mnemónicos: goto, movlw, etc.
- En la tercera el operando, registro o llamada: c1, f, start, etc.
- Y por último en la cuarta, ***siempre*** después de un punto y coma, los comentarios, que sirven nada más que para ayudar a la memoria del programador o facilitar la comprensión del código por otra persona ajena al propio programador. No tienen utilidad alguna en el ensamblado, pero si mucha para los programadores.

En la ventana en blanco abierta en el proyecto que se ha definido, puede introducir el código fuente de la figura 7.8 como programa de ejemplo. Para escribir el código se utiliza la tecla ***TAB*** para desplazarse entre columnas.

Por último, recuerde que en los cuadros seleccionables del nodo tenía marcado ***CASE SENSITIVE*** lo cual quiere decir que el código debe ser escrito de una única forma; o mayúsculas o minúsculas, o se hará referencia a etiquetas diferentes. Lo que significa que ***start*** no es lo mismo que ***Start***. Guarde los cambios en el menú ***FILE SAVE***.

## TEMA 7.5. ENSAMBLADO

Nuevamente debe recordar que no se compila sino se ensambla, porque recuerde que solamente se compila desde lenguajes de alto nivel.

Cabe mencionar que si no se realiza el ensamblado correspondiente al programa, el MPLAB no permite realizar la simulación. Es por ello que se hace necesario tratar este tema de manera anticipada al de simulación.

### 7.5.1. EL ENSAMBLADOR MPASM[62]

El MPASM, que se incluye con el MPLAB, permite el ensamblado condicional, de diferentes fuentes y lista de formatos. MPASM permite generar varios formatos de código objeto que soportan las herramientas de desarrollo de Microchip así como los programadores relacionados con ellas sin salir de MPLAB.

Con el MPASM se procede a ensamblar un programa que haya creado o editado con el MPLAB. Este permite escribir el código y ensamblarlo para producir como salida un fichero *.HEX que servirá para grabarlo en el PIC o utilizarlo por el simulador.

Accediendo al menú **PROJECT - BUILD ALL** sale la ventana de ensamblado, la cual indica que se ha construido un archivo con extensión .HEX (lo que no quiere decir que funcione correctamente). Recuerde que no hace falta salvar los cambios en el código cuando se ensambla ya que se guardan los posibles cambios automáticamente.

La figura 7.9 muestra los resultados de ensamblado de un programa, si tuviera un error en esta ventana solo debe dar doble clic en la línea de Error y el cursor se posará en la línea del programa con el error, solo debe corregirlo. Cada vez que realice alguna corrección del programa fuente, debe acceder al menú **PROJECT - BUILD ALL** para ensamblarlo de nuevo.

---

[62] Op cit [50]

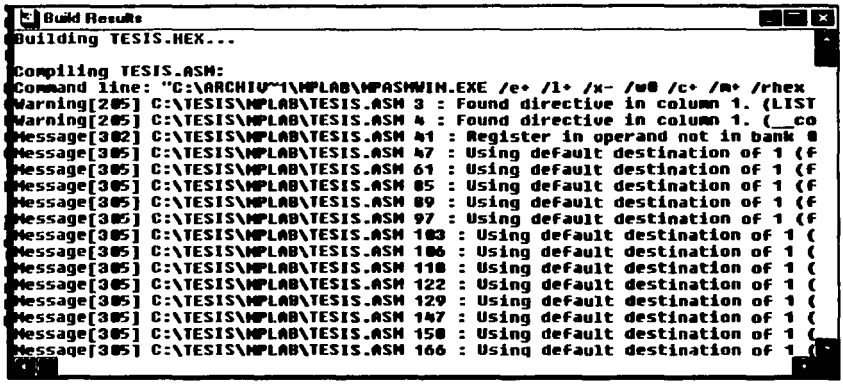



*FIGURA 7.9 VENTANA DE RESULTADOS DE COMPILACIÓN*

El mensaje Warning solo hace referencia a un aviso pero no afecta al programa. Si no se presentan errores al momento de ensamblar se creará, como ya se dijo, el archivo *.HEX que es el que se graba en el PIC. Si observa en el directorio donde guardó el proyecto, verá el archivo. En estos momentos ya podría utilizar este archivo para grabarlo en el PIC, pero antes necesita saber si funciona bien el programa, así que tendrá que simular antes para comprobarlo.

Como dato, respecto al ensamblado o compilado, cada vez que aparezca la ventana de la figura 7.9 e indique que ha tenido algún error "error" (uno o más), siempre hay que corregir el primero, porque el resto de los errores pueden producirse por culpa del primero, así que solucionando éste podrían desaparecer el resto o al menos alguno de ellos.

## TEMA 7.6. VENTANAS EN LA SIMULACIÓN

Existe la opción de ver los registros y más detalles concernientes al funcionamiento del programa, esto se encuentra básicamente en el menú Window.

### 7.6.1. MENÚ WINDOW

En este menú puede abrir la memoria de programa, la memoria EEPROM, el reloj interno del PIC, una ventana de observación de los registros que a Ud. le interesan, el ***WATCH WINDOW,*** etc.

Primeramente está **PROGRAM MEMORY**, la ventana de la memoria del programa puede visualizar localizaciones en el rango de la memoria del programa para el procesador actualmente seleccionado. Dicho de otra forma, muestra la posición de todos los registros con que cuenta el programa en curso y, conforme este se ejecuta, los valores de los registros van cambiando al valor asignado en el propio programa. Puede dejar esta ventana abierta siempre y mover y volver a clasificar según el tamaño dicha ventana.

La memoria del programa se puede visualizar de tres formas diferentes. El formato deseado se elige a través del menú del sistema.

- **Hex Code Display**: visualiza la memoria del programa como datos en hexadecimal. Esta opción es la más útil al usar un programador del dispositivo (es el código que se utiliza para grabar los PIC).
- **Machine Code Display**: visualiza el código desensamblado sin la información simbólica.
- **Disassembly Display**: visualiza el código desensamblado con símbolos o sea, con los nombres que se han asignado en el programa.

La ventana de la memoria del programa (figura 7.10) está solamente disponible en modo emulador y en modo simulador.

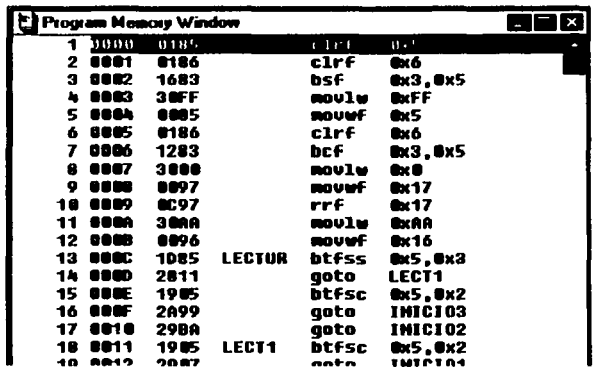



*FIGURA 7.10 VENTANA DE MEMORIA DE PROGRAMA*

La opción **EEPROM MEMORY** visualiza la ventana que se muestra en la figura 7.11 en la que se observa la memoria de datos de la EEPROM para un microcontrolador que tenga dicha memoria. El PIC16F84 es un ejemplo de un dispositivo que utilice memoria EEPROM. La ventana puede estar abierta siempre, moverse por la pantalla y recolocarla según el tamaño. Está única y exclusivamente para información y no podrá cambiar los valores visualizados, a menos que en el programa tenga una rutina para escribir en esta memoria.

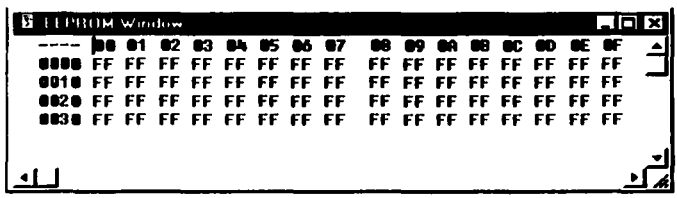



*FIGURA 7.11 VENTANA DE MEMORIA DE DATOS DE EEPROM*

Con **ABSOLUTE LISTING** puede observar todos los datos de direcciones, mnemónicos, mensajes de compilación y demás respecto al programa que se está simulando

Con la opción **STACK** se abre una ventana que visualiza el contenido de la pila, las direcciones que se guardan en ésta. El número de niveles disponibles depende del tipo del procesador que emule, pero en este caso es el microcontrolador PIC16F84 el cual contienen 8 niveles en el stack. La ventana de la pila puede estar abierta siempre, moverla y colocada de nuevo según su tamaño. El contenido de la pila se puede visualizar con o sin números de línea. El formato deseado se elige a través del menú del sistema.

Con **SPECIAL FUNCTION REGISTERS** se permite visualizar el contenido de los registros especiales de la función (SFRs) para el procesador que emule.

Realmente de poco sirve ver una simulación si no se muestra ninguna información sobre los registros. Para eso está el menú **WINDOWS – SPECIAL FUNCTION REGISTER** el cual abrirá la ventana de registros especiales o específicos del PIC16F84 (figura 7.12) que ya debe estar configurado en **DEVELOPMENT MODE**.

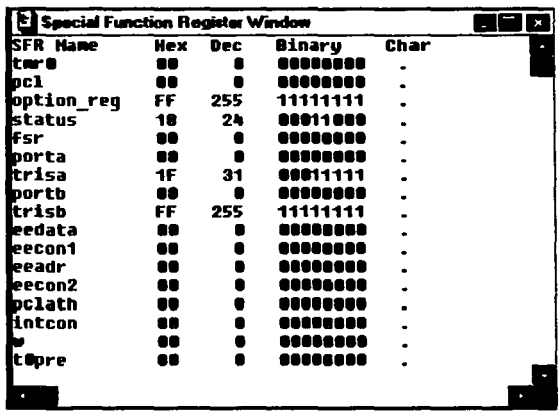



*FIGURA 7.12 VENTANA DE REGISTROS ESPECIALES*

En esta ventana se ven todos los registros especiales del microcontrolador. También se ve que cuando cambian de valor, según el programa simulado, cambian de color azul a rojo.

La opción **FILE REGISTER** sirve para ver absolutamente todos los registros. Tiene que ir al menú **WINDOWS – FILE REGISTER**. Así se abrirá una ventana en hexadecimal. Sobre la barra de titulo de esta nueva ventana, verá que a la izquierda del titulo hay un icono que son tres hojas superpuestas. Si hace clic con el puntero del ratón encima, se desplegará un menú, en donde podrá elegir " **Symbolic Display**" y aparecerá la ventana que se muestra en la figura 7.13:

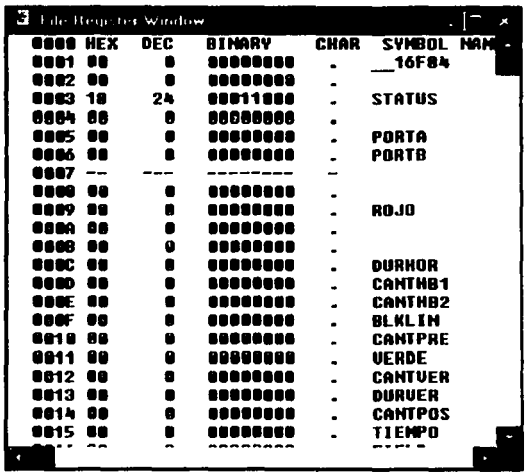



*FIGURA 7.13 VENTANA DE REGISTROS GENERALES*

Se muestran todos los registros del microcontrolador, tanto los especiales como los de uso general, incluso los que se renombran en el programa. Hasta aquí todo correcto. Pero para el caso de que el programa sea mucho muy grande, ver esta ventana se hace muy complicado.

Para buscar una solución a esto, hay otra ventana "especial" llamada " ***Watch Window***" y que se guardará con el proyecto.

En el menú ***window - watch window - new watch window*** aparecerá una ventana en blanco y al mismo tiempo sobre esta última, la ventana ***Add Watch Symbol***, ya que no hay todavía ningún símbolo cargado. Escriba en el cuadro de diálogo el registro que desea ver.

Al dar clic en el botón ***Add*** tal y como está ahora se mostrará por defecto en hexadecimal. Si desea verlo en otro formato (decimal, binario, etc.), antes de presionar ***Add*** pulse en ***Properties*** y seleccione el de su gusto.

Y para finalizar con la pantalla de añadir símbolos pinche en ***Close***. Ahora ya tiene la ventana con los registros que realmente le interesa ver (figura 7.14). Para agregar o editar registros se hace a través del icono de la barra de título de la ventana ***watch window*** o a través del menú ***window - watch window*** en donde se mostrará ***add active watch, edit active***...etc., haciendo referencia a **active** porque es posible tener más de una ventana ***watch window***.

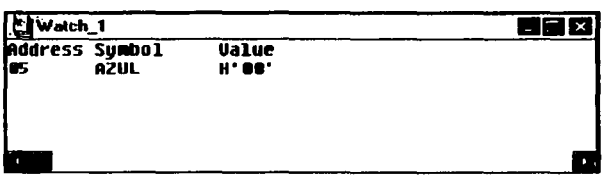



*FIGURA 7.14 VENTANA DE WATCH*

Para guardar la ventana de ***watch window*** y no tener que construirla nuevamente cada vez que se desea utilizar, se hace con ***save watch window***. Pero si cierra el proyecto, se le preguntará si quiere guardar los cambios. Al responder que si se guardará automáticamente la ventana con el nombre por defecto ***watch_1.wdt***.

La opción **SHOW SYMBOL LIST** visualiza los símbolos del código fuente. Estos símbolos son *.COD en el proyecto e incluyen todos los símbolos conocidos en MPLAB. Los símbolos (figura 7.15) incluyen constantes y escrituras de la etiqueta. La lista de símbolo de es un rectángulo de diálogo de información únicamente.

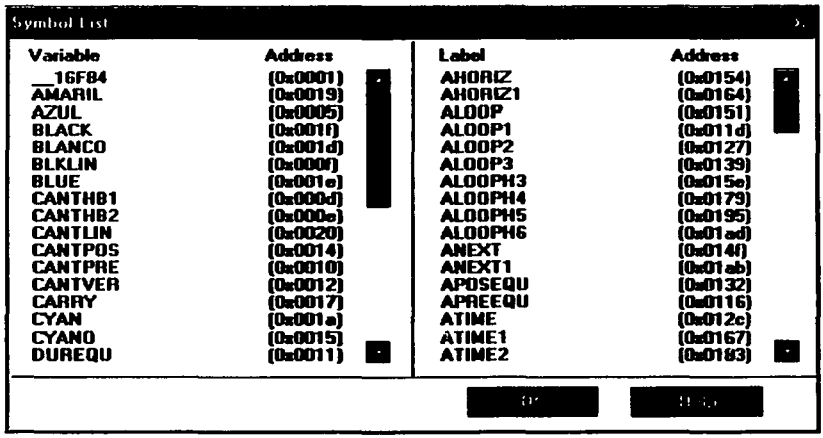



*FIGURA 7.15 VENTANA SHOW SYMBOL LIST*

Con **STOPWATCH** se visualiza el valor actual del contador de ciclo. El cronómetro del sistema cuenta el número de los ciclos de reloj que el procesador ejecuta. El cronómetro permite medir el tiempo de ejecución del código. Éste se calcula basándose en la frecuencia de reloj del PIC (ver figura 7.16).

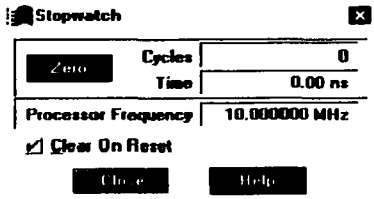



*FIGURA 7.16 VISUALIZACIÒN DEL STOPWATCH*

Con **PROJECT** visualiza la ventana del proyecto si éste está abierto. Visualiza la lista de ficheros actualmente en el proyecto. Si se ha compilado el proyecto, enseña una lista de todos los ficheros incluidos en el proyecto. Si no, la ventana muestra solamente el fichero del proyecto principal.

Con **ABSOLUTE LISTING** se muestra toda la información relacionada con el ensamblado del programa.

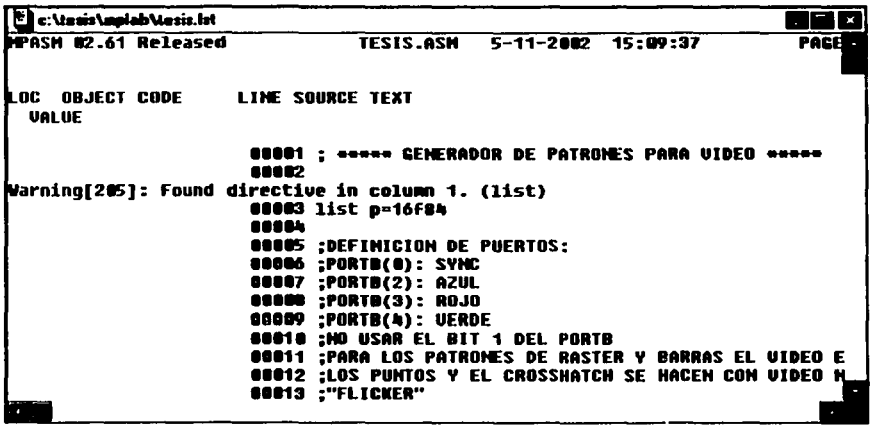



*FIGURA 7.17 VENTANA ABSOLUTE LISTING*

En cuanto a las opciones **TILE HORIZONTAL, TILE VERTICAL, CASCADE, ICONIZE ALL** y **ARRANGE ICONS,** solo sirven para organizar las ventanas en el proyecto que se encuentra abierto, tal y como sucede con las ventanas de Windows.

## TEMA 7.7. EL SIMULADOR MPSIM

El simulador MPSIM permite aislar problemas de código y depurar diseños. Simula las funciones principales así como la mayoría de los periféricos de las familias de microcontroladores PIC16/17. Este se incluye con el software MPLAB.

### 7.7.1. SIMULACIÓN[63]

Ahora llegó el momento de simular, depurar o debuguear (hace referencia a eliminar gusanos "bugs"). Si el proyecto a depurar está cerrado, se abre desde el menú **project** y seleccione **open project**, elija el proyecto que desee depurar. Recordar que si desea guardar el proyecto, debe hacerlo en el menú **project** y después **close project**.

Es la única forma de cerrarlo. No es suficiente con cerrar la ventana de código para cerrar el proyecto, eso lo puede observar cuando la cierre y se siga viendo en la barra de título el proyecto.

La primera vez que guarda o cierra el proyecto, saldrá la pregunta si quiere guardar los cambios hechos, y estos cambios incluyen hasta las posiciones de las ventanas de código en el área de trabajo, y así cuando lo abra la próxima vez, aparecerán las ventanas y demás partes integrantes del mismo, en las mismas posiciones donde estaban antes de guardarlo.

Una vez abierto el proyecto y con el cursor en la ventana de código, en el menú **DEBUG – RUN - RESET** se inicializa el sistema. Una vez inicializado, vea que está situada la pantalla en subrayado negro, en la primera instrucción que va a ejecutar el PIC.

La línea que aparece resaltada es la que se va a ejecutar, **NO** que ya está ejecutada. Para volver a simular o regresar al punto de partida, de nuevo se pulsa **RESET**.

---

[63] http://www.superpic.com/MPLAB.htm, (España, 2002)

Una vez sobre la primera instrucción a ejecutar (lo que hay arriba son directivas para el ensamblador y no forman parte del código asm) debe ir al menú **DEBUG – RUN - STEP** o desde el acceso directo que está en el botón que tiene el dibujo de dos pies. Este es el botón para ejecutar paso a paso. A la derecha hay otro igual pero con una línea entre los dos pies. Esta función se llama **STEP OVER** y sirve para ejecutar paso a paso igual que el anterior, pero si encuentra algún CALL lo ejecuta todo de golpe como si de una sola línea se tratara. También se puede acceder a **STEP** a través de la tecla **F7**.

Si observa la barra de status, verá que ha cambiado el valor del **Program Counter (PC)**. Si sigue presionando **F7** repetidamente, podrá observar como se ejecuta el proceso paso a paso.

Para hacer que funcione a tiempo real (tiempo real para el programa de simulación, no la velocidad real del pic durante la simulación). Debe ir nuevamente al menú **DEBUG – RUN - RUN** o pulse el botón del semáforo verde o **F9**, y verá como la barra de estado cambia su color a amarillo. Esto indica que el programa está funcionando en tiempo real. Para detener el proceso presione el semáforo rojo, o **F5**. Se detendrá donde estaba ejecutándose en el momento de pulsar el semáforo o la tecla **F5**.

También está la función **ANIMATE** que no es nada más que un **AUTO STEP**. La puede seleccionar en el menú **DEBUG – RUN - ANIMATE** o **control F9**, y verá en forma automática como se ejecutan las líneas. Lo puede detener también con **F5**.

#### *7.7.1.1. MENÚ DEBUG:*

Aunque ya se ha descrito la forma de simular un programa, a continuación se presenta una descripción más detallada del menú correspondiente a la simulación.

El menú **DEBUG**, que se muestra en la figura 7.18, es el que se utiliza para simular un programa, depurarlo, colocar break points, correr el programa paso a paso, hacer un reset, etc.

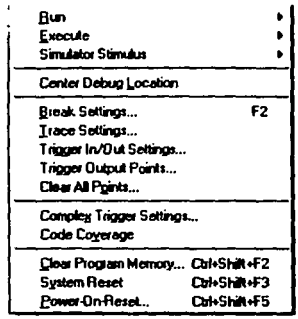



*FIGURA 7.18 MENÚ DEBUG*

En el submenú **RUN** están varias opciones:

- **RUN** que hecha a andar el programa a toda velocidad, o sea que es solo para cuando hay algunos breakpoints colocados y poder detenerlo.
- **RESET** resetea el programa y comienza nuevamente desde el principio.
- **HALT** detiene el programa.
- **HALT TRACE** detiene un traceo que se esté haciendo.
- **ANIMATE** es igual a la ejecución paso a paso pero en forma automática, algo así como autostep de cualquier debuger.
- **STEP** ejecuta un paso sólo [STEP OVER], pero que no salta dentro de ningún CALL, únicamente lo ejecuta todo de golpe.
- **UPDATE ALL REGISTER** actualiza el valor de todos los registros.
- **CHANGE PROGRAM COUNTER** cambia el valor del ProgramCounter y además se cambiará la posición del programa.

- **EXECUTE** permite ejecutar un código de operación (OpCode), en forma totalmente independiente a la secuencia del programa. Puede hacer por ejemplo un GOTO a un lado, ejecutarlo y listo. En la ventana desplegable se guardan los últimos códigos ejecutados para no tener que escribirlos de nuevo.

- **CONDITIONAL BREAK** ejecuta un breakpoint[64] en forma condicional. Hay muchas formas de ejecutar un breakpoint. Por ejemplo: parar cuando una línea a sido ejecutada un número de veces determinado, o sólo porque se llego hasta allí, o cuando un registro adquiere determinado valor, etc.

El submenú **SIMULATOR STIMULUS** permite que instale señales de estímulo del reloj y que el simulador responda a los acontecimientos de ficheros en el PC. Los ficheros se pueden escribir con el editor de MPLAB, o cualquier otro editor o procesador de textos conveniente, y se deben guardar en el mismo directorio que el proyecto actual.

El simulador de estímulos genera las señales para el simulador. Puede fijar los contactos altos o bajos, e inyecta valores directamente en los registros. Hay cuatro tipos de estímulos:

- Diálogo de estímulo asíncrono (un diálogo interactivo con las señales de control en los contactos de la entrada de información). Los estímulos del tipo **ASYNC STIMULUS** permiten simular la entrada de las señales a determinados pines o puertos de distintas formas.

En la figura 7.19 se muestra la pantalla de estímulo asíncrono, observe que el primer botón está asignado a la terminal 1 del puerto A (RA1) y la T significa que el tipo de estímulo es alto cada vez que se oprima. Estos botones se configuran con el botón derecho del mouse encima de cualquiera de los botones que se muestran.

---

[64] Para mayores detalles diríjase al subcapítulo 7.7.1.2 BREAK POINTS O PUNTOS DE PARADA

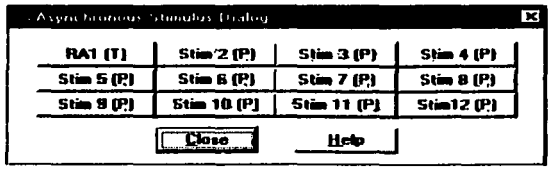



*FIGURA 7.19 VENTANA DE SIMULACIÓN DE PUERTOS*

- El estímulo Pin (el contenido de un fichero del texto describe señales de entrada en los contactos). También llamados Synchronous Stimulus, permiten en forma automática simular el estado de los pines de los puertos. Consiste en crear un archivo de texto con la extensión *.STI en el cual en la primer columna van las posiciones del programa donde quiere simular esas entradas a los pines, y en la primer fila los pines a los cuales va a "estimular". Todo esto es para poder simular un puerto como si fuese en la vida real.

- El registros de estímulos (el contenido de un fichero del texto se utiliza para fijar valores 8-bit directamente en un registro). El **REGISTER STIMULUS** es exactamente lo mismo que con los pines, solo que lleva la extensión *.REG y sirve que para una determinada posición del programa se cargue un registro con el valor que le interese.

- Estímulo del reloj (una fuente regular, programable y periódica de los impulsos). El **CLOCK STIMULUS** hace que para un determinado rango del reloj, un pin tome un valor determinado, etc.

La opción **CENTER DEBUG LOCATION** en el menú debug sirve para posicionar el cursor en el lugar donde está la ejecución del programa por si no se sabe donde está la línea en ejecución.

En cuanto a **TRIGGER IN/OUT SETTINGS** y a **TRIGGER OUTPUT POINTS** son opciones que no se tratarán en estos temas, debido a que se hace necesario el uso del emulador del MPLAB.

Por otro lado **CLEAR ALL POINTS** limpia todos los break/trace points. Mientras que **COMPLEX TRIGGER SETTINGS** es para configurar las funciones complejas del emulador (lo cual no se trata dentro de estos temas).

En cuanto a **CLEAR PROGRAM MEMORY** es para borrar la memoria de programa. La memoria de programa es el buffer que hay en la maquina para poder almacenar un programa propiamente dicho para ser simulado o programado[65].

El **SYSTEM RESET** es para hacer un reset del sistema como si comenzara el programa.

En **POWER ON RESET** se meten datos aleatorios como los que podrían ser en la realidad cuando se produce un PowerOnReset, y ver de esta forma el comportamiento del programa.

#### *7.7.1.2. BREAK POINTS O PUNTOS DE PARADA*

Los breakpoints sirven para detener la simulación del procesador en el punto del programa donde se indique.

Para ello se debe situar en la ventana de código, y colocar el cursor sobre la instrucción deseada y una vez allí pinchar con el botón derecho del ratón para abrir el menú contextual, donde se elige ***Break Point(s).*** Verá que la línea cambia de color (a rojo), indicando que hay un ***break point*** en dicha línea.

[65] El MPLAB usa el PICMASTER para grabar la memoria de los PIC's, en el caso de esta tesis no es así, para esto se usará el software NOPPP según se describe en el Capitulo 8

Tiene que asegurarse que en la barra de estado (en la parte de abajo) en donde dice ***Bk*** esté en ***ON***. Si está en ***OFF*** nunca se detendrá en ningún break point. Para cambiarlo se da doble clic ahí mismo y se cambiará de un estado a otro. También podrá hacerlo en el menú ***option - development mode*** y pinchar en la solapa ***Break Point*** y marcar la casilla ***Global Break Enable***.

Al hacer de nuevo reset del sistema, presionando ***F9*** verá como la simulación del programa se detiene cada vez que llegue al ***break point***.

También hay muchas otras opciones, como por ejemplo ***RUN TO HERE***, que hace que se ejecute el código hasta allí sin necesidad de colocar un ***break point***.

Los ***Break Points*** se pueden colocar en la ventana de código, en ***window - Program Memory*** (es el buffer que se mencionó antes) o en ***window - Absolute Listing***.

***Source Files***: si va al menú ***window - Project Window*** se abrirá una ventana en donde se muestran todos los archivos que forman parte del proyecto. Haciendo clic en ellos se abrirá el editor del mismo.

En el menú PROJECT se encuentran opciones que se describen a continuación:

- **OPEN PROJECT** con el que se abre un proyecto existente.
- **CLOSE PROJECT** para cerrar un proyecto.
- **SAVE PROJECT** para guardar un proyecto.

El submenú **EDIT PROJECT** es muy importante, como verá sirve para seleccionar la herramienta de desarrollo, los path, los nodos, el target name, el linker, las librerías, etc.

El submenú **MAKE PROJECT** ensambla el proyecto. No se refiere al programa, dado que el programa puede estar conformado por partes de librerías de rutinas aparte de las líneas de código propias. Cuando se ensambla, se genera un archivo con extensión *.HEX, y es el que se usa para grabar el PIC.

- **BUILD ALL** no tiene en cuenta ninguna fecha al pasar a ensamblador.
- **BUILD NODE** convierte a ensamblador algún nodo que haya asociado al proyecto. Únicamente ensamblará el nodo.
- **INSTALL LANGUAGE TOOL** permite configurar el lenguaje que va a utilizar, que en este caso es el **ASM** de Microchip (figura 7.20). Obviamente, tiene muchas funciones, incluso el C .

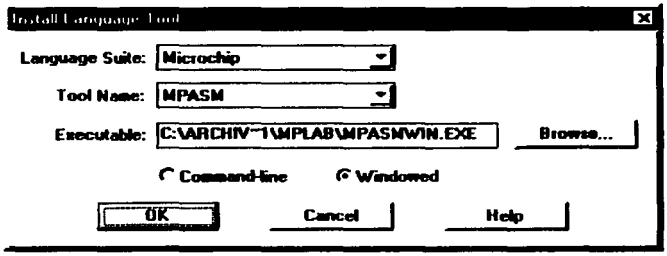



*FIGURA 7.20 VENTANA INSTALL LANGUAGE TOOL*

Y por último tiene el acceso directo a los últimos proyecto abiertos (en el caso de haber abierto alguno anteriormente).

El simulador o debuger[66] para poder funcionar, utiliza el archivo *.HEX generado a partir de una compilación o ensamblado del código fuente. Además este mismo archivo generado (*.HEX) es el que se usa para grabar el PIC. El ensamblador produce otros ficheros aparte del *.HEX para otros usos que no son objeto de estos temas.

#### *7.7.1.3. MENÚ EDIT.*

En el menú **EDIT** puede hacer todo lo referente a las acciones de tipo texto como:

- **UNDO** deshace la última acción.
- **CUT** corta lo que tenga seleccionado.
- **COPY** copia lo que tenga seleccionado.
- **PASTE** pega lo que había cortado o copiado anteriormente.
- **SELECT ALL** selecciona todo.
- **SELECT WORD** selecciona la palabra sobre la que esta el cursor.
- **DELETE LINE** borra la línea sobre la que esta el cursor.
- **DELETE EOL** borra desde donde esta el cursor hasta el final de la línea.
- **GOTO LINE** desplaza el cursor a una línea determinada.
- **FIND** busca algún texto que tenga dentro del programa.
- **REPLACE** reemplaza el texto que anteriormente buscó.

---

[66] Está palabra hace referencia a la palabra del inglés BUG que en español significa Gusano. Básicamente, **debug se refiere** a la depuración de los programas, es algo como quitar los gusanos dentro de él.

- **REPEAT FIND** repite la búsqueda si es que hay otra palabra igual.
- **REPEAT REPLACE** repite el reemplazo del texto.
- **MATCH BRACE** localiza y coloca el cursor al principio o final de un emparejamiento de código.

El submenú **TEMPLATE** sirve para crear secciones de texto que sirven para incluir en el programa varias veces sin necesidad de teclear de nuevo lo mismo.

Dentro del submenú **TEXT** está la opción **TRANSPOSE** que sirve para intercambiar entre si dos letras, las cuales están una a cada lado del cursor. Por ejemplo: si el cursor está entre las letras A y F de la palabra AFTER, al hacer TRANSPOSE quedará FATER.

Las demás opciones son:

- **UPPER CASE** hará mayúsculas todas las letras seleccionadas.
- **LOWER CASE** hará minúsculas todas las letras seleccionadas.
- **INDENT** mantiene la sangría de márgenes de la línea anterior.
- **UN-INDENT** no mantiene la sangría de márgenes de la línea anterior.

Hasta aquí lo que de manera básica es útil para comenzar un programa y depurarlo, esto no quiere decir que solo esto sirva del MPLAB, solo significa que es lo esencial. El resto de los menús y submenús, de ser necesario se pueden aprender con un poco de práctica. Además, es muy importante la destreza de quien realiza el programa pues ello es esencial para un buen funcionamiento del programa a ensamblar.

# CAPITULO 8

# SOFTWARE Y HARDWARE

# "NOPPP"

# CAPITULO 8. SOFTWARE Y HARDWARE "NOPPP"

El software NOPPP, (por sus siglas en inglés No Piece Programar PIC o programador para PIC's sin piezas, haciendo alusión a que requiere de muy pocos componentes de hardware), es un software muy simple y efectivo para cargar los programas a diferentes tipos de PIC's entre los que se encuentran los PIC16C83, 16C84 y el que conciernen a esta tesis, es decir, el PIC16F84

El hardware que requiere el NOPPP (figura 8.2) es inusualmente simple y utiliza componentes fácilmente localizables y por lo tanto su costo es muy bajo, motivos por los cuales se ha preferido la utilización de este software a otros que sirven para el mismo caso. En forma básica solo requiere de dos diodos rápidos de señal, un transistor y cuatro resistencias. Su fuente de alimentación debe ser regulada doble de CD de 13 V y 5 V (aproximadamente), lo que se puede lograr solo con un regulador de 12 volts y uno de cinco así como de un par de diodos. Por último se requiere de un conector para el puerto paralelo de la PC. Sin embargo cabe mencionar que posteriormente se añadirían algunos componentes extras al circuito, sin que ello afecte su funcionamiento.

## TEMA 8.1. SEÑALES PARA LA PROGRAMACIÓN DEL PIC16F84[67]

La programación del PIC16F84 se realiza a través de una PC, un software y un hardware simples NOPPP(No Piece Programmer PIC).

[67] "Todo sobre PICs & microprocesadores y microcontroladores", Vallejo Horacio D, H. Picerno Alberto, Prado Federico, Rodríguez Luis H., Quark S.R.L., 2001, Argentina, Pag 64

El PIC tienen una pata que predispone el dispositivo para leer o para escribir. Si la pata 4 del PIC está a un potencial comprendido entre 13 y 14 V (aprox. 13.4 V), el PIC está preparado para escribir los datos que provienen desde la PC. Si la pata 4 está por debajo de 6 V, el dispositivo está previsto para ser leído. Los datos a leer o escribir se ponen/obtienen de la pata 13 del PIC, con la pata 12 del mismo que opera como clock.

Un PIC se lee/escribe accediendo a las diferentes posiciones de memoria por la pata por la que se obtienen/ingresan los datos (pata 13). La señal primero elige la posición de memoria a ser leída/escrita, y en cuanto esa posición está accesible se escriben/leen los datos. El clock que se coloca en la pata 12 del PIC sirve para indicar en qué momento se debe transferir la información. Los datos pueden estar sobre la pata 13 todo el tiempo que se desee, ya que no serán escritos ni leídos por la PC hasta que se produzca un cambio de estado (de alto a bajo) en la pata 12.

Primeramente se genera una serie de 6 bits que indican la posición de la memoria a ser leída/escrita, posteriormente se genera un estado de alta impedancia en la línea de datos en la que el clock está apagado. Por último ingresan/egresan los datos con el clock encendido. La secuencia descrita debe seguirse invariablemente. Dicha secuencia se muestra de forma gráfica a continuación en la figura 8.1

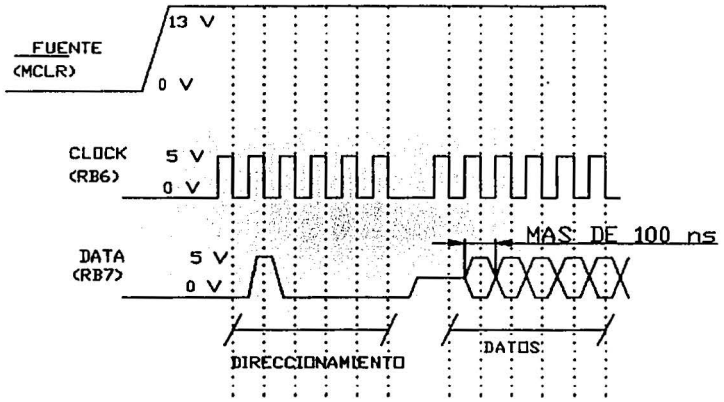



*FIGURA 8.1 SEÑALES PARA LA LECTURA/ESCRITURA DEL PIC16F84*

Esta figura es muy básica pero muy ilustrativa, pero en caso de requerir mayor detalle al respecto puede obtener los datos necesarios en la página web de Microchip..

## TEMA 8.2. DESCRIPCIÓN DEL HARDWARE[68]

En el PIC16F84, el pin $\overline{\text{MCLR}}$ se pone a +5V para el funcionamiento normal pero debe ser de +13.2V (aprox.) para grabación, y a 0V para resetear. Realmente los +13.2 V no "queman una EEPROM", el voltaje superior es meramente una señal para activar el circuito interno de programación de la memoria flash. La salida D0 de la PC controla esta señal. No hay peligro para el chip si se aplica esta señal en un momento inadecuado.

[68] Op cit [67] Pag 66

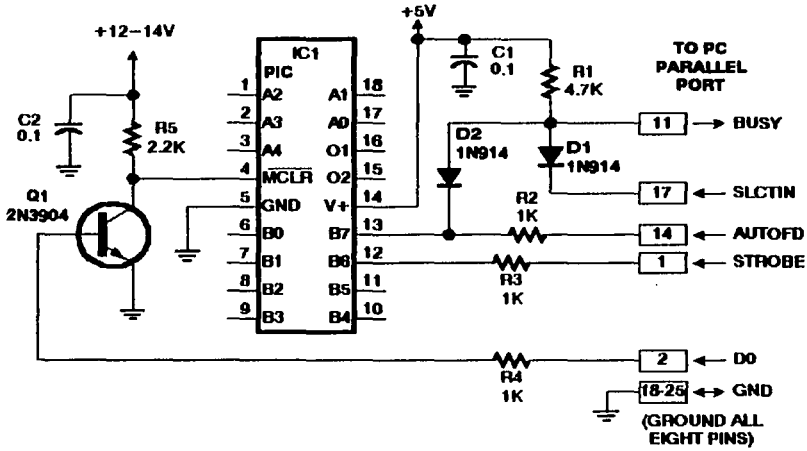



*FIGURA 8.2 CIRCUITO NOPPP* [60]

El PIC se comunica mediante protocolo serie síncrono de dos líneas (más masa). El Pin B6 es la señal de strobe; los pulsos en este pin le indican al PIC cuando debe recibir o transmitir cada bit de datos. El Pin B7 se utiliza como entrada y salida. Cuando el PIC está recibiendo datos desde la PC, la señal SLCTIN es mantenida a nivel bajo y por lo tanto D2 no conduce por lo que D1 y R1 no se utilizan en este momento y el PIC recibe los datos mediante la señal AUTOFD.

[60] Los números de Pin corresponden al conector de 25 pines en la PC y los condensadores están en microfaradios.

Cuando el PIC está enviando datos, las señales SLCTIN y AUTOFD (figura 8.2) están a nivel alto, D1 no conduce y D2 y R1 proporcionan la polarización (pull-up). La resistencia R2 más la resistencia interna de la línea AUTOFD (dentro del puerto de la PC normalmente 4.7k, aunque a veces mucho menos en los nuevos puertos paralelos CMOS) proporcionan algo de Pull-up adicional. La PC lee la información a través de la línea BUSY, que es 0.6V mayor que la salida del PIC debido al diodo D2. El puerto paralelo de la PC tiene (o debería tener) entradas CMOS o Schmitt y no debería necesitar verdaderos niveles lógicos TTL.

R2 y R3 ayudan a reducir las interferencias aislando la capacidad de entrada del PIC, de modo que circule menos corriente durante transiciones bruscas. El PIC tiene entradas del tipo Schmitt, que no impiden la reducción del tiempo de subida (rise time). R4 protege la base de Q1.

La fuente de alimentación puede ser como la que se muestra en la figura 8.3, considerando los rangos de voltaje de entrada del regulador de 12 volts que Ud. compre.

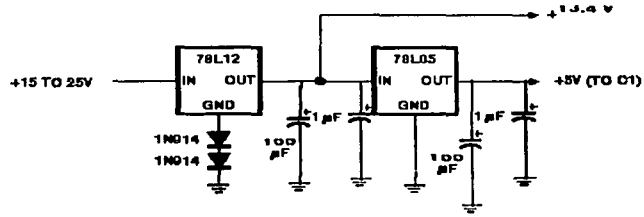



*FIGURA 8.3 FUENTE PARA EL HARDWARE NOPPP*

Los diodos provocan que la masa (tierra) del regulador de 12 volts caiga en aproximadamente 1.4 volts, provocando a la salida del mismo un voltaje cercano a los 13.4 V. Por otro lado los capacitores están con la finalidad de reducir el ruido a la salida de la fuente. Por último puede colocar LED's con su respectiva resistencia a la salida de cada alimentación con la finalidad de poder verificar rápidamente si hay o no voltaje de salida.

## TEMA 8.3. EL PAPEL DEL NOPPP EN EL ENSAMBLADO

Como ya se vio anteriormente, las características de hardware del NOPPP describe que el sistema a ocupar en el PIC no es de costo alto, ni mucho menos de conocimiento extenso sobre microcontroladores, lo único que queda por dejar claro en esta pequeña parte introductoria al uso del NOPPP, es saber en donde existe la relación de Ensamblado – Cargado en el PIC, para poder realizar cualquier proyecto.

Cuando se necesita ensamblar, para empezar basta con el uso del NOPPP y con el MPLAB[70], que son dos programas sencillos y que gozan de ser gratuitos y tan livianos que entran en cualquier disco duro de las PCs, pues no ocupan mucho espacio.

Al MPLAB junto con el MPASM lo conoce por medio de una breve introducción en capítulos anteriores (por parte de los autores de esta tesis, se piensa que es suficiente para entender su uso conjunto con el NOPPP). Mientras que el NOPPP es el software que hace funcionar a nuestro cargador de PIC's.

¿Qué idioma entiende el PIC?, éste entiende un solo idioma, que es el Binario que consta de unos y ceros, el PIC maneja un idioma con un total de 35 palabras de 14 bits, donde los primeros 6 bits indican el tipo de operación a realizar y los últimos 8 bits indican las variables o números involucrados en la operación a realizar, parece mentira que con tan pocas palabras se pueda hacer tanto. Por ejemplo una sentencia completa utilizada para rotar los bits de una palabra seria:

00110110001011

[70] Vease el Capitulo 7 correspondiente al MPLAB para un mejor conocimiento.

Donde los últimos 8 dígitos dependen de la variable considerada, si bien el PIC solo conoce el código binario, el NOPPP entiende también el código hexadecimal, que es algo muy simple de escribir, por lo que se puede considerar que el NOPPP traduce el código hexadecimal en código binario o código fuente del PIC.

Imagínese el esfuerzo que significa programar recordando estas 35 sentencias en código binario, que se hace más simple cuando programa por medio de una clave más sencilla, o también llamados mnemónicos, lo que hace que cuando programe utilice mejor este lenguaje. A este sencillo lenguaje, que se describió en capítulos anteriores, conocido como lenguaje ensamblador o en ingles "assembler", y que en este caso para el PIC16F84 se ha utilizado el MPLAB en forma conjunta y automática con el MPASM.

## TEMA 8.4. USO DEL SOFTWARE DEL NOPPP[71]

Para poder empezar físicamente utilizando el NOPPP debe de tener en su poder una copia del software, mismo que puede descargar de la web[72], y que cuando lo tenga en su poder debe usted instalarlo. El NOPPP utiliza el sistema DOS para su ejecución.

Cuando conecte el hardware NOPPP al puerto paralelo de la PC, con la fuente de alimentación del mismo desconectada, debe cuidar que el PIC no esté en la base porta chip, si lo está debe retirarlo con la fuente apagada. Por otro lado deberá ejecutar el programa NOPPP desde su computadora donde aparecerá una pantalla similar a la figura 8.4:

---

[71] Op cit [67] Pags 75-78

[72] http://www.covingtoninnovations.com/noppp/noppp-sp.html , U.S.A, 2002

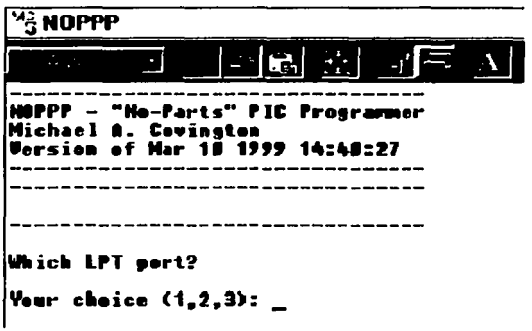



*FIGURA 8.4 PANTALLA INICIAL DEL NOPPP*

Esta es la pantalla inicial la cual pregunta en que puerto esta conectado el hardware del NOPPP, y da las opciones 1, 2 y 3. En la mayoría de las máquinas (salvo que se haya predispuesto lo contrario o que existan mas de una salida en paralelo) siempre se usa el puerto 1. Así que al pulsar la tecla número 1 se pasa a la siguiente pantalla. Posiblemente antes de presentar la siguiente pantalla, se muestre una recomendación del NOPPP, (dependiendo de la versión que sea) de que, si cuenta con un solo puerto, puede configurar el programa para que por defualt la elección sea el puerto 1[73]. Dependiendo del usuario se tomará la anterior modificación, por lo que ahora se presenta la siguiente pantalla (figura 8.5), en donde se confirma que se está utilizando el puerto 1, además de que el programa invita a conectar la alimentación a 5V, pero manteniendo el PIC sin ubicar en el zócalo.

[73] Por medio de la modificación del archivo *autoexec.bat* en la carpeta del NOPPP, con la instrucción SET PPLPT=1.

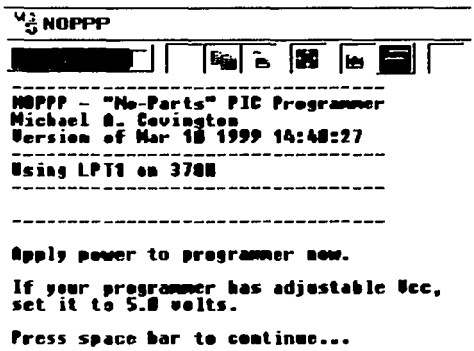



*FIGURA 8.5 PANTALLA DE APLICACIÓN DE ALIMENTACIÓN AL NOPPP*

Luego en la parte de abajo de la pantalla puede observar que debe apretar la barra espaciadora para ir a la siguiente pantalla, que se muestra en la figura 8.6. Aquí posiblemente aparecerá una pantalla indicando que no se encuentra conectado el hardware, si está conectado se debe proceder a realizar un test, opción que se elige en la pantalla mostrada en la figura 8.6.

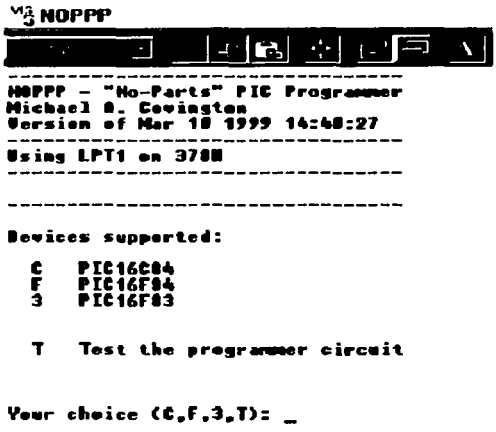



*FIGURA 8.6 PANTALLA PARA SELECCIÓN DEL PIC*

En esta pantalla el programa indica que solo soporta tres tipos diferentes de PICs e invita a optar por uno de los tres pulsando las teclas C, F ó 3, (en el caso de está tesis la opción será F). También se puede hacer una verificación del funcionamiento del cargador pulsando la tecla T. Esto es lo primero que se debe realizar, un test del hardware NOPPP

### 8.4.1. POSIBLES FALLAS QUE SE PUEDEN PRESENTAR

En esta parte se verán los problemas que se pueden presentar al cargar el PIC, pueden ser de software, por ejemplo cuando el archivo que tiene disponible no es el que entiende el cargador de PIC's, (los únicos archivos que sirven directamente son los que tienen extensión .HEX si su archivo tiene otra extensión deberá convertirlo), o de hardware.

En cuanto a problemas de hardware del cargador *"quien asegura que no se haya cometido un error de armado y se pusieron componentes que no corresponden al diagrama propuesto"*, seguramente existen cargadores mucho más modernos y versátiles que el NOPPP, pero este tiene una ventaja fundamental, que se autodiagnóstica, es decir, que da información para probarlo paso a paso y reparar lo que está mal. El modo de entregar esta información es a través de la pantalla del monitor que irá indicando que mediciones hacer sobre la plaqueta del cargador de PIC's.

### 8.4.2. AUTODIAGNÓSTICO DEL NOPPP

Una vez en el programa NOPPP (recuerde que lo debe hacer desde el sistema DOS, también se puede hacer desde una ventana de Windows ejecutando el icono de NOPPP), con la debida alimentación y después de haber seleccionado la opción de test (T), verificará que el **dispositivo cargador** esté o no bien conectado en el puerto paralelo o, si está conectado, que el puerto paralelo pudiera estar funcionando mal.

En caso de que se muestre la pantalla de precaución de hardware mientras ejecuta el NOPPP, (NOPPP hardware not fund), si está conectado, significa que su PC tiene un puerto paralelo de ida solamente o que el puerto funciona incorrectamente, o sea, no transmite. En este caso ignore esta pantalla que le va a aparecer siempre (puede aparecer esta pantalla también si está mal armado el cargador o el cable de conexión), ignore la indicación ya que en el proceso posterior aparecerá el modo de reparar la falla. Si desea salir del programa debe pulsar las teclas Ctrl y C al mismo tiempo. Para continuar ignorando el mensaje, pulse la barra espaciadora.

Acepte la propuesta de realizar el autodiagnóstico "T". Tal vez recibirá la pantalla que se muestra en la figura 8.7.


*FIGURA 8.7 PANTALLA DE ADVERTENCIA CORRESPONDIENTE AL PIC*

Esto significa que el cargador no recibe tensión de fuente a pesar de que de entrada se indicó que se encendiera la fuente, como recomendación para evitar futuros problemas se recomienda agregar al cargador un simple LED (rojo) con un resistor de 2.2KΩ en serie para usarlo como indicador piloto de la fuente de 13.4 V.

Conecte la tensión de entrada de 13.4V, también seria conveniente agregar un LED (verde) con un resistor de 1KΩ en serie para medir la tensión de la fuente de 5V, de este modo de un vistazo rápido sabrá que el PIC esta alimentado correctamente y ahorrará problemas porque un corte de tensión en la fuente puede causar un mensaje de error al cargar la primer posición de la memoria del dispositivo. En una palabra que el programa no puede determinar si la fuente se corto después de realizar la prueba de la pantalla "de PIC no encontrado", e intenta cargar la memoria sin fuente, como no lo logra se equivoca y diagnostica que no se puede cargar la primera posición de la memoria y detiene la carga:

Con los dos LEDs puede verificar las fuentes mirando el LED rojo para la fuente de programación de 13.4 V y el LED verde para la fuente del PIC a 5V, al mismo tiempo los LED ayudarán a proteger el PIC evitando que lo conecte o desconecte con la fuente activa, si la fuente está activa y se sigue el proceso de auto diagnóstico se verá la pantalla de la figura 8.8:

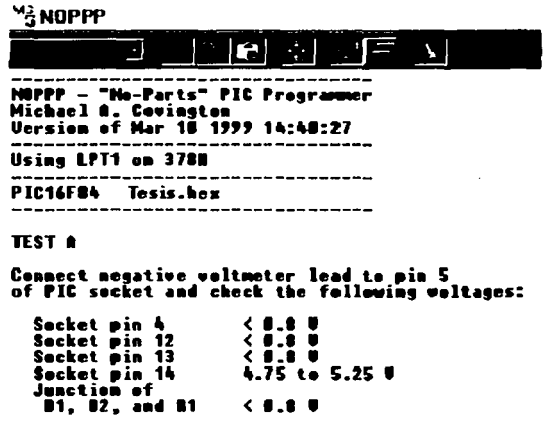



*FIGURA 8.8 PANTALLA DE PRIMER AUTODIAGNÓSTICO DEL PIC*

Esta pantalla invita a conectar la terminal negativa de un multímetro a la pata 5 del PIC y a tomar mediciones de los pines 4, 12 y 13 que deben estar a un potencial bajo menor a 0.8v y la pata 14 debe estar un potencial comprendido entre los 4.75v y los 5.25v. Al mismo tiempo la unión de los diodos D1 y D2 con el resistor R1 debe estar a potencial bajo menor de 0.8v.

Estas mediciones deben realizarse sin colocar el PIC en el zócalo, es decir, que el diagnóstico se hace solo sobre los componentes periféricos para evitar que un PIC dañado altere la rutina de reparación. Recuerde medir la tensión de la fuente del PIC para estar seguro de las mediciones y de la tensión adecuada a 5V con tolerancia máxima de 250mV.

Si se presentan algunos problemas a continuación se muestran unos tips para resolverlos:

- Si la tensión de la pata 13 da más de 800mV es porque seguramente el cable de tierra no esta conectado a la misma (muchas veces este cable es de color negro) y por lo tanto D1 no esta conduciendo por R1, probablemente este cortado el cable de tierra, o este mal conectado a la pata 17 del conector que va a el puerto de la PC.

- También puede ocurrir que el cable esté conectado a tierra pero el diodo D1 este abierto o invertido, para evitar esta duda verifique la tensión de la unión de los diodos D1 y D2.

- Si usted tiene una tensión de la pata 4 (tensión de grabación) alta, o sea cerca de los 13V, probablemente tendrá cortado un cable del puerto o mal conectada la pata 2 del conector, también puede ocurrir que el transistor, ubicado como Q1 en el diagrama, esté abierto o no esté polarizado adecuadamente.

- Luego si tiene tensiones altas en las patas 12 y 13 deberá buscar corto circuitos en las patas del zócalo del PIC o en el circuito impreso o en el conector.

Si las tensiones medidas son correctas presione la barra espaciadora para el Test B y aparecerá la siguiente pantalla como muestra la figura 8.9:

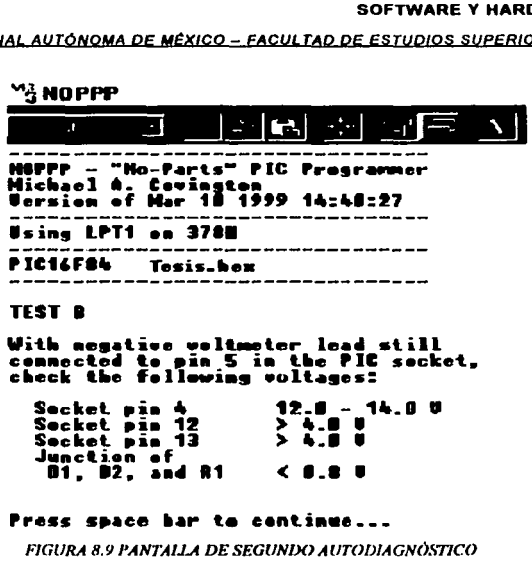



*FIGURA 8.9 PANTALLA DE SEGUNDO AUTODIAGNÓSTICO*

En esta pantalla se pide que se midan las tensiones con respecto a la pata 5, de las patas 4, 12 y 13 pero esta vez con el transistor Q1 cortado y con potenciales altos de 5V, desde la PC, las tensiones medidas en las patas 12 y 13 provienen de la PC y dependen de su fuente reguladora interna y del puerto de salida paralelo, lo importante en esta prueba es que la tensión supere los 4V para que el PIC interprete que hay un estado alto en su entrada de DATOS y CLOCK,

En el test A se verificó el estado bajo de la patas y en este se verifica el estado alto completando las pruebas de las patas 12 y 13, la tensión medida en la pata 4 debe corresponder con la tensión de fuente de la fuente de grabación; es decir, que debe tener un valor de 13.4V aproximadamente, ya que el transistor Q1 no debe conducir.

La unión de los diodos D1 y D2 (el cable de salida de datos que va a la pata 11 del conector) deberá mantenerse a un potencial inferior a los 800mV asegurándose que el diodo D2 no está en corto circuito o invertido, observando que en este test el diodo queda polarizado en inversa.

Si la tensión de la pata 4 esta baja o no existe, probablemente el resistor marcado en el diagrama como R5 está cortado o mal conectado o con un valor equivocado, también puede ocurrir que el transistor tenga un corto circuito entre el emisor y el colector. Revise las tensiones de los cables que van a la pata 12 y 13 si no tienen la tensión indicada revise las resistencias correspondientes (R2 y R3) a cada uno de los cables y sus diodos que pueden estar invertidos o en corto circuito.

Si las tensiones son correctas y todo esta en normalidad, presione la barra espaciadora. En algunas versiones del software NOPPP puede existir, dentro del test, una pantalla que indica se conecte una resistencia de **470 ohms** entre el pin 5 y 13 del zócalo y se midan valores **menores a 0.7 V** en la resistencia (pin 13). Este valor no siempre es necesario. En forma práctica, puede ser que resulte un valor cercano a los 2 V sin que ello afecte la carga del PIC16F84. Sin embargo, si desea obtener los valores solicitados basta con que realice cambios en las resistencias que llegan al pin 13 a modo de obtener los valores deseados (se recomienda hacer un pequeño análisis mediante las leyes de Kirchoff)

En esta pantalla se simula un estado bajo saliendo de la pata de *data* del PIC, la acción del PIC para sacar datos se simula conectando un resistor de 470Ω entre las patas 13 y la pata 5.

Al terminar este test (si es que se muestra) presione la barra espaciadorea para ver la pantalla de la figura 8.10 que muestra el Test C, donde se simulara un estado bajo de la salida de datos cuando se escribe un dato alto.

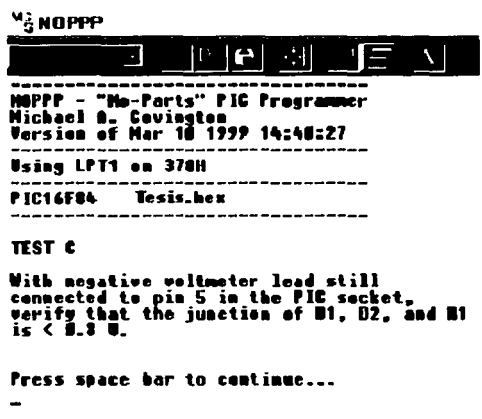



*FIGURA 8.10 PANTALLA DE TERCER AUTODIAGNÓSTICO*

Esto significa que no debe haber retorno directo por el cable de salida de datos cuando se escribe debido a que el diodo D1 esta polarizado con terminal negativa a tierra, para evitar que la tensión sobre él supere los 600mV, esta tensión puede ser algo elevada para los puertos de la PC que podrían llegar a interpretar este estado bajo como si fuera estado alto, aquí se aconseja cambiar los diodos D1 y D2 por diodos de germanio.

Si la tensión es correcta en esta prueba pulse la barra espaciadora para ver la siguiente pantalla (figura 8.11):

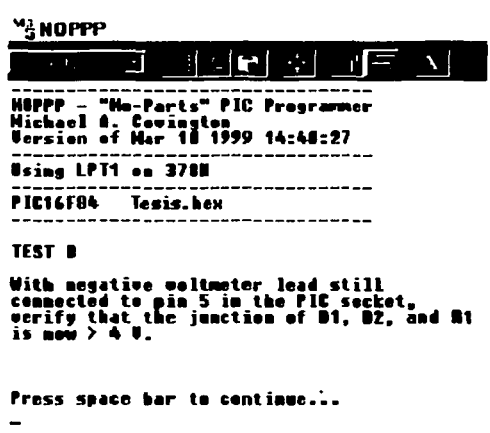



*FIGURA 8.11 PANTALLA DE CUARTO AUTODIAGNÓSTICO*

En la unión de D1, D2 y R1, las tensiones correspondientes deben ser iguales a 4.4V, se debe suponer que los diodos D1 y D2 podrían afectar esta medición así que tome en cuenta las caídas de tensión.

Si las mediciones son correctas puede pasar a la última pantalla del autodiagnóstico que se muestra en la figura 8.12 donde se avisa que la secuencia de pruebas de voltajes está terminada y se aconseja que revise las conexiones de los pines, y que el cable de la PC al cargador no sea mayor a 1 metro.

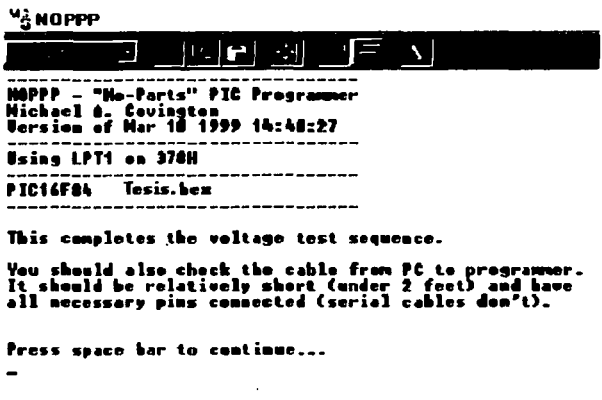



*FIGURA 8.12 PANTALLA FINAL DE PRUEBAS DE AUTODIAGNÓSTICO*

En las pruebas desde la A a la D se utilizaron estados altos o bajos de los cables que se resumen a continuación:

| NÚMERO DE TEST | CABLE DE FUENTE DE GRABACION | CABLE DE CLOCK | CABLE DE DATOS | CABLE DE LECTURA / ESCRITURA |
|---|---|---|---|---|
| A | ALTO | BAJO | BAJO | BAJO |
| B | BAJO | ALTO | ALTO | BAJO |
| C | ALTO | BAJO | ALTO | BAJO |
| D | ALTO | BAJO | ALTO | ALTO |

Durante este capítulo se han visto las distintas fallas que se pueden presentar y una de las mayores es cuando se produce un error en posición de memoria 0000. Es cuando debe checar las conexiones y principalmente que las fuentes estén conectadas, que ese problema se resuelve con los diodos verde y rojo.

Otro problema grande es cuando se da un mensaje de error por mala elección o elección errada del tipo de PIC. Este mensaje de error le especificará en la pantalla que se cometió un error en la palabra de configuración del programa y que este no corresponde al PIC elegido, lo que debe hacer es presionar la barra espaciadora y elegir correctamente el PIC en la pantalla de funciones del NOPPP.

## TEMA 8.5. VARIACIONES DEL HARDWARE

El circuito y el software originalmente son suficientemente fiables. Sin embargo, para mayor compatibilidad con unos poco puertos paralelos que tienen problemas con el circuito original, puede hacer algunos ligeros cambios:

- Cambiando los diodos 1N914 por 1N34 (o 1N34A) se obtienen niveles lógicos mejores, y por lo tanto mayor inmunidad al ruido, sin costo adicional.

Si utiliza el circuito original y funciona correctamente no necesita hacer ningún cambio. Si tiene un puerto paralelo no estándar o cables de poca calidad, estas modificaciones le ayudarán.

Otro motivo de realizar cambios al hardware original del NOPPP es cuando no se obtienen los datos requeridos por el software durante el test, lo cual se debe, en muchas ocasiones, al tipo de PC utilizada. Si este es su caso, sólo debe analizar los valores de resistencias, voltajes y corrientes a partir de la salida de su PC. Las resistencias son las que producen la mayoría de las caídas de voltaje y son estas las que se pueden modificar con más facilidad. Así mismo, debe de tener muy bien entendido de donde provienen las corrientes que pasan por cada resistencia y los voltajes que las producen, para saber a que valor aproximado se debe cambiar una resistencia.

Como nota adicional, no siempre existe mal funcionamiento por la variación en los valores solicitados por el test, puede funcionar bien el hardware aun con otros valores, esto solo se comprueba de forma práctica, es decir, cargando un PIC y probándolo. Sin embargo, la decisión es suya, pues esta determinación podría quemar el PIC.

## TEMA 8.6. CARGA DEL ARCHIVO .HEX AL PIC.

Con lo tratado hasta el momento, se supone que el cargador de PIC's funciona correctamente, pulse en la pantalla de selección del PIC la letra "F" y dará la siguiente pantalla (figura 8.13):

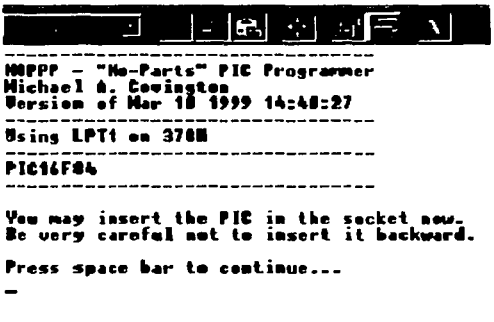



*FIGURA 8.13 PANTALLA DE INSERTADO DEL PIC EN EL ZOCALO*

En esta pantalla sólo se indica que es el momento de insertar el PIC en el zócalo. En este caso debe observar que la fuente esté conectada, se hace la recomendación de que cuando sea necesario insertar el PIC se desconecte la fuente se coloque el PIC en el zócalo y posteriormente se vuelva a encender la fuente; ya que si se hace caso omiso a la indicación de la pantalla que indica que el PIC no debe estar sobre el zócalo (en el inicio de este subcapitulo), el software dará una indicación del tipo "programmer no conected" (programador desconectado) y abortará la carga.

La pantalla de la figura 8.14 se puede considerar como la última de la serie de predisposiciones que se requieren para la iniciación del NOPPP, en esta pantalla puede elegir la operación que desea realizar que son seleccionables por las teclas significan lo siguiente:

**L)** ***Cargar el NOPPP.*** Con un archivo que debe ser del tipo hexadecimal, este formato no es el formato que maneja el PIC, por lo tanto antes de cargarlo debe generar el archivo HEX[74] partiendo del archivo mnemotécnico que se genera con el MPLAB.

**S)** ***Seleccionar el tipo de PIC.*** En este caso el PIC ya fue seleccionado pero podría necesitar un cambio si debe programar más de un tipo de PIC. Aun así, puede oprimir la tecla "S" y volver a elegir el PIC16F84.

**E)** ***Borrar un PIC previamente grabado.*** Este tipo de PIC no necesita un borrado previo a la grabación; pero puede ser necesario borrarlo por razones de seguridad.

**P)** ***Programar el PIC.*** En esta función se programa el PIC y es donde se empieza a cargar el programa

**V)** ***Verificar el programa cargado en un PIC.*** En esta función se carga un programa hexadecimal y el cargador verifica que el programa almacenado en el PIC sea igual. Debe tener en cuenta que solo se puede realizar la verificación cuando el programa no este protegido.

[74] Para mayor referencia consulte el Capitulo 7 del MPLAB.

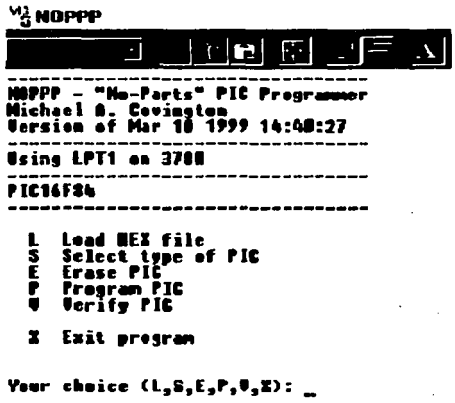



*FIGURA 8.14 PANTALLA DE FUNCIONES DEL NOPPP*

### 8.6.1. CARGA DEL PIC CON ARCHIVOS .HEX

Para poder cargar un PIC no necesita tener cargado ningún utilitario más que el NOPPP en su computadora y por supuesto el archivo del programa con extensión ***.hex.***

Después de haber seleccionado el PIC16F84, el NOPPP regresará a la pantalla principal que muestra las opciones de L, S, E, P y V donde si quiere cargar un PIC debe antes cargar el programa ***.hex*** en el NOPPP, en este caso elija la letra "L" de "load hexadecimal file" cuya traducción es cargar archivo hexadecimal

En la pantalla que aparecerá se debe escribir el nombre de archivo a cargar, notando que el programa no da la posibilidad de buscar el archivo en otro directorio que no sea aquel donde se cargo el NOPPP, es decir, se recomienda al usuario que copie su archivo del directorio del MPLAB (en el caso de la tesis recuérdese que se creó un archivo con extensión hex, en el capitulo de MPLAB, llamado ***tesis.hex***) y péguelo en la carpeta donde tiene los archivos del NOPPP y así podrá ejecutar y cargar el programa en el NOPPP.

Entonces teclee el nombre del programa sin olvidar la terminación hex (en este caso será ***tesis.hex)***, y el archivo se cargara en el NOPPP. El resultado de la carga se observará en la pantalla.

En esta pantalla se pueden observar algunos detalles del tamaño del archivo **.hex** desglosado, cabe mencionar que dependiendo del tamaño del programa variará la estructura de la pantalla, también en ésta se puede ver la cantidad de palabras de programa, de configuración, de identificación y de datos. También aparece una indicación de que el programa completo fue cargado, por último se indica que se vuelva a apretar la barra espaciadora, con lo cual aparece de nuevo la pantalla inicial que da diferentes opciones antes vistas.

Ahora si, es el momento de cargar el PIC eligiendo "P" (PROGRAM PIC), se observará un movimiento en la pantalla y unos segundos después aparecerá un cartel indicando que la carga fue exitosa, recordando que el NOPPP irá colocando los datos en la memoria y luego los ira leyendo; si el dato anterior se cargo correctamente, pasará a cargar el siguiente y así sucesivamente hasta llegar el último, posteriormente el programa indicará que se debe cortar la fuente y luego sacar el PIC del zócalo, cumpla con las indicaciones y coloque el PIC programado en el zócalo del proyecto correspondiente y por último, reponga la fuente y observe su proyecto, verá que si funciona será una meta bien obtenida.

### 8.6.2. CARGADO EXITOSO DEL PIC

La pantalla más esperada antes de este capítulo es la de pantalla de cargado con éxito (Programing complete).

El mensaje se divide en tres partes, la última parte indica que la carga de memoria de programa se completo exitosamente. Al final se indica que para utilizar el PIC cargado en un proceso debe comprobar su funcionamiento correcto con la mínima y máxima tensión de la fuente Vcc (según las características del PIC).

La comprobación por software se realiza pulsando la barra espaciadora y eligiendo luego la letra "V" de verificación, si tiene cargado el programa con el cual se grabó el PIC se producirá una rápida verificación del programa cargado dentro del PIC.

Cabe mencionar que estas pantallas cambiarán dependiendo de la versión de NOPPP que se ocupe, del programa cargado, del tipo de PIC usado, en especial para quien habla español existe una versión en español en la red, solo es cuestión de buscarla. Ahora volviendo al tema, si va a ocupar el PIC previamente cargado debe volver a realizar este proceso de verificación llevando la tensión de Vcc a un valor de 4.75 y 5.25V, después pruebe su PIC dejándolo inactivo un tiempo y después vea si sigue cargado solo así se puede asegurar su buen funcionamiento para un proyecto grande.

Por último, volviendo a la pantalla de elección de funciones del NOPPP, da la opción de borrado con la letra "E" (erase PIC). El PIC16F84 no necesita de borrado antes de volver a ser grabado, pero para tener seguridad al cargarlo, este sofware solo permite grabar el PIC si se borra antes eligiendo esta opción y volviendo a elegir el tipo de PIC para comenzar un nuevo proyecto.

Por lo que respecta a este capítulo del NOPPP se considera que ya se mencionó lo importante y elemental para el dominio de este cargador.

# CAPITULO 9

# OTROS MICROCONTROLADORES DE 8 BITS

# CAPITULO 9. OTROS MICROCONTROLADORES DE 8 BITS[75]

En esta sección se hará mención de los microcontroladores más conocidos que se utilizan en la mayoría de proyectos, haciendo referencia solo a los microcontroladores de 8 bits, dado que esta tesis solo se enfoca al desarrollo de un dispositivo de 8 bits como es el caso del PIC16F84.

No se pretende realizar un análisis a fondo de cada microcontrolador a tratar en este tema, se intenta tomar solamente las características más importantes de cada uno y así poder compararlos con el microcontrolador PIC16F84.

Dentro de los controladores de 8 bits más conocidos están los de: National, Philips, Motorola, Nec, Intel y Microchip como los más comunes y clásicos, que aunque todos hayan sido fabricados para casi la misma capacidad y las mismas funciones, existen diferencias entre ellos. En general las especificaciones son similares para todos los microcontroladores, pero haciendo hincapié en una característica especifica se ven las diferencias.

## TEMA 9.1. LA SERIE COP DE NATIONAL

La línea COP emplea una arquitectura tipo Harvard con buses de instrucciones y datos separados que permite que la mayor parte de las instrucciones (el 77%) puedan ser ejecutados en un único ciclo de instrucción, al realizarse simultáneamente el acceso al espacio de datos (memoria RAM y periféricos), determinan con ello un promedio de instrucciones próximo a 1MIPS (un millón de instrucciones por segundo).

---

[75] "Todo sobre PICs & microprocesadores y microcontroladores", Vallejo Horacio D, H. Picerno Alberto, Prado Federico, Rodríguez Luis H., Quark S.R.L., 2001, Argentina, pag 95-121

Otra características importantes de los dispositivos de la serie COP, es que poseen espacio de programas tipo OTP (One Time Programing) con protección de escritura y lectura, además de que son diseñados para que tengan emisiones de poca radiación, con interrupciones opcionales de 4 a 8 pines, tienen espacio de almacenamiento en EPROM, posee varias opciones de oscilador que van desde un oscilador RC interno al que se le puede alterar la frecuencia mediante un capacitor externo, hasta un oscilador externo a cristal resonador con 4 tipos diferentes (como el caso del PIC) que son HS, XT, LP, RC, poseen reseteo interno seleccionable, tiene 12 salidas de alta corriente y tiene el WATCHDOG.

Los COP también poseen un monitor de clock justo como en el PIC, 16 pines de entrada / salida en los de tipo de 8 bits, (contiene en total desde 20 a 28 pines según su versión).

Las características de su CPU son que tiene versatilidad de configuración, tiempo cíclico de instrucciones, posee 8 interruptores de servicio de fuentes múltiples con vector, interruptor externo, y timer de inactividad. En el manejo de bits utiliza el puntero SP de 8 bits, tiene mapeo de memoria e instrucciones aritméticas en código BCD.

En las características periféricas tiene una lógica de despertador de entradas múltiples, posee un timer de 16 bits con dos registros, un contador de eventos externos así como una interfaz serial; las características de I/O son que tiene una salida TRI-STATE y tiene entradas de alta impedancia; el diseño CMOS es estático completamente, opera a un simple abastecimiento de entre 2.7V hasta 5.5V tiene un rango de temperatura de –40 a 125°c, en operación normal es de 25°c.

Dentro de todos los COP incluso los más pequeños existen dos timers llamados TO y T1, el T0 opera como un contador simple de 12 bits en modo libre y descendente y es usado como salida controlada de los modos de bajo consumo, mientras que el T1 es de 16 bits y ofrece un cambio de versatilidad al ser operado en 8 modos distintos, esta es una característica que solo pocos controladores de 8 bits ofrecen, esto permite que en sus variaciones se acceda al conteo de eventos externos así como la detección de flancos y algún modo de generación de señales.

Si se analizan en el mercado las ofertas de varios dispositivos, sólo se encontraran las opciones que propone Microchip con un costo semejante al COP y, en un costo mucho más elevado, Motorola e Intel.

## TEMA 9.2. LOS MICROCONTROLADORES DE PHILIPS

Con excepción del 83C751, todos los dispositivos de esta familia pueden manejar hasta 64 bytes, tanto de programa como de memoria de datos, el controlador contiene el CPU de 8 bits optimizada para aplicaciones de control, capacidad de procesamiento Booleano, 32 líneas bidireccionales, RAM de datos de 128 bytes on-chip, 5 entradas de interrupciones con dos niveles de prioridad entre otros.

El microcontrolador 80C51 es la versión CMOS del 8051 siendo totalmente compatible con los demás controladores de otras generaciones atrasadas. La memoria de programa ROM o EPROM puede ser leída, pero no grabada, el espacio de memoria de programa es amplio más que de los de su genero, en las versiones sin ROM toda la memoria de programa es externa.

La memoria de datos RAM ocupa un espacio separado para el direccionamiento a partir del programa de memoria, las memorias externas de datos pueden combinarse con memorias de programas si se desea, a través de la aplicación de señales y en las entradas de algunas compuertas, por lo que se utiliza la lectura para la memoria de datos.

La memoria de programa goza de ventajas mucho más extensas que las de un procesador de su clase, posee interrupciones externas, y manejo de subrutinas en tratamientos de interrupción, semejante al PIC16F84, la memoria de datos contiene una configuración de hardware que permite acceder hasta 2k bytes de una memoria RAM externa, el CPU genera señales de escritura y lectura a medida que sean necesarias durante el acceso a RAM externa, las direcciones para acceder los datos de la memoria externa pueden utilizar tanto como 1 o 2 bytes de extensión.

El set de instrucciones del 80C51 incluye una amplia selección de instrucciones de bit único y los 128 bytes de esta área pueden ser directamente dirigidos por estas instrucciones, así permite la manipulación directa del bit en sistemas lógicos y de control que requieren un procesamiento booleano.

Los modos de direccionamiento son: el modo directo, el indirecto, por medio de instrucciones del registrador, de instrucciones especificas de constantes inmediatas, de direccionamiento indexado, por instrucciones aritméticas, por instrucciones lógicas, por transferencia de datos, por instrucciones booleanas o usando la instrucción de salto.

Las interrupciones pueden ser habilitadas o deshabilitadas individualmente por la colocación o retiro de un bit en la parte de la SFR denominada activación de interrupción, este registro contiene también un bit de inhabilitación de todas las interrupciones al mismo tiempo, mientras que cada fuente de interrupción puede programarse individualmente para presentar uno o dos niveles de prioridad por utilización de un bit en el SFR denominado como interruptor de prioridad.

## TEMA 9.3. LOS MICROCONTROLADORES DE MOTOROLA

El microcontrolador Motorola MC68HC705J1 posee arquitectura Von Neumann, un manejo de frecuencia de 20 MHz como rango máximo, trabajando en rango medio de operación de 5.5V y 20 mA.

Tiene una capacidad de memoria de 1200x8 y un manejo de RAM de datos de 64 bytes, la EPROM está construida con una matriz de transistores (como en otros) tipo MOFSET, su borrado es por medio de luz ultravioleta, no de forma eléctrica como en el PIC.

El microcontrolador MC68HC705J1 en el modo normal, la velocidad está directamente determinada por la frecuencia del cristal o filtro cerámico del clock, pero aparte del modo de funcionamiento normal goza, a diferencia de los demás, otros dos modos de funcionamiento que es el WAIT y el STOP estos modos están previstos para que el dispositivo reduzca dramáticamente el consumo, ya que hasta éste se puede alimentar con pilas cuando trabaja como controlador en forma remota.

El RESET se puede producir automáticamente o como una condición buscada exteriormente con un pulsador, utiliza el sistema WATCHDOG de la misma manera que el resto de los de su clase, tiene dos puertos paralelos de entrada y salida, o sea alrededor de 14 pines, tiene un temporizador programable de 16 etapas, su utilidad es para demorar la acciones del micro de manera sencilla y con una precisión dada por el cristal.

La CPU realiza el procesamiento de datos de entrada y salida en función de las ordenes del programa, goza de una ALU, un acumulador, un registro de índice, un apuntador de pila, un contador de programa, y un registro de condición (entre los más comerciales), utiliza el mismo sistema de memorias que el resto de microcontroladores, la diferencia que lo hace especial es que en su set de instrucciones se pueden hacer maravillas pues tiene una estructura tipo CISC pero muy eficiente en el diseño y programación, lo que hace que este dispositivo sea casi de un uso completamente industrial.

Existe la versión OTP, que significa (One Time Programing) memoria de única programación, lo que hace que una vez grabados los dispositivos realizan la misma función.

Para poder comparar las características fundamentales de los microcontroladores antes mencionados, se muestra la Tabla 18 con un resumen sus aspectos más importantes.

| CARACTERÍSTICA | NATIONAL | PHILIPS | MOTOROLA | MICROCHIP PIC16F84 |
|---|---|---|---|---|
| *Tipo de Arquitectura* | *HARVARD* | *VON- NEUMANN* | *VON- NEUMANN* | *HARVARD* |
| Memoria de Programa (bytes) | 1 k | 1k | 1.240k | 1k |
| RAM Datos (bytes) | 64 | 64 | 64 | 68 |
| EPROM Datos (bytes) | 64 | 64 | 64 | 64 |
| Frecuencia Máxima (MHz) | 10 | 12 | 20 | 10 |
| Voltaje Medio (V) | 5.5 | 6.5 | 5.5 | 5.0 |
| Corriente Máxima (mA) | 10 | 20 | 20 | 10 |
| Temperatura Media (°c) | 20° | 25° | 25° | 25° |
| Pines I/O | 16 | 16 | 14 | 13 |
| Tipo de Reloj Oscilador | HS, XT, LP, RC | HS | HS, RC | HS, XT, LP, RC |
| Tipo de Set | RISC | RISC | CISC | RISC |
| Nota: El costo de cada microcontrolador varia mucho dependiendo el lugar donde se compre pero el PIC16F84 oscila alrededor de los $70, lo cual lo hace mucho muy accesible para los estudiantes. | | | | |

*TABLA 18 COMPARACION ENTRE MICROCONTROLADORES MÁS COMUNES*

Quizá la mayor ventaja del PIC se encuentra en que puede ser borrado de manera eléctrica (no con UV), lo que permite que se puedan hacer muchas pruebas o proyectos sin la necesidad de tener los accesorios para el borrado UV que aumentaría el costo de cualquier proyecto. Además la amplia información que se encuentra acerca del mismo, el software y hardware que se pueden utilizar para crear programas, depurarlos y cargarlos a la memoria del PIC representan las mayores ventajas de los PIC's frente al resto de los microcontroladores.

Aunque muchos controladores tienen capacidades mayores a la del PIC16F84, que se trata en esta tesis, debe tener claro que lo que se ha propuesto hasta el momento es el uso de un microcontrolador de manera didáctica (que no significa que no sirva de manera industrial), y a criterio de los tesistas el PIC16F84 es uno de los que mejores características tienen para este fin.

# CAPITULO 10

# USO PRÁCTICO.
# GENERADOR DE VIDEO

# CAPITULO 10. USO PRÁCTICO. GENERADOR DE VIDEO

Como parte de esta tesis a continuación se desarrolla un proyecto, en donde el objetivo es la construcción de un generador de patrones de video compuesto, cuyas aplicaciones se extienden desde el campo de la enseñanza de la señal de video hasta la reparación y ajuste de equipos receptores de televisión, utilizando los patrones como: Barras de color, Raster, Cross-hatch y Puntos.

Para desarrollar este proyecto es necesario dividirlo en varias partes, donde la primera es la generación de sincronismo y patrones, y posteriormente la generación de video compuesto.

No se pretende dar una explicación a fondo de los conceptos a considerar para este proyecto, ya que no es la finalidad de esta tesis. Sin embargo, se hace una breve referencia de lo que se necesita para realizarlo, pero sobre todo con la finalidad de entender el programa que se usará en el microcontrolador.

**Los colores en televisión**[76]

La imagen en un televisor a colores se forma mediante la emisión de luz resultante de la excitación de la película de fósforo, que recubre internamente la pantalla, al ser alcanzada por un haz de electrones que barre periódicamente la superficie visible.

[76] Televisión Básica 1, Van Valkenburg, Nooger & Neville, Inc.. Editorial Continental S. A, 1973, México Pag 68

Si se habla de "emisión de luz", inmediatamente debe pensar en procesos "aditivos", lo cual lleva a concluir que en televisión los colores primarios son el Rojo, Verde y Azul (RVA o en inglés RGB). Efectivamente, dentro del tubo de televisión se emiten tres haces de electrones, destinados cada uno a excitar una franja de fósforo en la pantalla, la cual responderá emitiendo un color característico al fósforo empleado. Naturalmente, como no podía ser de otro modo, estos colores son Rojo, Verde y Azul.

Los demás colores en la televisión se pueden obtener combinando estos tres colores primarios en distintas proporciones.

Por lo tanto, un generador de video básico debería hacer lo siguiente:

- Tener tres salidas, una para cada color.
- Cada una de estas salidas se debería conectar a la correspondiente dentro del sistema.
- El equipo generará combinaciones en sus salidas.

La mayoría de los equipos de televisión y video no poseen entradas directas de colores, Lo habitual es que los equipos hogareños tengan una entrada de "Video Compuesto", denominada usualmente como "VIDEO IN". Por esta razón, este generador debe poder convertir los componentes de color en ese "Video Compuesto".

## VIDEO COMPUESTO[77]

Las señales de salida de color contienen toda la información de la imagen, pero ocuparían un ancho de banda considerable si se transmitieran una a una.

---

[77] Op cit [76] Pag 68

Para reducir este ancho de banda, además de mantener la compatibilidad entre transmisiones en "Blanco y Negro" y "Color", se creó la señal de Video Compuesto.

Se deben generar todos los sincronismos necesarios para generar correctamente la imagen en la pantalla del televisor, "sincronizar" el haz de electrones que barre la pantalla con el barrido generado en el estudio de televisión, si no se hiciera esto, las imágenes aparecerían cortadas, con barras inclinadas, con colores incorrectos (basta con ver un "canal codificado" para tener una idea de lo que se quiere decir, ya que una manera de codificar la señal es quitarle los sincronismos). Los sincronismos son dos en este caso el sincronismo vertical y el sincronismo horizontal.

Para poder indicar la sincronización hay que ubicar líneas de barrido tanto verticales como horizontales, donde hay más frecuencia es en las líneas horizontales, donde el número de líneas dentro de una imagen son parte de las normas que rigen los estados de transmisión para los países.

Las normas establecen los parámetros que deben seguir tanto los equipos transmisores de señal como los receptores, a fin de que se establezca una comunicación segura y sin errores entre ambos. Entre los muchos parámetros que se fijan considere solo los que atañen a este proyecto.

Primeramente la imagen que se transmite se divide en líneas horizontales, estas líneas serán posteriormente transmitidas al receptor de televisión para que este las reproduzca secuencialmente en la pantalla. Sin embargo, no se transmiten las líneas consecutivamente (línea 1, 2, 3, 4, ...) sino que primero se transmiten las impares (línea 1, 3, 5, ...) y luego las pares (línea 2, 4 ,6, ...). Del mismo modo recorre el haz de electrones la pantalla, reproduciendo primero las líneas impares y luego las pares. Por este motivo se requiere que el haz recorra 2 veces toda la pantalla para formar una imagen completa. A las dos "semi-imágenes" se les denomina Campos (Impar y Par respectivamente) y a la imagen completa Cuadro.

Cuando se hace el barrido de la pantalla se realiza en dos formas, el Barrido Entrelazado y el Barrido no Entrelazado. El Barrido Entrelazado su nombre proviene del "entrelazamiento" de las líneas de barrido de dos campos sucesivos, la gran ventaja que tiene este barrido es el de obtener una mayor frecuencia de repetición de imágenes sin aumentar el ancho de banda.

Mientras que el Barrido No Entrelazado se utiliza en imágenes fijas, simplemente superponen las líneas de barrido campo tras campo, esta técnica es muy utilizada en monitores de computación y en generadores de patrones cuando se utilizan señales como el "Cross-hatch" (líneas) y "Puntos".

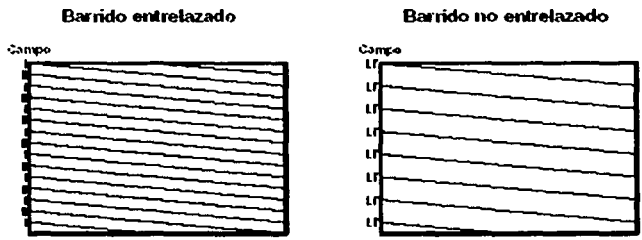



*FIGURA 10.1TIPOS DE BARRIDO DE LA SEÑAL DE VIDEO*

Observe que en el caso del barrido entrelazado, la primera línea horizontal comienza a la mitad de la pantalla, en tanto que en el barrido no entrelazado se comienza con una línea completa. Se menciona esto ya que es muy importante para poder realizar el programa que lleve a cabo los sincronismos.

### 10.1.1. NORMA NTSC[78]

Para asegurarse de generar correctamente el video que se desea es necesario analizar la norma NTSC, ya que esta influye en la realización de proyecto para delimitar ciertos tiempos necesarios para la generación de video. Cabe aclarar que los valores expresados dentro del parantesis son una variación hecha en los parámetros solo con fines prácticos y, en ocasiones, fuera de norma, ello con la finalidad de hacer más fácil el cálculo de los tiempos al momento de realizar el programa para este proyecto:

**Frecuencia Horizontal (H):** 15750 KHz, es decir, duración de una línea: 63.5 µseg (64 µseg)

**Sincronismo Horizontal (H Sync): de 4.19 a 5.71** µseg (4.8 µseg).

**Frecuencia Vertical (V):** 60 Hz

**Sincronismo Vertical (V Sync.):** 2.5 líneas horizontales

**Líneas Horizontales:** 525 por cuadro (262.5 por campo)

**Subportadora de Color (SC):** 3.579545 MHz

**"BURST" de Color:** 9 a 11 ciclos de SC

**"Front Porch":** 1.9 µseg.

**"Back Porch":** 5 µseg.

**"Pos sincronismos":** 11 o 12 pulsos según el campo y el patron a generar.

---

[78] http://www.sct.gob.mx

**Pulsos de pre y pos-ecualización (o de igualación 5 c/u): de 2.29 a 2.54** µseg (2.4µseg).

En la figura 10.2 se observa esquema que representa una señal real de video, utilizando los dos métodos de barrido mencionados.

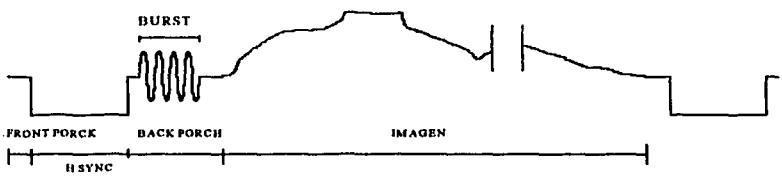



*FIGURA 10.2 CARACTERÍSTICAS DE LA SEÑAL DE VIDEO*

El pulso de sincronismo vertical abarca 2½ líneas horizontales (figura 10.3). Además, se ve algo no mencionado hasta ahora; unos pulsos de Pre y Pos-Ecualización.

La detección del sincronismo vertical en el televisor se realiza integrando la señal de video que llega. La integración se lleva a cabo mediante un circuito RC, y se monitorea la carga del capacitor para decidir cuando disparar el vertical. Haciendo que los barridos verticales no se disparen siempre en un mismo punto, provocando inestabilidades en la imagen (temblor vertical). Para evitar esto se colocan los pulsos de pre y pos-ecualización, cuya función es llevar la carga del capacitor a valores fijos antes y después del pulso vertical

Los pulsos de pre y pos-ecualización tienen una frecuencia igual al doble de la horizontal y su duración es la mitad de la del H Sync (figura 10.3). Se aplican durante un tiempo igual a 2½ líneas horizontales, los pulsos positivos que aparecen dentro del pulso de sincronismo vertical se denominan "Serrated Pulses", y su función es mantener enganchado al oscilador horizontal durante este período, su duración es igual al H Sync.

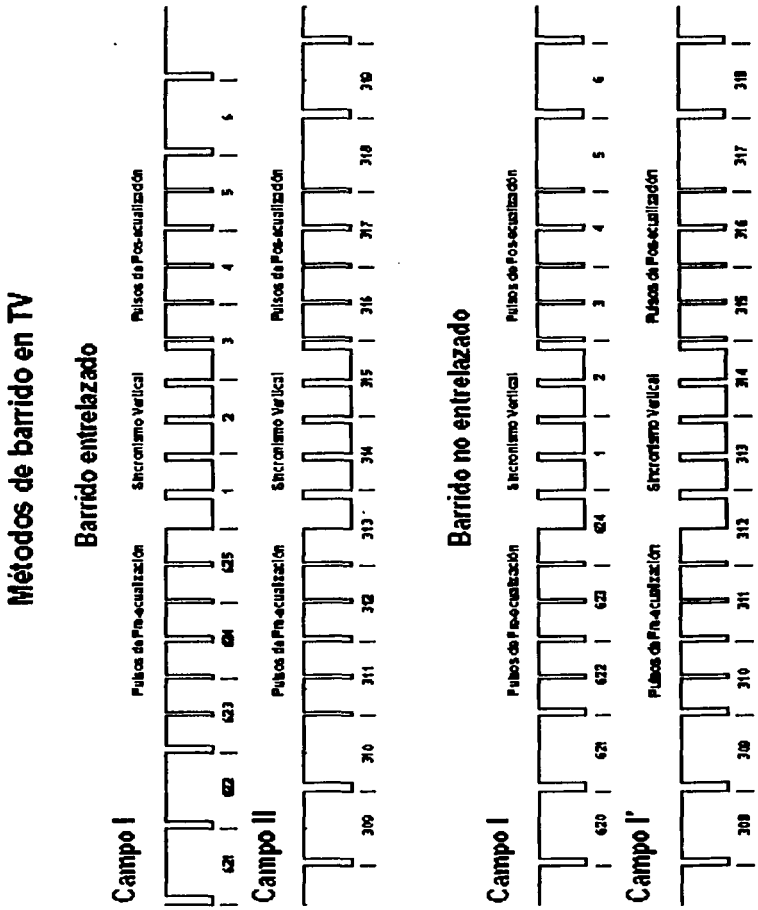



*FIGURA 10.3 PULSOS EN LA SEÑAL DE VIDEO*

Con estos conceptos se puede empezar con la construcción del generador de patrones de video.

## TEMA 10.2. CONSTRUCCIÓN DEL GENERADOR DE VIDEO

Como se vió anteriormente un generador de patrones de video no solo debe entregar componentes de color de la señal, si no que también debe generar un video compuesto, con todo lo que esto implica; generar sincronismos, obtener la señal de luminancia, generar subportadora de color y modularla, que esto se realiza gracias a un circuito integrado, el MC1377p y con un cristal como se verá después.

Lo primero que se define es la cantidad y tipo de patrones que deberá generar el equipo, ya que esto determinará las características, y por lo tanto la complejidad del sistema a desarrollar. En este caso el equipo será capaz de generar cuatro patrones básicos:

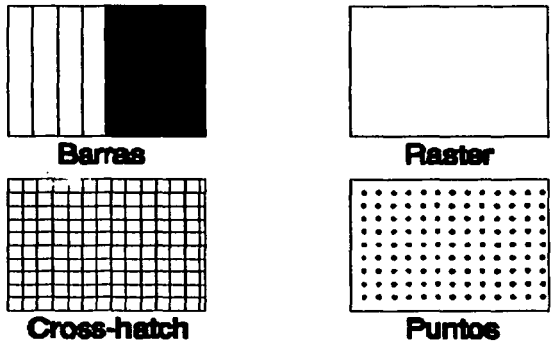



*FIGURA 10.4DIFERENTES PATRONES DE VIDEO*

Además, en este generador se permitirá el control de algunos colores en la pantalla lo que hace que se amplié la cantidad de patrones que pueden ser generados y así adoptar diferentes combinaciones de colores (mediante las llaves S1, S2 y S3). Para seleccionar cual de los cuatro patrones básicos generará el equipo se utilizan dos llaves nombradas S4 y S5, cuya combinación determinara el patrón según la siguiente tabla:

| LLAVES | S4 OFF | S4 ON |
|---|---|---|
| S5 OFF | BARRAS | RASTER |
| S5 ON | CROSS-HATCH | PUNTOS |

### 10.2.1. GENERACIÓN DE SINCRONISMOS Y PATRONES

La generación de la base de tiempo, los sincronismos y los cuatro patrones básicos estarán a cargo de un microcontrolador (PIC16F84-10), el microcontrolador deberá hacer lo siguiente:

- Generar una base de tiempos estable, de donde debe obtener todos los tiempos requeridos por los sincronismos.
- Generar en uno de sus terminales, el correspondiente al **Bit 0 del PORTB**, todos los **sincronismos** requeridos por la norma de televisión adoptada (NTSC), sin agregar video a esta señal (sincronismos puros).
- Generar en las terminales las señales que correspondan con el patrón que deba mostrarse a la salida. Estos terminales no tendrán sincronismos (video puro).
- Aceptar en dos de sus terminales, configurados como entradas, las órdenes provenientes de las llaves S4 y S5, a modo de poder seleccionar el patrón a generar. Estas entradas corresponden a dos Bits del PORTA, los siguientes:

  PORTA (2) = S4

  PORTA (3) = S5

El programa básicamente se compone de cuatro bloques independientes de generación de señal, las llaves S4 y S5 generaran la combinación que se deseará ver en la pantalla, mientras que el patrón que se eligió no será mas que uno de los cuatro bloques elegidos por medio de la combinación de las llaves. Según que combinación se encuentre activada en ese momento, el programa se dirigirá a uno de los cuatro bloques de video mencionados donde se generará uno de los patrones.

En cada uno de estos bloques se comienza por generar los pulsos de pre-ecualización, luego el sincronismo vertical con sus correspondientes "Serrated Pulses", seguido de los pulsos de pos-ecualización.

Posteriormente se realiza la selección de campo par/impar, en uno de los campos la primera línea horizontal luego del sincronismo vertical es completa, mientras que en el otro campo debe ser solo media línea.

Es de destacar que en dos de los patrones (Cross-hatch y Puntos) se trabaja con barrido no entrelazado, para evitar el fenómeno de temblor vertical (o "flicker") de las líneas fijas. Luego de esto se hacen 3 o 4 líneas horizontales sin video (según el campo), pero respetando correctamente los tiempos de sincronismo, luego de generar el sincronismo horizontal y respetar el tiempo de back porch, en las líneas aparece la información que corresponda a la señal mostrada.

Recuerde que cada línea horizontal dura un tiempo total de 64 µseg., donde se incluyen 4.8 seg. de H Sync, 1.9 µseg. de Front Porch y 5 µseg. de Back Porch. Por lo tanto solo quedan 52.3 µseg para mostrar video, y ese es el tiempo útil.

Para el cálculo de tiempos dentro del programa el uso del microcontrolador PIC16F84-10/P facilita mucho esta tarea, ya que basta con "contar ciclos de reloj" para obtener todos los tiempos correctos. Utilizando un cristal de 10 MHz, y sabiendo que cada ciclo de instrucción son 4 ciclos de reloj, se obtiene el tiempo de un ciclo de instrucción:

$$Tosc = 1/fosc$$

$$Tins = Tosc \times 4$$

$$Tins = 1/10\ MHz \times 4 = 0.4\ \mu seg.$$

Si cada ciclo de instrucción dura 0.4 µseg., entonces para obtener el pulso de sincronismo horizontal basta con contar 12 ciclos: **12 x 0.4 µseg. = 4.8 µseg.**

Del mismo modo se obtiene que la duración de una línea horizontal completa es de 160 ciclos de instrucción: **160 x 0.4 µseg. = 64 µseg.**

Básicamente, esto es lo que hace el programa del PIC, cuenta instrucciones y pone a nivel alto o bajo, según corresponda, el Bit 0 del PORTB. Se estableció que durante el pulso de sincronismo (H o V) este bit estará a nivel bajo (0V) y el resto del tiempo a nivel alto (5V).

## TEMA 10.3. REALIZACIÓN PRÁCTICA DEL GENERADOR DE VIDEO

Ahora se debe combinar la señal de colores con la señal de sincronismos y con ambas generar video Compuesto bajo la norma NTSC[79], señal que sí puede inyectarse a los equipos mencionados como televisión y video.

Una vez obtenida la señal de video Compuesto se ajusta el nivel y la impedancia de la misma pasando por un circuito buffer, conformado por Q1, R14 y R15, como se observa en el circuito que se muestra en la figura 10.5.

[79] En este proyecto se utiliza un cristal de 3.582056 MHz para que el equipo genere señal.

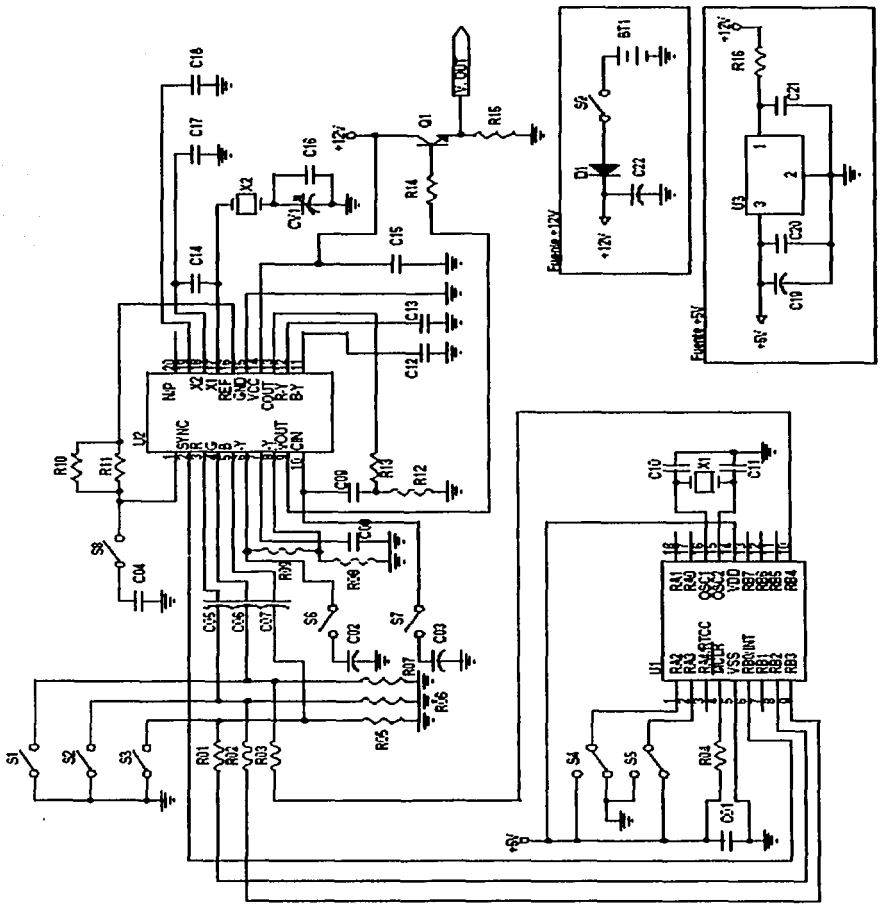



*FIGURA 10.5 CIRCUITO COMPLETO PARA EL GENERADOR DE VIDEO*

Con esto concluye la generación de señal y la descripción del circuito, solo queda por mencionar que ambos integrados se alimentan con tensiones diferentes, por lo que se puede observar una alimentación principal de 12V, destinada al sector de video y una alimentación secundaria de 5V, derivada de la primera, destinada al microcontrolador. Dichas alimentaciones se pueden obtener de las baterías mencionadas o en su defecto de la misma fuente que se utilizó para el NOPPP, únicamente se deben eliminar los diodos en el regulador de 12 V y conectar éste directamente a tierra.

En el ANEXO 2 se presentan los componentes del generador de patrones de video para realizar la construcción de este equipo. Con respecto al programa para el pic, se encuentra en su totalidad en el ANEXO 3, basta copiarlo adecuadamente en el MPLAB, ensamblarlo y cargarlo en el PIC, utilizando las herramientas descritas en los capítulos anteriores[80].

En el momento de volcar el programa en el PIC no olvidar poner la opción de operación con cristal "HS". De otro modo, el cristal no oscilará.

[80] Vease el Capitulo 7 y Capitulo 8 de esta tesis.

# CONCLUSIONES

## CONCLUSIONES

Con lo expuesto dentro de este trabajo se ha visto como la electrónica ha cambiado desde que apareció el microprocesador, y como ciertamente un microcontrolador del tipo PIC reúne las características necesarias para considerarlo como un elemento muy adecuado en aspectos de enseñanza de la electrónica. El grado de utilidad dependerá de las características de las posibles prácticas a realizar dentro de una institución.

El microcontrolador PIC16F84 es realmente una gran opción para su uso en el ámbito didáctico. A comparación de otros de características semejantes, el PIC16F84 cuenta con un set de instrucciones pequeño y fácil de aprender, además de la ventaja de su bajo costo y que el hardware y software necesarios para lograr el funcionamiento adecuado de él son muy económicos, en el caso del software resulta gratuito. Su memoria tipo FLASH y la tipo EEPROM, tienen mayor flexibilidad de uso, de tal forma que no necesita de instrumentos extras para su borrado.

Esta afirmación se realiza con base en lo visto en los capítulos anteriores, no por ello se debe excluir el posible uso de otro tipo de microcontroladores con mayor capacidad y con otras características. Sin embargo, según lo expuesto en esta tesis, se observa que la amplia información y el gran interés que existe sobre este controlador en el mundo electrónico son muy extensos y, por ende, es posible encontrar asesoría hasta por Internet.

En cuanto a la experimentación práctica, se logró llegar a los resultados esperados, apoyándose en la teoría de capítulos anteriores.

Este estudio, así como su importancia, pretenden ser relevantes tanto en la época en que se realizó (en la que los microcontroladores forman parte importante del mercado electrónico), como a futuro debido a que este tipo de componentes son mejorados y sustituidos en el mercado con frecuencia, además de que la tecnología permite avanzar más rápido en este tipo de temas. Por lo tanto en un futuro no muy lejano, pudiera ser que este dispositivo sea sustituido por otro mejor o de más alto rendimiento, pero suele suceder que los componentes que vendrán, tendrán un gran apoyo en las bases del que forma parte del estudio concerniente a esta tesis, el microcontrolador PIC16F84.

Antes de concluir, se espera que esta tesis llene las expectativas del lector y pueda entender el funcionamiento del dispositivos aquí tratado, esperamos que el lector pueda en un futuro aumentar y modificar el contenido, para que después lo muestre como seguimiento de un este trabajo, y así otros futuros lectores puedan entender y analizar el amplio mundo de los microcontroladores.

Por todo lo expuesto, la conclusión para esta tesis es:

"el microcontrolador PIC16F84 resulta muy adecuado para su uso en el ámbito didáctico"

Por lo tanto la hipótesis propuesta para esta tesis: se ha cumplido satisfactoriamente.

Cabe destacar que, debido al rápido avance de la electrónica, puede ser muy posible que en el mercado no encuentre el controlador PIC16F84 mencionado en esta tesis, sino el PIC16F84A-20. Sin embargo, lo expuesto hasta el momento resulta útil con éste último, sin hacer modificaciones considerables, solo tome en cuenta que en cualquier parte del texto en donde encuentre PIC16F84 Ud. deberá leer PIC16F84A.

**S. RODRIGUEZ**
**A. MONTIEL**

# ANEXOS

# ANEXOS

**Anexo 1**: Datos técnicos del fabricante (preliminar)[81].

| Cross reference of device specs for oscillator configurations and frequencies of operation (commercial devices) | |
|---|---|
| OSC | PIC16F84-10 |
| RC | VDD: 4.5V to 5.5V<br>IDD: 1.8 mA typ. at 5.5V<br>IPD: 1.0 μA typ. at 5.5V WDT dis<br>Freq: 4..0 MHz max. |
| XT | VDD: 4.5V to 5.5V<br>IDD: 1.8 mA typ. at 5.5V<br>IPD: 1.0 μA typ. at 5.5V WDT dis<br>FREQ: 4.0 MHZ MAX. |
| HS | VDD: 4.5V to 5.5V<br>IDD: 10 mA max. at 5.5V typ.<br>IPD: 1.0 μA typ. at 4.5V WDT<br>Freq: 10 MHz max. |
| LP | Do not use in LP mode |
| The shaded sections indicate oscillator selections which are tested for functionality, but not for MIN/MAX specifications. It is recommended that the user select the device type that ensures the specifications required. | |

| DC Characteristics | Standard Operating Conditions (unless otherwise stated) |
|---|---|
| Power Supply Pins | Operating temperature 0°C ≤ TA ≤ +70°C (commercial)<br>-40°C ≤ TA ≤ +85°C (industrial) |

[81] Considere que esta información es una selección de la que brinda el fabricante y no está completa. Para mayor información diríjase a la página de Microchip.

| Sym | Characteristic | Min | Typ† | Max | Units | Conditions |
|---|---|---|---|---|---|---|
| VDD | Supply Voltage | 4.0 | — | 6.0 | V | XT, RC and LP osc configuration |
| | | 4.5 | — | 5.5 | V | HS osc configuration |
| VDR | RAM Data Retention Voltage(1) | 1.5* | — | — | V | Device in SLEEP mode |
| VPOR | VDD start voltage to Ensure internal Power-on Reset signal | — | VSS | — | V | |
| SVDD | VDD rise rate to ensure Internal Power-on Reset signal | 0.05* | — | — | V/ms | |
| IDD | Supply Current(2) | | | | | RC and XT osc configuration(4) |
| | | — | 1.8 | 4.5 | mA | FOSC = 4.0 MHz, VDD = 5.5V |
| | | — | 7.3 | 10 | mA | FOSC = 4.0 MHz, VDD = 5.5V (During Flash programming) |
| | | | | | | HS osc configuration (PIC16F84-10) |
| | | — | 5 | 10 | mA | FOSC = 10 MHz, VDD = 5.5V |
| IPD | Power-down Current(3) | — | 7.0 | 28 | µA | VDD = 4.0V, WDT enabled, industrial |
| | | — | 1.0 | 14 | µA | VDD = 4.0V, WDT disabled, commercial |
| | | — | 1.0 | 16 | µA | VDD = 4.0V, WDT disabled, industrial |

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only

and are not tested.

Note

1: This is the limit to which VDD can be lowered in SLEEP mode without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors such as I/O pin loading and switching rate, oscillator type, internal code execution pattern, and temperature also have an impact on the current consumption.

The test conditions for all IDD measurements in active operation mode are:

OSC1=external square wave, from rail to rail; all I/O pins tristated, pulled to VDD, T0CKI = VDD,

MCLR' = VDD; WDT enabled/disabled as specified.

3: The power down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins in hi-impedance state and tied to VDD and VSS.

4: For RC osc configuration, current through Rext is not included. The current through the resistor can be estimated by the formula IR = VDD/2Rext (mA) with Rext in kOhm.

| DC Characteristics | Standard Operating Conditions (unless otherwise stated) | | | | |
|---|---|---|---|---|---|
| All Pins Except | Operating temperature 0°C ≤ TA ≤ +70°C (commercial) | | | | |
| Power Supply Pins | -40°C ≤ TA ≤ +85°C (industrial) | | | | |
| **Sym** | **Characteristic** | **Min** | **Typ†** | **Max** | **Units** | **Conditions** |
| VIL | Input Low Voltage | | | | | |
| | I/O ports | | | | | |
| | with TTL buffer | VSS | — | 0.8 | V | 4.5 V ≤ VDD ≤ 5.5 V( |
| | | VSS | — | 0.16VDD | V | entire range(4) |
| | with Schmitt Trigger buffer | VSS | — | 0.2VDD | V | entire range |
| | MCLR', RA4/T0CKI | VSS | — | 0.2VDD | V | |
| | OSC1 (XT, HS and LP modes)( | VSS | — | 0.3VDD | V | |
| | OSC1 (RC mode) | VSS | — | 0.1VDD | V | |
| VIH | Input High Voltage | | | | | |
| | I/O ports | | | | | |
| | with TTL buffer | 2.4 | — | VDD | V | 4.5 V ≤ VDD ≤ 5.5V( |
| | | 0.48VDD | — | VDD | V | entire range(4) |
| | with Schmitt Trigger buffer | 0.45VDD | — | VDD | | entire range |
| | MCLR', RA4/T0CKI, OSC1 (RC mode) | 0.85VDD | — | VDD | V | |
| | OSC1 (XT, HS and LP modes)(1) | 0.7 VDD | — | VDD | V | |
| VHYS | Hysteresis of Schmitt Trigger inputs | TBD | — | — | V | |
| IPURB | PORTB weak pull-up current | 50* | 250* | 400* | µA | VDD = 5.0V, VPIN = VSS |
| IIL | Input Leakage Current( | | | | | |
| | I/O ports | — | — | ±1 | µA | VSS ≤ VPIN ≤ VDD, Pin at hi-impedance |
| | | — | — | | | |
| | MCLR', RA4/T0CKI | — | — | ±5 | µA | VSS ≤ VPIN ≤ VDD |
| | OSC1 | | | ±5 | µA | VSS ≤ VPIN ≤ VDD, XT, HS and LP osc configuration |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Output Low Voltage | | | | | |
| VOL | I/O ports | — | — | 0.6 | V | IOL = 8.5 mA, VDD = 4.5V |
| | OSC2/CLKOUT | — | — | 0.6 | V | IOL = 1.6 mA, VDD = 4.5V |
| | | | | | | |
| VOH | Output High Voltage | | | | | |
| | I/O ports(3) | VDD-0.7 | — | — | V | IOH = -3.0 mA, VDD = 4.5V |
| | OSC2/CLKOUT | VDD-0.7 | — | — | V | IOH = -1.3 mA, VDD = 4.5V |
| | **Data EEPROM Memory** | | | | | |
| ED | Endurance | 1M | 10M | — | E/W | 25°C at 5V |
| VDRW | VDD for read/write | VMIN | — | 6.0 | V | VMIN = Minimum operating voltage |
| TDEW | Erase/Write cycle time | — | 10 | 20* | ms | |
| | **Program Flash Memory** | | | | | |
| EP | Endurance | 100 | 1000 | — | E/W | |
| VPR | VDD for read | VMIN | — | 6.0 | V | VMIN = Minimum operating voltage |
| VPEW | | 4.5 | — | 5.5 | V | |
| TPEW | VDD for erase/write | — | 10 | — | ms | |
| | Erase/Write cycle time | | | | | |

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only

and are not tested.

Note

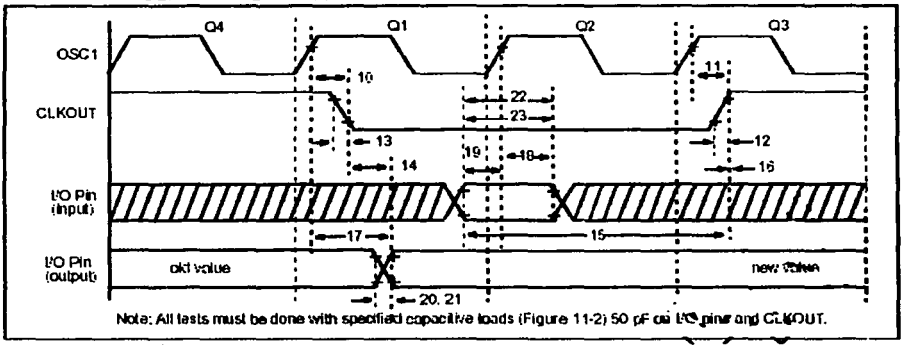
1: In RC oscillator configuration, the OSC1 pin is a Schmitt Trigger input. Do not drive the PIC16F8X with an external clock while the device is in RC mode, or chip damage may result.

2: The leakage current on the MCLR' pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as coming out of the pin.

4: The user may choose the better of the two specs.

**CLKOUT AND I/O TIMING**




Note: All tests must be done with specified capacitive loads (Figure 11-2) 50 pF on I/O pins and CLKOUT.

| Parameter No. | Sym | Characteristic | Min | Typ† | Max | Units | Coditions |
|---|---|---|---|---|---|---|---|
| 10 | TosH2ckL | OSC1↑ to CLKOUT↓ | - | 15 | 30 | ns | Note 1 |
| 11 | TosH2ckH | OSC1↑ to CLKOUT↑ | - | 15 | 30 | ns | Note 1 |
| 12 | TckR | CLKOUT rise time | - | 15 | 30 | ns | Note 1 |
| 13 | TckF | CLKOUT fall time | - | 15 | 30 | ns | Note 1 |
| 14 | TckL2ioV | CLKOUT ↓ to Port out valid | - | - | 0.5Tcy+20 | ns | Note 1 |
| 15 | TioVckH | Port in valid before CLKOUT ↑ | 0.30Tcy+30 | - | - | ns | Note 1 |
| 16 | TokH2iol | Port in hold after CLKOUT ↑ | 0 | - | - | ns | Note 1 |
| 17 | TosH2ioV | OSC1↑ (Q1 cycle) Port out valid | - | - | 125 | ns | |
| 18 | TosH2iol | OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time) | 10 | - | - | ns | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 19 | TioV2osH | Port input valid to OSC1↑ (I/O in setup time) | -75 | - | - | ns | |
| 20 | TioR | Port output rise time | - | 10 | 35 | ns | |
| 21 | TioF | Port output fall time | - | 10 | 35 | ns | |
| 22 | Tinp | INT pin high or low time | 20 | - | - | ns | |
| 23 | Trbp | RB7:RB4 change INT high or low time | Tosc | - | - | ns | |

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

**Note 1:** Measurements are taken in RC Mode where CLKOUT output is 4 x TOSC.

**ANEXO 2:** Materiales para el generador de patrones de video.

| | |
|---|---|
| R01 | 3K9 |
| R02 | 3K9 |
| R03 | 3K9 |
| R04 | 1K |
| R05 | 1K |
| R06 | 1K |
| R07 | 1K |
| R08 | 1K |
| R09 | 1K |
| R10 | 68K |
| R11 | 82K |
| R12 | 10K |
| R13 | 2K2 |
| R14 | 4K7 |
| R15 | 2K7 |
| R16 | 100 |
| C01 | 0.1µ |
| C02 | 100µ/16V |
| C03 | 100µ/16V |
| C04 | 1500p |
| C05 | 10µ/25V |
| C06 | 10µ/25V |
| C07 | 10µ/25V |
| C08 | .02µ |
| C09 | .01µ |
| C10 | 15p |
| C11 | 15p |
| C12 | 0.1µ |

| | |
|---|---|
| C13 | 0.1µ |
| C14 | 220p |
| C15 | 0.1µ |
| C16 | 18p |
| C17 | 150p |
| C18 | .02µ |
| C19 | 100µ/16V |
| C20 | 0.1µ |
| C21 | 0.1µ |
| C22 | 100µ/16V |
| CV1 | TRIMMER 5-45p |
| D1 | 1N4007 |
| Q1 | BF494C |
| U1 | PIC16F84-10 |
| U2 | MC1377 |
| U3 | LM78L05 |
| X1 | 10.000MHz |
| X2 | 3.582056MHz |
| S1 | LLAVE 2 POSICIONES |
| S2 | LLAVE 2 POSICIONES |
| S3 | LLAVE 2 POSICIONES |
| S4 | LLAVE 2 POSICIONES |
| S5 | LLAVE 2 POSICIONES |
| S6 | LLAVE 2 POSICIONES |
| S7 | LLAVE 2 POSICIONES |
| S8 | LLAVE 2 POSICIONES |
| S9 | LLAVE 2 POSICIONES |
| BT1 | 8 x AA PILAS ALCALINAS para hacerlo portátil |

**ANEXO 3**: Programa para el pic del generador de video

```
; ***** GENERADOR DE PATRONES PARA VIDEO *****

LIST   P=16F84A
__config      02            ;CODIGO PROTEGIDO, SIN WDT, CON PWRT Y OSCILADOR HS
;DEFINICION DE PUERTOS:
      ;PORTB(0): SYNC
      ;PORTB(2): AZUL
      ;PORTB(3): ROJO
      ;PORTB(4): VERDE
;NO USAR EL BIT 1 DEL PORTB
;PARA LOS PATRONES DE RASTER Y BARRAS EL VIDEO ES ENTRELAZADO
;LOS PUNTOS Y EL CROSSHATCH SE HACEN CON VIDEO NO ENTRELAZADO PARA EVITAR
;EL"FLICKER"

            CBLOCK 0X0C           ;VARIABLES

DURHOR,CANTHB1,CANTHB2,BLKLIN,CANTPRE,DUREQU,CANTVER,DURVER,CANTPOS
TIEMPO,FIELD,CARRY,WHITE,YELLOW,CYAN,GREEN,MAGEN,RED,BLUE,BLACK,CANTLIN
      ENDC

PORTA      EQU  5
TRISA      EQU  85H
PORTB      EQU  6
TRISB      EQU  86H
STATUS     EQU  3
RP0        EQU  5
BLANCO     EQU  B'00011101'
AMARIL     EQU  B'00011001'
CYANO      EQU  B'00010101'
VERDE      EQU  B'00010001'
MAGENT     EQU  B'00001101'
ROJO       EQU  B'00001001'
AZUL       EQU  B'00000101'
NEGRO      EQU  B'00000001'
;
      CLRF       PORTA          ;TODOS LOS BITS EN 0
      CLRF       PORTB          ;TODOS LOS BITS EN 0
      BSF        STATUS,RP0     ;SELECCIONA BANCO DE REGISTROS 1
      MOVLW      B'11111111'
      MOVWF      TRISA          ;TODOS LOS BITS DEL PUERTO A COMO ;ENTRADAS
      CLRF       TRISB^80H      ;TODOS LOS BITS DEL PUERTO B COMO SALIDA
```

```
        BCF     STATUS,RP0      ;SELECCIONA BANCO DE REGISTROS 0
;
        MOVLW   0
        MOVWF   CARRY           ;VARIABLE CONTROLAR EL ESTADO DEL CARRY
        RRF     CARRY           ;CARRY FLAG A "0"
        MOVLW   B'10101010'
        MOVWF   FIELD           ;CONTROL DEL CAMPO
LECTURA
        BTFSS   PORTA,3         ;SE LEE EL TECLADO
        GOTO    LECT1           ;SE USAN LOS BITS 2 Y 3 DEL PUERTO A
        BTFSC   PORTA,2         ;FUNCION: BIT3 BIT2
        GOTO    INICIO3         ;BARRAS 0 0
        GOTO    INICIO2         ;RASTER 0 1
LECT1
        BTFSC   PORTA,2         ;CROSSHATCH 1 0
        GOTO    INICIO1         ;PUNTOS 1 1

;***** BARRAS DE COLOR *****

INICIO
        RRF     FIELD           ;CARRY PASA AL BIT 7 DE FIELD, BIT 0 AL CARRY
        MOVLW   D'11'           ;LINEAS SIN VIDEO LUEGO DE LA POSECUALIZACION.
        BTFSS   FIELD,0         ;SI ES EL CAMPO 1 SE HACEN SOLO 11 LINEAS
        MOVLW   D'12'           ;12 LINEAS EN EL CAMPO 2
        MOVWF   BLKLIN
        MOVLW   D'120'
        MOVWF   CANTHB1         ;CANTIDAD DE LINEAS HORIZONTALES EN UN BLOQUE
        MOVLW   D'2'
        MOVWF   CANTHB2         ;CANTIDAD DE BLOQUES
        MOVLW   5
        MOVWF   CANTPRE         ;PULSOS DE PREECUALIZACION
        MOVLW   5
        MOVWF   CANTVER         ;PULSOS DE SINCRONISMO VERTICAL
        MOVLW   5
        MOVWF   CANTPOS         ;PULSOS DE POSECUALIZACION
PREEQU
        BCF     PORTB,0         ;DURACION: 2,4µs ABAJO
        MOVLW   D'23'
        MOVWF   DUREQU
        NOP
        NOP
        NOP
```

```
        BSF     PORTB,0
LOOP1
        DECFSZ  DUREQU          ;SE COMPLETAN LOS 32µs ARRIBA
        GOTO    LOOP1
        NOP
        NOP
        DECFSZ  CANTPRE
        GOTO    PREEQU
NOP
VERT
        BCF     PORTB,0
        MOVLW   D'22'
        MOVWF   DURVER
LOOP2
        DECFSZ  DURVER
        GOTO    LOOP2
        BSF     PORTB,0         ;DURACION: 4.8µs ARRIBA ("SERRATED PULSES")
        MOVLW   2
        MOVWF   TIEMPO
TIME
        DECFSZ  TIEMPO
        GOTO    TIME
        NOP
        DECFSZ  CANTVER
        GOTO    VERT
        NOP
POSEQU
        BCF     PORTB,0
        MOVLW   D'23'
        MOVWF   DUREQU
        NOP
        NOP
        NOP
        BSF     PORTB,0
LOOP3
        DECFSZ  DUREQU
        GOTO    LOOP3
        NOP
        NOP
        DECFSZ  CANTPOS
        GOTO    POSEQU
        NOP
```

```
;SE EMPIEZAN A BARRER LAS LINEAS HORIZONTALES
;LA PRIMERA LINEA ES COMPLETA EN EL CAMPO 1, EN TANTO QUE ES SOLO MEDIA LINEA
;EN EL CAMPO 2, Y NO COMIENZA CON UN PULSO DE SINCRONISMO

        RLF     PORTB       ;1 O 1/2 LINEA H SEGUN EL CAMPO
        NOP                 ;SE PASA EL CARRY AL BIT 0 DEL PUERTO B
        NOP                 ;CAMPO 1: 1 LINEA Y PULSO DE SINC (CARRY=0).
        NOP                 ;CAMPO 2: 1/2 LINEA SIN PULSO DE SINC (CARRY=1)
        NOP
        NOP
        NOP
        NOP
        MOVLW   D'21'       ;TIEMPO PARA 1/2 H (80 CICLOS TOTAL)
        BTFSS   PORTB,0     ;SI HAY H SYNC (CAMPO 1) SE AGREGA MAS TIEMPO
        ADDLW   D'27'       ;TIEMPO PARA 1 H (160 CICLOS TOTAL)
        MOVWF   DURHOR
        BSF     PORTB,0
        BTFSS   FIELD,0
        GOTO    NEXT        ;SE PIERDE 1 CICLO MAS (SOLO 1/2 H)
NEXT
        BCF     PORTB,1
        NOP
LOOP
        DECFSZ  DURHOR
        GOTO    LOOP
        NOP

;SE HACEN 11 O 12 LINEAS PARA CUMPLIR CON  525 LINEAS DE LA NORMA NTSC
;SI ES EL CAMPO 1 SE HACEN SOLO 11 LINEAS, YA QUE ANTES SE HIZO 1 DE MAS

HORIZ
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
TIME3
        DECFSZ  TIEMPO
        GOTO    TIME3
        NOP
        NOP
        MOVLW   D'48'
        MOVWF   DURHOR
```

```
        BSF     PORTB,0         ;BIT 0 ALTO
LOOPH3
        DECFSZ  DURHOR
        GOTO    LOOPH3
        NOP
        DECFSZ  BLKLIN
        GOTO    HORIZ
        NOP
;SE HACEN 2 BLOQUES DE 120 LINEAS HORIZONTALES
;2*(120+1)=242 LINEAS

HORIZ1
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
TIME1
        DECFSZ  TIEMPO
        GOTO    TIME1
        NOP
        NOP
        MOVLW   D'31'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        MOVLW   5
        MOVWF   WHITE
        MOVWF   YELLOW
        MOVWF   CYAN
        MOVWF   GREEN
        MOVWF   MAGEN
        MOVWF   RED
        MOVWF   BLUE
        MOVWF   BLACK
        MOVLW   BLANCO
        MOVWF   PORTB
WHITE1
        DECFSZ  WHITE
        GOTO    WHITE1
        MOVLW   AMARIL
        MOVWF   PORTB
```

```
YELLO1
        DECFSZ      YELLOW
        GOTO        YELLO1
        MOVLW       CYANO
        MOVWF       PORTB
CYAN1
        DECFSZ      CYAN
        GOTO        CYAN1
        MOVLW       VERDE
        MOVWF       PORTB
GREEN1
        DECFSZ      GREEN
        GOTO        GREEN1
        MOVLW       MAGENT
        MOVWF       PORTB
MAGEN1
        DECFSZ      MAGEN
        GOTO        MAGEN1
        MOVLW       ROJO
        MOVWF       PORTB
RED1
        DECFSZ      RED
        GOTO        RED1
        MOVLW       AZUL
        MOVWF       PORTB
BLUE1
        DECFSZ      BLUE
        GOTO        BLUE1
        MOVLW       NEGRO
        MOVWF       PORTB
BLACK1
        DECFSZ      BLACK
        GOTO        BLACK1
        NOP
        NOP
        NOP
        NOP
        DECFSZ      CANTHB1
        GOTO        HORIZ1
        NOP
HORIZ2
        BCF         PORTB,0         ;BIT 0 BAJO
```

```
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
TIME2
        DECFSZ  TIEMPO
        GOTO    TIME2
        MOVLW   D'120'
        MOVWF   CANTHB1
        MOVLW   D'31'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        MOVLW   5
        MOVWF   WHITE
        MOVWF   YELLOW
        MOVWF   CYAN
        MOVWF   GREEN
        MOVWF   MAGEN
        MOVWF   RED
        MOVWF   BLUE
        MOVWF   BLACK
        MOVLW   BLANCO
        MOVWF   PORTB
WHITE2
        DECFSZ  WHITE
        GOTO    WHITE2
        MOVLW   AMARIL
        MOVWF   PORTB
YELLO2
        DECFSZ  YELLOW
        GOTO    YELLO2
        MOVLW   CYANO
        MOVWF   PORTB
CYAN2
        DECFSZ  CYAN
        GOTO    CYAN2
        MOVLW   VERDE
        MOVWF   PORTB
GREEN2
        DECFSZ  GREEN
        GOTO    GREEN2
```

```
        MOVLW   MAGENT
        MOVWF   PORTB
MAGEN2
        DECFSZ  MAGEN
        GOTO    MAGEN2
        MOVLW   ROJO
        MOVWF   PORTB
RED2
        DECFSZ  RED
        GOTO    RED2
        MOVLW   AZUL
        MOVWF   PORTB
BLUE2
        DECFSZ  BLUE
        GOTO    BLUE2
        MOVLW   NEGRO
        MOVWF   PORTB
BLACK2
        DECFSZ  BLACK
        GOTO    BLACK2
        NOP
        NOP
        NOP
        NOP
        DECFSZ  CANTHB2
        GOTO    HORIZ1
        NOP

;ESTA ULTIMA LINEA/MEDIA LINEA,  LA USO PARA CARGAR VARIABLES

        BCF     PORTB,0         ;BIT 0 PASA A NIVEL BAJO
        NOP                     ;PIERDO TIEMPO PARA HACER LOS 4,8µs
        NOP
        MOVLW   0               ;NO USAR EL BIT 1 DEL PORTB, BIT 0 = SYNC
        BTFSC   FIELD,0
        MOVLW   1
        MOVWF   CARRY
        NOP
        MOVLW   D'15'
        BTFSC   FIELD,0
        ADDLW   D'24'
        MOVWF   DURHOR
```

```
        BSF     PORTB,0         ;BIT 0 PASA A NIVEL ALTO
        BTFSS   FIELD,0
        GOTO    NEXT1
NEXT1
        NOP
        NOP
LOOPH5
        DECFSZ  DURHOR
        GOTO    LOOPH5
        RRF     CARRY           ;CARRY = 1 SI 1 H, CARRY = 0 SI 1/2 H
        BTFSC   PORTA,2
        GOTO    LECTURA
        BTFSC   PORTA,3
        GOTO    LECTURA
        NOP
        NOP
        NOP
        NOP
        NOP
        GOTO    INICIO

;***** RASTER *****

INICIO1
        RRF     FIELD
        MOVLW   D'11'
        BTFSS   FIELD,0
        MOVLW   D'12'
        MOVWF   BLKLIN
        MOVLW   D'120'
        MOVWF   CANTHB1
        MOVLW   D'2'
        MOVWF   CANTHB2
        MOVLW   5
        MOVWF   CANTPRE
        MOVLW   5
        MOVWF   CANTVER
        MOVLW   5
        MOVWF   CANTPOS
APREEQU
        BCF     PORTB,0         ;DURACION: 2.4µs ABAJO
        MOVLW   D'23'
```

```
        MOVWF   DUREQU
        NOP
        NOP
        NOP
        BSF     PORTB,0
ALOOP1
        DECFSZ  DUREQU          ;SE COMPLETAN LOS 32µs ARRIBA
        GOTO    ALOOP1
        NOP
        NOP
        DECFSZ  CANTPRE
        GOTO    APREEQU
        NOP
AVERT
        BCF     PORTB,0
        MOVLW   D'22'
        MOVWF   DURVER
ALOOP2
        DECFSZ  DURVER
        GOTO    ALOOP2
        BSF     PORTB,0         ;DURACION: 4.8µs ARRIBA ("SERRATED PULSES")
        MOVLW   2
        MOVWF   TIEMPO
ATIME
        DECFSZ  TIEMPO
        GOTO    ATIME
        NOP
        DECFSZ  CANTVER
        GOTO    AVERT
        NOP
APOSEQU
        BCF     PORTB,0
        MOVLW   D'23'
        MOVWF   DUREQU
        NOP
        NOP
        NOP
        BSF     PORTB,0
ALOOP3
        DECFSZ  DUREQU
        GOTO    ALOOP3
        NOP
```

```
        NOP
        DECFSZ      CANTPOS
        GOTO        APOSEQU
        NOP
        RLF         PORTB           ;1 O 1/2 LINEA H SEGUN EL CAMPO
        NOP
        NOP
NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW       D'21'           ;TIEMPO PARA 1/2 H (80 CICLOS TOTAL)
        BTFSS       PORTB,0
        ADDLW       D'27'           ;TIEMPO PARA 1 H (160 CICLOS TOTAL)
        MOVWF       DURHOR
        BSF         PORTB,0
        BTFSS       FIELD,0
        GOTO        ANEXT           ;SE PIERDE 1 CICLO MAS (SOLO 1/2 H)
ANEXT
        BCF         PORTB,1
        NOP
ALOOP
        DECFSZ      DURHOR
        GOTO        ALOOP
        NOP
AHORIZ
        BCF         PORTB,0
        MOVLW       2
        MOVWF       TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
ATIME3
        DECFSZ      TIEMPO
        GOTO        ATIME3
        NOP
        NOP
        MOVLW       D'48'
        MOVWF       DURHOR
        BSF         PORTB,0         ;BIT 0 ALTO
ALOOPH3
        DECFSZ      DURHOR
        GOTO        ALOOPH3
        NOP
```

```
        DECFSZ  BLKLIN
        GOTO    AHORIZ
        NOP
AHORIZ1
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
ATIME1
        DECFSZ  TIEMPO
        GOTO    ATIME1
        NOP
        NOP
        MOVLW   D'44'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   B'00011101'
        MOVWF   PORTB
ALOOPH4
        DECFSZ  DURHOR
        GOTO    ALOOPH4
        MOVLW   B'00000001'
        MOVWF   PORTB
        DECFSZ  CANTHB1
        GOTO    AHORIZ1
        NOP
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
ATIME2
        DECFSZ  TIEMPO
        GOTO    ATIME2
        MOVLW   D'120'
        MOVWF   CANTHB1
```

```
        MOVLW   D'44'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   B'00011101'
        MOVWF   PORTB
ALOOPH5
        DECFSZ  DURHOR
        GOTO    ALOOPH5
        MOVLW   B'00000001'
        MOVWF   PORTB
        DECFSZ  CANTHB2
        GOTO    AHORIZ1
        NOP

;ESTA ULTIMA LINEA/MEDIA LINEA,  LA USO PARA CARGAR VARIABLES

        BCF     PORTB,0         ;BIT 0 PASA A NIVEL BAJO
        NOP                     ;PIERDO TIEMPO PARA
        NOP                     ;HACER LOS 4,8µS
        MOVLW   0               ;NO USAR EL BIT 1 DEL PORTB, BIT 0 = SYNC
        BTFSC   FIELD,0
        MOVLW   1
        MOVWF   CARRY
        NOP
        MOVLW   D'15'
        BTFSC   FIELD,0
        ADDLW   D'24'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 PASA A NIVEL ALTO
        BTFSS   FIELD,0
        GOTO    ANEXT1
ANEXT1
        NOP
```

```
        NOP
ALOOPH6
        DECFSZ      DURHOR
        GOTO        ALOOPH6
        RRF         CARRY           ;CARRY = 1 SI 1 H, CARRY = 0 SI 1/2 H
        BTFSS       PORTA,2
        GOTO        LECTURA
        BTFSC       PORTA,3
        GOTO        LECTURA
        NOP
        NOP
        NOP
        NOP
        NOP
        GOTO        INICIO1

;***** CROSSHATCH *****

INICIO2
        RRF         FIELD
        NOP
        NOP
        MOVLW       D'11'
        MOVWF       BLKLIN
        MOVLW       D'25'
        MOVWF       CANTHB1
        MOVLW       D'9'
        MOVWF       CANTHB2
        MOVLW       4
        MOVWF       CANTPRE         ;SOLO 4 PULSOS POR SER VIDEO NO ENTRELAZADO
        MOVLW       5
        MOVWF       CANTVER
        MOVLW       5
        MOVWF       CANTPOS
BPREEQU
        BCF         PORTB,0         ;DURACION: 2.4µs ABAJO
        MOVLW       D'23'
        MOVWF       DUREQU
        NOP
        NOP
        NOP
        BSF         PORTB,0
```

```
BLOOP1
        DECFSZ    DUREQU          ;SE COMPLETAN LOS 32µs ARRIBA
        GOTO      BLOOP1
        NOP
        NOP
        DECFSZ    CANTPRE
        GOTO      BPREEQU
        NOP
BVERT
        BCF       PORTB,0
        MOVLW     D'22'
        MOVWF     DURVER
BLOOP2
        DECFSZ    DURVER
        GOTO      BLOOP2
        BSF       PORTB,0         ;DURACION: 4.8µs ARRIBA ("SERRATED PULSES")
        MOVLW     2
        MOVWF     TIEMPO
BTIME
        DECFSZ    TIEMPO
        GOTO      BTIME
        NOP
        DECFSZ    CANTVER
        GOTO      BVERT
        NOP
BPOSEQU
        BCF       PORTB,0
        MOVLW     D'23'
        MOVWF     DUREQU
        NOP
        NOP
        NOP
        BSF       PORTB,0
BLOOP3
        DECFSZ    DUREQU
        GOTO      BLOOP3
        NOP
        NOP
        DECFSZ    CANTPOS
        GOTO      BPOSEQU
        NOP
        NOP                       ;1/2 LINEA H (NO ENTRELAZADO)
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   D'21'           ;TIEMPO PARA 1/2 H (80 CICLOS TOTAL)
        NOP
        NOP
        MOVWF   DURHOR
        NOP
        NOP
        NOP
        NOP
        BCF     PORTB,1
        NOP
BLOOP
        DECFSZ  DURHOR
        GOTO    BLOOP
        NOP
BHORIZ
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
BTIME3
        DECFSZ  TIEMPO
        GOTO    BTIME3
        NOP
        NOP
        MOVLW   D'48'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
BLOOPH3
        DECFSZ  DURHOR
        GOTO    BLOOPH3
        NOP
        DECFSZ  BLKLIN
        GOTO    BHORIZ
        NOP
BHORIZ1
        BCF     PORTB,0
```

```
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
BTIME1
        DECFSZ  TIEMPO
        GOTO    BTIME1
        NOP
        NOP
        MOVLW   9
        MOVWF   CANTLIN
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
BLOOPHA
        MOVLW   B'00011100'
        ADDWF   PORTB
        SUBWF   PORTB
        NOP
        MOVLW   2
        MOVWF   DURHOR
BLOOPH4
        DECFSZ  DURHOR
        GOTO    BLOOPH4
        DECFSZ  CANTLIN
        GOTO    BLOOPHA
        NOP
        MOVLW   B'00011100'
        ADDWF   PORTB
        SUBWF   PORTB
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DECFSZ  CANTHB1
```

```
        GOTO    BHORIZ1
        NOP
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
BTIMEZ
        DECFSZ  TIEMPO
        GOTO    BTIMEZ
        NOP
        NOP
        MOVLW   D'44'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   B'00011101'
        MOVWF   PORTB
BLOOPHZ
        DECFSZ  DURHOR
        GOTO    BLOOPHZ
        MOVLW   B'00000001'
        MOVWF   PORTB
        NOP
        NOP
        NOP
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
BTIME2
        DECFSZ  TIEMPO
        GOTO    BTIME2
        MOVLW   D'25'
        MOVWF   CANTHB1
        MOVLW   D'44'
        MOVWF   DURHOR
```

```
        BSF     PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   B'00011101'
        MOVWF   PORTB
BLOOPH5
        DECFSZ  DURHOR
        GOTO    BLOOPH5
        MOVLW   B'00000001'
        MOVWF   PORTB
        DECFSZ  CANTHB2
        GOTO    BHORIZ1
        NOP

;ESTA ULTIMA MEDIA LINEA,  LA USO PARA CARGAR VARIABLES

        BCF     PORTB,0         ;BIT 0 PASA A NIVEL BAJO
        NOP                     ;PIERDO TIEMPO PARA HACER LOS 4,8µs
        NOP
        MOVLW   0               ;NO USAR EL BIT 1 DEL PORTB, BIT 0 = SYNC
        BTFSC   FIELD,0
        MOVLW   1
        MOVWF   CARRY
        NOP
        MOVLW   D'15'
        NOP
        NOP
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 PASA A NIVEL ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
BLOOPH6
```

```
        DECFSZ    DURHOR
        GOTO      BLOOPH6
        RRF       CARRY          ;CARRY = 1 SI 1 H, CARRY = 0 SI 1/2 H
        BTFSC     PORTA,2
        GOTO      LECTURA
        BTFSS     PORTA,3
        GOTO      LECTURA
        NOP
        NOP
        NOP
        NOP
        NOP
        GOTO      INICIO2

;***** PUNTOS *****

INICIO3
        RRF       FIELD
        NOP
        NOP
        MOVLW     D'11'
        MOVWF     BLKLIN
        MOVLW     D'25'
        MOVWF     CANTHB1
        MOVLW     D'9'
        MOVWF     CANTHB2
        MOVLW     4
        MOVWF     CANTPRE
        MOVLW     5
        MOVWF     CANTVER
        MOVLW     5
        MOVWF     CANTPOS
CPREEQU
        BCF       PORTB,0        ;DURACION: 2,4µs ABAJO
        MOVLW     D'23'
        MOVWF     DUREQU
        NOP
        NOP
        NOP
        BSF       PORTB,0
CLOOP1
        DECFSZ    DUREQU         ;SE COMPLETAN LOS 32µs ARRIBA
```

```
        GOTO    CLOOP1
        NOP
        NOP
        DECFSZ  CANTPRE
        GOTO    CPREEQU
        NOP
CVERT
        BCF     PORTB,0
        MOVLW   D'22'
        MOVWF   DURVER
CLOOP2
        DECFSZ  DURVER
        GOTO    CLOOP2
        BSF     PORTB,0         ;DURACION: 4.8µs ARRIBA ("SERRATED PULSES")
        MOVLW   2
        MOVWF   TIEMPO
CTIME
        DECFSZ  TIEMPO
        GOTO    CTIME
        NOP
        DECFSZ  CANTVER
        GOTO    CVERT
        NOP
CPOSEQU
        BCF     PORTB,0
        MOVLW   D'23'
        MOVWF   DUREQU
        NOP
        NOP
        NOP
        BSF     PORTB,0
CLOOP3
        DECFSZ  DUREQU
        GOTO    CLOOP3
        NOP
        NOP
        DECFSZ  CANTPOS
        GOTO    CPOSEQU
        NOP
        NOP                     ;1/2 LINEA H
        NOP
        NOP
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        MOVLW   D'21'           ;TIEMPO PARA 1/2 H (80 CICLOS TOTAL)
        NOP
        NOP
        MOVWF   DURHOR
        NOP
        NOP
        NOP
        NOP
        BCF     PORTB,1
        NOP
CLOOP
        DECFSZ  DURHOR
        GOTO    CLOOP
        NOP
CHORIZ
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
CTIME3
        DECFSZ  TIEMPO
        GOTO    CTIME3
        NOP
        NOP
        MOVLW   D'48'
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 ALTO
CLOOPH3
        DECFSZ  DURHOR
        GOTO    CLOOPH3
        NOP
        DECFSZ  BLKLIN
        GOTO    CHORIZ
        NOP
CHORIZ1
        BCF     PORTB,0
        MOVLW   2
        MOVWF   TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
```

```
CTIME1
        DECFSZ      TIEMPO
        GOTO        CTIME1
        NOP
        NOP
        MOVLW       D'48'
        MOVWF       DURHOR
        BSF         PORTB,0         ;BIT 0 ALTO
CLOOPHZ
        DECFSZ      DURHOR
        GOTO        CLOOPHZ
        NOP
        DECFSZ      CANTHB1
        GOTO        CHORIZ1
        NOP
        BCF         PORTB,0
        MOVLW       2
        MOVWF       TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
CTIMEZ
        DECFSZ      TIEMPO
        GOTO        CTIMEZ
        NOP
        NOP
        MOVLW       9
        MOVWF       CANTLIN
        BSF         PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
CLOOPHA
        MOVLW       B'00011100'
        ADDWF       PORTB
        SUBWF       PORTB
        NOP
        MOVLW       2
        MOVWF       DURHOR
```

```
CLOOPH4
        DECFSZ      DURHOR
        GOTO        CLOOPH4
        DECFSZ      CANTLIN
        GOTO        CLOOPHA
        NOP
        MOVLW       B'00011100'
        ADDWF       PORTB
        SUBWF       PORTB
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BCF         PORTB,0
        MOVLW       2
        MOVWF       TIEMPO          ;PIERDO TIEMPO PARA HACER LOS 4,8µs
CTIME2
        DECFSZ      TIEMPO
        GOTO        CTIME2
        MOVLW       D'25'
        MOVWF       CANTHB1
        MOVLW       9
        MOVWF       CANTLIN
        BSF         PORTB,0         ;BIT 0 ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
CLOOPHB
        MOVLW       B'00011100'
        ADDWF       PORTB
        SUBWF       PORTB
```

```
        NOP
        MOVLW   2
        MOVWF   DURHOR
CLOOPH5
        DECFSZ  DURHOR
        GOTO    CLOOPH5
        DECFSZ  CANTLIN
        GOTO    CLOOPHB
        NOP
        MOVLW   B'00011100'
        ADDWF   PORTB
        SUBWF   PORTB
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DECFSZ  CANTHB2
        GOTO    CHORIZ1
        NOP

;ESTA ULTIMA MEDIA LINEA, LA USO PARA CARGAR VARIABLES

        BCF     PORTB,0         ;BIT 0 PASA A NIVEL BAJO
        NOP                     ;PIERDO TIEMPO PARA HACER LOS 4,8µs
        NOP
        MOVLW   0               ;NO USAR EL BIT 1 DEL PORTB, BIT 0 = SYNC.BTFSC
FIELD,0
        MOVLW   1
        MOVWF   CARRY
        NOP
        MOVLW   D'15'
        NOP
        NOP
        MOVWF   DURHOR
        BSF     PORTB,0         ;BIT 0 PASA A NIVEL ALTO
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
CLOOPH6
        DECFSZ      DURHOR
        GOTO        CLOOPH6
        RRF         CARRY           ;CARRY = 1 SI 1 H, CARRY = 0 SI 1/2 H
        BTFSS       PORTA,2
        GOTO        LECTURA
        BTFSS       PORTA,3
        GOTO        LECTURA
        NOP
        NOP
        NOP
        NOP
        NOP
        GOTO        INICIO3
        END
```

Como nota adicional, no es necesario generar todo el programa, se pueden eliminar algunos bloques de patrones de video pero solo si conoce el funcionamiento adecuado del programa deberá hacer modificaciones o puede alterar todo el funcionamiento.

# BIBLIOGRÁFIA

## BIBLIOGRÁFIA


- "Análisis Básico de circuitos eléctricos", E. Jhonson, David. (traducción: Guillermo Lopéz Portillo Sánchez),. Prentice-Hall Hispanoamericana S.A., 1978
- "Aplicaciones de los microcontroladores PIC de Microchip", Angulo Usategui J. Ma., Cuenca E. Martín, Angulo Martínez J., McGraw Hill, 1998, México.
- "Curso completo de automatización industrial moderna", Martínez Sánchez Victoriano,. RA-MA, 1992,España.
- "Lineamientos generales para la elaboración de proyectos de investigación y para la redacción", Cid Capetillo, Iliana; de los Angelews Márquez, María; UNAM 1999 1a ed, México, Pag 11, 14.
- "Microcontroladores PIC, la solución en un chip", Angulo Usategui J. Ma., Cuenca E. Martín, Angulo Martínez J, Paraninfo, 1997, España.
- "Microcontroladores PIC", Vallejo Horacio D. Quark S.R.L., 2002, Argentina.
- "Microchip PIC Microcontrollers Data Book", Microchip Technology Inc., Microchip, 1997. U.S.A.
- "Saber electrónica No. 135", Vallejo Horacio D, Televisa S.A. de C.V, 2001, México.
- "Saber electrónica No. 136", H. Picerno, Alberto.,. Televisa S.A. de C.V, 2001, México
- "Televisión Básica 1", Van Valkenburg, Nooger & Neville, Inc.. Editorial Continental S. A, 1973, México.
- "Todo sobre PICs & microprocesadores y microcontroladores", Vallejo Horacio D, H. Picerno Alberto, Prado Federico, Rodríguez Luis H., Quark S.R.L., 2001, Argentina

## FUENTES DE INTERNET


- http://club.telepolis.com/fremiro/mplab.htm (España, 2002)
- http://garaje.ya.com/deore/Pics/pic1.htm (2002)
- http://pub75.ezboard.com (2002)
- http://www.fer.nu/electronica/mplab-sp.html
- http://www.hobbiepic.com/ (U.S.A, 2002)
- http://www.iespana.es/portosin/ (España, 2002)
- http://www.losa.com.uy (2002)
- http://www.microcihip.com (U.S.A, 2002)
- http://www.mikroelektronika.co.yu (2002)
- http://www.sct.gob.mx (México, 2002)
- http://www.superpic.com/MPLAB.htm (España, 2002)

# ÍNDICE DE TABLAS

## ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

## INDICE DE FIGURAS