



Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Programación y Desarrollo de Sistemas

Academia de Algoritmia y Programación



Programación de Microcontroladores PIC-Microchip



Araujo Díaz David

México, D.F. año 2004

Contenido

	Página
1. Introducción	1
1.1 Descripción del microcontrolador	1
1.2 Tipos de memoria ROM (Read Only Memory)(Memoria de programa)	2
1.3 Memoria de Datos RAM (Random Access Memory)	2
1.4 Líneas de E/S para los controladores de periféricos	2
1.5 Recursos auxiliares	3
1.6 Programación de microcontroladores	3
1.7 Arquitectura interna de un PIC	4
2. Microcontrolador	5
2.1 Arquitectura de los PIC	5
2.2 Memoria de programa	7
2.3 Contador de programa y pila	7
2.4 Memoria de datos RAM	7
2.5 Direccionamiento de la memoria de datos	9
2.6 Disposición de la terminales	10
3. Registros	11
3.1 Registro de estado (STATUS)	11
3.2 Registro de opciones (OPTION)	11
3.3 Perro Guardián (Watchdog)	12
3.4 Modo de reposo ó de bajo consumo (SLEEP)	12
3.5 Temporizadores	12
3.6 Ejemplo del uso de registros y puertos	14
4. Puertos de E/S y EEPROM	15
4.1 Frecuencia de funcionamiento (Reloj)	15
4.2 Tipos de osciladores	15
4.3 Reinicialización ó RESET	16
4.4 Puertos de E/S	16
4.5 Palabra de configuración	17
4.6 Palabra de identificación (ID)	18
4.7 Memoria de datos EEPROM	18
5. Recursos auxiliares	21
5.1 Interrupciones	21
5.2 Convertidor Analógico/Digital (A/D)	23
5.3 Receptor/Trasmisor serial	24
6. Aplicaciones	27
6.1 Luces secuenciales	27
6.2 Circuito de desarrollo para el PIC16X84	28
6.2.1 Codificación de caracteres para la tarjeta de desarrollo para el PIC16X84	29
6.2.2 Contador ascendente de 4 dígitos	30
6.2.3 Reloj digital de 12 horas	30
6.2.4 Marquesina de mensajes con alarma sonora	30
6.2.5 Contador ascendente/descendente de 4 dígitos	30
6.3 Cerradura electrónica con pantalla LCD	31
6.4 Generador de barras de gris (PIC16F84)	31

	Página
Apéndice A: Cuadro de Instrucciones	33
A.1 Resumen de instrucciones	33
A.1.1 Instrucciones que manejan registros	33
A.1.2 Instrucciones de control y operandos inmediatos	33
A.1.3 Instrucciones que manipulan bits	34
A.2 Cuadro de instrucciones	34
 Apéndice B: Programador de microcontroladores	 43
B.1 Circuitos electrónico	43
B.2 Circuito impreso	43
B.3 Componentes	44
B.4 Programa	44
B.5 Dispositivos	45
 Referencias	 47

Capítulo 1: Introducción

Los microcontroladores han ganado todos los espacios en nuestra vida cotidiana, los podemos encontrar en el trabajo, en el transporte, en el hogar, etc. debido a que son sencillos, modernos, rápidos baratos, se pueden escribir y borrar sus programas muchas veces, existe una buena documentación y las herramientas de programación y desarrollo resultan económicas. Los microcontroladores de **Microchip**, se presentan en una amplia variedad, lo que permite que sean empleados desde aplicaciones básicas hasta muy complejas empleando el mismo conjunto de instrucciones, lo que involucra la función de un ingeniero de diseño; el cual debe de elegir un modelo de microcontrolador que mejor satisfaga las necesidades de un proyecto con el mínimo presupuesto.

1.1 Descripción del microcontrolador

Diferencia entre Microprocesador y Microcontrolador

- Un **microprocesador (mP)** es un sistema abierto, con el cual es posible construir una computadora con las características que se desee, acoplando los módulos necesarios.
- Un **microcontrolador (mC)** es un sistema cerrado, contiene una computadora completa por lo que sus prestaciones están limitadas a lo que contiene y pocas veces se pueden modificar.

Partes de un mC

Las partes principales de un μ C son:

- Procesador
- Memoria no volátil que contiene el programa
- Memoria de lectura/escritura para almacenar datos
- Terminales de entrada/salida
 - Puertos paralelos
 - Puertos serie
 - Otras comunicaciones SPI, I²C, USB, etc.
- Recursos auxiliares
 - Circuitos de reloj
 - Temporizadores
 - Perro Guardián (Watchdog)
 - Convertidores A/D y D/A
 - Comparadores analógicos
 - Protección contra fallos de alimentación
 - Estado de reposo o de bajo consumo

Características de los mC

La arquitectura del procesador es del tipo **Harvard**, a diferencia de la arquitectura tradicional **von Neumann**, caracterizada por que la CPU se conecta con una memoria única donde existen datos e instrucciones a través de un sistema de buses (**Figura 1.1**).

En la **arquitectura Harvard** son independientes las memorias de instrucciones y la memoria de datos, por lo que a cada una se accede a través de su propio bus, por lo que se propicia el paralelismo (**Figura 1.1**).

Los μ C poseen una arquitectura **RISC** (Computadoras con un conjunto reducido de instrucciones), que se identifican por que poseen un pequeño conjunto de instrucciones, y que todas ellas se ejecutan en un solo ciclo de reloj (excepto las de salto que pueden tomar dos ciclos).

Como existe paralelismo, es posible descomponer por etapas cada instrucción, por lo que es posible trabajar con varias instrucciones a la vez, esto se conoce como **segmentación del procesador** (pipe-line).

Las **líneas de E/S** que se adaptan con los periféricos manejan información en paralelo y se agrupan en conjuntos de ocho, que reciben el nombre de **Puertos**. Hay modelos con líneas que soportan la comunicación en serie; otros disponen de conjuntos de líneas que implementan puertos de comunicación para diversos protocolos, como el **I²C** (Inter-Integrated Circuit), el **USB** (Universal Serial Bus), **USART** (Universal Synchronous Asynchronous Receiver / Transmitter), **SPI** (Serial Peripheral Interface), etc.

1.5 Recursos auxiliares

Según las aplicaciones a las que orienta el fabricante cada modelo de microcontrolador, incorpora una diversidad de complementos que refuerzan la potencia y la flexibilidad del dispositivo. Entre los recursos más comunes encontramos:

- **Circuito de reloj**, encargado de generar los impulsos que sincronizan el funcionamiento de todo el sistema.
- **Temporizadores**, orientados a controlar tiempos.
- **Perro Guardián** (*watchdog*), destinado a provocar una reinicialización cuando el programa queda bloqueado.
- **Conversores A/D y D/A**, para poder recibir y enviar señales analógicas.
- **Comparadores analógicos**, para verificar el valor de una señal analógica.
- Sistema de protección ante **fallos de la alimentación**.
- **Estado de Reposo**, en el que el sistema queda congelado y el consumo de energía se reduce al mínimo.

1.6 Programación de microcontroladores

La utilización de los lenguajes más cercanos a la máquina (de bajo nivel) representan un considerable ahorro de código en la confección de los programas, lo que es muy importante dada la estricta limitación de la capacidad de la memoria de instrucciones. Los programas bien realizados en **lenguaje ensamblador** optimizan el tamaño de la memoria que ocupan y su ejecución es muy rápida ([Apéndice A](#)).

Los lenguajes de alto nivel más empleados con microcontroladores son el **C** y el **BASIC**, de los que existen varias empresas que comercializan versiones de compiladores e intérpretes para diversas familias de microcontroladores.

Siempre que se diseña con circuitos integrados programables se precisan herramientas para la puesta a punto del hardware y del software.

Con referencia al software, además de los compiladores o intérpretes de los lenguajes usados, es muy interesante disponer de simuladores software, que consisten en programas que simulan la ejecución de instrucciones representando el comportamiento interno del procesador y el estado de las líneas de E/S. Como se simula por software al procesador, el comportamiento no es idéntico aunque proporciona una aproximación aceptable, especialmente cuando no es esencial el trabajo en tiempo real.

Microchip pone libremente a disposición de sus usuarios, a través de Internet (www.microchip.com), ensambladores como el **MPASM** y simuladores como el **MPSIM**.

Respecto a las **herramientas hardware**, una indispensable es el grabador, encargado de escribir el programa en la memoria del microcontrolador. Existen grabadores muy completos capaces de trabajar con muchos modelos de diferentes familias, pero su elevado precio los aleja de los usuarios personales. Para estos últimos existen bastantes versiones de sencillos grabadores, específicos para ciertos modelos de microcontroladores, que son gobernados desde una computadora personal ([Apéndice B](#)).

La elección de los **microcontroladores PIC** (Programmable Integrated Circuit) de **Microchip**, es por que son sencillos de utilizar, existe buena información, presentan un buen precio, tienen un buen promedio de parámetros (velocidad, consumo, alimentación, código compacto, etc.), existen herramientas de desarrollo accesibles y permiten un diseño rápido.

1.7 Arquitectura interna de un PIC

El diagrama a bloques de los microcontroladores **PIC16F873** y **PIC16F876** se muestra en la **Figura 1.2**.

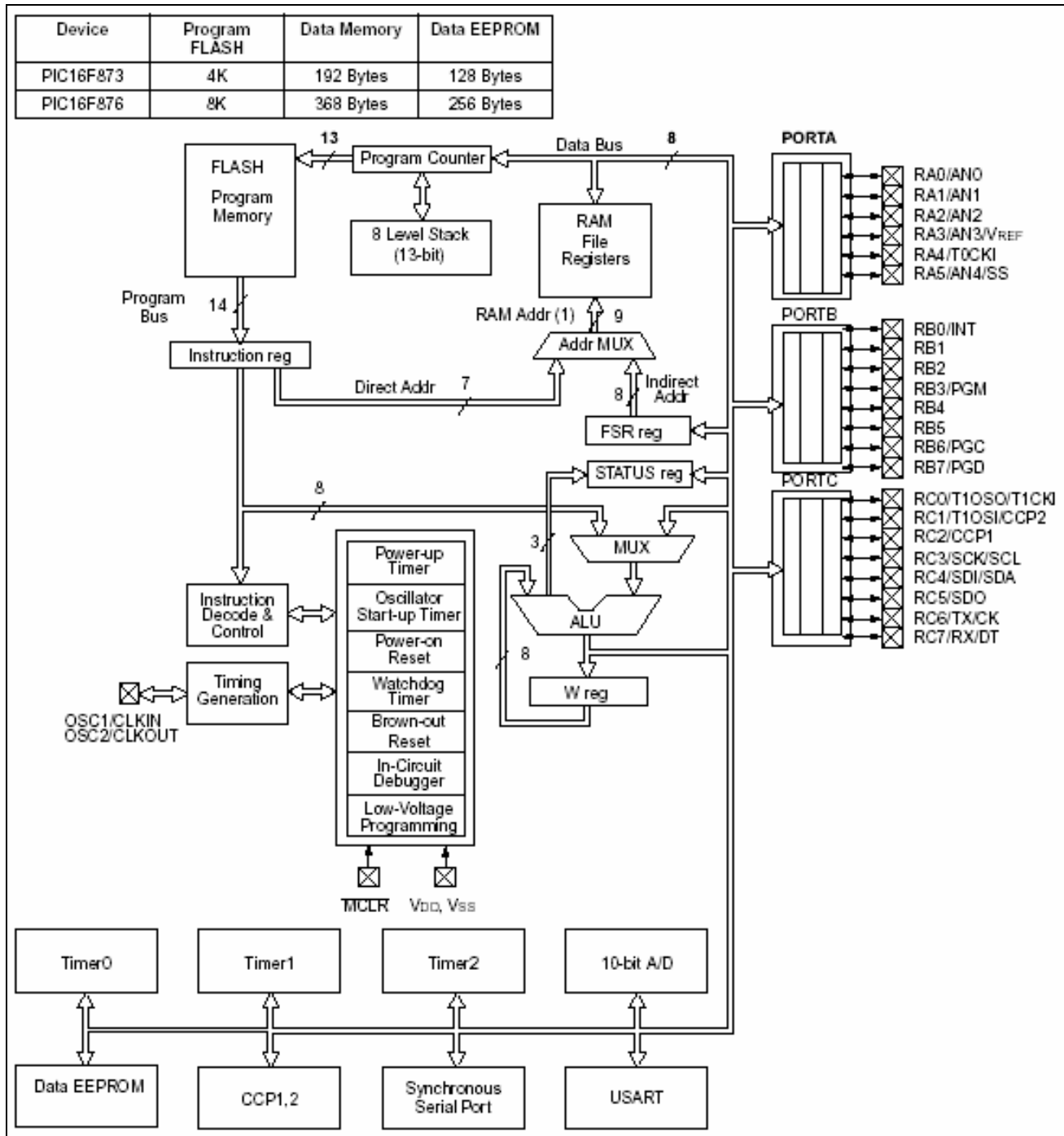


Figura 1.2: Diagrama a bloques de los microcontroladores **PIC16F873** y **PIC16F876**.

Capítulo 2: Microcontrolador

2.1 Arquitectura de los PIC

Para lograr una compactación de código óptima y una velocidad superior a la de sus competidores los microcontroladores PIC incorporan en su procesador tres de las características más avanzadas en los grandes computadoras:

- Procesador **RISC**.
- Procesador **Segmentado**.
- Arquitectura **Harvard**.

Con la incorporación de estos recursos los PIC son capaces de ejecutar en **un ciclo** de instrucción todas las instrucciones, excepto las de salto, que tardan el doble. Una condición imprescindible es la simetría y ortogonalidad en el formato de las instrucciones, que en el caso de los PIC de la gama media tienen una longitud de 14 bits. De esta forma se consigue una compactación en el código del programa para un **PIC16X84** **2.24** veces superior al de un **68HC05**, funcionando a la misma frecuencia.

El cuadro de instrucciones se reduce a **35** y sus modos de direccionamiento se han simplificado al máximo (**Apéndice A**).

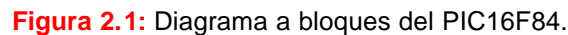
Con la **estructura segmentada** se pueden realizar simultáneamente las dos fases en que se descompone cada instrucción. Al mismo tiempo que se está desarrollando la fase de **ejecución** de una instrucción se realiza la fase de **búsqueda** de la siguiente.

El aislamiento y diferenciación de los dos tipos de memoria permite que cada uno tenga la longitud y el tamaño más adecuados. De esta forma en el **PIC16F84** la longitud de los datos es de un byte, mientras que la de las instrucciones es de **14 bits**.

Otra característica relevante de los PIC es el manejo intensivo del **Banco de Registros**, los cuales participan de una manera muy activa en la ejecución de las instrucciones. Por ejemplo la ALU efectúa sus operaciones lógico-aritméticas con dos operandos, uno que recibe desde el registro **W** (Work), que hace las veces de **Acumulador** en los microprocesadores convencionales, y otro que puede provenir de cualquier registro o del propio código de la instrucción. El resultado de la operación puede almacenarse en cualquier registro o en **W**. Esta funcionalidad da un carácter completamente ortogonal a las instrucciones que pueden utilizar cualquier registro como **operando fuente** y **destino**. La memoria de datos RAM implementa en sus posiciones los registros específicos y los de propósito general.

La arquitectura interna del **PIC16F84** se presenta en la **Figura 2.1** y consta de siete bloques fundamentales.

1. **Memoria de programa** Flash de **1K x 14bits**.
2. **Memoria de datos** formada por dos áreas. Una RAM donde se alojan **22 registros de propósito específico** y **68 de propósito general** y una **EEPROM de 64 bytes**.
3. Camino de datos con una **ALU de 8 bits** y registro de trabajo **W**, del que normalmente recibe un operando y envía el resultado. El otro operando puede provenir del bus de datos o del propio registro de instrucción (literal).
4. Diversos **recursos** conectados al bus, como puertos, temporizadores, etc.
5. **Base de tiempos** y **circuitos auxiliares**.
6. Direccionamiento de la memoria de programa en base al **contador de programa** ligado a una **pila de 8 niveles** circular.
7. Direccionamiento **directo** e **indirecto** a la memoria RAM.



Las operaciones de E/S con los periféricos las soportan los **Puertos A y B**. Existe un **Temporizador**, TMR0, para encargarse de las funciones de control de tiempos. Finalmente, hay circuitos auxiliares que dotan al procesador de interesantes posibilidades de seguridad, reducción del consumo y reinicialización.

2.2 Memoria de programa

La arquitectura de los PIC de la gama media admite un mapa de memoria de programa capaz de contener **8,192 instrucciones** de **14 bits** cada una. Este mapa se divide en páginas de **2,048** posiciones. Para direccionar **8K** posiciones se necesitan **13 bits**, que es la longitud que tiene el Contador de Programa. Sin embargo, el **PIC16F84** sólo tiene implementadas 1K posiciones, por lo que ignora los 3 bits de más peso del **Contador de Programa (PC)**.

En la gama media, la verdadera aportación del **PIC16F84** es la utilización de una memoria de programa del tipo Flash, capaz de ser escrita y borrada eléctricamente.

2.3 Contador de Programa y la Pila

El rango de direcciones que cubre el **PIC16X84** en su memoria de programa llega desde la $0x0000$ a la $0x03FF$, o sea, un total de 1,024 posiciones.

En el **PC** se ignoran los 3 bits de más peso, de forma que apuntar a la dirección $0x33$ es lo mismo que hacerlo a la $0x033$, $0x833$, etc.

Al igual que todos los registros específicos que controlan la actividad del procesador, el **Contador de Programa** está implementado sobre un par de posiciones de la memoria RAM. Cuando se escribe el **Contador de Programa** como resultado de una operación de la **ALU**, los 8 bits de menos peso del **PC** residen en el registro **PCL**, que ocupa, repetido, la posición 2 de los dos bancos de la memoria de datos. Los bits de más peso, **PC<12:8>**, residen en los 5 bits de menos peso del registro **PCLATH**, que ocupa la posición $0x0A$ de los dos bancos de la memoria **RAM**.

En las instrucciones **GOTO** y **CALL** de la gama media los 11 bits de menos peso del **PC** provienen del código de la instrucción y los otros dos de los bits **PCLATH <4:3>**.

Con los 11 bits que se cargan en el **PC** desde el código de las instrucciones **GOTO** y **CALL**, se puede direccionar una página de 2K de la memoria. Los bits restantes **PC<12:11>** tienen la misión de apuntar una de las 4 páginas del mapa de memoria y, en los modelos de PIC que alcanzan ese tamaño, dichos bits proceden de **PCLATH<4:3>**.

La **Pila** es una zona aislada de las memorias de instrucciones y datos. Tiene una **estructura LIFO**, en la que el último valor guardado es el primero que sale. Tiene **8 niveles de profundidad** cada uno con 13 bits. Funciona como una **memoria circular**, de manera que el valor que se obtiene al realizar el noveno desempilado (**pop**) es igual al que se obtuvo en el primero.

La instrucción **CALL** y las interrupciones originan la carga del contenido del **PC** en el nivel superior o tope de la **Pila**. El contenido del nivel superior se saca de la Pila al ejecutar las instrucciones **RETURN**, **RETLW** y **RETFIE**. El contenido del registro **PCLATH** no es afectado por la entrada o salida de información de la **Pila**.

2.4 Memoria de datos RAM

La memoria de datos del **PIC16F84** dispone de dos zonas diferentes:

- **Área de RAM estática o SRAM**, donde reside el **Banco de Registros Específicos (SFR)** y el **Banco de Registros de Propósito General (GPR)**. El primer banco tiene 22 posiciones de tamaño de un byte, aunque dos de ellas no son operativas, y el segundo 68 registros de propósito general.
- **Área EEPROM** de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se pierden al desconectar la alimentación.

La zona de memoria RAM se halla dividida en dos bancos (**banco 0** y **banco 1**) de 128 bytes cada uno. En el **PIC16F84** sólo se hallan implementadas físicamente las 48 primeras posiciones de cada banco, de las cuales las 12 primeras están reservadas a los **Registros de Propósito Específico (SFR)**, que son los encargados del control del procesador y sus recursos. Algunos de dichos registros se hallan repetidos en la

misma dirección de los dos bancos, para simplificar su acceso (INDF, ESTADO, FSR, PCLATH e INTCON). La posición apuntada por la dirección 0x7 y la apuntada por la 0x87 no son operativas. Los 68 registros restantes de cada banco se destinan a Registros de Propósito General y en realidad sólo son operativos los 68 del **banco 0** porque los del **banco 1** se mapean sobre el **banco 0**, es decir, cuando se apunta a un registro general del **banco 1**, se accede al mismo del **banco 0**. (Figura 2.2).

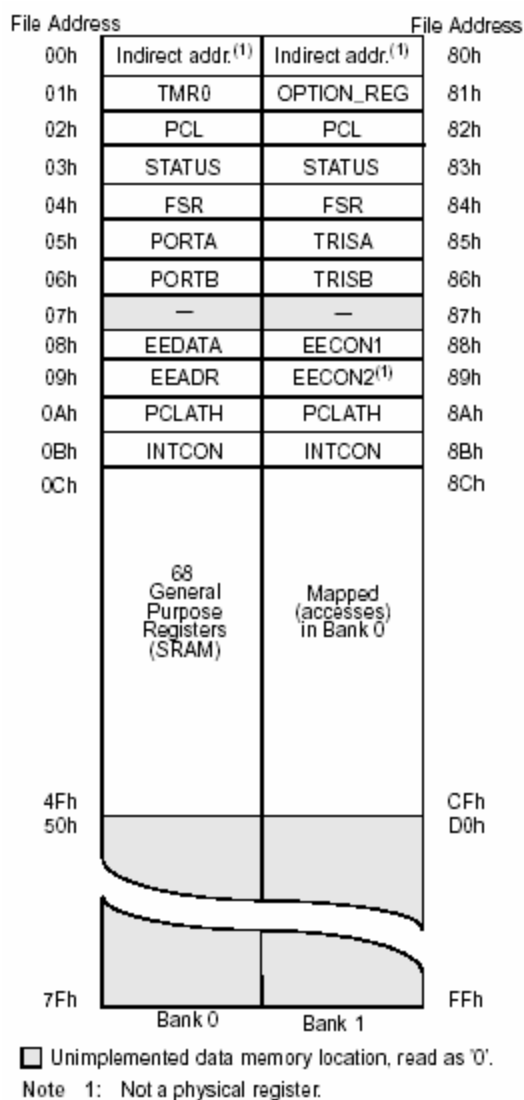


Figura 2.2: Mapa de registros del PIC16F84.

Para seleccionar el banco a acceder hay que manipular el bit 5 (**RP0**) del registro ESTADO. Si **RP0** = 1 se accede al **banco 1** y si **RP0** = 0 se accede al **banco 0**. Tras un Reset se accede automáticamente al **banco 0**. Para seleccionar un registro de propósito general no hay que tener en cuenta el estado del bit **RP0**, porque al estar mapeado el **banco 1** sobre el **banco 0**, cualquier direccionamiento de un registro del **banco 1** corresponde al homólogo del **banco 0**. En el direccionamiento directo a los registros **GPR** se ignora el bit de más peso, que identifica el banco y sus direcciones están comprendidas entre el valor 0x0C y 0x02F.

Los registros **SFR** se clasifican en dos grupos. En uno se incluyen aquellos que controlan el núcleo del microcontrolador (ESTADO, OPTION, INTCON, etc.) y en el otro los que caracterizan la operatividad de los recursos auxiliares y periféricos. La **Figura 2.3** presenta la estructura de estos registros y los valores que toman después de un Reset.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- --	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS ⁽²⁾	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA ⁽⁴⁾	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB ⁽⁵⁾	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC ⁽¹⁾				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- --	11
81h	OPTION_REG	RBP0	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS ⁽²⁾	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register				---1 1111	16	
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- --	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC ⁽¹⁾				---0 0000	11	
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Figura 2.3: Registros Especiales del PIC16F84.

2.5 Direccionamiento de la memoria de datos

En los **PIC** de la gama media la memoria de datos está organizada para alojar un máximo de 4 bancos de 128 bytes cada uno. El **PIC16F84** sólo tiene implementados los 48 primeros bytes de los **bancos 0 y 1**. En el resto de los **PIC** de esta familia se destinan dos bits del registro **ESTADO** (**RP0** y **RP1**) para determinar el banco y otros siete para elegir una de las 128 posiciones del banco seleccionado.

Direccionamiento Directo.- El operando que utiliza la instrucción en curso se referencia mediante su dirección, que viene incluida en el **código OP** de la misma, concretamente en los 7 bits de menos peso. El banco a acceder lo determinan los bits **RP0** y **RP1** del registro **ESTADO**. En el caso del **PIC16F84** sólo se usa el bit **RP0** al tener implementados únicamente dos bancos.

Direccionamiento Indirecto.- Este modo de direccionado se usa cuando en una instrucción se utiliza como operando el registro **INDF**, que ocupa la dirección 0 de ambos bancos. En realidad el registro **INDF** no está implementado físicamente y cuando se le hace referencia, se accede a la dirección de un banco especificada con los 7 bits de menos peso del registro **FSR**. El bit de más peso de **FSR** junto al bit **IRP** del registro **ESTADO** se encarga de seleccionar el banco a acceder, mientras que los 7 bits de menos peso apuntan a la posición. Como sólo hay dos bancos en el **PIC16F84** en este modo de direccionamiento, el bit **IRP** = 0 siempre.

2.6 Disposición de terminales

A continuación se muestra la distribución de las terminales de algunos PIC (**Figura 2.4**).

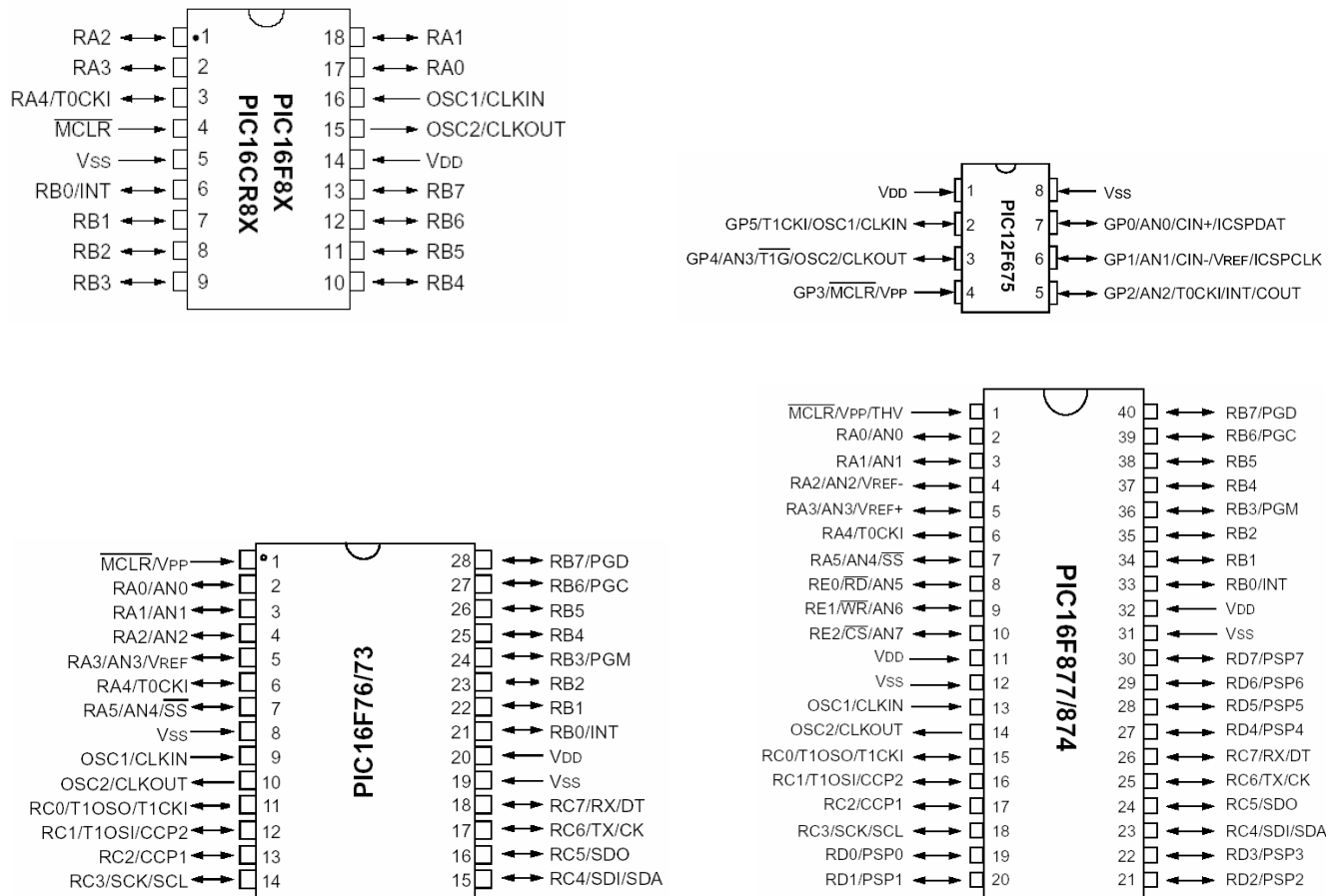


Figura 2.4: Disposición de las terminales de diferentes PIC.

Capítulo 3: Registros

3.1 Registro de estado (STATUS)

El registro de estado (**STATUS**) tiene tres funciones importantes:

- Avisa incidencias generadas en la Unidad Aritmética y Lógica (C, DC y Z).
- Indica el estado del **RESET** (~TO y ~PD).
- Selecciona el banco a acceder a la memoria de datos (IRP, RP0 y RP1).

Registro de estado (**STATUS**):

7	6	5	4	3	2	1	0
IRP	RP1	RP0	~TO	~PD	Z	DC	C
R/W	R/W	R/W	R	R	R/W	R/W	R/W

- **C.**- es el **acarreo**, 1 cuando se produce acarreo con las instrucciones ADDWF y ADDLW. Es **cero** si no se ha producido acarreo.
- **DC.**- **Acarreo medio**, es el que se produce entre nibbles.
- **Z.**- bandera de **cero**, si es 1 el resultado de la última operación es **cero**. Si es **cero** el resultado de la última operación no fue **cero**.
- **~PD.**- **Power Down**, es 1 después de alimentar al μC , ó al ejecutar la instrucción CLRWD. Se pone a **cero** cuando se ejecuta SLEEP.
- **~TO.**- **Time Out**, se pone a 1 después de conectar el μC ó al ejecutar las instrucciones CLRWD y SLEEP. Se pone a **cero** por un desbordamiento del perro guardián.
- **RP1 y RP0.**- para la **selección del banco** de direccionamiento, cuando RP0 vale 1 se accede al **banco 1** y si vale 0 se accede al **banco 0**. Después de RESET se tiene $RP0 = 0$.
- **IRP.**- Sirve para determinar el banco de memoria de datos seleccionado.

3.2 Registro de opciones (OPTION)

Se encarga de controlar los temporizadores y el divisor de frecuencia. Registro de estado (**OPTION**):

7	6	5	4	3	2	1	0
~RBPO	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

- **PS2:PS0** determinan el valor del divisor de frecuencia.

PS2	PS1	PS0	División de TMR0	División WDT
0	0	0	2	1
0	0	1	4	2
0	1	0	8	4
0	1	1	16	8
1	0	0	32	16
1	0	1	64	32
1	1	0	128	64
1	0	1	256	128

- **PSA.**- Asignación del divisor de frecuencia. 1 = WDT y 0 = TMR0.
- **TOSE.**- Tipo de flanco de T0CKI. 1 = descendente y 0 = ascendente.
- **TOCS.**- Tipo de reloj para TMR0. 1 = Pulsos a través de T0CKI (contador) y 0 = Reloj interno $F_{osc}/4$ (temporizador).
- **INTEDG.**- Flanco activo de interrupción externa. 1 = descendente y 0 = ascendente.
- **~RBPO.**- Resistores de elevación para el **puerto B**. 1 = Desactivas y 0 = Activadas.

3.3 Perro Guardián (Watchdog)

Es un contador interno de 8 bits, que origina un `RESET` cuando se desborda. Su control de tiempos es independiente a `TMR0` y se basa en una red RC. Su actuación es opcional y puede bloquearse mediante el bit `WDTE` de la **palabra de configuración** al programar el microcontrolador.

El tiempo nominal con la que se halla programado el perro guardián es de 18ms, pero puede ampliarse utilizando el divisor de frecuencia hasta unos 2.3s. Para evitar que el perro guardián se desborde es necesario borrarlo con `CLRWDT`.

3.4 Modo de reposo o de bajo consumo (SLEEP)

Se caracteriza por un bajo consumo de energía, se recomienda para periodos de espera prolongados, el microcontrolador consume en este modo menos de 1µA, por lo que puede alimentarse con una pequeña batería cerca de dos años.

Cuando se entra al modo `Sleep`, el microcontrolador se congela, se detiene el oscilador principal, las terminales de E/S, se mantienen en el estado anterior a `Sleep`, y las que no se encuentran conectadas pasan a alta impedancia. El procesador sin pulsos de reloj se congela y deja de ejecutar instrucciones hasta que despierte y salga del estado de reposo.

Sin embargo el perro guardián continúa activo, al entrar a él, se borra pero sigue activo. Existen tres formas de salir del modo de reposo.

- Activación externa a través del `Reset`.
- Desbordamiento del perro guardián, si se quedo activo al entrar al estado de reposo.
- Generación de una interrupción.

Cuando el PIC despierta se desarrolla una secuencia del oscilador, que retarda 1024 ciclos para estabilizar la frecuencia de trabajo y luego se pasa a ejecutar la instrucción `sleep(PC+1)`.

Los bits `~PD` y `~TO` se emplean para conocer la causa del `Reset` que despierta al sistema. Cuando se ejecuta `Sleep`, `~PD` = 0 y cuando se desborda el perro guardián `~TO` = 0.

3.5 Temporizadores

Los PIC disponen de un procesador rápido y potente. En él se incluye la memoria de programa, la de datos, la ALU, la Unidad de Control y algunos registros especiales. Si dentro de un microcontrolador únicamente existiese un procesador sólo se podrían ejecutar instrucciones lógico aritméticas y de transferencia. Pero un microcontrolador es mucho más que un procesador, es una **computadora integral**, en el que además del procesador hay puertos de E/S para conectarse con periféricos, canales de comunicación, temporizadores para controlar tiempos, sistema de interrupciones capaz de detectar anomalías o sucesos especiales, sistemas de seguridad, modo de funcionamiento con bajo consumo y un largo etcétera de recursos que configuran la potencia integral de un computador.

Los PIC contienen todos los recursos posibles aunque los fabricantes no los incluyen en todos los modelos, sino que los alternan para poderse ajustar óptimamente a las necesidades de cada diseño.

Los **PIC16X8X** contienen pocos recursos en comparación a otros familiares, pero poseen los suficientes para resolver gran parte de las aplicaciones típicas de los microcontroladores. Sobre todo disponen de una memoria de programa tipo EEPROM (**PIC16C84**) ó Flash (**PIC16F8X**), que les posibilita regrabar el programa hasta su total puesta a punto.

3.5.1 Temporizador/Contador TMR0

Una de las labores más habituales en los programas de control de dispositivos suele ser determinar intervalos concretos de tiempo, y recibe el nombre de **temporizador** (`timer`) el elemento encargado de realizar

esta función. También suele ser frecuente contar los impulsos que se producen en el exterior del sistema, y el elemento destinado a este fin se denomina contador.

Si las labores del temporizador o contador las asignamos al programa principal robarían mucho tiempo al procesador en detrimento de actividades más importantes. Por este motivo se diseñan recursos específicamente orientados a estas misiones.

Los **PIC16X8X** poseen un temporizador/contador de 8 bits, llamado **TMR0**, que actúa de dos maneras diferentes:

- Como **contador de sucesos**, que están representados por los impulsos que se aplican a la terminal **RA4/T0CKI**. Al llegar al valor **0xFF** se desborda el contador y, con el siguiente impulso, pasa a **0x00**, advirtiendo esta circunstancia activando una bandera y/o provocando una interrupción.
- Como **temporizador**, cuando se carga en el registro que implementa al recurso un valor inicial se incrementa con cada ciclo de instrucción (**F_{osc}/4**) hasta que se desborda, o sea, pasa de **0xFF** a **0x00** y avisa poniendo a 1 una bandera y/o provocando una interrupción.

Para que el **TMR0** funcione como contador de impulsos aplicados a la patita **T0CKI** hay que poner a 1 el bit **TOCS**, que es el que ocupa la posición 5 del registro **OPTION**. En esta situación, el registro **TMR0**, que es el ubicado en la dirección 1 del **banco 0** de la memoria de datos, se incrementa con cada flanco activo aplicado en la terminal **T0CKI**. El tipo de flanco activo se elige programando el bit **TOSE**, que es el que ocupa la posición 4 del registro **OPTION**. Si **TOSE = 1**, el flanco activo es el descendente, y si **TOSE = 0**, es el ascendente. Cuando se desea que **TMR0** funcione como temporizador el bit **TOCS = 0**.

En realidad, los **PIC16X8X** y los de la gama baja disponen de dos temporizadores, el **TMR0** y el **Perro Guardián** (Watchdog). El primero actúa como principal y sobre él recae el control de tiempos y el conteo de impulsos. El otro vigila que el programa no se bloquee y para ello cada cierto tiempo comprueba si el programa se está ejecutando normalmente. En caso contrario, si el control está detenido en un bucle infinito a la espera de algún acontecimiento que no se produce, el **Perro Guardián** se desborda, lo que se traduce en un **Reset** que reinicializa todo el sistema.

A menudo el **TMR0** y el **Perro Guardián** precisan controlar largos intervalos de tiempo y necesitan aumentar la duración de los impulsos de reloj que les incrementa. Para cubrir este requisito se dispone de un circuito programable denominado **Divisor de frecuencia**, que divide la frecuencia utilizada por diversos rangos.

Para programar el comportamiento del **TMR0**, el **Perro Guardián** (**WDT**) y el **Divisor de frecuencia** se utilizan algunos bits del registro **OPTION** y de la **Palabra de Configuración**.

El **Divisor de frecuencia** puede usarse con el **TMR0** o con el **WDT**. Con el **TMR0** actúa como **Predivisor**, es decir, los impulsos pasan primero por el **Divisor** y luego se aplican al **TMR0**, una vez aumentada su duración. Con el **Perro Guardián** actúa después, realizando la función de **Postdivisor**. Los impulsos, que divide por un rango el **Divisor de frecuencia**, pueden provenir de la señal de reloj interna (**F_{osc}/4**) o de los que se aplican a la terminal **T0CKI**.

El **TMR0** se comporta como un registro de propósito especial (**SFR**) ubicado en la dirección 1 del **banco 0** de la memoria de datos. En igual dirección, pero en el **banco 1**, se halla el registro **OPTION**.

TMR0 puede ser leído y escrito en cualquier momento al estar conectado al bus de datos. Funciona como un contador ascendente de 8 bits. Cuando funciona como temporizador conviene cargarlo con el valor de los impulsos que se quiere temporizar, pero expresados en complemento a 2. De esta manera al llegar al número de impulsos deseado se desborda y al pasar a **0x00** se activa el **TOIF** y/o se produce una interrupción. Para calcular los tiempos a controlar con el **TMR0** se tienen:

$$\begin{aligned} \text{Temporización} &= 4 * T_{osc} (\text{Valor cargado en TMR0}) * (\text{Rango del divisor}) \\ \text{Valor a cargar en TMR0} &= \text{Temporización} / (4 * T_{osc} * \text{Rango del divisor}) \end{aligned}$$

3.6 Ejemplo del uso de registros y puertos

El **Programa 3.1** ilumina una serie de leds en secuencia principio-fin-principio. El circuito con un PIC16F73 se muestra en la **Figura 3.1**.

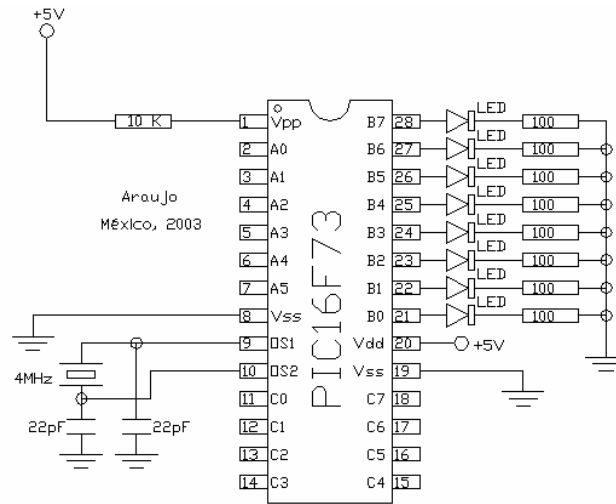


Figura 3.1: Circuito de marquesina.

```

LIST P = 16F73
; Puertos y registros del Microcontrolador
PB      EQU      0x06      ; Puerto B
TRISB   EQU      0x86      ; Registro del puerto B
OPTREG  EQU      0x81      ; Registro OPTION
STATUS  EQU      0x03      ; Registro de estado
CARRY   EQU      0x00      ; Bandera de Acarreo
RP0     EQU      0x05      ; Selección del banco de direccionamiento (Status)
MSB     EQU      0x07      ; Posición del BIT más a la izquierda
; Programa Principal
        CLRFB      PB      ; Leds apagados
        BSF        STATUS,RP0 ; Banco de registros 1
        CLRFB      TRISB^0x80 ; Puerto B configurado para salida
        MOVLW      0x0A      ; Configuración de OPTION
        MOVWF      OPTREG^0x80 ; PRESCALER (1:4) asignado al WatchDog
        BCF        STATUS,RP0 ; Banco de registros 0
        INCF       PB,0x01    ; Encender LED de la derecha
        BCF        STATUS,CARRY ; Carry = 0
IZQDA   SLEEP        ; Esperar WDT
        RLF        PB,0x01    ; Encender siguiente LED a la izquierda
        BTFSS      PB,MSB     ; Alcanzado final por la izquierda?
        GOTO       IZQDA      ; Si no, siguiente
DRCHA   SLEEP        ; Esperar WDT
        RRF        PB,0x01    ; Encender siguiente LED a la derecha
        BTFSS      PB,0x00    ; Alcanzado final por la derecha?
        GOTO       DRCHA      ; NO: repetir
        GOTO       IZQDA      ; SI: Comienza un nuevo ciclo.
END      ; Fin

```

Programa 3.1: Programa de marquesina.

Capítulo 4: Puertos de E/S y EEPROM

4.1 Frecuencia de funcionamiento (Reloj)

La frecuencia de trabajo del microcontrolador es un parámetro fundamental a la hora de establecer la velocidad en la ejecución de instrucciones y el consumo de energía.

Cuando un **PIC16X8X** funciona a 10 MHz, le corresponde un ciclo de instrucción de 400 ns, puesto que cada instrucción tarda en ejecutarse cuatro períodos de reloj, o sea, $4 \times 100 \text{ ns} = 400 \text{ ns}$. Todas las instrucciones del PIC se realizan en un ciclo de instrucción, menos las de salto, que tardan el doble.

Los impulsos de reloj entran por la terminal **OSC1/CLKIN** y se dividen por 4 internamente, dando lugar a las señales **Q1, Q2, Q3 y Q4**. Durante un ciclo de instrucción, que comprende las 4 señales mencionadas, se desarrollan las siguientes operaciones:

- **Q1:** Durante este impulso se incrementa el Contador de Programa.
- **Q4:** Durante este impulso se busca el código de la instrucción en la memoria del programa y se carga en el Registro de Instrucciones.
- **Q2-Q3:** Durante la activación de estas dos señales se produce la decodificación y la ejecución de la instrucción.

Para conseguir ejecutar cada instrucción en un ciclo de instrucción (excepto las de salto, que tardan dos), se aplica la técnica de la **segmentación o pipe-line**, que consiste en realizar en **paralelo** las dos fases que comprende cada instrucción.

En realidad, cada instrucción se ejecuta en dos ciclos: en el primero se lleva a cabo la **fase de búsqueda** del código de la instrucción en la memoria del programa, y en el segundo se **decodifica** y la **ejecuta** (fase de ejecución). La estructura segmentada del procesador permite realizar al mismo tiempo la fase de ejecución de una instrucción y la de búsqueda de la siguiente. Cuando la instrucción ejecutada corresponde a un salto no se conoce cuál será la siguiente hasta que se complete, por eso en esta situación se sustituye la fase de búsqueda de la siguiente instrucción por un **ciclo vacío**, originando que las instrucciones de salto tarden en realizarse dos ciclos de instrucción.

La **técnica de la segmentación** unida a la **arquitectura Harvard** del procesador permite a los PIC superar la velocidad de sus competidores directos. Así, por ejemplo, el **PIC16F84** es **1.54** veces más rápido que el microcontrolador de **Motorola 68HC05** cuando ambos funcionan a la misma frecuencia de **4 MHz**.

Ejemplo: Un **PIC16X84** que funciona a 4 MHz ejecuta un programa de 1,000 instrucciones, de las cuales el 25% son de salto. Calcular el tiempo que tarda en ejecutarse.

$$T_{\text{ciclo de instrucción}} = 4 * T_{\text{reloj}} = 4 * 250\text{ns} = 1\text{ms}$$

$$T_{\text{programa}} = 750 * 1\text{ms} + 250 * 2\text{ms} = 1,250\text{ms}$$

4.2 Tipos de osciladores

Los PIC admiten cuatro tipos de osciladores externos para aplicarles la frecuencia de funcionamiento. El tipo empleado debe especificarse en dos bits (**FOSC1** y **FOSC0**) de la **Palabra de Configuración**.

- **Oscilador tipo RC.-** Se trata de un oscilador de bajo costo formado por un simple resistor y un condensador. Proporciona una estabilidad mediocre de la frecuencia, cuyo valor depende de los valores de los dos elementos de la red **RC**.
- **Oscilador tipo HS.-** Se trata de un oscilador que alcanza una alta velocidad comprendida entre 4 y 20MHz y está basado en un cristal de cuarzo o un resonador cerámica.
- **Oscilador tipo XT.-** Es un oscilador de cristal o resonador para frecuencias estándar comprendidas entre 100 KHz y 10MHz.

- **Oscilador tipo LP.**- Oscilador de bajo consumo con cristal o resonador diseñado para trabajar en un rango de frecuencias de 35 a 200 KHz. El cristal de cuarzo o el resonador cerámica se coloca entre las terminales `OSC1` y `OSC2`.

4.3 Reinicialización ó RESET

Cuando se aplica un nivel lógico bajo en la terminal `~MCLR` el microcontrolador reinicializa su estado. Dos acciones importantes se producen en la reinicialización o `Reset`:

- El **Contador de Programa** se carga con la `dirección 0`, apuntando la primera dirección de la memoria de programa en donde deberá estar situada la primera instrucción del programa de aplicación. La mayoría de los registros de estado y control del procesador toman un estado conocido y determinado.
- Se puede ocasionar el `Reset` de varias maneras, que estudiaremos más adelante.

El circuito más simple para provocar un `Reset` manualmente, se obtiene colocando un interruptor entre la terminal `~MCLR` y tierra.

4.4 Puertos de E/S

Los **PIC16X8X** sólo disponen de dos puertos de E/S. El **puerto A** posee 5 líneas, `RA0-RA4`, y una de ellas soporta dos funciones multiplexadas. Se trata de la `RA4/TOCKI`, que puede actuar como línea de E/S o como terminal por la que se reciben los impulsos que debe contar `TMR0`. El **puerto B** tiene 8 líneas, `RB0 - RB7`, y también tiene una con funciones multiplexadas, la `RB0/INT`, que, además de línea típica de E/S, también sirve como la terminal por la que se reciben los impulsos externos que provocan una **interrupción**

Cada línea de E/S puede configurarse independientemente como entrada o como salida, según se ponga a 1 ó a 0, respectivamente, el bit asociado del registro de configuración de cada puerta (`TRISA` y `TRISB`). Se llaman `PUERTO A` y `PUERTO B` los registros que guardan la información que entra o sale por el puerto y ocupan las direcciones 5 y 6 del **banco 0** de la memoria de datos. Los registros de configuración `TRISA` y `TRISB` ocupan las mismas direcciones pero en el **banco 1**.

Puerto A

Las líneas `RA3-RA0` admiten niveles de entrada TTL y de salida CMOS. La línea `RA4/TOCKI` dispone de un circuito Schmitt Trigger que proporciona una buena inmunidad al ruido y la salida tiene drenador abierto. `RA4` multiplexa su función de E/S con la de entrada de impulsos externos para el `TMR0`.

Cuando se lee una línea del **Puerto A** (instrucción `movfw puertoA`) se recoge el nivel lógico que tiene en ese momento. Las líneas cuando actúan como salidas sus terminales sacan el nivel lógico que se haya cargado por última vez en el registro `PUERTO A`. La escritura de una puerta implica la operación lectura/modificación/escritura. Primero se lee el puerto, luego se modifica el valor y finalmente se escribe en la memoria de salida.

Cuando se saca un nivel lógico por una línea del **Puerto A**, primero se deposita en la línea correspondiente del bus interno de datos y se activa la señal `WRITE`, lo que origina el almacenamiento de dicho nivel en el multivibrador de datos.

Si una línea actúa como entrada, el nivel lógico depositado en ella desde el exterior pasa a la línea correspondiente del bus interno de datos cuando se activa la señal `READ` y se hace conductor el dispositivo triestado que les une. Al programarse como entrada, los dos transistores MOS de salida quedan bloqueados y la línea en alta impedancia. Téngase en cuenta que cuando se lee una línea de entrada se obtiene el estado actual que tiene su terminal correspondiente y no el valor que haya almacenado en la báscula de datos. La información presente en una línea de entrada se muestrea al iniciarse el ciclo de instrucción y debe mantenerse estable durante su desarrollo.

Al **reinicializarse** el PIC todos los bits de los registros `TRIS` quedan a 1, con lo que las líneas de los puertos quedan configuradas como **entradas**.

Cada línea de **salida** puede suministrar una corriente máxima de **20 mA** y si es **entrada** puede absorber hasta **25 mA**. Al existir una limitación en la disipación máxima de la potencia del circuito se restringe la corriente máxima de **absorción** del **Puerto A a 80 mA** y la de **suministro a 50 mA**. El **puerto B** puede **absorber** un máximo de **150 mA** y **suministrar** un total de **100 mA**.

Con `movf puerto,w` se lee un puerto y con la instrucción `movwf puerto` se escribe. También existen instrucciones para modificar el valor de un bit particular correspondiente a una línea de un puerto con las instrucciones `bsf puerto,bit` (pone a 1 el bit indicado del puerto) y `bcf puerto,bit`. Existen instrucciones de salto condicionales que verifican el valor de un bit de un puerto y brincan si vale 1 (`btfss`) o si vale 0 (`btfsc`).

Puerto B

Consta de 8 líneas bidireccionales de E/S, `RB7-RB0`, cuya información se almacena en el registro `PUERTOB`, que ocupa la dirección 6 del **banco 0**. El registro de configuración `TRISB` ocupa la misma dirección en el **banco 1**.

La línea `RB0/INT` tiene dos funciones multiplexadas. Además de terminal de E/S, actúa como terminal para la petición de una interrupción externa, cuando se autoriza esta función mediante la adecuada programación del registro `INTCON`.

A todas las líneas de este puerto se las permite conectar un **resistor pull-up** de elevado valor con el positivo de la alimentación. Para este fin hay que programar en el registro `OPTION` el bit `~RBPUE = 0`, afectando la conexión del resistor a todas las líneas. Con el **Reset** todas las líneas quedan configuradas como entradas y se desactivan los resistores pull-up.

Las 4 líneas de más peso, `RB7-RB4`, pueden programarse para soportar una misión especial. Cuando las 4 líneas actúan como entradas se les puede programar para generar una **interrupción** si alguna de ellas cambia su estado lógico. Esta posibilidad es muy práctica en el control de teclados.

El estado de las patitas `RB7-RB4` en modo entrada se compara con el valor antiguo que tenían y que se había almacenado durante la última lectura del **Puerto B**. El cambio de estado en alguna de esas líneas origina una interrupción y la activación de la bandera `RB1F`.

La línea `RB6` también se utiliza para la grabación serie de la memoria de programa y sirve para soportar la señal de reloj. La línea `RB7` constituye la entrada de los datos en serie.

4.5 Palabra de Configuración

La **palabra de configuración** es una posición reservada de la memoria de programa situada en la dirección `0x2007` y accesible únicamente durante el proceso de grabación. Al escribirse el programa de la aplicación es necesario grabar el contenido de esta posición de acuerdo con las características del sistema.

En la **Figura 4.1** se muestra la distribución y asignación de los 14 bits de la **Palabra de Configuración** de los **PIC16F8X**.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	~PWRTE	WDTE	FOSC1	FOSC0

Figura 4.1: Palabra de Configuración del PIC16F8X.

Cada bit tiene las siguientes funciones:

CP: Bits de protección de la memoria de código.

- 1: No protegida
- 0: Protegida. El programa no se puede leer, evitando copias. Tampoco se puede sobrescribir. Además evita que pueda ser accedida la EEPROM de datos y, finalmente, si se modifica el bit **CP** de 0 a 1, se borra completamente la EEPROM.

PWRT: Activación del temporizador *power-up*. El temporizador **power-up** retrasa 72 ms la puesta en marcha o *Reset* que se produce al conectar la alimentación al PIC, para garantizar la estabilidad de la tensión aplicada.

- 1: Desactivado
- 0: Activado

WDTE: Activación del perro guardián.

- 1: Activado el WDT
- 0: Desactivado

FOSC1-FOSC0: Selección del oscilador utilizado.

- 1-1: Oscilador **RC**
- 1-0: Oscilador **HS**
- 0-1: Oscilador **XT**
- 0-0: Oscilador **LP**

4.6 Palabra de Identificación (ID)

Son 4 posiciones reservadas de la memoria de programa ubicadas en las direcciones $0x2000-0x2003$ que no son accesibles en el funcionamiento normal del microcontrolador y sólo pueden ser leídas y escritas durante el proceso de grabación.

Sólo se utilizan los 4 bits de menos peso de cada **palabra de identificación (ID)**, en donde se almacena un valor que puede consistir en un número de serie, códigos de identificación, numeraciones secuenciales o aleatorias, etc.

4.7 Memoria de datos EEPROM

Los **PIC16X8X** tienen 64 bytes de memoria EEPROM de datos, donde se pueden almacenar datos y variables que interesa que no se pierdan cuando se desconecta la alimentación al sistema. Soporta 1,000,000 de ciclos de escritura/borrado y es capaz de guardar la información sin alterarla más de 40 años.

La memoria EEPROM no está mapeada en la zona de la memoria de datos donde se ubican los registros *SFR* y *GPR*. Para poder leerla y escribirla durante el funcionamiento normal del microcontrolador hay que utilizar 4 registros del banco *SFR*: *EEDATA*, *EEADR*, *EECON1* y *EECON2*.

En el **registro EEADR**, ubicado en la dirección 9 del **banco 0**, se carga la dirección a acceder de la EEPROM de datos. Las 64 posiciones de un byte ocupan las direcciones de un mapa que comienza en $0x00$ y termina en $0x3F$, por eso los 2 bits de más peso del registro *EEADR* siempre valen 0.

En el **registro EEDATA**, ubicado en la dirección 8 del **banco 0**, se depositan los datos que se leen o se escriben.

El **registro EECON1**, que ocupa la dirección $0x88$ de la memoria de datos, o la dirección $0x08$ del **banco 1**, tiene misiones de control de las operaciones en la EEPROM y la distribución de sus bits se presenta en la **Figura 4.2**.

7	6	5	4	3	2	1	0
---	---	---	EEIF	WRERR	WREN	WR	RD

Figura 4.2: Registro EECON1.

RD: Lectura.

- 1: Se pone a 1 cuando se va a realizar un ciclo de lectura de la EEPROM. Luego pasa a 0 automáticamente.

WR: Escritura.

- 1: Se pone a 1 cuando se inicia un ciclo de escritura de la EEPROM. Cuando se completa el ciclo pasa a 0 automáticamente.

WREN: Permiso de escritura.

- 1: Permite la escritura de la EEPROM.
- 0: Prohíbe la escritura.

WRERR: Bandera de error en escritura.

- 1: Se pone a 1 cuando una operación de escritura ha terminado prematuramente.
- 0: La operación de escritura se ha completado correctamente.

EEIF: Bandera de final de operación de escritura.

- 1: Cuando esta bandera se pone a 1 indica que la operación de escritura se ha completado con éxito. Se pone a 0 por programa.
- 0: La operación de escritura no se ha completado.

El registro `EECON2` en realidad no está implementado físicamente. Al leerlo todos sus bits son 0. Sólo se emplea como un dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.

Proceso de lectura

Se inicia un ciclo de lectura colocando la dirección a acceder en el registro `EEADR` y poniendo el bit `RD` = 1 en el registro `EECON1`. El dato leído estará disponible en el registro `EEDATA` en el siguiente ciclo y permanecerá en él hasta que se realice una nueva lectura o escritura en la EEPROM.

Ejemplo: Implementar un programa para realizar la lectura de una posición, `lect`, de la EEPROM de datos.

```

bcf    estado,rp0        ; Banco 0
movlw  lect              ; Dirección a leer
movwf  eeadr
bsf    estado,rp0        ; Banco 1
bsf    eecon1,rd         ; Lectura
bcf    estado,rp0        ; Banco 0
movf   eedata,w          ; En W queda el valor leído de la EEPROM

```

Proceso de escritura

Para escribir una posición de la EEPROM de datos el usuario debe seguir una determinada secuencia de instrucciones en las que participa el registro `EECON2`. Este registro, que en realidad no se halla implementado físicamente, sólo asume funciones de seguridad en el proceso, cargándose en él dos valores concretos: `0x55` y `0xAA`. La duración típica de un **ciclo de escritura** es de **10 ms**, que es notablemente larga en comparación con la velocidad del PIC.

El ciclo de escritura comienza cargando en `EEADR` la dirección de la posición a escribir y en el registro `EEDATA` el valor a grabar.

Ejemplo: Escribir en la dirección `escri` de la EEPROM de datos el valor `dato`.

```
bcf    estado,rp0        ; Banco 0
movlw  escri
movwf  eeadr             ; Escribe la dirección en eeadr
movlw  dato
movwf  eedata           ; Escribe el dato en eedata
bsf    estado,rp0       ; Banco 1
bsf    eecon1,wren      ; Permiso de escritura
movlw  0x55
movwf  eecon2           ; 0x55 se escribe en eecon2
movlw  0xAA
movwf  eecon2           ; 0xAA se escribe en eecon2
bsf    eecon1,wr        ; Comienza escritura
```

Al acabar el proceso de escritura el bit `WR` pasa a tener un valor de 0 automáticamente, mientras que la bandera `EEIF` se pone a 1. Este último bit hay que ponerlo a 0 posteriormente mediante software.

Una buena precaución es verificar si la escritura de la EEPROM ha sido correcta, para lo cual se suele restar el dato escrito con el que existe en el registro `EEDATA`. Si no se ha producido error la bandera `Z` pasa a 1.

Capítulo 5: Recursos Auxiliares

5.1 Interrupciones

Las **llamadas a subrutinas** mediante la instrucción `CALL` son desviaciones del flujo de control del programa originadas por instrucciones, por lo que se consideran síncronas. Se producen cada vez que se ejecuta dicha instrucción.

Las **interrupciones** son desviaciones del flujo de control del programa originadas asincrónicamente por diversos sucesos que no se hallan bajo la supervisión de las instrucciones. Dichos sucesos pueden ser externos al sistema, como la generación de un flanco o nivel activo en una terminal del microcontrolador, o bien internos, como el desbordamiento de un contador.

El comportamiento del microcontrolador ante la interrupción es similar al de la instrucción `CALL` de llamada a subrutina. En ambos casos se detiene la ejecución del programa en curso, se salva la dirección actual del **PC** en la **Pila** y se carga el **PC** con una dirección, que en el caso de `CALL` viene acompañando a la propia instrucción, y en el caso de una interrupción es una dirección reservada de la memoria de código, llamada **Vector de Interrupción**.

La **RSI** suele comenzar guardando en la memoria de datos algunos registros específicos del procesador. Concretamente aquellos que la **RSI** va a emplear y va a alterar su contenido. Antes del retorno al programa principal se recuperan los valores guardados y se restaura completamente el estado del procesador. Algunos procesadores salvan estos registros en la **Pila**, pero los PIC no disponen de instrucciones para meter (`push`) y sacar (`pop`) información de la **Pila**, utilizando para este fin registros de propósito general de la memoria de datos.

Los **PIC16X8X** pueden ser interrumpidos por 4 causas diferentes, pero todas ellas desvían el flujo de control a la dirección `0x0004`, por lo que otra de las operaciones iniciales de la **RSI** es averiguar cuál de las posibles causas ha sido la responsable de la interrupción en curso. Para ello se exploran las banderas de las fuentes de interrupción.

Otro detalle importante en la **RSI** de los **PIC16X8X** es que estos microcontroladores poseen un bit **GIE** (Global Interrupt Enable) que cuando vale 0 **prohíbe** todas las interrupciones. Pues bien, al comenzar la **RSI** dicho bit **GIE** se pone automáticamente a 0, con objeto de no atender nuevas interrupciones hasta que se termine la que ha comenzado. En el retorno final de la interrupción, **GIE** pasa a valer automáticamente 1 para volver a tener en cuenta las interrupciones. Dicho retorno de interrupción se realiza mediante la instrucción `RETFIE`.

Antes del retorno conviene borrar la bandera de la causa de interrupción que se ha atendido, porque si bien las banderas se ponen a 1 automáticamente en cuanto se produce la causa que indican, la puesta a 0 se hace por programa.

Causas de interrupción

Los **PIC16X8X** tienen 4 causas o fuentes posibles de interrupción:

- Activación de la terminal `RB0/INT`.
- Desbordamiento del temporizador `TMR0`.
- Cambio de estado en una de las 4 terminales de más peso del **Puerto B**.
- Finalización de la **escritura** de EEPROM de datos.

Cuando ocurre cualquiera de los 4 sucesos indicados se origina una petición de interrupción, que si se acepta y se atiende comienza depositando el valor del **PC** actual en la **Pila**, poniendo el bit **GIE** = 0 y cargando en el **PC** el valor `0x0004`, que es el **Vector de Interrupción** donde se desvía el flujo de control.

Cada fuente de interrupción dispone de una **bandera** ó **flag**, que es un bit que se pone automáticamente a 1 cuando se produce. Además cada fuente de interrupción tiene otro bit de permiso, que según su valor permite o prohíbe la realización de dicha interrupción.

Registro de Control de Interrupciones INTCON

La mayor parte de las banderas y bits de permiso de las fuentes de interrupción en los **PIC16X8X** están implementados sobre los bits del registro **INTCON**, que ocupa la dirección **0x0B** del **banco 0**, hallándose duplicado en el **banco 1** (**Figura 5.1**).

7	6	5	4	3	2	1	0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Figura 5.1: Registro INTCON.

GIE: Permiso Global de Interrupciones

- 1: Permite la ejecución de todas las interrupciones, cuyos bits de permiso individuales también las permitan.
- 0: Prohíbe todas las interrupciones.

EEIE: Permiso de Interrupción por fin de la escritura en la EEPROM.

- 1: Permite se origine una interrupción cuando termina la escritura de la EEPROM de datos.
- 0: Prohíbe que se produzca esta interrupción.

TOIE: Permiso de Interrupción por desbordamiento del TMR0.

- 1: Permite una interrupción al desbordarse el TMR0.
- 0: Prohíbe esta interrupción.

INTE: Permiso de Interrupción por activación de la patita RB0/INT.

- 1: Permite la interrupción al activarse RB0/INT.
- 0: Prohíbe esta interrupción.

RBIE: Permiso de interrupción por cambio de estado en RB7:RB4.

- 1: Permite esta interrupción.
- 0: Prohíbe esta interrupción.

TOIF: Bandera de desbordamiento del TMR0.

- 1: Se pone a 1 cuando ha ocurrido el desbordamiento.
- 0: Indica que el TMR0 no se ha desbordado.

INTF: Bandera de activación de la terminal RB0/INT.

- 1: Se pone a 1 al activarse RB0/INT.
- 0: Indica que RB0/INT aún no se ha activado.

RBIF: Bandera de cambio de estado en las patitas RB7:RB4.

- 1: Pasa a 1 cuando cambia el estado de alguna de estas 4 líneas.
- 0: No ha cambiado el estado de RB7:RB4.

Siempre que se produzca una interrupción por cualquier causa, $GIE = 0$ y PC se carga con el valor $0x0004$, que es el **Vector de Interrupción**. Para conocer qué causa ha provocado la interrupción se exploran las banderas, tres de las cuales se ubican en el registro $INTCON$ y la cuarta, $EEIF$, que se pone a 1 cuando finaliza la escritura de la EEPROM, se halla en el bit 4 del registro $EECON1$.

Las banderas deben ponerse a 0 por programa antes del retorno de la interrupción y son operativas aunque la interrupción esté prohibida con su bit de permiso correspondiente.

Interrupción Externa INT

La fuente de **interrupción externa** es sumamente importante para atender acontecimientos externos en **tiempo real**. Cuando ocurre alguno de ellos activa la terminal $RB0/INT$ y se hace una petición de interrupción. Entonces, de forma automática, el bit $INTF = 1$ y, si el bit de permiso $INTE = 1$, se autoriza el desarrollo de la interrupción.

Mediante el bit 6, llamado $INTDEG$, del registro $OPTION$ se puede seleccionar cuál será el flanco activo en $RB0/INT$. Si se desea que sea el ascendente se escribe un 1 en dicho bit, y si se desea que sea el descendente se escribe un 0.

El procesador explora la bandera $INTF$ al final del primer ciclo de reloj de cada ciclo de instrucción. Recuérdese que cada ciclo de instrucción constaba de 4 ciclos de reloj: Q_1 , Q_2 , Q_3 y Q_4 . Al terminar Q_1 se exploran las banderas produciéndose un período de latencia de 3 ó 4 ciclos de instrucción desde el momento que hay una bandera activa hasta que se inicializa la interrupción.

5.2 Convertidor Analógico/Digital (A/D)

En la **conversión A/D** se siguen los siguientes pasos:

1. Se configure el modulo **A/D**:
 - Configurar las terminales analógicas, el voltaje de referencia y las E/S digitales ($ADCON1$)
 - Seleccionar el tipo de reloj para la conversión **A/D** ($ADCON0$)
 - Iniciar el modulo **A/D** ($ADCON0$)
2. Configurar la interrupción del convertidor **A/D** (si se desea):
 - Limpiar el bit $ADIF$
 - Activar el bit $ADIE$
 - Activar el bit $PEIE$
 - Activar el bit GIE
3. Seleccionar el canal de entrada del convertidor **A/D** ($ADCON0$).
4. Esperar el tiempo necesario de acuerdo al periodo de adquisición.
5. Iniciar la conversión:
 - Activar el bit $GO/DONE$ ($ADCON0$)
6. Esperar a que la conversión **A/D** se complete, verificando:
 - Que el bit $GO/DONE$ se encuentre limpio (interrupciones deshabilitadas)
 - Esperar la interrupción del convertidor **A/D**
7. Leer el resultado de la conversión **A/D** ($ADRES$), y limpiar el bit $ADIF$ si es necesario.
8. Si se requiere otra conversión repetir el proceso desde los pasos 3 ó 4 según las necesidades.

Ejemplo: el siguiente segmento de código inicia el convertidor A/D.

```
; Inicia el convertidor A/D, selecciona los canales
; CH0 al CH3 como entradas analógicas, fosc/2 y CH3 como lectura
Inicia_AD
    bsf    STATUS,RP0          ; Selecciona el banco 1
    clrf   ADCON1              ; Selecciona CH0-CH3 como entradas analógicas
    bcf    STATUS,RP0          ; Selecciona el banco 0
    movlw  B'10000001'         ; Selecciona Fosc/32,CH0
    movwf  ADCON0              ; e inicia el convertidor A/D
    clrf   ADRES               ; Limpia Registro de Resultados
    return                    ; Retorno de subrutina
```

5.3 Receptor/Transmisor serial

Los pasos siguientes son necesarios para realizar una **transmisión asíncrona**:

1. Iniciar el registro `SPBRG`, para elegir la velocidad de transmisión deseada.
2. Habilitar el puerto serie asíncrono limpiando el bit `SYNC` y activando el bit `SPEN`.
3. Si se desean interrupciones, habilitar el bit `TXIE`.
4. Si la transmisión es de 9-bit, activar el bit `TX9`.
5. Habilitar la transmisión, activando el bit `TXEN`, cuando se inicie activar el bit `TXIF`.
6. Si se ha seleccionado la transmisión de 9 bits, el bit 9 se carga en el bit `TX9D`.
7. Cargar el **dato** en el registro `TXREG` (inicia transmisión).
8. Si se usan interrupciones, asegurarse que los bits `GIE` y `PEIE` de `INTCON` se encuentren activos.

Los pasos siguientes son necesarios para realizar una **recepción asíncrona**:

1. Iniciar el registro `SPBRG`, para elegir la velocidad de recepción deseada.
2. Habilitar el puerto serie asíncrono limpiando el bit `SYNC` y activando el bit `SPEN`.
3. Si se desea activar as interrupciones, activar el bit `RCIE`.
4. Si la recepción es a 9 bits, activar el bit `RX9`.
5. Habilitar la recepción, activando el bit `CREN`.
6. La bandera `RCIF` se activa cuando la recepción se ha completado y una interrupción puede ser generada si el bit `RCIE` esta activo.
7. Leer el registro `RSTA` para cargar el 9 bit (si está habilitado) y determinar si un error ha ocurrido durante la recepción.
8. Leer el **dato** recibido leyendo el registro `RREG`.
9. Si un error ha ocurrido limpiar la bandera `CREN`.
10. Si se usan interrupciones, asegurarse que los bits `GIE` y `PEIE` de `INTCON` se encuentren activos.

Ejemplo: A continuación se muestra el circuito (**Figura 5.2**) que captura una señal analógica y la envía a través del **puerto serie del PIC** a un **puerto serie de la PC** (**Programa 5.1**).

```
; Empleo de un canal A/D y de la comunicación por RS-232
; Este programa muestra la conversión A/D a través del canal 1 (CH0).

    LIST                P = 16F73          ; Microcontrolador empleado
    ERRORLEVEL          -302
    __CONFIG            B'00011001'        ; Oscilador a Cristal XT

W           EQU         0x00              ; Referencia al acumulador
F           EQU         0x01              ; Referencia al registro fuente
STATUS      EQU         0x03              ; Registro de Estado
```

Programa 5.1: Comunicación serie PIC-PC (Parte 1/3).

```

ADCON0    EQU    0x1F        ; Registro 0 del convertidor A/D
ADCON1    EQU    0x9F        ; Registro 1 del convertidor A/D
ADRES     EQU    0x1E        ; Registro de resultados del convertidor A/D
RP0       EQU    0x05        ; Dirección del banco para direccionamiento
adini     EQU    0x02        ; Variable de inicio de conversión
PIR1      EQU    0x0C        ; Registro para comunicación serie RS232
RCSTA     EQU    0x18        ; Registro para comunicación serie RS232
TXREG     EQU    0x19        ; Registro para comunicación serie RS232
D1        EQU    0x20        ; Variable temporal de pérdida de tiempo 1
D2        EQU    0x21        ; Variable temporal de pérdida de tiempo 2
D3        EQU    0x22        ; Variable temporal de pérdida de tiempo 3

        ORG    0x00        ; Inicio de la memoria
        goto   inicio        ; Salta vectores de interrupción
        org    0x10          ; Inicia almacenamiento desde la localidad 04h

inicio
        call   Inicia_AD      ; Inicia el convertidor A/D
        call   ini_SERIE_TX   ; Inicia el puerto serie

actualiza
        bcf    STATUS,RP0     ; Banco 0
        movf   ADRES,W        ; (DATO) Lee el valor del convertidor A/D
        movwf  TXREG          ; Carga el valor en TXREG e inicia transmisión
        call   Retardo        ; Pierde un tiempo
        bcf    ADCON0,adini   ; Limpia banderas de interrupción
        bsf    ADCON0,adini   ; inicia una nueva conversión

a_1
        btfss  ADCON0,adini   ; Terminó la conversión A/D
        goto   actualiza      ; Si, se actualiza el valor
        goto   a_1           ; No, sigue esperando

; Puerto SERIE: 9600bps, 8bits de datos, Paridad = Ninguna
ini_SERIE_TX                ; Inicia el puerto serie
        bsf    STATUS,RP0     ; Banco 1
        movlw  0x19           ; Carga W con 25dec (9600 bpd)
        movwf  TXREG          ; Configura velocidad del puerto 9600bps
        bsf    RCSTA,2        ; BRGH = 1
        bcf    RCSTA,4        ; SYNC = 0
        bcf    STATUS,RP0     ; Banco 0
        bsf    RCSTA,7        ; SPEN = 1
        bsf    STATUS,RP0     ; Banco 1
        bcf    PIR1,4         ; TXIE = 0
        bcf    RCSTA,6        ; TX9 = 0
        bsf    RCSTA,5        ; TXEN = 1
        bsf    PIR1,4         ; TXIF = 1
        return              ; Retorno de Subrutina

; Inicia el convertidor A/D, selecciona los canales
; CH0 al CH3 como entradas analógicas, fosc/2 y CH3 como lectura
Inicia_AD
        bsf    STATUS,RP0     ; Selecciona el banco 1
        clrf   ADCON1         ; Selecciona CH0-CH3 como entradas analógicas
        bcf    STATUS,RP0     ; Selecciona el banco 0
        movlw  B'10000001'    ; Selecciona Fosc/32,CH0
        movwf  ADCON0         ; e inicia el convertidor A/D
        clrf   ADRES          ; Limpia Registro de Resultados
        return              ; Retorno de subrutina

```

Programa 5.1: Comunicación serie PIC-PC (Parte 2/3).

```

; Rutina de pérdida de tiempo
Retardo
    movlw    0x02          ; Mueve 09 a W
    movwf    D3            ; Carga D3 con 0F
    movlw    0xFF          ; Mueve FF a W
    movwf    D2            ; Carga D2 con 0F
    movlw    0xFF          ; Mueve FF a W
    movwf    D1            ; Carga D1 con 0F
SD1   decfsz  D1, F        ; Decrementa F
      goto   SD1          ; Si no es cero brinca a SD
      decfsz  D2, F        ; Decrementa F
      goto   SD1          ; Si no es cero brinca a SD
      decfsz  D3, F        ; Decrementa F
      goto   SD1          ; Si no es cero brinca a SD
      return              ; Si es cero regresa

END                                ; Fin del programa

```

Programa 5.1: Comunicación serie PIC-PC (Parte 3/3).

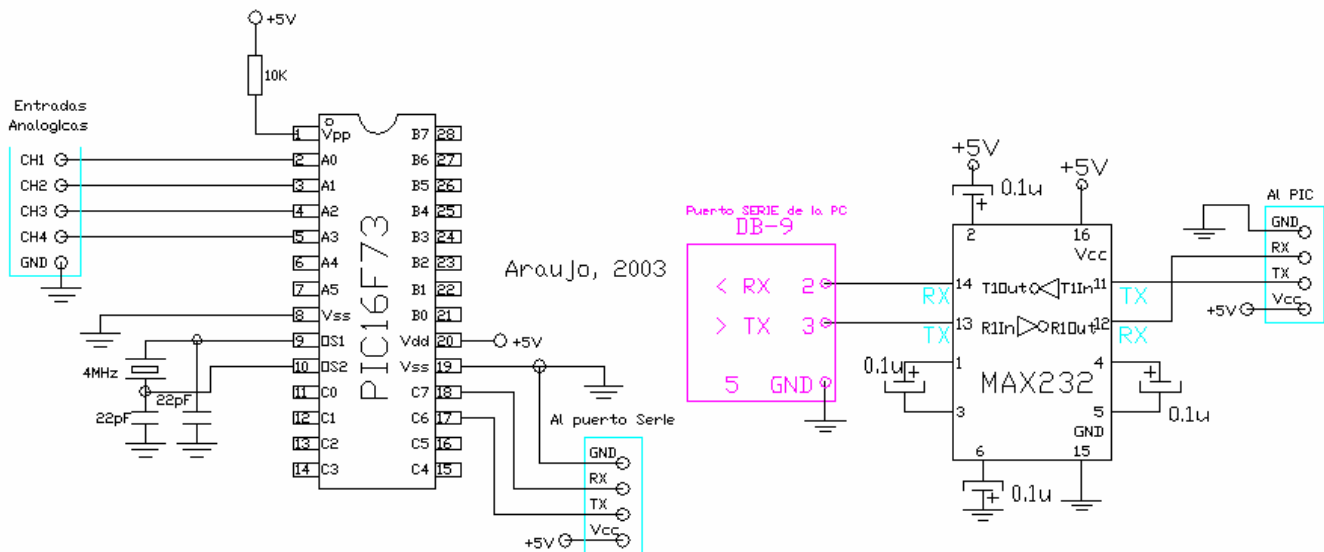


Figura 5.2: Puerto serie PIC-PC.

Capítulo 6: Aplicaciones

Aquí describimos una serie de circuitos prácticos que ejemplifican el uso de PIC.

6.1 Luces secuenciales

El **Programa 6.1** realiza el encendido secuencial de LED, a través del **puerto B** del **PIC16F84**, el circuito se muestra en la **Figura 6.1**, se emplea el tipo de **oscilador interno RC**.

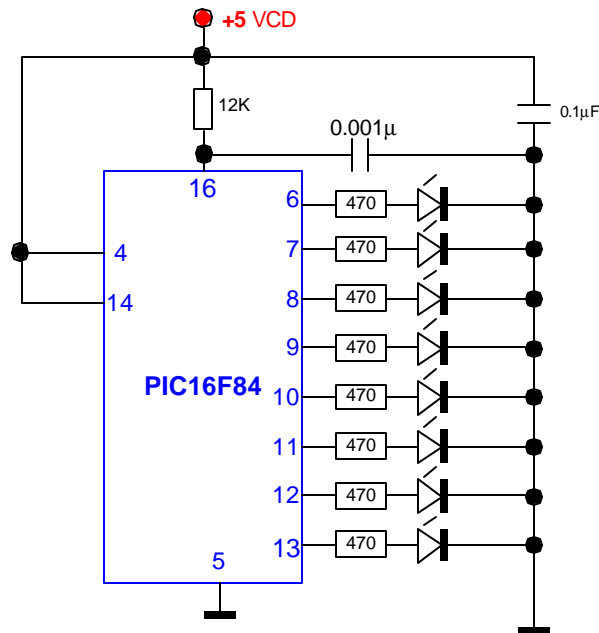


Figura 6.1: Circuito de prueba para el PIC.

```

LIST P=16F84
PORTB EQU 6
TRISB EQU 86H
OPTREG EQU 81H
STATUS EQU 3
CARRY EQU 0
RP0 EQU 5
MSB EQU 7
CLRF PORTB ; Posición del BIT más a la izquierda
BSF STATUS,RP0 ; LEDs apagados
CLRF TRISB^80H ; Banco de registros 1
MOVLW 0AH ; Puerto B configurado para salida
MOVWF OPTREG^80H ; PRESCALER (1:4) asignado al WatchDog
BCF STATUS,RP0 ; Banco de registros 0
INCF PORTB,F ; Encender LED de la derecha
BCF STATUS,CARRY ; Carry=0
LEFT SLEEP ; Esperar WDT
RLF PORTB,F ; Encender siguiente LED a la izquierda
BTFSF PORTB,MSB ; ¿Alcanzado final por la izquierda?
GOTO LEFT ; Si no, siguiente
RIGHT SLEEP ; Esperar WDT
RRF PORTB,F ; Encender siguiente LED a la derecha
BTFSF PORTB,0 ; ¿Alcanzado final por la derecha?
GOTO RIGHT ; NO: repetir
GOTO LEFT ; SI: Comienza un nuevo ciclo.
END

```

Programa 6.1: Programa en ensamblador para el circuito de prueba.

6.2 Circuito de desarrollo para el PIC16X84

En la **Figura 6.2** se muestra el circuito de la tarjeta de desarrollo de **PIC16X84**. Esta tarjeta puede ser empleada para desarrollar diversos proyectos, sin necesidad de armar diferentes circuitos.

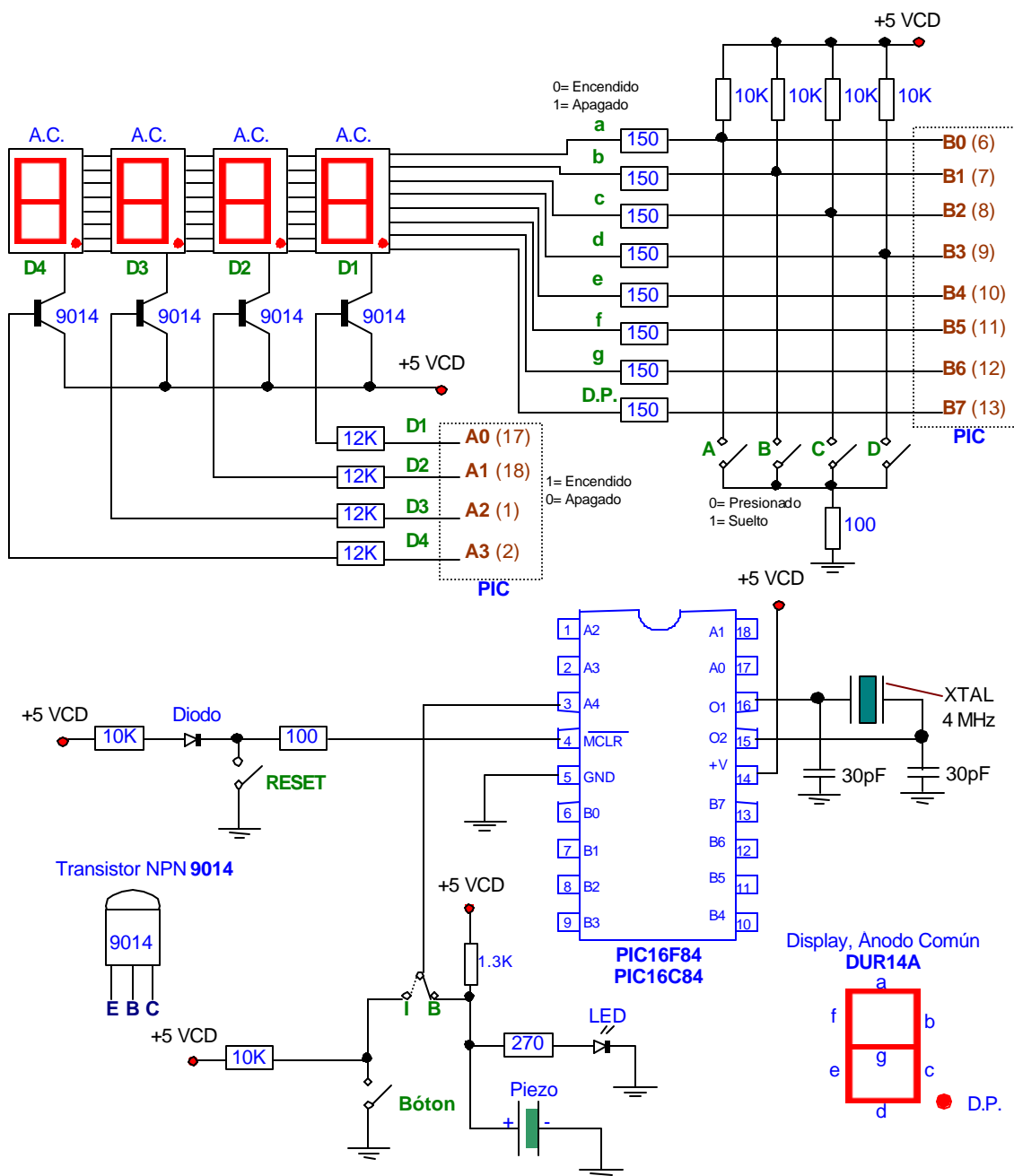


Figura 6.2: Tarjeta de desarrollo para PIC.

6.2.1 Codificación de caracteres en la tarjeta de desarrollo para el PIC16X84

En la **Tabla 6.1** se encuentra la lista de los caracteres que es posible generar en cada uno de los dígitos de la tarjeta (un cero enciende el segmento).

Carácter	D.P. g f e	d c b a	Hexadecimal
0	1100	0000	C0
1	1111	1001	F9
2	1010	0100	A4
3	1011	0000	B0
4	1001	1001	99
5	1001	0010	92
6	1000	0010	82
7	1111	1000	F8
8	1000	0000	80
9	1001	0000	90
a	1000	1000	88
b	1000	0011	83
c	1010	0111	A7
d	1010	0001	A1
e	1000	0110	86
f	1000	1110	8E
g	1001	0000	90
h	1000	1011	8B
i	1111	1011	FB
j	1111	0001	F1
l	1100	0111	C7
n	1010	1011	AB
o	1010	0011	A3
p	1000	1100	8C
q	1001	1000	98
r	1010	1111	AF
s	1001	0010	92
t	1000	0111	87
u	1110	0011	E3
z	1010	0100	A4
Punto Decimal	0111	1111	7F
Guión Alto	1111	1110	FE
Signo menos	1011	1111	BF
Guión Bajo	1111	0111	F7
Espacio en blanco	1111	1111	FF

Tabla 6.1: Caracteres en la Tarjeta de PIC.

6.2.2 Contador ascendente de cuatro dígitos

Para el contador cada uno de los botones de la tarjeta tiene la función que se muestra en la [Tabla 6.2](#), el [Programa 6.2](#), se tiene el código en lenguaje ensamblador, este no se muestra por que es extenso, pero se encuentra en el disco.

Botón	A	B	C	D	RESET	A4
Función	Inicio	No usado	No usado	No usado	RESET	Entrada (I)

Tabla 6.2: Función de los botones para el contador.

6.2.3 Reloj Digital de 12 horas

Para el reloj digital cada uno de los botones de la tarjeta tiene la función que se muestra en la [Tabla 6.3](#), el [Programa 6.3](#), se tiene el código en lenguaje ensamblador, este no se muestra por que es extenso, pero se encuentra en el disco.

Botón	A	B	C	D	RESET	A4
Función	Inicio	Avance lento	Avance rápido	No usado	RESET	No usado

Tabla 6.3: Función de los botones para el reloj.

6.2.4 Marquesina de mensajes con alarma sonora

Para la marquesina cada uno de los botones de la tarjeta tiene la función que se muestra en la [Tabla 6.4](#), el [Programa 6.4](#), se tiene el código en lenguaje ensamblador, este no se muestra por que es extenso, pero se encuentra en el disco.

Botón	A	B	C	D	RESET	A4
Función	Inicio	Mensaje 1	Mensaje 2	Mensaje 3	RESET	Altavoz (B)

Tabla 6.4: Función de los botones para la marquesina.

6.2.5 Contador Ascendente/Descendente de cuatro dígitos

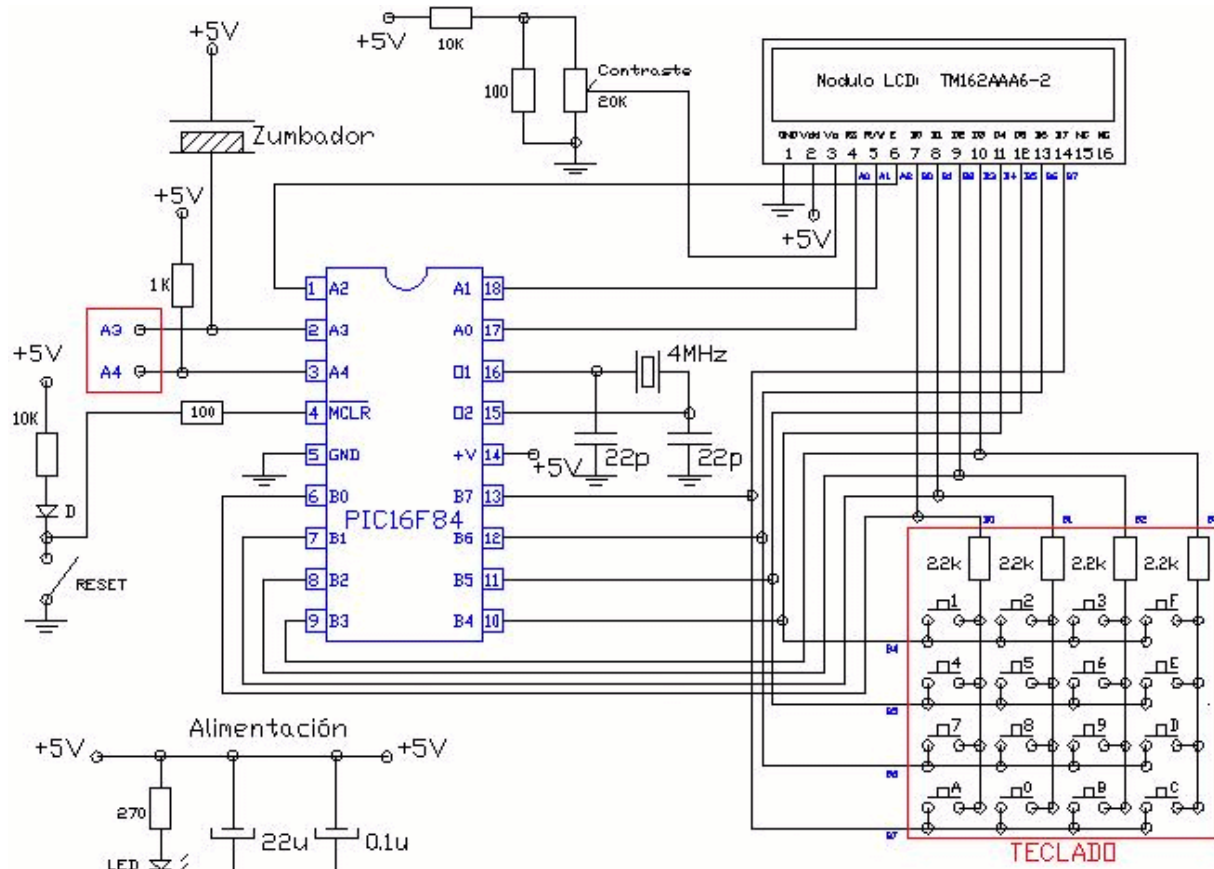
Para el contador ascendente/descendente, cada uno de los botones de la tarjeta tiene la función que se muestra en la [Tabla 6.5](#), el [Programa 6.5](#), se tiene el código en lenguaje ensamblador, este no se muestra por que es extenso, pero se encuentra en el disco.

Botón	A	B	C	D	RESET	A4
Función	Inicio	Ascendente	Descendente	No usado	RESET	No usado

Tabla 6.5: Función de los botones para el contador ascendente/descendente.

6.3 Cerradura electrónica con pantalla LCD

Este es un circuito de control de acceso, el cual emplea muchas de las características importantes de un PIC. El sistema permite acceder mediante una contraseña, pero permite cambiar la contraseña. Emplea la EEPROM para almacenarla, además ilustra la forma en que se emplean el teclado y la transferencia de información a un LCD, a través de líneas compartidas (**Figura 6.3** y **Programa 6.6** en disco). Se encuentran implementadas todas las rutinas de comunicación con el módulo LCD, las cuales se pueden emplear para otros modelos de módulos LCD similares.



Araujo, México, Año 2001

Figura 6.3: Cerradura electrónica con PIC y LCD.

6.4 Generador de barras de gris (PIC16F84)

El programa genera una pantalla que contiene:

- Marquesina con 7 caracteres visibles
- 8 Barras de gris
- Reloj con 4 dígitos

El circuito se observa en la **Figura 6.4**. Se generan ocho barras de gris mediante el convertidor analógico/digital formado por los resistores de las terminales 1, 2 y 18, por la terminal 17 se obtiene el sincronismo y por la terminal 10 los pulsos de los caracteres (**Programa 6.7** en disco).

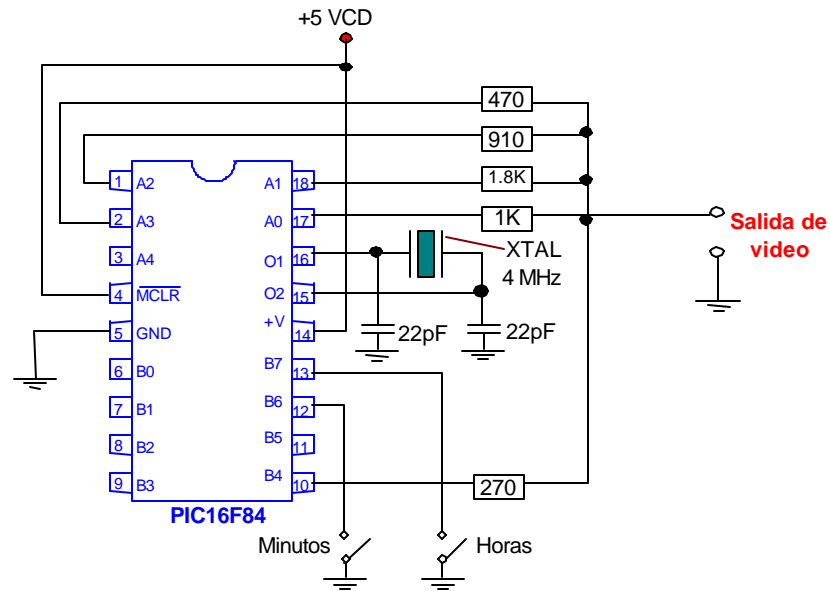


Figura 6.4: Circuito de para el Generador de Barras de Gris.

La señal de vídeo generada debe de ajustarse a las normas del sistema de televisión correspondiente, las características para los sistemas de televisión **PAL-N** y **NTSC-M** se muestran en la **Tabla 6.6**.

Característica	NTSC-M
Frecuencia de campo	60 Hz
Frecuencia horizontal (H)	15,750 Hz
Información	63.5 μ s
Intervalo de borrado	21H
Línea	58.8 μ s
Líneas por campo	262.5 líneas
Periodo de campo	16.7 ms
Pulso de Borrado	10.9 μ s
Pulso de sincronismo	4.7 μ s
a) Igualadores largos	3H en 6 pulsos
b) Igualadores cortos	3H en 6 pulsos
c) Igualadores largos	3H en 6 pulsos

Tabla 6.6: Características de la señal de vídeo.

La pantalla resultante se muestra en la **Figura 6.5**, según el sistema empleado.



Figura 6.5: Salida del Generador de Barras de Gris.

Apéndice A: Cuadro de Instrucciones

En este apartado, se describe el cuadro mínimo de instrucciones comunes para la mayoría de los microcontroladores de la marca **Microchip**. Se presenta un resumen con las instrucciones y a continuación la descripción detallada de cada una de ellas.

A.1 Resumen de Instrucciones

A.1.1 Instrucciones que manejan registros

Nemónicos	Operandos	Descripción	Ciclos	Banderas	Notas
ADDWF	f, d	Suma w y f	1	C, DC, Z	1, 2, 4
ANDWF	f, d	AND w con f	1	Z	2, 4
CLRF	f	Borra f	1	Z	4
CLRWF	---	Borra w	1	Z	---
COMF	f, d	Complementa f	1	Z	---
DECf	f, d	Decrementa f	1	Z	2, 4
DECFSZ	f, d	Decrementa f, si es 0 salta	1 (2)	Ninguno	2, 4
INCF	f, d	Incrementa f	1	Z	2, 4
INCFsZ	f, d	Incrementa f, si es 0 salta	1	Ninguno	2, 4
IORWF	f, d	OR entre w y f	1	Z	2, 4
MOVF	f, d	Mueve f	1	Z	2, 4
MOVWF	f	Mueve w a f	1	Ninguno	1, 4
NOP	---	No operación	1	Ninguno	---
RLF	f, d	Rota f a la izqda. a través del acarreo	1	C	2, 4
RRF	f, d	Rota f a la dcha. a través del acarreo	1	C	2, 4
SUBWF	f, d	Resta a f el reg. w	1	C, DC, Z	1, 2, 4
SWAPF	f, d	Intercambia los nibbles f	1	Ninguno	2, 4
XORWF	f, d	XOR de w con f	1	Z	2, 4

A.1.2 Instrucciones de control y de operandos inmediatos

Nemónicos	Operandos	Descripción	Ciclos	Banderas	Notas
ANDLW	k	AND inmediato con w	1	Z	---
ADDLW	k	Suma k + w	1	C, DC, Z	---
CALL	k	Llamada a subrutina	2	Ninguno	1
CLRWDt	---	Borra Watchdog	1	~TO, ~PD	---
GOTO	k	Salto incondicional	2	Ninguno	---
IORLW	k	OR inmediato con w	1	Z	---
MOVLW	k	Mueve a w un valor inmediato	1	Ninguno	---
OPTION	---	Carga el registro OPTION	1	Ninguno	---
RETFIE	---	Retorno de interrupción	2	---	---
RETLW	k	Retorno y carga de w	2	Ninguno	---
RETURN	---	Retorno de Subrutina	2	---	---
SLEEP	---	Pasa a estado de reposo	1	~TO, ~PD	---
TRIS	f	Carga el registro	1	Ninguno	3
SUBLW	k	Resta w - k	1	C, DC, Z	---
XORLW	k	OR exclusiva a w	1	Z	---

A.1.3 Instrucciones que manipulan Bits

Nemónicos	Operandos	Descripción	Ciclos	Banderas	Notas
BCF	f, b	Borra bit de f	1	Ninguno	2, 4
BSF	f, b	Pone a 1 el bit de f	1	Ninguno	2, 4
BTFSC	f, b	Comprueba un bit de f y salta si vale 0	1 (2)	Ninguno	---
BTFSS	f, b	Comprueba un bit de f y salta si vale 1	1 (2)	Ninguno	---

Notas:

1. El noveno bit del PC siempre es 0, en todas las instrucciones que escriban sobre el PC, con excepción de GOTO.
2. Al modificar un registro de E/S con una operación sobre él mismo, el valor usado es el que se halle presente en las terminales del puerto.
3. La instrucción TRIS f, siendo f = 5, 6 ó 7, origina que el contenido del registro w se escriba en los FF triestado que configuran bs puertos A, B ó C, respectivamente. Un "1" fuerza a la terminal a ponerse en alta impedancia y desactiva los buffers de salida.
4. Si se ejecuta esta instrucción sobre el TMR0 y d=1, será borrado el divisor de frecuencia (prescaler) si está asignado al TMR0.

A.2 Cuadro de Instrucciones

• **ADDLW Suma W y una literal**

Sintaxis: [Label] ADDLW k

Operandos: $0 \leq k \leq 255$

Operación: $(W) + k \Rightarrow (W)$

Banderas afectadas: C, DC, Z

Descripción: Suma el contenido del registro w y la literal k.

Ejemplo: ADDLW 0x04

Antes de la instrucción: W = 0x02 k = 0x04

Después de la instrucción: W = 0x06

• **ADDWF Suma W y f**

Sintaxis: [Label] ADDWF f, d

Operandos: $d \in [0, 1], 0 \leq f \leq 31$

Operación: $(W) + (f) \Rightarrow (dest)$

Banderas afectadas: C, DC, Z

Descripción: Suma el contenido del registro w y el registro f. Si d es 0, el resultado se almacena en el registro w. Si d es 1 el resultado se almacena en el registro f.

Ejemplo: ADDWF REG, 0

Antes de la instrucción: W = 0x17 REF = 0xC2

Después de la instrucción: W = 0xD9 REG = 0xC2

• **ANDWF W AND f**

Sintaxis: [Label] ANDWF f, d

Operandos: $d \in [0, 1], 0 \leq f \leq 31$

Operación: $(W) \text{ AND } (f) \Rightarrow (dest)$

Banderas afectadas: Z

Descripción: Realiza la operación lógica AND entre el registro w y el registro f. Si d es 0 el resultado se almacena en el registro w. Si d es 1, el resultado se almacena en el registro f.

Ejemplo: ANDWF REG, 1

Antes de la instrucción: W = 0x17 REG = 0xC2

Después de la instrucción: W = 0x17 REG = 0x02

- ANDLW** **W AND literal**
Sintaxis: [Label] ANDLW k
Operandos: $0 \leq k \leq 255$
Operación: $(W) \text{ AND } (k) \Rightarrow (W)$
Banderas afectadas: Z

Descripción: Realiza la operación lógica **AND** entre el registro **w** y la constante **k**. El resultado se almacena en el registro **w**.

Ejemplo: ANDLW 0x5F
 Antes de la instrucción: W = 0xA3
 Después de la instrucción: W = 0x03

- BCF** **Borra un bit**
Sintaxis: [Label] BCF f,b
Operandos: $0 \leq f \leq 31; 0 \leq b \leq 7$
Operación: $0 \Rightarrow (f)$
Banderas afectadas: Ninguna
Descripción: Borra el bit **b** del registro **f**.
Ejemplo: BCF REG,7
 Antes de la instrucción: REG = 0xC7
 Después de la instrucción: REG = 0x47

- BSF** **Activa un bit**
Sintaxis: [Label] BSF f,b
Operandos: $0 \leq f \leq 31; 0 \leq b \leq 7$
Operación: $1 \Rightarrow (f)$
Banderas afectadas: Ninguna
Descripción: Activa el bit **b** del registro **f**.
Ejemplo: BSF REG,0x0A
 Antes de la instrucción: REG = 0x0A
 Después de la instrucción: REG = 0x8A

- BTFSC** **Prueba el bit y si es cero salta**
Sintaxis: [Label] BTFSC f,b
Operandos: $0 \leq f \leq 31; 0 \leq b \leq 7$
Operación: Salta si $(f) = 0$
Banderas afectadas: Ninguna
Descripción: Si el bit **b** del registro **f** es 0, salta una instrucción y se continúa con la ejecución.
Ejemplo:

```
COM    BTFSC REG,1
FALSE  GOTO  PROCESA_X
TRUE   ---
```

Antes de la instrucción: PC = Dirección(COM)
 Después de la instrucción: Si REG <1> = 0 PC = Dirección (TRUE)
 Si REG <1> = 1 PC = Dirección (FALSE)

- BTFSS** **Prueba el bit y si es uno salta**
Sintaxis: [Label] BTFSS f,b
Operandos: $0 \leq f \leq 31; 0 \leq b \leq 7$
Operación: Salta si $(f) = 1$
Banderas afectadas: Ninguna
Descripción: Si el bit **b** del registro **f** es 1, salta una instrucción y se continúa con la ejecución.
Ejemplo:

```
COM    BTFSS REG,6
FALSE  GOTO  PROCESA_X
TRUE   ---
```

Antes de la instrucción: PC = Dirección(COM)

Después de la instrucción: Si REG <6> = 0 PC = Dirección (FALSE)
Si REG <6> = 1 PC = Dirección (TRUE)

- CALL** **Salto a subrutina**
Sintaxis: [Label] CALL k
Operandos: $0 \leq k \leq 255$
Operación: (PC) + 1 => stack; k = PC
Banderas afectadas: Ninguna

Descripción: Salto a subrutina. La dirección de retorno se guarda en la pila (**stack**). La constante **k** de 8 bits forma la dirección de salto y se carga en los bits <7:0> del **PC**. Los bits <10:9> del **PC** se cargan con los bits <6:5> del registro de estado (**STATUS**). **PC** pone <8> a 0.

Ejemplo: ORG CALL SUBROUTINA
 Antes de la instrucción: PC = ORG
 Después de la instrucción: PC = SUBROUTINA

- CLRF** **Borra un registro**
Sintaxis: [Label] CLRF f
Operandos: $0 \leq f \leq 32$
Operación: $0x00 \Rightarrow (f); 1 \Rightarrow Z$
Banderas afectadas: Z
Descripción: Se borra el contenido del registro **f** y la bandera **z** se activa.
Ejemplo: CLRF REG
 Antes de la instrucción: REG = 0x5F
 Después de la instrucción: REG = 0x00 Z = 1

- CLRW** **Borra el registro W**
Sintaxis: [Label] CLRW
Operandos: Ninguno
Operación: $0x00 \Rightarrow W; 1 \Rightarrow Z$
Banderas afectadas: Z
Descripción: Se borra el contenido del registro de trabajo **w** y la bandera **z** se activa.
Ejemplo: CLRW
 Antes de la instrucción: W = 0x5F
 Después de la instrucción: W = 0x00 Z = 1

- CLRWD** **Borra el Watchdog**
Sintaxis: [Label] CLRWD
Operandos: Ninguno
Operación: $0x00 \Rightarrow WDT; 1 \Rightarrow \sim TO; 1 \Rightarrow \sim PD$
Banderas afectadas: $\sim TO, \sim PD$
Descripción: Borra el contenido del registro **watchdog** y el **prescaler**. Los bits $\sim TO$ y $\sim PD$ del registro de estado se ponen a 1.

Ejemplo: CLRWD
 Después de la instrucción: WDT = 0 (Contador)
 WDT = 0 (Prescaler)
 Bit de estado TO = 1
 Bit de estado PD = 1

- COMF** **Complementa f**
Sintaxis: [Label] COMF f,d
Operandos: $d \in [0,1], 0 \leq f \leq 31$
Operación: (f) => (dest)
Banderas afectadas: Z
Descripción: El contenido del registro **f** se complementa. Si **d** es 0, el resultado se almacena en el

registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: COMF REG,0
 Antes de la instrucción: REG = 0x13
 Después de la instrucción: REG = 0x13 W = 0xEC

- **DECF** **Decrementa f**
Sintaxis: [Label] DECF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 31$
Operación: $(f) - 1 \Rightarrow (dest)$
Banderas afectadas: Z

Descripción: Se decrementa en una unidad el contenido del registro **f**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: DECF CONT,1
 Antes de la instrucción: CONT = 0x01 z = 0
 Después de la instrucción: CONT = 0x00 z = 1

- **DECFSZ** **Decrementa y salta si es cero**
Sintaxis: [Label] DECFSZ f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 32$
Operación: $(f) - 1 \Rightarrow d$; Salta si Resultado = 0
Banderas afectadas: Ninguna

Descripción: El contenido del registro **f**, se decrementa. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**. Si el resultado (**R**) es cero, se salta la siguiente instrucción, y se continúa con la ejecución.

Ejemplo: COM1 DECFSZ REG,0
 GOTO NO_ES_0

 CONTINUA ---
 Antes de la instrucción: PC = Dirección(COM)
 Después de la instrucción: REG = REG - 1 Si REG = 0 PC = (CONTINUA)
 Si REG \neq 0 PC = (COM1 + 1)

- **GOTO** **Salto incondicional**
Sintaxis: [Label] GOTO k
Operandos: $0 \leq k \leq 511$
Operación: $k \Rightarrow PC \Rightarrow \langle 8:0 \rangle$
Banderas afectadas: Ninguna

Descripción: Se trata de un salto incondicional. Los 9 bits de la constante **k**, se cargan en los bits $\langle 8:0 \rangle$ del PC y forman la dirección de salto. Los bits $\langle 9:0 \rangle$ del PC se cargan con los bits $\langle 6:5 \rangle$ del registro de estado.

Ejemplo: ORG GOTO destino
 Antes de la instrucción: PC = 0
 Después de la instrucción: PC = Destino

- **INCF** **Incrementa f**
Sintaxis: [Label] INCF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 31$
Operación: $(f) + 1 \Rightarrow (dest)$
Banderas afectadas: Z

Descripción: Se incrementa en una unidad el contenido del registro **f**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: INCF CONT,1
 Antes de la instrucción: CONT = 0xFF z = 0
 Después de la instrucción: CONT = 0x00 z = 1

- **INCFSZ** **Incrementa f y salta, si es cero**

Sintaxis: [Label] INCFSZ f,d

Operandos: $d \in [0,1]$, $0 \leq f \leq 31$

Operación: $(f) + 1 \Rightarrow (dest)$; Salta si Resultado = 0

Banderas afectadas: Ninguna

Descripción: Se incrementa en una unidad el contenido del registro **f**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**. Si el resultado (**R**) es cero, se salta la siguiente instrucción, y se continúa con la ejecución.

Ejemplo:

```
COM1  INCFSZ REG,0
      GOTO NO_ES_0
      CONTINUA    ---
```

Antes de la instrucción: PC = Dirección(COM)

Después de la instrucción: REG = REG + 1 Si REG = 0 PC = (CONTINUA)
 Si REG ≠ 0 PC = (COM1 + 1)

- **IORLW** **W OR literal**

Sintaxis: [Label] IORLW k

Operandos: $0 \leq k \leq 255$

Operación: $(W) \text{ OR } (k) \Rightarrow (W)$

Banderas afectadas: Z

Descripción: Realiza la operación lógica **OR** entre el registro **w** y la constante **k**. El resultado se almacena en el registro **w**.

Ejemplo: IORLW 0x35

Antes de la instrucción: W = 0x9A

Después de la instrucción: W = 0xBF

- **IORWF** **OR entre W y f**

Sintaxis: [Label] IORWF f,d

Operandos: $d \in [0,1]$, $0 \leq f \leq 31$

Operación: $(W) \text{ OR } (f) \Rightarrow (dest)$

Banderas afectadas: Z

Descripción: Realiza la operación lógica **OR** entre el registro **w** y el registro **f**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: IORWF REG,0

Antes de la instrucción: R = 0x13 W = 0x91

Después de la instrucción: R = 0x13 W = 0x93 Z = 0

- **MOVF** **Mover a f**

Sintaxis: [Label] MOVF f,d

Operandos: $d \in [0,1]$, $0 \leq f \leq 31$

Operación: $(f) \Rightarrow (dest)$

Banderas afectadas: Z

Descripción: El contenido del registro **f** se mueve al destino **d**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**. Se puede emplear para verificar el registro **f**, pues se afecta la bandera de cero **Z**.

Ejemplo: MOVF REG,0

Después de la instrucción: W = REG

- **MOVLW** **Carga una literal en W**

Sintaxis: [Label] MOVLW k

Operandos: $0 \leq k \leq 255$

Operación: $(k) \Rightarrow W$

Banderas afectadas: Ninguna

Descripción: El registro **w** se carga con el valor de la literal **k**.

Ejemplo: MOVLW 0x5A

Después de la instrucción: W = 0x5A

- MOVWF** **Mover W a f**
Sintaxis: [Label] MOVWF f
Operandos: $0 \leq f \leq 31$
Operación: (W) => (dest)
Banderas afectadas: Ninguna
Descripción: El contenido del registro **w** al registro **f**.

Ejemplo: MOVWF REG

Antes de la instrucción: REG = 0xFF W = 0x4F

Después de la instrucción: REG = 0x4F W = 0x4F

- NOP** **No operación**
Sintaxis: [Label] NOP
Operandos: Ninguno
Operación: No operación
Banderas afectadas: Ninguna
Descripción: No realiza ninguna operación. Consume un ciclo de reloj.
Ejemplo: NOP

- OPTION** **Carga el registro OPTION**
Sintaxis: [Label] OPTION
Operandos: Ninguno
Operación: (W) => OPTION
Banderas afectadas: Ninguna
Descripción: El contenido del registro **w** se carga en el registro **OPTION**.
Ejemplo: OPTION
Antes de la instrucción: W = 0x07
Después de la instrucción: OPTION = 0x07

- RETLW** **Retorno, carga W**
Sintaxis: [Label] RETLW k
Operandos: $0 \leq k \leq 255$
Operación: (k) => W; TOS => PC
Banderas afectadas: Ninguna

Descripción: El registro **w** se carga con el valor de la constante **k**. El **PC** se carga con el contenido del tope de la pila (**TOS**). Se consumen dos ciclos de reloj

Ejemplo: CALL TABLA ; W contiene el offset de la tabla

TABLA ADDWF PC ; W offset

RETLW k1 ; Inicio de la tabla

RETLW k2

RETLW kn ; Fin de la tabla

Antes de la instrucción: W = 0x07

Después de la instrucción: W = Valor de k8

- RETFIE** **Retorno de interrupción**
Sintaxis: [Label] RETFIE
Operandos: ---
Operación: TOS => PC, 1 => GIE
Banderas afectadas: Ninguna
Descripción: Regresa de una interrupción.

- RLF** **Rota f a la izquierda**
Sintaxis: [Label] RLF f,d
Operandos: $d \in [0,1], 0 \leq f \leq 31$
Operación: Rota a la izquierda f
Banderas afectadas: C

Descripción: El contenido del registro f se rota una posición a la izquierda. El bit de más peso de f pasa al acarreo (C), y el acarreo se introduce al bit de menos peso de f . Si d es 0, el resultado se almacena en el registro w . Si d es 1 el resultado se almacena en el registro f .

Ejemplo: RLF REG,0
 Antes de la instrucción: REG = 1110 0110
 C = 0
 Después de la instrucción: REG = 1110 0110
 W = 1100 1100
 C = 1

- RRF** **Rota f a la derecha**
Sintaxis: [Label] RRF f,d
Operandos: $d \in [0,1], 0 \leq f \leq 31$
Operación: Rota a la derecha f
Banderas afectadas: C

Descripción: El contenido del registro f se rota una posición a la derecha. El bit de menos peso de f pasa al acarreo (C), y el acarreo se introduce al bit de más peso de f . Si d es 0, el resultado se almacena en el registro w . Si d es 1 el resultado se almacena en el registro f .

Ejemplo: RRF REG,0
 Antes de la instrucción: REG = 1110 0110
 C = 0
 Después de la instrucción: REG = 1110 0110
 W = 0110 0110
 C = 0

- RETURN** **Retorno de subrutina**
Sintaxis: [Label] RETURN
Operandos: ---
Operación: TOS => PC
Banderas afectadas: Ninguna

Descripción: Retorna de una subrutina. Extrae de la pila TOS y lo carga en el contador de programa.

- SLEEP** **Pasa a estado de reposo**
Sintaxis: [Label] SLEEP
Operandos: Ninguno
Operación: 0x00 => WDT; 0 => WDT prescaler; 1 => ~TO; 0 => ~PD
Banderas afectadas: ~TO, ~PD, GPWUF

Descripción: Pasa al microcontrolador al estado de reposo. Al salir, activa el bit de estado ~TO y borra ~PD. El WDT y el prescaler se borran. Finalmente se detiene el oscilador.

Ejemplo: SLEEP

- SUBWF** **Resta f - W**
Sintaxis: [Label] SUBWF f,d
Operandos: $d \in [0,1], 0 \leq f \leq 32$
Operación: (f) - (W) => (dest)
Banderas afectadas: C, DC, Z

Descripción: Resta por el método de complemento a 2 el contenido del registro f menos el contenido del registro w . Si d es 0, el resultado se almacena en el registro w . Si d es 1 el resultado se almacena en el registro f . Si el resultado es negativo la bandera de acarreo C se pone en 0.

Ejemplo: SUBWF REG,1
 Antes de la instrucción: REG = 0x01 W = 0x02
 Después de la instrucción: REG = 0xFF W = 0xFF

• **SUBLW** **Resta k - W**

Sintaxis: [Label] SUBLW k

Operandos: $0 \leq k \leq 255$

Operación: $k - (W) \Rightarrow (W)$

Banderas afectadas: C, DC, Z

Descripción: Resta por el método de complemento a 2 el contenido del registro **k** menos el contenido del registro **w**.

Ejemplo: SUBLW 0x04
 Antes de la instrucción: k = 0x04 W = 0x08
 Después de la instrucción: W = 0x02

• **SWAPF** **Intercambio de los nibbles de f**

Sintaxis: [Label] SWAPF f,d

Operandos: $d \in [0,1], 0 \leq f \leq 31$

Operación: $(f <3:0>) \Rightarrow (dest <7:4>)$

$(f <7:4>) \Rightarrow (dest <3:0>)$

Banderas afectadas: Ninguna

Descripción: Los cuatro bits de más peso del registro **f** se intercambian con los cuatro bits de menos peso del mismo registro. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: SWAPF REG,0
 Antes de la instrucción: REG = 0x5A
 Después de la instrucción: REG = 0xA5 W = 0xA5

• **TRIS** **Carga el registro TRIS**

Sintaxis: [Label] TRIS f

Operandos: $5 \leq f \leq 7$

Operación: $(W) \Rightarrow \text{Registro TRIS } f$

Banderas afectadas: Ninguna

Descripción: El contenido del registro **w** se carga en uno de los registros **TRIS** (**TRISA**, **TRISB**, **TRISC**), según **f** valga 5, 6 ó 7.

Ejemplo: TRIS PORTA
 Antes de la instrucción: W = 0xA5
 Después de la instrucción: TRISA = 0xA5

• **XORLW** **W XOR literal**

Sintaxis: [Label] XORLW k

Operandos: $0 \leq k \leq 255$

Operación: $(W) \text{ XOR } (k) \Rightarrow (W)$

Banderas afectadas: Z

Descripción: Realiza la operación lógica **xor** entre el registro **w** y la constante **k**. El resultado se almacena en el registro **w**.

Ejemplo: XORLW 0xAF
 Antes de la instrucción: W = 0xB5
 Después de la instrucción: W = 0x1A

• **XORWF** **W XOR f**

Sintaxis: [Label] XORWF f,d

Operandos: $d \in [0,1], 0 \leq f \leq 31$

Operación: $(W) \text{ XOR } (f) \Rightarrow (dest)$

Banderas afectadas: Z

Descripción: Realiza la operación lógica **XOR** entre el registro **w** y el registro **f**. Si **d** es 0, el resultado se almacena en el registro **w**. Si **d** es 1 el resultado se almacena en el registro **f**.

Ejemplo: XORWF REG,1

Antes de la instrucción: REG = 0xAF W = 0xB5

Después de la instrucción: REG = 0x1A W = 0xB5

A.3 Variables

PC: Contador de programa

GPWUF: GPIO bit reset

TO: Bit Time Out

PD: Bit Power Down

Z: Bit de cero

DC: Bit de Carry/Borrow (ADDWF y SUBWF)

C: Bit de Carry (ADDWF, SUBWF, RRF y RLF)

B.3 Componentes

La distribución de los componentes se presenta a continuación (**Figura B.3**).

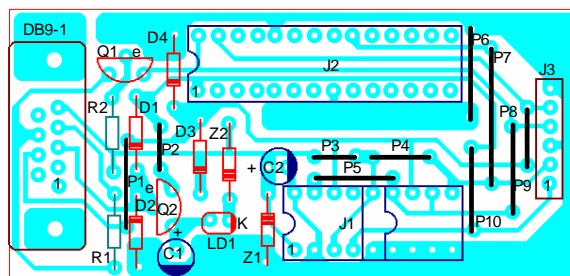


Figura B.3: Circuito impreso del programador (componentes).

La lista de componentes se describe en la **Tabla B.1**.

Nombre	Descripción
Q1, Q2	Transistor NPN PN2222
R1	Resistor 1.5K Ω @ 1/4W
R2	Resistor 10K Ω @ 1/4W
D1, D2, D3, D4	Diodos rectificadores 1N4936
Z1	Diodo zener de 5.1V 1N4733
Z2	Diodo zener de 6.2V 1N4735
LD1	Led de color rojo
C1	Capacitor electrolítico 47 μ F @ 16V
C2	Capacitor electrolítico 33 μ F @ 16V
J1	Base para C.I. de 18 terminales
J2	Base para C.I. de 28 terminales
J3	Tira de pines (6 terminales)
P1 a P10	Puentes de alambre
DB9-1	Conector DB9 hembra

Tabla B.1: Lista de componentes.

B.4 Programa

El programa empleado para usar este dispositivo es el conocido como **IC-PROG**, el cual debe de ser configurado para un programador de tipo **JDM** (**Figura B.4**).

Cada vez que se coloque o se tome un dispositivo del programador, es necesario desconectarlo del puerto, pues existe la posibilidad de dañarlo.

Para programar un PIC se sigue el siguiente procedimiento:

- Se escribe el programa en **lenguaje ensamblador** (en cualquier editor de texto) (Programa *.asm).
- Se ensambla con el **MPASM** de Microchip y se verifica que no se presenten errores (Genera *.hex).
- Se ejecuta el programa **IC-PROG** y se carga el programa *.hex.
- Se elige el **tipo de oscilador** empleado (Por ejemplo **XT** para un cristal).
- Se verifican los **bits de configuración**.
- Se **graba** el dispositivo.
- Se **verifica**, el programa almacenado en el PIC.

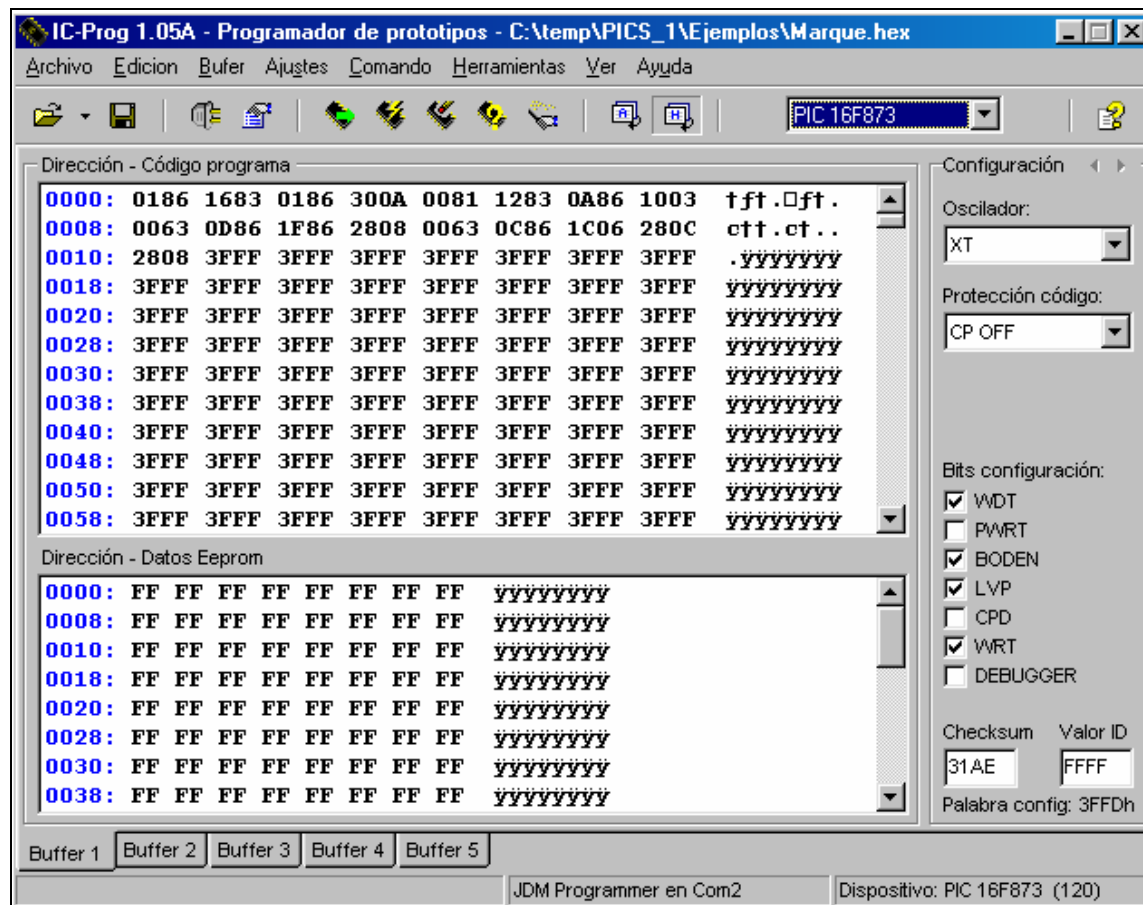


Figura B.4: Aspecto del programa para grabar PIC.

B.5 Dispositivos

Están soportados los siguientes dispositivos:

- **EEPROM 4-Wire:** 59C11, 59C22, 59C13
- **EEPROM I2C:** 24C01, 24C02, 24C04, 24C08, 24C16, 24C32, 24C64/65, 24C128, 24C256, 24C512, PCF8572 o 8572 = 24C01, PCF8582 o 8582 = 24C02, PCF8592 o 8592 = 24C04, SDE2516 = 24C??, DE2526 = 24C??, SDA2546 = 24C??, SDA2586 = 24C??, SDA3546 = 24C??, SDA3586 = 24C?? y 24LC21 = 24C01 (La patita 7 (V_{CLK}) debe conectarse a V_{CC}). Se encuentran soportadas tanto la serie C como la LC. La serie CS no es soportada.
- **Microcontroladores Flash:** 89C1051, 89C2051 y 89C4051.
- **EEPROM IM-Bus:** NVM3061, MDA2060, MDA2061 y MDA2062
- **Dispositivos PIC Microchip:** 12C508, 12C508A, 12C509, 12C509A, 12CE518, 12CE519, 12C671, 12C672, 12CE673, 12CE674, 16C433, 16C54, 16C56, 16C61, 16C62A, 16C62B, 16C63, 16C63A, 16C64A, 16C65A, 16C65B, 16C66, 16C67, 16C71, 16C72, 16C72A, 16C73A, 16C73B, 16C74A, 16C76, 16C77, 16C84, 16F83, 16F84, 16F84A, 16C505, 16C620, 16C621, 16C622, 16C622A, 16F627, 16F628 Ponga RB4 a GND, 16C715, 16F870, 16F871, 16F872, 16F873, 16F874, 16F876, 16F877 Ponga RB3 a GND, 16C923 y 16C924.
- **EEPROM Microwire:** 93C06, 93C46, 93C57, 93C56, 93C66, 93C76, 93C86, 93C13 = 93C06 y 93C14 = 93C46. Se encuentran soportadas tanto la serie C como la LC. La serie CS no es soportada.
- **EEPROM Serie Modernas:** AK6420, AK6440, AK6480, BR9010, BR9020 y BR9040, BR9080.
- **Dispositivos Scenix:** SX18AC y SX28AC
- **EEPROM Spi:** 25010, 25020, 24040, 25080, 25160, 25320, 25640,
- **Microcontroladores Spi:** 90S1200, 90S2313, 90S2323, 90S2333, 90S2343, 90S4414, 90S4433, 90S4434, 90S8515, 90S8535, 89S53 y 89S8252.

Referencias

- [1] www.microchip.com
- [2] www.ic-prog.com
- [Angulo 99] Angulo Usategui, José Ma., Angulo Martínez, Ignacio; **“Microcontroladores PIC. Diseño práctico de aplicaciones”**, 2^{da} ed., Ed. McGraw-Hill, España, 1999, 295pp.
- [Microchip 01] **“Four Channel Digital Voltmeter with Display and Keyboard”**, Microchip, 2002, 30pp.
- [Microchip 02] **“Frequency Counter Using PIC16C5X”**, Microchip, 1997, 11pp.
- [Microchip 03] **“Implementation of an Asynchronous Serial I/O”**, Microchip, 1997, 19pp.
- [Microchip 04] **“Instruction Set PICmicro MID-RANGE MCU FAMILY”**, Microchip, 1997, 48pp.
- [Microchip 05] **“PIC16F7X Data Sheet 28/40-Pin, 8-Bit CMOS Flash”**, Microchip, 2002, 174pp.
- [Microchip 06] **“PIC16F84A, Data Sheet, 18-pin Enhanced FLASH/EEPROM, 8-bit Microcontroller”**, Microchip, 2001, 88pp.
- [Microchip 07] **“PIC16F87X 28/40-Pin 8-Bit CMOS Flash Microcontrollers”**, Microchip, 1998, 200pp.
- [Microchip 08] **“PIC16F87X, Data Sheet, 28/40-Pin 8-Bit CMOS FLASH, Microcontrollers”**, Microchip, 2001, 218pp.
- [Microchip 09] **“PIC16F87XA Data Sheet, 28/40 pin Enhanced FLASH Microcontrollers”**, Microchip, 2001, 222pp.
- [Microchip 10] **“PIC16F8X 18-Pin Flash / EEPROM 8-Bit Microcontrollers”**, Microchip, 1998, 124pp.
- [Motorola 01] **“M68HC11E Series Programming Reference Guide”**, Motorola, 2002, 60pp.

