MICROCONTROLADOR

PIC 16C84

ESCUELA DE INGENIERIA

TECNICA INDUSTRIAL

BADAJOZ

SOFTWARE PIC 16C84

- **INTRODUCCION AL MICROCONTROLADOR PIC 16C84**

Casi desconocidos hace cinco años, los microcontroladores PIC 16CXX de Microchip actualmente se encuentran en aparatos muy diversos como son: programadores domésticos, telemandos, termostatos electrónicos, etc.

El motivo de su éxito se encuentra en su coste especialmente bajo, sus altas prestaciones y su reducido juego de instrucciones (RISC), así como su estructura interna del tipo Harvard.

HARVARD: Casi todos los microcontroladores actuales utilizan la estructura de Von Neuman en la que la memoria de programa contiene instrucciones y datos mecclados y se dispone de un único bus llamado bus de datos por el que circulan a la vez los códigos de las instrucciones y los datos asociados a ellas.

Esta arquitectura presenta algunos problemas cuando se demanda rapidez, con lo que es preferible utilizar la arquitectura denominada Harvard en la que instrucciones y datos están perfectamente diferenciados y se emplean buses distintos.

Esta estructura no modifica nada desde el punto de vista del usuario y la velocidad de ejecución de los programas es impresionante.

RISC: Significa Reduced Instruction Set Computer (Ordenador con juego de instrucciones reducido)

Pero no solamente significa esto sino que debe disponer de una estructura pipeline , lo cual permite ejecutar como mínimo una instrucción mientras busca la siguiente, además también permite aumentar la velocidad de ejecución en relación a los microcontroladores clásicos denominados CISC (Completex I.S.C).

Además los circuitos RISC, deben ejecutar todas las instrucciones a la misma velocidad.

El PIC 16C84 es un circuito RISC con lo que ganan en velocidad de trabajo y no es preciso trabajar con un centenar de instrucciones, sino tan solo 35, sin que esto disminuya las prestaciones.

2− SEÑALES DISPONIBLES

2.1− ALIMENTACIÓN

VSS: Masa general.

VDD: Alimentación (positiva) debe estar comprendida entre 4 y 6v.

En elementos con el sufijo HS estará entre 4,5 y 5,5v.

En modo SLEEP deberá tener una alimentación de 1,5v para que la RAM interna conserve sus contenidos.

En general todas las reglas aplicables a la alimentación de microprocesadores son

aplicables también aquí como por ejemplo seria el condensador de desacoplo (0,1uF).

- **RELOJES**

hay dos pines de reloj que son OSC1 y OSC2 pudiéndose utilizar varios tipos de reloj.

En las versiones con ventana, el tipo de reloj utilizado se puede seleccionar mediante la programación de un registro especial accesible únicamente al programar el circuito, esto se puede hacer con reloj de cristal de cuarzo o como célula RC.

En las versiones sin ventana o OTP el reloj viene predeterminado en la cápsula teniendo cuatro encapsulados distintos un función de dicho reloj interno:

XT: Oscilador de cuarzo estandart con una frecuencia máxima de 4 Mhz.

HS: Lo mismo pero de hasta 20 Mhz (High Speed).

RC: Oscilador RC funciona hasta a 4 Mhz pero con una estabilidad menor que el cuarzo.

LP: (Low power) versión para cuarzo, especial para aplicaciones de bajo consumo, su

frecuencia está limitada a 200 Mhz.

- **RESET**

La señal reset tiene como pin el denominado MLCR y se le añade una circuiteria para el reset manual en el caso de que la velocidad de crecimiento de la señal de alimentación sea < 0,05 v/ms o si se requiere el acceso al reset de forma manual.

- **RA0 − RA3**

Son los pines de entrada −salida del 0 al 3 del puerto A.

- **RA4**

Pin 4 de entrada−salida del puerto A común con la entrada de reloj externo del temporizador To.

- **RB0/INT**

Entrada−salida o del puerto B común con la entrada de interrupciones externas.

- **RB1 − RB7**

Entada−salida de la uno a la siete del puerto B.

2.7.1 − RB4 − RB7

Soportan además la función de interrupción por cambio de estado.

Figura 1.3 *Patillaje del encapsulado 16C84*

- **ARQUITECTURA INTERNA**

La memoria de programa esta organizada en palabras de 14 bits, el tamaño de la pila es de 8 niveles. Los registros están organizados siempre del mismo modo, estando en las direcciones bajas los registros especiales y de recursos y en las altas los de propósito general. El PC contiene 13 bits por lo que no se pueden direccionar 8 Kbytes de memoria, como no hay mas que 1 Kbyte, el hecho de direccionar mas allá de este valor hace que se vuelva al principio, como si los bits de mayor peso se ignoraran.

- **LOS REGISTROS INTERNOS**

Este gráfico representa la cartografía de la memoria de estos registros, obsérvese la utilización de dos paginas de memoria, la primera o página cero, contiene los registros fundamentales de 00 a 1F.

La segunda o página una, contiene registros menos importantes, o asociados a registros de la página cero

00	Direccionamiento indirecto	Direccionamiento indirecto	80
01	RTCC	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	SFR	FSR	84
05	PORT A	TRIS A	85
06	PORT B	TRIS B	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATCH	PCLATCH	8A
0B	INTCON	INTCON	8B
OC 2F	REGISTRO DE PROPOSITO GENERAL	IDEM PAGINA CERO	8C AF
30 7F			B0 FF

PÁGINA 0 PÁGINA 1

Tabla 1.1 *Organización de los registros internos.*

Así el registro TRISA asociado al PORT A se encuentra en la misma dirección dentro de la página.

Determinados registros figuran en las dos páginas, por lo que son accesibles de la misma forma y sin restricción, para facilitar de este modo la programación del circuito.

- **REGISTRO INDF (Dirección 00) O registro de indirección**

No tiene existencia física, por lo que no se podrá acceder a él. En realidad este registro sirve únicamente para especificar la utilización del direccionamiento indirecto, por ejemplo de la forma siguiente:

ADDWF INDF.W Significa que se va a añadir, al contenido del registro W, el contenido del registro apuntado por el registro FSR de dirección 04. Se realiza entonces el Direccionamiento indirecto con respecto al contenido de FSR. Sirviéndose de INDF únicamente como modo de notación.

4.2− REGISTRO PCL (Dirección 02)

Es el PC o contador de programa, o mejor dicho los 8 bits de menor peso del PC, puesto que el PC debe tener un tamaño de 13 bits, sus bits de mayor peso se llevan al registro PCLATCH, situado en las posiciones 0A y 8A.

Si el PC es destino de una instrucción, el contenido de PCLATCH se tiene en cuenta automáticamente.

Para las instrucciones GOTO y CALL, tiene lugar la misma operación, teniendo en cuenta el hecho de que el PC está codificado con 11 bits en la propia instrucción.

Los ocho registros de pila no aparecen en la gráfica anterior por no estar situados en el mismo espacio de memoria que los demás. Son registros de trece bits capaces de contener íntegramente al PC. Su utilización es automática, ya que el PC se introduce en la pila durante la ejecución de una instrucción CALL o de una interrupción, y se extrae de la pila durante la ejecución del retorno correspondiente. Estos registros de pila deben considerarse como un buffer de memoria circular, lo que significa que, si se introduce más de 8 valores del PC, el noveno valor tomará la posición del primero, y así sucesivamente.

Obsérvese que ningún bit de registro indica que la pila está llena, por lo que debe tener cuidado de que no se desborde, excepto si quiere realizar operaciones de retorno especiales.

Figura 1.4 *Principio de determinación de los bits*

de mayor peso del PC por PCLATCH

- **REGISTRO STATUS (Dirección 03) o Registro de Estado**

Este registro contiene un cierto número de bits de estado de la unidad central, pero también los bits de selección de páginas, que ahora se denominan RP1 y RP0. Se pueden leer o escribir como cualquier otro registro, dando por supuesto que determinados bits de estado no se modificarán después de un intento de escritura. Cada bit de este registro tiene un significado particular que es el que sigue.

  - **BIT 0** o bit de carry (C). Bit de acarreo para las operaciones de suma y resta. Se pone a uno mediante las instrusiones ADDWF y SUBWF, si se genera un acarreo en el bit de mayor peso.

Este bit también lo utilizan las instrucciones de rotación.

  - **BIT 1** o bit DC (Digit Carry). Este es un bit de acarreo de dígito, para por ejemplo la aritmética en BCD. Se pone a uno con las instrucciones ADDWF y SUBWF, si se genera un acarreo del bit 3 al grupo de cuatro bits superior.

- **BIT 2** o bit Z (zero). Este bit se pone a uno si el resultado de la operación aritmética o lógica ejecutada es nulo.
- **BIT 3** o bit PD (Power Down). Este bit se pone a uno durante la conexión a la alimentación del circuito, o durante la ejecución de una instrucción CLRWDT relativa al temporizador watchdog. Se pone a cero mediante una instrucción SLEEP.
- **BIT 4** o bit TO (Time Out). Este bit se pone a uno durante una conexión a la alimentación o durante la ejecución de una instrucción CLRWDT o SLEEP. Se pone a cero cuando el temporizador Watchdog se desborda.
- **BIT 5 Y 6** o bits RP0 y RP1. Estos bits sirven para seleccionar las páginas de memoria de programa. La tabla 1.5 precisa su modo de utilización, que es perfectamente lógico sabiendo que cada página contiene 128 bytes.
- **BIT 7** o bit IRP. Este bit está previsto para un futuro direccionamiento de paginado indirecto, pero no se utiliza en el 16C84. Tan solo se usa para compatibilidad con las futuras versiones, por lo que se debe poner a cero.

Tabla 1.2 *Programación de los bits de*

selección de página del STATUS

4.4− REGISTRO FSR (Dirección 04) o registro de selección de registro.

El contenido del SFR se utiliza para el direccionamiento indirecto. Este registro contiene 8 bits, tamaño suficiente para las versiones actuales del PIC 16CXX; no obstante, es preciso saber que en direccionamiento indirecto, puede construirse una dirección de 9 bits utilizando el contenido de este registro y el bit IRP del registro de estado. Esta función no se utiliza en el 16C84, sino que está prevista para extensiones futuras, por ello el que se aconseje el no uso del bit IRP.

4.5− REGISTRO PCLATCH (Dirección 0A y 8A)

Se cita este registro sólo como recordatorio, ya que lo vimos en el PCL.

4.6− REGISTRO INTCON (Dirección 0B y 8B)

Este registro sirve para el control global de las interrupciones y para indicar la procedencia de algunas de ellas, gracias a los bits de estado. Se dispone de cuatro potenciales recursos de interrupción.

- Una fuente externa a través del pin RB0/INT.
- El desbordamiento del temporizador 0.
- Un cambio de estado en los pines RB4 a RB7.
- Programación de la EEPROM de datos.

Cada bit del registro INTCON tiene un significado concreto que es el que sigue:

- **BIT 0** o bit RBIF (RB Interrupt Flag). Si se pone a 1, este bit indica un cambio de estado en una de las líneas de RB4 a RB7 del puerto B.
- **BIT 1** o bit INTF (Interrupt Flag). Si se pone a 1, este bit indica una interrupción provocada por la línea RB0/INT del puerto B.
- **BIT 2** o bit T0IF (Timer 0 Interrupt Flag). Si se pone a 1, este bit indica un desbordamiento del temporizador 0.
- **BIT 3** o bit RBIE (RB Interrupt Enable). Si se pone a uno este bit, autoriza las interrupciones provocadas por un cambio de estado de las líneas RB4 a RB7 del puerto B.
- **BIT 4** o bit INTE (Interrupt Enable). Si se pone a 1, este bit autoriza las interrupciones provocadas

por la línea RB0/INT del puerto B.

- **BIT 5** o bit T0IE (Timer 0 Interrupt Enable). Si está a uno, este bit autoriza las interrupciones debidas al desbordamiento del temporizador 0.
- **BIT 6** o bit EEIE (EEPROM Interrupt Enable). Si está a uno, este bit autoriza las interrupciones que proceden de la memoria EEPROM de datos.
- **BIT 7** o bit GIE (Global Interrupt Enable). Si se pone a uno, este bit autoriza todas las interrupciones que estén enmascaradas mediante sus bits individuales de activación, a cero las inhibe.

Cada activador individual debe ponerse a cero por software.

Solamente hay un vector indicador de interrupción (dirección 0004), por lo que se debe comprobar estos bits en el programa de interrupción para saber cual es la fuente de la misma.

Cuando llega una interrupción, el PIC pone el bit GIE a cero, de forma que no se perturbe el tratamiento de la interrupción en curso, debido a otras interrupciones eventuales. Este bit se pone automáticamente a uno al terminar el programa de interrupción, con la ejecución de la instrucción RETFIE.

Los indicadores de interrupciones correspondientes permanecen funcionales incluso cuando no se han autorizado.

- **− PUERTOS PARALELO**

Se dispone de dos puertos paralelo A y B. Las líneas de estos puertos se pueden programar individualmente como entradas o como salidas, y se utilizan casi de la misma forma.

Teniendo en cuenta que el encapsulado se propone con 18 pines, determinadas líneas de estos puertos se comparten con otros recursos internos.

4.7.1− PUERTO A

Dispone de un ancho de 5 bits. Las líneas RA0 a RA3 respetan el esquema de la figura.

A la salida está provista de un buffer CMOS, entrada y salida pasan por un latch.

La línea RA4 adopta una estructura diferente, su salida es de tipo drenador abierto, y la entrada está provista de un Trigger Schmitt. Es común con la entrada externa del temporizador 0.

El sentido de trabajo de todas las líneas de este puerto se controla mediante el registro TRISA, en el que un bit a cero activa la línea correspondiente como salida, y un bit a 1 como entrada, Evidentemente, después de un reset, todos los bits del registro TRISA quedan a uno.

- **PUERTO B**

El puerto B es un puerto bidireccional de 8 bits completo, en el que sólo una línea se comparte con otro recurso interno. Las líneas RB0 a RB3 adoptan la estructura interna indicada en la figura mientras que las líneas RB4 a RB7 la de la figura . La razón de ser de esta diferencia radica en el hecho de que es posible programar la generación de una interrupción durante un cambio de estado de una cualquiera de las líneas RB4 a RB7.

Todas las líneas del puerto B disponen de una resistencia de pull−up de alto valor, conectada a la alimentación. Esta resistencia se puede activar o no gracias al bit RBPU del registro OPTION. Dicha activación afecta a todas las líneas del puerto B y se desactiva tras un reset al igual que las líneas configuradas

como salida.

El sentido de trabajo de cada una de las líneas de este puerto es controlado por el registro TRISB, en el que un bit a cero hace que la línea correspondiente se active como salida, y un bit a 1 como entrada, tras un reset todos los bits se ponen a 1.

- **− TEMPORIZADOR**

El PIC 16C84 no contiene mas que un temporizador analógico. Genera una interrupción cada vez que el temporizador se desborda de FF a 00. El modo de funcionamiento y programación se realiza a través del registro OPTION.

Puede recibir su señal directamente, o a través de un predivisor, siendo hecha la selección mediante el bit PSA. Si se selecciona el predivisor, su tasa de división se programa a través de tres bits: PS0, PS1, PS2 .

En todos los casos, la señal utilizada puede proceder de una cualquiera de las dos fuentes posibles, dependiendo del estado del bit de selección RTS. O procede del oscilador de reloj después de la división por cuatro, por lo que está a la frecuencia del reloj de instrucción, o del pin ToCKI. La puerta OR EXCLUSIVO, controlada por el bit RTE, permite seleccionar si el incremento del registro TMR0 se producirá sobre el flanco de subida o bajada.

Los bits RTE, RTS, PSA y PS0 a PS2 que se emplean para configurar este temporizador están contenidos en el registro OPTION. Cada bit de este registro tiene un significado especial que es el que sigue:

REG. OPTION

- **BITs 0, 1, 2.** Bits PS0, PS1, PS2. Estos bits sirven para definir la tasa de división del predivisor , como muestra la tabla, esta tasa difiere dependiendo de que el predivisor este asignado al temporizador 0 o al watchdog.

Tabla 1.3 *Programación de los bits de*

selección de tasa del predivisor en el registro OPTION

- **BIT 3** o bit PSA (Prescaler Assignment). Si este bit está a cero, el predivisor está asignado al temporizador 0. Si está a uno estará asignado al watchdog.
- **BIT 4** o bit RTE (Timer 0 signal Edge). Si este bit está a cero, el contenido del registro TMR0 se incrementará en un flanco de subida de la señal aplicada al pin RA4/ToCK1; si está a uno, se incrementará con el flanco de bajada.
- **BIT 5** o bit RTS (Timer 0 signal source). Si este bit está a cero, el temporizador utilizará el reloj interno. Si está a uno, utilizará la señal aplicada al pin RA4/T0CK1.
- **BIT 6** o bit INTEDG (Interrupt edge). Este bit define el sentido del flanco que provocará la interrupción a través del pin externo INT. Un uno activa el flanco de subida y un cero un flanco de bajada.
- **BIT 7** o bit RBPU (RB Pull Up enable). Si está a cero, este bit activa las resistencias de pull up a nivel alto, previstas en la entrada del puerto B.

(Estos últimos dos bits no se usan en el temporizador 0)

El registro TMR0 se incrementa en una unidad con cada impulso de reloj seleccionado mediante el registro OPTION. Cada vez que llega al valor FF, vuelve a 00 generando una interrupción, si se ha autorizado, y continua su ciclo indefinidamente.

El registro TMR0 se puede leer o escribir directamente con cualquier instrucción, con el fin de conocer su posición actual, o para inicializarlo en un estado determinado. Es importante saber que después de cualquier escritura en este registro, es necesario un retardo de dos ciclos de instrucción para que se retome la incrementación. Este retraso es independiente de la fuente de reloj usada. Las instrucciones concernidas son MOVF TMRO o CLRF TMR0.

Para comprobar el paso por cero sin inferir en el desarrollo regular del recuento, es aconsejable utilizar, por ejemplo, una instrucción MOVF TMR0,W, que no hace mas que una lectura.

El reloj interno deja de funcionar en el modo SLEEP, por lo que no se puede contar con sus interrupciones en este modo ni por consiguiente, que salga de este modo de funcionamiento por medio de dicha interrupción.

Obsérvese que todas las instrucciones que escriben en el TMR0 ponen a cero al predivisor, cuando éste está asignado al temporizador.

- **− TEMPORIZADOR WATCHDOG**

Está provisto de su propio oscilador interno autónomo con célula RC interna, por lo que no necesita ningún componente externo y continua funcionando incluso cuando el reloj del PIC se para durante, por ejemplo, la instrucción SLEEP.

Este temporizador cuenta de forma permanente, y cuando se desborda, es decir cuando llega a FF, hace que se genere un reset del microcontrolador. Si el PIC está en modo sleep cuando se produce este desbordamiento, tiene por efecto que se salga de este modo.

Si no se desea utilizar el watchdog, es posible desactivarlo escribiendo un 0 en un bit específico del circuito durante su programación. Este bit no es direccionable, pero todos los programadores saben acceder a él.

El tiempo típico de desbordamiento de este temporizador es de 18 ms, pero puede variar algo con la tensión de alimentación y la temperatura. Si no es muy largo, el predivisor del que hemos hablado puede asignarse al watchdog gracias al bit PSA del registro OPTION. En estas condiciones, las tasas de predivisión definidas por este registro varían , pero teniendo en cuenta el máximo permitido de 128 ms, es posible obtener tiempos de desbordamiento de hasta 2.5 s.

Las instrucciones CLRWDT y SLEEP ponen a cero al watchdog y a uno el bit T0 del registro de estado. Esto evita que se generen resets indeseados, como por ejemplo cuando se está en modo sleep. Estas instrucciones también ponen a cero el predivisor, si este está asignado al watchdog.

Figura *Temporizador Watchdog.*

- **− MEMORIA EEPROM DE DATOS**

El PIC 16C84 contiene una EEPROM de datos de 16 bytes. Esta memoria no forma parte del espacio direccionable y solo es accesible para lectura y escritura a través de dos registros: EEDATA en la dirección 08 para los datos, y un registro EEADR en la dirección 09 para las direcciones. Dos registros de control, EECON1 en la dirección 88 y EECON2 en la dirección 89, permiten definir el modo de funcionamiento de esta memoria.

Esta memoria no emplea ningún recurso de alimentación externo, y funciona en todo el rango de alimentación autorizado. Su duración típica de programación es de 10 ms. Se controla mediante un temporizador interno, el cual le descarga a usted de realizar cualquier medida de tiempo por software.

Si se desea escribir en una posición de memoria ya ocupada, automáticamente se borra el contenido que había y se introduce el nuevo dato, por lo que no hay comando de borrado.

El principio de utilización de esta EEPROM en modo lectura es muy simple, y sigue los siguientes pasos:

- Escritura de la dirección que hay que leer en el registro EEADR
- Puesta a uno del bit RD del registro EECON1
- Lectura del dato así direccionado en el registro EEDATA

Teniendo en cuenta la velocidad de funcionamiento de esta memoria, el dato está disponible en EEDATA después de que el bit RD se pone a uno, por lo que es posible leerlo mediante la instrucción que sigue inmediatamente a la que pone a este bit a uno.

La escritura en esta memoria, que consiste en realidad en una programación, ya que estamos trabajando con una EEPROM, es algo mas complicada por evidentes razones de seguridad. Este proceso se debe desarrollar de la forma siguiente:

- Escritura de la dirección en la que desea escribir en el registro EEADR.
- Escritura del dato en el registro EEDATA.

Ejecución de un programa tipo:

MOVLW 55

MOVWF EECON2

MOVLW AA

MOVWF EECON2

BSF EECON1,WR

Esta última instrucción inicia el proceso de escritura. Cuando termina, el bit EEIF está a uno y si la última instrucción EEPROM ha sido activada por medio del bit EEIE del registro INTCON, se genera dicha interrupción. El bit EEIF debe ponerse a cero por software.

El programa consiste en escribir 55 seguido de AA en el registro EECON2 como medida de seguridad , con el fin de evitar que un programa que tenga un despiste pueda programar la EEPROM por accidente, manipulando simplemente los bits del EECON1.

El registro de control EECON1 contiene un determinado número de bits de control, cuyo significado es el que sigue:

- **BIT 0** o bit RD (Read Data). Este bit debe ponerse a uno para leer un dato. El circuito lo pone automáticamente a cero.
- **BIT 1** o bit WR (Write data). Este bit debe ponerse a uno para escribir un dato. El circuito lo pone automáticamente a cero.
- **BIT 2** o bit WREN (Write Enable). Si este bit está a cero, impide cualquier escritura en la EEPROM. Debe ponerse a uno para autorizar la escritura.
- **BIT 3** o bit WRERR (Write Error). Este bit se pone a uno si se produce un error de escritura después de una parada prematura (reset o watchdog). En estas condiciones, los contenidos de EEDATA y EEADR no varían, para permitir una repetición correcta de la operación.

- **BIT 4** o bit EEIF (EEPROM Interupt flag). Este bit se pone a uno al terminar una escritura y debe ponerse a cero por software

El registro EECON2 no tiene existencia física, por lo que no se puede leer, tan solo sirve para el proceso de protección en escritura detallado anteriormente.

- **− RESET E INTERRUPCIONES**
- **− FUENTES DE RESET**

Un reset puede ser provocado por diversas fuentes:

- Una conexión a la alimentación del circuito.
- Una acción sobre el pin MLCR mientras que el circuito está en modo normal.
- Una acción sobre el pin MLCR mientras que el circuito está en modo SLEEP.
- Un desbordamiento del watchdog, mientras el circuito está en modo normal.
- Un desbordamiento del watchdog, mientras el circuito está en modo SLEEP.

El comportamiento del circuito y el estado de los registros afectados por un reset son diferentes dependiendo de la situación que se produzca. Es posible distinguir por software el origen del reset. Para ello, basta con leer los bits TO y PD del registro de estado como se muestra el la tabla adjunta:

TO	**PD**	**ORIGEN DEL RESET**
0	0	Watchdog en modo SLEEP
0	1	Watchdog en funcionamiento normal
−−	0	MLRC en modo SLEEP
1	1	Alimentación
−−	−−	MLRC en funcionamiento normal

Tabla 1.4 Determinación del origen de un reset por medio de la lectura de los bits T0 y PD.

En todos los casos, al salir del modo SLEEP mediante el watchdog o una interrupción, el contador de programa PC se pone a 000. Por tanto, la primera instrucción ejecutable de su programa debe encontrarse en esta dirección.

Cuando se sale del modo SLEEP mediante el temporizador watchdog, el PC se incrementa en una unidad para pasar a la instrucción que sigue a la instrucción SLEEP, igual que cuando se sale de este modo mediante una interrupción (si el bit GIE del registro INTCON está a uno).

(u) = no cambia (x) = desconocido (−) = no existe (?) = depende de otras condiciones

Tabla 1.5 Estado de los registros después de los diferentes tipos de reset posibles.

Las posibles fuentes de reset afectan de forma diversa a los contenidos de los diferentes registros de control, de estado o de datos. En la descripción de cada registro hemos indicado el estado de los bits después de un reset de alimentación. En la tabla anterior encontrará estas mismas indicaciones, pero más completas.

Es aconsejable examinar esta tabla después de un reset antes de cualquier utilización, excepto en el caso de que reinicialice cada registro.

- **− FUENTES DE INTERRUPCIÓN**

Se dispone de cuatro fuentes de interrupción distintas, que se autorizan o no mediante la puesta a uno de los correspondientes bits del registro INTCON (cada una de las cuales dispone de su propio flag independientemente de que hayan sido autorizadas o no).

La siguiente figura muestra una representación electrónica de la lógica de interrupciones. Se ve claramente que cada bit de indicación de interrupción, se tiene en cuenta si la correspondiente puerta AND se abre mediante el bit de autorización, XXXE, que corresponde. También se pueden autorizar todas en bloque , o no , gracias al bit GIE del registro INTCON.

Ver como se puede hacer salir al PIC del modo SLEEP o generar una verdadera interrupción si está en modo normal.

Figura *Esquema del principio de la lógica de interrupción.*

Existe un único vector de interrupción en la dirección 004. Sea cual sea la interrupción, el PC se carga mediante 004 y a continuación, debe comprobar los diferentes indicadores para saber cuál es el elemento que interrumpe.

Durante esta operación, el bit GIE está a cero, para impedir que se tenga en cuenta cualquier nueva interrupción. Después de la ejecución de una instrucción de retorno RETFIE, se pone automáticamente a uno. No está previsto ningún mecanismo de puesta a cero de los bits de indicación de interrupción, por lo que es su programa de tratamiento de interrupciones el que debe encargarse de poner estos bits a cero. Si no es así, no podrá salir de ellas.

Durante la interrupción, el único registro que se salvaguarda en la pila es el PC. Si se desea preservar el contenido de otro registro, tendrá que hacerlo de forma manual.

6. − DIRECCIONAMIENTOS

Los modos de Direccionamientos son muy pocos y como veremos a continuación, muy a menudo son parte integrante de la propia instrucción.

- **INMEDIATO**

El dato manipulado por la instrucción se codifica con la propia instrucción. En este caso, el dato en cuestión se denomina *literal*, nombre que mantendremos a lo largo de todo el capítulo, con el fin de ser coherentes con las denominaciones adoptadas por *Microchip* en todos sus documentos.

MOVLW k Coloca el literal *k*, que es un valor cualquiera codificado con 8 bits, en el registro . de trabajo *w*.

- **DIRECTO**

Es el modo más usado, ya que como hemos visto en los capítulos anteriores, la memoria RAM está dividida en registros específicos y en un conjunto de registros de propósito general. Este modo consiste en codificar el nombre del o de los registros en cuestión directamente en la instrucción.

MOVWF *f* Desplaza el contenido del registro *w* al registro *f*

El registro *f* se referencia mediante su número codificado con 5 o 7 bits, siendo este número en realidad la dirección del byte de la RAM correspondiente, después de la utilización del mecanismo de paginación de la memoria, si es necesario.

- **BIT A BIT**

Manipulación de un bit individual en cualquier registro. Este modo de direccionamiento no se utiliza nunca solo, sino que siempre va emparejado con el modo de direccionamiento directo.

BCF *f* , *b* Pone a cero el bit número *b* del registro *f*

El registro *f* se codifica como ya indicamos anteriormente, y *b* es el número del bit de este registro, codificado sólo con tres bits, ya que no puede variar más que de 0 a 7.

- **INDIRECTO**

Es el modo más potente y de el ya hemos hablado anteriormente en los registros INDF y FSR.

Recordemos rápidamente que el registro FSR es el de selección de registro, en el que se introduce el número del registro direccionado. Lo único extraño del PIC es el modo de notación correspondiente que utiliza el registro INDF, como por ejemplo en:

MOVWF *fo* Desplaza el contenido de *w* al registro apuntado por el registro FSR.

La notación *fo* únicamente sirve para indicar el direccionamiento indirecto y por tanto, la utilización del registro FSR como puntero. Incluso cuando esto le parezca pobre, sobre todo si está acostumbrado a otros microcontroladores, éstos son todos los modos de direccionamientos del PIC 16C84. Sin embargo, teniendo en cuenta la eficacia de la estructura de los circuitos y su juego de instrucciones, estos modos le permitirán realizar un buen trabajo.

Antes de concluir este apartado, observe que nos hemos olvidado del tradicional modo RELATIVO, el cual permite en las arquitecturas clásicas los saltos para por ejemplo, los bucles condicionales. Cuando lea la descripción de las instrucciones de bucle, comprenderá por qué no existe este modo de direccionamiento en este circuito.

- **JUEGO DE INSTRUCCIONES**

Con el fin de proporcionarle un texto tan practico como sea posible, para cada instrucción que se presenta se indica:

- Su sintaxis, indicando la notación de forma precisa.
- Su codificación con 14 bits

Este último punto, no le interesa en lo que concierne a la programación propiamente dicha, ya que estos bits los genera el ensamblador, por supuesto: pero le permitirán comprender cómo se llegan a codificar las instrucciones y los modos de direccionamiento. Además, también le permitirá comprender los límites de los tamaños impuestos a determinados datos, límites que se deben exclusivamente al número de bits que se pueden utilizar.

- Número de palabras y el número de ciclos de máquina utilizados. Puede así comprobar que el principio fundamental de la arquitectura RISC se respeta, con una instrucción por ciclo, salvo raras ocasiones.
- La representación esquemática de la operación realizada por la instrucción.
- Los bits del registro de estado a los que afecta esta operación.
- Comentarios extraídos de la documentación del fabricante o de nuestra experiencia personal cuando sus explicaciones son algo confusas.

La notación adoptada para los datos y direcciones manipuladas por las instrucciones es muy sencilla, y es como sigue:

- *f* representa un número de registro, codificado con siete bits
- *b* representa en número de bit, codificado con tres bits. El bit 0 es siempre el bit de menor peso.
- *k* representa un dato, el literal mencionado antes. EL número de bits que se emplea para su codificación es variable, dependiendo de la instrucción.

Un determinado número de instrucciones (ADDWF, ANDWF, etc.) utilizan una notación especial que es:

ADDWF *f,d*

f es el número del registro, como ya sabemos, y *d* puede tomar el valor 0 o 1, y cambia el comportamiento de la instrucción. Si d está a cero, el resultado se introduce en el registro W, mientras que si *d* está a uno, el resultado se introduce en el registro *f*. Las instrucciones de este tipo son, en realidad, dobles ya que, dependiendo del valor de *d*, realizan:

W operación *f* −−−−− W si *d* = 0

W operación *f* − − − − − *f* si *d* = 1

El segundo punto que nos gustaría resaltar concierne a las instrucciones de bucle condicional. Por ejemplo:

BTFSC *f,b*

Ya sabemos que esta instrucción comprueba el bit *b* del registro *f*. El significado de esta instrucción es Bit Test Skip if Clear, por lo que hace un salto (skip) si el bit está a cero (clear). Veamos como ocurre esto:

- Si el bit está a uno, es decir, si la condición que se comprueba no se cumple, el programa continua su desarrollo normal de forma secuencial

- Si el bit está a cero, es decir, si la condición que se comprueba se cumple, el programa salta a la instrucción que sigue a BTFSC en el programa.

Si desea, como generalmente es el caso, saltar a alguna otra parte del programa dependiendo del resultado de la prueba, generalmente adoptará la siguiente sintaxis:

GOTO *k* o **CALL** *k*

Procediendo de esta forma, si la condición es falsa, debe ejecutar la parte de programa llamada por GOTO o CALL, mientras que si la condición es verdadera, el programa continua con la instrucción que sigue a GOTO o a CALL ya que, en ese caso, se salta justamente esta línea.

Obsérvese que este método, que puede parecer curioso a primera vista, en realidad es muy práctico para tratar los bits de los registros en función de condiciones externas. Así, si un interruptor está conectado a una línea de un puerto paralelo y en función de su estado, debe posicionar un relé controlado por otra línea del puerto paralelo, bastará con hacer:

BTFSC *f,b*

Donde *b* es el número de bit que corresponde al interruptor sobre el puerto en cuestión.

BCF *f,b*

Donde *b* es el número de bit que activa el relé del puerto en cuestión.

Este tratamiento especial de las instrucciones de prueba permiten comprender por qué, al principio del capítulo, le indicábamos que realmente el direccionamiento relativo no existía.

Las demás instrucciones son más clásicas, pero no obstante, nos detendremos un momento en las instrucciones CALL, que es una llamada a una subrutina ,y GOTO, que es un salto incondicional.

En estos dos casos anteriores, la dirección llamada parece que está codificada con un número insuficiente de bits. En realidad, está codificada con 14 bits, pero los bits que faltan proceden del PCLATCH.

Tenga también cuidado en las llamadas a subrutinas, de no anidar más llamadas que los niveles que contiene la pila, ocho en nuestro caso.

NEMÓNICO	OPERACIÓN	DESCRIPCIÓN	14 BITS MSb LSb		EXAD	RELOJ	BITS DE ESTADO
ADDWF f,d	w+f d	Suma W y f	00 0111	dfff ffff	07ff	1	C,DC,Z
ANDWF f,d	w&f d	AND entre W y f	00 0101	dfff ffff	05ff	1	Z
CLRF f	0 f	Pone a cero f	00 0001	1fff ffff	018f	1	Z
CLRW −	0 w	Pone a cero W	00 0001	0000 0011	0100	1	Z
COMF f,d	^f d	Complementa f	00 0101	dfff ffff	09ff	1	Z
DECF f,d	f−1 d	Decrementa f	00 0011	dfff ffff	03ff	1	Z
DECFSZ f,d	f−1 d skip if zero	Dec f, salta si es 0	00 1011	dfff ffff	0Bff	1, (2)	NONE
INCF f,d	f+1 d	Incrementa f	00 1010	dfff ffff	0Aff	1	Z
INCFSZ f,d	f+1 d skip if zero	Inc f, salta si es 0	00 1111	dfff ffff	0Fff	1, (2)	NONE
IORWF f,d	w v f d	OR inclusivo W con f	00 0100	dfff ffff	04ff	1	Z
MOVF f,d	f d	Mueve f. mira si es 0	00 1000	dfff ffff	08ff	1	Z
MOVWF f,d	w d	Mueve W a f	00 0000	1fff ffff	008f	1	NONE
NOP −	−−−	No operación	00 0000	0xx0 0000	0000	1	NONE
RLF f,d		Rota izq f con carry	00 1101	dfff ffff	0Dff	1	C
RRF f,d		Rota drch f con carry	00 1100	dfff ffff	0Cff	1	C
SUBWF f,d		Resta W con f	00 0010	dfff ffff	02ff	1	C,DC,Z
SWAPF f,d		4LSB con 4MSB	00 1110	dfff ffff	0Eff	1	NONE
XORWF f,d	w + f d or exc	OR exc W con f	00 0110	dfff ffff	06ff	1	Z
Instrucciones en los registros bit por bit							
BCF f,d	o f(d)	Borra f	01 00bb	bfff ffff	1bff	1	NONE
BSF f,d	1 f(d)	Pone a 1 f	01 01bb	bfff ffff	1bff	1	NONE
BTFSC f,d		Salta si b(f)=0 1 instr	01 10bb	bfff ffff	1bff	1, (2)	NONE
BTFSS f,d		Salta si b(f)=1 1 instr	01 11bb	bfff ffff	1bff	1, (2)	NONE

Instrucciones con literales y de control							
ADDLW f,d	**k + w w**	**Añade literal k a W**	**11 111x**	**kkkk kkkk**	**3Ekk**	**1**	**C,DC,Z**
ANDLW f,d	**k & w w**	**AND literal con W**	**11 1001**	**kkkk kkkk**	**39kk**	**1**	**Z**
CALL f,d		**Llama a subrutina**	**10 0kkk**	**kkkk kkkk**	**2kkk**	**2**	
CLRWDT f,d		**Borra el watchdog**	**00 0000**	**0110 0100**	**0064**	**1**	**TO, PD**
GOTO f,d		**Salta a nueva direcc**	**10 1kkk**	**kkkk kkkk**	**2kkk**	**2**	**NONE**
IORLW f,d	**k v w w**	**OR incl literal con W**	**11 1000**	**kkkk kkkk**	**38kk**	**1**	**Z**
MOVLW f,d	**k w**	**Carga literal en W**	**11 00xx**	**kkkk kkkk**	**30kk**	**1**	**NONE**
RETFIE f,d		**Retorno interrupción**	**00 0000**	**0000 1001**	**0009**	**2**	**NONE**
RETLW f,d		**RETFIE + MOVLW**	**11 01xx**	**kkkk kkkk**	**34kk**	**2**	**NONE**
RETURN f,d	**TOS PC**	**Retorno subrutina**	**00 0000**	**0000 1000**	**0008**	**2**	**NONE**
SLEEP f,d	**0WDT stop oscila**	**Modo SLEEP**	**00 0000**	**0110 0011**	**0063**	**1**	**TO, PD**
SUBLW f,d	**k − w w**	**Resta W con literal**	**11 110x**	**kkkk kkkk**	**3Ckk**	**1**	**C,DC,Z**
XORLW f,d	**k + w w or excl**	**OR exc literal con W**	**11 1010**	**kkkk kkkk**	**3Akk**	**1**	**Z**
OPTION	**W OPTION regis**	**CargaW en reg *option***	**00 0000**	**0110 0010**	**0062**	**1**	**NONE**
TRIST f	**Tristate port f**	**Carga W en reg trist**	**00 0000**	**0110 0fff**	**006f**	**1**	**NONE**

ADDL*W*	***Añade el contenido de W al literal k, y almacena el resultado en W.***
ADDWF	**Añade el contenido de W al de f, y almacena el resultado en W si d = 0, y en f si d = 1.**
ANDLW	**Efectúa un AND lógico entre el contenido de W y el literal k, y almacena el resultado en W.**
ANDWF	**Efectúa un AND lógico entre el contenido de W y el contenido de f y coloca el resultado en W si: d = 0, y en f si d = 1.**
BCF	**Pone a cero el bit número b de f**
BSF	**Pone a uno el bit número b de f**
BTFSC como	**Si el bit número b de f es nulo, la instrucción que sigue a ésta se ignora y se tratará una NOP. En este caso, y solo en el, la instrucción precisará dos ciclos para ejecutarse**
BTFSS	**Si el bit número b de f está a uno, la instrucción que sigue a ésta se ignora y se tratará como una NOP. En este caso, y solo en el, la instrucción precisará dos ciclos para ejecutarse**
CALL	**Salvaguarda la dirección de vuelta en la pila y después llama a la subrutina situada en la** **dirección cargada en el PC. También hay que posicionar correctamente el registro PCLATCH antes de ejecutarla.**

CLRF	**Pone el contenido de f a cero y activa el bit Z.**
CLRW	**Pone el contenido del registro W a cero y activa el bit Z.**
CLRWDT	**Pone a cero el registro contador del temporizador Watchdog, así como el predivisor.**
COMF	**Hace el complemento de f bit a bit. El resultado se almacena de nuevo en: f si d = 1, y en W si d = 0 (en este caso, f no varía).**
DECF	**Decrementa el contenido de f en una unidad. El resultado se almacena de nuevo en: f si d = 1, y en W si d = 0 (en este caso, f no varía).**
DECFSZ	**Decrementa el contenido de f en una unidad. El resultado se almacena de nuevo en: f si d = 1, y en W si d = 0 (en este caso, f no varía). Si el resultado es nulo, se ignora la siguiente instrucción y en ese caso, ésta instrucción dura dos ciclos.**
GOTO	**Llama a la subrutina situada en la dirección cargada en el PC También hay que posicionar correctamente el registro PCLATCH antes de ejecutarla.**
INCF	**Incrementa el contenido de f en una unidad. El resultado se almacena de nuevo en: f si d = 1, y en W si d = 0 (en este caso, f no varía).**
INCFSZ	**Incrementa el contenido de f en una unidad. El resultado se almacena de nuevo en: f si d = 1, y en W si d = 0 (en este caso, f no varía). Si el resultado es nulo, se ignora la siguiente instrucción y en ese caso, ésta instrucción dura dos ciclos.**
IORLW	**Efectúa un OR lógico inclusivo entre el contenido de W y el literal k, y almacena el resultado en W.**
IORWF	**Efectúa un OR lógico inclusivo entre el contenido de W y el contenido de f, y almacena el resultado en: f si d = 1, y en W si d = 0**
MOVF	**Desplaza el contenido de f a f si d = 1 ó a W si d = 0 Este desplazamiento que parece inútil, permite comprobar el contenido de f con respecto a cero, ya que esta instrucción actúa sobre el bit Z.**
MOVLW	**Carga W con el literal k.**
MOVWF	**Carga f con el contenido de W.**
NOP	**Solamente consume tiempo de máquina, (un ciclo).**
OPTION	**Carga el registro OPTION con el contenido de W. Esta instrucción no debe usarse, tan solo existe por compatibilidad con circuitos superiores.**
RETFIE	**Carga el PC con el valor que se encuentra en la parte alta de la pila, asegurando así la vuelta de la instrucción. Pone a uno el bit GIE, para autorizar de nuevo que se tengan en cuenta las interrupciones. Dura dos ciclos.**
RETLW	**Carga W con el literal k, y después carga el PC con el valor que se encuentra en la parte superior de la pila, efectuando así un retorno de la subrutina. Dura 2 ciclos**
RETURN	**Carga el PC con el valor que se encuentra en la parte alta de la pila, efectuando así una vuelta de subrutina. Se trata de la instrucción RETLW simplificada. Dura 2 ciclos**
RLF	**Rotación de un bit a la izquierda del contenido de f, pasando por el bit de acarreo C. Si d = 1 el resultado se almacena en f, si d = 0 el resultado se almacena en W.**
RRF	**Rotación de un bit a la izquierda del contenido de f, pasando por el bit de acarreo C. Si d = 1 el resultado se almacena en f, si d = 0 el resultado se almacena en W.**
SLEEP	**Pone el circuito en modo sleep con parada del oscilador.**
SUBLW	**Sustrae el contenido de W del literal k, y almacena el resultado en W. La sustracción se realiza en complemento a dos.**
SUBWF	**Sustrae el contenido de W del contenido de f, y almacena el resultado en: W si d = 0 y en f si d = 1. La sustracción se realiza en complemento a dos.**
SWAPF	

	Intercambia los cuatro bits de mayor peso con los cuatro de menor peso de f, y almacena el resultado en f si d = 1, o en W si d = 0.
TRIS	**Carga el contenido de W en el registro tris del puerto f. Esta instrucción no debe usarse, tan solo existe por compatibilidad con circuitos superiores.**
XORLW	**Efectúa un OR lógico exclusivo entre el contenido de W y el literal k, y almacena el resultado en W.**
XORWF	**Efectúa un OR lógico exclusivo entre el contenido de W y el contenido de f, y almacena el resultado en f si d = 1 y en W si d = 0.**

- **− SISTEMA DE DESARROLLO**
- **ENSAMBLADOR Y COMPILADOR**

En primer lugar, un sistema de desarrollo está formado por un ensamblador y uno o varios compiladores adaptados al lenguaje de alto nivel que se desea utilizar para programar.

El ensamblador traduce las instrucciones que se han escrito usando los nemónicos del lenguaje máquina, a código binario ejecutable por el microcontrolador. La secuencia de nemónicos se llama listado o código fuente del programa, mientras que el código binario se llama objeto o ejecutable.

El compilador traduce las instrucciones que se han escrito en lenguaje de alto nivel, que constituyen también lo que se llama listado o código fuente, a código binario ejecutable por el microcontrolador que constituye el código objeto.

En un sistema de desarrollo bien concebido, es primordial que el compilador utilizado para el lenguaje de alto nivel tenga una interfaz perfecta con el ensamblador, de forma que se puedan insertar subrutinas en lenguaje máquina en el mismo seno del programa de alto nivel.

Estos dos programas deben ejecutarse sobre una máquina denominada *host*, ó plataforma de desarrollo. Esta máquina puede ser de diversas formas, la solución más generalizada, es un compatible PC el cual permite minimizar la inversión.

Una vez que el programa se ha escrito y ensamblado o compilado sobre la máquina *host*, se está en posesión de un binario ejecutable. Es casi indispensable probar este programa, haciéndole funcionar en condiciones tan próximas como sea posible a las de utilización real. Para hacer esto, existen varias posibles soluciones.

- **EMULADOR Y SIMULADOR**

Utilizar un emulador es la solución más costosa, es un montaje especial, que puede ser muy complejo y que se comporta exactamente como el microcontrolador al que reemplaza.

Como el emulador es una versión fragmentada del microcontrolador al que reemplaza, se tiene acceso a las distintas señales internas de éste y en particular, se puede saber por cuales direcciones pasa el programa, que ocurre en los diversos registros de los periféricos internos, etc. Del mismo modo, se puede introducir puntos de parada, para leer el estado de ciertas memorias o de ciertos registros, y ejecutar en tiempo real.

La segunda solución no siempre permite realizar todas las pruebas necesarias. Consiste en emplear un simulador, el cual es un programa escrito especialmente para el microcontrolador que se va a simular.

Por tanto, el simulador es un producto mucho más sencillo que el emulador, ya que no es más que un programa. Por tanto, su precio de coste es mucho más bajo.

El simulador realiza la ejecución del programa, aproximadamente, de 10 a 100 veces menos deprisa que lo haría el mismo programa directamente sobre el microcontrolador, por eso determinadas operaciones, en las que son necesarios tiempos muy precisos o críticos, no se puede probar mediante la simulación. No obstante, bien utilizado permite desarrollar aplicaciones interesantes mediante una inversión mínima.

- **DESARROLLO SOFTWARE CON MEDIOS REDUCIDOS**

Para determinado número de aplicaciones, y en el caso que nos ocupa, a veces es posible no hacer uso del emulador gracias a la comercialización por parte de *Microchip,* de kits de desarrollo que contienen un buen ensamblador y simulador.

- **− EJEMPLO**

Diseñar un programa que de una frecuencia de salida de 1 KHz por el pin B0 del puerto B, el cual estará ofreciendo un pulso positivo proporcional a una entrada exterior, y el resto a cero hasta completar la duración total de la frecuencia anteriormente mencionada.

El pulso positivo tendrá una duración máxima de un milisegundo y su relación con la entrada será lineal como se muestra en la figura.

El valor máximo de la entrada, será de 255 en digital, que corresponderá con una duración de un milisegundo del pulso positivo, y el mínimo de un valor 0, que corresponderá con un pulso positivo de duración nula.

La entrada mencionada se dará por el puerto A.

Cada vez que se desborde el temporizador decrementaré un registro cargado inicialmente a FF en digital, con lo que estaré contando 256 x 255 pulsos de reloj (65280 pulsos).

Tengo que dar una frecuencia de salida de 1 KHz, por lo tanto, como la entrada máxima de 255 debe corresponder con un pulso positivo de un milisegundo, tendré:

65280 −−−−−−−−− 1 KHz

1 −−−−−−−−− ? 0.015318Hz

Le asignaré una tasa de división de 256, por lo que la frecuencia del reloj deberá ser de 3.92156 Hz.

El registro OPTION, tras la puesta en marcha del circuito, están a uno todos sus bits, por lo que debo poner el bit 3 de este registro a cero para asignar dicha predivisión. El registro OPTION ocupa la posición 81 de la página uno dentro de los registros internos.

El PORT A inicialmente está configurado como entrada por tener todos los bits del registro TRISA a uno, y como quiero sacar los pulsos por el bit uno de dicho puerto, pondré a cero dicho bit del registro TRISA.

El registro TRISA ocupa la posición 85 de la página uno dentro de los registros internos.

INICIO:

BCF 81,03 ; pone a cero el bit 3 del registro OPTION con lo que asigno el

predivisor al temporizador. PS0,PS1,PS2 tienen el valor 1,1,1 en el

registro OPTION.

MOVLW FF ; carga el registro de trabajo (W) con el literal FF.

MOVWF 1C ; carga el literal FF en un registro de propósito general (1C).

MOVF f6,01 ; pone el registro f6 (PORT B) en el registro W, actuando esta

operación sobre el bit de cero en el caso de que en el PORT B

estuviese el valor cero.

BTFSC f3, 02 ; ignora la siguiente instrucción, tratándola como una operación NOP,

en el caso de que el bit 2 del registro f3 (STATUS) sea cero, es decir

si la ultima operación es distinta de cero.

GOTO INICIO ; salta a inicio solamente en el caso de que la entrada del puerto B sea

nula.

BCF 85,01 ; Pone a cero el bit 1 del registro TRISA, asignándolo así como salida.

BSF f5, 01 ; pone a uno el bit 1 del PORT A, o bit RA1, que será la salida del

sistema.

MOVWF 0C ; carga el registro 0C con el contenido del registro W que es el valor

leído en el PORT B.

CUENTA:

GOTO ENTRADA ; salta a la posición de memoria entrada

CUENTA1:

BCF 0B, 02 ; pone a cero el bit 2 del registro INTCON

(debe hacerse por software).

DEC 1C, 01 ; decrementa el registro 1C, el cual estaba inicialmente cargado con el

literal FF.

DECFSZ 0C, 01 ; decrementa 0C y guarda en 0C, el cual estaba inicialmente cargado

con el valor del PORT B y saltará la siguiente instrucción en el caso

de que el resultado sea cero.

GOTO CUENTA ; salta a la posición de memoria cuenta, solamente no leerá esta

instrucción en el caso de que el registro 0C llegue a cero.

BCF f5, 01 ; pone a cero el bit 1 del PORT A o bit RA1.

GOTO FINAL ; salta a la posición de memoria final.

ENTRADA:

BTFSC 0B, 02 ; saltará la siguiente instrucción, si el bit 2 del registro INTCON es cero,

es decir, en el caso de que no se halla producido overflow en el

temporizador.

RETURN ; retorna a la última llamada de subrutina producida. Carga el PC con el

valor que se encuentra en la parte superior de la pila. (*)

GOTO ENTRADA ; salta a la posición de memoria entrada.

GOTO CUENTA1 ; (*)

FINAL:

BTFSS 0B, 02 ; saltará la siguiente instrucción, si el bit 2 del registro INTCON es uno,

es decir, en el caso de que si se halla producido overflow en el

temporizador.

GOTO FINAL ; salta a la posición de memoria final, esta instrucción se leerá en el

caso de que aun no se halla producido overflow.

BCF 0B, 02 ; pone a cero el bit 2 del registro INTCON (flag de overflow)

DECFSZ 1C, 01 ; decrementa el registro 1C, el cual estaba inicialmente cargado con el

literal FF y saltará la siguiente instrucción si el resultado es cero.

GOTO FINAL ; salta a la posición de memoria final.

GOTO INICIO ; comienza de nuevo el programa.

(*) Los ocho registros de pila deben considerarse como un buffer de memoria circular, lo que significa que, si se introducen más de ocho valores del PC, el noveno valor tomará la posición del primero, y así sucesivamente.

La posición de memoria final se usa para completar el ciclo y que éste tenga la frecuencia deseada,

y solamente se accederá a ella una vez se complete el ciclo de salida en alto del PORTA.

La posición de memoria entrada se usa para detectar el overflow del timer, y tendrá ciclos con la posición cuenta para ir decrementando los registros 0C y 1C

33