Arquitectura de Computadores

Diferencias entre Microprocesadores y Microcontroladores

Conjunto de Instrucciones de los Microcontroladores

Subrutina de temporización para el Microcontrolador PIC

Juana Orozco Díaz

Andrés Felipe Echavarría

Fabián Pertuz

Luxardo Asís

Profesor:

Evert de los Ríos Trujillo

Universidad del Magdalena

Programa de Ingeniería de Sistemas

Santa Marta D.C.T.H

2013

**Diferencias entre Microprocesadores y Microcontroladores**

Hay varias diferencias entre un microcontrolador y un microprocesador, la primera y más importante es la funcionalidad.

Para que un microprocesador sea funcional requiere conectarse a una memoria y a dispositivos de entrada y salida (E/S). Para aclarar esto piensen en la tarjeta principal de su computadora personal, además de un buen microprocesador, requieren de memoria RAM y en la tarjeta principal (motherboard) tenemos un “súper” circuito integrado llamado “chipset” que entre otras cosas se encarga de proporcionar las funciones de entrada y salida, tales como el control de los puertos serial, paralelo, USB, IEEE-1394, Ethernet y de las ranuras ISA, PCI, AGP y más.

En cambio en un microcontrolador tenemos tanto las capacidades de procesamiento, almacenamiento de datos e instrucciones (memoria) y de comunicación con el entorno (E/S) dentro de un sólo chip, es decir que no requiere de otros componentes para funcionar. De hecho puede considerarse como una microcomputadora en un sólo chip.

Otra diferencia es la arquitectura (los bloques funcionales internos y la forma en que se conectan), normalmente la de los microprocesadores es la tradicional de Von Neumann, mientras que la de los microcontroladores es la Harvard y aunque ya no hay arquitecturas “puramente” CISC o RISC ya que actualmente comparten muchas características, la de los microcontroladores es RISC. No debemos de olvidar que la arquitectura Harvard se distingue por tener una memoria para los datos y otra para las instrucciones, cada una con propio bus y capacidad.

La configuración mínima básica de un Microprocesador está constituida por un Micro de 40 Pines, Una memoria RAM de 28 Pines, una memoria ROM de 28 Pines y un decodificador de direcciones de 18 Pines

El costo para un sistema basado en Microcontrolador es mucho menor, mientras que para el Microprocesador, es muy alto en la actualidad.

El tiempo de desarrollo de su proyecto electrónico es menor para los Microcontroladores.

En los microcontroladores tradicionales todas las operaciones se realizan sobre el acumulador. La salida del acumulador está conectada a una de las entradas de la Unidad Aritmética y Lógica (ALU), y por lo tanto este es siempre uno de los dos operandos de cualquier instrucción, las instrucciones de simple operando (borrar, incrementar, decrementar, complementar), actúan sobre el acumulador.

En los microcontroladores PIC, la salida de la ALU va al registro W y también a la memoria de datos, por lo tanto el resultado puede guardarse en cualquiera de los dos destinos.

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

**Conjunto de instrucciones**

El conjunto de instrucciones de los microprocesadores PIC 16C5X consiste en un pequeño repertorio de solo 33 instrucciones de 12 bits, que pueden ser agrupadas para su estudio en tres a cinco grupos. En este curso se ha optado por clasificarlas, desde el punto de vista del programador, en cinco categorías bien definidas de acuerdo con la función y el tipo de operandos involucrados. En primer lugar se agrupan las instrucciones que operan con bytes y que involucran algún registro de la memoria interna. En segundo lugar se analizaran las instrucciones que operan solo sobre el registro W y que permiten cargarle una constante implícita o incluida literalmente en la instrucción (literales). En tercer lugar se agrupan las instrucciones que operan sobre bits individuales de los registros de la memoria interna. En cuarto lugar se clasifican las instrucciones de control de flujo del programa, es decir las que permiten alterar la secuencia lineal de ejecución de las instrucciones. Por último se agrupan unas pocas instrucciones que llamaremos especiales, cuyas funciones o tipos de operandos son muy específicos y no encajan en ninguna de las clasificaciones anteriores.

**Instrucciones de Byte que operan con Registros**

Estas instrucciones pueden ser de simple o doble operando de origen. El primer operando de origen será siempre el registro seleccionado en la instrucción, el segundo, en caso de existir, será el registro W. El destino, es decir donde se guardara el resultado, será el registro seleccionado o el W, según se seleccione con un bit de la instrucción.

El formato genérico de estas instrucciones es el siguiente:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | **6** | **7** | **8** | **9** | **10** | **11** |
|  |  |  |  |  |  | **d** | **f** | **f** | **f** | **f** | **f** |

Los bits 0 a 4 (5 bits), denominados “f” permiten seleccionar uno de 32 registros de la memoria interna. El bit 5, denominado “d”, permite especificar el destino del resultado. Si d = 1 el resultado se guardara en el registro seleccionado. Si d = 0 el resultado se guardara en W. Los bits 6 a 11 identifican la instrucción específica a realizar.

Las instrucciones siguientes son las tres operaciones lógicas de doble operando:

**ANDWF f, d**; operación **AND** lógica, destino = **W ­ f**

**IORWF f, d**; operación **OR** lógica, destino = **W ­ f**

**XORWF f, d**; operación **XOR** lógica, destino = **W ­ f**

Los nombres mnemónicos de estas instrucciones provienen de: AND W con F, Inclusive OR W con F y XOR W con F. Las que siguen son las cuatro operaciones aritméticas y lógicas sencillas de simple operando:

**MOVF f, d**; movimiento de datos, destino = **f**

**COMF f, d**; complemento lógico, destino = **NOT f**

**INCF f, d; incremento** aritmético, destino = **f + 1**

**DECF f, d**; decremento aritmético, destino = **f – 1**

Los mnemónicos de estas instrucciones provienen de: MOVe File, COMplement File, INCrement File y DECrement File.

En las siete instrucciones anteriores el único bit afectado de la palabra de estado del procesador es el Z, que se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valor.

A continuación siguen las dos instrucciones de rotación de bits a través del CARRY:

**RLF f, d**; rotación a la izquierda, destino = **f ROT** ­

**RRF f, d**; rotación a la derecha, destino = **f ROT** ­

En estas operaciones (Rotate Left File y Rotate Right File) los bits son desplazados de cada posición a la siguiente, en sentido derecho o izquierdo. El desplazamiento es cerrado, formando un anillo, con el bit C (CARRY) de la palabra de estado.

En estas dos instrucciones, el único bit afectado de la palabra de estado del procesador es el bit C, que tomará el valor que tenía el bit 7 o el bit 0, según sea el sentido del desplazamiento.

Estas instrucciones son muy útiles para la manipulación de bits, y además para realizar operaciones aritméticas, ya que en numeración binaria, desplazar un número a la izquierda es equivalente a multiplicarlo por 2, y hacia la derecha, a dividirlo por 2.

La instrucción siguiente realiza el intercambio de posiciones entre los cuatro bits menos significativos y los cuatro más significativos (nibble bajo y nibble alto).

**SWAPF f, d**; intercambia nibbles, destino = **SWAP f**

Esta instrucción (SWAP File) no afecta ninguno de los bits de la palabra de estado del procesador.

Esta instrucción es muy útil para el manipuleo de números BCD empaquetados, en los que en un solo byte se guardan dos dígitos BCD (uno en cada nibble).

Las dos operaciones que siguen son la suma y la resta aritméticas:

**ADDWF f, d; suma** aritmética, destino = **f + W**

**SUBWF f, d; resta** aritmética, destino = **f - W**

Estas operaciones (ADD W a F y SUBstract W de F) afectan a los tres bits de estado C, DC y Z.

El bit Z se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valorLa suma se realiza en aritmética binaria pura sin signo. Si hay un acarreo del bit 7, es decir que el resultado es mayor que 255, el bit C (carry) resulta 1, en caso contrario resulta 0. Si hay un acarreo del bit 3, es decir que la suma de las dos mitades (nibbles) menos significativas (bits 0 a 3) resulta mayor que 15, se pone en 1 el bit DC (digit carry), en caso contrario se pone en 0.

Ejemplos:

1010 0010 1101 0000

+ 0100 1111 C DC Z + 0110 1111 C DC Z

1111 0001 0 1 0 0011 1111 1 0 0

La resta se realiza sumando, en binario puro sin signo, el registro f más el complemento a dos de W (el complemento a 1, o complemento lógico, más 1)

Ejemplos:

f ­ 0100 0100 0010 1000

W­­ - 0010 1000 C DC Z - 0100 0100 C DC Z

0001 1100 1 0 0 1110 0100 0 1 0

­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­

­ Equivalente a:

f ­ 0100 0100 0010 1000

cmp.2 W ­ + 1101 1000 C DC Z + 1011 1100 C DC Z

0001 1100 1 0 0 1110 0100 0 1 0

Los bits de estado C y DC toman el valor normal correspondiente a la suma de f con el complemento a 2 de W. De esta manera el significado para la operación de resta resulta invertido, es decir que C (carry) es 1 si no hubo desborde en la resta, o dicho de otra manera, si el contenido de W es menor que el de f. El bit DC se comporta de manera similar, es decir que DC es 1 si no hubo desborde en la mitad menos significativa, lo que equivale a decir que el nibble bajo del contenido de W es menor que el del registro f.

Las instrucciones que siguen son de simple operando, pero son casos especiales ya que el destino es siempre el registro seleccionado:

**CLRF f**; borrado de contenido, **f = 0**

**MOVWF f**; copia contenido W ­ f, **f = W**

La instrucción CLRF (CLeaR File) afecta solo al bit Z que resulta siempre 0.

La instrucción MOVWF (MOVe W a F) no afecta ningún bit de la palabra de estado.

**Instrucciones de Byte que operan sobre W y Literales**

Estas instrucciones se refieren todas al registro W, es decir que uno de los operandos de origen y el operando de destino son siempre el registro W. En las instrucciones de este grupo que tienen un segundo operando de origen, este es siempre una constante de programa literalmente incluida en la instrucción, llamada constante literal o simplemente literal.

El formato genérico de estas instrucciones es el siguiente:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|  |  |  |  | **k** | **k** | **k** | **k** | **k** | **k** | **k** | **k** |

Los bits 0 a 7 especifican la constante literal de 8 bits que se utilizara en la operación.

Las tres instrucciones que siguen son las operaciones lógicas tradicionales, similares a las que ya vimos anteriormente, pero realizadas entre una constante de programa y el registro W:

**IORLW k**; operación OR lógica, W = **W ­ k**

**ANDLW k**; operación AND lógica, W = **W ­ k**

**XORLW k;** operación XOR lógica, W = **W ­ k**

En estas tres instrucciones (Inclusive OR Literal W, AND Literal W y XOR Literal W) el único bit afectado de la palabra de estado del procesador es el Z, que se pone en 1 si el resultado de la operación es 00000000, y se pone en 0 si el resultado tiene cualquier otro valor.

La instrucción que sigue sirve para cargar una constante de programa en el registro W:

**MOVLW k**; carga constante en W, **W = K**

Esta (MOVe Literal W) instrucción no afecta ninguno de los bits de estado del procesador.

La instrucción que sigue (CLeaR W) no correspondería incluirla en este grupo, y pertenece en realidad al primero, el de las instrucciones que operan sobre registros, ya que se trata de un caso especial de la instrucción CLRF, con destino W, y f = 0.

La incluimos aquí porque como se le ha asignado un mnemónico particular referido específicamente al registro W, creemos que, desde el punto de vista del programador, es más útil verla dentro del grupo de instrucciones referidas a W.

**CLRW**; borra el contenido de W, **W = 0**

Al igual que en la instrucción CLRF, el único bit de estado afectado es el Z que resulta 1.

**Instrucciones de Bit**

El formato genérico de estas instrucciones es el siguiente:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|  |  |  |  | **b** | **b** | **b** | **f** | **f** | **f** | **f** | **f** |

Los bits 0 a 4 (5 bits), denominados “f”, permiten seleccionar uno de 32 registros de la memoria interna. Los bits 5 a 7, denominados “b”, permiten especificar el número de bit (0 a 7) sobre el que se operara. Estas instrucciones operan solamente sobre el bit especificado, el resto de los bits del registro no son alterados. Estas instrucciones no tienen especificación de destino, ya que el mismo es siempre el registro seleccionado.

**BCF f, b**; borra el bit b de f; bit **f (b) = 0**

**BSF f, b.**; coloca en uno el bit b de f; bit **f (b) = 1**

Estas instrucciones (Bit Clear File y Bit Set File) no afectan ningún bit de la palabra de estado del procesador.

**Instrucciones de Control**

**GOTO k;** salto a la posición k (9 bits) del programa

Esta es la típica instrucción de salto incondicional a cualquier posición de la memoria de programa (que en la mayoría de los microprocesadores convencionales se llama JUMP). La constante literal k es la dirección de destino del salto, es decir la nueva dirección de memoria de programa a partir de la cual comenzarán a leerse las instrucciones después de ejecutar la instrucción GOTO. Esta instrucción simplemente carga la constante k en el registro PC (contador de programa). La única complicación de esta instrucción es que la constante k es de solo 9 bits, mientras que el registro PC es de 11 bits, ya que en el 16C57 debe permitir direccionar una memoria de programa de 2 K. Los dos bits faltantes, bit 9 y 10 del PC, son tomados respectivamente de los bits de selección de página PA0 y PA1 de la palabra de estado. Este comportamiento particular hace que la memoria de programa aparezca como dividida en páginas de 512 posiciones como se verá más adelante. El programador debe tener en cuenta que antes de ejecutar una instrucción GOTO es posible que haya que programar los bits PA0 y PA1.

La que sigue es la instrucción de llamado a subrutina:

**CALL k**; salto a la subrutina en la posición k (8 bits)

Su comportamiento es muy similar al de la instrucción GOTO, salvo que además de saltar guarda en el stack la dirección de retorno de la subrutina (para la instrucción RETLW).

Esto lo hace simplemente guardando en el stack una copia del PC incrementado, antes de que el mismo sea cargado con la nueva dirección k. La única diferencia con la instrucción GOTO respecto de la forma en la que se realiza el salto, es que en la instrucción CALL la constante k tiene solo 8 bits en vez de 9. En este caso también se utilizan PA0 y PA1 para cargar los bits 9 y 10 del PC, pero además el bit 8 del PC es cargado siempre con 0. Esto hace que los saltos a subrutina solo puedan realizarse a posiciones que estén en las primeras mitades de las páginas mencionadas. El programador debe tener en cuenta este comportamiento y asegurarse de ubicar las posiciones de inicio de las subrutinas en las primeras mitades de las páginas. La instrucción que aparece a continuación es la de retorno de subrutina:

**RETLW k**; retorno de subrutina con constante k, W = k

Esta (RETurn con Literal in W) instrucción produce el retorno de subrutina con una constante literal k en el registro W. La operación que realiza consiste simplemente en sacar del stack un valor y cargarlo en el PC. Ese valor es el PC incrementado antes de realizar el salto, de la última instrucción CALL ejecutada, por lo tanto es la dirección de la instrucción siguiente a dicho CALL... Dado que el stack es de 11 bits, el valor cargado en el PC es una dirección completa, y por lo tanto se puede retornar a cualquier posición de la memoria de programa, sin importar como estén los bits de selección de página. Esta instrucción además carga siempre una constante literal en el registro W. Ya que esta es la única instrucción de retorno de subrutina de los PIC16C5X, no hay en estos microprocesadores forma de retornar de una subrutina sin alterar el registro W. Por otro lado, y con una metodología especial de programación, un conjunto de sucesivas instrucciones RETLW puede ser usado como una tabla de valores constantes incluida en el programa (Ej.: tablas BCD/7 seg., hexa/ASCII, etc.).

A continuación se presentan las dos únicas instrucciones de “salteo” (skip) condicional.

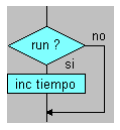
Estas instrucciones son los únicos medios para implementar bifurcaciones condicionales en un programa. Son muy generales y muy poderosas ya que permiten al programa tomar decisiones en función de cualquier bit de cualquier posición de la memoria interna de datos, y eso incluye a los registros de periféricos, los puertos de entrada/salida e incluso la palabra de estado del procesador. Estas dos instrucciones reemplazan y superan a todo el conjunto de instrucciones de salto condicional que poseen los microprocesadores sencillos convencionales (salto por cero, por no cero, por carry, etc.).

**BTFSC f, b**; salteo si bit = 0, **bit = f (0)** ­ saltea

**BTFSS f, b**; salteo si bit = 1**, bit = f (1)** ­ saltea

**BTFSC** (Bit Test File and Skip if Clear) saltea la próxima instrucción si el bit b del registro f es cero. La instrucción BTFSS (Bit Test File and Skip if Set) saltea si el bit es 1. Estas instrucciones pueden usarse para realizar o no una acción según sea el estado de un bit, o, en combinación con GOTO, para realizar una bifurcación condicional.

Ejemplo 1:



**btfsc** flags, run; sí ha arrancado el reloj

Incf tiempo; incremento contador de tiempo

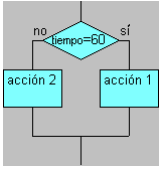
Ejemplo 2:

**movf** tiempo, **w**; testeo por tiempo = 60

**xorlw** 60

**btfss** STATUS, Z

**goto** acc\_2; salto si tiempo <> 60

 - - - - - - - - - - - - - ; acción 1

- - - - - - - - - - - - -

- - - - - - - - - - - - -

**goto** acc\_fin

acc\_2

- - - - - - - - - - - - - ; acción 2

- - - - - - - - - - - - -

- - - - - - - - - - - - -

acc\_fin;

Acá se unen los caminos

Las instrucciones que siguen son casos especiales de las de incremento y decremento vistas anteriormente. Estas instrucciones podrían categorizarse dentro del grupo de instrucciones orientadas a byte sobre registros (primer grupo), ya que efectivamente operan sobre los mismos, y el formato del código de la instrucción responde al de ese grupo, pero, a diferencia de las otras, pueden además alterar el flujo lineal del programa y por eso se les incluyó en este grupo.

**DECFSZ f, d**; decrementa y saltea sí 0, destino= f - 1, = 0 ­ saltea

**INCFSZ f, d;** incrementa y saltea sí 0, destino= f + 1, = 0 ­ saltea

Estas dos instrucciones (DECrement File and Skip if Zero, e INCrement File and Skip if Zero) se comportan de manera similar a DECF e INCF, salvo que no afectan a ningún bit de la palabra de estado. Una vez realizado el incremento o decremento, si el resultado es 00000000, el microprocesador salteara la próxima instrucción del programa. Estas instrucciones se utilizan generalmente en combinación con una instrucción de salto (GOTO), para el diseño de ciclos o lazos (loops) de instrucciones que deben repetirse una cantidad determinada de veces.

Ejemplo:

**clrf** 10; pongo cero en la posición 10 de la memoria interna

**loop**; lo que sigue se ejecutará 256 veces

.....................................

.....................................

.....................................

**incfsz** 10,1; incremento la posición 10 hasta que llegue a 0

**goto loop**; si no llego a cero voy a repetir la secuencia

; cuando llegue a cero salteo el goto

; y sigue la continuación del programa

**Instrucciones Especiales**

En este grupo se reunieron las instrucciones que controlan funciones específicas del microprocesador o que actúan sobre registros especiales no direccionados como memoria interna normal.

La instrucción que sigue es la típica NO OPERATION, existente en casi todos los microprocesadores.

**NOP**; no hace nada, consume tiempo

Esta instrucción solo sirve para introducir una demora en el programa, equivalente al tiempo de ejecución de una instrucción. No afecta ningún bit de la palabra de estado.

La siguiente es una instrucción específica de control de los puertos de entrada/salida.

**TRIS f**; carga el tristate control, **TRISf = W**

Esta instrucción (TRIState) carga el registro de control de los buffers tristate de un puerto de entrada salida (data dirección register), con el valor contenido en W. El parámetro f debe ser la dirección de memoria interna del puerto, aunque el valor W no será cargado en el puerto sino en el registro de tristate del mismo. Los valores válidos para f son 4 y 5 en los 16C54/56 y 4, 5 y 6 en los 16C55/57. Esta instrucción no afecta ningún bit de la palabra de estado.

La siguiente instrucción sirve para programar el registro OPTION que controla el RTCC y prescaler

**OPTION;** carga el registro OPTION, **OPTION = W**

El registro OPTION no es accesible como memoria interna y solo se lo puede programar con esta instrucción. Esta instrucción no afecta ningún bit de la palabra de estado.

La instrucción que sigue borra el contador del watch dog timer. Este registro tampoco está accesible como memoria, y esta es la única instrucción que lo modifica.

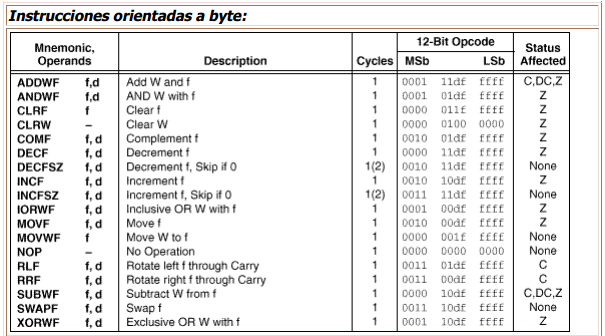
**CLRWDT;** borra el watch dog timer, WDT = 0

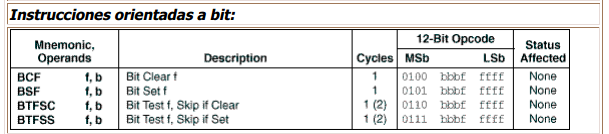
Esta instrucción, además, coloca en uno los bits PD (power down) y TO (time-out) de la palabra de estado.

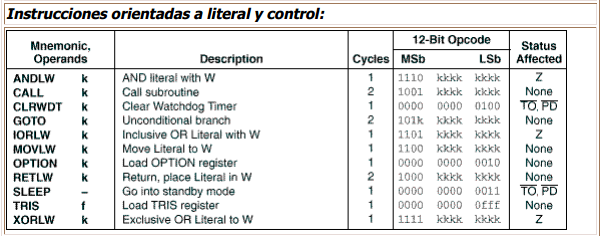
La siguiente es una instrucción especial de control del microcontrolador que lo pone en el modo power down. En este modo el microprocesador se detiene, el oscilador se apaga, los registros y puertos conservan su estado, y el consumo se reduce al mínimo. La única forma de salir de este estado es por medio de un reset o por time-out del watch dog timer.

**SLEEP**; coloca el µC en modo sleep, WDT = 0 Esta instrucción, además, borra el bit PD (power down) y setea el bit TO (time-out) de la palabra de estado.

Resumen de instrucciones (clasificación según el fabricante en tres grupos):







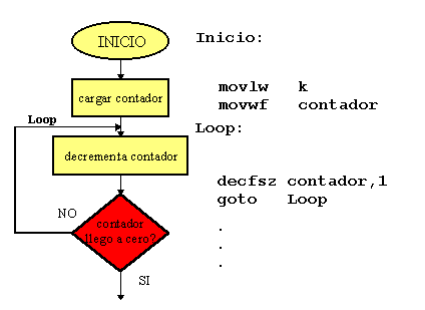
En esta tabla de resumen del conjunto de instrucciones se pueden observar los mnemónicos, la explicación, el número de ciclos, el código de máquina y los bits afectados del registro STATUS para cada una de las instrucciones.

**Subrutina Temporizador**

Temporización Existen momentos dentro de la programación en los que se necesita realizar un retardo de tiempo. Los retardos de tiempo se pueden obtener mediante hardware o por medio de ciclos repetitivos basados en software. La precisión de los retardos generados por software depende en esencia del tipo de oscilador que se utilice como base de tiempo en el microcontrolador, la mayor precisión se obtiene de los cristales de cuarzo. La velocidad a la que se ejecuta el código (instrucciones) depende de la velocidad del oscilador y del número de ciclos de máquina ejecutados. Las instrucciones necesitan 1 ó 2 ciclos de máquina para ser ejecutadas. Un ciclo de máquina es un tiempo utilizado por el microcontrolador para realizar sus operaciones internas y equivale a cuatro ciclos del oscilador. Por tanto: Tciclo máq.= 4 \* Tosc ­­­ Tciclo máq = 4 / fosc.

El número de ciclos de máquina utilizados por una instrucción para ser ejecutada depende de la misma. Las instrucciones que modifican el contador de programa necesitan dos (2) ciclos de máquina, mientras que todas las demás necesitan tan solo uno (1).

El hecho de generar ciclos repetitivos por medio del programa y calcular el tiempo total de ejecución nos puede ayudar a generar tiempos precisos.



**Operación # de ciclos**

La carga de k en W 1

La carga de W en el contador 1

El decremento del contador mientras no llegue a cero k-1

El decremento del contador cuando llegue a cero 2

El salto a Loop 2 \* (k-1)

Total: 3\*k+1

Por cada instrucción agregada debe incluirse en la cuenta total el número de ciclos correspondiente a dicha instrucción. Trabajando a 4 Mhz y asumiendo que k se remplaza por el valor 15d en el ejemplo tendríamos un tiempo igual a:

Número de ciclos = (3\*15) +1 = 46 ciclos de máquina, Tciclo máq.= 4 / 4 Mhz = 1 µ segundo, el tiempo total del ejemplo entonces será 46µ segundos.