

¿Qué es un microcontrolador?

- Es un pequeño computador alojado dentro de un chip.

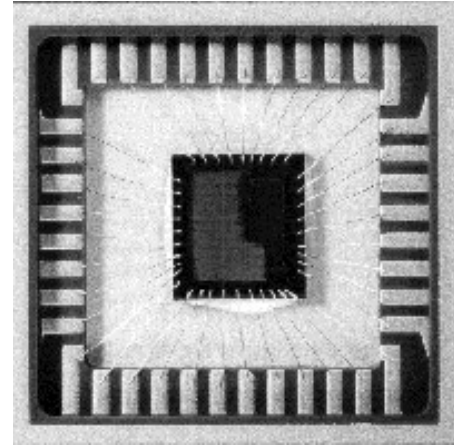
Contiene todos sus componentes básicos (sin teclado, ni ratón, ni monitor) a una escala mucho menor.

Un microcontrolador está pensado para comunicarse con otro *hardware*, mientras que un PC (*personal computer*) está pensado para comunicarse con personas.

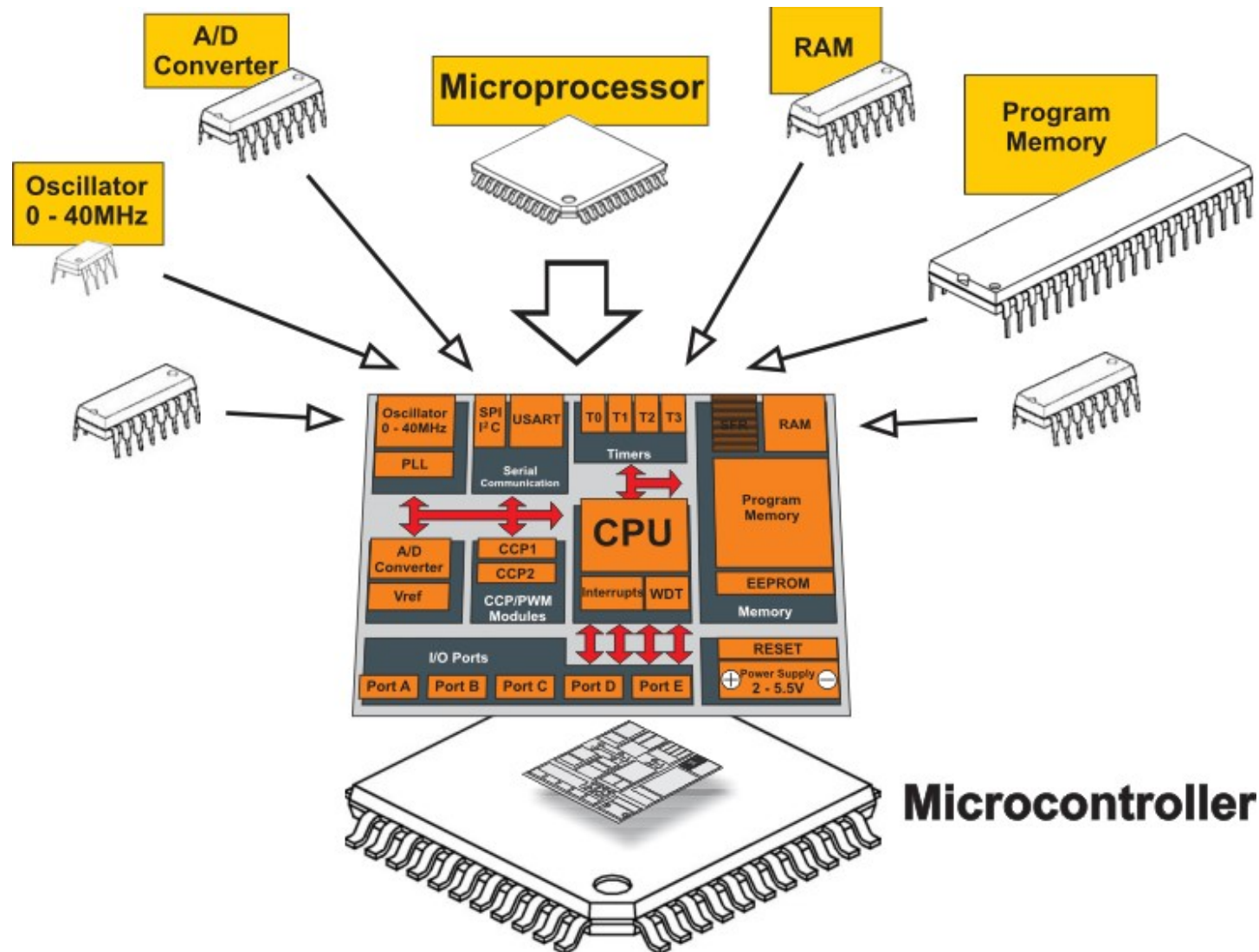
- Es un dispositivo programable destinado a realizar una determinada tarea (o tareas) dentro del sistema al que pertenece. La tarea que realiza se establece por medio del código que se guarda en su memoria de programa (*program memory*).

micro \equiv pequeño

controlador \equiv se utiliza en aplicaciones de '*control*'(este término resulta ambiguo).



- Un μC es por si solo un computador (no necesita de ningún otro componente para funcionar \equiv ejecutar instrucciones)...pero necesita otros circuitos con los que comunicarse (que le proporcionen datos y que reciban los resultados de los datos procesados por el μC !!!)



- Los microcontroladores (μ Cs) están diseñados para tener un bajo coste.

- Entre los componentes de un microcontrolador cabe destacar:

- _ CPU

- _ ROM: almacena el programa (FLASH) y los datos fijos (constantes) (EEPROM).

- _ RAM: almacena datos e instrucciones temporalmente durante la ejecución del programa (SRAM).

- _ Puertos de entrada/salida: son el medio que tiene el μ C para comunicarse con otros circuitos (enviar y recibir datos).

- _ Temporizadores para contar eventos o medir tiempos

- _ Circuitos de interrupción

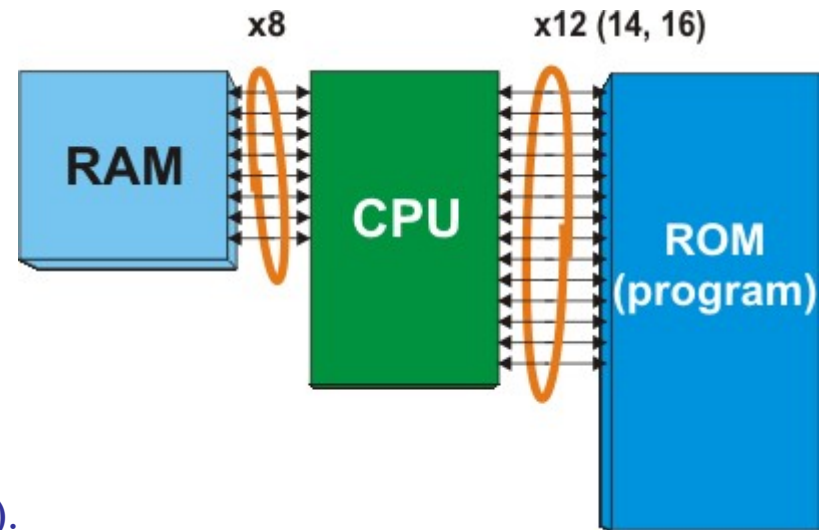
- _ Buses: comunicación interna

La inmensa mayoría de los μC que se pueden encontrar en el mercado se caracterizan por tener:

- Un repertorio de instrucciones reducido \equiv arquitectura RISC (*Reduced Instruction Set Computer*)
- Un cauce segmentado (*pipelining*)
- Una arquitectura *Harvard* o *Super-Harvard* (o *Hardvard modificada*)

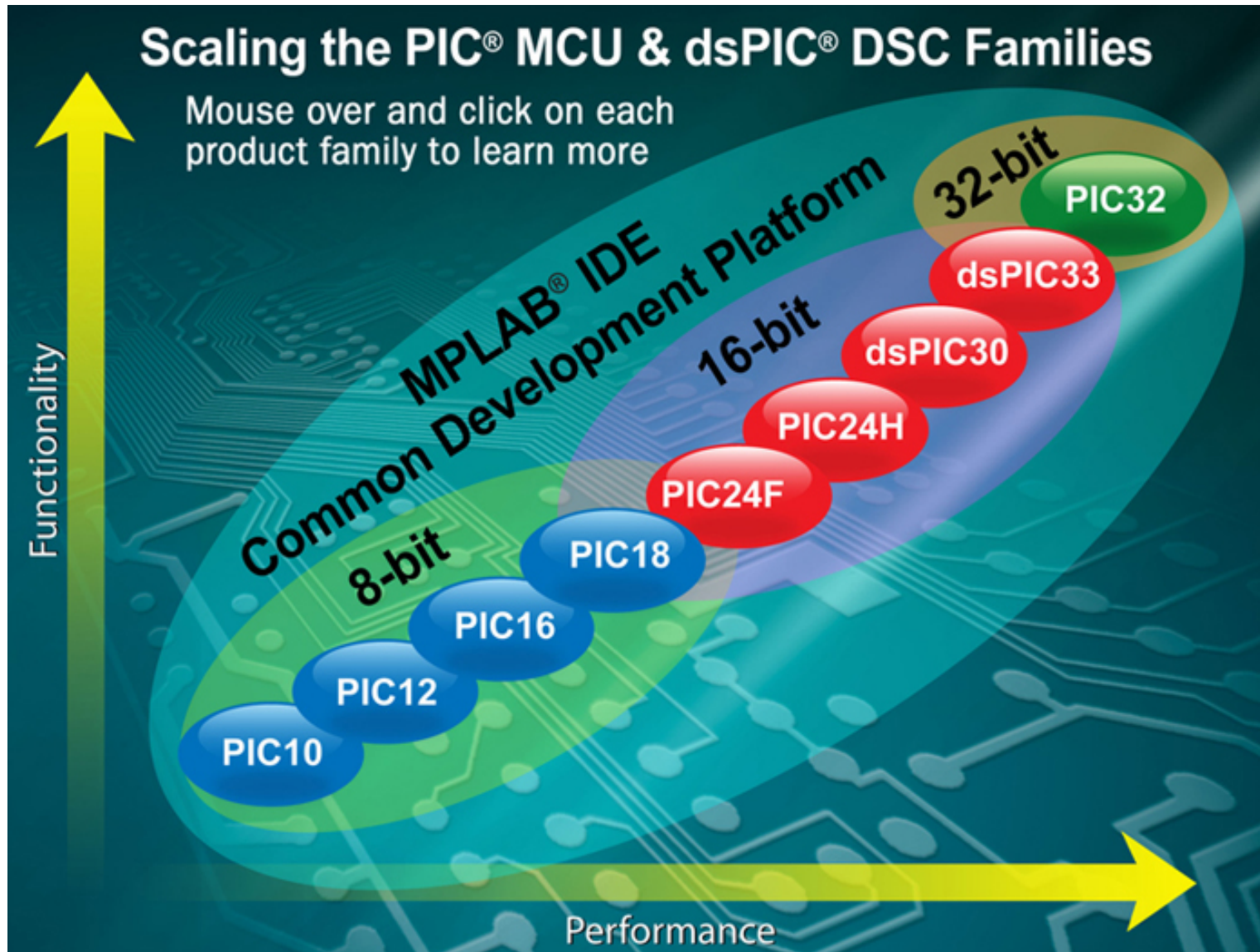
Arquitectura Harvard

- ✓ Se utilizan memorias independientes para guardar los datos y las instrucciones.
- ✓ Hay por lo menos dos buses internos que permiten el acceso simultaneo a la memoria de datos (RAM) y a la de instrucciones (ROM).



Si la ejecución de una instrucción requiere una operación de lectura/escritura en la memoria de datos, esta arquitectura permite que la CPU busque la siguiente instrucción a ejecutar en la memoria de programa mientras se realiza la operación de lectura/escritura en la memoria de datos. Esto reduce el tiempo de ejecución de los programas a costa de un hardware más complejo.

Familias de microcontroladores de Microchip

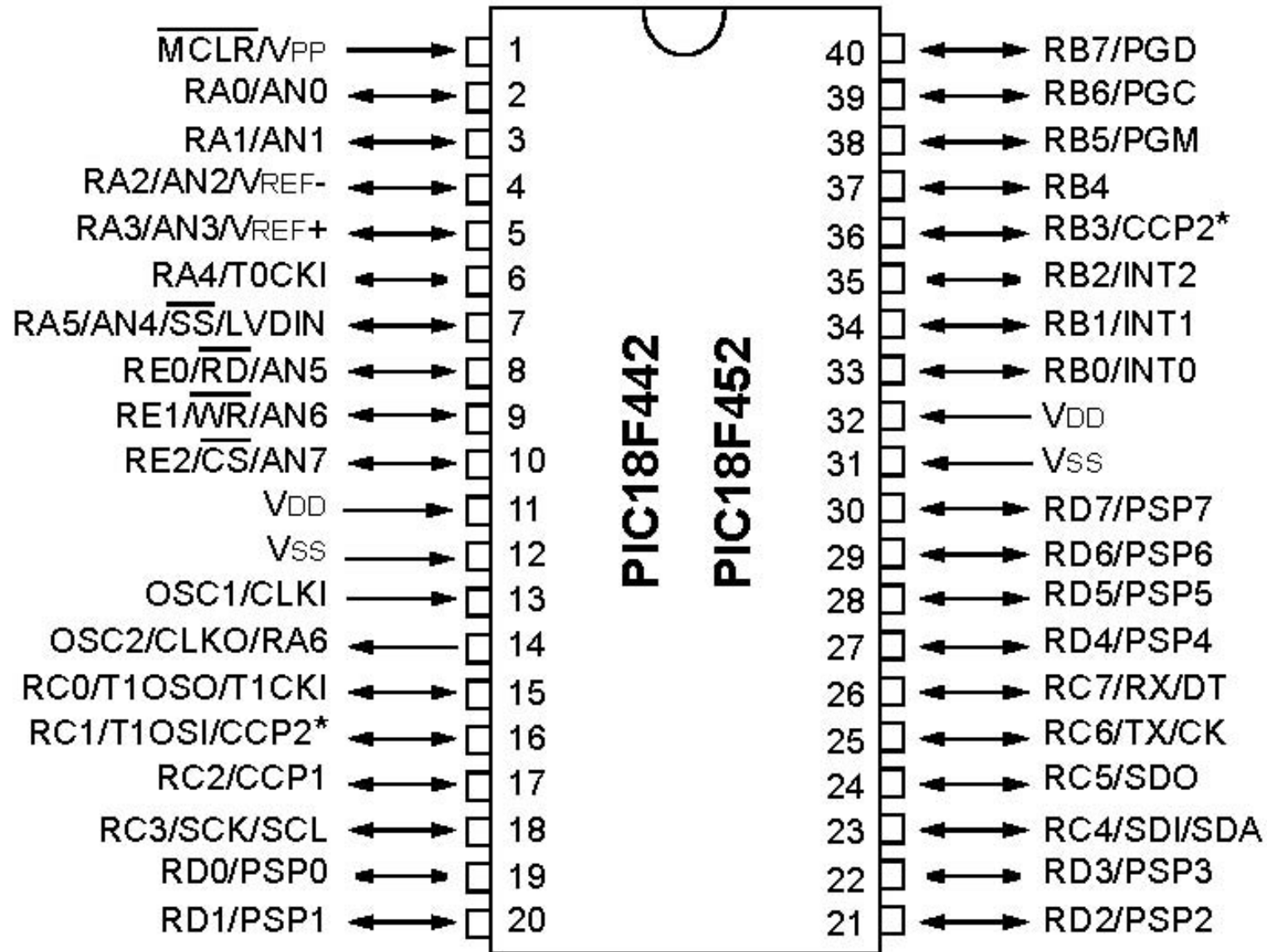


$$T_{CY} = 4T_{osc}$$

Características básicas de la serie PIC18FXX2

Features	PIC18F242	PIC18F252	PIC18F442	PIC18F452
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	16K	32K	16K	32K
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	17	17	18	18
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin PLCC 44-pin TQFP	40-pin DIP 44-pin PLCC 44-pin TQFP

$$T_{CY} = 4T_{osc}$$

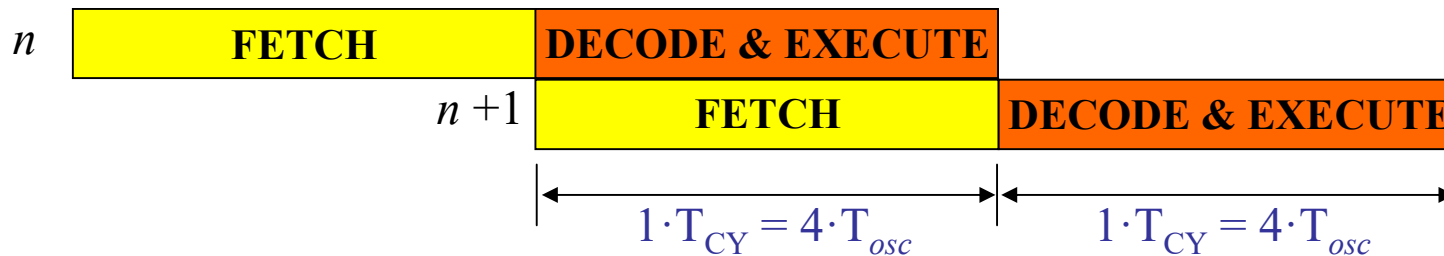


- El contador de programa (PC: *program counter*) se incrementa cada $4 \cdot T_{osc} = 1 \cdot T_{CY}$ ($T_{CY} \equiv \text{instruction cycle}$)

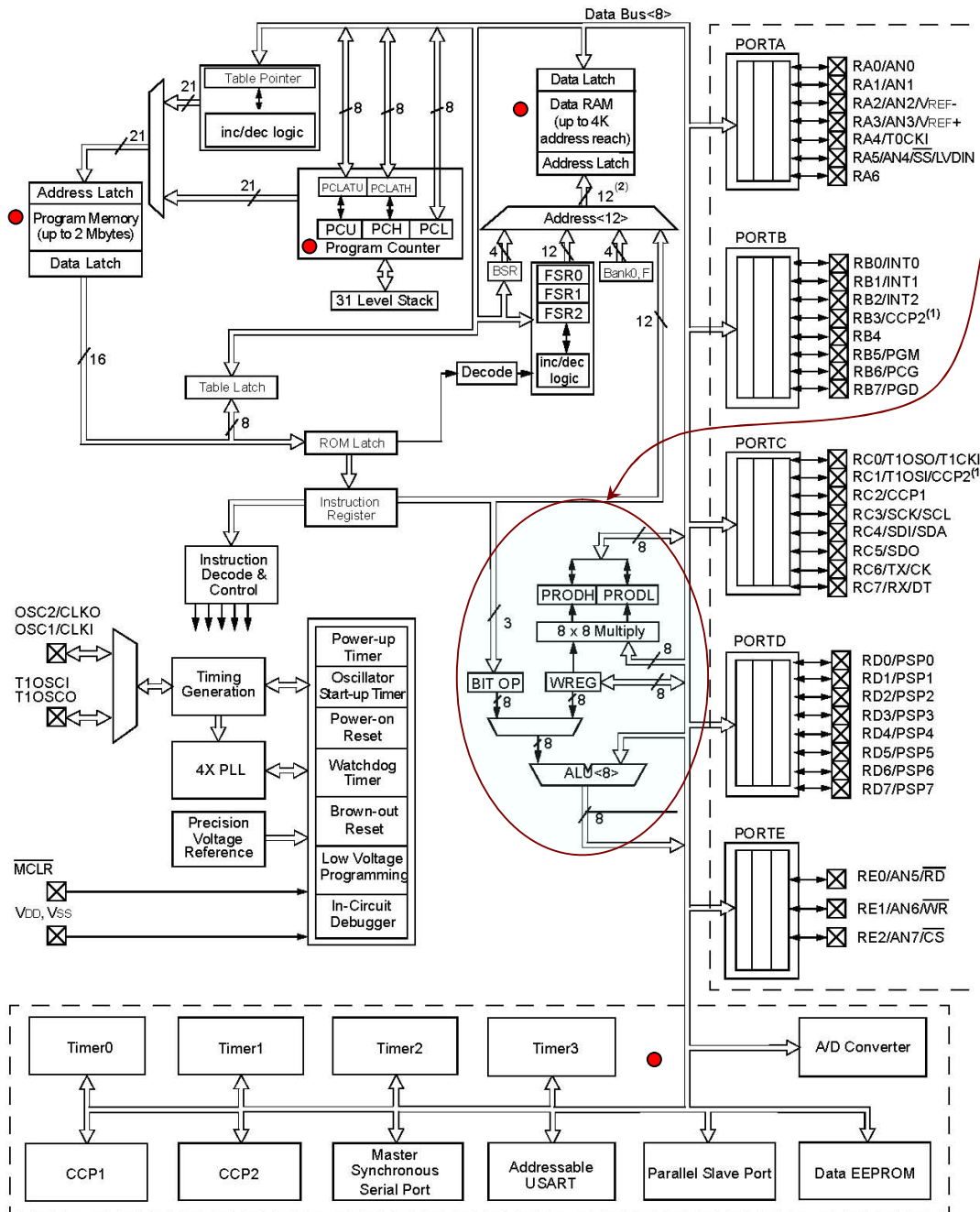
- Tiene un cauce segmentado de 2 niveles (las instrucciones se ejecutan en 2 etapas):

Fetch: búsqueda de la instrucción a ejecutar (dura $1 \cdot T_{CY} = 4 \cdot T_{osc}$)

Decode & execute: decodificación y ejecución de la instrucción (dura $1 \cdot T_{CY} = 4 \cdot T_{osc}$)



Curiosidad: de media se ejecuta 1 instrucción cada 4 periodos de la señal de reloj (T_{osc}). Así, por ejemplo, con una señal de reloj de frecuencia $f_{osc} = 8\text{MHz}$, se ejecutan una media de $2 \cdot 10^6$ instrucciones/segundo



La CPU consta de:

- ALU de 8 bits
- WREG registro de trabajo de 8 bits
- Multiplicador hardware de 8x8 (el resultado de la multiplicación se guarda en los registros PRODH y PRODL)

El bus de direcciones de la memoria de programa es de 21 bits (puede direccionar 2MB). El PIC18F452 tiene una memoria de programa de 32KB \Rightarrow sólo requiere un bus de 15 bits (los restantes 6 bits no se usan)

La memoria de programa contiene una pila de 31 niveles, la cual se usa habitualmente para guardar la dirección de retorno de las interrupciones y de las subrutinas.

La memoria de datos tiene un bus de direcciones de 12 bits (puede direccionar 4KB). Está formada por registros de función especial (*SFR*) y registros de propósito general, organizados en bancos. El PIC18F452 tiene una memoria de 1536 Bytes

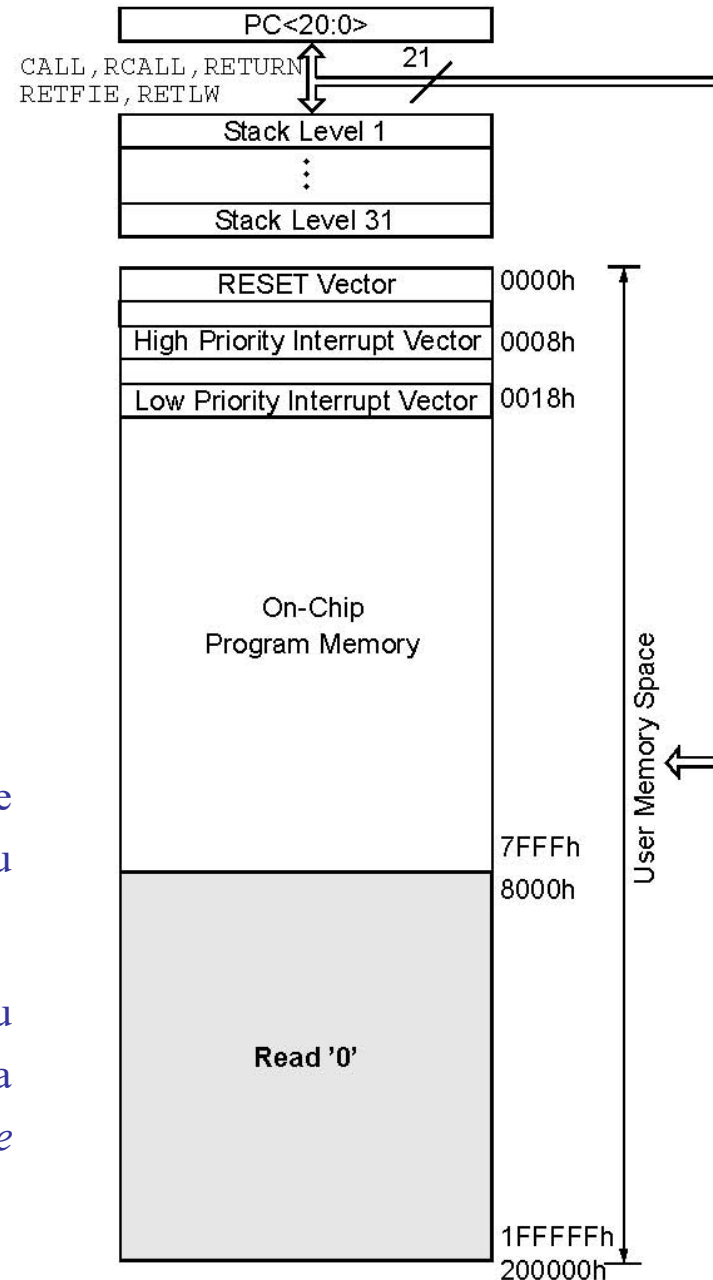
En la parte inferior del diagrama de bloques anterior se puede ver:

- 4 timers/counters (Timers 0, 1, 2 y 3)
- 2 módulos Capture/Compare/PWM (CCP1 y CCP2)
- 2 módulos de comunicación serie (MSSP, USART)
- 1 convertidor A/D de 10 bits (8 canales de entrada)
- 1 memoria EEPROM de 256 Bytes (para guardar datos que no van a cambiar frecuentemente de valor, de modo que no ocupen espacio en la memoria del programa)

Organización de la Memoria de Programa (PIC18F452)

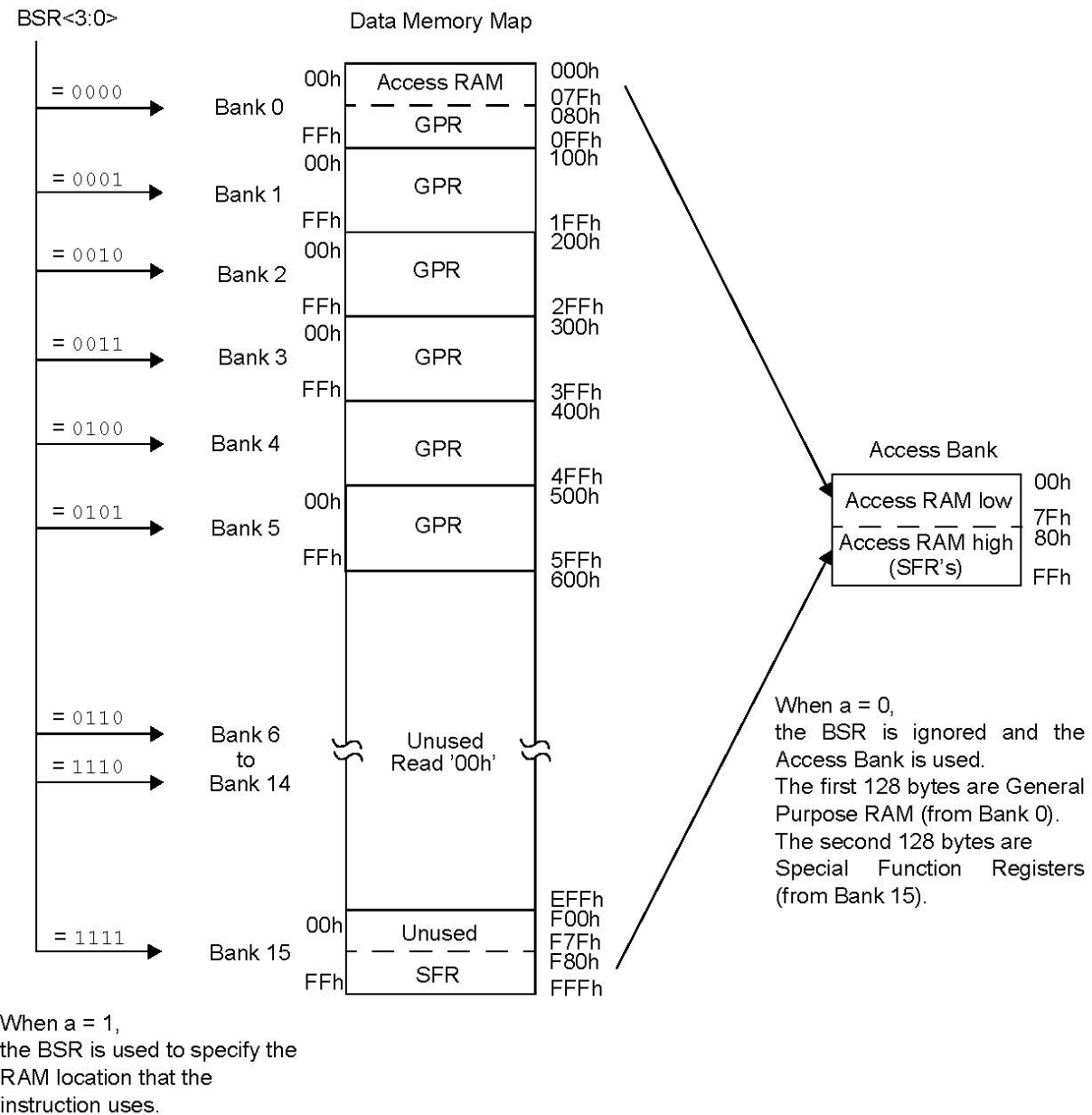
Notas:

- Un *mapa de memoria* es un diagrama que muestra, en este caso, cómo un μC utiliza su memoria interna.
- Cada μC tiene distribuida/organizada su memoria de una forma particular. Dicha distribución se indica mediante un *mapa de memoria*.



- El espacio de memoria del que se dispone para guardar el código del programa abarca las direcciones comprendidas entre la 00000h y la 7FFFh.
- Operaciones de lectura en las posiciones comprendidas entre la dirección 8000h y la dirección 1FFFFFFh proporcionará el valor 0.
- Las direcciones 0008h y 0018h están reservadas para las interrupciones de alta y de baja prioridad, respectivamente. Las rutinas de servicio de las interrupciones (*ISR*) deben comenzar en una de éstas direcciones.
- El PIC18F452 dispone de una pila de 31 niveles (o entradas), que se utiliza para guardar las direcciones de retorno de las llamadas a funciones y de la atención a las interrupciones.

Organización de la Memoria de Datos (PIC18F452)



- El bus de direcciones de la memoria de datos es de 12 bits \Rightarrow puede direccionar hasta 4MB.

- La memoria está formada por 16 bancos de 256 Bytes cada uno. Sólo se pueden utilizar los 6 primeros bancos.

El PIC18F452 tiene una memoria de 1536 bytes (6 bancos X 256 bytes cada uno) que ocupan desde la dirección 000H hasta la dirección 5FFH.

- Los registros de función especial (*SFR* \equiv *special function register*) ocupan la mitad superior del banco 15 (son 101 registros). Dichos registros controlan el funcionamiento de diversos dispositivos, como: *timers/counters*, *A/D converter*, *interrupts*, *USART*, etc.

- Los registros de propósito general (*GPR* \equiv *general purpose registers*) se utilizan para guardar datos y como memoria temporal en las aplicaciones de usuario.

Puertos PIC18F452

PORTA: RA6_(osc2), RA5, RA4, RA3, RA2, RA1, RA0

PORTB: RB7, RB6, RB5, RB4, RB3, RB2, RB1, RB0

PORTC: RC7, RC6, RC5, RC4, RC3, RC2, RC1, RC0

PORTD: RD7, RD6, RD5, RD4, RD3, RD2, RD1, RD0

PORTE: RE2, RE1, RE0

Nota: los terminales de los puertos pueden realizar más de una función, entre las que se incluye la de actuar como entrada/salida digital.

- *Nota de C:* Configuración de **un terminal** (pin) de un puerto como *salida* digital

$\text{TRISX.B}\alpha = 0 \rightarrow$ el terminal α del puerto X es una *salida* digital

Ejemplo 1:

$\text{TRISC.B5} = 0;$ //Se configura el terminal RC5 como una salida

$\text{PORTC.B5} = 1;$ // A partir del momento en el que se ejecuta ésta instrucción, en
// el terminal RC5 (de puerto C) hay un nivel de tensión alto.

Regla nemotécnica: $0 \approx O \equiv \text{Output}$

- *Nota de C:* Configuración de un terminal (pin) de un puerto como **entrada** digital

$\text{TRISX.B}\alpha = 1 \rightarrow$ el terminal α del puerto **X** es una **entrada** digital

Ejemplo 1:

$\text{TRISA.B3} = 1;$ //Se configura el terminal RA3 como una entrada

Ejercicio 1: ¿Qué terminales del puerto D quedan configurados como entradas y cuales como salidas después de ejecutar la siguiente instrucción?

$\text{TRISD} = 0xF5;$

Ejercicio 2: ¿Qué valor tomará la variable x después de que se ejecuten las siguientes instrucciones? (la respuesta no es tan sencilla como parece!!!)

```
TRISC.B4 = 1;
if (PORTC.B4 == 1)
    x = 20;
else
    x = 10;
```

Regla nemotécnica: $1 \approx I \equiv \text{Input}$

PORTA:

- Al encender el PIC o al dejar de activar el MCLR se cumple lo siguiente:
 - _ RA5, RA3:RA0 se configuran como *entradas analógicas* (y se leen como 0s)
 - _ RA6 y RA4 se configuran como salidas digitales
- RA4 como salida digital es de tipo '*drenador abierto*' y como entrada digital es una entrada *schmitt trigger*.
- Algunos terminales pueden funcionar como *entradas/salidas digitales* o bien como *entradas analógicas*. El comportamiento de dichos terminales se establece mediante los 4 bits menos significativos del registro ADCON1.
- Hay que poner a 1 el correspondiente bit del registro TRIS de los terminales que se configuren para actuar como entradas analógicas.

PORTA 18F452

TABLE 9-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2/VREF-	bit2	TTL	Input/output or analog input or VREF-.
RA3/AN3/VREF+	bit3	TTL	Input/output or analog input or VREF+.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/ \overline{SS} /AN4/LVDIN	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input, or low voltage detect input.
OSC2/CLKO/RA6	bit6	TTL	OSC2 or clock output or I/O pin.

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 9-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	-x0x 0000	-u0u 0000
LATA	—	LATA Data Output Register							-xxx xxxx	-uuu uuuu
TRISA	—	PORTA Data Direction Register							-111 1111	-111 1111
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by PORTA.

PORTB:

- Al encender el PIC o al dejar de activar el MCLR todos los terminales del puerto B se configuran como *entradas digitales* y las resistencias de *pullup* internas están desactivadas.
- Cada terminal tiene una resistencia de *pullup* interna que se activa poniendo a cero el bit INTCON2.RBPU y configurando dicho terminal como una entrada (digital). Las resistencias de *pullup* se desactivan cuando el correspondiente terminal se configura como una salida.

PORTB 18F452

Name	Bit#	Buffer	Function
RB0/INT0	bit0	TTL/ST ⁽¹⁾	Input/output pin or external interrupt input0. Internal software programmable weak pull-up.
RB1/INT1	bit1	TTL/ST ⁽¹⁾	Input/output pin or external interrupt input1. Internal software programmable weak pull-up.
RB2/INT2	bit2	TTL/ST ⁽¹⁾	Input/output pin or external interrupt input2. Internal software programmable weak pull-up.
RB3/CCP2 ⁽³⁾	bit3	TTL/ST ⁽⁴⁾	Input/output pin or Capture2 input/Compare2 output/PWM output when CCP2MX configuration bit is enabled. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5/PGM ⁽⁵⁾	bit5	TTL/ST ⁽²⁾	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Low voltage ICSP enable pin.
RB6/PGC	bit6	TTL/ST ⁽²⁾	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST ⁽²⁾	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

- Note**
- 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
 - 2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
 - 3: A device configuration bit selects which I/O pin the CCP2 pin is multiplexed on.
 - 4: This buffer is a Schmitt Trigger input when configured as the CCP2 input.
 - 5: Low Voltage ICSP Programming (LVP) is enabled by default, which disables the RB5 I/O function. LVP must be disabled to enable RB5 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

PORTC:

- Al encender el PIC o al dejar de activar el MCLR todos los terminales del puerto C se configuran como *entradas digitales*.

PORTC 18F452

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin, Timer1 oscillator input, or Capture2 input/Compare2 output/PWM output when CCP2MX configuration bit is set.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I ² C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or Data I/O (I ² C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin, Addressable USART Asynchronous Transmit, or Addressable USART Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin, Addressable USART Asynchronous Receive, or Addressable USART Synchronous Data.

Legend: ST = Schmitt Trigger input

TABLE 9-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
LATC	LATC Data Output Register								xxxx xxxx	uuuu uuuu
TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

PORTD:

·Al encender el PIC o al dejar de activar el MCLR todos los terminales del puerto D se configuran como *entradas digitales*.

PORTD 18F452

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit0.
RD1/PSP1	bit1	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit1.
RD2/PSP2	bit2	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit2.
RD3/PSP3	bit3	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit3.
RD4/PSP4	bit4	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit4.
RD5/PSP5	bit5	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit5.
RD6/PSP6	bit6	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit6.
RD7/PSP7	bit7	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit7.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffer when in Parallel Slave Port mode.

TABLE 9-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
LATD	LATD Data Output Register								xxxx xxxx	uuuu uuuu
TRISD	PORTD Data Direction Register								1111 1111	1111 1111
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

PORTE:

· Al encender el PIC o al dejar de activar el MCLR todos los terminales del puerto E se configuran como *entradas analógicas*. Para configurarlas como terminales digitales el μC debe ejecutar la siguiente instrucción:

ADCON1 = 0x07;

o bien

ADCON1 = 0x06;

Nota: en el caso de utilizar el convertidor AD del μC no se debe ejecutar ninguna de las instrucciones anteriores. Hay que estudiar el tema 17 de las hojas de datos del μC .

· Los terminales del puerto E, actuando como entradas, son de tipo *schmitt trigger*.

PORTE 18F452

Name	Bit#	Buffer Type	Function
RE0/ $\overline{\text{RD}}$ /AN5	bit0	ST/TTL ⁽¹⁾	Input/output port pin or read control input in Parallel Slave Port mode or analog input: $\overline{\text{RD}}$ 1 = Not a read operation 0 = Read operation. Reads PORTD register (if chip selected).
RE1/ $\overline{\text{WR}}$ /AN6	bit1	ST/TTL ⁽¹⁾	Input/output port pin or write control input in Parallel Slave Port mode or analog input: $\overline{\text{WR}}$ 1 = Not a write operation 0 = Write operation. Writes PORTD register (if chip selected).
RE2/ $\overline{\text{CS}}$ /AN7	bit2	ST/TTL ⁽¹⁾	Input/output port pin or chip select control input in Parallel Slave Port mode or analog input: $\overline{\text{CS}}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 9-10: SUMMARY OF REGISTERS ASSOCIATED WITH PORTE

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -000	---- -000
LATE	—	—	—	—	—	LATE Data Output Register			---- -xxx	---- -uuu
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTE.

Interrupts (interrupciones):

- Una interrupción consiste en un suceso que suspende (temporalmente) la ejecución normal del programa que está ejecutando el μC para que se ejecute una función, previamente definida por el programador, denominada '*rutina de atención a la interrupción*' (**ISR** \equiv *Interrupt service routine*).

Cuando el μC termina de ejecutar la rutina de atención a una interrupción, continúa con la ejecución del programa principal en donde lo dejó para atender a la interrupción.

- El uso de interrupciones evita que el μC tenga que estar periódicamente comprobando si determinados sucesos internos o externos han ocurrido o no. Lo cual, a su vez, evita que se produzcan retardos o paradas innecesarias en la ejecución de los programas.

Nota: no es aconsejable realizar consultas periódicas (*polling* \equiv comprobación continua). Sin embargo, es muy aconsejable utilizar interrupciones.

Ejemplos de aplicaciones en las que las interrupciones pueden ser útiles:

- ✓ Aplicaciones que requieren una atención inmediata del sistema de control o de supervisión. *Ejem.:* una caída de tensión, un incendio, etc.
- ✓ Realizar una serie de tareas cada cierto tiempo (\equiv de forma periódica): se pueden utilizar interrupciones para que avisen al μ C cuándo debe realizar una determinada tarea.
- ✓ Aplicaciones multitarea: a cada tarea se le asigna un tiempo de CPU continuo dado. Si la ejecución del código de una tarea supera el tiempo (continuo) máximo asignado, se puede utilizar una interrupción para que interrumpa la ejecución de dicho código.
- ✓ Atención rápida a dispositivos periféricos: en algunas aplicaciones es importante saber cuándo un periférico finaliza la tarea que está realizando.

Ejem.: finalización de una conversión A/D.

Características de las interrupciones de los μC de Microchip:

Los μC de Microchip tienen dos fuentes de interrupción: internas y externas

- Internas (*core*):

- _ INT0, INT1, INT2 por flanco de subida o de bajada de la señal externa aplicada.

- _ Desbordamiento del Timer 0.

- _ PORTB (RB7, RB6, RB5, RB4) por cambio de nivel de señal externa aplicada.

▪ **Externas (peripheral):** son todas las que no son de tipo *core*. Para habilitar estas interrupciones es necesario (pero no suficiente) poner el bit INTCON.PEIE a 1.

_ desbordamiento de los *timers* TMR1, TMR2, TMR3

_ finalización de una conversión A/D.

_ comparador

_ módulo CCPx

_ escritura en la EEPROM/FLASH

_ transmisión o recepción por UART

_ high/low voltage detect

_ etc. (todas las que tiene el μC y que no se indican en la diapositiva anterior)

Cada fuente de interrupción (excepto INT0) tiene 3 bits asociados para controlar su funcionamiento:

- *flag bit* (xxxIF): se pone a 1 siempre que ocurra el suceso asociado a dicha interrupción.

Ejemplo: el bit PIR1.ADIF se pone a 1 siempre que el convertidor AD finaliza una conversión (con independencia de si la interrupción del convertidor AD está habilitada o no). En el caso de que las interrupciones del convertidor AD estén habilitadas, cada vez que finaliza una conversión AD, el μC interrumpirá (temporalmente) la ejecución del código que esté ejecutando en dicho momento para ejecutar la rutina asociada a la interrupción del convertidor AD [con el compilador de MikroC se ejecuta la función *interrupt()*]

- *enable bit* (xxxIE): sirve para habilitar/deshabilitar una fuente de interrupción

Ejemplo: PIE1.ADIE = 1; \rightarrow se habilitan las interrupciones del convertidor AD. Siempre que el bit (flag) PIR1.ADIF se ponga a 1, el μC dejará de ejecutar el código que esté ejecutando en ese momento para ejecutar la rutina asociada a la finalización de una conversión AD [con el compilador de MikroC se ejecuta la función *interrupt()*]. 33

- *priority bit* (xxxIP): establece si la interrupción asociada a dicho bit es de alta o bien es de baja prioridad. En un PIC18F452, sólo hay 2 niveles de prioridad (alto y bajo). Las interrupciones con un nivel de prioridad *alto* se caracterizan porque pueden interrumpir la ejecución de la rutina de servicio de una interrupción que tenga un nivel de prioridad *bajo*.

Ejemplo: IPR1.ADIP = 0; → la interrupción creada por la finalización de una conversión AD no puede interrumpir la ejecución de la rutina de servicio de otra interrupción que se esté ejecutando en ese momento. En el caso de que ocurra un hecho como éste, la rutina de servicio de la interrupción del convertidor AD se ejecutará cuando finalice la ejecución de la rutina que se esté ejecutando.

IPR1.ADIP = 1; → en este caso, la interrupción creada por la finalización de una conversión AD puede interrumpir la ejecución de la rutina de servicio de una interrupción que tenga un nivel de prioridad *bajo* que se esté ejecutando cuando el bit ADIF se pone a 1.

Nota: no existe el bit INT0IP. La interrupción INT0 tiene siempre un nivel *alto* de prioridad.

Para que una interrupción sea aceptada por la CPU del μC deben cumplirse las siguientes condiciones:

1^a: Debe estar habilitada dicha interrupción (*Ejem.:* `INTCON.INT0IE = 1`)

2^a: Si la interrupción es de tipo *peripheral*, entonces hay que poner a 1 el bit `INTCON.PEIE`

3^a: Hay que poner a 1 el bit `INTCON.GIE` (*GIE* \equiv *global interrupt enable*)

Importante: el bit `xxxIF` (*interrupt flag*) de una interrupción debe ponerse a 0 por software. Dicha operación debe incluirse entre las instrucciones de la rutina de atención a la interrupción.

Curiosidad: se puede poner a 1 el bit `xxxIF` (*interrupt flag*) de una interrupción por software... es decir, se puede provocar artificialmente una interrupción por software.

Interrupciones por cambio de nivel en los terminales del puerto B:

- Los terminales RB0/INT0, RB1/INT1 y RB2/INT2 se pueden configurar, *de forma independiente*, para provocar una *interrupción por flanco* (\equiv por el cambio de la tensión presente en dichos terminales de nivel alto a nivel bajo o bien de nivel bajo a nivel alto).

Notas:

_ Las interrupciones asociadas a los terminales RB0, RB1 y RB2 no se pueden hacer sensibles a ambos tipos de flancos a la vez.

_ *Para que un cambio de nivel en un terminal RB0, RB1 o RB2 provoque una interrupción es necesario que el correspondiente terminal esté configurado como entrada [además de que la correspondiente interrupción (INTx) esté habilitada]*

Configuración de la interrupción INT0 (RB0):

```
void interrupt() //rutina de servicio de interrupciones (MikroC)
```

```
{
```

```
.....
```

```
    INTCON.INT0IF = 0; // se borra el flag de la interrupción INT0
```

```
}
```

```
void main()
```

```
{
```

```
.....
```

```
    TRISB.B0 = 1; //se configura RB0 como entrada
```

```
    INTCON2.INTEDG0 = x; //la interrupción la provoca un flanco de subida (x=1)/ bajada (x=0)
```

```
    INTCON.INT0IF = 0; // se pone el flag de la interrupción INT0 a 0
```

```
    INTCON.INT0IE = 1; // se habilita la interrupción INT0
```

```
    INTCON.GIE = 1; // se habilitan las interrupciones en general
```

```
.....
```

```
}
```

Configuración de la interrupción INT1 (RB1):

```
void interrupt() //rutina de servicio de interrupciones (MikroC)
{
    .....
    INTCON3.INT1IF = 0; // se borra el flag de la interrupción INT1
}
```

```
void main()
{
    .....
    TRISB.B1 = 1; // se configura RB1 como entrada
    INTCON2.INTEDG1 = x; // la interrupción la provoca un flanco de subida (x=1) / bajada (x=0)
    INTCON3.INT1IF = 0; // se pone el flag de la interrupción INT1 a 0
    INTCON3.INT1IE = 1; // se habilita la interrupción INT1
    INTCON.GIE = 1; // se habilitan las interrupciones en general
    .....
}
```

Configuración de la interrupción INT2 (RB2):

```
void interrupt() //rutina de servicio de interrupciones (MikroC)
{
    ....
    INTCON3.INT2IF = 0; // se borra el flag de la interrupción INT2
}
```

```
void main()
{
    ....
    TRISB.B2 = 1; // se configura RB1 como entrada
    INTCON2.INTEDG2 = x; //la interrupción la provoca un flanco de subida (x=1)/ bajada (x=0)
    INTCON3.INT2IF = 0; // se pone el flag de la interrupción INT2 a 0
    INTCON3.INT2IE = 1; // se habilita la interrupción INT2
    INTCON.GIE = 1; // se habilitan las interrupciones en general
    .....
}
```

- Los terminales **RB4**, **RB5**, **RB6** y **RB7** se pueden configurar para provocar una interrupción por cambio de nivel. Cumpliéndose que:
 - De estos 4 terminales, sólo los cambios de nivel en los configurados como *entradas* pueden provocar una interrupción.
 - Se producirá una interrupción siempre que la tensión presente en *cualquiera* de estos 4 terminales (**que hayan sido configurados como entradas**) describa un flanco de *subida* o de *bajada* (y la interrupción esté habilitada: $\text{INTCON.RBIE} = 1$). Esta característica no se puede modificar.
 - Para que se borre el *flag* de esta interrupción ($\text{INTCON.RBIF} = 0$;) es necesario leer previamente el contenido del registro **PORTB**. Esto no ocurre con el *flag* de ninguna otra interrupción en el PIC18F452
- *Polling of PORTB is not recommended while using the interrupt on change feature.*

Configuración de la interrupción RB (RB4, RB5, RB6 o RB7)

```
void interrupt() // rutina de servicio de la interrupción (MikroC)
{
    ....
    x = PORTB; // hay que leer el puerto B antes de borrar el flag
    INTCON.RBIF = 0; // se borra el flag
}
```

```
void main()
{
    ....
    TRISB.Bx = 1; // x = 4, 5, 6, 7
    x = PORTB; // x es una variable de 8 bits.
    INTCON.RBIF = 0; // se pone el flag a 0
    INTCON.RBIE = 1; // se habilita la interrupción por cambio de nivel
    INTCON.GIE = 1; // se habilitan las interrupciones en general
    .....
}
```

Nota: Sólo los terminales configurados como entradas pueden provocar una interrupción.

Temporizadores – Contadores (*timers-counters*)

Un módulo temporizador/contador está formado básicamente por un *contador* digital como los que se estudian en *Sistemas Digitales*, con la particularidad de que su funcionamiento se controla por *software*. Dicho contador cuenta flancos de subida (o de bajada) de una señal binaria que se hace llegar a su entrada.

La capacidad de contar de forma controlada los flancos de una señal binaria hace que, en la práctica, los módulos *timer-counters* se utilicen en aplicaciones como:

- _ *Medir el tiempo transcurrido entre dos instantes dados.*
- _ *Temporizador*: provocar una interrupción cuando haya transcurrido un tiempo dado
- _ Contar eventos-sucesos.
- _ Medir la frecuencia (o el periodo) de una señal.
- _ Generar señales.
- _ etc.

Filosofía – nomenclatura de Microchip en relación a los módulos ‘timers-counters’

- *En las hojas de datos de los μC de Microchip se habla de ‘timers’ cuando estos circuitos se utilizan para contar flancos (de subida) de una señal **periódica**. Dicha señal suele generarse a partir de la señal de reloj del μC ($F_{osc}/4$)*
- *En las hojas de datos de los μC de Microchip se habla de ‘counters’ cuando estos módulos se utilizan para contar flancos (de subida) de una señal **no-periódica**. Dicha señal suele ser externa al μC . Así, por ejemplo, se puede configurar el Timer 0 para contar los flancos de subida (o de bajada) de la señal aplicada a la patilla RA4 de un PIC18F452.*

- Medida del tiempo transcurrido entre dos instantes dados:

“Se puede medir el tiempo que transcurre entre dos instantes t_1 y t_2 contando los flancos de subida (o de bajada) que describe una señal digital periódica entre dichos instantes de tiempo”. Cumpliéndose que:

$$t_2 - t_1 \cong n \cdot T$$

siendo:

- n : el número de flancos contados entre los instantes t_1 y t_2
- $T = \text{cte.}$: periodo de la señal a la que se le cuentan los flancos

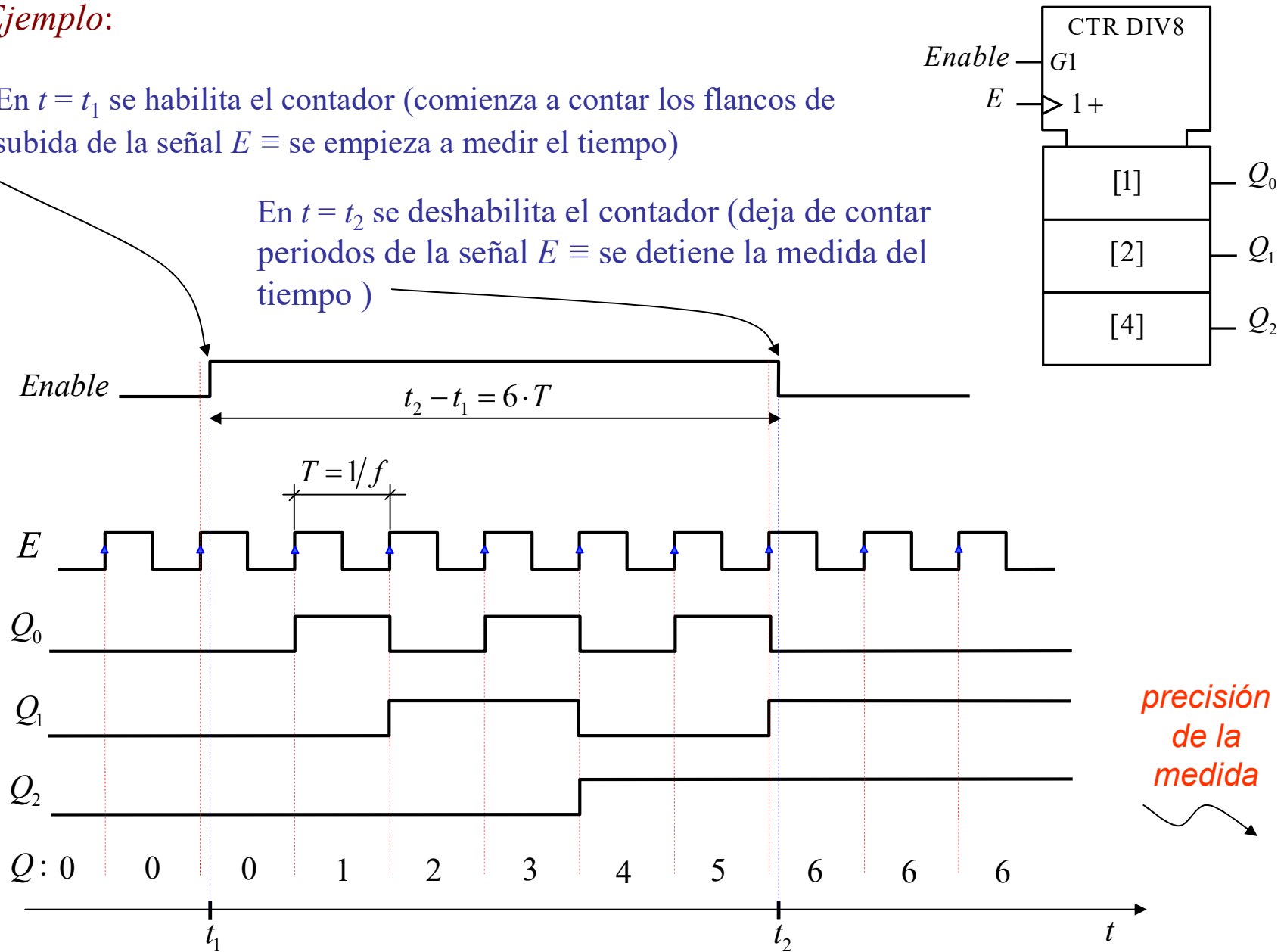
Nota: lo más cómodo y práctico es poner el contador a cero antes de ponerlo a contar flancos en $t = t_1$. De este modo el contenido del contador en $t = t_2$ es igual al en número n de flancos contados entre t_1 y t_2 (\cong número de periodos T transcurridos entre t_1 y t_2)

(ver diapositiva siguiente)

Ejemplo:

En $t = t_1$ se habilita el contador (comienza a contar los flancos de subida de la señal $E \equiv$ se empieza a medir el tiempo)

En $t = t_2$ se deshabilita el contador (deja de contar periodos de la señal $E \equiv$ se detiene la medida del tiempo)

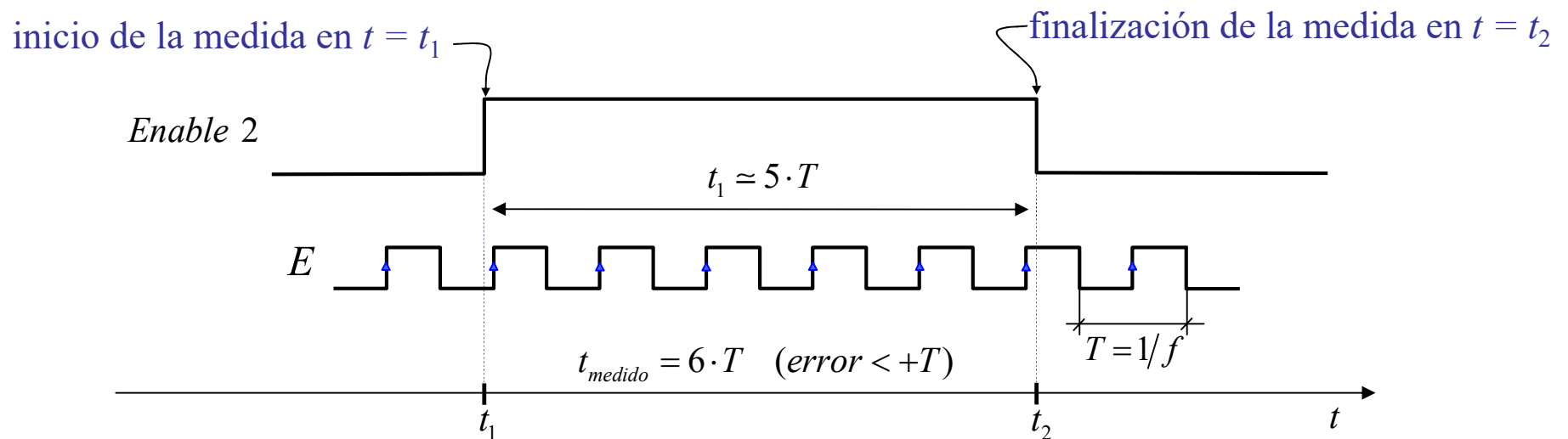
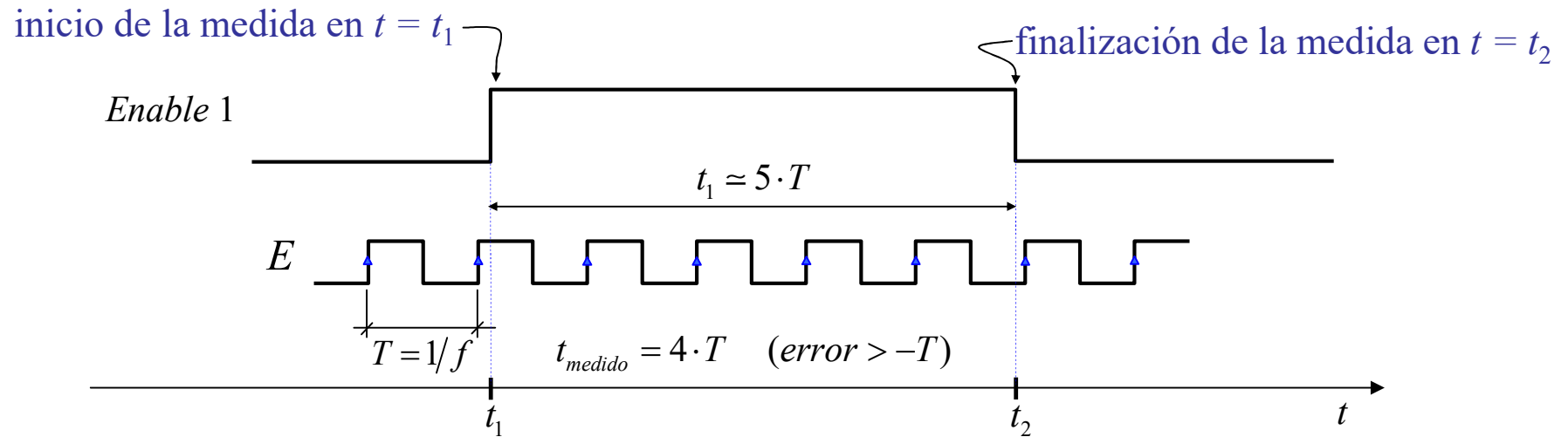


Con éste sistema de medida, el *error* que se comete está comprendido entre $-T < \text{error} < +T$, siendo T el periodo de la señal (E) a la que se le cuentan los flancos de subida (o de bajada).

Los factores que contribuyen a que el tiempo medido presente un *error* son:

- El instante en que se inicia la medida de tiempo con respecto al primer flanco de la señal (E) que se cuenta.
- El instante en el que finaliza la medida de tiempo con respecto al último flanco de la señal (E) que se ha contado.
- El que el tiempo a medir ($t_2 - t_1$) no sea un múltiplo entero del periodo T de la señal (E) a la que se le cuentan los flancos.
- El tiempo de ejecución de las instrucciones relacionadas con la medida del tiempo.

A continuación se muestra un ejemplo de los dos casos que presentan un mayor error (en valor absoluto) al utilizar este método para medir tiempo:



$$-T < \text{error medida} < +T$$

$$T_{CY} = 4T_{osc}$$

Conclusión: A la hora de medir un tiempo $t_2 - t_1$ dado hay que elegir la señal (E) a la que se le van a contar los flancos de modo que su periodo (T) sea despreciable frente al tiempo $t_2 - t_1$ a medir. Es decir, se debe cumplir que:

$$t_2 - t_1 \pm T \simeq t_2 - t_1$$

Ejemplo: $40 \cdot 10^{-3} + 8 \cdot 10^{-4} = 40,8 \cdot 10^{-3} \simeq 40 \cdot 10^{-3}$

$$40 \cdot 10^{-3} - 8 \cdot 10^{-4} = 39,2 \cdot 10^{-3} \simeq 40 \cdot 10^{-3}$$

Corolario: cuanto menor sea el valor del periodo T de la señal a la que se le cuentan los flancos, menor será el error que se pueda cometer al hacer una medida de tiempo

Los pasos a dar para medir el tiempo transcurrido entre dos instantes t_1 y t_2 son:

1º: Elegir un periodo de la señal a la que se le van a contar los flancos adecuado al tiempo $t_2 - t_1$ que se va a medir (\equiv dicho periodo debe ser despreciable frente al tiempo $t_2 - t_1$ a medir).

2º: Poner el contenido del *timer* a cero antes de iniciar la medida del tiempo.

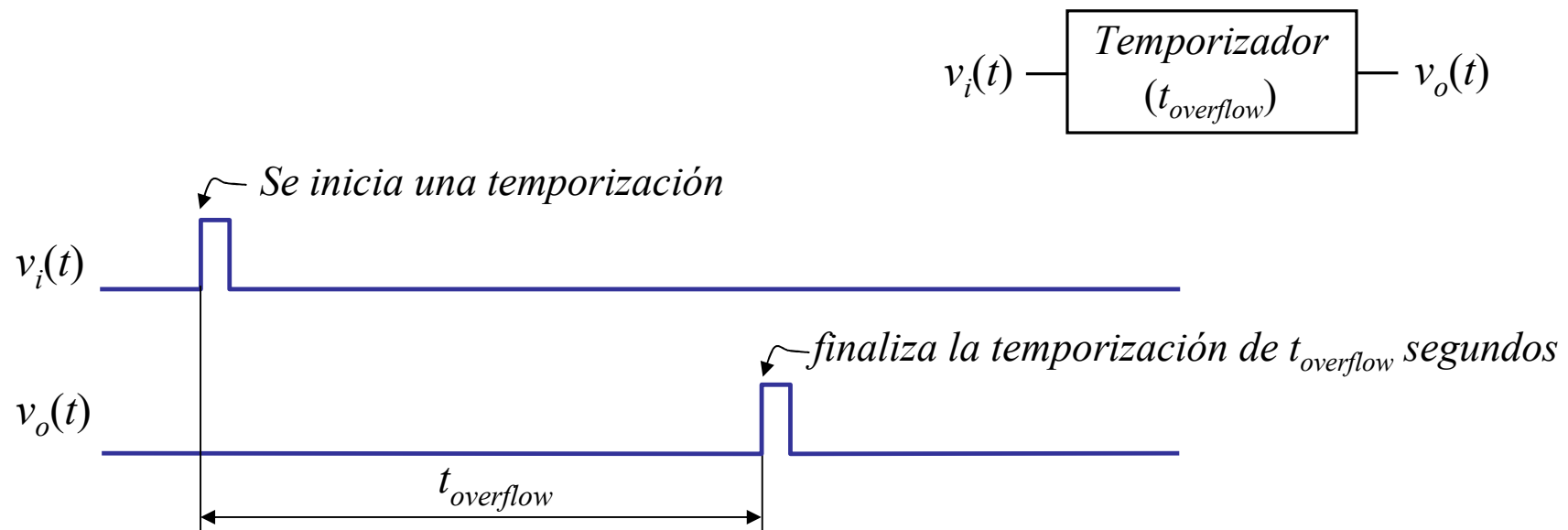
3º: Poner en funcionamiento el contador en el instante ($t = t_1$) en el que deba iniciarse la medida del tiempo. En el instante ($t = t_2$) en el que deba finalizar la medida del tiempo hay que inhibir el contaje o bien leer el contenido del contador.

4º: Para determinar el tiempo transcurrido entre t_1 y t_2 hay que leer el contenido (α) del contador y multiplicarlo por el periodo (T) de la señal a la que se le han contado los flancos de subida (o de bajada). Cumpliéndose que el tiempo transcurrido entre t_1 y t_2 será igual a $T \cdot \alpha$ ($t_2 - t_1 = T \cdot \alpha$).

- *Temporizador digital:*

El funcionamiento de un temporizador *no redisparable* se caracteriza porque su salida $[v_o(t)]$ indica que ha finalizado una temporización cuando ha transcurrido un tiempo dado ($t_{overflow}$) desde el instante en el que se ha iniciado dicha temporización.

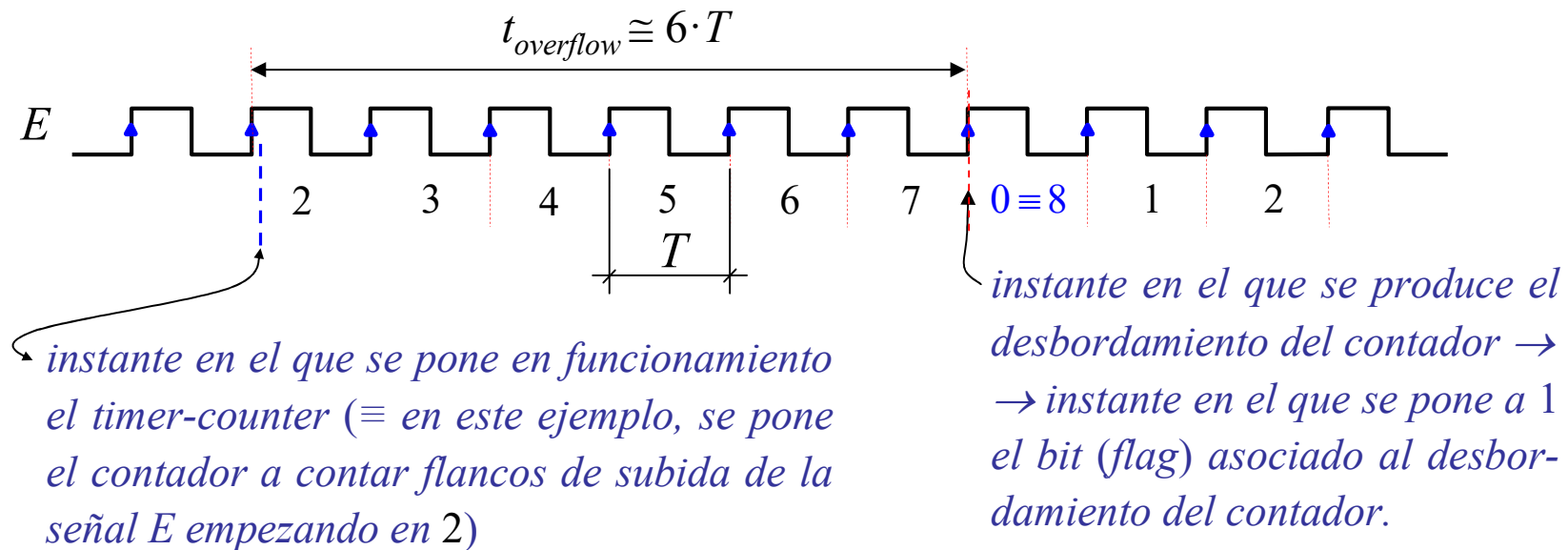
En la mayoría de los temporizadores, el tiempo que dura una temporización ($t_{overflow}$) se puede configurar (previamente) dentro de unos límites *máximo y mínimo* dados.



El concepto anterior de *temporizador no redispachable* se puede implementar utilizando un módulo *timer-counter* de un μC si se tiene en cuenta que cuando se produce un *desbordamiento del contador* se provoca una interrupción (¡en el caso de que esté habilitada!).

Ejemplo:

Nota: en este ejemplo se ha supuesto un contador de 3 bits



Nota: el tiempo que transcurre desde el instante en el que el timer-counter se pone en funcionamiento (\equiv se pone a contar flancos de subida de la señal E) hasta que se produce el desbordamiento del contador es aproximadamente igual a $6 \cdot T = (2^3 - 2) \cdot T$

De acuerdo con lo indicado en la página anterior, para configurar un *timer* para que temporice un tiempo dado ($t_{overflow}$) hay que hacer lo siguiente:

1º: *Elegir un periodo adecuado (T) de la señal a la que el timer-counter le va a contar los flancos de subida (cuanto más pequeño sea dicho periodo, menor será el error cometido en la temporización).*

2º: *Determinar el valor inicial (α) desde el que el contador debe comenzar a contar flancos de subida, para que transcurra un tiempo $t_{overflow}$ desde el instante en el que se ponga el contador a contar flancos hasta el instante en el que se produzca el desbordamiento del contador.*

3º: *Cargar en el contador el valor calculado (α) y habilitar la interrupción del timer*

4º: *Poner el contador a contar flancos (de subida) en el instante en el que se quiera iniciar una temporización. El μC se enterará de que ha transcurrido el tiempo a temporizar ($t_{overflow}$) porque el timer producirá una interrupción.*

Algo en lo que pensar: se puede medir el tiempo transcurrido entre dos instantes dados así como implementar un temporizador, sin tener que apagar nunca el módulo temporizador...

Nota: más adelante se explica en qué consiste el modo 16 bits en el Timer 0

Timer 0 (PIC18F452)

- Este módulo puede utilizar un contador de 8 bits o bien un contador de 16 bits para contar los flancos de subida de la señal que se haga llegar a su entrada. La elección se realiza por software (bit T0CON.T08BIT).
- El origen de la señal a la que se le cuentan los flancos de subida puede ser interno ($F_{osc}/4$) o externo (pin RA4).
- El contenido del contador del *Timer 0* se guarda en:
 - _ el registro TMR0L, cuando se utiliza un contador de 8 bits (modo 8 bits)
 - _ los registros TMR0H (byte alto) y TMR0L (byte bajo) cuando se utiliza un contador de 16 bits (modo 16 bits).
- El contenido de los registros TMR0H y TMR0L se puede modificar (escribir) y leer en cualquier momento. No siendo necesario detener el funcionamiento del *timer 0* ni para escribir ni para leer su contenido.

Nota: contenido del contador del *timer 0* \equiv contenido del *timer 0*

- El Timer 0 puede provocar una interrupción por ‘*overflow*’ (si se habilita):
 _ (modo 8 bits) la interrupción se produce durante la transición 0xFF→0x00 del contenido del contador.
 _ (modo 16 bits) la interrupción se produce durante la transición 0xFFFF→0x0000 del contenido del contador.
- El funcionamiento del *Timer 0* se configura y se controla por medio de los registros T0CON, INTCON, TMR0H y TMR0L.

TABLE 10-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
TMR0L	Timer0 Module Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISA	—	PORTA Data Direction Register							-111 1111	-111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

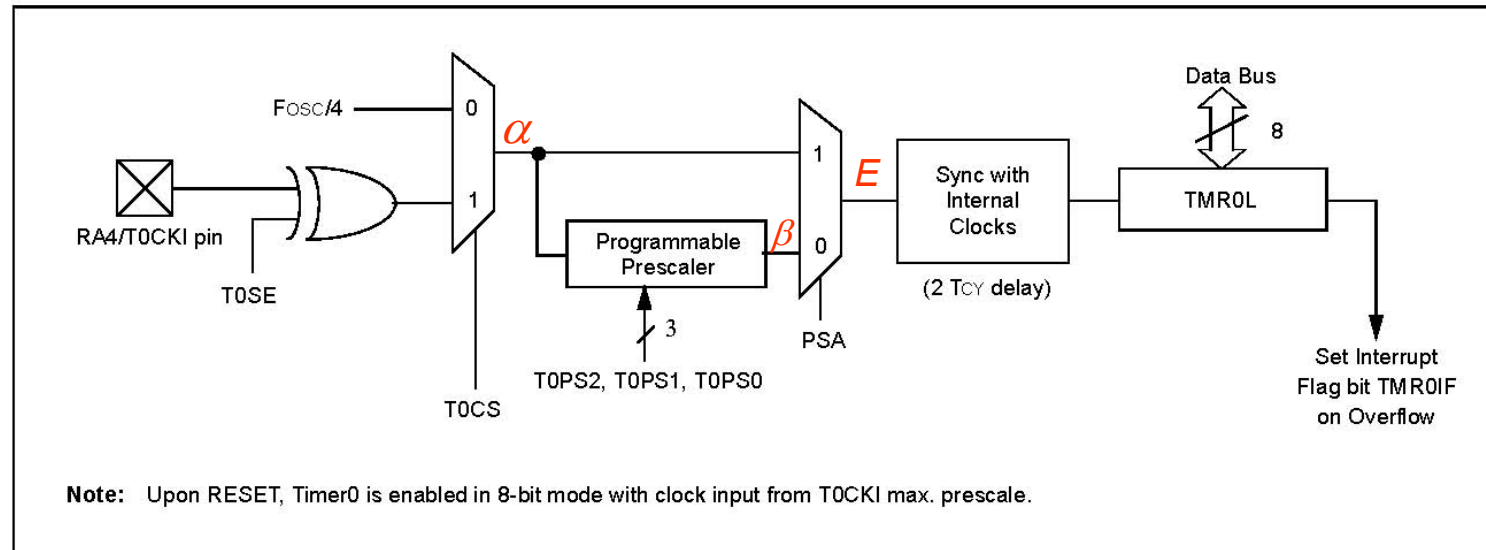
REGISTER 10-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

- bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0
- bit 6 **T08BIT:** Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter
- bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)
- bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin
- bit 3 **PSA:** Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.
- bit 2-0 **T0PS2:T0PS0:** Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

Timer 0 (modo 8 bits)

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE

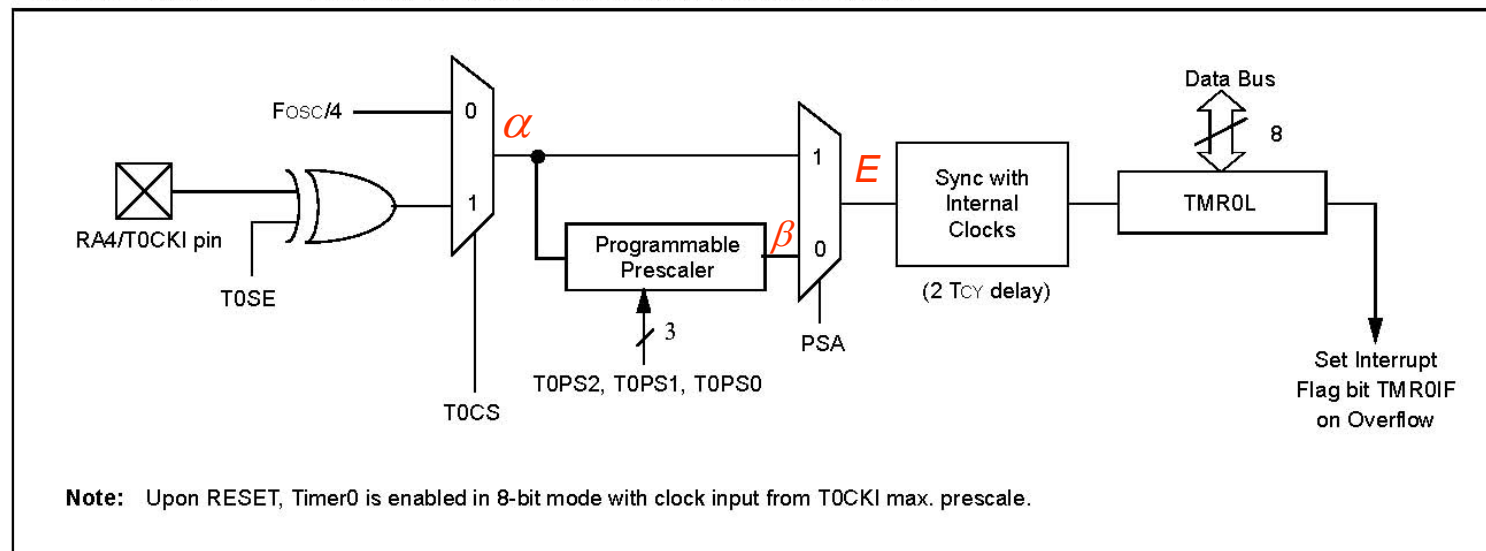


- Si se pone el bit T0CS a 0, entonces la señal a la que se le cuentan los flancos de subida procede de la señal de reloj del microcontrolador. La cual tiene un frecuencia igual a $F_{osc}/4$.
- Si se pone el bit T0CS a 1, entonces la señal a la que se le cuentan los flancos de subida procede de la señal aplicada al pin RA4 del microcontrolador.
- El contador cuenta los flancos de subida de la señal α en el caso de que se ponga a 1 el bit PSA (ver Figura 10-1). En el caso de que el bit PSA se ponga a 0, el contador cuenta los flancos de la señal β .

$$T_{CY} = 4T_{osc}$$

- La señal β se obtiene a partir de la señal α haciéndola pasar por un divisor de frecuencia (*Programmable Prescaler*). Los valores por los que se puede dividir la frecuencia de la señal α son: 2, 4, 8, 16, 32, 64, 128 y 256. El divisor elegido se indica por medio de los bits TOPS2, TOPS1 y TOPS0 del registro T0CON.
- El bit T0SE sirve para elegir si a la señal aplicada al terminal RA4 se le cuentan los flancos de subida (T0SE = 0) o de bajada (T0SE = 1)
- El contador cuenta los flancos de subida de la señal E . El contenido del contador (\equiv número de flancos contados) se guarda en todo momento en el registro TMR0L.

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



▪ Asumiendo que en el registro TMR0L se ha guardado un valor α a partir del cual el *Timer 0* comenzará a contar flancos de subida, el tiempo ($t_{overflow}$) que se tarda en producir el desbordamiento del contador a partir del instante en el que se ponga a contar flancos cumple lo siguiente:

H: Si la señal a la que se le cuentan los flancos de subida procede de la señal de reloj (f_{osc}) del μC , entonces se cumple lo siguiente:

$$t_{overflow} = (4/f_{osc}) \cdot prescaler \cdot (256 - \alpha)$$

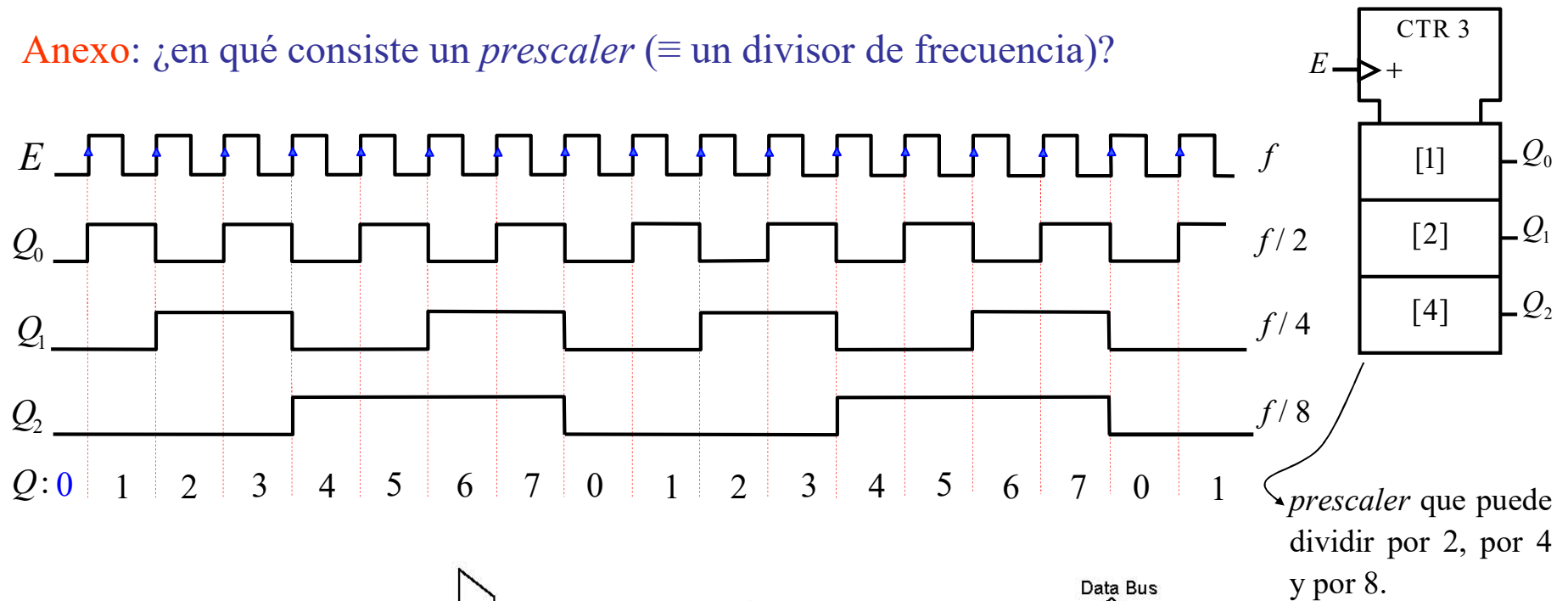
H: Si la señal a la que se le cuentan los flancos de subida procede de la señal aplicada al terminal RA4 del μC , entonces se cumple lo siguiente:

$$t_{overflow} = (1/f_{RA4}) \cdot prescaler \cdot (256 - \alpha)$$

Nota: ésta expresión sólo es útil si la señal aplicada a RA4 es periódica

siendo f_{RA4} la frecuencia de la señal aplicada al terminal RA4

Anexo: ¿en qué consiste un *prescaler* (\equiv un divisor de frecuencia)?



Nota: La señal α tiene un periodo $T_\alpha = 4 \cdot T_{osc}$

Si el bit $PSA = 0$, las señales E y E' tienen un periodo igual a $T = 4 \cdot T_{osc} \cdot \text{prescaler}$

(continuación de la página 59) En el caso de que la señal a la que se le cuentan los flancos de subida proceda de la señal de reloj del μC , el valor inicial (*alfa*) a guardar en el registro TMR0L cumple lo siguiente (ver diapositiva 59):

$$alfa = 256 - \frac{t_{overflow}}{4 \cdot T_{osc} \cdot prescaler} = 256 - \frac{t_{overflow} \cdot f_{osc}}{4 \cdot prescaler} \quad (alfa \lll 255)$$

prescaler : 1, 2, 4, 8, 16, 32, 64, 128, 256

Nota: Para hacer que el error cometido al realizar una temporización sea lo más pequeño posible es necesario que el periodo de la señal a la que se le cuentan los flancos de subida sea lo más pequeño posible.

Para que el periodo de la señal a la que se le cuentan los flancos de subida sea lo más pequeño posible hay que elegir el *prescaler* más pequeño posible. De la expresión anterior se deduce que cuanto menor sea el valor del *prescaler* elegido, menor será el valor el valor inicial (*alfa*) a guardar en el registro TMR0L.

Nota: el Timer 0 no puede ‘despertar’ el μC del modo *sleep*, debido a que está apagado durante dicho modo!.

Ejemplo: Configuración del timer 0 para que temporice 0.5 msec , siendo la frecuencia de reloj del microcontrolador $f_{osc} = 8\text{ MHz}$.

Solución

Utilizando los datos facilitados ($t_{overflow} = 0.5 \cdot 10^{-3}$, $f_{osc} = 8 \cdot 10^6 = 1/T_{osc}$) y probando con los distintos valores que puede tomar el *prescaler* del *Timer 0* en la siguiente expresión

$$alfa = 256 - \frac{t_{overflow}}{4 \cdot T_{osc} \cdot prescaler} = 256 - \frac{t_{overflow} \cdot f_{osc}}{4 \cdot prescaler} = 256 - \frac{0.5 \cdot 10^{-3} \cdot 8 \cdot 10^6}{4 \cdot prescaler}$$

se obtienen los resultados indicados en la página siguiente.



<i>prescaler</i>	<i>alfa</i>		$T = 4 \cdot T_{osc} \cdot \text{prescaler}$
1	-744	<i>no válido</i>	$0.5 \cdot 10^{-6}$
2	-244	<i>no válido</i>	10^{-6}
4	6	<i>válido</i>	$2 \cdot 10^{-6}$
8	131	<i>válido</i>	$4 \cdot 10^{-6}$
16	193.5	<i>no exacto</i>	$8 \cdot 10^{-6}$
32	224.75	<i>no exacto</i>	$16 \cdot 10^{-6}$
64	240.375	<i>no exacto</i>	$32 \cdot 10^{-6}$
128	248.1875	<i>no exacto</i>	$64 \cdot 10^{-6}$
256	252.09375	<i>no exacto</i>	$128 \cdot 10^{-6}$

$T = 4 \cdot T_{osc} \cdot \text{prescaler} = 4 \cdot (1/f_{osc}) \cdot \text{prescaler}$: es el periodo de la señal (*E*) a la que se le cuentan los flancos de subida.

alfa: es el valor que hay que cargar en el registro TMR0L. Es decir, es el valor desde el que el contador del *Timer* 0 comienza a contar flancos de subida de la señal (*E*) que llega a su entrada, la cual tiene un periodo $T = 4 \cdot T_{osc} \cdot \text{prescaler}$.

De los resultados anteriores se deduce que dos posibles soluciones son:

- $prescaler = 4$ y $alfa = 6$, siendo: $-2 \cdot 10^{-6} < error < +2 \cdot 10^{-6}$
- $prescaler = 8$ y $alfa = 131$, siendo: $-4 \cdot 10^{-6} < error < +4 \cdot 10^{-6}$

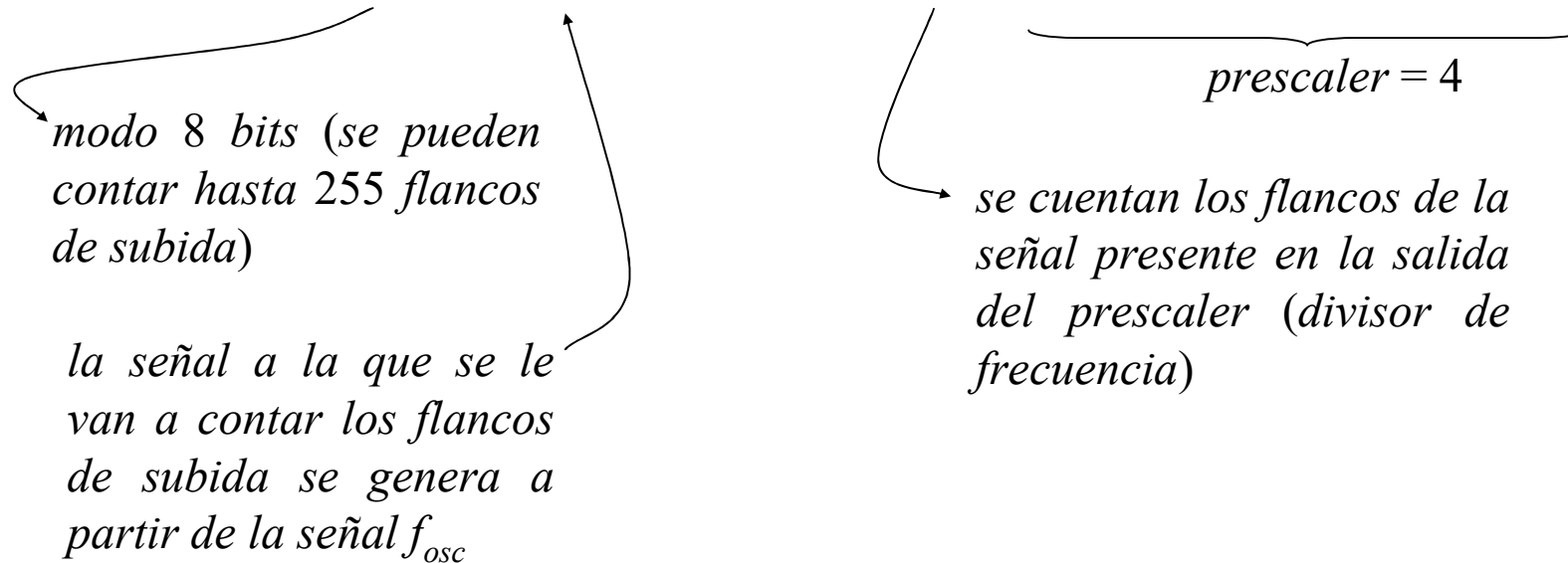
En la página siguiente se indican los valores a guardar en el registro T0CON correspondientes a la solución que da lugar a un menor error que está dada por:

- $prescaler = 4$
- $alfa = 6$ (\equiv valor inicial desde el que el contador debe comenzar a contar flancos de subida y que hay que guardar en el registro TMR0L antes de poner el contador a contar flancos).

Registro T0CON

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
--------	--------	------	------	-----	-------	-------	-------

TMR0ON	1	0	x	0	0	0	1
--------	---	---	---	---	---	---	---



Una vez guardado en el registro TMR0L el valor 6, desde que se pone el bit TMR0ON a 1 hasta que se produce el desbordamiento del contador transcurren 0.5 mseg.

Pregunta: Supón que has estado probando con diferentes frecuencias de reloj del μC y con diferentes *prescalers* para obtener un valor **entero**, lo más pequeño posible, a guardar en el registro TMR0L y que has llegado a la conclusión de que necesitas que el *prescaler* valga 1. ¿Cómo se puede lograr que el prescaler valga 1?

Pregunta: Supón que has estado haciendo cálculos para configurar el Timer 0 con el fin de que realice una temporización de $t_{overflow}$ segundos, y que no has podido obtener un valor **entero** a guardar en el registro TMR0L.

- i) ¿Qué puedes hacer en este caso?.
- ii) ¿Qué consecuencias tiene en la temporización el aproximar un valor con parte fraccionaria por el valor entero más próximo?

Nota: se pueden realizar temporizaciones sin tener que apagar nunca el Timer

Configuración de la interrupción del Timer 0 (por desbordamiento)

```
void interrupt() // rutina de servicio de la interrupción (MikroC)
```

```
{
```

```
.....
```

```
    INTCON.TMR0IF = 0; // se borra el flag
```

```
}
```

```
void main()
```

```
{
```

```
.....
```

```
    T0CON = ?; // se configura el Timer 0
```

```
    INTCON.TMR0IF = 0; // se pone el flag a 0
```

```
    INTCON.TMR0IE = 1; // se habilita la interrupción del Timer 0
```

```
    INTCON.GIE = 1; // se habilitan las interrupciones en general
```

```
    TMR0L = ?; //se carga el valor inicial (alfa) del ‘contador’
```

```
.....
```

```
    // ya se puede poner el Timer0 a realizar una temporización
```

```
.....
```

```
}
```

Muy importante: no se debe guardar un valor en el registro TMR0L antes de configurar el registro T0CON (ver página siguiente).

Importante: a la hora de configurar el *Timer 0* hay que tener en cuenta lo siguiente:

- Con $prescaler = \mu$ el contenido del *Timer 0* se incrementa cada $\mu \cdot 4 \cdot T_{osc} = \mu \cdot T_{CY}$
- Cada vez que ‘escribimos’ un valor en el registro *TMR0L* el contaje de los flancos de subida se inhibe durante $8 \cdot T_{osc} = 2 \cdot T_{CY}$ ($\equiv 2$ ciclos de instrucción) (si no se utiliza $f_{osc}/4$)
- El *prescaler* no es más que un contador de 8 bits que cuenta flancos (de subida o de bajada) de la señal que llega a su entrada. Cambiar el valor del *prescaler* (\equiv cambiar la salida del contador de 8 bits que se conecta a la entrada del contador del *Timer 0* para que este le cuente los flancos de subida) no hace que su contenido se ponga a cero (0).
- Cada vez que “nosotros” guardamos un valor en el registro *TMR0L*, el contenido del *prescaler* (contador de 8 bits) se pone a cero (0). De acuerdo con esto, *para que una temporización sea correcta primero hay que configurar/cambiar el prescaler y después hay que guardar en el registro TMR0L el valor desde el que el contador inicia el contaje.* Hacerlo al revés puede dar lugar a una temporización incorrecta.

Timer 0 (modo 16 bits)

- En este modo, el Timer 0 utiliza un contador de 16 bits para contar flancos de la señal que se haga llegar a su entrada. Esto implica que puede contar hasta $2^{16}-1 = 65535$ flancos de subida sin que se produzca un desbordamiento.
- Se tiene acceso al contenido del *contador* a través de los registros TMR0L y TMR0H. El orden de acceso a estos registros difiere según se trate de una operación de *lectura* o de *escritura* (ver diagrama de bloques en la siguiente página).

[illegible]

- $$x = x + (\text{TMR0H} \ll 8);$$


- TMR0L = 23497;

- La relación que existe entre el valor (*alfa*) guardado en los registros TMR0H y TMR0L y el tiempo ($t_{overflow}$) que tarda en producirse el desbordamiento del contador del *Timer* 0 a partir del instante en el que se pone a contar flancos (T0CON.TMR0ON = 1) cumple lo siguiente:

Н: Si la señal a la que se le cuentan los flancos de subida procede de la señal de reloj (f_{osc}) del μC , entonces se cumple lo siguiente:

$$t_{overflow} = (4/f_{osc}) \cdot prescaler \cdot (65536 - alfa)$$

Н: Si la señal a la que se le cuentan los flancos de subida procede de la señal aplicada al terminal RA4 del μC , entonces se cumple lo siguiente:

Nota: ésta expresión sólo es útil si
 la señal aplicada a RA4 es periódica

$$t_{overflow} = (1/f_{RA4}) \cdot prescaler \cdot (65536 - alfa)$$

siendo f_{RA4} la frecuencia de la señal aplicada al terminal RA4

En el caso de que la señal a la que se le cuentan los flancos de subida se genere a partir de la señal de reloj del μC ($f_{osc}/4$), el valor inicial *alfa* que hay que guardar en los registros TMR0H y TMR0L para que el *Timer* 0 tarde un tiempo $t_{overflow}$ en producir una interrupción (desde el instante en el que se pone en funcionamiento) cumple lo siguiente (ver diapositiva anterior):

$$alfa = 65536 - \frac{t_{overflow}}{4 \cdot T_{osc} \cdot prescaler} = 65536 - \frac{t_{overflow} \cdot f_{osc}}{4 \cdot prescaler} \quad (alfa \lll 65535)$$

prescaler: 1, 2, 4, 8, 16, 32, 64, 128, 256

$t_{overflow}$: tiempo que transcurre desde que el *Timer* 0 se pone a contar flancos de subida (empezando en el valor *alfa*) hasta que el contenido del contador realiza el cambio 0xFFFF \rightarrow 0x0000 (\equiv hasta que el *Timer* 0 produce una interrupción, si está habilitada)

Nota: para que el error que se pueda cometer al realizar una temporización sea el menor posible, el periodo de la señal a la que se le cuentan los flancos de subida debe ser el menor posible. Esto equivale a hacer que el valor inicial *alfa* a guardar en los registros TMR0H y TMR0L sea lo más pequeño posible.

Dicho de otro modo, cuanto menor sea el valor del *prescaler*, menor será el periodo de la señal a la que se le cuentan los flancos de subida y, por lo tanto, menor será el máximo error que se puede cometer al realizar la temporización.

Ejemplo: Configuración del timer 0 para que temporice 2 segundos, siendo la frecuencia de reloj del microcontrolador $f_{osc} = 8 \text{ MHz}$.

Solución

Utilizando los datos facilitados ($t_{overflow} = 2$, $f_{osc} = 8 \cdot 10^6 = 1/T_{osc}$) y probando con los distintos valores que puede tomar el *prescaler* del *Timer 0* (1, 2, 4, 8, 16, 32, 64, 128, 256) en la siguiente expresión

$$alfa = 65536 - \frac{t_{overflow}}{4 \cdot T_{osc} \cdot prescaler} = 65536 - \frac{t_{overflow} \cdot f_{osc}}{4 \cdot prescaler} = 65536 - \frac{2 \cdot 8 \cdot 10^6}{4 \cdot prescaler}$$

se obtienen los resultados indicados en la página siguiente.



<i>prescaler</i>	<i>alfa</i> (TMR0H-TMR0L)		$T = 4 \cdot T_{osc} \cdot \text{prescaler}$
1	-3934464	<i>no válido</i>	$0.5 \cdot 10^{-6}$
2	-1934464	<i>no válido</i>	10^{-6}
4	-934464	<i>no válido</i>	$2 \cdot 10^{-6}$
8	-434464	<i>no válido</i>	$4 \cdot 10^{-6}$
16	-184464	<i>no válido</i>	$8 \cdot 10^{-6}$
32	-59464	<i>no válido</i>	$16 \cdot 10^{-6}$
64	3036	<i>exacto</i>	$32 \cdot 10^{-6}$
128	34286	<i>exacto</i>	$64 \cdot 10^{-6}$
256	49911	<i>exacto</i>	$128 \cdot 10^{-6}$

$T = 4 \cdot T_{osc} \cdot \text{prescaler} = 4 \cdot (1/f_{osc}) \cdot \text{prescaler}$: es el periodo de la señal a la que se le cuentan los flancos de subida.

alfa : representa el valor que hay que cargar en los registros TMR0H y TMR0L \equiv es el valor desde el que el contador del *Timer* 0 comienza a contar los flancos de subida de la señal que llega a su entrada.

De los resultados anteriores se deduce que hay 3 posibles soluciones:

- $prescaler = 64$ y $alfa = 3036$, siendo: $-32 \cdot 10^{-6} < error < +32 \cdot 10^{-6}$
- $prescaler = 128$ y $alfa = 34286$, siendo: $-64 \cdot 10^{-6} < error < +64 \cdot 10^{-6}$
- $prescaler = 256$ y $alfa = 49911$, siendo: $-128 \cdot 10^{-6} < error < +128 \cdot 10^{-6}$

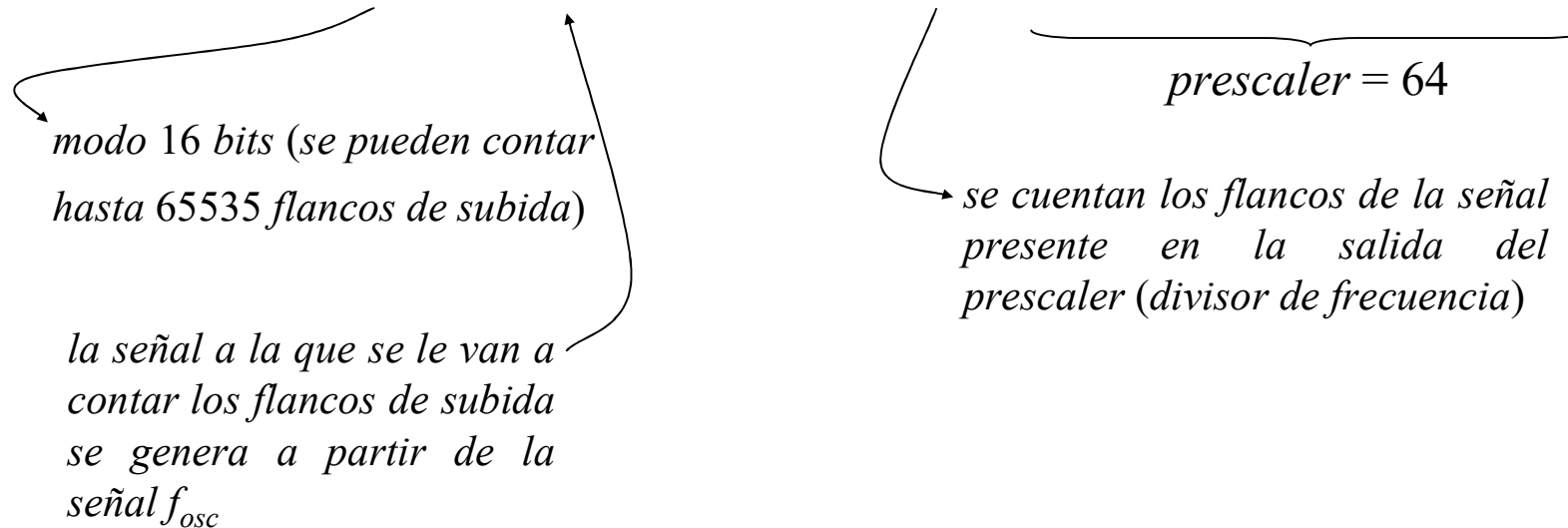
En la siguiente página se indican los valores a guardar en el registro T0CON correspondientes a la solución que da lugar a un menor error en la temporización:

- $prescaler = 64$
- $alfa = 3036 \equiv$ valor inicial desde el que el contador del *Timer 0* debe comenzar a contar flancos de subida. Dicho valor debe guardarse en los registros *TMR0H* y *TMR0L* antes de poner en funcionamiento el *Timer 0*.

Registro T0CON

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
--------	--------	------	------	-----	-------	-------	-------

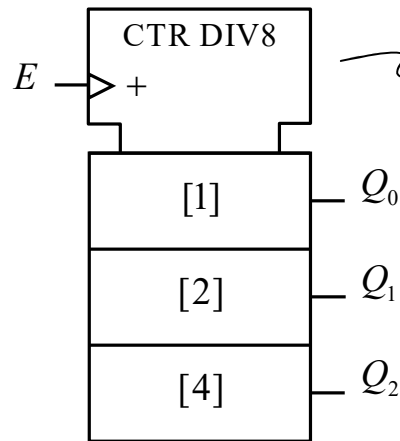
TMR0ON	0	0	x	0	1	0	1
--------	---	---	---	---	---	---	---



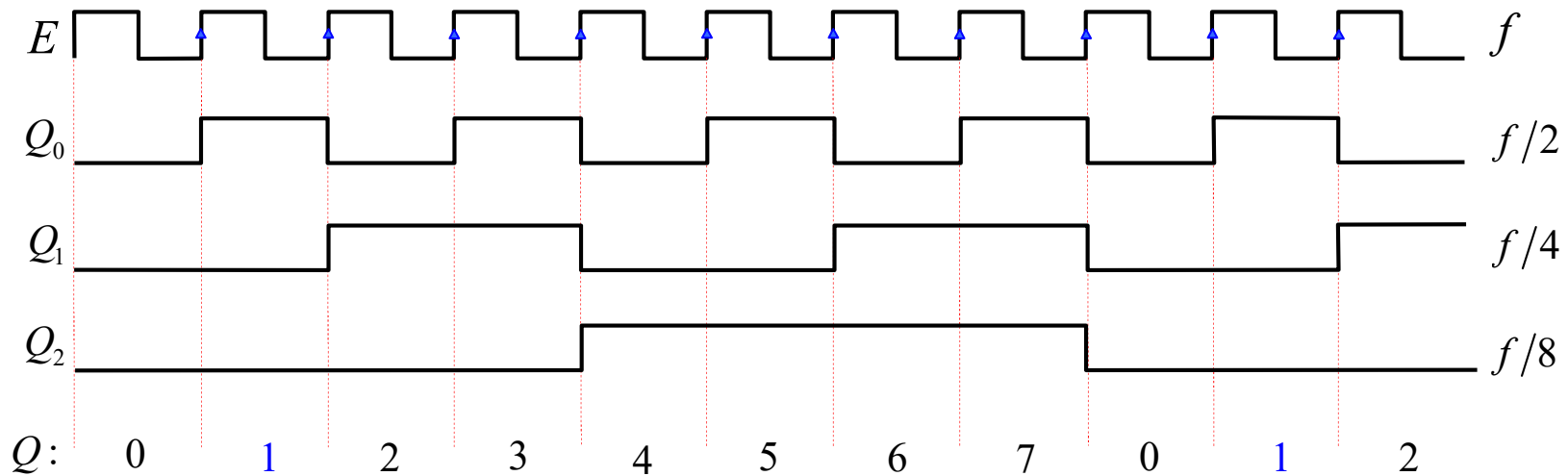
Una vez guardado en los registros TMR0H y TMR0L el valor alfa = 3036, desde que se pone el bit TMR0ON a 1 hasta que se produce el desbordamiento del contador transcurren 2 seg.

Importante: a la hora de configurar el *Timer 0* hay que tener en cuenta lo siguiente:

- Con $prescaler = \mu$ el contenido del *Timer 0* se incrementa cada $4 \cdot T_{osc} \cdot \mu = T_{CY} \cdot \mu$
- Cada vez que guardamos un valor en el registro *TMR0L*, el conteo de los flancos de subida se inhibe durante $8 \cdot T_{osc} = 2 \cdot T_{CY}$ ($\equiv 2$ ciclos de instrucción)
- El *prescaler* no es más que un contador de 8 bits que cuenta flancos (de subida o de bajada) de la señal que llega a su entrada. Cambiar el valor del *prescaler* (\equiv cambiar la salida del *prescaler* que se conecta a la entrada del contador del *Timer 0* para que este le cuente los flancos de subida) **no** hace que su contenido se ponga a cero (0).
- Cada vez que “nosotros” guardamos un valor en el registro *TMR0L*, el contenido del *prescaler* (contador de 8 bits) se pone a cero (0). De acuerdo con esto, *para que una temporización sea correcta cuando se utiliza el prescaler primero hay que configurar el prescaler y después hay que guardar en los registros TMR0H y TMR0L el valor desde el que se debe iniciar el conteo de los flancos de subida. Hacerlo al revés puede dar lugar a una temporización incorrecta.*



Este contador se puede utilizar como un *prescaler* que puede dividir la frecuencia de la señal (E) que le llega a su entrada por 2, 4 u 8, en función de la salida (Q_0 , Q_1 , Q_2) que se elija.



Nota: cambiar el valor del *prescaler* equivale a cambiar la salida del contador del *prescaler* que se hace llegar a la entrada del contador del *Timer 0*. El problema está en que el contenido del contador del *prescaler* no se pone a cero al realizar el cambio de salida.

Tipos de datos admitidos por el compilador MikrocPRO de Mikroelektronika

Tipo	Tamaño	Rango de valores
(unsigned) <i>char</i>	8 bits	0 ··· 255
signed <i>char</i>	8 bits	-128 ··· +127
unsigned short (<i>int</i>)	16 bits	0 ··· 65535
(signed) short (<i>int</i>)	16 bits	-32768 ··· +32767
(signed) <i>int</i>	32 bits	-2147483648 ··· +2147483647
unsigned (<i>int</i>)	32 bits	0 ··· 4294967295
(signed) long (<i>int</i>)	64 bits	-9223372036854775808 ··· +9223372036854775807
unsigned long (<i>int</i>)	64 bits	0 ··· 18446744073709551615
<i>float</i>	32 bits	$\pm 1.17549435082 \cdot 10^{-38}$ to $\pm 6.80564774407 \cdot 10^{38}$
<i>double</i>	64 bits	$\pm 1.17549435082 \cdot 10^{-38}$ to $\pm 6.80564774407 \cdot 10^{38}$
long double	128 bits	$\pm 1.17549435082 \cdot 10^{-38}$ to $\pm 6.80564774407 \cdot 10^{38}$