

Microcontroladores

Semana 3

Semestre 2025-2

Profesor: Kalun José Lau Gan

Preguntas previas:

- Tengo errores al momento de realizar el primer ejemplo con el MPLAB X v6.15, me arroja “lexical error”

```
movlw 0x10  
movlw A0H  
movlw 0A0H ✓
```

Muy probablemente error de formato de un número declarado en el código XC8 PIC Assembler

- ¿Cuál es la diferencia entre BRA y GOTO?
 - BRA saltos cortos (ocupa 2 bytes al usarlo)
 - GOTO salto largos (ocupa 4 bytes al usarlo)

Preguntas previas:

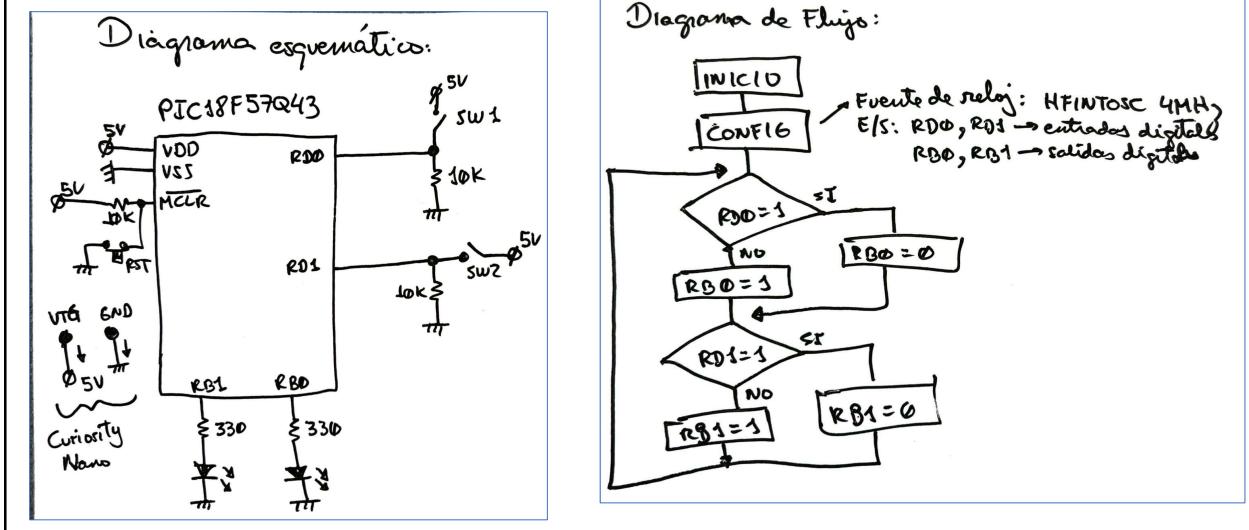
- He visto videos de Youtube y algunos libros (el del pic16f84 supongo de José Angulo Usategui) donde hay algunas discrepancias en la representación de números en el Assembler
 - No hay discrepancias, un número en cualquiera de sus representaciones numéricas (hex, bin o dec) siempre es el mismo valor, muy posible que sea por la mala interpretación de las instrucciones empleadas.
 - movlb -> rango de 0 a 63
 - movlw -> rango de 0 a 255
 - lfsr -> rango de 0 a 16383
- ¿Es mejor emplear el ASM para cuando use periféricos como I2C, SPI, UART frente al XC8? Porque estuve haciendo unas pruebas de esos módulos en ASM y no me salía
 - No te debe de salir por no configurar correctamente los módulos, pero se recomienda utilizar lenguaje de alto nivel ya que luego de emplear esos módulos muy posiblemente necesites hacer operaciones matemáticas (escalamiento, filtrado, promediacióñ, control de errores, etc) el cual de hacerlo en ASM te va a demorar demasiado tiempo en implementarlo, cosa que en XC8 alto nivel sería mas rápido de hacer ya que puedes hacer operaciones aritméticas complejas y de precisión con float.

Preguntas previas:

- No me compila el proyecto en el MPLABX y no sale donde esta el error en la ventana de output
 - Muy posiblemente se ha colocado caracteres restringidos en el nombre y/o en la ruta del proyecto.
 - Ej. Semana_245 <- No debe de colocarse el punto en el nombre
 - Ej. Semana2_3 <- De preferencia no dejar espacios, rellenarlo con guion bajo
- No esta habilitado el botón de grabación del microcontrolador en el MPLABX
 - No has conectado el Curiosity Nano, o el cable no tiene capacidad de transferencia de datos
 - Te has olvidado de escoger la herramienta (propiedades del proyecto: Tool)
- No se colorean las palabras en el código fuente.
 - Primero deben de grabar todos los archivos fuente presionando el botón de diskette.

Preguntas previas:

- ¿Puede mostrarnos el diagrama esquemático y el diagrama de flujo de la asignación de la semana 2?



Preguntas previas

- Cantidad de vistas de las grabaciones de clase de los laboratorios de la semana 2:

Analytics

Total views: 14

Last viewed: April 14th, 2025 5:50:44 AM

[Close](#)

Analytics

Total views: 8

Last viewed: April 14th, 2025 8:51:50 AM

[Close](#)

Preguntas previas 2025-2

- En el MPLAB X hay alguna forma de saber el valor asignado a una variable?
 - En lenguaje Assembler (XC8 PIC Assembler) no se manejan variables, solo valores asignados a registros.

Agenda:

- Estructura interna de un microcontrolador
- El módulo de oscilador
- Memoria de programa del microcontrolador PIC18F57Q43
- Memoria de datos del microcontrolador PIC18F57Q43
- Instrucciones básicas en XC8 PIC Assembler para el PIC18F57Q43
- ~~El contador de programa (PC) del CPU del microcontrolador PIC18F57Q43~~
- ~~Acceso a datos almacenados en la memoria de programa empleando el puntero de tabla (TBLPTR)~~
- ~~Acceso mediante punteros FSRx/INDFx a la memoria de datos~~
- ~~Interface a display de siete segmentos~~
- ~~Instrucciones de comparación numérica en XC8 PIC Assembler~~

¿Y el Raspberry Pi qué cosa es, un microprocesador o un microcontrolador?

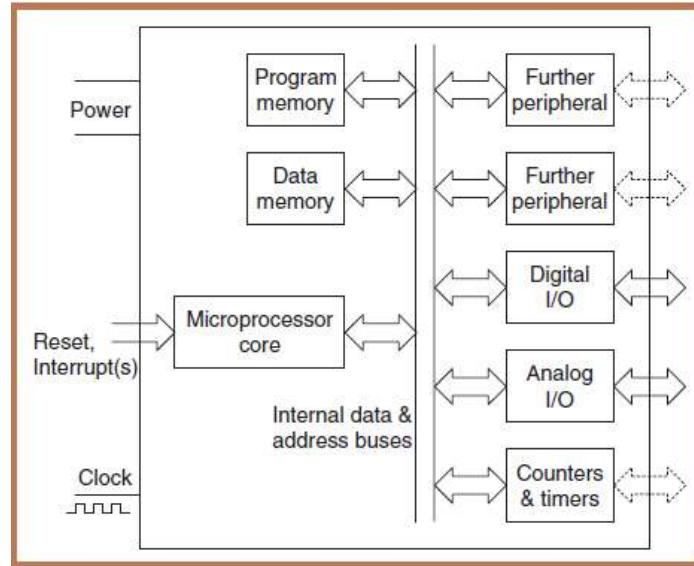


- No es un microcontrolador, tampoco es un microprocesador!
- Computador de Placa Reducida (SBC – Single Board Computer)

¿Cuál es la diferencia entre microprocesador y microcontrolador?

- **El microprocesador** es la unidad central de proceso de un computador o un microcontrolador.
- **El microcontrolador** contiene todos los elementos para un funcionamiento autónomo, es decir, contiene un microprocesador, memorias y dispositivos de E/S.

Estructura interna de un microcontrolador (introducción a la arquitectura de computadores)



- CPU: Encargado de ejecutar las instrucciones
- Soporte del CPU: módulo de reloj, dispositivos de seguridad, etc.
- **Memoria de programa:** No volátil, almacena programa y datos constantes
- **Memoria de datos:** Volátil, almacena datos temporales y contiene los registros SFR
- Entradas y salidas
- Periféricos

¿Por qué la memoria de datos siempre es del tipo volátil?

- La memoria de datos tiene que equiparar la velocidad de trabajo del CPU. La RAM es la que puede llegar a cumplir dicho requerimiento.



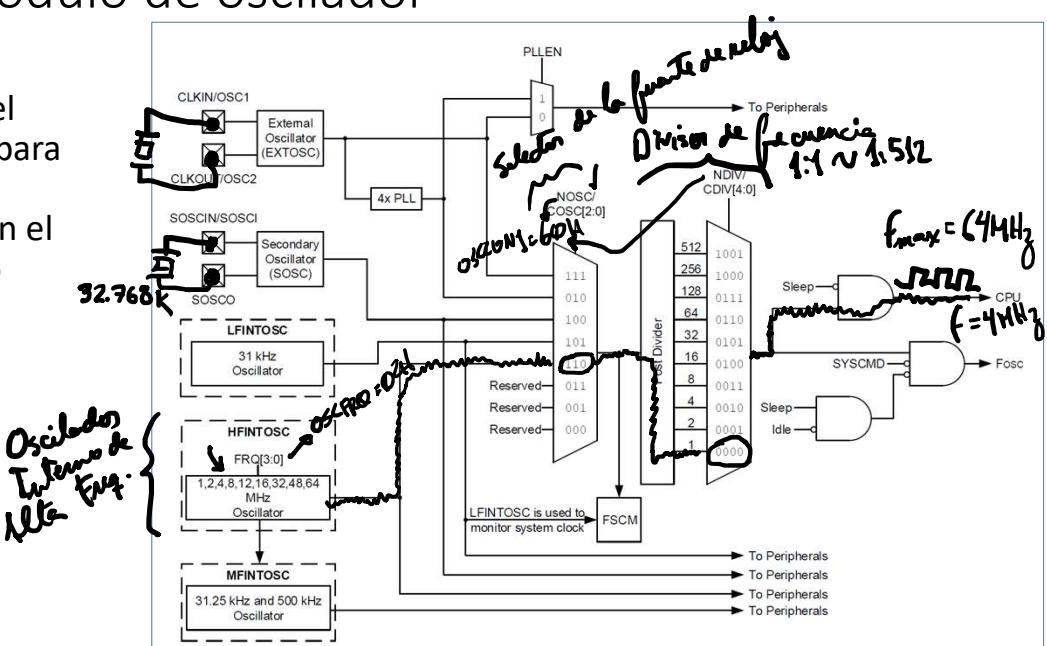
Lectura:

Hoja técnica (datasheet) del PIC18F57Q43

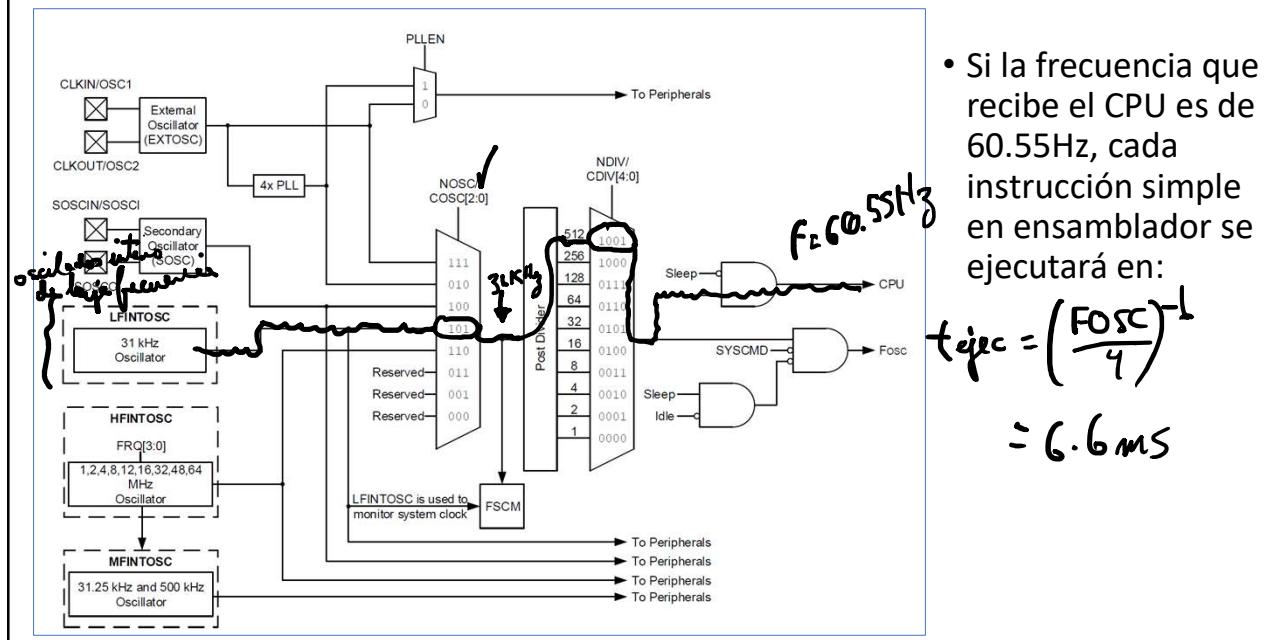
- Capítulo 12 – Módulo de oscilador
- Capítulo 19 – Las entradas y salidas
 - Revisar también el capítulo 21 relacionado con el PPS
- Capítulo 09 – Organización de la memoria

El módulo de oscilador

- Nosotros usaremos el HFINTOSC para nuestros ejemplos en el laboratorio

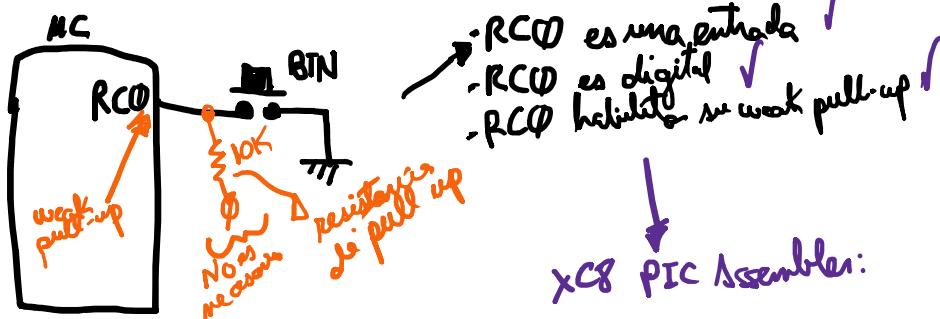


¿Cuál es la mínima frecuencia posible?



Recordando la configuración de los puertos de E/S:

- Definir las configuraciones al siguiente caso:



Según CLD: Los entradas no deben de estar al aire!

xc8 PIC Assembler:

```
bsf TRIS, 0, 1
bcf ANSEL, 0, 1
bsf WPUC, 0, 1
```

El Microcontrolador PIC18F57Q43 de Microchip

- Referencia: Capítulo 09 de hoja técnica
- Estructura de la memoria de **programa** del PIC18F57Q43:
 - 128Kbyte de capacidad (000000H-01FFFFH)
 - Data EEPROM (1Kbyte) se encuentra mapeado en 380000H
 - Bits de configuración están mapeadas en 300000H – 300009H
 - Rango de direcciones: 000000H-3FFFFH (22 bits – 4Mbyte de direcciones)
- Recordar que la dirección 000000H es el vector de RESET

Address	Memory
00 0000h to 00 3FFh	Program Flash Memory (64 KW) ⁽¹⁾
00 4000h to 00 7FFFh	
00 8000h to 00 FFFFh	
01 0000h to 01 FFFFh	
02 0000h to 1F FFFFh	Not Present ⁽²⁾
20 0000h to 20 003Fh	User IDs (32 Words) ⁽³⁾
20 0040h to 2B FFFFh	Reserved
2C 0000h to 2C 00FFh	Device Information Area (DIA) ⁽⁴⁾
2C 6100h	Reserved
2F FFFFh	
30 0000h to 30 009h	Configuration Bytes ⁽⁵⁾
30 00A0h to 37 FFFFh	Reserved
38 0000h to 38 03Fh	Data EEPROM (1024 Bytes)
38 0400h to 3B FFFFh	Reserved
3C 0000h to 3C 009h	Device Configuration Information
3C 00A0h to 3F FFFFh	Reserved
3F FFFFh to 3F FFFFh	Revision ID (1 Word) ^(3,4,5)
3F FFFFh to 3F FFFFh	Device ID (1 Word) ^(3,4,5)

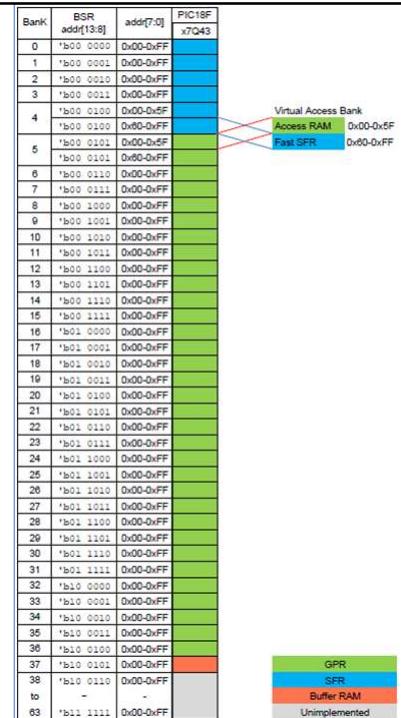
Notes:

1. Storage Area Flash is implemented as the last 128 Words of User Flash, if enabled.
2. The addresses do not roll over. The region is read as "0".
3. Not code-protected.
4. Hard-coded in silicon.
5. This region cannot be written by the user and it is not affected by a Bulk Erase.

El Curiosity Nano PIC18F57Q43 de Microchip

- Estructura de la memoria de **datos** del PIC18F57Q43:
 - Ancho de 8 bits
 - Memoria del tipo volátil (se borra el contenido en un PoR – Power-on Reset), **agrupado en bancos**.
 - A diferencia del PIC18F45K50, la memoria RAM (GPR) esta mapeada a partir del Bank 5 (500H) y los registros de funciones especiales (SFR) se encuentran entre Bank 0 y Bank 4
 - Los SFR son los registros que permiten configurar algún recurso del microcontrolador (ej fuente de reloj, manipulación de las E/S)
 - Tener en cuenta que la RAM de datos es de 8Kbyte
 - El rango total de direcciones: 0000H-3FFFH (14 bits - 16384 direcciones)

b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
1	0	0	1	0	1	0	0	0	0	0	0	0	0	2500H
1	0	0	1	0	1	1	1	1	1	1	1	1	1	2400H 24FFH



La importancia de escoger el banco correcto en la memoria de datos

- Los registros ó direcciones en la memoria de datos se encuentran clasificados en bancos (Bank), son los bits mas significativos (bit 8 al bit 13) del valor de dirección.
- Es importante seleccionar el banco correcto antes de manipular un registro en la memoria de datos.
- Por ejemplo: Deseo manipular el puerto RE0 como salida digital

ejemplo:

```

movlb 4H      ;me voy al Bank4
bcf TRISE, 0, 1 ;RE0 sea salida
bcf ANSELE, 1, 1 ;RE0 sea digital

```

Recordando: Estructuras anidadas

En el lenguaje C:

```

unsigned char x-var = 0;           ↑ 8 bits
for (x-var = 0; x-var < 10; x-var++) {
    }                                } se va a repetir 10 veces
                                         ↑ límite → 255

```

Cómo hago para hacer mas repeticiones teniendo como límite el tipo de variable?

```

unsigned char x-var=0, y-var=0;
for (x-var=0; x-var<2000; x-var++) {
    for (y-var=0; y-var<2000; y-var++) {
        }                                } ¿Cuántos veces se repite? → 4000000
    }
}

```

Recordando saltos a sub-rutinas

```

ORG 000000H
bra 000100H

ORG 000100H
bsf LATD, 0, 1
call 000200H
bcf LATD, 0, 1
call 000200H
bra 000100H

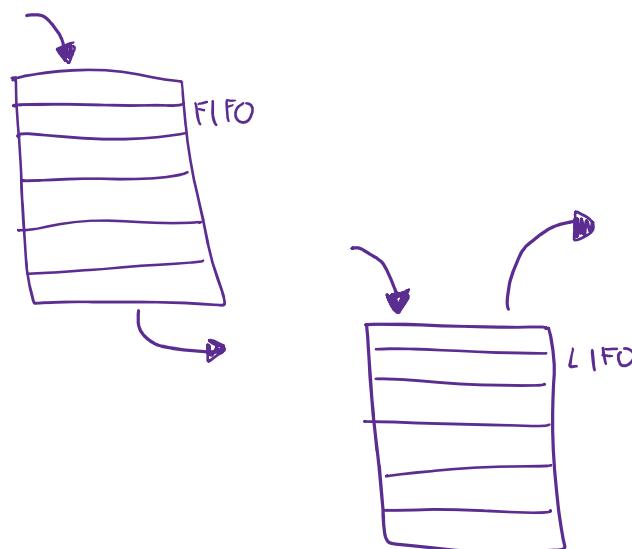
ORG 000200H
nop
nop
nop
return
    
```

;salto a subrutina

- Las subrutinas se usan para ejecutar en mas de una oportunidad una serie instrucciones. De esta manera se economiza espacio en la memoria de programa.
- ¿Cómo sabe el CPU que debe de retornar de donde saltó?
 - Existe una memoria el cual almacena la dirección de retorno y que le va a permitir al CPU regresar una vez atendido la eventualidad.
 - Dicha memoria se le conoce como “stack” o de pila (acción de apilar).

Recordando: memorias “stack” o de pila (apilamiento)

- Estructura FIFO
(first in – first out)
- Estructura LIFO
(last in – first out)



Aspectos relacionados con el obsoleto MPASM y el actual XC8 PIC Assembler

- El año 2014 Microchip dejó de lado el MPASM para dar lugar al XC8 PIC Assembler (PIC-AS)
- Nuevos diseños deberán emplear XC8 PIC Assembler en lugar de MPASM.
- XC8 PIC Assembler es un lenguaje de bajo nivel (orientado a la máquina), **nosotros debemos de conocer primero cómo funciona la máquina** para luego hacer que funcione mediante la codificación de un programa en Assembler y dar solución al problema planteado.

MPASM vs XC8 PIC Assembler: Partes de un programa

MPASM:

```

list p=18f4550
#include<p18f4550.inc>

; aqui declaramos los bits
CONFIG FOSC = XT_XT
CONFIG PWRT = ON
CONFIG BOR = OFF
CONFIG WDT = OFF
CONFIG PBADEN = OFF
CONFIG LVP = OFF

org 0x0000
goto configuracion

org 0x0020
configuracion:
    bsf TRISB, 0
    bcf TRISD, 0

principal:
    btfss PORTB, 0
    goto principal
    btg LATD, 0

otro:
    btfsc PORTB, 0
    goto otro
    goto principal

end
  
```

Labels

Directivas

Configuraciones

Programa

XC8 PIC ASM:

```

PROCESSOR 18F4550
#include "cabecera.inc"

PSECT rstVect,class=CODE, reloc=2, abs
ORG 0000H

rstVect:
    goto configuracion
    ORG 0020H

configuracion:
    bsf TRISB, 0
    bcf TRISD, 0

principal:
    btfss PORTB, 0
    goto principal
    btg LATD, 0, 0

otro:
    btfsc PORTB, 0
    goto otro
    goto principal

end rstVect
  
```

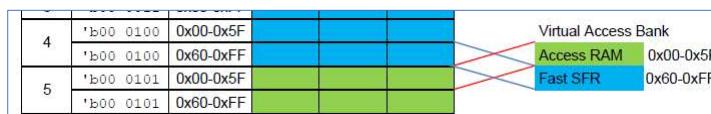
Nota: Los bits de configuración se alojaron en un archivo header llamado "cabecera.inc"

¿Por qué aprender Assembler (más difícil) si en el C también se puede hacer?

- En el assembler se prioriza en la eficiencia (espacio empleado en la memoria de programa y de datos, control detallado del consumo energético y del funcionamiento en general)
- En el C se prioriza en menor tiempo de desarrollo
- Hay que recordar que el uso de lenguaje de alto nivel siempre será menos eficiente frente al assembler (va a ocupar mas espacio, lo vas a ver como caja negra al microcontrolador)

Access-Bank vs BSR

- Son las formas para acceder a la memoria de datos
- **Access-Bank** es una forma directa (pero limitada) de acceder a la memoria datos, se usa para tener acceso a una porción del Bank4 (460H-4FFH) y una porción de memoria RAM (500H-55FH) y evitar estar cambiando de bancos de manera frecuente



- **BSR (Bank Select Register)** es la forma estándar para acceder a la memoria de datos, previamente se tiene que seleccionar el banco de entre 0 y 63 usando el registro selector de bancos BSR ó usando directamente la instrucción “movlb”

Repertorio de instrucciones en Assembler del PIC18

- Revisar capítulo 44 de la hoja técnica del PIC18F57Q43

Table 44-2. Standard Instruction Set							
Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word			Status Affected	Notes
			MSb		Lsb		
BYTE-ORIENTED FILE REGISTER INSTRUCTIONS							
ADDWF	f, d, a Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N
ADDWF	f, d, a Add WREG and Carry bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N
ANDWF	f, d, a And WREG with f	1	0001	01da	ffff	ffff	Z, N
CLRF	f, a Clear f	1	0110	10la	ffff	ffff	Z
COMF	f, d, a Complement f	1	0001	11da	ffff	ffff	Z, N
DECf	t, d, a Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N
INCF	t, d, a Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N
IORWF	t, d, a Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N
MOVF	f, d, a Move f to WREG or f	1	0101	00da	ffff	ffff	Z, N
MOVF	f ₀ , f _d Move f ₀ (12-bit source) to f _d (12-bit destination)	2	1100	f ₁ f ₀ ,f _d	f ₁ f ₀ ,f _d	f ₁ f ₀ ,f _d	None
MOVFF	f ₀ , f _d Move f ₀ (14-bit source) to f _d (14-bit destination)	3	1111	f ₃ f ₂ f ₁ f ₀	f ₃ f ₂ f ₁ f ₀	f ₃ f ₂ f ₁ f ₀	None
MOVF	f, a Move WREG to f	1	0110	11la	ffff	ffff	None
MULWF	f, a Multiply WREG with f	1	0000	001a	ffff	ffff	None
NEGF	f, a Negate f	1	0110	11da	ffff	ffff	C, DC, Z, OV, N
RLCF	t, d, a Rotate Left through Carry	1	0011	01da	ffff	ffff	C, Z, N
RNCF	t, d, a Rotate Left through (No Carry)	1	0010	00da	ffff	ffff	Z, N
RNRCF	t, d, a Rotate Right through Carry	1	0011	00da	ffff	ffff	C, Z, N
RNRCF	t, d, a Rotate Right through (No Carry)	1	0100	00da	ffff	ffff	Z, N
SETF	f, a Set f	1	0110	10la	ffff	ffff	None
SUBWF	f, d, a Subtract from WREG with Borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N
SUBWF	f, d, a Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N
SUBWBF	f, d, a Subtract WREG from with Borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N
SWAPP	f, d, a Swap nibbles in f	1	0011	10da	ffff	ffff	None
XORWF	f, d, a Exclusive OR WREG with f	1	0011	10da	ffff	ffff	Z, N

Repertorio de instrucciones en Assembler del PIC18

- Revisen las instrucciones que se han empleado en el ejemplo de la semana pasada con esta tabla de instrucciones:

- o movlb -> seleccionar el banco de trabajo, ej movlb 0H ;vas al Bank0
 - o movlw -> mover un literal hacia el registro W, ej movlw 60H
 - o movwf -> mover el contenido de W hacia un registro (memoria de datos)
 - o bsf -> setear un bit de un registro (poner a uno lógico)
 - o bcf -> resetear un bit de un registro (poner a cero lógico)
 - o btfss -> preguntar si un bit de un registro es igual a uno
 - o bra -> salto incondicional, ej bra inicio

Detalle de una instrucción con opción {a}

- Ejemplo:

Trabajando con Access bank:
movwf TRISB, 0

Trabajando con BSR:
(previamente especificar en el BSR cuál es el banco de acceso)

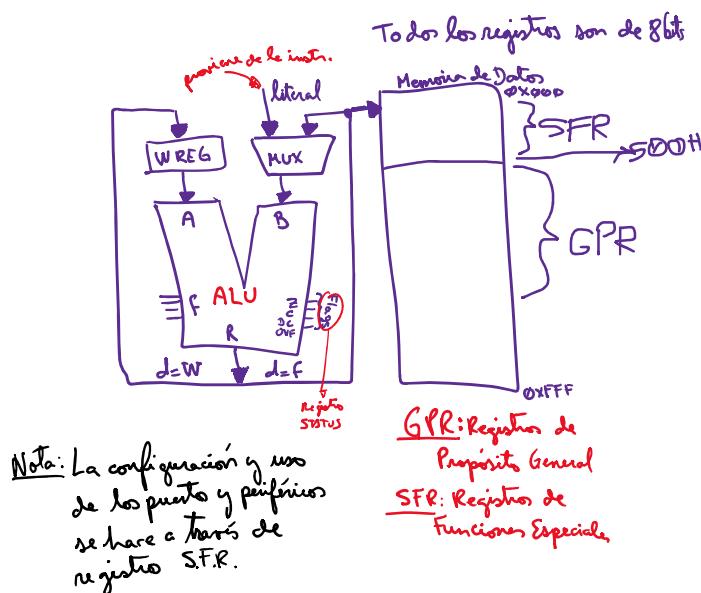
movwf TRISB, 1

movwf TRISB, a
movwf TRISB, b

- No todas las instrucciones tienen el parámetro {a}, revisar capítulo 44 del datasheet

MOVWF		Move W to f							
Syntax	MOVWF f { ,a }								
Operands	$0 \leq f \leq 255$ $a \in [0, 1]$								
Operation	$(W) \rightarrow f$								
Status Affected	None								
Encoding	0110 111a ffff ffff								
Description	Move data from W to register f. Location f can be anywhere in the 256-byte bank. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \leq 95$ (5Fh). See Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode for details.								
Words	1								
Cycles	1								
Q Cycle Activity:									
Q1	Q2	Q3	Q4						
Decode	Read W	Process Data	Write register 'f'						
Example: MOVWF REG, 0									
Before Instruction									
W = 4Fh									
REG = FFh									
After Instruction									
W = 4Fh									
REG = 4Fh									

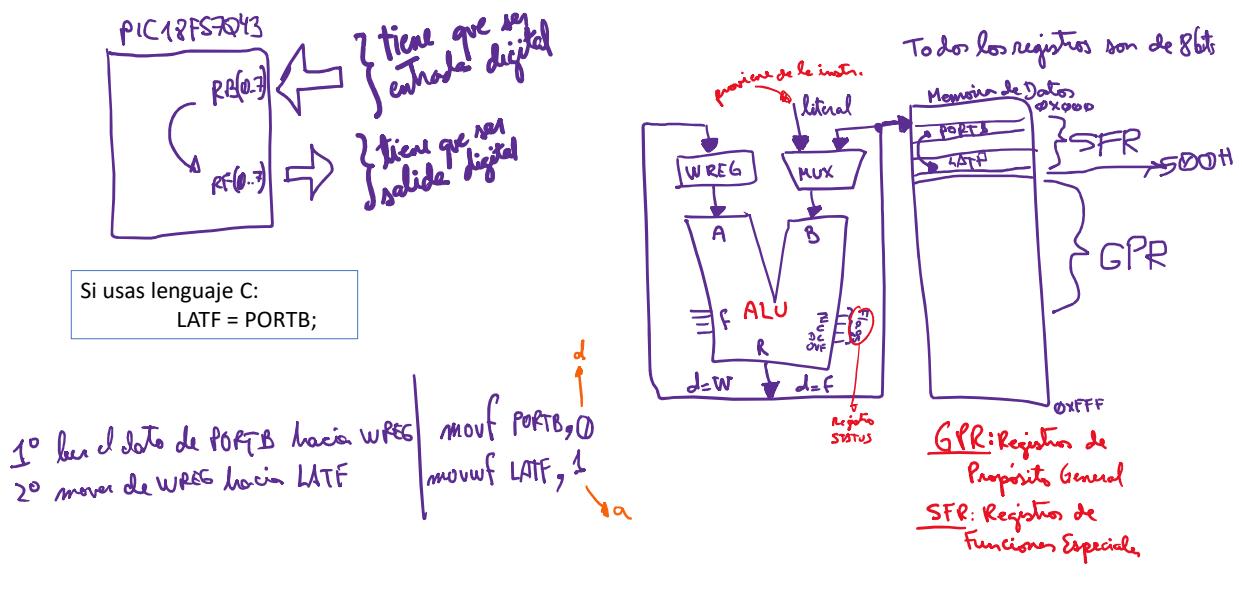
El flujo de datos en el microcontrolador PIC18F57Q43



Comentarios y Notas:

- En los programas desarrollados en XC8 PIC Assembler, la mayor cantidad de instrucciones serán las de movimiento de datos.
 - Es preferible trabajar con BSR (a=1), debido a que en el modo Access Bank solo se tiene una parte del registros del Bank4 y una parte de GPR del Bank5, si tu quieres emplear algún otro registro fuera de ese grupo, igual tendrás que el BSR.

Ejemplo: Pasar un dato del puerto B al puerto F



Registro STATUS

- Encuentras las banderas de la ALU, se actualizan cuando ocurre alguna operación aritmética ó lógica en este dispositivo.

7.7.7 STATUS								
Name:	STATUS							
Address:	0x4D8							
STATUS Register								
Bit	7	6	5	4	3	2	1	0
Access		TO	PD	N	OV	Z	DC	C
Reset	1	1	R	0	0	0	0	0

de la NTJ

- Bandera "N": Cuando en una operación que realiza la ALU el resultado fué un número negativo
 Bandera "OV": Cuando ocurre un desbordamiento del dato que se ha incrementado
 Bandera "Z": Cuando en una operación que realiza la ALU el resultado salió cero (00000000B ó 00H ó 0D)
 Bandera "DC": El digit carry
 Bandera "C/~B": Bit carry/~borrow, empleado en las operaciones aritméticas de suma y resta

Instrucciones básicas en XC8 PIC Assembler

- movlb -> para establecer el banco donde se va a trabajar
- movlw -> para mover un literal hacia el registro Wreg
- movwf -> para mover contenido del Wreg hacia un registro
- movff -> para mover el contenido de un registro a otro registro
- bsf, bcf -> para colocar 1 ó 0 a un bit (7-0) de un registro
- btfss, btfsc -> para probar si un bit es uno ó cero
- nop -> no operación (pierde el tiempo según t_ejec)
- decf, incf -> para decrementar/incrementar el valor de un registro
- incfsz, decfsz -> incremento/decremento con pregunta si llegó a cero
- setf, clrf -> coloca todos a uno/cero los bits de un registro
- comf -> complementa todos los bits de un registro
- bra, goto -> saltos, bra = cortos, goto = largos
- call, return -> saltos a subrutina (con opción a retornar)

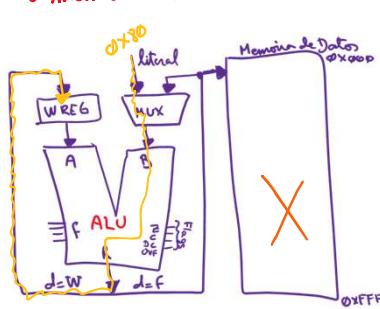
Instrucciones básicas en XC8 PIC Assembler

- Instrucciones de movimiento de datos

`movlw [literal]`

- mover un literal hacia WREG

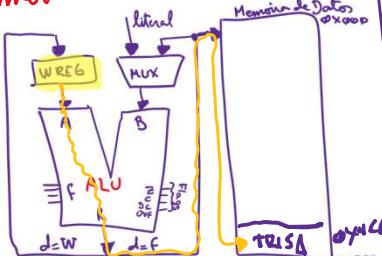
Ej: $maw \times 80$



movwf [registro], a

- mover el contenido de WREG
hacia [registro] a:1

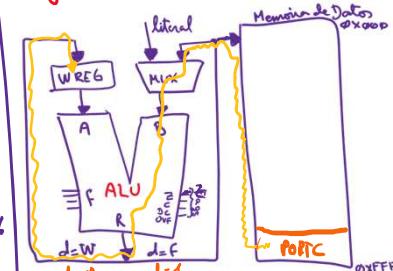
Ej: movwf TRISA, f $a=1 \rightarrow BSR$



movf [registro], d, a

mover el contenido de
y lo muerde según "d".

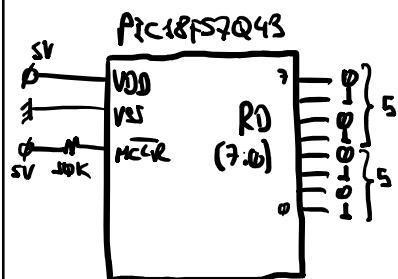
Ej: $\text{movf PORTC, } \emptyset, 1$



Note: movf [registro], f sirve para act. flag

Ejemplo: Escribir el número 55H en el puerto D del microcontrolador PIC18F57Q43

- Primero: hacer que el puerto D sea salida digital
 - Segundo: cargar el numero 55H en el registro W
 - Tercero: mover el contenido de registro W hacia LATD

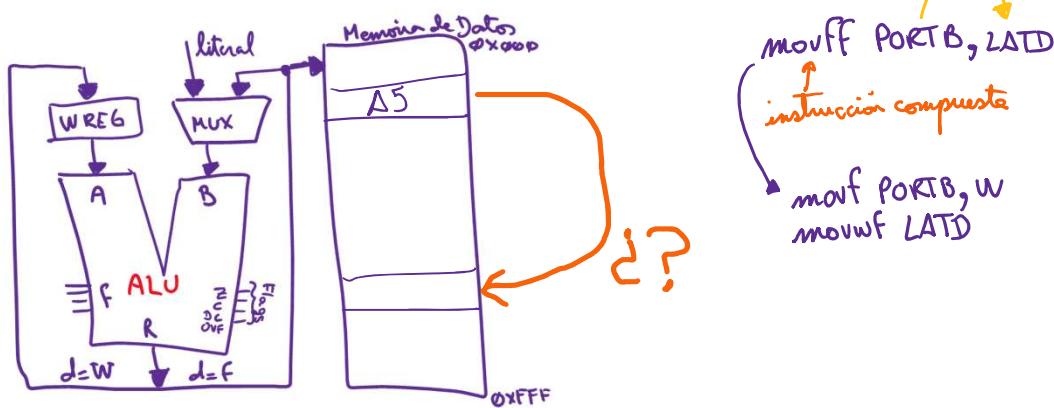


```
inicio:    movlb 4H      ;yendo al Bank4
           movlw 00H      ;cargo literal 00H a Wreg
           movwf TRISD, 1 ;muevo contenido de Wreg hacia TRISD
           movwf ANSEL0, 1 ;muevo contenido de Wreg hacia ANSEL0
           movlw 55H      ;muevo literal 55H hacia Wreg
           movwf LATD, 1  ;muevo contenido de Wreg hacia LATD

           end
```

Instrucción movff [registro1], [registro2]

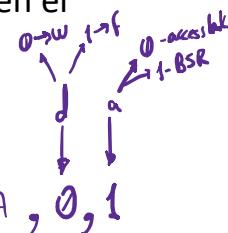
- Mueve el contenido de registro1 hacia registro2
- Ocupa el doble y demora el doble**



¿Instrucción movfw?

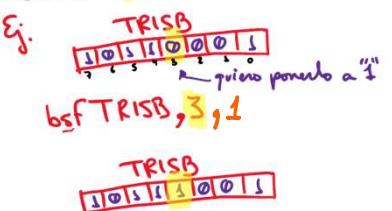
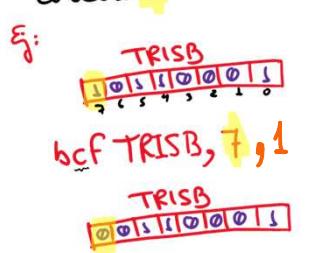
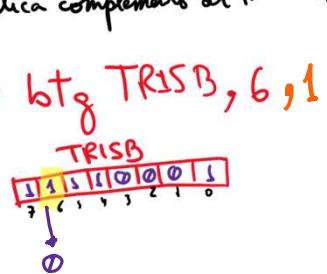
- ¡Instrucción no documentada en la hoja técnica!
- Instrucción “legacy”, instrucción de compatibilidad
- Esta no es una instrucción en sí, sino una “pseudo-instrucción” del lenguaje assembler.
- Lo que hace es mover el contenido de un registro y lo coloca en el Wreg.

Ej: movfw TRISA simila
 \rightarrow movf TRISA, 0, 1

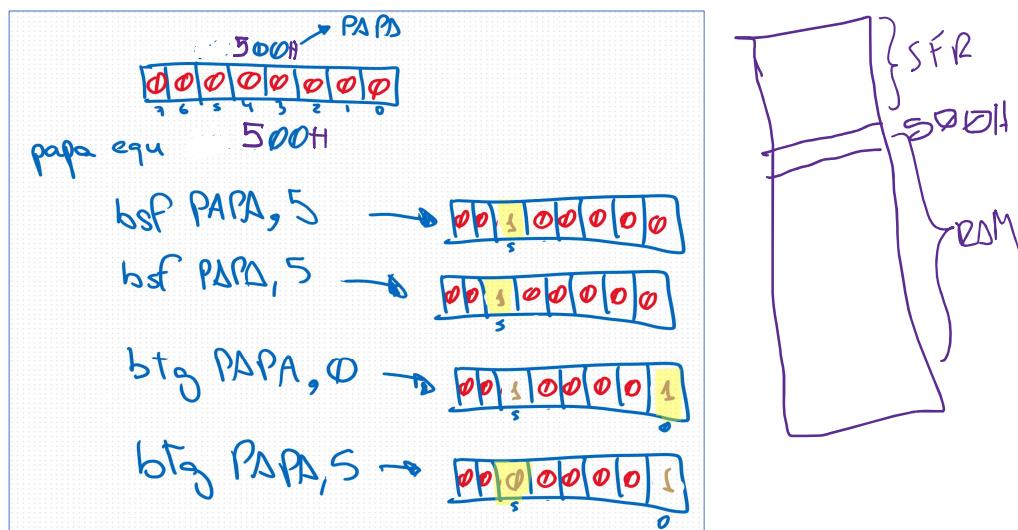


Instrucciones básicas en XC8 PIC Assembler

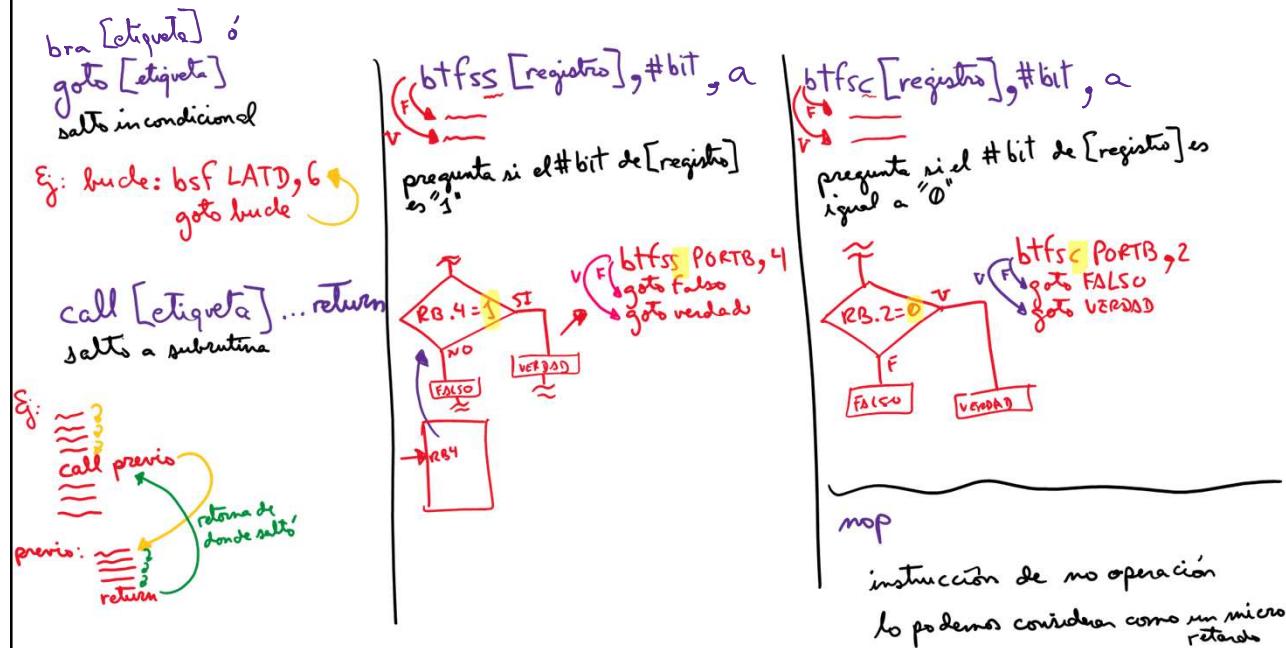
- Instrucciones de manipulación de un bit determinado, en un registro
- Hay que recordar que todos los registros son de ocho bits

<p><u>bsf</u> [registro], #bit, ^ posición coloca a "1" el #bit de [registro]</p> <p>Ej:  </p> <p><u>bsf</u> TRISB, 3, 1</p>	<p><u>bcf</u> [registro], #bit, ^ coloca a "0" el #bit de [registro]</p> <p>Ej:  </p> <p><u>bcf</u> TRISB, 7, 1</p>	<p><u>btg</u> [registro], #bit, ^ aplica complemento al #bit de [registro]</p> <p>Ej: <u>btg</u> TRISB, 6, 1</p> 
--	---	--

Ejemplo de uso de instrucciones de manipulación de bits en un registro



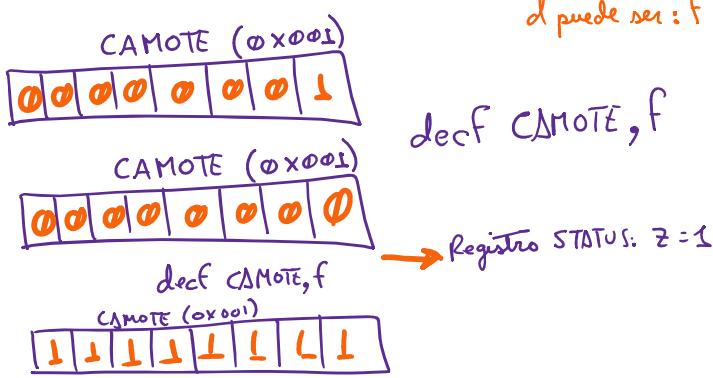
Instrucciones básicas en XC8 PIC Assembler



Instrucciones básicas en XC8 PIC Assembler

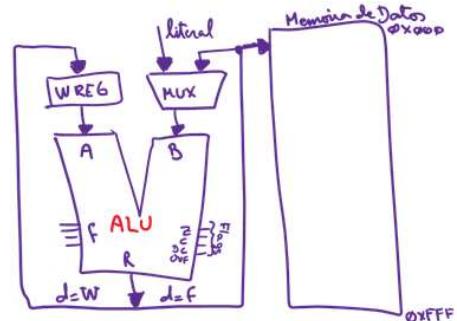
- Instrucción decf / incf
 - Decremento (decf) o incremento (incf) de registro, ambos de **uno en uno**

Ej: decf [registro], d, a incf [registro], d, a
"d" puede ser: f ó w



decf CAMOTE, f

Registro STATUS: Z=1



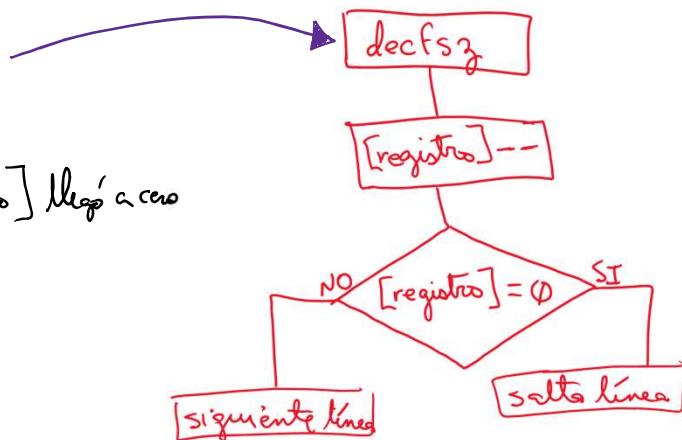
Instrucciones básicas en XC8 PIC Assembler

decfsz [registro], d

decrementa y pregunta si [registro] llegó a cero

incfsz [registro], d

incrementa y pregunta si [registro] llegó a cero



Instrucciones setf [registro], clrf [registro], comf [registro]

- **setf [registro]** : Coloca todos los bits del registro indicado a uno lógico

ejemplo: TRISB
10110001

setf TRISB

TRISB
11111111

- **clrf [registro]** : Coloca todos los bits del registro indicado a cero lógico

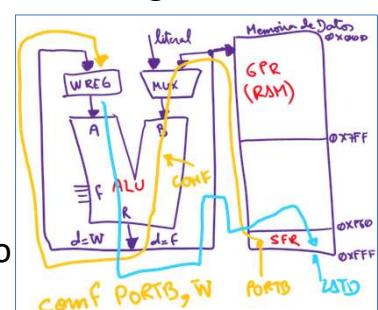
ejemplo:

clrf TRISB

TRISB
10110001

TRISB
00000000

- **comf [registro]** : Complementa todos los bits del registro



Ejemplo con la instrucción "comf"

ZANAHORIA (505H) = 55H

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

comf ZANAHORIA, 1, 1

ZANAHORIA (505H)

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

 0AAH

ZANAHORIA (505H) = 55H

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

comf ZANAHORIA, 0, 1

ZANAHORIA (505H)

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

 55H

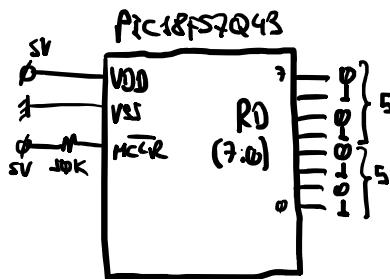
WREG

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

 0AAH

Ejemplo: Escribir el número 55H en el puerto D del microcontrolador PIC18F57Q43

- Optimizando con la instrucción clrf



```

inicio:    movlb 4H          ;yendo al Bank4
          clrf TRISD, 1      ;TRISD todos sus bits a cero
          clrf ANSELD, 1      ;ANSELD todos sus bits a cero
          movlw 55H           ;muevo literal 55H hacia Wreg
          movwf LATD, 1        ;muevo contenido de Wreg hacia LATD
          end

```

Ejemplo: Puerto D todos sus bits como salida digital:

Opción 1: usando bcf	Opción 2: usando valor numérico	Opción 3: usando clrf
$\text{bcf TRISD}, 0$ $\text{bcf TRISD}, 1$ \vdots $\text{bcf TRISD}, 7$ R ocho instrucciones	movlw 00H movwf TRISD, 1 una sola instrucción	clrf TRISD, 1 una sola instrucción !

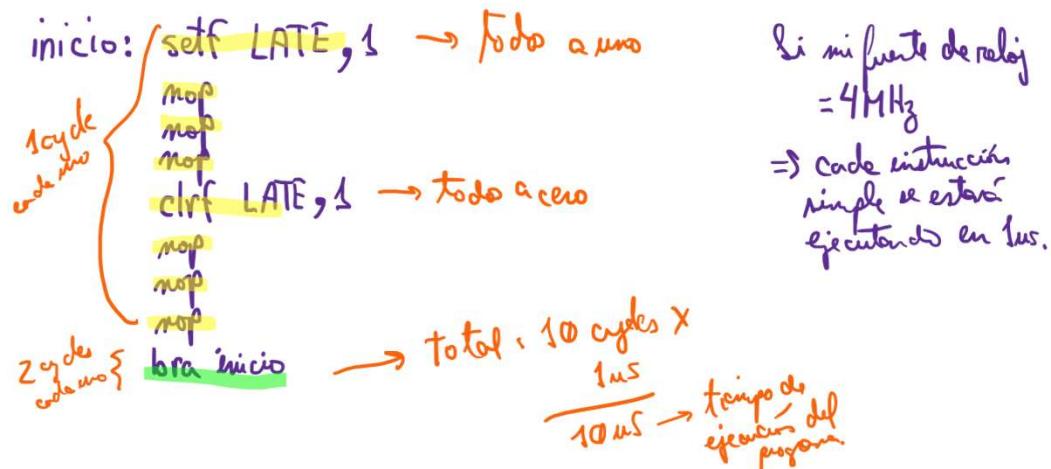
Tiempo de ejecución de las instrucciones en XC8 PIC Assembler

- Se utiliza la siguiente fórmula:

$$t_{opc} = \left(\frac{f_{osc}}{4} \right)^{-1} \text{ s}$$

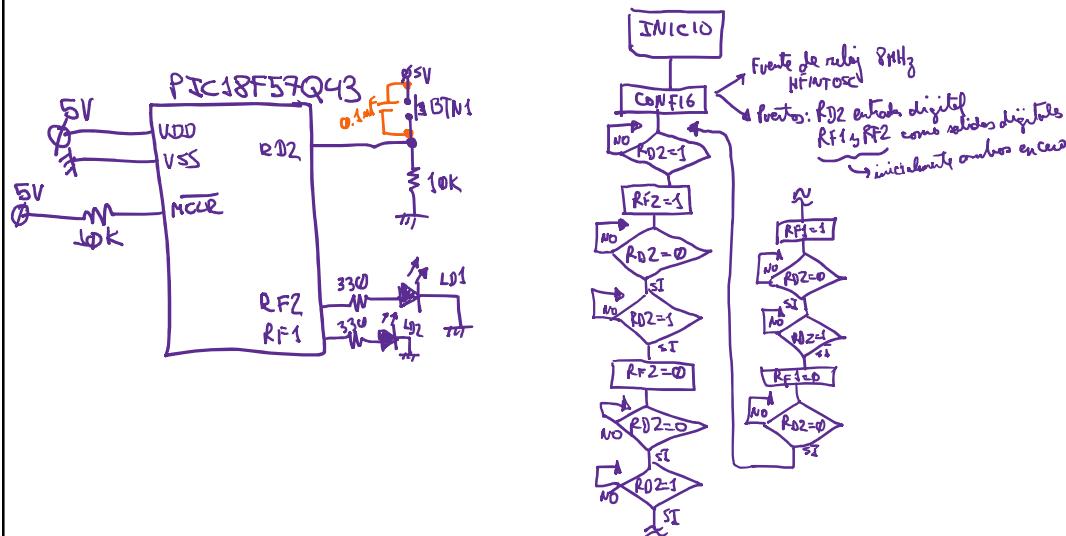
- Hay instrucciones simples, dobles y especiales (revisar 44.0 de la datasheet)
- Recordando la relación periodo vs frecuencia: $f = \frac{1}{T}$

Ejemplo de tiempo de ejecución



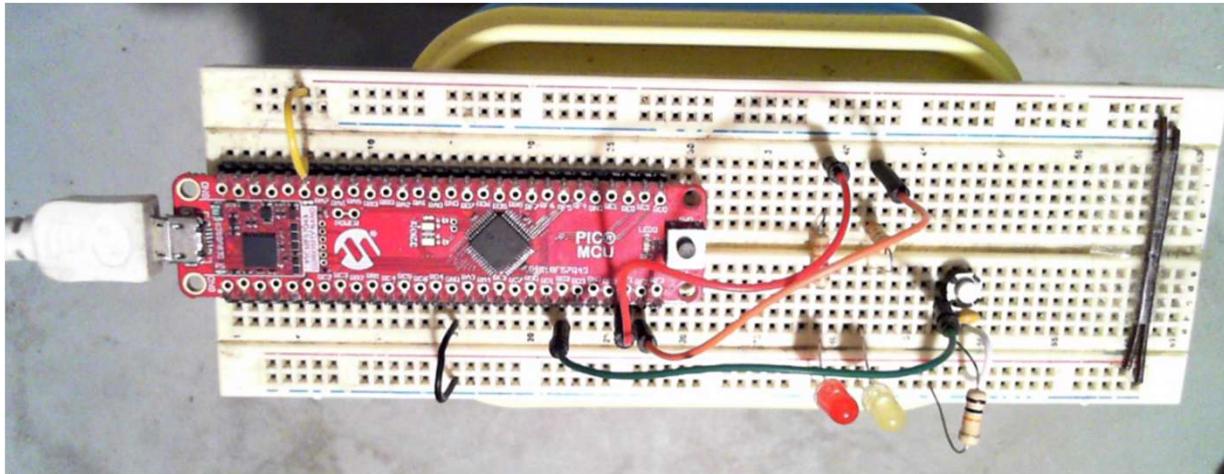
Ejemplo 02: Desarrollar un dispositivo LATCH con el Curiosity Nano PIC18F57Q43

- Latch: Dispositivo enclavador – Mantiene el valor de salida hasta que pulses el botón



Ejemplo 02: Desarrollar un dispositivo LATCH con el Curiosity Nano PIC18F57Q43

- Latch: Dispositivo enclavador – Mantiene el valor de salida hasta que pulses el botón



Ejemplo 02: Desarrollar un dispositivo LATCH con el Curiosity Nano PIC18F57Q43

- Latch: Dispositivo enclavador – Mantiene el valor de salida hasta que pulses el botón

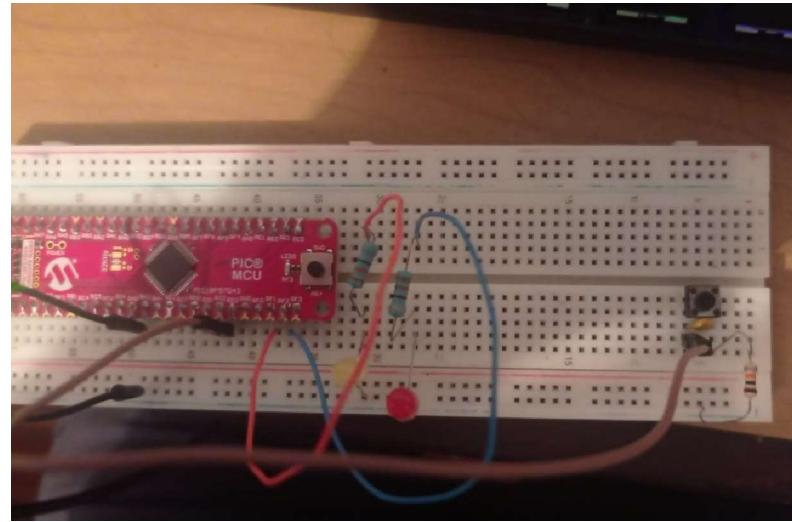
```

1  PROCESSOR 18F57Q43
2  #include "cabecera.inc"
3
4  PSECT upcino, class=CODE, reloc=2, abs
5  upcino:
6    ORG 000000H      ;vector de reset
7    bra configuro   ;salto a label configuro
8
9  configuro:        ;label configuro
10   movlb 0H          ;al BANK0
11   movlw 60H
12   movwf OSCCON1, 1  ;HFINTOSC y 1:1
13   movlw 03H
14   movwf OSCFRQ, 1   ;HFINTOSC a 8MHz
15   movlw 40H
16   movwf OSCEN, 1    ;HFINTOSC habilitado
17   movlb 4H          ;al BANK4
18   bcf TRISD, 2, 1   ;RD2 como entrada
19   bcf ANSEL0, 2, 1   ;RD2 como digital
20   bcf TRISF, 1, 1   ;RF1 como salida
21   bcf ANSEL1, 1, 1   ;RF1 como digital
22   bcf TRISF, 2, 1   ;RF2 como salida
23   bcf ANSEL2, 2, 1   ;RF2 como digital
24   bcf LATF, 1, 1    ;RF1 en cero
25   bcf LATF, 2, 1    ;RF2 en cero
26 inicio:           ;regreso a label inicio
27   btfss PORTD, 2, 1 ;pregunto si presione el boton
28   bra inicio         ;falso, no presione el boton y vuelvo a preguntar
29   bsf LATF, 1, 1     ;verdad, enciendo primer LED
30
31   otrol:
32     btfsc PORTD, 2, 1 ;pregunto si deje de presionar el boton
33     bra otrol         ;falso y vuelvo a preguntar
34   otroal:
35     btfss PORTD, 2, 1 ;pregunto si presione el boton
36     bra otroal         ;verdad, apago el LED
37   otroa2:
38     btfsc PORTD, 2, 1 ;pregunto si solte el boton
39     bra otroa2         ;verdad, apago el LED
40   otro2:
41     btfss PORTD, 2, 1 ;pregunto si presione el boton
42     bra otro2         ;falso, no presione el boton y vuelvo a preguntar
43     bsf LATF, 2, 1     ;verdad, enciendo primer LED
44   otro3:
45     btfsc PORTD, 2, 1 ;pregunto si deje de presionar el boton
46     bra otro3         ;falso y vuelvo a preguntar
47   otroa3:
48     btfss PORTD, 2, 1 ;pregunto si pulse el boton
49     bra otroa3         ;verdad, apago el LED
50   otro4:
51     btfsc PORTD, 2, 1 ;pregunto si deje de presionar el boton
52     bra otro4         ;verdad, apago el LED
53     bra inicio         ;regreso a label inicio
54
55   end upcino
56

```

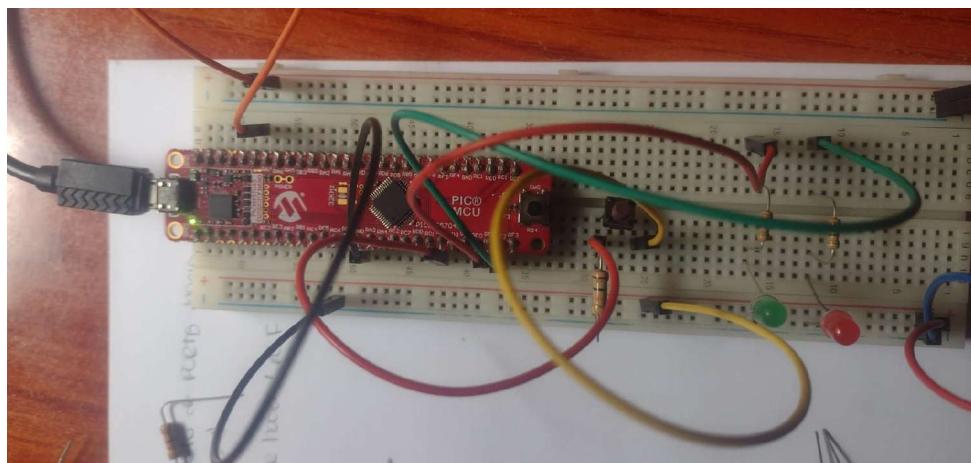
Error en la implementación

- “Profesor compila pero no funciona”
- Ubica el error:



Error en la implementación

- “Profesor compila pero no funciona”
- Ubica el error:

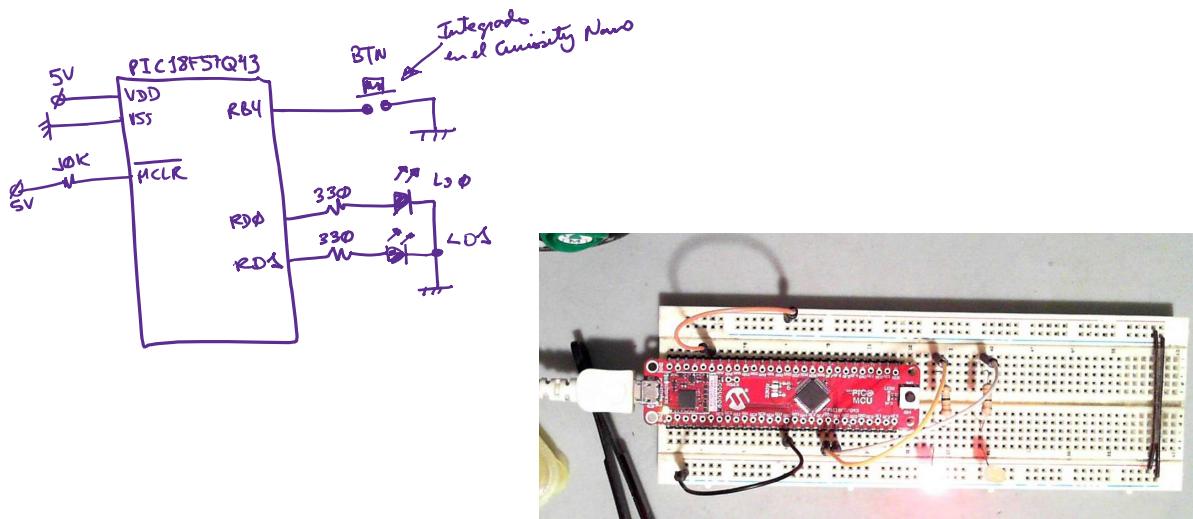


Problemas encontrados

- No funciona bien, parece que hay pulsaciones falsas en el botón
- Los botones tienen problema de rebote, por lo tanto se tiene que implementar una estrategia “anti-rebote”

Ejemplo 03: Señal de tren con dos LEDs controlados por pulsador con el Curiosity Nano PIC18F57Q43

- Diseño del hardware



Ejemplo 03: Señal de tren con dos LEDs controlados por pulsador con el Curiosity Nano PIC18F57Q43

- Código en XC8 PIC Assembler

```

1      PROCESSOR 18F57Q43
2      #include "cabecera.inc"
3
4      PSECT upcino, class=CODE, reloc=2, abs
5      upcino:
6          ORG 000000H
7          bra configuro
8
9          ORG 000020H
10     configuro:
11         movlb 0H
12         movlw 60H
13         movwf OSCCON1, 1
14         movlw 02H
15         movwf OSCFRQ, 1
16         movlw 40H
17         movwf OSCEN, 1
18
19         movlb 4H
20         bsf TRISB, 4, 1
21         bcf ANSELB, 4, 1
22         bsf WPUB, 4, 1
23         movlw 0FCH
24         movwf TRISD, 1           ;RD0 y RD1 como salidas
25         movwf ANSELD, 1         ;RD0 y RD1 como digitales
26         movlw 01H
27         movwf LATD, 1           ;RD0 encendido y RD1 apagado

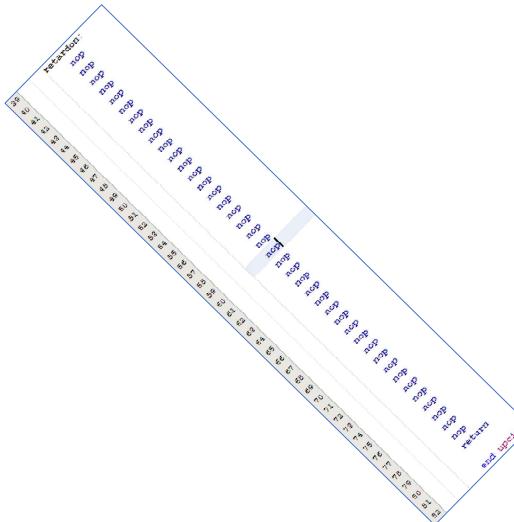
29      inicio:
30          btfsc PORTB, 4, 1   ;Pregunta si RB4 es cero (si presione el boton)
31          bra inicio         ;falso, regresa a preguntar
32          comf LATD, 1, 1     ;complemento a registro
33          call retardon
34      otro:
35          btfss PORTB, 4, 1   ;Pregunta si RB4 es uno (si solte el boton)
36          bra otro            ;falso, vuelvo a preguntar
37          bra inicio

```

```
39      retardon:  
40      pop  
41      pop  
42      pop  
43      pop  
44      pop  
45      pop  
46      pop  
47      pop  
48      pop  
49      pop  
50      pop  
51      pop  
52      pop  
53      pop  
54      pop  
55      pop  
56      pop  
57      pop  
58      pop  
59      pop  
60      pop  
61      pop  
62      pop  
63      pop  
64      pop  
65      pop  
66      pop  
67      pop  
68      pop  
69      pop  
70      pop  
71      pop  
72      pop  
73      pop  
74      pop  
75      pop  
76      pop  
77      pop  
78      pop  
79      pop  
80      return  
81  
82      end unsime
```

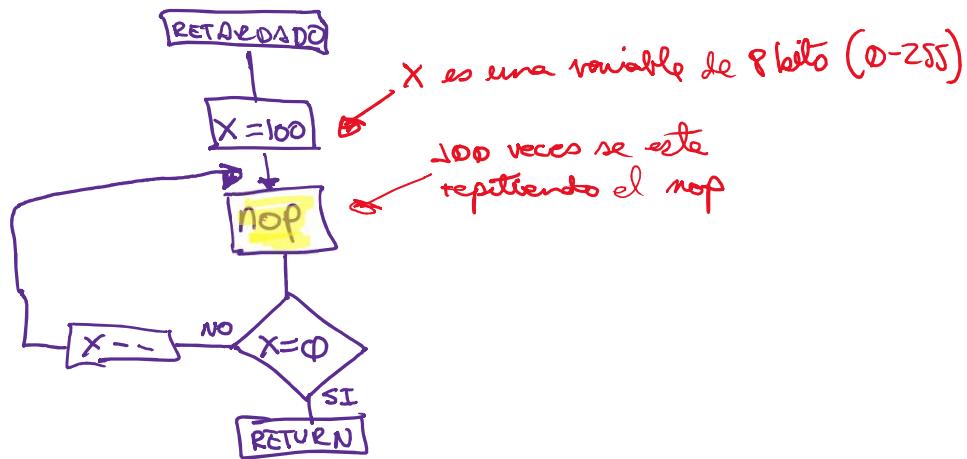
¿Cómo hago el antirrebote del botón sin tener que escribir tanto “nops”?

- Escribir 40000 nops no es práctico ni eficiente!



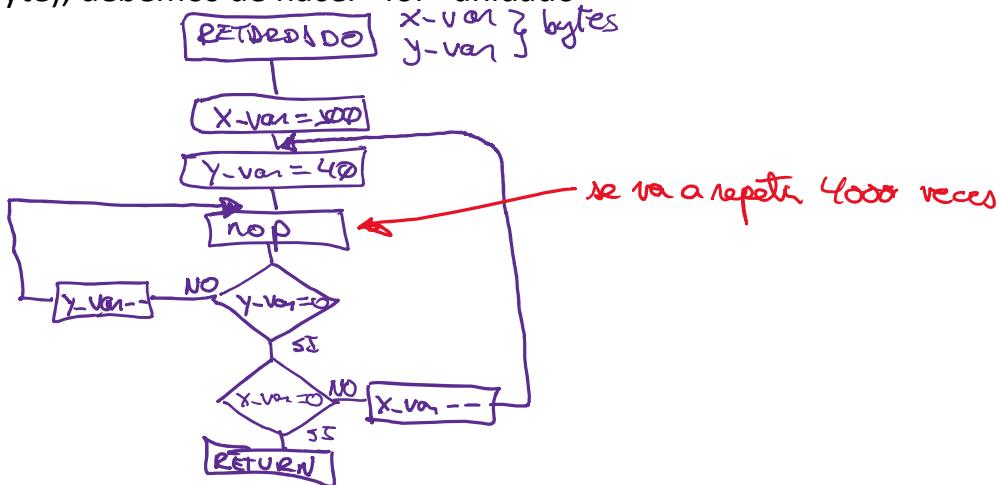
¿Cómo generamos el retardo de 40ms para el antirrebote del botón?

- Usar muchos nops ocupa demasiado espacio
- Se puede armar una rutina de bucle de repetición (for)



¿Cómo generamos el retardo de 40ms para el antirrebote del botón?

- Con un solo bucle de repetición solo llegamos a 255 (valor máximo en un byte), debemos de hacer "for" anidado



¿Cómo generamos el retardo de 40ms para el antirrebote del botón?

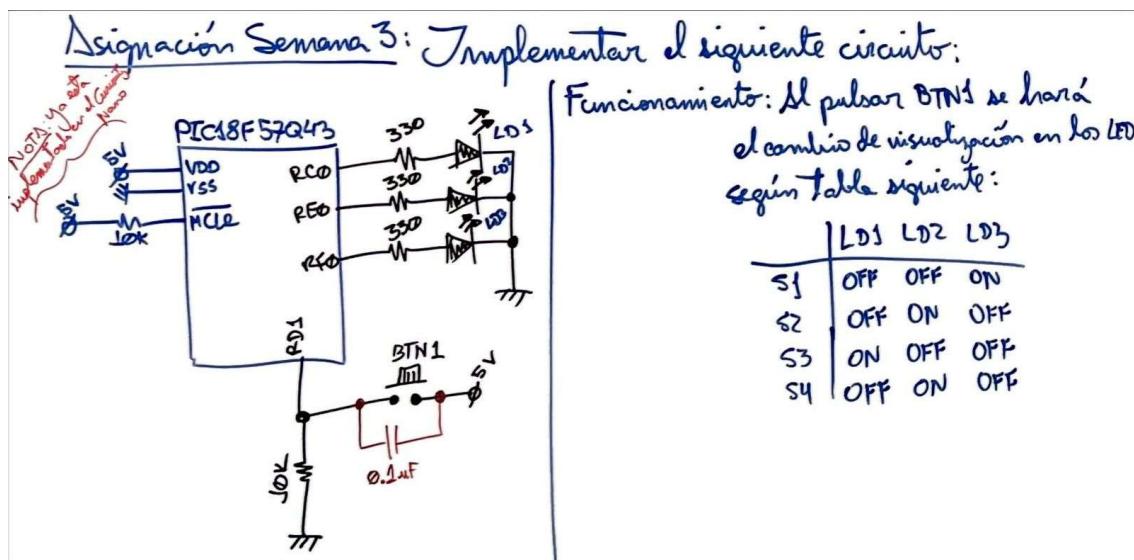
- **x_var** y **y_var** son etiquetas asignadas a posiciones GPR de la memoria de datos, recordando que los GPR empiezan a partir de la dirección 500H
- El código fue obtenido del diagrama de flujo anterior, tener en cuenta que al usar BSR se tiene que estar cambiando de bancos ya que el registro STATUS se encuentra en el BANK4 (4D8H) y los GPRs **x_var** y **y_var** están ubicados en BANK5 (500H y 501H respectivamente)
- Reemplazarlo en los códigos anteriores para obtener un mejor filtro antirrebote

```

40  retardado:
41      movlb 5H
42      movlw 100
43      movwf x_var, 1
44
45  otro3:
46      movlw 40
47      movwf y_var, 1
48      nop           ;se va a ejecutar 40000 veces
49      movf y_var, 1, 1
50      movlb 4H
51      btfss STATUS, 2, 1
52      bra y_var_noescero
53      movlb 5H
54      movf x_var, 1, 1
55      movlb 4H
56      btfss STATUS, 2, 1
57      bra x_var_noescero
58      return
59  y_var_noescero:
60      movlb 5H
61      decf y_var, 1, 1
62      bra otro2
63  x_var_noescero:
64      movlb 5H
65      decf x_var, 1, 1
66      bra otro3

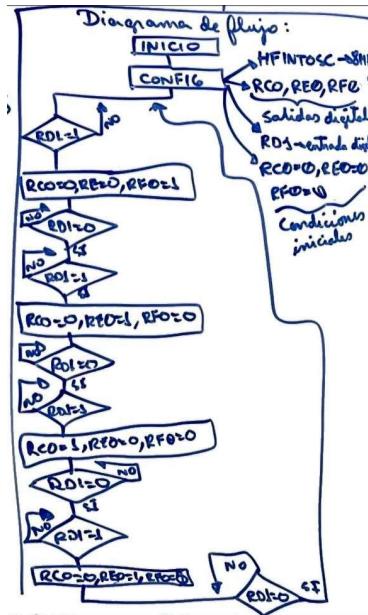
```

Ejemplo 2024-1



Ejemplo 2024-1

- Diagrama de flujo:



```

1      PROCESSOR 18F57Q43 ;directiva de procesador
2      #include "cabecera.inc"

3
4      PSECT upcino, class=CODE, reloc=2, abs ;apertura de program section
5      upcino:
6          ORG 000000H ;etiqueta upcino
7          bra configuro ;vector de reset
8
9          ORG 000100H ;zona de programa de usuario
10         configuro:
11             movlb 0H ;etiqueta configuro
12             movlw 60H ;al BANK0
13             movwf OSCCON1, 1 ;NOSC=110 (HFINTOSC), NDIV=0000 (1)
14             movlw 03H
15             movwf OSCFRC, 1 ;HFINTOSC a 8MHz
16             movlw 40H
17             movwf OSCEN, 1 ;HFINTOSC enabled
18             movlb 4H ;al BANK4
19             bcf TRISD, 1, 1 ;RDI como entrada
20             bcf ANSELD, 1, 1 ;RDI como digital
21             bcf TRISC, 0, 1 ;RC0 como salida
22             bcf ANSELC, 0, 1 ;RC0 como digital
23             bcf TRISE, 0, 1 ;RE0 como salida
24             bcf ANSELE, 0, 1 ;RE0 como digital
25             bcf TRISE, 0, 1 ;RF0 como digital
26             bcf ANSELF, 0, 1 ;RF0 como digital
27             bcf LATC, 0, 1 ;RC0 en cero
28             bcf LATE, 0, 1 ;RE0 en cero
29             bcf LATF, 0, 1 ;RF0 en cero
30
31         inicio:
32             btfs PORTD, 1, 1 ;Pregunto si pulse el BTN1
33             bra inicio ;no pulse BTN1 y me regreso
34             bcf LARC, 0, 1 ;RC0 en cero
35             bcf LATE, 0, 1 ;RE0 en cero
36             bcf LATF, 0, 1 ;RF0 en uno
37         otrol:
38             btfc PORTD, 1, 1 ;Pregunto si dejo de pulsar BTN1
39             bra otrol ;sigue BTN1 pulsado y regreso
40         otro2:
41             btff PORTD, 1, 1 ;Pregunto si pulse el BTN1

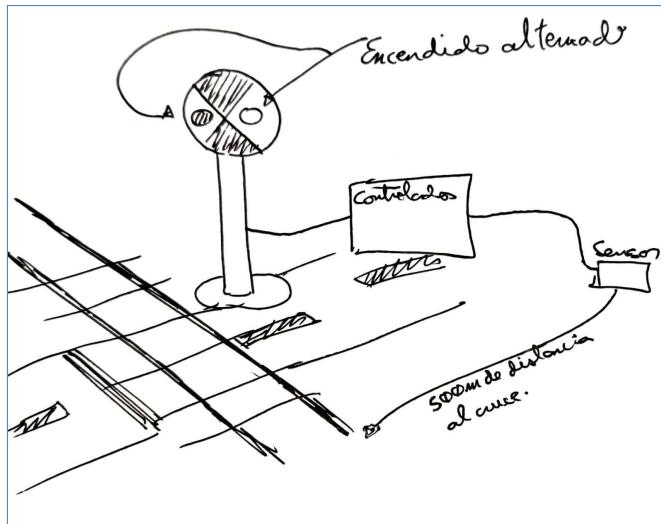
```

Ejemplo 2024-1

- Código:

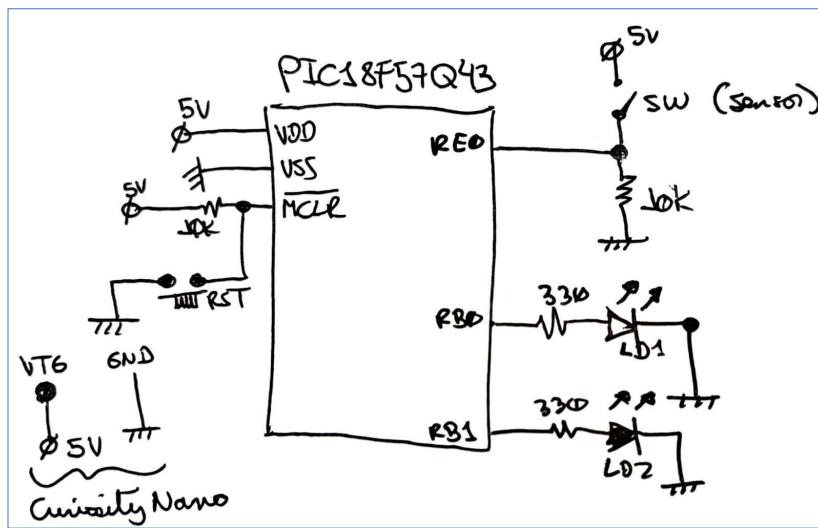
Ejemplo 2 – 2024-2

- Desarrollar una aplicación de una señal de cruce de tren con control de encendido a través de un pulsador.



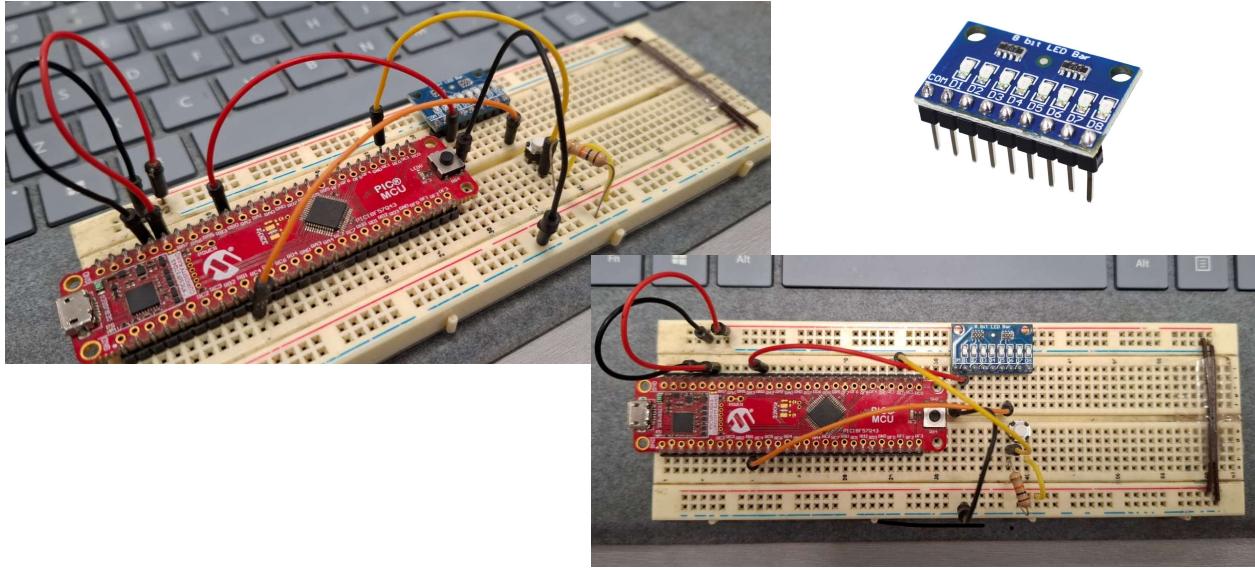
Ejemplo 2 – 2024-2

- Diseño del hardware – Diagrama esquemático



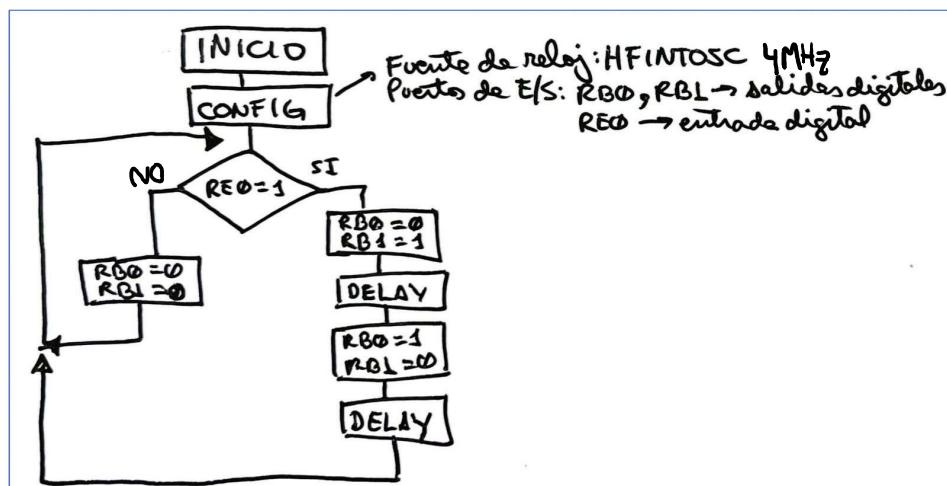
Ejemplo 2 – 2024-2

- Diseño del hardware – Implementación



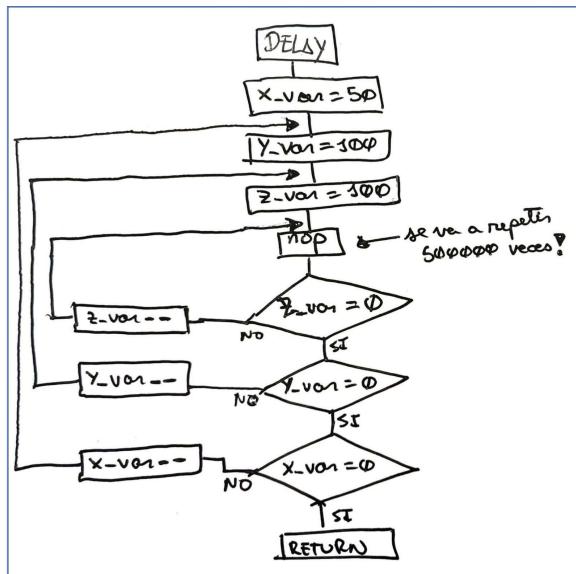
Ejemplo 2 – 2024-2

- Diseño de software: Diagrama de flujo



Ejemplo 2 – 2024-2

- Diseño de software: Diagrama de flujo



Ejemplo 2 – 2024-2

- Diseño de software: Codificación en lenguaje XC8 PIC Assembler

```

1      PROCESSOR 18F57Q43
2      #include "cabecera.inc"
3
4      ;asignacion de nombres a los registros GPR para la rutina de retardo
5      x_var EQU 500H
6      y_var EQU 501H
7      z_var EQU 502H
8
9      PSECT upcino, class=CODE, reloc=2, abs
10     upcino:
11         ORG 000000H      ;vector de reset
12         bra configuro   ;salto a label configuro
13
14         ORG 000100H      ;zona de programa de usuario
15         configuro:
16             ;configuracion de la fuente de reloj
17             movlb 0H          ;nos vamos al Bank0
18             movlw 60H
19             movwf OSCCON1, 1   ;NOSC=HFINTOSC, NDIV 1:1
20             movlw 02H
21             movwf OSCFRC, 1    ;HFINTOSC a 4MHz
22             movlw 40H
23             movwf OSCEN, 1      ;HFINTOSC habilitado
24             ;configuracion las E/S
25             movlb 4H
26             bsf TRISE, 0, 1    ;REO como entrada
27             bcf ANSELB, 0, 1    ;REO como digital
28             ;opcion 1 manipulacion individual de bits
29             ;bcf TRISE, 0, 1    ;RB0 como salida
30             ;bcf TRISB, 1, 1    ;RB1 como salida
31             ;bcf ANSELB, 0, 1    ;RB0 como digital
32             ;bcf ANSELB, 1, 1    ;RB1 como digital
33             ;opcion 2 manipulacion numerica al registro
34             movlw 0FCH
35             movwf TRISB, 1        ;RB0 y RB1 como salidas
36             movwf ANSELB, 1        ;RB0 y RB1 como digitales
37
38         inicio_train:
39             btfss PORTE, 0, 1  ;Pregunto si RE0 es uno
40             bra es_falso        ;Falso, salta a label es_falso
41             movlw 01H            ;Verdad (efecto de luces del cruce de tren)
42             movwf LATB, 1        ;RB1 apagado, RB0 encendido
43             call retardo        ;llamo rutina de retardo
44             movlw 02H
45             movwf LATB, 1        ;RB1 encendido, RB0 apagado
46             call retardo        ;llamo rutina de retardo
47             bra inicio_train    ;salta a label inicio_train
48             es_falso:
49                 clrf LATB, 1    ;todo el RB en cero (apagado RB1 y RB0)
50                 bra inicio_train  ;salta a label inicio_train
51
52         retardo:
53             movlb 5H            ;nos vamos al Bank5
54             movlw 50
55             movwf x_var, 1
56             punto_2:
57                 movlw 50
58                 movwf y_var, 1
59             punto_1:
60                 movlw 50
61                 movwf z_var, 1
62             punto_0:
63                 nop                ;se va a repetir 125000 veces
64                 movlw 0
65                 cpfsq x_var, 1
66
67         falso_z:
68             movlw 0
69             cpfsq y_var, 1
70             bra falso_y
71             movlw 0
72             cpfsq x_var, 1
73             bra falso_x
74             movlb 4H              ;regresamos al Bank0
75             return
76
77         falso_x:
78             deef z_var, 1, 1
79             bra punto_0
80             falso_y:
81             deef y_var, 1, 1
82             bra punto_1
83             falso_z:
84             deef x_var, 1, 1
85             bra punto_2
86
87         end upcino.
  
```

Ejemplo 2 – 2024-2

- Diseño de software: Codificación en lenguaje XC8 PIC Assembler

```

1      PROCESSOR 18F57Q43          34    inicio_tren:
2      #include "cabecera.inc"    35        btfss PORTE, 0, 1 ;pregunto si RE0 es 1
3      ;Asignacion de labels a GPR 36        bra es_falsozao ;viene aqui cuando es Falso
4      x_var EQU 500H              37        bra es_verdad ;viene aqui cuando es Verdadero
5      y_var EQU 501H              38        es_verdad:
6      z_var EQU 502H              39            bsf LATB, 0, 1 ;enciende RB0
7      ;PSECT upcino, class=CODE, reloc=2, abs 40            bcf LATB, 1, 1 ;apago RB1
8      upcino:                   41            call retardado ;llamada a subrutina retardado
9      ORG 000000H ;vector de RESET 42            bcf LATB, 0, 1 ;apago RB0
10     bra configuro ;salto a label configuro 43            bsf LATB, 1, 1 ;enciende RB1
11     ORG 000100H ;zona de programa de usu 44            call retardado ;llamada a subrutina retardado
12     configuro:                45            brc inicio_tren ;retorno al inicio_tren
13     ;configuracion la fuente de reloj 46        es_falsozao:
14     movlb 0H ;nos vamos al Bank0 47            bcf LATB, 0, 1 ;apago RB0
15     movlw 60H                  48            bra LATB, 1, 1 ;apago RB1
16     movwf OSCCON1, 1 ;NOSC=HFINTOSC, NDIV 49            call retardado ;llamada a subrutina retardado
17     movlw 02H                  50            bcf LATB, 0, 1 ;apago RB0
18     movwf OSCFRC, 1 ;HFINTOSC a 4MHz 51            bsf LATB, 1, 1 ;enciende RB1
19     movlw 40H                  52            call retardado ;llamada a subrutina retardado
20     movwf OSCEN, 1 ;HFINTOSC habilitado 53            brc inicio_tren ;retorno al inicio_tren
21     ;configuracion las E/S 54        punto_tres:
22     movlb 4H ;saltamos al Bank4 55            movlw 50
23     bcf TRISE, 0, 1 ;RE0 es entrada 56            movwf x_var, 1 ;x_var valor d
24     bcf ANSELB, 0, 1 ;RB0 es digital 57            btfss STATUS, 2, 1 ;preguntar si Z=1
25     bcf ANSELB, 1, 1 ;RB1 es digital 58            bra z_var_noescero ;Falso salta a label z_var_noescero
26     punto_dos:                59            movwf y_var, 1, 1 ;evaluo x_var
27     nop                      60            movlw 1
28     punto_uno:                61            btfss STATUS, 2, 1 ;preguntar si Z=1
29     punto_tres:               62            bra x_var_noescero ;Falso salta label x_var_noescero
30     ;Recordando bcf [registro], #bit, access 63            return
31     btfss STATUS, 2, 1 ;pregunto si Z=1 64        z_var_noescero:
32     bra z_var_noescero ;Falso salta a label z_var_noescero 65            decf z_var, 1, 1 ;decrementamos z_var
33     movwf y_var, 1, 1 ;Verdad, evaluar y_var 66            bra punto_uno ;salto a label punto_uno
34     btfss STATUS, 2, 1 ;preguntar si Z=1 67        y_var_noescero:
35     bra y_var_noescero ;Falso salta a label y_var_noescero 68            decf y_var, 1, 1 ;decrementamos y_var
36     punto_dos:                69            bra punto_dos ;salto a label punto_dos
37     nop                      70            decf x_var, 1, 1 ;decremento x_var
38     punto_uno:                71            bra punto_tres
39     punto_tres:               72            end upcino
40     movwf z_var, 1, 1 ;evaluar z_var 73
41     call retardado ;llamada 74
42     movlw 02H                  75
43     movwf LATB, 1 ;RB1=0, 76
44     call retardado ;llamada 77
45     bra inicio_tren 78
46     retardado:               79
47     movlb 5H ;al Bank5 80
48     movlw 30 81
49     movwf x_var, 1 82
50     punto_tres:               83
51     movlw 30 84
52     movwf y_var, 1 85
53     punto_dos:               86
54     movlw 30 87
55     movwf z_var, 1 88
56     punto_ultimo:             89
57     nop 90
58     movwf z_var, 1, 1 91
59     movlb 4H 92
60     btfss STATUS, 2, 1 93
61     bra zvar_noescero 94
62     movlw 5H 95
63     movwf y_var, 1, 1 96
64     btfss STATUS, 2, 1 97
65     bra yvar_noescero 98
66     movlw 5H 99
67     btfss STATUS, 2, 1 100
68     bra xvar_noescero 101
69     movlw 5H 102
70     btfss STATUS, 2, 1 103
71     bra zvar_noescero 104
72     decf z_var, 1, 1 105
73     bra punto_ultimo 106
74     yvar_noescero: 107
75     decf y_var, 1, 1 108
76     bra punto_dos 109
77     xvar_noescero: 110
78     decf x_var, 1, 1 111
79     bra punto_tres 112
80     end upcino 113

```

Ejemplo 2 – 2024-2

- Diseño de software: Codificación en lenguaje XC8 PIC Assembler

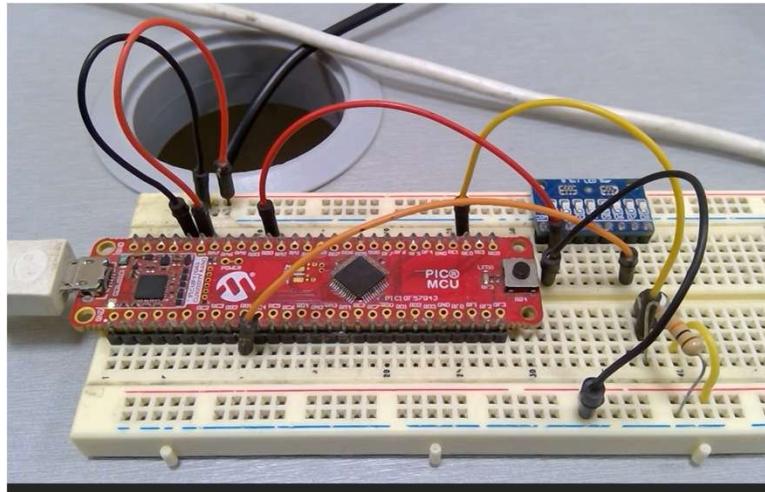
```

1      PROCESSOR 18F57Q43          29    bcf ANSELB, 0, 1 ;RE0 com 57    nop
2      #include "cabecera.inc"    30    movwf z_var, 1, 1 58    movlb 4H ;Al Bank4
3      x_var EQU 500H              31    inicio_tren: 59    btfss STATUS, 2, 1 ;pregunto si z_var llego a c
4      y_var EQU 501H              32    btfss PORTE, 0, 1 ;pregunto 60    bra zvar_noescero ;falso zvar no llego a cero
5      z_var EQU 502H              33    bra es_falso ;viene a 61    movlw 5H ;Al bank5
6      ;PSECT upcino, class=CODE, reloc=2, abs 34    bra es_verdad ;viene a 62    movwf y_var, 1, 1 ;verdad y prosigo a ls sigui
7      upcino:                   35    es_falso: 63    movlb 4H ;Al Bank4
8      ORG 000000H ;vector de reset 36    clrf LATB, 1 ;apagamo 64    btfss STATUS, 2, 1 ;pregunto si y_var llego a c
9      configuro:                37    bra inicio_tren 65    bra yvar_noescero ;falso yvar no llego a cero
10     ORG 000100H ;zona de usuario 38    es_verdad: 66    movlw 5H ;Al bank5
11     configuro:                39    movlw 01H 67    movwf x_var, 1, 1
12     ;configuracion de la fuente de reloj 40    movwf LATB, 1 ;RB1=0, 68    movlb 4H ;Al Bank4
13     movlb 0H ;al Bank0 41    call retardado ;llamada 69    btfss STATUS, 2, 1 ;pregunto si x_var llego a c
14     movlw 60H                  42    movlw 02H 70    bra xvar_noescero ;falso xvar no llego a cero
15     movwf OSCCON1, 1 ;NOSC=HFINTOSC NDIV 43    retardado: 71    return
16     movlw 02H                  44    movlb 5H ;al Bank5 72    zvar_noescero:
17     movwf OSCFRC, 1 ;HFINTOSC 45    movlw 30 73    movlb 5H ;Al bank5
18     movlw 40H                  46    call retardado ;llamada 74    decf z_var, 1, 1 ;decremento z_var
19     movwf OSCEN, 1 ;HFINTOSC enabled 47    bra inicio_tren 75    bra punto_ultimo
20     ;configuracion las E/S 48    movwf x_var, 1 76    yvar_noescero:
21     movlb 4H ;al Bank4 49    movlw 30 77    movlb 5H ;Al bank5
22     movlw 0xfc ;en binario 1111110 50    call retardado ;llamada 78    decf y_var, 1, 1 ;decremento y_var
23     movwf TRISB, 1 ;RB0 y RB1 como sal 51    movlw 30 79    bra punto_dos
24     movwf ANSELB, 1 ;RB0 y RB1 como dig 52    movwf z_var, 1 80    xvar_noescero:
25     bcf TRISE, 0, 1 ;RE0 como entrada 53    punto_dos: 81    movlb 5H ;Al bank5
26     punto_ultimo:             54    movlw 30 82    decf x_var, 1, 1 ;decremtno x_var
27     punto_tres:               55    movwf z_var, 1 83    bra punto_tres
28     punto_dos:               56    punto_ultimo: 84    end upcino
29     nop 85
30     movwf z_var, 1, 1 86
31     movlb 4H 87
32     btfss STATUS, 2, 1 88
33     bra zvar_noescero 89
34     movlw 5H 90
35     btfss STATUS, 2, 1 91
36     bra yvar_noescero 92
37     movlw 5H 93
38     btfss STATUS, 2, 1 94
39     bra xvar_noescero 95
40     movlw 5H 96
41     btfss STATUS, 2, 1 97
42     bra zvar_noescero 98
43     decf z_var, 1, 1 99
44     bra punto_ultimo 100
45     yvar_noescero: 101
46     decf y_var, 1, 1 102
47     bra punto_dos 103
48     xvar_noescero: 104
49     decf x_var, 1, 1 105
50     bra punto_tres 106
51     end upcino 107

```

Ejemplo 2 – 2024-2

- Diseño de software: Funcionamiento, al presionar el botón iniciará la secuencia de encendido de los LEDs



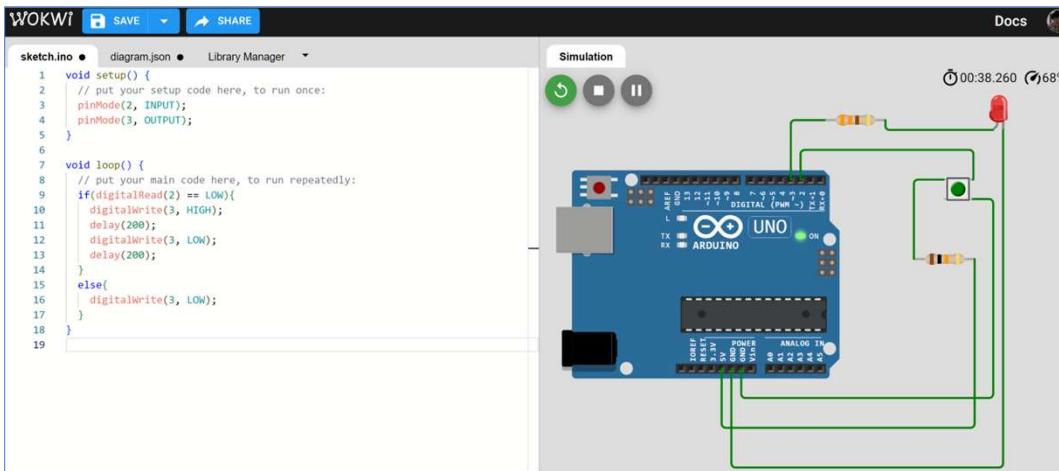
Ejemplo 2 – 2024-2

- Asignación: Mejorar el diseño agregando una entrada para que se puede manipular la velocidad de la alternancia de encendido de los LEDs

Ejemplo 2 (2025-1)

- Prender y apagar un LED, controlando su parpadeo con un pulsador activo en bajo, el parpadeo debe de activarse solo cuando se mantenga presionado el pulsador.

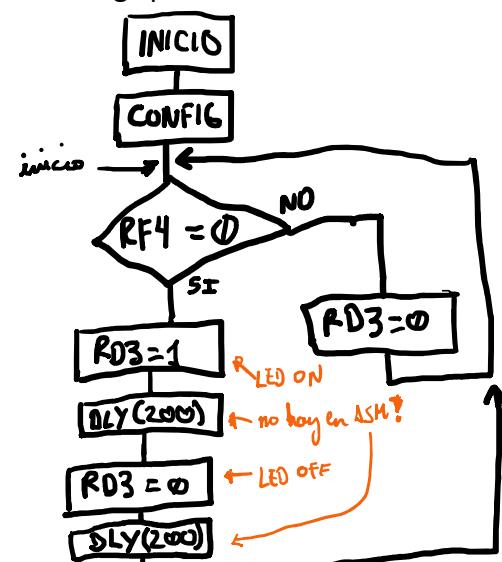
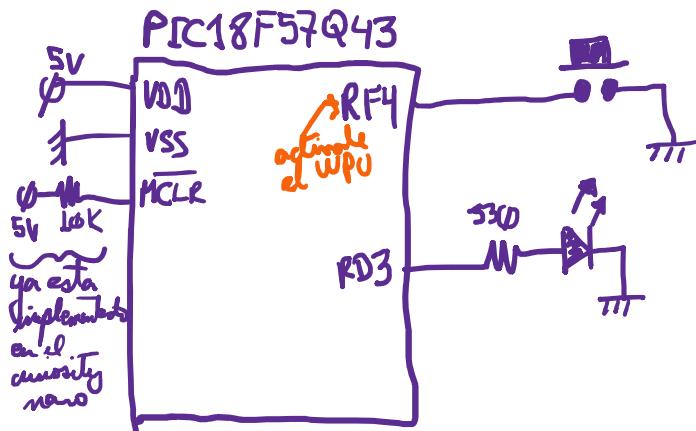
En Arduino:



Ejemplo 2 (2025-1)

- Prender y apagar un LED, controlando su parpadeo **con un pulsador activo en bajo**, el parpadeo debe de activarse solo cuando se mantenga presionado el pulsador.

Empleando el PIC18F57Q43:



Ejemplo 2 (2025-1)

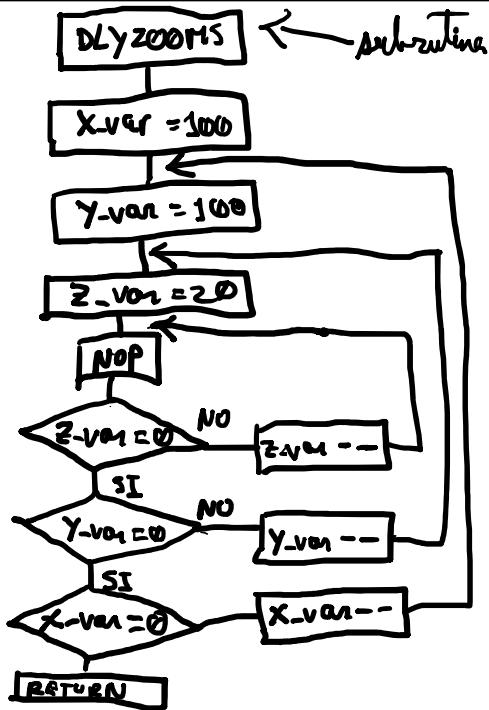
- Tenemos un pequeño inconveniente: en assembler no hay rutina de retardo implementado.

~~Tiene la instrucción "nop"~~

Si $f_{osc} = 4MHz$, ese `nop` se ejecutará en 1 μs .

Tu necesitas 200000 μs ,
250000 `nops`?

Tenemos que construir una
estructura de repetición
anidada para ese `nop`

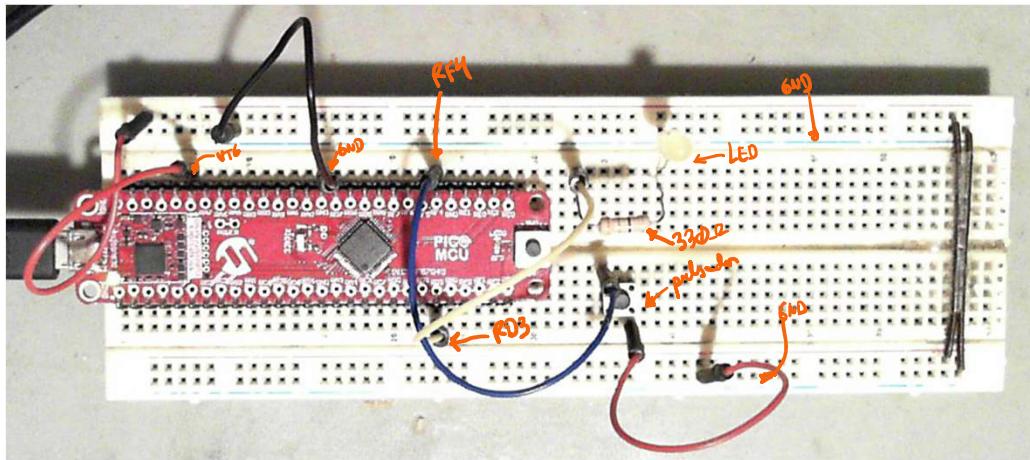


Ejemplo 2 (2025-1)

- Prender y apagar un LED, controlando su parpadeo con un pulsador activo en bajo, el parpadeo debe de activarse solo cuando se mantenga presionado el pulsador.

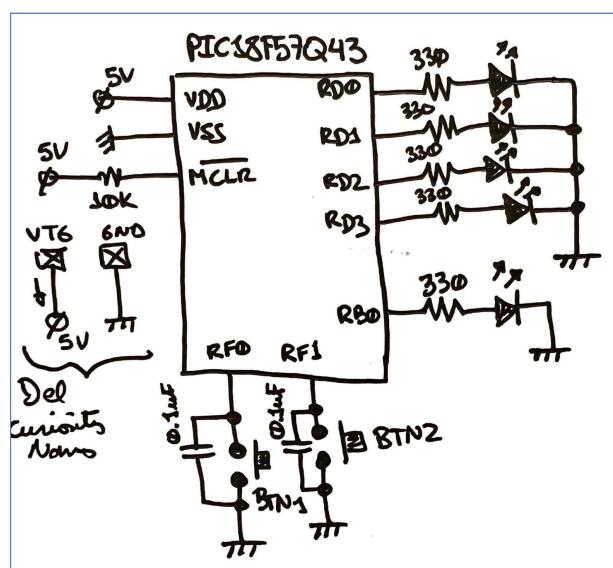
<pre> 1 ;Programa del ejemplo de la semana 2 - 2025-1 2 PROCESSOR 18F57043 3 #include "cabecera.inc" 4 5 PSECT upcino, class=CODE, reloc=2, abs 6 ;declarar etiquetas para los GPR 7 x_var EQU 500H 8 y_var EQU 501H 9 z_var EQU 502H 10 11 upcino: 12 ORG 000000H 13 bra configuro 14 15 ORG 000100H 16 configuro: 17 ;Configuracion del modulo de oscilador 18 movlb 0H 19 movlw 60H 20 movwf OSCCON1, 1 ;NOSC=HFINTOSC, NDIV 21 movlw 02H 22 movwf OSCFRC, 1 ;HFINTOSC a 4MHz 23 bsf OSCEN, 6, 1 ;HFINTOSC enabled 24 25 ;Configuracion de los puertos de E/S 26 movlb 4H 27 ;RF4 como entrada digital con pullup activado 28 bsf TRISF, 4, 1 ;RF4 entrada 29 bcf ANSELF, 4, 1 ;RF4 digital 30 bsf WPUF, 4, 1 ;RF4 pullup enabled 31 ;RD3 como salida digital 32 bcf TRISD, 3, 1 ;RD3 salida 33 bcf ANSELD, 3, 1 ;RD3 digital </pre>	<pre> 35 inicio: 36 btfc PORTF, 4, 1 ;pregunto si presione el botón 37 brcf nopalos ;falso, salta a nopalos 38 bra nospresione ;verdad, salta a nospresione 39 40 nopalos: 41 bcf LATD, 3, 1 ;apago el LED 42 bra inicio ;salto de retorno a inicio 43 44 nospresione: 45 bcf LATD, 3, 1 ;enciendo el LED 46 call dly200ms ;salto a subrutina dly200ms 47 bcf LATD, 3, 1 ;apago el LED 48 call dly200ms ;salto a subrutina dly200ms 49 bra inicio ;salto de retorno a inicio 50 51 ;subrutina de retardo 52 dly200ms: 53 movlb 5H 54 movlw 50 55 movwf z_var, 1 56 57 llegada_uno: 58 movlw 50 59 movwf y_var, 1 60 61 llegada_dos: 62 movlw 20 63 movwf z_var, 1 64 65 llegada_tres: 66 nop 67 ;pregunta si z_var llegó a cero 68 movf z_var, 1, 1 69 movlb 4H 70 btffs STATUS, 2, 1 ;pregunta si se levanto bandera Z 71 bra noselevantol 72 movlb 5H 73 ;pregunta si y_var llegó a cero 74 movf y_var, 1, 1 75 movlb 4H 76 btffs STATUS, 2, 1 ;pregunta si se levanto bandera Z 77 bra noselevantol 78 movlb 5H 79 btffs STATUS, 2, 1 ;pregunta si se levanto bandera Z 80 bra noselevantol 81 82 llegada_tres: 83 decf z_var, 1, 1 ;decremento de z_var 84 bra llegada_tres 85 86 noselevantol: 87 movlb 5H 88 decf y_var, 1, 1 ;decremento de y_var 89 bra llegada_dos 90 91 noselevantol: 92 movlb 5H 93 decf x_var, 1, 1 ;decremento de x_var 94 bra llegada_uno 95 96 end upcino </pre>
--	---

Ejemplo 2 (2025-1)



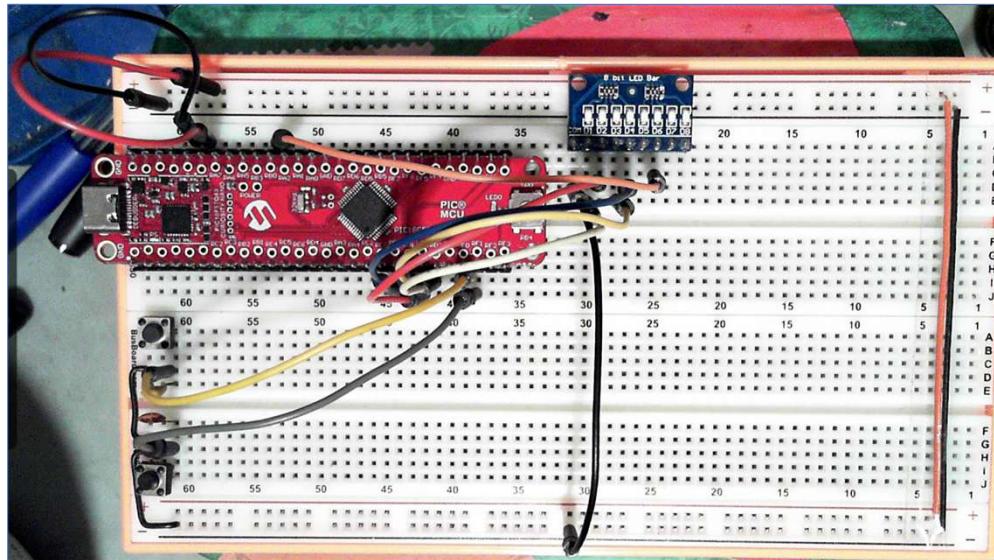
Ejemplo 2 (2025-2)

- Implementar el siguiente circuito:



Ejemplo 2 (2025-2)

- Implementar el siguiente circuito:



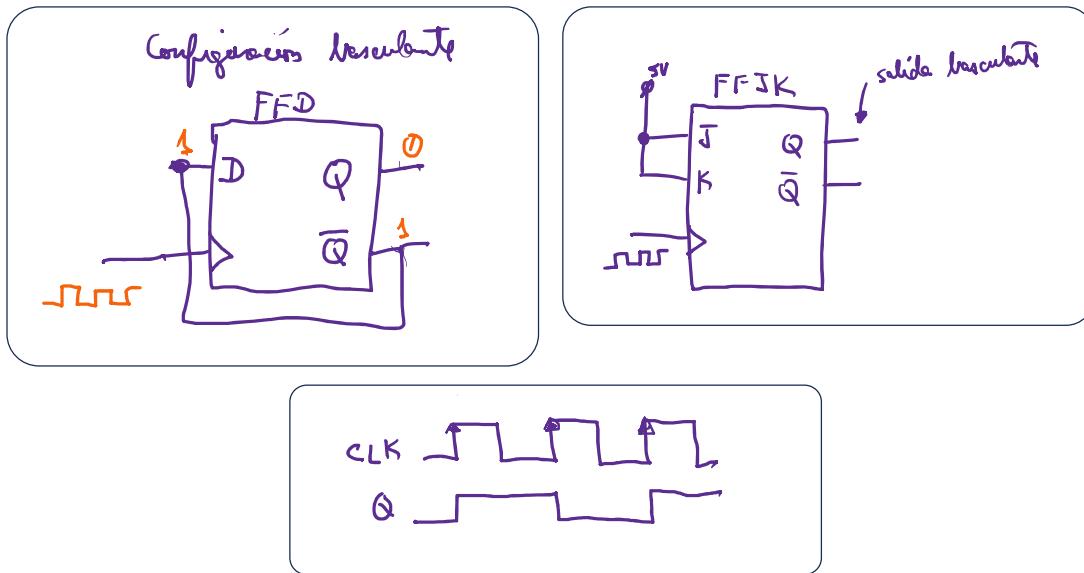
Ejemplo 2 (2025-2)

- Desarrollar con el circuito anterior efectos luminosos usando los pulsadores BTN1 y BTN2 como control según siguiente tabla:

Pulsador	Acción
BTN1 (RF0)	Latch al LED en RBO
BTN2 (RF1)	Intercambio entre dato 5H y 0AH en RD(3:0)

5H = 0101B
0AH = 1010B

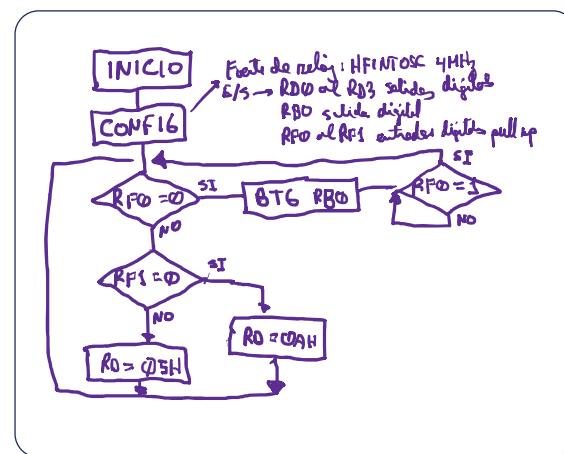
Recordando la función de Latch en circuitos lógicos digitales



Ejemplo 2 (2025-2)

- Desarrollar con el circuito anterior efectos luminosos usando los pulsadores BTN1 y BTN2 como control según siguiente tabla:

Pulsador	Acción
BTN1 (RF0)	Latch al LED en RB0
BTN2 (RF1)	Intercambio entre dato 5H y 0AH en RD(3:0)



Ejemplo 2 (2025-2)

Analizando:

- Fuente de reloj: HFINTOSC a 4MHz
- Puertos RD0 al RD3 como salidas digitales
- Puerto RB0 como salida digital
- Puertos RF0 y RF1 como entradas digitales con pullup interno activado.

```
;Configuracion del modulo de oscilador
;OSCCON1 = 60H NOSC=HFINTOSC, NDIV=1:1
;OSCFRQ = 02H HFINTOSC a 4MHz
;OSCEN = 40H HFINTOSC enabled
;Configuracion de las E/S:
;TRISD = 0F0H RD0 al RD3 como salidas
;ANSELD = 0F0H RD0 al RD3 como digitales
;TRISB.0 = 0 RB0 como salida
;ANSELB.0 = 0 RB0 como digital
;TRISF = 0FFH Todos los RF como entrada
(opcional)
;ANSELF = 0FCH RF1 y RF0 como digitales
;WPUF = 03H RF1 y RF0 con pullup activadas
```

Ejemplo 2 (2025-2)

<pre> 1 PROCESSOR 18F57043 2 #include "cabecera.inc" 3 4 PSELECT upcinos, class=CODE, reloc=2, abs 5 upcinos: 6 ORG 000000H 7 bra configuro 8 9 ORG 000100H 10 configuro: 11 ;conf la fuente de reloj 12 movlb 0H 13 movlw 60H 14 movwf OSCCON1, 1 ;NOSC=HFINTOSC, NDIV 15 movlw 02H 16 movwf OSCFRQ, 1 ;HFINTOSC a 4MHz 17 movlw 40H 18 movwf OSCEN, 1 ;HFINTOSC enabled 19 ;conf las E/S 20 movlb 4H 21 movlw 0FOH 22 movwf TRISD, 1 ;RD0 al RD3 como salidas 23 movwf ANSELD, 1 ;RD0 al RD3 como digitales 24 bcf TRISB, 0, 1 ;RB0 como salida 25 bcf ANSELB, 0, 1 ;RB0 como digital 26 setf TRISF, 1 ;Todo el puerto F como entradas 27 movlw 0FCH 28 movwf ANSELF, 1 ;RF1 y RF0 como digitales 29 movlw 03H 30 movwf WPUF, 1 ;RF1 y RF0 con pullup activadas </pre>	<pre> 32 inicio: 33 btfsc PORTF, 0, 1 ;Pregunto si RF0 es cero (si presione el boton1) 34 bra preguntaRF1 35 btg LATB, 0, 1 ;falso, salta a etiqueta pregunta RF1 36 otrol: 37 btfss PORTF, 0, 1 ;Pregunto si RF0 es uno (si solte el boton1) 38 bra otrol 39 bra inicio 40 preguntaRF1: 41 btfsc PORTF, 1, 1 ;Pregunto si RF1 es cero (si presione el boton2) 42 bra imprime5 43 movlw 0AH 44 movwf LATD, 1 ;falso, salta a label imprime5 45 bra inicio 46 imprime5: 47 movlw 05H 48 movwf LATD, 1 ;verdad, a imprimir A en RD 49 bra inicio 50 51 end upcinos </pre>
--	--

Ejercicios complementarios de manipulación de puertos con el PIC18F57Q43:

Tener en consideración el procedimiento para el desarrollo de los ejercicios (análisis del problema y requerimientos, diseño de hardware, diagrama de flujo, código en XC8 PIC Assembler y pruebas)

1. Colocar LEDs en todos los pines de RD y de RB (con su respectiva resistencia de 330Ω). Escribir 5AH en RD y 0A5H en RB.
2. Implementar una XOR de un bit empleando RD0 y RD4 como entradas y RB2 como la salida.
3. Recibir un dato de 8 bits en RD y replicarlo en complemento por RB
4. Con el algoritmo antirrebote basado en la generación de retardo de 40ms con bucles anidados visto en el último ejemplo 04. Ampliar dicho retardo hasta 500ms aproximadamente y realizar una aplicación de parpadeo de un LED conectado en el RA0.

Recomendaciones al momento de implementar el circuito en físico con el Curiosity Nano PIC18F57Q43:

- Verificar continuidad en los cables jumper antes de ser utilizados en el circuito.
- Verificar que el cable USB-microUSB tenga capacidad de transferencia de datos.
- Verificar que la PC haya detectado correctamente el Curiosity Nano PIC18F57Q43
- Verificar que el proyecto creado en el MPLABX se haya seleccionado el Curiosity Nano PIC18F57Q43 (ventana de propiedades del proyecto y “connected hardware tool”)
- No olvidar que el header file tiene extensión *.inc y el source file tiene extensión *.s
- Revisar que se haya configurado el Curiosity Nano PIC18F57Q43 para que entregue voltaje de 5V a través del pin VTG (ventana de propiedades del PKOB nano dentro de propiedades del proyecto y opción Power)
- Revisar siempre los mensajes en la ventana de “output” por posibles fallos en el evento de compilación y/o evento de programación.
- Tener a la mano un multímetro para verificar voltajes y continuidad de conexiones en el prototipo implementado

Ejercicios adicionales:

- Desarrollar un titilador de un LED conectado en RE0 en el cual su periodo de parpadeo dependerá del estado de un switch conectado en RB0, si RB0=1 el periodo será de 500ms, si RB0=0 el periodo será de 100ms.
- Desarrollar una señal de cruce de tren con entrada de activación mediante el uso de un switch.
- Desarrollar una “vela electrónica” con entrada de activación, dicha entrada tendrá como sensor de luz a un L.D.R.

Fin de la sesión