

# Microcontroladores

## Semana 3

Semestre 2023-2

Profesor: Kalun José Lau Gan

1

### 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)

2

## Preguntas previas:

- He visto videos de youtube y algunos libros (el del pic16f84 supongo de jose 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, pero se recomienda utilizar lenguaje de alto nivel ya que luego de emplear esos módulos muy posiblemente necesites hacer operaciones matemáticas (escalamiento, filtrado, promediación, 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.

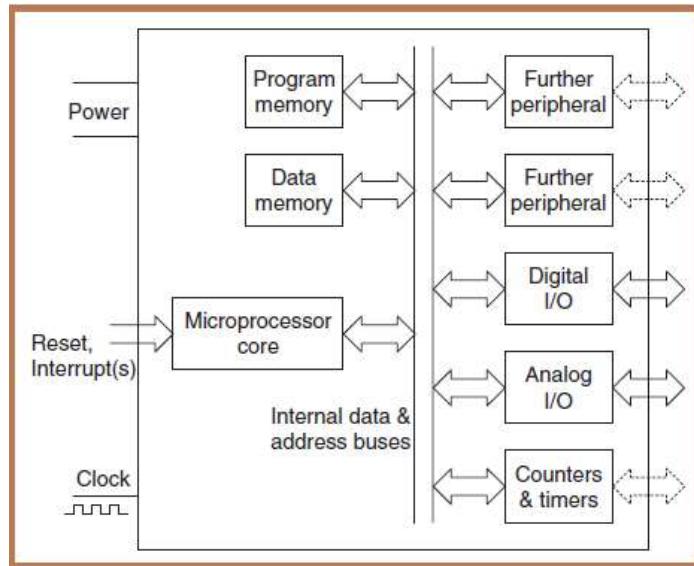
3

## Agenda:

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

4

## Estructura interna de un microcontrolador



- 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

5

## El Curiosity Nano PIC18F57Q43 de Microchip

- 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

Address	Device
00 0000h to 00 3FFFh	PIC18F57Q43
00 4000h to 00 7FFFh	Program Flash Memory (64 KW) <sup>(1)</sup>
00 8000h to 00 FFFFh	
01 0000h to 01 FFFFh	
02 0000h to 1F FFFFh	Not Present <sup>(2)</sup>
20 0000h to 20 003Fh	User ID <sup>(3)</sup> (32 Words) <sup>(3)</sup>
20 0040h to 2B FFFFh	Reserved
2C 0000h to 2C 009Fh	Device Information Area (DIA)
2C 0100h to 2F FFFFh	Reserved
30 0000h to 30 0009h	Configuration Bytes <sup>(2)</sup>
30 000Ah to 37 FFFFh	Reserved
38 0000h to 38 03FFh	Data EEPROM (1024 Bytes)
38 0400h to 3B FFFFh	Reserved
3C 0000h to 3C 0009h	Device Configuration Information
3C 000Ah to 3F FFFFh	Reserved
3F FFFFCh to 3F FFFDh	Revision ID (1 Word) <sup>(2,4,5)</sup>
3F FFFEh to 3F FFFFh	Device ID (1 Word) <sup>(2,4,5)</sup>

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.

6

# El Curiosity Nano PIC18F57Q43 de Microchip

- Estructura de la memoria de datos del PIC18F57Q43:
    - Memoria del tipo volátil (se borra el contenido en un PoR)
    - A diferencia del PIC18F45K50, la memoria RAM esta mapeada a partir del Bank 5 (500H) y los registros de funciones especiales (SFR) se encuentran entre Bank 0 y Bank 4
    - Tener en cuenta que la RAM de datos es de 8Kbyte

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	125FFH		
														2400H		
														24FFH		

Bank	BSR addr[13:8]	addr[7:0]	PI18F x70243	
0	'b00 0000	0x00-0xFF		
1	'b00 0001	0x00-0xFF		
2	'b00 0010	0x00-0xFF		
3	'b00 0011	0x00-0xFF		
4	'b00 0100	0x00-0xFF		
	'b00 0100	0x00-0xFF		
5	'b00 0101	0x00-0xFF		
	'b00 0101	0x00-0xFF		
6	'b00 0110	0x00-0xFF		
7	'b00 0111	0x00-0xFF		
8	'b00 1000	0x00-0xFF		
9	'b00 1001	0x00-0xFF		
10	'b00 1010	0x00-0xFF		
11	'b00 1011	0x00-0xFF		
12	'b00 1100	0x00-0xFF		
13	'b00 1101	0x00-0xFF		
14	'b00 1110	0x00-0xFF		
15	'b00 1111	0x00-0xFF		
16	'b01 0000	0x00-0xFF		
17	'b01 0001	0x00-0xFF		
18	'b01 0010	0x00-0xFF		
19	'b01 0011	0x00-0xFF		
20	'b01 0100	0x00-0xFF		
21	'b01 0101	0x00-0xFF		
22	'b01 0110	0x00-0xFF		
23	'b01 0111	0x00-0xFF		
24	'b01 1000	0x00-0xFF		
25	'b01 1001	0x00-0xFF		
26	'b01 1010	0x00-0xFF		
27	'b01 1011	0x00-0xFF		
28	'b01 1100	0x00-0xFF		
29	'b01 1101	0x00-0xFF		
30	'b01 1110	0x00-0xFF		
31	'b01 1111	0x00-0xFF		
32	'b10 0000	0x00-0xFF		
33	'b10 0001	0x00-0xFF		
34	'b10 0010	0x00-0xFF		
35	'b10 0011	0x00-0xFF		
36	'b10 0100	0x00-0xFF		
37	'b10 0101	0x00-0xFF		
38	'b10 0110	0x00-0xFF		
to	-	-		
63	'b11 1111	0x00-0xFF		
				GPR
				SFR
				Fast SFR
				Unimplemented

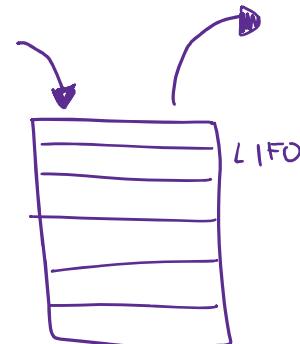
7

Memorias “Stack” o de pila (apilamiento)

- Estructura FIFO  
(first in – first out)



- Estructura LIFO  
(last in – first out)



8

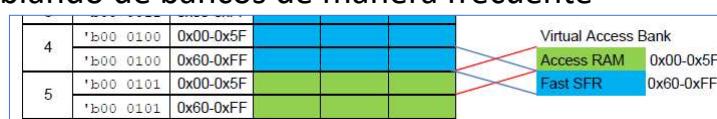
## Aspectos relacionados con el MPASM y el XC8 PIC Assembler

- El año 2014 Microchip dejó de lado el MPASM para dar lugar al XC8 PIC Assembler (PIC-AS)
- Nuevos diseños deberán emplean XC8 PIC Assembler en lugar de MPASM.
- XC8 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

9

## 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”

10

## 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 *access bank*  
movwf TRISB, b *BSR*

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

MOVWF	Move W to f		
Syntax	MOVWF f [,a]		
Operands	0 ≤ f ≤ 255 a ∈ [0, 1]		
Operation	(W) → 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 ≤ 95 (5Fh). See Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Mode for details.		
Words	1		
Cycles	1		
Q Cycle Activity:			
Q1	Q2	Q3	Q4
Decode	Read W	Process Data	Write register 'f'
<b>Example: MOVWF REG, 0</b>			
<b>Before Instruction</b>			
W = 4Fh REG = FFh			
<b>After Instruction</b>			
W = 4Fh REG = 4Fh			

11

## MPASM vs XC8 PIC Assembler: Partes de un programa

MPASM:

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

; aqui declaramos los bits de configuración
CONFIG FOSC = XT_XT
CONFIG PWRT = ON
CONFIG BOR = OFF
CONFIG WDT = OFF
CONFIG PBADEN = OFF
CONFIG IVP = 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, 0
    bcf TRISD, 0, 0

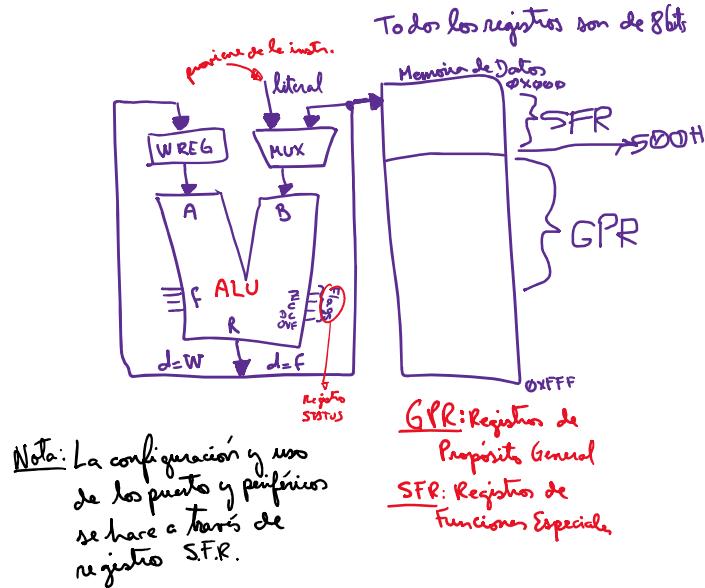
principal:
    btfss PORTB, 0
    goto principal
    btg LATD, 0, 0
otro:
    btfsc PORTB, 0
    goto otro
    goto principal

end rstVect
```

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

12

## El flujo de datos en el microcontrolador PIC18F57Q43



13

## 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	0	0	0	0	0
	de la ALU							

14

## 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/decremeno 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)

15

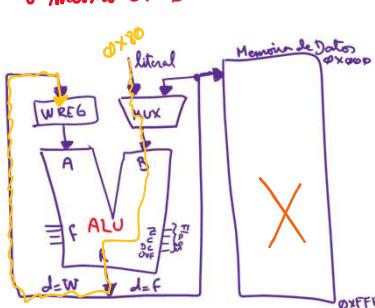
## Instrucciones básicas en XC8 PIC Assembler

### • Instrucciones de movimiento de datos

#### movlw [literal]

-mover un literal hacia WREG

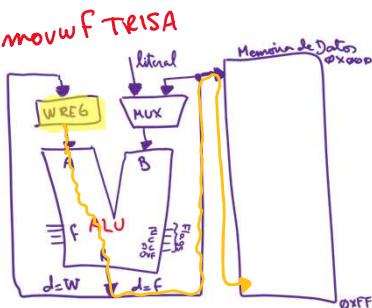
Ej: movlw 0x80



#### movwf [registro], d , a

-mover el contenido de WREG hacia [registro]

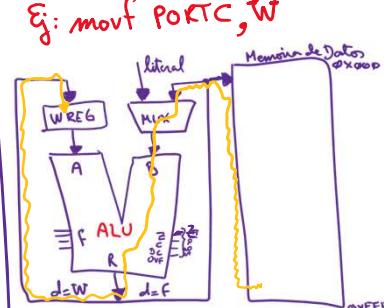
Ej: movwf TRISA



#### movf [registro], d , a

mover el contenido de [registro] y lo muere según "d".

Ej: movf PORTC,W

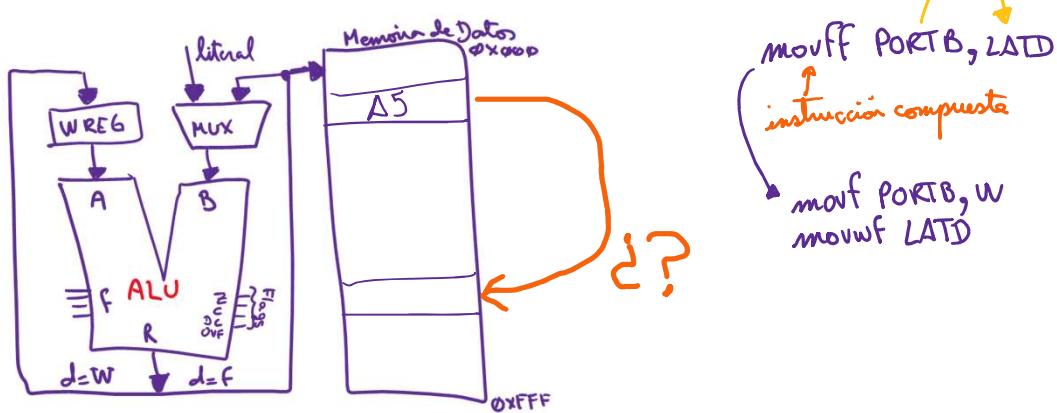


Note: movf [registro], f sirve para act. flag

16

Instrucción movff [registro1], [registro2]

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



17

## ¿Instrucción movfw?

- ¡Instrucción no documentada en la hoja técnica!
  - Instrucción “legacy”
  - 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: morfw TRISA       similar  
                        → morf TRISA, w

18

## Instrucciones básicas en XC8 PIC Assembler

- Instrucciones de manipulación de bits en un registro

bsf [registro], #bit, ~  
coloca a "1" el #bit de [registro]

Ej:  
TRISB  
  
 quiero ponerlo a "1"  
bsf TRISB, 3

TRISB

bcf [registro], #bit, ~  
coloca a "0" el #bit de [registro]

Ej:  
TRISB  
  
bcf TRISB, 7  
TRISB

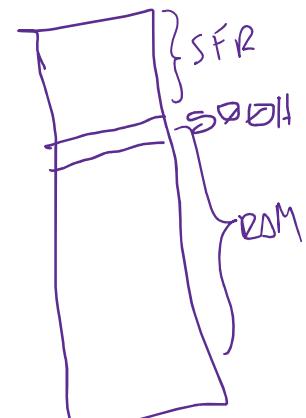
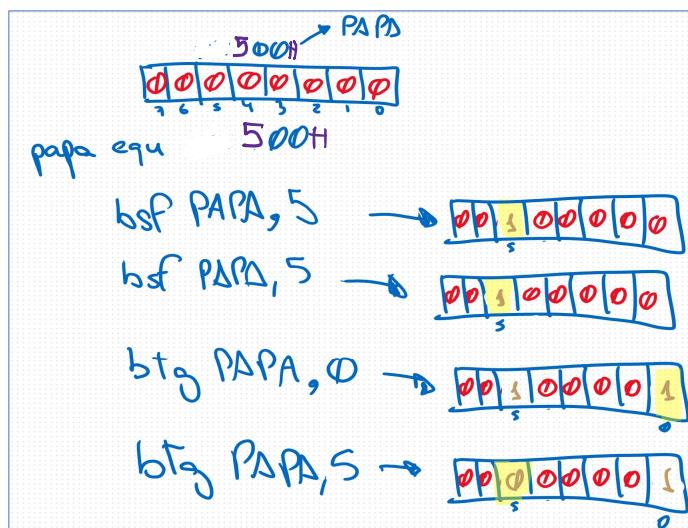
btg [registro], #bit, ~  
aplica complemento al #bit de [registro]

Ej: btg TRISB, 6

TRISB  
  
 ↓  
 0

19

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



20

## Instrucciones básicas en XC8 PIC Assembler

`bra [etiqueta]` ó  
`goto [etiqueta]`  
 salto incondicional

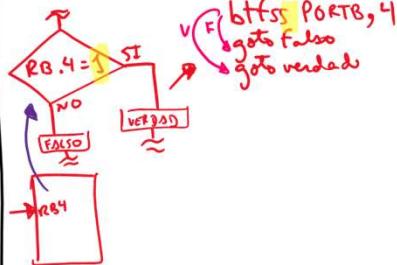
Ej: bucle: `bsf LATD, 6`  
`goto bucle`

`call [etiqueta] ... return`  
 salto a subrutina

Ej:  
`call previo`  
 previo: `return`  
 retorna de donde saltó

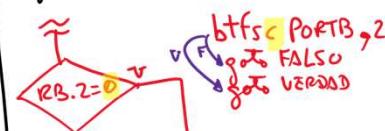
`btfss [registro], #bit, a`

pregunta si el #bit de [registro] es "1"



`btfsc [registro], #bit, a`

pregunta si el #bit de [registro] es igual a "0"



`mop`

instrucción de no operación

lo podemos considerar como un micro retraso

21

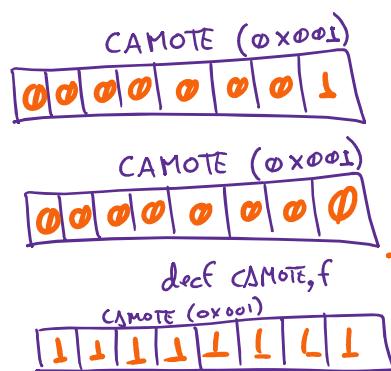
## Instrucciones básicas en XC8 PIC Assembler

- Instrucción decf / incf

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

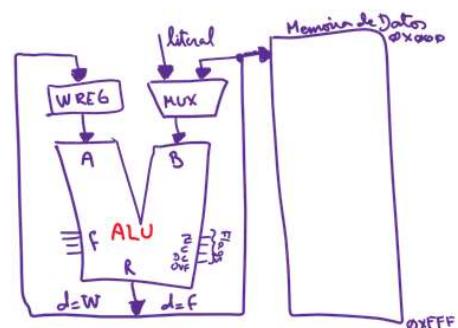
Ej: `decf [registro], d, a`    `incf [registro], d, a`

"d" puede ser: f ó w



`decf CAMOTE, f`

Registro STATUS: Z = 1



22

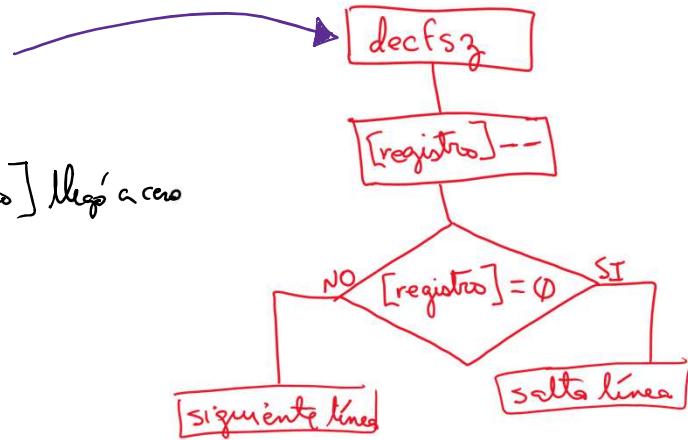
## Instrucciones básicas en XC8 PIC Assembler

*decfsz [registro], d*

decrementa y pregunta si [registro] llegó a cero

*incfsz [registro], d*

incrementa y pregunta si [registro] llegó a cero



23

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

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

*ejemplo:* TRISB

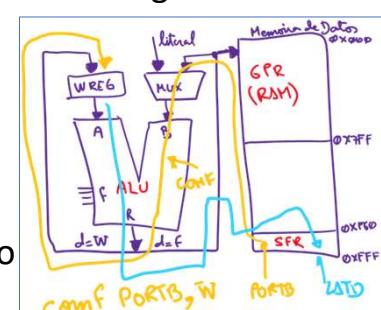
`setf TRISB`

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

*ejemplo:* TRISB

`clrf TRISB`

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



24

# Tiempo de ejecución de las instrucciones en XC8 PIC Assembler

- Se utiliza la siguiente fórmula:

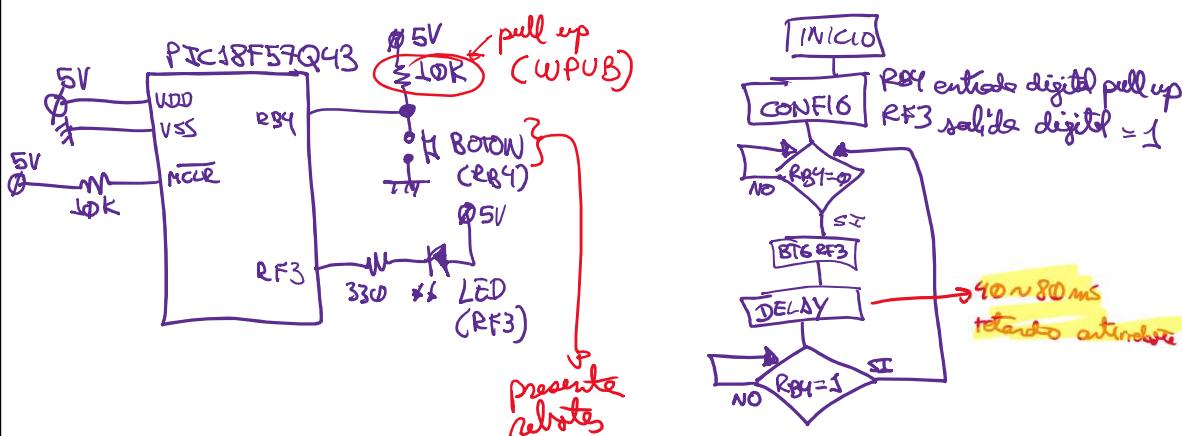
$$t_{osc} = \left( \frac{r_{osc}}{4} \right)^{-\frac{1}{2}}$$

- Hay instrucciones simples, dobles y especiales (revisar 26.0 de la datasheet)
  - Recordando la relación periodo vs frecuencia:  $f = \frac{1}{T}$

25

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

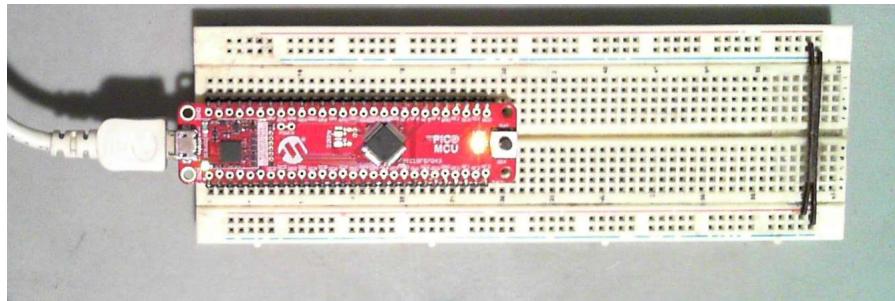
- Latch: Dispositivo enclavador – Mantiene el valor de salida hasta que pulses el botón



26

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

- Hardware (solo el Curiosity Nano)



27

## Ejemplo 02: Desarrollar un dispositivo LATCH 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    bcf TRISF, 3, 1
24    bcf ANSELF, 3, 1
25    bsf LATF, 3, 1      ;LED apagado en un inicio

27   inicio:
28    btfsc PORTB, 4, 1  ;Pregunta si RB4 es cero (si presione el boton)
29    bra inicio         ;falso, regresa a preguntar
30    btg LATF, 3, 1     ;complemento a RF3
31    call retardon
32   otro:
33    btfss PORTB, 4, 1  ;Pregunta si RB4 es uno (si solte el boton)
34    bra otro           ;falso, vuelvo a preguntar
35    bra inicio

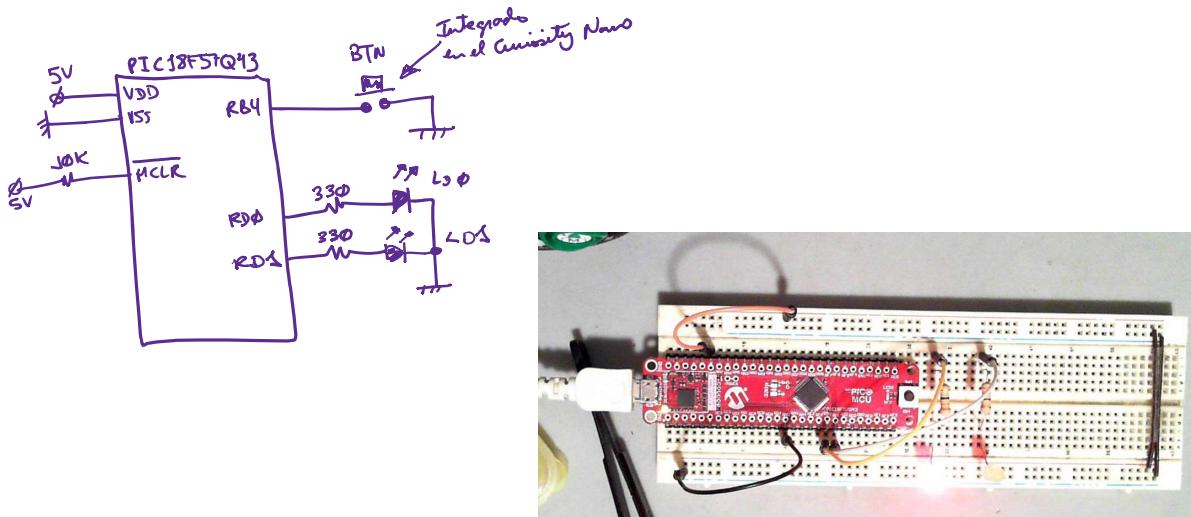
37   retardon:
38    nop
39    nop
40    nop
41    nop
42    nop
43    nop
44    nop
45    nop
46    nop
47    nop
48    nop
49    nop
50    nop
51    nop
52    nop
53    nop
54    nop
55    nop
56    nop
57    nop
58    nop
59    nop
60    nop
61    nop
62    nop
63    nop
64    nop
65    nop
66   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

```

28

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

- Diseño del hardware



29

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

- Código en XC8 PIC Assembler

```

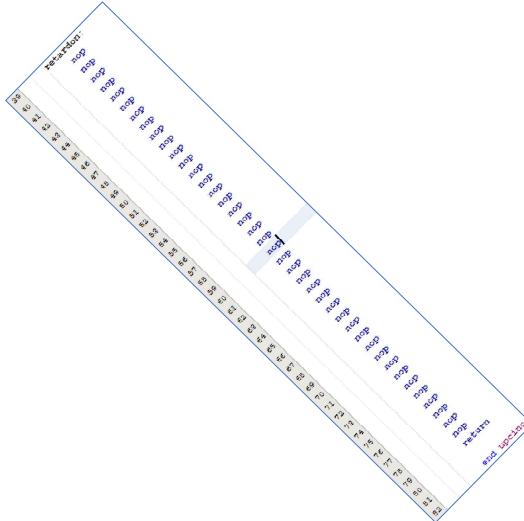
1      PROCESSOR 18F57Q43
2      #include "cabecera.inc"
3
4      PSECT upcino, class=CODE, reloc=2, abs
5      upcino:
6          ORG 000000H
7          bra configuro
8
9          ORG 000020H
10         configuro:
11             movlb 0H
12             movlw 60H
13             movwf OSCCON1, 1
14             movlw 02H
15             movwf OSCFRCQ, 1
16             movlw 40H
17             movwf OSCEN, 1
18
19             movlb 4H
20             bsf TRISB, 4, 1
21             bcf ANSELB, 4, 1
22             bsf WDUB, 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
28
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

```

30

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

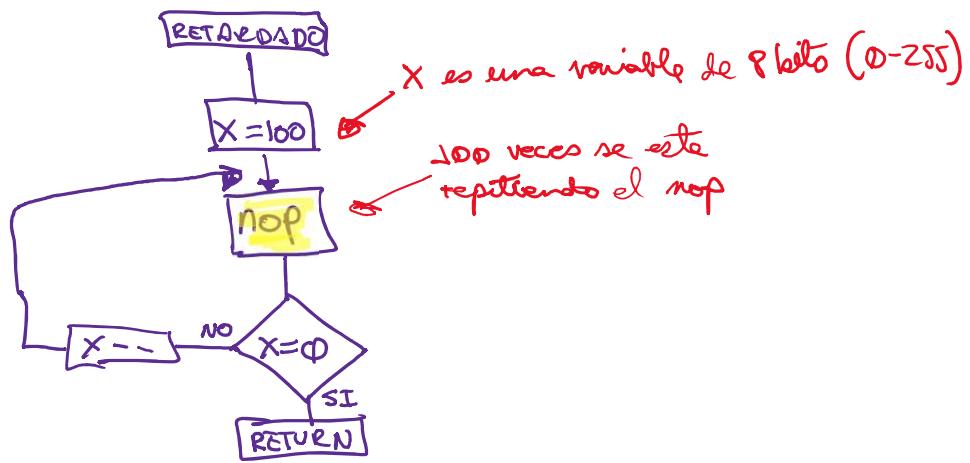
- Escribir 40000 nops no es práctico ni eficiente!



31

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

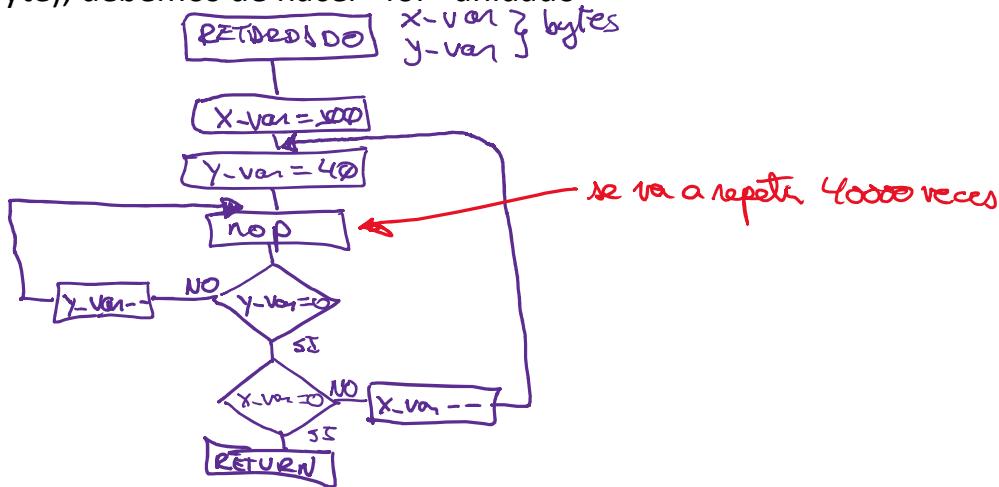
- Usar muchos nops ocupa demasiado espacio
  - Se puede armar una rutina de bucle de repetición (for)



32

## ¿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



33

## ¿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     otro3:
45         movlw 40
46         movwf y_var, 1
47     otro2:
48         nop
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
    
```

34

## 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  $330\Omega$ ). 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.

35

## 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

36

## 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.

37

## Fin de la sesión

38