

Microcontroladores

Semana 3

Semestre 2024-1

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 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 numéricicas (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, 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.

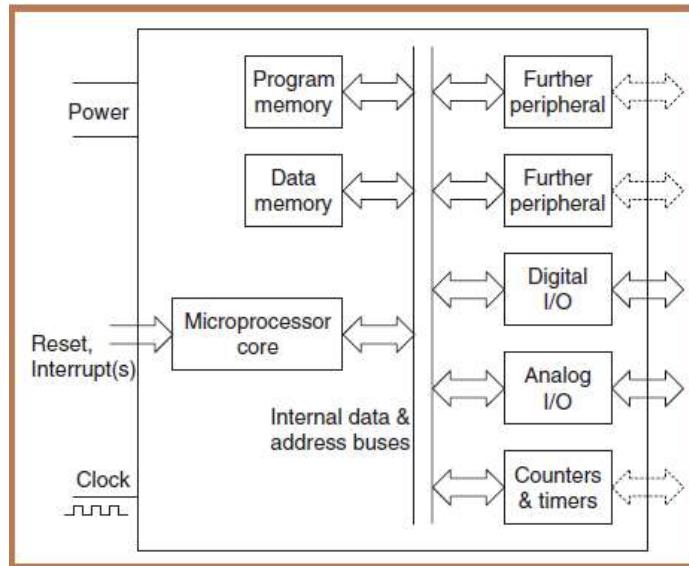
3

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 Microcontrolador 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
 - Rango de direcciones: 000000H-3FFFFH (22 bits – 4Mbyte de direcciones)

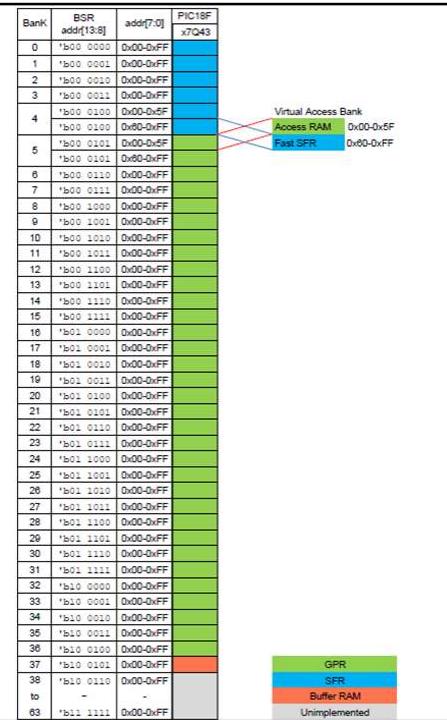
Address	Device
00 0000h to 00 3FFFh	PIC18F57Q43
00 4000h to 00 7FFFh	Program Flash Memory (64 KW) ⁽¹⁾
00 8000h to 00 FFFFh	
01 0000h to 01 FFFFh	
02 0000h to 1F FFFFh	Not Present ⁽²⁾
20 0000h to 20 003Fh	User IDs (32 Words) ⁽³⁾
20 0040h to 2B FFFFh	Reserved
2C 0000h to 2C 00FCh	Device Information Area (DIA) ⁽⁴⁾
2C 0100h to 2F FFFFh	Reserved
30 0000h to 30 0009h	Configuration Bytes ⁽⁵⁾
30 000Ahn to 37 FFFFh	Reserved
38 0000h to 38 002Fh	Data EEPROM (1024 Bytes)
38 0040h 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) ^(2,A)
3F FFFF Eh to 3F FFFFh	Device ID (1 Word) ^(2,A)

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 – Power-on Reset)
 - A diferencia del PIC18F45K50, la memoria RAM está 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)



7

Recordando: Estructuras anidadas

En el lenguaje C:

```

unsigned char x_var = 0;
for(x_var = 0; x_var < 10; x_var++) {
}

```

limite → 255

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

unsigned char $\lambda_var = \emptyset$, $\gamma_var = \emptyset$.
 for ($x_var = \emptyset$; $x_var < 2000$; x_var++) {
 for ($y_var = \emptyset$; $y_var < 2000$; y_var++) {
 } i Cuántas veces se repite? $\rightarrow 4000 \times 4000$
 }
 }

8

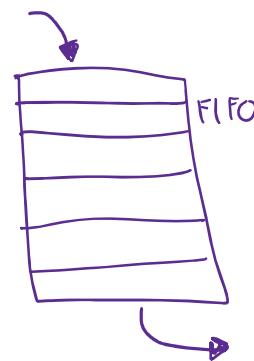
Recordando saltos a sub-rutinas

- ¿Cómo sabe el CPU que debe de retornar de donde saltó?

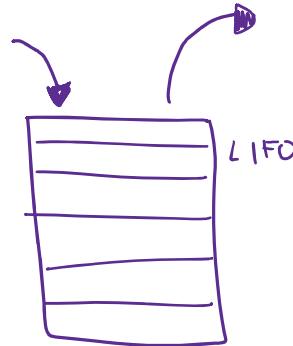
9

Memorias “Stack” o de pila (apilamiento)

- Estructura FIFO
(first in – first out)



- Estructura LIFO
(last in – first out)



10

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.

11

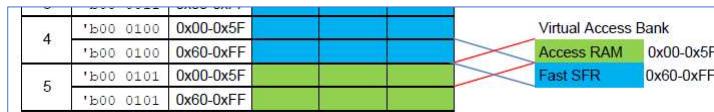
¿Por qué aprender Assembler (más difícil) si en el C también se puede hacer?

- En el assembler se prioriza en la eficiencia (espacio empleado en la memoria de programa y de datos, control detallado del consumo energético)
- 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)

12

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”

13

Repertorio de instrucciones en Assembler del PIC18

- Revisar capítulo 44 de la hoja técnica del PIC18F57Q43

14

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 ≤ 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					
Decode		Read W		Process Data					
Q4		Write register f'							
Example: MOVWF REG, 0									
Before Instruction									
W = 4Fh									
REG = FFh									
After Instruction									
W = 4Fh									
REG = 4Fh									

15

MPASM vs XC8 PIC Assembler: Partes de un programa

MPASM:

```
list p=18f4550
#include<p18f4550.inc>
; Aquí 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, 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

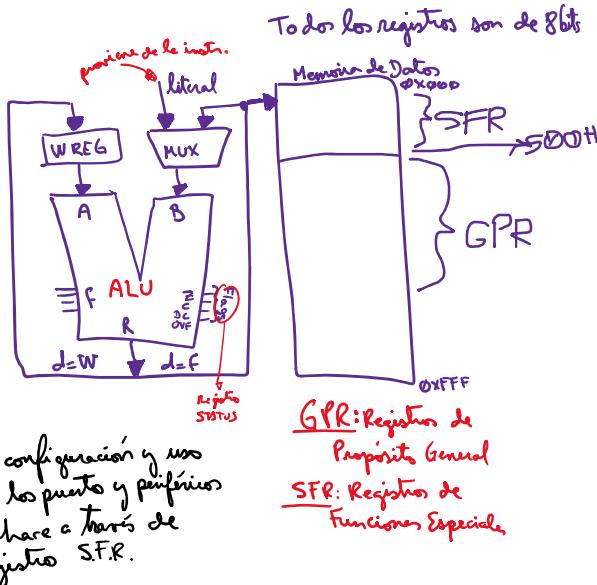
otro:
    btfsc PORTB, 0
    goto otro
    goto principal

end rstVect
```

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

16

El flujo de datos en el microcontrolador PIC18F57Q43



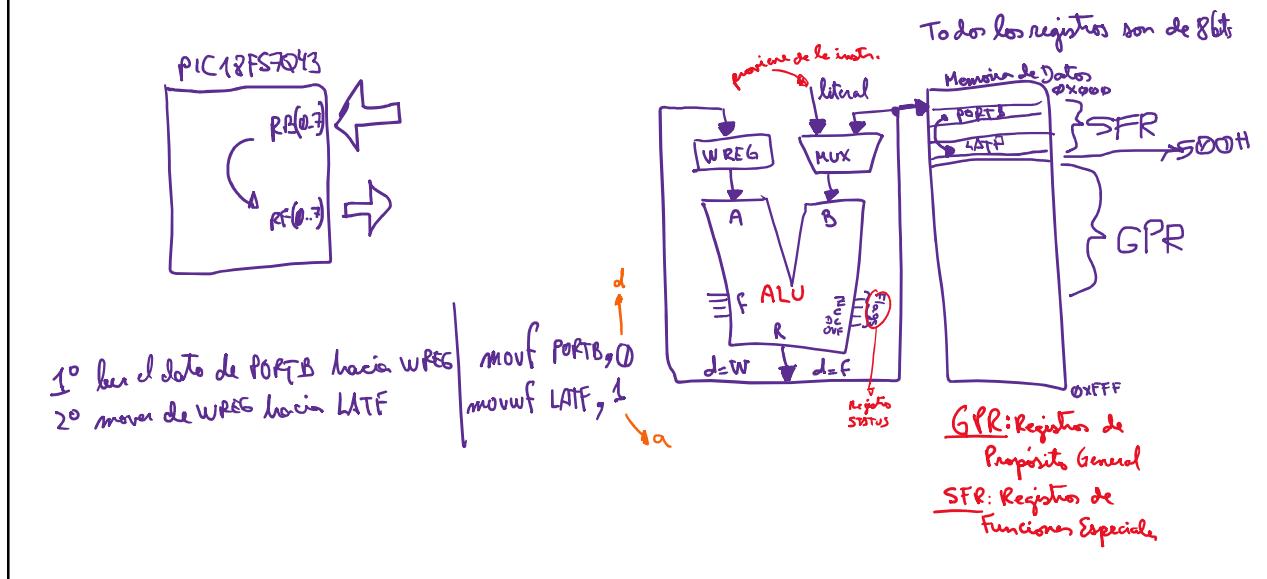
17

Comentarios y Notas:

- En los programas desarrollados en XC8 PIC Assembler, la mayor cantidad de instrucciones serán las de movimiento de datos.

18

Ejemplo: Pasar un dato del puerto B a al puerto F



19

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	0

de la ALU

20

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)

21

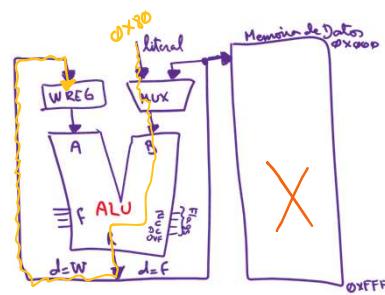
Instrucciones básicas en XC8 PIC Assembler

- Instrucciones de movimiento de datos

movlw [literal]

-mover un literal hacia WREG

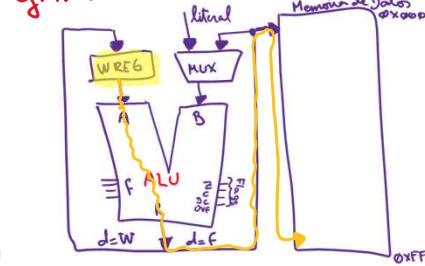
Ej: movlw 0x80



movwf [registro], a

-mover el contenido de WREG hacia [registro]

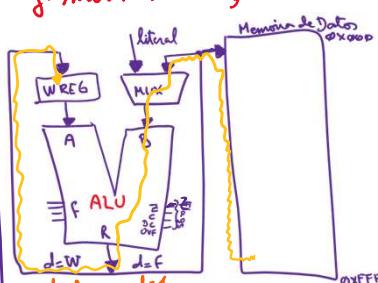
Ej: movwf TRISA



movf [registro], d, a

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

Ej: movf PORTC, W

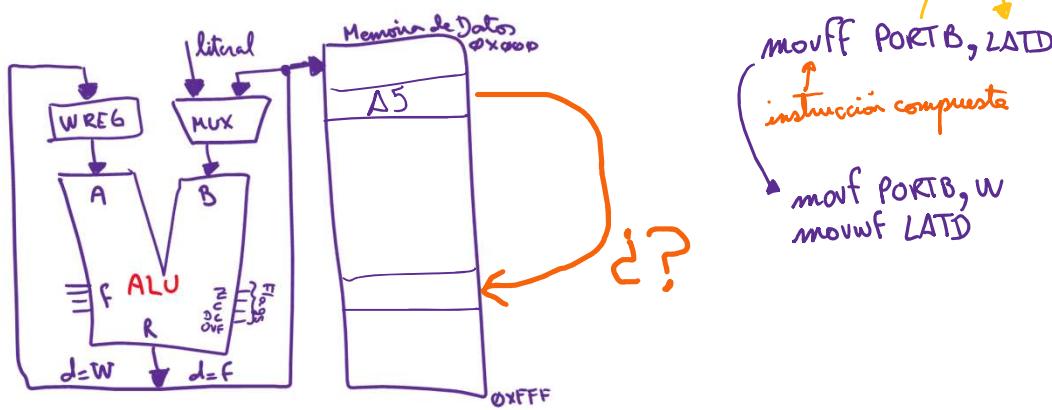


Note: movf [registro], f aviso para not. flag

22

Instrucción movff [registro1], [registro2]

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



23

¿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: movfw TRISA simila
 \rightarrow movf TRISA, w

24

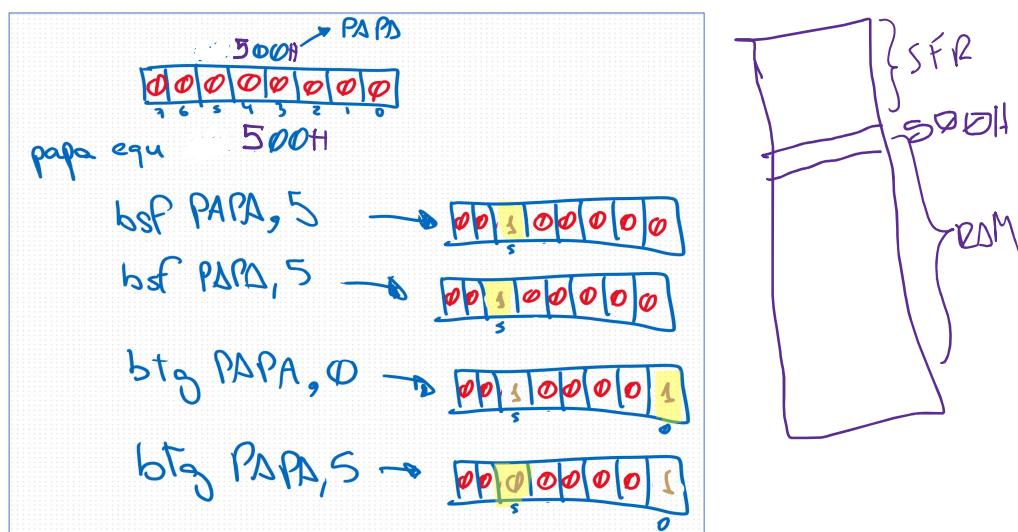
Instrucciones básicas en XC8 PIC Assembler

- Instrucciones de manipulación de un bit determinado, en un registro

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

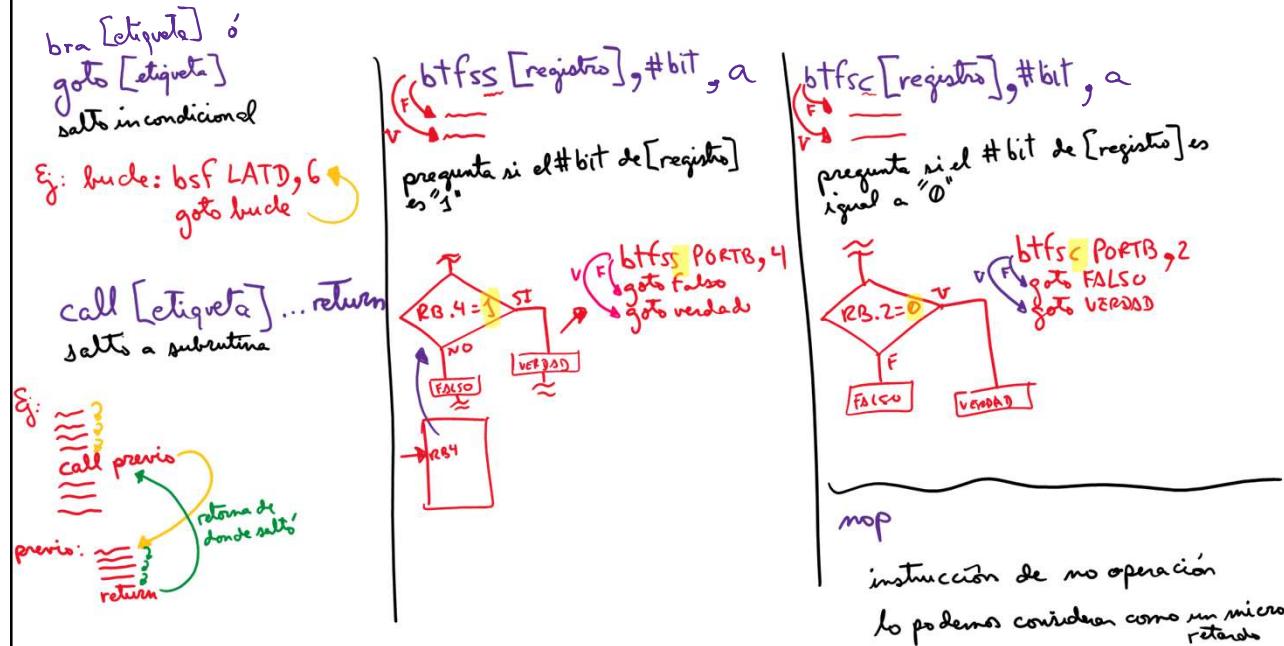
25

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



26

Instrucciones básicas en XC8 PIC Assembler

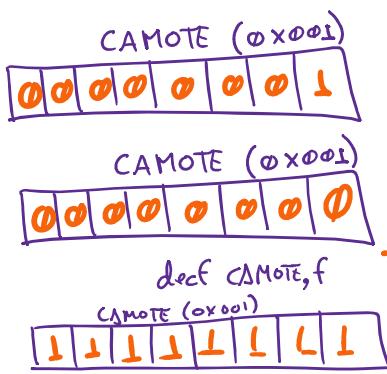


27

Instrucciones básicas en XC8 PIC Assembler

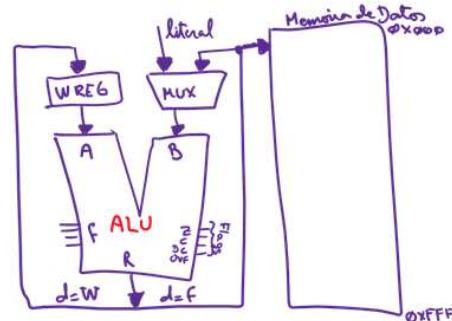
- Instrucción decf / incf
 - Decremento (decf) o incremento (incf) de registro, ambos de **uno en uno**

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



decf CAMOTE, f

Registro STATUS: Z=1

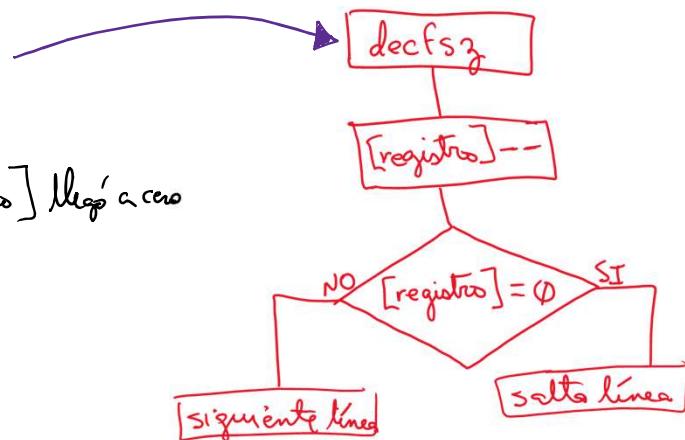


28

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

29

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

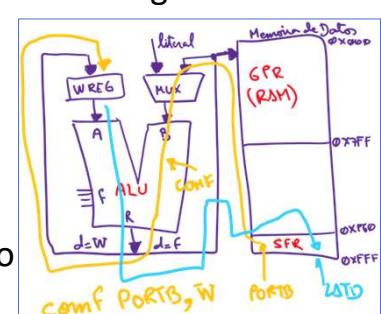
- **clrf [registro]** : Coloca todos los bits del registro indicado a cero lógico

ejemplo: TRISB
10110001

clrf TRISB

TRISB
00000000

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



30

Tiempo de ejecución de las instrucciones en XC8 PIC Assembler

- Se utiliza la siguiente fórmula:

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

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

$$f = \frac{1}{T}$$

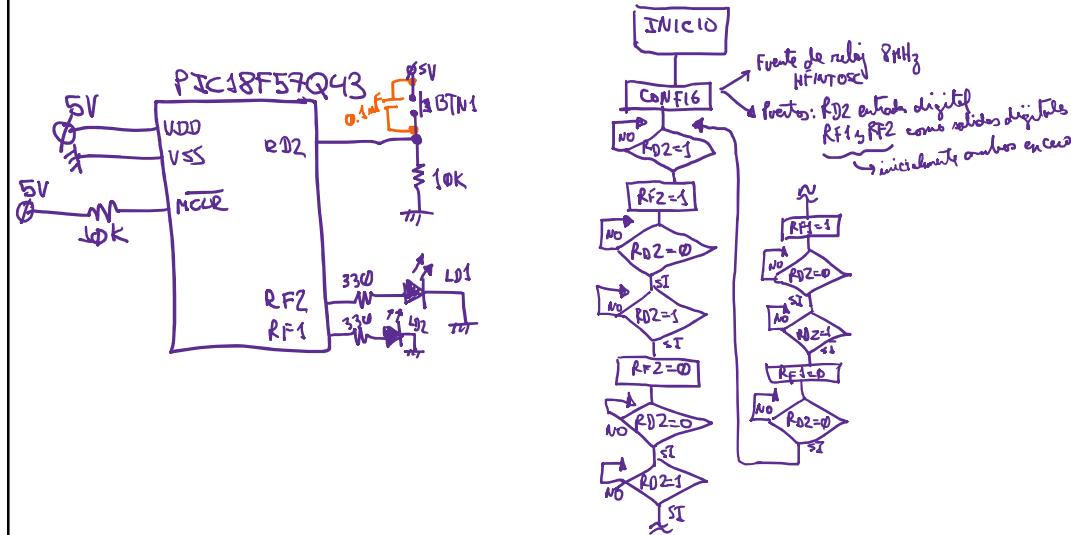
31

Ejemplo de tiempo de ejecución

32

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

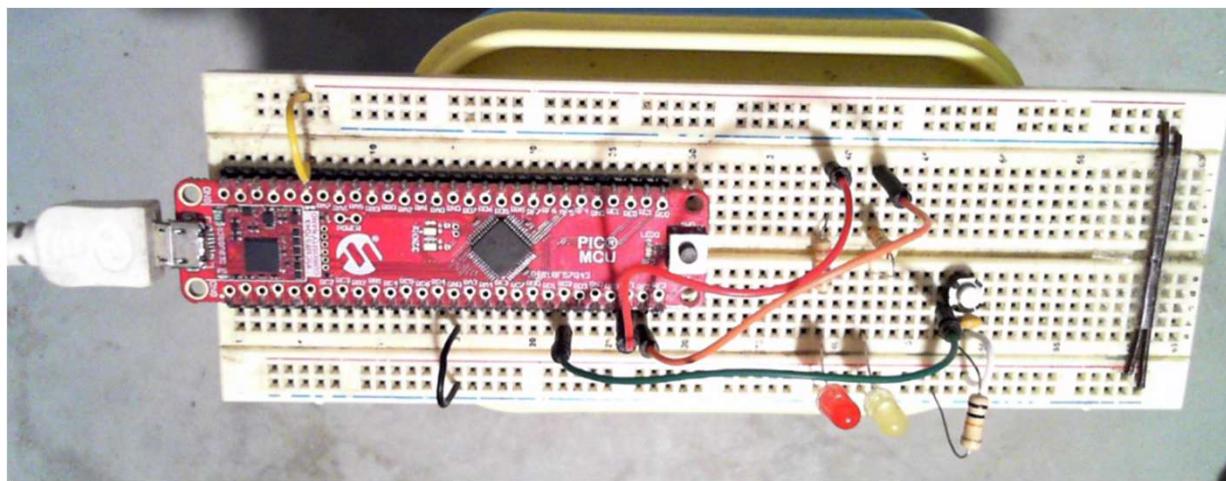
- Latch: Dispositivo enclavador – Mantiene el valor de salida hasta que pulses el botón



33

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

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



34

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

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

```

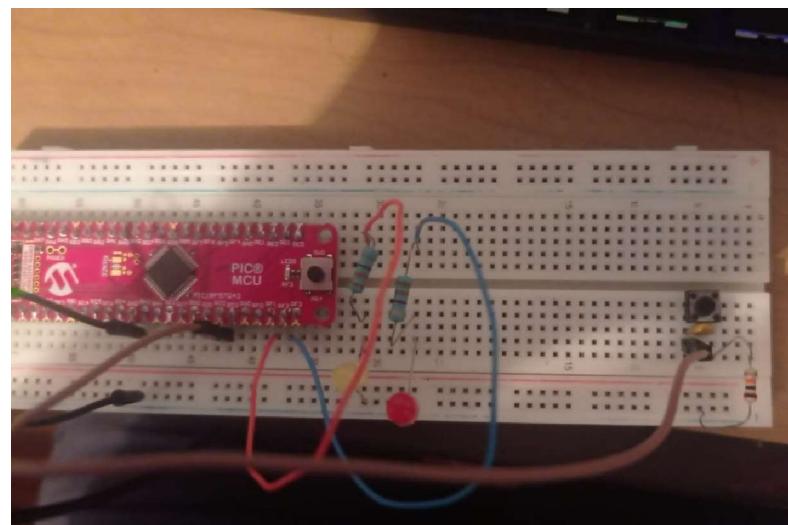
1  PROCESSOR 18F57Q43
2  #include "cabecera.inc"
3
4  PSECT upcino, class=CODE, reloc=2, abs
5  upcino:
6    ORG 000000H          ;vector de reset
7    bra configuro        ;salto a label configuro
8
9  configuro:             ;label configuro
10   movlb 0H              ;al BANK0
11   movlw 60H
12   movwf 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   bsf TRISD, 2, 1       ;RD2 como entrada
19   bcf ANSELB, 2, 1       ;RD2 como digital
20   bcf TRISF, 1, 1       ;RF1 como salida
21   bcf ANSELF, 1, 1      ;RF1 como digital
22   bcf TRISF, 2, 1       ;RF2 como salida
23   bcf ANSELF, 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
35 otroal:
36   btfss PORTD, 2, 1      ;pregunto si presione el boton
37   bra otroal              ;verdad, apago el LED
38
39 otro2:
40   btfsc PORTD, 2, 1      ;pregunto si solte el boton
41   bra otro2              ;falso, no presione el boton y vuelvo a preguntar
42
43 otro3:
44   btfsc PORTD, 2, 1      ;pregunto si deje de presionar el boton
45   bra otro3              ;falso y vuelvo a preguntar
46
47 otro4:
48   btfss PORTD, 2, 1      ;pregunto si pulse el boton
49   bra otro4              ;verdad, apago el LED
50
51 otro3:
52   btfsc PORTD, 2, 1      ;pregunto si deje de presionar el boton
53   bra otro3              ;falso y vuelvo a preguntar
54
55   bra inicio             ;regreso a label inicio
56
56 end upcino

```

35

Error en la implementación

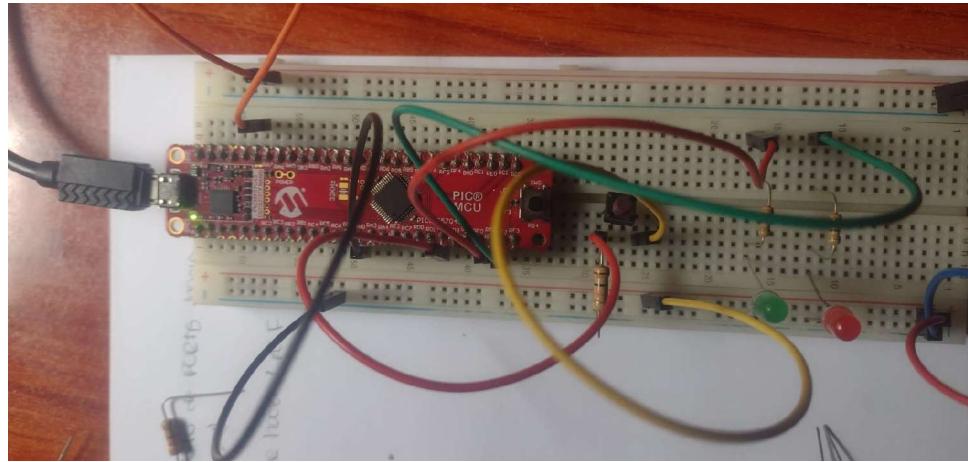
- “Profesor compila pero no funciona”
- Ubica el error:



36

Error en la implementación

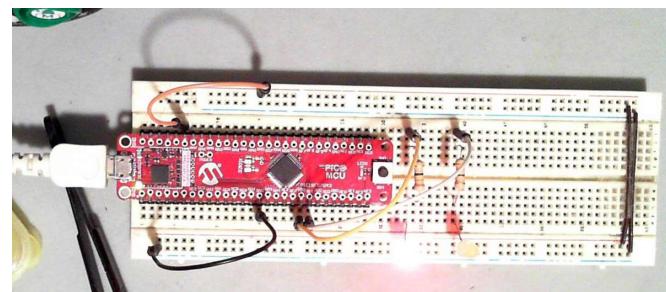
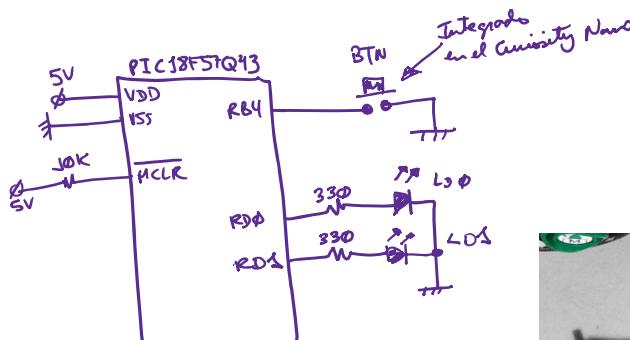
- “Profesor compila pero no funciona”
- Ubica el error:



37

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

- Diseño del hardware



38

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

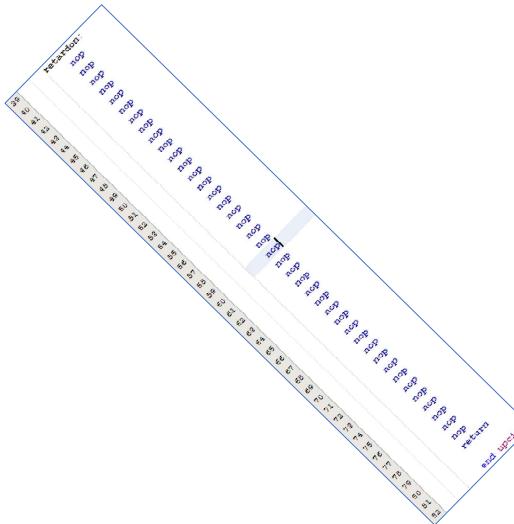
39      retardon:
40          nop
41          nop
42          nop
43          nop
44          nop
45          nop
46          nop
47          nop
48          nop
49          nop
50          nop
51          nop
52          nop
53          nop
54          nop
55          nop
56          nop
57          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          nop
79          nop
80          return
81
82      end upcino

```

39

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

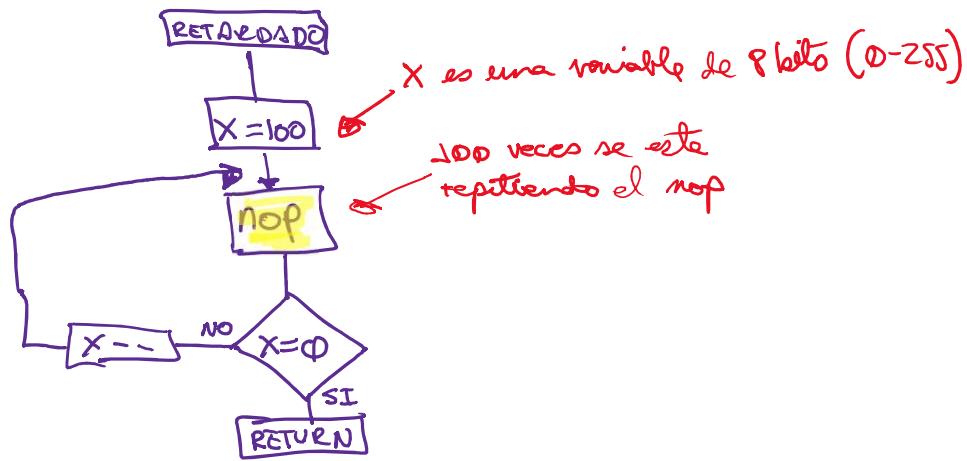
- Escribir 40000 nops no es práctico ni eficiente!



40

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

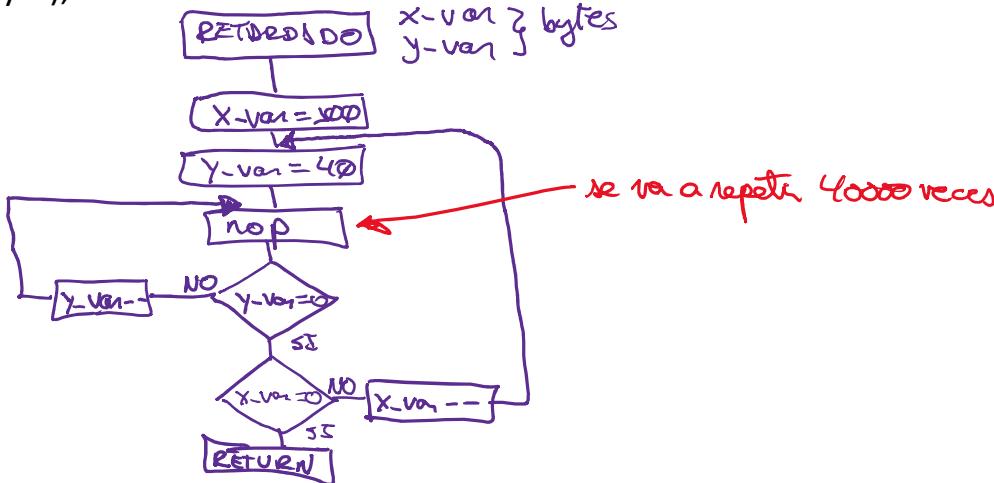
- Usar muchos nops ocupa demasiado espacio
- Se puede armar una rutina de bucle de repetición (for)



41

¿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



42

¿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

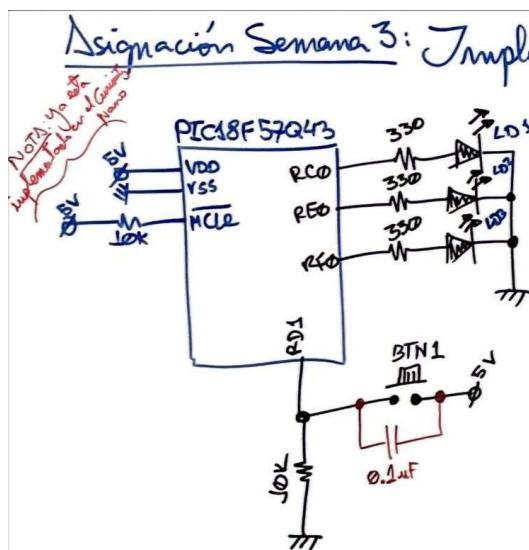
```

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

```

43

Ejemplo 2024-1



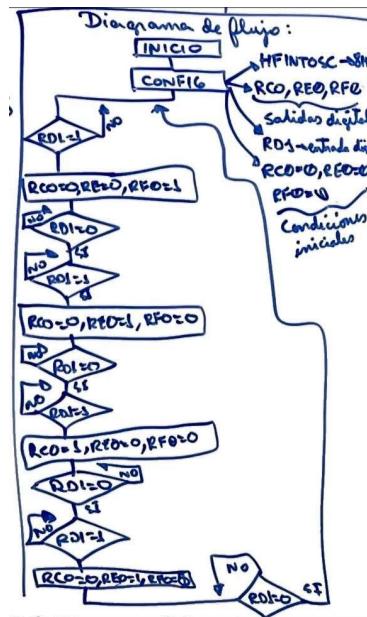
Funcionamiento: Al pulsar BTN1 se hará el cambio de visualización en los LED's según tabla siguiente:

	LD1	LD2	LD3
S1	OFF	OFF	ON
S2	OFF	ON	OFF
S3	ON	OFF	OFF
S4	OFF	ON	OFF

44

Ejemplo 2024-1

- Diagrama de flujo:



45

Ejemplo 2024-1

- Código:

```

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
7    bra configuro
8
9    ORG 000100H
10   configuro:
11    movlw 0H
12    movlw 60H
13    movwf OSCCON1, 1
14    movlw 03H
15    movwf OSCFRQ, 1
16    movlw 40H
17    movwf OSCEN, 1
18    movlw 4H
19    bcf TRISD, 1, 1
20    bcf ANSELD, 1, 1
21    bcf TRISC, 0, 1
22    bcf ANSELC, 0, 1
23    bcf TRISE, 0, 1
24    bcf ANSELE, 0, 1
25    bcf TRISE, 0, 1
26    bcf ANSELF, 0, 1
27    bcf LATC, 0, 1
28    bcf LATE, 0, 1
29    bcf LATF, 0, 1
30
31  inicio:
32    btfss PORTD, 1, 1
33    bra inicio
34    bcf LATC, 0, 1
35    bcf LATE, 0, 1
36    bcf LATF, 0, 1
37  otro1:
38    btfsc PORTD, 1, 1
39    bra otro1
40  otro2:
41    btfss PORTD, 1, 1
42    bra otro2
43    bcf LATC, 0, 1
44    bsf LATE, 0, 1
45    bcf LATF, 0, 1
46  otro3:
47    btfsc PORTD, 1, 1
48    bra otro3
49  otro4:
50    btfss PORTD, 1, 1
51    bfa LATC, 0, 1
52    bsf LATE, 0, 1
53    bcf LATF, 0, 1
54  otro5:
55    btfsc PORTD, 1, 1
56    bra otro5
57    bfa LATC, 0, 1
58  otro6:
59    btfss PORTD, 1, 1
60    bra otro6
61    bcf LATC, 0, 1
62    bsf LATE, 0, 1
63    bcf LATF, 0, 1
64  otro7:
65    btfsc PORTD, 1, 1
66    bra otro7
67    bra inicio
68
69    end upcino
  
```

46

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.

47

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

48

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.

49

Fin de la sesión

50