

Microcontroladores

Laboratorio Semana 2

Semestre: 2022-1

Profesor: Kalun José Lau Gan

1

Preguntas previas:

- ¿Por qué estudiamos Assembler?
 - Manipulamos directamente los recursos disponibles en el microcontrolador
 - Mejora la eficiencia en diferentes aspectos: velocidad, consumo energético, dissipación de calor, costos reducidos, tamaño final del PCB
- ¿Cuál es el lenguaje mas usado en microcontroladores?
 - El lenguaje C y variantes
- Había leído el capítulo 7 del datasheet sobre el Data EEPROM. ¿Qué es y como se usa?
 - Es una memoria no-volátil que es considerado como un periférico en la arquitectura del microcontrolador PIC18F4550. No está mapeado su contenido en la memoria de datos
 - Para acceder a dicha memoria se emplean cuatro registros SFR según se detalla en la hoja técnica:

- EECON1
- EECON2
- EEDATA
- EEADR

2

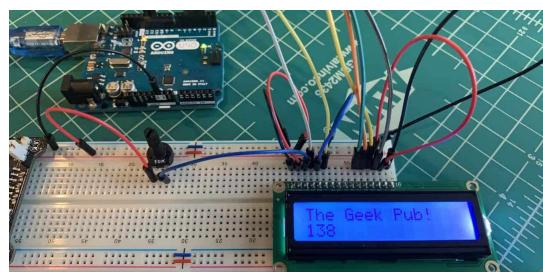
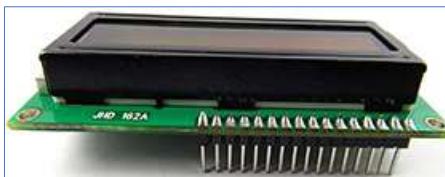
Preguntas previas:

- Por error conseguí un cristal de 8MHz. ¿Podré implementar los ejemplos que se propongan en la clase?
 - Si, pero debes de configurar de manera distinta los bits de configuración: FOSC = HS y CPUDIV = OSC2_PLL3 para que funcione el CPU a 4MHz.
- Me vino con una placa verde el Pickit3. ¿Vamos a emplearlo?
 - No va a ser necesario emplearlo puesto a que vamos a colocar de manera permanente el microcontrolador en el breadboard y la conexión con el pickit3 va a ser de manera directa.
- El osciloscopio solicitado opcionalmente en la lista de materiales aproximadamente en que semana lo vamos a emplear?
 - Desde la cuarta o quinta semana
- No pude conseguir el cable telefónico AWG22 pero he conseguido UTP Cat6, habrá algún problema?
 - No hay problema alguno salvo lo trenzado que están los pares, que se tendrá que “enderezar” para poder usarlos en el breadboard.

3

Preguntas previas:

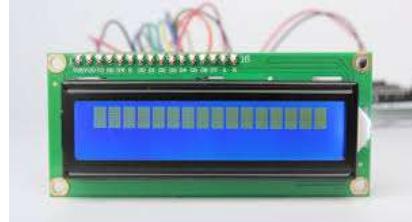
- ¿Cuáles pueden ser las consecuencias “negativas” en cambiar de frecuencia de trabajo de 4MHz a 20MHz?
 - Mayor consumo de energía
 - En el caso de leer el estado de un pulsador, será mas sensible a los rebotes
 - Al trabajar a mayor frecuencia se debe de tener mayor cuidado en el diseño del PCB
 - Ground planes, short paths, etc
- ¿Cómo es lo de soldar las conexiones en el LCD?



4

Preguntas previas:

- ¿Cómo compruebo si mi LCD esta funcionando?
 - Energizamos el display conectando: pin1-GND, pin2-VCC, pin3-GND, pin15-VCC, pin16-GND.



- He conseguido el PCKIT3.5+ . Habrá algún problema?
 - No hay problema alguno siempre y cuando el MPLAB X lo reconozca correctamente y pueda grabar el microcontrolador.



5

Preguntas previas:

- ¿El bit PBADEN para qué sirve?
 - Es el habilitador para que los pines 4 al 0 del puerto B funcionen como E/S digitales (OFF) o como entradas analógicas para el conversor A/D (ON).
- Con el programador viene un CD. ¿Es necesario instalarlo?
 - No, el MPLABX se encarga de todo.

6

Agenda

- El flujo de datos en el microcontrolador PIC18F4550
- Instrucciones básicas en XC8 PIC Assembler
- Ejemplos básicos con el microcontrolador PIC18F4550 en lenguaje Assembler
- Nuestra primera implementación física de un circuito basado en el microcontrolador PIC18F4550

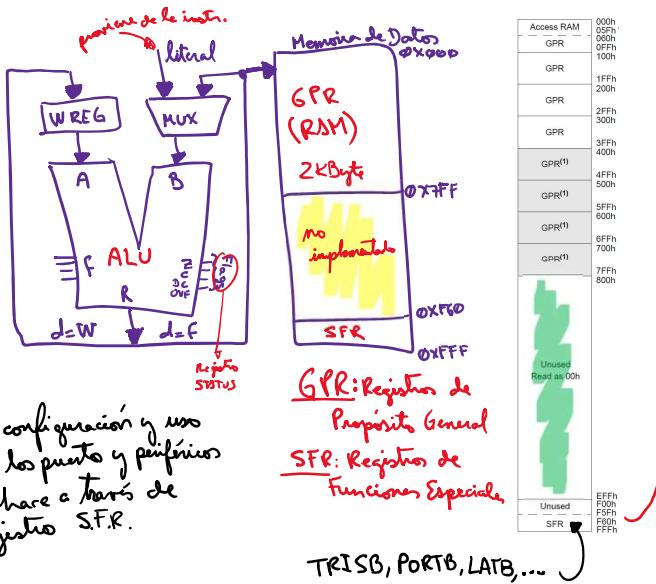
7

Aspectos relacionados con el MPASM y el XC8 PIC Assembler

- El año 2020 Microchip dejó de lado el MPASM para dar lugar al XC8 PIC Assembler (PIC-AS)
- El XC8 Assembler tiene bugs reportados por lo que indicaremos en los ejemplos el cómo solucionarlos (EDIT 28/03/2022 ya fue solucionado en las versiones de MPLABX 6.00 y XC8 2.36)
- 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

8

El flujo de datos en el microcontrolador PIC18F4550



Todos los registros son de 8bits

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFn	INDEX2 ⁽¹⁾	F8Fh	CCCR1H	F9Fh	IPR1	F7Fh	UEP15		
FEFn	TOSU	F8Eh	POSTINC0 ⁽¹⁾	F8Eh	CCCR1L	F9Eh	IPR14		
FDFn	TOSL	F8Dh	POSTDEC0 ⁽¹⁾	F8Dh	CPCR1CON	F9Dh	IPR13		
FFCh	STKPTR	F8Ch	PREINC2 ⁽¹⁾	F8Ch	CPCR2H	F9Ch	IPR12		
FFBh	PCLATU	F8Bh	PLUSW2 ⁽¹⁾	F8Bh	CPCR2L	F9Bh	OSCTUNE		
FFAh	PCLATH	F8Ah	CCP2CON	F8Ah	— ⁽²⁾	F7Ah	UEP10		
FF9h	PCL	F89h	FSR2L	F89h	— ⁽²⁾	F79h	UEP9		
FF8h	TBLPTRU	F88h	BAUDCON	F88h	— ⁽²⁾	F78h	UEP8		
FF7h	TBLPTRH	F87h	TMROH	F87h	ECCP1DEL	F77h	UEP7		
FF6h	TBLPTR	F86h	TMROL	F86h	ECCP1IAS	F76h	TRISE ⁽³⁾		
FF5h	TABLAT	F85h	TOCON	F85h	CVRCON	F75h	TRISD ⁽³⁾		
FF4h	PRODH	F84h	CMCON	F84h	TRISC	F74h	UEP4		
FF3h	PROCL	F83h	OSCCON	F83h	TRISB	F73h	UEP3		
FF2h	HVDCON	F82h	TMRS3	F82h	TRISA	F72h	UEP2		
FF1h	INTCON2	F81h	WDTCON	F81h	— ⁽²⁾	F71h	UEP1		
FF0h	INTCON3	F80h	RCON	F80h	— ⁽²⁾	F70h	UEP0		
FEFn	INDF9 ⁽¹⁾	FCFh	TMRIH	FAFh	SPRNG	F6Fh	UCFG		
FEFn	POSTINC1 ⁽¹⁾	FCFh	TMRL	FAEh	RCREG	F6Eh	UADDR		
FEFn	POSTDEC0 ⁽¹⁾	FCDh	T1CON	FADh	TXREG	F6Dh	LATE ⁽³⁾		
FECh	PREINC1 ⁽¹⁾	FCCh	TMR2	FACh	TXSTA	F6Ch	USTAT		
FEBh	PLUSW0 ⁽¹⁾	FCBh	PR2	FABh	RCSTA	F6Bh	UIEIE		
FEAh	FSR0H	FCAh	T2CON	FAAh	— ⁽²⁾	F6Ah	UIER		
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEDAR	F69h	UIE		
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F68h	UIR		
FE7h	INDF1 ⁽¹⁾	FC7h	SSPSTAT	FA7h	ECON2 ⁽¹⁾	F67h	UFIRM		
FE6h	POSTINC1 ⁽¹⁾	FC6h	SSPCON1	FA6h	ECON1	F66h	UFML		
FE5h	POSTDEC0 ⁽¹⁾	FC5h	SSPCON2	FA5h	— ⁽²⁾	F65h	SPPCON ⁽³⁾		
FE4h	PREINC1 ⁽¹⁾	FC4h	ADRESH	FA4h	— ⁽²⁾	F64h	SPREPS ⁽³⁾		
FE3h	PLUSW1 ⁽¹⁾	FC3h	ADRESL	FA3h	— ⁽²⁾	F63h	SPPCFG ⁽³⁾		
FE2h	FSR1H	FC2h	ADCON1	FA2h	IPR	F62h	SPPDATA ⁽³⁾		
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F61h	— ⁽²⁾		
FE0h	BSR	FC0h	ADCON2	FA0h	PORTA	F60h	— ⁽²⁾		

9

MPASM vs XC8 PIC Assembler: Partes de un programa

MPASM:

```

list p=18f4550
#include<pic18f4550.inc>

; aqui declaramos los bi
CONFIG FOSC = XT_XT
CONFIG PWRT = ON
CONFIG BOR = OFF
CONFIG WDT = OFF
CONFIG PBADEN = OFF
CONFIG IVP = OFF

.org 0x0000
goto configuro

.org 0x0020
configuro:
    bsf TRISB, 0
    bcf TRISD, 0

principal:
    btfs PORTB, 0
    goto principal
    btg LATD, 0
otro:
    btfc PORTB, 0
    goto otro
    goto principal
end

```

XC8 PIC ASM:

```

PROCESSOR 18F4550
#include "cabecera.inc"

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

rstVect:
    goto configuro
    ORG 0020H

configuro:
    bsf TRISB, 0, 0
    bcf TRISD, 0, 0

principal:
    btfs PORTB, 0
    goto principal
    btg LATD, 0, 0
otro:
    btfc PORTB, 0
    goto otro
    goto principal

end rstVect

```

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

10

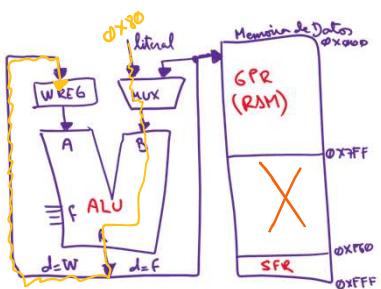
Instrucciones básicas en XC8 ASM

- Instrucciones de movimiento de datos

movlw [literal]

- mover un literal hacia WREG

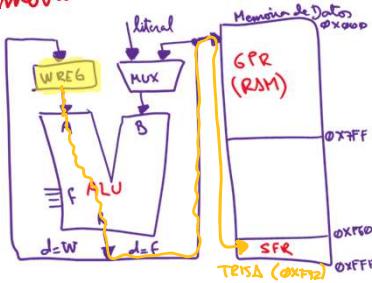
Ej: *movlw 0x80*



movwf [registro]

- mover el contenido de WREG hacia [registro]

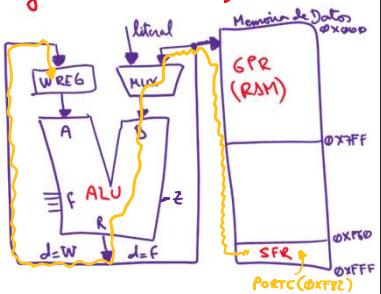
Ej: *movwf TRISA*



movf [registro], d

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

Ej: *movf PORTC, W*

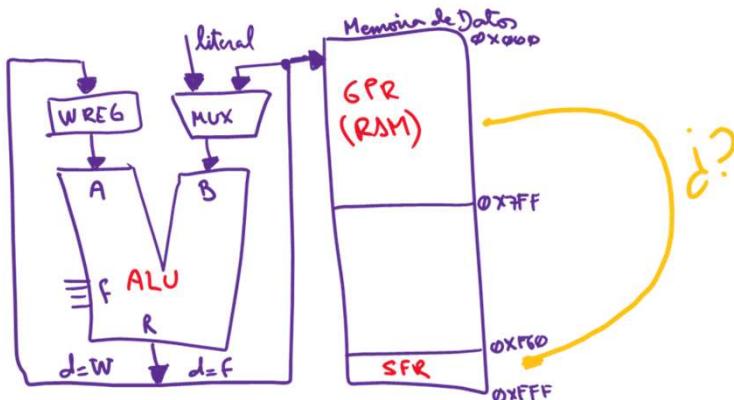


Nota: *movf [registro], F* sirve para act. flag

11

Instrucción movff [registro1], [registro2]

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



movff PORTB, LATD
instrucción compuesta
movf PORTB, W
movwf LATD

12

¿Instrucción movfw?

- ¡Instrucción no documentada en la hoja técnica!
- 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

13

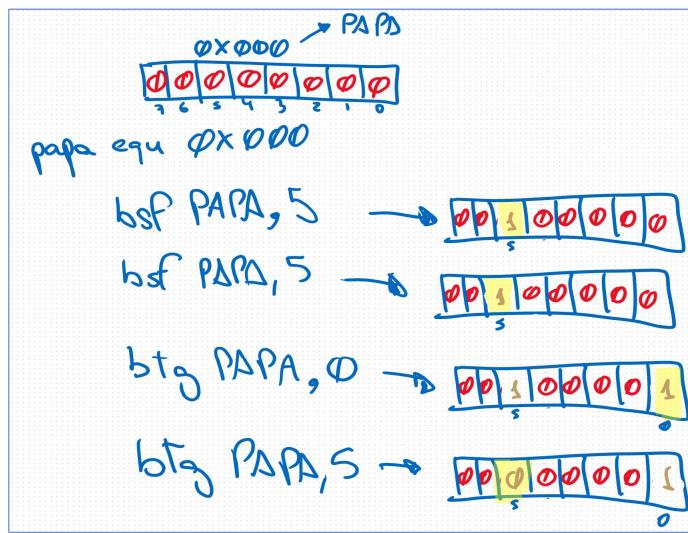
Instrucciones básicas en XC8 ASM

- Instrucciones de manipulación de bits en un registro

<p><u>bsf</u> [registro], #bit <i>posición</i> coloca a "1" el #bit de [registro]</p> <p>Ej: <u>TRISB</u> <i>quiero ponerlo a "1"</i></p> <p><u>bsf TRISB, 3</u></p> <p><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></p> <p><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> <p><u>TRISB</u> </p>
---	--	---

14

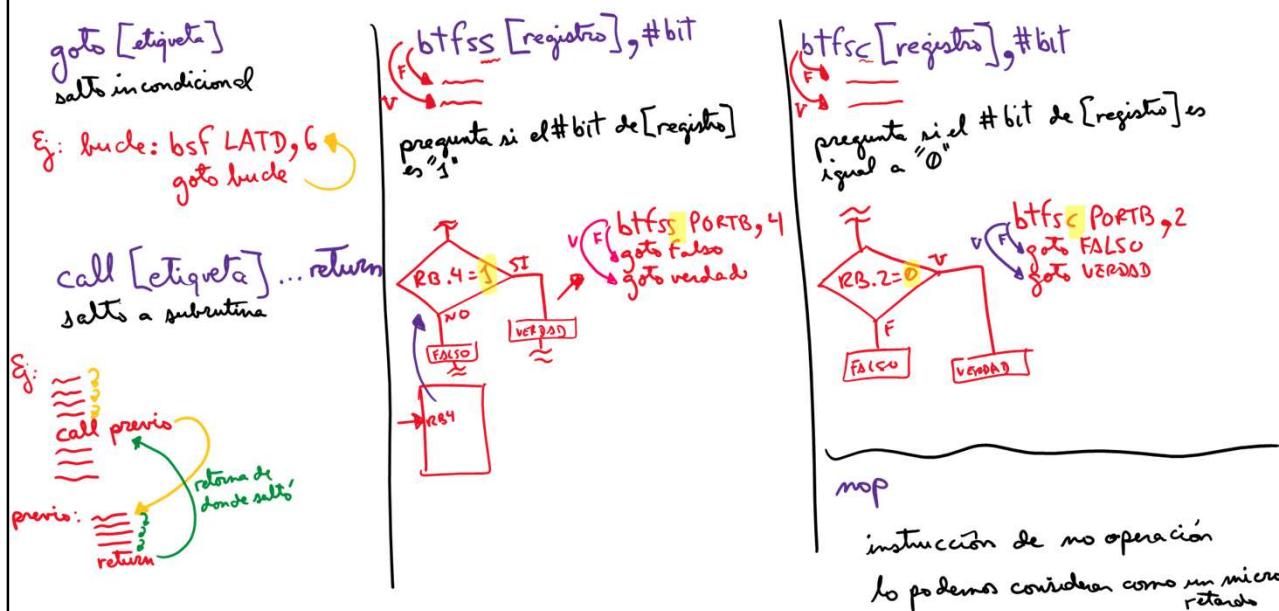
Ejemplo de uso de instrucciones de manipulación de bits en un registro



P bits	
Access RAM	000h - 05Fh
GPR	060h - 0FFh
GPR	100h
GPR	1FFh - 200h
GPR	2FFh - 300h
GPR	3FFh - 400h
GPR ⁽¹⁾	4FFh - 500h
GPR ⁽¹⁾	5FFh - 600h
GPR ⁽¹⁾	6FFh - 700h
GPR ⁽¹⁾	7FFh - 800h
Unused	Read as 00h
Unused	EFFFh - F00h
SFR	F5Fh - F60h
	FF0h - FFFh

15

Instrucciones básicas en XC8 ASM

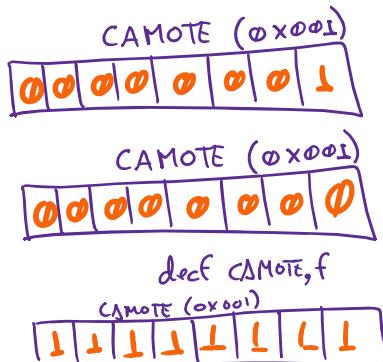


16

Instrucciones básicas en XC8 ASM

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

Ej: decf [registro], d

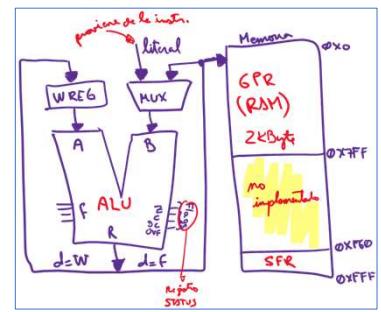


"d" puede ser: f ó w

decf CAMOTE, f

→ Registro STATUS: Z = 1

incf [registro], d



17

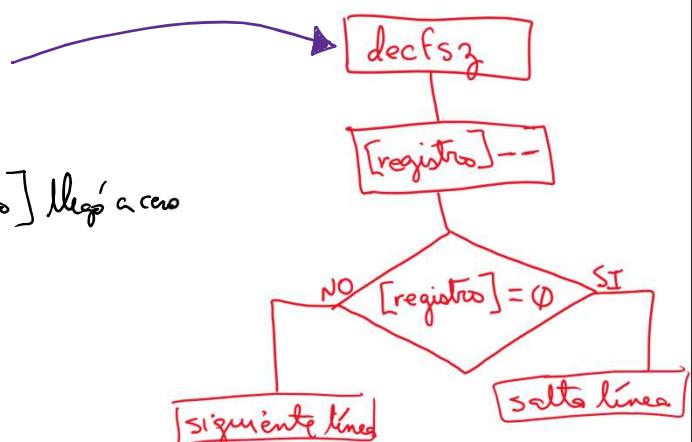
Instrucciones básicas en XC8 ASM

decfsz [registro], d

decrementa y pregunta si [registro] llegó a cero

incfsz [registro], d

incrementa y pregunta si [registro] llegó a cero



18

Instrucciones setf [registro] y clrf [registro]

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

ejemplo:

setf TRISB
⇒

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

ejemplo:

clrf TRISB
⇒

19

Tiempo de ejecución de las instrucciones en XC8 PIC Assembler

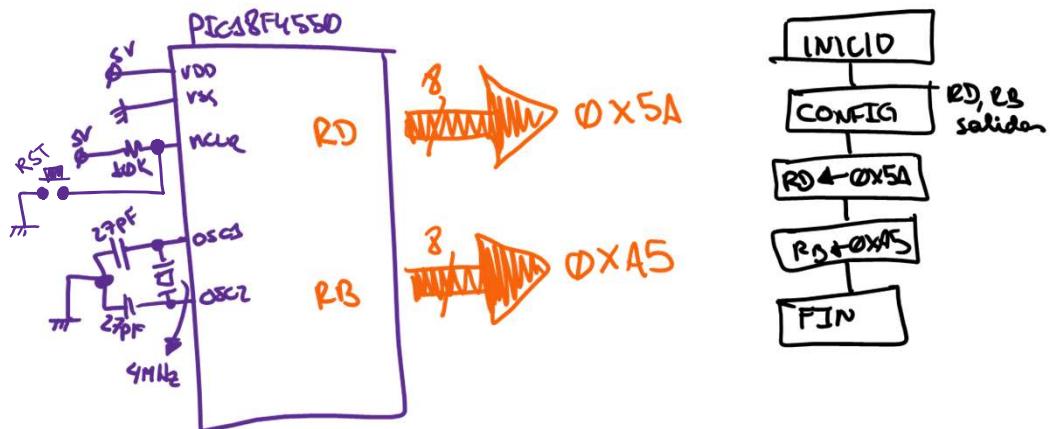
- Se utiliza la siguiente fórmula:

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

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

20

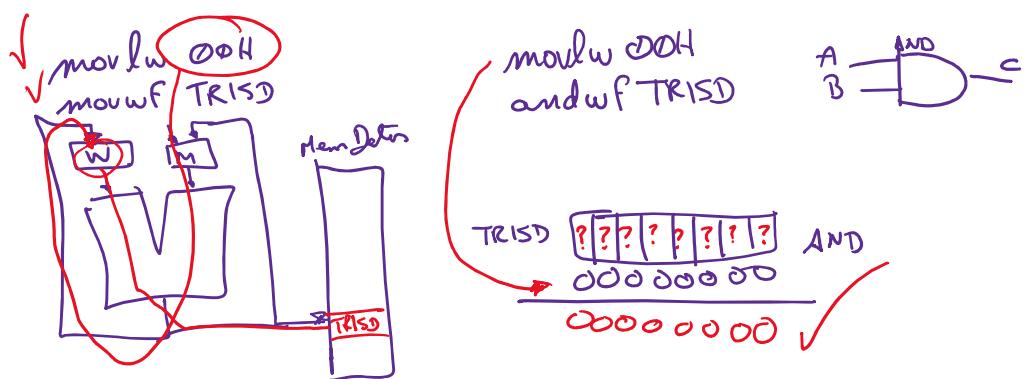
Ejemplo: Escribir 0x5A en RD y 0xA5 en RB



21

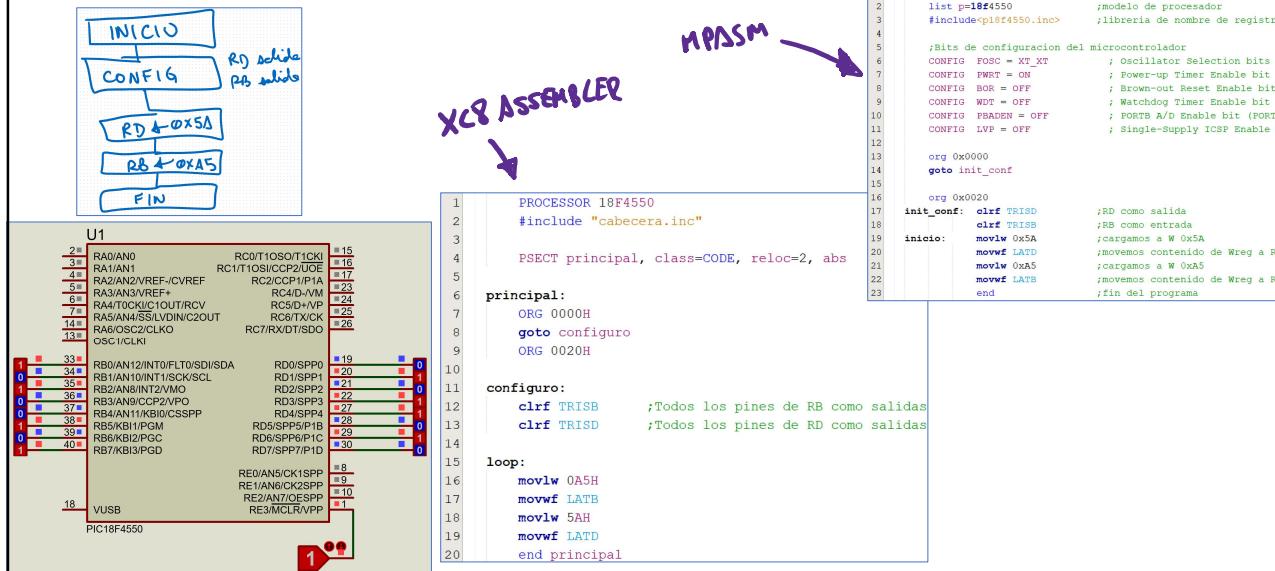
Análisis:

- El objetivo: Pines del RD como salidas



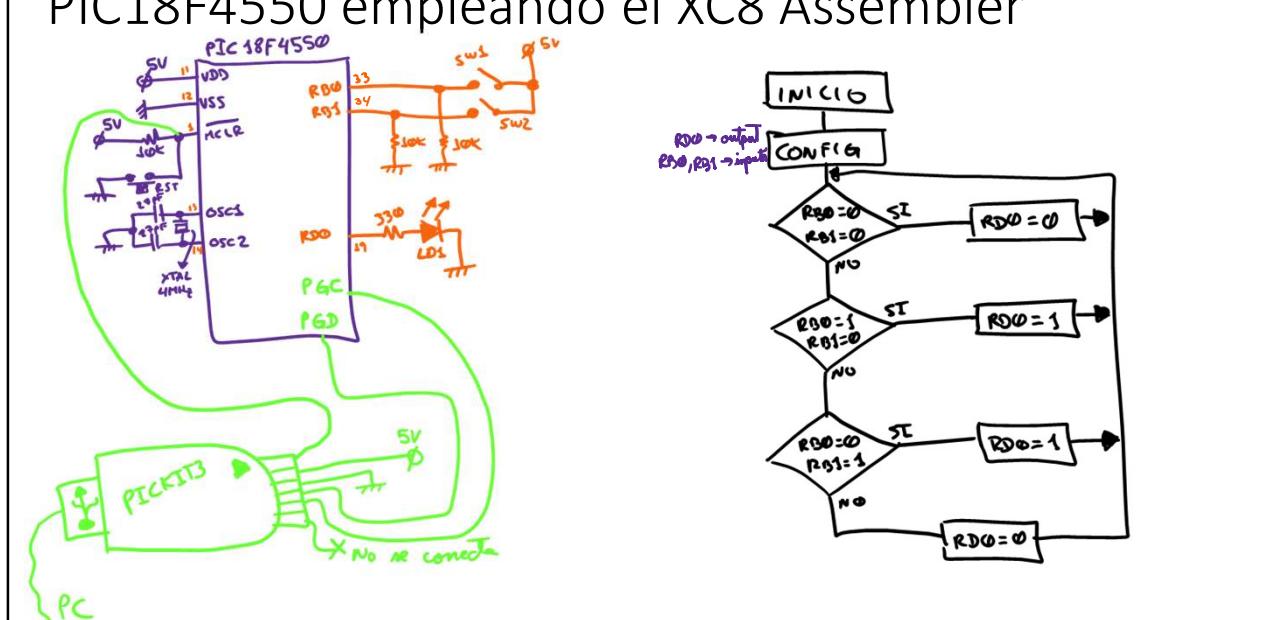
22

Ejemplo: Enviar un dato 0x5A al puerto RD y un dato 0xA5 al puerto RB



23

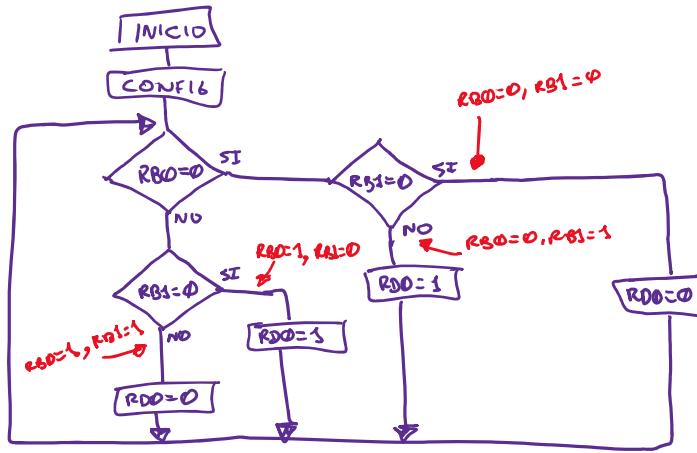
Ejemplo: Implementar una XOR de un bit en el PIC18F4550 empleando el XC8 Assembler



24

Ejemplo: Implementar una XOR de un bit en el PIC18F4550 empleando el XC8 Assembler

- Desarrollando mas a detalle el diagrama de flujo: Teniendo en cuenta que se usará la instrucción "btfs" para las preguntas a cada puerto de entrada de la XOR



25

Ejemplo: Implementar una XOR de un bit en el PIC18F4550 empleando el XC8 Assembler

- Tener en cuenta que en cabecera.inc se encuentra los bits de configuración y la llamada a la librería general "xc.inc"

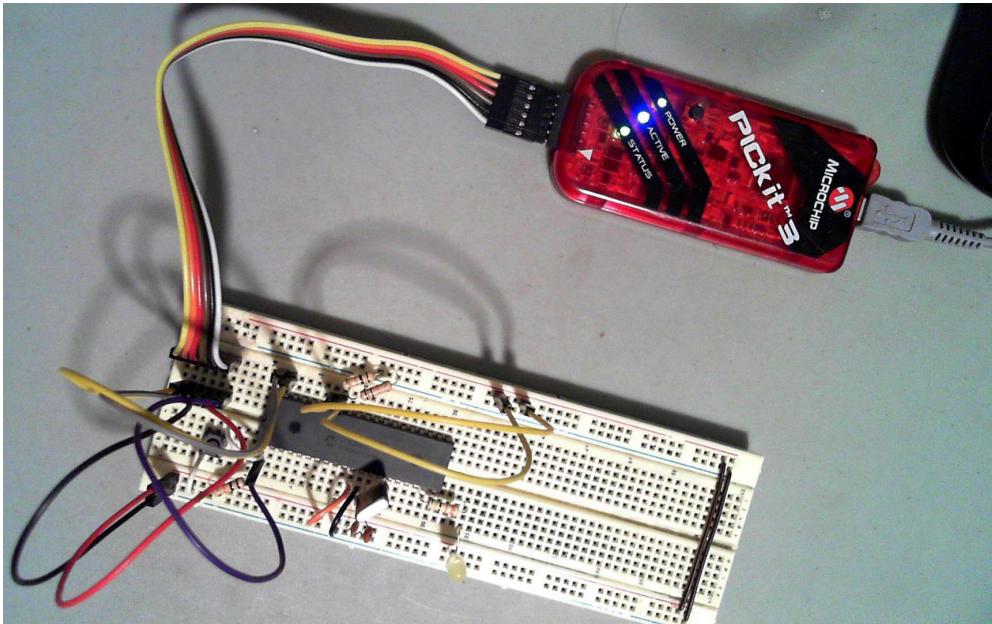
```

1  PROCESSOR 18F4550
2  #include "cabecera.inc"
3
4  PSECT muynuevo,class=CODE,reloc=2, abs
5  ORG 0000H
6  muynuevo:
7      goto configuracion
8  ORG 0020H
9  configuracion:
10     bcf TRISD, 0, 0 ;RD0 como salida
11
12  inicio:   btfs PORTB, 0, 0 ;Pregunto RB0=0
13      goto falsos1 ;Salto a falsos1 cuando es falso
14      btfs PORTB, 1, 0 ;Pregunto RB1=0
15      goto falsos2 ;Salto a falsos2 cuando es falso
16      bcf LATD, 0, 0 ;RB0=0 RB1=0 por tanto RD0=0
17      goto inicio ;Salto a inicio
18  falsos1:  btfs PORTB, 1, 0 ;Pregunto RB1=0
19      goto falsos3 ;Salto a falsos3 cuando es falso
20      bsf LATD, 0, 0 ;RB0=1 RB1=0 por tanto RD0=1
21      goto inicio ;Salto a inicio
22  falsos2:  bsf LATD, 0, 0 ;RB0=0 RB1=1 por tanto RD0=1
23      goto inicio ;Salto a inicio
24  falsos3:  bcf LATD, 0, 0 ;RB0=1 RB1=1 por tanto RD0=0
25      goto inicio ;Salto a inicio
26
27      end muynuevo

```

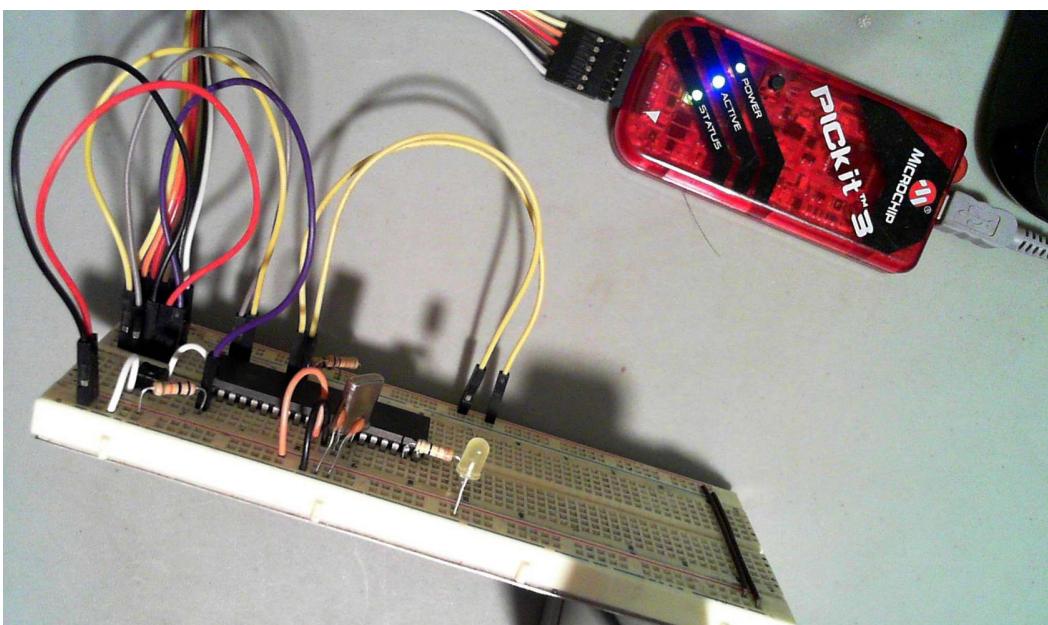
26

Implementación del circuito:



27

Implementación del circuito:



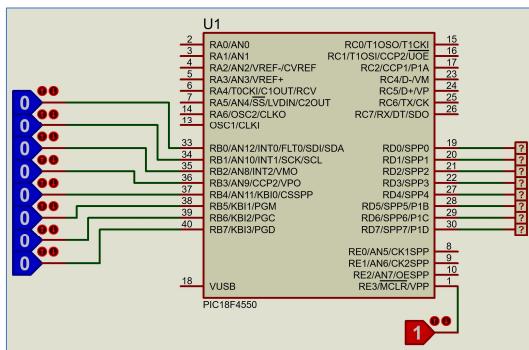
28

Ejemplos adicionales

- Tener en cuenta que los ejemplos siguientes están desarrollados en MPASM por lo que se tendrán que migrar al nuevo XC8 PIC Assembler para que puedan compilarse correctamente.

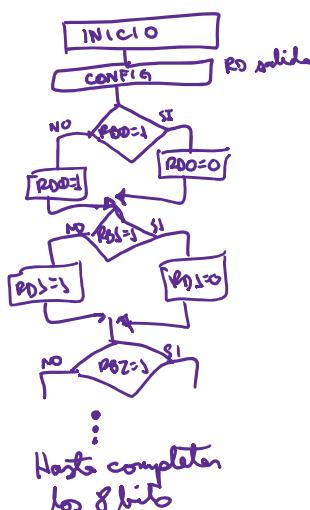
29

Ejemplo: Recibir un dato en RB y replicarlo en complemento a RD



Nota:

- Se ha seguido el ejemplo de un negador de un bit pero replicándolo siete veces más para así obtener un negador de 8 bits solicitado.
- El código resultante es extenso. Por lo que se evaluará una mejor alternativa empleando la instrucción **COMF** que se verá en la siguiente lámina



```

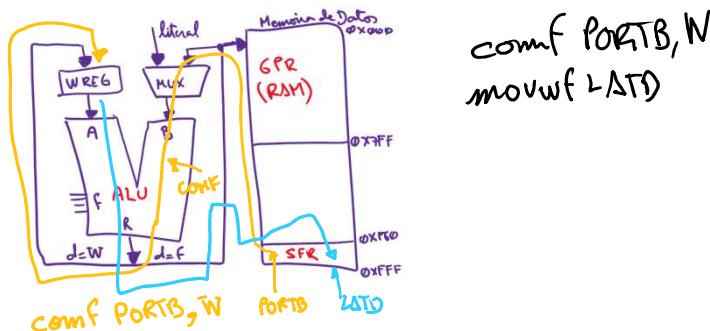
13      org 0x0000
14      goto init_conf
15
16      org 0x0020
17      init_conf: clrf TRISD
18      inicio: btfss PORTE, 0
19      goto falso0
20      bcf LATD, 0
21      goto siguiente0
22      falso0: bsf LATD, 0
23      siguiente0: btfss PORTE, 1
24      goto falso1
25      bcf LATD, 1
26      goto siguiente1
27      falso1: bsf LATD, 1
28      siguiente1: btfss PORTE, 2
29      goto falso2
30      bcf LATD, 2
31      goto siguiente2
32      falso2: bsf LATD, 2
33      siguiente2: btfss PORTE, 3
34      goto falso3
35      bcf LATD, 3
36      goto siguiente3
37      falso3: bsf LATD, 3
38      siguiente3: btfss PORTE, 4
39      goto falso4
40      bcf LATD, 4
41      goto siguiente4
42      falso4: bsf LATD, 4
43      siguiente4: btfss PORTE, 5
44      goto falso5
45      bcf LATD, 5

```

30

Instrucción COMF [registro], d

- Aplica complemento al registro indicado y el resultado lo puede almacenar en Wreg o en el mismo registro



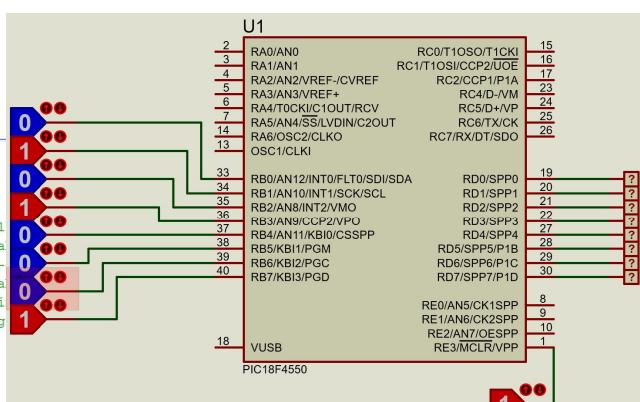
31

Optimizando el ejemplo del negador de 8 bits empleando instrucción comf:

```

2      list p=18f4550          ;modelo de procesador
3      #include<p18f4550.inc>    ;libreria de nombre de registros
4
5      ;Bits de configuracion del microcontrolador
6      CONFIG FOSC = XT_XT        ; Oscillator Selection bits (XT oscil
7      CONFIG PWRTE = ON          ; Power-up Timer Enable bit (PWRT ena
8      CONFIG BOR = OFF           ; Brown-out Reset Enable bits (Brown-
9      CONFIG WDT = OFF           ; Watchdog Timer Enable bit (WDT disa
10     CONFIG PBADEN = OFF        ; PORTB A/D Enable bit (PORTB4:0> pi
11     CONFIG LVPEE = OFF         ; Single-Supply ICSP Enable bit (Sing
12
13     org 0x0000
14     goto init_conf
15
16     org 0x0020
17     init_conf: cirf TRISD      ;RD como salida
18     inicio:      comf PORTB, W   ;complemento de PORTB y lo almacena en Wreg
19                           ;muestra el contenido de Wreg a RD
20     goto inicio
21     end                  ;fin del programa

```



Del código extenso visto anteriormente ahora se reduce a pocas líneas gracias a comf

32

Ejemplo: Titilar un LED conectado en RD2

Hardware:

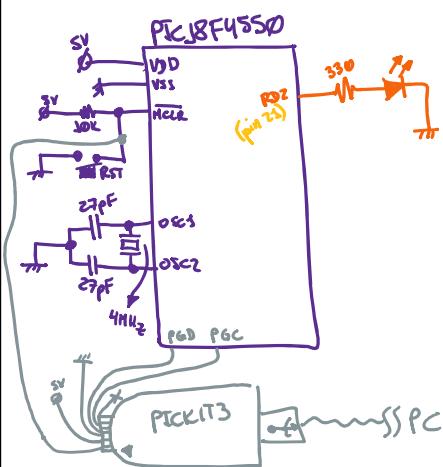
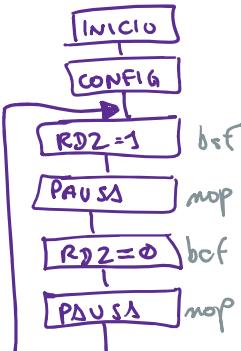


Diagrama de Flujos:



Note:
 $T_{ejec\ instr.} = \left(\frac{F_{osc}}{4}\right)^{-1}$
 $F_{osc} = 4MHz$
 $\Rightarrow T_{ejec\ instr.} = 1\mu s$

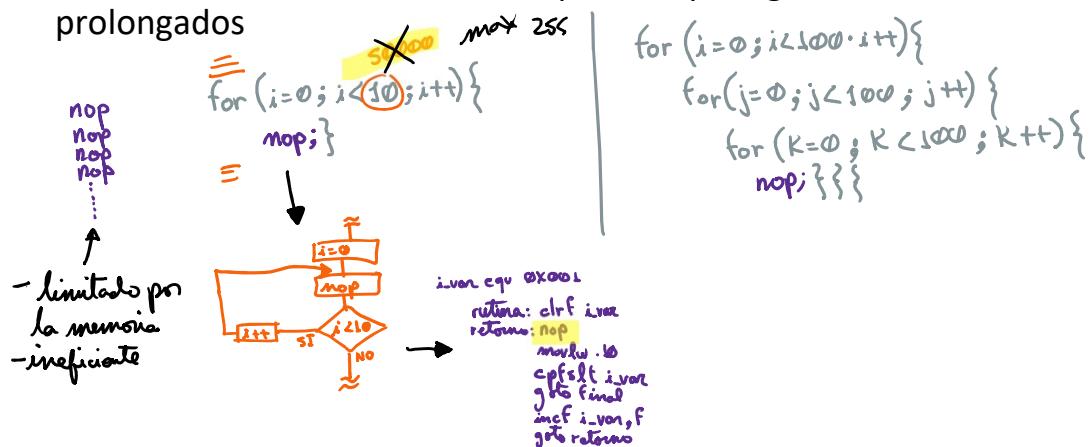
Tenemos que alargar el tiempo
¿Puedo colocar 50000 'nop'?

NO!

33

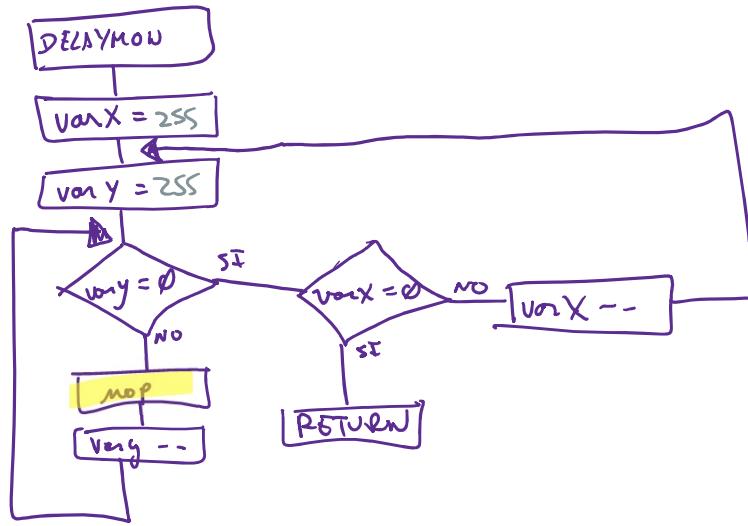
Ejemplo: Titilar un LED conectado en RD2

- Debemos de crear un bucle de repetición para generar retardos más prolongados



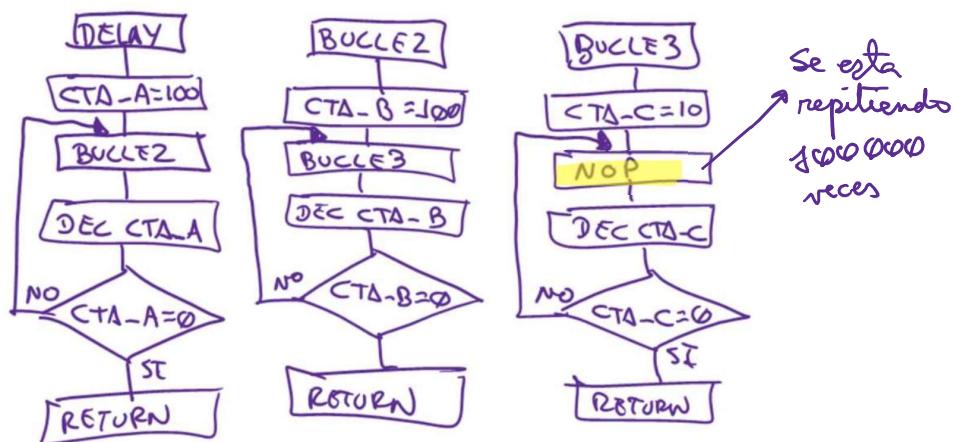
34

Propuesta de algoritmo con dos anillos de repetición



35

Propuesta de algoritmo con tres anillos de repetición



36

Código del titilador de LED en MPASM:

```

1      list p=18f4550
2      #include <p18f4550.inc>      ;libreria de nombre de la
3
4      CONFIG  FOSC = XT XT
5      CONFIG  CPUDIV = OSC1_PLL2
6      CONFIG  PWRT = ON           ; Power-up Timer Enabled
7      CONFIG  BOR = OFF          ; Brown-out Reset Enabled
8      CONFIG  WDT = OFF          ; Watchdog Timer Enabled
9      CONFIG  PBADEN = OFF       ; PORTB A/D Enable bit
10     CONFIG  LVP = OFF          ; Single-Supply ICSP Enabled
11
12     cblock 0x000
13     var_i
14     var_j
15     var_k
16     endc
17
18     org 0x0000                ;Vector de reset
19     goto configuro
20
21     org 0x0020                ;Zona de programa de usuario
22 configuro:
23     bcf TRISD, 2              ;RD0 como salida
24
25 inicio:
26     bsf LATD, 2               ;prendo el led
27     call delaymon
28     bcf LATD, 2               ;apago el led
29     call delaymon
30     goto inicio

```

```

32     delaymon:
33         movlw .50
34         movwf var_i
35     otrol:
36         call bucle1             ;Salto a subrutina
37         decfsz var_i,f
38         goto otrol
39         return
40
41     bucle1:
42         movlw .55
43         movwf var_j
44     otro2:
45         nop
46         nop
47         call bucle2             ;Salto a subrutina
48         decfsz var_j,f
49         goto otro2
50         return
51
52     bucle2:
53         movlw .20
54         movwf var_k
55     otro3:
56         nop
57         decfsz var_k,f
58         goto otro3
59         return
60         end

```

37

Prototipo físico del circuito:



38

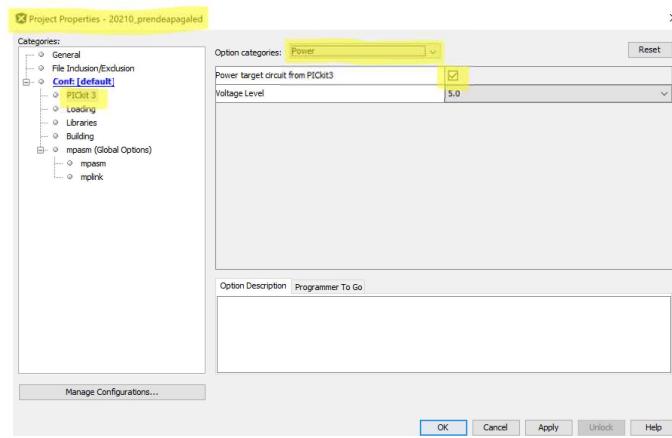
Recomendaciones al momento de implementar el circuito en físico:

- Verificar continuidad en los cables jumper utilizados en el circuito
- Verificar que la resistencia en pin MCLR hacia 5V sea de 10K
- Verificar que la PC haya detectado correctamente el PICKIT3
- Revisar siempre los mensajes en la ventana de “output” por posibles fallos en el evento de compilación y evento de programación.
- Tener a la mano un multímetro para verificar voltajes y continuidad en el prototipo.

39

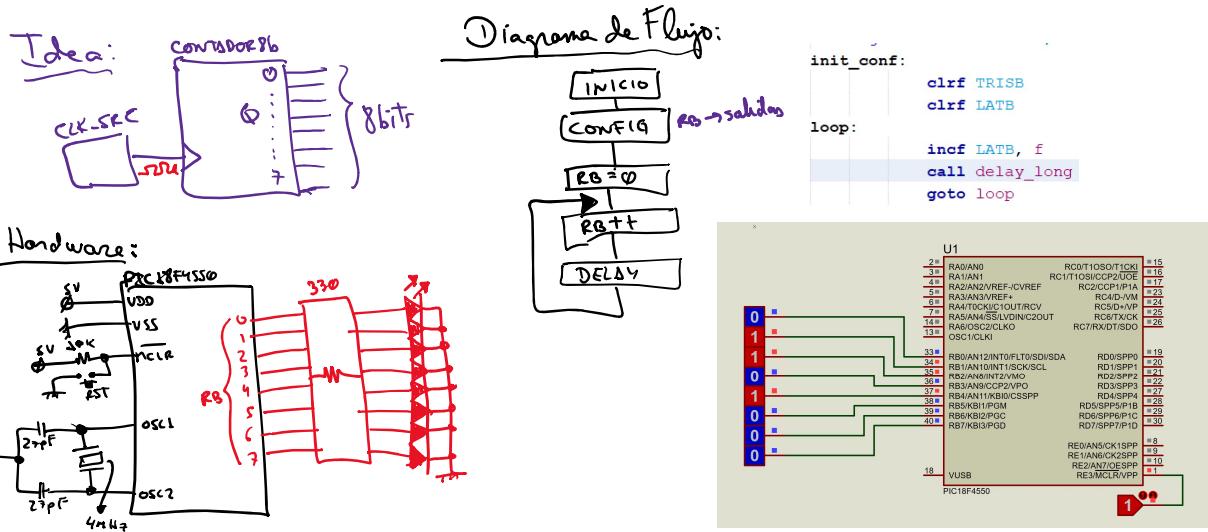
Recomendaciones al momento de implementar el circuito en físico:

- Si se está empleando el PICKIT3 como programador y deseas que éste administre el voltaje de alimentación al circuito de prueba:



40

Modificar el titileo de LED para que se visualice un contador autoincremental de 8 bits



41

Ejercicios adicionales:

Tener en cuenta que se debe de seguir el procedimiento visto en clase (idea, desarrollo del circuito, diagrama de flujo, código en MPASM, simulación, prototipo físico)

- 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.
- Desarrollar una “vela electrónica” con entrada de activación, dicha entrada tendrá como sensor de luz a un L.D.R.

42

Fin de la sesión!