

# Microcontroladores

## Laboratorio Semana 2

Semestre: 2023-1

Profesor: Kalun José Lau Gan

### 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 8 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 PIC18F45K50. 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

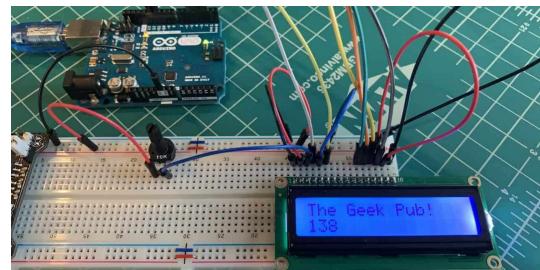
## Preguntas previas:

- 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.



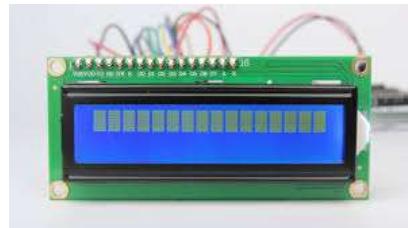
## Preguntas previas:

- ¿Cuáles pueden ser las consecuencias “negativas” en cambiar de frecuencia de trabajo de 4MHz a 48MHz?
  - 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?



## 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.



## Preguntas previas:

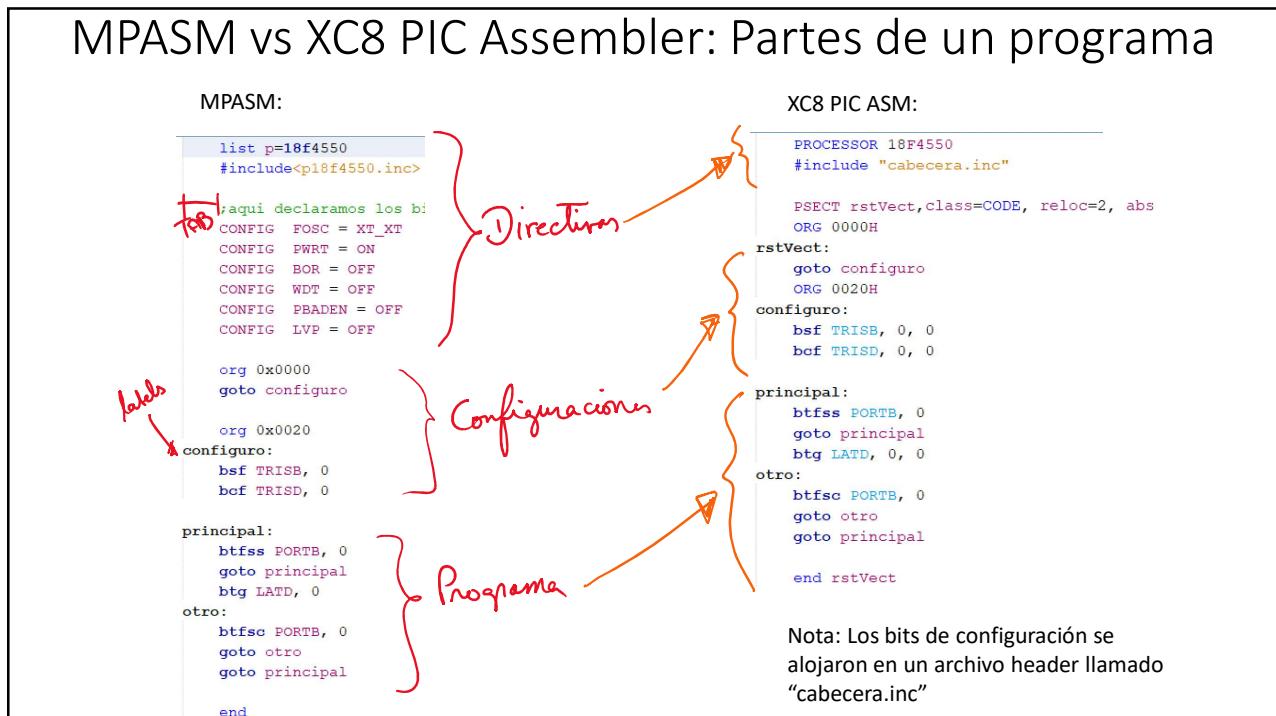
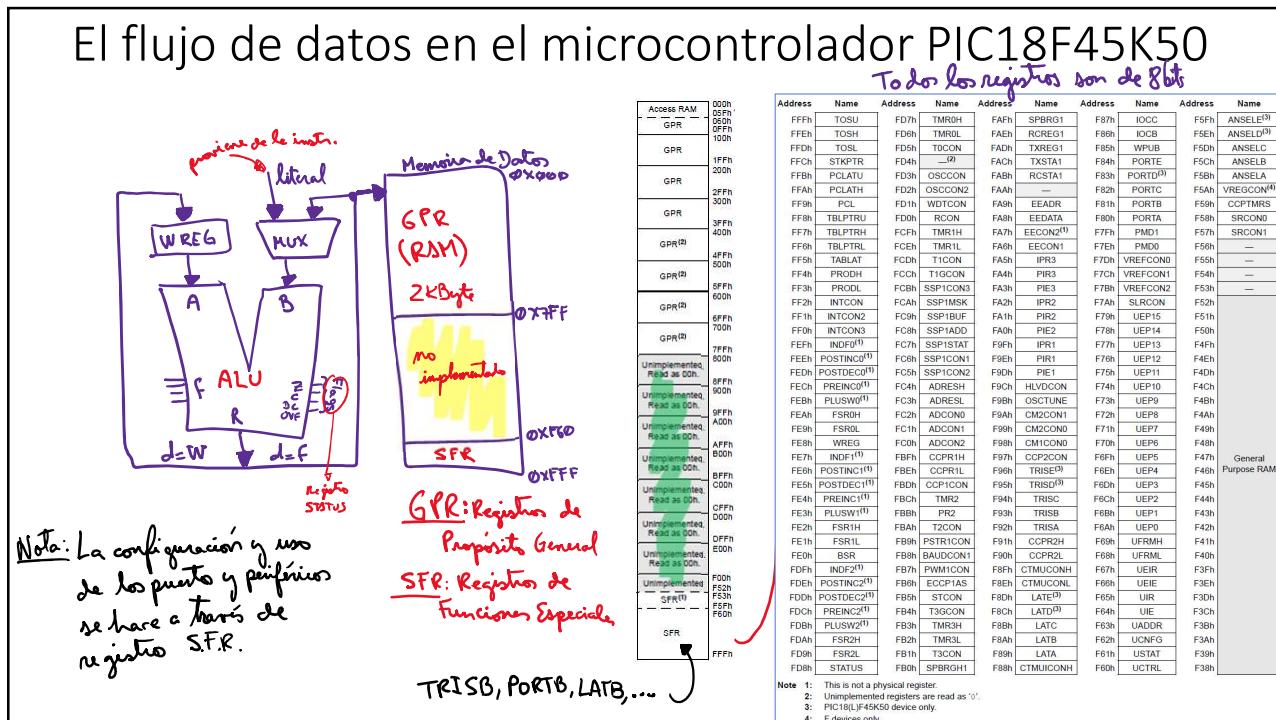
- ¿El bit PBADEN para qué sirve?
  - Es el habilitador para que los pines 4 al 0 del puerto B funcionen como E/S digitales (OFF) o como entradas analógicas para el conversor A/D (ON).
- Con el programador viene un CD. ¿Es necesario instalarlo?
  - No, el MPLABX se encarga de todo.

## Agenda

- El flujo de datos en el microcontrolador PIC18F45K50
- Instrucciones básicas en XC8 PIC Assembler
- Ejemplos básicos con el microcontrolador PIC18F45K50 en lenguaje Assembler
- Nuestra primera implementación física de un circuito basado en el microcontrolador PIC18F45K50

## 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



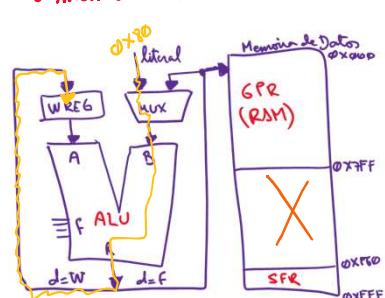
## Instrucciones básicas en XC8 ASM

- Instrucciones de movimiento de datos

**movlw [literal]**

- mover un literal hacia WREG

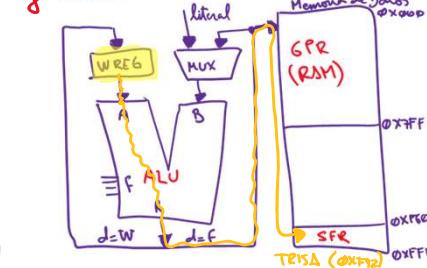
Ej: movlw 0x80



**movwf [registro]**

- mover el contenido de WREG hacia [registro]

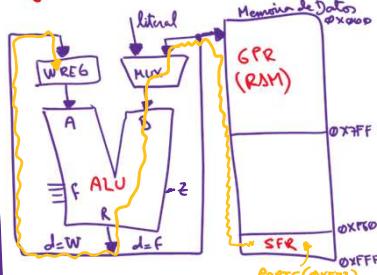
Ej: movwf TRISA



**movf [registro], d**

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

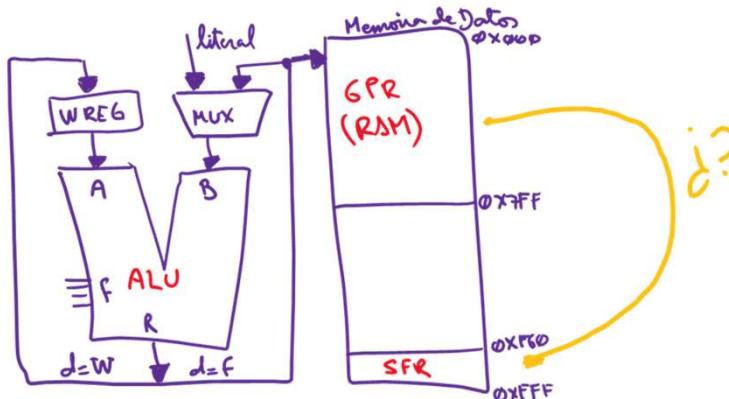
Ej: movf PORTC, W



Note: movf [registro], F sirve para not. flag

## 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

## ¿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      *similar*       $\rightarrow$  movf TRISA, w

## Instrucciones básicas en XC8 ASM

- Instrucciones de manipulación de bits en un registro

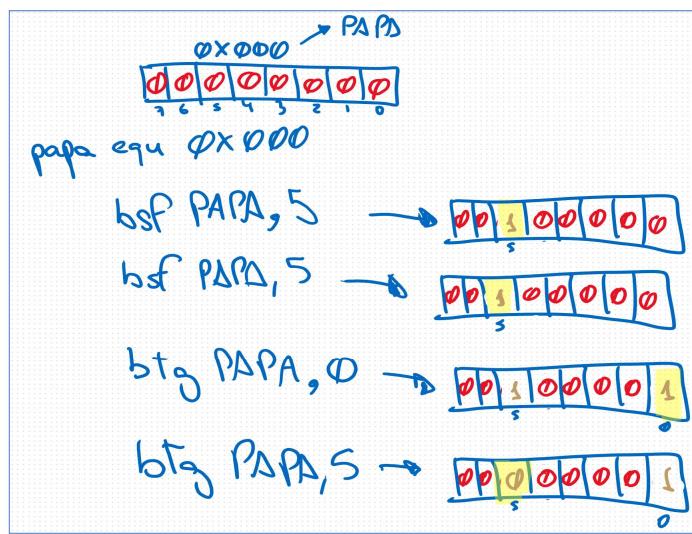
bsf [registro], #bit *posición*  
coloca a "1" el #bit de [registro]  
Ej:  
  
 bsf TRISB, 3

bcf [registro], #bit  
coloca a "0" el #bit de [registro]  
Ej:  
  
 bcf TRISB, 7

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

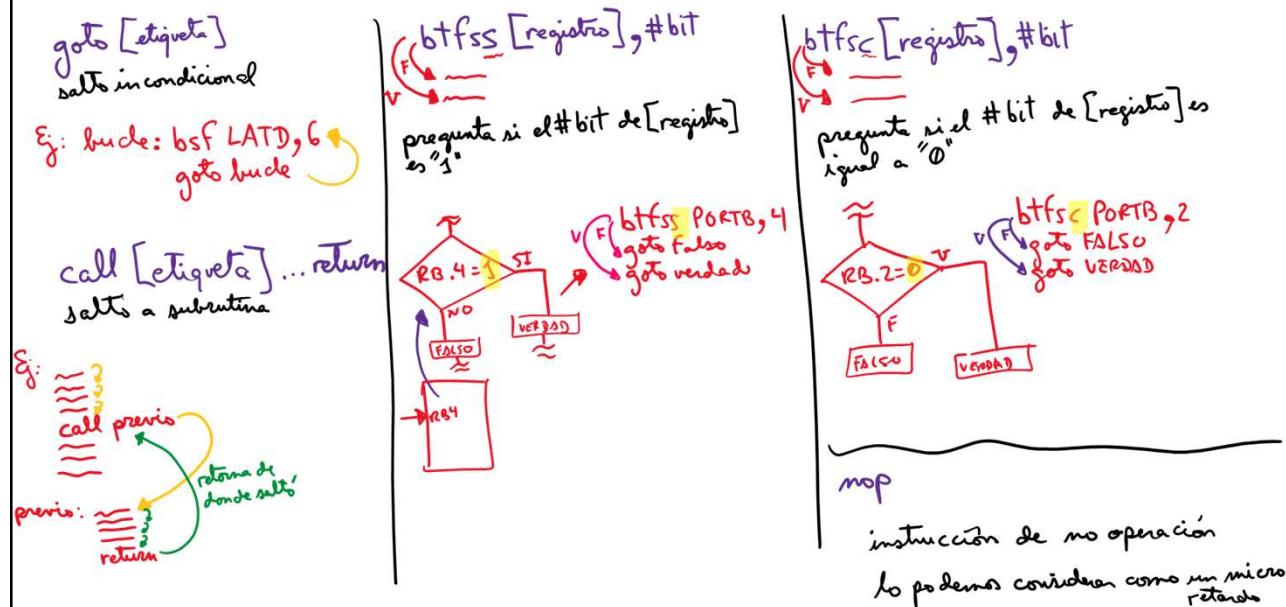
Ej: btg TRISB, 6

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



Registers 000h - FFFh	
00000000h	000h
GPR	000h
GPR	001h
GPR	002h
GPR	003h
GPR(2)	004h
GPR(2)	005h
GPR(2)	006h
GPR(2)	007h
Unimplemented.	Read as 00h.
SFR	FFFh

## Instrucciones básicas en XC8 ASM



## Instrucciones básicas en XC8 ASM

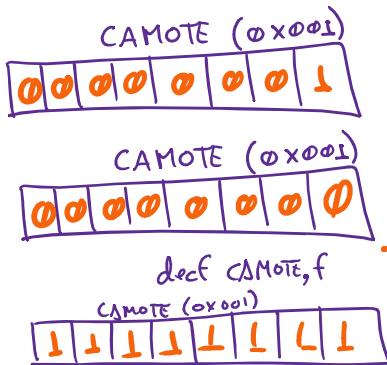
- Instrucción decf / incf

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

Ej: decf [registro], d

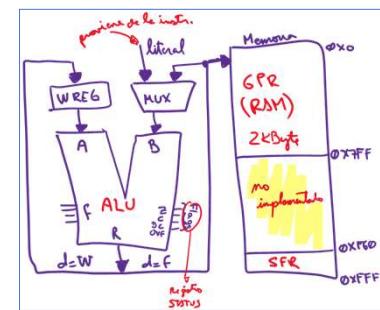
incf [registro], d

"d" puede ser: f ó w



decf CAMOTE, f

Registro STATUS: Z=1



## Instrucciones básicas en XC8 ASM

decfsz [registro], d

decrementa y pregunta si [registro] llegó a cero

decfsz

[registro] --

NO [registro] = 0

SI

Siguiente línea

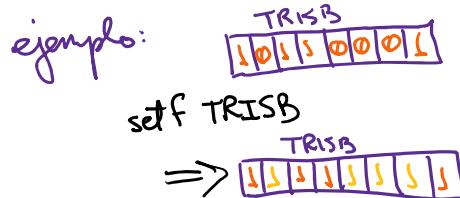
Salta línea

incfsz [registro], d

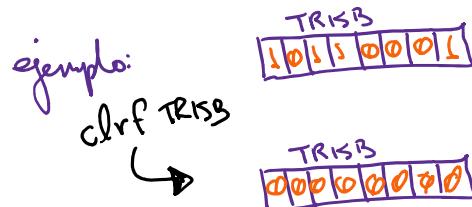
incrementa y pregunta si [registro] llegó a cero

## Instrucciones setf [registro] y clrf [registro]

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



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



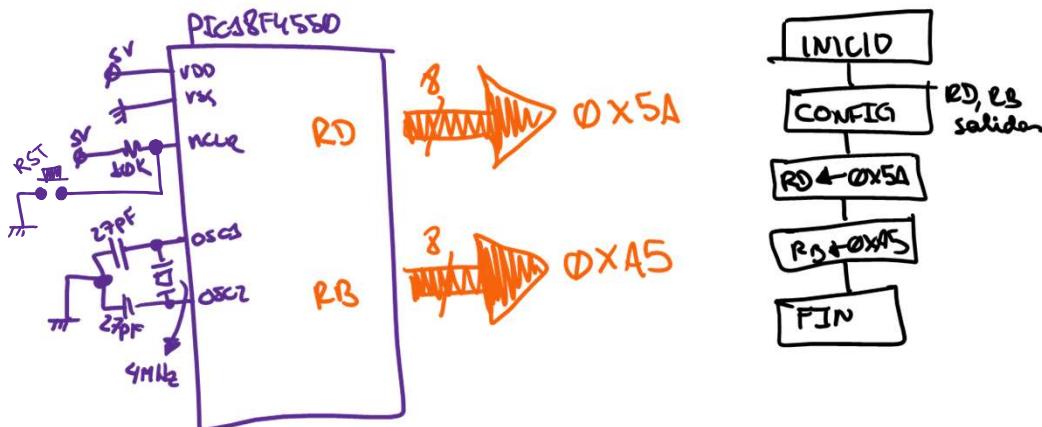
## Tiempo de ejecución de las instrucciones en XC8 PIC Assembler

- Se utiliza la siguiente fórmula:

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

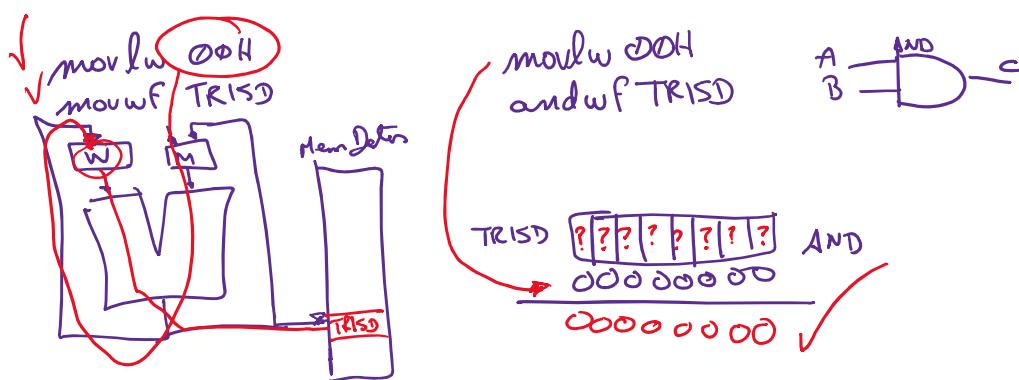
- Hay instrucciones simples, dobles y especiales (revisar 26.0 de la datasheet)
- Recordando la relación periodo vs frecuencia:  $f = \frac{1}{T}$

Ejemplo: Escribir 0x5A en RD y 0xA5 en RB usando el PIC18F4550

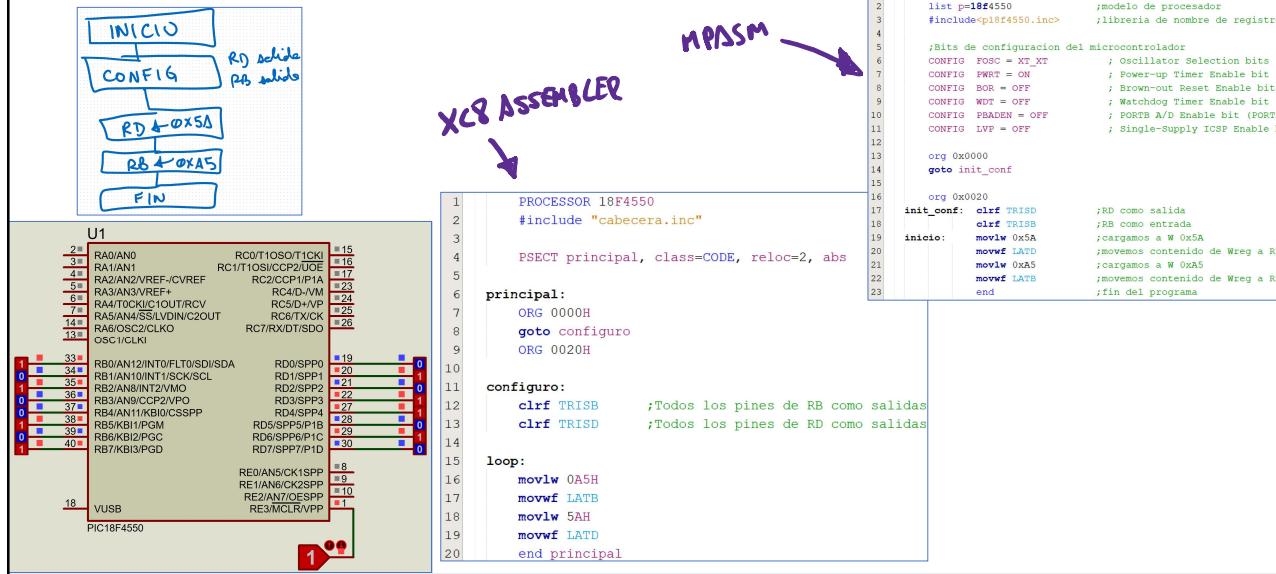


## Análisis:

- El objetivo: Pines del RD como salidas



Ejemplo: Enviar un dato 0x5A al puerto RD y un dato 0xA5 al puerto RB usando el PIC18F4550



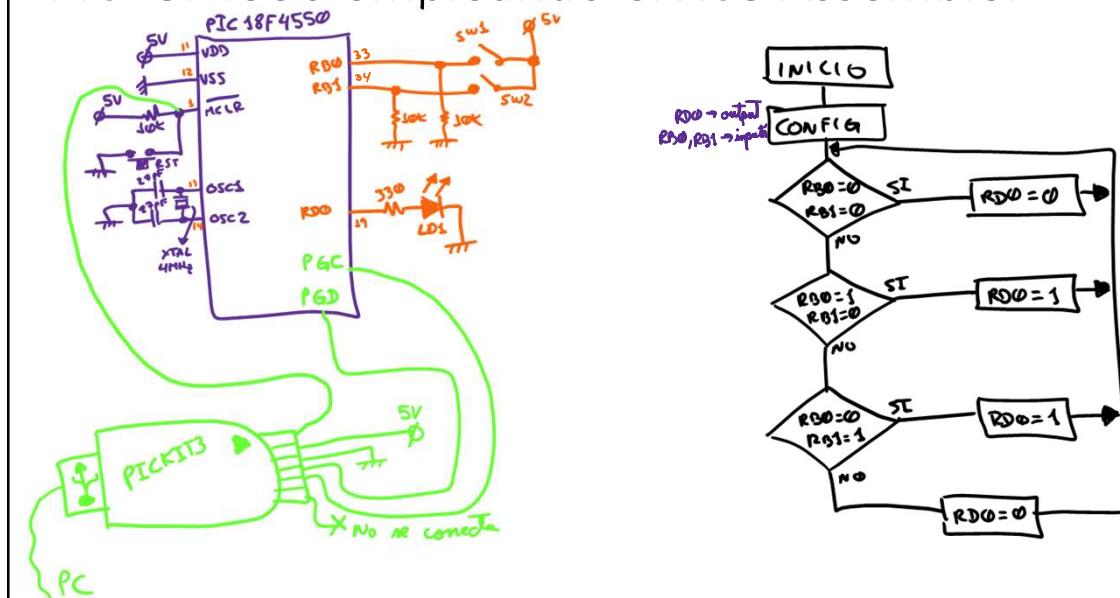
Ejemplo: Escribir 0x5A en RD y 0xA5 en RB usando el PIC18F45K50



Ejemplo: Enviar un dato 0x5A al puerto RD y un dato 0xA5 al puerto RB usando el PIC18F45K50

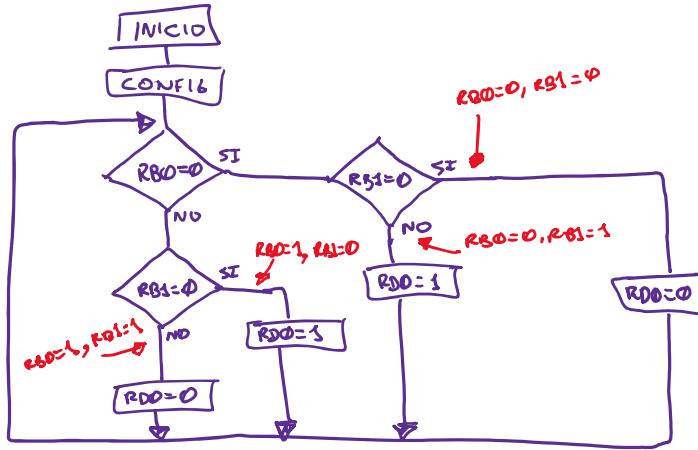
XC8 ASSEMBLER

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



## 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



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

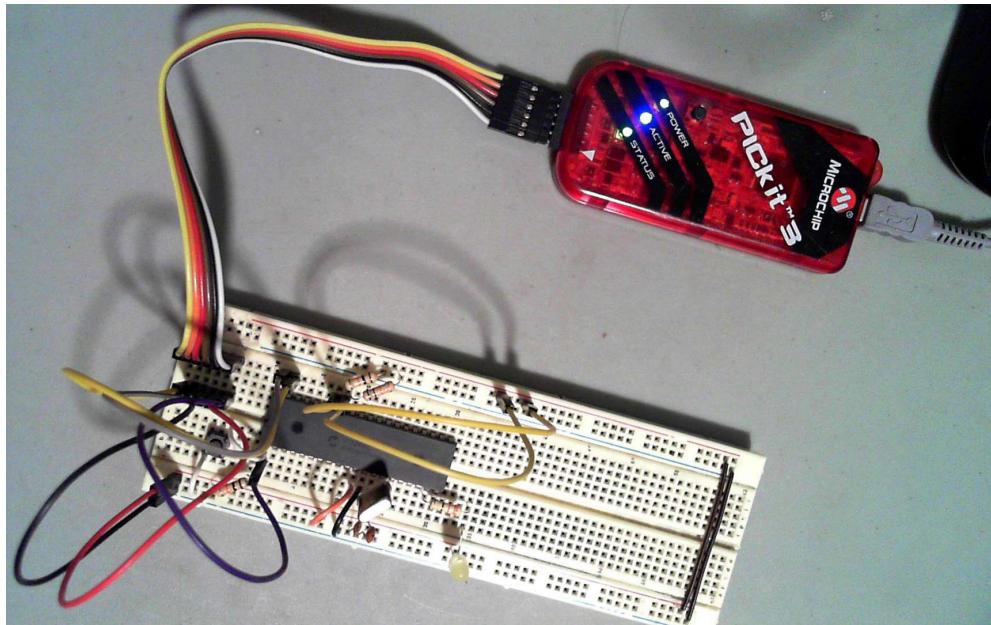
- Tener en cuenta que en cabecera.inc se encuentra los bits de configuración y la llamada a la librería general “xc.inc”

```

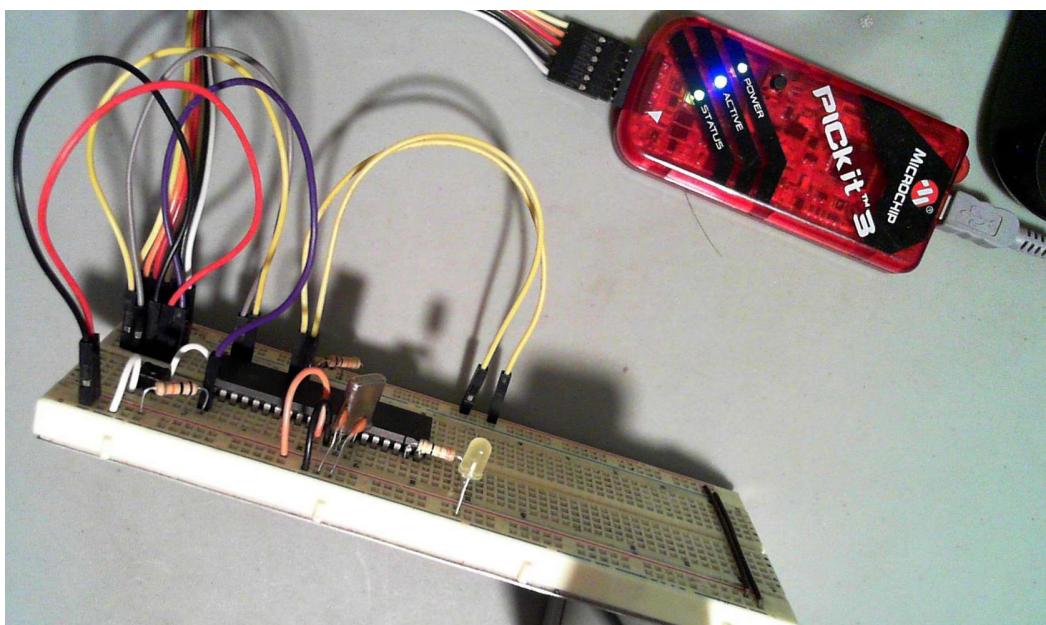
1      PROCESSOR 18F4550
2      #include "cabecera.inc"
3
4      PSECT muynuevo, class=CODE, reloc=2, abs
5      ORG 0000H
6      muynuevo:
7          goto configuro
8      ORG 0020H
9      configuro:
10         bcf TRISD, 0, 0 ;RD0 como salida
11
12     inicio:   btfs PORTB, 0, 0 ;Pregunto RB0=0
13         goto falsol ;Salto a falsol cuando es falso
14         btfs PORTB, 1, 0 ;Pregunto RB1=0
15         goto falso2 ;Salto a falso2 cuando es falso
16         bcf LATD, 0, 0 ;RB0=0 RB1=0 por tanto RD0=0
17         goto inicio ;Salto a inicio
18     falsol:   btfs PORTB, 1, 0 ;Pregunto RB1=0
19         goto falso3 ;Salto a falso3 cuando es falso
20         bsf LATD, 0, 0 ;RB0=1 RB1=0 por tanto RD0=1
21         goto inicio ;Salto a inicio
22     falso2:   bsf LATD, 0, 0 ;RB0=0 RB1=1 por tanto RD0=1
23         goto inicio ;Salto a inicio
24     falso3:   bcf LATD, 0, 0 ;RB0=1 RB1=1 por tanto RD0=0
25         goto inicio ;Salto a inicio
26
27 end muynuevo

```

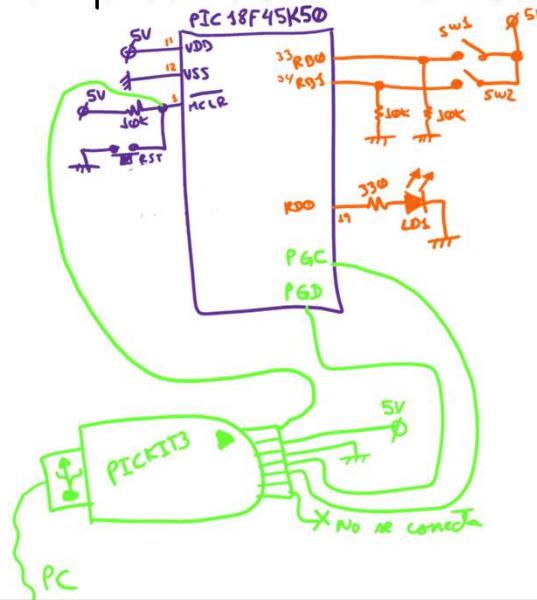
## Implementación del circuito:



## Implementación del circuito:



Ejemplo: Implementar una XOR de un bit en el PIC18F45K50 empleando el XC8 Assembler



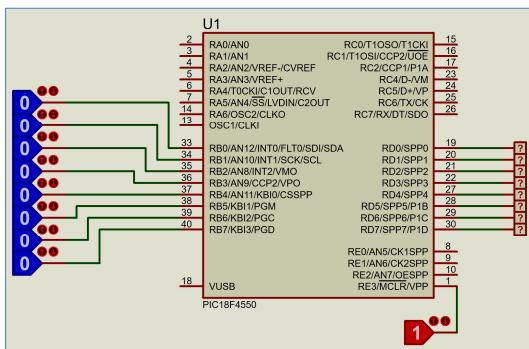
Ejemplo: Implementar una XOR de un bit en el PIC18F45K50 empleando el XC8 Assembler

- Código en XC8 PIC Assembler

## Ejemplos adicionales

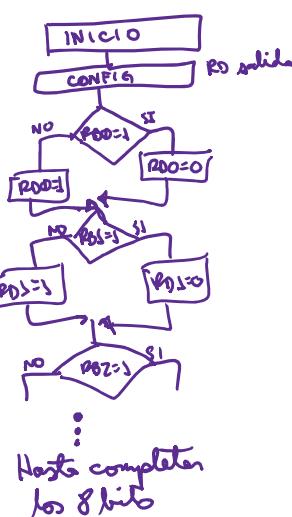
- Tener en cuenta que los ejemplos siguientes:
  - El microcontrolador es el PIC18F4550 por lo que se deberá de modificar los bits de configuración, BSR y demás parámetros de acorde a lo explicado en clase acerca del PIC18F45K50
  - Están desarrollados en MPASM por lo que se tendrán que migrar al nuevo XC8 PIC Assembler para que puedan compilarse correctamente.

## 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



```

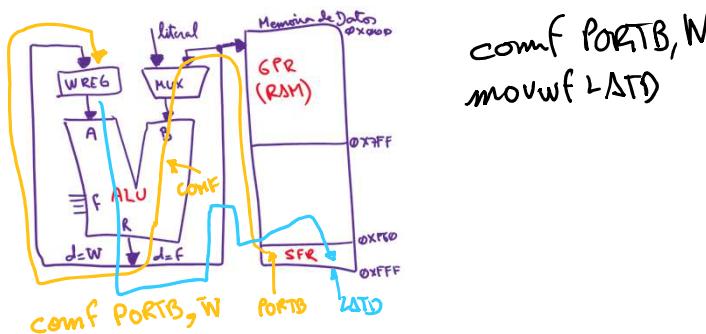
13          org 0x0000
14          goto init_conf
15
16
17          org 0x0020
18          init_conf: clrf TRISD
19          inicio: btfss PORTB, 0
20          goto falso0
21          bcf LATD, 0
22          goto siguiente0
23          falso0: bsf LATD, 0
24          siguiente0: btfss PORTB, 1
25   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

```

;

## Instrucción COMF [registro], d

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

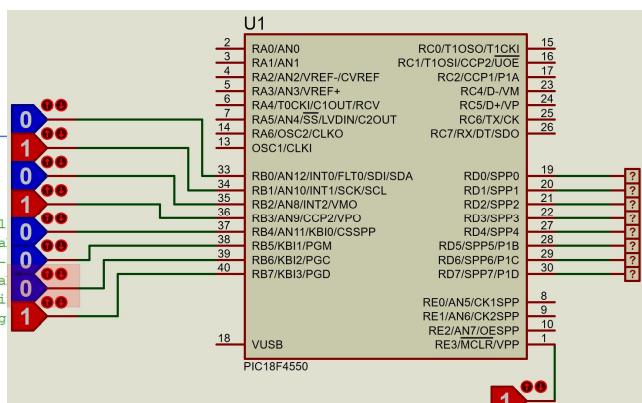


Optimizando el ejemplo del negador de 8 bits empleando instrucción comf:

```

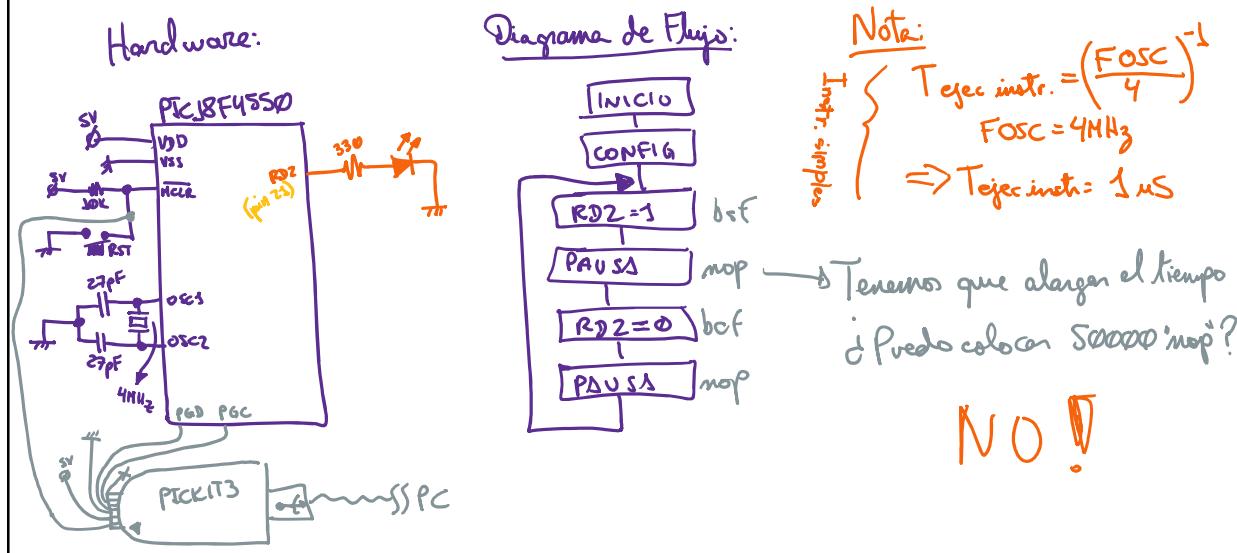
2  list p=18f4550          ;modelo de procesador
3  #include<cp18f4550.inc> ;librería de nombre de registros
4
5  ;bits de configuración del microcontrolador
6  CONFIG FOSC = XT_XT      ; Oscillator Selection bits (XT oscil
7  CONFIG PWRT = ON         ; Power-up Timer Enable bit (PWRT ena
8  CONFIG BOR = 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 (PORTB<4:0> pi
11 CONFIG LVP = OFF         ; Single-Supply ICSP Enable bit (Sing
12
13 org 0x0000
14 goto init_conf
15
16 org 0x0020
17 init_conf: cirlf TRISD   ;RD como salida
18 inicio:    comf PORTB, W ;complemento de PORTB y lo almacena en Wreg
19           movwf LATD       ;mueve el contenido de Wreg a RD
20           goto inicio
21           end                ;fin del programa

```



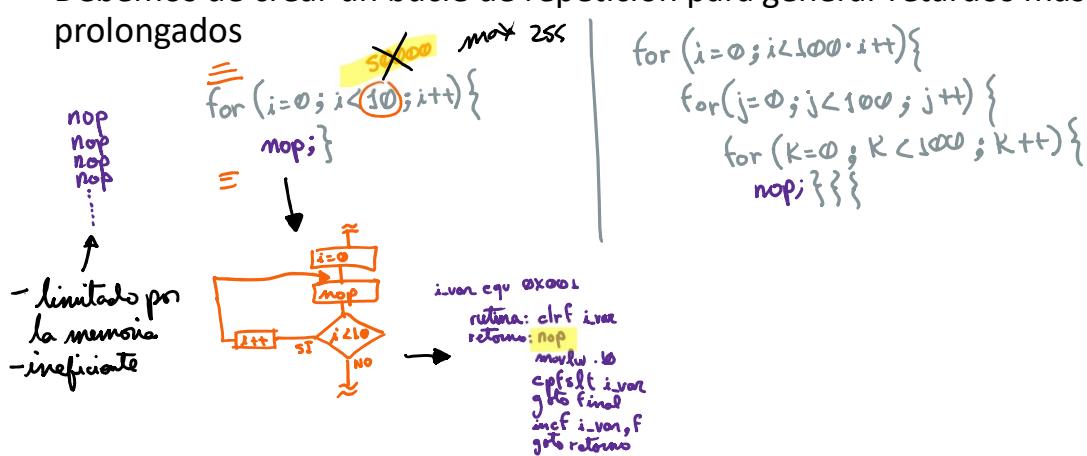
Del código extenso visto anteriormente ahora se reduce a pocas líneas gracias a comf

## Ejemplo: Titilar un LED conectado en RD2

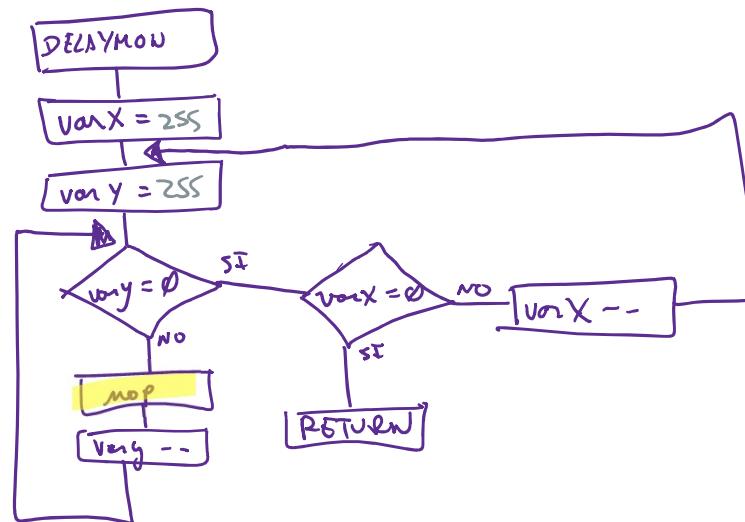


## Ejemplo: Titilar un LED conectado en RD2

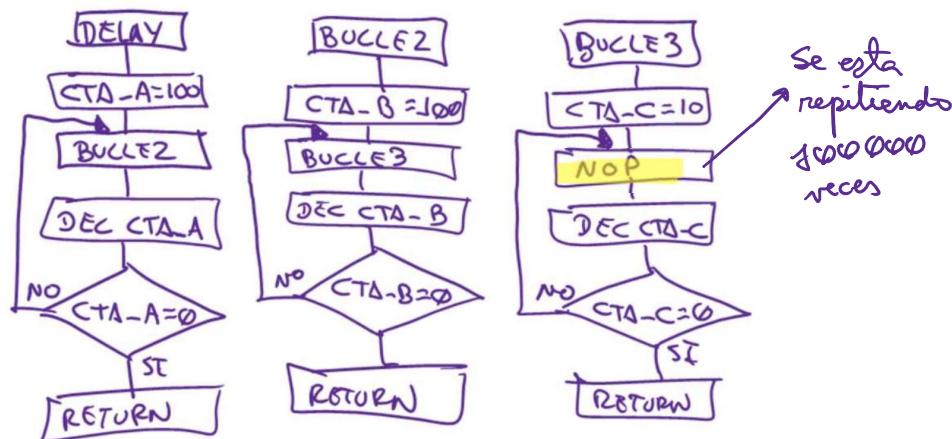
- Debemos de crear un bucle de repetición para generar retardos mas prolongados



## Propuesta de algoritmo con dos anillos de repetición



## Propuesta de algoritmo con tres anillos de repetición



## Código del titilador de LED en MPASM:

```

1      list p=18f4550
2      #include <p18f4550.inc>      ;libreria de nombre de los
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 configuracion
20
21     org 0x0020                ;Zona de programa de usuario
22     configuracion:
23         bcf TRISD, 2           ;RDO 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
31
32     delaymon:
33         movlw .50
34         movwf var_i
35         otro1:
36             call bucle1          ;Salto a subrutina
37             decfsz var_i,f
38             goto otro1
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

```

## Prototipo físico del circuito:

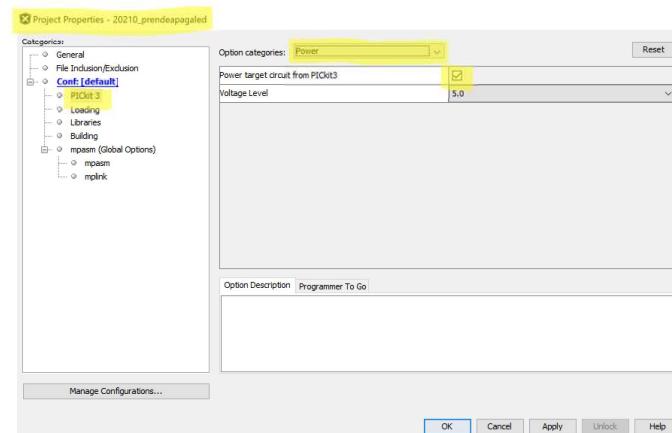


## Recomendaciones al momento de implementar el circuito en físico:

- 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.

## 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:



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

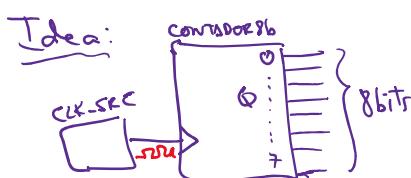
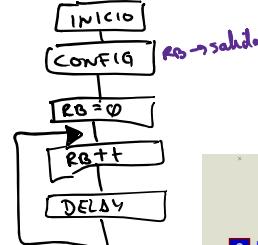


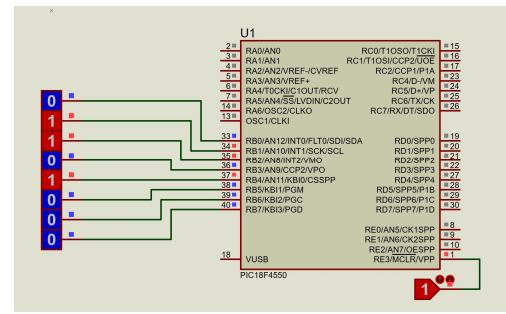
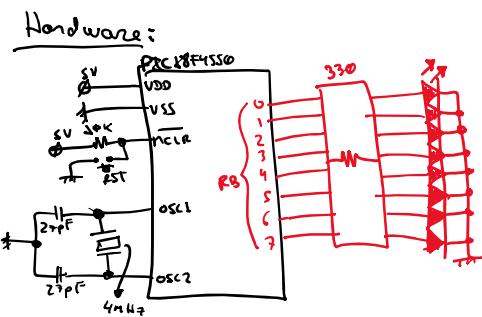
Diagrama de Flujo:



```

init_conf:
    clrf TRISB
    clrf LATB

loop:
    incf LATB, f
    call delay_long
    goto loop
    
```



## 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.

Fin de la sesión!