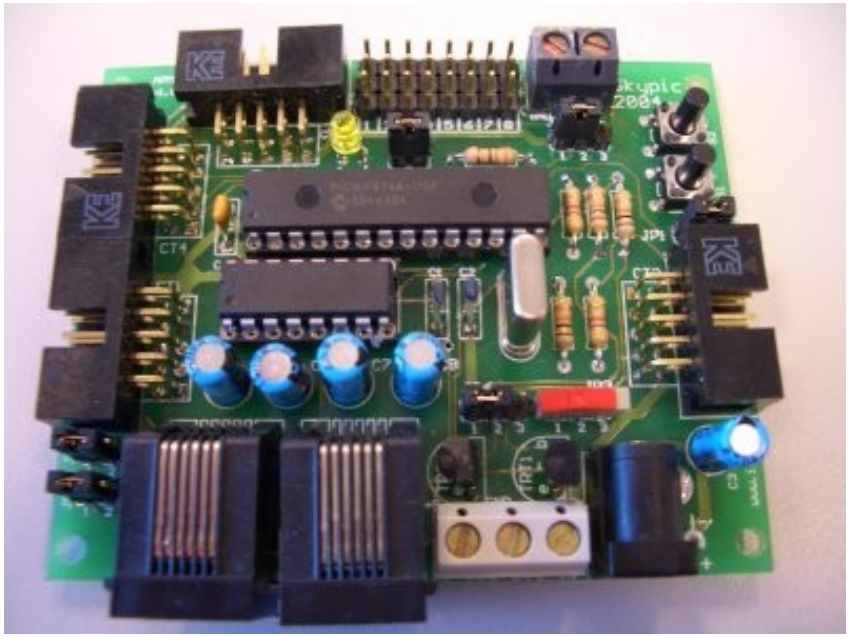
PIC 16F87X



Juan González

Escuela Politécnica Superior
Universidad Autónoma de Madrid

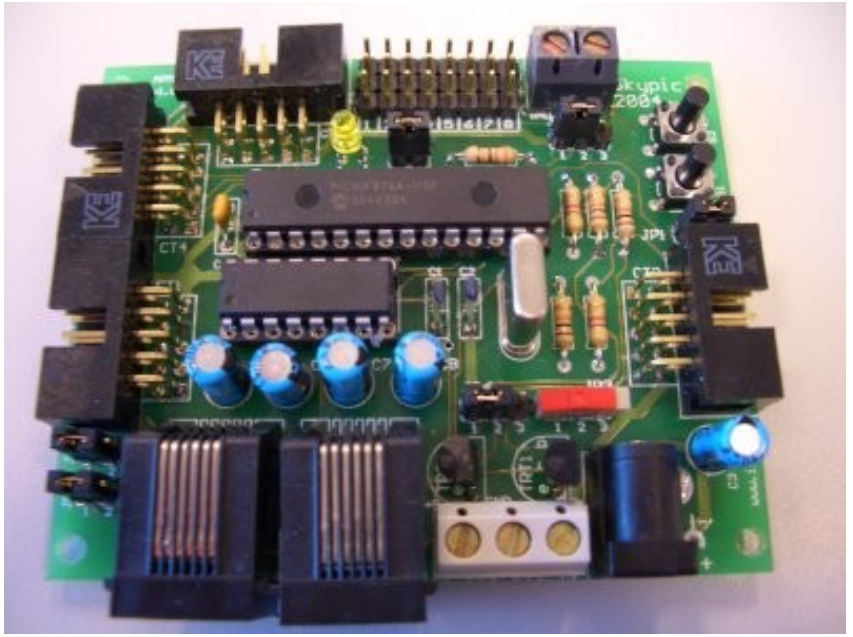
Andrés Prieto-Moreno

Flir Networked Systems

Ricardo Gómez

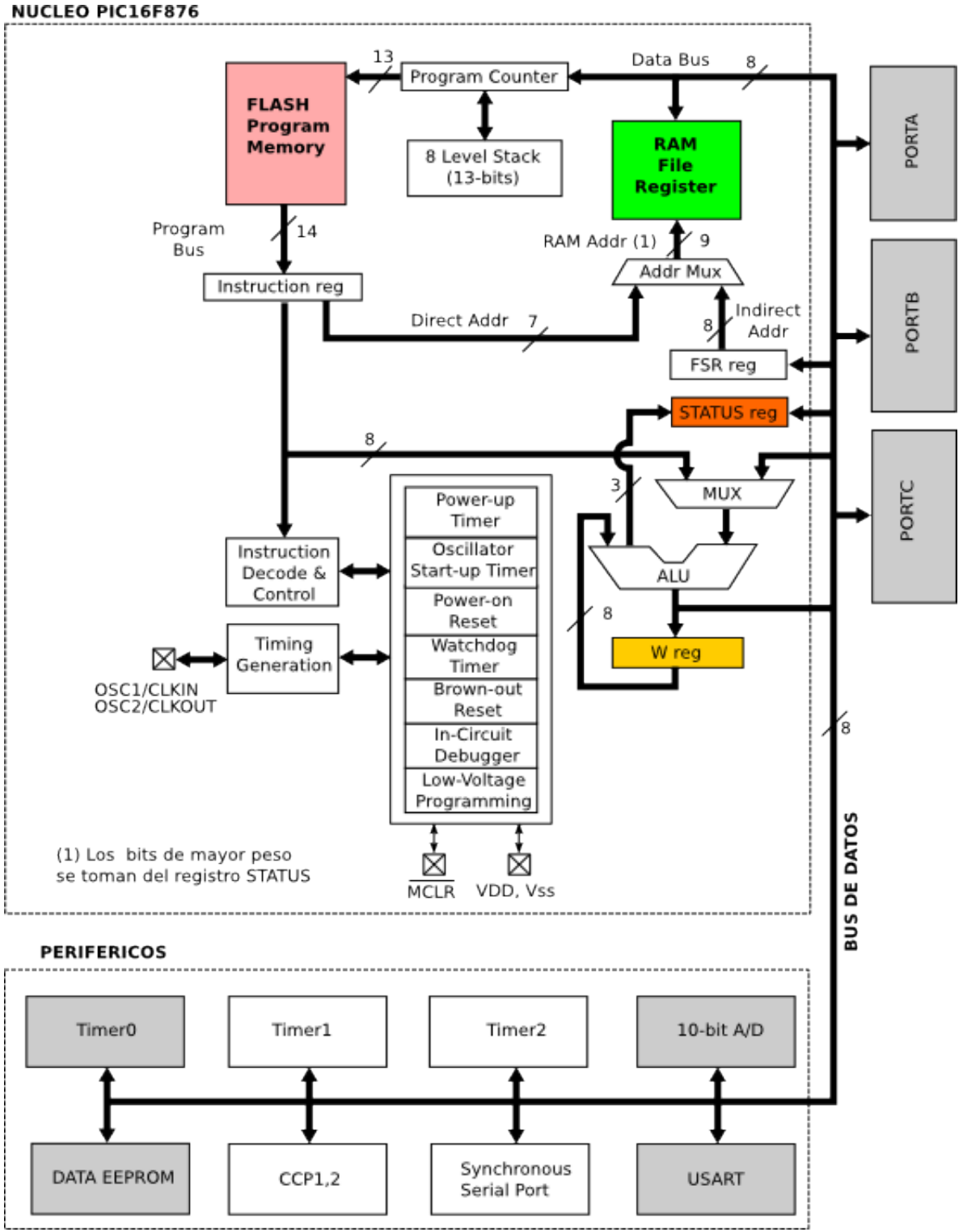
Flir Networked Systems

PIC 16F87X



MÓDULO 9:

Introducción a la programación en lenguaje ENSAMBLADOR



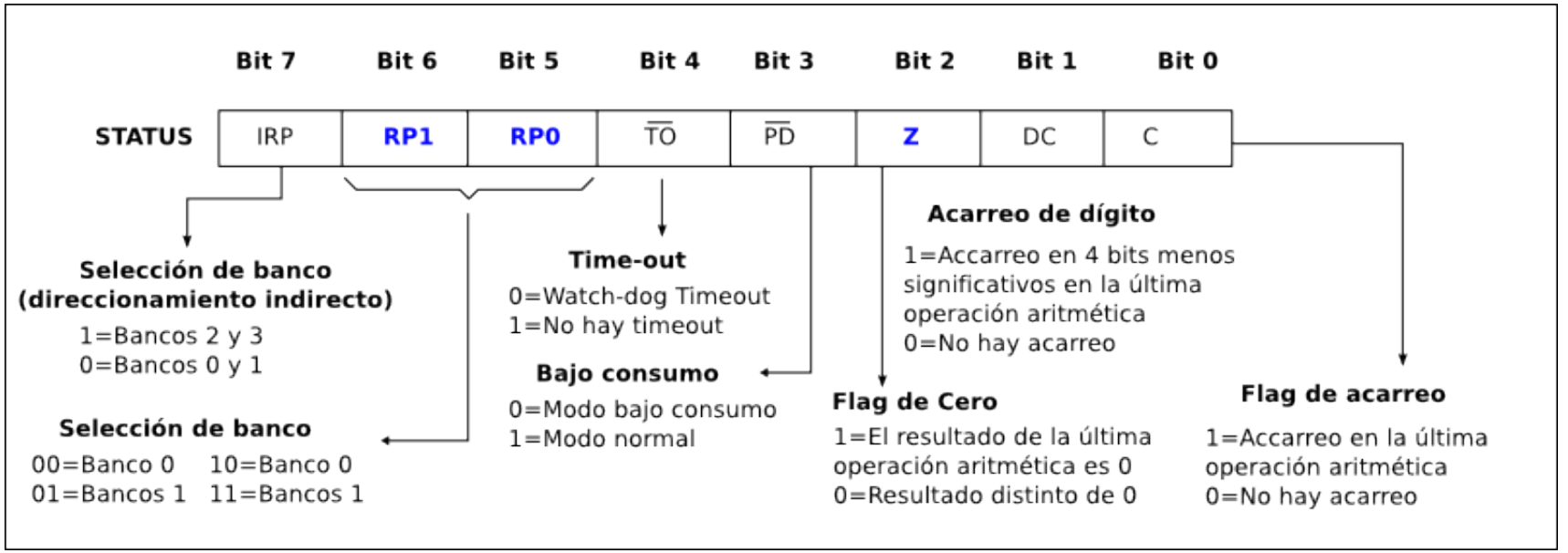
Núcleo del PIC16F876

Introducción

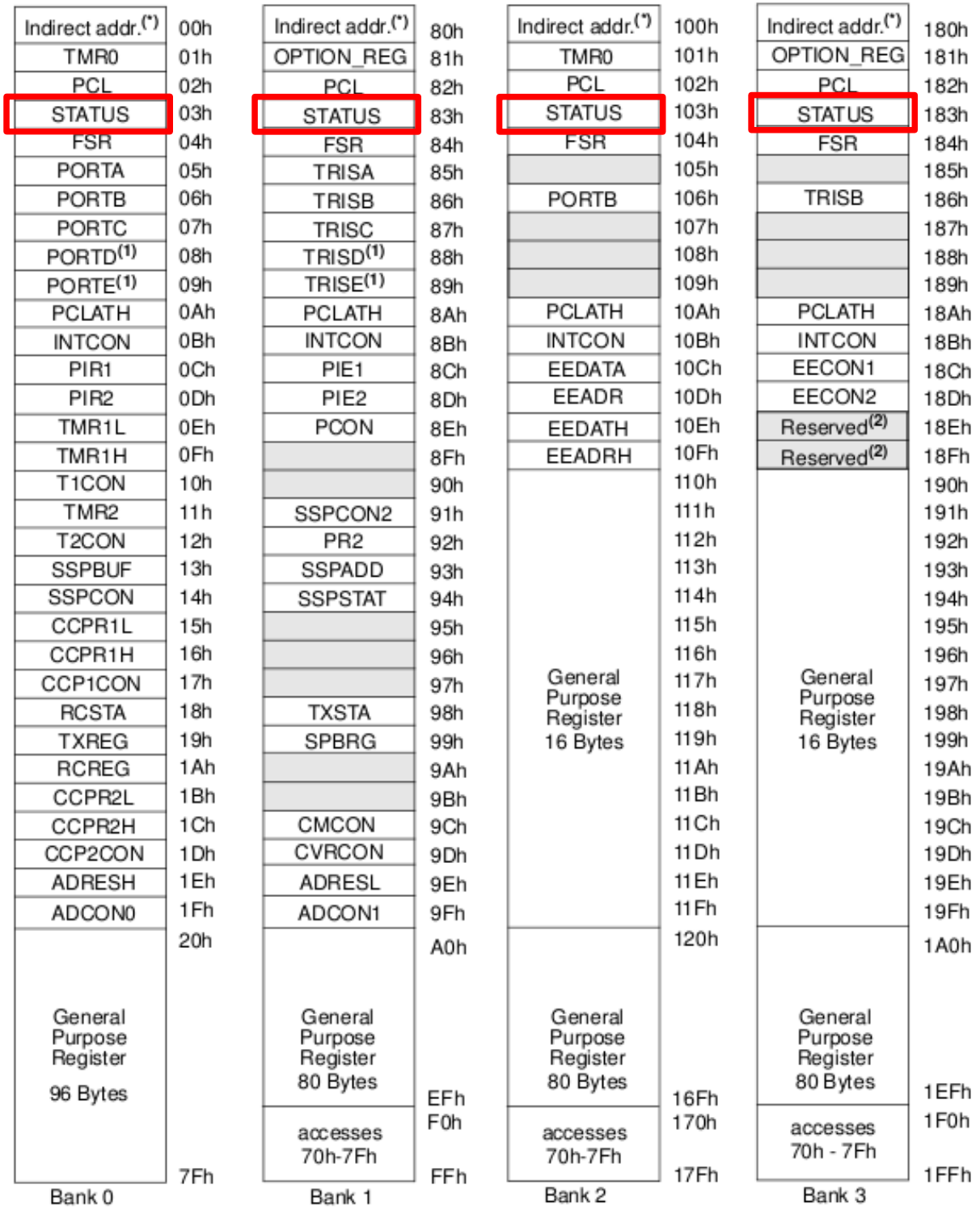
- Registros INTERNOS del PIC
  - W: Acumulador (Working register). Está conectado a una de las entradas de la ALU por lo que se usa como operando en todos los cálculos
  - STATUS: Registro de estado. Selección de los bancos de registros y flags de la ALU. **Acceso desde TODOS los bancos**
  - INDF: Registro de direccionamiento indirecto. **Acceso desde todos los bancos**

- Memoria RAM
  - Capacidad de **512 Bytes**
  - Dividida en 4 bancos de **128 bytes**: Banco 0, 1, 2 y 3
  - Antes de acceder a un registro hay que especificar el banco, mediante los bits RP0 y RP1 del registro de Status

Registro de status

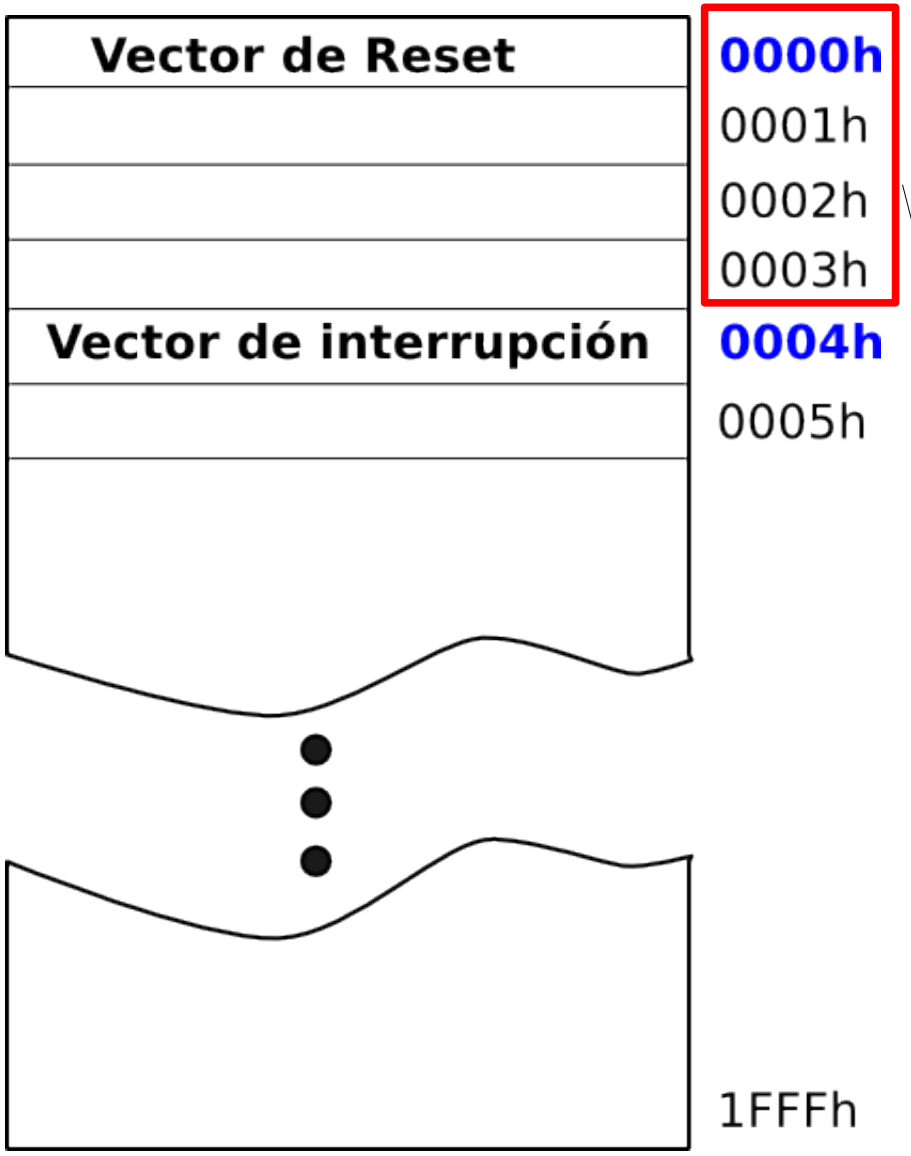


¡¡¡Registro accesible desde todos los BANCOS!!!



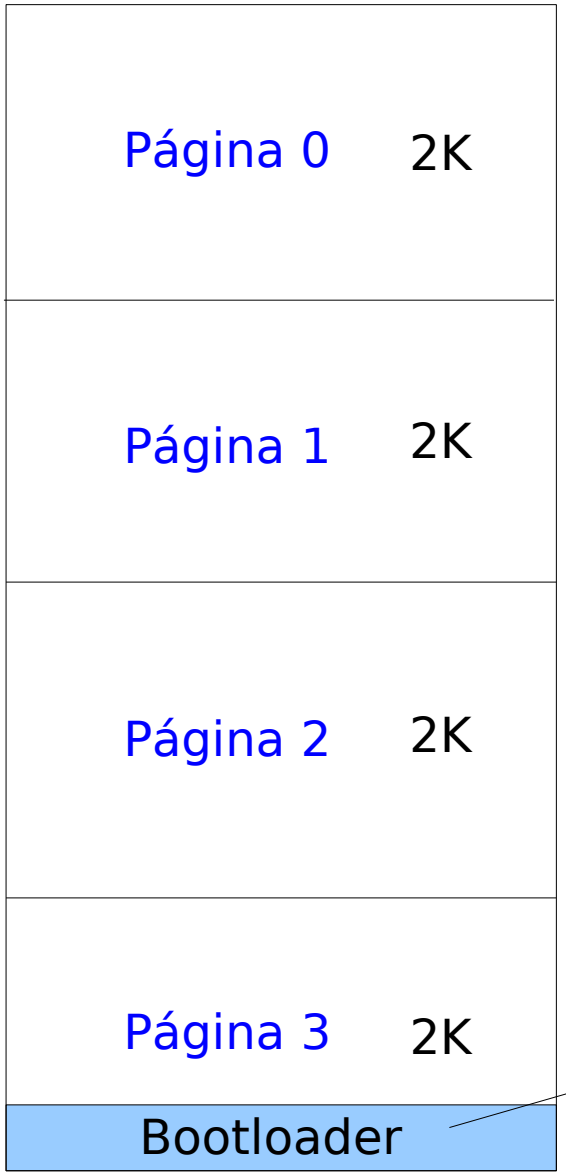
Bancos de registros

Mapa de memoria (flash)



- El PIC siempre empieza a ejecutar instrucciones a partir de la dirección 0000h (Vector de Reset)
- Cuando se produce una **interrupción** (la que sea) SIEMPRE salta a la dirección 0004h (Vector de interrupción)
- Hay 4 palabras disponibles para ejecutar un **código de inicialización** (Por ejemplo saltar al bootloader)

Mapa de memoria (flash) (II)



- Memoria dividida en 4 bancos de **2K** cada uno
- Para cambiar de un banco a otro hay que modificar el registro **PCLATH**

El bootloader (opcional) se graba en la parte alta de la memoria, en el banco 3

Juego de instrucciones

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected
				MSb			LSb	
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
MOVWF	f	Move W to f	1	00	0000	lfff	ffff	
NOP	-	No Operation	1	00	0000	0xx0	0000	
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	
LITERAL AND CONTROL OPERATIONS								
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk	
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk	
RETFIE	-	Return from interrupt	2	00	0000	0000	1001	
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk	
RETURN	-	Return from Subroutine	2	00	0000	0000	1000	
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z

Plantilla de programa

plantilla.asm

```
    INCLUDE "p16f876a.inc"

      __CONFIG  _RC_OSC  &  _WDT_ON  &
_PWRTE_OFF  &  _BODEN_ON  &  _LVP_ON  &
_CPD_OFF  &  _WRT_OFF  &  _DEBUG_OFF  &
_CP_OFF


  ORG 0
  ...
  goto start


  ORG 0x04
  ...
  retfie


start
  ...
fin goto fin


END
```

Cabecera. Indicar PIC a emplear

Establecer los fusibles de configuración

El programa comienza en la dirección 0

Poner código de arranque (opcional)

Saltar al comienzo del programa

Rutina de atención a las interrupciones

Código de atención a las interrupciones

Retorno de interrupción

Comienzo del programa principal

Nuestro código

Terminamos. Bucle infinito

Programa “Hola mundo”

ledon.asm

```
     INCLUDE "p16f876a.inc"
     ORG 0
     BSF STATUS,RP0
     BCF TRISB,1
     BCF STATUS,RP0
     BSF PORTB,1
fin  GOTO fin
     END
```

Encender el LED!!!

- No se usan interrupciones
- No se establece la palabra de configuración

Comienzo del programa

Primero configuramos el bit TRISB1 a 0, para que el pin RB1 sea de salida.

Para ello hay que acceder al BANCO 1: RP0=1

Ahora ponemos el bit RB1 a 1. Pero el registro PORTB se encuentra en el BANCO 0: RP0=0

Bit Set

Poner un bit de un registro a '1'

BSF Registro,Bit → **Número de bit (0-7)**

Dirección del registro

Bit Clear

Poner un bit de un registro a '0'

BCF Registro,Bit → **Número de bit (0-7)**

Dirección del registro

Hola mundo para Bootloader

¿Qué pasa si queremos usar un Bootloader para cargar los programas?

Antes de que nuestro programa arranque tiene que llamarse al Bootloader

ledon2.asm

```
   INCLUDE "p16f876a.inc"
   ORG 0
   CLRF    STATUS
   MOVLW   0
   MOVWF   PCLATH
   GOTO    start

start
    BSF STATUS,RP0
    BCF TRISB,1
    BCF STATUS,RP0
    BSF PORTB,1
fin GOTO fin

    END
```

Hay que añadir este código de arranque

Al cargarse el programa mediante el Bootloader, este lo modificará para que se llame primero al bootloader, y luego al programa del usuario

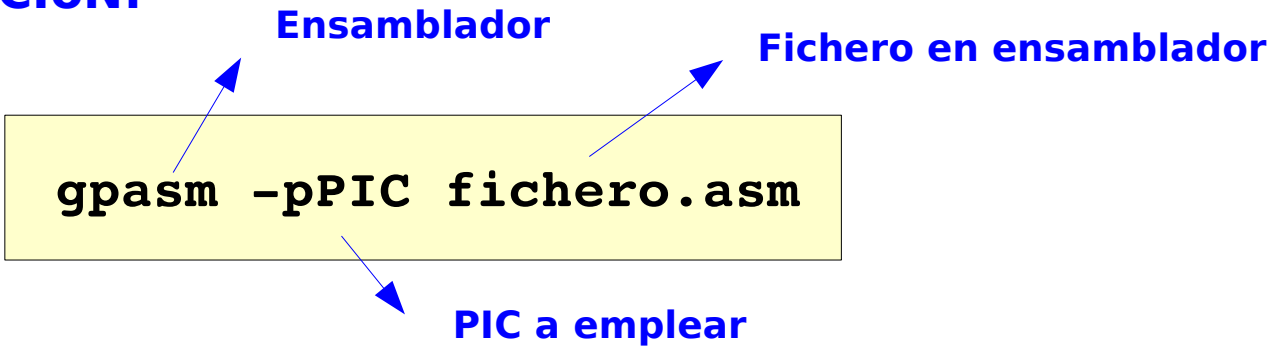
Si no hay bootloader, este código de arranque es "inofensivo". Simplemente selecciona la página 0 y salta al comienzo de nuestro programa

Nuestro programa hola mundo que enciende el led de la tarjeta Skypic

Compilando con las GPUTILS

- Las GPUTILS (GNU Pic Utilities) son las **herramientas libres** y multiplataformas para ensamblar y enlazar
- Son **100% compatibles con el MPLAB**

UTILIZACIóN:



EJEMPLO:

```
gpasm -pp16f876a ledon2.asm
```

Ejercicio: vamos a practicar

- Modificar el programa ledon2.asm para que además del bit RB1 se enciendan el **bit RB7** y se vea por los leds

- Primero habrá que **modificar el editor** para que nos permita invocar al ensamblador

Sacando datos por el puerto B

outputb.asm

Sacar un byte por el puerto B

```
  INCLUDE "p16f876a.inc"
  #define VALOR 0xAA

  ORG 0
  ...
  GOTO    start

start
  BSF  STATUS,RP0
  CLRF TRISB
  BCF STATUS,RP0
  MOVLW VALOR
  MOVWF PORTB
fin goto fin
 END
```

Definir una constante. Igual que en lenguaje C

Comienzo del programa. Lo primero sería el código de inicialización del Bootloader, igual que en el ejemplo anterior

Comienzo del programa

Acceder al banco 1

Poner todos los bits de TRISB a 0. Ahora todos los pines son de salida

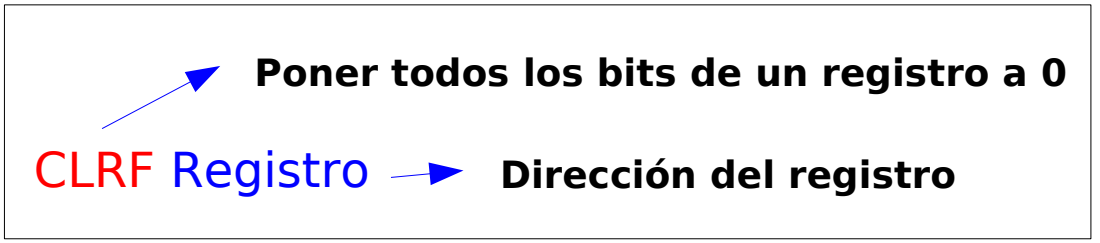
Acceder al banco 0

Cargar la constante “VALOR” en el acumulador (W)

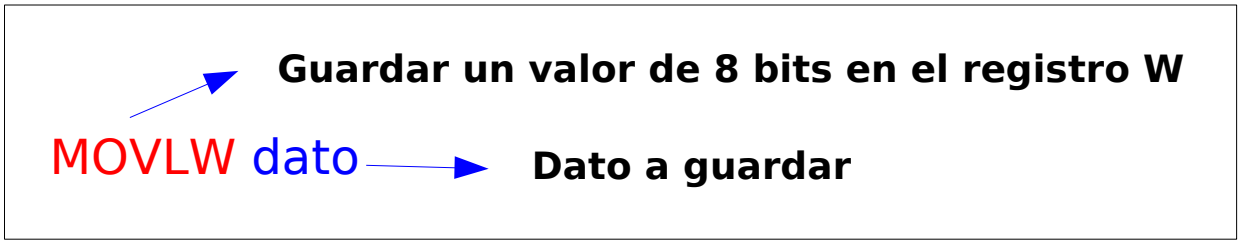
Sacar el acumulador por el puerto B

Instrucciones de transferencia y borrado

Clear



Move literal



Move

Comprobación de bits

pulsador.asm

```
   INCLUDE "p16f876a.inc"
   ...
start
   BSF STATUS,RP0
   BCF TRISB,1
   BCF STATUS,RP0
bucle
   BTFSC PORTB,0
   goto no_pulsado

   BSF PORTB,1
   goto bucle

no_pulsado
   BCF PORTB,1
   goto bucle

END
```

Programa que enciende el led al pulsar el botón y lo apaga al soltarlo

Inicialización (igual que ejemplos anteriores)

Cambiar al banco 1

Configurar RB1 para salida

Cambiar al banco 0

Comprobar el estado de RB0 y saltar si está a 0. Es decir, saltar si el pulsador está apretado

Si se ejecuta esta instrucción, es que no se ha realizado el salto y por tanto RB0 estaba a 1 (no pulsado). Saltar a la etiqueta correspondiente

Este es el código que se ejecuta cuando RB0 está a 0. Se enciende el led

Repetir el proceso para volver a comprobar el estado del pulsador

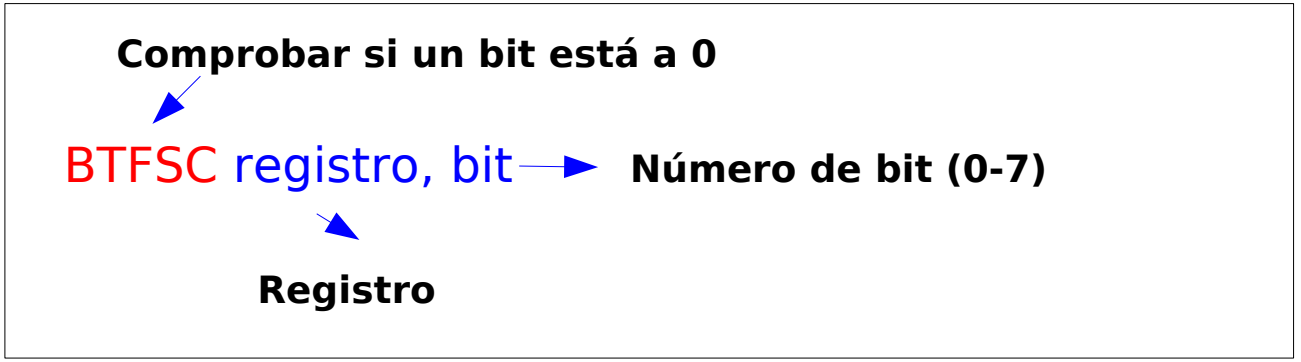
Este es el código que se ejecuta cuando RB0 está a 1

Apagar el led

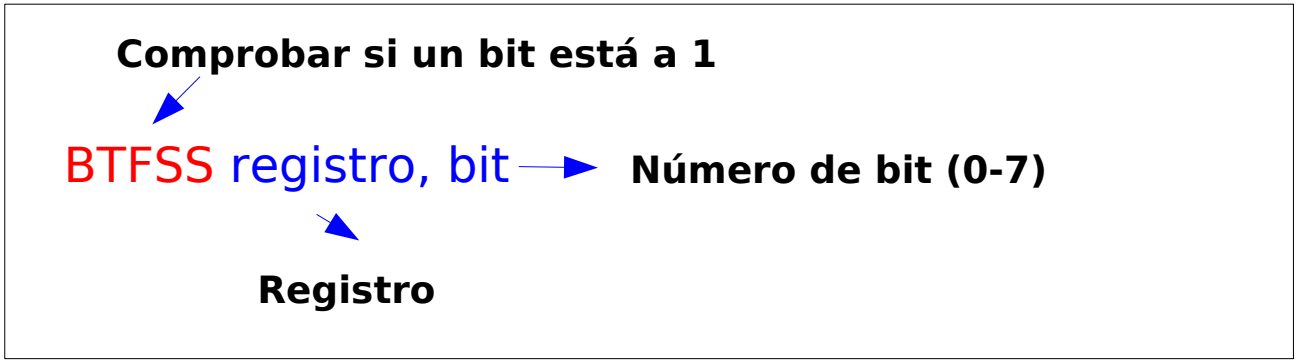
Repetir el proceso para volver a comprobar el estado del pulsador

Instrucciones de comprobación de bits

Bit Test (0)



Bit Test (1)



bucles

- Los bucles se implementan con la instrucción DECFSZ

```
bucle
  ...
  DECFSZ variable,F
  goto bucle
```

La variable se decrementa. Cuando variable==0, finaliza el bucle

SINTÁXIS:

Decrementar y saltar si es igual a cero

DECFSZ registro, destino

- **F: registro=registro -1**
- **W: W=registro-1**

Registro a usar

Bucles: Rutina de pausa (I)

ledp.asm (parte I)

Hacer parpadear el led (RB1)

```
  INCLUDE "p16f876a.inc"
  cblock 0x20
    CONTH
    CONTL
  endc
  ...
start
  BSF STATUS,RP0
  BCF TRISB,1
  BCF STATUS,RP0
main
  MOVLW 0x02
  XORWF PORTB,F
  CALL pausa
  GOTO main
...
```

Declaración de variables. A partir de la dirección 0x20

Parte alta del contador para hacer la pausa

Parte baja del contador para hacer la pausa

Configurar RB1 para salida

Bucle principal

Cargar el valor 0x02 en W

Hacer la operación PORTB=PORTB xor 0x02 para cambiar el pin RB1 de estado

Llamar a la subrutina de espera

Repetir. Bucle infinito

Bucles: Rutina de pausa (II)

ledp.asm (parte II)

```
pausa
  MOVLW 0xFF
  MOVWF CONTH
bucle1
  MOVLW 0xFF
  MOVWF CONTL
bucle2
  DECFSZ CONTL,F
  goto bucle2

  DECFSZ CONTH,F
  goto bucle1
  RETURN

END
```

Poner parte alta del contador a 0xFF

Bucle exterior

Poner parta baja del contador a 0xFF

Bucle interior

Decrementar CONTL y salir del bucle cuando llegue a 0

Decrementar CONTH y salir del bucle cuando llegue a 0

Se llega aquí cuando CONTH=0 y CONTL=0. En total se han realizado 0xFFFF iteraciones

Retorno de la subrutina

Puerto serie (I)

sci-eco.asm (I)

Hacer eco por el puerto serie

```
INCLUDE "p16f876a.inc"
...
start
  BSF STATUS,RP0
  MOVLW 0x81
  MOVWF SPBRG
  MOVLW 0x24
  MOVWF TXSTA
  BCF STATUS,RP0
  MOVLW 0x90
  MOVWF RCSTA
  BSF STATUS,RP0
  CLRF TRISB
  BCF STATUS,RP0
main
  CALL leer_car
  CALL enviar
  MOVWF PORTB
  GOTO main
...
```

Primero configuramos el puerto serie

Acceso al banco 1

Baudios: 9600

Configurar transmisor

Acceso al banco 0

Configurar receptor

Acceso al banco 1

Configurar puerto B para salida

Acceso al banco 0

Bucle principal

Esperar a que llegue un carácter

Hacer el eco

...y sacarlo por los leds del puerto B

Puerto serie (II)

sci-eco.asm (II)

```
leer_car
  BTFSS PIR1,RCIF
  GOTO leer_car
  MOVFW RCREG
  RETURN

enviar
wait
  BTFSS PIR1,TXIF
  goto wait
  MOVWF TXREG
  RETURN

END
```

Subrutina de recepción de datos por el SCI

Esperar hasta que RCIF=1

Equivalente a while(RCIF==0) en C

Leer carácter recibido y guardarlo en W

Retorno de la subrutina

Subrutina de transmisión de datos por el SCI

Se transmite el carácter que esté en W

Esperar hasta que TXIF=1

Equivalente a while(TXIF==0) en C

Transmitir el carácter

Retorno de la subrutina

Interrupciones (I)

sci-int1.asm

```
INCLUDE "p16f876a.inc"
  ORG 0
  ...
  GOTO    start

  ORG 0x04
  MOVLW 0x02
  XORWF PORTB,F
  MOVFW RCREG
  RETFIE

start
  ...
  BSF STATUS,RP0
  BCF TRISB,1
  BSF PIE1,RCIE
  BSF INTCON,PEIE
  BSF INTCON,GIE
  BCF STATUS,RP0
  BSF PORTB,1
main  GOTO main
  END
```

Rutina de atención a las interrupciones. Se ejecuta cuando llegue un carácter por el sci

Cambiar el led de estado

Leer el carácter recibido para limpiar el flag de interrupción

Retorno de interrupción

Programa principal

Inicializar puerto serie. Igual que ejemplo anterior

Acceso al banco 1

Configurar RB1 para salida

Activar la interrupción de dato recibido

Acceso al banco 0

Encender el led

El programa principal no hace nada

Interrupciones (II)

• El programa anterior funciona porque sólo se están utilizando las interrupciones para hacer una tarea

• Si en el bucle principal se hiciera otra tarea, no funcionaría correctamente

• Las interrupciones **pueden llegar en cualquier momento** y nos pueden cambiar los valores de los registros W y STATUS.

Regla de oro:

> Al comienzo de la rutina de atención a la interrupción hay que guardar los registros W y STATUS
>
> Al finalizar la rutina hay que restablecer sus valores

• Vamos a hacer un ejemplo con dos TAREAS

- TAREA 1: Hacer eco de lo recibido por puerto serie mediante interrupciones
- TAREA 2: Que parpadee el led

Interrupciones (III)

sci-int2.asm (I)

```
INCLUDE "p16f876a.inc"
  cblock 0x20
    CONTH
    CONTL
    save_w
    save_status
  endc
  ORG 0
  ...
  GOTO    start

  ORG 0x04
  MOVWF save_w
  SWAPF STATUS,W
  MOVWF save_status
  CALL leer_car
  CALL enviar
  SWAPF save_status,W
  MOVWF STATUS
  SWAPF save_w, F
  SWAPF save_w, W
  RETFIE
```

Declaración de variables

Variables para usar en la rutina de pausa

Variable para guardar el registro W

Variable para guardar el registro STATUS

TAREA 1: hacer eco

Guardar el contexto (registros W y STATUS)

Leer carácter recibido

Hacer eco

Recuperar el contexto

Interrupciones (IV)

sci-int2.asm (II)

```
start
   ...
   BSF STATUS,RP0
   BCF TRISB,1
   BSF PIE1,RCIE
   BSF INTCON,PEIE
   BSF INTCON,GIE
   BCF STATUS,RP0
   BSF PORTB,1

main
   MOVLW 0x02
   XORWF PORTB,F
   CALL pausa
   GOTO main
```

Configuración puerto serie

Acceso al banco 1

Configurar RB1 para salida (led)

Activar interrupción de dato recibido

Acceso al banco 0

Encender led

TAREA 2: Que parpadee el led

Cambiar led de estado

Pausa

Ejercicio:

- Hacer un **programa en ensamblador** para hacer parpadear el led de la Skypic usando el timer 0, **mediante interrupciones.**

- Los valores para configurar el timer0 los podéis sacar del ejemplo timer0-pausa10ms.c, Y para las interrupciones: timer0-onda10Khz-int.c

Ensamblador y C en el SDCC (I)

sdcc-asm.c

Hacer parpadear el led. Parte en C y parte en ASM

```
#include <pic16f876a.h>
...

void main(void)
{
  TRISB1=0;
  T0CS=0; PSA=0;
  PS2=1; PS1=1; PS0=1;
  RB1=0;
  while(1) {

    _asm
      MOVLW 0x02
      XORWF PORTB,F
    _endasm;

    pausa(10);
  }
}
```

Funciones de pausa (ejemplos anteriores)

Configurar RB1 para salida

Configurar Timer 0 para hacer pausas

Apagar led

Comienzo del bloque en ensamblador

Cambiar led de estado

Fin del bloque en ensamblador

Pausa de 100ms

Ensamblador y C en el SDCC (II)

- Mezclar código en C y ASM puede ser peligroso.

EXPERIMENTO:

- En el ejemplo anterior eliminar la línea que pone RB1=0;

```
void main(void)
{
  TRISB1=0;
  T0CS=0; PSA=0;
  PS2=1; PS1=1; PS0=1;
  RB1=0;
  ...
```

Eliminar esa “inocente” línea

- ¿Funciona el código? ¿Qué ocurre?

Ensamblador y C en el SDCC (III)

- Falla porque estamos accediendo a bancos de registros diferentes

```
void main(void)
{
  TRISB1=0;
  TOCS=0; PSA=0;
  PS2=1; PS1=1; PS0=1;

  while(1) {

    _asm
      MOVLW 0x02
      XORWF PORTB,F
    _endasm;

    pausa(10);
  }
}
```

Esos registros están en un banco distinto al 0

El puerto B está en el banco 0

El compilador “no lo sabe” y no nos cambia al banco 0

Sin embargo, cuando está la instrucción RB1=0, el compilador cambia al banco 0 y por eso al ejecutar nuestro código en ensamblador funciona

Conclusión: Mucho cuidado cuando se mezclan ambos lenguajes

Ensamblador y C en el SDCC (IV)

- ¿Para qué se usa la mezcla entre C y ASM principalmente?

Para tener control sobre los ciclos que la CPU ejecuta

```
while(1) {
  ...
  _asm
    nop
  _endasm;
  ...
}
```

Meter una instrucción “nop” que sabemos que tarda 200ns en ejecutarse (con cristal de 20Mhz)

Ejecutar instrucciones especiales

```
while(1) {
  ...
  _asm
    sleep
  _endasm;
  ...
}
```

Poner el pic en bajo consumo

```
while(1) {
  ...
  _asm
    clrwdt
  _endasm;
  ...
}
```

Reset del “Watch-Dog”