

2020

Arquitectura, Set de Instrucciones y Prog. en Assembler

Primer Parcial: 2/10 – 5/10

Recuperatorio:

Segundo Parcial:

Recuperatorio:

Entrega TP1 :

Entrega TP2:

Entrega TP3:

Entrega TP4:

Material Disponible en Web personal y Campus UNLa

Para reflexionar antes de la cursada

Los alumnos no “saben” porque el profesor enseña.

*Los alumnos generan su aprendizaje a partir de la aportación de los profesores que no son más que **facilitadores** de conocimiento.*

*El **saber hacer** se consigue cuando los alumnos logran conectarse con lo que se quiere.*

Compromiso, dedicación, intencionalidad y tiempo. Es el “costo” que los alumnos están dispuesto a pagar para alcanzar el objetivo de lo que se quiere.

Todo tiene un costo... nada es gratis o mágico.

Web Personal

www.licrgarcia.wixsite.com/misitio

The screenshot shows a Wix website editor interface. At the top, there are two tabs: "wix Panel de Control | Wix.com" and "wix Editor Wix - misitio". The address bar displays the URL <https://editor.wix.com/html/editor/web/renderer/edit/9908cdb6-a92f-4bf6-8408-49beec88ab9d?metaSiteId=289a854a-828f-4bb1-8e04-d3ea64822868&editorSessionId=e05f5e29-3d7a-41f7-a75f-fcb3ecdb4c8f&esi=e05f5e29-3d7a-41f7-a75f-fcb3ecdb4c8f&is...>. The browser toolbar includes icons for back, forward, search, and download.

The main content area shows a preview of a website. The header features a teal background with the Wix logo, a "Visitantes" counter (0 5 0 2 1), and a yellow button for "Lic. Roberto García" with a globe icon. The menu has links for "INICIO", "ASIGNATURAS", and "CONTACTO". The "ASIGNATURAS" section lists courses: "UNLa Curso Ingreso 2009", "UNLa Organización de Computadoras", "UNLa Arquitectura de Computadoras", "ITMR Bases de Datos", "UNLZ TIC", "UNLZ Sistemas de Información", "ISFT 172 Sistemas de Información I", "ISFT 172 Sistemas de Información II", "ISFT 172 Bases de Datos", "ISFT 172 Investigación Operativa", and "ITEL Modelos y Sistemas". The footer contains a link to the document's URL and standard system status icons.

Web Personal

www.licrgarcia.wixsite.com/misitio

Wix Panel de Control | Wix.com wix Editor Wix - misitio X +

https://editor.wix.com/html/editor/web/renderer/edit/9908cdb6-a92f-4bf6-8408-49beec88ab9d?metaSiteId=289a854a-828f-4bb1-8e04-d3ea64822868&editorSessionId=...

Aplicaciones wix Wix.com Google SIU Sueldo UNLZ Campus UNLa UNLa - SIGEBA Campus UNLZ UNLa - Sueldos Home | Edmodo Servicios ABC - Suel...

Otros favoritos

Wix Estás actualmente en modo vista previa Guardado Volver al editor

UNLa Arquitectura de Computadoras Notas FINAL 2do. Llamado Julio 2019

PPT de Clases expositivas

PPT

01 Arquitectura de Computadoras 2018 Pub

Bibliografía y Apuntes

PDF PDF

Trabajos Practicos Notas Examenes

https://docs.wixstatic.com/ugd/1b3847_bf4ca689a1ae4ba794eb99eba2bf1203.pptx...

Objetivos

- Introducción básica sobre la arquitectura de computadoras basada en un microcontrolador.
 - Introducción a las instrucciones de programación
 - Organización de la memoria
 - Escritura de un programa simple
-

Agenda

- Arquitectura Básica
 - Revisión del Set de Instrucciones
 - Modos de direccionamiento y Organización de la Memoria
 - Características Especiales
 - Ejercicios
-

Clase 1

Definición de un sistema programable

- En términos generales un sistema programable es un dispositivo o conjunto de dispositivos de propósito general, que según sea necesario se programa para resolver distintos problemas.
- El ejemplo mas conocido de sistema programable es un PC.

Aplicaciones de un sistema microprogramable

- Los sistemas programables tienen una gran variedad de aplicaciones, ya que simplemente variando la programación, se les puede indicar que realicen una función u otra, siendo las mas importantes:
 - **Aplicaciones informáticas**
 - **Cálculo matemático:**
 - **Automatización de Electrodomésticos**
 - **Automatización de Funciones en Automotores**
 - **Etc.**

Componentes de un Microprogramable

Un sistema microprogramable esta formado por los siguientes componentes:

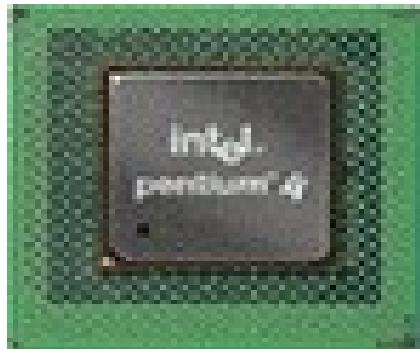
- *Hardware*
- *Software*
- *Firmware*

Definiciones

- **Hardware:** Es el conjunto de circuitos electrónicos que forman el sistema programable o, dicho de otra forma, **es la parte física del sistema.**
- **Software:** Es el **conjunto de programas y aplicaciones** formado por instrucciones y rutinas que se utilizan para programar y coordinar al sistema programable. También se denomina software al conjunto de lenguajes empleados para elaborar dichos programas.
- **Firmware:** Es un "**software**" **grabado en la estructura electrónica** del sistema programable y que el usuario en principio no puede alterar.

En los ordenadores personales, la BIOS (Basic Input/Output System) utiliza firmware y contiene un grupo de programas que sirven de intermediario entre el software y hardware.

Sistemas Micropogramables



Microprocesadores



Microcontroladores

La clasificación de los sistemas microprogramables se basa en:

- El número de distintos circuitos integrados que lo forman,
- Su capacidad de trabajo
- **El tratamiento de programas y datos basados en CPU**

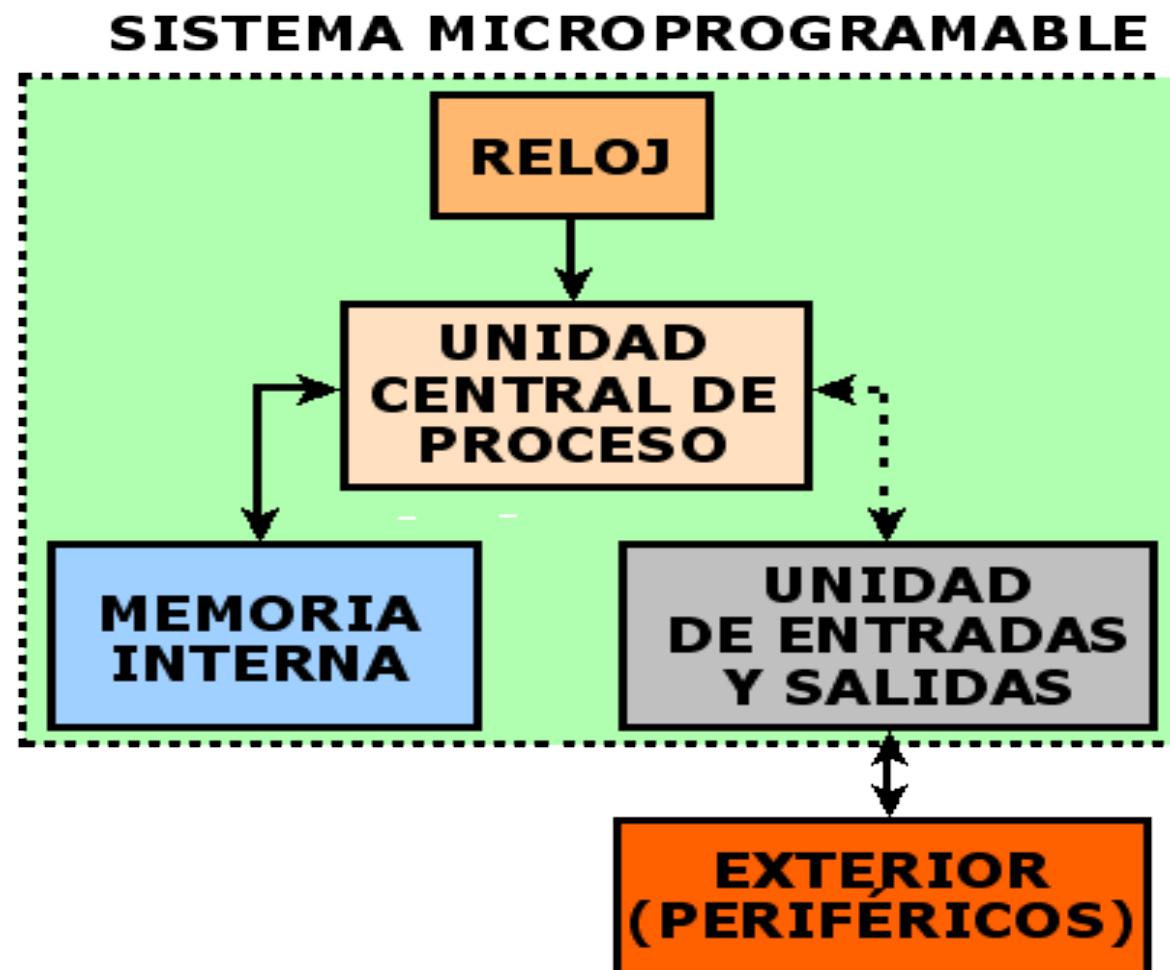
PLD's



Son los circuitos integrados con función lógica fija. internamente no tienen la estructura de microprocesadores y microcontroladores pues no están basados en una CPU que lea un programa de una memoria.

Estructura Básica de un Microprogramable

Microprocesadores Microcontroladores



Estructura Básica de un Microprogramable

Reloj: es un generador de ondas cuadradas periódicas, utilizado para que todo el sistema esté sincronizado.

Unidad Central de Proceso o CPU: Es la parte mas importante del sistema microprogramable. Es donde se realiza la **interpretación y ejecución de las instrucciones**, se generan todas las órdenes de control para gobernar todo el sistema y se realizan las **operaciones aritméticas y lógicas**.

Todo ello se realiza con los datos procedentes de la Memoria Interna o de registros internos. También, es la encargada de realizar todas las transferencias de datos hacia la memoria o desde esta.

La CPU está formada por:

Unidad Aritmética-Lógica

Acumuladores y Registros

Unidad de Control

Todas sus funciones se realizan en sincronía con la señal del reloj, por ello, la frecuencia del reloj define la velocidad del sistema.

Estructura Básica de un Microprogramable

Memoria Central o Interna: En los dispositivos que la forma, se encuentran los programas que debe utilizar el sistema microprogramable, los datos necesarios y los resultados que se generan.

Unidad de entrada/salida: Permite la comunicación del sistema microprogramable con el exterior. Su función fundamental es la de adaptar las diferentes velocidades y códigos utilizados por los elementos externos del sistema y el interior.

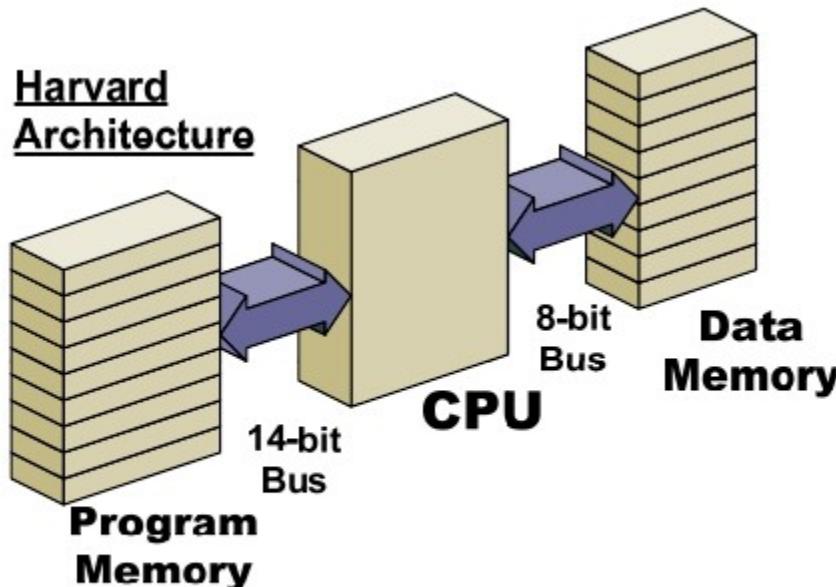
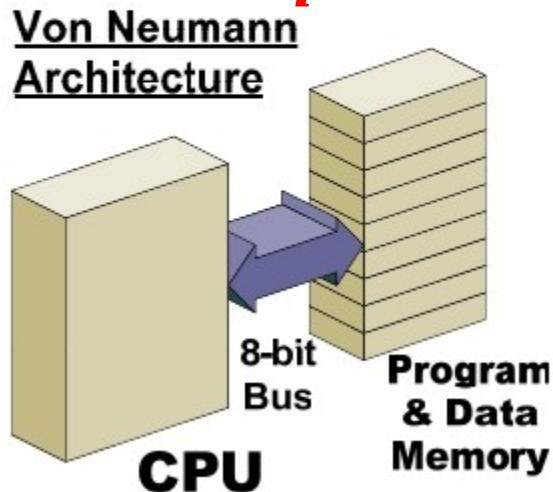
Periféricos: No forman parte del sistema microprogramable. Es un conjunto de dispositivos que realizan un trabajo en el exterior del sistema.

Estos periféricos pueden ser de entrada o de salida, aunque existen algunos que realizan ambas.

Un sistema microprogramable puede controlar una multitud de dispositivos de manera que también pueden considerarse periféricos un motor eléctrico, una electroválvula, un sensor de proximidad, un brazo robótico, etc.

Estructura Basica de un Microprogramable

Arquitectura Von Newmann y Harvard



Arquitectura Von Newmann:

- Busqueda de instrucciones y datos desde **una memoria simple**
- Ancho de Banda de operación Limitado

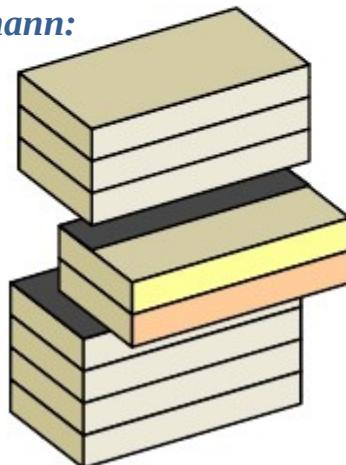
Arquitectura Harvard:

- Usa **dos memorias separadas** para Datos e Instrucciones
- Ancho de Banda de operación mejorado
- Permite diferentes anchos de Bus

Longitud de la palabra de Instrucción

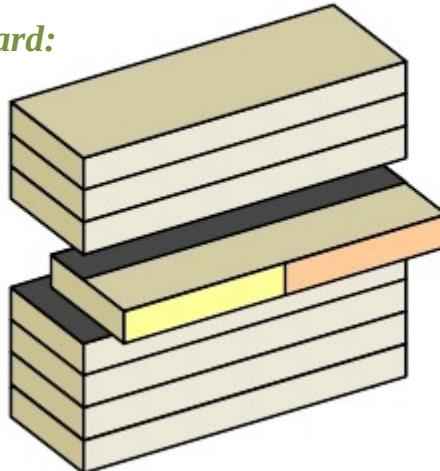
Memoria de Programa de 8 Bit

Arquitectura Von Newmann:



Memoria de Programa de 14 Bits

Arquitectura Harvard:



Instrucción de 8 bit sobre MCU de 8 Bits

Ejemplo: Freescale ‘Cargar Acumulador A’:

- 2 Localizaciones de Memoria
- 2 Ciclos de Instrucciones para Ejecutarse

ldaa #k

1	0	0	0	0	1	1	0
k	k	k	k	k	k	k	k

Limitado ancho
de banda

Incrementa los
requerimientos
de Memoria

14-bits de Instrucción sobre PIC16 MCU de 8 bits

Ejemplo: ‘Mover un valor Literal al registro Wr’

- 1 Localización en Memoria de Programa
- 1 Ciclo de instrucción para ejecutarse

movlw k

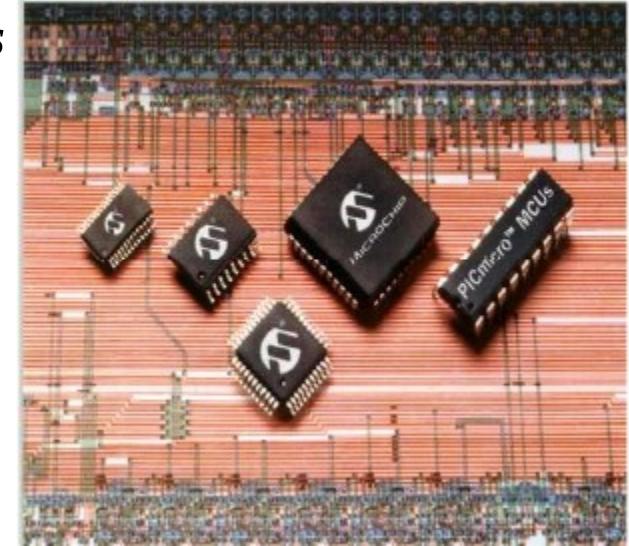
1	1	0	0	0	0	k	k	k	k	k	k	k	k
---	---	---	---	---	---	---	---	---	---	---	---	---	---

Buses separados Permiten diferentes
anchos

2k x 14 ies equivalente a 4k x 8

Características de Arquitectura Microcontroladores PIC.

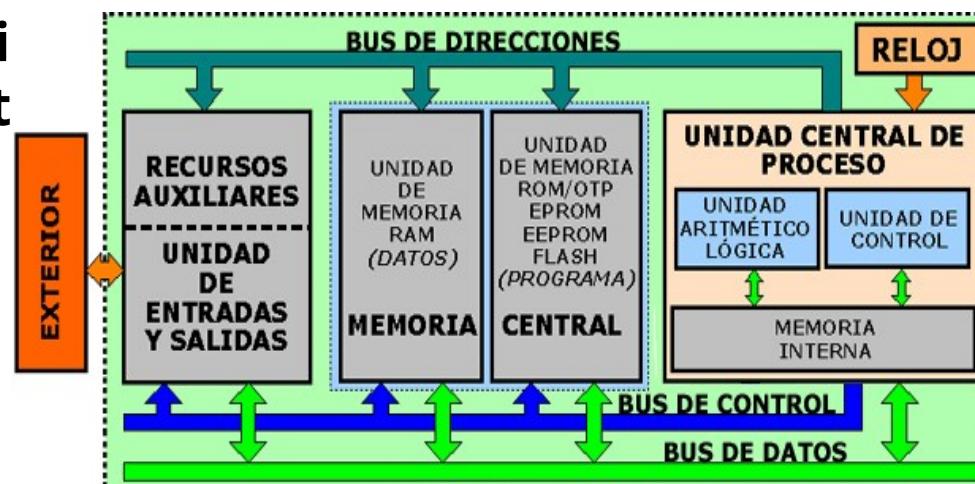
- *Arquitectura Harvard*
- *Pipelining (canalización) de Instrucciones*
- *Archivos de registros*
- *Instrucciones de un ciclo*
- *Instrucciones de una palabra*
- *Longitud de la palabra de Instrucciones*
- *Set de Instrucciones reducido*
- *Set de Instrucciones ortogonal*



Clase 2

Arquitectura Detallada de un Microcontrolador

- Procesador o CPU (Unidad Central de Proceso).
- Memoria Central:
 - Memoria de programa de tipo ROM/EPROM/EEPROM/Flash .
 - Memoria de datos de tipo RAM.
- Buses de control, datos y direcciones.
- Líneas de E/S para comunicarse con el exterior.
- Recursos auxiliares (temporizadores, Puertas Serie y Paralelo, Conversores Analógico/Digital, Conversores Digital/Analógico, etc.).
- Generador de i
funcionamiento



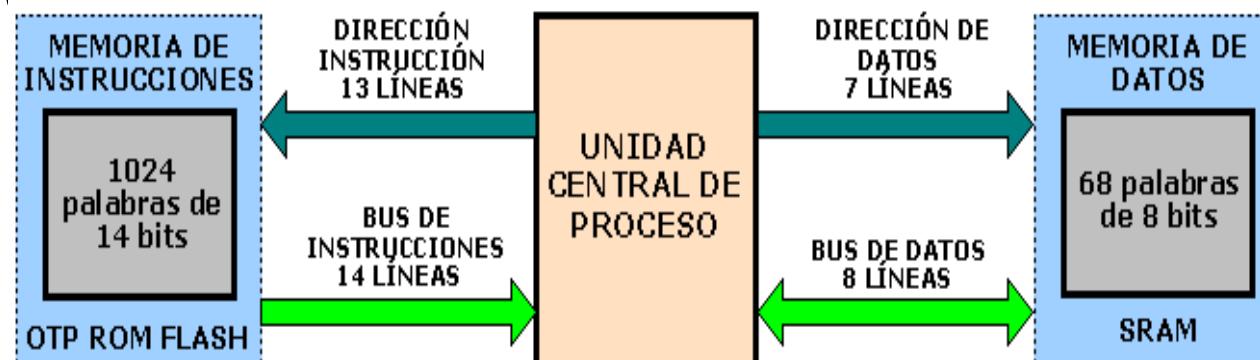
Organización y Gestión de la Memoria

Memoria de instrucciones o programa Es donde se almacena el programa a ejecutar. Esta memoria solo podrá ser leída por el PIC, que va leyendo las instrucciones del programa almacenado en esta memoria y las va ejecutando. Al apagar el microcontrolador esta memoria no se borra.

Memoria de datos RAM Incluye los registros que configuran el comportamiento del microcontrolador y los registros que almacenan los valores de las variables del programa. Al apagar el microcontrolador esta memoria no mantiene los datos.

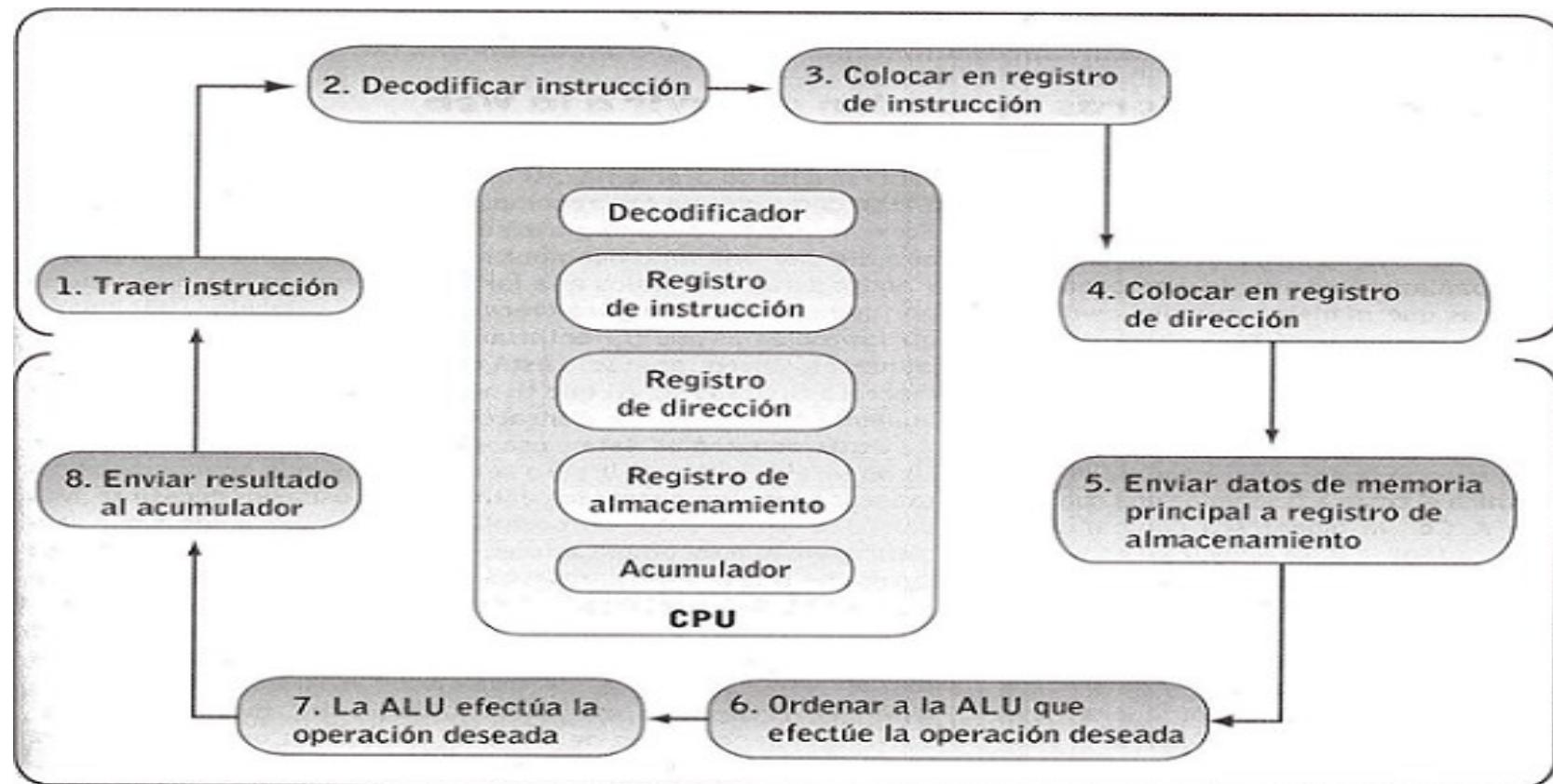
Memoria de datos EEPROM. Es un espacio de memoria EEPROM en la que se pueden guardar variables que se desea conservar aunque se apague el sistema.

ARQUITECTURA DE LA FAMILIA 16X84

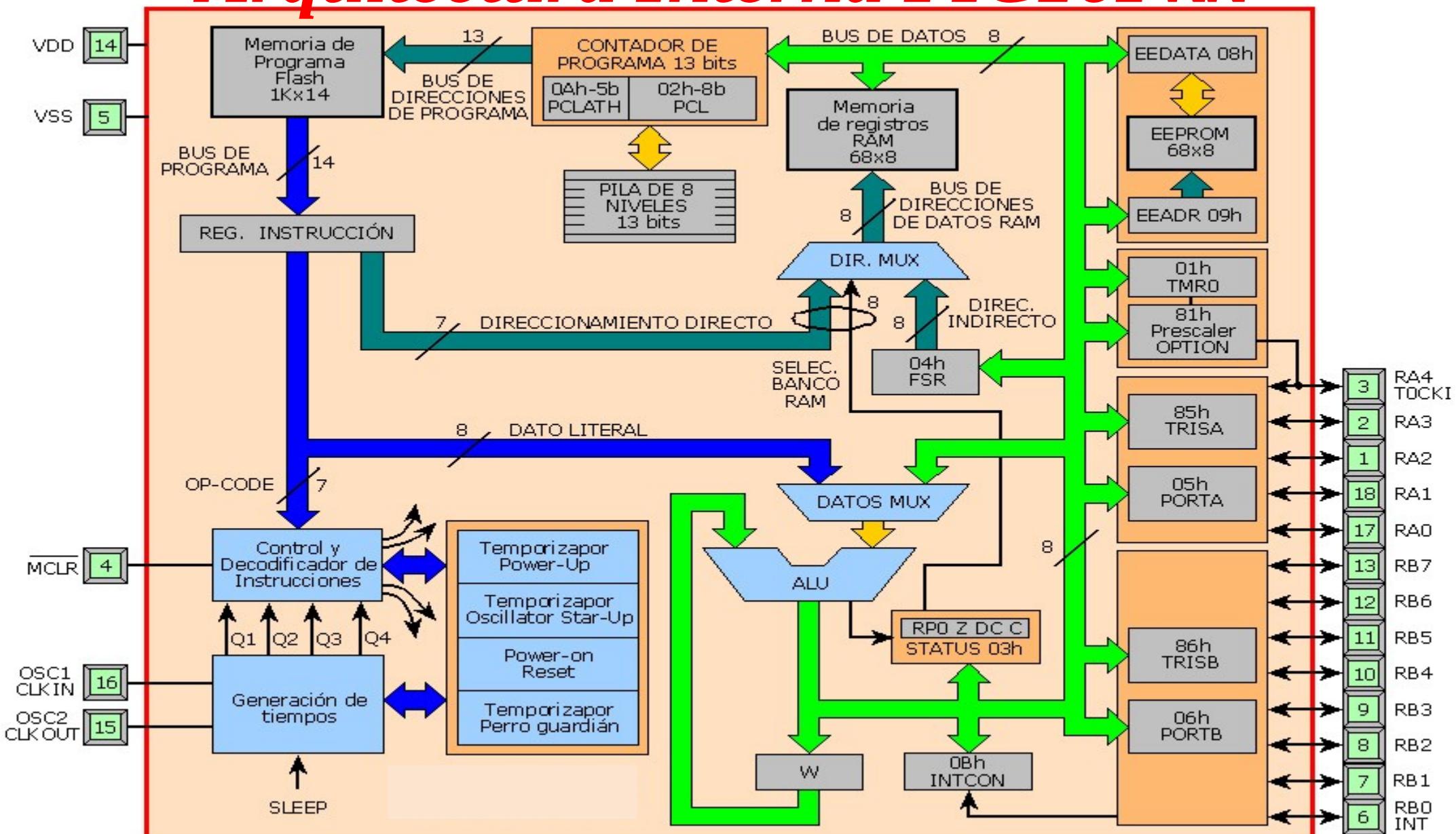


Ciclo de Máquina

Pasos del ciclo de máquina. Tiene dos etapas de operación principales: el **ciclo de instrucción** y el **ciclo de operación**. Cada ciclo consta de varios pasos que se requieren para procesar una sola instrucción de máquina en la CPU.



Arquitectura Interna PIC16Fxx



Arquitectura Interna PIC16Fxxx - Memoria de Programa

Espacio de Memoria del Usuario es donde irá el programa, desde la dirección 0000h hasta la 3FFh (3FFh en decimal es 1023).

Reset Vector es la primera dirección (0000h) a la que se dirige el PIC al encenderlo o al resetearlo y donde debe estar siempre la primera instrucción.

Vector de Interrupción es la dirección (0004h) a la que se dirige el PIC cuando se produce una interrupción, esto es, un evento que permite sacar al PIC de la ejecución normal del programa para ejecutar una subrutina de atención a la interrupción.

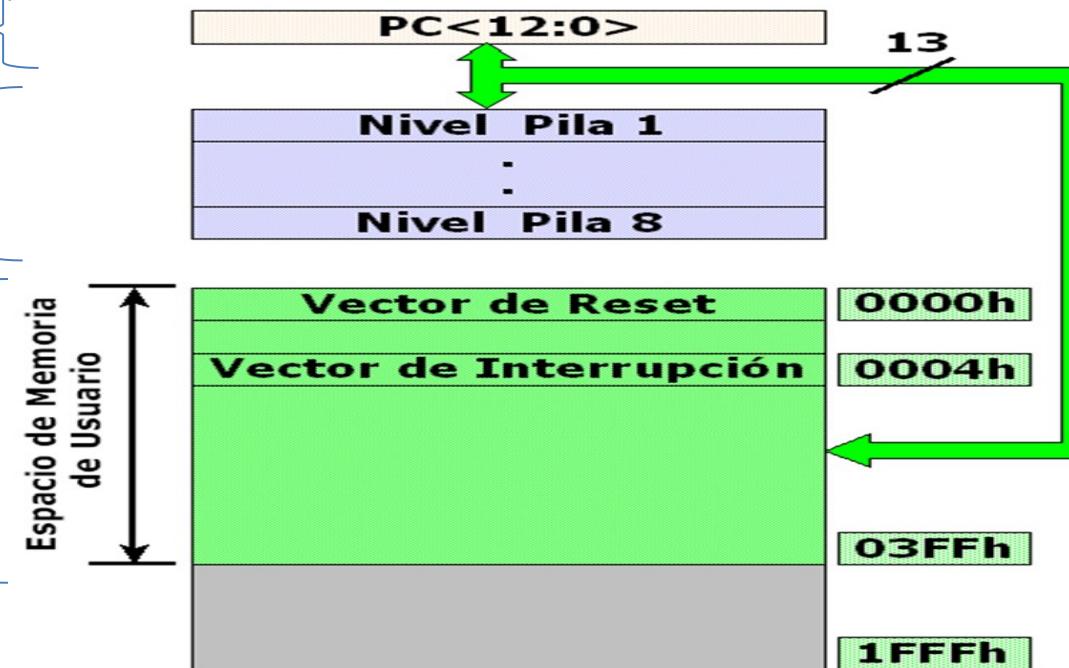
PC (Contador de Programa) es un registro de 13 bits que apunta a la dirección de la memoria de programa que contiene la instrucción a ejecutar.

Niveles de la pila de 1 a 8 son los niveles de la pila, que se utiliza cuando se ejecutan subrutinas.

*Contador
del Programa*

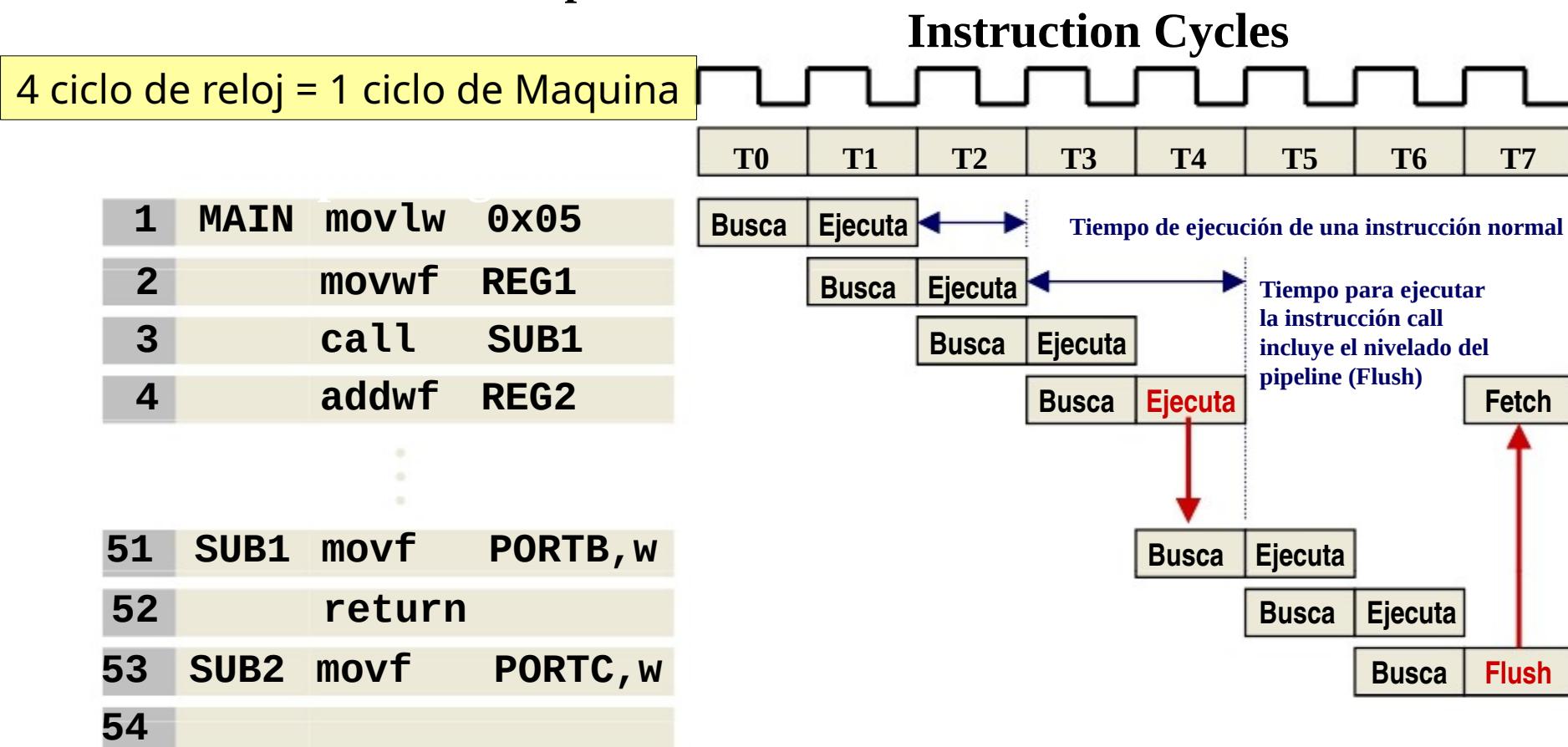
Pila

*Memoria
del Usuario*



Pipelining de Instrucciones

La búsqueda de instrucciones se superpone con la ejecución de instrucciones previamente buscadas



Pipelining de Instrucciones

Búsca Instrucción

movlw 0x05

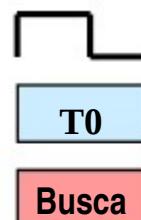
Ejecuta Instrucción

-

Ciclos de Instrucción

Programa Ejemplo

1	MAIN	movlw	0x05
2		movwf	REG1
3		call	SUB1
4		addwf	REG2
⋮			
51	SUB1	movf	PORTB,w
52		return	
53	SUB2	movf	PORTC,w
54		return	



Pipelining de Instrucciones

Búsca Instrucción

movwf REG1

Ejecuta Instrucción

movlw 0x05

Programa Ejemplo

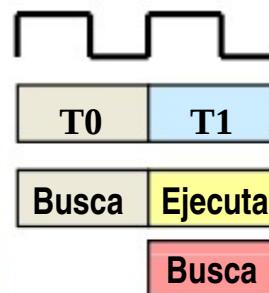
1 MAIN movlw 0x05

2 movwf REG1

3 call SUB1

4 addwf REG2

Ciclos de Instrucción



51 SUB1 movf PORTB,w
52 return
53 SUB2 movf PORTC,w
54 return

Pipelining de Instrucciones

Búsca Instrucción

call SUB1

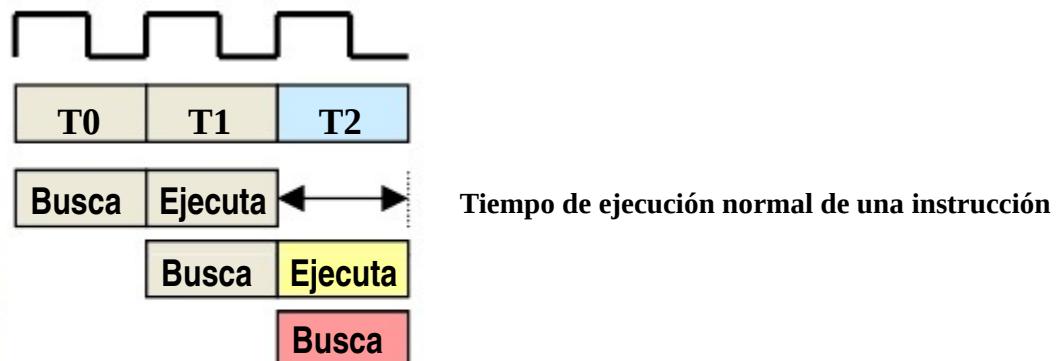
Ejecuta Instrucción

movwf REG1

Programa Ejemplo

```
1 MAIN    movlw 0x05
2          movwf REG1
3          call    SUB1
4          addwf  REG2
```

Ciclos de instrucción



```
51 SUB1   movf    PORTB,w
52          return
53 SUB2   movf    PORTC,w
54          return
```

Pipelining de Instrucciones

Búsca Instrucción

addwf REG2

Ejecuta Instrucción

call SUB1

Programa Ejemplo

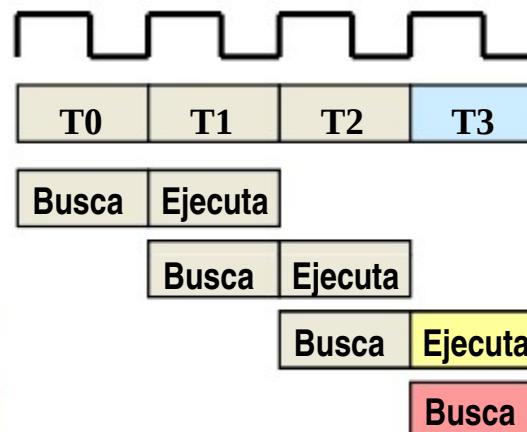
1 MAIN movlw 0x05

2 movwf REG1

3 call SUB1

4 addwf REG2

Ciclos de Instrucción



51 SUB1 movf PORTB,w

52 return

53 SUB2 movf PORTC,w

54 return

Pipelining de Instrucciones

Búsca Instrucción

movf PORTB, w

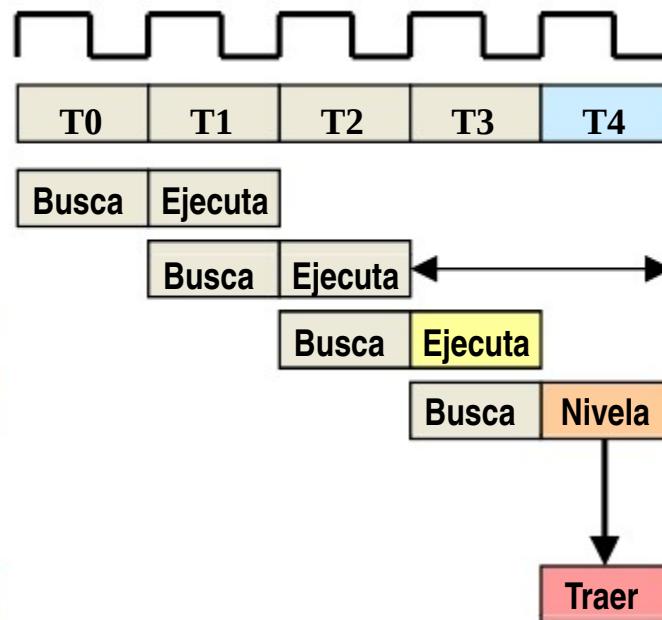
Ejecuta Instrucción

call SUB1

Programa Ejemplo

1	MAIN	movlw	0x05
2		movwf	REG1
3		call	SUB1
4		addwf	REG2
51	SUB1	movf	PORTB, w
52		return	
53	SUB2	movf	PORTC, w
54		return	

Ciclos de Instrucción



Tiempo de ejecución
del call incluido el
flush del pipeline

Pipelining de Instrucciones

Búsca Instrucción
return

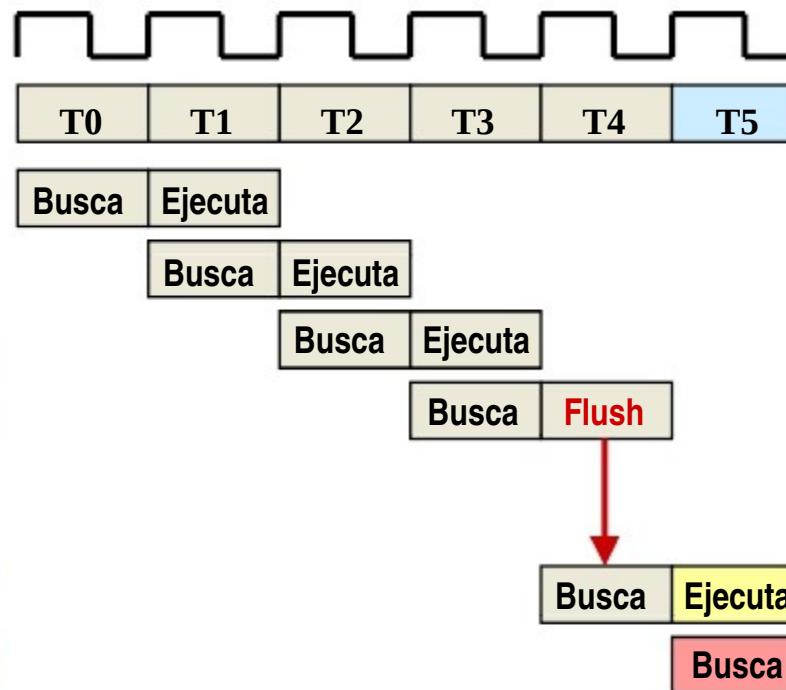
Ejecuta Instrucción
movf PORTB, w

Programa Ejemplo

1 MAIN movlw 0x05
2 movwf REG1
3 call SUB1
4 addwf REG2

51 SUB1 movf PORTB, w
52 return
53 SUB2 movf PORTC, w
54 return

Ciclos de Instrucción



Pipelining de Instrucciones

Búsca Instrucción

movf PORTC, w

Ejecuta Instrucción

return

Programa Ejemplo

1 MAIN movlw 0x05

2 movwf REG1

3 call SUB1

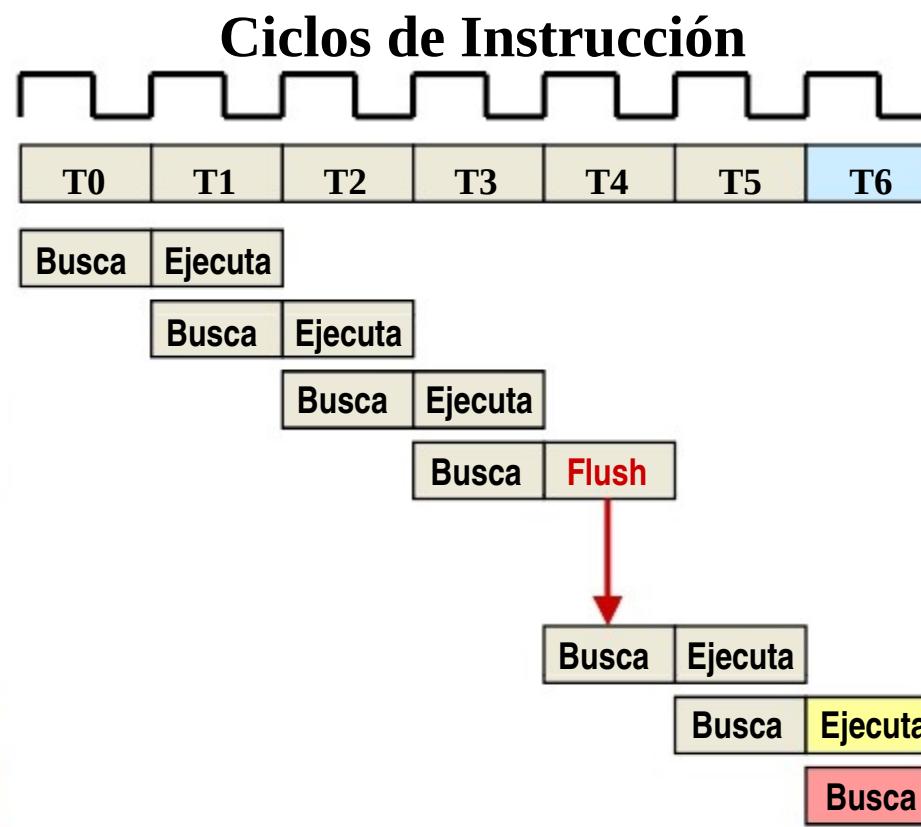
4 addwf REG2

51 SUB1 movf PORTB, w

52 return

53 SUB2 movf PORTC, w

54 return



Pipelining de Instrucciones

Búsca Instrucción

addwf REG2

Ejecuta Instrucción

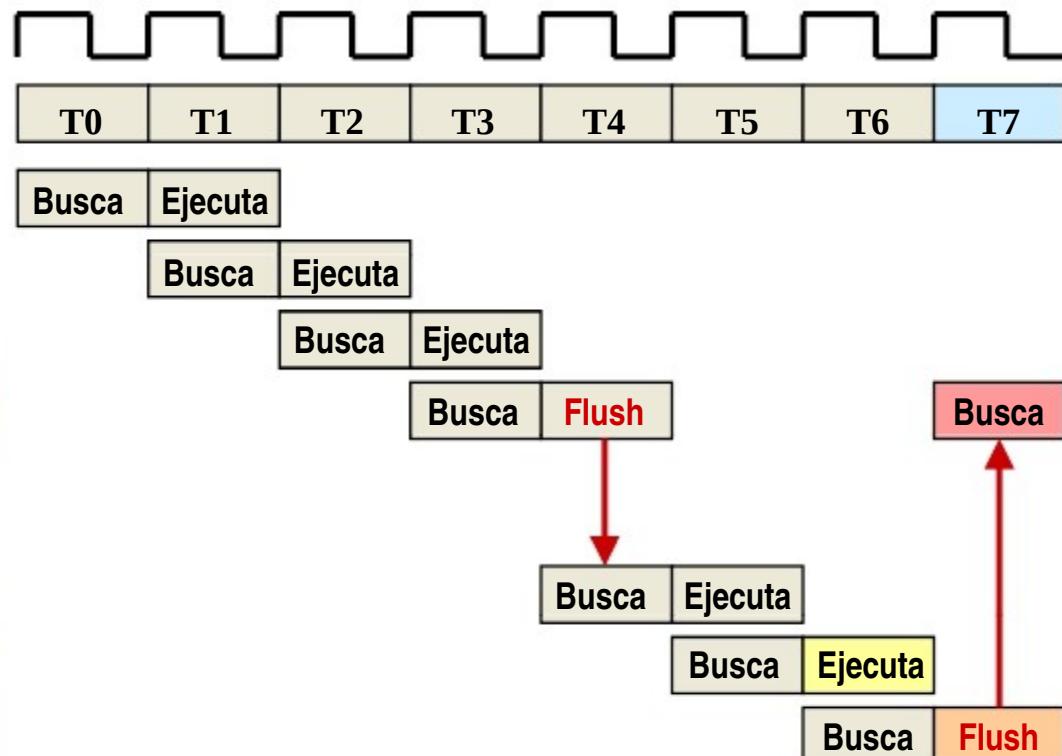
return

Programa Ejemplo

1 MAIN movlw 0x05
2 movwf REG1
3 call SUB1
4 addwf REG2

51 SUB1 movf PORTB,w
52 return
53 SUB2 movf PORTC,w
54

Ciclos de Instrucción



Clase 3

Arquitectura de Computadoras

Organización y gestión de la memoria

- *Memoria RAM*
- *Registros de Trabajo*

Memoria de Datos

Almacena los datos variables y los resultados temporales. Debe permitir lectura y escritura. En un microcontrolador se suele encontrar:

- **RAM:** Memoria de lectura y escritura muy rápida y volátil. Algunas posiciones de la memoria se tratan como registros.
- **EEPROM:** Memoria de lectura y escritura lenta pero no volátil.

Unidad Aritmético Lógica



La ALU

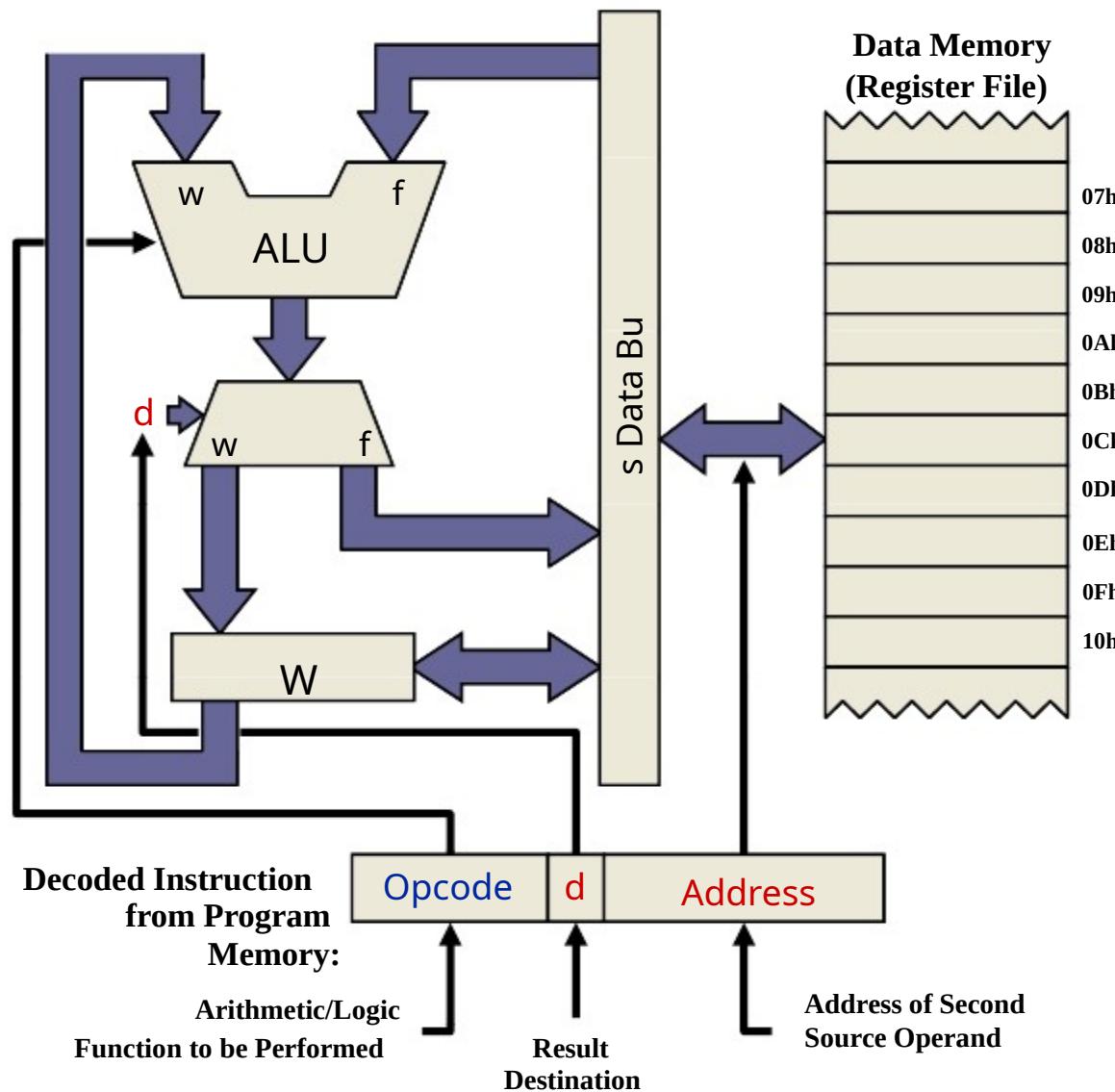
El PIC16F84A posee una ALU (Unidad Aritmético Lógica) de 8 bits capaz de realizar operaciones de desplazamientos, lógicas, sumas y restas. Su salida va al **registro de trabajo W** y también a la **memoria de datos**, por lo tanto el resultado puede guardarse en cualquiera de los dos destinos. **Dependiendo de la instrucción ejecutada**, La ALU puede afectar a los bits de **Acarreo**, **Acarreo Digital (DC)** y **Cero (Z)** del **Registro de Estado (STATUS)**.

Registro de trabajo

El acumulador o registro de trabajo (W) **es el registro mas utilizado de todos**.

No se trata de un registro de la RAM ya que **no tiene dirección** pero se usa constantemente para mover datos y dar valores a las variables (registros). Por ejemplo, si queremos copiar la información del registro 0Ch en el registro 0Dh **no podremos hacerlo directamente**, deberemos usar una instrucción para cargar el valor del registro 0Ch en el acumulador y después otra instrucción para cargar el valor del acumulador en el registro 0Dh.

Concepto del Archivo de Registros



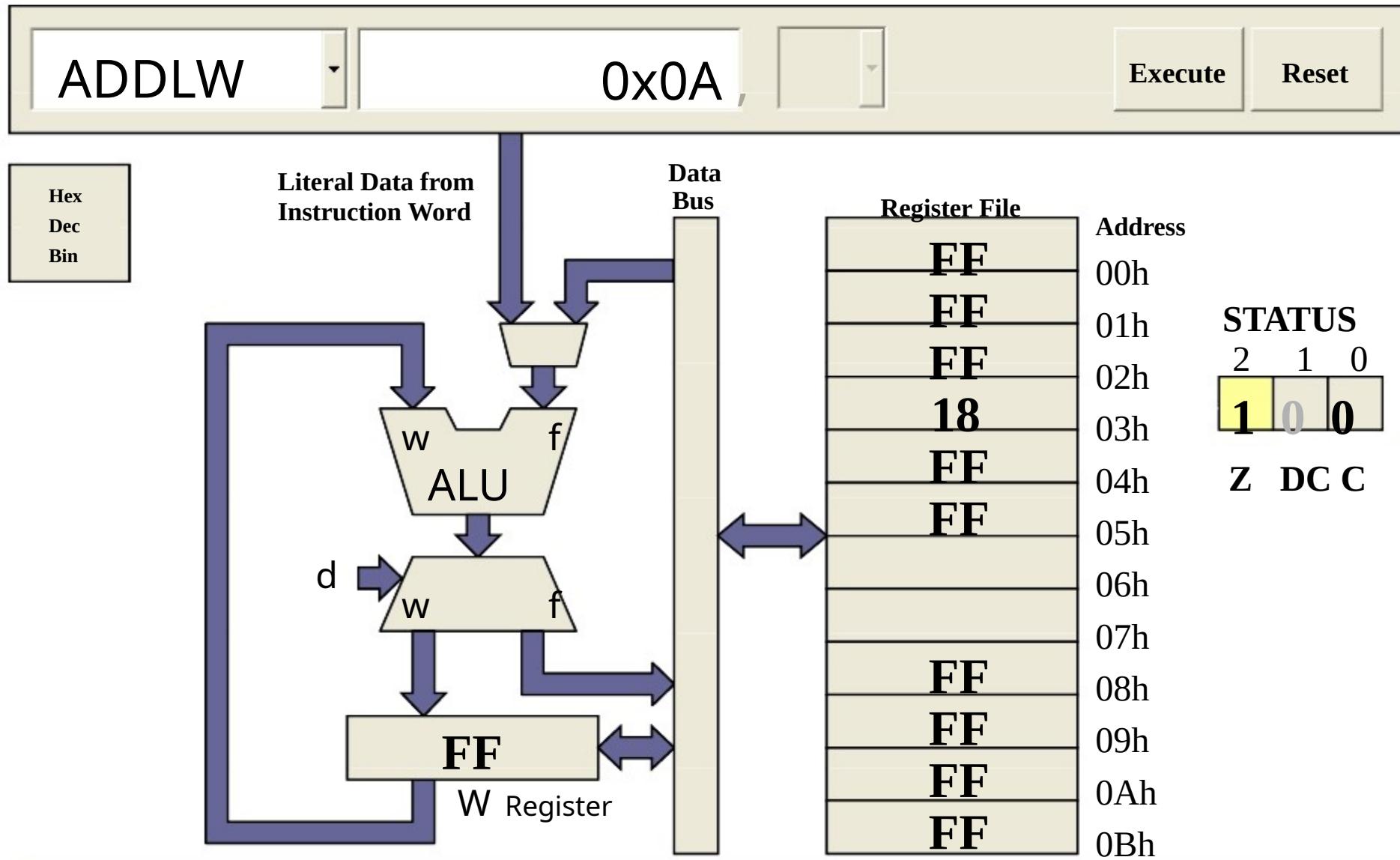
Concepto de Archivo de Registro: Todos los datos de memoria son parte del archivo de registro, cualquier localización puede ser operada directamente

Todos los periféricos están mapeados como una serie de registros

Set de instrucciones Ortogonal: Todas las instrucciones pueden operar sobre cualquier localización de memoria de datos

La longitud del formato de la palabras de instrucción permite un direccionamiento directo del archivo de registros

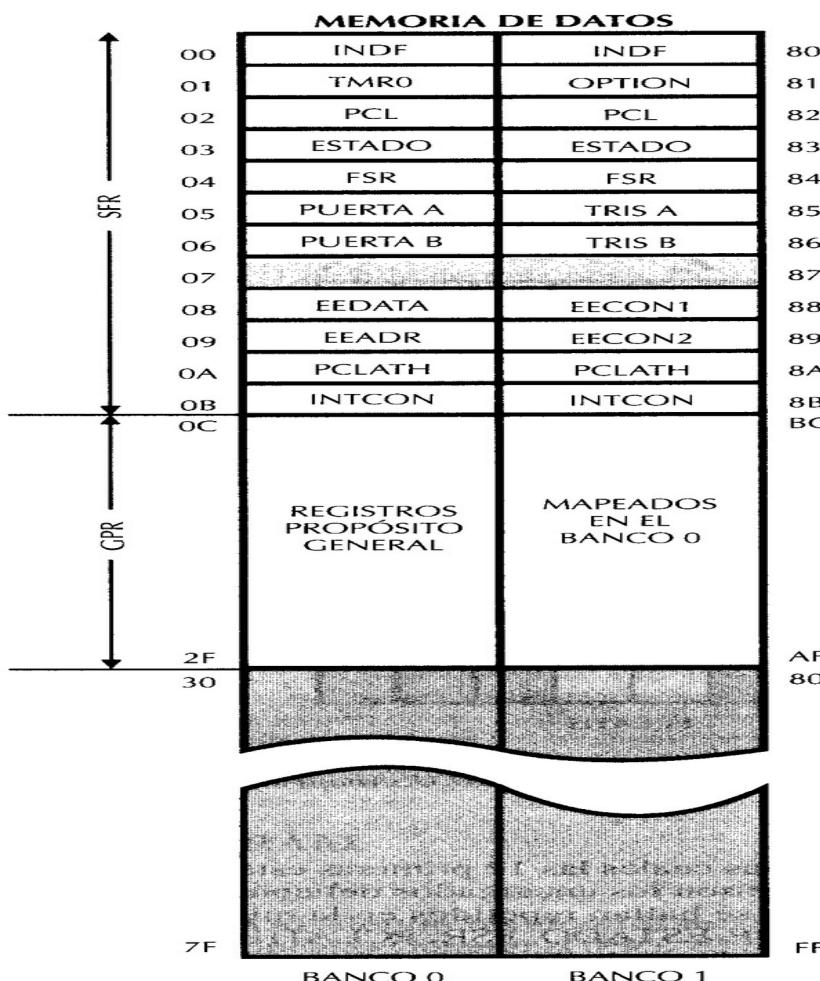
Instérprete Visual PIC16



Clase 4

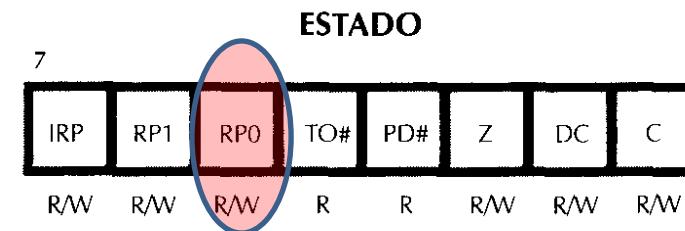
Arquitectura Interna PIC16Fxx - Memoria de Datos

Está organizada en páginas o bancos de registro. Para cambiar de página se utiliza un bit del registro STATUS (RP0).



Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind. ¹	Dir. Ind. ¹	80h
01h	TMRO	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	-	-	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2 ¹	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 REGISTROS DE PROPÓSITO GENERAL		8Ch
4Fh	MAPEADOS (ACCESO) EN EL BANCO 0		Cfh
50h			D0h
7Fh			FFh

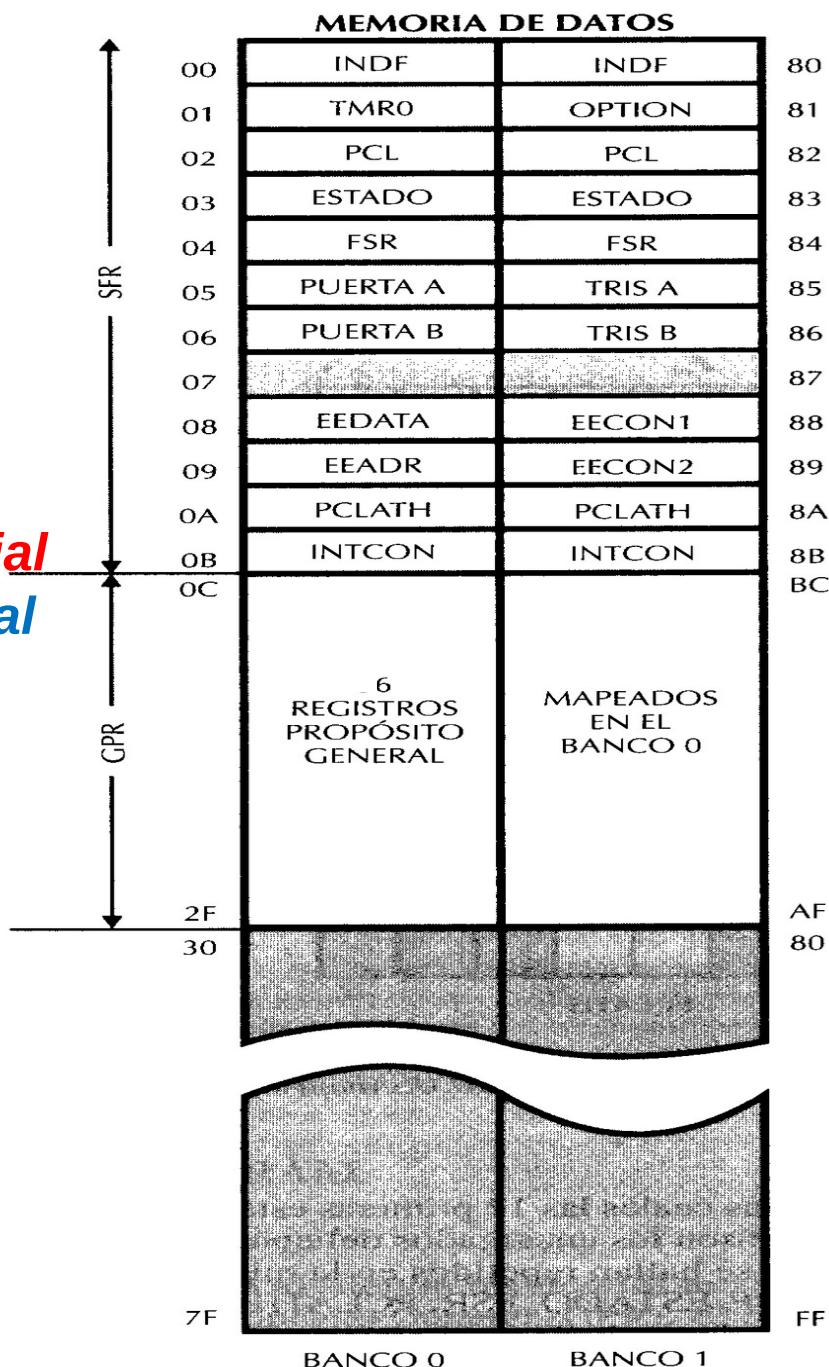
Localización de memoria no implementada, se lee como '0'
Nota 1: No es un registro físico



- Cada banco se divide a su vez en dos áreas:
- SFR (Registros de Funciones Especiales)
 - GPR (Registros de Propósito General)

Memoria de Datos

SFR: Registros de Propósito Especial
GPR: Registros de Propósito General



Registros de Trabajo

Organización de la Memoria de Datos

Banco 0

000	INDF
001	TMR0
002	PCL
003	STATUS
004	FSR
005	PORTA
006	PORTB
007	PORTC
008	PORTD
009	PORTE
00A	PCLATH
00B	INTCON
00C	PIR1
00D	PIR2

Banco 1

080	INDF
081	OPTION_REG
082	PCL
083	STATUS
084	FSR
085	TRISA
086	TRISB
087	TRISC
088	TRISD
089	TRISE
08A	PCLATH
08B	INTCON
08C	PIE1
08D	PIE2

Banco 2

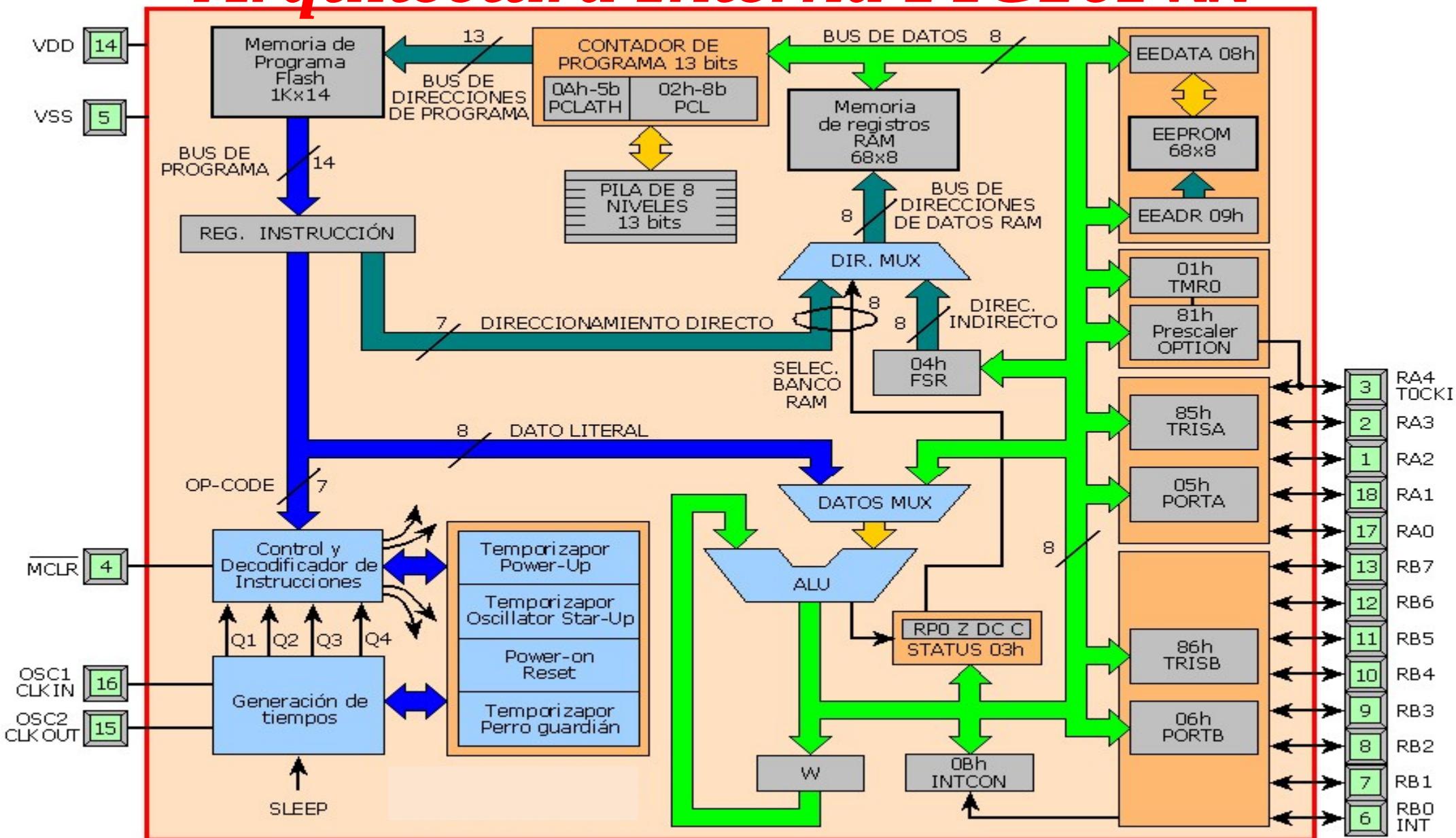
100	INDF
101	TMR0
102	PCL
103	STATUS
104	FSR
105	PORTB
106	PORTC
107	PORTD
108	PORTE
109	PCLATH
10A	INTCON
10B	EEDATA
10C	EEADR

Banco 3

180	INDF
181	OPTION_REG
182	PCL
183	STATUS
184	FSR
185	TRISB
186	TRISC
187	TRISD
188	TRISE
189	PCLATH
18A	INTCON
18B	EECON1
18C	EECON2

Device Specific Registers

Arquitectura Interna PIC16Fxx



Registro de STATUS

IRP	RP1	RP0	— TO	— PD	Z	DC	C
bit 7				bit 0			
R/W	R/W	R/W	R	R	R/W	R/W	R/W

IRP: Registro Selector de Bancos (usado para el Dir. Indirecto)

0 = Bank 0, 1 1 = Bank 2, 3

RP1:RP0: Bits Selectores de Bancos de Registros

— 00 = Bank 0, 01 = Bank 1, 10 = Bank 2, 11 = Bank 3

TO: bit Time-out

— 0 = ocurrió un WDT time-out

PD: bit Power-down

— 0 = ejecución de una instrucción SLEEP

Z: bit Cero

— 1 = El resultado de la operación aritmética es cero

DC: Digit carry / borrow bit

— 1 = Acarreo en el cuarto bit

C: Carry / borrow bit

— 1 = Acarreo en el Bit de Mayor Peso

Registro INDF (00h y 80h)

- El registro INDF (Indirect File) que ocupa la posición 00 no tiene existencia física, por lo que no se podrá acceder a él.
- En realidad este registro sirve únicamente para especificar la utilización del direccionamiento indirecto junto con el registro FSR.

Ver el direccionamiento indirecto en "El PIC16F84A".

Registro TMR0 (01h)

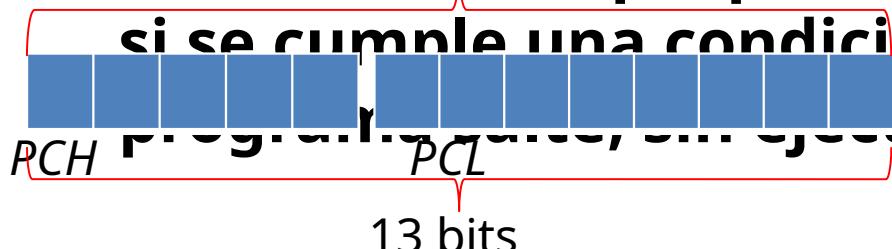
- El registro TMR0 (Timer 0) almacena el valor del contador TMR0, que está funcionando continuamente e incrementando el valor. Se incrementa en una unidad con cada impulso de reloj
- Las opciones que controlan este contador residen en el registro OPTION.
- Cada vez que llega al valor FF, vuelve a 00 generando una interrupción.
- El registro TMR0 se puede leer o escribir directamente con cualquier instrucción, con el fin de conocer su posición actual, o para inicializarlo en un estado determinado.
- Es importante saber que después de cualquier escritura en este registro, es necesario un retardo de dos ciclos de instrucción para que se retome la incrementación. Este retraso es independiente de la fuente de reloj usada. Las instrucciones que se utilizan son:

El Contador de programa PC

PCL(002h y 082h)

PCLATH (00Ah y 08Ah)

- Dos Registros por que uno solo no puede representar los 14 bits del PC
- Este registro contiene la dirección de la próxima instrucción a ejecutar.
- Se incrementa automáticamente al ejecutar cada instrucción
- Algunas instrucciones (que llamaremos de control) cambian el contenido del PC alterando la secuencia lineal de ejecución.
- Dentro de estas instrucciones se encuentran **GOTO** y **CALL** que permiten cargar en forma directa un valor constante en el PC
- Otras instrucciones de control son los **SKIP** o saltos condicionales, que producen un incremento adicional del PC si se cumple una condición específica, haciendo que al ejecutar la instrucción siguiente, el Registro PCL se actualice con el valor del Registro PCLATH.



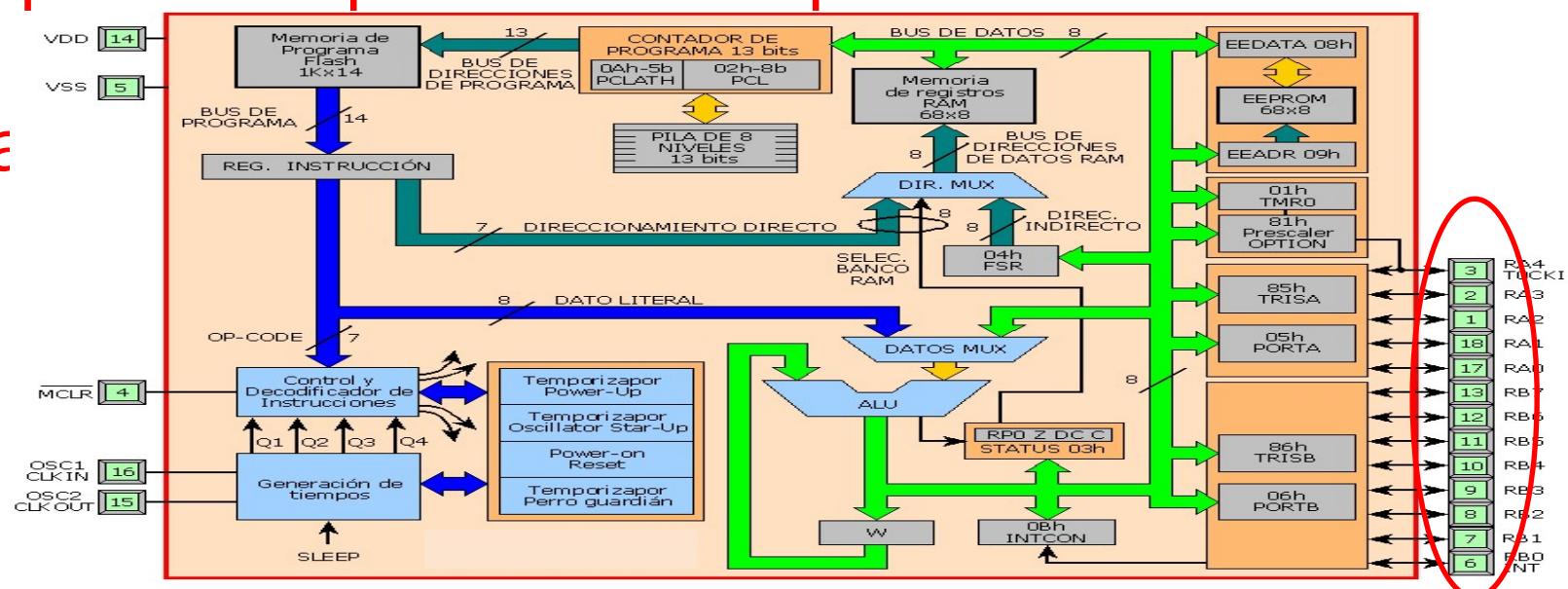
Registro FSR (04h y 84h)

- El contenido del FSR se utiliza para el direccionamiento indirecto junto con el registro INDF. Este registro contiene 8 bits. Ver el direccionamiento indirecto en "El PIC16F84A".

Registro TRISA y TRISB (85h y 86h)

- Estos registros son idénticos para el puerto A y el puerto B, con la diferencia de que uno será de 5 bits y otro de 8 bits, el mismo número de bits que tiene cada puerto.
- Los registros TRIS, también son llamados así, sirven para configurar si los bits de cada puerto serán de entrada o de salida.

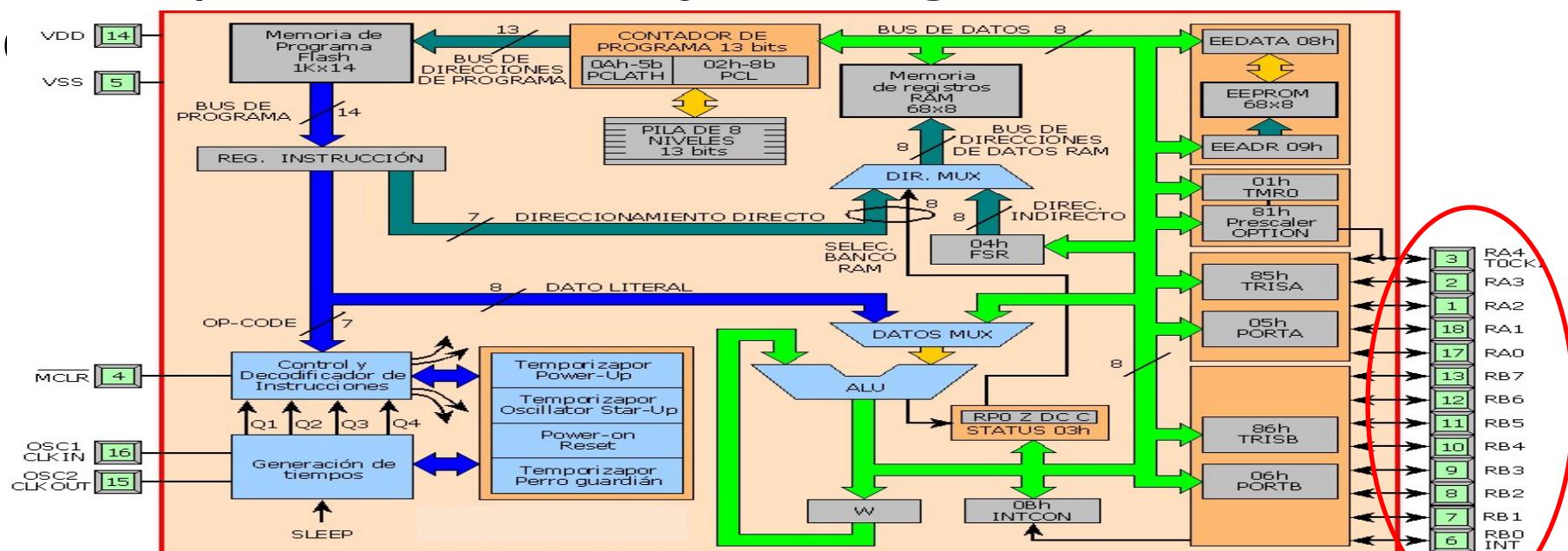
1: Los pines del puerto correspondiente serán de entrada
0: Los pines del puerto correspondiente serán de salida



Clase 5

Registro PORTA y PORTB (05h y 06h)

- Estos registros contienen los niveles lógicos (0/1) de los pines de E/S.
- Cada bit se puede leer o escribir según el pin correspondiente se haya configurado como entrada o



bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Puerto B

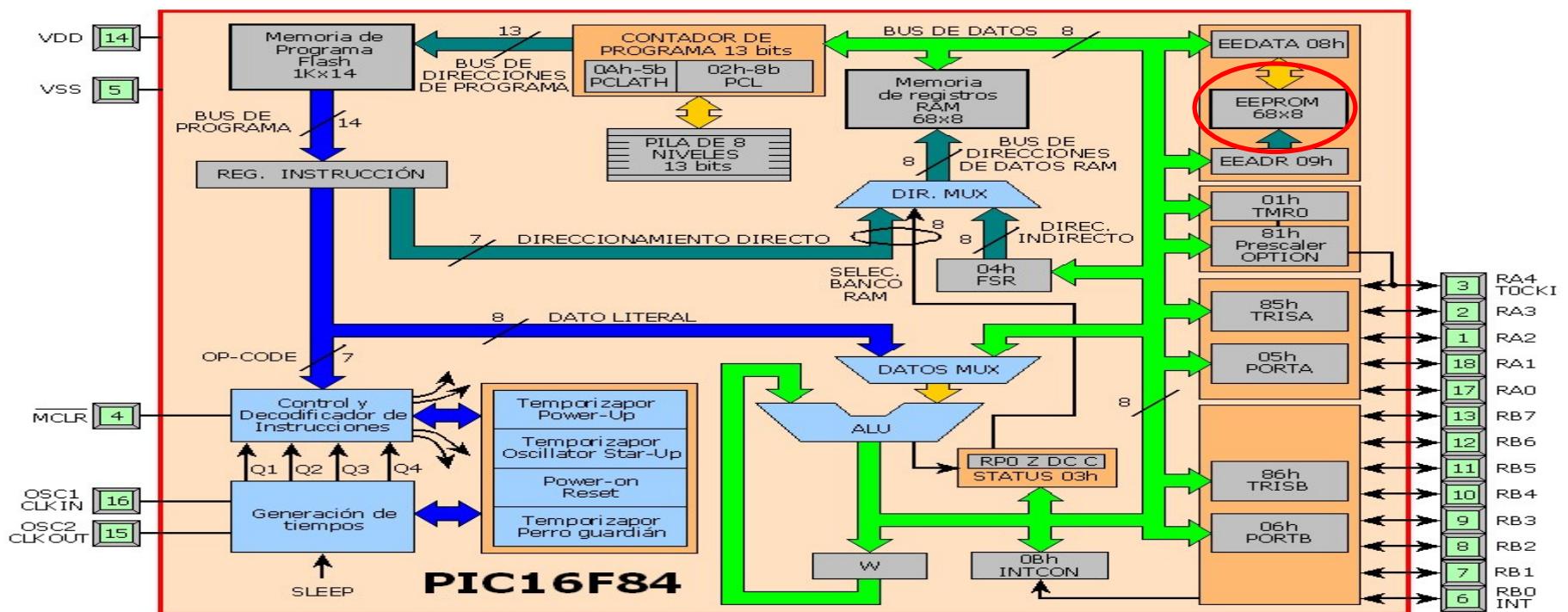
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
X	X	X	RA4	RA3	RA2	RA1	RA0

Puerto A

Registro EEDATA (08h)

Registro EEADR (09h)

- El registro **EEDATA** (Datos de EEPROM) guarda el contenido de una posición de la memoria EEPROM de datos antes de su escritura o después de su lectura, según leamos o escribamos en ella.
- El registro **EEADR** (Dirección de EEPROM) guarda la dirección de la posición de memoria EEPROM cuando queramos acceder a ella.



Registro EECON1 (88h)

Este registro contiene configuraciones importantes acerca de la escritura y la lectura de la EEPROM de datos.

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
-	-	-	EEIF	WRERR	WREN	WR	RD
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

U (Unimplemented), No implementado. Se lee como 0.

Bit 4 (flag): EEIF. Bit de interrupción de escritura en la memoria EEPROM (EEPROM Interrupt Flag)

Bit 3 (flag), WRERR. Bit de error de escritura (Write Error)

Bit 2, WREN. Bit de habilitación de escritura. (Write Enable)

Bit 1, WR. Bit de control de escritura (Write Data)

Bit 0, RD. Bit de control de lectura (Read Data)

Registro EECON2 (89h)

e registro no está implementado físicamente, por lo cual no se puede leer.
sólo sirve para un proceso de protección de escritura que consiste en
añadir en él unos datos específicos, con el fin de evitar que un programa p
or pueda programar la EEPROM, manipulando simplemente los bits
EECON1.

Registro INTCON (0Bh y 8Bh)

- Este registro contiene los bits de selección de fuentes de interrupción, el bit de activación global de interrupciones y varios flag que indican la causa de una interrupción.
- Se utiliza para el control global de las interrupciones y para indicar la procedencia de algunas de ellas.
- Hay cuatro potenciales recursos de interrupción:
 - Una fuente externa a través del pin RB0/INT.
 - El desbordamiento del temporizador 0
 - Un cambio de estado en los pines RB4
 - Programación de la EEPROM de datos.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7, GIE: Habilitación global de interrupciones.

Bit 6, EEIE: Habilitación de las Interrupciones de la memoria EEPROM.

Bit 5, TOIE: Habilitación de la interrupción del temporizador por desbordamiento (Timer 0 Interrupt Enable)

Bit 4, INTE: Habilitación de la entrada de interrupción externa (Interrupt Enable) por patilla RB0/INT.

Bit 3, RBIE: Habilitación de las interrupciones del puerto B.

Bit 2 (flag), TOIF: Bit de interrupción de desbordamiento del TMRO.

Registro OPTION (80h)

(1)

- El registro OPTION (o registro de opciones) se emplea para programar las opciones del temporizador TMR0, el tipo de flanco con el que se detecta una interrupción y la activación de las resistencias de polarización del puerto B.
- Ocupa la posición 81h de la página 1 del banco de registros. Debe escribirse usando la instrucción especial OPTION. Esta instrucción carga el contenido de W en el registro OPTION.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Registro OPTION (80h)

(1)

Bit 7, /RBPU (RB Pull Up). Conexión de las resistencias de polarización del Puerto B. Se conectan todas cuando el puerto B actua como entrada.

1: Todas las resistencias son desconectadas.

0: Las resistencias se activan de forma individual.

Bit 6, INTDEG (INTerrupt EDGe). Selecciona el tipo de flanco para la interrupción externa. Este bit indica el tipo de flanco de la señal externa que ha de provocar una interrupción en la patilla RB0/INT.

1: La interrupción es producida por el flanco ascendente o de subida.

0: La interrupción es producida por el flanco descendente o de bajada.

Bit 5, T0CS (Timer 0 Signal Source). Selección de la fuente de reloj para el TMR0.

1: TMR0 se usa en modo contador de los pulsos introducidos a través de RA4/T0CKI

0: TMR0 se usa en modo temporizador haciendo uso de los pulsos de reloj internos (Fosc/4).

Bit 4, T0SE (Timer 0 Signal Edge). Tipo de flanco activo de T0CKI (patilla RA4/T0CKI).

1 = El TMR0 se incrementa con el flanco descendente de la señal aplicada a RA4/T0CK1.

0 = El TMR0 se incrementa con el flanco ascendente.

Bit 3, PSA (PreScaler Assignment). Se usa para la asignación del divisor de frecuencias o Prescaler.

1 = El divisor de frecuencia se asigna al WDT.

0 = El divisor de frecuencia se asigna a TMR0.

Registro OPTION (80h) (2)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

, /RBPU (RB Pull Up). Conexión de las resistencias de polarización del Puerto B. Se conectan todas cuando el puerto B actua como entrada.

, INTDEG (INTerrupt EDGe). Selecciona el tipo de flanco para la interrupción externa.

bit indica el tipo de flanco de la señal externa que ha de provocar una interrupción en la patilla RB0/INT.

, T0CS (Timer 0 Signal Source). Selección de la fuente de reloj para el TMR0.

, T0SE (Timer 0 Signal Edge). Tipo de flanco activo de T0CKI (patilla RA4/T0CKI).

, PSA (PreScaler Assignment). Se usa para la asignación del divisor de frecuencias o Prescaler.

, 0, 1 y 2, PS0, PS1 y PS2 (Prescaler Rate Select Bits). Configura la tasa del valor del divisor de frecuencia del prescaler. Difiere dependiendo que se haya asignado al TMR0 o al WDT.

PS2	PS1	PS0	Divisor TMR0	Divisor WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Cuestionario

(20 minutos)

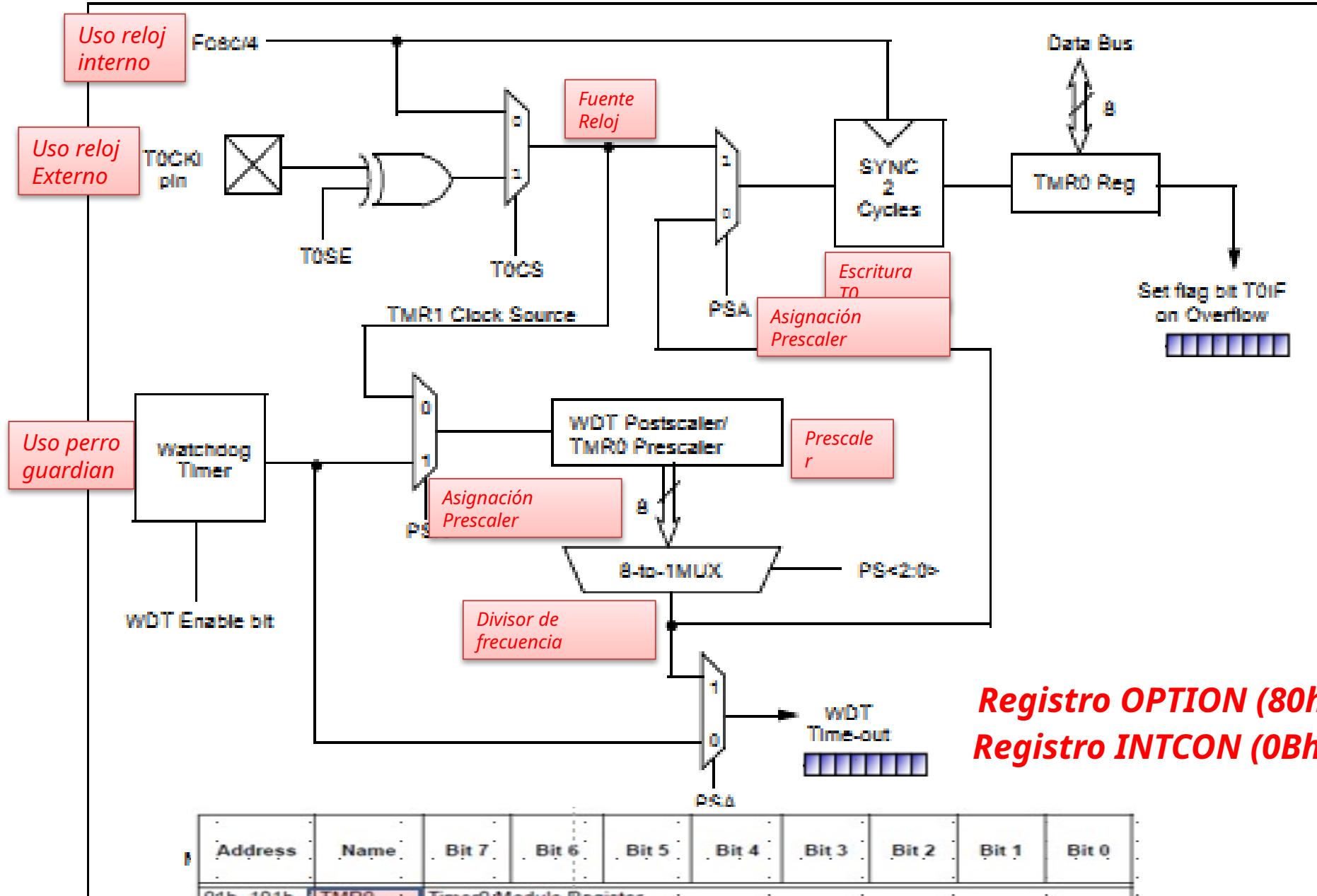
- *Mencione los registros trabajo y describa su utilización.*
- *Brevemente describa*
 - *El uso del registro W*
 - *El uso del registro Status*

Clase 6

Timer

Calculo de Eventos / Tiempos

FIGURE 6-1: BLOCK DIAGRAM OF THE TIMER0/WDT



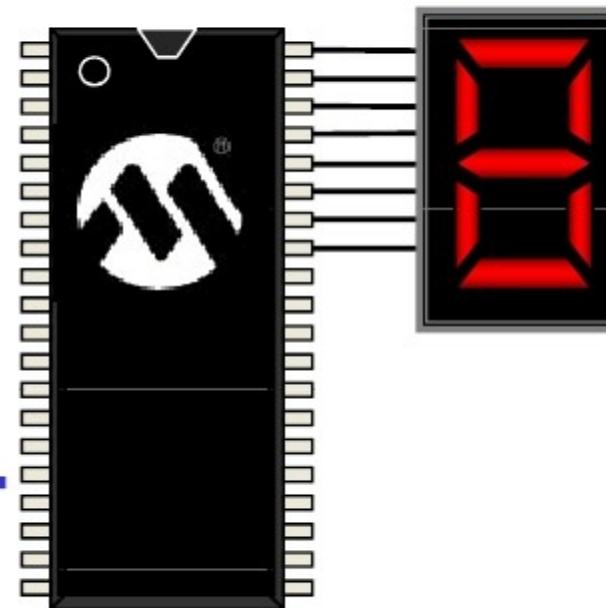
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h, 101h	TMR0								
	Timer0 Module Register								
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81h, 181h	OPTION ⁽²⁾	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Timers 0

Timers son usados para:

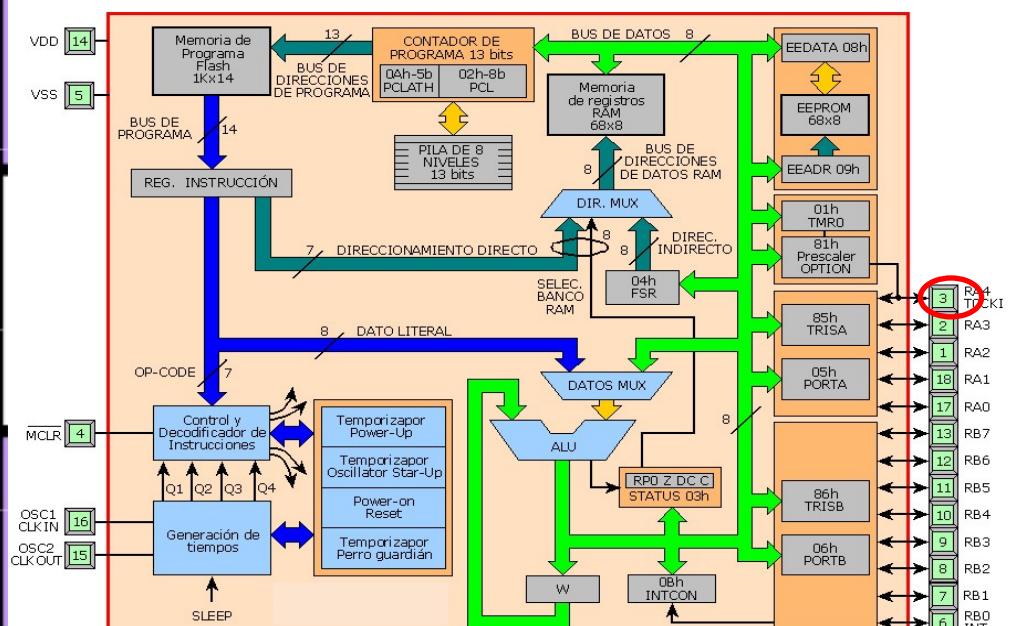
- tiempo de referencia para eventos
- contar el número de eventos
- generación de formas de onda etc...

- **Timer0** *Vamos a utilizarlo en nuestro curso*
- *Timer1*
- *Timer2*

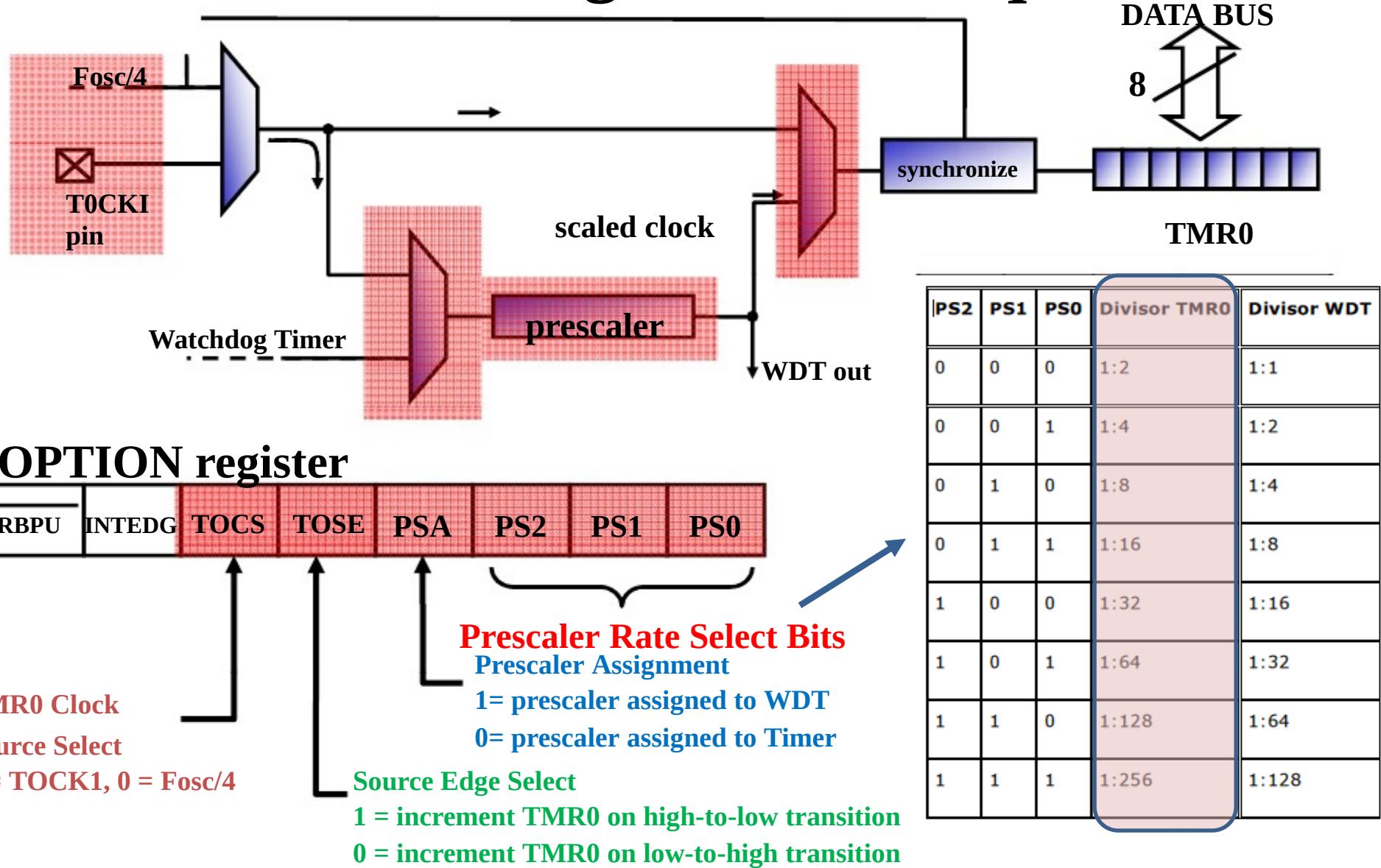


Características del Timer 0

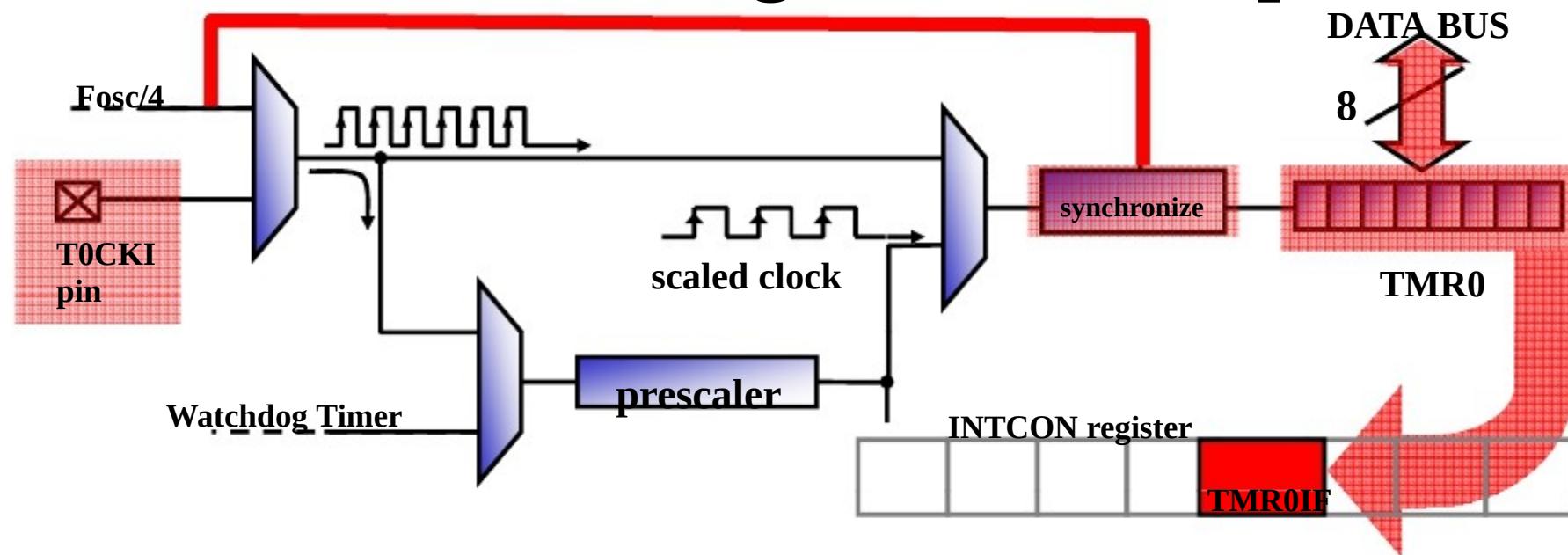
	SIZE OF REGISTER	8-bits (TMR0)
CLOCK SOURCE (Internal)	Fosc/4	
CLOCK SOURCE (External)	T0CKI pin	
CLOCK SCALING AVAILABLE (Resolution)	Prescaler 8-bits (1:2 1:256)	
INTERRUPT EVENT and FLAG LOCATION	On overflow FFh 00h (TMR0IF in INTCON)	
CAN WAKE PIC FROM SLEEP?	NO	



Timer 0 Diagrama en Bloques



Timer 0 Diagrama en Bloques



- Si la fuente de clock es externa puede sincronizarse (TOCKI) para el clock interno
- Timer 0 es de lectura y escritura
- Timer 0 el flag es seteado sobre un desborde del TMR0 (FF to 00)

Timer0 Inicialización

;Make sure the Timer0 count
;register (TMR0) is clear

```
banksel    TMR0  
        dcf      TMR0
```

;Clear Timer 0 interrupt flag
bcf INTCON,TMR0IF

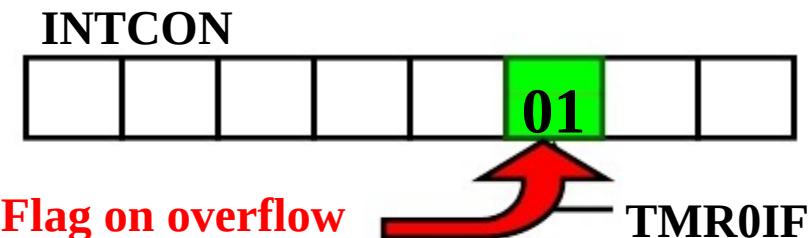
;Setup the Option register to
;increment Timer0 from internal
;clock with a prescaler of 1:16

```
banksel    OPTION_REG  
        movlw    b'00000011'  
        movwf    OPTION_REG
```

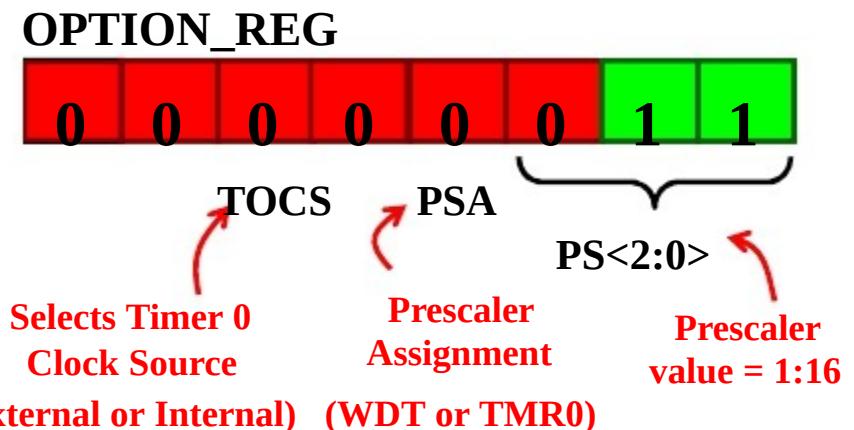
;The TMR0 interrupt is disabled, do
;polling on the flag bit (TMR0IF)

```
btfs      INTCON,TMR0IF  
        goto    $-1
```

<continue>



This interrupt flag will set on
Timer0 overflow even if
interrupts are disabled



Uso del Timer 0 para la familia PIC16Fxxx

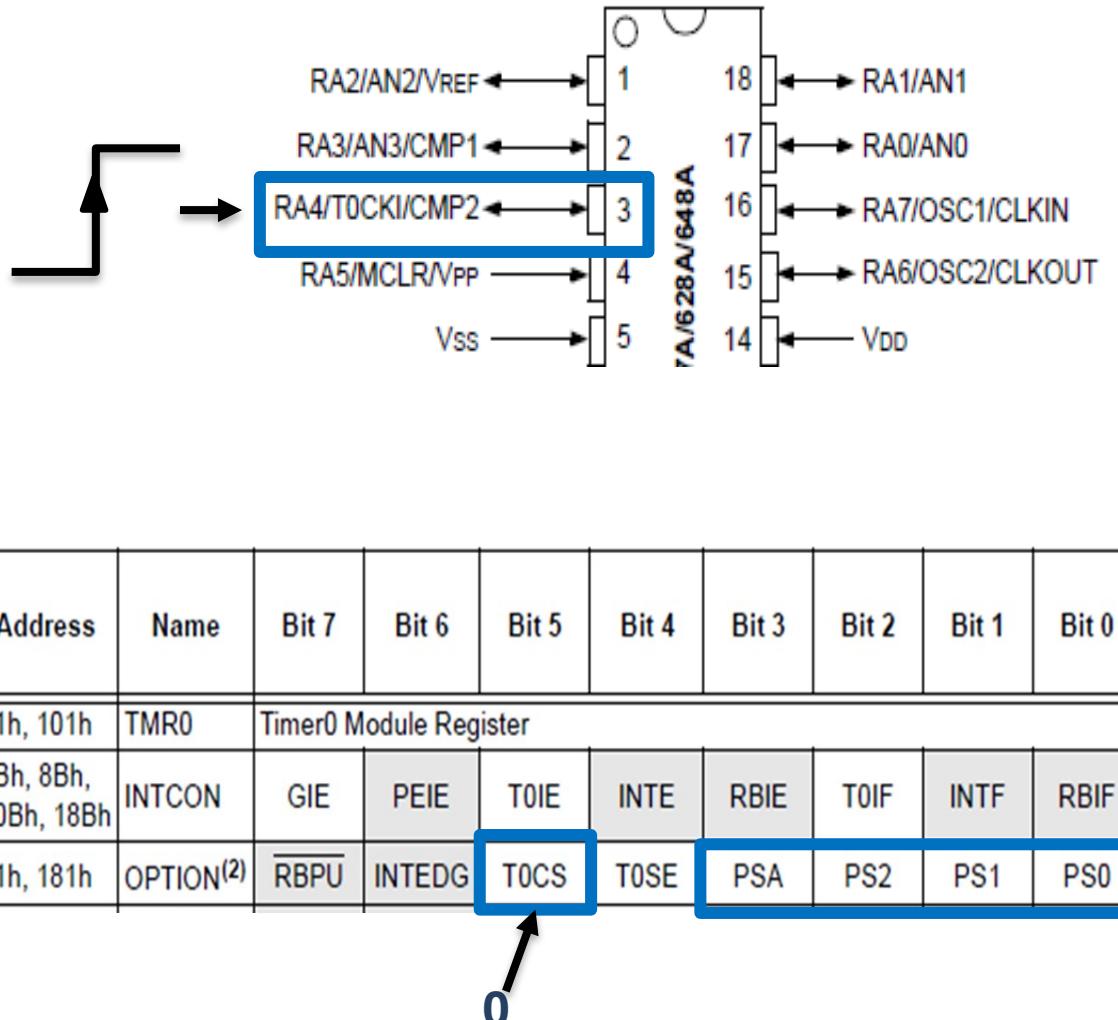
Registros que se utilizan

- TMR0: Registro que almacena el valor del contador.
- INTCON: Registro utilizado para habilitar las interrupciones del Timer.
- OPTION: Registro de configuración del preescaler, entre otras cosas que veremos a continuación.

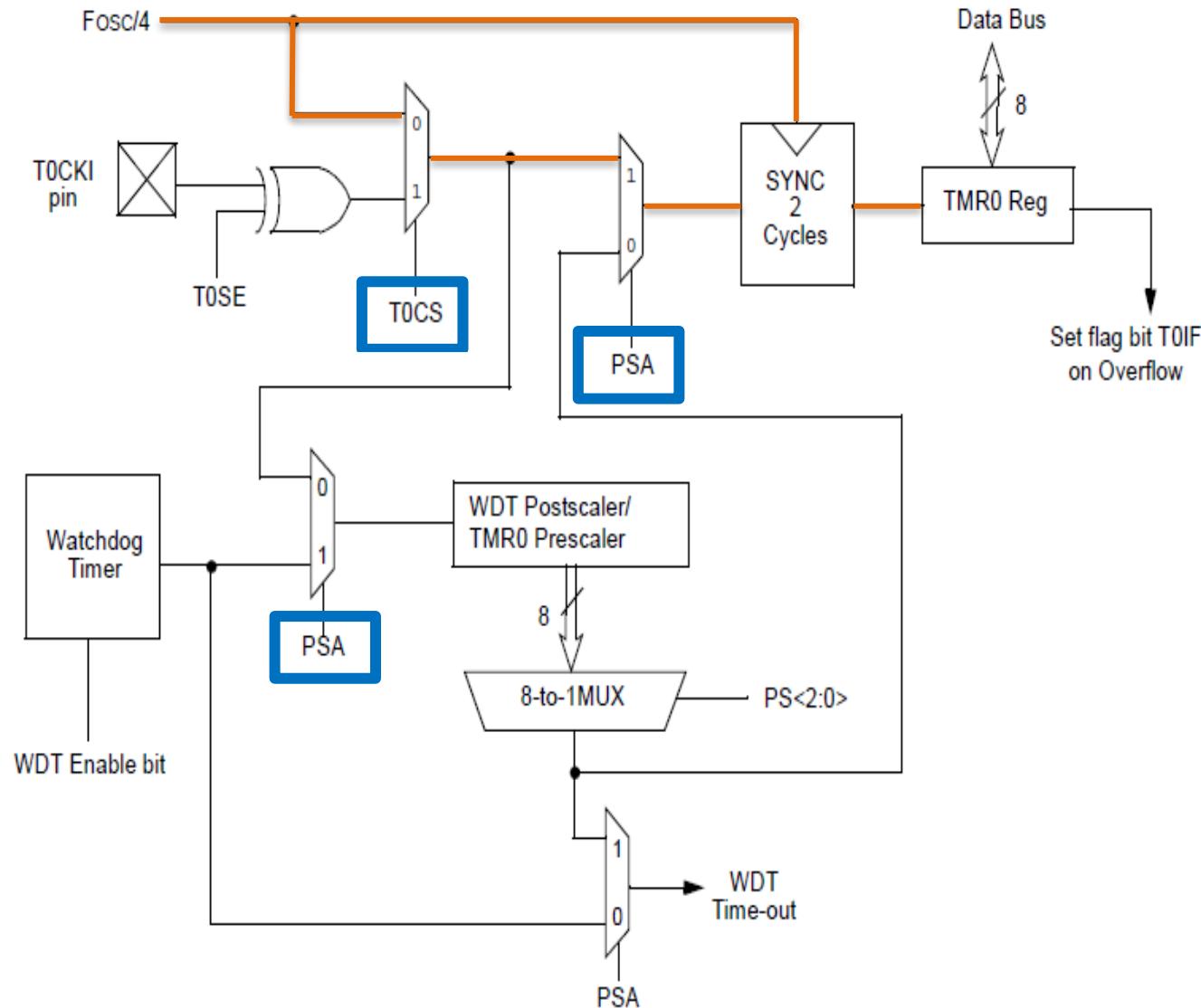
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h, 101h	TMR0	Timer0 Module Register							
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
81h, 181h	OPTION ⁽²⁾	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Timer 0 como Contador de Eventos

- El registro TMR0 se incrementa en 1 cada vez que ingresa una señal en RA4.
- Para utilizarlo debe escribirse un 0 lógico en el bit TOCS del registro OPTION.
- Puede utilizarse el preescaler para dividir la frecuencia de conteo.



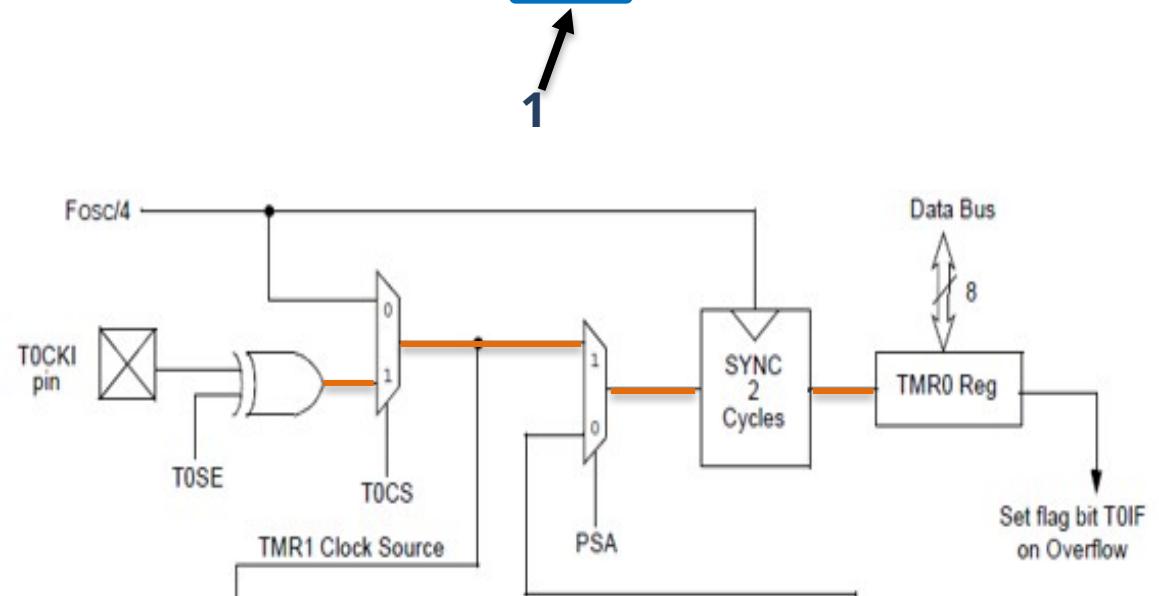
Timer 0 como Contador de Tiempos



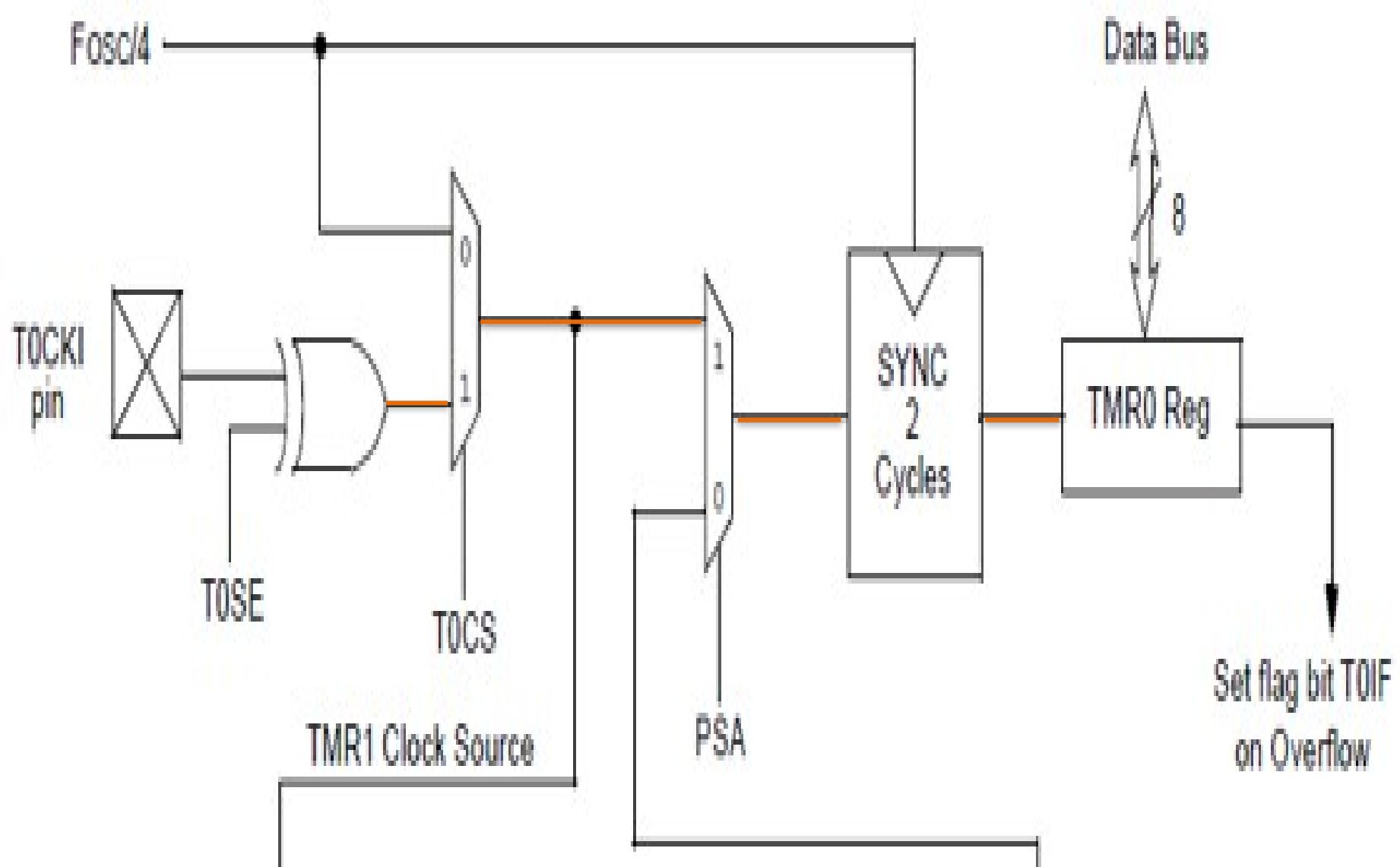
Timer 0 como Contador de Eventos

- El registro TMR0 se incrementa en 1 cada vez que hay un ciclo de máquina.
- Un ciclo de máquina equivale a 4 ciclos de reloj.
- Para utilizarlo debe escribirse un 1 lógico en el bit TOCS del registro OPTION.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h, 101h	TMR0	Timer0 Module Register							
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
81h, 181h	OPTION ⁽²⁾	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0



Timer 0 como Temporizador



Timer 0 como Temporizador

¿Cómo se calcula el valor del registro TMR0 según el tiempo "T" que quiero?

1. Sabemos que un ciclo de reloj es de 4MHz ($0,25\mu s$), por lo tanto un ciclo de máquina será de 1MHz o $1\mu s$.
2. En base a esto, sin activar el preescaler, la fórmula sencillamente sería la siguiente:

Entonces, sin usar el preescaler, ¿cuál es el tiempo máximo que podemos calcular?

Referencias unidades de tiempo

μs = microsegundo, es igual a la millonésima parte de un segundo.

ms = milisegundo= es igual a la milésima parte de un segundo.

Timer 0 como Temporizador

Entonces, sin usar el preescaler, ¿cuál es el tiempo máximo que podemos calcular?

Si se utiliza el preescaler (PSA=0), se pueden lograr demoras más grandes y generalmente más útiles como por ejemplo para el multiplexado de displays (Sí, lo van a tener que hacer en algún momento).

- Usando los bits PSA, PS0, PS1 y PS2 la fórmula se modifica de la siguiente manera:

Tiempo

¿Y ahora hasta qué valor puede llegar el Tiempo máximo?

Timer 0 como Temporizador

- Ejemplo 1: Identificar qué bits y registros se tienen que usar para generar un delay de 20ms y definir sus valores.

Pasos a seguir:

1. ¿Hay que usar preescaler?
2. En caso de que se requiera, identificar el valor del preescaler.
3. Despejar el valor de TMR0.
4. Identificar los valores de los registros a usar.

Timer 0 como Temporizador

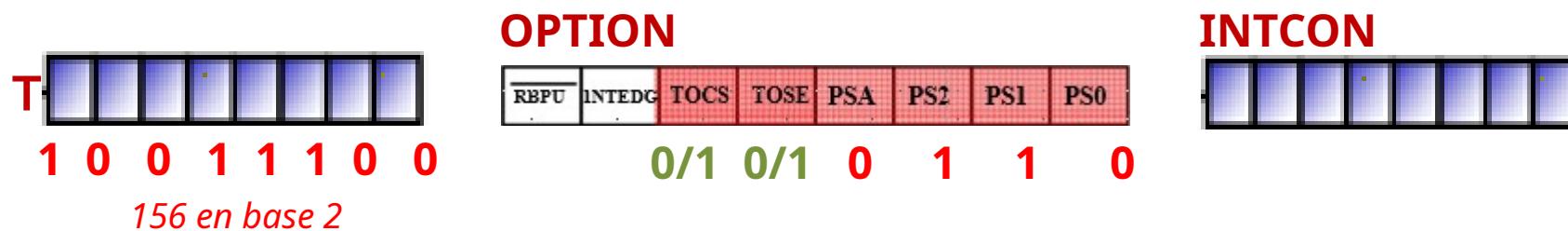
Para calcular un tiempo 20 ms:

1. Sí, se debe usar preescaler. ¿Por qué?
2. Mínimo se requerirá usarlo en 1:128. ¿Por qué?
3. De la fórmula presentada anteriormente, tomando a "T" en μs , se despeja:

Para nuestro ejemplo:

$$TMRO = 256 - \text{Tiempo} \quad 256 - 100 = 156^{(10)}$$

4. ¿Qué valores deben tomar los registros del controlador?



Timer 0 como Temporizador

Ejercitación:

1. Resolver el ejercicio de ejemplo con un preescaler de 1:256.
2. Realizar una demora de 50ms. Completar todos los pasos vistos en el ejemplo.
3. Idem ejercicio anterior con una demora de 1ms.
4. Sabiendo que el tiempo máximo que se puede tomar es de 65ms, ¿como se podría generar una demora de 1 segundo utilizando el Timer 0?

Cuestionario

(20 minutos)

- *Describa el uso del Timer 0*
- *Mencione los registros trabajo que se utilizan para programar el Timer 0.*
- *Genere un tiempo de 35 miliseg. (Registros **Option TMR0** y **INTCON**)*

Clase 9

Tipos de Direccionamiento

PIC16 Modos de Direccionamiento

Accesos a Memoria de Datos:

- **Directo** addwf <dirección del dato>, <d>
- **Indirecto** addwf INDF, <d>
- **Inmediato (Literal)** movlw <constante>

Accesos a Memoria de Programa:

- **Absoluto** goto <Dir. Mem. programa>
- **Relativo** addwf PCL, f

Direccionamiento Directo

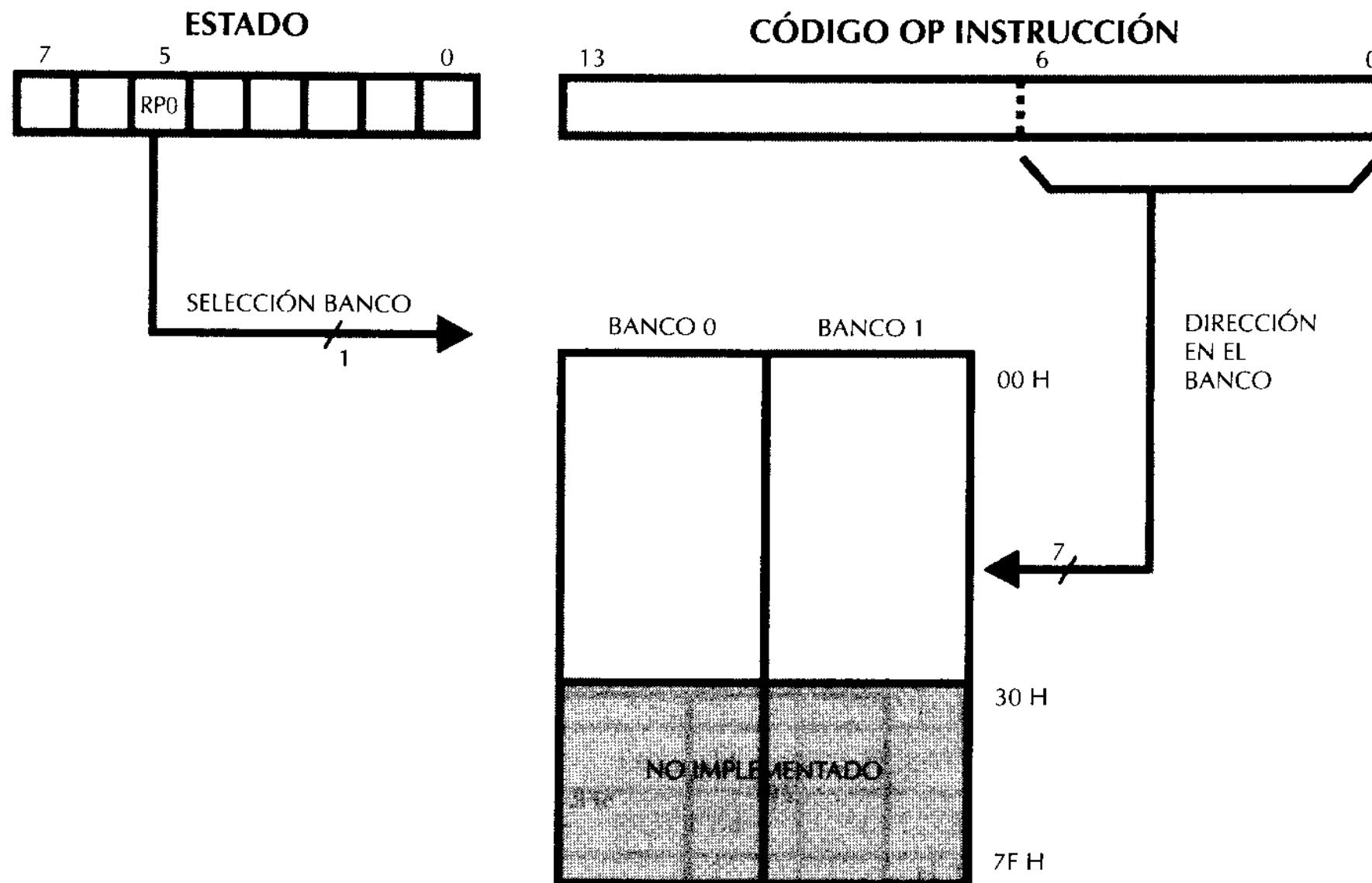


Figura 6.10. Direccionamiento directo en el PIC16C(F)84.

Direccionamiento Directo

Ejemplo: Inicializar bits 0~3 como salida en el PORTB

Registro W :

F0

9-Bit Dirección Efectiva:

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

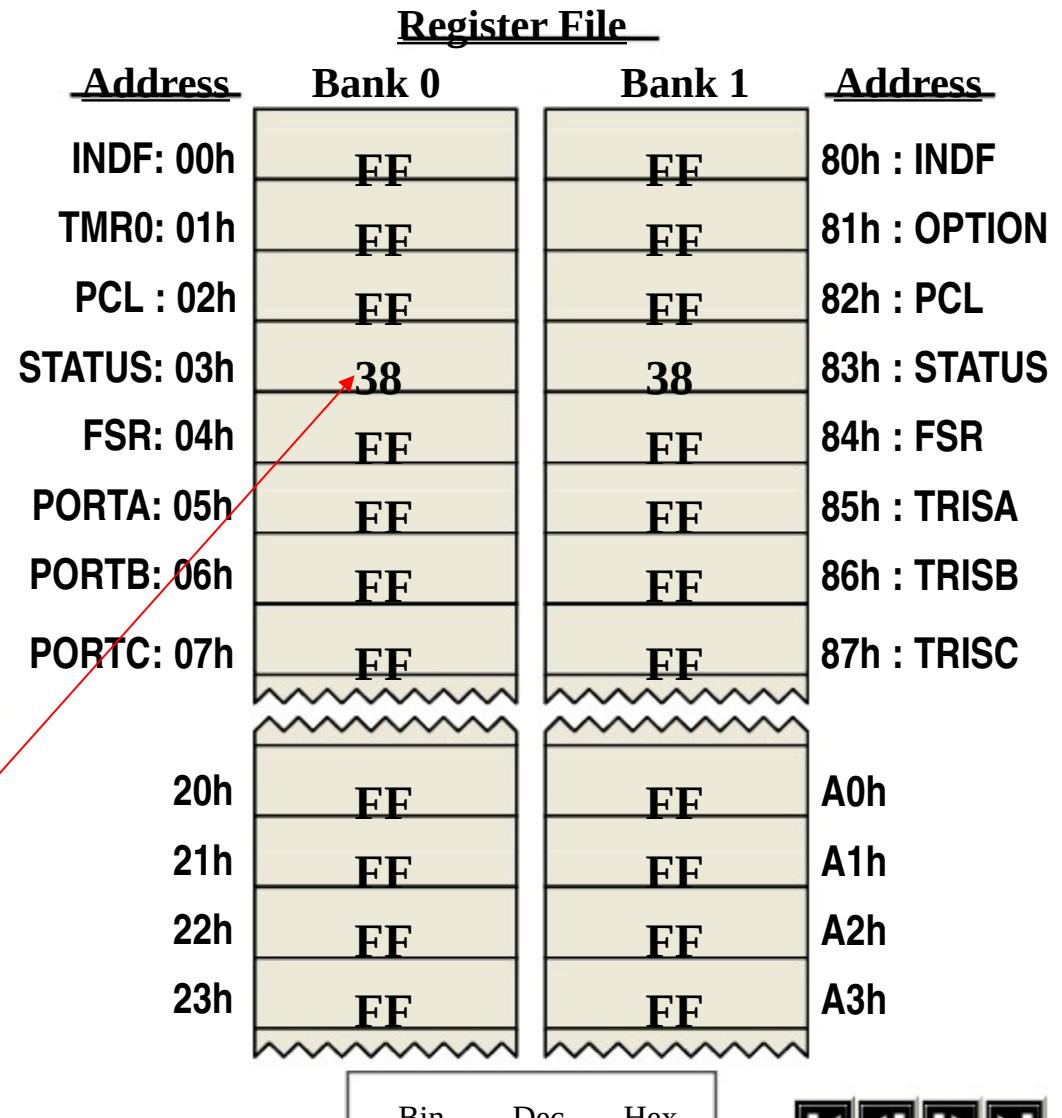
RP1RP0

7-bits desde la Instrucción

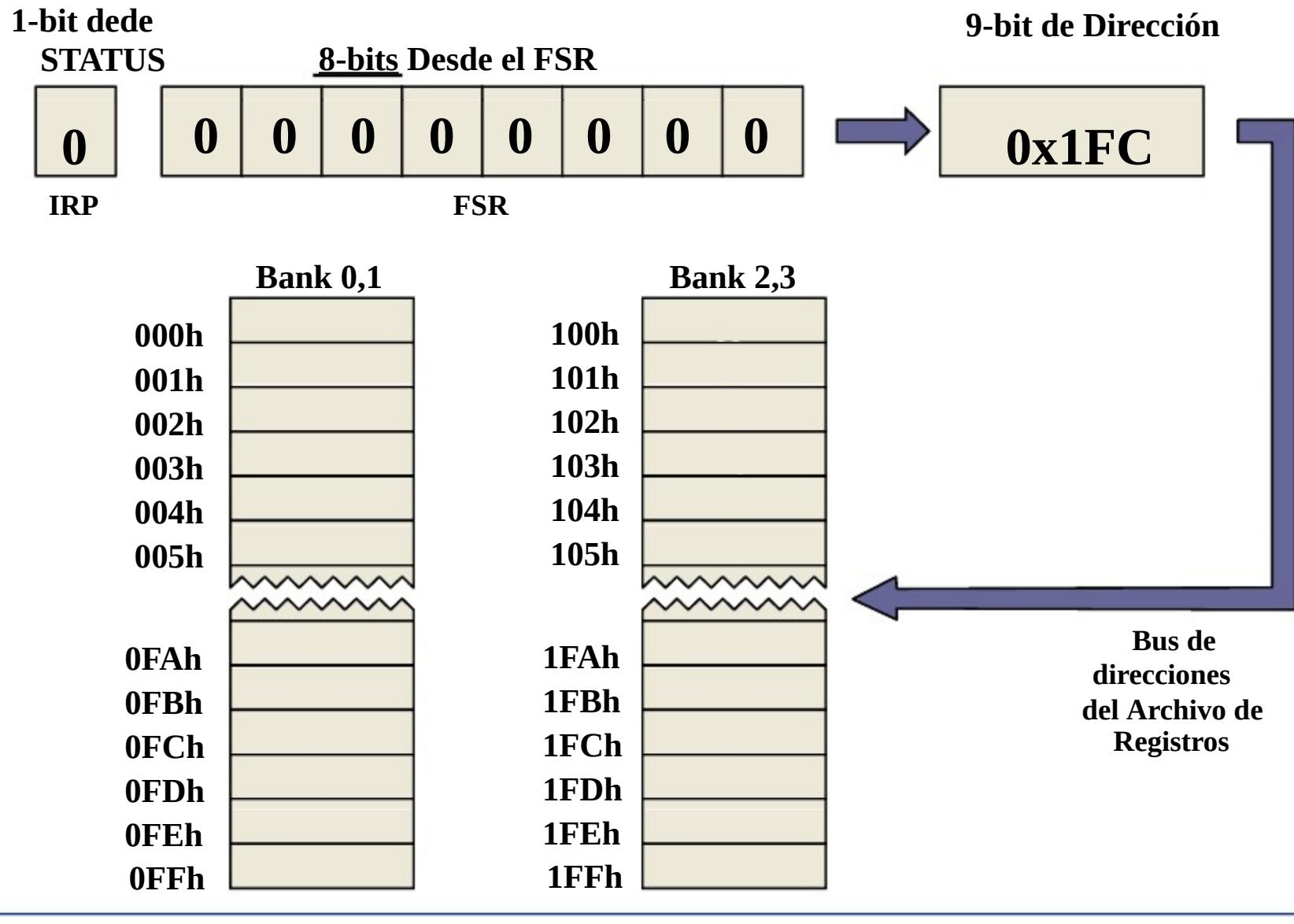
```

bsf    STATUS, RP0
movlw b'11110000'
movwf TRISB
bcf    STATUS, RP0
clrf    PORTB

```



Direccionamiento Indirecto

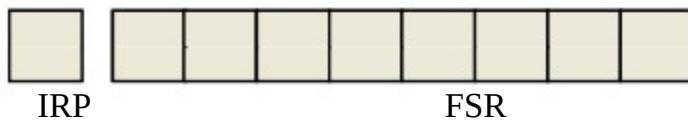


Direccionamiento Indirecto

W Register:

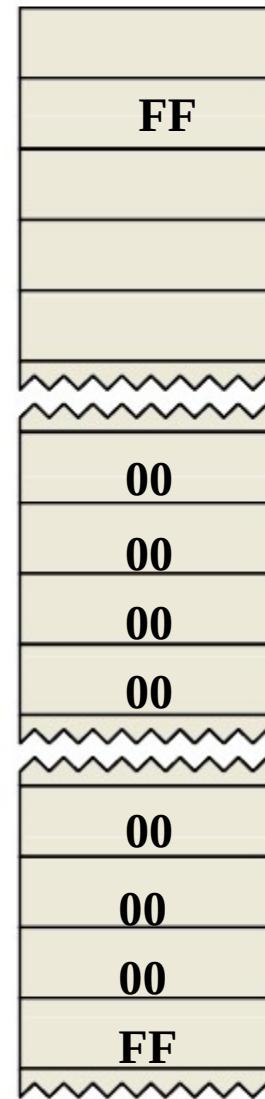


9-Bit Effective Address:



	bcf	STATUS, IRP
	movlw	0x20
	movwf	FSR
LOOP	clrf	INDF
	incf	FSR, f
	btfss	FSR, 7
	goto	LOOP
	<next instruction>	

Register File Address



Direccionamiento Absoluto

Instrucciones CALL and GOTO :

	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PC Direccionamiento Absoluto (Program Memory)

- saltar a otra localización de memoria de programa fuera de la secuencia del PC
- Llamar a una Subrutina

Usado por las instrucciones CALL y GOTO

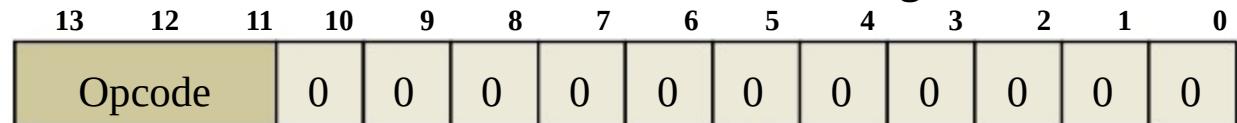
- 11-bits de los 13 bits requeridos están codificados en la instrucción
- 2 bits adicionales son aportados por el registro PCLATH

Usado cuando se realiza un Salto computado

- La dirección del salto es calculada por el programa
- La dirección computada es escrita directamente dentro del PC

Direccionamiento Absoluto

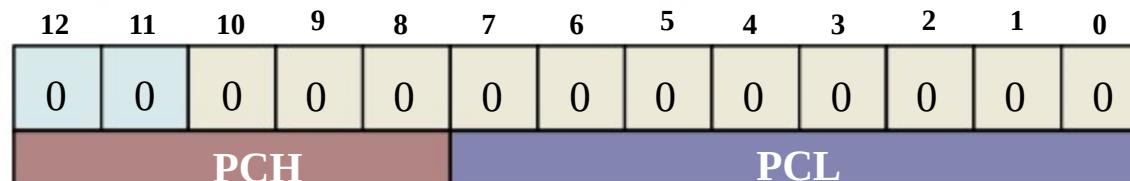
Instrucción de 14-Bit CALL o GOTO en Memoria de Programa



Registro PCLATH en Memoria de Datos



2-Bits desde el PCLATH



Contador de Programa de 13-Bit

Direccionamiento Absoluto

Registro PCLATH									
7	6	5	4	3	2	1	0		
-	-	-	0	0	0	0	0		

Instrucción CALL en Memoria de Programa															
13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Opcode	0	0	0	0	0	0	0	0	0	0	0	0	0		

Registro W
FF

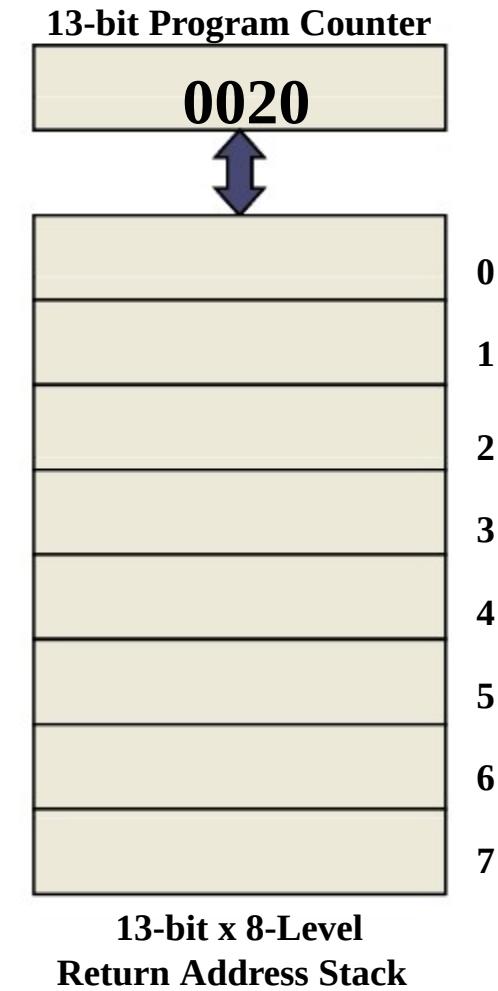
Contador de Programa - PCH:PCL
- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
org 0x0020
movlw      HIGH MiSubrutina
movwf      PCLATH
call       MiSubrotina
...
org 0x1250
MiSubrutina <aquí comienza la subrutina>
...
return
```



CALL / RETURN Stack

0020		movlw HIGH MySub1
0021		movwf PCLATH
0022		call MySub1
0023		call MySub4
0024		bsf PORTB, 7
...		...
1000	MySub1	bsf PORTB, 0
1001		call MySub2
1002		return
1003	MySub2	bsf PORTB, 1
1004		call MySub3
1005		return
1006	MySub3	bsf PORTB, 2
1007		return
1008	MySub4	bsf PORTB, 3
1009		call MySub2
100A		return

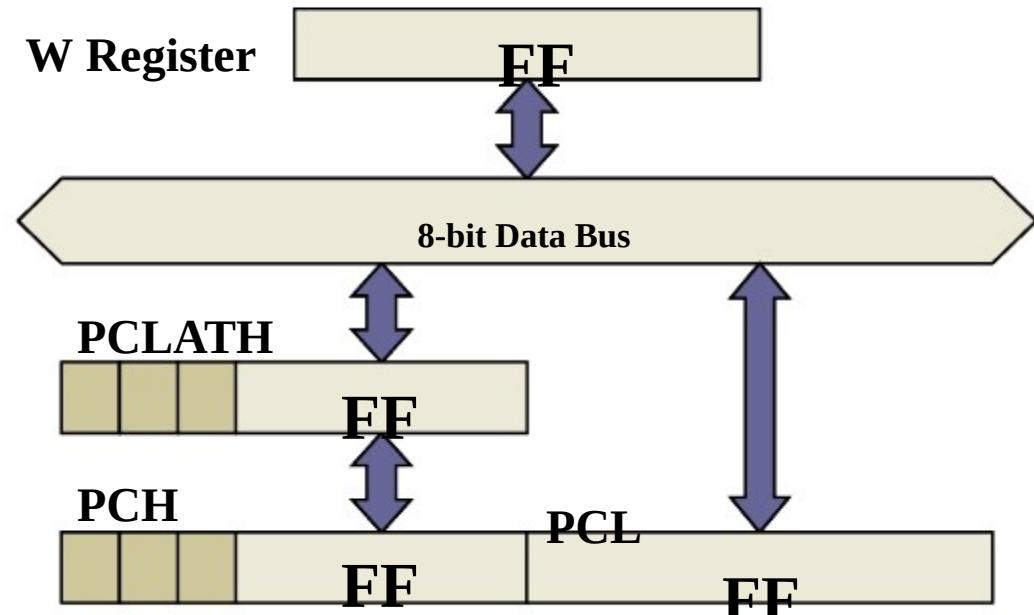


Direccionamiento relativo

Para escribir al PC:

Escribir byte alto en el PCLATH

Escribir byte bajo en el PCL
(PCH siempre debe ser cargado con el PCLATH)



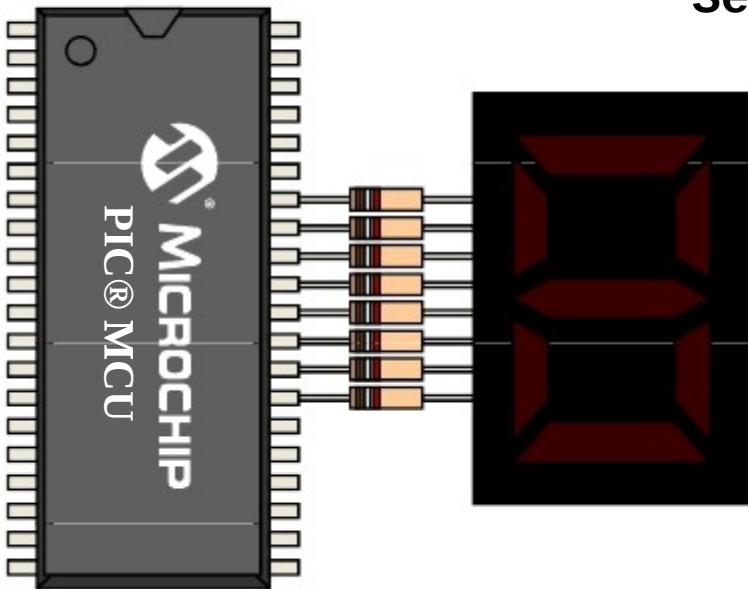
movlw	HIGH 0x1250
movwf	PCLATH
movlw	LOW 0x1250
movwf	PCL



Direccionamiento relativo:

Tablas de lectura

Ejemplo: Uso de Tablas de lectura para decodificación de BCD a 7-segmentos para excitar un display a LED



```
ORG      0x0020 ; Page 0
movlw    HIGH SevenSegDecode
movwf    PCLATH
movlw    .5
call     SevenSegDecode
movwf    PORTB
...
ORG      0x1800 ; Page 3
```

SevenSegDecode:

```
addwf   PCL,f
retlw   b'00111111' ;0
retlw   b'00000110' ;1
retlw   b'01011011' ;2
retlw   b'01001111' ;3
retlw   b'01100110' ;4
retlw   b'01101101' ;5
retlw   b'01111101' ;6
retlw   b'00000111' ;7
retlw   b'01111111' ;8
retlw   b'01101111' ;9
```



Clase 8 a 19

PROGRAMACIÓN

Programa Ensamblador

El programa ensamblador es un software que se encarga de traducir los nemonicos y símbolos alfanuméricos del programa escrito en ensamblador por el usuario a código maquina.

El programa escrito en lenguaje ensamblador recibe el nombre de archivo fuente. Suele tener la extensión *.asm.****

El programa ensamblador proporciona un archivo en lenguaje maquina que suele tener la extensión *.hex.****

Programa Ensamblador

El ensamblador mas utilizado para los PIC's es el MPASM que trabaja dentro del entorno del software llamado MPLAB que se describirá mas adelante.

- Disponible en www.microchip.com

Archivos Resultante del Ensamblado

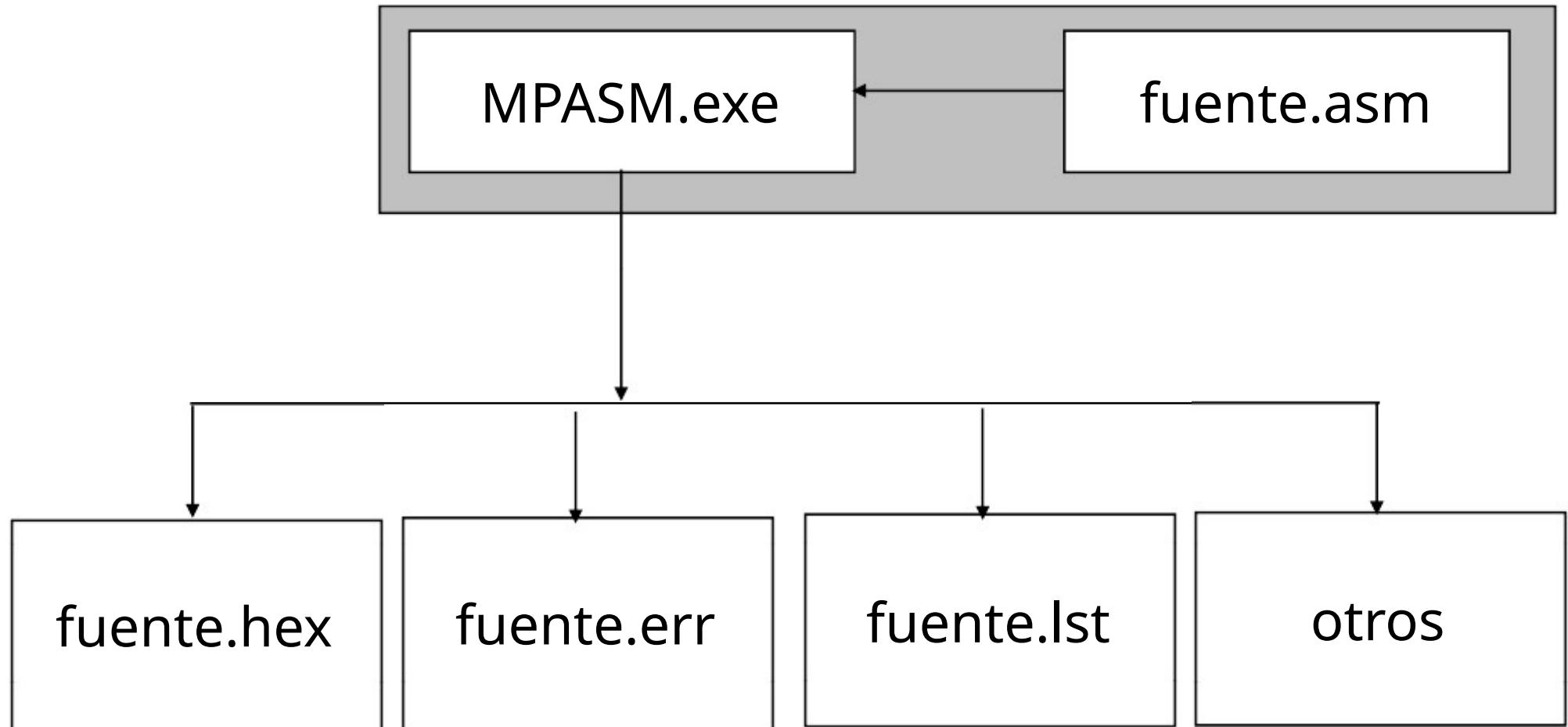
Tras el ensamblado del archivo *.asm se producen varios archivos.

Archivo ejecutable o hexadecimal. Es un archivo con la extensión *.hex . Contiene los códigos maquina del programa que servirán para grabar la memoria de programa del microcontrolador

Archivo de errores. Es un archivo con la extension *.err . Contiene los errores producidos durante el proceso de ensamblado.

Archivo de reporte. Es un archivo con la extensión *.lst . Contiene toda la información del programa: codigo fuente, códigos maquina, direcciones de cada instrucción, errores producidos, etc.

Archivos Resultante del Ensamblado



Código Fuente

El código fuente esta compuesto por una sucesión de líneas de programa.

Todos los archivos fuentes poseen una estructura similar independientemente del procesador utilizado.

- Campo de etiquetas**
- Campo de códigos de operación**
- Campo de operándos y datos**
- Campo de comentarios**

Código Fuente - Ejemplo

```
C:\Wis Programas PIC16F84A\Ensam_03.asm

1 ;Por el puerto B se obtiene el datos de las cinco lineas del Puerto A al que esta conectado
2 ;
3 ;un array de interruptores. Por ejemplo si por el puerto A se introduce xxx11001, por we puerto ;B
4 ;aparecera xxx11001
5 ;
6 ; ZONA DE DATOS
7
8     _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC ; configuracion para el grabador
9     LIST P=16F84A          ;procesador
10    INCLUDE <P16F84A.INC>
11
12    ;ZONA DE CODICOS
13    Inicio    bsf    STATUS, RP0      ;el programa comienza en la direccion 0 de memoria
14                clrf   TRISB      ;pone a 1 el bit 5 del registro status. Acceso a banco 1
15                movlw  b'11111111'  ;Todas las salidas de TRISB se cargan con 0=output
16                movwf  TRISA      ;Se carga w con 1111 1111 binario
17                bcf    STATUS, RP0      ;Todas las salidas de TRISA se cargan con 1=Input
18                Principal
19                movf   PORTA,W      ;pone a 0 el bit 5 del registro status. Acceso a banco 0
20                movwf  PORTB      ;lee el puerto A
21                goto   Principal    ;el contenido de w se carga en PORTAB
22
23    END
24
```

Código Fuente - Ejemplo

10			
11	;ZONA DE CODIGOS		
12	ORG	0	;el programa comienza
13	Inicio	b <u>sf</u> STATUS, RPO	;pone a 1 el bit 5
14		clrf TRISB	;Todas las salidas
15		movlw b'11111111'	;Se carga w con 11
16		movwf TRISA	;Todas las salidas
17		bcf STATUS, RPO	;pone a 0 el bit 5
18	Principal		
19		movf PORTA,W	;lee el puerto A
20		movwf PORTB	;el contenido de w
21		goto Principal	
22			

↑ Campo de etiquetas

 Campo de códigos de operación

 Campo de operándos y datos

 Campo de comentarios

Directivas del Ensamblador

Directiva : EQU

Define constantes en assembler

Es una directiva de asignación .

El valor <expr> es asignado a la etiqueta <label>

La directiva EQU permite al programador "igualar" nombres personalizados a datos o direcciones.

<label> EQU <expr>

Ejemplos:
temp EQU 12
DATO EQU 22
PORT_A EQU 5
START EQU 0
CARRY EQU 3
TIEMPO EQU 5
Bank_1 EQU BSF STATUS,RP0

Directiva : ORG

Set program origin

Indica al programa ensamblador en que dirección inicia el programa.

ORG 0

Esta directiva dice al ensamblador a partir de que posición de memoria de programa se situarán las siguientes instrucciones. Rutinas de comienzo, subrutinas de interrupción y otros programas deben comenzar en locaciones de memoria fijados por la estructura del microcontrolador.

Directiva : CONFIG

Esta directiva le sirve a MPLAB durante la grabación del microcontrolador.

_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

Significa

No hay protección de código

_CP_OFF

No se habilita el Watchdog

_WDT_OFF

Se habilita el reset mediante Power Up Timer

_PWRTE_ON

Se utiliza el oscilador de cristal de cuarzo

_XT_OSC

Directiva : Include <P16F684A.INC>

Indica el archivo en donde se localizan las etiquetas que nombran a los diferentes registros y el valor que le corresponde a cada uno de ellos.

Directiva : LIST P=16F628A

Indica el tipo de microprocesador utilizado

Directiva: END

End program block

Indica el fin del programa

Guía de Lectura

(Obligatoria)

1.- 03 Directivas del ensamblador.PDF

2.- 05 Técnicas de Programación.PDF

Constante Numéricas y Alfanuméricas

TIPO	SINTAXIS	Ejemplo
Decimal	D'<cantidad>' d'<cantidad>' . <cantidad><cantidad>	Movlwl D'109' Movlw d'109' Movlw.109 109
Hexadecimal	H'<cantidad>' h'<cantidad>' 0x<cantidad> <cantidad>H <cantidad>h	MovlwH'6D' momovlwlh'6D' movlw0x6D movlw6DH movlw6Dh
Octal	O'<cantidad>' o'<cantidad>'	MovlwO'155' movlwo'155'

Constante Numéricas y Alfanuméricas

TIPO	SINTAXIS	Ejemplo
Binario	B'<cantidad>' b'<cantidad>'cantidad	MovlwB'01101101' movlw b'01101101'
ASCII	A'<carácter>'<carácter> a'<carácter>,'<carácter>'<carácter>	MovlwA'M'M movlw a'M' movlw 'MM
String	"<string>"	DT"EstudiaDPE"

Repertorio de Instrucciones

Es un juego reducido de 33 instrucciones simples y rápidas

La mayoría de las instrucciones se ejecutan en 4 ciclos de reloj, menos las de salto que requieren 8 ciclos.

Las instrucciones son ortogonales. Casi todas las instrucciones pueden usar cualquier operando.

Todas las instrucciones tiene la misma longitud, 12 bits y todos los datos son de 8 bits.

Repertorio de Instrucciones

Instrucciones de Carga

Instrucciones Aritméticas

Instrucciones Lógicas

Instrucciones de bit

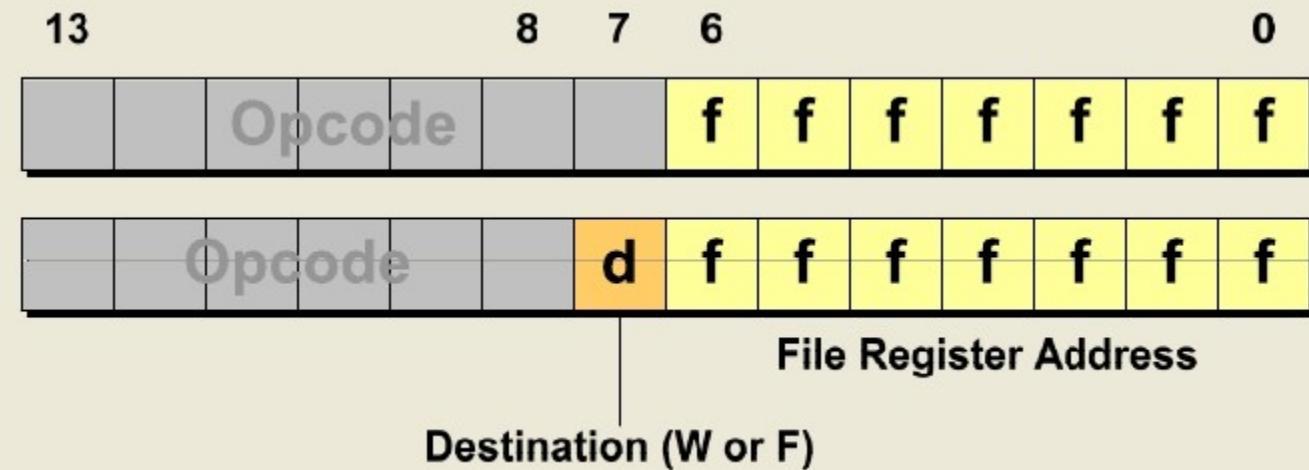
Instrucciones de salto

Instrucciones para manejo de subrutinas

Instrucciones especiales

Introducción al Set de Instrucciones

Byte Oriented Operations



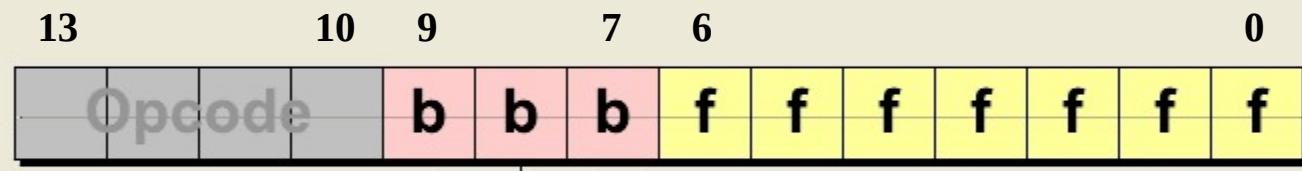
ADDWF 0x25, W

File Register Address

Destination

Introducción al Set de Instrucciones

Bit Oriented Operations



File Register Address

Bit Position (0-7)

BSF 0x25, 3



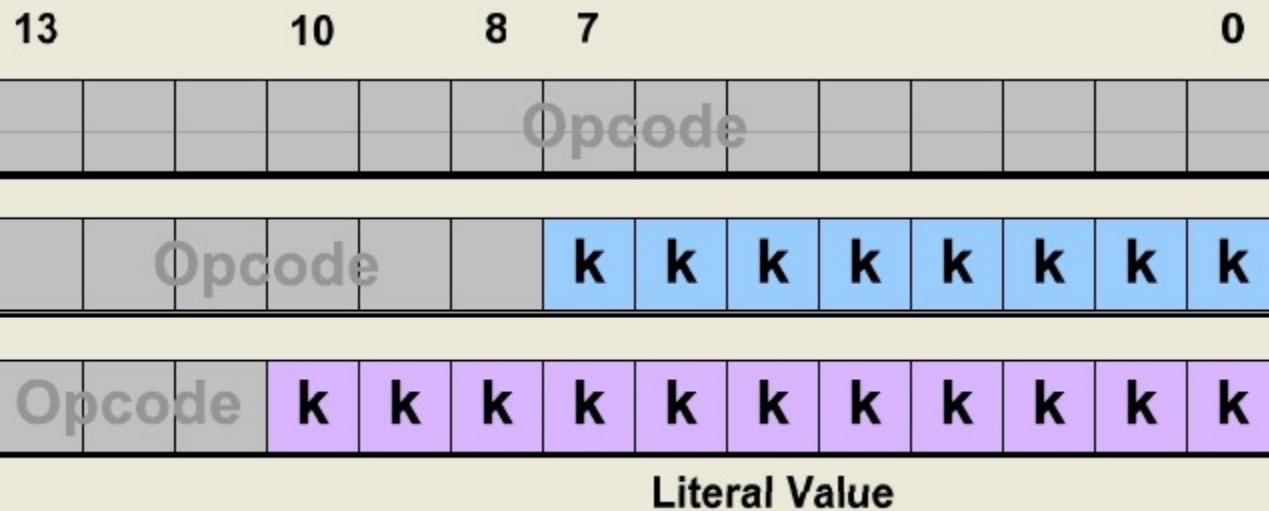
File Register Address



Bit Position

Introducción al Set de Instrucciones

Literal and Control Operations



MOVLW 0x55



Literal Value

Introducción al Set de Instrucciones

Operaciones Basadas en Bytes (Registros)

Diagram of a 16-bit instruction format:

- Fields 13 to 0 are shown.
- Fields 13 to 10 are labeled "Código de Operación (opcode)".
- Field 9 is labeled "d".
- Fields 8 to 3 are labeled "Registro (f= file register)".

Operaciones Orientadas a bits

13	12	11	10	9	8	7	6	5	4	3	2	1	0
Código Operación(opcode)				b= bit				Registro (f= file register)					

Operaciones Orientadas a Literales

The diagram illustrates a 16-bit instruction format. The bits are numbered from 13 down to 0. Bits 13 through 4 are labeled "Código de Operación (opcode)" and bits 3 through 0 are labeled "k=Literal".

Para las operaciones call y goto

A horizontal bar divided into 14 segments, indexed from 13 to 0 above it. The first segment (index 13) is highlighted in yellow and labeled "opcode". The last three segments (indices 2, 1, 0) are also highlighted in yellow.

Set de instrucciones de la Linea Base

Operaciones orientadas a Byte		
addwf	f,d	Add W and f
andwf	f,d	AND W with f
clrf	f	Clear f
clrw	-	Clear W
comf	f,d	Complement f
decf	f,d	Decrement f
decfsz	f,d	Decrement f, Skip if 0
incf	f,d	Increment f
incfsz	f,d	Increment f, Skip if 0
iorwf	f,d	Inclusive OR W with f
movf	f,d	Move f
movwf	f	Move W to f
nop	-	No Operation
rlf	f,d	Rotate Left f through Carry
rrf	f,d	Rotate Right f through Carry
subwf	f,d	Subtract W from f
swapf	f,d	Swap nibbles in f
xorwf	f,d	Exclusive OR W with f

Operaciones Orientadas a Bit		
bcf	f,b	Bit Clear f
bsf	f,b	Bit Set f
btfsc	f,b	Bit Test f, Skip if Clear
btfss	f,b	Bit Test f, Skip if Set
Operaciones con Literales y control		
andlw	k	AND literal with W
call	k	Call subroutine
clrwdt	-	Clear Watchdog Timer
goto	k	Go to address
iorlw	k	Inclusive OR literal with W
movlw	k	Move literal to W
option	-	Load OPTION Register
retlw	k	Return with literal in W
sleep	-	Go into standby mode
tris	f	Load TRIS Register
xorlw	k	Exclusive OR literal with W

01 El lenguaje ensamblador del PIC16F84A V1.pdf



CLRF				CLRF
Clearf				
Operación	00 h → f 1 → Z			
Sintaxis	[Etiqueta] CLRF f			
Operadores	0 < f < 127			
Ciclos	1			
OPCODE	00	0001	1fff	<u>Efff</u>
Descripción	Se borra el contenido del registro f y el <u>flag</u> Z se activa			

El OPCODE de CLRF f es en binario "0000011fffffff" donde "fffffff" se sustituiría por el registro que se quiera borrar. f es una de las abreviaturas que se utilizan para describir las instrucciones del PIC usados en el lenguaje ensamblador y que son:

- f Representa un registro cualquiera de la memoria de datos.
- w Registro de trabajo (Working Register).
- b Dirección de un bit dentro de un registro de 8 bits (0-7).
- l ó k Literal o constante de 8 bits.
- d Bit de destino, 0 ó 1.
- x Los bits que estén representados por este tipo de dato no tienen ninguna función y su valor lo define el compilador.

A continuación se explican con más detalle:

Configuración Básica MP LAB

mplab_capitulo1.PDF

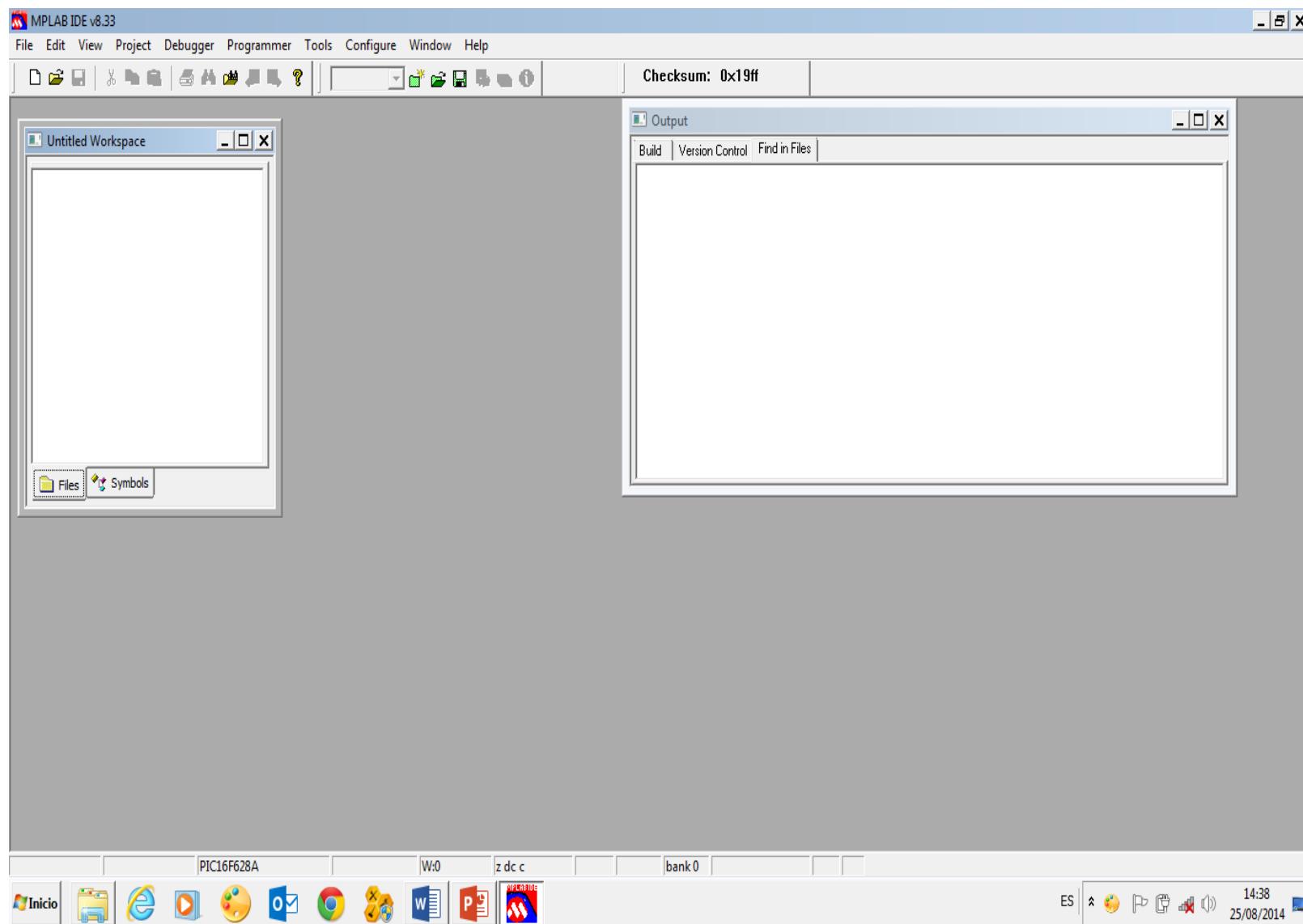
mplab_capitulo2.PDF

mplab_capitulo3.PDF

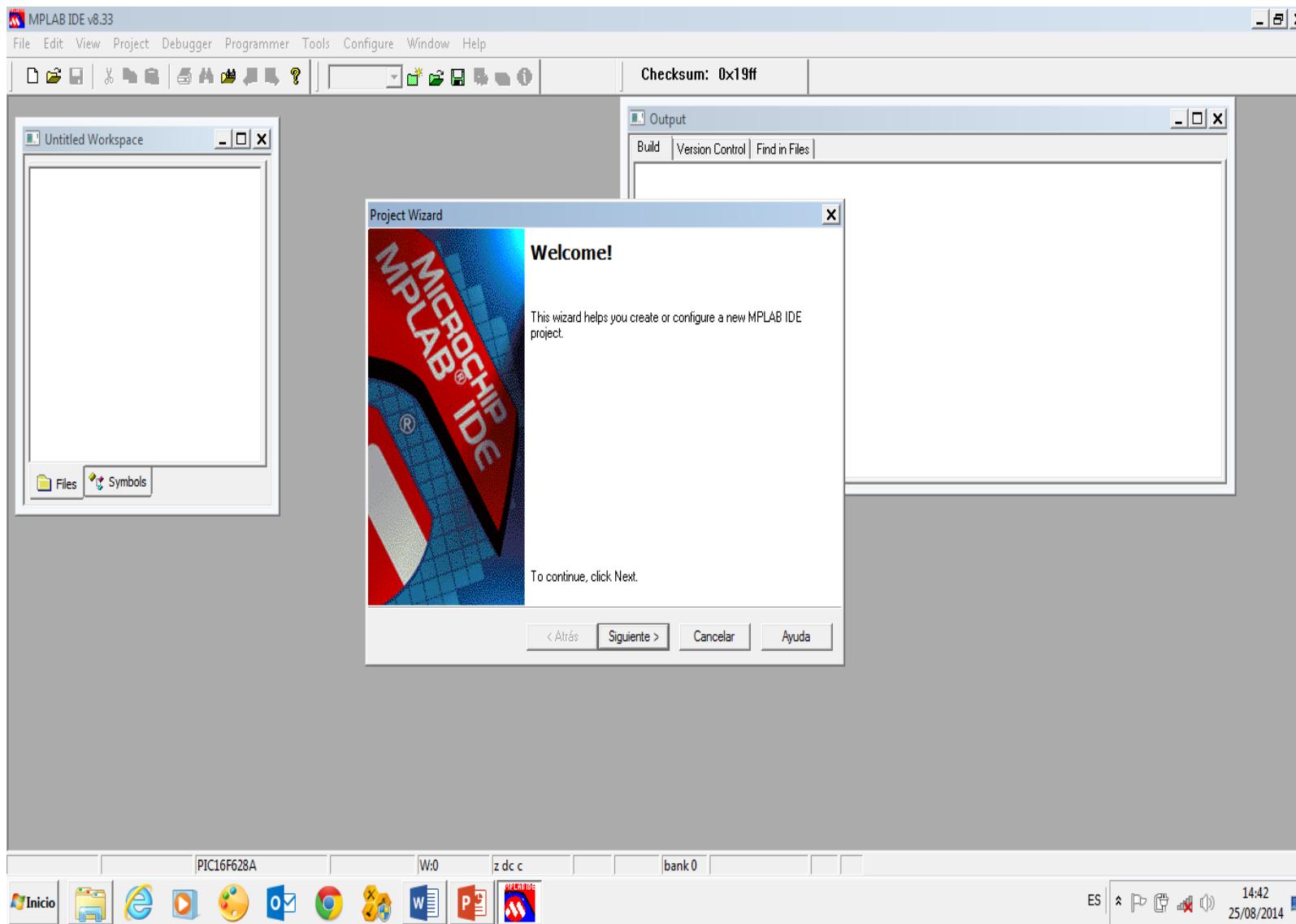
mplab_capitulo4.PDF

mplab_capitulo5.PDF

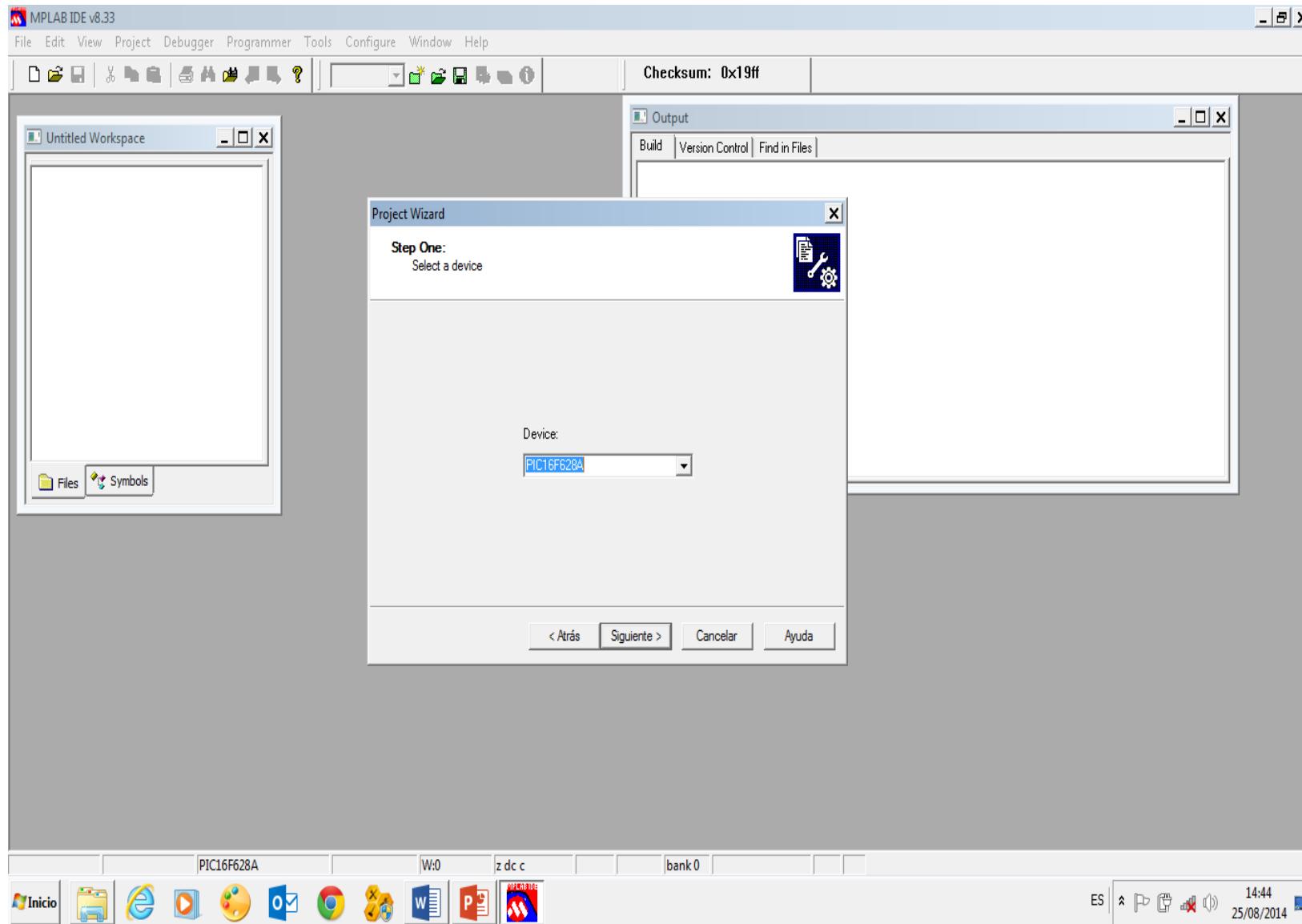
MPLab Pantalla Inicial



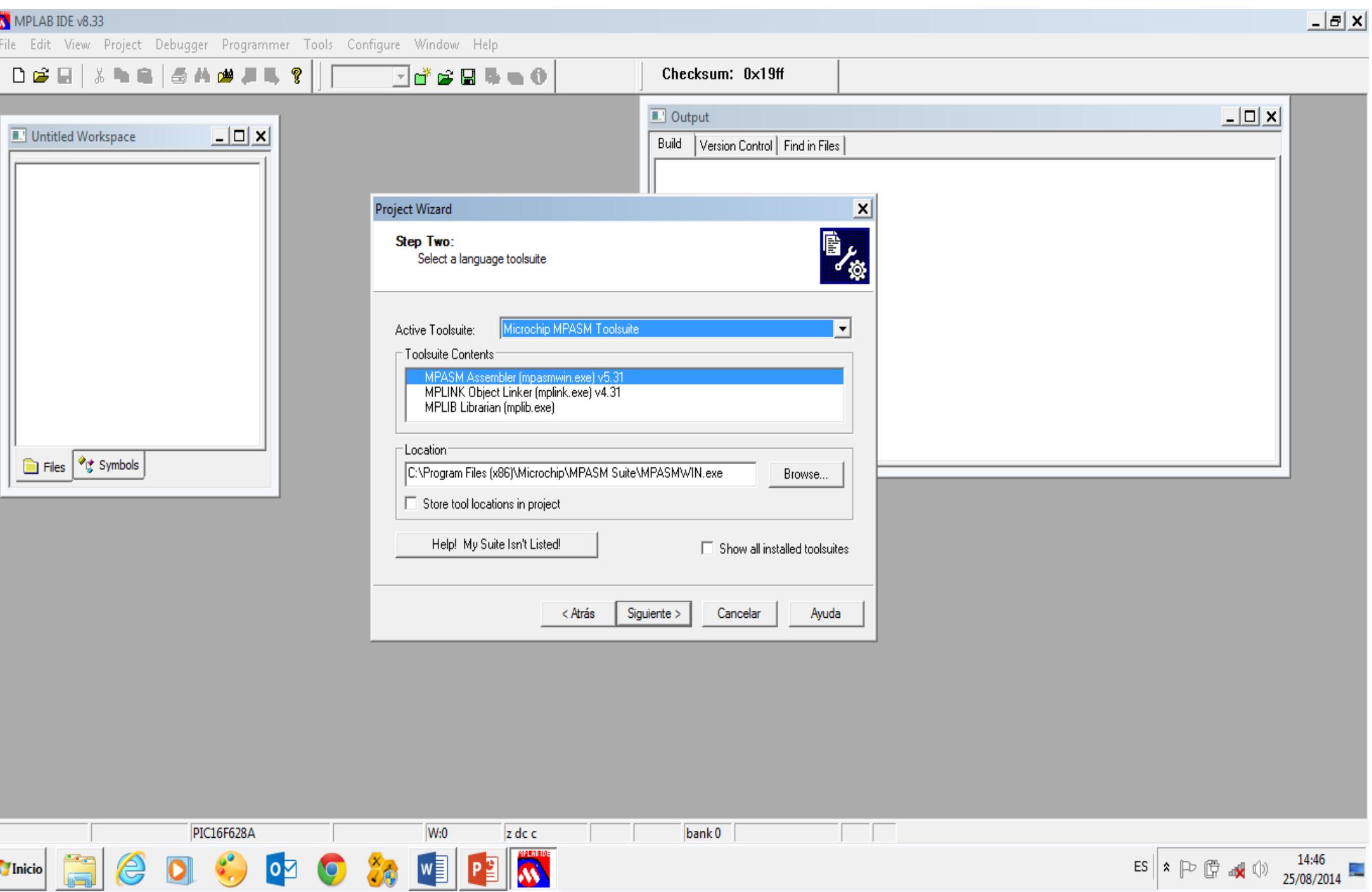
MPLab - Project Wizard



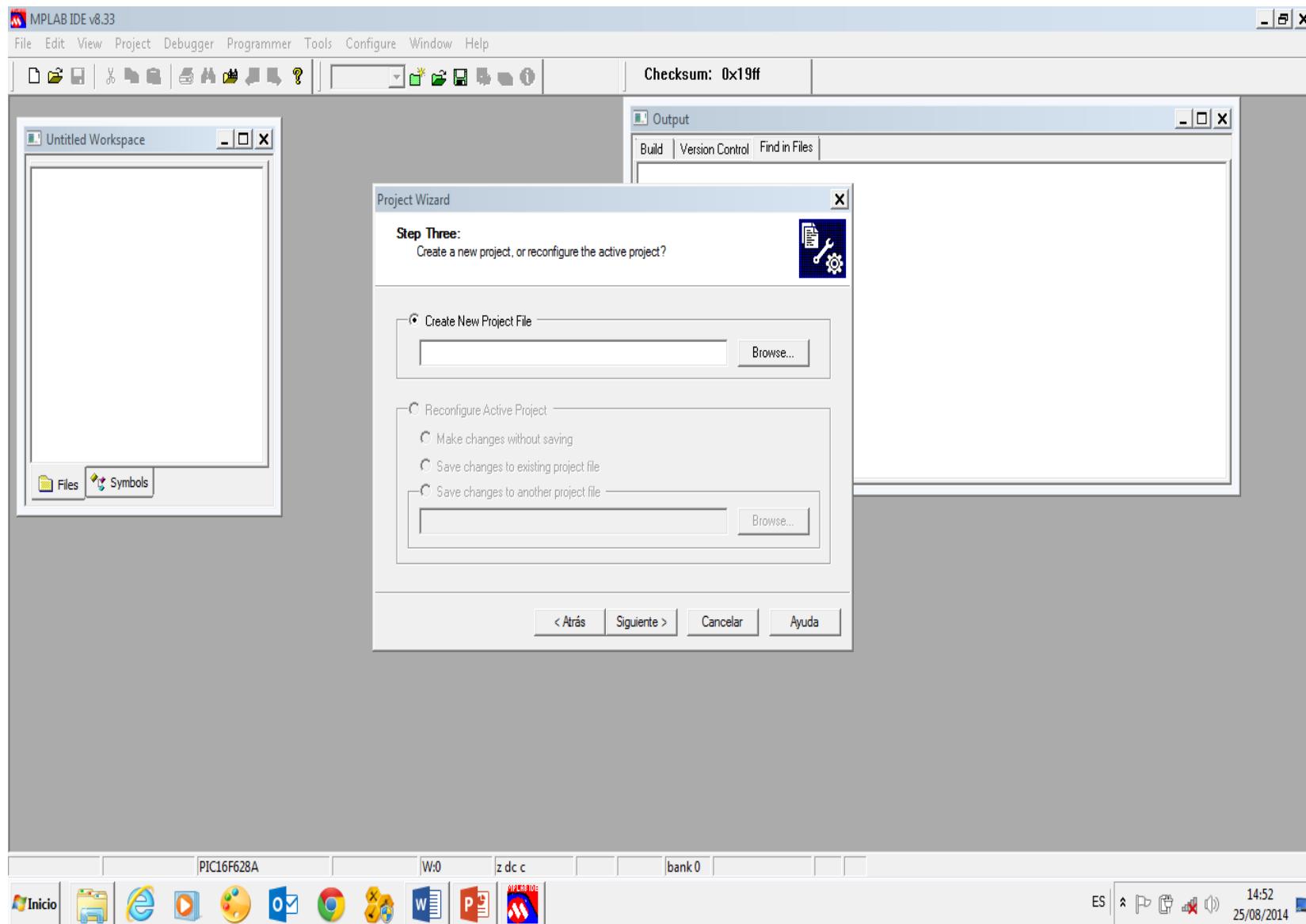
MPLab – Seleccionar Microcontrolador

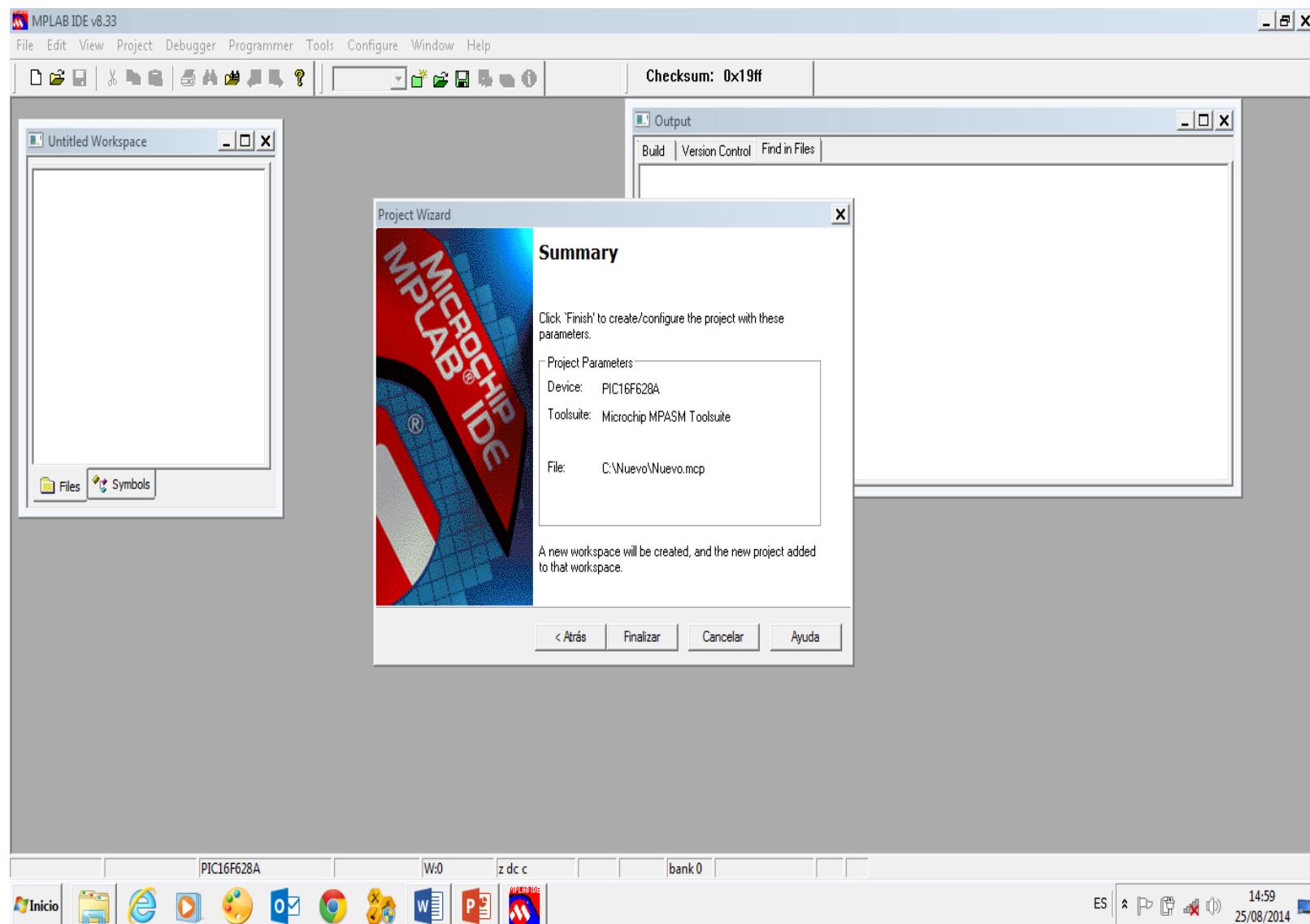


MPLab – Seleccionar Lenguaje de

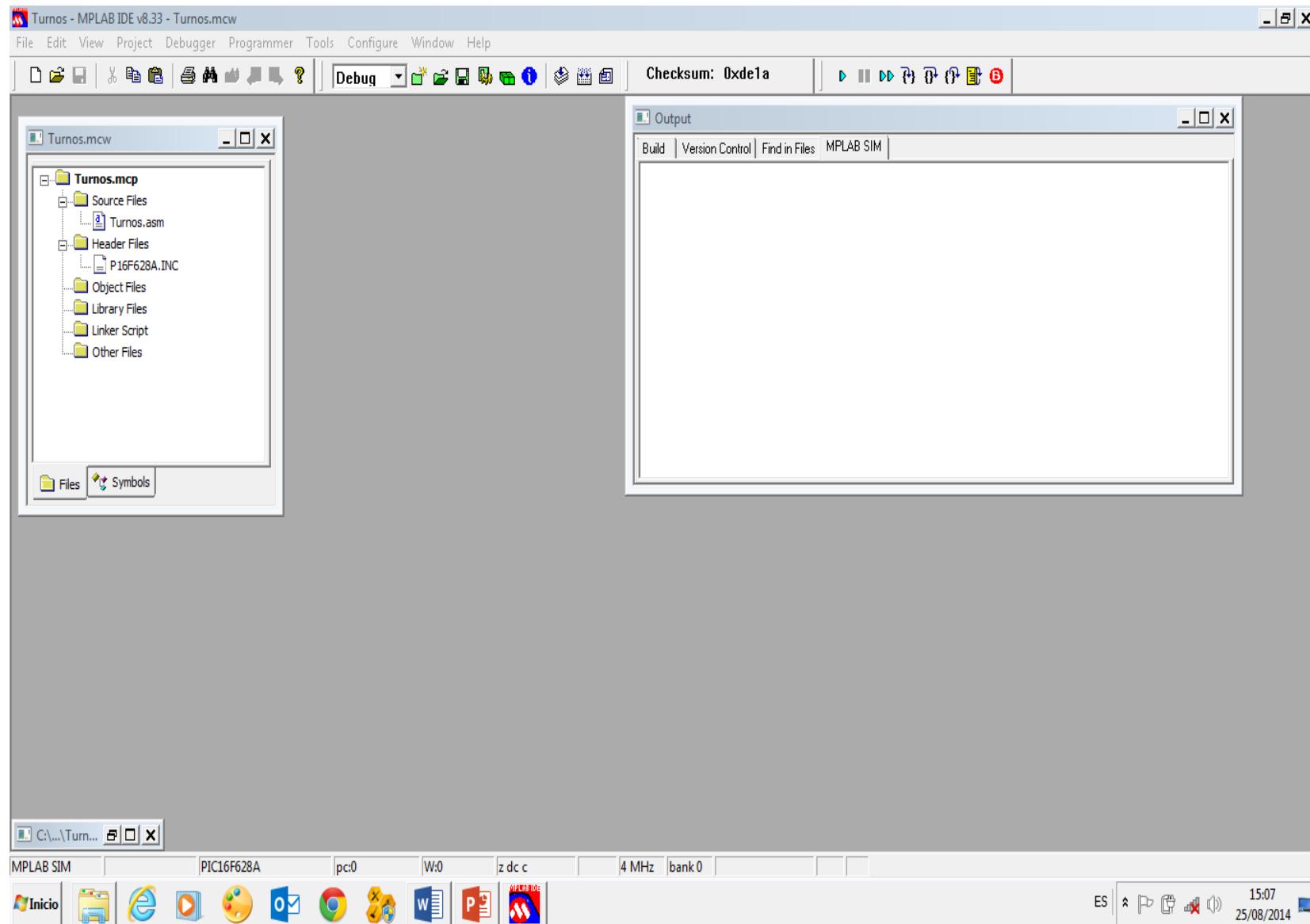


MPLab – Identificar Proyecto



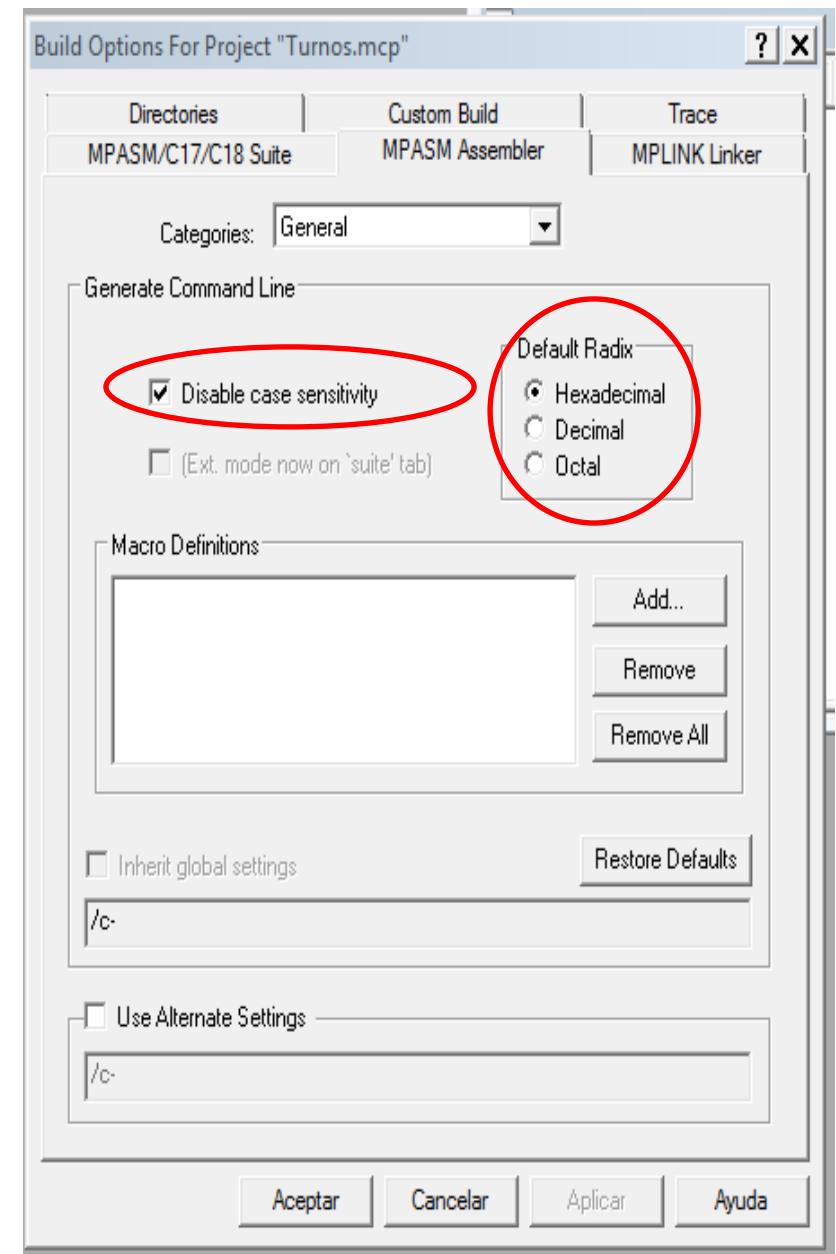


MPLab - Estructura de un Proyecto



Menú Project, Build options

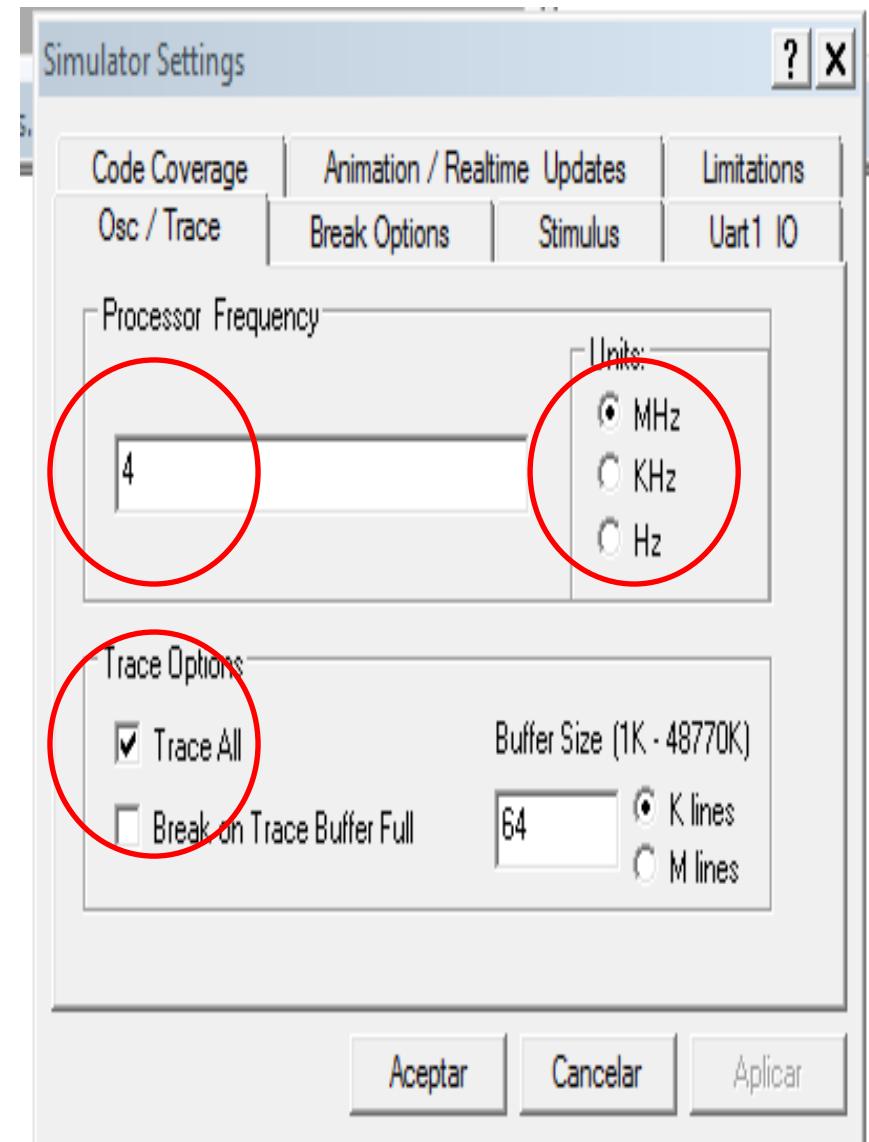
1. Ir al menú Project, Build options y elegir la solapa para configurar el MPASM Assembler.
2. Tildar la opción para deshabilitar la distinción entre mayúsculas y minúsculas, de forma que si editamos el programa con minúsculas, el ensamblador reconozca el archivo de encabezado que habitualmente está escrito en mayúsculas.
3. Verificar la opción de que trabaje en hexadecimal por defecto.



Simulador - Seteo

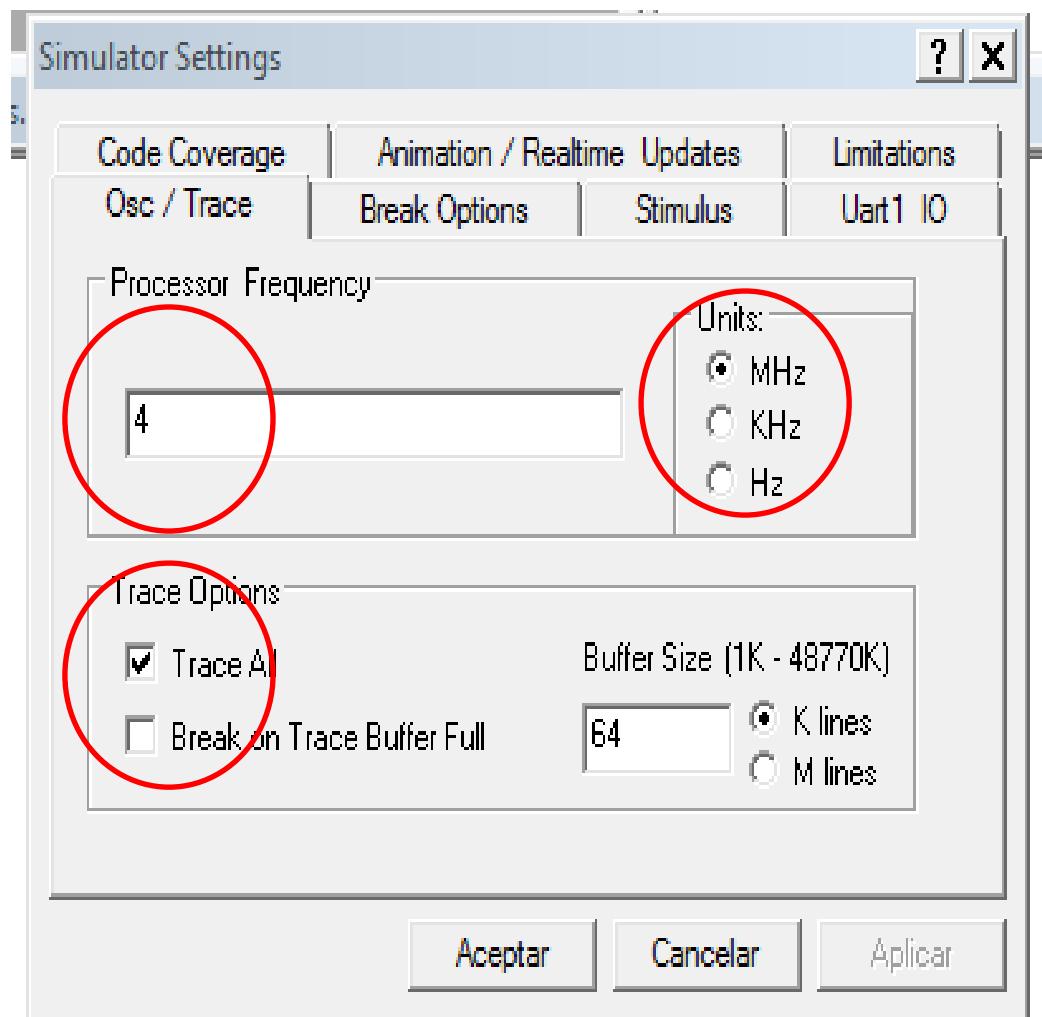
Lo siguiente es elegir la herramienta para depuración (debugger)

1. Tildamos MPLabSim.
2. Después de esta elección debería aparecer una barra de herramientas que nos permitirá compilar y correr la compilación, ejecutar instrucción por instrucción, colocar puntos de detención, etc.; es decir serán las herramientas de depuración de nuestro programa (subrutina, función).



Simulador - Seteo

- Lo que deberemos elegir a continuación es la frecuencia del reloj para la simulación: menú Debugger (depuración), Settings (configuración), solapa Osc/Trace: elegir la frecuencia en el valor que usaremos comúnmente: 4 MHz.
- Se podría elegir la otra



Simulador - Seteo

Ahora sigue configurar al PIC, a través de la “palabra de configuración”. Esta palabra puede estar definida en el programa o aquí.

Esta palabra establece los siguientes parámetros para el funcionamiento del PIC:

1. El tipo de reloj (oscilador) que va a usar (interno, externo, por cristal, etc.) Nosotros usaremos el RC interno porque no necesitamos una precisión extrema y no inutilizamos dos de las patas de E/S del PIC.
2. El timer denominado “perro guardián” lo usaremos deshabilitado
3. El timer de encendido también deshabilitado
4. Habilitaremos el borrado total en un reset.
5. No interesa el siguiente que es la detección de apagado
6. La siguiente opción debe habilitarse para poder luego programar el PIC con tensiones normales (de lo contrario hace falta mayor tensión para programarlo)
7. Deshabilitamos la protección de lectura de la EEPROM
8. Deshabilitamos la protección del código del programa

Configuration Bits				
<input checked="" type="checkbox"/> Configuration Bits set in code.				
Address	Value	Field	Category	Setting
2007	3F10	OSC	Oscillator	INTOSC: I/O on RA6/OSC2/CLKOUT, I/O on RA7/OSC1/C
		WDT	Watchdog Timer	Off
		PUT	Power Up Timer	Enabled
		MCLRE	Master Clear En	Disabled
		BODEN	Brown Out Detect	Disabled
		LVP	Low Voltage Program	Disabled
		CPD	Data EE Read Protection	Disabled
		CP	Code Protect	Off

Practica Programación Assembler

Programas Simple

Estructura de un Programa Simple

Su estructura en un programa ejemplo muy simple:

```
;*****  
;Programa E001.asm                                Fecha: 3 Diciembre 2004  
;Este programa suma dos valores inmediatos (7+8) y el resultado  
;lo deposita en la posición 0x10  
;Revisión: 0.0                                     Programa para PIC16F84  
;Velocidad de reloj: 4 MHz                          Instrucción: 1Mz=1 us  
;Perro Guardián: deshabilitado                    Tipo de Reloj: XT  
;Protección de código: OFF  
;*****  
      LIST      p=16F84 ;Tipo de PIC  
*****  
RESULTADO    EQU 0x10      ;Define la posición del resultado  
*****  
        ORG 0          ;Comando que indica al Ensamblador  
                  ;la dirección de la memoria de programa  
                  ;donde situar la siguiente instrucción  
*****  
INICIO       movlw   0x07      ;Carga primer sumando en W  
             addlw   0x08      ;Suma W con segundo sumando  
             movwf   RESULTADO ;Almacena el resultado  
  
END         ;Fin del programa fuente
```

Cabecera

Tipo de microcontrolador

Definición de constantes

Comentarios

Programa

Hemos visto la estructura general. Ahora veremos la posición de los elementos del código por 4 columnas:

Etiquetas Operación Operandos Comentarios

INICIO

movlw 0x07

;Carga primer sumando en W

addlw 0x08

;Suma W con segundo sumando

movwf RESULTADO

;Almacena el resultado

MOVLW					MOVLW
Move literal to w					
Operación	$k \rightarrow w$				
Sintaxis	[Etiqueta] MOVLW k				
Operadores	$0 < k < 255$				
Ciclos	1				
OPCODE	11	00xx	kkkk	kkkk	
Descripción	El registro w se carga con el valor de 8 bits del literal k				

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	X	-	-

EJEMPLO:

MOVLW 0x5A

Al ejecutarse:

w = 5A h

ADDLW					ADDLW
ADD Literal to w					
Operación	$w + k \rightarrow w$				
Sintaxis	[Etiqueta] ADDLW k				
Operadores	$0 < k < 255$				
Ciclos	1				
OPCODE	11	111x	kkkk	kkkk	
Descripción	Suma el contenido del registro w al literal k, y almacena el resultado en w. Si se produce acarreo el flag C se pone a "1".				

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	X	X	X

C Se pone a 1 si se produce un Acarreo desde el bit de mayor peso.

DC Se pone a 1 si se genera un Acarreo del bit 3 al bit 4.

Z Se pone a 1 si el resultado de la operación es cero.

EJEMPLO:

ADDLW 0x15

Si antes de la instrucción:

w = 10h = 0001 0000 b

Al ejecutarse la instrucción

w = 10 h + 15 h = 25 h

w = 0001 0000 b + 0001 0101 b = 0010 0101 b

0001 0000 b

0001 0101 b

0010 0101 b

MOVWF				MOVWF
Move w to f				
Operación	w → f			
Sintaxis	[Etiqueta] MOVWF f			
Operadores	0 < f < 127			
Ciclos	1			
OPCODE	00	0000	1fff	ffff
Descripción	Mueve el contenido del registro w al registro f			

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	-	-	-

EJEMPLO:

MOVWF OPCION

Si antes de la instrucción:

OPCION = FF h

w = 4F h

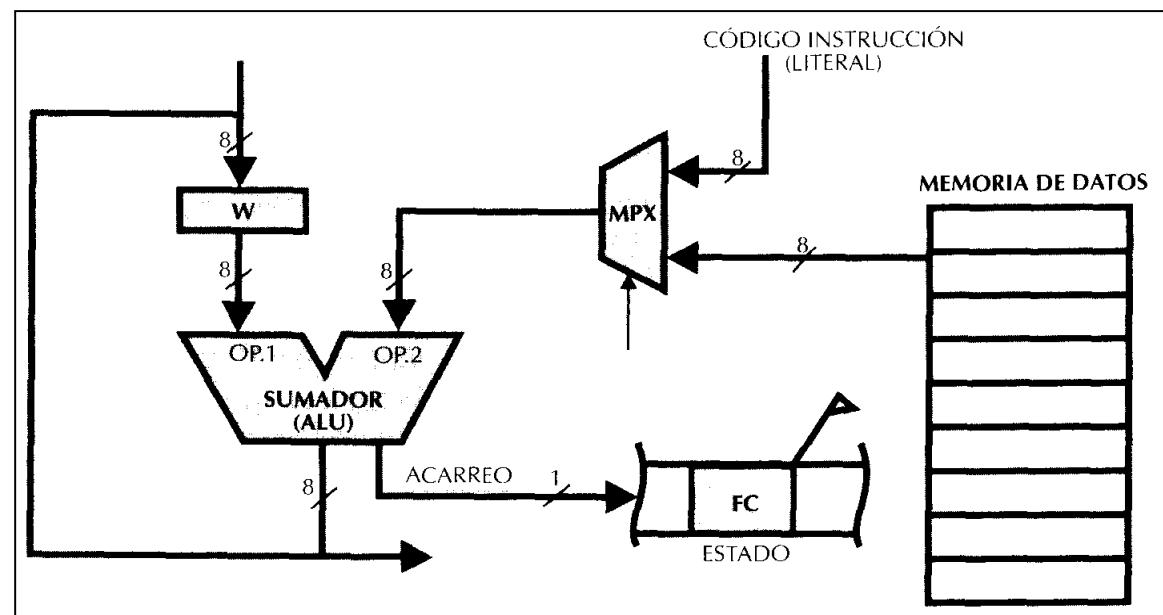
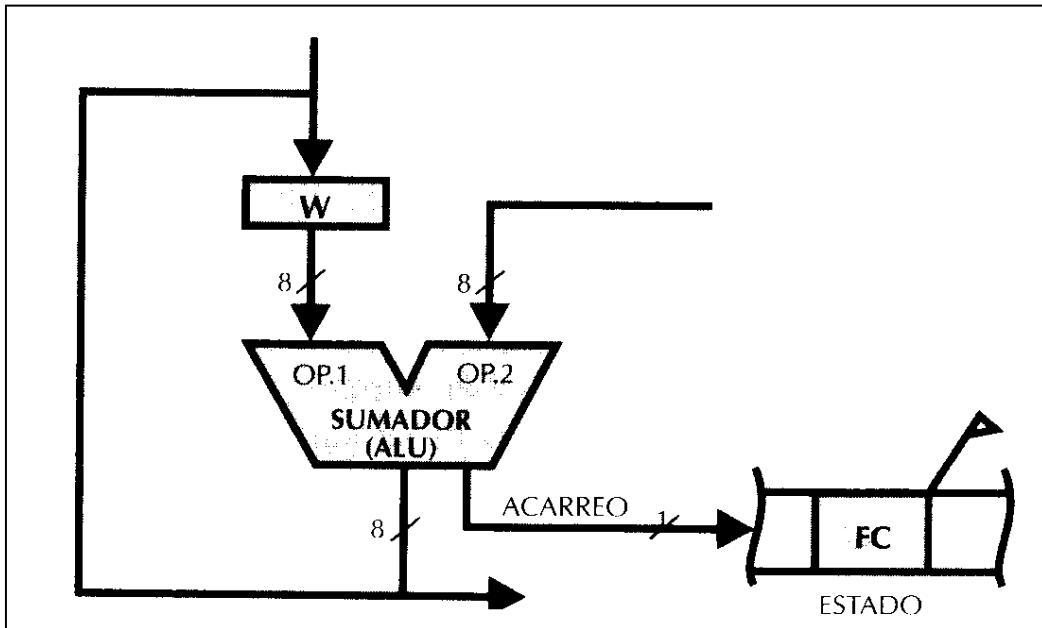
Al ejecutarse:

OPCION = 4F h

w = 4F h

Primer Programa Assembler

Primer Programa ASSEMBLER



Primer Programa ASSEMBLER (14 bits)

ENUNCIADO

El ejercicio maneja tres posiciones de la memoria de datos. En la dirección 0Ch se depositará el primer operando, en la 0Dh el segundo y en la 0Eh el resultado de la suma de los dos primeros operandos. Como valor para el primer operando se va a usar 5 y como segundo operando el 2.

```
LIST P = 16F628      ; Indica el modelo PIC que se utiliza  
                     ; Es una directiva del Ensamblador  
                     ; ZONA de ETIQUETAS  
  
OPERANDO1 EQU 0x0c ; Define la posición del operando 1  
OPERANDO2 EQU 0x0d ; Define la posición del operando 2  
RESULTADO EQU 0x0e ; Define la posición del resultado  
  
ORG0               ; Comando que indica al ensamblador  
                     ; la dirección de memoria de programa  
                     ; donde se situará la siguiente instrucción  
  
movlw 05            ; 5 => W (primera instrucción)  
movwf OPERANDO1    ; W => operando1  
movlw 02            ; 2 => W  
movwf OPERANDO2    ; W => operando2  
movfw OPERANDO1    ; operando 1 => W  
addwf OPERANDO2, 0 ; W + operando2 => W  
movwf RESULTADO     ; W => resultado  
END                 ; Directiva de fin de programa
```

Directivas del Compilador

- Directivas del Compilador en mayúsculas
- Nombre de variables en mayúsculas
- Nemáticos en minúsculas
- Valores de operando (constantes)
- Programas bien estructurados

Primer Programa ASSEMBLER

(Comportamiento de las instrucciones)

<i>movff,d:</i>	Mueve el contenido del operando fuente, que es una posición de memoria de datos, al destino, que bien puede ser el <u>registro W</u> cuando <u>d=0</u> , o el propio fuente cuando <u>d=1</u> .
<i>movwf f:</i>	Mueve el contenido del registro W a la posición de la memoria de datos identificada por <u>f</u> . Realiza la transferencia <u>W => f</u> .
<i>movlw k:</i>	Mueve el literal <u>k</u> incluido en el código de la instrucción al registro W. Realiza la transferencia <u>k => W</u> .
<i>addwf f,d:</i>	Suma el contenido del <u>registro W</u> con el de <u>f</u> y deposita el resultado en <u>W</u> si el valor de <u>d=0</u> . Si <u>d=1</u> lo deposita en <u>f</u> . _
<i>addlw k:</i>	Suma al contenido del <u>registro W</u> el literal que acompaña a la instrucción y deposita el resultado en el <u>registro W</u> (<u>W + k => W</u>)

Practica Programación Assembler

Practica 1

```

;*****
;Practica 1 - Encendido de un Led
;Este programa enciende un led conectado al puerto B del micro en el placa
;*****
#include <p16F628.inc>      ;Incluimos el archivo que contiene los registros del
                           ;micro que utilizaremos

org 0                      ;Instruccion al compilador, esta indica que el codigo
                           ;precedente
                           ;a esta instruccion se situara en la direccion
                           ;indicada, en este
                           ;caso la direccion 0 Hexa

Inicio:
bsf    STATUS, RP0      ;selecciono banco 1
bcf    TRISB, 0          ;Configuro el pin RB0 como salida
bcf    STATUS, RP0      ;Volvemos al banco 0
bsf    PORTB, 0          ;Enciendo el Led
goto   $                  ;Me quedo esperando
end

```

BSF		BSF		
Bit Set F				
Operación	1 → (f)			
Sintaxis	[Etiqueta] BSF f,b			
Operadores	0 < f < 127 0 < b < 7			
Ciclos	1			
OPCODE	01	11bb	bfff	ffff
Descripción	Pone a 1 el bit b del registro f			

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	-	-	-

EJEMPLO:

BSF FLAG_REG, 7

Si antes de la instrucción el registro tiene el valor:

FLAG_REG = 0A h = 0000 1010 b

Al ejecutarse la instrucción, el registro queda con el valor:

FLAG_REG = 8A h = 1000 1010 b

BCF			BCF
Bit Clear F			
Operación	$0 \rightarrow (f)$		
Sintaxis	[Etiqueta] BCF f,b		
Operadores	$0 < f < 127$ $0 < b < 7$		
Ciclos	1		
OPCODE	01	00bb	bfff
Descripción	Pone a cero el bit número b del registro f.		

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	-	-	-

EJEMPLO:

BCF FLAG_REG, 7

Si antes de la instrucción el registro:

FLAG_REG = C7 h = 1100 0111 b

Al ejecutarse la instrucción, el registro queda con el valor:

FLAG_REG = 47b = 0100 0111 b

GOTO					GOTO
Unconditional Branch					
Operación	$k \rightarrow PC <10:0>$ $(PCLATH <4:3>) \rightarrow (PC <12:11>)$				
Sintaxis	[Etiqueta] GOTO k				
Operadores	0 < k < 2047				
Ciclos	2				
OPCODE	10	1kkkk	kkkk	kkkk	
Descripción	Salto incondicional, normalmente se utiliza para llamar a la subrutina situada en la dirección que se carga en PC. El modo de cálculo de la instrucción carga desde el bit 0 al 10 de la constante k en el PC y los bits 3 y 4 del registro PCLATH en los 11 y 12 del PC				

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	-	-	-

EJEMPLO:

GOTO SEGUIR

Al ejecutarse:

PC = dirección SEGUIR

Practica Programación Assembler

Practica 2

Técnicas de Programación

1.- Movimientos de Datos (DATO)

- Mover los datos de un registro a al registro W
- Mover un literal (constante) al registro W
- Mover un literal a un registro.

2.- Utilizar las instrucciones apropiadas para

- Poner en cero los bits de un registro
- Poner en cero los bits del registro W
- Poner en cero el bit 5 del registro DATO
- Poner en uno el bit 4 del registro DATO

3.- Sumar un valor 19 residente en W al registro Dato que debe estar en cero.

4.- Sumar el Valor de W con el registro DATO guardar en el resultado en W.

5.- Idem anterior pero guardando en registro DATO1.

6.- Incrementar en 1 el valor del registro DATO y guardar el resultado DATO1 y W.

7.- Decrementar en 1 el valor del registro DATO y guardar el resultado DATO1 y W.

8.- Complementar el valor de Dato y guardar el resultado en DATO1 y W.

9.- Comparar el valor del registro DATO y DATO1; si DATO > DATO1 Guardar en DATO3 el Valor 1.
Si DATO < DATO1 guardar en DATO3 el Valor 0.

INDICAR LOS VALORES ASOCIADOS AL REG. STATUS CARRY (C) CERO (Z) y ACARREO DE DÍGITO (DC) SEGÚN CORRESPONDA

```

; ****
; Practica 2: Parpadeo
; En la primera practica vimos como encender un led, ahora haremos una rutina de encendido y apagado del led.
; Para poder ver el parpadeo debemos agregar un retardo a cada encendido y apagado del led.
; ****
#include <p16F628.inc>

Delay1          ; Definimos 2 registros que usaremos
Delay2          ; en los retardos Delay1 y Delay2

    org 0      ; Iniciar el Programa en la dirección 0 de la memoria de programa
Inicio:
    bsf      STATUS,RP0      ; seleccionar banco 1 para su utilización
    bcf      TRISB,0         ; Definir RB0 como salida
    bcf      STATUS,RP0      ; Regresar al banco 0

LoopPrincipal:           ; Crear subrutina LoopEncendido
    bsf      PORTB,0         ; Encendemos led conectado a RB0

LoopEncendido:           ; Crear subrutina LoopEncendido
    decfsz  Delay1,f        ; Decremento Delay1 hasta llegar a cero
    goto    LoopEncendido   ; Cada loop toma 3 ciclos de maquina * 256 loops= 768 instrucciones
    decfsz  Delay2,f        ; el proximo loop toma 3 ciclos en volver al primer loop, asi 256 veces
    goto    LoopEncendido   ; (768+3) * 256 = 197376 instrucciones / con ciclos de 1uSeg = 0.197 seg
    bcf      PORTB,0         ; apagamos el led en RB0

LoopApagado:             ; Crear subrutina LoopApagado
    decfsz  Delay1,f        ; hacemos el mismo retardo anterior
    goto    LoopApagado
    decfsz  Delay2,f
    goto    LoopApagado
    goto    LoopPrincipal   ; y volvemos todo de nuevo...
    end

```

DECFSZ		DECFSZ		
<u>Decrement f , Skip if 0</u>				
Operación	$f - 1 \rightarrow d$, salta si resultado = 0			
Sintaxis	[Etiqueta] DECFSZ <u>f,d</u>			
Operadores	$0 < f < 127$ $d [0,1]$			
Ciclos	1 (2)			
OPCODE	00	1011	<u>ffff</u>	<u>ffff</u>
Descripción	Decrementa el contenido del registro f en una unidad, el resultado se almacena en f si d=1 y en w si d=0, en este caso, f no varía. Si el resultado es cero, se ignora la siguiente instrucción y, en ese caso la instrucción tiene una duración de dos ciclos.			

Registro de STATUS

PA2	PA1	PA0	TO#	PD#	Z	DC	C
-	-	-	-	-	-	-	-

EJEMPLO:

INICIO

DECFSZ CNT_1

GOTO

LOOP

CONTINUAR

Si antes de la instrucción:

PC = dirección INICIO

Al ejecutarse: CNT = CNT - 1

Si CNT = 0 entonces PC = dirección CONTINUAR

Si CNT no = 0 entonces PC = dirección INICIO + 1

CLRF				CLRF
Clear f				
Operación	$00 \text{ h} \rightarrow f$ $1 \rightarrow Z$			
Sintaxis	[Etiqueta] CLRF f			
Operadores	$0 < f < 127$			
Ciclos	1			
OPCODE	00	0001	1fff	ffff
Descripción	Se borra el contenido del registro f y el flag Z se activa			

El OPCODE de CLRF f es en binario "0000011fffff" donde "fffff" se sustituiría por el registro que se quiera borrar. f es una de las abreviaturas que se utilizan para describir las instrucciones del PIC usadas en el lenguaje ensamblador y qué son:

- f Representa un registro cualquiera de la memoria de datos.
- w Registro de trabajo (Working Register).
- b Dirección de un bit dentro de un registro de 8 bits (0-7).
- l ó k Literal o constante de 8 bits.
- d Bit de destino, 0 ó 1.
- x Los bits que estén representados por este tipo de dato no tienen ninguna función y su valor lo define el compilador.

Primer TP (Manejo con display) Contadores

1. *Descripción del TP*
2. *Circuito Electrico*
3. *Diagrama global*
4. *Diagrama Detallado*
5. *Codificación con descripción de bloque y sentencias*
6. *Prueba Proteus*

A continuación se publica los archivos con la programación y el simulador Proteus para probar el programa. de acuerdo a lo desarrollado en clase.

Diagramas de flujo.vsd (78336)

Proyecto 1.pdsprj (15136)

TP-turnos-solucion.asm (2732)

Descripción del TP. TP turnos.doc (97792)

Se solicita efectuar las siguientes actividades.

- 1.- Proyecto en MPLab
 - 2.- Generar la descripción funcional de la programación del programa TP-turnos.asm
 - 3.- Mediante la utilización de la IDE MPLab efectuar la ejecución paso a paso del programa mostrando la variación de los valores de las variables para efectuar la validación funcional de la solución presentada.
 - 4.- Mediante el simulador Proteus se procederá a efectuar la prueba de la solución presentada (el contador de turnos debe comenzar con el valor inicial 00)
 - 4.- en los puntos 3 y 4 se deberá efectuar las capturas de pantallas que evidencie la actividad solicitada con los comentario de la representación de la misma.
 - 5.- Si hubiere que efectuar algún ajuste funcional
- 6.- Responder**
- a) cómo funciona la tabla de conversión
 - b) explicar por qué se elijen esos valores.

Palabra de configuración

CP	-	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	-	-	PWRTE	WDTEN	FOSC1	FOSC0
bit 1										bit 0			

Localizada en la memoria de Programa fuera del alcance del Contador de Programa

Usada para programar las características del dispositivo:

- Code Protection
 - Oscillator Mode
 - Watchdog Timer
 - Power Up Timer
 - Brown Out Reset
 - Low Voltage Programming
 - Flash Program Memory Write

Solo leible durante el proceso de grabación en los PIC16

PIC16 Opciones del Oscilador

XT	Standard frequency crystal oscillator	100kHz - 4MHz
HS	High frequency crystal oscillator	4MHz - 20MHz
LP	Low frequency crystal oscillator	5kHz - 200kHz
RC	External RC oscillator	DC - 4MHz
INTRC	Internal RC oscillator	4 or 8 MHz ± 2%

G

Un oscilador seleccionable ofrece mayores posibilidades al diseñador:

- LP Oscilador de baja frecuencia
- RC or INTRC provee una solución de ultra bajo costo
- XT optimizado para la mayoría de las frecuencias de oscilador comúnmente usadas
- HS optimizado para excitar cristales de alta frecuencia

G

Son pautadas solo las gamas de velocidades

POR, , OST, , PWRT

POR: Power On Reset

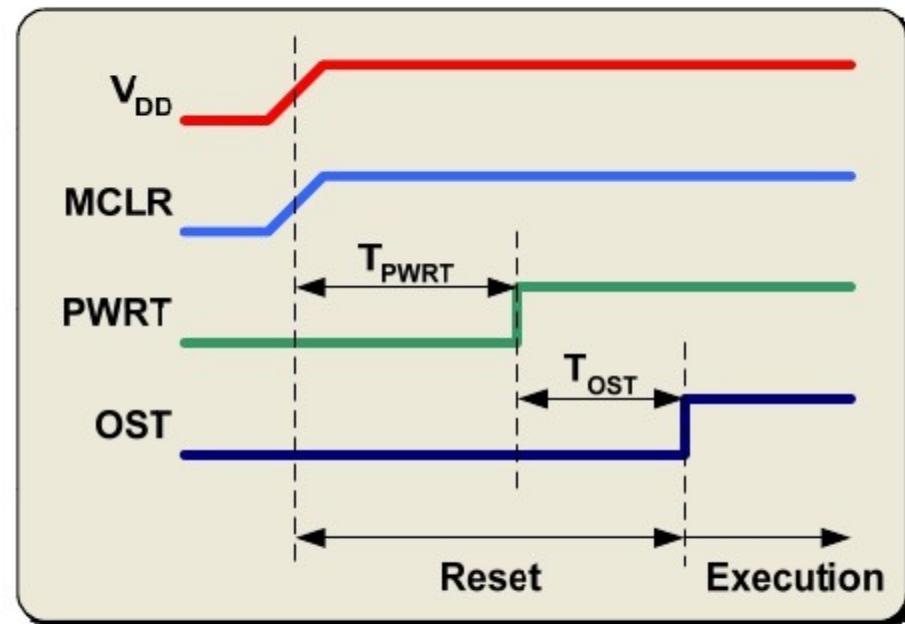
- Con MCLR conectado a VDD, es generado un pulso de Reset cuando es detectada la subida de Vdd

PWRT: Power Up Timer

- Mantiene al dispositivo Reseteado por 72ms (nominal) (despues del POR)

OST: Oscillator Start-up Timer

- Mantiene al dispositivo Resteado por 1024 ciclos para permitir al cristal estabilizar su frecuencia y amplitud; no activo en modo RC ; usado despues del POR o al despertar del SLEEP



Modo Sleep

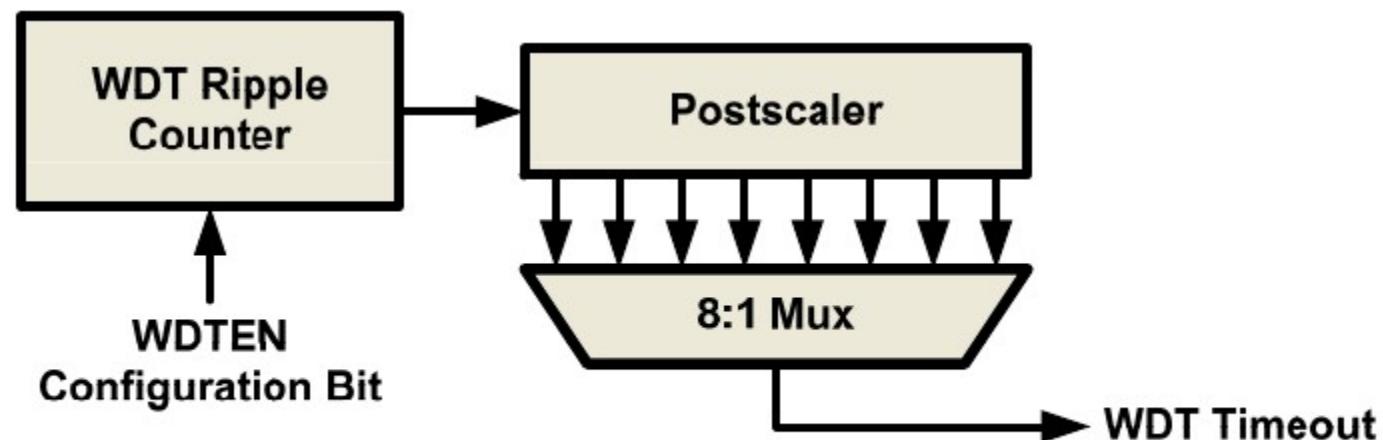
El procesador puede ser puesto en Modo Bajo consumo por medio de la ejecución de una instrucción SLEEP

- El oscilador del sistema es detenido
- El estado del procesador es mantenido (diseño estático)
- Watchdog timer continua funcionando, si esta habilitado
- Minima corriente de mantenimiento ($0.1 - 2.0\mu A$ typical)

Eventos que despiertan al procesador del modo SLEEP	
MCLR	Pulso sobre el MCLR (pulled low)
WDT	Watchdog Timer llegó a final de cuenta
INT	INT sobre el pin de interrupción
TMR1	Interrupción del Timer 1 (or also TMR3 on PIC18)
ADC	Interrupción por final de conversión del A/D
CMP	Interrupción por cambio de la salida del Comparador
CCP	Evento en la entrada de captura
PORTB	Interrupción por cambio en el PORTB
SSP	Interrupción en el Synchronous Serial Port (I ₂ C Mode) Start / Stop Bit detect
PSP	Lectura /Escritura en el PSP

Watchdog Timer

- Ayuda a recuperarse del mal funcionamiento del software
- Usa par funcionar su propio oscilador RC sobre el chip
- WDT es borrado po la instrucción CLRWDT
- Habilitado el WDT no puede ser desactivarse por software
el desborde del WDT resetea al dispositivo
- Período del time out esProgramable : 18ms to 3.0s typ
- Opera en modo SLEEP; sobre el time out, despierta la CPU



BOR –Brown Out Reset

- **Cuando se produce una variación del voltaje, Resetea al dispositivo**
- **Previene operaciones erráticas o inesperadas**
- **Elimina la necesidad de un circuito BOR externo**



PBOR – Programmable Brown Out Reset

Configuración opcional (seteado en la programación)

- No puede ser habilitado/ deshabilitado por software

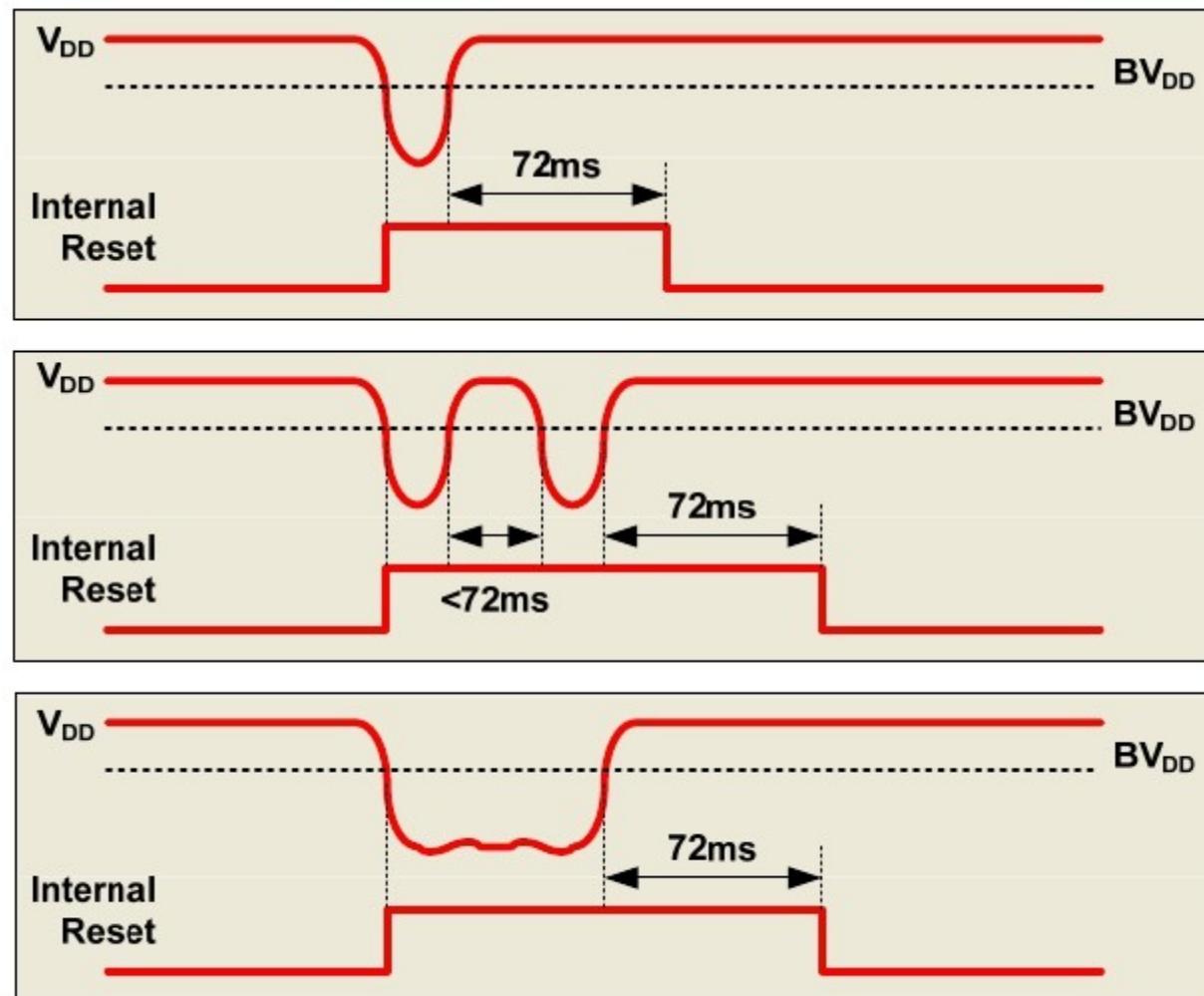
Cuatro puntos seleccionables BV_{DD} :

- 2.5V – Minimum V_{DD} for OTP PICmicro® MCUs
- 2.7V
- 4.2V
- 4.5V

Para otros thresholds, usar un supervisor de CPU externo (MCP1xx, MCP8xx/TCM8xx, or TC12xx)

(P)BOR – Brown Out Reset

Mantiene al PIC® MCU en reset hasta ~72ms despues que V_{DD} subió por encima del valor de threshold



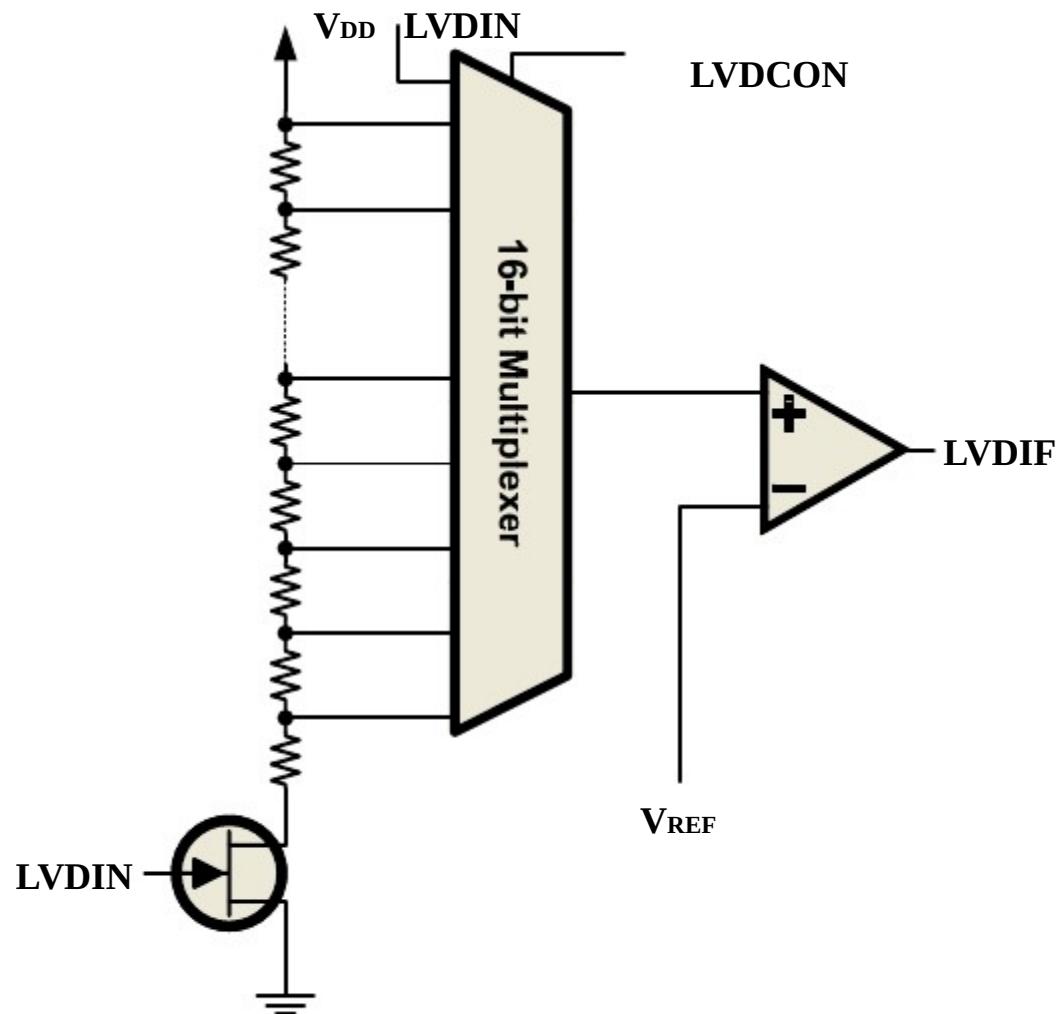
PLVD – Detector de Bajo Voltaje Programable

Es mejor que un
brown out

16 puntos
seleccionables:

- 1.8V up to 4.5V
in 0.1 to 0.2V
steps
- External analog
input

conectado al V_{REF} Interno



Programación Serie en Circuito ICSP™

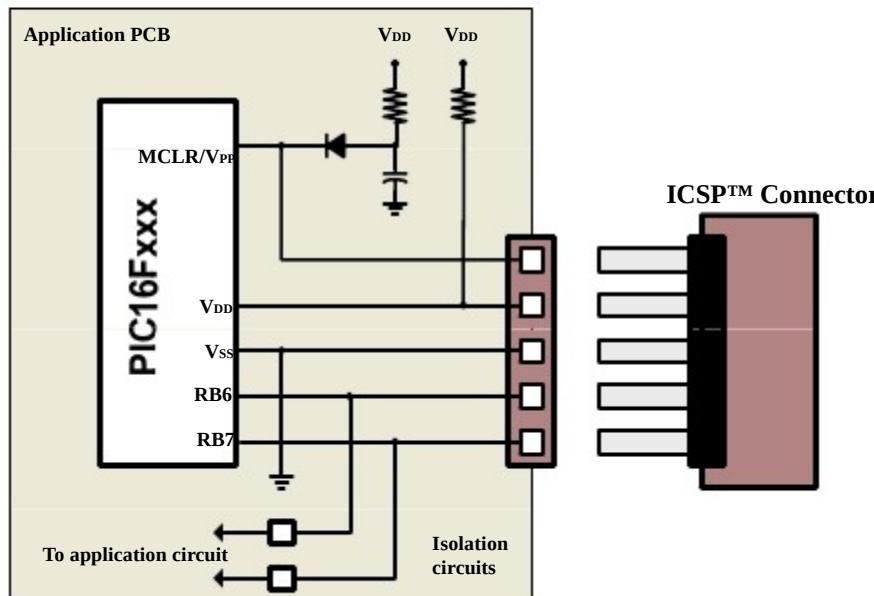
Solo requiere 2 pines para ser programado

Conveniente para hacer programación en sistemas

- Calibration Data
- Serialization Data

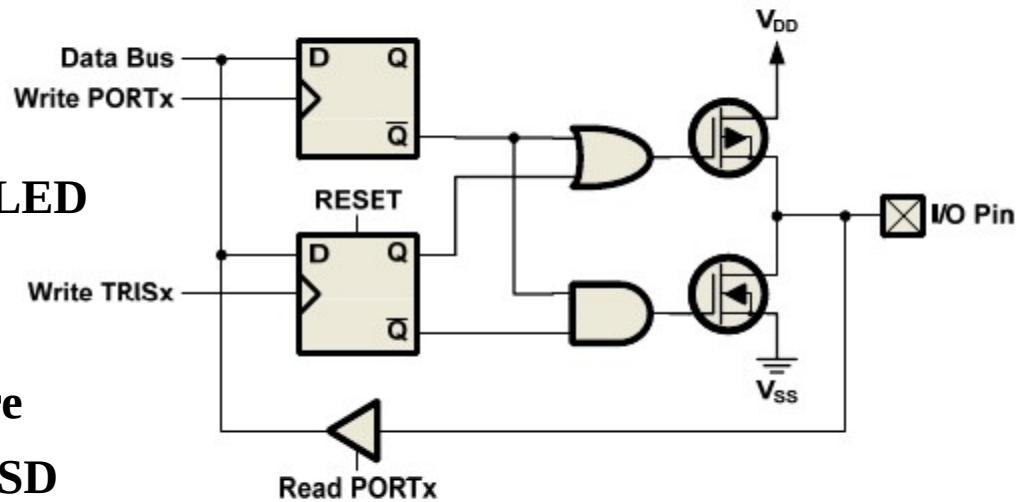
Soportado por MPLAB® PM3 & ICD2

Pin	Function
V _{PP}	Programming Voltage = 13V
V _{DD}	Supply Voltage
V _{SS}	Ground
RB6	Clock Input
RB7	Data I/O & Command Input

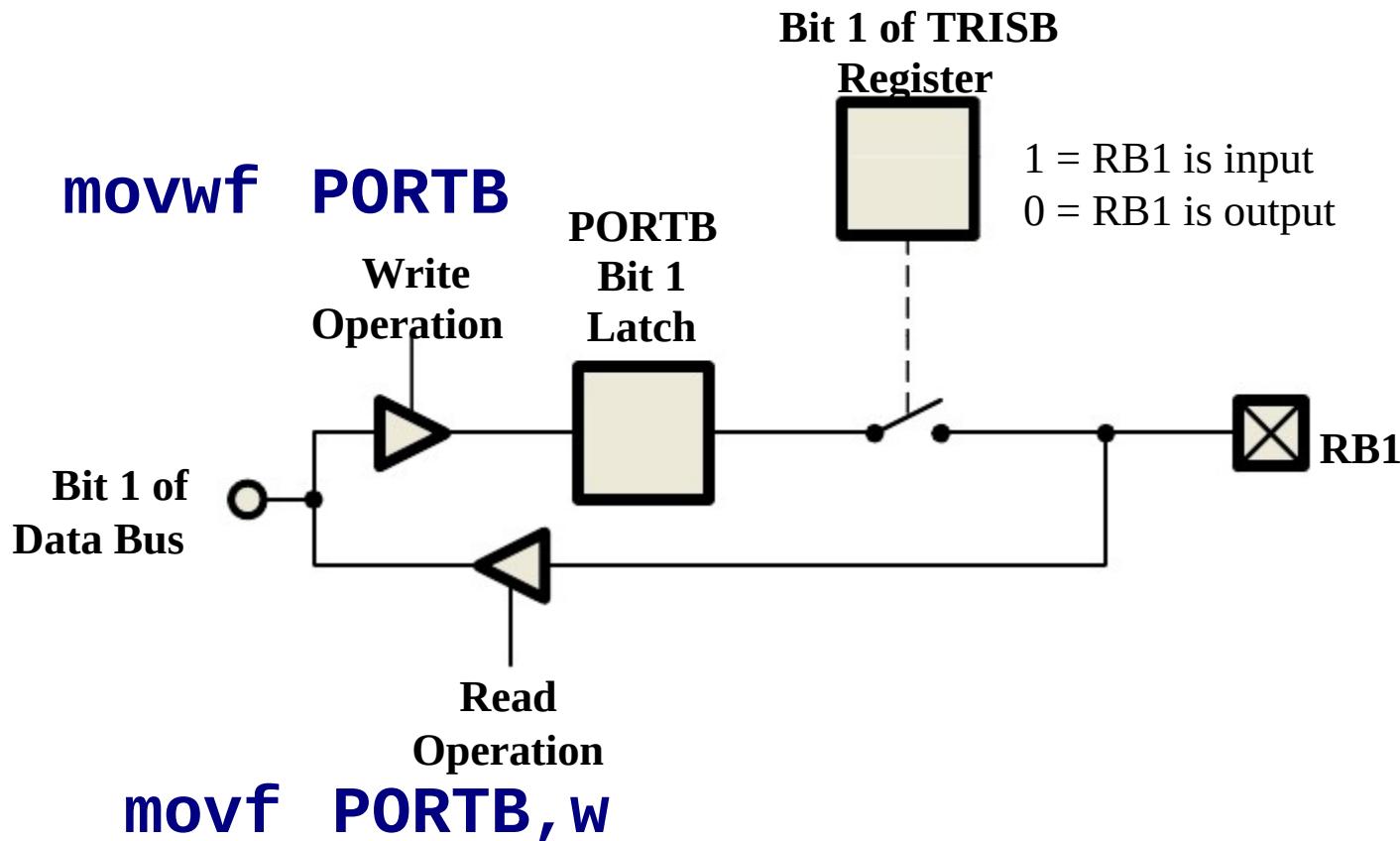


I/O Ports

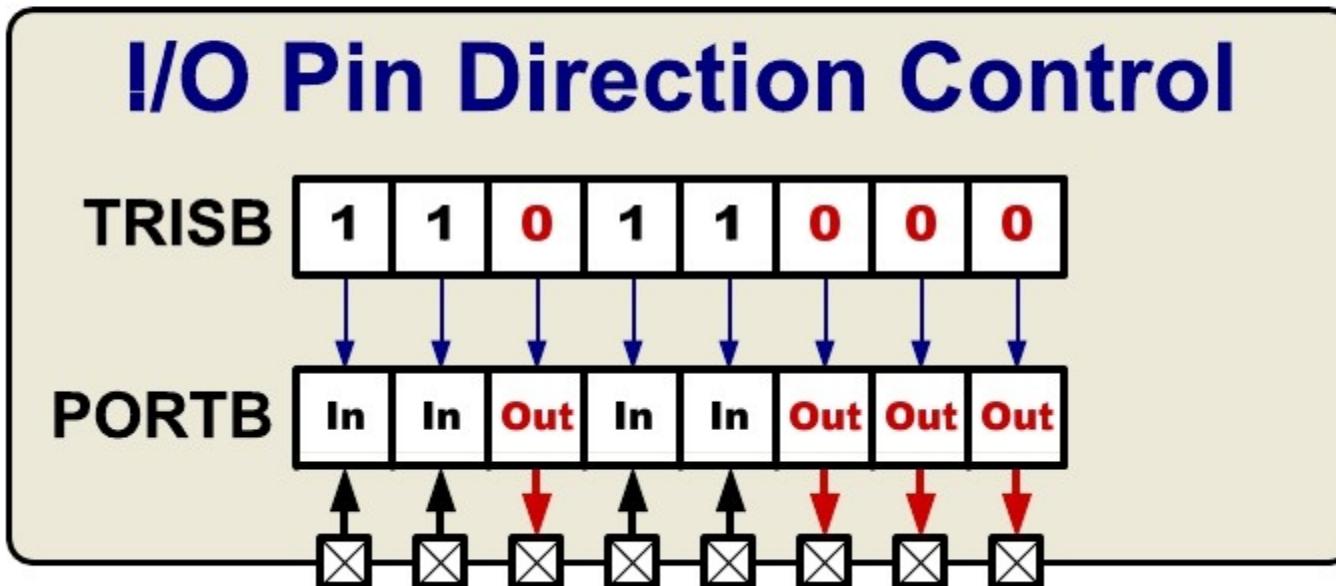
- Alta capacidad de corriente
- Pueden excitar directamente un LED
- Directa manipulación de bits
- Cada Pin puede ser direccionado independientemente por software
- Todo los pines tiene protección ESD
- Pin RA4 es open drain
- Todos los pines I/O por default son entradas(Alta impedancia) sobre el arranque
- Todos los pines estan multiplexados sobre entradas analogicas sobre el arranque (de los dispositivos que las tengan)



Pin I/O Diagramma Conceptual



I/O Ports



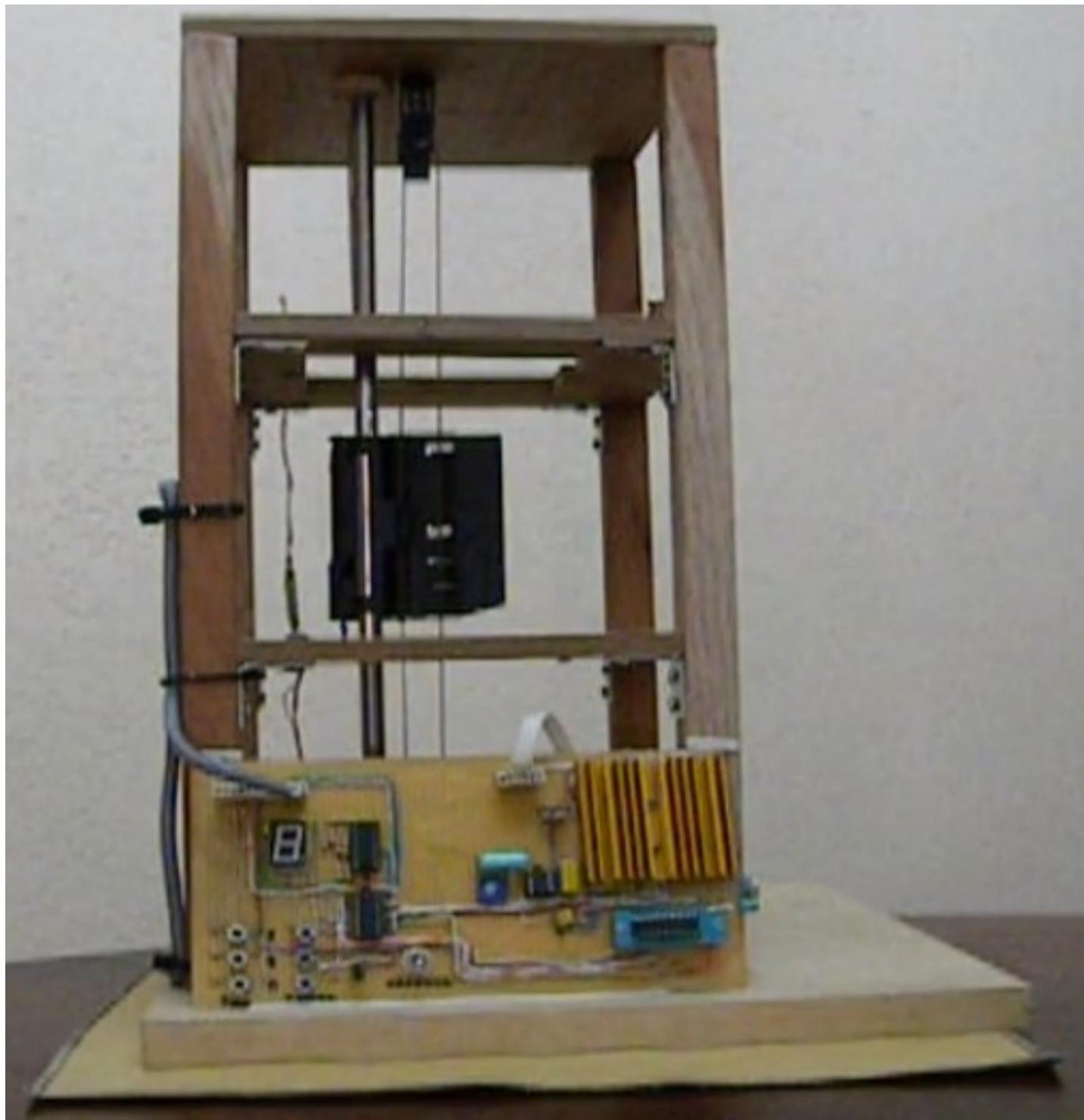
- Bit n en TRISx controla la dirección del dato Bit n en el PORTx
- 1 = Entrada, 0 = Salida

Arquitectura de Computadoras

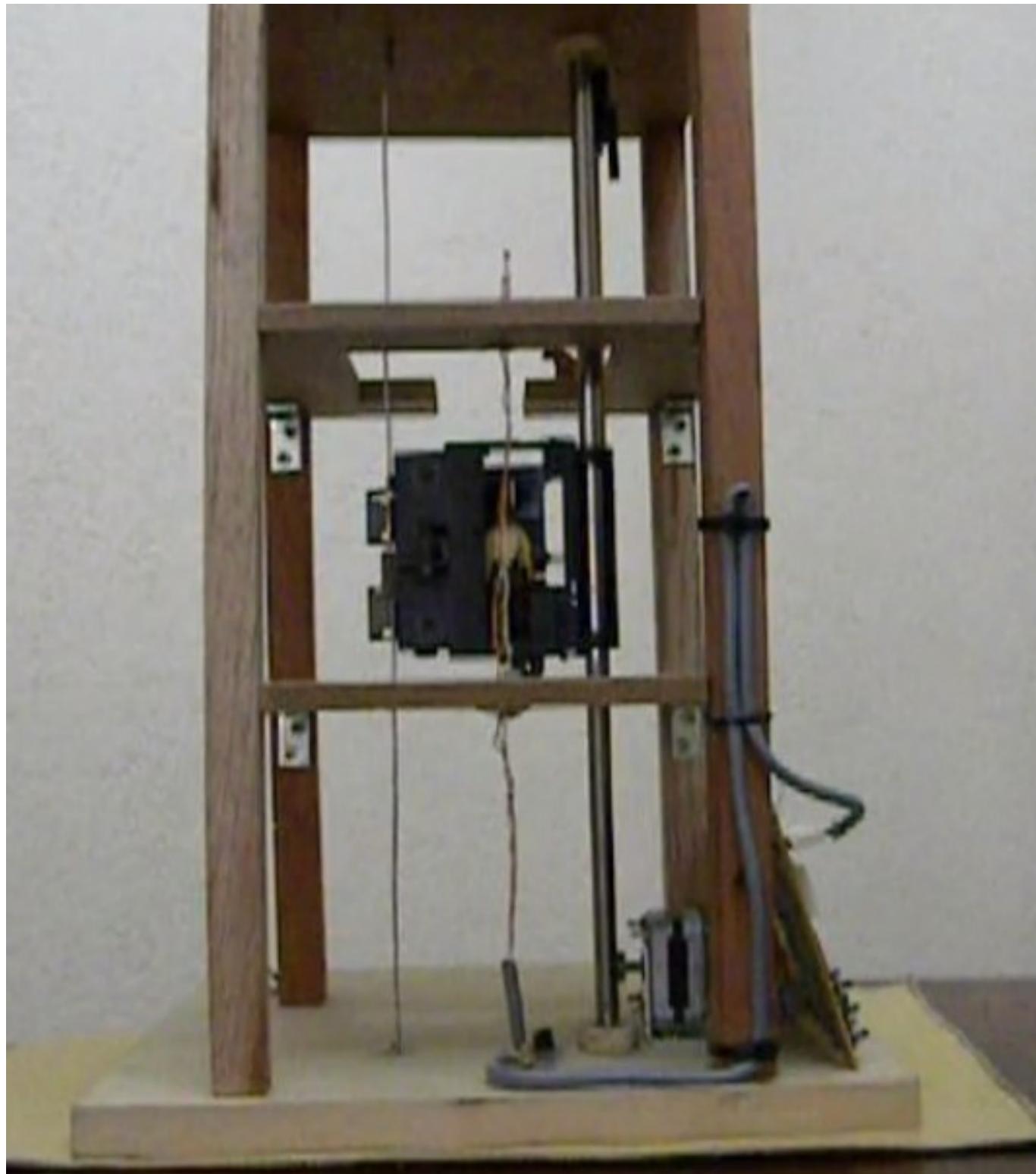
Trabajo Teórico - Práctico

Desarrollo de Ascensor

Viernes 15/11







Ascensor – Plaqueta

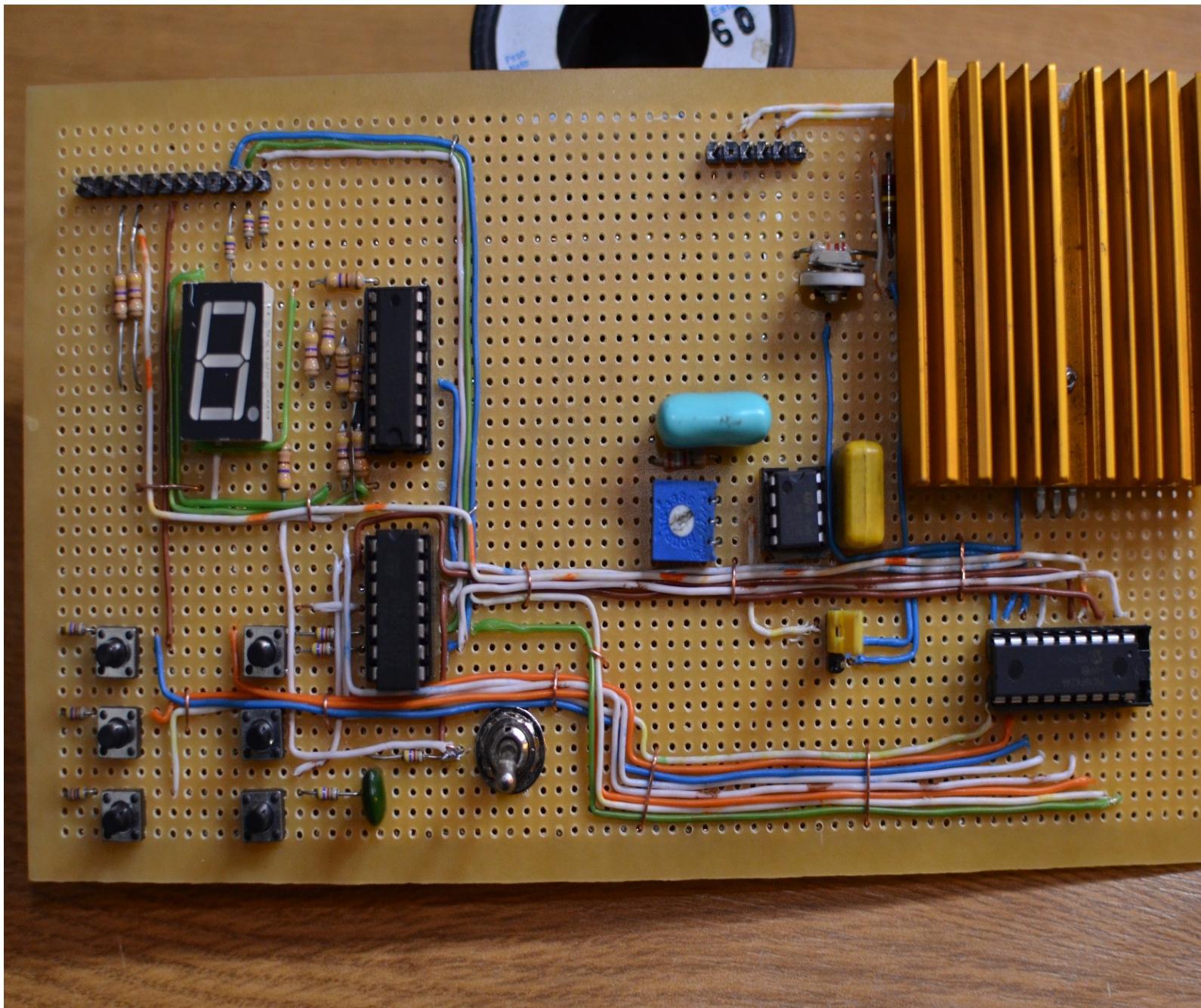


Diagrama Eléctrico Ascensor

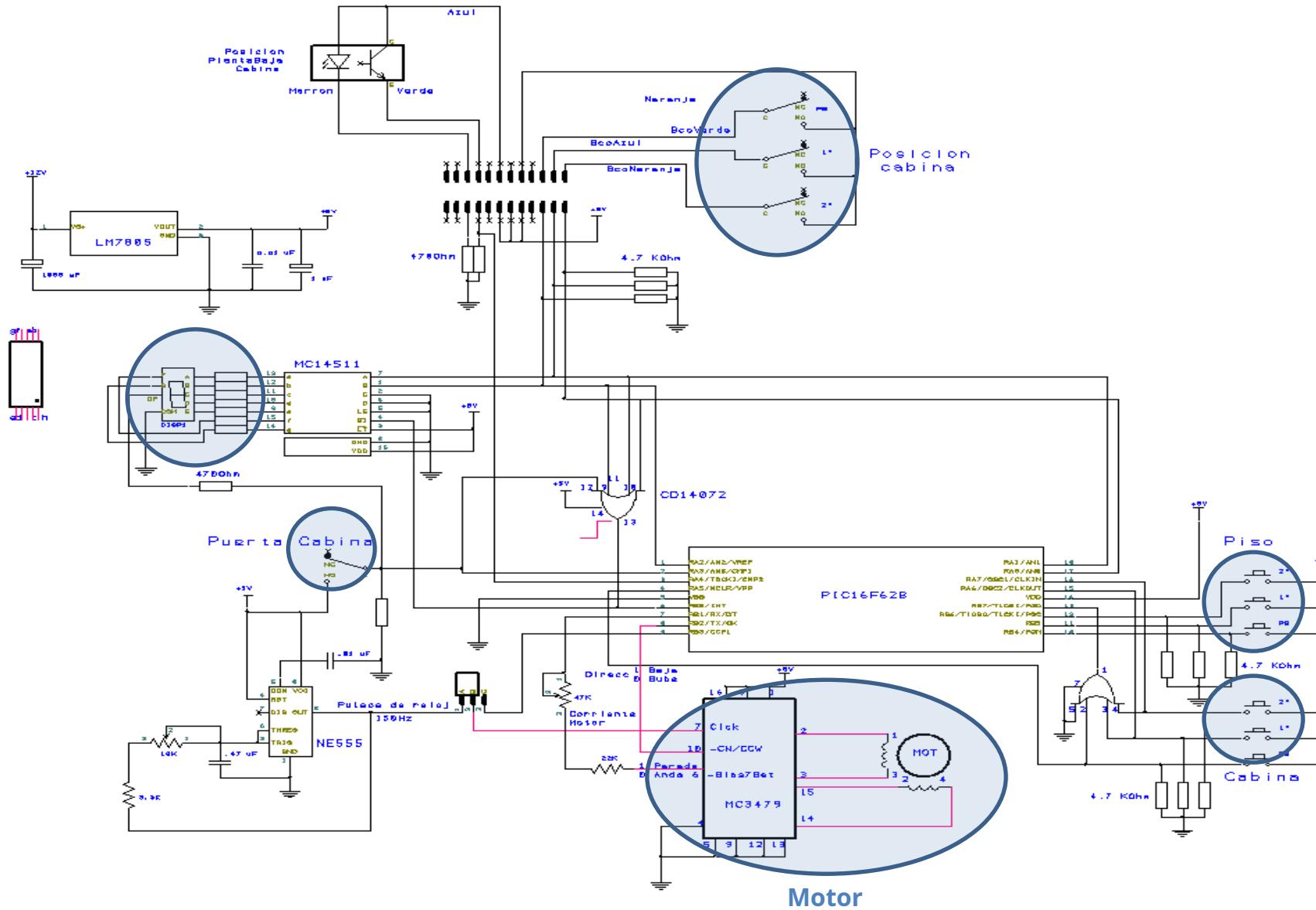
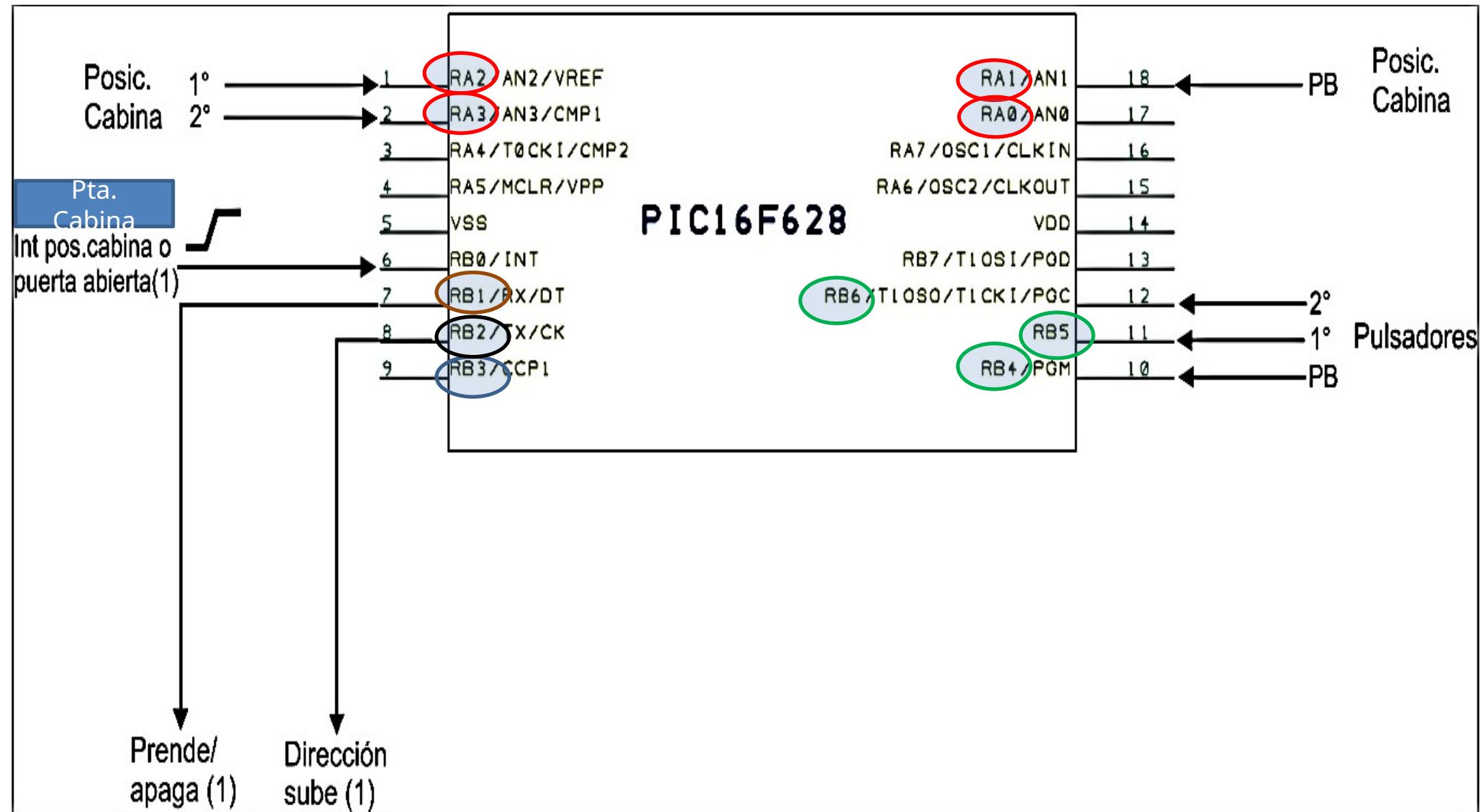
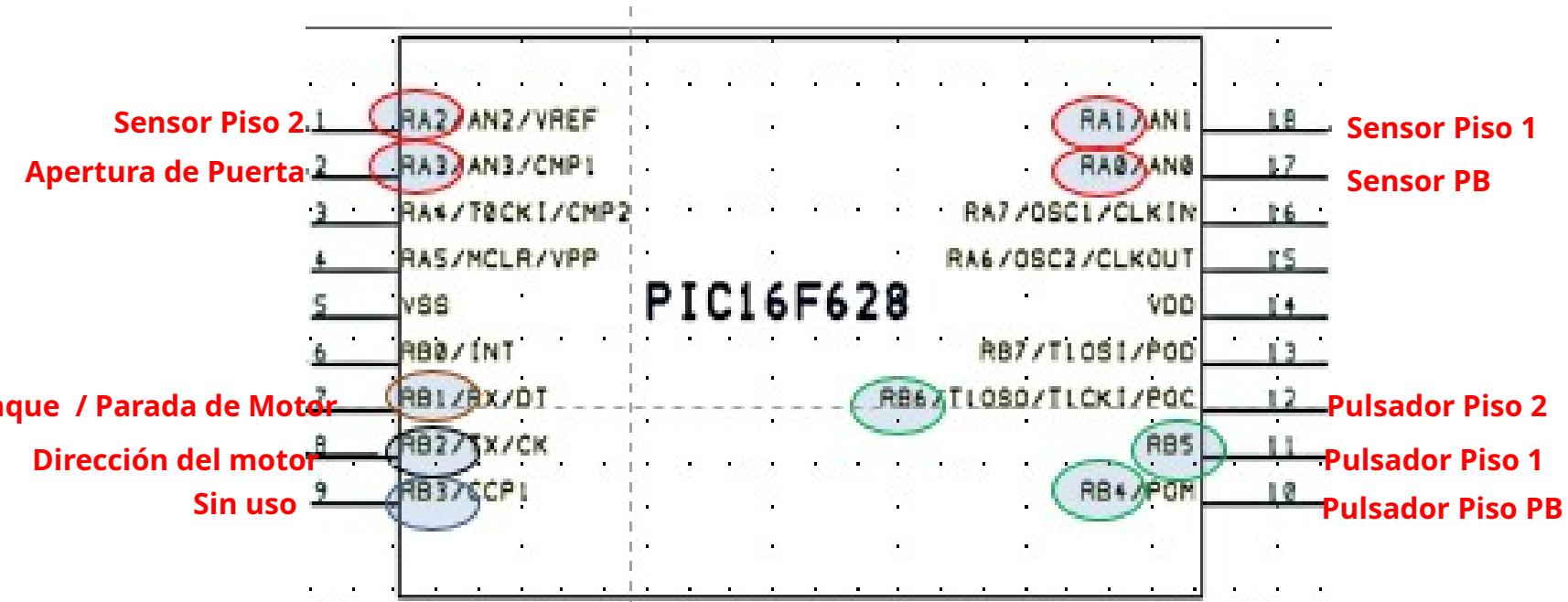


Diagrama de Utilización



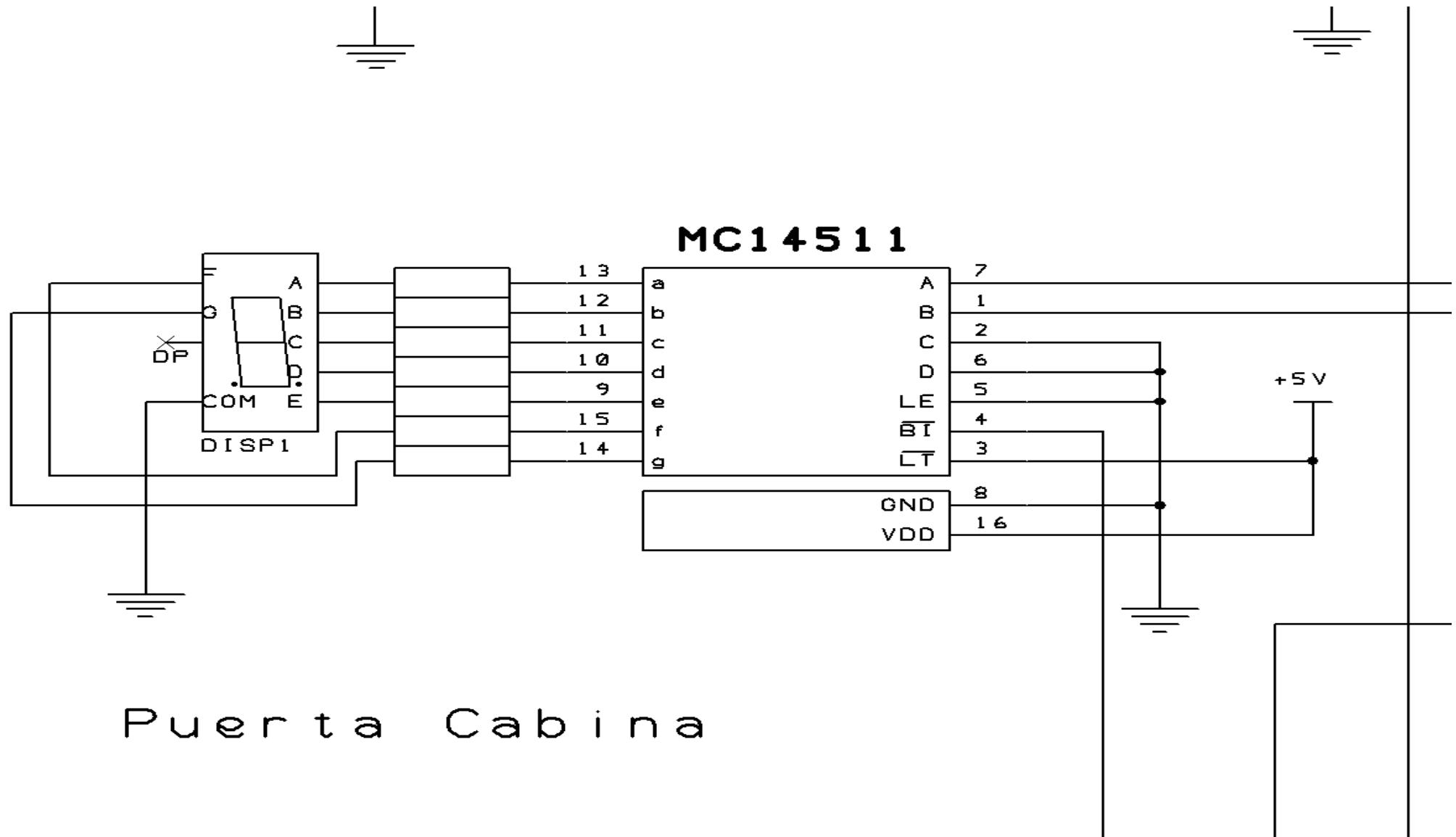


Apertura de Puerta 0 Cerrada 1 Abierta

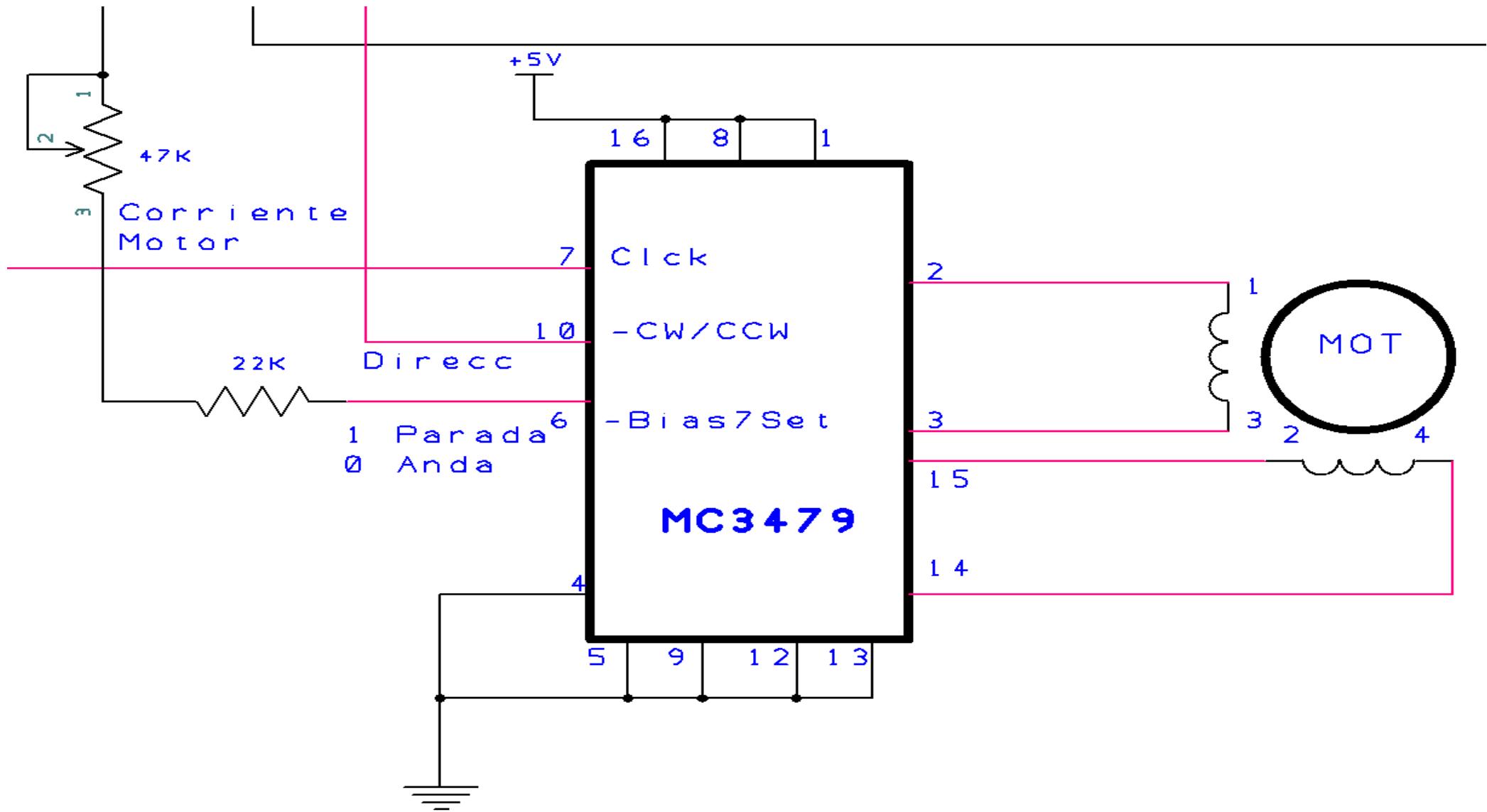
Arranque / Parada de Motor 0 Arranca 1 Parada

Dirección del motor 1 Sube 0 Baja

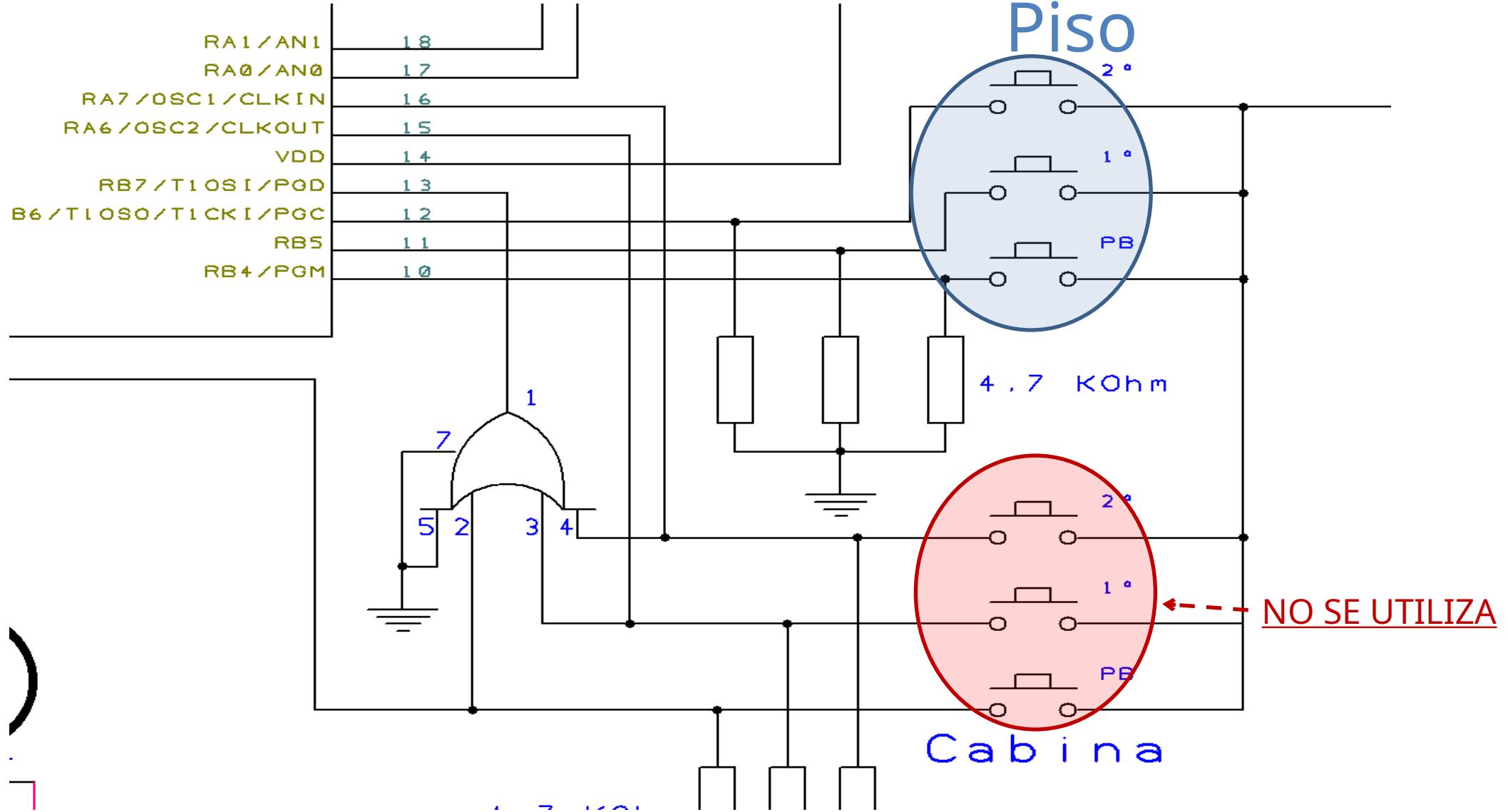
Display 7 segmentos



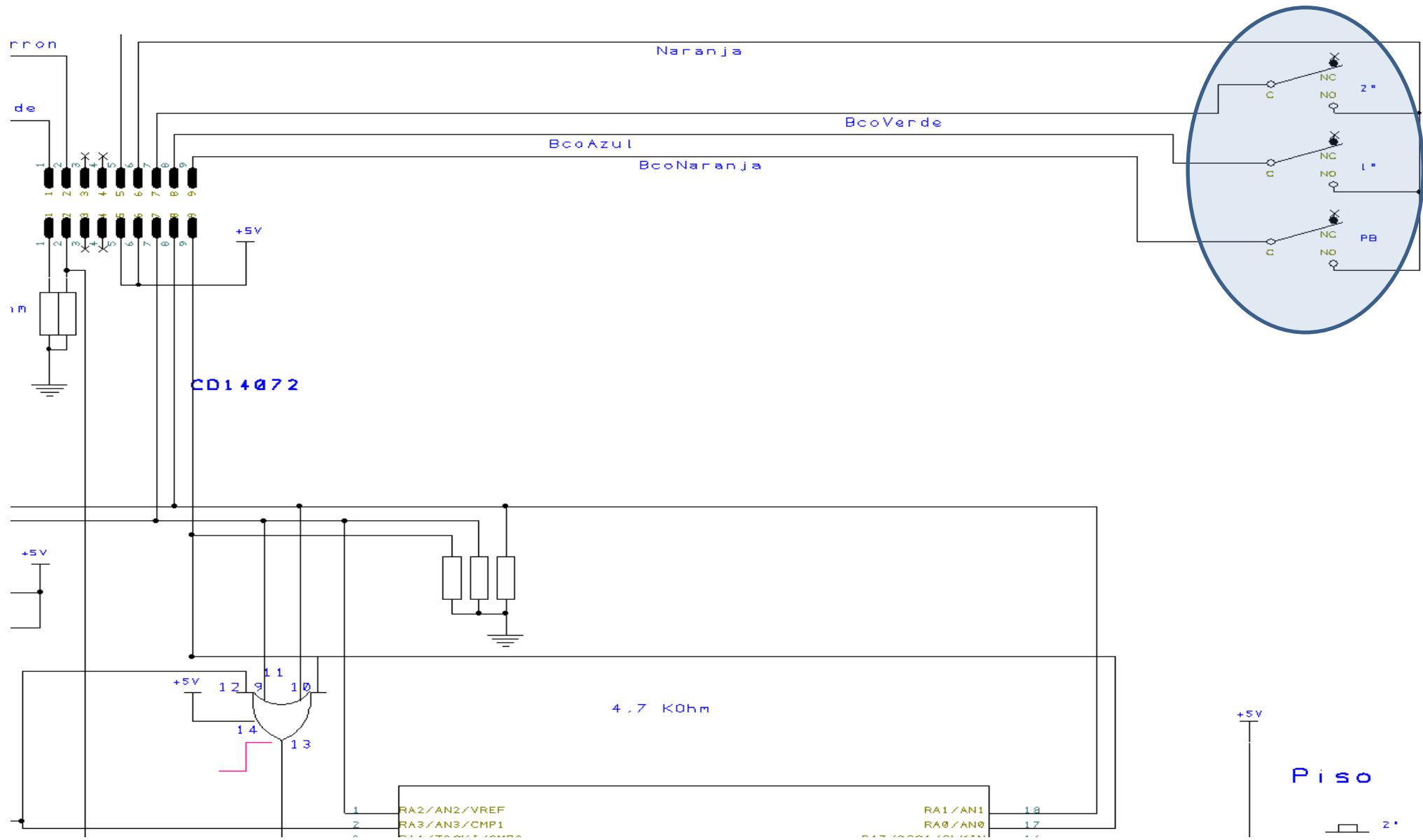
Conexión Motor



Pulsores



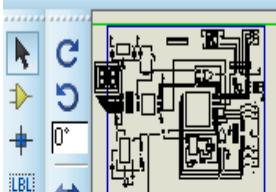
Sensores de Piso





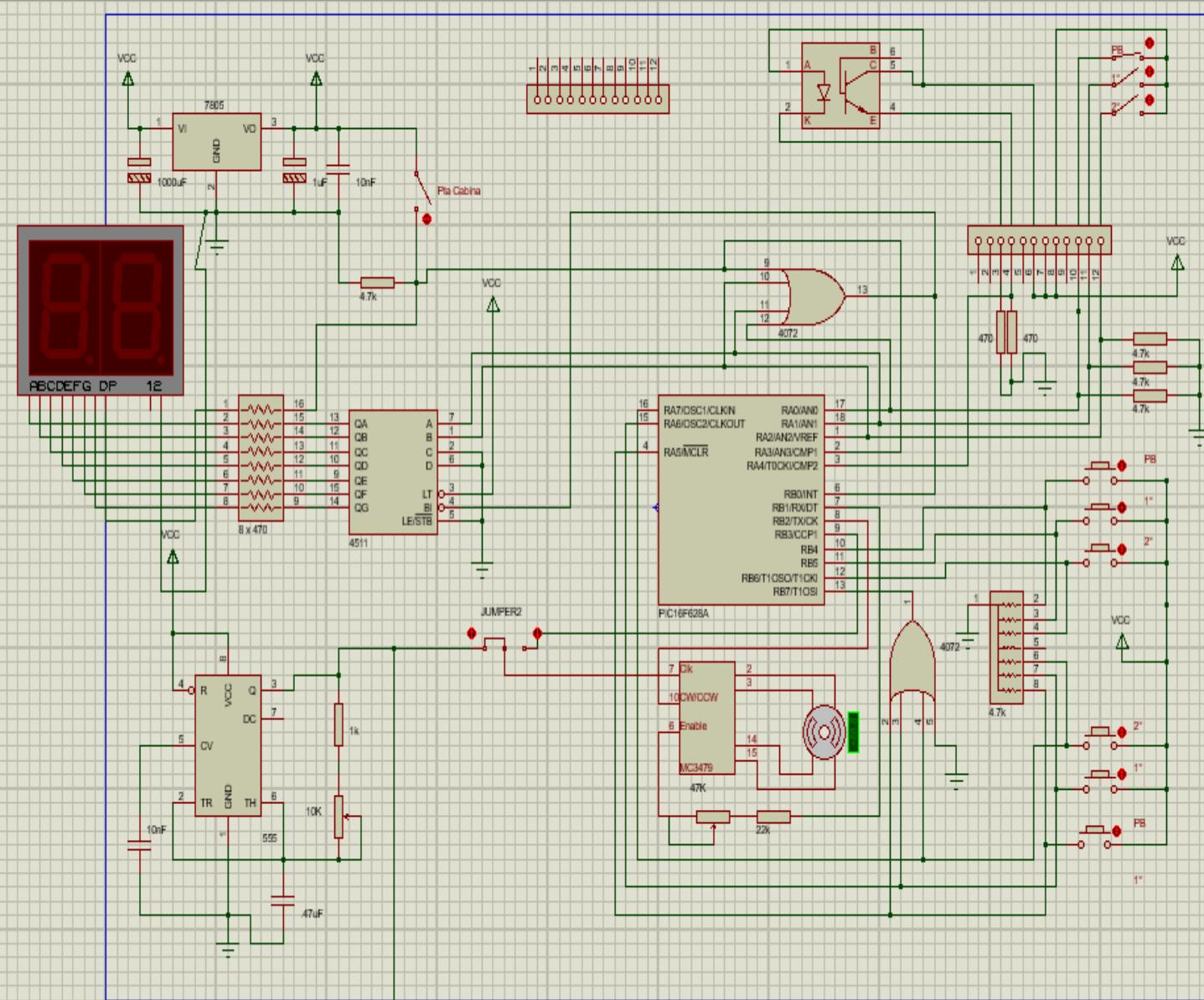
Schematic Capture x

PCB Layout x



PL DEVICES

4N25
7SEG-MPX1-CC
7SEG-MPX2-CC
555
3009Y-1-103LF
4072
4511
7805
66226-012
BUTTON
CAP
CAP-ELEC
JUMPER2
L297
MONORES1N
MOTOR-BISTEPPER
PIC16F628A
RES
RES8SIPB
RES16DIPIS
SW-SPST-MOM
[4072]



No Messages

Root sheet 1

-6100.0

-200.0 th



15:06

Diagrama de Flujo

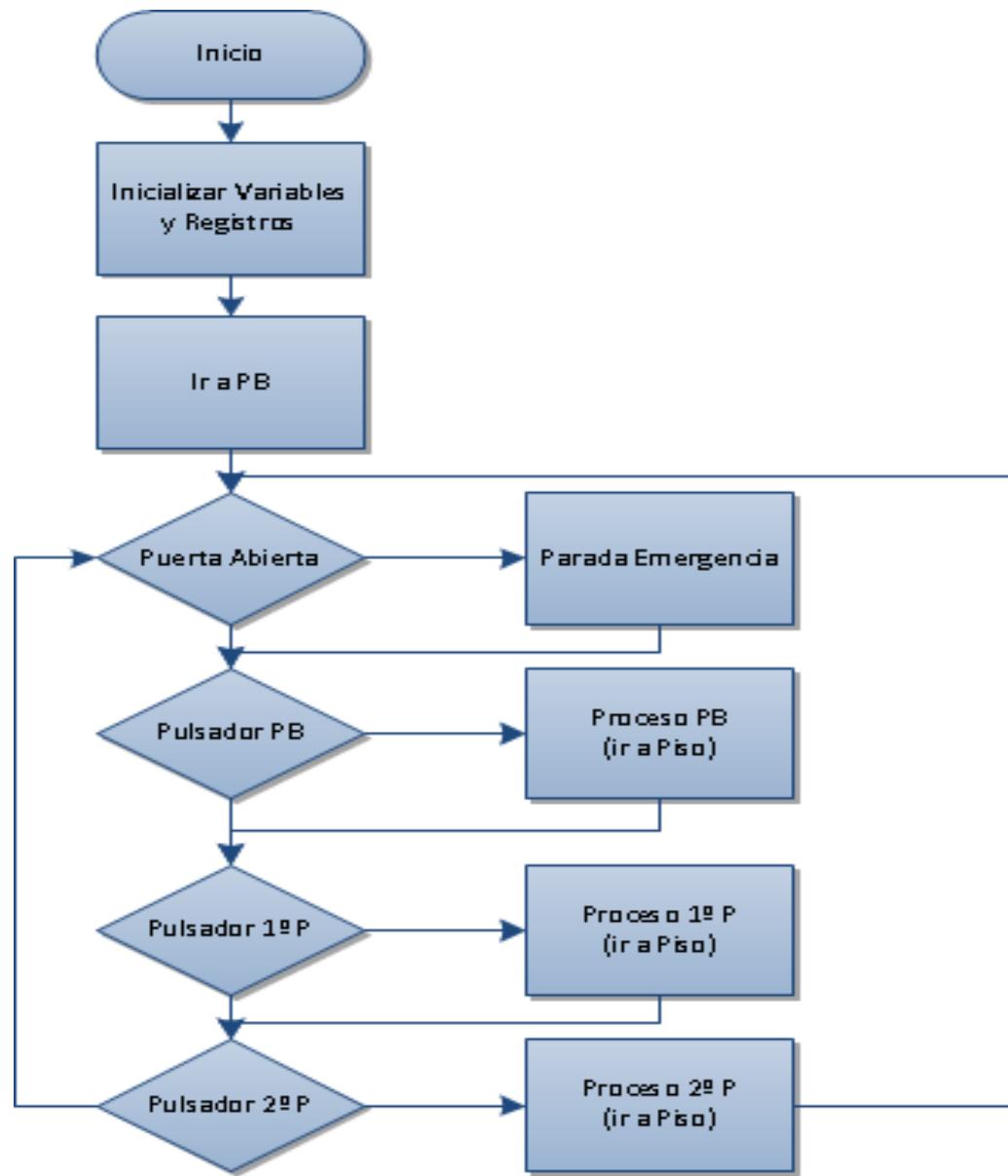


Diagrama de Flujo



Diagrama de Flujo

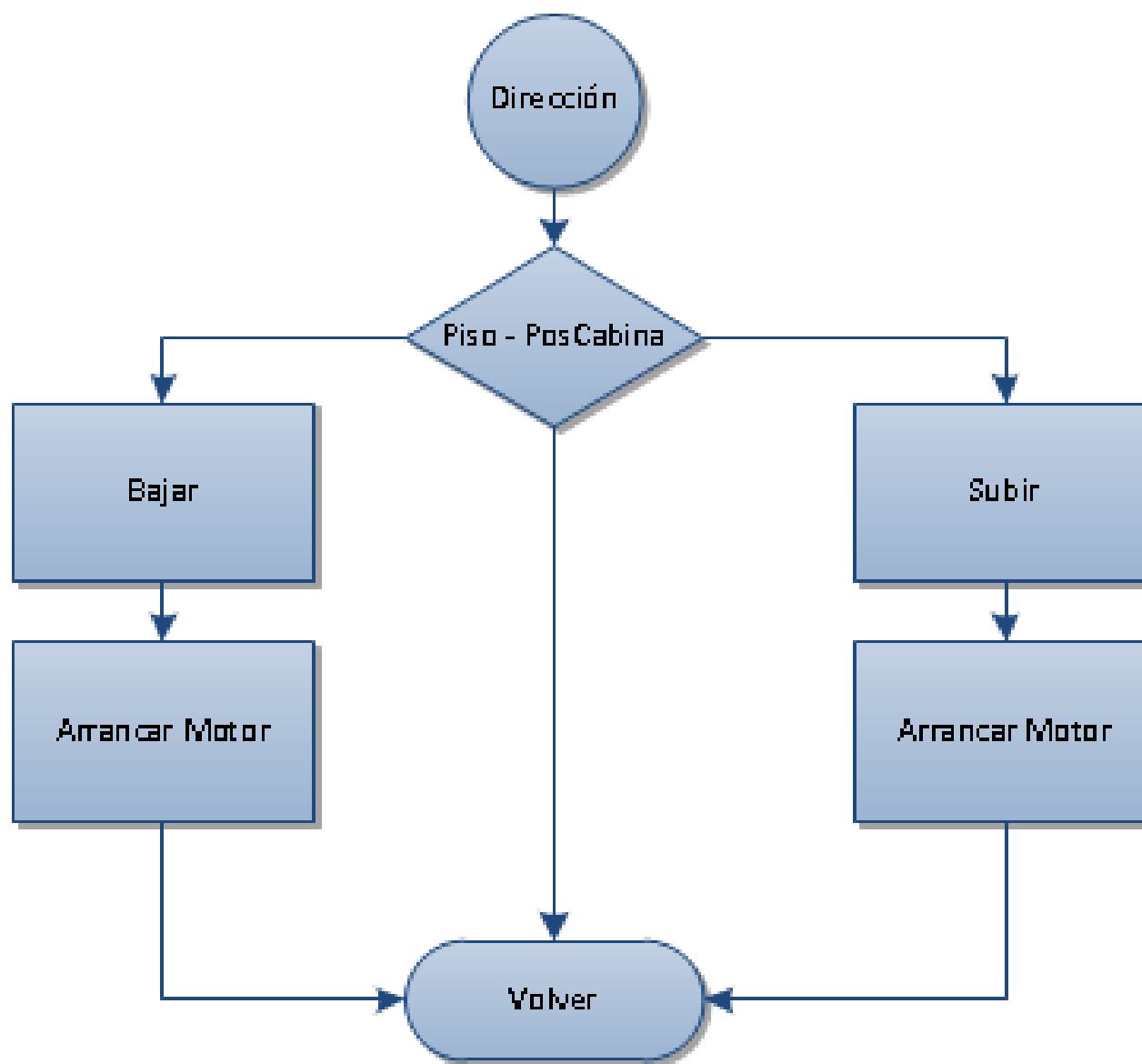
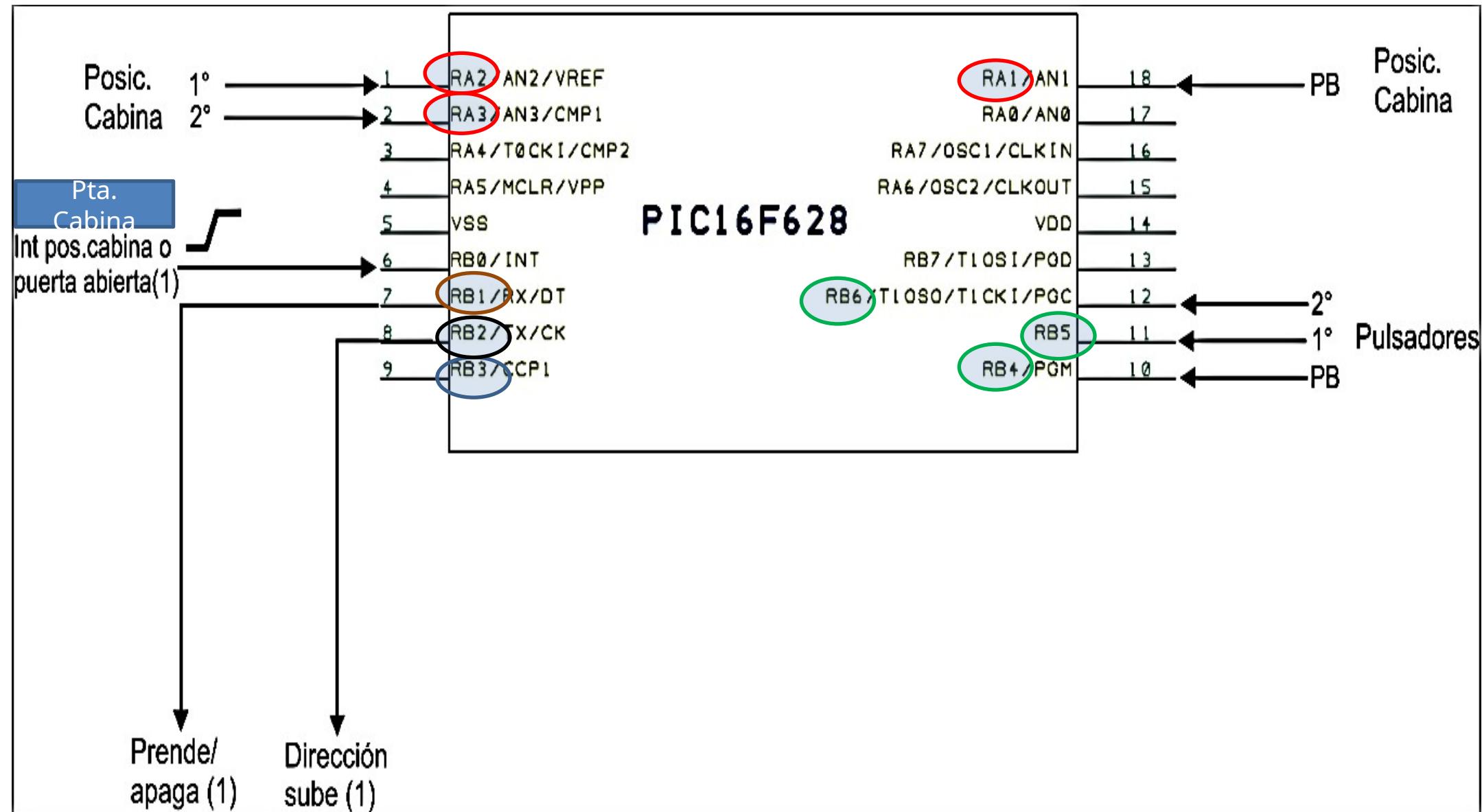


Diagrama de Utilización



Registros del PICF16

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind. ¹	Dir. Ind. ¹	80h
01h	TMRO	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	-	-	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2 ¹	89h
0Ah	PCLATH	PCLATH	8Ah
OBh	INTCON	INTCON	8Bh

Registros a utilizar

Banco 0										
Direcc	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
00h	INDF	Dirección de FSR (no es físicamente un registro)								
01h	TMRO	Contador/Temporizador de 8 bits								
02h	PCL	8 bits LSB del PC								
03h	STATUS	IRP	RP 1	RP 0	TO	PD	Z	DC	G	
04h	FSR	Puntero para el Direccionamiento Indirecto								
05h	PORTA	-	-	-	R44 TOCKI	R43	R42	R41	R40	
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0 INT	
07h	-	No implementado								
08h	EEDATA4	Registro de Datos EEPROM								
09h	EEADR	Registro de Direcciones EEPROM								
0Ah	PCLATH	-	-	-	5 bits MSB del PC					
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	

Sensor de PB

Posición de Cabina
RA0 (PB)
RA1 (1)
RA2 (2)

Puerta Cabina
Abierta (1)
Cerrada (0)

Motor Prendido (0)
Apagado (1)

Dirección del motor
Sube (0) Baja (1)

Pulsadores de Pared
RB4 (PB)
RB5 (1)
RB6 (2)

Registros a utilizar

Banco 1

Direcc	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0						
80h	INDF	Dirección de FSR (no es físicamente un registro)													
81h	OPTION	RBPU	INTE	TOCS	TOSE	PSA	PS2	PS1	PS0						
82h	PCL	8 bits LSB del PC													
83h	STATUS	IRP	RP1	RP0	TO	PB	Z	DC	C						
84h	FSR	Puntero para el Direccionamiento Indirecto													
85h	TRISA	-	-	-	Dirección de datos del Puerto A										
86h	TRISB	Dirección de los datos del Puerto B													
87h	-	No Implementado													
88h	ECON1	-	-	-	EEIF	WRR	WEN	WR	RD						
89h	ECON2	2º REGISTRO DE Control de la EEPROM													
8Ah	PCLATH	-	-	-	5 bits MSB del PC										
8Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF						

Entregables

Para cada sub-ejercicio se debe desarrollar y entregar

- Diagrama de flujo
- Proyecto completo en MPLAB
- Opcional: Simulación en Proteus
- Todo esto debe entregarse vía mail como se informa en el documento del trabajo práctico

Programa Ascensor

```
;*****
;* DECLARACIONES DE PROCESADOR, VARIABLES Y CONFIGURACION
;*****  
#include<p16f628a.inc> ; Archivo de reconocimiento de registros  
especiales y sus bits.  
Piso equ 0x20 ; Declaración de registro para la ubicación de la cabina  
Cabina equ 0x21 ; Declaración de registro para ubicación de la cabina  
intmem equ 0x70 ; Declaración de registro para estado actual del motor y  
dirección  
  
_config 3f10 ; Oscilador INTRA I/O  
 ; Watchdog OFF  
 ; Power Up TIM ON  
 ; Master Clear DISABLE  
 ; Brown Out DISABLE  
 ; Data Read DISABLE Protege datos NO CAMBIAR NUNCA  
 ; Code Protect OFF Protege datos NO CAMBIAR NUNCA
```

Programa Ascensor

```
;*****
;* INICIO DEL PROGRAMA
;*****  
  
;*****  
inicio          org      0x05  
                  goto    inicio  
                  clrf    porta      ;Borra registros del porta  
                  ;RA4= Sensor PB (0= PB; 1= No llegó)  
                  ;RA3= Puerta Cabina Abierta (1) Cerrada (0)  
                  ;RA2= Posición Cabina 2°  
                  ;RA1= Posición Cabina 1°  
                  ;RA0= Posición Cabina PB  
                  movfw   portb     ;Mueve el contenido del portb a W  
                  ;RB7= Posible Interrupción Por Cambio en Pulsadores  
de  
                  ;Cabina (No se usa)  
                  ;RB6= Pulsador 2°  
                  ;RB5= Pulsador 1°  
                  ;RB4= Pulsador PB  
                  ;RB3= No se usa Futuro Motor Paso a Paso  
                  ;RB2= Dirección de motor baja (1) sube (0)  
                  ;RB1= Prende (0) / Apaga el motor (1)  
                  ;RB0= Interrupción Posic. Cabina / Puerta abierta (NO  
SE USA)  
                  clrw      ;Se asegura que el registro W se encuentre borrado
```

Programa

	movwf	portb	;Mueve el contenido del portb a W ;RB7= No se usa ;RB6= Pulsador 2° ;RB5= Pulsador 1° ;RB4= Pulsador PB ;RB3= No se usa ;RB2= Dirección de motor sube (1) ;RB1= Prende (0) / Apaga el motor (1) ;RB0= Puerta abierta (1)
	clrf	piso	;Borra dirección de memoria Piso
	clrf	cabina	;Borra dirección de memoria ubicación de cabina
	movlw	0x07	;Identifica a todas patas como E/S Mueve valor 0 al bit 7 de W (Se asegura que el bit 7 del portb tiene 0)
banco 1	movwf	cmcon	;porta como i/o
	bsf	status,rp0	;Pone en 1 el bit RP0 (selector de banco de registros) indica
entradas	movlw	0xff	;Mueve al registro W el valor FF-> W= 11111111 (Para que?)
	movwf	trisa	;Mueve el contenido de W (11111111) al registro TRISA porta
	movlw	0xf1	;Mueve al registro W el valor F1-> W= 11110001
	movwf	trisb	;Pone en portb 1,2 y 3 como salidas
	bcf	status,rp0	Pone en 0 el bit RP0 (selector de banco de registros) indica
banco 0	movfw	portb	;Mueve el contenido del registro portb al registro W
EN MP Lab	movlw	0x0e	;ESTAS INSTRUCCIONES SON PARA VER COMPORTAMIENTO (Recurso de Programador)
EN MP Lab	movwf	portb	;ESTAS INSTRUCCIONES SON PARA VER COMPORTAMIENTO

Programa Ascensor

```
;*****  
;* ENVIA LA CABINA A PB EN EL PRIMER CICLO DE TRABAJO  
;*****  
apb          btfsc    porta,0      ;Verifica que el bit 0 del porta sea = 1 (Verifica si Cabina  
está PB)  
                           ;Si es =1 va a la etiqueta SIGUE si=0 Salta el GOTO  
                           ;El bit 0 de porta controla que la cabina se encuentre en  
PB(1)          goto     sigue        ;si, seguir  
                           ;No Esta en PB Pone el bit 2 del puerto b = 1  
                           ;El bit 2 del puerto controla la dirección del motor  
(1=abajo)      bsf      portb,2  
sigue         bcf      portb,1      ;anda motor  
                           goto     apb       ;bucle hasta que llega a PB  
                           bsf      portb,1      ;para motor  
                           movfw   porta      ;Pone en PORTA 00000111  
                           andlw   0x07      ;Pone en PORTA 00000111  
                           movwf   cabina     ;leo porta y cargo posicion de cabina  
nada          call    ptaabierta   ;LOGICA PRINCIPAL DEL PROGRAMA  
                           call    pulsadores  ;LOGICA PRINCIPAL DEL PROGRAMA  
                           goto    nada       ;LOGICA PRINCIPAL DEL PROGRAMA
```

Programa Ascensor

```
;*****  
;*CONTROLA LA APERTURA DE LA PUERTA  
;*****  
  
Ptaabierta          btfss    porta,3      ;puerta de cabina abierta?  
                      return  
                      call     tiempo       ;no, salga del proceso  
                      btfss    porta,3      ;antirrebote de contactos  
                      return  
                      movfw   portb        ;sigue abierta?  
                      movwf   intmem      ;no, falsa alarma, salga del proceso  
                      return  
                      ;si, puerta abierta  
                      ;guardar estado actual del motor y direcc  
  
emergencia         bsf      portb,1      ;parar el motor  
                      btfsc   porta,3      ;esperar a que cierren la puerta  
                      goto    emergencia  
                      movfw   intmem      ;recompomer el estado anterior  
                      movwf   portb        ;volver al punto de llamada  
                      return
```

Programa Ascensor

```
;*****  
;* RUTINA DE PULSADORES  
;*****  
pulsadores    btfsc    portb,4          ;pruebo pulsador de PB accionado  
                 call     tiempo           ;espero tiempo de rebotes del pulsado  
                 btfsc    portb,4          ;confirmo accionamiento de pulsador  
                 call     PB               ;pulsador de PB confirmado, ir al proceso  
                 btfsc    portb,5          ;espero tiempo de rebotes del pulsado  
                 call     tiempo           ;pulsador de 1 piso confirmado, ir al proceso  
                 btfsc    portb,5          ;espero tiempo de rebotes del pulsado  
                 call     primero          ;pulsador de 2 piso confirmado, ir al proceso  
                 btfsc    portb,6          ;ningun pulsador accionado, volver al progr.  
                 call     tiempo           ;espero tiempo de rebotes del pulsado  
                 btfsc    portb,6          ;ningun pulsador accionado, volver al progr.  
                 return
```

Programa Ascensor

```
;*****  
;* PROCESO EN PLANTA BAJA  
;*****  
PB          swapf   portb,w      ;paso los bits de las puertas a menor peso  
            andlw   0x07        ;aseguro el valor  
            movwf   piso         ;para guardarla en memoria de piso  
            call    direccion    ;llamo al proceso que fijara la direccion  
  
otrapb       call     ptaabierta   ;alguien abrio la puerta?  
            btfss   porta,0         
            goto   otrapb        ;pruebo hasta que la cabina llegue a PB  
            bsf    portb,1        ;llego, paro el motor  
            movlw   0x01          
            movwf   cabina        ;cargo la memoria con la posicion de cabina  
            clrf    piso         ;borro piso para proxima llamada  
            return
```

Programa Ascensor

```
;*****
;* PROCESO EN PRIMER PISO
;*****
```

primero

	swapf	portb,w	;paso los bits de las puertas a menor peso
	andlw	0x07	;aseguro el valor
	movwf	piso	;para guardarla en memoria de piso
	call	direccion	;llamo al proceso que fijara la direccion

otrap

	call	ptaabierta	;alguien abrio la puerta?
	btfss	porta,1	
	goto	otrap	;pruebo hasta que la cabina llegue a PB
	bsf	portb,1	;llego, paro el motor
	movlw	0x02	
	movwf	cabina	;cargo la memoria con la posicion de cabina
	clrf	piso	;borro piso para proxima llamada
	return		

Programa Ascensor

```
;*****  
;* PROCESO EN SEGUNDO PISO  
;*****  
segundo          swapf   portb,w  
                  andlw   0x07  
movwf    piso  
call      direccion  
  
otras           call     ptaabierta  
                  btfss   porta,2  
                  goto    otras  
                  bsf     portb,1  
                  movlw   0x04  
                  movwf   cabina  
                  clrf    piso          ;Mismo proceso que para PB  
                  return
```

Programa Ascensor

```
;*****
;* PROCESO PARA DETERMINAR LA DIRECCION PARA MOVER LA CABINA
;*****
```

direccion	movfw cabina	
	subwf piso,w	;Se resta Posición de CABINA - Sensor de PISO activado
	btfsc status,z	;si la resta da cero la cabina y el pulsador
	return	;da cero, corresponden al mismo piso, volver
	btfss status,c	;Se verifica el valor del bit de Carry en STATUS
	goto baja	;Por negativo (bit Carry = 0 hay que subir
	goto sube	;Por Positivo (bit de Carry = 1 hay que bajar
sube	bcf portb,2	;direccion sube
	bcf portb,1	;motor anda
	return	;vuelve
baja	bsf portb,2	;direccion baja
	bcf portb,1	;motor anda
	return	;vuelve

Programa Ascensor

```
;*****  
;CONTROL DE TIEMPO PARA REBOTES EN PULSADORES  
;*****  
tiempo      clrf    t1con  
              movlw   0xb1  
              movwf   tmr1h  
              movlw   0xdf  
              movwf   tmr1l           ; se configura a Timer1 para 20 mseg aprox  
              bsf     t1con,0         ;arranca temporizador  
  
mas        btfss   pir1,0       ;termino?  
              goto   mas          ;no, sigo esperando  
              bcf    pir1,0       ;si, borrar bandera indicadora de fin  
              bcf    t1con,0       ;parar contador  
              return  
end
```

Guía de los conceptos necesarios para la resolución el primer TP

Arquitectura de Computadoras,
2016

Descripción de la consigna

- El primer trabajo práctico de programación se divide en tres ejercicios, cada uno de los cuales contiene subdivisiones. A grandes rasgos, abordan las siguientes temáticas:
 - EJERCICIO 1: Configuración de puertos - Manejo de LEDs - Demoras sin Timer
 - EJERCICIO 2: Manejo de pulsadores – Interrupciones + Conceptos del Ejercicio 1
 - EJERCICIO 3: Manejo de Display – Uso del Timer 0 + Conceptos de los ejercicios anteriores

Entregables

Para cada sub-ejercicio se debe desarrollar y entregar

- Diagrama de flujo
- Proyecto completo en MPLAB
- Opcional: Simulación en Proteus
- Todo esto debe entregarse vía mail como se informa en el documento del trabajo práctico

Ejercicio 1: Demoras sin Timer

0

• delay_1ms		
•	movlw	
• .250		
•	movwf	
• CONT		
• loop	nop	1 ciclo
•	decfsz	
• CONT, f		1 ciclo
•	goto	
• loop		2 ciclos
•	return	
		<hr/>
		4 x 250

¿Cómo se calcula?

- Se considera que cada ciclo de máquina es de 1 microsegundo.
- Se busca en la hoja de datos cuántos ciclos dura cada instrucción.
- Se hace la cuenta considerando cuántas veces voy a ejecutar cada instrucción.
- En este caso aproximadamente nos da 1000 microsegundos, o 1 milisegundo

Ejercicio 1: Demoras sin Timer

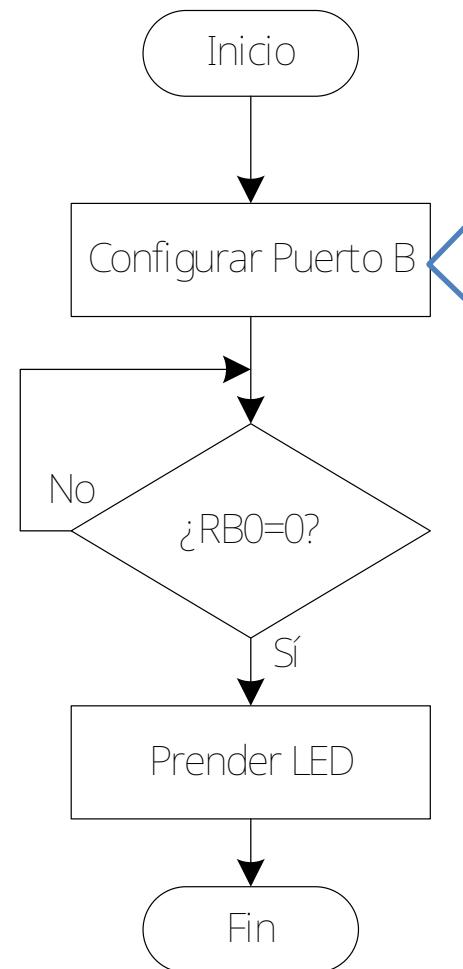
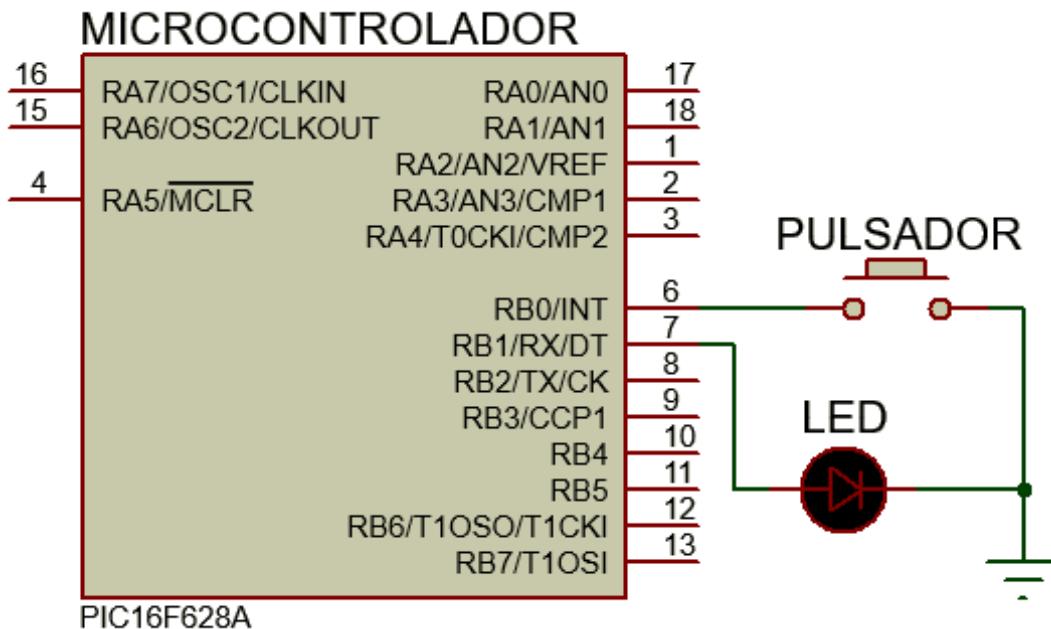
0

¿Cómo incrementamos ese tiempo?

- Se deben anidar contadores.
- Así, el tiempo de demora final es la multiplicación de las demoras de los contadores anidados
- En el ejemplo se cuentan aproximadamente 250 milisegundos, ya que se llama 250 a la demora de 1 milisegundo.
- Sólo se recomienda usar CALL y llamadas a subrutinas en caso de que no se usen demasiados contadores.

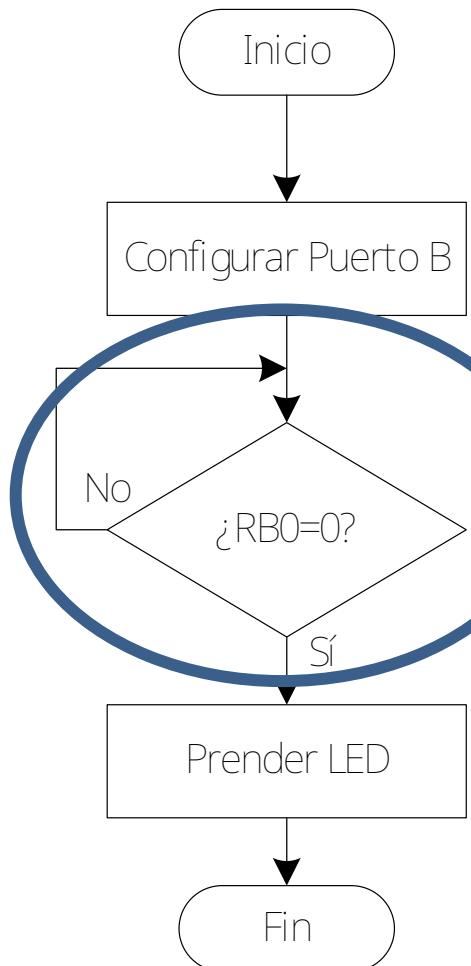
- delay_250ms
- movlw .250
- movwf CONT2
- loop CALL delay_1ms
- decfsz CONT,f
- goto loop
- return

Ejercicio 2: Manejo de Pulsadores



En este ejercicio se utilizarán los pull-ups internos del PIC. Para activarlos se debe usar la instrucción:
bcf OPTION, RBPU

Ejercicio 2: Manejo de Pulsadores



- Sin el uso de interrupciones se utiliza el método de “Polling”, nombrado así por estar constantemente “preguntando” algo.
- Ventaja: Implementación simple
- Desventaja: mientras se espera que se pulse el botón no se puede estar haciendo otra cosa simultáneamente.

esperar

btfsc PORTB,

RB0

goto esperar

bsf PORTB,

RB1

Ejercicio 2: Manejo de Pulsadores

- Por las propiedades mecánicas de los pulsadores pueden ocurrir rebotes los cuales pueden confundir al microcontrolador, el cual interpretará que se pulsó muchas veces el botón cuando sólo ocurrió una vez.
- En este caso se deben usar demoras “Anti-rebote” que garanticen que efectivamente una persona tocó el botón y no se está procesando un rebote.

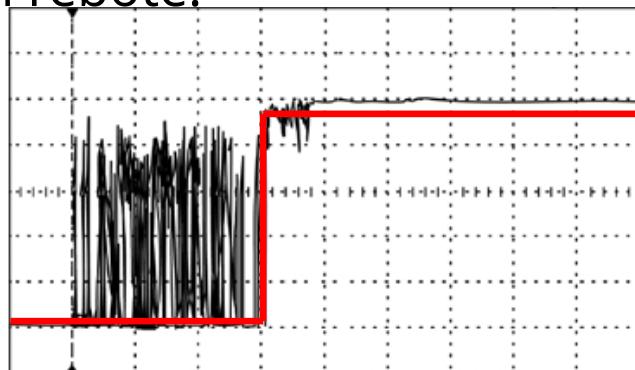


Gráfico de lo que ocurre al presionar un botón. En rojo se ve lo esperado vs. lo que ocurre físicamente

```
btfsc PORTB, RB0  
goto esperar  
call demora_20ms  
btfsc PORTB, RB0  
goto esperar  
bsf PORTB, RB1
```

Ejercicio 2: Manejo de Pulsadores

- Para evitar las desventajas de método de polling se utilizan interrupciones.
- Particularmente en este caso vamos a usar la interrupción por RB0.
- Esta se configura de la siguiente manera:
 - `bsf INTCON, GIE ; Habilito las interrupciones globales`
 - `bsf INTCON, INTE ; Habilito la interrupción por RB0`
 - La información sobre estos bits y los correspondientes al resto de las interrupciones se encuentran en el datasheet del PIC.

Ejercicio 2: Manejo de Pulsadores

- Al utilizar interrupciones debemos definir nuestra rutina de interrupción. Es una buena práctica que esto se haga al inicio del código de la siguiente manera:
 - org 0x00
 - goto inicio
 - org 0x04
 - bcf INTCON, GIE ; inhabilito las interrupciones globales
 - bcf INTCON, INTF ; bajo el flag de la interrupción
 - bsf PORTB, RB0 ; realizo la rutina, en este caso prender el LED
 - bsf INTCON, GIE ; habilito nuevamente las interrupciones
 - retfie ; Return especial para las rutinas de interrupción
 - Inicio ...

Ejercicio 2: Manejo de Pulsadores

- El programa principal quedaría conformado únicamente por la configuración de los puertos y las interrupciones, ya que el encendido del LED se hace en la interrupción:
- Inicio bsf INTCON,GIE ; habilito las interrupciones globales
- bsf INTCON,INTE; habilito la interrupción por RB0
- bsf STATUS,RP0 ; me muevo al Banco 1
- movlw b'00000001' ; configuro el TRISB: RB0 entrada y el resto salida
- movwf TRISB
- bcf STATUS, RP0 ; regreso al Banco 0
- goto \$; esto indica que se hace un loop en la misma posición
- end

Ejercicio 2: Manejo de Pulsadores

- En este caso no es necesario, pero para programas más complejos es recomendable guardar los valores del registro w y STATUS en registros temporales al inicio de la rutina de interrupción, para que no se altere el funcionamiento del programa principal.

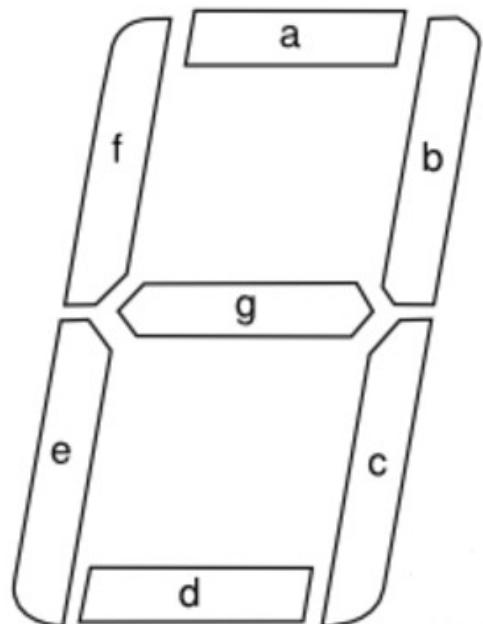
- org 0x04
- bcf INTCON,GIE ; inhabilito la interrupción por RB0
- bcf INTCON,INTF ; bajo el flag de la interrupción
- movwf AUXW ; guardo el valor de w en un
registro auxiliar
- movf STATUS,w
- movwf AUXSTATUS ; guardo el valor del STATUS en un
registro auxiliar
- ...

Ejercicio 2: Manejo de Pulsadores

- Antes de volver de la interrupción, se retornan los valores de los registros auxiliares
 - ...
 - `movf AUXSTATUS,w ; regreso el valor del STATUS`
 - `movwf STATUS`
 - `movf AUXW,w ; regreso el valor de w`
 - `retfie`

Ejercicio 3: Manejo de Displays

- Un display de 7 segmentos esta compuesto por LEDS controlados por distintos terminales identificados de la letra “a” a la letra “g”, como se indica a continuación:



- Suponiendo que los segmentos se encienden con uno, para formar el número 3 debería enviar un 1 a aquellos puertos conectados con los terminales “a”, “b”, “c”, “d” y “g”

Ejercicio 3: Manejo de Displays

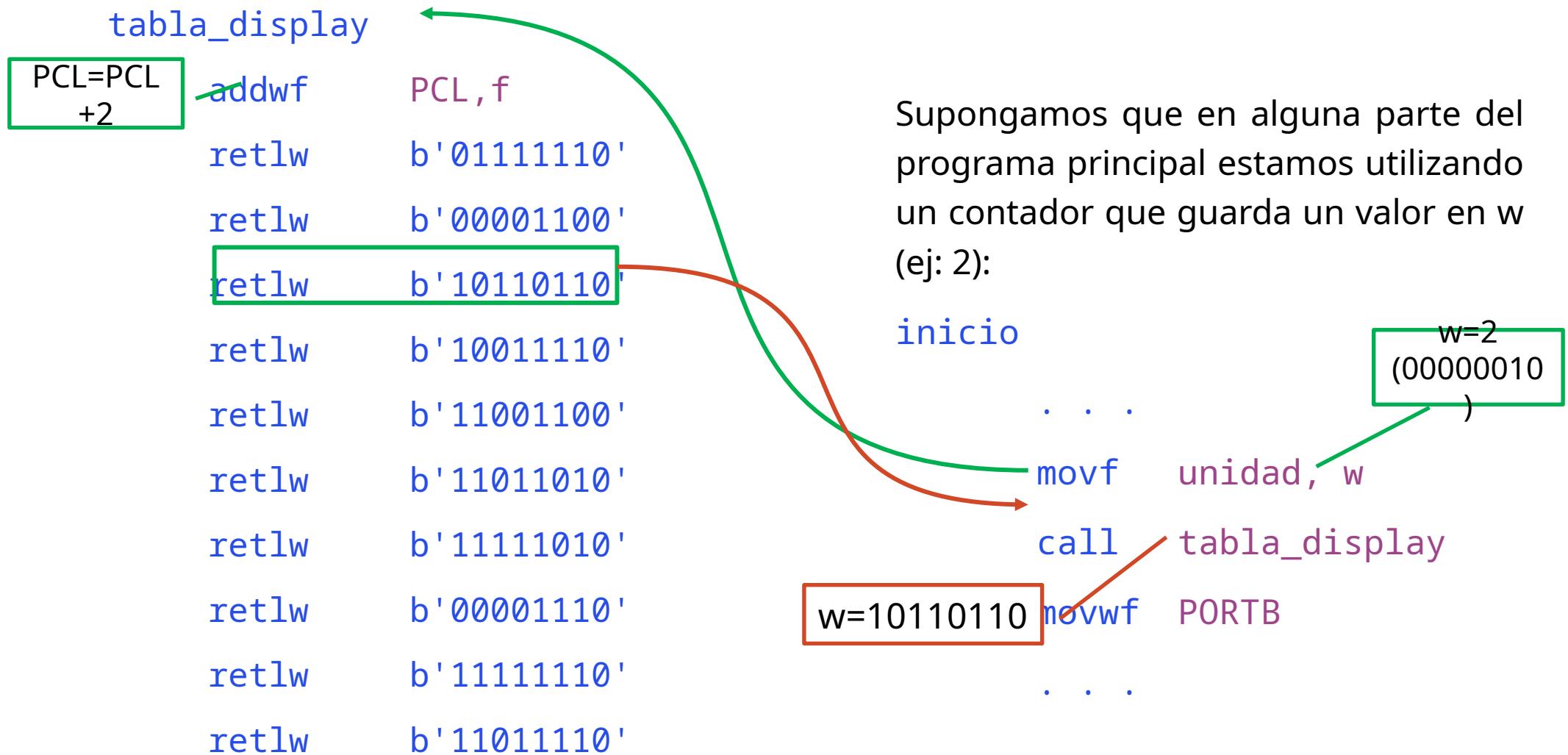
- En todos los ejercicios, los segmentos del display estarán al Puerto B (RB1-RB7). Entonces, los números se formarían de la siguiente manera:

Segmento	g	f	e	d	c	b	a	-
Puerto B	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
0	0	1	1	1	1	1	1	X
1	0	0	0	0	0	1	1	X
2	1	0	1	1	0	1	1	X
3	1	0	0	1	1	1	1	X
4	1	1	0	0	1	1	0	X
5	1	1	0	1	1	0	1	X
6	1	1	1	1	1	0	1	X
7	0	0	0	0	1	1	1	X
8	1	1	1	1	1	1	1	X
9	1	1	0	1	1	1	1	X

Ejercicio 3: Manejo de Displays

- De la tabla anterior puede verse que no existe una lógica matemática que nos permita hacer un pasaje sencillo de un número en binario a su representación en el display (Por ejemplo “00000010” a “10110110”). Para este tipo de problemas se implementan tablas utilizando el contador de programa (PCL) y una instrucción especial llamada “RETLW”.

Ejercicio 3: Manejo de Displays



Temas del segundo parcial

(Viernes 1/11 - Lunes 4/11)

1.- Genere las instrucciones necesarias para:

Ejemplo:

Sume el contenido del registro **w** al literal 10, y almacena el resultado en **w**.

Indique las banderas que se debe utilizar para controlar esta operación indicando los posibles valores y su significado

2.- Genere la/s instrucciones para realizar las siguientes operaciones matemáticas .

Ejemplo:

Sumar el valor de $w= 5$ al registro **DATO** guardándolo en **w** o en **DATO**
Restar al valor del registro **DATO= 5** el valor de $w= 10$ guardándolo en **w** o en **DATO**

3.-Utilización de banderas para controlar operaciones lógicas y aritméticas dentro del microcontrolador.

Ejemplo:

Se solicita indicar las banderas que se utilizan en las operaciones en una resta .

4.- Resolver un diagrama de flujo; plantear una solución posible.

5-. Dado un pseudocódigo e indicando una funcionalidad simple generar la instrucciones necesarias

Ejemplo:

Campo 1	Campo 2	Campo 3	PSEUDOCODIGO
			Instrucción al compilador para indicar que el código precedente a esta instrucción se situará en la dirección 0 Hexa

Temas del segundo parcial

(26/10)

6.- *Manejo de pulsadores.*

7.- *Calculo de tiempos con y sin uso de TIMER 0 (Programación)*

Resolución 2do. Parcial

- 1) Completar el siguiente cuadro con las instrucciones que permitan realizar lo descrito en la columna de comentarios. En cada fila debe ir solamente una instrucción o directiva.

Campo 1	Campo 2	Campo 3	Comentarios
	#INCLUDE	<P16F628A.INC>	;Agregar el archivo p16f628a.inc para utilizar en el proyecto
	_CONFIG	3F10	;Configurar el microcontrolador de acuerdo a las definiciones ;dadas en clase
	ORG	0X00	;El código precedente a esta linea se situará en la dirección 0
	BSF	STATUS,RP0	;Seleccionar el banco 1 para trabajar sobre el mismo
	BCF	TRISB,0	;Configurar el pin RB0 como salida
	BCF	STATUS,RP0	;Seleccionar el banco 0 para trabajar sobre el mismo
	BSF	PORTB, 0	;Encender led en RB0 (1 – encendido)
LOOP	GOTO	LOOP	;Generar un bucle infinito
	END		;Indicar al compilador el final del código

Resolución 2do. Parcial

- 2) Completar la siguiente subrutina para comprobar que un botón conectado al puerto RB2 ha sido presionado. En cada fila debe ir solamente una instrucción o directiva.

Campo 1	Campo 2	Campo 3	COMENTARIOS
BOTON			;Comienzo de subrutina
	BTFS	PORTB,2	
	GOTO	BOTON	
	CALL	DELAY_32ms	;Verifico si el botón ha sido apretado utilizando la técnica de polling junto con una demora antirrebote de 32ms.
	BTFS	PORTB,2	
	GOTO	BOTON	
	RETURN		;Cuando se presiona el botón se regresa de la subrutina

Ver PPT 208

Resolución 2do. Parcial

- 3) Desarrollar la subrutina de demora anti-rebote de aproximadamente 32 milisegundos que se emplearía en el ejercicio anterior, sin la utilización de ningún timer y con un clock de 4MHz. Considerar que las variables que se van a usar ya están declaradas (no es necesario usar ningún EQU).

DELAY

	MOVLW	.32
	MOVWF	CONT2
D2	MOVLW	.250
	MOVWF	CONT1
D1	NOP	
	DECFSZ	CONT1, F
	GOTO	D1
	DECFSZ	CONT2, F
	GOTO	D2
	RETURN	

Ver PPT 205

Resolución 2do. Parcial

RECUPERATORIO Lunes 5

- 4) Se requiere crear un circuito que prenda y apague un LED cada 1 segundo. Desarrollar el programa completo para dicho circuito utilizando el registro TIMER 0 como contador de tiempos y usando interrupciones para prender y apagar el LED, el cual está conectado al puerto RB5.

```
#INCLUDE    <P16F628A.INC>
__CONFIG    3F10
```

```
CONT_TMR0 EQU 0X20
```

```
ORG      0X00
GOTO    INICIO
ORG      0X04
BCF     INTCON, GIE
INCFL   CONT_TMR0, F
MOVFL   CONT_TMR0, W
XORLW   .20
BTFS    STATUS, Z
GOTO    VOLVER
COMF    PORTB, F
CLRF    CONT_TMR0
```

```
VOLVER
MOVLW   .61
MOVWF   TMR0
BCF     INTCON, T0IF
RETIE
```

```
INICIO
MOVLW   .61
MOVWF   TMR0
BSF    STATUS, RP0
BCF    TRISB, 5
BCF    OPTION_REG, T0CS
BCF    OPTION_REG, PSA
BSF    OPTION_REG, PS0
BSF    OPTION_REG, PS1
BSF    OPTION_REG, PS2
BSF    INTCON, GIE
BSF    INTCON, T0IE
BCF    STATUS, RP0
CLRF    CONT_TMR0
CLRF    PORTB
GOTO    $
```

Prende Led
X Interrupción

Apaga Led

Ver PPT 69 / 74
40/ 55