PIC®
Programable Integrates Circuit o Peripheral Interface Controller

un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine.

Es diferente a un microprocesador, dado que el PIC posee en su interior un microprocesador pero posee interfaces con el exterior, por lo que puede funcionar como un sistema en un solo chip

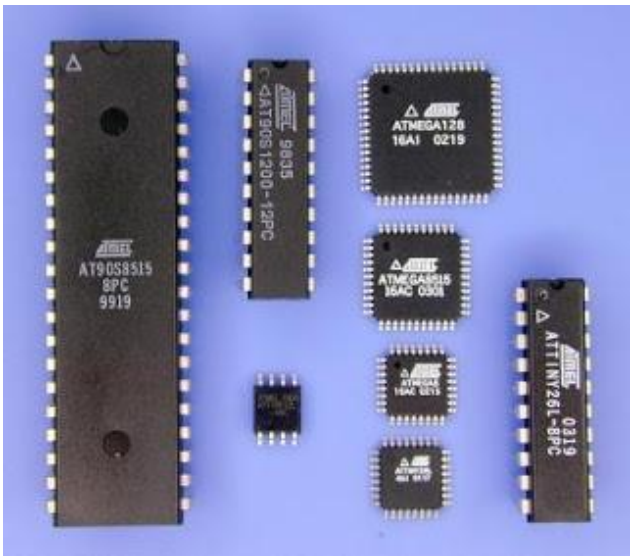
, su

Empresa	8 bits	16 bits	32 bits
Atmel	AVR (mega y tiny), 89Sxxxx familia similar 8051		SAM7 (ARM7TDMI), SAM3 (ARM Cortex-M3), SAM9 (ARM926), AVR32
Freescale (antes Motorola)	68HC05, 68HC08, 68HC11, HCS08	68HC12, 68HCS12, 68HCSX12, 68HC16	683xx, PowerPC, ColdFire
Holtek	HT8		
Intel	MCS-48 (familia 8048) MCS51 (familia 8051) 8xC251	MCS96, MXS296	x
National Semiconductor	COP8	x	x
Microchip	Familia 10f2xx Familia 12Cxx Familia 12Fxx, 16Cxx y 16Fxx 18Cxx y 18Fxx	PIC24F, PIC24H y dsPIC30FXX,dsPIC33F con motor dsp integrado	PIC32
NXP Semiconductors (antes Philips)	80C51	XA	Cortex-M3, Cortex-M0, ARM7, ARM9
Renesas (antes Hitachi, Mitsubishi y NEC)	78K, H8	H8S, 78K0R, R8C, R32C/M32C/M16C	RX, V850, SuperH, SH-Mobile, H8SX
STMicroelectronics	ST 62, ST 7		STM32 (ARM7)
Texas Instruments	TMS370	MSP430	C2000, Cortex-M3 (ARM), TMS570 (ARM)
Zilog	Z8, Z86E02		

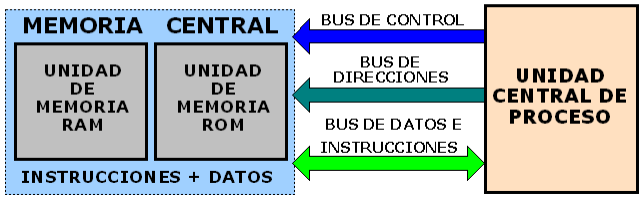
, su

Empresa	8 bits	16 bits	32 bits
Atmel	AVR (mega y tiny), 89Sxxxx familia similar 8051		SAM7 (ARM7TDMI), SAM3 (ARM Cortex-M3), SAM9 (ARM926), AVR32
Freescale (antes Motorola)	68HC05, 68HC08, 68HC11, HCS08	68HC12, 68HCS12, 68HCSX12, 68HC16	683xx, PowerPC, ColdFire
Holtek	HT8		
Intel	MCS-48 (familia 8048) MCS51 (familia 8051) 8xC251	MCS96, MXS296	x
National Semiconductor	COP8	x	x
Microchip	Familia 10f2xx Familia 12Cxx Familia 12Fxx, 16Cxx y 16Fxx 18Cxx y 18Fxx	PIC24F, PIC24H y dsPIC30FXX,dsPIC33F con motor dsp integrado	PIC32
NXP Semiconductors (antes Philips)	80C51	XA	Cortex-M3, Cortex-M0, ARM7, ARM9
Renesas (antes Hitachi, Mitsubishi y NEC)	78K, H8	H8S, 78K0R, R8C, R32C/M32C/M16C	RX, V850, SuperH, SH-Mobile, H8SX
STMicroelectronics	ST 62, ST 7		STM32 (ARM7)
Texas Instruments	TMS370	MSP430	C2000, Cortex-M3 (ARM), TMS570 (ARM)
Zilog	Z8, Z86E02		

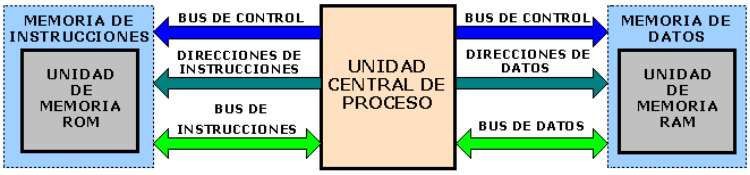
- OBJETIVO
- COSTO Y FACILIDAD DE OBTENCION
- ANCHO DE PALABRA (8, 16 y 32 bits)
- VELOCIDAD
- TIPO Y CANTIDAD DE PORTS



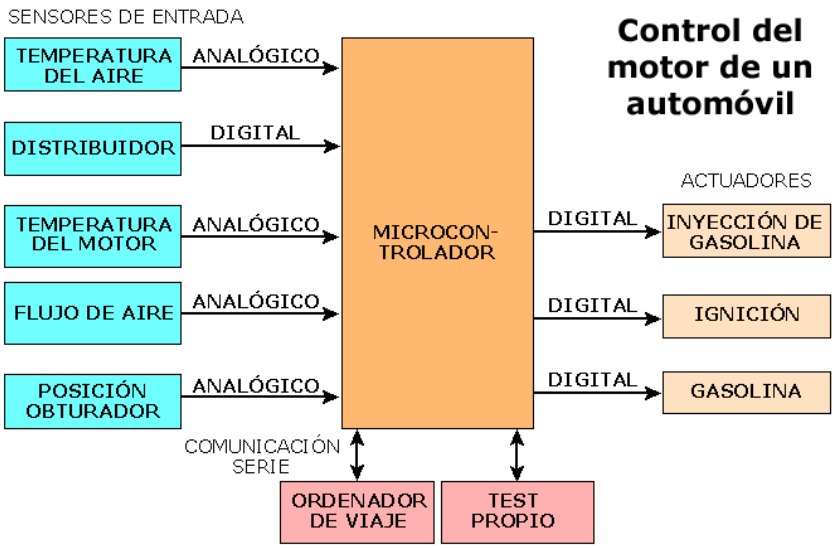
ARQUITECTURA VON NEUMANN

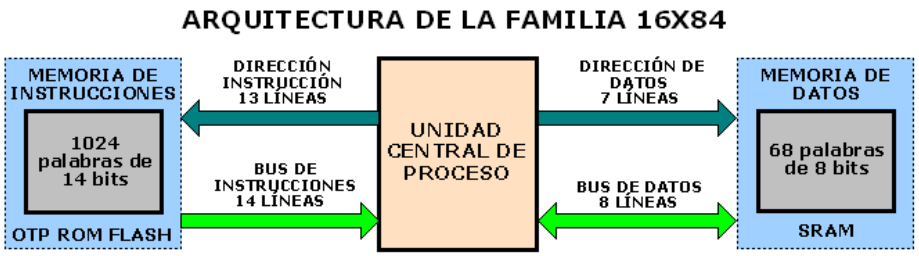


ARQUITECTURA HARVARD



MICROCONTROLADOR EN EL AUTOMOVIL

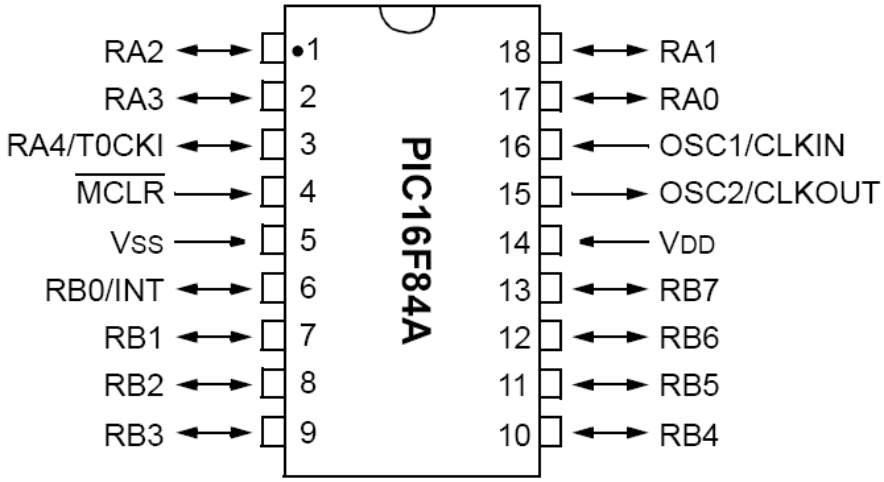




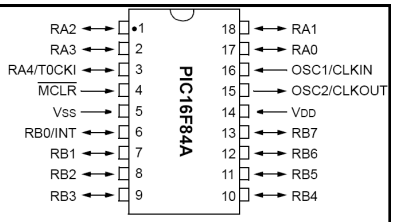
16C84A

- 2 puertos
- PortA tiene 5 pines numerados RA0 a RA4
- PortB tiene 8 pines numerados RB0 a RB7
- Pin RA4 lo que significa que solo puede conectarse a 0 Volt (no a 5 Volts).
- Conectaremos el LED a una fuente de 5 Volts (ánodo del led), y la salida (cátodo) a la resistencia en serie y esta al RA4.

(el pin del ánodo es el mas largo y el cátodo tiene una superficie plana junto al pin

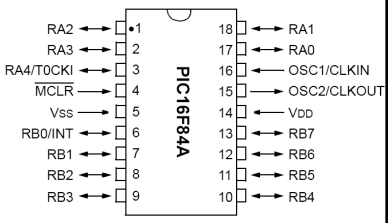


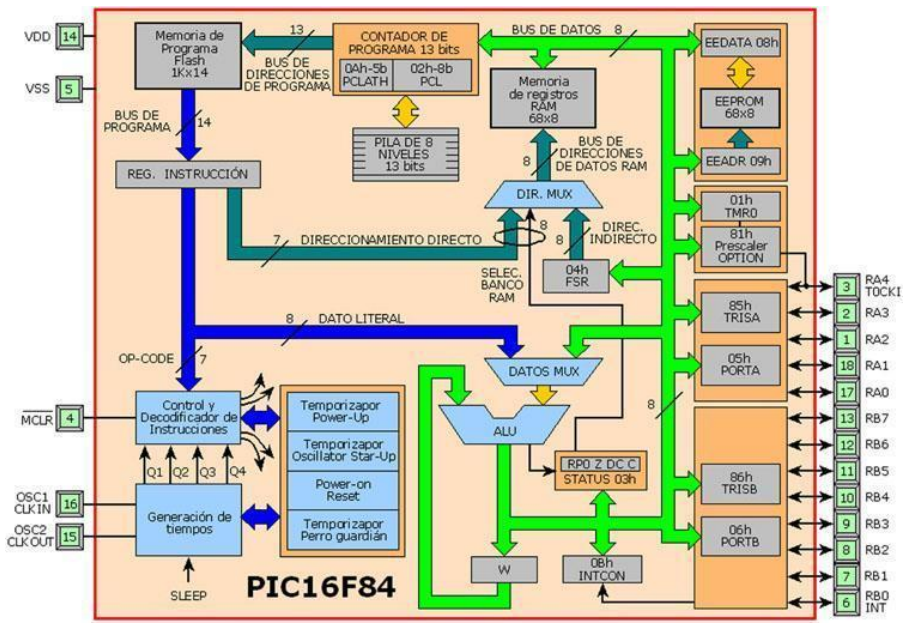
CONFIGURACION



- VDD (14) alimentacion
  5 V
- VSS (5) masa
- RA0 a RA 3 (17, 18, 1 y 2) puertos de entrada/salida
- RA4 (3) entrada/salida a colector abierto
- RB0 a RB7 (6 al 13) puertos de entrada/salida
- VPP (4) carga de memoria de programa cuando se exita con 15 volts.

- $\overline{\text{MCLR}}$ (4) Reset
- TOCKI (3) Temporizador
- INT (6) interrupcion
- OSC1 (16) clock imput
- OSC2 (15) clock output





Bancos de Memoria

Está organizada en dos páginas o bancos de registro, banco 0 y banco 1

Para cambiar de página se utiliza un bit del registro STATUS (RP0).

Cada banco se divide a su vez en dos áreas:

RFS (Registros de Funciones Especiales)

RGP (Registros de Propósito General)

La primera es la de RFS (Registros de Funciones Especiales) que controlan el funcionamiento del dispositivo. Estos se emplean para el control del funcionamiento de la CPU y de los periféricos.

El segundo área (68 bytes SRAM) es la de RGP (Registros de Propósito General), y puede accederse a ellos tanto directa como indirectamente haciendo uso del registro FSR.

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Bancos de Memoria

Banco 0

- Este banco está formado por 80 bytes, desde la posición 00 hasta la 4Fh.
- El área RFS consta de 12 registros que serán utilizados por funciones especiales del microcontrolador. Comienza en la dirección 00h y termina en la 0Bh.
- El Área RGP consta de 68 registros de memoria RAM que serán utilizados para almacenar datos temporales requeridos por los programas. Comienza en la dirección 0Ch y termina en la posición 4Fh. Esta parte es la memoria de registros de propósito general.

Banco 1

- Este banco tiene las mismas dimensiones que el anterior, pero su uso es menor, ya que no tiene banco para registros de propósito general. Solamente tiene una sección de registros especiales que van de la posición 80h a la 8Bh (desde la 128 a 139)

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

REGISTROS

REGISTRO Dirección	FUNCION
00 IND0	Direccionamiento indirecto de datos
01 TMRO	TIMER 0/counter register. Se usa para registrar cuentas o eventos. Puede ser incrementado por la aplicación de un pulso externo aplicado al pin TOCK I
02 PCL	Contador de Programa
03 STATUS	Registro de estado
04 FSR	REGISTRO SELECTOR DE REGISTROS. Asociado con el registro IND0, se utiliza para seleccionar indirectamente los otros registros disponibles. Si en el programa no se utilizan llamadas indirectas, este registro se puede utilizar como un registro de propósito general.

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

17

REGISTROS

REGISTRO Dirección	FUNCION
05 PORTA	PUERTO de ENTRADA/SALIDA. Sólo se utilizan los 5 lsb´s (RA0~ RA4). Este también puede ser programado como una entrada de clock El registro que contiene el sentido (entrada o salida) de los pines del puerto esta localizado en la pagina 1 (Banco 1), en la posición 85h y se llama TRISA
06 PORTB	PUERTO de ENTRADA/SALIDA de 8 BITS (RB0~RB7). Similar al anterior pero de 8 bits. El registro de control para la configuración de la función de sus pines se localiza en la pagina 1 (Banco 1), en la dirección 86h y se llama TRISB.
07	Sin Uso

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

18

REGISTROS

REGISTRO Dirección	FUNCION
08 EEDATA	REGISTRO de DATOS de la EEPROM. Este registro contiene el dato que se va a escribir en la memoria EEPROM de datos o el que se leyó de ésta.
09 EEADR	REGISTRO de DIRECCION de la EEPROM. Aquí se mantiene la dirección de la EEPROM de datos que se van a trabajar, bien sea para una operación de lectura o para una de escritura
0A PCLATH	REGISTRO de la parte alta del contador de programa y no se puede acceder directamente.
0B INTCON	REGISTRO para el CONTROL de INTERRUPCIONES..

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Registro STATUS (03h y 83h)

Bit	7	6	5	4	3	2	1	0
	IRP	RP1	RP0	T0	PD	Z	DC	C

- Bits 7:

  Selector de página para direccionamiento indirecto. Este bit no se utiliza efectivamente en el PIC 16F84, por lo que se puede utilizar como un bit de propósito general.

- BITs 5 y 6 Paginadores
  - 00 selecciona banco 0
  - 01 selecciona banco 1
  - 10 selecciona banco 2
  - 11 selecciona banco 3
- Bit 4 T0 Time out bit (o bit de finalización del temporizador)
- Bit 3 PD Power Down Bit (o de bajo consumo)

  Se usa la instrucción Sleep

Registro STATUS (03h y 83h)

Bit	7	6	5	4	3	2	1	0
	IRP	RP1	RP0	T0	PD	Z	DC	C

- **Bit 2 Z:** bit de cero

  Se setea cuando el resultado de una operación es o no cero.

  Se utiliza para verificar cuando una cuenta llega o no a cero.

```
Ej.   movlw.    10   ; pone en w <= 150
      subwf     REG, w     resta w de REG, Bit 2 = 1 si REG=10;
      btfss     status, zerobit  ;  salta si el  flag dio 1
```

- **Bit 1 DC:** Carry / borrow

  Flag de acarreo de dígito. Luego de operaciones aritméticas se activa si hay acarreo entre el bit 3 y 4, es decir cuando hay acarreo entre el nibble de menor y de mayor peso.

- **Bit 0 C:** Carry / borrow bit

  Indica si hubo acarreo luego de una operación de suma o resta.

Tipos de ROM

Versión Flash

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. A diferencia de las memoria de tipo ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que la EEPROM. Esta versión es idónea para la enseñanza y la Ingeniería de diseño.

Versión OTP

("One Time Programmable") "Programable una sola vez". Sólo se puede grabar una vez por el usuario sin la posibilidad de borrar lo que se graba. Resulta mucho más económica en la implementación de prototipos y pequeñas series.

Versión QTP

Es el propio fabricante el que se encarga de grabar el código en todos los chips que configuran pedidos medianos y grandes.

Versión SQTP

El fabricante solo graba unas pocas posiciones de código para labores de identificación, numero de serie, palabra clave, checksum, etc.

Registros que se utilizan con la EEPROM

Registro EEDATA (08h)
Registro de Datos, lectura/escritura 8 bits

Registro EEADR (09h)
Registro de Dirección, de 0h a 3Fh, 64 bytes

Registro EECON1 (88h)
Registro de Control 1

Registro EECON2 (89h)
Registro de Control 2

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Memoria de datos EEPROM

Esta memoria está basada en tecnología EEPROM, y tiene una longitud de 8 bits, del mismo modo que la memoria de datos. Su tamaño es de 64 bytes y está situada en un bloque distinto y aislado de la de datos.

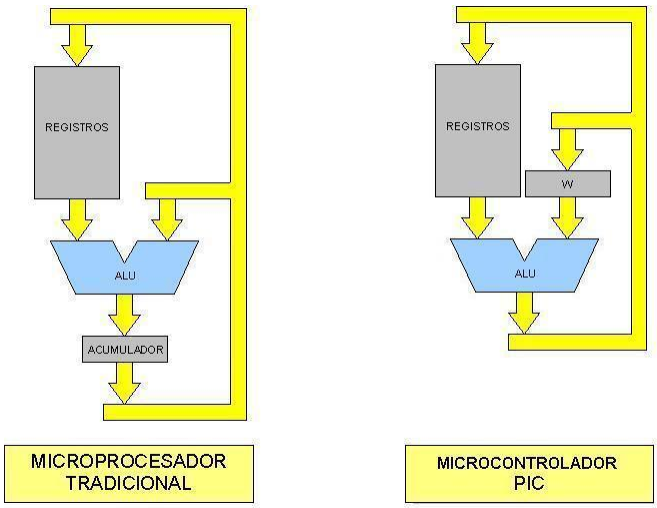
Los 64 bytes EEPROM de Memoria de Datos no forman parte del espacio normal direccionable, y sólo es accesible en lectura y escritura a través de dos registros, para los datos el EEDATA que se encuentra en la posición 0008h del banco de registros RAM y para las direcciones el EEADR en la 0009h. Para definir el modo de funcionamiento de esta memoria se emplean dos registros especiales, el EECON1 en la dirección 0088h y el EECON2 en 0089h.

REGISTRO W

- Es el corazón del microcontrolador.
- Para mover datos desde el ARCHIVO A al ARCHIVO B hay que mover los datos desde A a W y luego de W a B.
- En W también se realiza el movimiento de datos en las operaciones lógicas y matemáticas

Registro W

diagrama simplificado de la arquitectura interna de los datos en la CPU de los microcontroladores PIC y de los microprocesadores tradicionales relacionado con la ALU:



Escribiendo Software para PICs

- Podemos
  - Usar un editor de texto genérico
  - Usar MPLAB u otra aplicación de un nivel más alto de abstracción
- El PIC como todo micro utiliza un lenguaje binario para su funcionamiento.
  - Presentando una herramienta de compilación que permite la transformación de lenguaje mnemónico a uno de máquina

Pasos en la programación del PIC

El PIC funcionará si se le carga un programa en la MEMORIA DE PROGRAMA
Este deberá estar en lenguaje de máquina (binario)

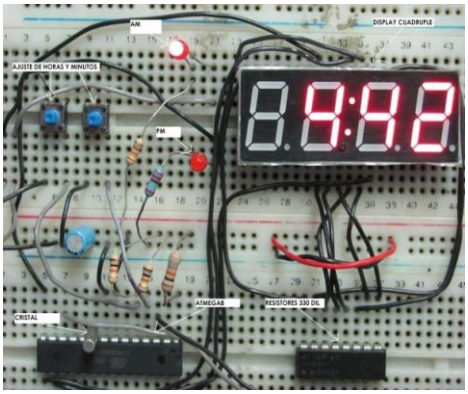
Normalmente el usuario escribe el programa en un lenguaje entendible (assembler), y mediante un programa editor (p.e. MPLAB) genera un archivo de texto con extension .asm

Luego otro programa, el compilador (p.e. MPSAM), se encarga de convertirlo a código de máquina

Este último dependiendo de la herramienta de desarrollo puede ser simulado su comportamiento antes de ser grabado en el PIC.

Por último se transfiere desde la PC al PIC, con un programa del tipo NOPPP, utilizando desde un grabador pequeño hasta la misma baseboard donde se pueda alojar el micro.

¿Dónde programar?

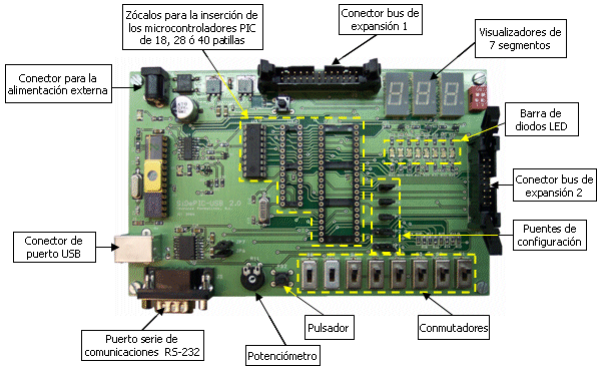


Protoboard

¿Dónde programar?

Alux 1.1

¿Dónde programar?

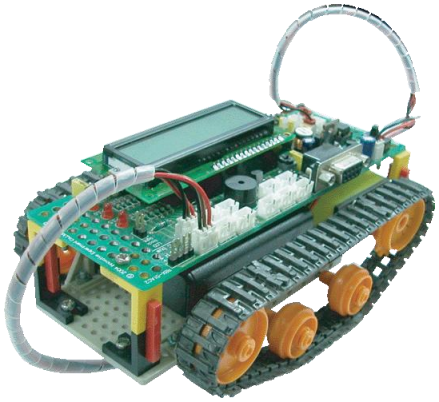


SiDePic-USB

¿Dónde programar?

EduMic - EduPic

¿Dónde programar?



iBOX-3

¿Dónde programar?

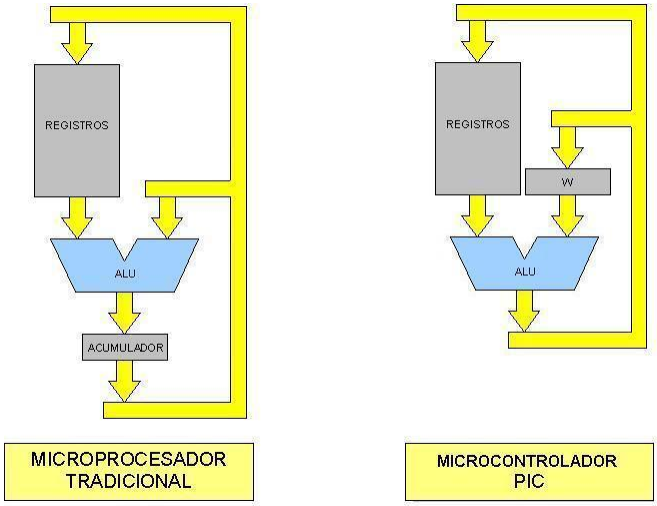
Aliatron e-Biz

¿Dónde programar?

EasyPic v7

Registro W

diagrama simplificado de la arquitectura interna de los datos en la CPU de los microcontroladores PIC y de los microprocesadores tradicionales relacionado con la ALU:



SET DE INSTRUCCIONES

Tres tipos diferentes de instrucciones

-ORIENTADAS A REGISTROS (o a bytes)

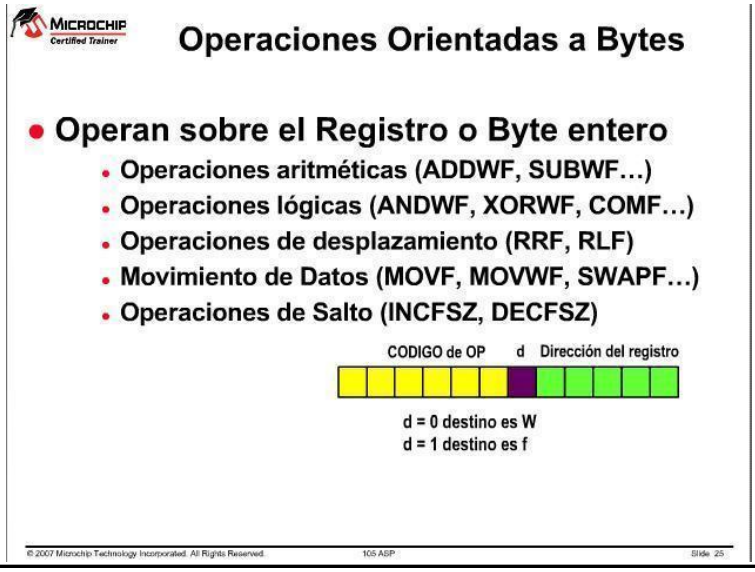
-ORIENTADAS A BITs

-ORIENTADAS A CONTROL Y AL MANEJO DE DATOS LITERALES

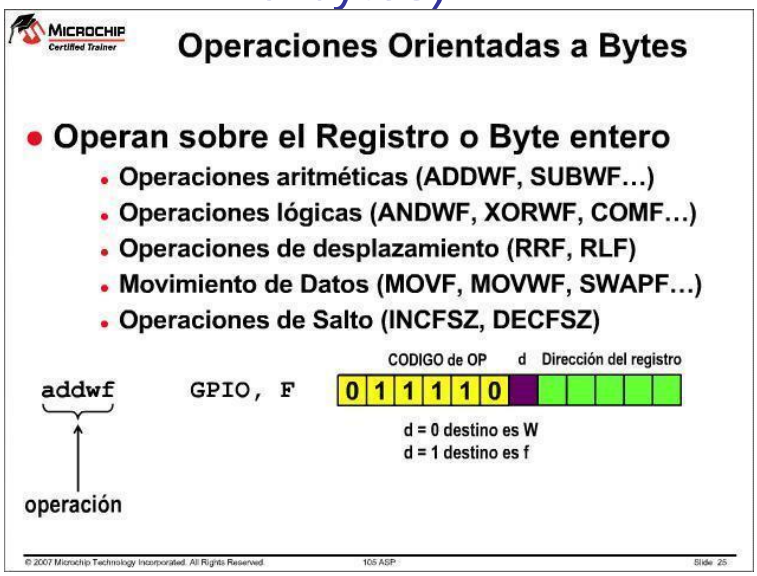
ORIENTADAS A REGISTROS (o a bytes)

MNEMÓNICO	OPERANDOS	DESCRIPCIÓN
ADDWF	f,d	w + f → d
ANDWF	f,d	w AND f → d
CLRF	f	00 h → f
CLRW	-	00 h → w
COMF	f,d	Complemento de f → d
DECF	f,d	f - 1 → d
DECFSZ	f,d	f - 1 → d (si es 0 salta)
INCF	f,d	f + 1 → d
INCFSZ	f,d	f + 1 → d (si es 0 salta)
IORWF	f,d	w OR f → d
MOVF	f,d	f → d
MOVWF	f	w → f
NOP	-	No operación
RLF	f,d	Rota f izq por carry → d
RRF	f,d	Rota f dcha por carry → d
SUBWF	f,d	f - w → d
SWAPF	f,d	Intercambia nibbles de f → d
XORWF	f,d	w XOR f → d

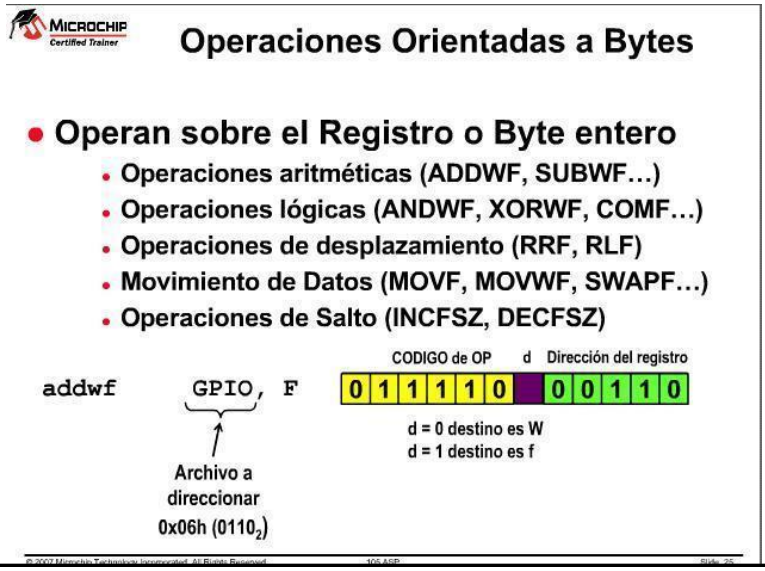
ORIENTADAS A REGISTROS (o a bytes)



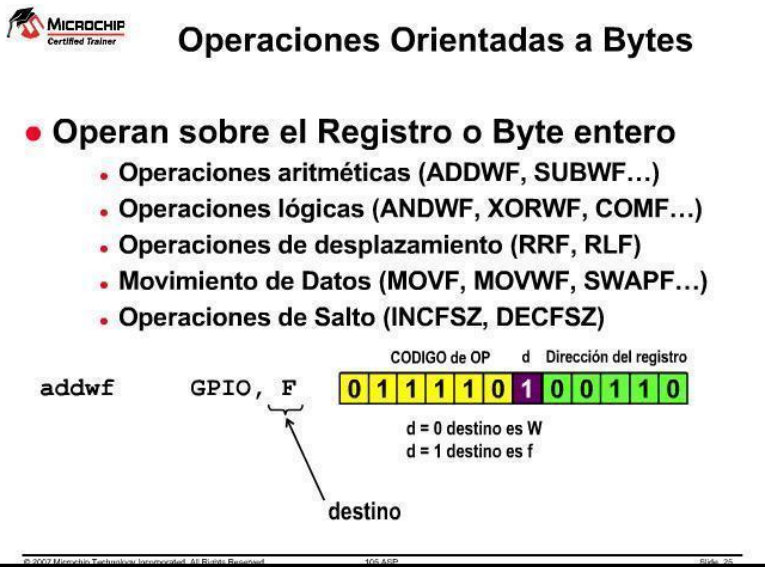
ORIENTADAS A REGISTROS (o a bytes)



ORIENTADAS A REGISTROS (o a bytes)



ORIENTADAS A REGISTROS (o a bytes)



ORIENTADAS A BITs

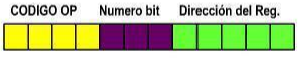
MNEMÓNICO	OPERANDOS	DESCRIPCIÓN
BCF	f,b	Pone a 0 bit b de registro f
BSF	f,b	Pone a 1 bit b de registro f
BTFSC	f,b	Salto si bit b de reg. f es 0
BTFSS	f,b	Salto si bit b de reg. f es 1

ORIENTADAS A BITs

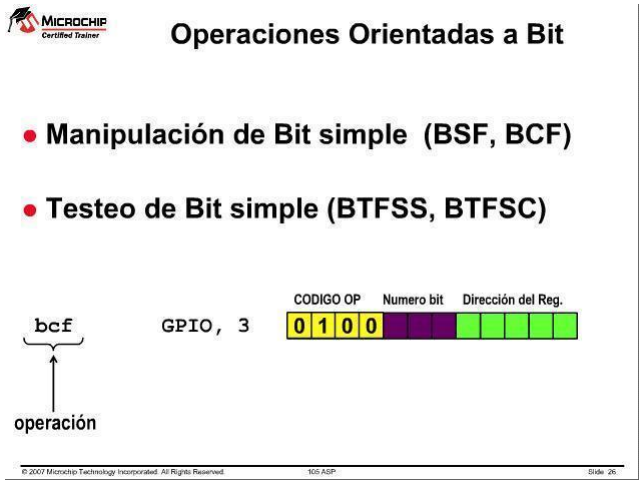
MICROCHIP Certified Trainer

Operaciones Orientadas a Bit

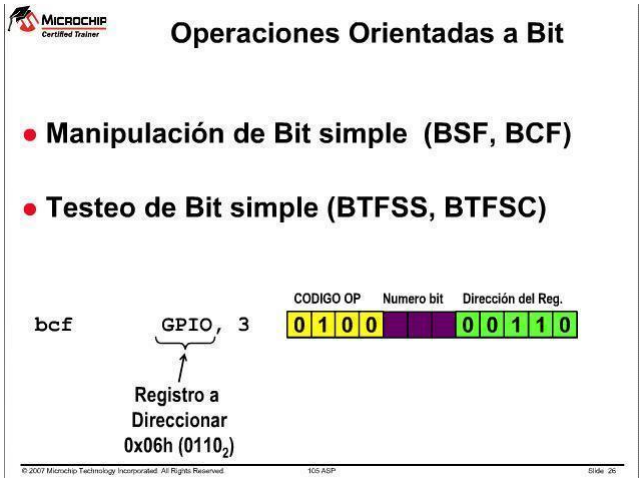
- **Manipulación de Bit simple (BSF, BCF)**
- **Testeo de Bit simple (BTFSS, BTFSC)**



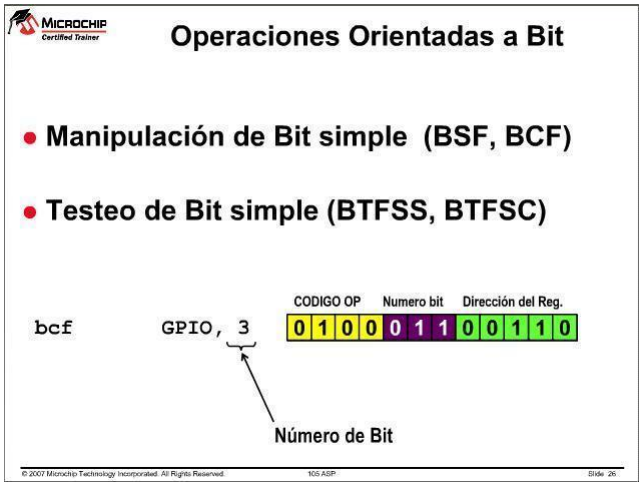
ORIENTADAS A BITs



ORIENTADAS A BITs



ORIENTADAS A BITs



ORIENTADAS A LITERAL Y CONTROL

MNEMÓNICO	OPERANDOS	DESCRIPCIÓN
ADDLW	k	w + k → w
ANDLW	k	w AND k → w
CALL	k	Llamada a subrutina k
CLRWDT	-	Borra temporizador del WDT
GOTO	k	Ir a dirección k
IORLW	k	w OR k → w
MOVLW	k	k → w
RETFIE	-	Retorno de una interrupción
RETLW	k	Retorno con k en w
RETURN	-	Retorno de una subrutina
SLEEP	-	Modo Standby
SUBLW	k	k - w → w
XORLW	k	w XOR k → w

ORIENTADAS A LITERAL

Operaciones con Literales y de Control

- **Instrucción Literal:**
  - Emplea valores definidos por el usuario
  - Operaciones lógicas (A

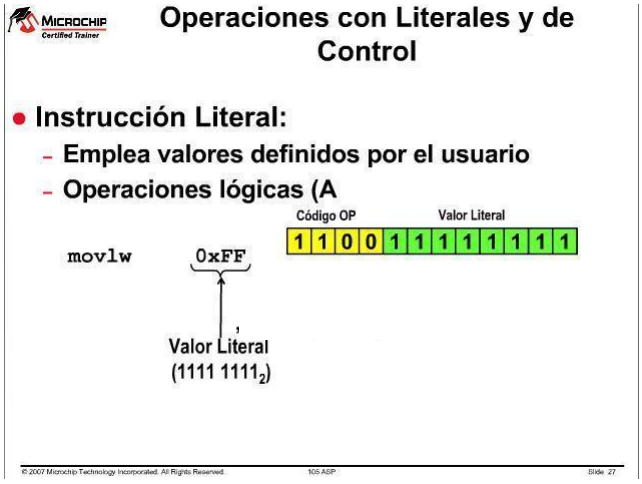
ORIENTADAS A LITERAL

Operaciones con Literales y de Control

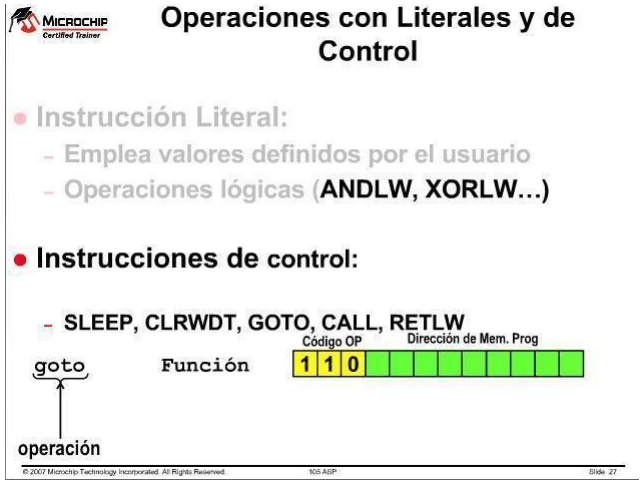
- **Instrucción Literal:**
  - Emplea valores definidos por el usuario
  - Operaciones lógicas (A

© 2007 Microchip Technology Incorporated. All Rights Reserved.
105 ASP
Slide 27

ORIENTADAS A LITERAL



ORIENTADAS A CONTROL



ORIENTADAS A CONTROL

Operaciones con Literales y de Control

- Instrucción Literal:
  - Emplea valores definidos por el usuario
  - Operaciones lógicas (**ANDLW, XORLW…**)
- **Instrucciones de control:**
  - **SLEEP, CLRWDT, GOTO, CALL, RETLW**

Dirección de Memoria

ORIENTADAS A CONTROL

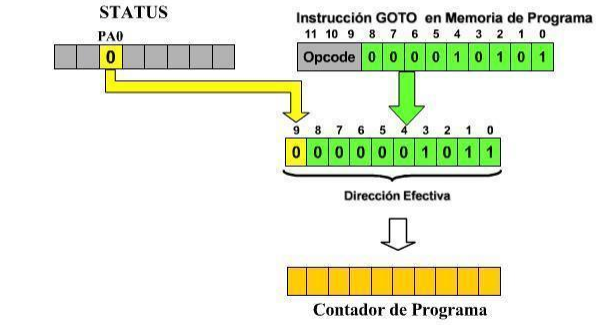
Operaciones con Literales y de Control

- Instrucción Literal:
  - Emplea valores definidos por el usuario
  - Operaciones lógicas (**ANDLW, XORLW…**)
- **Instrucciones de control:**
  - **SLEEP, CLRWDT, GOTO, CALL, RETLW**

Los bits de la paginación vienen del STATUS

Paginación GOTO

Como se implementa la Paginación

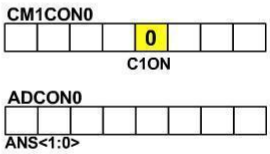


 105 ASP Slide 11

Configuración de puertos I/O

Configurando todos los pines como I/O Digital

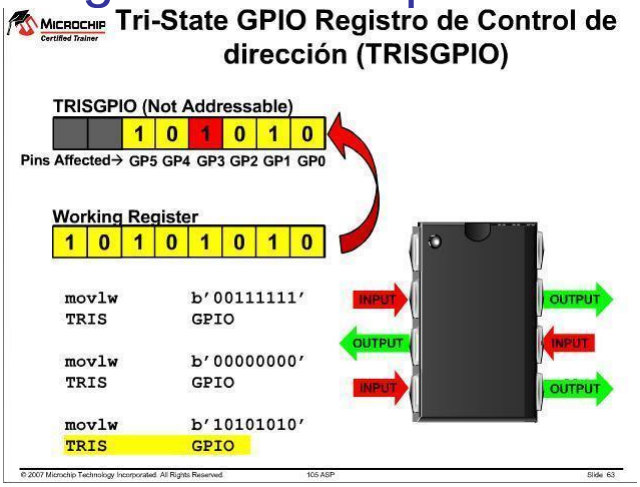
- **Borrar CM1CON0<C1ON> bit (desactiva el comparador)**
- **Borrar bits ANS<1:0> (Todos pines digital)**



 105 ASP Slide: 65

Configuración de puertos I/O

Configuración de puertos I/O



PROGRAMANDO

- Partir un problema largo en gran cantidad de pequeños problemas.

COMBINANDO DATOS E INSTRUCCIONES

- El PIC utiliza un numero binario de 14 bits
- El 16F84 tiene 1024 posiciones de 14 bits para almacenamiento

ESCRIBIENDO EN ASSEMBLER

- La escritura en ASSEMBLER se denomina ¨CODIGO FUENTE¨.
- Cuando el programa lo traduce a hex se denomina ¨CODIGO OBJETO¨.

CODIGO FUENTE

LABELS	1ra columna
MNEMONICS	2da. Columna en adelante
OPERANDOS	Luego del anterior, separado por un espacio.
COMENTARIOS	Luego del anterior, separado por ;

Directivas del assembler

Al utilizar un programa ensamblador podemos introducir además instrucciones o comando que proporciona el propio ensamblador. Estos comandos generalmente se utilizan para simplificar la tarea de programar, y reciben el nombre de directivas

- **Directiva EQU**

  El nombre viene de la palabra "equal", (igual)". La directiva EQU permite al programador "igualar" nombres personalizados a datos o direcciones. Los nombres utilizados se refieren generalmente a direcciones de dispositivos, datos numéricos, direcciones de comienzo, direcciones fijas, posiciones de bits, etc

- **Directiva ORG**

  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 #INCLUDE**

  Esta directiva indica que archivos deberán tomarse en cuenta a la hora de compilar el código. Normalmente se usa para incluir el archivo de PIC que el ensamblador tiene entre sus archivos, con el cual el compilador será capaz de reconocer todos los registros especiales y sus bits

- **Directiva LIST**

  Este comando sirve para que el compilador tenga en cuenta sobre qué procesador se está trabajando. Este comando debe estar en todo proyecto, situado debajo del "include", con la siguiente sintaxis.

  LIST P=PIC16F84A

- **Directiva END**

  Al igual que las dos anteriores, esta debe ir incluida una sola vez en todo el programa. En concreto, esta debe situarse al final, para indicar al ensamblador que el programa ha finalizado.

- **Directiva #DEFINE**

  Se usa para crear pequeñas macros. Con estas macros se pueden poner nombres a pequeños fragmentos de código que nos facilitarán la realización y comprensión del algoritmo.

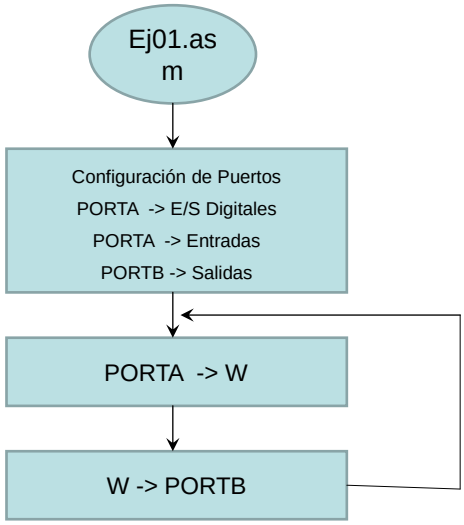
1. En todo programa se debe decir que PIC se utilizara para poder incluir las librerias correspondientes

```
list       p: 16f84
include <p 16f84.inc>
_CONFIG_RC_OSC8_WDT_OFF
```

2. Las sentencias se realizan en columnas

1ra. Nombra variables o coloca etiquetas
2da. Instrucción a ejecutar
3ra. Datos necesarios para ejecutar la instrucción de la 2da columna
4ta. Datos utiles para el programador no tenidos en cuenta por el PIC

PROGRAMA




```
;-------------------------------------------------------------------
;Programa Ej01.asm
;Este programa:
:           -Configura las entradas del PORTA como DIGITALES
;           -Lee contenido de PORTA y lo vuelca al PORTB
;PIC16F87X
;-------------------------------------------------------------------

            List      p=16F8t6                ;Tpo de procesador

            include   "P16F876.INC“           ;Definición de registros internos

            ORG       0x00                    ;Dirección de comienzo

            goto INICIO

INICIO      . . . . . .
```

```
INICIO  clrf    PORTB           ;Borra latch salida PORTB
        bsf     STATUS,RP0      ;Selecciona banco 1
        movlw   b'00000110‘      ;Carga literal
        movwf   ADCON1          ;PORTA E/S digitales

        clrf    TRISB           ;Configura PORTB como salida
        movlw   b'00011111'     ;Carga literal
        movwf   TRISA           ;Configura PORTA como entrada
        bcf     STATUS,RP0      ;Selecciona banco 0
BUCLE   movf    PORTA, W        ;W <- PORTA
        movwf   PORTB           ;PORTB <- W
        goto    BUCLE           ;Loop
        END
```

Recordando….

Bit	7	6	5	4	3	2	1	0
	IRP	RP1	RP0	T0	PD	Z	DC	C

- Bits 7:

  Selector de página para direccionamiento indirecto. Este bit no se utiliza efectivamente en el PIC 16F84, por lo que se puede utilizar como un bit de propósito general.
- BITs 5 y 6 paginadores
  - 00 selecciona banco 0
  - 01 selecciona banco 1
  - 10 selecciona banco 2
  - 11 selecciona banco 3
- Bit 4 T0 Time out bit

  Time Out o bit de finalización del temporizador.
- Bit 3 PD Powder Down Bit

  Power Down o bit de bajo consumo. Se coloca en 0 por la instrucción sleep

Recordando….

Dir. de registro	BANCO 0	BANCO 1	Dir. de registro
00h	Dir. Ind.[1]	Dir. Ind.[1]	80h
01h	TMR0	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	MAPEADOS (ACCESO) EN EL BANCO 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Localización de memoria no implementada, se lee como '0'

Nota 1: No es un registro físico

PROGRAMAS CARGADORES DE PICs

Para que el PIC funcione hay que cargarle un programa en la correspondiente MEMORIA DE PROGRAMA

Este debera estar en lenguaje de maquina en codigo binario (en realidad lo cargamos en hexadecimal y el micro lo convierte internamente en binario)

Normalmente el usuario escribe el programa en un lenguaje entendible (assembler), y mediante un programa editor (MPLAB) genera un archivo con extension .asm

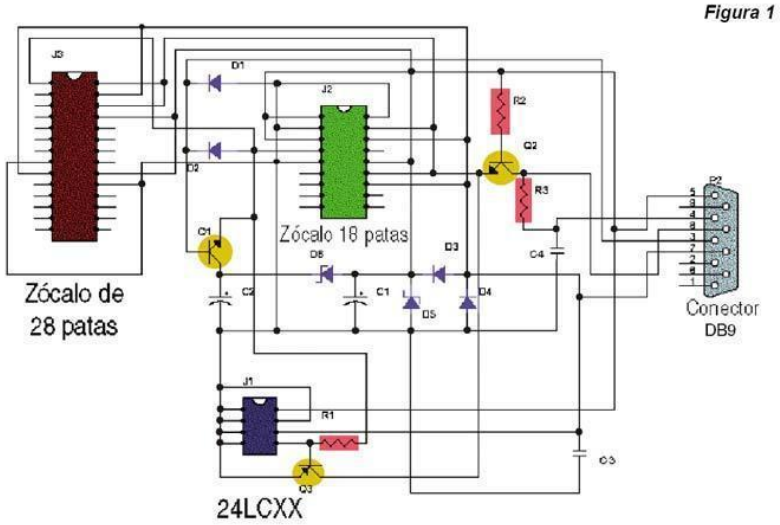
Otro programa, denominado ensamblador (MPSAM), se encarga de traducir este al lenguaje hexadecimal

Otro programa (NOPPP) me permite cargar este archivo en el PIC

Existen aplicaciones que me permiten simular el funcionamiento del circuito

CARGADORES DE PICs

Figura 1



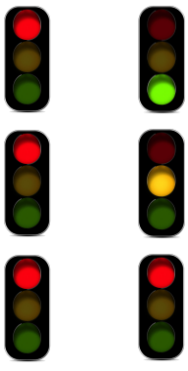
CARGADORES DE PICs

DEFINIR EL PROBLEMA

MEDIANTE UN PIC
MANEJAR UN SEMÁFORO
DE DOS VÍAS HECHO CON LEDS

DIAGRAMA DEL PROBLEMA

Semáforo 1 **Semáforo 2**



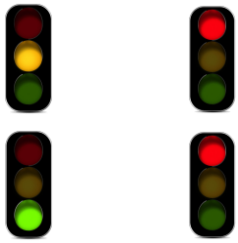
Duración: 5 segundos.

Duración: 1 segundo.

Duración: 1 segundo.

DIAGRAMA DEL PROBLEMA

Semáforo 1 **Semáforo 2**

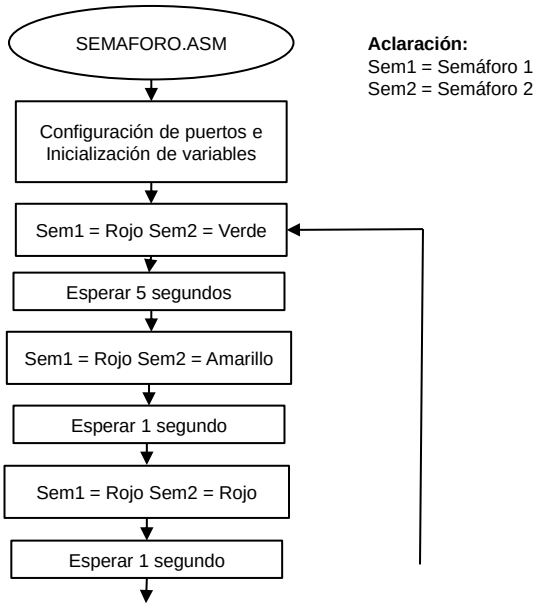


Duración: 1 segundo.

Duración: 5 segundos.

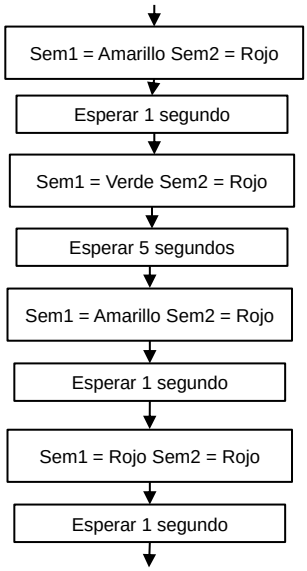
Y se repite.

DIAGRAMA DE FLUJO



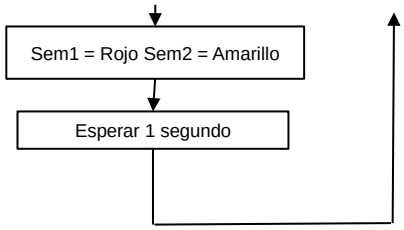
Aclaración:
Sem1 = Semáforo 1
Sem2 = Semáforo 2

DIAGRAMA DE FLUJO



Aclaración:
Sem1 = Semáforo 1
Sem2 = Semáforo 2

DIAGRAMA DE FLUJO



Aclaración:
Sem1 = Semáforo 1
Sem2 = Semáforo 2

CÓDIGO ASSEMBLER DEL PROBLEMA

```
;-------------------------------------------------------
; SEMAFORO.ASM
;-------------------------------------------------------
LIST P=16F84A
#INCLUDE <P16F84A.INC>

tempo1      EQU    0x20    ; temporizador1
tempo2      EQU    0x21    ; temporizador2
tempo3      EQU    0x22    ; temporizador3

ORG  0x00                  ; comienzo del programa

BSF  STATUS, RP0   ; cambia al banco 1 de registros
CLRF TRISB         ; configura el puerto B como salida
BCF  STATUS, RP0   ; cambia al banco 0 de registros
```

```
ciclo       MOVLW  b'100001'     ; ROJO y VERDE
            CALL   luz5s
            MOVLW  b'100010'     ; ROJO y AMARILLO
            CALL   luz1s
            MOVLW  b'100100'     ; ROJO y ROJO
            CALL   luz1s
            MOVLW  b'010100'     ; AMARILLO y ROJO
            CALL   luz1s
            MOVLW  b'001100'     ; VERDE y ROJO
            CALL   luz5s
            MOVLW  b'010100'     ; AMARILLO y ROJO
            CALL   luz1s
            MOVLW  b'100100'     ; ROJO y ROJO
            CALL   luz1s
            MOVLW  b'100010'     ; ROJO y AMARILLO
            CALL   luz1s
            GOTO   ciclo
```

```
luz5s       MOVWF  PORTB
            MOVLW  5
            CALL   tempo
            RETURN

luz1s       MOVWF  PORTB
            MOVLW  1
            CALL   tempo
            RETURN

tempo       CLRF   tempo1 ; pone tempo1 en 0
            CLRF   tempo2 ; pone tempo2 en 0
            MOVWF  tempo3 ; carga en tempo3 el valor de W
bucle       DECFSZ tempo1 ; resta 1 a tempo1
            GOTO   bucle  ; si no es 0 va a bucle
            DECFSZ tempo2 ; resta 1 a tempo2
            GOTO   bucle  ; si no es 0 va a bucle
            DECFSZ tempo3 ; resta 1 a tempo3
            GOTO   bucle  ; si no es 0 va a bucle
            RETURN        ; finaliza el bucle
END
```

DISEÑO DEL TEMPORIZADOR

1 instrucción → 4 ciclos de reloj

Frecuencia de trabajo = 1MHz
1s → 1.000.000 ciclos de reloj → 250.000 instrucciones (simples)

Costo de la subrutina *tempo*

Cada iteración:

DECFSZ → 1 ciclo
GOTO → 2 ciclos (GOTO + NOP)

Total: 3 ciclos

Iteraciones de *tempo1* = 256
Iteraciones de *tempo2* = 256
Total de ciclos de instrucciones utilizados = 256*256*3 = 196.608 ≅ 250.000

CICLO DE PROGRAMACIÓN DE UN PIC 16F84A