SISTEMAS ELECTRÓNICOS Y AUTOMÁTICOS
PRACTICAS DE MICROCONTROLADORES PIC

PRÁCTICA 2:

Programas básicos. Tipos de direccionamiento.

- ***Tipos de direccionamiento***
- ***Ejemplos y ejercicios***

---

1. Objetivos

- Introducir al alumno en la programación en ensamblador
- Conocer los tipos de direccionamiento en los PICs
- Simular el funcionamiento de programas-ejemplo.

2. Tipos de direccionamiento de la memoria de datos

Para direccionar la memoria de datos se emplean 3 modos de direccionamiento: **Inmediato**, **Directo** e **Indirecto**.

2.1. Direccionamiento Inmediato

Es aquel en el que el dato manipulado por la instrucción se codifica junto con ella. En este caso el dato se denomina literal. Por ejemplo, la instrucción:

MOVLW K

coloca el literal K en el registro de trabajo W. El literal utilizado puede ser cualquier valor de 8 bits y la base en la que se exprese es opcional[1].

2.2. Direccionamiento Directo

Es el modo más utilizado (acceso a la memoria de datos). Consiste en codificar el nombre del o los registros en cuestión directamente en la instrucción. Por ejemplo, la instrucción:

MOVWF f

desplaza el contenido de W al registro f. El registro f se referencia mediante su número codificado en 5 o 7 bits. Este "número" es en realidad la dirección del byte de la RAM correspondiente. Antes hay que colocarse en el banco adecuado. En el direccionamiento ***Directo*** los 7 bits de menos peso del código OP de la instrucción proporcionan la dirección en la posición o dirección dentro del banco, mientras que los bits RP1 y RP0 del registro de estado (STATUS) seleccionan el banco.

[1] La especificación de la base en la que se expresan los datos u operandos es opcional: d'65', b'01000001',0x41, o'101' y 'A' son el mismo dato. El sufijo **d** o **D** se emplea para expresar un valor en decimal. El sufijo **b** o **B** expresa un valor en binario. Para expresar un valor en hexadecimal se emplea el sufijo **0x** ó **0X**. Un número en base octal se representa mediante el sufijo **o** ó **O**. Finalmente, un valor se puede expresar mediante un caracter ASCII si se encierra entre comillas simples como 'A'.

2.3. Direccionamiento Indirecto

El direccionamiento ***Indirecto*** es el más potente. Emplea los registros INDF (posición 00h de la memoria de datos) y el registro FSR (*F*ile *S*elect *R*egister, posición 04h de la memoria de datos) o registro de selección de registro, en el que se introduce el número de registro seleccionado.

La idea del direccionamiento ***Indirecto*** es que la dirección de memoria del registro al que se quiere acceder se introduce en el registro FSR. Los 7 bits de menos peso de FSR seleccionan la posición y el bit de más peso, junto con el bit IRP del registro STATUS, selecciona el banco. En el PIC16F84 el IRP debe mantenerse a 0.
Cuando se quiera operar sobre el registro cuya dirección de memoria está almacenada en FSR, se usará en registro INDF. Por ejemplo:

MOVWF INDF

INDF no está implementado físicamente, por lo que no se accederá realmente a él. Si en cualquier instrucción se opera con este registro, en realidad se estará operando con la dirección a la que apunte el contenido de FSR.

El direccionamiento indirecto nos permite crear rutinas generales que no usen registros específicos. Las direcciones de dichos registros se asignarán antes de entrar en la rutina.

Lo único extraño es el modo de notación MOVWF INDF. Esta instrucción desplaza el contenido del registro de trabajo W al registro apuntado por FSR. Ya se ha indicado que ha de introducirse en FSR el número del registro direccionado, que será empleado como puntero. Este modo de direccioamento permite acceder a 256 direcciones.

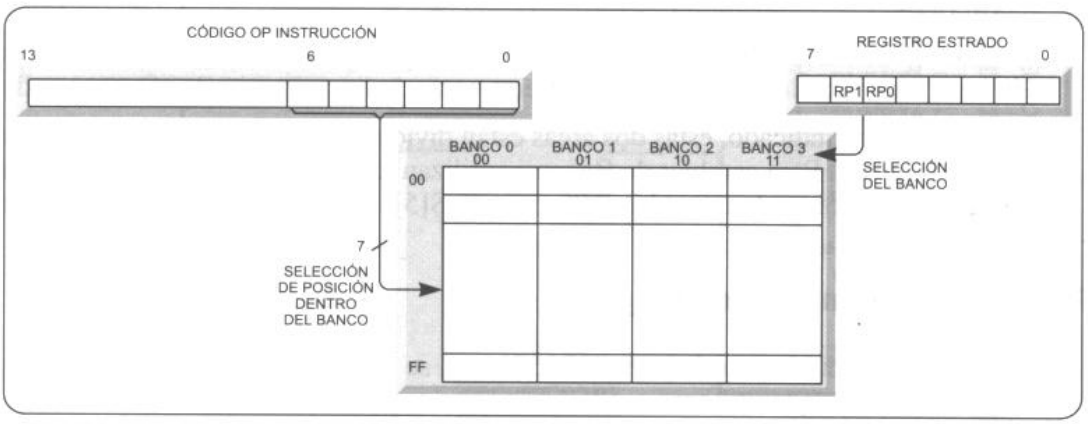

Figura 7.9. *Direccionamiento directo. La posición del dato la proporcionan los 7 bits de menor peso del código OP y la del banco los bits RP0 y RP1 ESTATUS.*

Figura 1. Direccionamiento directo.

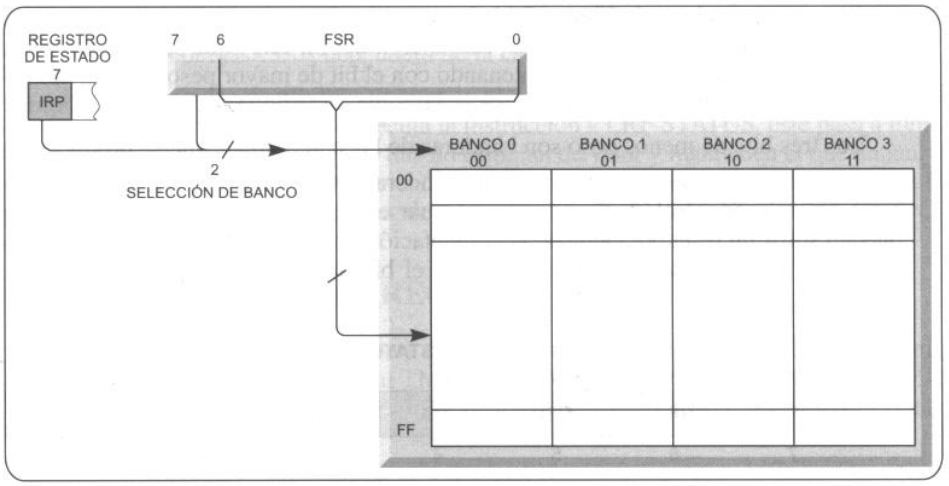

Figura 7.10. *Direccionamiento indirecto. Los 7 bits de menos peso de FSR apuntan a la posición y el bit de más peso junto con el bit IRP del registro de estado indican el banco.*

Figura 2. Direccionamiento indirecto.

2.5 Indirect Addressing; INDF and FSR Registers

The INDF register is not a physical register. Addressing INDF actually addresses the register whose address is contained in the FSR register (FSR is a *pointer*). This is indirect addressing.

EXAMPLE 2-1: INDIRECT ADDRESSING

- Register file 05 contains the value 10h
- Register file 06 contains the value 0Ah
- Load the value 05 into the FSR register
- A read of the INDF register will return the value of 10h
- Increment the value of the FSR register by one (FSR = 06)
- A read of the INDF register now will return the value of 0Ah.

Reading INDF itself indirectly (FSR = 0) will produce 00h. Writing to the INDF register indirectly results in a no-operation (although STATUS bits may be affected).

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: HOW TO CLEAR RAM USING INDIRECT ADDRESSING

```
        movlw   0x20    ;initialize pointer
        movwf   FSR     ;to RAM
NEXT    clrf    INDF    ;clear INDF register
        incf    FSR     ;inc pointer
        btfss   FSR,4   ;all done?
        goto    NEXT    ;NO, clear next
CONTINUE
         :              ;YES, continue
```

An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-3. However, IRP is not used in the PIC16F84A.

Figura 3. Direccionamiento indirecto en el PIC16F84A.

3. Ejercicios

3.1. Ejercicio:

Cread un proyecto y simulad el funcionamiento del siguiente programa que compara dos números A y B. Si A=B, el resultado es 0. Si A > B, el resultado es A-B. Si A < B el resultado es A+B. Hay que destacar que, al no haber instrucciones de comparación, ésta se realiza mediante restas.

```
;EJEMPLO 1
            List  p=16F84           ;Tipo de procesador
            include     "P16F84.INC"      ;Definiciones de registros
                                          ;internos

Dato_A            equ   0x10        ;Variable del dato A
Dato_B            equ   0x11        ;Variable del dato B
Resultado   equ   0x12        ;Variable para el resultado

            org   0x00        ;Vector de Reset
            goto  Inicio

            org   0x05        ;Salva el vector de interrupción

Inicio      movf  Dato_B,W    ;Carga el dato B
            subwf Dato_A,W    ;Resta/compara con dato A
            btfsc STATUS,Z    ;Son iguales (Z=1)??
            goto  A_igual_B   ;Si
            btfsc STATUS,C    ;No. A mayor que B (C=0)??
            goto  A_mayor_B   ;Si

A_menor_B   movf  Dato_A,W    ;No, A es menor que B
            addwf Dato_B,W    ;Suma a más B
            movwf Resultado   ;Guarda el resultado
            goto  Stop

A_mayor_B   movwf Resultado   ;
            goto  Stop

A_igual_B   clrf  Resultado   ;Pone a 0 el resultado

Stop        nop               ;Poner breakpoint de parada
            nop

            end               ;Fin del programa fuente
```

3.2. Ejercicio:

Diseñad un programa que realice la suma del contenido de las posiciones de memoria 0x10, 0x12 y 0x14 y compare el resultado de la suma con el contenido de la posición de memoria 0x16. Si el resultado de la comparación es 0 que guarde el resultado de la suma en la posición de memoria 0x20, si es mayor que 0 (suma>[0x16]), que decremente los contenidos de las posiciones de memoria 0x10, 0x12 y 0x14 en una unidad y repita el proceso de la comparación; y si la comparación es menor que 0 (suma<[0x16]), que aumente los contenidos de las posiciones de memoria 0x10, 0x12 y 0x14 en una unidad y repita el proceso de la comparación.

3.3. Ejercicio: *Direccionamiento Indirecto.*

Cread un proyecto y simulad el funcionamiento del siguiente programa que almacena el patrón 33 en 15 posiciones contiguas de la memoria de datos, empezando desde la dirección 0x10.

```
      ;EJEMPLO 2

            List  p=16F84          ;Tipo de procesador
            include     "P16F84.INC"      ;Definiciones de registros internos

Contador    equ   0x0c        ;Contador interno
Primera     equ   0x10        ;Posición inicial

            org   0x00        ;Vector de Reset
            goto  Inicio

            org   0x05        ;Salva el vector de interrupción

Inicio      movlw .15
            movwf Contador    ;Carga el contador con 15 (en decimal)
            movlw Primera
            movwf FSR         ;Inicia puntero con dirección inicial
            movlw 0x33        ;Carga patrón a almacenar

Bucle       movwf INDF        ;Almacena patrón en pos. indicada por FSR
            incf  FSR,F       ;Incrementa el puntero FSR
            decfsz Contador,F ;Decrementa contador hasta llegar a 0
            goto  Bucle       ;Contador no es 0

Stop        nop               ;Poner breakpoint de parada
            nop

            end               ;Fin del programa fuente
```

3.4. Ejercicio:

Modificad el programa anterior para que almacena el patrón AA, 10 veces seguidas en la memoria de datos, empezando desde la dirección 0x30.

3.5. Ejercicio:

Modificad el programa anterior para que almacena el patrón 012345, 10 veces seguidas en la memoria de datos, empezando desde la dirección 0x10.

3.6. Ejercicio:

Cread un proyecto y simulad el funcionamiento del siguiente programa que realiza el producto de dos números de 8 bits generando un resultado de 16 bits. El programa emplea la misma mecánica que al hacer un producto sobre el papel. Hay que destacar que el programa se ejecuta siempre en el mismo intervalo de tiempo, sean cuales fueren los operandos

```
;EJEMPLO 3

            List  p=16F84           ;Tipo de procesador
            include     "P16F84.INC"      ;Definiciones de registros internos

            cblock      0x10        ;Inicio de definición de variables

            Multiplicando           ;Variable para el multiplicando
            Multiplicador           ;Variable para el multiplicador
            Resultado_H       ;Parte alta del resultado
            Resultado_L       ;Parte baja del resultado
            Estatus_Temp            ;Reg. de estado temporal
            Contador          ;Variable con número de veces a operar

            endc              ;Fin de definiciones

            org   0x00        ;Vector de Reset
            goto  Inicio

            org   0x05        ;Salva el vector de interrupción

Inicio            clrf  Resultado_H
            clrf  Resultado_L ;Pone a 0000 el resultado inicial
            movlw 0x08
            movwf Contador    ;Inicia el contador con 8
            bcf   STATUS,C    ;Borra el carry

Bucle       movf  Multiplicando,W   ;Carga el multiplicando
            btfsc Multiplicador,0   ;Es 1 el bit de menos peso del multiplicador ??
            addwf Resultado_H,F     ;Si, se suma el multiplicando

            rrf   Resultado_H,F
            rrf   Resultado_L,F     ;Desplazamiento a la derecha del resultado

;Rota a la derecha el multiplicador sin que se modifique el flag Carry

Rota_sin_Carry    movf  STATUS,W
            movwf Estatus_Temp      ;Salva temporalmente el carry
            rrf   Multiplicador,F   ;Desplaza a la derecha el multiplicador
            movf  Estatus_Temp,W
            movwf STATUS            ;Recupera el carry original

            decfsz      Contador,F  ;Repite el bucle 8 veces
            goto  Bucle

Stop        nop               ;Poner breakpoint de parada
            nop

            end               ;Fin del programa fuente
```

4. Juego de instrucciones de 14 bits:

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.