

Microcontroladores

Semana 4

Semestre 2023-2

Por Kalun José Lau Gan

1

Preguntas previas

- En el ejercicio de generación de retardo de 40ms de la semana 3. ¿Cómo se declaran las etiquetas de los GPR?
 - En la parte superior del programa debajo de la declaración del PSECT se coloca:

```
x_var EQU 500H  
y_var EQU 501H
```
- Hay una nueva versión del XC8: v2.45 (lanzado el 31/08/2023)
- ¿Cómo puedo saber en que banco se encuentra un registro?
 - Revisando en la hoja técnica, generalmente los registros de manipulación de E/S se encuentran en el BANK4, los registros que manipulan la configuración del módulo de oscilador en el BANK0, la memoria RAM esta mapeada a partir del BANK5 y así en sucesivo.

2

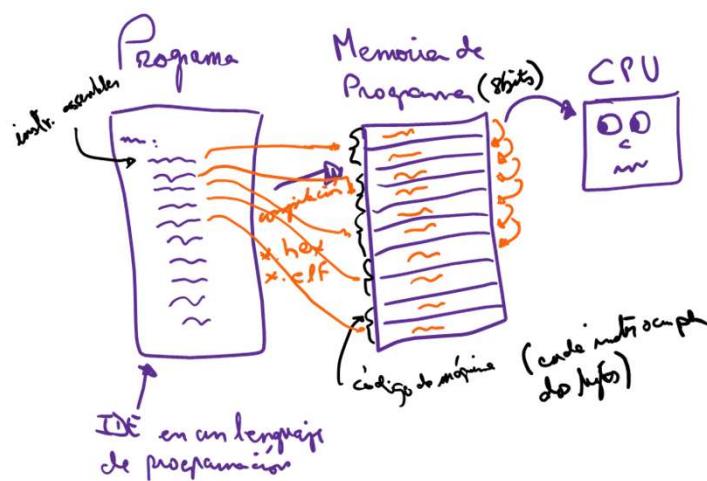
Agenda

- El contador de programa
- Los displays de siete segmentos
- Datos constantes en la memoria de programa
- El puntero de tabla (TBLPTR)
- Instrucciones de comparación numérica

3

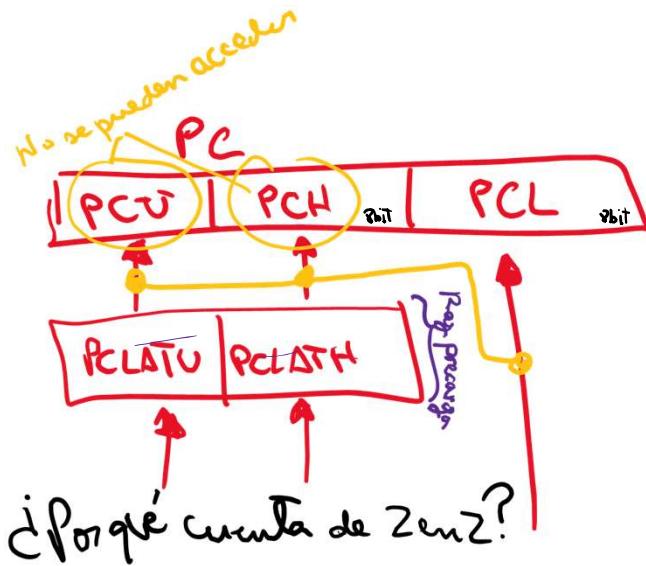
Recordando:

- La ejecución de un programa en el microcontrolador en de manera **secuencial** (una instrucción a la vez) y **ordenada** (uno detrás de otro).



4

El Contador de Programa (PC)



- Indispensable para la ejecución secuencial de las instrucciones.
- Su función es de alojar la dirección de la siguiente instrucción que el CPU va a ejecutar.
- Según hoja técnica, consta de 21 bits separados en tres registros: PCU, PCH y PCL.
- PCU y PCH no son accesibles, para que se pueda escribir la dirección de 21bits es necesario seguir paso previo que es la de precargar los datos en los registros previos PCLATH y PCLATU, y solamente se transferirán hacia PCU y PCH respectivamente cuando PCL le cargue un dato.
- Al escribir en PCL se subirán PCLATH y PCLATU para así subir los 21 bits a la vez
- En el PIC18F57Q43, se tiene solamente 17 bits ya que la memoria es de 128Kbyte

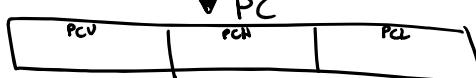
5

Ejemplo

- Cargar dirección 00288AH en PC:

```

    clrf PCLATH      ← Precarga
    movlw 28H
    movwf PCLATH      ← Precarga
    movlw 8AH
    movwf PCL          ← Se carga PCL
                        PCLATH hacia el PC
  
```



9.8 Register Summary - Memory Organization

Address	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x04D9	FSR2	7:0								FSRH[5:0]
0x04D8	PLUSW1	15:8								PLUSW[7:0] FSRH[5:0]
0x04DC	PREINC2	7:0								PREINC[7:0]
0x04D0	POSTDEC2	7:0								POSTDEC[7:0]
0x04D1	POSTINC2	7:0								POSTINC[7:0]
0x04E0	INDF	7:0								IND[7:0]
0x04E3	BSR	7:0								BSR[5:0]
0x04E1	FSR1	7:0								FSR1[7:0] FSRH[5:0]
0x04E3	PLUSW1	7:0								PLUSW[7:0]
0x04E4	PREINC1	7:0								PREINC[7:0]
0x04E5	POSTDEC1	7:0								POSTDEC[7:0]
0x04E6	POSTINC1	7:0								POSTINC[7:0]
0x04E7	IND1	7:0								IND[7:0]
0x04E8	WREG	7:0								WREG[7:0]
0x04E9	FSR0	7:0								FSR0[7:0] FSRH[5:0]
0x04E5	PLUSW0	7:0								PLUSW[7:0]
0x04E6	PREINC0	7:0								PREINC[7:0]
0x04E7	POSTDEC0	7:0								POSTDEC[7:0]
0x04E8	POSTINC0	7:0								POSTINC[7:0]
0x04E9	IND0	7:0								IND[7:0]
0x04F0	Reserved									
0x04F3	PCL	7:0								PCL[7:0]
0x04FA	PCLAT	7:0								PCLATH[7:0]
0x04FC	STKPTR	7:0								STKPTR[8:0]
0x04FD	TOS	7:0								TOS[7:0]
		23:16								TOS[20:16]

6

Ejemplo

- Si ejecutamos "goto inicio" en el siguiente programa:

```

13      org 0x0000
14      goto init_conf
15
16      org 0x0020
17      init_conf: clrf TRISD
18      inicio:    comf PORTB, W
19          movwf LATD
20          goto inicio
21      end

```

↓ hace un salto a 0x0020

↓ haz un salto a 0x0022

↓ modifica el PC

Direcciones de la memoria de programa

```

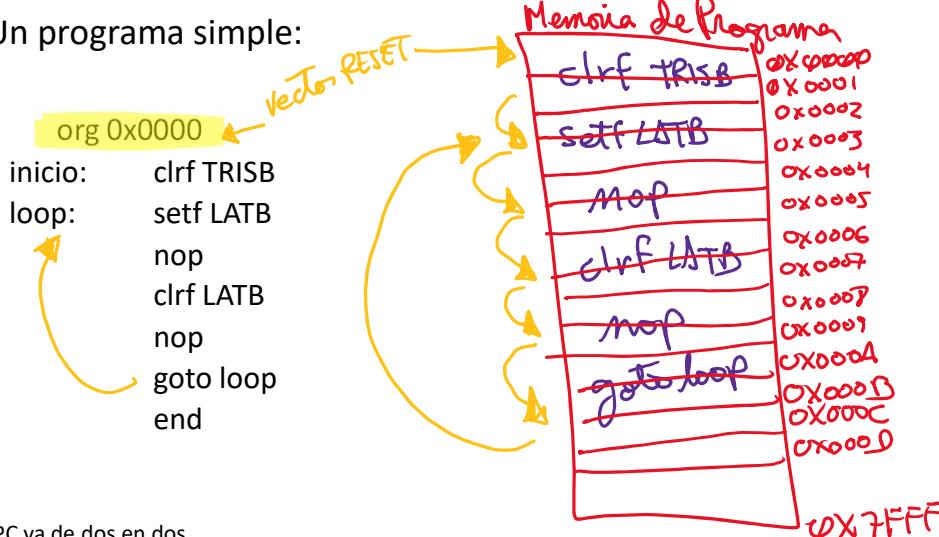
org 0x0000
goto 0x0020
org 0x0020
clrf 0xF95
org 0x0022
comf 0xF81, 0
movwf 0xF8C
goto 0x0022
end

```

7

¿Cómo funciona el PC?

- Un programa simple:



8

¿Cómo funciona el PC?

- Se tiene el siguiente programa

```
org 0x0000   
inicio: clrf TRISD  
        movlw 0x05  
        movwf LATD  
        goto inicio  
        end
```

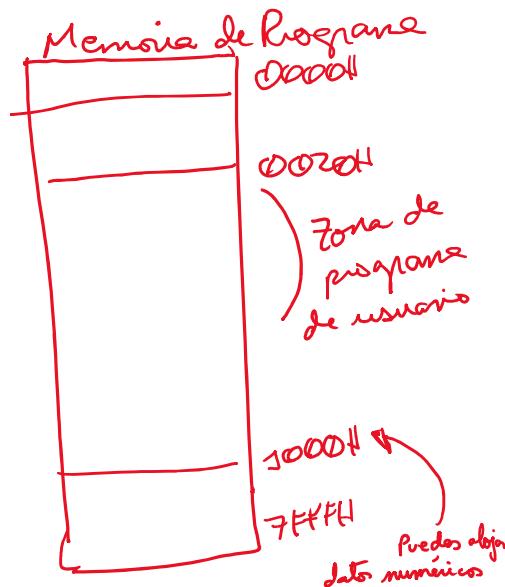
Nota: Las instrucciones en MPASM ocupan 2 bytes
Nota: Se comprueba que PC va de dos en dos, no
puede contener una dirección impar



9

Información alojada en la memoria de programa:

- En la memoria de programa podemos alojar no solamente instrucciones de un programa, sino también datos que serán constantes (no se podrán modificar cuando el microcontrolador entre en operación)



10

Otro ejemplo de PC:

- Las últimas 4 instrucciones emulan un salto a una posición de memoria, como si fuera una instrucción *bra* o *goto*

```

ORG 000000H
bra 000080H ;00H

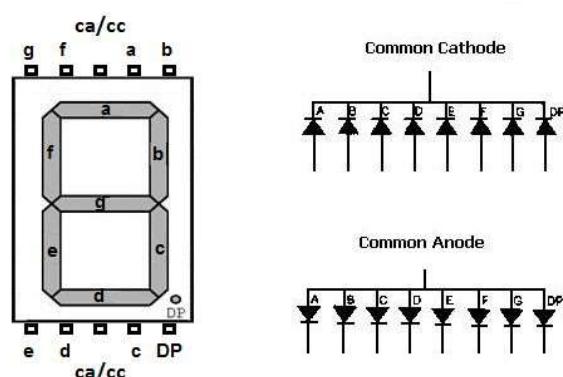
ORG 000080H
movlb 4H ;80H
bcf TRISA, 0, 1 ;82H
bcf ANSELA, 0, 1 ;84H
btg LATA, 0, 1 ;86H
nop ;88H
clrf PCLATU ;
clrf PCLATH ;
movlw 86H ;
movwf PCL ; hace un salto a 86H

```

11

El display de siete segmentos

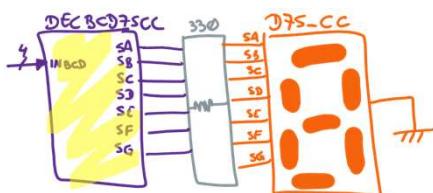
- Dos tipos: ánodo común y cátodo común
- Tablas de decodificación diferentes entre ellos.
- Es necesario colocar resistencias limitadoras de corriente para cada segmento que compone el display (100ohm a 1kohm).
- Evitar usar solo una resistencia en el pin común del display ya que al cambiar de carácter mostrado se tendrá un efecto de cambio de intensidad luminosa.



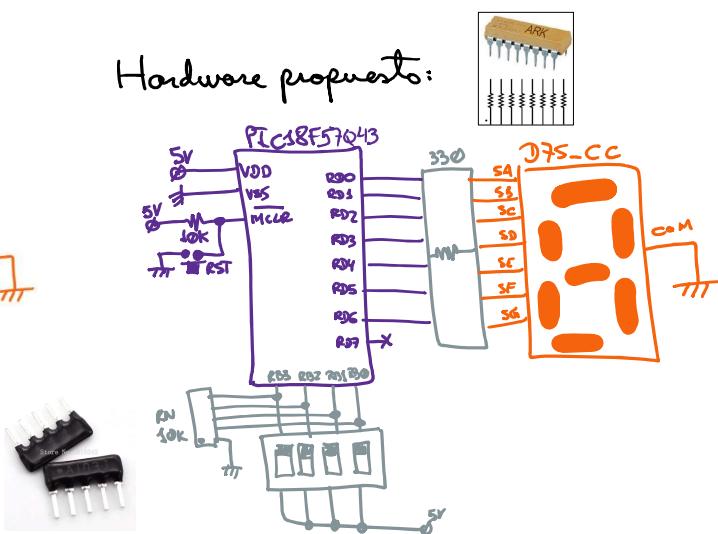
12

Ejemplo de decodificador BCD a 7 segmentos cátodo común con PC

Idea:



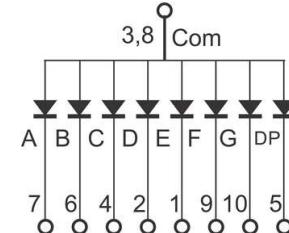
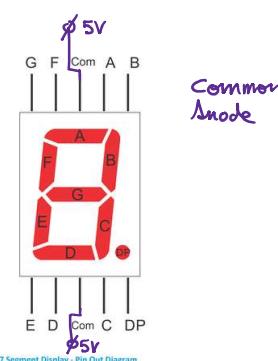
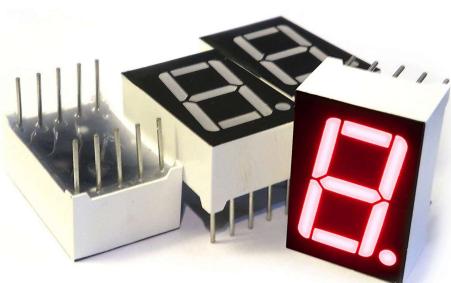
Hardware proposals:



13

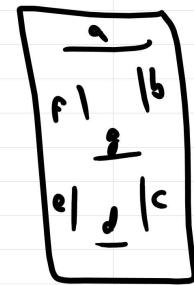
Observación:

- Los pines denominados “comunes” (Com) son el mismo nodo, la misma conexión!



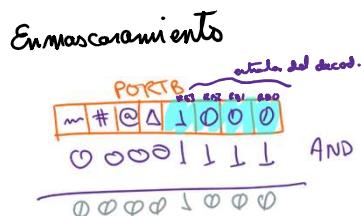
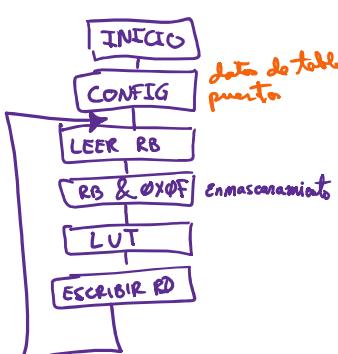
Desarrollo de la tabla de decodificación para display de 7 segmentos cátodo común:

CC	X	SG	SF	SE	SD	SC	SB	SA	HEX
	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	0	1	1	1	0x67



15

Diagrama de flujo



Código previo (empleando PC como LUT)

```

31    loop:
32        movf PORTB, w
33        andlw 0x0F
34        movwf valor_entrada
35        call tabla_pc
36        movwf LATD
37        goto loop

38    tabla_pc:
39        movf valor_entrada, w
40        addwf PCL, f
41        addwf PCL, f
42        (0)retlw 0x3F
43        (1)retlw 0x06
44        (2)retlw 0x5B
45        (3)retlw 0x4F
46        retlw 0x66
47        retlw 0x6D
48        retlw 0x7D
49        retlw 0x07
50        retlw 0x7F
51        retlw 0x67
52
53    end
  
```

Diagram illustrating the PC as a LUT. The PC is used to index into a table of addresses (0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F). The PC is modified by adding the current value of W (which is the masked input) to it twice. The resulting address is then used to jump to the corresponding table entry.

Problema: Debido a que se esta empleando el PC como LUT, al interactuar los datos de entrada (PORTB) con PC se obtendrán saltos a direcciones **impares**!

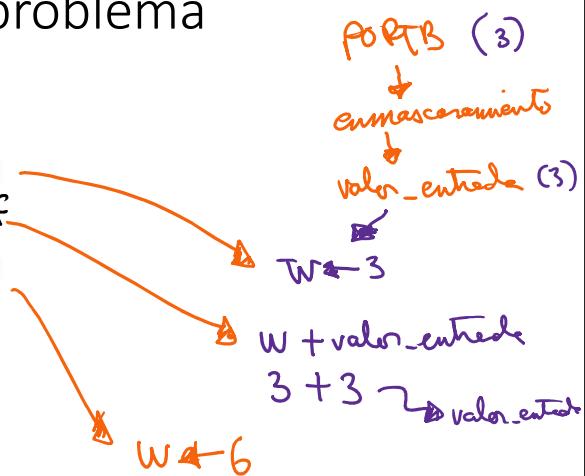
16

Propuesta de arreglo de problema

```

39  tabla_pc:
40      movf valor_entrada, w
41      addwf valor_entrada, f
42      movf valor_entrada, w
43      addwf PCL, f

```



Nota: Para que se realicen los saltos correctos empleando el PC (como LUT) se propone que luego de obtener el valor del dato de entrada de PORTB, éste dato se sumará a si mismo de tal modo que sea el doble, se obtenga las direcciones de salto en número par para el PC y funcione correctamente la LUT

17

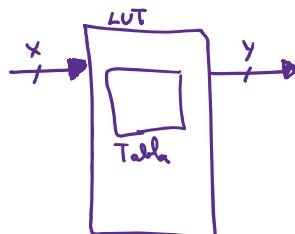
Acerca de la memoria de programa

- En esta memoria no solamente permite el alojamiento de instrucciones.
- También se puede alojar datos constantes, estos datos constantes solo se graban en un evento de programación del microcontrolador, luego en operación no se pueden modificar ni borrar.

18

Tablas de búsqueda (lookup tables - LUT)

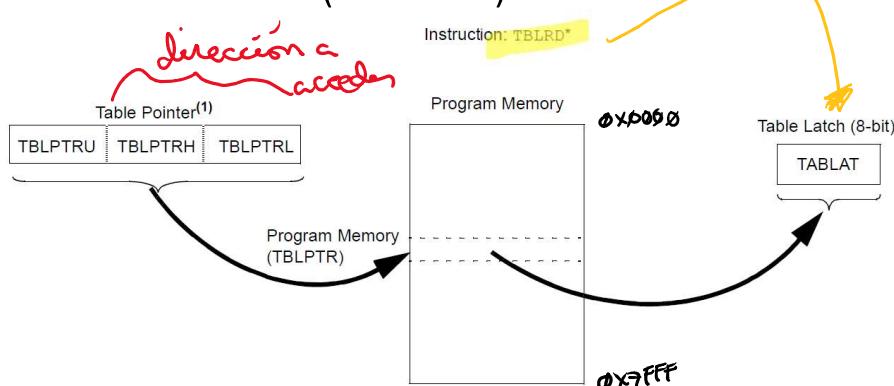
- Decodificadores implementados en software



- Dos maneras de implementar LUT en MPASM-PIC18
 - Utilizando la funcionalidad del PC (preferible no usar)
 - Utilizando el TBLPTR

19

El puntero de tabla (TBLPTR)

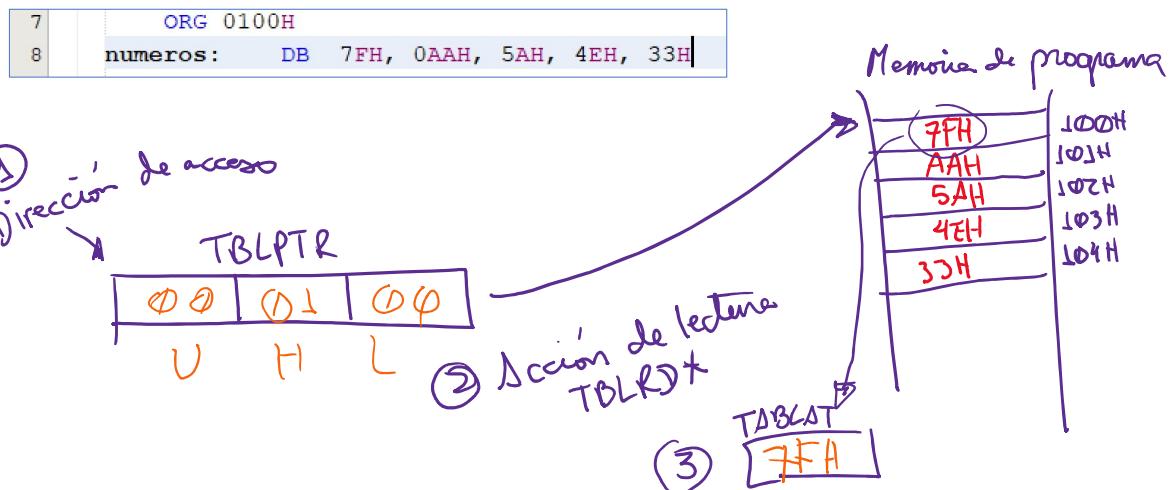


Note 1: Table Pointer register points to a byte in program memory.

- Al igual que el PC, el TBLPTR también es de 21 bits (para el caso del PIC18F4550 15 bits)
- Se emplea como única herramienta para acceder a la memoria de programa y leer (también escribir pero es más complicado) su contenido están en operación el microcontrolador.
- El puntero debe de tener la dirección de apunte antes de hacer el proceso de lectura con TBLRD*. Luego de la acción de lectura, el contenido de la celda apuntada se alojará en el registro TABLAT

20

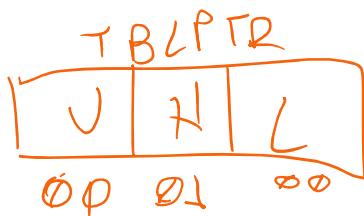
Operación con el TBLPTR



21

Operación con el TBLPTR

- En este programa se busca obtener el contenido de 102H en la memoria y trasladarlo al puerto D



```

1  PROCESSOR 18F4550
2  #include "cabecera.inc"
3
4  PSECT principal, class=CODE, reloc=2, abs
5
6  principal:
7      ORG 0100H
8      numeros: DB 7FH, OAAH, 5AH, 4EH, 33H
9
10     ORG 0000H
11     goto configuro
12     ORG 0020H
13
14 configuro:
15     clrf TRISD      ;Puerto D como salida
16     clrf TBLPTRU
17     movlw HIGH numeros
18     movwf TBLPTRH
19     movlw LOW numeros
20     movwf TBLPTRL    ;TBLPTR apuntando a 000100H
21
22 loop:
23     movlw 02H
24     addwf TBLPTRL, 1    ;Dirección de apunte modificada a 000102H
25     TBLRD*              ;Leo contenido apuntado por TBLPTR
26     movf TABLAT, 0        ;Muevo el contenido de TABLAT hacia WReg
27     movwf LATD            ;Muevo contenido de WReg hacia LATD
28     goto loop
29 end principal

```

22

Modificación del ejemplo del decodificador anterior empleando TBLPTR

```

1      ...
2      #include "cabecera.inc"
3
4      PSECT rstVect, class=CODE, reloc=2, abs
5
6      ORG 0500H
7      cadena: DB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67, 0x79, 0x79, 0x79, 0x79, 0x79
8
9      ORG 0000H
10     rstVect: goto configuracion
11
12     ORG 0020H
13     configuracion: movlw 80H
14             movwf TRI5D ;RD6:RD0 como salidas
15             clrf TBLPTRU
16             movlw HIGH cadena
17             movwf TBLPTRH
18             movlw LOW cadena
19             movwf TBLPTRL ;Asignamos dirección a TBLPTR (500H)
20             movf PORTB, w
21             andlw OFH
22             movwf TBLPTRL
23             TBLRD*
24             movff TABLAT, LATD
25             goto inicio
26
27     end rstVect

```

- Como segunda opción para implementar una LUT es empleando el puntero de tabla (TBLPTR) donde accederá a determinado dato ubicado en la memoria de programa dependiendo de la dirección asignada.

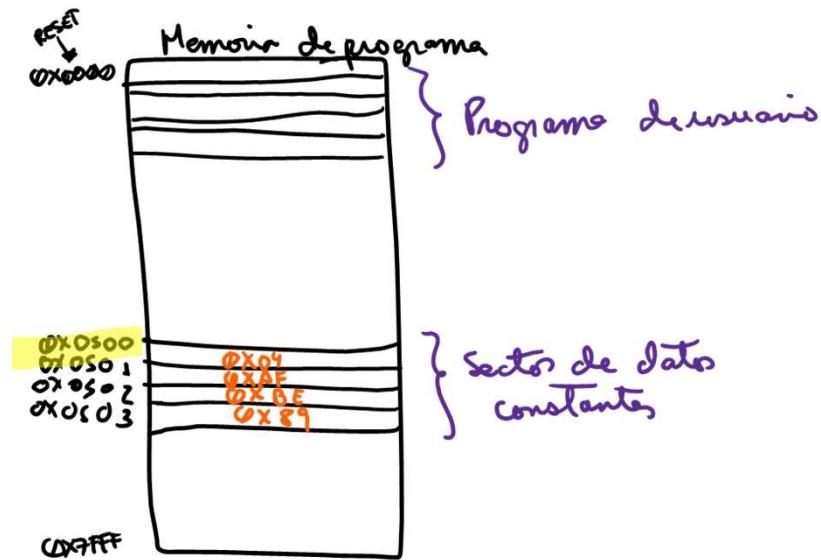
23

Ejemplo:

- Desarrollar un programa donde se tenga almacenado los siguientes datos en la **memoria de programa**:
 - 00500H: 04H
 - 00501H: AFH
 - 00502H: BEH
 - 00503H: 89H
- Elaborar un algoritmo que permita leer los datos anteriores y arrojarlas de manera secuencial a través de RD con periodo de NOP

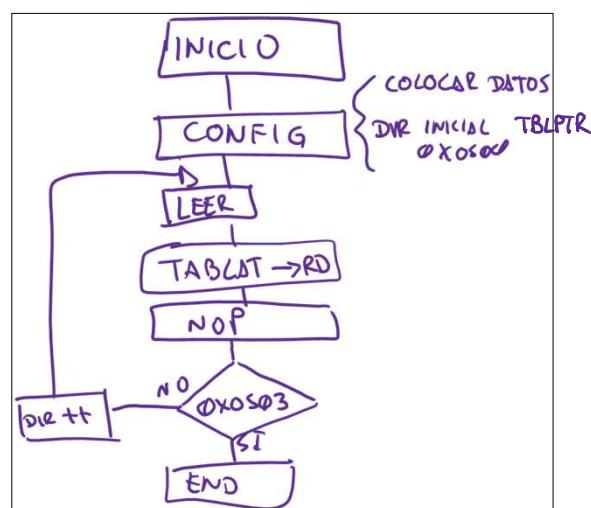
24

Desarrollo del ejemplo:



25

Diagrama de flujo:



26

Resumen de instrucciones con toma de decisión:

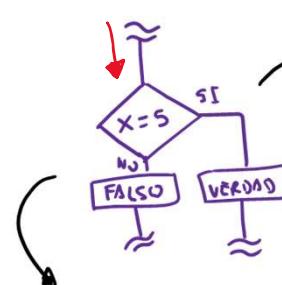


- btfss, btfsc
 - Prueba el #bit de [registro] si es 0 ó 1
- decfsz, incfsz
 - Decrementa o incrementa [registro] y pregunta si es cero.
- dcfsnz, icfsnz
 - Decrementa o incrementa [registro] y pregunta si **no** es cero
- cpfsgt, cpfseq, cpfslt
 - Comparaciones numéricas entre [registro] y W
- tstfsz
 - Prueba [registro] y salta si es cero
- Saltos Branch
 - Saltos condicionales

27

Instrucciones de comparación numérica (CPFSEQ, CPFSLT, CPFSGT)

- $\text{CPFSEQ} \rightarrow f = W_{\text{reg}}$
 $\text{CPFS LT} \rightarrow f < W_{\text{reg}}$
 $\text{CPFS GT} \rightarrow f > W_{\text{reg}}$



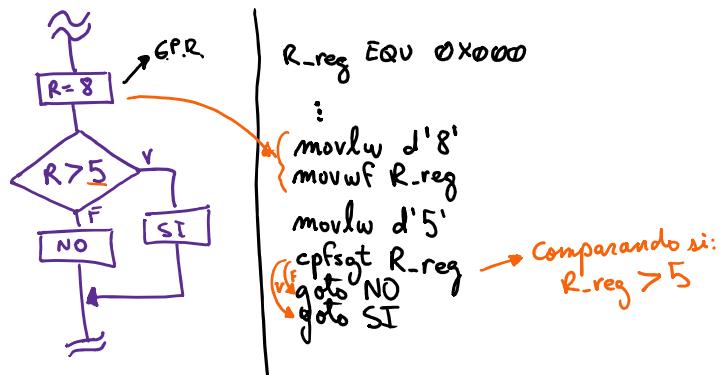
XC8 ASM
En MPASM:
 Usaremos CPFSEQ:
 $\text{CPFSEQ } [\text{reg}]$

Nota: El valor a comparar debe de estar previamente en Wreg
 En lenguaje de alto nivel:
 $\text{if } (x = 5) \{$
 $\quad \text{[VERDAD]}$
 $\text{else} \{$
 $\quad \text{[Falso]}$
 $\}$

$(f) = w$
 $x = 5$
 $\text{movlw } 5$
 $\text{cpfseq } x - \text{reg}$
 gotofalso
 gotoverdад

28

Ejercicio: Pasar a XC8 ASM el siguiente diagrama de flujo:

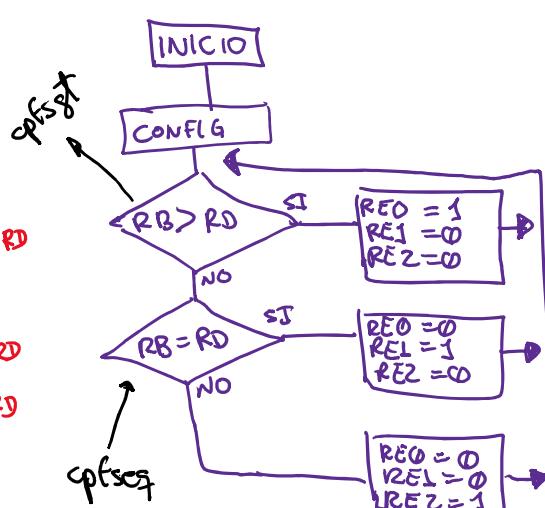
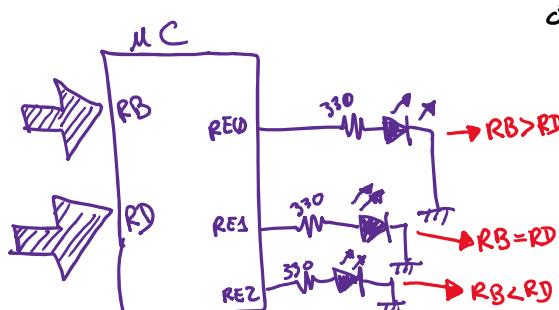


Nota: En las instrucciones cpfsgt -- el segundo parámetro de la comparación debe de estar en Wreg

29

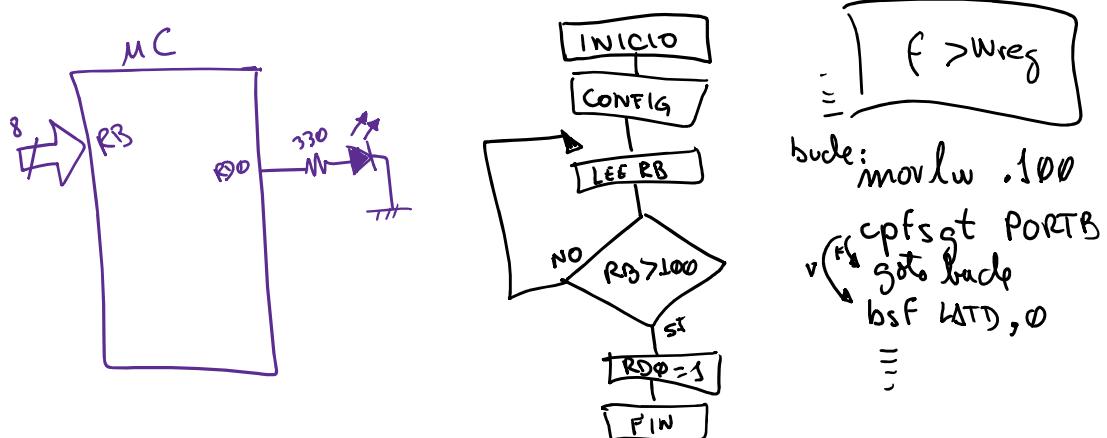
Ejercicio:

- Implementar un comparador de magnitud de dos números de 8 bits:



30

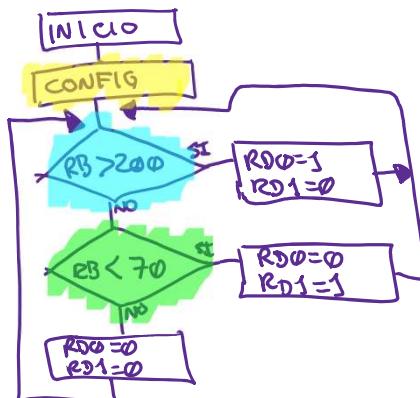
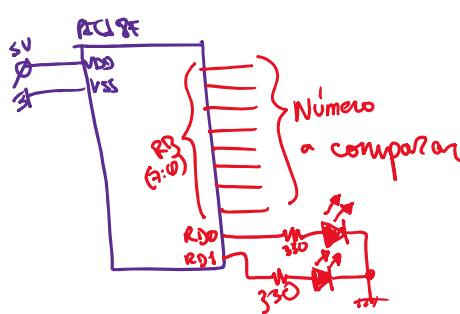
Ejercicio: Leer el valor de RB y colocar a uno el puerto RD0 únicamente cuando dicho valor sea mayor de 100.



31

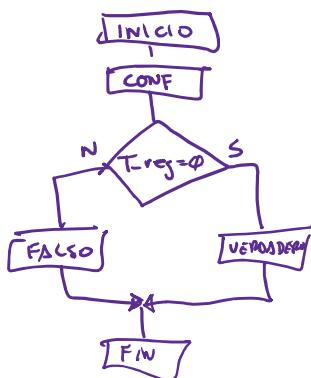
Ejercicio:

- Desarrollar un programa para que compare lo que se está ingresando en RB y arroje lo siguiente: RD0=1 cuando RB>200 y RD1=1 cuando RB<70, cuando no se cumplan las dos condiciones las dos salidas permanecerán en cero.



32

Ejemplo: Pregunta si T_reg (GPR) es igual a cero, si es cierto va a etiqueta verdadero, si no es cierto va a etiqueta falso



Opción 1:

inicio: movlw .0
cpfseq T-reg
gof Falso
gof VERDADERO

Opción 2:

inicio: tstdsz T-reg
gof Falso
gof VERDADERO

Opción 3:

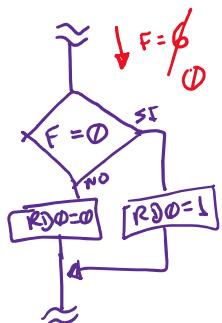
inicio: movf T-reg, W
sublw .0
btfss STATUS, Z
gof Falso
gof VERDADERO

Opción 4:

inicio: movf T-reg, W
sublw .0
b3 VERDADERO
Falso: =
Verdadero: =

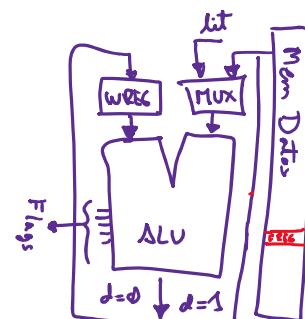
33

Aplicación de sublw para tomas de decisión:



movf F-reg, 0
sublw 0,0
btfss STATUS, 2

$\begin{array}{c} \text{F-reg} \\ \text{---} \\ \text{-F-reg} \\ \text{---} \\ \text{Z} \end{array}$
bit 2 (Z)
¿Se llevan más algun flag?
 $Z=0 \quad N=1$



34

Los saltos BRANCH

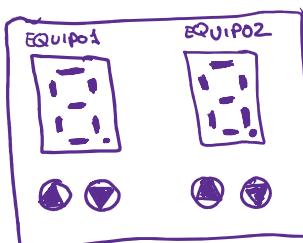
- Son saltos cortos condicionales en base a los flags del registro STATUS

BC	n	Branch if Carry
BN	n	Branch if Negative
BNC	n	Branch if Not Carry
BNN	n	Branch if Not Negative
BNOV	n	Branch if Not Overflow
BNZ	n	Branch if Not Zero
BOV	n	Branch if Overflow
BRA	n	Branch Unconditionally
BZ	n	Branch if Zero

35

Ejercicios adicionales:

- Desarrollar un visualizador de mensaje “SOY FELIZ” a través de un display de siete segmentos del tipo cátodo común a razón de una letra a la vez y con periodo de cambio de 500ms.
- Desarrollar un tablero de score para una cancha deportiva, el cual se tenga dos displays y cuatro pulsadores clasificados en un display y dos pulsadores para cada equipo, un pulsador será para incrementar la cuenta y el otro para decrementar la cuenta.



36

Fin de la sesión