Universidad Nacional Autónoma de México

Facultad de Ingeniería

Microcomputadoras

Profesor: M.I. Rubén Anaya García

Proyecto 2. Manejo de LCD

Suárez Espinoza Mario Alberto

Semestre 2020-2

Entrega: miércoles 1 de abril de 2020

Requerimientos:

1. El proyecto consistirá en un microcontrolador que controlará a una pantalla LCD 16x2.
2. El sistema tendrá 6 modos de funcionamiento diferentes, estos son:
   0) Mostrar “Hola” como mensaje.
   1) Mostrar el nombre de la persona que desarrolló el proyecto. En este caso “Mario Suarez”.
   2) Mostrar el número que ingresa a un puerto del microcontrolador en formato hexadecimal.
   3) Mostrar el número que ingresa a un puerto del microcontrolador en formato binario.
   4) Mostrar el número que ingresa a un puerto del microcontrolador en formato decimal.
   5) Mostrar el dibujo del logo representativo del deporte de la UNAM (un puma).

Diseño:

Los algoritmos utilizados están divididos por modo

Modo 0. Hola

Pseudocódigo:

```
Inicio
Envia el comando 0x80 a la LCD
Envia el ascii del carácter ‘H’ como dato a la LCD
Envia el ascii del carácter ‘o’ como dato a la LCD
Envia el ascii del carácter ‘l’ como dato a la LCD
Envia el ascii del carácter ‘a’ como dato a la LCD
Envia el ascii del carácter ‘ ’ como dato a la LCD
Envia el ascii del carácter ‘M’ como dato a la LCD
Envia el ascii del carácter ‘u’ como dato a la LCD
Envia el ascii del carácter ‘n’ como dato a la LCD
Envia el ascii del carácter ‘d’ como dato a la LCD
Envia el ascii del carácter ‘o’ como dato a la LCD
Envia el ascii del carácter ‘!’ como dato a la LCD
indice ← 5
Mientras indice > 0
     Envia el ascii del carácter ‘ ’ como dato a la LCD
     indice ← indice - 1
Envia el comando 0xC0 a la LCD
indice ← 16
```

```
Mientras indice > 0
     Envia el ascii del carácter ‘ ’ como dato a la LCD
     indice ← indice - 1
Fin
```

Comentarios acerca del pseudocódigo:

El comando 0x80 coloca al apuntador a la DDRAM en la dirección 0x00, mientras que el comando 0xC0 0x80 coloca al apuntador a la DDRAM en la dirección 0x40.

El algoritmo básicamente consiste enviar los caracteres ascii en código duro que se desean imprimir. Para refrescar las posiciones de la LCD cada que se cambia de modo, y que no se traslapen los caracteres entre modo, se imprime el carácter de espacio en todas aquellas posiciones que no se están ocupando, y para ello se ocupan ciclos. El paso de enviar caracteres con espacios se ocupa en todos los modos, así que a partir del siguiente se omitirá en la documentación.

Modo 1. Nombre

Pseudocódigo:

```
Inicio
Envia el comando 0x80 a la LCD
Envia el ascii del caracter ‘M’ como dato a la LCD
Envia el ascii del caracter ‘a’ como dato a la LCD
Envia el ascii del caracter ‘r’ como dato a la LCD
Envia el ascii del caracter ‘i’ como dato a la LCD
Envia el ascii del caracter ‘o’ como dato a la LCD
Envia el ascii del caracter ‘ ’ como dato a la LCD
Envia el ascii del caracter ‘S’ como dato a la LCD
Envia el ascii del caracter ‘u’ como dato a la LCD
Envia el ascii del caracter ‘a’ como dato a la LCD
Envia el ascii del caracter ‘r’ como dato a la LCD
Envia el ascii del caracter ‘e’ como dato a la LCD
Envia el ascii del caracter ‘z’ como dato a la LCD
Fin
```

Comentarios acerca del pseudocódigo:

Es básicamente el mismo algoritmo que para el modo 0, pero con un mensaje diferente.

Modo 2. Hexadecimal

Pseudocódigo:

```
Inicio modo 2
     Envia el comando 0x80 a la LCD
     numerador ← PORTD
     Llama a la subrutina division_hexa
     Decodifica e imprime el número contenido en cocien_hex
     Decodifica e imprime el número contenido en res_hex
     Envia el ascii del caracter 'H' como dato a la LCD

Fin modo 2

Inicio division_hexa
     cocien_hex ← numerador / 0x10
     res_hex ← numerador mod 0x10
Fin division_hexa

Inicio deco_num: //Subrutina para decodificar e imprimir
     Si deco_var < 0x0A
          deco_var ← deco_var + 0x30
     En caso contrario
          deco_var ← deco_var + 0x37
     Envia deco_var como dato a la LCD
Fin deco_num
```

Comentarios acerca del pseudocódigo:

El puerto D es de entrada. En este puerto está conectado el switch de 8 bits que servirá para ingresar un número al sistema. Este número es almacenado en el registro numerador para que, mediante el algoritmo de la división entera, separar los 4 bits más significativos en un registro y los otros 4 bits en otro. El algoritmo consiste en hacer divisiones consecutivas, en este caso entre 0x10, lo cual separara al número en cociente y resto, en el cociente están los 4 bits más significativos, y en el resto los menos.

Al momento de trasladar este algoritmo al microcontrolador PIC16F877A se debe implementar un algoritmo adicional, debido a que este no posee una instrucción para realizar divisiones enteras. Para el caso de este trabajo, se presenta el algoritmo de división entera como un anexo al final del documento. Después de tener dividido al número de entrada en dos registros, estos se envían a un decodificador, el cual trasladará a los números a su correspondiente representación en código ASCII. Debido a la distribución, la decodificación resulta sencilla debido a que los números del 0 al 9 aparecen consecutivos, así como las letras A – E, entonces a los números se les aplica un desplazamiento dependiendo si el número está en el rango [0x01,0x09] o [0x0A,0x0F]. Debido a la naturaleza de este decodificador, es posible ocuparlo también para los números decimales, lo que permite la reutilización de código.

Por último, se imprime el carácter ‘H’ para denotar que se trata de un número hexadecimal.

Modo 3. Binario

Pseudocódigo:

```
Inicio modo 3
     Envia el comando 0x80 a la LCD
     indice ← d’8’
     input ← PORTD
     Mientras indice > 0
          Si bit7(input) = 1
               Envia el ascii del caracter ‘1’ como dato a
               la LCD
          Si no
               Envia el ascii del caracter ‘0’ como dato a
               la LCD
          Rota input a la izquierda
          indice ← indice - 1
     Envia el ascii del caracter ‘B’ como dato a la LCD

Fin modo 3
```

Comentarios acerca del pseudocódigo:

Los datos de entrada son guardados en el registro input. Lo que realiza el algoritmo es recorrer de uno en uno los bits del registro input, y si encuentra un 1, entonces imprime el caracter ‘1’, si encuentra un 0, imprime el caracter ‘0’. Por último, se imprime el carácter ‘B’ para indicar que el número está en formato binario.

Modo 4. Decimal

Pseudocódigo:

```
Inicio modo 4
     Envia el comando 0x80 a la LCD
     numerador ← PORTD
     Llama a la subrutina division_dec1
     numerador ← cocien_hex
     Llama a la subrutina division_dec2
     Decodifica e imprime el número contenido en cocien_dec
     Decodifica e imprime el número contenido en res_dec
     Decodifica e imprime el número contenido en res_hexa
     Envia el ascii del caracter ‘D’ como dato a la LCD

Fin modo 4

Inicio division_dec1
```

```
     cocien_hex ← numerador / 0x0A
     res_hex ← numerador mod 0x0A
Fin division_dec1

Inicio division_dec2
     cocien_dec ← numerador / 0x0A
     res_dec ← numerador mod 0x0A
Fin division_dec2
```

Comentarios acerca del pseudocódigo:

Este algoritmo es muy parecido al algoritmo utilizado en el modo 2, con la diferencia en que las divisiones se hacen entre 0x0A, el cual es la base de un número decimal. Además, por la naturaleza de los números decimales, se tiene que hacer una división extra, quedando dividido el número de 8 bits de la entrada en 3 registros, dónde cada uno contiene las centenas, decenas y unidades respectivamente en representación decimal.

Por último, se decodifican los números con la misma subrutina que se ocupa en el modo 2.

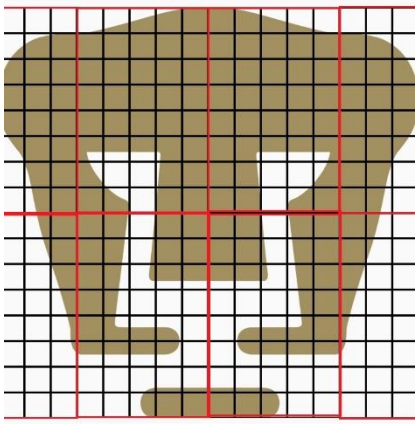
Modo 5. Puma

Consideraciones iniciales:

Para dibujar al puma, se debe representar en formato de pixeles encendidos o apagados. En este caso se definió utilizar una imagen de 16x16 pixeles (cuadrado máximo soportado por la LCD), y para ello se dividió a la imagen del puma en cuadrados de 16x16, posteriormente dependiendo de si el recuadro contenía en su mayoría relleno dorado o blanco es que se encendía o apagaba el píxel. Por último, para poder representar a todos estos bits, es necesario utilizar 8 caracteres definidos por el usuario.

La imagen muestra es la siguiente:

La imagen cuadriculada por caracteres es:



Pseudocódigo:

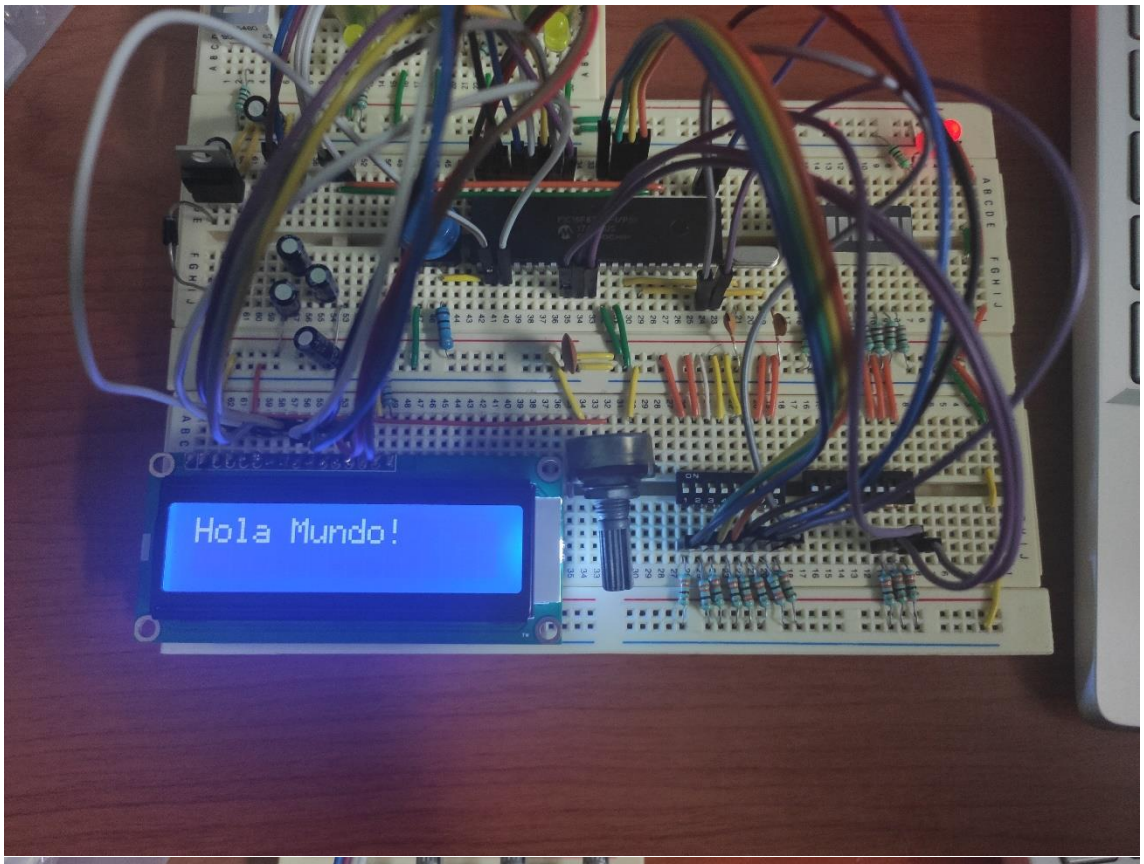
```
Inicio modo 5
     Llama a la subrutina set_custom_characters
     Envia el comando 0x80 a la LCD
     Para i=0x00 hasta i=0x07
          Envia i como dato a la LCD

Fin modo 5

Inicio set_custom_characters
     Envia el comando 0x40
     Envia como dato la codificación en bits de todas las
     filas que conforman a los caracteres persnozalizados
Fin set_custom_characters
```

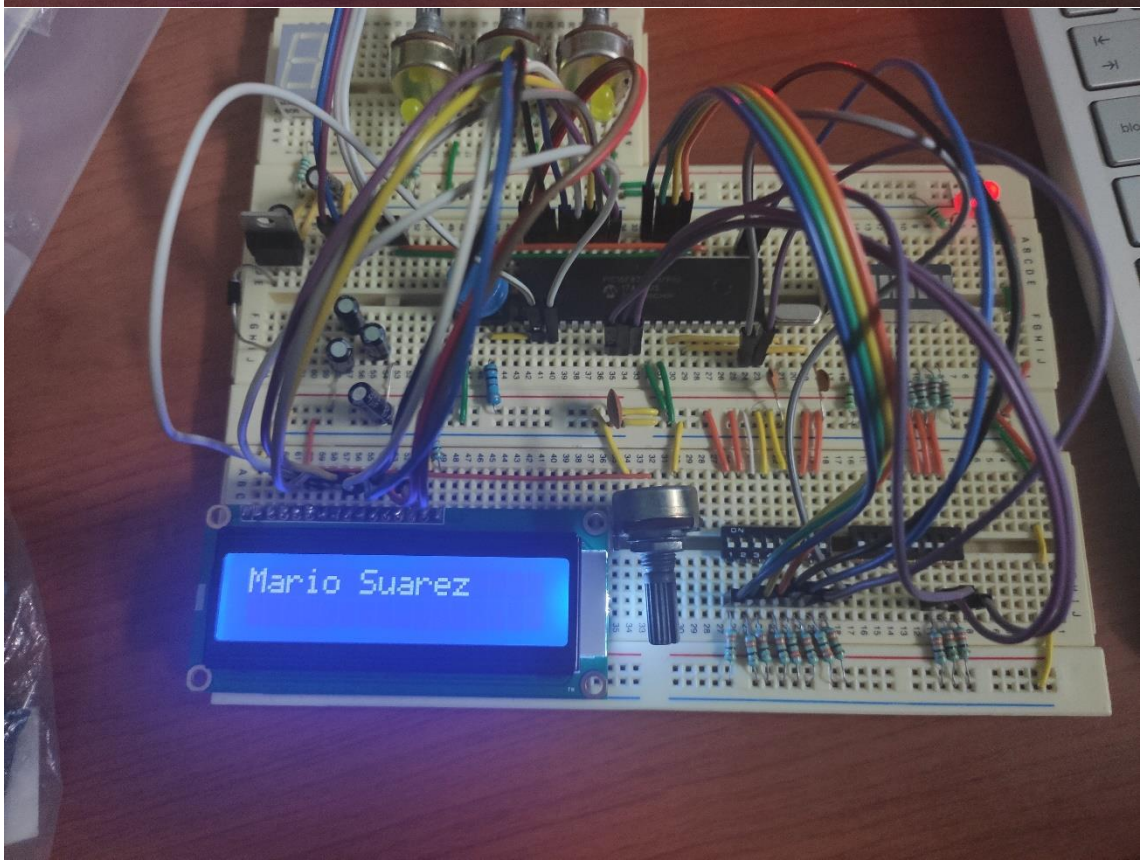
Comentarios acerca del pseudocódigo:

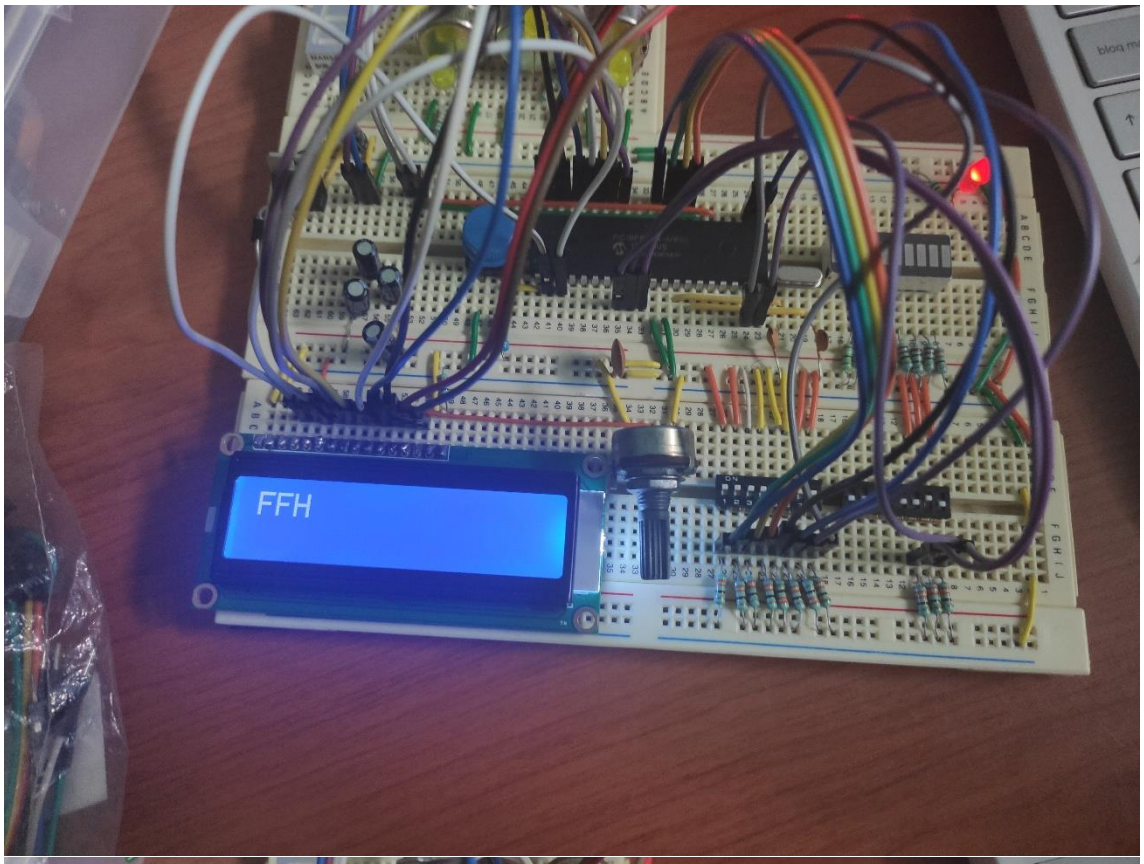
En subrutina set_custom_characters se almacenan los datos de los caracteres personalizados. Se utiliza el comando 0x40 el cual sitúa al puntero de la CGRAM en la dirección 0x00 para para poder empezar a escribir. Para cada fila de un carácter, se envía una codificación en 5 bits como dato a la LCD. Después de terminar la ejecución de la subrutina set_custom_characters, estos se imprimen de igual manera que los caracteres predefinidos (los almacenados en la CGROM). La dirección del primer caracter definido por el usuario es 0x00 y la última es la 0x07. Para esta aplicación se imprimen todos los caracteres predefinidos por el usuario.

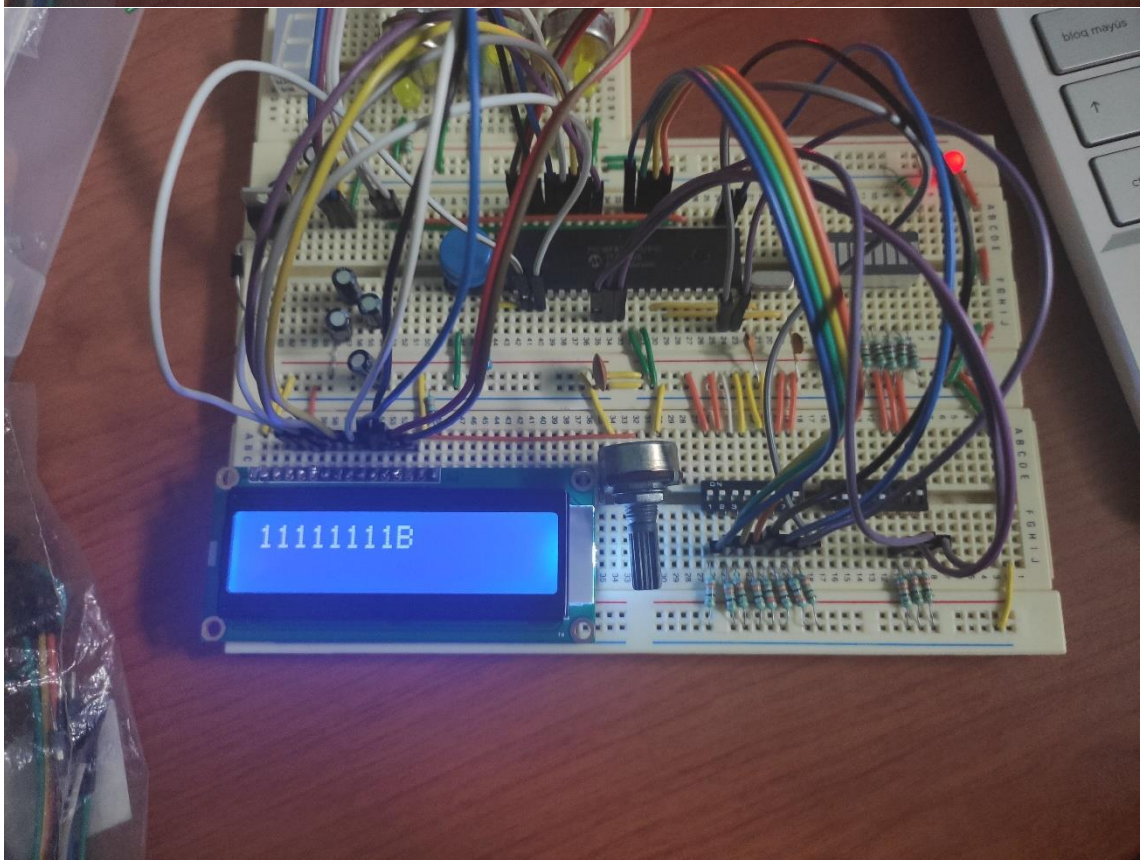
Resultados obtenidos:

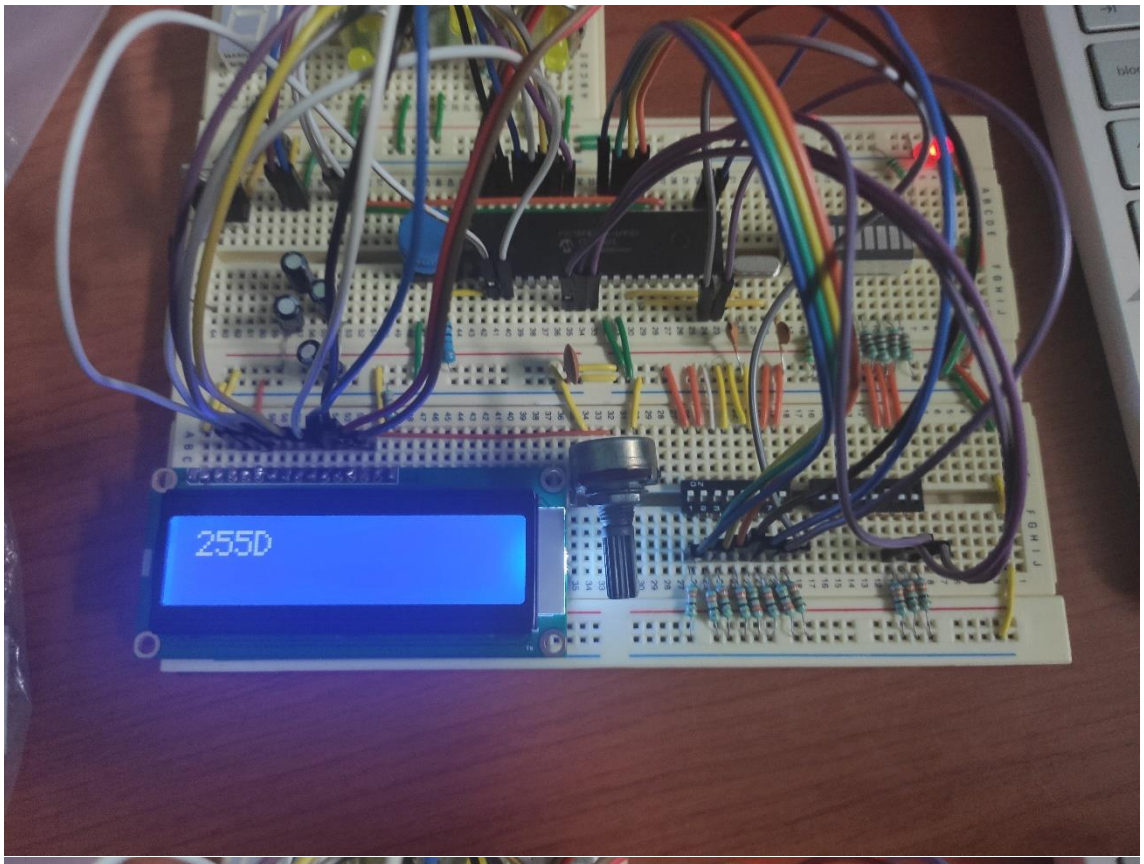
Se agregan imágenes de su funcionamiento

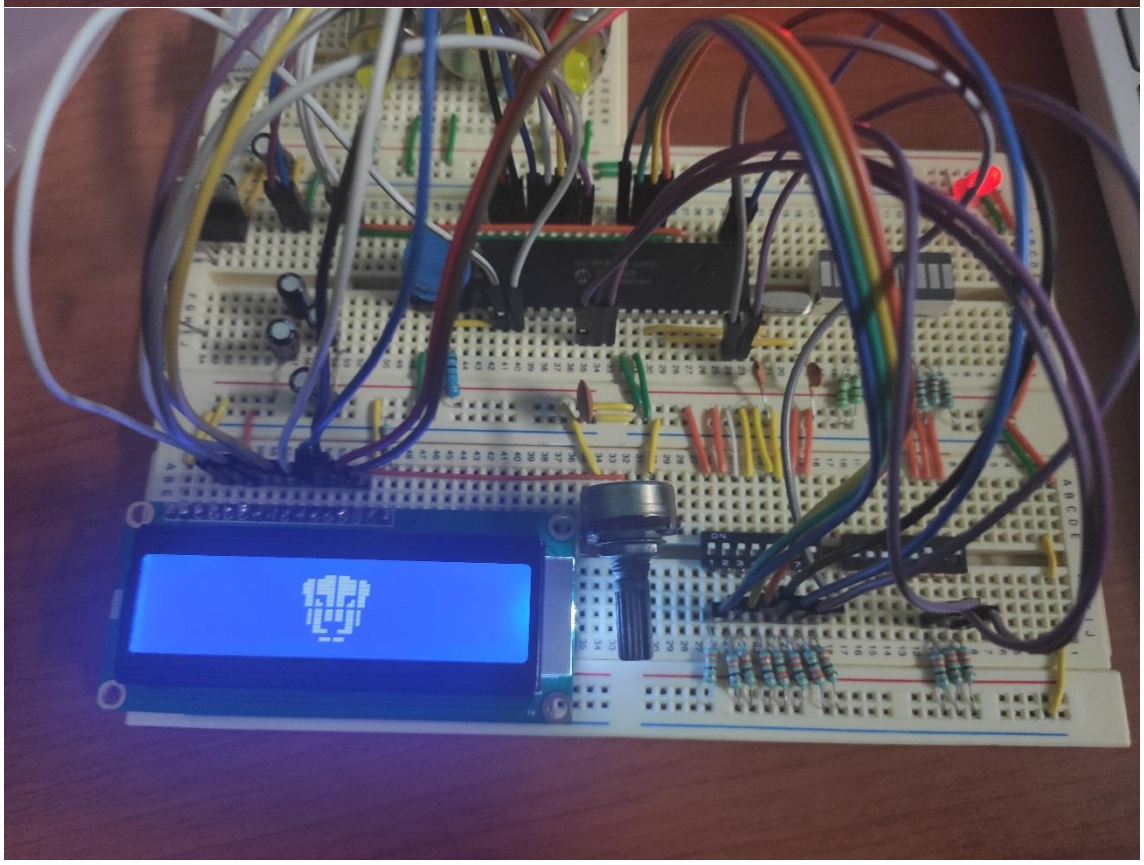

Hola Mundo!


Mario Suarez


FFH


11111111B


255D



Comentarios:

Considero que la realización de este proyecto fue complicada porque se tuvieron que realizar varios módulos por cada caso que se tenía. En lo personal, el entender el funcionamiento de la pantalla LCD fue difícil también, debido a que se tienen muchos comandos y su arquitectura es compleja, así que además del manual que se nos proporcionó, tuve que investigar bastante en internet, y al final se logró el cometido.

Una de las partes interesantes que tuve que resolver fue que para muchos algoritmos necesite usar divisiones, operaciones que no están incluidas dentro de las instrucciones base del microcontrolador, así que tuve que generar mis propios algoritmos utilizando operaciones de resta y suma.

Por último, creo que el poder dibujar diferentes caracteres personalizados en la LCD puede traer un sinfín de aplicaciones, entre las que destaca el programar juegos básicos.

Código ASM:

```
;Mario ALberto Suárez Espinoza
;masues64@gmail.com

; PORTB BUS DE DATOS B0-D0 ... B7-D7
; RS - A0
; E - A1
; R/W - GND

;Puerto A: Control display (2 pines de 6)
;Puerto B: Datos display (8 pines de 8)
;Puerto D: Entrada de datos al uC (8 pines de 8)
;Puerto E: Selección de modo (2 pines de 3)

    processor 16f877
    include<p16f877.inc>
valor equ h'20'
valor1 equ h'21'
valor2 equ h'22'
contador equ h'23'
dato equ h'24'
constA:  equ d'255'
constB:  equ d'255'
constC:  equ d'255'
```

```
regA:  equ h'25'
regB:  equ h'26'
regC:  equ h'27'
indice: equ h'28'
input: equ h'29'
deco_var: equ h'30'
numerador:  equ h'30'; registro que almacena al numerador
deno_hex:  equ h'10'; literal denominador hexadecimal
deno_dec:  equ h'0A'; literal denominador decimal
cocien_hex:  equ h'34'; registro que almacena el resultado de la operació
n hexadecimal
res_hex:  equ h'33'; registro que almacena al resto hexadecimal
cocien_dec:  equ h'31'; registro que almacena el resultado de la operació
n hexadecimal
res_dec:  equ h'32'; registro que almacena al resto hexadecimal

    org 0
    goto inicio
    org 5
inicio
    clrf PORTA
    CLRF PORTB
    banksel TRISB
    movlw b'000000000'
    movwf TRISB
    banksel ADCON1
    movlw 0x07
    movwf ADCON1
    banksel TRISA
    movlw b'000000000'
    movwf TRISA
    banksel TRISD
    movlw h'FF'
    movwf TRISD
    banksel TRISE
    movlw b'111'
    movwf TRISE

    bcf STATUS, RP1; Cambia a banco 0
    bcf STATUS, RP0

    call inicia_lcd

main:
    ;Caso0
    movlw h'00'
    subwf PORTE,0
    btfsc STATUS, Z
    goto caso0
```

```
    ;Caso1
    movlw h'01'
    subwf PORTE,0
    btfsc STATUS, Z
    goto caso1
    ;Caso2
    movlw h'02'
    subwf PORTE,0
    btfsc STATUS, Z
    goto caso2
    ;Caso3
    movlw h'03'
    subwf PORTE,0
    btfsc STATUS,Z
    goto caso3
    ;Caso4
    movlw h'04'
    subwf PORTE,0
    btfsc STATUS,Z
    goto caso4
    ;Caso5
    movlw h'05'
    subwf PORTE,0
    btfsc STATUS,Z
    goto caso5

    goto main


caso0: ; Imprime Hola Mundo
    movlw 0x80
    call comando
    movlw a'H'
    call datos
    movlw a'o'
    call datos
    movlw a'l'
    call datos
    movlw a'a'
    call datos
    movlw a' '
    call datos
    movlw a'M'
    call datos
    movlw a'u'
    call datos
    movlw a'n'
    call datos
    movlw a'd'
```

```
    call datos
    movlw a'o'
    call datos
    movlw a'!'
    call datos
    ;Ciclo para imprimir 6 espacios
    movlw d'5'
    movwf indice
c0_loop:
    movlw a' '
    call datos
c0_endLoop:
    decfsz indice
    goto c0_loop

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 16 espacios
    movlw d'16'
    movwf indice
c0_2_loop:
    movlw a' '
    call datos
c0_2_endLoop:
    decfsz indice
    goto c0_2_loop

    goto main


caso1:  ; Imprime Mario Suarez
    movlw 0x80
    call comando
    movlw a'M'
    call datos
    movlw a'a'
    call datos
    movlw a'r'
    call datos
    movlw a'i'
    call datos
    movlw a'o'
    call datos
    movlw a' '
    call datos
    movlw a'S'
    call datos
    movlw a'u'
```

```
    call datos
    movlw a'a'
    call datos
    movlw a'r'
    call datos
    movlw a'e'
    call datos
    movlw a'z'
    call datos
    ;Ciclo para imprimir 4 espacios
    movlw d'4'
    movwf indice
c1_loop:
    movlw a' '
    call datos
c1_endLoop:
    decfsz indice
    goto c1_loop

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 16 espacios
    movlw d'16'
    movwf indice
c1_2_loop:
    movlw a' '
    call datos
c1_2_endLoop:
    decfsz indice
    goto c1_2_loop

    goto main


caso2:  ; Imprime entrada en hexadecimal
    movlw 0x80
    call comando
    ;Salva la entrada en el registro numerador
    movf PORTD,0
    movwf numerador
    ;Divide el contenido del numerador en dos registros, cocien_hex y res
_hex
    call division_hexa
    ;Decodifica a las variables para imprimirlas
    movf cocien_hex,0
    call deco_num
    movf res_hex,0
    call deco_num
```

```
    movlw a'H'
    call datos
    ;Ciclo para imprimir 13 espacios
    movlw d'13'
    movwf indice
c2_loop:
    movlw a' '
    call datos
c2_endLoop:
    decfsz indice
    goto c2_loop

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 16 espacios
    movlw d'16'
    movwf indice
c2_2_loop:
    movlw a' '
    call datos
c2_2_endLoop:
    decfsz indice
    goto c1_2_loop

    goto main

division_hexa:
    clrf cocien_hex; coloca un cero en concien_hex

    movf numerador,0; coloca el contenido de numerador en W
    movwf res_hex; mueve el contenido de W a res_hex
    xorlw 0x00; compara el contenido de W con 0
    btfsc STATUS, Z; revisa si fue cero la operación anterior
    return; termina el algoritmo porque el numerador fue cero

    movlw deno_hex; coloca deno_hex en W
    subwf numerador,0; a 'numerador' le resta el contenido de W y lo alma
cena en W
    btfsc STATUS, C; revisa si 'deno_hex' es mayor que 'numerador'
    goto numerador_hexa_mayor; numerador>=deno_hex
    return; numerador<deno_hex, termina el algoritmo

numerador_hexa_mayor:
    movf numerador,0; mueve el contenido de numerador a W
    movwf res_hex; guarda el contenido de W en res_hex
    goto loop_division_hex

loop_division_hex:
```

```
    movlw deno_hex; coloca deno_hex en W
    subwf res_hex,1; resta el contenido de W a res_hex, y se almacena el
resultado en res_hex
    incf cocien_hex,1; incrementa en 1 a cocien_hex, el resultado se alma
cena en cocien_hex
    movlw deno_hex; coloca deno_hex en W
    subwf res_hex,0; a 'res_hex' le resta el contenido de W y lo almacena
 en W
    btfsc STATUS, C; revisa si 'y' es mayor que 'x'
    goto loop_division_hex; c>=y
    return;termina el algoritmo

deco_num: ;Subrutina que imprime el número hexadecimal en la pantalla LCD
    movwf deco_var
    movlw h'0A'; coloca 0x0A en W
    subwf deco_var,0; a 'deco_var' le resta el contenido de W y lo almace
na en W
    btfss STATUS, C; revisa si '0x0A' es mayor o igual que 'deco_var'
    goto imp_dec; deco_var<0x0A, por lo tanto imprime  deco_var + 0x30
    goto imp_hex; deco_var>=0x0A, por lo tanto imprime  deco_var + 0x37
imp_dec:
    movlw h'30'; W <- 0x30
    addwf deco_var,0; W <- deco_var + W
    call datos
    return
imp_hex:
    movlw h'37'; W <- 0x37
    addwf deco_var,0; W <- deco_var + W
    call datos
    return


caso3:  ; Imprime la entrada en binario
    movlw 0x80
    call comando
    ;Ciclo para recorrer todos los bits de los datos de entrada
    movlw d'8'
    movwf indice
    movf PORTD,0
    movwf input
c3_loop:
    btfss input,7
    goto c3_0
    goto c3_1
c3_endLoop:
    call datos
    rlf input,1
    decfsz indice
    goto c3_loop
```

```
    movlw a'B'
    call datos
    ;Ciclo para imprimir 7 espacios
    movlw d'7'
    movwf indice
c3_loop2:
    movlw a' '
    call datos
c3_endLoop2:
    decfsz indice
    goto c3_loop2

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 16 espacios
    movlw d'16'
    movwf indice
c3_2_loop:
    movlw a' '
    call datos
c3_2_endLoop:
    decfsz indice
    goto c3_2_loop

    goto main

c3_0:
    movlw a'0'
    goto c3_endLoop
c3_1:
    movlw a'1'
    goto c3_endLoop


caso4:  ; Imprime entrada en decimal
    movlw 0x80
    call comando
    ;Salva la entrada en el registro numerador
    movf PORTD,0
    movwf numerador
    ;Divide el contenido del numerador en dos registros, cocien_hex y res
_hex
    call division_dec1
    ;Salva el contenido del cocien_hex en el regitro numerador
    movf cocien_hex,0
    movwf numerador
```

```
    ;Divide el contenido de cocien_hex en dos registros, cocien_dec y res
_dec
    call division_dec2
    ;Decodifica a las variables para imprimirlas
    movf cocien_dec,0
    call deco_num
    movf res_dec,0
    call deco_num
    movf res_hex,0
    call deco_num
    movlw a'D'
    call datos
    ;Ciclo para imprimir 12 espacios
    movlw d'12'
    movwf indice
c4_loop:
    movlw a' '
    call datos
c4_endLoop:
    decfsz indice
    goto c4_loop

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 16 espacios
    movlw d'16'
    movwf indice
c4_2_loop:
    movlw a' '
    call datos
c4_2_endLoop:
    decfsz indice
    goto c4_2_loop

    goto main

division_dec1:
    clrf cocien_hex; coloca un cero en concien_hex

    movf numerador,0; coloca el contenido de numerador en W
    movwf res_hex; mueve el contenido de W a res_hex
    xorlw 0x00; compara el contenido de W con 0
    btfsc STATUS, Z; revisa si fue cero la operación anterior
    return; termina el algoritmo porque el numerador fue cero

    movlw deno_dec; coloca deno_dec en W
    subwf numerador,0; a 'numerador' le resta el contenido de W y lo alma
cena en W
```

```
    btfsc STATUS, C; revisa si 'deno_dec' es mayor que 'numerador'
    goto numerador_dec1_mayor; numerador>=deno_dec
    return; numerador<deno_dec, termina el algoritmo

numerador_dec1_mayor:
    movf numerador,0; mueve el contenido de numerador a W
    movwf res_hex; guarda el contenido de W en res_hex
    goto loop_division_dec1

loop_division_dec1:
    movlw deno_dec; coloca deno_dec en W
    subwf res_hex,1; resta el contenido de W a res_hex, y se almacena el
resultado en res_hex
    incf cocien_hex,1; incrementa en 1 a cocien_hex, el resultado se alma
cena en cocien_hex
    movlw deno_dec; coloca deno_dec en W
    subwf res_hex,0; a 'res_hex' le resta el contenido de W y lo almacena
 en W
    btfsc STATUS, C; revisa si 'y' es mayor que 'x'
    goto loop_division_dec1; c>=y
    return;termina el algoritmo

division_dec2:
    clrf cocien_dec; coloca un cero en concien_hex

    movf numerador,0; coloca el contenido de numerador en W
    movwf res_dec; mueve el contenido de W a res_dec
    xorlw 0x00; compara el contenido de W con 0
    btfsc STATUS, Z; revisa si fue cero la operación anterior
    return; termina el algoritmo porque el numerador fue cero

    movlw deno_dec; coloca deno_dec en W
    subwf numerador,0; a 'numerador' le resta el contenido de W y lo alma
cena en W
    btfsc STATUS, C; revisa si 'deno_dec' es mayor que 'numerador'
    goto numerador_dec2_mayor; numerador>=deno_dec
    return; numerador<deno_dec, termina el algoritmo

numerador_dec2_mayor:
    movf numerador,0; mueve el contenido de numerador a W
    movwf res_dec; guarda el contenido de W en res_dec
    goto loop_division_dec2

loop_division_dec2:
    movlw deno_dec; coloca deno_dec en W
    subwf res_dec,1; resta el contenido de W a res_dec, y se almacena el
resultado en res_dec
    incf cocien_dec,1; incrementa en 1 a cocien_dec, el resultado se alma
cena en cocien_dec
```

```
    movlw deno_dec; coloca deno_dec en W
    subwf res_dec,0; a 'res_dec' le resta el contenido de W y lo almacena
 en W
    btfsc STATUS, C; revisa si 'y' es mayor que 'x'
    goto loop_division_dec2; c>=y
    return;termina el algoritmo


caso5:
    ; Inicializa a los caracteres que se van a ocupar para imprimir al pu
ma
    call set_custom_characters

    movlw 0x80; Selecciona la dirección 0x00 de la DDRAM
    call comando

    ;Ciclo para imprimir 6 espacios
    movlw d'6'
    movwf indice
c5_1_loop:
    movlw a' '
    call datos
c5_1_endLoop:
    decfsz indice
    goto c5_1_loop
    movlw h'00'; código de la primera dirección de la CGRAM
    call datos
    movlw h'01'
    call datos
    movlw h'02'
    call datos
    movlw h'03'
    call datos
    ;Ciclo para imprimir 6 espacios
    movlw d'6'
    movwf indice
c5_2_loop:
    movlw a' '
    call datos
c5_2_endLoop:
    decfsz indice
    goto c5_2_loop

    movlw 0xC0; Selecciona la dirección 0x40 de la DDRAM
    call comando

    ;Ciclo para imprimir 6 espacios
    movlw d'6'
    movwf indice
```

```
c5_3_loop:
    movlw a' '
    call datos
c5_3_endLoop:
    decfsz indice
    goto c5_3_loop
    movlw h'04'; código de la primera dirección de la CGRAM
    call datos
    movlw h'05'
    call datos
    movlw h'06'
    call datos
    movlw h'07'
    call datos
    ;Ciclo para imprimir 6 espacios
    movlw d'6'
    movwf indice
c5_4_loop:
    movlw a' '
    call datos
c5_4_endLoop:
    decfsz indice
    goto c5_4_loop

    goto main

set_custom_characters:
    movlw 0x40; selecciona a la primera dirección de la CGRAM
    call comando
    ;Caracter 1
    movlw b'00000'
    call datos
    movlw b'00011'
    call datos
    movlw b'00111'
    call datos
    movlw b'00111'
    call datos
    movlw b'00111'
    call datos
    movlw b'00011'
    call datos
    movlw b'00011'
    call datos
    movlw b'00011'
    call datos
    ;Caracter 2
    movlw b'00111'
    call datos
```

```
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'00011'
call datos
movlw b'10011'
call datos
movlw b'11011'
call datos
;Caracter 3
movlw b'11100'
call datos
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'11111'
call datos
movlw b'11000'
call datos
movlw b'11001'
call datos
movlw b'11011'
call datos
;Caracter 4
movlw b'00000'
call datos
movlw b'11000'
call datos
movlw b'11100'
call datos
movlw b'11100'
call datos
movlw b'11100'
call datos
movlw b'11000'
call datos
movlw b'11000'
call datos
movlw b'11000'
call datos
;Caracter 5
```

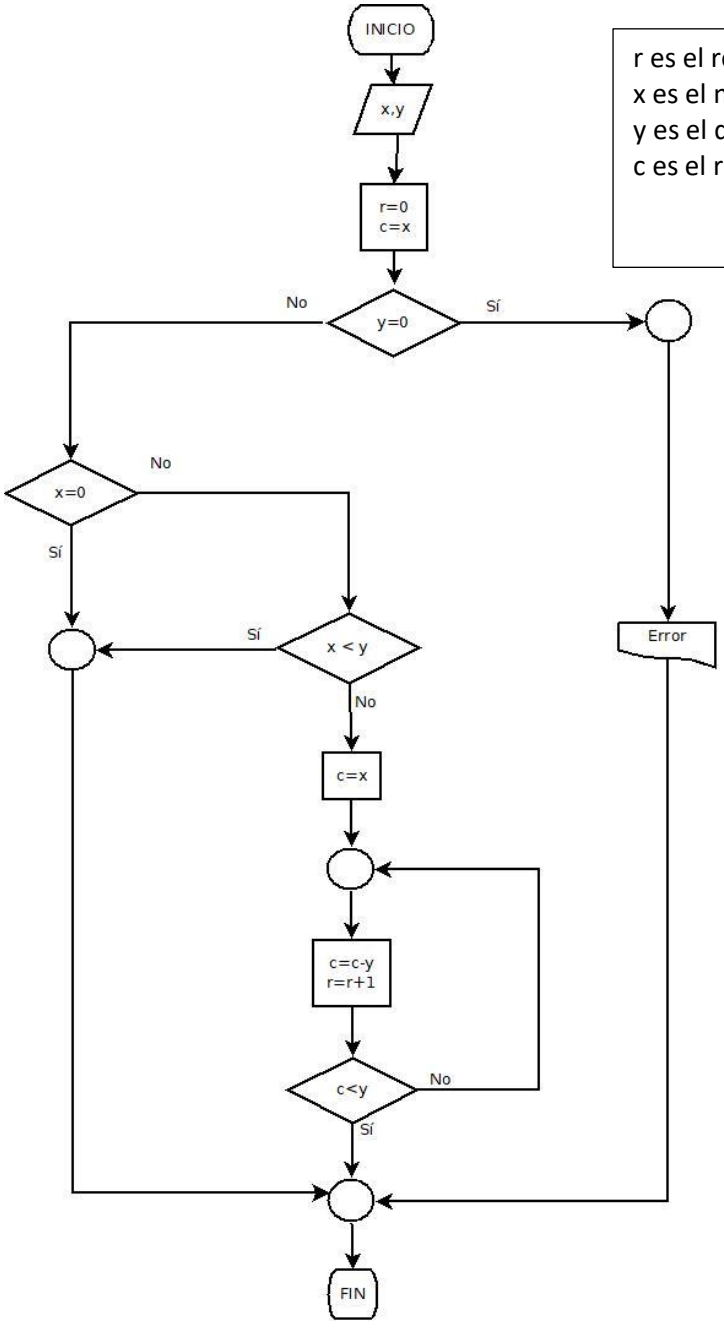
```
movlw b'00001'
call datos
movlw b'00001'
call datos
movlw b'00001'
call datos
movlw b'00001'
call datos
movlw b'00000'
call datos
movlw b'00000'
call datos
movlw b'00000'
call datos
movlw b'00000'
call datos
;Caracter 6
movlw b'11011'
call datos
movlw b'11011'
call datos
movlw b'11011'
call datos
movlw b'11000'
call datos
movlw b'11000'
call datos
movlw b'11110'
call datos
movlw b'00000'
call datos
movlw b'00111'
call datos
;Caracter 7
movlw b'11011'
call datos
movlw b'11011'
call datos
movlw b'11011'
call datos
movlw b'00011'
call datos
movlw b'00011'
call datos
movlw b'01111'
call datos
movlw b'00000'
call datos
movlw b'11100'
```

```
    call datos
    ;Caracter 8
    movlw b'10000'
    call datos
    movlw b'10000'
    call datos
    movlw b'10000'
    call datos
    movlw b'10000'
    call datos
    movlw b'00000'
    call datos
    movlw b'00000'
    call datos
    movlw b'00000'
    call datos
    movlw b'00000'
    call datos

    return


inicia_lcd
    movlw 0x30
    call comando
    call ret100ms
    movlw 0x30
    call comando
    call ret100ms
    movlw 0x38
    call comando
    movlw 0x0c
    call comando
    movlw 0x01
    call comando
    movlw 0x06
    call comando
    movlw 0x02
    call comando
    return

comando
    movwf PORTB
    call ret200
    bcf PORTA,0
    bsf PORTA,1
    call ret200
    bcf PORTA,1
    return
```

```
datos
    movwf PORTB
    call ret200
    bsf PORTA,0
    bsf PORTA,1
    call ret200
    bcf PORTA,1
    call ret200
    call ret200
    return

ret200
    movlw 0x02
    movwf valor1
loop
    movlw d'164'
    movwf valor
loop1
    decfsz valor,1
    goto loop1
    decfsz valor1,1
    goto loop
    return

ret100ms
    movlw 0x03
rr  movwf valor
tres movlw 0xff
    movwf valor1
dos movlw 0xff
    movwf valor2
uno decfsz valor2
    goto uno
    decfsz valor1
    goto dos
    decfsz valor
    goto tres
    return
    end
```

Anexo:

Se agrega el algoritmo utilizado para la división entera en ensamblador.



r es el registro donde se guarda el resultado
x es el numerador
y es el denominador
c es el resto