



# Universidad Nacional Autónoma de México

## Facultad de Ingeniería

## Microcomputadoras

## Profesor: M.I. Rubén Anaya García

# *Proyecto 3. Vólmetro Digital*

Suárez Espinoza Mario Alberto

Semestre 2020-2

Entrega: lunes 27 de abril de 2020

# Requerimientos:

1. El proyecto consistirá en un microcontrolador que controlará a una pantalla LCD 16x2.
2. El sistema tendrá 4 modos de funcionamiento diferentes, estos son:
   0) Despliegue de la conversión A/D en formato Decimal.
   1) Despliegue de la conversión A/D en formato Hexadecimal.
   2) Despliegue de la conversión A/D en formato Binario.
   3) Despliegue de la conversión A/D en Volts (usando tres dígitos).

# Diseño:

Los algoritmos utilizados están divididos por modo

Modo 0. Decimal

Pseudocódigo:

```
Inicio modo 0
     Envia el comando 0x80 a la LCD
     Llama a la subrutina conversionAD
     numerador ← W
     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 0

Inicio conversionAD
     Inicia la conversión Análogo-Digital
     Espera hasta que la conversión haya finalizado
     W ← ADRESH
Fin conversionAD

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:

El algoritmo lee los datos de la entrada análoga, y posteriormente transforma el dato leído a decimal. Una vez transformado el dato, este se decodifica e imprime en la LCD.

Modo 1. Hexadecimal

Pseudocódigo:

```
Inicio modo 1
     Envia el comando 0x80 a la LCD
     Llama a la subrutina conversionAD
     numerador ← W
     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 1

Inicio conversionAD
     Inicia la conversión Análogo-Digital
     Espera hasta que la conversión haya finalizado
     W ← ADRESH
Fin conversionAD


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 algoritmo lee los datos de la entrada análoga, y posteriormente transforma el dato leído a hexadecimal. Una vez transformado el dato, este se decodifica e imprime en la LCD.

Modo 2. Binario

Pseudocódigo:

```
Inicio modo 2
     Envia el comando 0x80 a la LCD
     Llama a la subrutina conversión AD
     indice ← d'8'
     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 2

Inicio conversionAD
     Inicia la conversión Análogo-Digital
     Espera hasta que la conversión haya finalizado
     W ← ADRESH
Fin conversionAD
```

Comentarios acerca del pseudocódigo:

El algoritmo lee los datos de la entrada análoga, y posteriormente transforma el dato leído a binario. Una vez transformado el dato, este se decodifica e imprime en la LCD.

Modo 3. Voltaje

Pseudocódigo:

```
Inicio modo 3
     Envia el comando 0x80 a la LCD
     Llama a la subrutina conversionAD
     contaVolt <- W
     Llama a la subrutina toVolts
     Decodifica e imprime el número contenido en regVol2
     Envia el ascii del carácter '.' como dato a la LCD
     Decodifica e imprime el número contenido en regVol1
     Decodifica e imprime el número contenido en regVol0
     Envia el ascii del carácter 'V' como dato a la LCD

Fin modo 3

Inicio conversionAD
     Inicia la conversión Análogo-Digital
     Espera hasta que la conversión haya finalizado
     W ← ADRESH
Fin conversionAD
```

```
Inicio toVolts
     reg2 ← 0
     reg1 ← 0
     reg0 ← 0
     Mientras contaVolt>0:
       Si regVol0 = 8
         regVol0 ← 0
         Si regVol1 = 9
           regVol1 ← 0
           Si regVol2 = 9
             regVol2 ← 0
           En caso contrario
             Incrementa regVol2
         En caso contrario
           Incrementa regVol1
       En caso contrario
         regVol0 ← regVol0 + 2
       Decrementa contaVolt
Fin toVolts
```

Comentarios acerca del pseudocódigo:

El algoritmo lee los datos de la entrada análoga, y posteriormente transforma el dato leído a volts. La conversión a volts se basa en el hecho de que la resolución del conversor es $res = \frac{5V-0V}{2^8} = 0.019531[V] \approx 0.02[V]$. Entonces, para conocer el valor de un número determinado, basta con multiplicar la resolución por tal número. Se establece entonces la función $y(x) = x * res$, dónde $res = 0.02$, $x$ es el número leído, $y$ es el número leído en Volts.

Para lograr la implementación de esta función, se realiza un contador, de 2 en 2, que utiliza 3 registros. Cada registro representa un dígito en decimal, por lo que se debe validar que llegue a máximo 9.

Una vez obtenido el valor en Volts, se decodifica e imprime con el formato deseado <dígito más significativo><.><dígito medio><dígito menos significativo><V>.

# Comentarios:

Para este proyecto, básicamente se usó el mismo código que en el proyecto pasado, por lo que era importante que el anterior funcionase y tuviese una arquitectura escalable. La adición mayor fue el uso del convertidor analógico - digital y transformar la salida de este convertidor a voltaje.

En lo personal, transformar a voltaje fue un proceso muy complicado, debido a que la forma más fácil de hacerlo era médiate una regla de tres (o regla de proporción), sin embargo, para poder realizar esto era necesario tener operaciones: suma, resta, multiplicación y división de al menos 16 bits, cosa que no había implementado previamente, y el implementarla me tomaría modificar la mayor parte del código que ya había escrito en el proyecto pasado. Para solucionar esto preferí utilizar un algoritmo contador, en el que por cada cuenta que se realiza, se suma un múltiplo de la sensibilidad del convertidor. Creo que este algoritmo no fue tan difícil de implementar y no fue necesario reescribir mi código pasado. Lo más difícil para mi fue pensar en esta alternativa.

# Código ASM:

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

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

; A3 entrada analógica

;Puerto A: Entrada analógica y control display (3 pines de 6)
;Puerto B: Datos display (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
regVol2: equ h'33'; registro que almacena a la cifra más significativa de
l voltaje
regVol1: equ h'34'; registro que almacena a la cifra media del voltaje
regVol0: equ h'35'; registro que almacena a la cifra más significativa de
l voltaje
contaVolt: equ h'36'; contador auxiliar para transformar a volts
```

```
    org 0
    goto inicio
    org 5
inicio
    clrf PORTA
    CLRF PORTB
    banksel TRISB
    movlw b'000000000'
    movwf TRISB
    banksel TRISA
    movlw h'07'
    movwf ADCON1
    banksel TRISA
    movlw b'000000100'
    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

    ;frecuencia de reloj frc
    ;entrada analógica 2
    ;enciende al convertidor AD
    movlw b'11010001'; configuración de adcon0
    movwf ADCON0

    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

    goto main

caso0:  ; Imprime entrada en decimal
    movlw 0x80
    call comando
    call conversionAD ; Transforma la entrada análoga a digital
    movwf numerador; Salva la entrada en el registro 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
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

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

conversionAD:
    bsf ADCON0, 2; inicia la conversión Análogo - Digital
    btfsc ADCON0, 2; pregunta si terminó la conversión
    goto $-1; si no ha terminado, se queda en un bucle
    ;Salva la entrada en el registro W
    movf ADRESH, 0; W <- ADRESH
    return

caso1:  ; Imprime entrada en hexadecimal
    movlw 0x80
    call comando

    call conversionAD ; Transforma la entrada análoga a digital

    ;Salva la entrada en el registro numerador
    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
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

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


caso2:  ; Imprime la entrada en binario
    movlw 0x80
    call comando

    call conversionAD ; Transforma la entrada análoga a digital
    movwf input
    ;Ciclo para recorrer todos los bits de los datos de entrada
    movlw d'8'
    movwf indice
c2_loop:
    btfss input,7
    goto c2_0
    goto c2_1
c2_endLoop:
    call datos
    rlf input,1
```

```
    decfsz indice
    goto c2_loop

    movlw a'B'
    call datos
    ;Ciclo para imprimir 7 espacios
    movlw d'7'
    movwf indice
c2_loop2:
    movlw a' '
    call datos
c2_endLoop2:
    decfsz indice
    goto c2_loop2

    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 c2_2_loop

    goto main

c2_0:
    movlw a'0'
    goto c2_endLoop
c2_1:
    movlw a'1'
    goto c2_endLoop


caso3:
    movlw 0x80
    call comando

    call conversionAD ; Transforma la entrada análoga a digital
    movwf contaVolt; contaVolt <- W
    incf contaVolt;
    call toVolts; transforma a volts

    ;Decodifica a las variables para imprimirlas
    movf regVol2,0
```

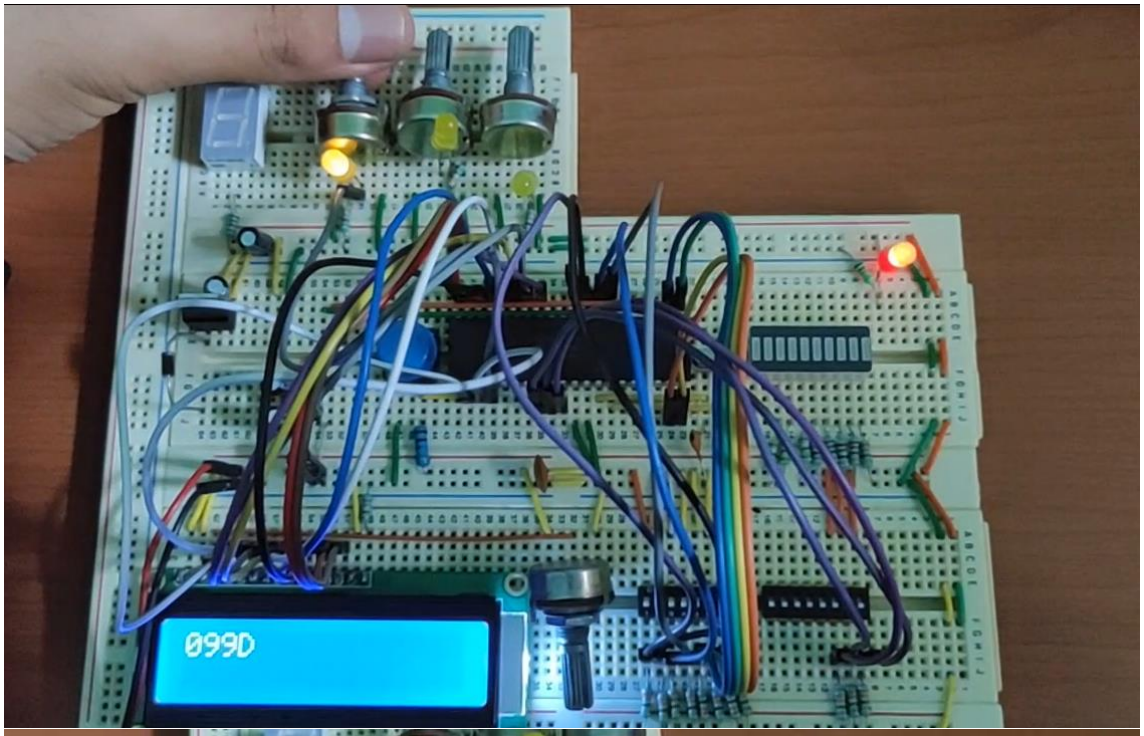
```
    call deco_num
    movlw a'.'
    call datos
    movf regVol1,0
    call deco_num
    movf regVol0,0
    call deco_num
    movlw a'V'
    call datos

    ;Ciclo para imprimir 7 espacios
    movlw d'11'
    movwf indice
c3_loop:
    movlw a' '
    call datos
c3_endLoop:
    decfsz indice
    goto c3_loop

    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

toVolts:
    clrf regVol2; regVol2 <- 0
    clrf regVol1; regVol1 <- 0
    clrf regVol0; regVol0 <- 0
finLoopVolt:
    decfsz contaVolt,1
    goto loopVolt
    return; termina el algoritmo
loopVolt:
    movlw h'08'
    subwf regVol0,0
    btfss STATUS, Z; ¿regVol0 = 8?
    goto noRegVol08; regVol0 != 8
    clrf regVol0; regVol0 = 8
```

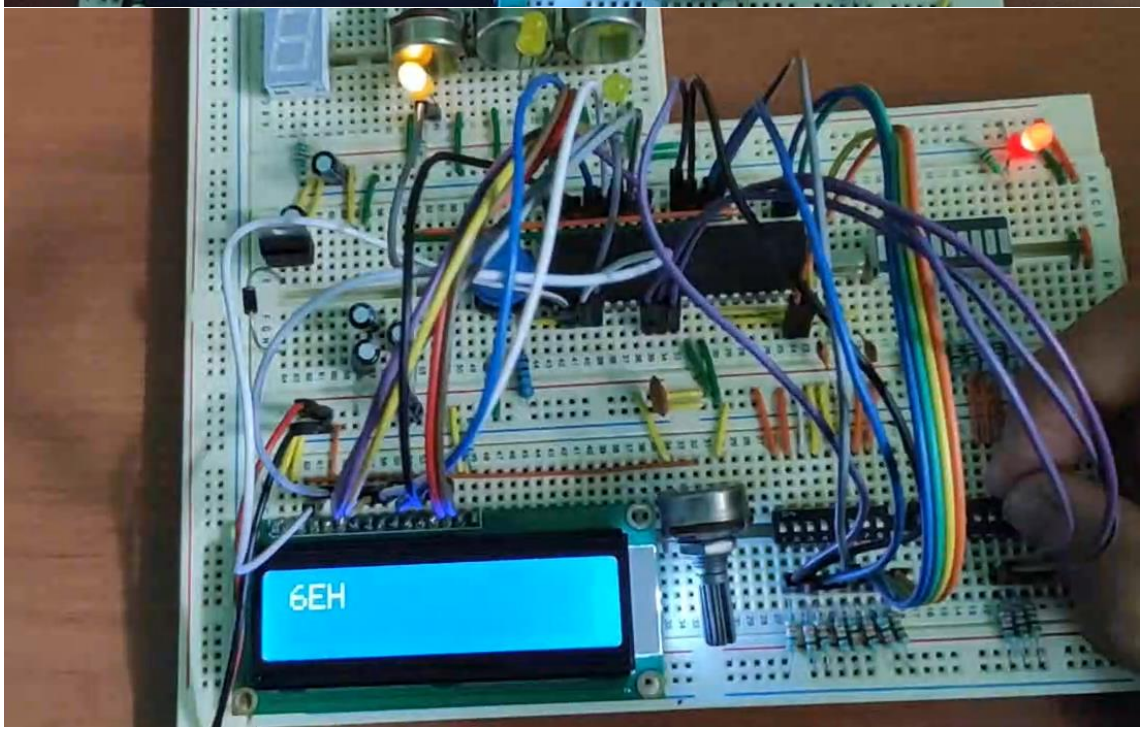
```
    movlw h'09'
    subwf regVol1,0
    btfss STATUS, Z; ¿regVol1 = 9?
    goto noRegVol19; regVol1 != 9
    clrf regVol1; regVol1 = 9

    movlw h'09'
    subwf regVol2,0
    btfss STATUS, Z; ¿regVol2 = 9?
    goto noRegVol29; regVol2 != 9
    clrf regVol2; regVol2 = 9
    goto finLoopVolt

noRegVol08:
    movlw h'02'
    addwf regVol0,1; regVol0 <- regVol0 + 2
    goto finLoopVolt

noRegVol19:
    incf regVol1,1; regVol1 <- regVol1 + 1
    goto finLoopVolt

noRegVol29:
    incf regVol2,1; regVol2 <- regVol2 + 1
    goto finLoopVolt


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
```

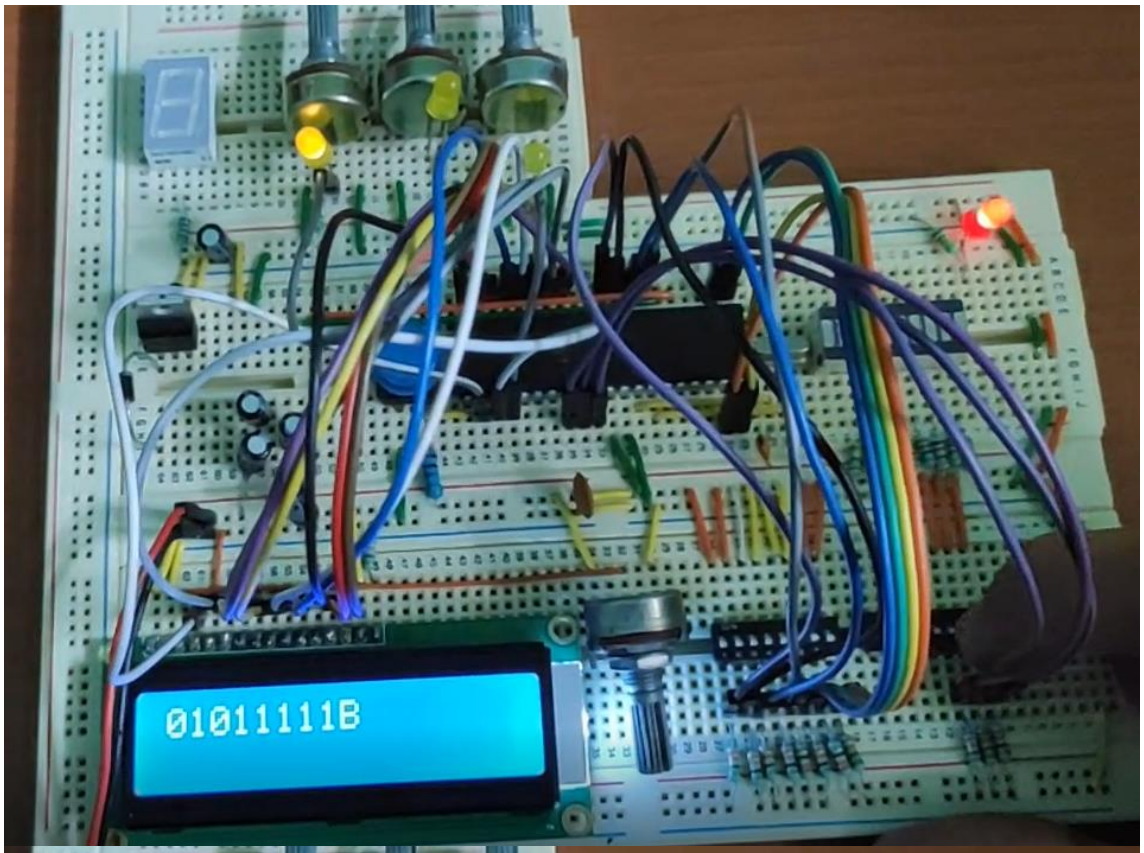
```
    call ret200
    bcf PORTA,0
    bsf PORTA,1
    call ret200
    bcf PORTA,1
    return

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
```

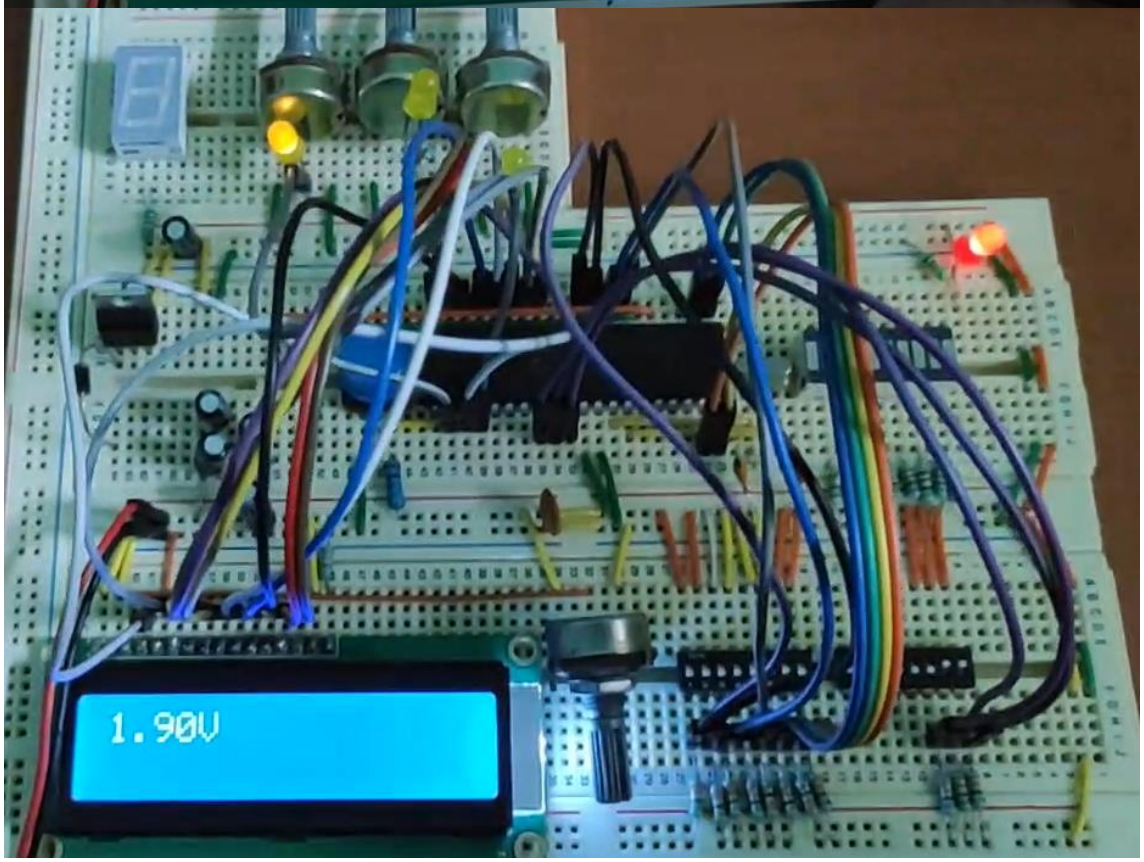
# Resultados obtenidos:

Se agregan imágenes de su funcionamiento



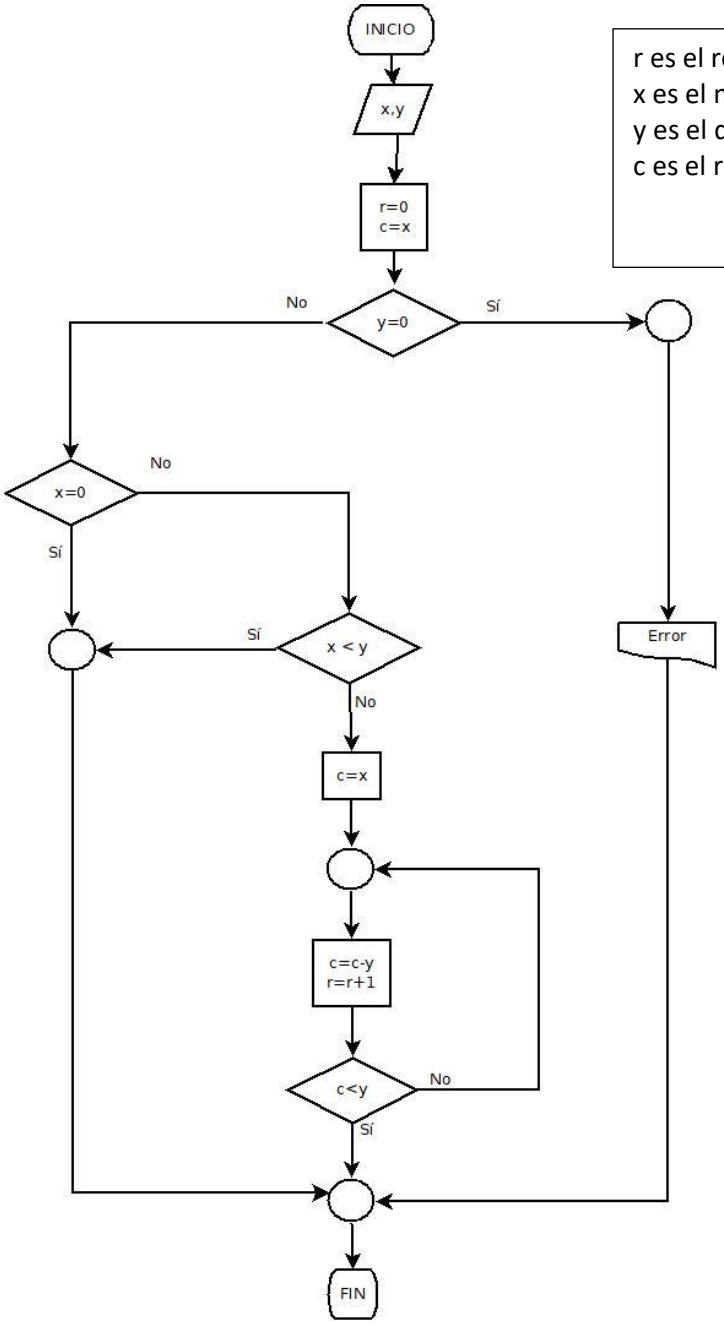




01011111B


1.90V

# 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