TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ENSENADA

Programa de Ingeniería en Sistemas Computacionales

Tarea

Calculadora 1ra Parte

Autor:
Jose Alberto Santiago Rios
No Control: 15760547

Profesor:
ME Oscar Martin Tirado Ochoa

Lenguajes de Interfaz (Enero-Junio 2020)

Ensenada B.C. México
14 Mayo de 2020

- **Actividad**: Desarrollar na sumadora que comprenda de 0 a 255. Ademas de algunas funciones como la resta.

1. Desarrollo

1.1. Arquitectura Proteus

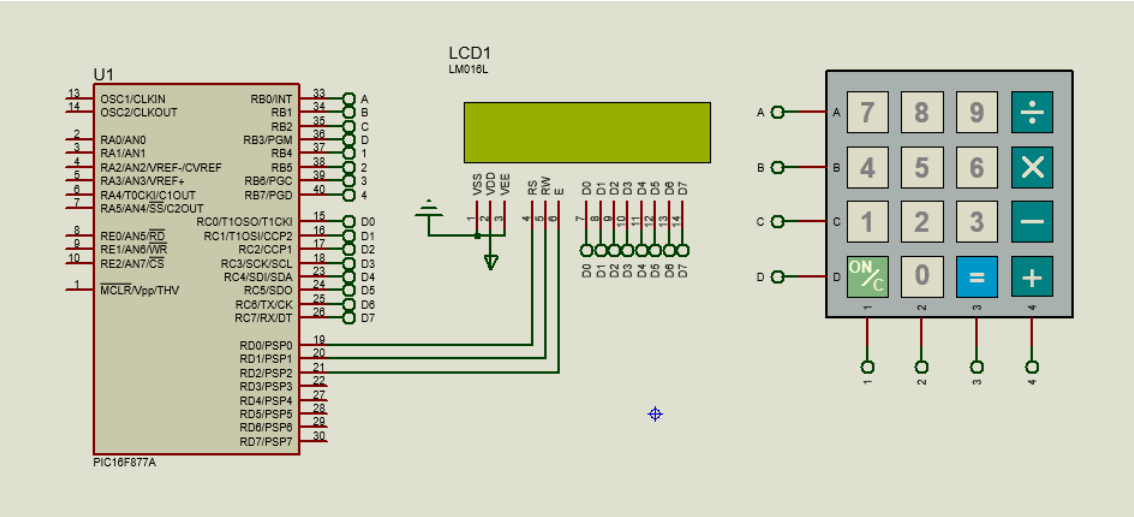

imagen 01 Arquitectura de Circuito

Para la arquitectura del circuito me base en un tutorial de internet debido al desconocimiento de poder comprender en su totalidad el armado con este tipo de componentes, mas que nada con el LCD.

1.2. Captura de numeros a operar

En esta primera capsula obtenemos ademas de ofrecerle a lña memoria temporal **W** el numero correspondiente, mediante la funcion **valora_unidades** obtendremos el valor real del numero digitado.Posteriormente se dibujara la tecla en el lcd en formato Ascci

```
0 references
UNO:
    movlw 0X01                  ; VALOR DEL NUMERO
    call valora_unidades        ; El numero acumulado
    movwf segundo_operando      ;  w -> segundo operando = 23
    movlw 0x31                  ; 1 en ascci
    call dibuja_display             ; dibujas display
0 references
return

0 references
DOS:
    movlw 0X02
    call valora_unidades
    movwf segundo_operando
    movlw 0x32
    call dibuja_display
0 references
return
```

imagen 02 Codigo de por presión sobre tecla

La funcion **valora_unidades** verifica mediante la bandera *unidades* cuantos digitos se han presionado antes, esto lo hace antes de que cualquier operador resete esta bandera. Cada función trabaja mediante **GOTO** para que cuando encuentre el return se regrese con el valor real del digito (Al mencionar valor real del digito nos referimos a un numero completo Ej 210)

Nota: La calculadora esta diseñada solo para trabajar con 3 digitos.... por el moemento

```
187  ;---------------FUNCIONES DE DECENAS--[ INICIO]-------------------
     0 references
188  valora_unidades:
189
190      btfss unidades,0           ; 001 -> skip
191      goto funcion_unidades
192      btfss unidades,1           ; 001 -> skip
193      goto funcion_decenas
194      btfss unidades,2           ; 111 -> skip
195      goto funcion_centenas
196
```

imagen 03 Valora Unidades

funcion_unidades manda la señal a la bandera en el bit 0 y guarda en su espacio de memoria correspondiente. **funcion_decenas** manda la señal a la bandera en el bit 1 y hace una copia de lo que contiene **w**, esto debido a que w siempre contendra unidad, pero necesitamos ese espacio para trasladar lo que hay en unidad al espacio de memoria **decena**. Para obtener la cantidad real de decena lo que realizamos es un loop que suma 10 a **w** por cada ciclo, el cual estara manipulado por lo que contiene decena, ese resultado sera sumado a lo que tiene unidad y de esta manera tenemos el numero real de dos digitos.

```
198  funcion_unidades: ; 000
199      bsf unidades,0 ; 001
200      movwf unidad
     0 references
201  return
     0 references
202  funcion_decenas: ;23
203      bsf unidades,1      ; unidades 011 w=9
204      movwf aux_unidad    ; aux_unidad = unidad 3
205
206      movf  unidad,0      ; w=2
207      movwf decena        ; decena = 2
208      movwf contador      ; contador
209      movf  aux_unidad,0  ; w = unidad : 2
210      movwf unidad
211      ; decena 2 unidad 3
212      movlw 0X0A          ; w=10
213      movwf aux_decena    ; contador = 10
214      MOVLW 0X00          ; w=0
215
         0 references
216      LOOPD:      ; 2 +2= 4
217          ADDWF aux_decena,W  ; w = w + 10
218          DECF contador,1 ;   n veces que corresponde a las decenas
219          BTFSS STATUS,Z  ; pasa 0
220          GOTO LOOPD
221          ; W = 20
222      ADDWF unidad,W ;w = 23
223      clrf aux_decena
```

imagen 04 Codigo de por presión sobre tecla

La **función_centenas** tiene exactamente la misma lógica, adapatada para una variable mas, la centena.

1.3. Operaciones

Tanto como para la Suma como para la Resta, tiendo al alcance los numeros reales de nuestra operación estas se genera con un **ADDWF** si es suma o un **SUBWF** para la resta . En el caso de la Multiplicación se crea un ciclo que sumara a **W** el **primer_operando** las veces que diga **segundo_operando** con un decremento, de esta forma cuando STATUS,Z se encuentre en 1, es decir que segundo_operando sea 0 se saldra.

```
430  COND00: ; CONDICION 00 =SUMA=
431      MOVF  primer_operando,W
432      ADDWF segundo_operando,W ; 58 ; 10
433      call separa
434
     0 references
435  RETURN
436
437          ;RESTA
     0 references
438  COND01: ;CONDICION 01
439
440      MOVF segundo_operando,W
441      SUBWF primer_operando,W
442      call separa
     0 references
443  RETURN
444
445  ;Multiplicacioooon
     0 references
446  COND10: ;CONDICION 10
447      MOVLW 0X00
         0 references
448      LOOP2:
449          ADDWF primer_operando,W
450          DECF segundo_operando,F
451          BTFSS STATUS,Z
452          GOTO LOOP2
453      call separa
     0 references
454  RETURN
```

imagen 05 Suma, Resta y Multiplicación

En el caso de la división me encontre con muchos problemas y la unica forma de poder resolverlo fue apyarme en 2 auxiliares, los contadores, y sumar el segundo operador hasta que al restarlo por el primero se desbordara o diera 0, y el primer contador incrementaba. Segi la lógica "cuantas veces le cabe un numero al otro"

```
456  ;Division
     0 references
457  COND11:;CONDICION 11
458
459
460      MOVF segundo_operando,W ;w = 4
461      movwf contador2
462      clrf contador ; cont = 0
463
         0 references
464      DIVI:
465          movf contador2,w
466          subwf primer_operando,W ;
467          btfss STATUS, C ; 10-7
468          goto SALIRDIV ; si
469          incf contador
470          MOVF segundo_operando,W ;w = 4
471          addwf contador2,w
472          movwf contador2
473          GOTO DIVI
474
     0 references
475  SALIRDIV:
476      movf contador,W
477      call separa
478      clrf contador
     0 references
479  RETURN
480
```

imagen 05 Suma, Resta y Multiplicación

2. Mostar en display

Para separar la cantidad del resultado, verifico si esta es menor a 100 y menor a 10, de esta manera se que si es menor a 10 solo debo mostrar en el display un numero, con el **SEPARAUNIDAD**

```
0 references
separa:
    movwf resultado ; 33
    movlw 0X0A  ; 10 Hexa  w =10
    subwf resultado,W ; 10
    btfss STATUS, C ; 23
    goto SEPARAUNIDAD
    movlw 0X64  ; 100 Hexa  w =100
    subwf resultado,W ; 10
    btfss STATUS, C ; 23
    goto SEPARADECENA
    goto SEPARACENTENA

0 references
SEPARAUNIDAD:
    movf resultado,W
    addlw 0x30
    call dibuja_display
0 references
RETURN
```

imagen 06 Separa

Para separar el caso en mas de 10 se va a ir a **SEPARADECENA** y del modo mas abstracto y rudo, hara un conteo y restara el resultado de uno por uno, por cada 10 numeros aumnetara uno en decena y y lo depositara a su variable, y el resto que quede lo depositara en unidad.

```
SEPARADECENA:
    ; resultado = resultado
    clrf aux_decena
    clrf aux_unidad     ; unid = 0
    clrf condicion
    0 references
    FOR_DECENA:
        movlw 0X0A          ; w = 10
        movwf contador      ; contador = 10

        0 references
        FOR_UNIDAD:
            incf aux_unidad,1  ; 5
            decfsz resultado,f
            BTFSC STATUS,Z
            GOTO SALIR
            decfsz contador,f
            goto FOR_UNIDAD

        clrf aux_unidad     ; unid = 0
        incf aux_decena,1
        goto FOR_DECENA

    0 references
    SALIR:
    movlw 0X0A
    subwf aux_unidad,W
    btfsc STATUS,Z
    call incrementa_ud
    movf aux_decena,W
    addlw 0x30
    call dibuja_display
```

imagen 07 Separa Decenas

La **SEPARACENTENA** usa la misma lógica.

2.1. PRUEBAS

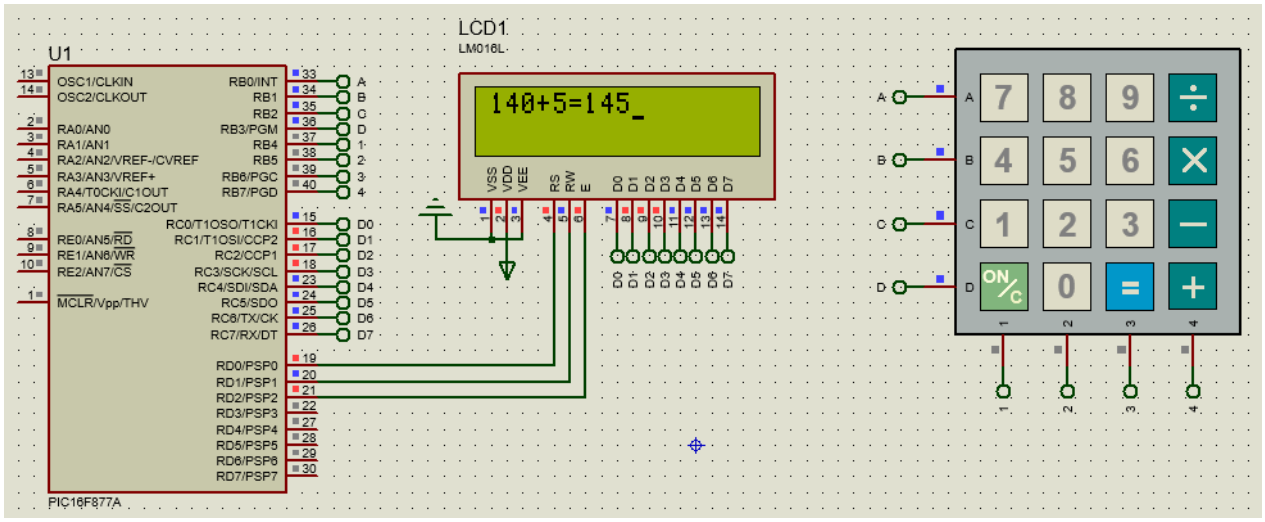

imagen Test 1

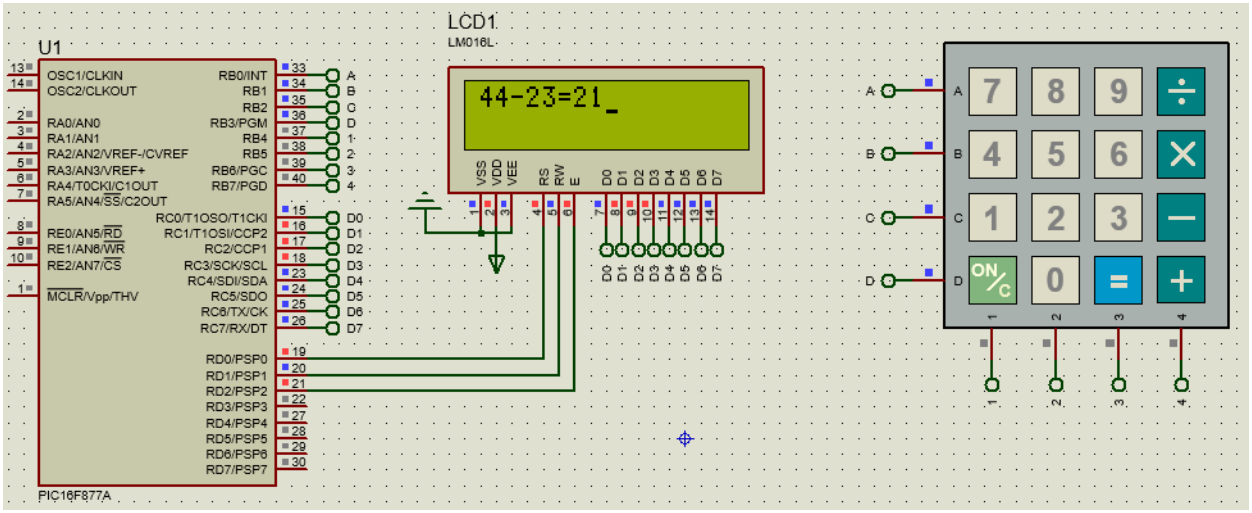

imagen Test 2

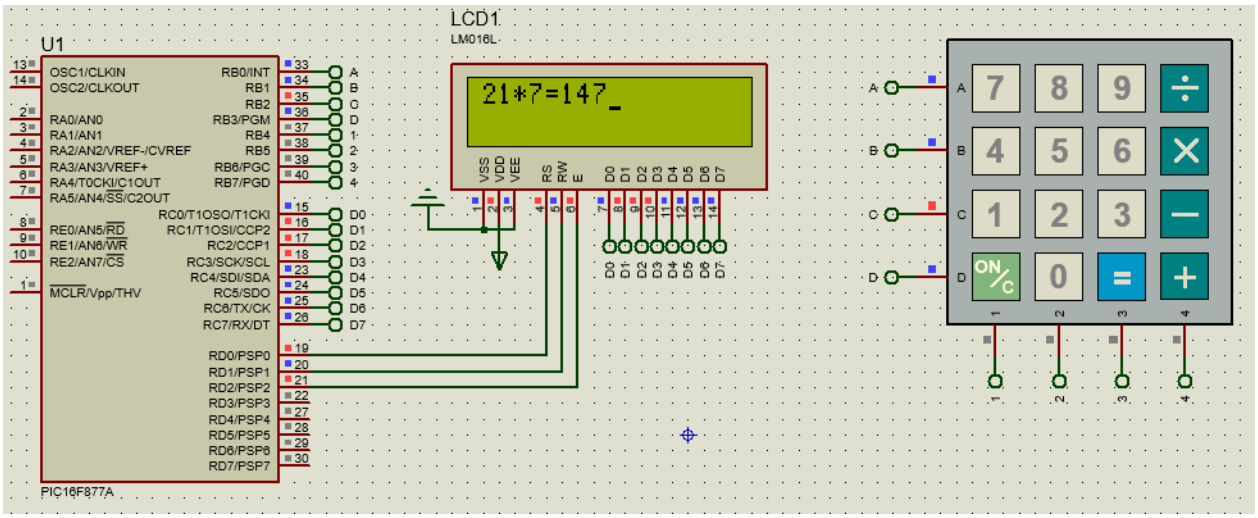

imagen Test 3

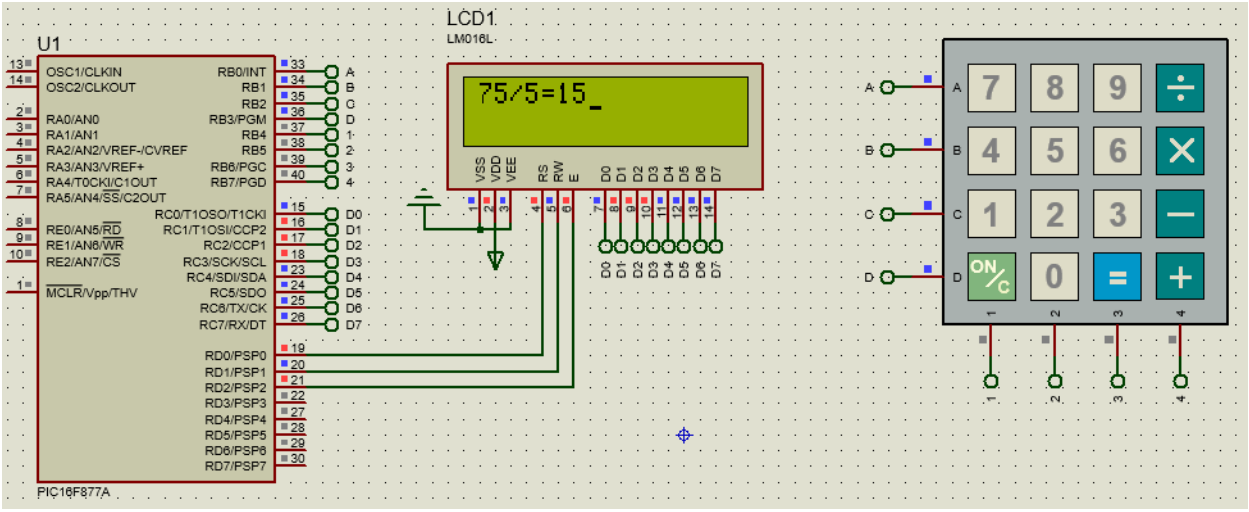

imagen Test 4

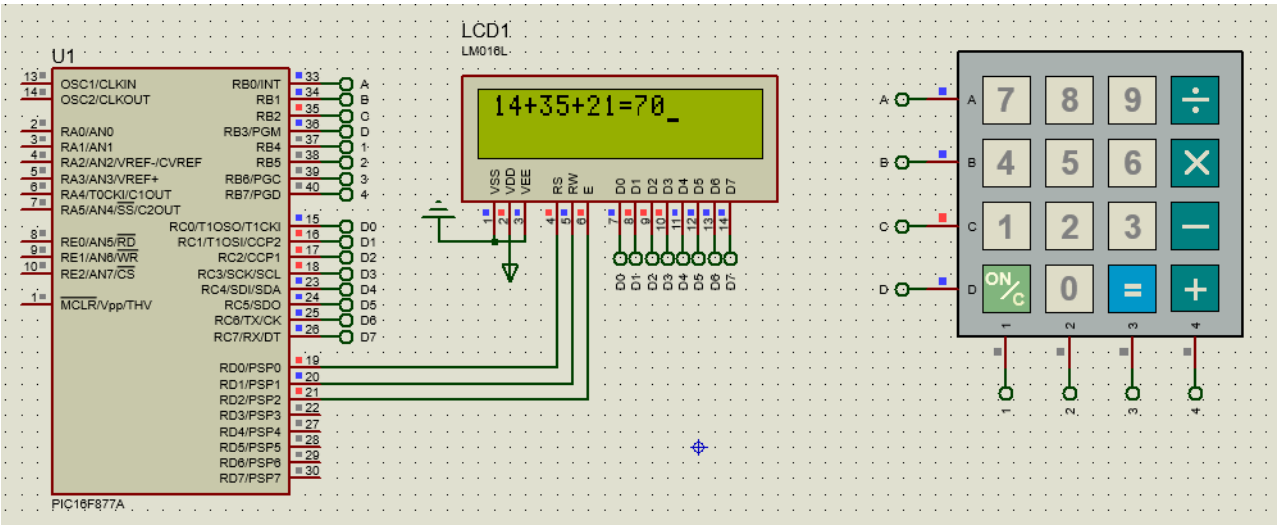

imagen Test 5

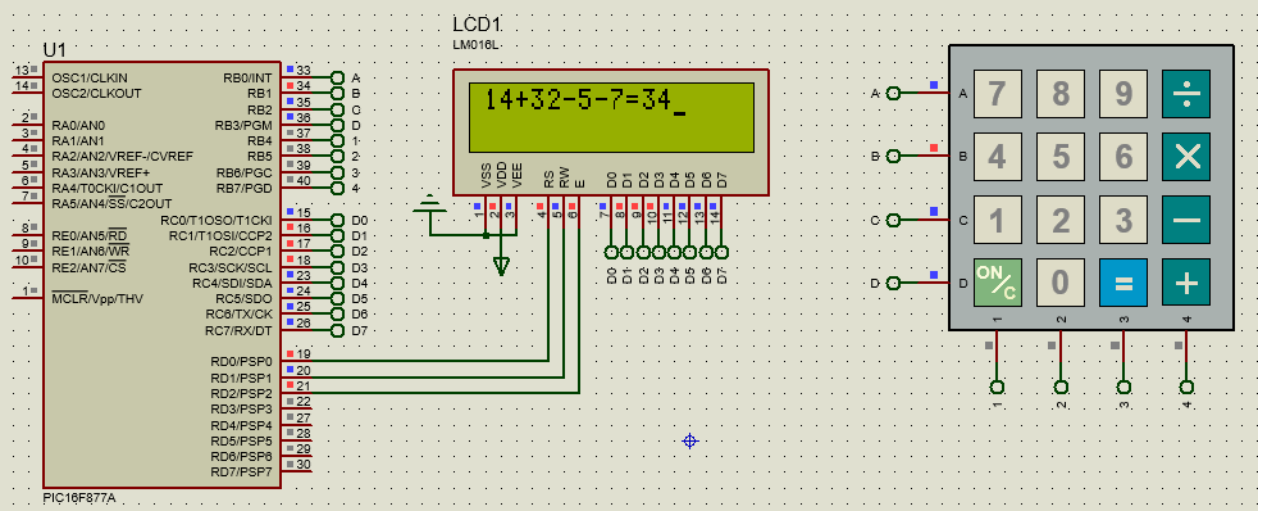

imagen Test 6

3. Conclusión

Las dos funciones principales para hacer esto fueron las llamadadas goto y call, que junto con el return me hicieron modular cada parte del codigo y cada funcion por separado para poder dividiir todos los posibles casos que puede realizar el usuario.

De esta manera hacen mas facil la escalabilidad del codigo, aun hay cosas que merecen mas eficiencia como las lecturas de los resultados pero pensando en que el usuario tiene mas tiempo de leer los resultados que de escribirlo, esto compenza un poco lo burdo del programa. Aun asi estoy pensando en mejorar esa parte.

La filosofia que busco logar con esta lógica es no realizar cosas raras y partir de las bases de las operaciones basicas que realizamos dia a dia, el proximo reto sera consultar lecturas asiaticas para ver si puedo implementar algo con su estilo, claramente si esto resulta tener mas eficiencia.

El código ejecuta:

- Suma
- Resta
- Multiplicación
- División
- Suma seriada
- Resta seriada
- suma y Resta Seriada

El código no ejecuta:

- Desborde de 8bit
- Multiplicación Seriada
- Division Seriada
- Manejar Numeros Negativos

Nota: Esta vez solicite asitencia de mi compañera Rebecca Soriano para la comprension del display. Ademas me ayudo a comprender el funcionamiento del **goto** y el **call** con el que base la mayor parte de mi codigo.