Universidad Autónoma de San Luis Potosí
Facultad de Ciencias
Ingeniería Física

Alarma de seguridad con tablero electrónico y comunicación con PC.

Lenguaje Ensamblador: Proyecto final

Rebeca Alvarado Contreras
18 de enero del 2021

I. Introducción

A continuación se dará una breve explicación de algunos temas fundamentales para la implementación de la alarma con tablero electrónico.

I-A. Direccionamiento indirecto

El registro 00 (`INDF`) es usado para direccionamiento indirecto. La dirección del registro deseado se escribe en el registro `FSR`(file select register. Cuando los datos se escriben o se leen de `INDF`, en realidad se escriben o leen del registro al que está apuntando `FSR`. Esto es muy útil para leer o escribir en un bloque continuo de registro de propósito general `GPR`, por ejemplo, al guardar datos que se leen continuamente de un puerto. En la Figura 2 se muestra la comparación entre direccionamiento indirecto o directo. [1]

Sin embargo, puesto que se necesitan 9 bits para direccionar a todos los registros (000 - 1FF), se usa el bit `IRP` del registro de `STATUS` como bit extra. [1]

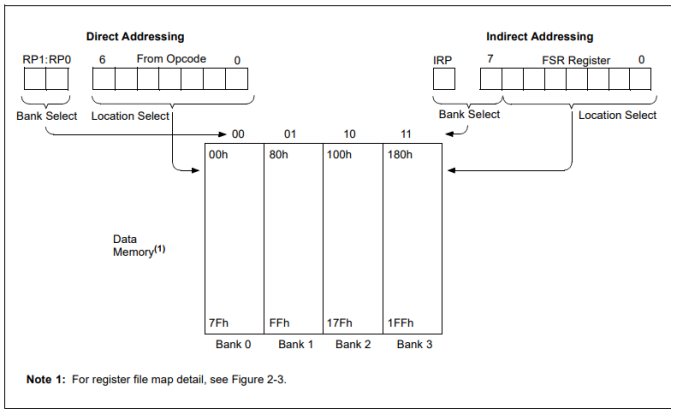

Figura 1: Comparación entre direccionamiento directo e indirecto.

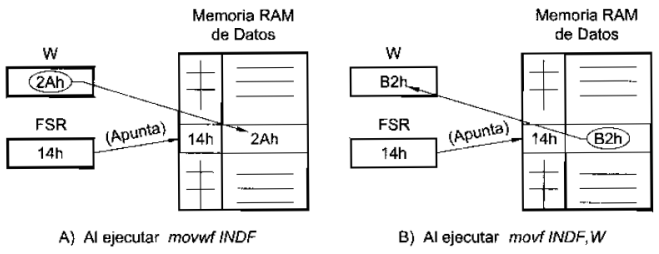

Figura 2: Funcionamiento de las instrucciones de direccionamiento indirecto. [2]

I-B. Macros

Una macro es un bloque de código que es insertado en el programa al usar su etiqueta/nombre como instrucción. Usar una macro es equivalente a crear una nueva instrucción desde instrucciones estánda, haciendo,[3] haciendo un bloque de código predefinido e insertado como un código fuente cuando sea requerido.[1]

La directiva `MACRO` define el inicio de bloque con una etiqueta, y la directiva `ENDM` lo termina. La ventaja de usar una macro sobre una subrutina, aunque incrementa la longitud del código, hace la misma función pero reduciendo el tiempo de ejecución al eliminar las instrucciones `return` y `call`. Las subrutinas usarán menor memoria, ya que son ensambladas una sola vez.[3]

Pero la ventaja más destacable es que se pueden definir argumentos en la macro (como en una función de mayor nivel), y al usar la macro definida, estos argumento se sustituyen con los valores requeridos.

I-C. UART

La forma más común de de comunicar cualquier dispositivo con un ordenador es a través del puerto serie. En la comunicación UART (Universal Asynchronous Receiver/Transmitter) , solo se requierene dos cables para transmitir dats entre dos UARTs; donde los datos fluyen del pin Tx (transmisión) al pin Rx (recepción) del UART receptor.

En comunicación UART se transmiten los datos asíncronamente, lo cual significa que no hay una señal de reloj para sincronizar la salida de los bits del transmisor al muestreo de los bits por el receptor. En vez de una reloj de señal, se agrega un bit de inicio y parada al paquete de datos transferido.[4]

El protocolo establecido por la norma RS232 envía la información estructurada en 4 partes (véase Figura 3):

- Bit de inicio (start): Es un paso de 1 a 0 lógico, indicándole al receptor que la transmisión ha comenzado, y a partir de entonces, debe de leer las señales de la línea a distancias concretas de tiempo en función de la velocidad fijada por emisor y receptor.

- Bits de datos: Los bits de datos son enviados al receptor después del bit de inicio. Se transmite primero el bit menos significativo.

- Bit de paridad (parity): Según la configuración de la transmisión, un noveno bit puede ser enviado para descubrir errores en la transmisión; aunque en aplicaciones simples no suele ser empleado.
- Bit de parada (stop): Después del último bit enviado, se queda en en 1 lógico, indicando la finalización de la transmisión de la palabra de datos.

Un aspecto importante a tomar en cuenta en cualquier comunicación, sobre todo en UART dado que no hay un reloj, es la velocidad de transmisión (cantidad de información enviada por unidad de tiempo); donde la más empleada es el **baudios** (bits por segundo), específicamente, 9600 baudios.

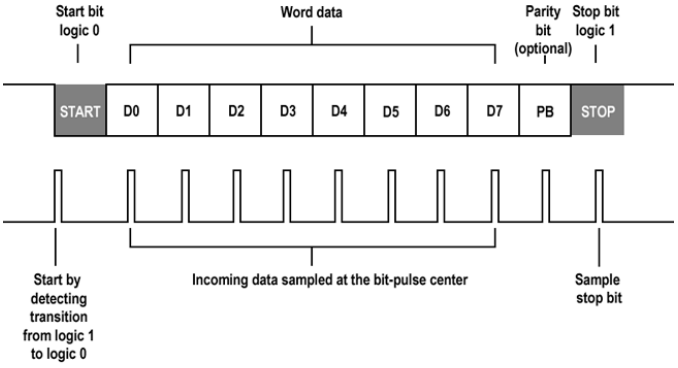

Figura 3: Estructura de bits transmitidos/recibidos para comunicación UART.

Sin embargo, para los estándares actuales es lento. [2]

El receptor y transmisor deben ser inicializados a la misma velocidad de transmisión, número de bits de datos, bits de parada y de paridad.[1]

Aunque los conectores vienen en formatos de 25 y 9 líneas, para comunicarse con un microcontroladores suficiente con tres líneas (véase Figura 4):

- Línea de transmisión (TxD)
- Línea de recepción (RxD)
- Pin de tierra

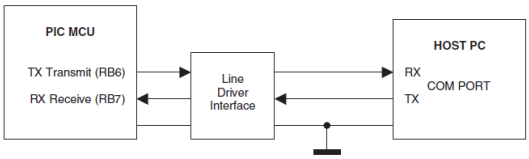

Figura 4: Interfaz entre microcontroladory computadora.

El *PIC16F877A*tiene un módulo integrado de USART, se tienen dos modos de operación: asíncrona y síncrona, accesado por los pines RC6 y RC7, para UART en específico, sirven para transmitir y recibir respectivamente. En la hoja de datos del *PIC16F877A*se tiene una descripción sobre el uso del UART, los registros involucrados, etc.

II. Objetivos

Implementar un tablero electrónico de una alarma de seguridad, en donde, con el botón S2 se programa la alarma para ser activada (aceptar interrupciones), y solo puede ser desactivada poniendo un pin de seguridad de 4 caracteres (dado), a través del teclado; pero, solo se tienen 3 oportunidades para ingresar el pin correcto; de no ser el caso, se esperará una señal de la computadora para regresar a su estado inicial. Si se dispara la alarma, es decir, se detecta una interrupción, se mandará una señal al ordenador de alerta y el buzzer será activado. En cada caso, se mostrarán los mensajes correspondientes.

El pin que dará acceso será definido por el usuario al inicio, mediante la terminal de la computadora.

Se establece comunicación con la computadora y microcontrolador mediante comunicación serial (UART).

III. Material y equipo utilizado

- El MCU *PIC16F877A*.
- *PICkit 3* para programar el microcontrolador.
- Tarjeta de desarrollo para crear la interfaz entre el MCU y los demás componentes, tales como: fuente de poder, LED's, el ya mencionado programador, entre otros.
- *MPLAB* para hacer el ensamblaje del programa, y su respectiva depuración.
- Adaptador USB - serial (DB9).

IV. Diagrama esquemático del circuito

En trabajos anteriores, ya se han presentado los esquemas de los circuitos empleados (LEDs, bocina, pantalla LCD, teclado, etc.) Esta vez solo difiere el uso del módulo USART, para esto, se requiere el circuito integrado MAX232 (véase Figura 5), el cual convierte entre niveles TTL y RS232.

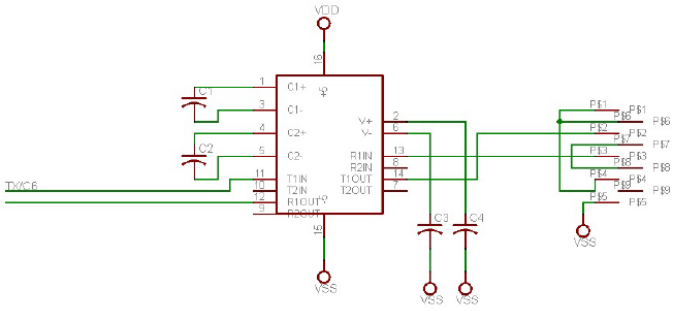

Figura 5: Esquema del circuito para el IC MAX232.

V. Código documentado

En trabajos pasados ya se ha visto el código para funcionamiento del LCD y teclado, por lo que solo se presentarán los que conciernen a la alarma en específico.

V-A. *Macros definidas*

```
movlf       MACRO   const,varReg
;mover constante a registro
            movlw   const
            movwf   varReg

            ENDM
```

Figura 6: Macro `movlf`

V-B. *Librería* `UART_LIB`

V-C. *Programa principal*

```
movff       MACRO   original,destino
;mover constante a registro
            movfw   original
            movwf   destino

            ENDM
```

Figura 7: Macro `movff`

```
mdelay      MACRO       veces_delay

            local       delayaso

            movlw       veces_delay
            movwf       CONTDEL

delayaso    call        DELAY
            decfsz      CONTDEL,f
            goto        delayaso

            ENDM
```

Figura 8: Macro `mdelay`

```
comparar    MACRO   num1,num2, Ltrue, Lfalse
;saber si son iguales
            movfw   num1
            subwf   num2,W

            btfss           STATUS, Z
            goto            Lfalse

            goto            Ltrue

            ENDM
```

Figura 9: Macro `comparar`

```
msg_cte     MACRO   posicion,TABLA_MSG

;0x80: primera fila
;0xC0:segunda fila

;va a desplegar un mensaje constante, dada la tabla que contiene los caracteres
;después de despleagrlo, se va a la primera posicion de segunda fila

;DATO es una variable definida en el main
            local   LAZO
            local   FINMEN

            ;call       BORRA_Y_HOME
            movlw       posicion
            call        LCD_REG

            movlw       0x00            ; Despliega mensaje
            movwf       DATO
LAZO        call        TABLA_MSG
            iorlw       0x00
            btfsc       STATUS,Z
            goto        FINMEN
            call        LCD_DATO
            incf        DATO,F
            goto        LAZO

FINMEN      movlw       0xC0            ; Primera posición de segunda fila
            call        LCD_REG

            ENDM
```

Figura 10: Macro `msg_cte`

```
read_arr    MACRO   arreglo, elemento,numx
;solo es para leer el elemento I del arreglo
            movlw   arreglo     ;dirección del elemento 1 del arreglo
            addwf   elemento,W  ;dirección del elemento i-ésimo gaurdada en W
            movwf   FSR         ;ahora ya estoy apuntando a ese elemento

            movfw   INDF
            movwf   numx

            ENDM
```

Figura 11: Macro `read_arr`

```
write_arr   MACRO   arreglo, elementoCont,var
            ;Arreglo es el apuntador/dirección del arreglo
            ;elemento es el contador de cuátos escribió
            ;var es la variable desde la cual va a leer y poner en INDF

            movlw   arreglo         ;dirección del elemento 1 del arreglo
            addwf   elementoCont,W  ;dirección del elemento i guardada en W
            movwf   FSR             ;ahora ya estoy apuntando a ese elemento

            ;aqui quiero escribir en indf el código ascii de la letra
            movfw   var
            movwf   INDF            ;estoy escribiendo en INDF

            incf    elementoCont,f  ;está listo para escribir en el sig elemento

            ENDM
```

Figura 12: Macro `write_arr`

```
#DEFINE  TX_EN  bsf     TXSTA,TXEN      ;habilitar transmisión (banco 1)
#DEFINE  TX_DIS bcf     TXSTA,TXEN      ;deshabilitar transmisión (banco 1)

#DEFINE  RX_EN  bsf     RCSTA,CREN      ;habilitar transmisión (banco 0)
#DEFINE  RX_DIS bcf     RCSTA,CREN      ;deshabilitar transmisión (banco 0)

#DEFINE  SC_EN  bsf     RCSTA,SPEN      ;habilitar comunicación serial (banco 0)
```

Figura 13: Macros para UART

```
INIT_UART
        bcf     STATUS,RP1
        bsf     STATUS,RP0      ;selecciona banco 1

        bsf     TRISC,6
        bsf     TRISC,7

        ;configurar baud rate
        movlw   .129            ;129 a BRG (baud rate 9600)
        movwf   SPBRG

        ;modo 8 bits, asincrono, high speed, habilitar transmisión
        movlw   b'10100100'
        movwf   TXSTA

        bcf     STATUS,RP0      ;banco 0
        ;habilitar comunicación serial
        RX_EN                   ;habilitar recepción
        movlw   b'10010000'
        movwf   RCSTA

        bcf     PIR1,RCIF; LIMPIAMOS FLAG RX

        return
```

Figura 14: Subrutina `INIT_UART`

```
UART_ENVIAR
        ; el contenido a enviar debe de estar en W
        bcf     STATUS,RP0

wtT     btfss   PIR1,TXIF       ;TXREG vacio?
        goto    wtT             ;no

        movwf   TXREG           ;mover a txreg

        return
```

Figura 15: Subrutina `UART_ENVIAR`

```
UART_RECIBIR
;se asume que externamente se checa la bandera RCIF
;los datos recibidos se ponen en la variable varRx
        bcf     STATUS,RP0                      ;banco 0

        ;primero veo si hay overrun error
        btfsc   RCSTA,OERR      ;hay overrun error?
        call    RX_OERR         ;si-> desactivar error

        movfw   RCREG           ;no->lo que sea que recibi lo pongo en W
        movwf   varRx

        return
```

Figura 16: Subrutina `UART_RECIBIR`

```
RX_OERR
        ;si hay overrun error, hay que resetear CREN

        RX_DIS
        RX_EN
        return
```

Figura 17: Subrutina `RX_OERR`

```
main        call        INICIALIZA              ;Rutina de inicialización
            call        INIT_UART               ;Inicializa uart

;Definir contraseña
            call        NUEVO_PIN
             mdelay     0xAF
;Esperar a que se active la alarma

            ;reiniciar:vidas, leds, flags, ptr, deplegar msg, desactivar ints
WT_0        msg_cte     0x00,WAIT_ARMAR     ;mensaje de espera
            call        INIT_RES            ;
            movlf       0x00, INTCON        ;desactivar interrupciones

WT_ARMAR
            btfsc       PORTA,5
            goto        WT_ARMAR        ;esperar a que se arme

    ;ya fue activada la alarma
            call        ACTIVADA
```

Figura 18: Programa principal: `main` (parte 1)

```
LOOP1       call        KBD_INI         ; Inicializa puerto para leer teclado
            call        KBD_BARRE       ; Efectua barrido de teclado
            btfss       BARRIDO,4       ; Verifica si se presionó tecla
            goto        CHECAR          ; Checa si se escribieron 4 carcteres

            call        KBD_LCD         ; Reconoce tecla, y exhibe


            ;checar si ya se presionaron 4 teclas
            ;EI_ARR debería de estar en 4

CHECAR      movfw       EI_ARR
            sublw       0x04
            btfss       STATUS,Z                ;si es 0->comparar arrays
            goto        LOOP2                   ;no->continua
```

Figura 19: Programa principal: `main` (parte 2)

```
            ;aquí es donde comparo ARR_CO y ARR_U
            call        COMPARAR_ARRAY
            btfss       FLAGS,0
            goto        SHOW_ERROR

        ;contraseña correcta
            call        PIN_OK
            goto        WT_0
```

Figura 20: Programa principal: `main` (parte 3)

```
        ;contraseña incorrecta
SHOW_ERROR
            call        PIN_MAL
            decfsz      VIDAS,f
            goto        PASS            ;aun tiene vidas

        ;contraseña incorrecta 3 veces
            call        INCORRECTO_3

        ;esperar a recibir comando 'H' para reiniciar
            call        DESBLOQUEAR
            goto        WT_0

        ;reiniciar para seguir revisando teclado

PASS        clrf        EI_ARR
            msg_cte     0xC0,NADA

LOOP2       movlw       0x7D            ; Retardo 20 ms.
            movwf       T_DELAY
            call        DELAY
            goto        LOOP1

            END                                 ; directiva 'fin del programa'
```

Figura 21: Programa principal: `main` (parte 4)

```
INIT_RES
                clrf        PORTB
                clrf        FLAGS
                clrf        EI_ARR                  ;limpiar ei_arr
                movlf       0x03, VIDAS
                msg_cte     0xC0,NADA
                movlf       0x00, INTCON            ;DESACTIVAR INTS

    return
```

Figura 22: Subrutina `INIT_RES`

```
NUEVO_PIN
;solo regreso al presionar S2

nueva_otra  clrf        EI_ARR
            msg_cte     0xC0, NADA

wtR_pin     btfss       PIR1,RCIF       ;algo se recibió?
            goto        wtR_pin         ;no->seguir esperando

            call        UART_RECIBIR    ;el caracter recibido está en Rx y W
            call        LCD_DATO        ;ver carcater recibido

            ;ahora escribir dato en ARR_CO
            write_arr   ARR_CO, EI_ARR, varRx

            ;ahora hay que preguntar si ya fueron los 4 dígitos
            movfw       EI_ARR
            sublw       0x04
            btfss       STATUS,Z        ;ya puso 4 digitos
            goto        wtR_pin         ;no->espera el otro caracter


            ;si presiono S3 significa que la contraseña se define y continuo
WT_S3       btfsc       PORTA,4
            goto        WT_S2           ;no se presionó, checo S2
            goto        FIN_PIN         ;continuar

            ;si presiono S2 significa, defino otra contraseña
WT_S2       btfsc       PORTA,5
            goto        WT_S3
            goto        nueva_otra

FIN_PIN     return
```

Figura 23: Subrutina `NUEVO_PIN`

```
COMPARAR_ARRAY
            clrf        MATCH           ;limpio match
            clrf        elementoI       ;limpio comtador de elementos

WLCOMPARAR

            read_arr    ARR_CO,elementoI, aNUM_1        ;pone el elemento I en NUM1

            read_arr    ARR_U,elementoI, aNUM_2         ;pone el elemento I en NUM2

            ;comparo NUM1 y NUM2->si son iguales
            comparar    aNUM_1, aNUM_2, MatchMM, SIGUEMM

MatchMM ;incrementar Match si son iguales
            incf        MATCH, f

SIGUEMM incf        elementoI,f

            ;veo si elementoI llegó al límite (4 elementos)
            comparar    elementoI, LIMITE, FINC, WLCOMPARAR
                                                    ;continua otra iteración
            ;sí , ya acabé el array
            ;ahora, solo veo si los arrays son iguales

FINC    comparar    MATCH, LIMITE, IGUAL, DIFF

IGUAL   bsf         FLAGS,0
        return

DIFF    bcf         FLAGS,0
        return
```

Figura 24: Subrutina `COMPARAR_ARRAY`

```
PIN_OK      msg_cte     0x80,ACEPTADO
            mdelay      0xFF

            return
```

Figura 25: Subrutina `Activada`

```
PIN_MAL
            msg_cte     0x80,MSG_ERROR
            mdelay      0xFF
            bsf         PORTB, L_ERROR  ;prender led 6,indicando error
            return
```

Figura 26: Subrutina `PIN_MAL`

```
DESBLOQUEAR

wtRF        btfss       PIR1,RCIF       ;algo se recibió?
            goto        wtRF            ;no->seguir esperando

            call        UART_RECIBIR    ;si->ver qué se recibió

            xorlw       'H'             ;el carcter recibido está en W
            btfss       STATUS,Z
            goto        wtRF

            return
```

Figura 27: Subrutina `DESBLOQUEAR`

```
INTER
        movwf   VAR_AUX         ;guardar lo que tenia en W

        btfss   INTCON,INTF
        goto    INT_TMRO
        call    TRIGGER
        bcf     INTCON,INTF
        goto    FIN_INT

INT_TMRO btfss  INTCON,TOIF     ;TMRO se encargarpa de hacer las señales
        goto    INTER
        call    SERV_TMRO
        bcf     INTCON,TOIF

FIN_INT movfw   VAR_AUX

        retfie
```

Figura 28: Rutina de servicio

```
TRIGGER
    bsf PORTB,1 ;prender LED

    movlw   'E'
    call    UART_ENVIAR

    bsf FLAGS,F_INT
    movlf   0xA0, INTCON


    return
```

Figura 29: Subrutina `TRIGGER`

```
SERV_TMRO       btfsc   PORTC,0             ; Verifica estado de RA3
                goto    APAGA
                bsf     PORTC,0             ; RA3 = 1
                goto    CONTINUA
APAGA           bcf     PORTC,0             ; RA3 = 0
CONTINUA        movf    FREC,W
                movwf   TMRO                ; Carga cuenta en TMRO
                movlw   0xA0                ;activar interrupciones
                movwf   INTCON


    return
```

Figura 30: Subrutina `SERV_TMR0`

```
KBD_LCD
    call    KBD_VALID           ; Valida tecla presionada
    btfss   STATUS,Z            ; Verifica si la tecla fue válidada
    goto    KB_LC_FIN           ; Si tecla no válida va a LOOP2

    call    KBD_NUM             ; Convierte código de tecla a numérico
    call    KBD_LIB             ; Espera a que la tecla sea liberada
    call    HEX_ASCII           ; Convierte valor numérico a ASCII

    movwf   LETRA               ;aqui guardo el código ascii de la letra
    write_arr   ARR_U, EI_ARR, LETRA    ;escribo en el arreglo

    call    LCD_DATO            ; Despliega dato en LCD
KB_LC_FIN       return
```

Figura 31: Subrutina `KBD_LCD`

VI. Desarrollo

Para implementar la alarma se emplearon las siguientes librerías:

- *kbd_cxx.asm*- Librería para teclado.
- *lcd_cxx.asm*- Librería para pantalla LCD.
- Kbd_lcd.asm - Código principal (interfaz teclado-LCD).
- macros_alarma.inc
- mensajes_tablas.asm
- UART_LIB.asm

La razón de usar comunicación serial asíncrona entre PIC-PC, además de que es sencilla de comprender, es que se puede hacer interfaz con varios módulos (por ejemplo, módulos de Internet, Bluetooth, etc), por lo que la PC estaría funcionando como un suplente/simulador de tales módulos.

VI-A. Macros

Un elemento muy importante que sirvió para hacer el código más legible y versátil, fue el uso de macros; ya que gracias a la posibilidad de agregar argumentos; por lo que un pedazo de código puede ser adaptado a la situación.

Se presenta una breve descripción de las macros definidas.

VI-A1. Macro `movlf`: En la Figura 10 se muestra una simple macro para mover un valor literal a un registro, no solo el W.

VI-A2. Macro `movff`: En la Figura 7 se muestra una simple macro para mover un valor de un registro a otro registro.

VI-A3. Macro `mdelay`: En la Figura 7 se muestra una simple macro para llamar m veces la subrutina `DELAY`, haciendo un retardo variable.

VI-A4. Macro `comparar`: En la Figura 9 se muestra una macro que toma dos números, los resta, y si resultan ser iguales (el resultado de la resta es 0), salta a la dirección de la etiqueta `Ltrue`, de lo contrario salta a `Lfalse`.

VI-A5. Macro `msg_cte`: El código de la Figura 9 es una macro que despliega un mensaje constante dada la posición para la pantalla LCD (80h para primera línea y C0h para la segunda), y una tabla con los caracteres del mensaje. Tiene la misma estructura que el código revisado en la práctica pasada.

VI-A6. Macro `read_arr`: El código de la Figura 12 es una macro que dada la dirección de memoria del inicio de un arreglo (`arreglo`), a través de direccionamiento indirecto, se lee el i-ésimo elemento (`elementoCont`) el valor de `var`.

VI-A7. Macro `write_arr`: El código de la Figura 12 es una macro que dada la dirección de memoria del inicio de un arreglo (`arreglo`), a través de direccionamiento indirecto, se escribe en el i-ésimo elemento (`elementoCont`) el valor de `var`. Muy similar a la anterior, pero en la macro se prepara para comenzar escribir en el siguiente elemento.

VI-B. Librería `UART_LIB`

Para establecer la comunicació serial con el *PIC16F877A*solo se siguieron los pasos recomendados por su hoja de datos.

VI-B1. Subrutina `INIT_UART`: Como es indicado en la hoja de datos, los pones RC6 y RC7 deben de estar en 1 para establecerlos para UART. De igual manera, se indica que para establecer una velocidad de 9600 baudios, con un oscilador de 20MHz, el registro `SPBRG` debe de tener un valor de 129.

El registro `TXSTA` se configura en modo 8 bits, asíncrono, oscilador de alta velocidad y se habilita la transmisión. El registro `RCSTA` se configura para habilitar la comunicación

serial, habilitando la recepción.

VI-B2. Subrutina UART_ENVIAR: Para enviar un dato, este debe moverse al registro TXREG, pero es necesario checar que este registro esté vacío mediante la banderaTXIF del registro PIR1. Se asume que el dato a enviar se encuentra en el registro W.

VI-B3. Subrutina U ART_ENVIAR: Ahora, cuando un dato es recibido la bandera RCIF del registro PIR1 se pone a 1. Pero en esta subrutina, se asume que se checa externamente la bandera. Asumiendo que la bandera indica que se ha recibido algo; primero se hace el manejo de errores (overrun error), ya que si no se maneja, es posible de repente ya no se reciban datos. Una vez manejado el error, si lo hubo, los datos recibidos contenidos en le registro RCREG se mueven a la variable varRx, aunque también estarán disponibles en el regsitro W.

VI-B4. Subrutina RX_OERR: Esta subrutina es llamada si se detectó un error, en tal caso, para desactivarlo se debe de desactivar y volver a activar la recepción (CREN).

VI-C. Tablas de mensajes

Durante la ejecución del programa, se estarán desplegando mensajes en la pantalla LCD para indicar al usuario el estado. Mediante los métodos ya vistos de tablas de datos, se hizo una librería mensajes_tablas.asm con los siguientes mensajes:

- "Pin: "
- "OK "
- "N PIN"
- "ERROR"
- "BLOCK"
- "WT S2"
- "...."

Se establecieron pequeños mensajes de 5 caracteres debido a que se tuvo un problema, atribuido con el contador del programa causando que el programa dejara de funcionar. Además de que al ser todos de 5 caracteres, no se tendrá que borrar la LCD para que queden mensajes sobrepuestos.

VI-D. Código principal

En la Figura 32 se describe el esquema general del programa principal. Se empieza por definir la contraseña, subrutina NUEVO_PIN, a través de la terminal de la computadora, si se presiona S2 se escribe una nueva contraseña, S3 se continúa con el programa.

Se reinician los parámetros (subrutina INIT_RES): limpiar puerto B, oportunidades, apuntador de elementos, desplegar mensajes. Se espera a que presione S2, para activar la alarma, es decir, admitir interrupciones, además se prende el LED 7.

Entonces, se empieza a monitorear el teclado, en la rutina KBD_LCD después de la conversión de la tecla a su código ASCII, esta se guarda en el arreglo correspondiente ARR_U. Después de salir de la rutina, se checa si ya se ingresaron 4 teclas, determinado por la variable EI_ARR, la cual se incrementa cada vez que se escribe. Si no es así, se sigue escaneando a la tecla.

Si ya se ingresaron 4 teclas, se procede a comparar los arreglos con la subrutina COMPARAR_ARRAY, según el bit 0 de FLAG, se determina si la contraseña fue aceptada, en tal caso, se ejecuta PIN_OK e irá al estado de reinicio.

Por otra parte, si la contraseña es incorrecta se ejecuta PIN_MAL para desplegar error, y se decrementan las vidas; ahora se revisa si ya se agotaron las oportunidades, si no, se vuelve a esperar a que ingrese el pin. Sin embargo, si ya fueron agotadas se ejecuta INCORRECTO_3 en donde se envía una 'B' al ordenador como alerta, y espera a que se reciba una 'H' desde el puerto serial, solo entonces se desbloqueará y volverá al estado de reset.

En cuanto a las interrupciones, al detectarse una interrupción por RB0, se ejecuta TRIGGER para enviar una 'E' como alerta y activar la interrupción por Timer0. Nótese que las interrupciones no afectan al flujo del programa.

VI-D1. Comparación de arreglos: En la figura 24 se muestra el código responsable de hace la comparación de los arreglos ARR_C0 y ARR_U, la contraseña definida y el pin ingresado respectivamente. La comparación se lleva a cabo de la siguiente manera: con la macro read_arr se apunta al elemento i (variable EI_ARR) del primer arreglo y se guarda en NUM_1, los mismo para el otro arreglo, guardando el valor en NUM_2; después con la otra macro comparar se comparan ambos número guardados y si son iguales, se incrementa un contador MATCH, de cualquier manera, se incrementa EI_ARR para apuntar al siguiente elemento, y este también funciona como indicador de cuántos elementos se han revisado. Luego, este contador de elementos, se compara con el límite (solo se quieren leer 4 elementos), si aun no se han leído los 4, continua con la siguiente iteración, si sí, se compara el número de coincidencias con el número de elementos de los arreglos, si son iguales, significa que ambos arreglos contienen los mismo valores y se pone una bandera en FLAGS, 0.

VI-D2. Subrutina NUEVO_PIN: La subrutina de la Figura 31 se corre al inicio, se espera a recibir 4 caracteres a través del puerto serial, cada vez que se recibe uno se muestra en la LCD. Al terminar de enviar los 4 caracteres, se espera a que se presione ya sea la tecla S2 o S3, para escribir un nuevo pin, o para aceptar el pin escrito y coninuar.

VI-D3. Rutina de servicio INTER: En la rutina de la Figura 28 se muestr la rutina de servicio, en donde se halla la causa de interrupción, ya sea por RB0 o por el Timer0. En caso de ser por RB0, se llama a la rutina TRIGGER de

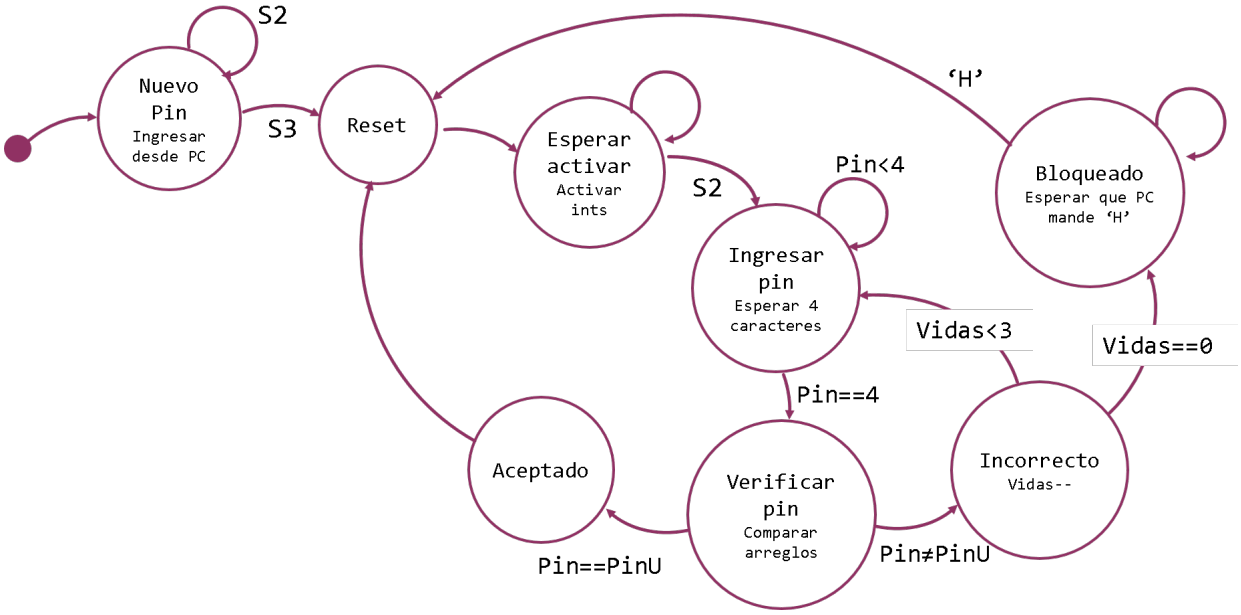

Figura 32: Máquina de estados.

la Figura 29, en la cual, se prende un LED indicando que la alarma fue disparada, y se activan las interrupciones por Timer0 para empezar a generar la señal para el buzzer (Figura 30.

VI-E. Comunicación PIC-PC

Además de emplear el conversor serial-USB, se usó la terminal mostrada en la Figura 33 para enviar y recibir los datos.

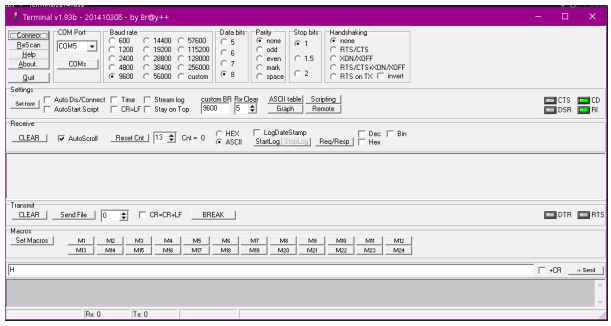

Figura 33: Terminal.

Finalmente, en cuanto al funcionamiento del programa, se ejecutó tal y cómo se esperaba siguiendo el flujo correspondiente.

VII. Conclusión

Se implementó exitosamente la alarma de seguridad con tablero electrónico, a través de una máquina de estado; además, estableciendo comunicación con la computadora para hacer un sistema comunicado, adecuado sobre todo para una alarma de seguridad con la cual se generarán alertas a otro dispositivo.

Debido a que se iban a ingresar/leer datos de manera continua y ordenada, un arreglo fue la opción más eficiente para almacenarlos, y para esto se requirió el uso de direccionamiento indirecto para apuntar y acceder a los diferentes elementos de los arreglos.

También, debido a que se estarían usando repetidamente ciertas secuencias de instrucciones, pero con parámetros variables (por ejemplo leer/escribir arreglos, mostrar mensajes, etc.), las macros fueron un elemento fundamental, debido a su flexibilidad y versatilidad, para hacer un código mejor estructurado y sencillo para el programador.

En lo que se refiere a la comunicación, realmente que el módulo UART esté integrado en el microcontrolador, hace un que el proceso sea muy sencillo; basta con inicializarlo de la forma indicada por la hoja de datos, y el proceso de recepción y transmisión se limitan a checar los registros y banderas correspondientes. Y a pesar de que UART ya no sea tan visto, hay módulo que emplean UART, tal es el caso del módulo Wifi ESP8266; en este caso, se usó la computadora a manera de simulación de estos módulos.

Aunque en general, la generación de interrupciones no causó una consideración en la lógica del programa; al emplear interrupciones hay que ser cuidadosos con el almacenamiento de información, ya que al generarse la interrupción la información

generada por el proceso interrumpido puede ser perdida, por lo que debe de ser almacenada al menos temporalmente para que pueda retomarse al reanudar el proceso interrumpido.

Referencias

[1] M. P. Bates, "9 serial communication," en *Interfacing PIC Microcontrollers*, Elsevier, 2014, ISBN: 978-0-08-099363-8. DOI: 10 . 1016 / C2012 - 0 - 02690 - 7. dirección: https : / / linkinghub . elsevier . com / retrieve / pii / C20120026907 (visitado 16-10-2020).

[2] Enrique Palacios, Fernando Remiro y Lucas López, *Microcontrolador PIC16F84. Desarrollo de Proyectos.* 1.ª ed. Alfaomega, 2004.

[3] M. P. Bates, "Chapter 6 - Programming techniques," en *PIC Microcontrollers. An Introduction to Microelectronics*, Elsevier, 2011, ISBN: 978-0-08-096911-4. DOI: 10 . 1016 / C2010 - 0 - 65255 - 2. dirección: https : / / linkinghub . elsevier . com / retrieve / pii / C20100652552 (visitado 17-10-2020).

[4] Scott Campbell. (). "Basics of UART Communication," Circuit Basics, dirección: https://www.circuitbasics.com/basics-uart-communication/ (visitado 18-01-2021).