TP 2

CUESTIONARIO - Ejercicio No 1: Display LCD

a) Características y comandos del LCD 1602A

Características principales:

- Pantalla LCD de 16 caracteres × 2 líneas
- Controlador HD44780 compatible
- Interfaz paralela de 4 u 8 bits
- Voltaje de operación: 5V
- Incluye generador de caracteres con:
  - 5×8 puntos por carácter
  - 240 caracteres programables (CGROM)
  - 8 caracteres definibles por usuario (CGRAM)
- Contraste ajustable por potenciómetro
- Retroiluminación LED (verde/azul/blanca dependiendo del modelo)

Comandos principales:

1. `Clear Display` (0x01) - Limpia la pantalla
2. `Return Home` (0x02) - Mueve cursor a posición inicial
3. `Entry Mode Set` (0x04-0x07) - Configura dirección del cursor
4. `Display On/Off` (0x08-0x0F) - Controla visualización/cursor/parpadeo
5. `Cursor/Display Shift` (0x10-0x1F) - Mueve cursor/desplaza pantalla
6. `Function Set` (0x20-0x3F) - Configura interfaz (4/8 bits), líneas, tamaño de fuente
7. `Set CGRAM Address` (0x40-0x7F) - Define dirección CGRAM
8. `Set DDRAM Address` (0x80-0xFF) - Define dirección DDRAM

b) Diagrama esquemático en Proteus

El diagrama debe incluir:

1. Microcontrolador (PIC16F877A u otro según kit)
2. LCD 1602A con conexiones:
    - VSS (GND), VDD (+5V), VO (contraste a potenciómetro)
    - RS, RW, E (pines de control)
    - D4-D7 (bus de datos en modo 4 bits)
    - A (ánodo LED backlight), K (cátodo LED backlight)
3. Teclado matricial 4×4 conectado a puertos del microcontrolador
4. Resistores de pull-up/pull-down según sea necesario

c) Compatibilidad de niveles lógicos

- El LCD 1602A opera con niveles TTL (0V para LOW, 5V para HIGH)
- Los puertos I/O del microcontrolador en el kit (como el PIC16F877A) también operan a 5V
- Ambos son perfectamente compatibles sin necesidad de conversores de nivel
- La corriente requerida por el LCD (≈1mA por pin) está dentro de la capacidad de los puertos del microcontrolador (≈25mA por pin)

d) Funciones de biblioteca para control del LCD

LCDinit():

- Inicializa el LCD en modo 4 bits
- Configura los pines de control y datos como salidas
- Envía secuencia de inicialización:
    1. Espera >15ms después de encendido
    2. Envía comandos Function Set (modo 4 bits, 2 líneas, caracteres 5×8)
    3. Configura display (on/off, cursor, parpadeo)
    4. Limpia pantalla

5. Establece modo de entrada (incremento cursor sin desplazamiento)

LCDgotoxy(x,y):

- Posiciona el cursor en coordenadas específicas
- Parámetros:
  - x: columna (0-15)
  - y: fila (0-1)
- Calcula dirección DDRAM según posición:
  - Línea 1: 0x80 + x
  - Línea 2: 0xC0 + x
- Envía comando "Set DDRAM Address" con la dirección calculada

LCDstring(*str, length):

- Escribe una cadena de caracteres en la posición actual del cursor
- Parámetros:
  - str: puntero al string a mostrar
  - length: longitud del string
- Itera enviando cada carácter usando la función LCDletra()
  - Maneja automáticamente el salto entre líneas cuando se llega al final de la primera línea

CUESTIONARIO - Ejercicio No 2: Teclado Matricial 4x4

a) Funcionamiento y conexionado del teclado matricial

Funcionamiento:

- El teclado matricial 4×4 consta de 16 teclas organizadas en 4 filas y 4 columnas
- Las teclas actúan como interruptores que conectan una fila con una columna cuando se presionan

- Para detectar teclas presionadas se usa técnica de barrido (scanning):
  1. Se ponen las filas como salidas y columnas como entradas (con pull-up)
  2. Se activa cada fila por turno (nivel bajo)
  3. Se leen las columnas para detectar qué tecla está presionada

Esquema de conexionado:

- 4 pines del MCU conectados a filas (salidas)
- 4 pines del MCU conectados a columnas (entradas con resistencias pull-up)
- Configuración típica en PIC:
  - Filas: PORTA0-3 (salidas)
  - Columnas: PORTA4-7 (entradas con pull-up habilitadas)

b) Función KEYPAD_Scan()

```
/*********************************************************
FUNCION PARA ESCANEAR UN TECLADO MATRICIAL Y DEVOLVER LA
TECLA PRESIONADA UNA SOLA VEZ. TIENE DOBLE VERIFICACION Y
MEMORIZA LA ULTIMA TECLA PRESIONADA
DEVUELVE:
0 -> NO HAYNUEVA TECLA PRESIONADA
1 -> HAY NUEVA TECLA PRESIONADA Y ES *pkey
*********************************************************/

uint8_t KEYPAD_Scan (uint8_t *pkey)
{
 static uint8_t Old_key, Last_valid_key=0xFF; // no hay tecla presionada;
 uint8_t Key,

 Key= KepadUpdate();
 if(Key==0xFF){
          Old_key=0xFF;       // no hay tecla presionada
          Last_valid_key=0xFF;
          return 0;
          }
 if(Key==Old_key) { //2da verificación
          if(Key!=Last_valid_key){ //evita múltiple detección
                    *pkey=Key;
                    Last_valid_key = Key;
                    return 1;
          }
 }
 Old_key=Key; //1era verificación
 return 0;
}
```

CUESTIONARIO - Ejercicio No 3: Interrupciones

a) Secuencia de interrupción

1. Completa la instrucción actual
2. Guarda el PC en la pila (return address)
3. Guarda el registro de estado (SREG)
4. Deshabilita interrupciones globales (I=0)
5. Salta al vector de interrupción correspondiente
6. Ejecuta la ISR (Interrupt Service Routine)
7. Restaura SREG y PC (instrucción RETI)
8. Continúa ejecución normal

b) Bit I en SREG

- El bit I (bit 7) en SREG habilita/deshabilita interrupciones globales
- Valor por defecto tras RESET: 0 (interrupciones deshabilitadas)
- Instrucciones para modificarlo:
  - `SEI` (Set Interrupt) - Habilita (I=1)
  - `CLI` (Clear Interrupt) - Deshabilita (I=0)

c) RESET y MCU Status Register

RESET:

- Reinicia el microcontrolador a estado inicial
- Formas de generarlo:
  1. Power-on Reset (al encender)
  2. Reset externo (pin RESET bajo)
  3. Brown-out Reset (caída de voltaje)
  4. Watchdog Timer Reset

5. Reset por software

MCU Status Register:

- Registro que indica la causa del último reset
- Bits típicos:
  - PORF: Power-on Reset Flag
  - EXTRF: External Reset Flag
  - BORF: Brown-out Reset Flag
  - WDRF: Watchdog Reset Flag

d) Latencia de interrupción

- Tiempo entre ocurrencia de interrupción y ejecución de ISR
- Mínimo: 4 ciclos de reloj (para interrupciones habilitadas)
- Máximo: depende de instrucción en ejecución (instrucciones largas pueden retrasarla)

e) Interrupciones anidadas

- Ocurren cuando una ISR es interrumpida por otra interrupción
- No es posible por defecto (I=0 al entrar en ISR)
- Para habilitar: poner I=1 manualmente dentro de ISR

f) Interrupción Externa (INTx)

Configuración:

- Se puede configurar por flanco (ascendente/descendente) o por nivel
- Diferencia:
  - Flanco: detecta cambio de estado (pulso)
  - Nivel: detecta estado mantenido (alto/bajo)

g) Interrupciones por cambio de estado (PORTx)

Características:

- Terminales que pueden generarlas: depende del MCU (ej. PORTB en PIC)

- Configuración:
    1. Habilitar interrupción por cambio en registro correspondiente
    2. Configurar pines como entradas
    3. Habilitar interrupción global

Vectores y prioridades:

- Vector único para todos los pines del puerto
- Prioridad generalmente menor que INTx

Diferencias con INTx:

- Detecta cambios en cualquier pin del puerto
- No distingue flancos (solo cambio de estado)
- Prioridad generalmente menor
- Requiere lectura de puerto para determinar pin que causó interrupción

CUESTIONARIO - Ejercicio No 4: Temporizadores

a) TIMER0: Componentes y modos de funcionamiento

Componentes principales:

- Registro TMR0 (8 bits) - Contador principal
- Prescaler (divisor de frecuencia programable)
- Fuente de reloj (interno/externo)
- Lógica de desborde e interrupción

Modos de funcionamiento:

1. **Modo temporizador**:
    - Incrementa con ciclo de instrucción (Fosc/4)
    - Usado para retardos y temporizaciones
2. **Modo contador**:

- Incrementa con flancos en pin T0CKI
- Puede contar flancos ascendentes o descendentes

b) Prescaler y cálculos de frecuencia

Preescaler:

- Divisor de frecuencia antes del contador
- Valores típicos: 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128, 1:256

Fórmulas:

- CLKn = CLKo / (4 × Prescaler)
- Frecuencia de desborde = CLKn / (2^n) donde n=8 (para TMR0 de 8 bits)
- Tiempo de desborde = 1/Frecuencia de desborde

Borrar bandera de desborde:

- Se borra leyendo el registro TMR0 o escribiendo en él

c) Configuración para retardo de 1ms y 1000ms

Para 1ms con Fosc=8MHz:

- Ciclo de instrucción = 1/(8MHz/4) = 0.5µs
- Con prescaler 1:8 → Tciclo = 4µs
- Cuentas necesarias = 1ms/4µs = 250
- Valor inicial = 256 - 250 = 6

```
OPTION_REG = 0b10000010; // Prescaler 1:8, fuente interna
TMR0 = 6;
```

Para 1000ms:

- No es posible directamente con TIMER0 (máximo ~32ms con prescaler 1:256)
- Solución: Usar contador de sobreflujos (necesitaría ~31 sobreflujos)

d) Interrupción periódica de 1ms

```
void Init_Timer0() {
    OPTION_REG = 0b10000010; // Prescaler 1:8
    TMR0 = 6;
    INTCONbits.TMR0IE = 1; // Habilitar interrupción
    INTCONbits.GIE = 1; // Habilitar interrupciones globales
}

void interrupt ISR() {
    if(INTCONbits.TMR0IF) {
        TMR0 = 6;
        INTCONbits.TMR0IF = 0;
        // Código a ejecutar cada 1ms
    }
}
```

CUESTIONARIO - Ejercicio No 5: Temporizador Asincrónico-RTC

a) Comparación TIMER0 vs TIMER2

Diferencias:

- TIMER2 tiene postscaler y periodo register (PR2)
- TIMER2 puede generar PWM
- TIMER0 puede usar fuente externa asincrónica
- TIMER2 es más preciso para aplicaciones periódicas

Similitudes:

- Ambos son temporizadores de 8 bits
- Tienen prescaler
- Generan interrupciones por desborde

b) Configuración para interrupción de 10ms con 32.768kHz

- Frecuencia = 32.768kHz / 4 = 8.192kHz
- Tciclo = 122.07µs
- Con prescaler 1:8 → Tciclo = 976.56µs
- Cuentas necesarias = 10ms/976.56µs ≈ 10.24 → usar 10
- Valor PR2 = 10

```
T2CON = 0b00000010; // Prescaler 1:8, timer ON
PR2 = 10;
PIE1bits.TMR2IE = 1;
```

c) Circuitos integrados RTC

DS1302:

- RTC con comunicación serial 3 hilos
- Bajo consumo (menos de 1µA en batería)
- Conexión: 3 pines (CLK, DAT, RST) + Vbat

DS3231:

- Mayor precisión (±2ppm de 0°C a +40°C)
- Comunicación I2C
- Temperatura compensada
- Conexión: 2 pines (SDA, SCL) + Vbat

Ventajas:

- Mantienen hora/fecha sin alimentación
- Mayor precisión que solución con cristal
- Liberan al MCU de tareas de temporización

CUESTIONARIO - Ejercicio No 6: Máquinas de Estado Finitos (MEF)

a) Modelo de Estados Finitos

Elementos:

- Estados finitos (Q)
- Eventos de entrada (Σ)
- Eventos de salida (Δ)
- Función de transición (δ: Q × Σ → Q)
- Función de salida (λ: Q → Δ para Moore, Q × Σ → Δ para Mealy)

Especificación:

- Diagrama de estados
- Tabla de transición de estados
- Descripción textual

b) Moore vs Mealy

Moore:

- Salidas dependen solo del estado actual
- Más fácil de diseñar y depurar
- Requiere más estados para ciertas funciones

Mealy:

- Salidas dependen de estado y entradas
- Más compacto (menos estados)
- Puede tener problemas con glitches

c) Implementación en C

Formas comunes:

1. Switch-case anidados
2. Tabla de transiciones
3. Orientado a objetos (si el lenguaje lo permite)

Ejemplo con switch-case:

```
typedef enum {S0, S1, S2} State;
State currentState = S0;

void UpdateFSM(uint8 input) {
    static uint8 output;

    switch(currentState) {
        case S0:
            if(input == 1) {
                currentState = S1;
                output = 0x01;
            }
            break;
        case S1:
            // Transiciones desde S1
            break;
        // ... otros estados
    }
}
```

d) Inicialización y actualización

Inicialización:

- Establecer estado inicial (usualmente definido en diseño)
- Resetear variables asociadas

Actualización:

- Evaluar entradas
- Determinar transición
- Ejecutar acciones asociadas
- Cambiar estado si corresponde

e) Comparación C vs VHDL

Similitudes:

- Misma lógica de estados

- Mismas transiciones

Diferencias:

- **VHDL**:
  - Implementación concurrente
  - Sensibilidad a señales de reloj
  - Sintaxis específica para máquinas de estado
- **C**:
  - Implementación secuencial
  - Requiere llamada explícita a función de actualización
  - Más flexible en manejo de datos complejos

Explicación

AVRDUDE graba en el micro el program

Drive CH340 convierte USB a serie

1. Conectar la placa (conector de abajo)
2. Abrir admin de disp, puertos COM y LPT, ver que puerto le dió.
3. Configurar la interfaz,:
   a. tool → external tool → crear un nuevo menu y configurar los parametro:
   b. Comando: ruta al AVRDUDE.exe,
   c. Argumento: esta en el help del AVRDUDE (avr/etc/avrdude.conf)
   d. Clickear la casilla Use output Windows,

ahora en tools te debería aparecer que configuraste (depende del nombre que le haya puesto)

Desenchufar y volver enchufar la placa despues de cargar el programa.

Librerias: main.h lcd.h teclado4×4.h

cada uno tiene su .c donde esta el codigo de cada funcion del.h

main.h → includes de avr io.h, interrupt.h y varios mas. las bibliotecas stnder, la frecuencia del micro

empezar a uitlizar ifndef. si esta definida no se ejecuta lo de debajo, si no esta lo hace.

leer hoja datos (deep seek)

leer lcd.c

dato: d7 d6 d5 d4 d3 d2 d1 d0

lo de abajo es para definir una macro

port_KP: (DATO & (1<<7))>>3 → equivale al Puerto 7 del dato. Es un puerto virtual para el Key Pad

```
uint8_t KepadUpdate(void)
{
    uint8_t r,c;

    PORTB|= 0x0F;

    for(c=0;c<4;c++)
    {
        DDRB&=~(0XFF);

        DDRB|=(0X80>>c);

        for(r=0;r<4;r++)
        {
            if(!(PINB & (0X08>>r)))
            {
                return (r*4+c);
            }
        }
    }

    return 0XFF; //tecla No presionada
}
```

Lo que tenemos q hacer es implementar la macro del puerto virtual y reemplazarlo por PINB para leer del teclado y luego imprimirlo en el lcd.