Universidad de San Carlos de Guatemala.
Facultad de ingeniería.
Escuela de ingeniería en ciencias y sistemas.
Área: Ciencias de la computación
Nombre del curso: Arquitectura de computadores y ensambladores 1
Catedrático: m.sc. Luis Fernando Espino Barrios
Auxiliar: Carlos Antonio Velásquez Castellanos

Manual Técnico

Proyecto #2

No.	Nombre:	Carné
1	Henderson Migdo Baten Hernandez	201019694
2	Selim Idair Ergon Castillo	201801300
3	Jemima Solmaira Chavajay Quiejú	201801521
4	Giovanni Saul Concohá Cax	202100229
5	Johan Moises Cardona Rosales	202201405
6	Estiben Yair Lopez Leveron	202204578
7	Santiago Julián Barrera Reyes	201905884

28 de junio, 2024

Contenido

Introducción 3
Arquitectura del Sistema 4
Sensores 4
Backend (Python) 4
Procesamiento de Datos (ARM) 5
Frontend 5
Flujo de Datos 5
Especificaciones del Hardware 6
Especificaciones del Software 7
Sistema Operativo 7
Estructura del código 10
Estructura Códigos Ensamblador 17
Resultado del código fuente 29
Costos 30
Lista de materiales, costos asociados y presupuesto total del proyecto. 30

Introducción

El cambio climático es uno de los mayores desafíos globales contemporáneos, manifestándose a través del aumento de las temperaturas, alteraciones en los patrones de precipitación y la creciente frecuencia e intensidad de eventos climáticos extremos. En este contexto, el monitoreo continuo y preciso de las condiciones meteorológicas locales es esencial para comprender y mitigar estos efectos adversos.

La implementación de una estación meteorológica, como la descrita en este proyecto, proporciona datos cruciales para la investigación climática, la gestión de recursos naturales y la protección de las comunidades frente a eventos climáticos adversos. Este proyecto tiene como objetivo desarrollar una estación meteorológica que pueda recolectar, procesar y visualizar datos meteorológicos en tiempo real, utilizando una combinación de sensores y tecnologías de procesamiento de datos.

Para lograrlo, se integrarán sensores para medir diversos parámetros atmosféricos (temperatura, humedad, velocidad del viento, luminosidad, calidad del aire y presión barométrica) con un sistema de procesamiento basado en un lenguaje ensamblador ARM, respaldado por un backend en Python y un frontend adaptable para la visualización de los datos. Este manual técnico documenta el diseño, implementación y funcionamiento de la estación meteorológica, proporcionando una guía detallada para su replicación y uso.

Arquitectura del Sistema

La estación meteorológica se compone de cuatro elementos fundamentales que trabajan en conjunto para recolectar, procesar y mostrar datos meteorológicos en tiempo real. A continuación, se detalla cada componente de la arquitectura del sistema:

Sensores

Los sensores son dispositivos encargados de recoger datos meteorológicos de diferentes parámetros atmosféricos. Los tipos de sensores utilizados y sus funciones son los siguientes:

- **Sensor de Temperatura**: Mide la temperatura ambiente en grados centígrados.
- **Sensor de Humedad**: Mide la humedad relativa en el aire.
- **Sensor de Velocidad del Viento (Anemómetro)**: Mide la velocidad del aire.
- **Sensor de Luminosidad**: Mide la cantidad de luz ambiental, indicando si está soleado o nublado.
- **Sensor de Calidad del Aire**: Mide los niveles de partículas o gases en el ambiente, indicando la calidad del aire.
- **Sensor de Presión Barométrica**: Mide la presión atmosférica.

Backend (Python)

El backend es responsable de la comunicación y el control de los sensores. Utilizando Python, el backend realiza las siguientes tareas:

- **Recibe Datos de los Sensores**: Obtiene los datos recogidos por cada sensor.
- **Envía Datos para Procesamiento**: Transfiere los datos obtenidos a un programa escrito en lenguaje ensamblador ARM para su análisis y procesamiento.
- **Almacena Datos**: Guarda los datos recolectados en archivos locales para su posterior análisis.

Procesamiento de Datos (ARM)

El procesamiento de datos se realiza mediante un programa escrito en lenguaje ensamblador ARM. Este componente se encarga de realizar análisis estadísticos sobre los datos recolectados. Las operaciones estadísticas incluyen:

- **Promedio**: Calcula el valor medio de los datos.
- **Mediana**: Encuentra el valor central en un conjunto de datos ordenados.
- **Desviación Estándar**: Mide la variabilidad o dispersión de los datos respecto a la media.
- **Máximo y Mínimo**: Identifica los valores máximos y mínimos.
- **Moda**: Determina el valor que más se repite en el conjunto de datos.
- **Contador**: Cuenta el número de datos registrados.
- **Contador de Calidad de Aire**: Diferencia entre el número de veces que la calidad del aire es buena o mala.

Frontend

El frontend es la interfaz que permite a los usuarios visualizar los datos procesados. Este componente puede ser desarrollado utilizando cualquier herramienta o framework y se encarga de:

- **Mostrar Datos en la Página Web**: Presenta los datos recolectados y procesados en una interfaz web amigable.
- **Selector de Sensores**: Permite al usuario seleccionar y visualizar datos específicos de cada sensor.
- **Actualización en Tiempo Real**: Muestra datos actualizados a intervalos regulares.

Flujo de Datos

- **Recolección de Datos**: Los sensores recolectan datos atmosféricos y los envían al backend en Python.
- **Almacenamiento y Envío para Procesamiento**: El backend almacena los datos y los envía al programa en ARM para su análisis.
- **Análisis Estadístico**: El programa en ARM realiza los cálculos estadísticos y devuelve los resultados al backend.
- **Visualización**: El backend envía los datos procesados al frontend, donde se muestran en una página web.

Especificaciones del Hardware

Para la implementación de la estación meteorológica, se requiere el uso de varios componentes de hardware. A continuación, se detallan las especificaciones de cada uno de los componentes necesarios:

- Raspberry Pi 3 o Superior
- Sensor de Temperatura
- Sensor de Humedad
- Sensor de Velocidad del Viento (Anemómetro)
- Fotoresistencia (Sensor de Luminosidad)
- Sensor de Calidad del Aire
- Sensor de Presión Barométrica

Especificaciones del Software

Para la implementación de la estación meteorológica, se requiere el uso de varias tecnologías y herramientas de software que permitirán la integración, procesamiento y visualización de los datos recolectados. A continuación, se detallan las especificaciones de cada uno de los componentes de software necesarios:

Sistema Operativo

- Distribución: Raspbian (basada en Debian)
- Versión: Raspbian Buster o posterior
- Requisitos:
- Compatible con Raspberry Pi 3 o superior
- Soporte para Python 3 y bibliotecas necesarias

Lenguaje de Programación

- Python 3.7 o superior
- RPi.GPIO: Para el control de los pines GPIO de la Raspberry Pi
- Adafruit_DHT: Para la lectura de datos del sensor DHT11/DHT22
- smbus: Para la comunicación I2C con sensores como BMP180/BMP280
- requests o flask: Para la comunicación entre backend y frontend
- Assembler (ARM)
- Arquitectura**:** ARMv7-A
- gcc-arm-none-eabi: Compilador para ARM
- gdb-multiarch: Depurador para ARM
- QEMU: Emulador para pruebas

Backend

El backend en Python es responsable de la comunicación con los sensores, recolección de datos y envío de los mismos para su procesamiento.

Funciones Principales:

- **Recolección de Datos:** Lee datos de los sensores y los almacena en archivos.
- **Comunicación:** Envía los datos al programa en lenguaje ensamblador ARM para su procesamiento.
- **Almacenamiento:** Guarda los datos en archivos locales para análisis posteriores.

Procesamiento de Datos

El procesamiento de datos se realiza en lenguaje ensamblador ARM. Este componente se encarga de realizar análisis estadísticos sobre los datos recolectados.

Operaciones Estadísticas:

- **Promedio**: Calcula el valor medio de los datos.
- **Mediana**: Encuentra el valor central en un conjunto de datos ordenados.
- **Desviación Estándar**: Mide la variabilidad o dispersión de los datos respecto a la media.
- **Máximo y Mínimo**: Identifica los valores máximos y mínimos.
- **Moda**: Determina el valor que más se repite en el conjunto de datos.
- **Contador**: Cuenta el número de datos registrados.
- **Calidad del Aire**: Diferencia entre el número de veces que la calidad del aire es buena o mala.

Frontend

El frontend es la interfaz que permite a los usuarios visualizar los datos procesados. Este componente puede ser desarrollado utilizando cualquier herramienta o framework.

Requisitos:

- **Frameworks Sugeridos**: Flask, Django, React, Angular, Vue.js

Funciones Principales:

- **Mostrar Datos**: Presenta los datos recolectados y procesados en una interfaz web amigable.
- **Selector de Sensores**: Permite al usuario seleccionar y visualizar datos específicos de cada sensor.
- **Actualización en Tiempo Real**: Muestra datos actualizados a intervalos regulares.

Comunicación entre Componentes

Protocolos:

- **I2C**: Utilizado para la comunicación entre la Raspberry Pi y algunos sensores (BMP180/BMP280).
- **GPIO**: Utilizado para la comunicación con sensores digitales (DHT11, DHT22, LDR, MQ-135, Anemómetro).
- **Archivos Locales**: Para el almacenamiento temporal de datos antes de su procesamiento.
- **HTTP/REST**: Opcional, para la comunicación entre backend y frontend si se usa un servidor web (Flask, Django).

Herramientas y Bibliotecas

- **Raspbian**: Sistema operativo base.
- **Python 3**: Lenguaje principal para el backend.
- **RPi.GPIO**: Control de GPIO.
- **Adafruit_DHT**: Lectura de sensores DHT11/DHT22.
- **smbus**: Comunicación I2C.
- **requests/flask**: Comunicación HTTP (opcional).
- **Assembler (ARM)**: Lenguaje para el procesamiento de datos.
- **gcc-arm-none-eabi**: Compilador.
- **gdb-multiarch**: Depurador.
- **QEMU**: Emulador.
- **Frontend**: Frameworks y herramientas para la interfaz de usuario.
- **Flask/Django**: Servidor web (opcional).
- **React/Angular/Vue.js**: Frameworks de frontend.

Estructura del código

app.py

Importación de Librerías

Se importan librerías necesarias para crear el servidor web y manejar los sensores.

Librería ctypes

Permite interactuar con bibliotecas compartidas y DLLs (Dynamic Link Libraries). Esto es útil para llamar funciones escritas en C, C++, o cualquier otro lenguaje que pueda compilarse en una biblioteca compartida. Con ctypes, puedes cargar estas bibliotecas, llamar funciones, y manipular datos en una manera que sea compatible con Python.

```
from flask import Flask, request, jsonify
from flask_cors import CORS

import board
import busio

import adafruit_dht
from adafruit_bmp280 import Adafruit_BMP280_I2C

import smbus2

import ctypes
import os

import sys
import time
```

Inicialización de la Aplicación

Se crea una instancia de Flask (app) y se configura CORS para permitir solicitudes desde cualquier origen.

```
app = Flask(__name__)
cors = CORS(app, resources={r"/api/*": {"origins": "*"}},
supports_credentials=True)
```

Biblioteca Compartida

Se carga una biblioteca compartida (calculos.so) para realizar cálculos estadísticos.

```
lib = ctypes.CDLL(os.path.abspath("calculos.so"))
```

Variables Globales

Variables globales para almacenar el estado y los datos de los sensores.

```
datos = []
resultados = []

sensor_aire = False
valor_sensoraire = 0
Switch_Sensoraire = False

sensor_temp = False

sensor_luminosidad = False

sensor_viento = False

sensor_barometrico = False
```

Configuración de Pines y Buses I2C

Asignación de pines GPIO y configuración de buses I2C para los sensores.

```
PIN_AIRE = 11
dht_device = None
PIN_VIENTO = 22
PIN_LUZ = 13

PCF8591_ADDRESS = 0x48
bus = smbus2.SMBus(1)

i2c = busio.I2C(board.SCL, board.SDA)
bmp280 = Adafruit_BMP280_I2C(i2c, address=0x76)
bmp280.sea_level_pressure = 1013.25
```

Rutas de la API

Definición de las rutas de la API para encender/apagar sensores y obtener datos.

```python
@app.route('/api/on', methods=['POST'])
def On_Sensor():
    data = request.get_json()
    sensor = data.get('sensor')
    # Lógica para encender el sensor...
    return jsonify({'message': sensor})

@app.route('/api/off', methods=['POST'])
def Off_Sensor():
    data = request.get_json()
    sensor = data.get('sensor')
    # Lógica para apagar el sensor...
    return jsonify({'message': sensor})

@app.route('/api/stats', methods=['GET'])
def Estadistics_Sensor():
    data = {
        "labels": ['Promedio', 'Mediana', 'DesviacionEstandar', 'Máximo',
'Mínimo', 'Moda'],
        "datasets": [
            {
                "label": 'Revenue',
                "backgroundColor": '#4e73df',
                "borderColor": '#4e73df',
                "data": [resultados[0], resultados[1], resultados[2],
resultados[3], resultados[4], resultados[5]],
            },
        ],
    }
    return jsonify(data)

@app.route('/api/data', methods=['GET'])
def Data_Sensor():
    calculos_estadisticos()
    data = {
        "Promedio": resultados[0],
        "Mediana": resultados[1],
        "DesviacionEstandar": resultados[2],
        "Máximo": resultados[3],
        "Mínimo": resultados[4],
        "Moda": resultados[5]
    }
    return jsonify(data)
```

Funciones para el Control de Sensores

Funciones para encender y apagar cada tipo de sensor y manejar la lectura de datos.

```
def On_Sensor():
    # Lógica para encender sensores según el tipo...

def Off_Sensor():
    # Lógica para apagar sensores según el tipo...

def On_aire():
    global sensor_aire
    sensor_aire = True
    print("Leyendo datos de la calidad de aire de la zona...")
    Datos_sensoraire()

def Off_aire():
    global Switch_Sensoraire
    Switch_Sensoraire = False
    bus_sensoraire.close()
    print("Sensor de calidad de aire apagado...")

def On_Temp_Hum(tipo_sensor):
    global sensor_temp, dht_device, datos
    dht_device = adafruit_dht.DHT11(board.D4)
    datos = []
    sensor_temp = True
    while sensor_temp:
        try:
            temperature_c = dht_device.temperature
            humidity = dht_device.humidity
            if tipo_sensor == 'temp':
                datos.append(temperature_c)
            else:
                datos.append(humidity)
        except RuntimeError as error:
            print(f"Error al leer el sensor: {error.args[0]}")
        time.sleep(2.0)

def Off_Temp_Hum():
    global sensor_temp
    sensor_temp = False

def On_Viento():
    sensor_viento = True
    global counter, last_value, start_time, THRESHOLD, datos
    datos = []
```

```
    last_value = read_analog(2)
    while sensor_viento:
        value = read_analog(2)
        if last_value < THRESHOLD and value >= THRESHOLD:
            counter += 1
        last_value = value
        elapsed_time = time.time() - start_time
        if elapsed_time >= 1.0:
            rpm = (counter / 20) / elapsed_time * 60
            print(f"Velocidad de rotación: {rpm:.2f} RPM")
            counter = 0
            start_time = time.time()
            datos.append(rpm)
            time.sleep(1)

def Off_Viento():
    global sensor_viento, bus
    sensor_viento = False
    bus.close()

def On_luminosidad():
    global sensor_luminosidad
    sensor_luminosidad = True
    datos.clear()
    while sensor_luminosidad:
        analog_value = Luminosidad_analogo(1)
        print("Valor analógico leído desde AO1: ", analog_value)
        time.sleep(10)

def Off_luminosidad():
    global sensor_luminosidad, bus
    sensor_luminosidad = False
    bus.close()

def On_Presure_Barometric():
    global sensor_barometrico, datos
    sensor_barometrico = True
    datos.clear()
    while sensor_barometrico:
        temperatura = bmp280.temperature
        presion = bmp280.pressure
        altitud = bmp280.altitude
        print(f"Temperatura: {temperatura:.2f} C")
        print(f"Presión: {presion:.2f} hPa")
        print(f"Altitud: {altitud:.2f} m")
        datos.append(presion)
        time.sleep(10)
```

```python
def Off_Presure_Barometric():
    global sensor_barometrico
    sensor_barometrico = False
```

Lectura y Clasificación de Datos de Sensores

Funciones para leer y clasificar los datos de los sensores.

```python
def leerADC(canal):
    bus_sensoraire.write_byte(PCF8591_ADDRESS, canal)
    valor_sensoraire = bus_sensoraire.read_byte(PCF8591_ADDRESS)
    return valor_sensoraire

def Clasificar_Calidad(aireVoltaje):
    if aireVoltaje < 1.5:
        return "Calidad de Aire Buena..."
    else:
        return "Calidad de Aire Mala..."

def Datos_sensoraire():
    global valor_sensoraire, Switch_Sensoraire, datos
    Switch_Sensoraire = True
    datos = []
    while Switch_Sensoraire:
        Valor_adc = leerADC(0x40)
        Voltage = Valor_adc / 255.0 * 3.3
        print(f"Voltaje obtenido: {Voltage: .2f} V")
        datos.append(Voltage)
        sesgo = Clasificar_Calidad(Voltage)
        print(sesgo)
        time.sleep(10)
    print("Sensor de calidad de aire desactivado...")
```

Cálculos Estadísticos

Realiza cálculos estadísticos utilizando la biblioteca compartida y almacena los resultados.

```
def calculos_estadisticos():
    global datos, lib, resultados
    resultados = []
    c_array = (ctypes.c_int * len(datos))(*datos)
    result_suma = lib.desviacion(c_array, len(datos))
    print(f"La suma de los valores es: {result_suma}")
    resultados.append(result_suma)
```

Ejecución del Servidor

Manejo de errores y ejecución del servidor en el puerto 8000 con modo de depuración activado.

```
try:
    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=8000, debug=True)
except KeyboardInterrupt:
    print("\nLectura finalizada por el usuario.")
    sys.exit(0)
except ImportError:
    print("\nError de importación de librerias.")
    sys.exit(1)
except Exception as e:
    print(f"Se genero un Error: {e}")
    sys.exit(1)
```

Estructura Códigos Ensamblador

Calculo.s (Desviación Estándar)

Definiciones Globales

Se definen las etiquetas `desviacion`, `conteo1`, y `conteo0` como globales para que puedan ser llamadas desde otros módulos.

```
.global desviacion, conteo1, conteo0
```

Función desviación

Calcula la desviación estándar de un arreglo de números flotantes.

```
desviacion:
    // x0 = puntero al arreglo, x1 = número de elementos
    mov w2, #0                // Inicializar el índice
    fmov s0, wzr              // Inicializar la suma a 0.0
    fmov s2, wzr              // Inicializar la suma de los cuadrados a 0.0
(s2 para precisión simple)

loop:
    ldr s1, [x0, x2, lsl #2]  // Cargar el siguiente valor del arreglo en s1
    fadd s0, s0, s1           // s0 += s1
    add x2, x2, #1            // Incrementar el índice
    cmp x2, x1                // Comparar el índice con el número de elementos
    b.lt loop                 // Si el índice es menor, repetir el bucle

    // Convertir x1 (número de elementos) a flotante en s1
    ucvtf s1, x1              // s1 = (float)x1
    fdiv s0, s0, s1           // s0 /= s1

    // Calcular la suma de los cuadrados de las diferencias
    mov w2, #0                // Reiniciar el índice
loop_square:
    ldr s3, [x0, x2, lsl #2]  // Cargar el siguiente valor del arreglo en s3
    fsub s4, s3, s0           // s4 = s3 - s0
    fabs s4, s4               // Valor absoluto de s4
    fmul s4, s4, s4           // s4 = s4 * s4 (elevar al cuadrado)
    fadd s2, s2, s4           // Sumar al acumulador de cuadrados
    add x2, x2, #1            // Incrementar el índice
    cmp x2, x1                // Comparar el índice con el número de elementos
```

```
    b.lt loop_square          // Si el índice es menor, repetir el bucle de
cuadrados

    // Dividir la suma de los cuadrados por el número de elementos
    fdiv s2, s2, s1           // s2 /= s1

    // Calcular la raíz cuadrada del resultado
    fsqrt s0, s2              // s0 = sqrt(s2)

    ret                       // Retornar el resultado en s0
```

Función conteo1

Cuenta el número de elementos iguales a 1 en un arreglo de enteros.

```
conteo1:
    // Los primeros dos argumentos (el arreglo y su longitud) se pasan en los
registros x0 y x1
    mov w2, #0                // Inicializar el contador a 0

    // Crear un bucle que recorra el arreglo
contador_loop:
    cbz x1, fin               // Si x1 (la longitud del arreglo) es 0, hemos
terminado

    ldr w3, [x0], #4          // Cargar el valor actual del arreglo en w3 y
avanzar el puntero del arreglo

    cmp w3, #1                // Comparar el valor actual del arreglo con 1
    beq sum                   // Si es igual a 1, ir a la etiqueta `sum`

    sub x1, x1, #1            // Decrementar el contador de elementos
restantes
    b contador_loop           // Volver al inicio del bucle

sum:
    add w2, w2, #1            // Incrementar el contador de elementos iguales
a 1
    sub x1, x1, #1            // Decrementar el contador de elementos
restantes
    b contador_loop           // Volver al inicio del bucle

fin:
    mov w0, w2                // Mover el valor del contador a x0
    ret                       // Retornar
```

Función conteo0

Cuenta el número de elementos iguales a 0 en un arreglo de enteros.

```
conteo0:
    // Los primeros dos argumentos (el arreglo y su longitud) se pasan en los
registros x0 y x1
    mov w2, #0                // Inicializar el contador a 0

    // Crear un bucle que recorra el arreglo
contador_loop:
    cbz x1, fin               // Si x1 (la longitud del arreglo) es 0, hemos
terminado

    ldr w3, [x0], #4          // Cargar el valor actual del arreglo en w3 y
avanzar el puntero del arreglo

    cmp w3, #0                // Comparar el valor actual del arreglo con 0
    beq sum                   // Si es igual a 0, ir a la etiqueta `sum`

    sub x1, x1, #1            // Decrementar el contador de elementos
restantes
    b contador_loop           // Volver al inicio del bucle

sum:
    add w2, w2, #1            // Incrementar el contador de elementos iguales
a 0
    sub x1, x1, #1            // Decrementar el contador de elementos
restantes
    b contador_loop           // Volver al inicio del bucle

fin:
    mov w0, w2                // Mover el valor del contador a x0
    ret                       // Retornar
```

Max.s (Maximo)

Definiciones Globales y Sección de Código

Define la etiqueta `findMax` como global y especifica que el código estará en la sección de texto (código ejecutable).

```
.global findMax
.section .text
```

Función `findMax`

Encuentra el valor máximo en un arreglo de enteros.

```
findMax:
    // Argumentos:
    // x0 - dirección del arreglo
    // x1 - longitud del arreglo

    // Si la longitud del arreglo es 0, devolver 0
    cbz x1, done
```

Finalización:

Mueve el valor máximo (x2) a x0 para devolverlo, retorna de la función (ret).

```
    done:
    // Mover el resultado (el máximo) a x0 para devolverlo
    mov x0, x2
    ret
```

Función findMin

Esta función encuentra el valor mínimo en un arreglo de enteros.

```
// findMin: Finds the minimum value in an array
// Inputs:
//   x0: Pointer to the array
//   x1: Length of the array
// Output:
//   x0: Minimum value
```

Entradas:

- `x0`: Puntero al arreglo.
- `x1`: Longitud del arreglo.

Salida:

- `x0`: Valor mínimo.

Código de la Función

Carga el primer elemento del arreglo en `w0`, que se usará para almacenar el valor mínimo encontrado hasta ahora.

```
findMin:
    // Initialize x0 (result) with the first element of the array
    ldr w0, [x0]
```

Bucle de Comparación:

Condición de Fin: Si `x1` (la longitud restante del arreglo) es 0, salta a `done` para terminar la función.

```
    // Create a loop to compare each element
    loop:
        // If x1 (length) is 0, we're done
        cbz x1, done
```

Cargar el Siguiente Elemento:

Carga el siguiente elemento del arreglo en w2 y avanza el puntero del arreglo x0 al siguiente elemento (4 bytes adelante para enteros de 32 bits).

```
    // Create a loop to compare each element
    loop:
        // If x1 (length) is 0, we're done
        cbz x1, done
```

Comparación y Actualización:

Comparar: Compara el valor actual en w2 con el valor mínimo actual en w0.

Seleccionar Condicionalmente: Si w2 es menor que w0 (lt - less than), w0 se actualiza con w2, de lo contrario, w0 permanece sin cambios.

```
        // Compare with the current minimum
        cmp w2, w0
        csel w0, w0, w2, lt  // Update minimum if w2 < w0
```

Actualizar Longitud y Continuar:

Decrementa x1 (longitud del arreglo restante) en 1.

Salta al inicio del bucle (b loop) para procesar el siguiente elemento.

```
        // Decrement length and continue
        sub x1, x1, #1
        b loop
```

Max.s (Máximo)

Función findMax

Esta función encuentra el valor máximo en un arreglo de enteros.

```
 // Encuentra el valor m?nimo en un arreglo de enteros
findMax:
    // Argumentos:
    // x0 - direcci?n del arreglo
    // x1 - longitud del arreglo
```

Verificación de Longitud

```
   // Si la longitud del arreglo es 0, devolver 0
    cbz x1, done
```

Inicialización de Variables

```
     // Inicializar x2 con el primer valor del arreglo (valor m?nimo inicial)
    ldr w2, [x0]

    // Ajustar x0 para que apunte al segundo elemento
    add x0, x0, #4
    sub x1, x1, #1
```

Bucle Principal

```
    // Crear un bucle que recorra el arreglo
    loop:
        // Si x1 (la longitud restante del arreglo) es 0, hemos terminado
        cbz x1, done

        // Cargar el valor actual del arreglo en w3
        ldr w3, [x0], #4
```

Comparar y Seleccionar el Máximo:

- Compara el valor actual (w3) con el valor máximo encontrado hasta ahora (w2).
- Usa la instrucción csel (conditional select) para actualizar w2 con el valor mayor entre w2 y w3.

```
        // Decrementar x1 y continuar con el siguiente elemento
        sub x1, x1, #1
        b loop
```

Finalización

```
    done:
    // Mover el resultado (el m�nimo) a x0 para devolverlo
    mov x0, x2
    ret
```

Moda. S (Moda)

Esta función encuentra el valor más frecuente en un arreglo de enteros.

```
// Encuentra la moda (valor más frecuente) en un arreglo de enteros
moda:
    // Argumentos:
    // x0 - dirección del arreglo
    // x1 - longitud del arreglo

    // Si la longitud del arreglo es 0, devolver 0 (o un valor indicativo de
que no hay moda)
    cbz x1, done
```

Inicialización de Variables Locales

```
    // Variables locales
    mov x2, #0       // Valor actual
    mov x3, #0       // Moda actual
    mov x4, #0       // Frecuencia del valor actual
    mov x5, #0       // Frecuencia máxima encontrada
```

Bucle Principal

```
    // Crear un bucle que recorra el arreglo
    loop:
        // Cargar el valor actual del arreglo en w2
        ldr w2, [x0], #4
```

Cargar el Valor Actual:

Carga el siguiente valor del arreglo en w2 y avanza el puntero del arreglo x0 al siguiente elemento (4 bytes adelante para enteros de 32 bits).

```
        // Si el valor actual es diferente al valor anterior
        cmp x2, w2
        bne update_count
```

Comparar con el Valor Anterior:

- Compara el valor actual (w2) con el valor anterior (x2).
- Si son diferentes (bne - branch if not equal), salta a update_count para actualizar las cuentas.

Incrementar Frecuencia:

- Si el valor es el mismo que el anterior, incrementa la frecuencia del valor actual (x4).
- Salta a continue_loop para continuar con el siguiente elemento.

```
        // Incrementar la frecuencia del valor actual
        add x4, x4, #1
        b continue_loop
```

Actualización de Cuentas

```
    update_count:
        // Si la frecuencia del valor actual es mayor que la frecuencia máxima
encontrada hasta ahora
        cmp x4, x5
        csel x3, x3, x2, gt  // Actualizar la moda si la frecuencia actual es
mayor
```

```
        mov x5, x4          // Actualizar la frecuencia máxima
        mov x2, w2          // Actualizar el valor actual
        mov x4, #1          // Reiniciar la frecuencia del nuevo valor
```

Actualizar Moda y Frecuencias:

- Compara la frecuencia del valor actual (x4) con la frecuencia máxima encontrada (x5).
- Si la frecuencia del valor actual es mayor (gt - greater than), actualiza la moda (x3) con el valor actual (x2).
- Actualiza la frecuencia máxima (x5) con la frecuencia del valor actual (x4).
- Actualiza el valor actual (x2) con el nuevo valor (w2).
- Reinicia la frecuencia del nuevo valor (x4) a 1.

```
    continue_loop:
        // Decrementar x1 y continuar con el siguiente elemento
        sub x1, x1, #1
        cbnz x1, loop
```

Continuar el Bucle

```
    continue_loop:
        // Decrementar x1 y continuar con el siguiente elemento
        sub x1, x1, #1
        cbnz x1, loop
```

Continuar con el Siguiente Elemento:

- Decrementa x1 (longitud del arreglo restante) en 1.
- Si x1 no es cero (cbnz - compare and branch if not zero), salta a loop para procesar el siguiente elemento.

Finalización

```
    done:
    // Si hemos terminado y el último valor tiene la frecuencia más alta,
actualizar la moda
    csel x0, x3, x2, gt
    ret
```

Contador.s

Función contadorDatos

Esta función cuenta la cantidad de elementos en un arreglo de enteros.

Comentarios Iniciales

```
// Cuenta la cantidad de datos en un arreglo de enteros
contadorDatos:
    // Argumentos:
    // x0 - dirección del arreglo
```

Inicialización de Variables

```
    // Inicializar x1 (contador) a 0
    mov x1, #0
```

Bucle Principal

```
    // Crear un bucle que recorra el arreglo
    loop:
        // Cargar el valor actual del arreglo en w2
        ldr w2, [x0], #4
```

Cargar el Valor Actual:

Carga el siguiente valor del arreglo en `w2` y avanza el puntero del arreglo `x0` al siguiente elemento (4 bytes adelante para enteros de 32 bits).

```
        // Incrementar el contador (x1) independientemente del valor cargado
        add x1, x1, #1
```

Incrementar el Contador:

Independientemente del valor cargado en w2, incrementa el contador x1 en 1.

```
        // Continuar con el siguiente elemento si no hemos llegado al final
        cbnz w2, loop
```

Continuar el Bucle:

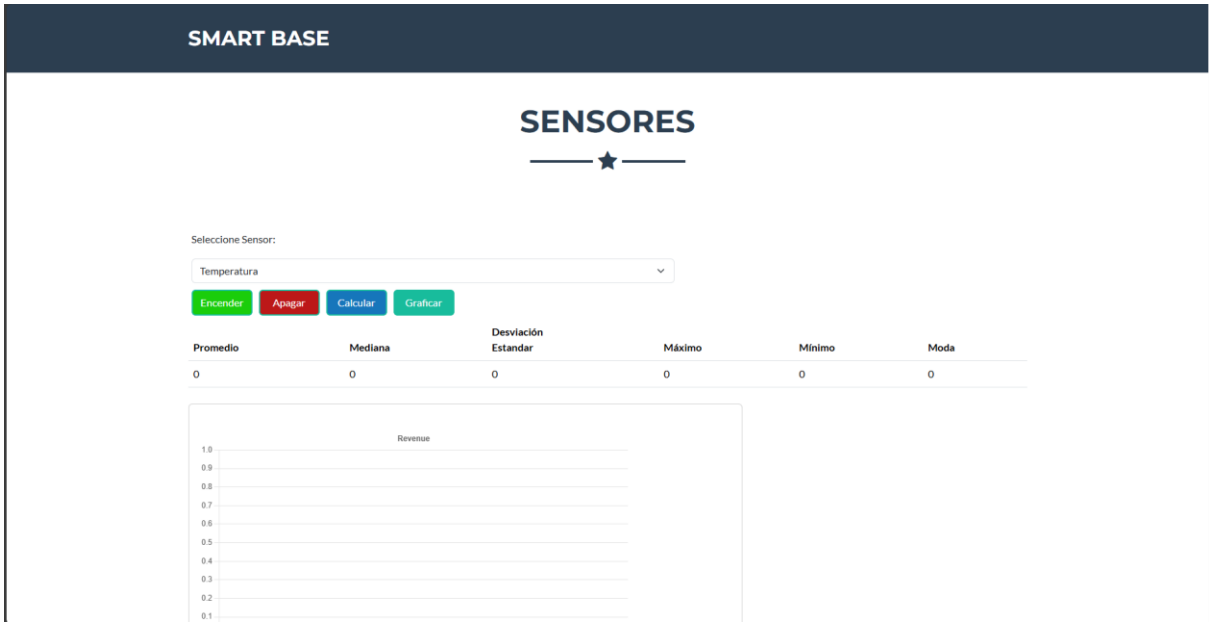
Si w2 no es cero (cbnz - compare and branch if not zero), salta a loop para procesar el siguiente elemento.

Finalización

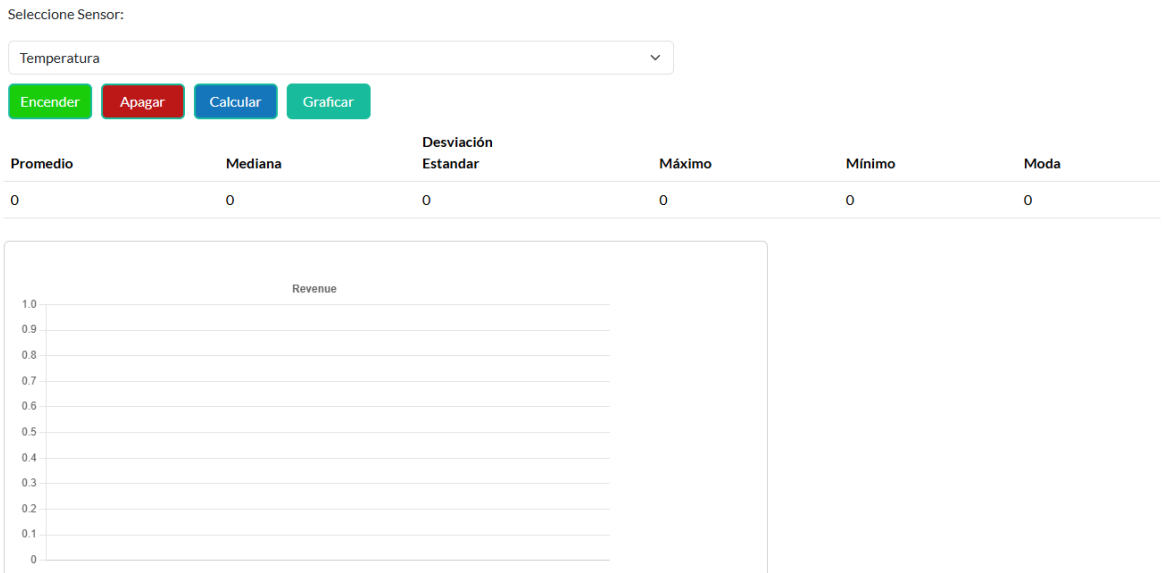
```
// Mover el contador (la cantidad de elementos) a x0 para devolverlo
 mov x0, x1
 ret
```

Resultado del código fuente

La interfaz que se muestra en la imagen es de una aplicación llamada “SMART BASE” que parece estar diseñada para interactuar con varios sensores. En la sección “SENSORES”, puedes seleccionar un sensor específico de una lista desplegable que incluye opciones como Temperatura, Presión, Humedad, Aire, Calidad del aire y Gases.



Una vez seleccionado un sensor, la interfaz muestra estadísticas para ese sensor en particular. Las estadísticas incluyen el Promedio, la Mediana, la Desviación Estándar, el Máximo, el Mínimo y la Moda. En la imagen proporcionada, todos estos valores se muestran como cero, lo que podría indicar que aún no se han recopilado datos del sensor seleccionado o que el sensor está apagado.



Costos

Lista de materiales, costos asociados y presupuesto total del proyecto.

Componentes:			
Descripción	**Unidades**	**Precio Unidad (Q)**	**Total**
Raspberry	1	Ya se tenía	0
Cargador Raspberry	1	Ya se tenía	0
Resistencias	6	0.50	3
Sensor de Temperatura y Humedad DHT11	1	34	34
Sensor de Calidad de aire MQ-135	1	39	39
Sensor de presión Barométrica BMP280	1	26	26
Módulo de foto interruptor para encoder óptico	1	29	29
Foto resistencia analógicos Digitales	1	10	10
Jumpers 20cm	20	1.25	25
Micro SD	1	Ya se tenía	0
Subtotal:			**166**

Materiales para maqueta:			
Descripción	**Unidades**	**Precio Unidad (Q)**	**Total**
Cartón Chip	1	23	23
Silicon Caliente	2	2	4
Papel Cartulina	2	6	12
Aerosol Negro y Gris	2	23	46
Kit Impresiones 3D	1	130	130
Fieltro Verde y Café	8	2.50	20
Subtotal:			**235**

Costo total del prototipo Q401.00