Documentación Técnica

Arquitectura por capas:

- **Nivel Usuario:** Es la lógica principal de la aplicación, el objetivo es interactuar con el usuario de manera sencilla y la complejidad la manejan las otras capas.
    - **Archivo**
        - **user/main.c:** Este archivo implementa la funcionalidad principal de la calculadora, consiste en un bucle infinito que permite realizar operaciones aritméticas continuas sin necesidad de volver a compilar el archivo.
    - **Responsabilidades Implementadas**
        - **Interacción con el usuario:** utiliza las funciones PRINT y READ para comunicarse.
        - **Procesamiento de entrada:** Solicita y a la vez captura dos números ya sean enteros o decimales.
        - **Operaciones aritméticas:** Realiza la suma de los números capturados sin importar si es entero o decimal.
        - **visualización de los resultados**
    - **Flujo de datos en esta Capa**
        - **Llamada a "PRINT":** El usuario quiere mostrar un mensaje por lo cual el string se envía a la capa de librería para ser procesada.
        - **Llamada a "READ":** El programa se detiene para que el usuario ingrese un valor.
        - **Calculo:** Se utiliza la operación matemática que se usa en C.
        - **Salida:** Se utiliza print para poder mostrar el resultado de la suma.

- **Nivel de Librería de Lenguaje:** Su función es proporcionar utilidades que facilitan operaciones de alto nivel sin que el programador de la aplicación tenga que evitar darles formato a los datos y de realizar las conversiones necesarias entre tipos de datos.
    - **Archivos**
        - **lib/stdio.c y lib/stdio.h:** Implementan las funciones centrales de entrada y salida formateada las cuales son "PRINT" y "READ"
            - **PRINT:** Analiza un string carácter por carácter, al detectar %d, %f, etc, extraerá el argumento

correspondiente y utiliza uart_itoa o uart_ftoa para convertir el valor en texto.

- **READ:** Gestiona la entrada de datos del usuario, usando un buffer captura la entrada a través de uart_gets_input para después transformar el string en valores tipo int o float.

- **lib/string.c y lib/string.h:** Manipula las cadenas de texto de la forma mas básica.
  - **my_strncpy:** permite realizar copias seguras de cadenas de texto limitando el número de caracteres.

- **Responsabilidades Implementadas:**
  - **Gestion de I/O:** Implementa lo necesario para procesar cadenas con varios datos en una sola instrucción.
  - **Cnversion de tipos de datos:** Actúa como intermediario ya que llama a las funciones de conversión en la capa OS para pasar de string a int o float y viceversa.
  - **Independencia del Hardware:** Esta capa no puede acceder a las direcciones de memoria del periférico UART.
- **Flujo de datos en esta capa:**
  - **Entrada desde el usuario:** La capa recibe una solicitud desde main.c con una cadena de formato (por ejemlo: "%f + %f = %f").
  - **Procesamiento:** Si se necesita hacer una lectura solicita datos a la capa OS, recibe el texto, lo convierte al tipo de dato correcto y lo devuelve al usuario.
  - **Salida hacia el hardware:** Si se hace una escritura, toma los datos del usuario, los convierte a caracteres ASCII y a nivel de OS se envían uno tras otro.

- **Nivel de OS:** Interactuar directamente con el hardware del procesador ARM y sus periféricos.
  - **Archivos:**
    - **os/os.c y os/os.h:** Gestiona la comunicación serial y las conversiones base de datos.
      - **Comunicación UART:** Implementa uart_putc u uart_getc para enviar y recibir bytes individuales manipulando los registros UART_DR y UART_FR.

- **Entrada de datos:** La funcion uart_gets_input captura secuencias de caracteres hasta detectar un salto de línea, almacenándolas en un buffer.
- **Conversion de datos:** Incluye funciones criticas para transformar datos crudos como uart_atoi/uart_itoa que realiza la conversión entre enteros y ASCII o uart_atof/uart_ftoa que realiza la conversión entre números de punto flotante y ASCII para que la calculadora pueda operar con decimales.
- **os/root.s:** código de arranque en lenguaje ensamblador, ya que como la memoria crece hacia abajo entonces el sp inicializa donde empieza la dirección de memoria de la RAM.
- **os/liker.ld:** Aquí se define el mapa de memoria.

- **Responsabilidades Implementadas:**
  - **Control de Hardware:** Acceso directo a periféricos mediante punteros a direcciones de memoria especificas.
  - **Gestion de entorno de ejecución:** Configuracion inicial del stack y salto al código C.
- **Flujo de datos en esta capa:**
  - **Entrada:** Los bits llegan al registro UART_DR, el os.c los lee cuando UART_FR_RXFE indica que hay datos disponibles y los agrupa en cadenas.
  - **Salida:** Recibe caracteres de la librería espera a que UART_FR_TXFF indique un espacio disponible para escribir el byte en el registro de transmisión.

Guia de compilación y ejecución:

El script compila cada capa por separado y enlaza con libgcc para soportar operaciones matemáticas de punto flotante. Se usan:

arm-none-eabi-gcc -c -Ios -Ilib -o main.o user/main.c

arm-none-eabi-gcc -nostartfiles -T os/linker.ld -o calculadora.elf root.o main.o os.o stdio.o string.o -lgcc

Demostración

```
aldo@aldo-VMware-Virtual-Platform:~/Escritorio/LabsCC7/Lab2_002Calculadora$ ./build_and_run.sh
Cleaning up previous build files...
Assembling root.s...
Compiling OS Level (os.c)...
Compiling Library Level (stdio.c and string.c)...
Compiling User Level (main.c)...
Linking object files with libgcc...
Converting ELF to binary...
Running QEMU...
Program: Add Two Numbers
-----------------------------
Enter first number: 50
Enter second number: 50
50 + 50 = 100
Enter first float number: 1.2
Enter second float number: 1.2
1.20 + 1.20 = 2.40
-----------------------------
Enter first number: 10
Enter second number: 40
10 + 40 = 50
Enter first float number: 6.2
Enter second float number: 1.1
6.19 + 1.10 = 7.29
-----------------------------
Enter first number: QEMU: Terminated
aldo@aldo-VMware-Virtual-Platform:~/Escritorio/LabsCC7/Lab2_002Calculadora$
```