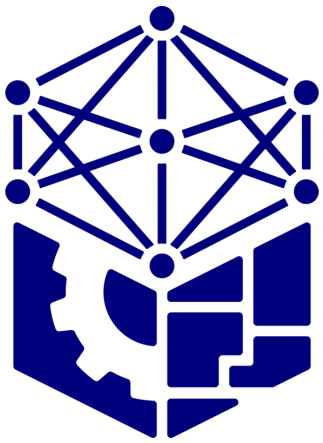
UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS



Lenguaje Ensamblador

Proyecto de implementacion de una calculadora usando assembly

DOCENTE: Emilio Palomino Olivera

ALUMNO: Daniel R. Alegria Sallo (215270)

Perú
Julio del 2025

Tabla de Contenido

Introduccion 3

Capitulo 1: Aspectos Generales 4
1.1. Descripcion del problema 4
1.2. Objetivos 4
1.2.1. Objetivo General 4
1.2.2. Objetivos Especificos 4
1.3. Limitaciones 5
1.4. Delimitaciones 5
1.5. Justificacion 6
1.6. Metodologia 6

Capitulo 2: Desarrollo del proyecto 8
2.1. Sprints 8
2.1.1. Sprint 1: Implementación Base 8
2.1.2. Sprint 2: Refinamiento de Interfaz y Lógica de Conversión 8
2.1.3. Sprint 3: Migración Completa a Ensamblador 9

Concluciones 10

Anexos 11
A Repositorio del Proyecto 11
A Codigo Fuente 11

Introduccion

La Unidad Aritmético-Lógica (ALU, por sus siglas en inglés) constituye uno de los componentes fundamentales de cualquier procesador moderno, siendo responsable de ejecutar todas las operaciones aritméticas y lógicas que requiere un sistema computacional. Su desarrollo se remonta a los primeros días de la computación electrónica, cuando pioneros como John von Neumann y su equipo conceptualizaron la arquitectura que llevaría su nombre en la década de 1940. Desde entonces, las ALUs han evolucionado considerablemente, incorporando capacidades cada vez más sofisticadas para el manejo de diferentes tipos de datos, incluyendo enteros, números en punto flotante, y operaciones en múltiples bases numéricas. Esta evolución ha sido impulsada por la constante demanda de mayor eficiencia computacional y la necesidad de procesar información de manera más precisa y versátil.

En el contexto académico y profesional actual, existe una creciente necesidad de comprender los fundamentos de las operaciones aritméticas a nivel de hardware, especialmente cuando se trata de sistemas que deben manejar diferentes bases numéricas y tipos de datos numéricos. Los calculadores tradicionales y las implementaciones de software de alto nivel a menudo abstraen estos detalles, limitando la comprensión profunda de cómo se ejecutan realmente las operaciones matemáticas en el procesador. Esta abstracción, si bien útil para el desarrollo de aplicaciones, representa un obstáculo para estudiantes y profesionales que buscan entender los mecanismos subyacentes de la computación aritmética, particularmente en escenarios que involucran conversiones entre bases, manejo de números negativos mediante representación en complemento a dos, y operaciones con números en punto flotante según estándares como IEEE 754.

Para abordar esta problemática, se propone el desarrollo de una calculadora implementada en lenguaje ensamblador que sea capaz de realizar operaciones aritméticas básicas entre números de diferentes bases, incluyendo el soporte completo para números en punto flotante y enteros con signo. La solución arquitectónica elegida consiste en un servidor web desarrollado en lenguaje C que sirve una interfaz gráfica HTML intuitiva, mientras que el núcleo computacional se ejecuta directamente en código ensamblador. Esta aproximación híbrida permite combinar la accesibilidad de una interfaz web moderna con la transparencia y control total que ofrece la programación en ensamblador, proporcionando así una herramienta educativa y práctica que expone los mecanismos fundamentales de las operaciones aritméticas a nivel de procesador, facilitando la comprensión de conceptos como la representación binaria, las conversiones de base, y la implementación de algoritmos aritméticos de bajo nivel.

Capitulo 1: Aspectos Generales

1.1. Descripcion del problema

La educación en ciencias de la computación y la ingeniería de sistemas enfrenta un desafío significativo en la enseñanza de los fundamentos de la aritmética computacional. Los estudiantes y profesionales a menudo interactúan con calculadoras y sistemas de alto nivel que ocultan completamente los procesos subyacentes de conversión entre bases numéricas, representación de números negativos, y manejo de operaciones en punto flotante. Esta abstracción, aunque funcional para el uso cotidiano, genera una brecha conceptual crítica que impide la comprensión profunda de cómo los procesadores ejecutan realmente estas operaciones.

El problema se manifiesta particularmente cuando se requiere trabajar con sistemas embebidos, optimización de algoritmos, o depuración de código a bajo nivel, donde el conocimiento detallado de la representación numérica y las operaciones aritméticas resulta fundamental. Las herramientas educativas existentes tienden a ser demasiado abstractas o, en el caso contrario, demasiado técnicas y poco accesibles para el aprendizaje progresivo. Además, la mayoría de las implementaciones disponibles no permiten la visualización transparente de los procesos internos de conversión entre diferentes bases numéricas (binaria, octal, decimal, hexadecimal) ni exponen claramente cómo se manejan los casos especiales como el desbordamiento aritmético, la representación en complemento a dos, o los estándares de punto flotante.

La ausencia de herramientas que combinen accesibilidad, transparencia educativa y precisión técnica representa un obstáculo significativo para la formación integral en computación de bajo nivel. Se requiere una solución que permita no solo realizar cálculos aritméticos entre diferentes bases numéricas, sino que también exponga de manera clara y comprensible los mecanismos internos de estas operaciones, facilitando así el aprendizaje de conceptos fundamentales como la arquitectura de procesadores, la representación de datos en memoria, y la implementación de algoritmos aritméticos a nivel de lenguaje ensamblador.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar una calculadora aritmética implementada en lenguaje ensamblador con interfaz web que permita realizar operaciones matemáticas entre números de diferentes bases numéricas, incluyendo soporte para números en punto flotante y enteros con signo, con el propósito de proporcionar una herramienta educativa que exponga los mecanismos fundamentales de la aritmética computacional a nivel de procesador.

1.2.2. Objetivos Especificos

- **Implementar algoritmos aritméticos en lenguaje ensamblador** que ejecuten las operaciones básicas (suma, resta, multiplicación, división) para números enteros y en punto flotante, siguiendo los estándares de representación IEEE
  754.

- **Desarrollar funciones de conversión entre bases numéricas** que permitan transformar números entre sistemas binario, octal, decimal y hexadecimal, manteniendo la precisión y manejando casos especiales como desbordamiento aritmético.
- **Crear una arquitectura híbrida servidor-cliente** utilizando un servidor HTTP implementado en lenguaje C que sirva una interfaz gráfica HTML intuitiva, estableciendo comunicación eficiente entre la interfaz de usuario y el núcleo computacional en ensamblador.
- **Implementar manejo robusto de números negativos** mediante representación en complemento a dos, asegurando compatibilidad con las operaciones aritméticas y conversiones de base para todo el rango de valores soportados.
- **Diseñar una interfaz de usuario educativa** que visualice claramente los procesos internos de cálculo, incluyendo representaciones binarias intermedias, pasos de conversión, y resultados en múltiples bases simultáneamente.
- **Validar la precisión y robustez del sistema** mediante pruebas exhaustivas que cubran casos límite, operaciones con diferentes combinaciones de bases, y verificación de cumplimiento con estándares aritméticos establecidos.

1.3. Limitaciones

El desarrollo de esta calculadora aritmética en lenguaje ensamblador presenta varias limitaciones inherentes que constituyen obstáculos significativos para la implementación completa del sistema. La **dependencia de la arquitectura de procesador** representa la principal restricción, ya que el código ensamblador está íntimamente ligado a la arquitectura específica del hardware objetivo, limitando la portabilidad del sistema y requiriendo reimplementación para diferentes plataformas.

La **complejidad de implementación de operaciones en punto flotante** constituye otro obstáculo considerable, especialmente en el manejo preciso de casos especiales como infinitos, NaN (Not a Number), y operaciones de redondeo según el estándar IEEE 754. Esta complejidad se incrementa cuando se requiere mantener consistencia entre diferentes bases numéricas y asegurar que las conversiones preserven la precisión matemática esperada.

Las **limitaciones de recursos de memoria y rendimiento** imponen restricciones adicionales, particularmente en el manejo de números de alta precisión y operaciones que requieren múltiples conversiones de base consecutivas. Además, la **interfaz entre el servidor C y el código ensamblador** presenta desafíos de sincronización y transferencia de datos que pueden afectar la estabilidad del sistema, especialmente durante operaciones concurrentes o con cargas de trabajo intensivas.

1.4. Delimitaciones

El alcance del proyecto se limita específicamente a la implementación de las **cuatro operaciones aritméticas fundamentales**: suma, resta, multiplicación y división. Esta restricción se establece para mantener la viabilidad del desarrollo en lenguaje ensamblador y asegurar una implementación robusta y bien optimizada de cada operación.

Las operaciones se desarrollarán para **números enteros con signo de 32 bits** y **números en punto flotante de precisión simple (32 bits)** según el estándar IEEE 754, excluyendo deliberadamente operaciones más complejas como exponenciación, logaritmos, funciones trigonométricas o raíces cuadradas. El sistema soportará conversiones entre **bases numéricas desde la base 2 hasta la base 16** (inclusivo), cubriendo todas las bases comúnmente utilizadas en computación, desde binaria hasta hexadecimal.

El **rango de valores soportados** se limita a los definidos por los tipos de datos de 32 bits, con manejo específico de casos de desbordamiento aritmético, excluyendo deliberadamente el soporte para valores especiales de punto flotante como infinito y NaN. No se incluirán operaciones con números de precisión extendida ni aritmética de múltiple precisión, manteniendo el enfoque en la demostración clara de los principios fundamentales de la aritmética computacional.

1.5. Justificacion

El desarrollo de esta calculadora aritmética en lenguaje ensamblador se justifica por la creciente necesidad de comprender los fundamentos de la computación a nivel de hardware en un contexto académico y profesional donde las abstracciones de alto nivel dominan la enseñanza. La implementación directa en ensamblador proporciona una **ventana transparente hacia los mecanismos internos** de las operaciones aritméticas, permitiendo observar cómo el procesador manipula realmente los datos binarios, ejecuta las conversiones de base, y maneja la representación de números negativos mediante complemento a dos.

La elección de una **arquitectura híbrida servidor-cliente** se fundamenta en la necesidad de combinar la accesibilidad moderna con la profundidad técnica. La interfaz web HTML permite un acceso universal sin requerir instalaciones específicas, mientras que el núcleo en ensamblador mantiene la transparencia educativa esencial. Esta aproximación facilita la visualización simultánea de representaciones en múltiples bases numéricas, exponiendo procesos que normalmente permanecen ocultos en implementaciones de software tradicionales.

1.6. Metodologia

El cronograma real del proyecto se distribuyó en **tres fases de desarrollo** distintas, abarcando un período total de aproximadamente 7 semanas con intervalos de inactividad planificados:

Fase 1: Implementación Base (7-8 de Junio 2025)

- Desarrollo inicial del servidor localhost en lenguaje C
- Implementación básica de la interfaz gráfica de usuario
- Integración inicial con FASM (Flat Assembler) para operaciones básicas
- Establecimiento de la arquitectura fundamental servidor-cliente

Fase 2: Refinamiento de Interfaz (11-13 de Julio 2025)

- Implementación completa de conversiones numéricas en C como prototipo
- Desarrollo y ajuste de la interfaz HTML con cajas de resultados
- Preparación del código C/ensamblador para implementación final

- Optimización de la experiencia de usuario y diseño visual

Fase 3: Implementación Final (24-27 de Julio 2025)

- Migración completa de operaciones de C hacia ensamblador (`ops.asm`)
- Implementación correcta de operaciones en punto flotante
- Refinamiento final del estilo de la interfaz gráfica
- Refactorización y reorganización de archivos fuente con reglas de compilación optimizadas
- Documentación técnica y preparación del sistema para distribución

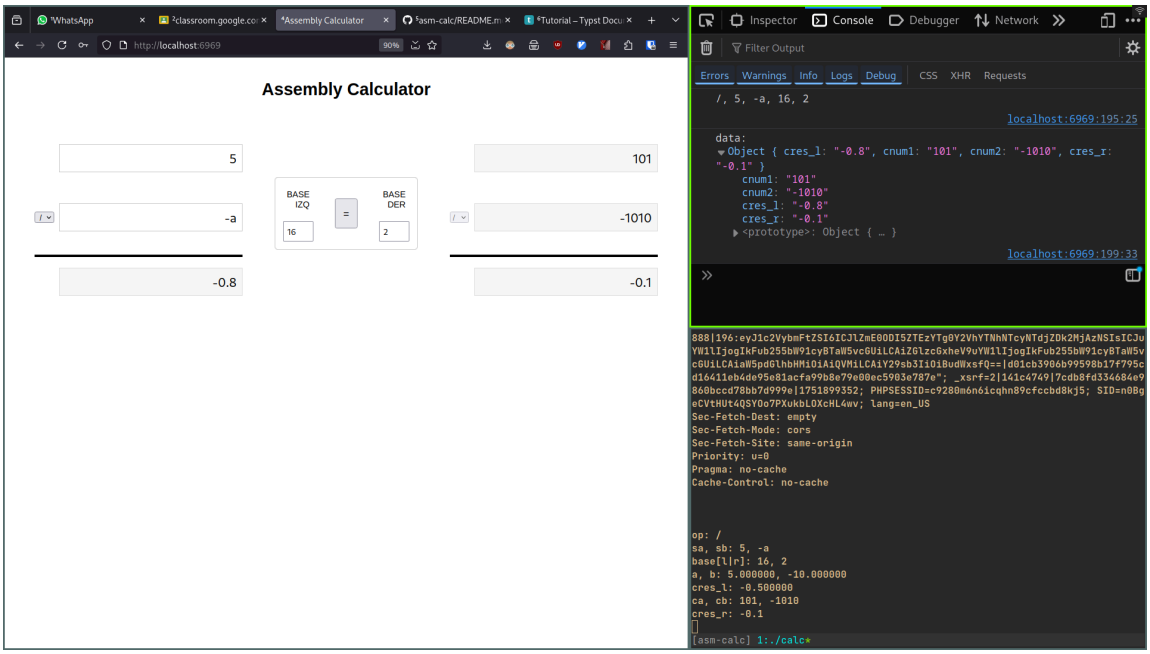


Figura 3: Programa final corriendo

Esta metodología iterativa permitió la evolución natural del proyecto desde un prototipo funcional hasta una implementación completa en ensamblador, manteniendo la funcionalidad en cada iteración mientras se profundizaba progresivamente en el nivel de implementación.

Capitulo 2: Desarrollo del proyecto

2.1. Sprints

2.1.1. Sprint 1: Implementación Base

El primer sprint se enfocó en establecer la arquitectura fundamental del sistema, desarrollando los componentes básicos necesarios para la comunicación servidor-cliente y las operaciones aritméticas iniciales. Durante esta fase, se implementó el **servidor HTTP en lenguaje C** utilizando sockets de Berkeley para manejar las peticiones web, estableciendo endpoints específicos para cada operación aritmética y configurando el sistema de archivos estáticos para servir la interfaz HTML.

El **código C inicial** incluyó la implementación de un parser básico para procesar las peticiones HTTP POST, extraer los parámetros numéricos y las bases especificadas, y formatear las respuestas en JSON para la interfaz cliente. Se desarrollaron funciones auxiliares para la conversión entre diferentes bases numéricas como prototipo, permitiendo validar la lógica de conversión antes de su migración a ensamblador. Paralelamente, se creó la **interfaz HTML básica** con formularios simples que permiten la entrada de dos operandos y la selección de operación aritmética, utilizando JavaScript para realizar peticiones asíncronas al servidor y mostrar los resultados.

La integración inicial con **FASM (Flat Assembler)** se limitó a operaciones básicas de suma y resta para enteros, estableciendo la interfaz de llamadas entre C y ensamblador mediante convenciones de llamada estándar. Este sprint concluyó con un sistema funcional capaz de realizar operaciones aritméticas básicas, aunque con limitaciones en el manejo de diferentes bases y tipos de datos.

2.1.2. Sprint 2: Refinamiento de Interfaz y Lógica de Conversión

El segundo sprint se centró en mejorar significativamente la experiencia de usuario mediante la actualización de la interfaz gráfica para soportar **entradas independientes de base numérica**. Se modificó el HTML para incluir dos campos de entrada separados para las bases de los operandos, permitiendo operaciones entre números de diferentes sistemas numéricos.

La interfaz se enriqueció con **cajas de resultados múltiples** que muestran simultáneamente el resultado en diferentes bases numéricas, proporcionando una visualización educativa completa del proceso de conversión. Se implementaron validaciones del lado cliente en JavaScript para verificar que los números ingresados sean válidos en sus respectivas bases, mejorando la robustez del sistema y la experiencia de usuario.

Durante esta fase se desarrolló el **núcleo matemático del sistema** en C, implementando funciones críticas como `str_to_base10f()` para convertir cadenas de cualquier base (2-16) a números en punto flotante, manejando casos especiales como números negativos, puntos decimales y dígitos hexadecimales. La función complementaria `base10f_to_str()` realizaba la conversión inversa desde punto flotante hacia representación de cadena en cualquier base, implementando algoritmos de separación de parte entera y fraccionaria con precisión

configurable. Estas funciones constituyeron la base para las operaciones transparentes entre diferentes bases numéricas que caracterizarían el sistema final.

2.1.3. Sprint 3: Migración Completa a Ensamblador

El tercer y final sprint se enfocó exclusivamente en la **traducción completa del código C hacia FASM**, transformando las funciones de referencia desarrolladas en el sprint anterior en implementaciones nativas de ensamblador. Las operaciones aritméticas básicas, que habían sido prototipadas como funciones simples en C (`asm_fadd`, `asm_fsub`, `asm_fmul`, `asm_fdiv`), fueron completamente reimplementadas en el módulo `ops.asm` utilizando instrucciones del coprocesador matemático x87.

La **migración de las funciones de conversión** representó el mayor desafío técnico, requiriendo la traducción de algoritmos complejos de manipulación de cadenas y aritmética en punto flotante desde C hacia ensamblador puro. Se mantuvieron exactamente los mismos algoritmos y lógica, pero implementándolos directamente con instrucciones de procesador para maximizar la transparencia educativa del sistema.

El sprint concluyó con **pruebas exhaustivas** comparando los resultados entre las implementaciones C originales y las nuevas versiones en ensamblador para validar la correctitud de la migración. Se realizó la refactorización final del código fuente, se optimizaron las reglas de compilación en el Makefile, y se estableció la estructura de proyecto definitiva. El resultado final fue un sistema completamente implementado en ensamblador que mantiene toda la funcionalidad desarrollada en los sprints anteriores, proporcionando máxima transparencia en los procesos de cálculo aritmético.

Concluciones

El desarrollo de la calculadora aritmética en lenguaje ensamblador con interfaz web alcanzó exitosamente todos los objetivos planteados, validando la viabilidad de crear herramientas educativas que combinen transparencia técnica con accesibilidad moderna.

La **implementación de algoritmos aritméticos en ensamblador** se completó satisfactoriamente para las cuatro operaciones básicas en números enteros y punto flotante. La migración desde prototipos en C hacia FASM expuso directamente las instrucciones del coprocesador x87, demostrando el manejo real de aritmética IEEE 754.

Las **funciones de conversión entre bases numéricas** lograron transformaciones precisas entre bases 2-16, manejando correctamente números negativos, decimales y casos de desbordamiento. Las implementaciones `str_to_base10f()` y `base10f_to_str()` proporcionan una base sólida para la comprensión de representaciones numéricas.

La **arquitectura híbrida servidor-cliente** demostró alta efectividad, combinando un servidor HTTP en C con interfaz HTML que comunica eficientemente con el núcleo en ensamblador. Esta aproximación logró accesibilidad universal manteniendo profundidad técnica.

La **implementación de números negativos** mediante complemento a dos se integró completamente con todas las operaciones y conversiones, asegurando compatibilidad para el rango completo de 32 bits soportado.

La **validación del sistema** mediante pruebas exhaustivas cubrió casos límite y diferentes combinaciones de bases. Las pruebas comparativas entre implementaciones C y ensamblador confirmaron la correctitud de la migración y la confiabilidad del sistema.

En conclusión, el proyecto demostró que es posible crear herramientas técnicamente profundas sin sacrificar accesibilidad, proporcionando una ventana única hacia los fundamentos de la aritmética computacional.

Anexos

A Repositorio del Proyecto

https://github.com/daniel-alegria3/asm-calc

A Codigo Fuente

calc.c

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdbool.h>
#include <math.h>

#define PORT 6969
#define BUFFER_SIZE 128*1024 // 128 kb?

//[ external functions
extern void str_to_base10f(char *str, float *num, int base);
extern void base10f_to_str(float num, char *str, int base);
extern float asm_fadd(float a, float b);
extern float asm_fsub(float a, float b);
extern float asm_fmul(float a, float b);
extern float asm_fdiv(float a, float b);
//]

void send_response(int csocket, const char *response);
void handle_request(int csocket, const char *request);

void signal_handler(int sig);

static int server_fd = -1;
static volatile sig_atomic_t shutdown_requested = 0;

int main() {

    //[ Setup interrupt signal handlers
    struct sigaction sa;
    sa.sa_handler = signal_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    sigaction(SIGINT, &sa, NULL);
    sigaction(SIGTERM, &sa, NULL);
    //]

    // Make a localhost TCP server
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == 0) {
```

```
        perror("socket failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // If server_fd is already in use, re-use it
    int opt = 1;
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Give the server_fd an address
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Accept client connection requests
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Server running on http://localhost:%d\n", PORT);
    printf("Press Ctrl+C to shut down gracefully\n");

    // Mainloop
    int client_socket;
    while (!shutdown_requested) {
                client_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen);

        if (client_socket < 0) {
            if (shutdown_requested) {
                printf("Server shutdown requested, stopping accept loop\n");
                break;
            }
            perror("accept");
            continue;
        }

        char buffer[BUFFER_SIZE] = {0};
        read(client_socket, buffer, BUFFER_SIZE);

        handle_request(client_socket, buffer);
```

```
        close(client_socket);
    }

    if (server_fd != -1) {
        close(server_fd);
        printf("Server socket closed\n");
    }

    printf("Server shutdown complete\n");
    return 0;
}


void send_response(int csocket, const char *response) {
    char http_response[BUFFER_SIZE];
    snprintf(http_response, sizeof(http_response),
        "HTTP/1.1 200 OK\r\n"
        "Content-Type: application/json\r\n"
        "Access-Control-Allow-Origin: *\r\n"
        "Content-Length: %zu\r\n"
        "\r\n"
        "%s",
        strlen(response), response);
    send(csocket, http_response, strlen(http_response), 0);
}

void handle_request(int csocket, const char *request) {
    printf("Raw request:\n%s\n", request);

    // Calculate operation
    if (strstr(request, "GET /calculate")) {
        char op;
        char sa[128+1], sb[128+1];
        int base_l, base_r;

        // Extract operation and operands from query string
          if (sscanf(request, "GET /calculate?op=%c&a=%[^&]&b=%[^&]&bl=%d&br=%d",
&op, sa, sb, &base_l, &base_r) == 5) {
            printf("\n");
            printf("op: %c\n", op);
            printf("sa, sb: %s, %s\n", sa, sb);
            printf("base[l|r]: %d, %d\n", base_l, base_r);

            float a, b;
            str_to_base10f(sa, &a, base_l);
            str_to_base10f(sb, &b, base_l);
            printf("a, b: %f, %f\n", a, b);

            float result;
            switch(op) {
                case '+': result = asm_fadd(a, b); break;
                case '-': result = asm_fsub(a, b); break;
                case '*': result = asm_fmul(a, b); break;
                case '/': result = asm_fdiv(a, b); break;
```

```
                default:
                    send_response(csocket, "{\"error\":\"Invalid operation\"}");
                    return;
            }
            printf("cres_l: %f\n", result);

            char cres_l[128+1];
            char cres_r[128+1];
            base10f_to_str(result, cres_l, base_l);
            base10f_to_str(a, sa, base_r);
            base10f_to_str(b, sb, base_r);
            base10f_to_str(result, cres_r, base_r);
            printf("ca, cb: %s, %s\n", sa, sb);
            printf("cres_r: %s\n", cres_r);

            char response[512];
            snprintf(response, sizeof(response),
                        "{ \"cres_l\":\"%s\", \"cnum1\":\"%s\", \"cnum2\":\"%s\",
\"cres_r\":\"%s\" }",
                cres_l, sa, sb, cres_r
            );
            send_response(csocket, response);
            return;
        }
    }

    // Serve HTML file for root path
    if (strstr(request, "GET / ")) {
        FILE *file = fopen("gui.html", "r");
        if (file) {
            fseek(file, 0, SEEK_END);
            long size = ftell(file);
            fseek(file, 0, SEEK_SET);

            char *html = malloc(size + 1);
            fread(html, 1, size, file);
            fclose(file);

            char http_response[BUFFER_SIZE + size];
            snprintf(http_response, sizeof(http_response),
                "HTTP/1.1 200 OK\r\n"
                "Content-Type: text/html\r\n"
                "Content-Length: %ld\r\n"
                "\r\n"
                "%s",
                size, html);

            send(csocket, http_response, strlen(http_response), 0);
            free(html);
            return;
        }
    }

    // Default 404 response
```

```
    send_response(csocket, "{\"error\":\"Not found\"}");
}


void signal_handler(int sig) {
    printf("\nReceived signal %d. Shutting down gracefully...\n", sig);
    shutdown_requested = 1;

    if (server_fd != -1) {
        close(server_fd);
        server_fd = -1;
    }
}
```

gui.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Assembly Calculator</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 650px;
            margin: 0 auto;
            padding: 20px;
        }

        .calc {
            display: flex;
            align-items: center;
            justify-content: center;
            gap: 40px;
            margin-top: 40px;
        }
        .grid {
            display: grid;
            grid-template-columns: 1fr 4fr;
            grid-template-rows: 100px 100px;
            gap: 10px;
            margin: 20px;
        }
        .cell {
            display: flex;
            align-items: center;
            justify-content: center;
            // border: 1px solid #ccc;
            // background-color: #f0f0f0;
        }
        .middle-grid-box {
            border: 1px solid #ccc;
            padding: 15px;
            border-radius: 5px;
```

```
            display: flex;
            align-items: center;
            justify-content: center;
            gap: 40px;
        }
        .middle-grid {
            display: flex;
            flex-direction: column;
            align-items: center;
            gap: 15px;
        }
        .display {
            padding: 10px;
            font-size: 1.5em;
            text-align: right;
            border: 1px solid #ccc;
        }

        button {
            width: 100%;
            box-sizing: border-box;
            padding: 15px;
            font-size: 1.2em;
        }
        input:disabled {
            background-color: #f5f5f5;
            color: #000;
            border-color: #ddd;
            cursor: not-allowed;
        }
        select:disabled {
            cursor: not-allowed;
        }
    </style>
</head>
<body>
    <h1 style="text-align: center;">Assembly Calculator</h1>
      <h2 style="text-align: center;">Universidad Nacional San Antonio Abad del Cusco</h1>
    <h3 style="text-align: center;">por Alegria Sallo Daniel Rodrigo (215270)</h3>

    <div class="calc">
        <div class="grid">
              <!-- <input type="text" id="operator" class="display" style="grid-column: span 1;" readonly> -->
            <!-- <input type="text" id="result" class="display" readonly> -->

            <div class="cell"></div>
            <div class="cell">
                <input type="text" id="num1" class="display" style="">
            </div>

            <div class="cell">
                <select style="text-align: center;" id="op">
```

```
                    <!-- <option value="">Op</option> -->
                    <option value="+">+</option>
                    <option value="-">-</option>
                    <option value="*">*</option>
                    <option value="/">/</option>
                    <!-- <option value="^">-</option> -->
                </select>
            </div>
            <div class="cell">
                <input type="text" id="num2" class="display" style="">
            </div>

             <div style="grid-column: span 2; border-top: 5px solid #000; margin:
10px 0;"></div>

            <div class="cell"></div>
            <div class="cell">
                <input type="text" id="cres_l" class="display" style="" disabled>
            </div>
        </div>

        <div class="middle-grid-box">
            <div class="middle-grid">
                <label for="base_l" style="
                    width: 100%;
                    box-sizing: border-box;
                    padding: 7px;
                    font-size: 1em;
                    text-align: right;
                    grid-column: span 6;
                ">BASE IZQ</label>
                <input type="text" id="base_l" style="
                    width: 100%;
                    box-sizing: border-box;
                    padding: 7px;
                    font-size: 1em;
                    text-align: left;
                    grid-column: span 1;
                "value=10>
            </div>

            <div class="middle-grid">
                <button onclick="calculate()" class="equals">=</button>
            </div>

            <div class="middle-grid">
                <label for="base_r" style="
                    width: 100%;
                    box-sizing: border-box;
                    padding: 7px;
                    font-size: 1em;
                    text-align: right;
                    grid-column: span 6;
                ">BASE DER</label>
```

```
                <input type="text" id="base_r" style="
                    width: 100%;
                    box-sizing: border-box;
                    padding: 7px;
                    font-size: 1em;
                    text-align: left;
                    grid-column: span 1;
                "value=10>
            </div>
        </div>

        <div class="grid">
            <div class="cell"></div>
            <div class="cell">
                <input type="text" id="out1" class="display" style="" disabled>
            </div>

            <div class="cell">
                <select id="op_mirror" disabled>
                    <!-- <option value="">Op</option> -->
                    <option value="+">+</option>
                    <option value="-">-</option>
                    <option value="*">*</option>
                    <option value="/">/</option>
                    <!-- <option value="^">-</option> -->
                </select>
            </div>
            <div class="cell">
                <input type="text" id="out2" class="display" style="" disabled>
            </div>

             <div style="grid-column: span 2; border-top: 5px solid #000; margin:
10px 0;"></div>

            <div class="cell"></div>
            <div class="cell">
                <input type="text" id="cres_r" class="display" style="" disabled>
            </div>
        </div>
    </div>

    <script>
        function calculate() {
            const op = document.getElementById('op');

            const num1 = document.getElementById('num1');
            const num2 = document.getElementById('num2');
            const cres_l = document.getElementById('cres_l');
            const base_l = document.getElementById('base_l');

            const out1 = document.getElementById('out1');
            const out2 = document.getElementById('out2');
            const cres_r = document.getElementById('cres_r');
            const base_r = document.getElementById('base_r');
```

```
            if (op.value === '/' && parseFloat(num2.value) == 0) {
                return;
            }

               if (num1.value !== '' && num2.value !== '' && op.value !== '' &&
base_l.value !== '' && base_r.value !== '') {
                        console.log(`${op.value}, ${num1.value}, ${num2.value},
${base_l.value}, ${base_r.value}`)
                    fetch(`/calculate?op=${op.value}&a=${num1.value}&b=${num2.value}
&bl=${base_l.value}&br=${base_r.value}`)
                        .then(response => response.json())
                        .then(data => {
                            console.log("data: ", data);
                            out1.value = data.cnum1;
                            out2.value = data.cnum2;
                            cres_l.value = data.cres_l;
                            cres_r.value = data.cres_r;
                        })
                        .catch(error => {
                            console.error('Error:', error);
                            // display.value = 'Error';
                        });
            }
            /*
            } else {
                (num1.value === '') ? (out1.value = '') : "";
                (num2.value === '') ? (out2.value = '') : "";
            }
            */
        }
        document.addEventListener('keydown', function(event) {
            if (event.key === 'Enter') {
                event.preventDefault();
                calculate();
            }
        });
    </script>

    <script>
        const select1 = document.getElementById('op');
        const select2 = document.getElementById('op_mirror');

        select1.addEventListener('change', () => {
            select2.value = select1.value;
        });
    </script>
</body>
</html>
```

ops.asm

```
format ELF64
```

```
section '.data' writable
    ; str_to_base10f
    base rq 1
    dig rd 1
    dot_found rb 1
    is_negative rb 1

    ; base10f_to_str
    base2 rq 1
    is_negative2 rb 1
    DIGITS db "0123456789ABCDEF", 0
    buffer rb 128+1

    ; xmm float constants
    align 4
    XMM_FLOAT_ZERO dd 0.0
    XMM_FLOAT_ONE dd 1.0
    ; XMM_NEGATE_FLOAT_MASK dq?? 0xE80000000

section '.text' executable

    public str_to_base10f
    public base10f_to_str

    public asm_fadd
    public asm_fsub
    public asm_fmul
    public asm_fdiv

str_to_base10f:
    ; rdi -> char *str
    ; rsi -> float *num
    ; rdx -> int base
    enter 0,0

    mov rbx, rsi
    mov rsi, rdi
    mov [base], rdx
    ; rbx -> float *num
    ; rsi -> char *str
    ; [base] -> int base

    ;[ init vars
    mov [dot_found], 0
    mov [is_negative], 0
    xorps xmm0, xmm0; f
    movss xmm1, [XMM_FLOAT_ONE]; div
    ;]

.str_loop:
    lodsb; al -> char ch
    cmp al, 0
    jz .exit_str_loop
```

```
    call switch_char

    cmp [dig], -1
    jz .str_loop

    cvtsi2ss xmm2, dword [base]
    cvtsi2ss xmm3, dword [dig]

    cmp [dot_found], 1
    jz ._dot_found
    mulss xmm0, xmm2
    addss xmm0, xmm3
    jmp ._dot_found_end
    ._dot_found:
    mulss xmm1, xmm2
    divss xmm3, xmm1
    addss xmm0, xmm3
    ._dot_found_end:

    jmp .str_loop
.exit_str_loop:

    cmp [is_negative], 1
    jnz .end
    mov eax, 80000000h  ; Sign bit mask for 32-bit float
    movd xmm4, eax      ; Move mask to xmm4
    xorps xmm0, xmm4    ; Flip sign bit of xmm0

.end:
    movss [rbx], xmm0

    leave
    ret

base10f_to_str:
    ; xmm0 -> float num
    ; rdi -> char *str
    ; rsi -> int base

    mov [base2], rsi ; -> base
    mov rsi, 0       ; -> i

    comiss xmm0, [XMM_FLOAT_ZERO]
    jnz .is_zero_end
    mov [rdi+rsi], byte '0'
    inc rsi
    jmp .end
    .is_zero_end:

    cmp [base2], 2
    jl .end
    cmp [base2], 16
    jg .end
```

```
mov [is_negative2], 0
comiss xmm0, [XMM_FLOAT_ZERO]
jnb .float_neg_end
mov [is_negative2], 1
mov eax, 80000000h  ; Sign bit mask for 32-bit float
movd xmm1, eax      ; Move mask to xmm4
xorps xmm0, xmm1    ; Flip sign bit of xmm0
.float_neg_end:

xor rcx, rcx
cvttss2si ecx, xmm0 ; decimal
cvtsi2ss xmm1, ecx
subss xmm0, xmm1

; rdi -> char *str
; ecx -> int dec
; xmm0 -> float f
; [base2] -> int base
; rsi -> int i

lea rbx, [buffer]
; rbx -> char *buffer

cmp ecx, 0
jnz .buff_loop_begin
mov [rbx+rsi], byte '0'
inc rsi
jmp .buff_loop_end
.buff_loop_begin:
push rdi
lea rdi, [DIGITS]
.buff_loop:
mov eax, ecx
cdq
idiv [base2]
mov ecx, eax

mov al, [rdi+rdx]

mov [rbx+rsi], al
inc rsi

cmp ecx, 0
jg .buff_loop
pop rdi
.buff_loop_end:

cmp [is_negative2], 0
jz ._is_negative_end
mov [rbx+rsi], byte '-'
inc rsi
._is_negative_end:
```

```
    ;; Reverse the string
    mov rcx, 0 ; ecx -> int j
    mov rdx, rsi ; edx -> int k
    sub rdx, 1
    .reverse_loop:
    mov al, [rbx+rdx]
    mov [rdi+rcx], al
    inc rcx
    dec rdx
    cmp rcx, rsi
    jl .reverse_loop

    ;; Decimal part
    comiss xmm0, [XMM_FLOAT_ZERO]
    jna .decimal_part_end

    mov [rdi+rsi], byte '.'
    inc rsi

    mov rcx, 6 ; int precision
    lea rbx, [DIGITS]
    xor rdx, rdx; int digit

    .div_loop:
    cmp rcx, 0
    jng .div_loop_end
    comiss xmm0, [XMM_FLOAT_ZERO]
    jna .div_loop_end
    dec rcx

    cvtsi2ss xmm1, [base2]
    mulss xmm0, xmm1
    cvttss2si edx, xmm0
    mov al, [rbx+rdx]

    mov [rdi+rsi], al
    inc rsi

    cvtsi2ss xmm1, edx
    subss xmm0, xmm1

    jmp .div_loop

    .div_loop_end:
    .decimal_part_end:

.end:
    mov [rdi+rsi], byte 0
    ret


;;;;;;; ARITMETIC PROCEDURES
asm_fadd:
    addss xmm0, xmm1
```

```
    ret

asm_fsub:
    subss xmm0, xmm1
    ret

asm_fmul:
    mulss xmm0, xmm1
    ret

asm_fdiv:
    ; xmm0, xmm1
    xorps xmm2, xmm2; xmm2 = 0.0
    comiss xmm1, xmm2; es el divisor = 0
    jz .div_by_zero

    divss   xmm0, xmm1
    ret

    .div_by_zero:
    xorps   xmm0, xmm0
    ret

;;;;;;; HELPER PROCEDURES
switch_char:
    ; expects: al
    ; changes: [dig], [dot_found], [is_negative]
    cmp al, ' '
    jz .char_space
    cmp al, '0'
    jz .char_zero
    cmp al, '1'
    jz .char_one
    cmp al, '2'
    jz .char_two
    cmp al, '3'
    jz .char_three
    cmp al, '4'
    jz .char_four
    cmp al, '5'
    jz .char_five
    cmp al, '6'
    jz .char_six
    cmp al, '7'
    jz .char_seven
    cmp al, '8'
    jz .char_eight
    cmp al, '9'
    jz .char_nine
    cmp al, 'a'
    jz .char_ten
    cmp al, 'A'
    jz .char_ten
    cmp al, 'b'
```

```
    jz .char_eleven
    cmp al, 'B'
    jz .char_eleven
    cmp al, 'c'
    jz .char_twelve
    cmp al, 'C'
    jz .char_twelve
    cmp al, 'd'
    jz .char_thirteen
    cmp al, 'D'
    jz .char_thirteen
    cmp al, 'e'
    jz .char_fourteen
    cmp al, 'E'
    jz .char_fourteen
    cmp al, 'f'
    jz .char_fifteen
    cmp al, 'F'
    jz .char_fifteen
    cmp al, '-'
    jz .char_hiphen
    cmp al, '.'
    jz .char_dot
    jmp .switch_char_end

.char_space:
    mov [dig], -1
    jmp .switch_char_end
.char_zero:
    mov [dig], 0
    jmp .switch_char_end
.char_one:
    mov [dig], 1
    jmp .switch_char_end
.char_two:
    mov [dig], 2
    jmp .switch_char_end
.char_three:
    mov [dig], 3
    jmp .switch_char_end
.char_four:
    mov [dig], 4
    jmp .switch_char_end
.char_five:
    mov [dig], 5
    jmp .switch_char_end
.char_six:
    mov [dig], 6
    jmp .switch_char_end
.char_seven:
    mov [dig], 7
    jmp .switch_char_end
.char_eight:
    mov [dig], 8
```

```
    jmp .switch_char_end
.char_nine:
    mov [dig], 9
    jmp .switch_char_end
.char_ten:
    mov [dig], 10
    jmp .switch_char_end
.char_eleven:
    mov [dig], 11
    jmp .switch_char_end
.char_twelve:
    mov [dig], 12
    jmp .switch_char_end
.char_thirteen:
    mov [dig], 13
    jmp .switch_char_end
.char_fourteen:
    mov [dig], 14
    jmp .switch_char_end
.char_fifteen:
    mov [dig], 15
    jmp .switch_char_end
.char_hiphen:
    mov [dig], -1
    mov [is_negative], 1
    jmp .switch_char_end
.char_dot:
    mov [dig], -1
    mov [dot_found], 1
    jmp .switch_char_end

.switch_char_end:
    ret
```