

Instituto Tecnológico de Costa Rica

Tarea 1

GeometryTEC

Documentación

CE1106 – Paradigmas de Programación

Estudiantes:

José Bernardo Barquero Bonilla (2023150476)

Jose Eduardo Campos Salazar (2023135620)

Jimmy Feng Feng (2023060347)

Alexander Montero Vargas (2023166058)

Prof. Marco Rivera Meneses

II Semestre

27 de Agosto de 2024

## **Descripción de los algoritmos desarrollados**

A continuación se detallan los algoritmos considerados y desarrollados para la solución del problema planteado en la tarea. Estos se especifican en el manejo de los números decimales, debido a que el ensamblador 8086 no tiene soporte para este tipo de números.

### **Algoritmos Considerados**

#### ***Potencias de 10***

El algoritmo basado en potencias de 10 tenía como objetivo simplificar el manejo de números decimales al tratarlos como enteros. Esta estrategia se enfoca en leer el número ingresado por el usuario hasta encontrar un separador decimal (punto o coma). Una vez identificado, se contaban los decimales presentes en la entrada del usuario. Posteriormente, se multiplicaba el número completo por la potencia de 10 correspondiente al número en decimales, convirtiéndose así en un número entero puro.

A pesar de la simplicidad de este algoritmo, se presentaba el problema del alto consumo de registros. Esto debido a que el 8086 dispone de un número limitado de registros, lo cual significaba que el manejo de potencias de 10 requería dedicar varios de ellos a almacenar y manipular estas potencias, especialmente cuando se trabajaba con números grandes. Además de que la multiplicación y división de números enteros por potencias de 10 consumía una cantidad considerable de memoria y procesamiento.

Respecto al rendimiento de este algoritmo considerado, hubiese sido ineficiente debido a la necesidad de manejar múltiples registros simultáneamente y la complejidad de las operaciones matemáticas involucradas.

#### ***Generalización del número***

La idea de generalizar el número, proponía tratar el número decimal como un único entero de hasta 6 dígitos, evitando la división entre la parte entera y la parte decimal durante las operaciones. Bajo esta idea, las operaciones matemáticas se realizarían utilizando los registros de enteros disponibles en el 8086, lo que simplifica el proceso al reducir la cantidad de pasos y operaciones intermedias.

El principal problema surgía al momento de generar la respuesta final, donde era necesario dividir el resultado para separar nuevamente la parte entera de la decimal. La división en ensamblador es una operación que puede ser compleja y propensa a posibles errores, especialmente cuando se trabaja con números grandes o cuando se necesita un alto grado de precisión. En este caso, la dificultad para identificar y separar de forma correcta los dígitos decimales después de la división, generaba resultados incorrectos y, por ende, un algoritmo ineficaz.

Respecto al rendimiento, éste algoritmo es la solución más directa a este problema, no obstante presenta esta ineficacia y poco fiable en la práctica debido a los problemas inherentes en la división e interpretación del resultado.

## ***Buffer de texto***

Este algoritmo basado en un *buffer* de texto proponía convertir el *input* del usuario en una cadena de texto, que luego sería almacenada en la pila para ser procesada. Esta idea permitiría tratar los números ingresados como simples caracteres, facilitando la manipulación de la entrada sin preocuparse inicialmente de su valor numérico. El procesamiento posterior involucraría convertir estos caracteres en números y realizar las operaciones necesarias.

Aunque el uso de un *buffer* de texto parece una solución viable, se presentaron varios problemas en el contexto del ensamblador 8086. Primero, el manejo de múltiples entradas se complicaba significativamente debido a la necesidad de gestionar adecuadamente la pila, especialmente considerando que las operaciones con la pila se realizan en orden inverso (LIFO). Además, la falta de *buffers* numéricos para almacenar y manipular las respuestas limitaba la capacidad del algoritmo para manejar los cálculos complejos y operaciones sucesivas. Además de la gestión del orden de los datos en la pila también incrementaba la complejidad y el riesgo a errores.

Este enfoque, hubiese introducido una sobrecarga significativa en el manejo de la pila y la memoria, lo que podría haber ralentizado el rendimiento del programa y aumentado la complejidad del código.

## **Algoritmo Desarrollado**

### ***Separación del número***

El algoritmo desarrollado se basa en la utilización de un único *buffer* para manejar tanto la parte entera como la parte decimal de los números. La estrategia consiste en acomodar los dígitos de derecha a izquierda en el *buffer*, comenzando por el menos significativo, con el separador decimal colocado en el tercer byte. Este enfoque permite un control más eficiente y preciso sobre los números al evitar la necesidad de separar la parte entera de la decimal en diferentes buffers. Esto facilita operaciones matemáticas como la multiplicación, donde el manejo del acarreo es crítico, ya que todo el cálculo se realiza dentro de un mismo espacio de memoria.

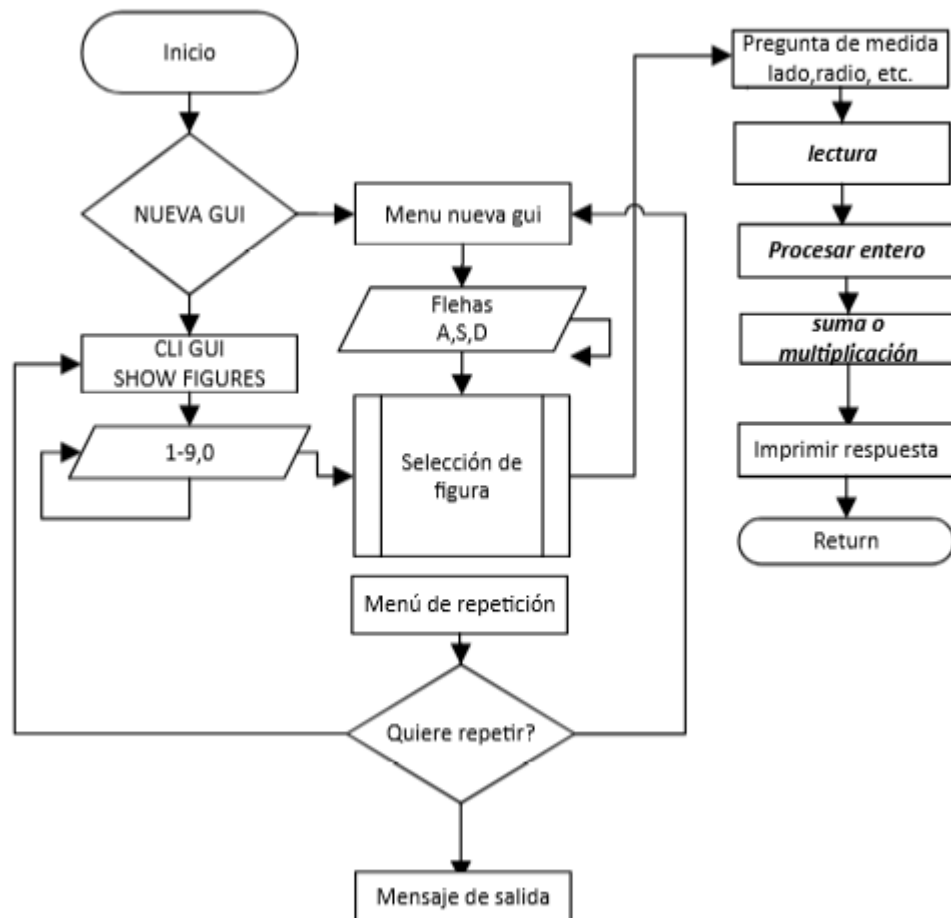
Esta alternativa fue implementada debido a su simplicidad y el mayor control que ofrecía sobre los números, en comparación con otras soluciones consideradas. Al utilizar un único *buffer*, el programador pudo manejar los números de manera más directa y evitar complicaciones asociadas con la coordinación de múltiples *buffers*. Además, el enfoque permitió optimizar el uso de los recursos limitados del 8086, como los registros y la memoria, ya que los buffers empleados eran de tamaño reducido (3 y 5 bytes). Este diseño también facilitó la implementación del algoritmo, ya que los números se podían manejar como una unidad coherente, simplificando el código y reduciendo el riesgo de errores.

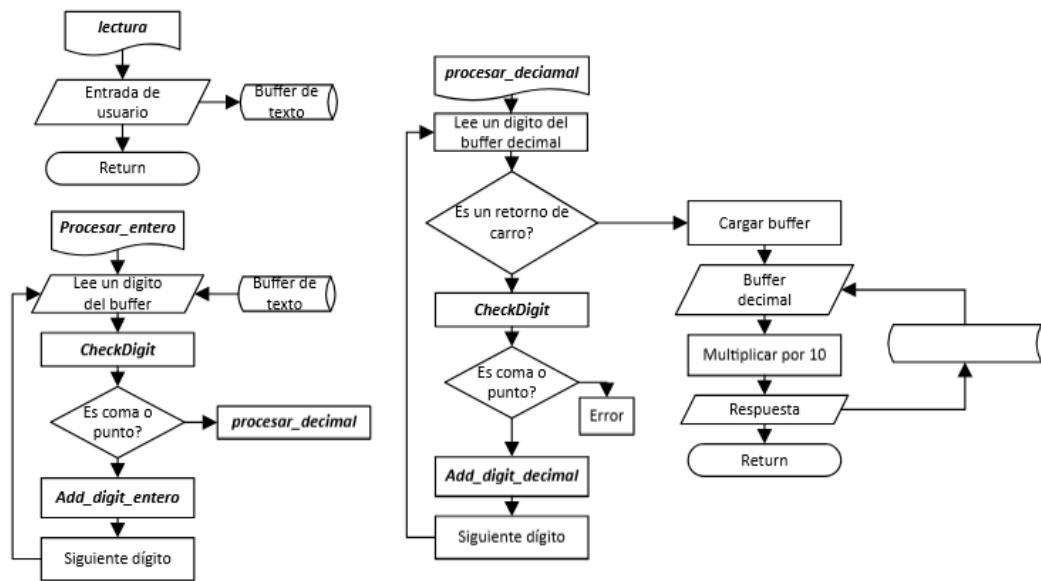
En términos de rendimiento, la utilización de un único *buffer* minimizó la carga sobre los registros y la memoria, lo cual es crucial en un entorno de recursos limitados. Además, la estructura del *buffer* permitió una manipulación directa de los números, lo que

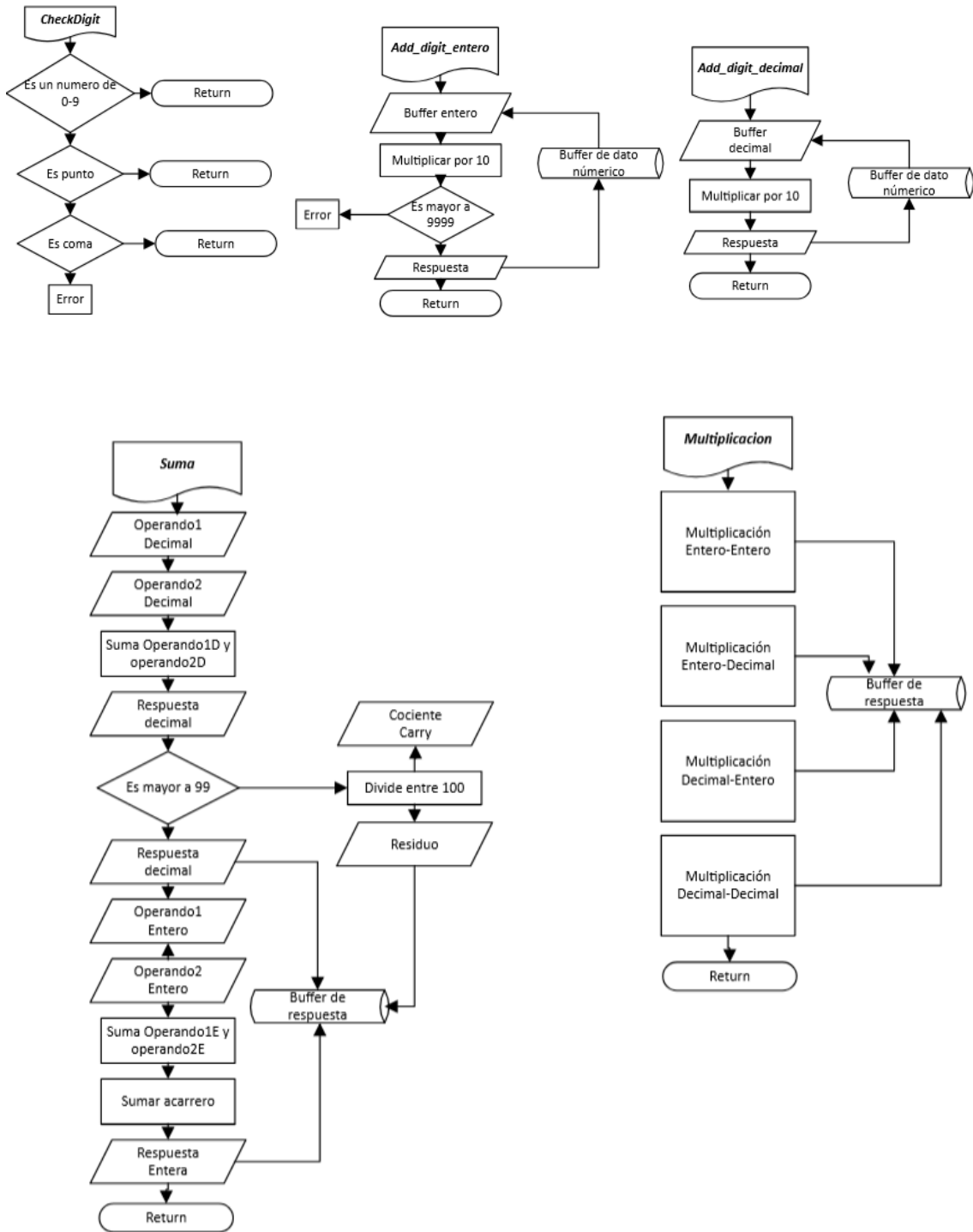
resultó en un procesamiento más rápido y menos propenso a errores durante las operaciones matemáticas. El control adicional que ofrecía este enfoque en la gestión de números y acarreo también contribuyó a mejorar la precisión y confiabilidad del programa.

### ***Diagrama***

A continuación se detalla un diagrama de flujo de la solución desarrollada para la solución del problema.







## **Descripción de funciones implementadas**

### **Suma**

La función suma se utiliza para el caso de que se deba hacer sumas de números grandes (2 bytes parte entera y 1 parte decimal). Toma dos buffers de 3 bytes de entrada y toma de salida uno de 3 bytes.

### **Multipliación**

Se creó una función de multiplicación para tener un mejor manejo con los números grandes que requieran más de dos 2 bytes de memoria, porque la entrada de la multiplicación es de 2 bytes más 1, debido a la parte decimal.

### **Cinco\_to\_tres**

La idea principal de esta función, es pasar un *buffer* de cinco bytes (que contiene una respuesta de tres bytes) a uno tres bytes, porque todos los métodos matemáticos aceptan solo tamaños de 3 bytes.

### **Lecture**

Esta función se emplea para la lectura de la entrada de texto del usuario, hasta que se presione *enter*.

### **Procesar\_entero**

La idea de esta función consiste en la lectura de la parte entera del número en el buffer de texto. Esta lectura, termina hasta que se tope con una “,” o un “.” para dar un salto a la parte decimal.

### **Check\_digit, check\_point, check\_comma**

Todas estas funciones permiten la verificación de que los datos ingresados corresponden a números, punto y coma, para evitar datos erróneos que contengan letras o caracteres no deseados.

### **Add\_digit\_entero**

Esta función, permite la creación del número entero, tomado el caracter leído en el registro DL y multiplica por 10 “dato\_n” para sumarle el siguiente dígito en DL.

### **Fin\_procesar\_entero**

Función que permite la detección de alguna coma o punto, para detener el análisis del dato y comenzar el análisis de la parte decimal.

### **Procesar\_decimal**

Como lo dice el nombre de la función, ésta permite el procesamiento de la parte decimal presente en el buffer hasta llegar a un salto de línea (*enter*).



### **Add\_digit\_decimal**

De igual forma que la parte entera, se realiza una multiplicación por 10 al “dato\_n” y se le suma el dígito almacenado en DL.

### **decimal\_deuno:**

Corrige el ingreso de decimales de solo 1 dígito, añadiendo el 0 como último decimal

### **Error**

Función creada con el fin de imprimir un mensaje de error en la terminal y llamar una función “repetir” para que la aplicación no se caiga.

### **Imprimir\_resp**

Esta función, permite procesar la respuesta obtenida después de todas las operaciones correspondientes. Hace uso de la pila, para almacenar el salto de línea, la parte decimal, coma o punto y la parte entera del número; para ser impresa de manera adecuada.

### **Imp\_msg**

Función utilizada para imprimir cualquier tipo de mensaje. La dirección del mensaje debe estar almacenada en el registro BX.

### **color\_msg:**

Función utilizada para imprimir cualquier tipo de mensaje a colores. La dirección del mensaje debe estar almacenada en el registro BX y el color en CL. El método ubica la posición de carro para imprimir una línea como solo de fondo y letra.

**new\_line:** Imprime un salto de línea.

### **Initial\_case**

Función que permite la impresión de un mensaje de bienvenida al usuario.

### **Case\_show\_figures**

Función que toma el *input* del usuario para las figuras (del 1 al 9) para saltar a la figura correspondiente

**Procesar\_rep:** procesa la entrada de las figuras del usuario. si no se cumple para ningún número, no cambie de pantalla

### **Reiniciar\_buffer**

Función creada para reiniciar los valores de los datos de cada *buffer* ingresado, dejando todos sus datos en 0.

### **Clear\_loop**

Función que también ayuda en la limpieza del *buffer*, realiza un *loop* para establecer en cero los bytes del mismo.

### **Repetir**

Esta función permite enviar un mensaje al usuario que desea repetir la selección de figuras o si desea salir del programa.

### **Done**

Imprime un mensaje de salida al usuario, y hace la interrupción del sistema para salirse del programa.

**print\_menu:** Imprime el nuevo menú de gui.

**input:** Maneja los inputs de movimiento y acción de la nueva interfaz.

## **Descripción de Estructuras de Datos Desarrolladas**

### **Buffers**

Los buffers se llenan de derecha a izquierda, como la lectura natural de los números. Los buffers se utilizaron para la lectura de los datos ingresados, además para el manejo de los números durante los cálculos, también se utilizaron para el control de la parte entera y decimal de los números. Se utilizaron buffers de 5 bytes y otros de 3 bytes.

### ***Ejemplos de caso de uso***

El buffer, es usado especialmente para el almacenamiento de los datos numéricos, tanto de la parte decimal, como de la parte entera. Estos, se leen para obtener su valor y realizar la operación correspondiente. Además, son utilizados en funciones de limpieza, impresión de mensajes, impresión de resultados y almacenamiento de datos temporales. Véase el apéndice B para una mejor visualización de los casos de uso.

## **Problemas encontrados**

### **Tamaño de número por *buffer* (de 2 hasta 4 bytes)**

En la realización de la tarea se presentó un problema con el tamaño de los números ingresados y los registros a utilizar, ya que algunos números ocupan más de un solo registro para ser almacenados.

### ***Solución***

Se realizó una partición del número entre varios registros, donde 4 registros corresponden a la parte entera y 1 registro para la parte decimal (suponiendo el número máximo 9999,99).

### **Manejo de decimales (principalmente operaciones)**

La arquitectura del ensamblador 8086 no tiene soporte para números flotantes, por lo tanto se presentó un gran problema con el manejo de los mismos, ya que al no tener este soporte, se debe realizar un manejo diferente de estos números de punto flotante.

### ***Solución***

Para la solución del problema, se decidió trabajar la parte decimal por aparte de la entera, teniendo en consideración de que la parte decimal debe ser menor a 99, porque si no, genera un acarreo que debe ser sumado en la parte entera. Para esto, se hace uso de un *buffer* por aparte, para el manejo de la parte decimal del número, que al final se adjuntará a la parte entera, tomando en consideración el acarreamiento.

### **Almacenamiento incorrecto de datos en los *buffers***

Algunos de los datos que se almacenan en los buffers, se guardaban al revés, lo cual provocaba un cálculo incorrecto tanto en el perímetro como el área. Esto se debe a la convención de almacenamiento llamada *endianess*, más conocido como *little-endian*, presente en el procesador 8086 (Patterson y Hennessy, 2005).

En el caso del *little-endian*, el byte menos significativo, se almacena primero, es decir, en la dirección de memoria más baja. Mientras que en el caso *big-endian*, en esta convención el byte más significado se almacena primero, en otras palabras, en la dirección de memoria más baja (Patterson y Hennessy, 2005).

### ***Solución***

Se realizó un ajuste manual, haciéndolo byte por byte y tomando en consideración la dirección del dato, para evitar este problema.

### **Limpieza incorrecta del *buffer***

A la hora de ejecución del código, se tenía un *buffer* que no se limpiaba después de realizar alguna operación y presentar el resultado en pantalla, lo cual generaba posteriormente, resultados erróneos en las demás operaciones realizadas, ya que el valor se acumulaba en el *buffer*.

### ***Solución***

Se realizó un ajuste en el código, para realizar la limpieza correcta de cada *buffer* y evitar este tipo de acumulación innecesaria de los datos.

## **Problemas conocidos sin solución**

### **Error porcentual en los resultados**

Debido a que se realiza una aproximación de los cálculos solicitados, los resultados obtenidos poseen un cierto porcentaje de error respecto al valor que debería dar. Esto se debe, a que se realiza una normalización al resultado después de cada operación, lo cual hace que se pierdan decimales significativos en los cálculos. Es por ello que se considera un

problema conocido sin solución, ya que un manejo completo de todos los decimales presentes en un resultado, resulta en algo muy costoso respecto a la memoria.

## Plan de Actividades

A continuación, se detallan todas las historias de usuario creadas para la realización de la tarea.

### Sprint 1

**Historia de Usuario #1:** Como usuario, quiero que al ejecutar la aplicación, se muestre un mensaje de bienvenida, para saber que la aplicación ha iniciado correctamente

+	7	User Story	Historia de Usuario #1: Como usuario, quiero que al eje...	New	3	Tarea 1 - CE1106\Sprint 1
	Task		Definir el mensaje de bienvenida	Closed		Tarea 1 - CE1106\Sprint 1
	Task		Mostrar el mensaje de bienvenida en pantalla	Closed		Tarea 1 - CE1106\Sprint 1

**Historia de Usuario #2:** Como usuario, quiero seleccionar la figura geométrica para la que deseo calcular el área y el perímetro, para realizar los cálculos específicos de la figura

5	User Story	Historia de Usuario #2: Como usuario, quiero selecciona...	New	5	Tarea 1 - CE1106\Sprint 1
	Task		Mostrar menú de figuras geometricas	Closed	Tarea 1 - CE1106\Sprint 1
	Task		Al presionar un numero correspondiente a una figura geo...	Closed	Tarea 1 - CE1106\Sprint 1

**Historia de Usuario #3:** Como usuario, quiero ingresar las medidas de la figura seleccionada, para que la aplicación pueda realizar los cálculos necesarios

3	User Story	Historia de Usuario #3: Como usuario, quiero ingresar l...	New	8	Tarea 1 - CE1106\Sprint 1
	Task		Imprimir un mensaje que me indique la medida que debo...	Closed	Tarea 1 - CE1106\Sprint 1
	Task		Leer la entrada del usuario y almacenarlo en un buffer de...	Closed	Tarea 1 - CE1106\Sprint 1
	Task		Convertir el buffer de texto en un buffer numérico para la...	Closed	Tarea 1 - CE1106\Sprint 1

**Historia de Usuario #4:** Como usuario, quiero que la aplicación me muestre el área y el perímetro calculado de la figura seleccionada, para saber los resultados de manera rápida y precisa.

1	User Story	Historia de Usuario #4: Como usuario, quiero que la apl...	New	8	Tarea 1 - CE1106\Sprint 1
Task	Metodo de multiplicación para calculos generales de las f...	Closed			Tarea 1 - CE1106\Sprint 1
Bug	Corrección de registros que se leen de manera incorrecta	Closed			Tarea 1 - CE1106\Sprint 1
Task	Manejo de números grandes	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para cuadrado	Active			Tarea 1 - CE1106\Sprint 1
Task	Método de cálculos para rectángulo	Active			Tarea 1 - CE1106\Sprint 1
Task	Método de cálculos para triángulo	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para rombo	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para pentágono	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para hexágono	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para círculo	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para trapecio	Active			Tarea 1 - CE1106\Sprint 1
Task	Metodo de cálculos para paralelogramo	Active			Tarea 1 - CE1106\Sprint 1

## Sprint 2

**Historia de Usuario #5:** Como usuario, quiero que la aplicación me pregunte si deseo continuar o salir después de cada cálculo, para decidir si realizar más cálculos o finalizar la sesión

8	User Story	Historia de Usuario #5: Como usuario, quiero que la apl...	New	3	Tarea 1 - CE1106\Sprint 2
Task	Mostrar mensaje de pregunta al usuario para saber si qui...	Closed			Tarea 1 - CE1106\Sprint 2
Task	Manejar la respuesta del usuario	Closed			Tarea 1 - CE1106\Sprint 2

**Historia de Usuario #6 (Opcional):** Como usuario, quiero que la interfaz gráfica sea fácil de entender y usar, para navegar de manera intuitiva entre las opciones del programa

6	User Story	Historia de Usuario #6 (Opcional): Como usuario, quier...	New	13	Tarea 1 - CE1106\Sprint 2
---	------------	---	-----	----	---------------------------

**Historia de Usuario #7:** Como usuario, quiero que se validen las entradas para evitar errores en la ejecución, para garantizar que la aplicación funcione sin fallos

4	User Story	Historia de Usuario #7: Como usuario, quiero que se val...	New	5	Tarea 1 - CE1106\Sprint 2
Task	Validación de comas y puntos	Closed			Tarea 1 - CE1106\Sprint 2
Task	Validación de caracteres numéricos	Closed			Tarea 1 - CE1106\Sprint 2
Task	Validación de no más de un punto o comma	Closed			Tarea 1 - CE1106\Sprint 2
Task	Validación de números menores que 10 000	Closed			Tarea 1 - CE1106\Sprint 2

## Conclusiones

El uso de *buffers* de 3 y 5 bytes permitió manejar de manera eficiente los números decimales y enteros, optimizando así los recursos limitados del entorno de ensamblador 8086.

La estrategia de separar la parte entera y la parte decimal en registros y buffers específicos resultó en una solución sencilla y eficaz para realizar los cálculos necesarios, cumpliendo con el objetivo principal de aproximación de los resultados.

La creación de funciones personalizadas para operaciones básicas como multiplicación y suma fue esencial para superar las limitaciones del lenguaje ensamblador, especialmente en el manejo de números grandes que se deben procesar por partes y no se pueden manejar de manera directa.

### **Recomendaciones**

Si se desea realizar operaciones con números de punto flotante, se puede utilizar el co-procesador 8087 para realizar este tipo de operaciones.


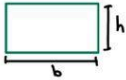
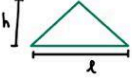
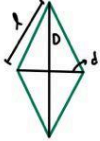

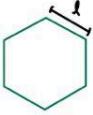

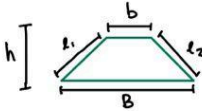
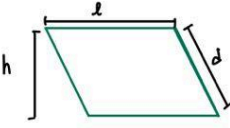
Se podría explorar la optimización del tamaño de los buffers o el uso de técnicas alternativas para mejorar aún más la eficiencia del manejo de números grandes y decimales.

Se puede considerar el uso de herramientas y/o técnicas avanzadas para el manejo de cálculos complejos en ensamblador, como la implementación de bibliotecas matemáticas optimizadas para el 8086.

## Bibliografía

- A GUEST. (13 de diciembre de 2019). *Untitled*. Pastebin. <https://pastebin.com/vadiukmk>
- Galeano, G., y Gómez, J. (1997). *8086 Interrupciones*. Grupo de Arquitectura de Computadores y Diseño Lógico, UEX.  
[http://ebadillo\\_computacion.tripod.com/ensamblador/8086\\_int.pdf](http://ebadillo_computacion.tripod.com/ensamblador/8086_int.pdf)
- gMendeZm. (2013). *[ITCR TEC] 224- [ASM x86] Colores (gmendezm)* [Video]. Youtube.  
<https://www.youtube.com/watch?v=ZS6dCR8LBA0>
- IBM. (2021). *ASCII, decimal, hexadecimal, octal, and binary conversion table*. IBM Documentation.  
<https://www.ibm.com/docs/es/aix/7.1?topic=adapters-ascii-decimal-hexadecimal-octal-binary-conversion-table>
- Learn Online. (2020). *LOOP INSTRUCTION | 8086 Instruction set with emu8086 | Looping example in 8086 assembly language* [Video]. Youtube.  
<https://www.youtube.com/watch?v=qb197N5jRxo>
- Patterson, D. y Hennessey, J. (2005). *Computer organization and design: The hardware/software interface* (3rd ed.). Morgan Kaufmann.  
<https://ia601209.us.archive.org/24/items/ComputerOrganizationAndDesign3rdEdition/-computer%20organization%20and%20design%203rd%20edition.pdf>
- Rcgldr. (6 de enero de 2017). *Assembler 8086 divide 32 bit number in 16 bit number*. StackOverflow.  
<https://stackoverflow.com/questions/41501491/assembler-8086-divide-32-bit-number-in-16-bit-number/41501934#41501934>
- Sep Roland. (20 de enero 2019). *Assembly 8086, LOOP instruction doesn't stop*. StackOverflow.  
<https://stackoverflow.com/questions/54268214/assembly-8086-loop-instruction-doesnt-stop/54281459#54281459>
- Tutorials Point. (s.f). *Assembly - Loops*. Tutorials Point.  
[https://www.tutorialspoint.com/assembly\\_programming/assembly\\_loops.htm](https://www.tutorialspoint.com/assembly_programming/assembly_loops.htm)

## Apéndice A. Fórmulas de Áreas y Perímetros Usados

	Perímetro	Área
	$P = 4 * l$	$A = l^2$
	$P = 2 * h + 2 * b$ $2(h + b)$	$A = b * h$
	$P = 3 * l$	$A = \frac{\sqrt{3} \cdot l \cdot l}{2}$
	<div>Verificar <math>\frac{d}{2} \leq l \leq \frac{D}{2}</math></div> $P = 4 \cdot l$	$A = \frac{D * d}{2}$
	$P = 5 * l$	<div><math>a = \frac{l}{2 \tan(36)} \rightarrow A = \frac{P \cdot a}{2}</math></div>
	$P = 6 * l$	<div><math>a = \frac{\sqrt{3}}{2} \cdot l \rightarrow A = \frac{P \cdot a}{2}</math></div>
	$P = 2\pi r$	$A = \pi \cdot r^2$
	$P = B + b + l_1 + l_2$ <div><math>0 &lt; h \leq \min(l_1, l_2)</math></div>	$A = \frac{B+b}{2} \cdot h$
	$P = 2d + 2l$ <div><math>0 &lt; h &lt; d</math></div>	$A = l \cdot h$



## Apéndice B. Casos de Uso de los *Buffers*

El buffer se llena de derecha a izquierda como la lectura natural de los números

Por lo que la manera correcta si por ejemplo se quiere guardar un 5 en un buffer de 2 bytes es:

0x0005

en lugar de

0x0500

En todos los buffer el último bytes se reserva para la parte decimal, así por ejemplo para almacenar el número pi en un buffer de 3 bytes queda de la forma

0x00030E

### ***multiplicacion:***

Entrada:

```
mov word [operando1], dir_buf_1 ;dirección del buffer 01
mov word [operando2], dir_buf_2 ;dirección del buffer 02
mov word [respuesta], dir_buf_respuesta ;dirección del buffer de
respuesta
call multiplicacion
```

### ***cinco\_to\_tres:***

Entrada y método de llamada

```
mov bx,[respuesta] ;dir de buffer de respuesta
call cinco_to_tres
```

recibe en bx la dirección del buffer de 5 bytes correspondiente a una respuesta de 3bytes almacenada en 5 bytes, ejemplo:

0x0000F452E2

Y lo convierte a un buffer de 3bytes

0xF452E2