Imagen que contiene Círculo

Descripción generada automáticamente

Universidad Tecnológica de Panamá

Centro Regional de Panamá Oeste

Facultad de ingeniería de Sistemas computacionales

Lic. en Ingeniería de Sistemas y Computación

Organización y Arquitectura de Computadoras

SEMESTRAL

Facilitador**:** Prof. Bolívar Quijada

Estudiantes:  
Joy Nelaton, 8-902-1282

Josué Pérez, 8-987-200

Julio Gómez, 8-956-1864

Grupo: 9IL-131

Fecha de entrega: 13/12/2023

**Introducción**

La resolución de ecuaciones cuadráticas es un problema fundamental en matemáticas y programación. A continuación, se presenta una implementación de un programa que hace uso tanto de C como ensamblador para efectuar el cálculo. El código en C proporciona una interfaz más legible y portable, mientras que el código en ensamblador se sumerge en operaciones a nivel de hardware para calcular las raíces de una ecuación cuadrática utilizando instrucciones de punto flotante.

1. **Código en C**

#include <stdio.h>

int quadratic(double, double, double, double \*, double \*);

int main() {

double a, b, c, root1, root2;

int res;

printf("\n - Programado por Joy Nelaton, Josue Perez, Julio Gomez - Lenguaje Ensamblador NASM Version 2.14.02 \n");

printf(" - Ecuacion Cuadratica en Assembler \n\n");

do {

printf("Ingresa el valor de a: ");

res = scanf("%lf", &a);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

do {

printf("Ingresa el valor de b: ");

res = scanf("%lf", &b);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

do {

printf("Ingresa el valor de c: ");

res = scanf("%lf", &c);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

if (quadratic(a, b, c, &root1, &root2)) {

printf("Las Raices son: x1 = %.10g, x2 = %.10g\n", root1, root2);

} else {

printf("Las Raices son imaginarias\n");

}

return 0;

}

1. **Análisis por sección del código en C**

#### 1**. Inclusión de Biblioteca, declaraciones de función y variables:**

#include <stdio.h>  
  
int quadratic(double, double, double, double \*, double \*);  
  
int main() {

printf("\n - Programado por Joy Nelaton, Josue Perez, Julio Gomez - Lenguaje Ensamblador NASM Version 2.14.02 \n");

printf(" - Ecuacion Cuadratica en Assembler \n\n");

}

* Se incluye la biblioteca estándar <stdio.h> para entrada/salida.
* Se declara la función quadratic() antes de su uso en main().

Se hace uso de cuatro argumentos en la función quadratic() para calcular las raíces de la ecuación cuadrática. La razón detrás de esto está relacionada con la necesidad de devolver más de un valor desde una función.

En el contexto de la ecuación cuadrática *ax2+bx+c=0*ax2+bx+c=0, se buscan dos raíces, lo que implica dos valores posibles para *x*x si la ecuación tiene solución real. Por lo tanto, para retornar ambos valores de raíz desde una función, se utilizan dos punteros (double \*) como argumentos adicionales.

* Los primeros tres argumentos (double, double, double) representan los coeficientes a, b, y c de la ecuación cuadrática.
* Los últimos dos argumentos (double \*, double \*) son punteros a variables en las cuales la función almacenará las raíces calculadas root1 y root2.

Esta elección permite que la función quadratic() realice los cálculos necesarios y guarde ambos resultados de raíz en las direcciones de memoria provistas por los punteros. Es una forma eficiente de devolver múltiples valores desde una función en C, aprovechando la capacidad de los punteros para modificar el contenido de variables fuera del ámbito de la función.

Las variables utilizadas en el código en C tienen roles específicos y se emplean para realizar operaciones y almacenar datos necesarios para resolver la ecuación cuadrática.

* **Variables para Coeficientes (a, b, c):** Estas variables almacenan los coeficientes de la ecuación cuadrática, que son fundamentales para realizar los cálculos necesarios.
* **Variables para Raíces (root1, root2):** Son utilizadas para almacenar las raíces calculadas por la función quadratic(). Dado que una ecuación cuadrática puede tener dos soluciones, se emplean dos variables separadas para almacenar las raíces resultantes.
* **Variable de Control (res):** Se utiliza para verificar si la entrada del usuario ha sido leída correctamente. Ayuda a asegurar que los valores ingresados para los coeficientes son números válidos antes de realizar los cálculos.

Estas variables son cruciales para el funcionamiento del programa, ya que almacenan los datos necesarios y los resultados obtenidos en el proceso de resolver la ecuación cuadrática. Permiten el flujo adecuado de datos entre el usuario, el programa y la función encargada de realizar los cálculos matemáticos.

#### **2. Interacción con el Usuario y Obtención de Coeficientes:**

do {

printf("Ingresa el valor de a: ");

res = scanf("%lf", &a);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

do {

printf("Ingresa el valor de b: ");

res = scanf("%lf", &b);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

do {

printf("Ingresa el valor de c: ");

res = scanf("%lf", &c);

while(getchar() != '\n'); // Limpia el buffer de entrada

if(res != 1) {

printf("Entrada invalida. Vuelve a intentarlo con un número valido.\n");

}

} while(res != 1);

Esta sección del código en C se encarga de interactuar con el usuario para obtener los coeficientes de la ecuación cuadrática *ax2+bx+c=0*ax2+bx+c=0.

* **do-while Loop:** Esta estructura de bucle permite solicitar al usuario el valor del coeficiente 'a'. El código dentro del bucle se ejecutará al menos una vez y se repetirá hasta que se obtenga un valor válido.
* **printf() y scanf():** printf() se utiliza para mostrar un mensaje que solicita al usuario ingresar el valor de 'a'. scanf() se emplea para leer el valor ingresado por el usuario y almacenarlo en la variable a. El valor leído se verifica mediante la variable res.
* **Limpieza del Buffer de Entrada:** La instrucción while(getchar() != '\n'); se utiliza para limpiar el buffer de entrada después de la lectura del valor a. Esto ayuda a eliminar cualquier carácter adicional (como el salto de línea) que pueda interferir con las lecturas posteriores.
* **Verificación de la Entrada:** Si scanf() no lee un valor válido (retorna diferente de 1), se muestra un mensaje de error indicando una entrada inválida. El bucle se repite hasta que se ingrese un valor válido.

Este segmento de código se repite tres veces, una para cada uno de los coeficientes a, b y c, permitiendo que el usuario ingrese valores para cada uno. Esta interacción garantiza que los coeficientes necesarios para resolver la ecuación cuadrática sean ingresados de manera válida y se almacenen en las variables correspondientes (a, b, c) para su posterior procesamiento.

**3. Manejo de Soluciones y Retorno de Valores:**

if (quadratic(a, b, c, &root1, &root2)) {

printf("Las Raices son: x1 = %.10g, x2 = %.10g\n", root1, root2);

} else {

printf("Las Raices son imaginarias\n");

}

return 0;

Esta sección del código en C se enfoca en manejar las soluciones obtenidas tras el cálculo de la ecuación cuadrática y en retornar los valores adecuados.

* **Llamada a la Función quadratic():** Se invoca la función quadratic() con los coeficientes a, b y c como argumentos y las direcciones de memoria de root1 y root2 como punteros para almacenar las raíces calculadas.
* **Evaluación del Valor Retornado:** Se utiliza una estructura condicional if-else para determinar si la función quadratic() ha retornado un valor verdadero (1) o falso (0).
* **Manejo de Soluciones:**
  + Si quadratic() devuelve verdadero, significa que se encontraron soluciones reales para la ecuación cuadrática. Se imprime el valor de las raíces (root1 y root2) utilizando printf() con un formato específico para mostrar los valores decimales con cierta precisión.
  + Si quadratic() devuelve falso, se imprime un mensaje indicando que las raíces son imaginarias.
* **Retorno desde la Función main():** Finalmente, el programa retorna 0 como indicador de finalización exitosa del programa.

1. **Código en ensamblador**

%define a qword [ebp+8]

%define b qword [ebp+16]

%define c qword [ebp+24]

%define root1 dword [ebp+32]

%define root2 dword [ebp+36]

%define disc qword [ebp-8]

%define one\_over\_2a qword [ebp-16]

section .data align=4

MinusFour dw -4

section .text

global \_quadratic

\_quadratic:

push ebp

mov ebp, esp

sub esp, 16 ; allocate 2 doubles (disc & one\_over\_2a)

push ebx ; must save original ebx

fild word [MinusFour]; stack -4

fld a ; stack: a, -4

fld c ; stack: c, a, -4

fmulp st1 ; stack: a\*c, -4

fmulp st1 ; stack: -4\*a\*c

fld b

fld b ; stack: b, b, -4\*a\*c

fmulp st1 ; stack: b\*b, -4\*a\*c

faddp st1 ; stack: b\*b - 4\*a\*c

ftst ; test with 0

fstsw ax

sahf

jb no\_real\_solutions ; if disc < 0, no real solutions

fsqrt ; stack: sqrt(b\*b - 4\*a\*c)

fstp disc ; store and pop stack

fld1 ; stack: 1.0

fld a ; stack: a, 1.0

fscale ; stack: a \* 2^(1.0) = 2\*a, 1

fdivp st1 ; stack: 1/(2\*a)

fst one\_over\_2a ; stack: 1/(2\*a)

fld b ; stack: b, 1/(2\*a)

fld disc ; stack: disc, b, 1/(2\*a)

fsubrp st1 ; stack: disc - b, 1/(2\*a)

fmulp st1 ; stack: (-b + disc)/(2\*a)

mov ebx, root1

fstp qword [ebx] ; store in \*root1

fld b ; stack: b

fld disc ; stack: disc, b

fchs ; stack: -disc, b

fsubrp st1 ; stack: -disc - b

fmul one\_over\_2a ; stack: (-b - disc)/(2\*a)

mov ebx, root2

fstp qword [ebx] ; store in \*root2

mov eax, 1 ; return value is 1

jmp short quit

no\_real\_solutions:

ffree st0 ; dump disc off stack

mov eax, 0 ; return value is 0

quit:

pop ebx

mov esp, ebp

pop ebp

Ret

1. **Análisis por sección del código en ensamblador**

### **Definiciones y Declaración del Procedimiento \_quadratic:**

* %define: Asigna nombres a posiciones de memoria en relación con ebp, permitiendo un acceso más sencillo a las variables locales.
* section .data align=4: Define una sección de datos con alineación de 4 bytes.
* MinusFour dw -4: Asigna el valor -4 a la variable MinusFour.

### **Cálculo del Discriminante y las Raíces de la Ecuación Cuadrática:**

* Se reserva espacio en la pila para variables locales (disc y one\_over\_2a) usando sub esp, 16.
* fild, fld, fmulp, entre otras instrucciones, realizan operaciones matemáticas con valores de punto flotante para calcular el discriminante.
* mov ebx, root1 y mov ebx, root2 mueven la dirección de memoria de root1 y root2 respectivamente a ebx. Esto se usa luego para almacenar las raíces calculadas.

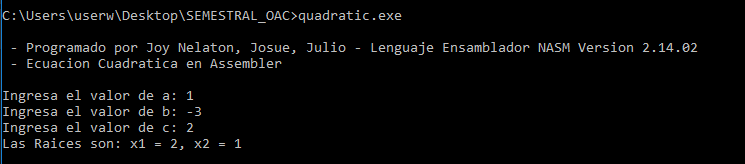
### **Manejo de Soluciones y Retorno de Valores:**

* ffree st0 libera el registro st0, utilizado en operaciones de punto flotante.
* mov eax, 0 y mov eax, 1 asignan valores a eax para indicar si existen soluciones reales (1) o si son imaginarias (0).

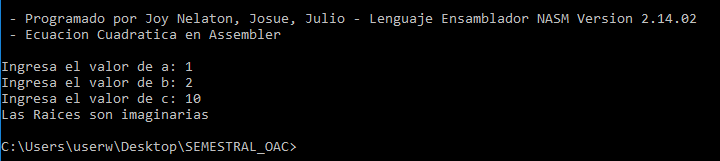
1. **Contraste entre la utilidad del código en C y el código en ensamblador**

El código en C se centra en la interacción con el usuario y la estructura general del programa, proporcionando una legibilidad y portabilidad mayores. A pesar de su legibilidad, puede ser menos eficiente en términos de rendimiento en comparación con el código en ensamblador, que está altamente optimizado y se ejecuta directamente en el nivel de la CPU. Mientras que el código en C utiliza funciones y estructuras de control de alto nivel para abstraer la lógica, el código en ensamblador opera directamente con la arquitectura de la máquina, aprovechando las capacidades específicas de la CPU para realizar operaciones matemáticas de manera eficiente.

1. **Prueba de ejecucion del programa**
2. **Prueba con valores para obtención de raíces reales (a = 1, b = -3, c = 2, Discriminante >=0)**



1. **Prueba con valores para obtención de raíces imaginarias (las mismas no se calculan, discriminante <0)**



**Conclusión**

La combinación de códigos en C y ensamblador para resolver la ecuación cuadrática demuestra la sinergia entre la comodidad de interacción del usuario brindada por C y la eficiencia en el procesamiento matemático ofrecida por el ensamblador. Mientras el código en C facilita la entrada de datos y proporciona una estructura más legible y portátil, el código en ensamblador está altamente optimizado para realizar los cálculos necesarios de manera eficiente y directa en el nivel de la CPU. Esta unión muestra cómo diferentes lenguajes pueden complementarse, capitalizando sus fortalezas individuales para abordar problemas específicos, asegurando una experiencia de usuario intuitiva junto con un rendimiento optimizado.