

# Actividad 6

**Christian Geovany Muñoz Rodríguez**

**Ingeniería en computación**

**Código: 221350605**

**Traductores de Lenguajes I – D04 (Martes y  
Jueves de 11 a 1)**

**Maestra: José Juan Meza Espinoza**

**Universidad de Guadalajara**

**Centro Universitario de Ciencias Exactas e  
Ingenierías**

**21 de Abril del 2023**



# Código:

```
#include <iostream>

using namespace std;

int suma( int $a, int $b ){

    int $result;

    __asm__ __volatile__(

        "movl %1, %%eax;"

        "movl %2, %%ebx;"

        "addl %%ebx,%%eax;"

        "movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ),
" " ( $b ));

    return $result ;

}

int resta( int $a, int $b ){

    int $result;

    __asm__ __volatile__(

        "movl %1, %%eax;"

        "movl %2, %%ebx;"

        "subl %%ebx,%%eax;"

        "movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ),
" " ( $b ));

    return $result ;

}
```

```
}
```

```
int gcd( int $a, int $b )
```

```
{
```

```
    int $result;
```

```
    __asm__ __volatile__(
```

```
        "movl %1, %%eax;"
```

```
        "movl %2, %%ebx;"
```

```
    "CONTD: cmpl $0, %%ebx;"
```

```
        "je DONE;"
```

```
        "xorl %%edx, %%edx;"
```

```
        "idivl %%ebx;"
```

```
        "movl %%ebx, %%eax;"
```

```
        "movl %%edx, %%ebx;"
```

```
        "jmp CONTD;"
```

```
    "DONE:    movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ),  
    "g" ( $b ));
```

```
    return $result ;
```

```
}
```

```
int producto(int a, int b, int c){
```

```
    int result;
```

```
    __asm__ __volatile__(
```

```
        "movl %1, %%eax;"
```

```

        "imull %2, %%eax;"
        "imull %3, %%eax;"
        "movl %%eax, %0;"
        : "=g" (result)
        : "g" (a), "g" (b), "g" (c)
        : "%eax"
    );
    return result;
}

```

```

int sub(int x, int y){
    return x - y;
}

```

```

int main(int argc, char** argv) {
    int a,b,c,d;
    cout<<"\nDigite el numero a: ";
    cin>>a;
    cout<<"\nDigite el numero b: ";
    cin>>b;
    cout<<"\nDigite el numero c: ";
}

```

```

    cin>>c;

    d=suma(a,b);

    cout<<"\nEl resultado de la suma de
"<<a<<"+"<<b<<"="<<d<<"\n";

    d=resta(a,b);

    cout<<"\nEl resultado de la resta de "<<a<<"-
"<<b<<"="<<d<<"\n";

    d=gcd(a,b);

    cout<<"\nEl resultado GCD("<<a<<","<<b<<")="<<d<<"\n";

    d=producto(a,b,c);

    cout<<"\nEl resultado de la multiplicacion de
"<<a<<"*"<<b<<"*"<<c<<"="<<d<<"\n";

```

```

asm("subl %%ebx, %%eax;"
    "movl %%eax, %%ecx;"
    : "=c" (c)
    : "a" (a), "b" (b)
    : /* lista clobber vacia */
);

```

```

printf("a = %d\nb = %d\nc = %d\n", a, b, c);

```

```

asm("addl %%ebx, %%eax;"
    "movl %%eax, %%ecx;"
    : "=c" (c)

```

```

        : "a"    (a), "b" (b)

        :                                /* lista clobber vacia */

    );

    printf("a = %d\nb = %d\nc = %d\n", a, b, c);

    /*La lista clobber es una lista de cadenas separadas por
    comas.

    Cada cadena es el nombre de un registro que el código
    ensamblador

    modifica potencialmente, pero para el cual el valor

    final no es importante. */

    system("pause");

    return 0;

}

```

## Desarrollo:

Este programa en C++ es un ejemplo de cómo utilizar instrucciones de lenguaje ensamblador en código C++ mediante el uso de la función **asm volatile**. La función asm permite integrar lenguaje ensamblador dentro de código en C++, lo que puede ser útil para realizar tareas específicas que no se pueden llevar a cabo de manera eficiente en C++. En este programa, se han implementado varias funciones en lenguaje ensamblador para realizar operaciones matemáticas básicas, como suma, resta, producto y cálculo del máximo común divisor (GCD).

En mi caso y según como marcaba la actividad, yo agregué la función de producto, la cual recibe 3 parámetros y retorna un solo resultado.

```

int suma( int $a, int $b ){
    int $result;
    __asm__ __volatile__(
        "movl %1, %%eax;"
        "movl %2, %%ebx;"
        "addl %%ebx, %%eax;"
        "movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ), "g" ( $b ));
    return $result ;
}

int resta( int $a, int $b ){
    int $result;
    __asm__ __volatile__(
        "movl %1, %%eax;"
        "movl %2, %%ebx;"
        "subl %%ebx, %%eax;"
        "movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ), "g" ( $b ));
    return $result ;
}

int gcd( int $a, int $b )
{
    int $result;
    __asm__ __volatile__(
        "movl %1, %%eax;"
        "movl %2, %%ebx;"
        "CONTD: cmpl $0, %%ebx;"
        "je DONE;"
        "xorl %%edx, %%edx;"
        "idivl %%ebx;"
        "movl %%ebx, %%eax;"
        "movl %%edx, %%ebx;"
        "jmp CONTD;"
        "DONE: movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ), "g" ( $b ));
    return $result ;
}

int producto(int a, int b, int c){
    int result;
    __asm__ __volatile__(
        "movl %1, %%eax;"
        "imull %2, %%eax;"
        "imull %3, %%eax;"
        "movl %%eax, %0;"
        : "=g" (result)
        : "g" (a), "g" (b), "g" (c)
        : "%eax"
    );
    return result;
}

int sub(int x, int y){
    return x - y;
}

```

La mayoría de estas funciones utilizan la instrucción de lenguaje ensamblador "mov" para mover valores entre registros y la instrucción "add" o "sub" para realizar la operación matemática deseada.

```

int suma( int $a, int $b ){
    int $result;

    __asm__ __volatile__(
        "movl %1, %%eax;"
        "movl %2, %%ebx;"
        "addl %%ebx, %%eax;"
        "movl %%eax, %0;" : "=g" ( $result ) : "g" ( $a ), "g" ( $b ));
    return $result ;
}

```

La función **volatile** se utiliza para indicar al compilador que no realice optimizaciones de código que puedan interferir con el código ensamblador. También se utiliza para indicar que el código ensamblador puede tener efectos secundarios, lo que significa que puede cambiar los valores de los registros de la CPU o de la memoria.

En la función principal del programa, se solicita al usuario que ingrese tres valores enteros, a, b y c. Luego, se realizan varias operaciones matemáticas utilizando las funciones previamente definidas.

```
int main(int argc, char** argv) {
    int a,b,c,d;
    cout<<"\nDigite el numero a: ";
    cin>>a;
    cout<<"\nDigite el numero b: ";
    cin>>b;
    cout<<"\nDigite el numero c: ";
    cin>>c;
    d=suma(a,b);
    cout<<"\nEl resultado de la suma de "<<a<<"+"<<b<<="<<d<<"\n";
    d=resta(a,b);
    cout<<"\nEl resultado de la resta de "<<a<<-"<<b<<="<<d<<"\n";
    d=gcd(a,b);
    cout<<"\nEl resultado GCD("<<a<<","<<b<<="<<d<<"\n";
    d=producto(a,b,c);
    cout<<"\nEl resultado de la multiplicacion de "<<a<<*"<<b<<*"<<c<<="<<d<<"\n";

    asm("subl %%ebx, %%eax;"
        "movl %%eax, %%ecx;"
        : "=c" (c)
        : "a" (a), "b" (b)
        : /* lista clobber vacia */
    );

    printf("a = %d\nb = %d\nc = %d\n", a, b, c);

    asm("addl %%ebx, %%eax;"
        "movl %%eax, %%ecx;"
        : "=c" (c)
        : "a" (a), "b" (b)
        : /* lista clobber vacia */
    );

    printf("a = %d\nb = %d\nc = %d\n", a, b, c);
    /*La lista clobber es una lista de cadenas separadas por comas.
    Cada cadena es el nombre de un registro que el código ensamblador
    modifica potencialmente, pero para el cual el valor
    final no es importante. */
    system("pause");

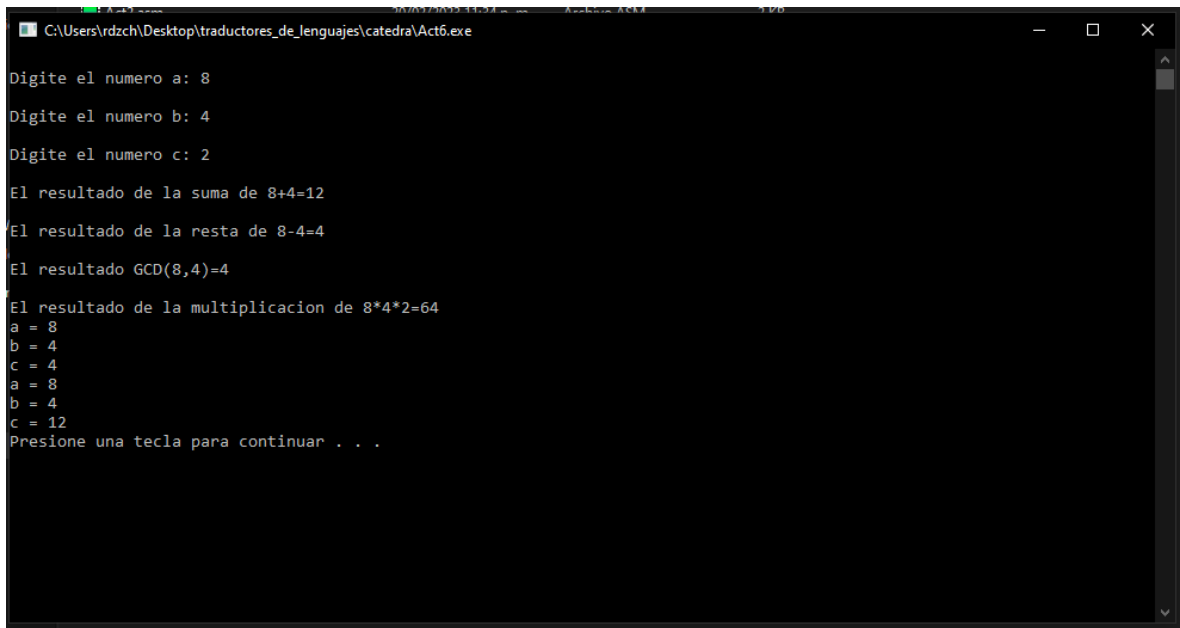
    return 0;
}
```

En resumen, el uso de la función `asm volatile` permite integrar código ensamblador en un programa C++ y realizar operaciones específicas de manera más eficiente.

Sin embargo, es importante tener en cuenta que el código ensamblador debe ser cuidadosamente diseñado para evitar efectos secundarios no deseados y se debe tener cuidado al trabajar con registros y memoria para evitar errores. Además, aunque el uso de lenguaje ensamblador en C++ puede ser útil en casos donde se requiere optimización del rendimiento o el uso de características de hardware específicas, también puede dificultar el mantenimiento y la portabilidad del código.

## Ejecución:





```
C:\Users\rdzch\Desktop\traductores_de_lenguajes\catedra\Act6.exe

Digite el numero a: 8
Digite el numero b: 4
Digite el numero c: 2
El resultado de la suma de 8+4=12
El resultado de la resta de 8-4=4
El resultado GCD(8,4)=4
El resultado de la multiplicacion de 8*4*2=64
a = 8
b = 4
c = 4
a = 8
b = 4
c = 12
Presione una tecla para continuar . . .
```

## Conclusión:

En conclusión, este programa es un ejemplo básico de cómo utilizar instrucciones de lenguaje ensamblador en C++ utilizando la sintaxis de GCC. Muestra cómo se pueden implementar funciones matemáticas básicas utilizando instrucciones ensambladoras y cómo se pueden integrar con el código C++. El uso de la función `asm volatile` puede ser beneficioso en algunos casos, pero su uso debe ser cuidadosamente considerado para evitar problemas de mantenimiento y portabilidad en el futuro.

## Bibliografía:

*Brey, B. B. (2006). Microprocesadores Intel : 8086/8088, 80186/80188, 80286, 80386 y 80486, Pentium, procesador Pentium Pro, Pentium II, Pentium III y Pentium 4: arquitectura, programación e interfaces. Pearson Educación.*