ЛАБОРАТОРНА РОБОТА 3.

РОБОТА ЗІ СТЕКОМ

**1.1 Мета роботи**

− Ознайомитися з основами роботи зі стеком у мові програмування Асемблер

− Проаналізувати використання стеку

**1.2 Завдання на роботу**

Для виконання лабораторної роботи необхідно дослідити як виглядає асемблерний код одразу після компіляції програми на С.

**1.3 Хід виконання**

Для дослідження коду, використаємо інструмент Compiler Explorer, який демонструє компіляцію та асемблерний код для програми (рис. 1), яка обчислює числа Фібоначчі рекурсивно. Ліва частина містить вихідний код C, а права — скомпільований асемблерний код (x86-64 GCC).

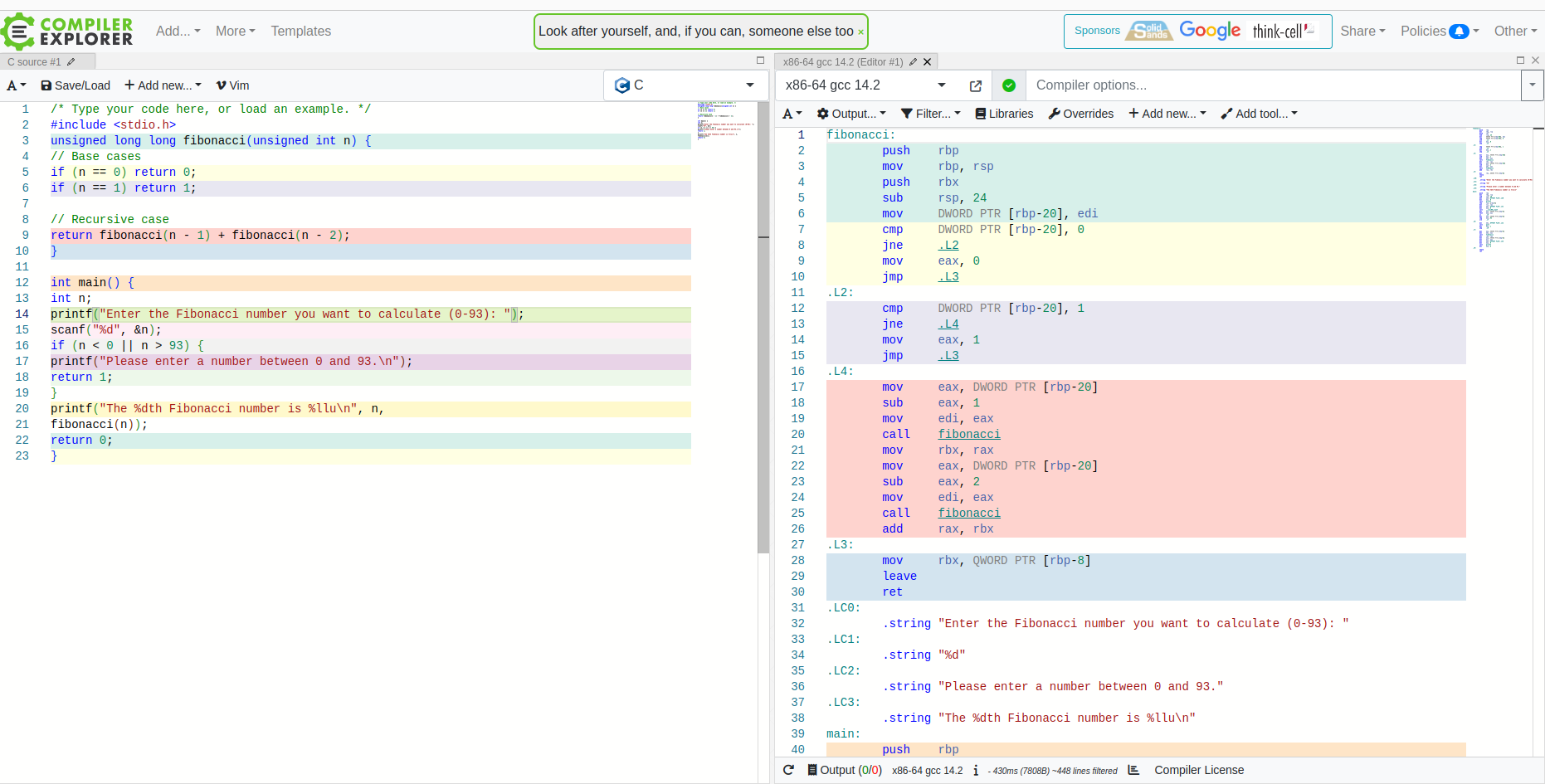
****

Рис. 1 **–** Аналіз коду за допомогою Compiler Explorer

Лістинг 1 - асемблерний код, задачі про числа Фібоначчі

|  |
| --- |
| fibonacci:  push rbp ; Зберегти базовий указник  mov rbp, rsp ; Встановити новий базовий указник  sub rsp, 24 ; Виділити пам'ять для локальних змінних  mov DWORD PTR [rbp-20], edi ; Зберегти n у стеку  cmp DWORD PTR [rbp-20], 0 ; Якщо n == 0  jne .L2  mov eax, 0 ; Результат = 0  jmp .L3  .L2:  cmp DWORD PTR [rbp-20], 1 ; Якщо n == 1  jne .L4  mov eax, 1 ; Результат = 1  jmp .L3  .L4:  sub eax, 1 ; Рекурсивний виклик fibonacci(n-1)  mov edi, eax  call fibonacci  mov rbx, rax ; Зберегти результат fibonacci(n-1)  sub eax, 2 ; Рекурсивний виклик fibonacci(n-2)  mov edi, eax  call fibonacci  add rax, rbx ; Сума результатів fibonacci(n-1) + fibonacci(n-2)  .L3:  leave ; Завершення функції  ret  main:  push rbp ; Зберегти базовий указник  mov rbp, rsp ; Встановити новий базовий указник  sub rsp, 16 ; Виділити пам'ять для локальних змінних  call printf ; Вивести запрошення  call \_\_isoc99\_scanf ; Отримати n від користувача  cmp DWORD PTR [rbp-4], 0 ; Перевірити, чи n < 0  js .L6  cmp eax, 93 ; Перевірити, чи n > 93  jle .L7  .L6:  call puts ; Вивести повідомлення про помилку  mov eax, 1 ; Завершити програму з кодом 1  jmp .L9  .L7:  call fibonacci ; Обчислити fibonacci(n)  call printf ; Вивести результат  .L9:  leave ; Завершення програми  ret |

### Пояснення:

1. Робота з RBP (base pointer):
   * RBP використовується для зберігання базового указника на стек. Це дозволяє легко звертатися до змінних у стеку через фіксовані зміщення.
   * Вхідні параметри (n) копіюються у стек (DWORD PTR [rbp-20]).
2. Рекурсія:
   * Кожен виклик fibonacci створює новий стековий кадр для збереження параметра та локальних змінних.
   * Результати рекурсивних викликів зберігаються в регістрах (наприклад, rbx для fibonacci(n-1)).
3. Використання регістрів:
   * EAX/RAX: Застосовується для повернення результату функції.
   * EDI: Передає вхідний параметр n у функцію fibonacci.
   * RBX: Зберігає проміжне значення fibonacci(n-1) між викликами.
4. Управління стеком:
   * sub rsp, 24: Виділення пам'яті для локальних змінних у стеку.
   * leave та ret: Завершують виконання функції, відновлюючи базовий указник та повертаючись до викликаної функції.

Висновки: У ході виконання лабораторної роботи, ознайомилась з основами роботи зі стеком у мові програмування Асемблер.