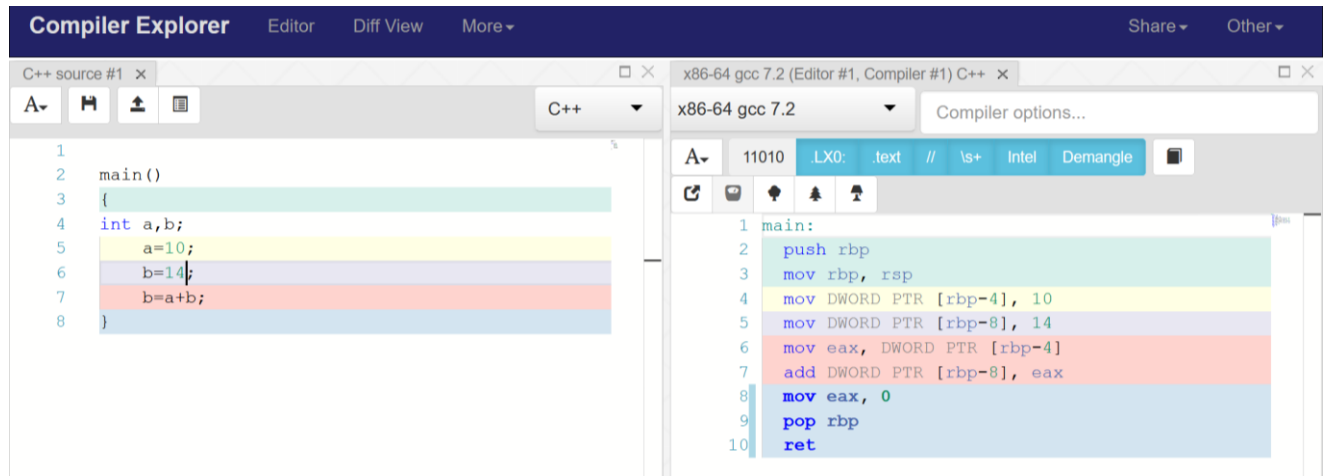


Compiler Explorer C to ASM

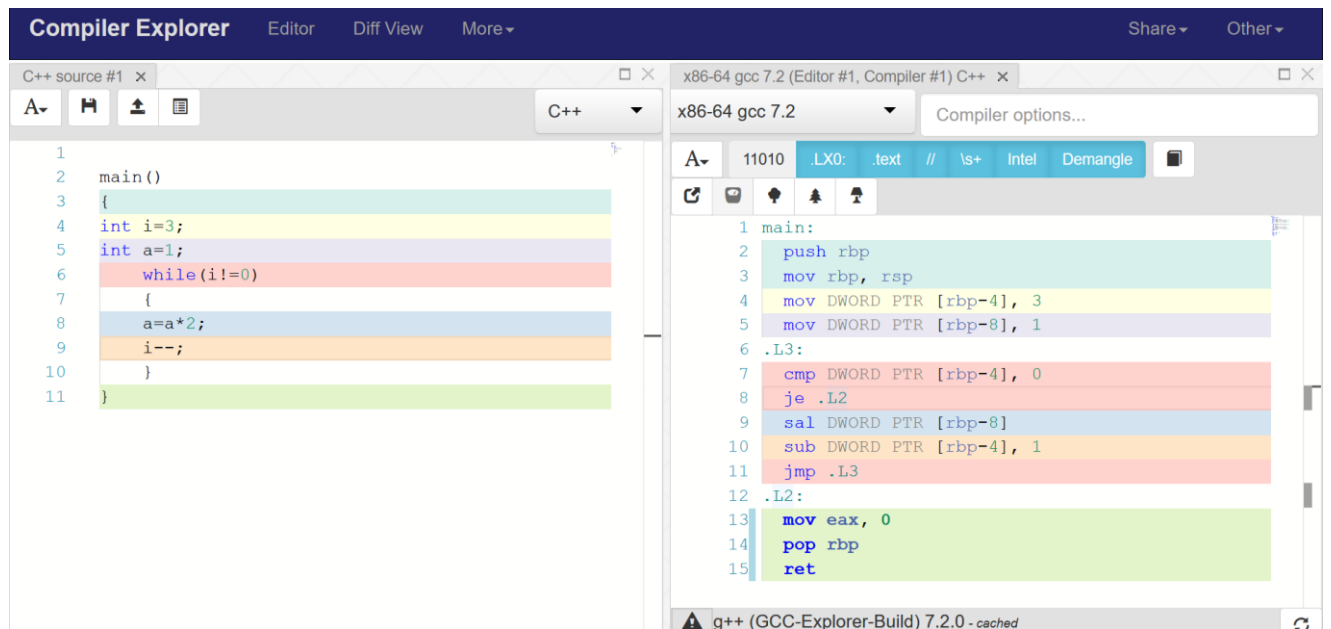
<https://godbolt.org/>



The screenshot shows the Compiler Explorer interface. On the left, the C++ source code is displayed in a file named 'C++ source #1'. It contains a simple program with a `main()` function that declares two integers `a` and `b`, initializes `a` to 10 and `b` to 14, and then prints the sum `a+b`. On the right, the x86-64 assembly output for GCC 7.2 is shown. The assembly code starts with `main:`, pushes `rbp`, moves `rsp` to `rbp`, and then uses `mov` instructions to store 10 and 14 at memory locations `[rbp-4]` and `[rbp-8]` respectively. It then loads these values into `eax`, adds them, and prints the result using `printf` (though the assembly shown is a simplified representation of the actual output).

```
1 main()
2 {
3     int a,b;
4     a=10;
5     b=14;
6     b=a+b;
7 }
```

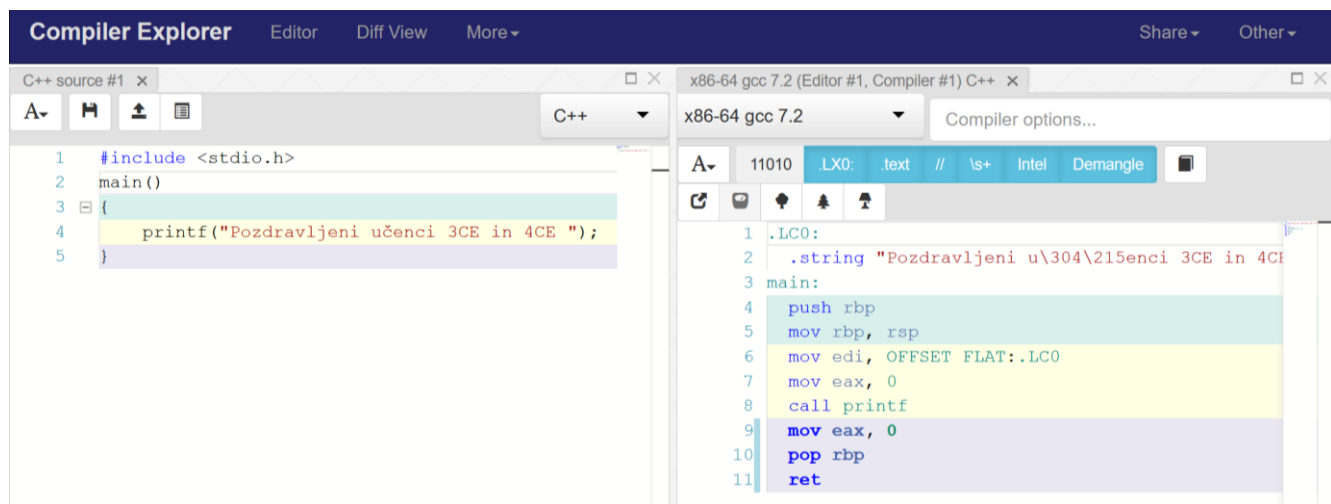
```
1 main:
2     push rbp
3     mov rbp, rsp
4     mov DWORD PTR [rbp-4], 10
5     mov DWORD PTR [rbp-8], 14
6     mov eax, DWORD PTR [rbp-4]
7     add DWORD PTR [rbp-8], eax
8     mov eax, 0
9     pop rbp
10    ret
```



This screenshot shows a more complex C++ program with a `while` loop. The source code on the left declares `i` as 3 and `a` as 1, then enters a `while` loop that doubles `a` and decrements `i` until `i` reaches 0. The assembly output on the right shows the corresponding x86-64 instructions. It includes a `while` loop implemented with `cmp`, `je`, `sal`, `sub`, and `jmp` instructions. The assembly also shows the final `mov` and `ret` instructions.

```
1 main()
2 {
3     int i=3;
4     int a=1;
5     while(i!=0)
6     {
7         a=a*2;
8         i--;
9     }
10 }
```

```
1 main:
2     push rbp
3     mov rbp, rsp
4     mov DWORD PTR [rbp-4], 3
5     mov DWORD PTR [rbp-8], 1
6     .L3:
7     cmp DWORD PTR [rbp-4], 0
8     je .L2
9     sal DWORD PTR [rbp-8]
10    sub DWORD PTR [rbp-4], 1
11    jmp .L3
12 .L2:
13    mov eax, 0
14    pop rbp
15    ret
```



The final screenshot shows a C++ program that includes `<stdio.h>` and uses `printf` to print a string. The source code on the left is straightforward, including the header and the `printf` call. The assembly output on the right shows the x86-64 instructions for this program. It starts with `.LC0:` defining the string constant, followed by `main:` which pushes `rbp`, moves `rsp` to `rbp`, and then uses `mov` and `call` instructions to print the string and return.

```
1 #include <stdio.h>
2 main()
3 {
4     printf("Pozdravljeni ućenci 3CE in 4CE ");
5 }
```

```
1 .LC0:
2     .string "Pozdravljeni u\304\215enci 3CE in 4CE"
3 main:
4     push rbp
5     mov rbp, rsp
6     mov edi, OFFSET FLAT:.LC0
7     mov eax, 0
8     call printf
9     mov eax, 0
10    pop rbp
11    ret
```