# Godbolt
# Compiler Explorer

December 2024

# Overview

- What is Compiler Explorer?
- Features
- Other Tools
- Summary

# What is **Compiler Explorer**?

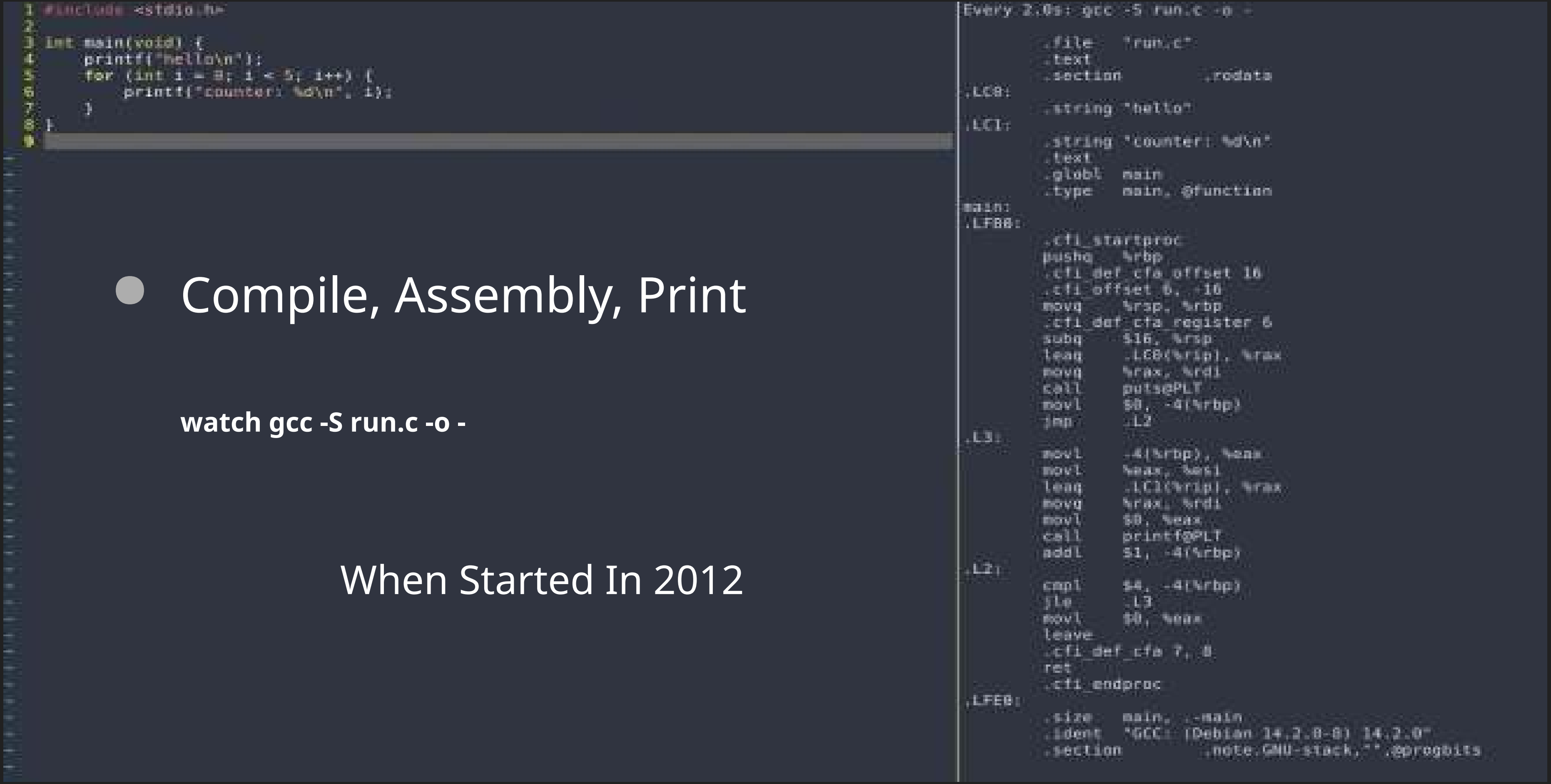
# What is Compiler Explorer?

github.com/mattgodbolt

- Created by Matt Godbolt
  - Former Google developer
  - Worked in the video game and Quant Trading industry
  - Unix Geek

# IDEA?

- Real time C/C++ Disassembly
    - Start in 2012
    - Just for C and C++ code
    - Simple Terminal commands
    - Only GCC on X86 machine
    - Grew in years



```c
#include <stdio.h>

int main(void) {
    printf("hello\n");
    for (int i = 0; i < 5; i++) {
        printf("counter: %d\n", i);
    }
}
```

```
Every 2.0s: gcc -S run.c -o -

        .file   "run.c"
        .text
        .section        .rodata
.LC0:
        .string "hello"
.LC1:
        .string "counter: %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, -4(%rbp)
        jmp     .L2
.L3:
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC1(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        addl    $1, -4(%rbp)
.L2:
        cmpl    $4, -4(%rbp)
        jle     .L3
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 14.2.0-8) 14.2.0"
        .section        .note.GNU-stack,"",@progbits
```

- Compile, Assembly, Print

**watch gcc -S run.c -o -**

When Started In 2012

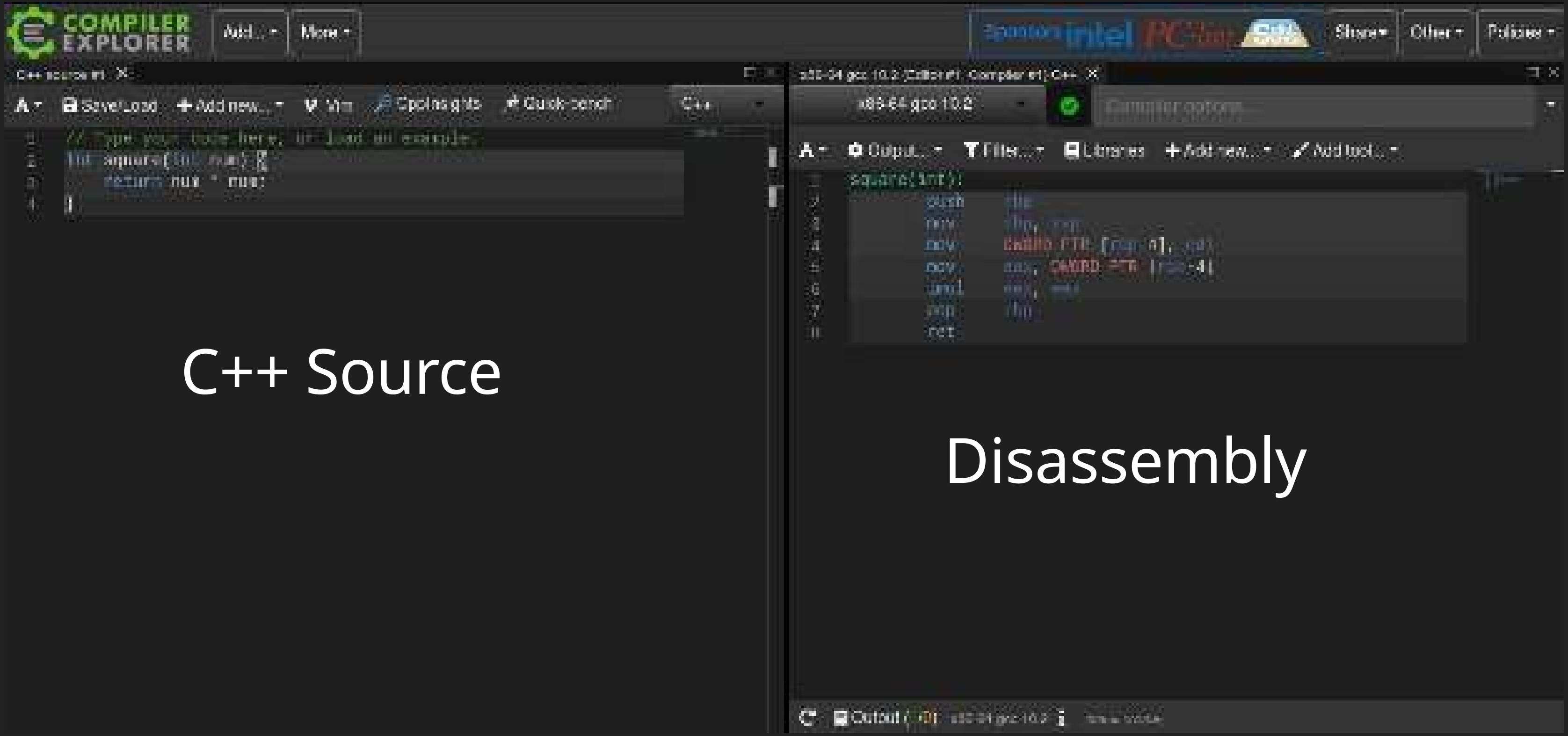
# New Version

COMPILER EXPLORER

- Web Application
  - Godbolt.org
  - Many Features
  - Extensibility
  - Compiler Testing Platform
  - Education Playground

Now  In 2024

# godbolt.org



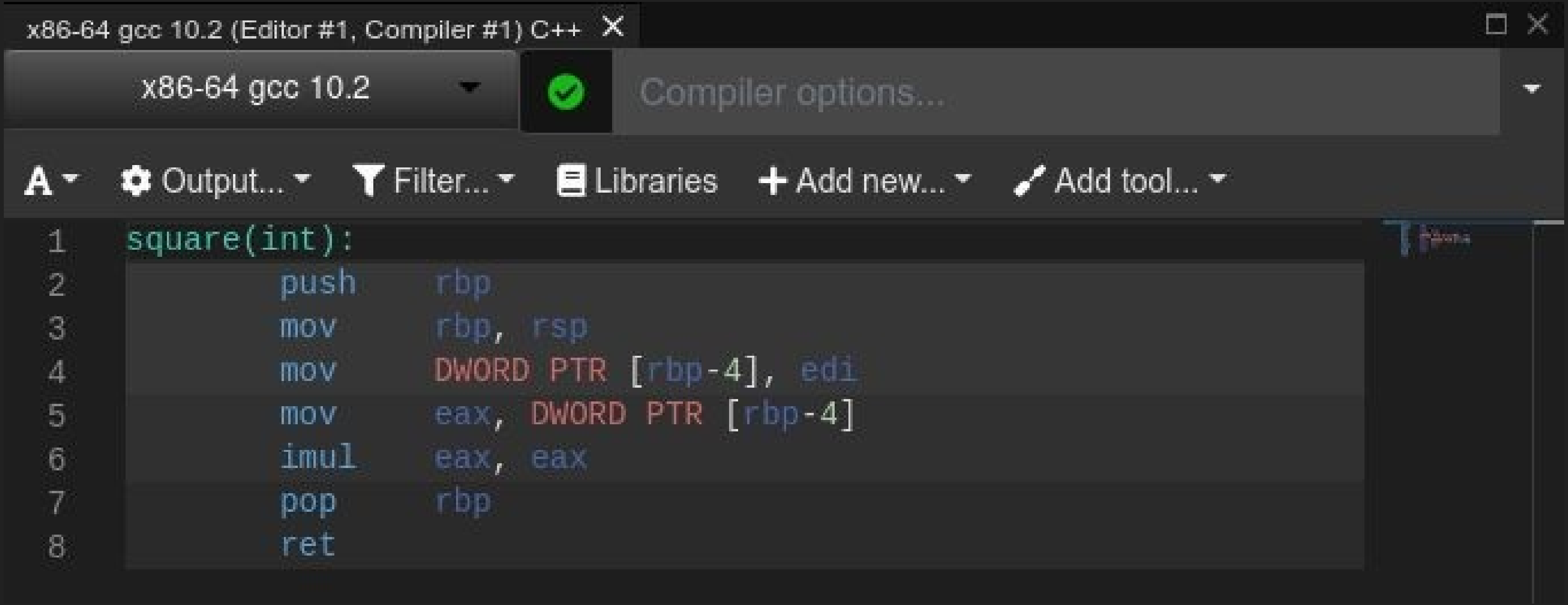
```
// Type your code here, or load an example.
int square(int num) {
    return num * num;
}
```

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

C++ Source

Disassembly

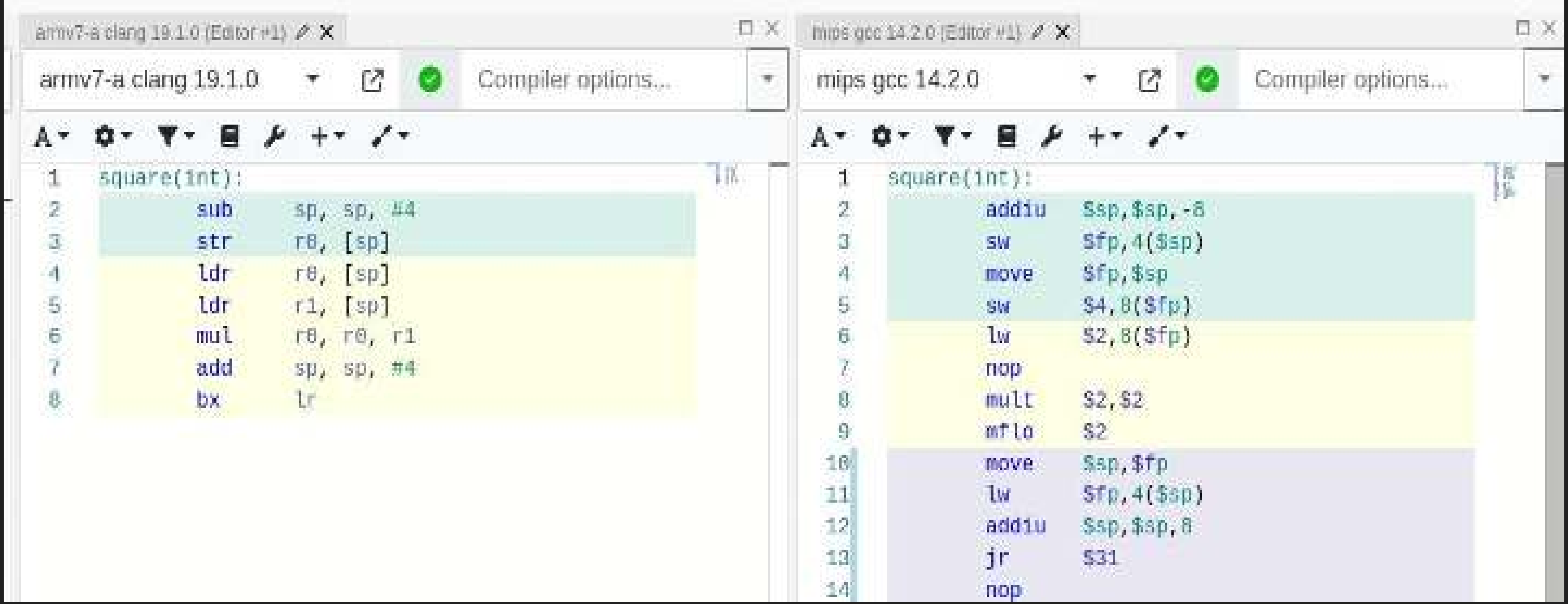
# Right side: Disassembly



```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

X86-64 gcc 10.2 compiler

# Right side: Disassembly



armv7-a clang 19.1.0 (Editor #1)

armv7-a clang 19.1.0

```
square(int):
        sub     sp, sp, #4
        str     r0, [sp]
        ldr     r0, [sp]
        ldr     r1, [sp]
        mul     r0, r0, r1
        add     sp, sp, #4
        bx      lr
```

mips gcc 14.2.0 (Editor #1)

mips gcc 14.2.0

```
square(int):
        addiu   $sp,$sp,-8
        sw      $fp,4($sp)
        move    $fp,$sp
        sw      $4,8($fp)
        lw      $2,8($fp)
        nop
        mult    $2,$2
        mflo    $2
        move    $sp,$fp
        lw      $fp,4($sp)
        addiu   $sp,$sp,8
        jr      $31
        nop
```

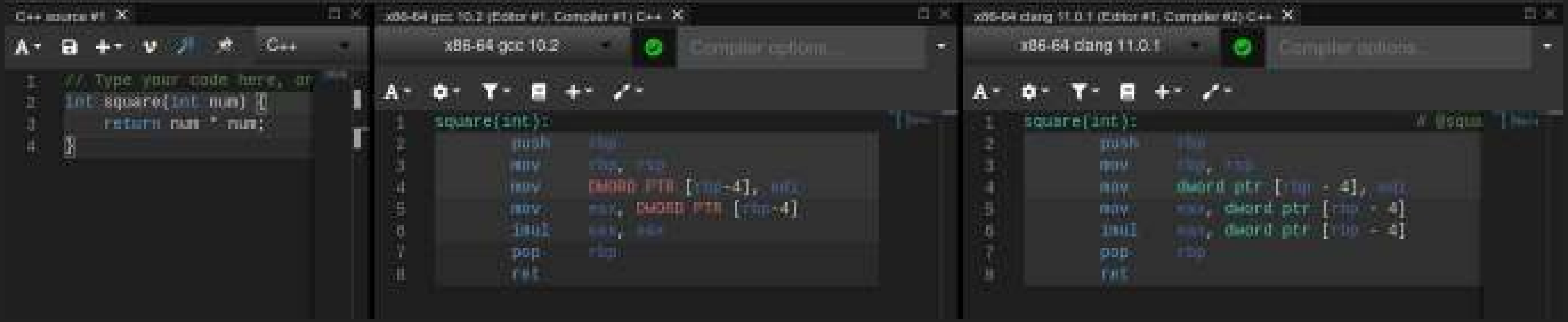
arm7 and MIPS

# Customization

source

gcc

clang



C++ source #1
C++
x86-64 gcc 10.2 (Editor #1, Compiler #1) C++
x86-64 gcc 10.2
Compiler options...
x86-64 clang 11.0.1 (Editor #1, Compiler #2) C++
x86-64 clang 11.0.1
Compiler options...

```cpp
// Type your code here, or
int square(int num) {
    return num * num;
}
```

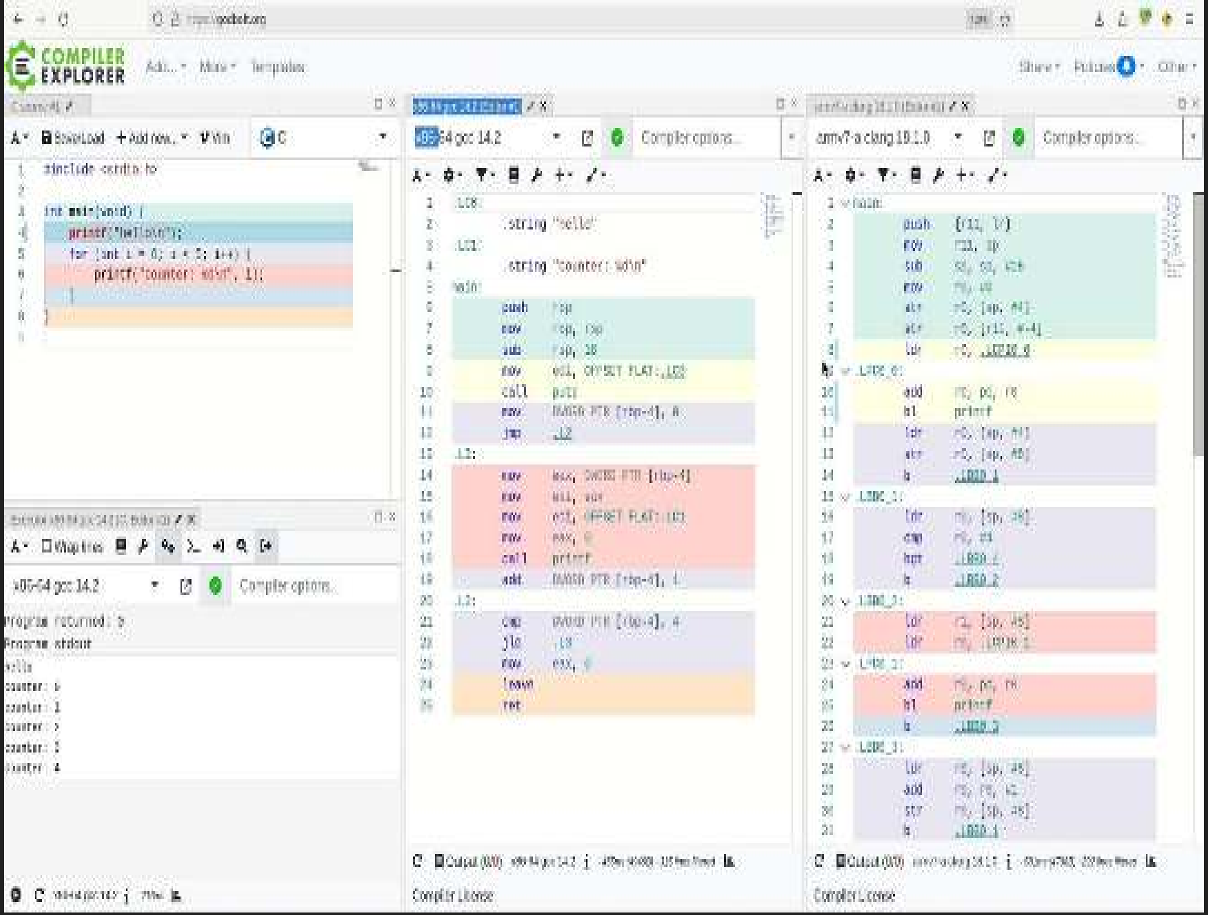
```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     dword ptr [rbp - 4], edi
        mov     eax, dword ptr [rbp - 4]
        imul    eax, dword ptr [rbp - 4]
        pop     rbp
        ret
```

Editor, Compiler

# Workbench

- Editor
- Execution
- Compiler
- Tool chain



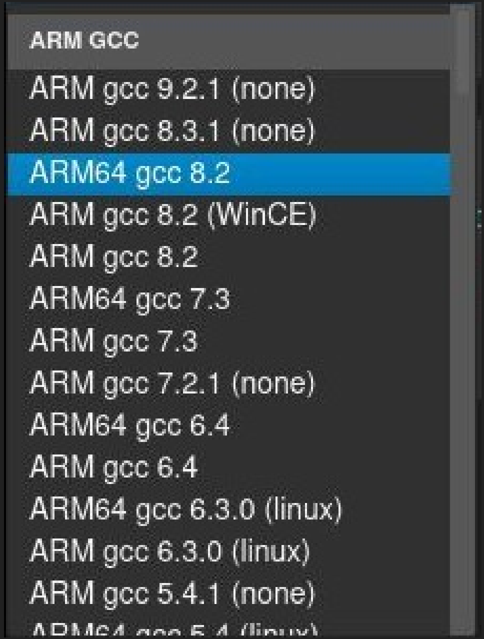
```c
#include <stdio.h>

int main(void) {
    printf("hello\n");
    for (int i = 0; i < 5; i++) {
        printf("counter: %d\n", i);
    }
}
```

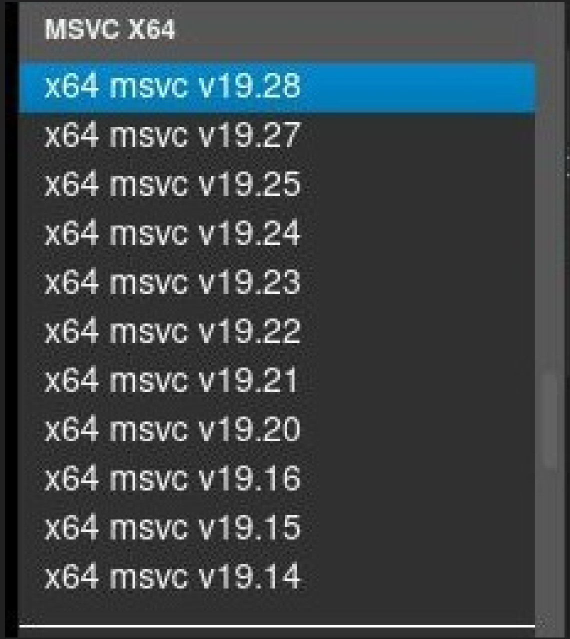
```
Program returned: 0
Program stdout
hello
counter: 0
counter: 1
counter: 2
counter: 3
counter: 4
```

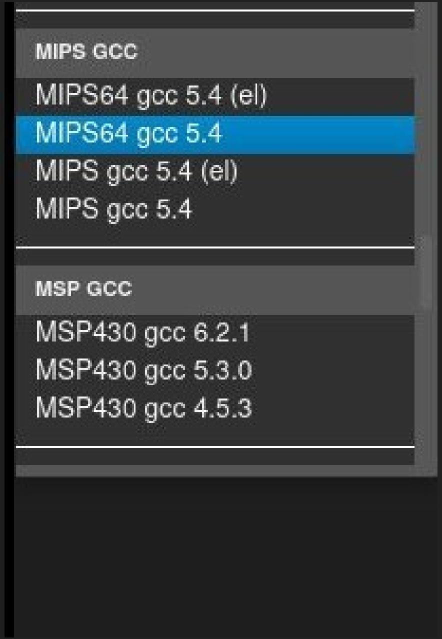
```
.LC0:
        .string "hello"
.LC1:
        .string "counter: %d\n"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR [rbp-4], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 4
        jle     .L3
        mov     eax, 0
        leave
        ret
```

```
main:
        push    {r11, lr}
        mov     r11, sp
        sub     sp, sp, #16
        mov     r0, #0
        str     r0, [sp, #4]
        str     r0, [r11, #-4]
        ldr     r0, .LCPI0_0
.LPC0_0:
        add     r0, pc, r0
        bl      printf
        ldr     r0, [sp, #4]
        str     r0, [sp, #8]
        b       .LBB0_1
.LBB0_1:
        ldr     r0, [sp, #8]
        cmp     r0, #4
        bgt     .LBB0_4
        b       .LBB0_2
.LBB0_2:
        ldr     r1, [sp, #8]
        ldr     r0, .LCPI0_1
.LPC0_1:
        add     r0, pc, r0
        bl      printf
        b       .LBB0_3
.LBB0_3:
        ldr     r0, [sp, #8]
        add     r0, r0, #1
        str     r0, [sp, #8]
        b       .LBB0_1
```

# Lots of options...



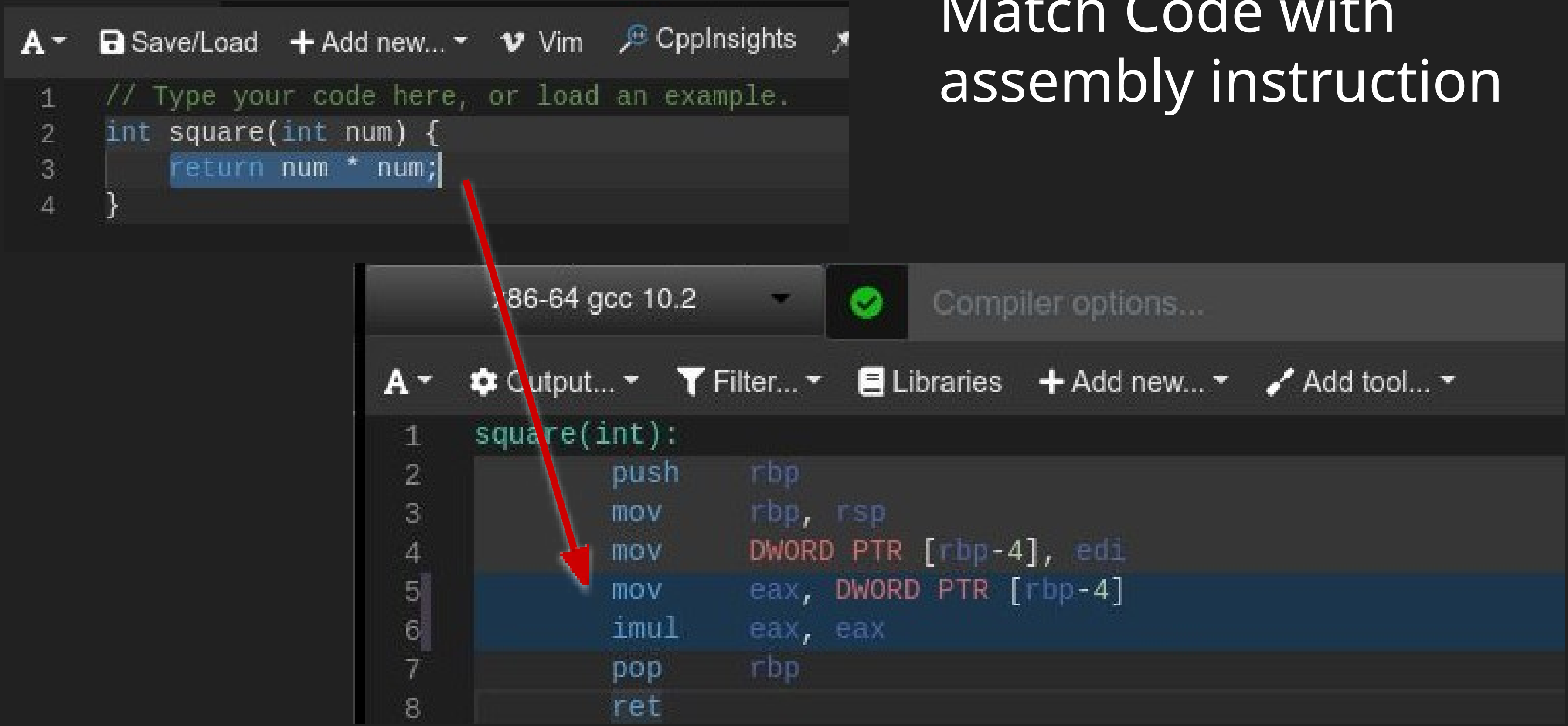




# Compiler Explorers Features

# Code Highlighting

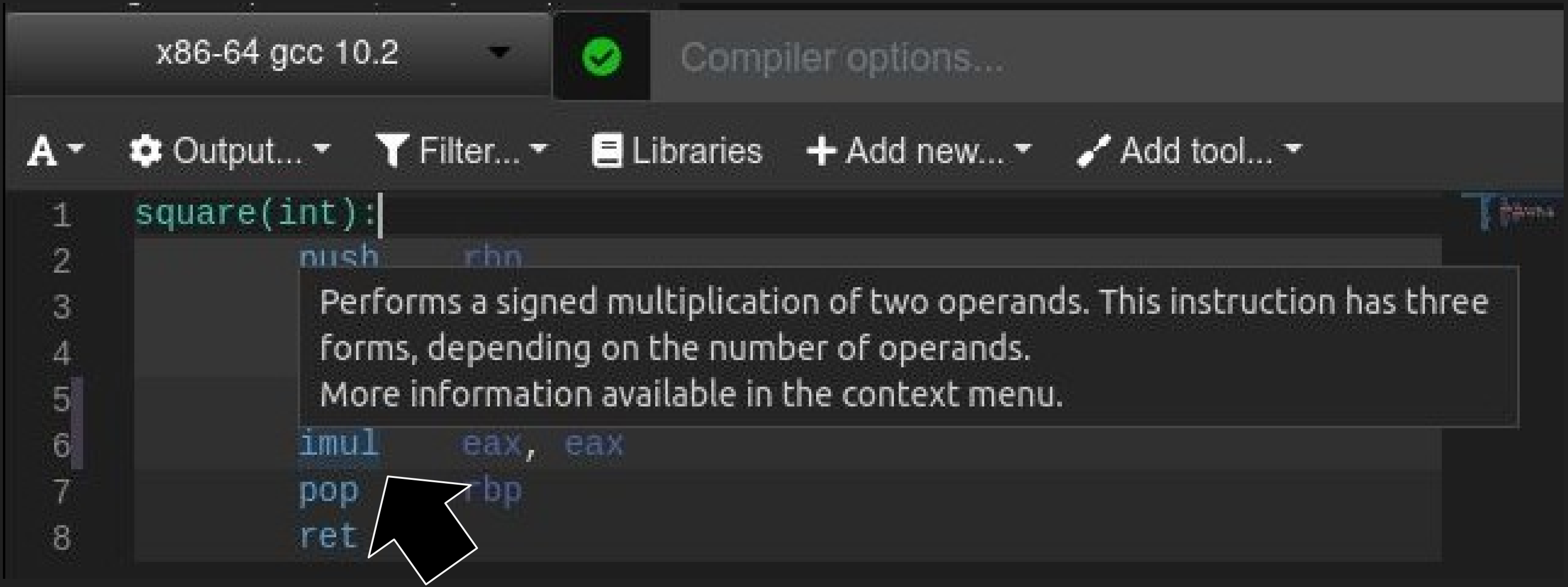
Match Code with assembly instruction



```
// Type your code here, or load an example.
int square(int num) {
    return num * num;
}
```

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

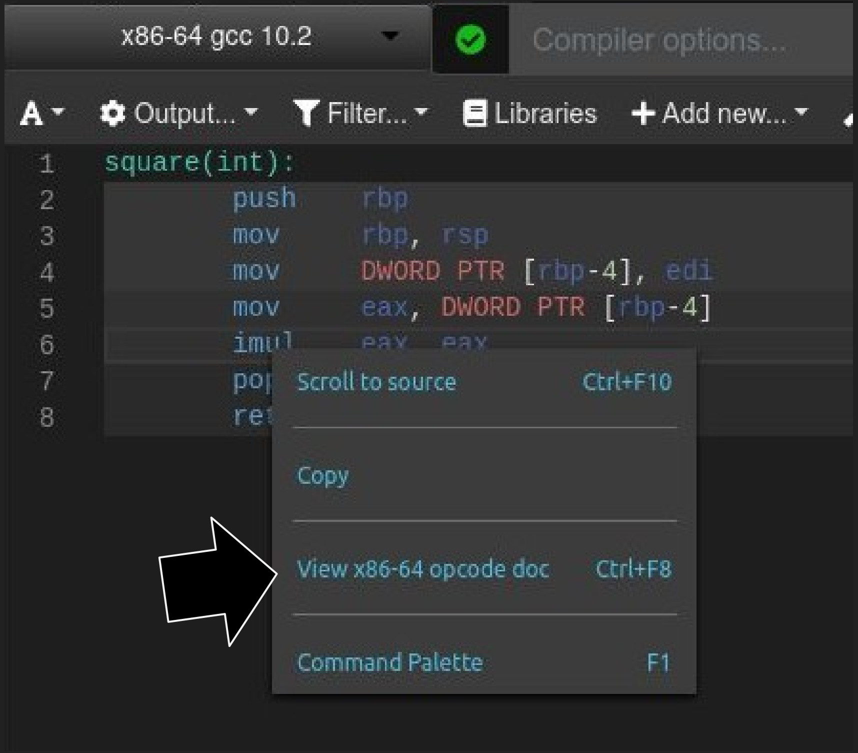
# Documentation



```
square(int):
        push    rbp
        imul    eax, eax
        pop     rbp
        ret
```

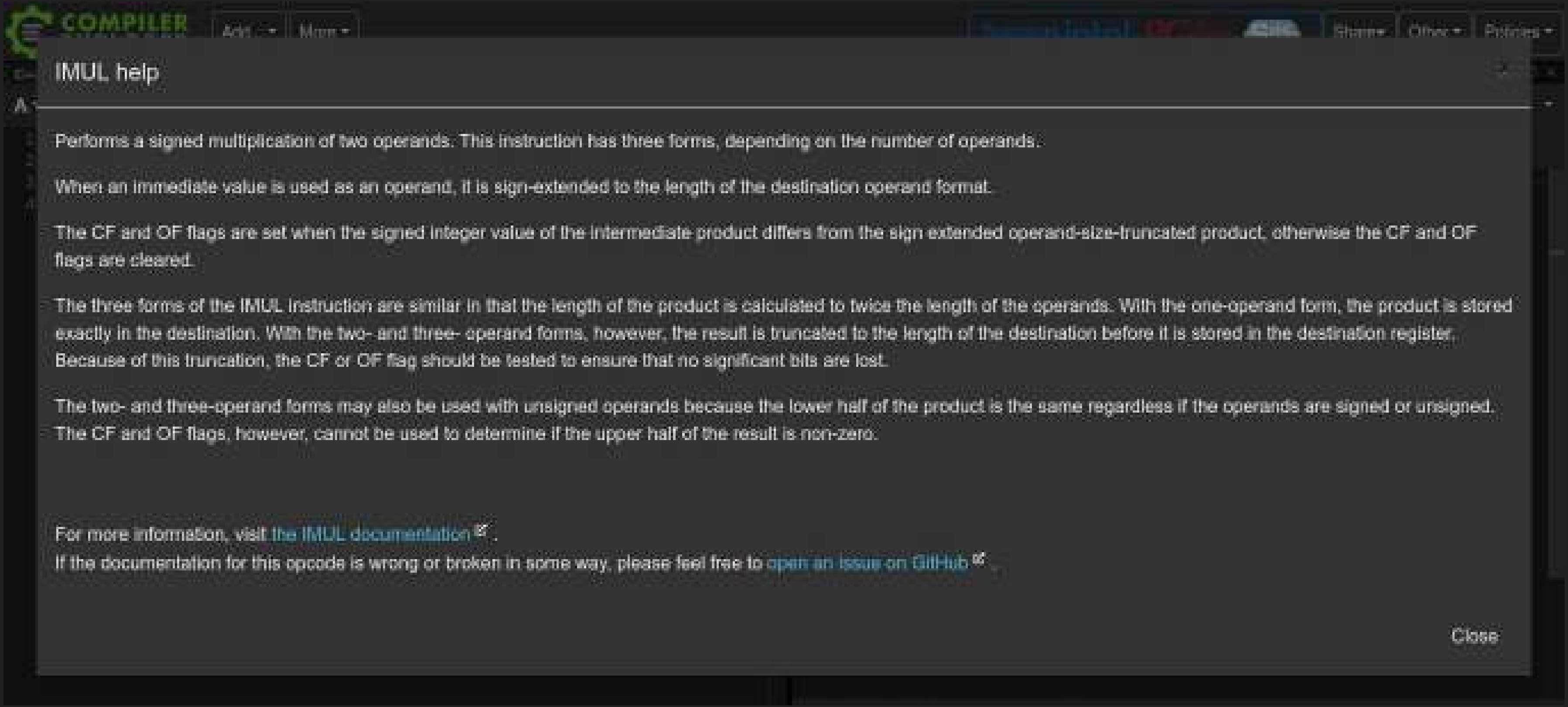
Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.
More information available in the context menu.

# Documentation



```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop
        ret
```

# Documentation



IMUL help

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The CF and OF flags are set when the signed integer value of the intermediate product differs from the sign extended operand-size-truncated product, otherwise the CF and OF flags are cleared.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three- operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

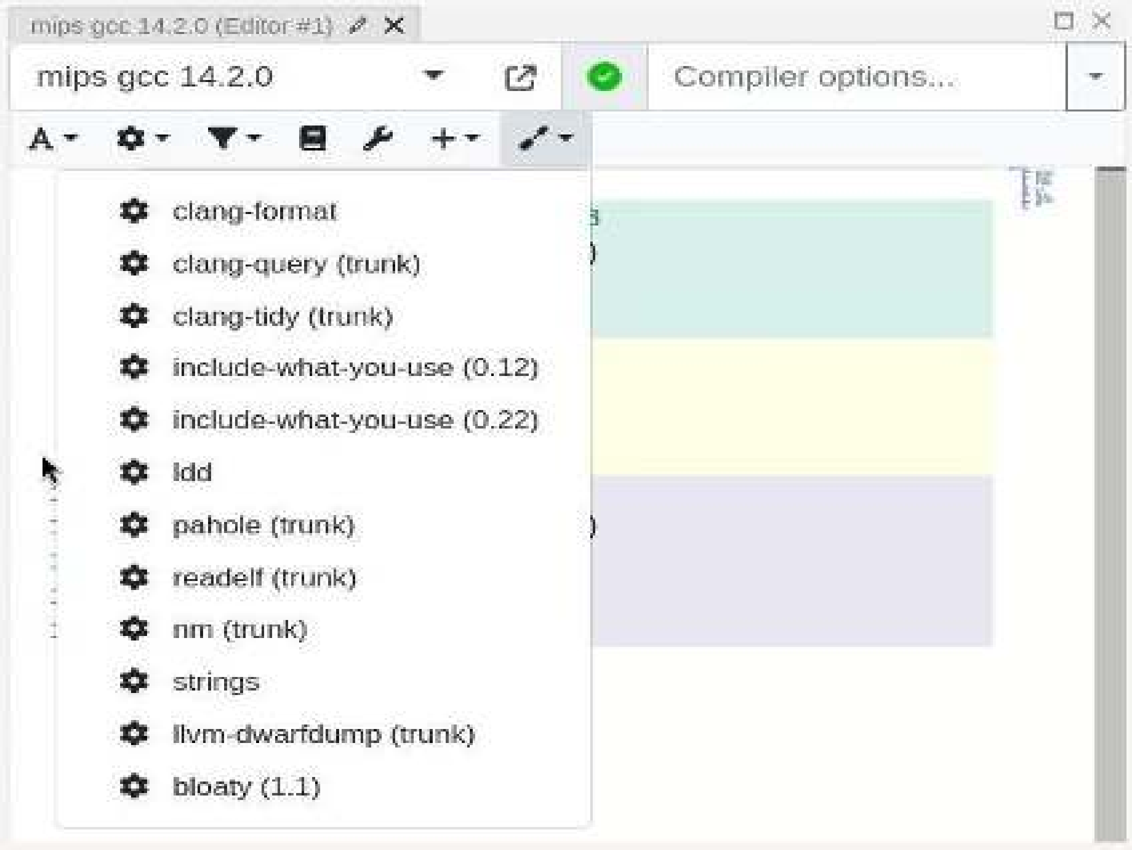
The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

For more information, visit the IMUL documentation.
If the documentation for this opcode is wrong or broken in some way, please feel free to open an issue on GitHub.

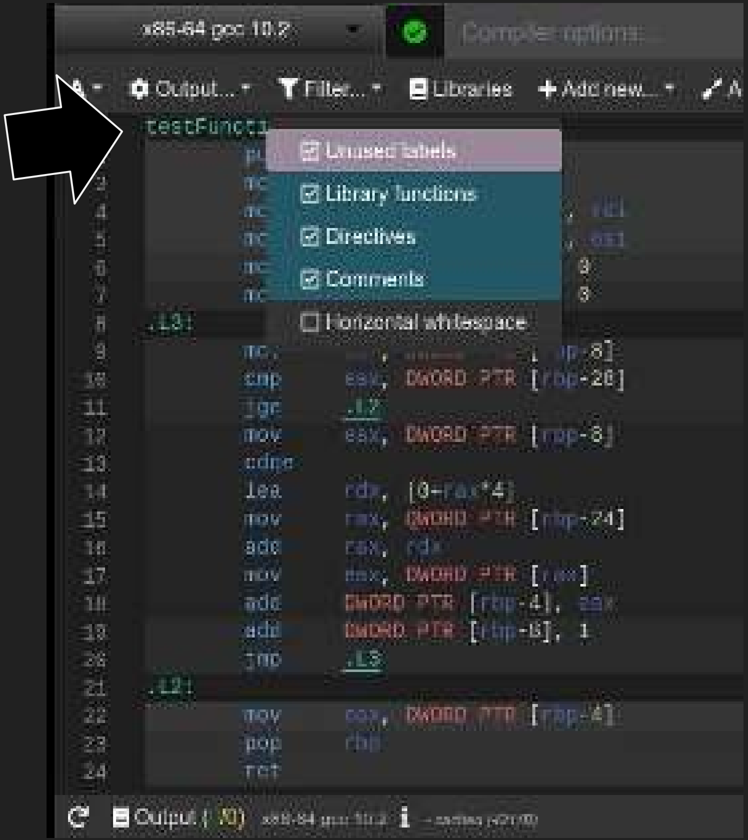
# Even more probing tools...

CLI toolbox

- ldd
- strings
- readelf



# Filtering options



☑ Unused labels

☑ Library functions

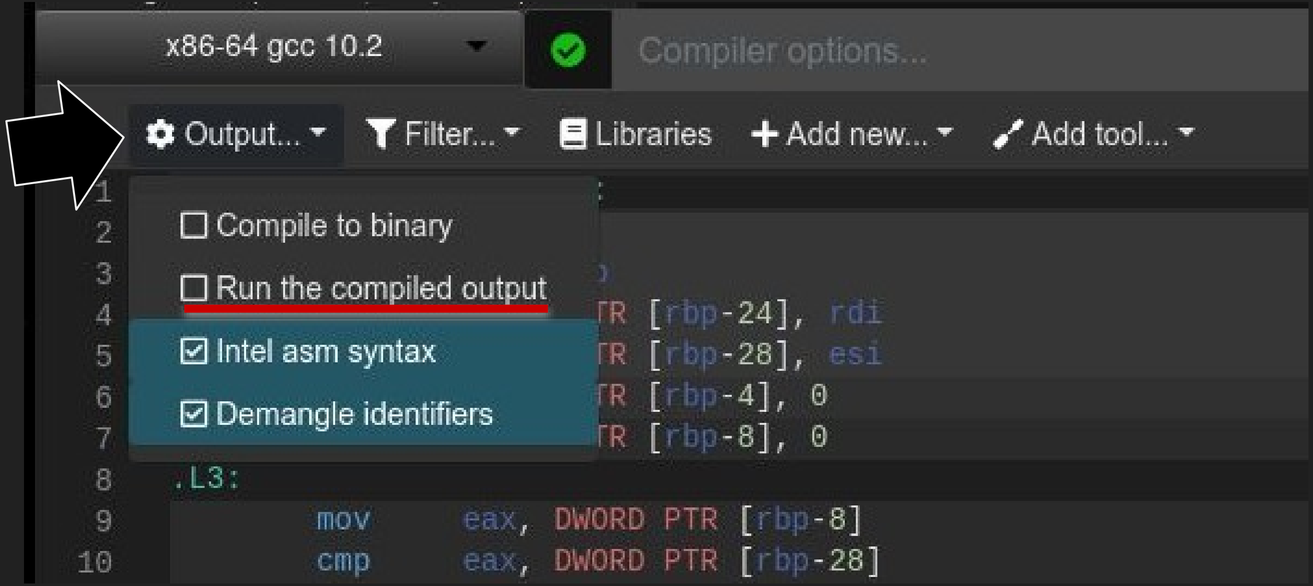
☑ Directives

☑ Comments

☐ Horizontal whitespace

```
testFuncti
        pu
        mo
        mo                      , rdi
        mo                      , esi
        mo                      0
        mo                      0
.L3:
        mo                      [ bp-8]
        cmp     eax, DWORD PTR [rbp-20]
        jge     .L2
        mov     eax, DWORD PTR [rbp-8]
        cdqe
        lea     rdx, [0+rax*4]
        mov     rax, QWORD PTR [rbp-24]
        add     rax, rdx
        mov     eax, DWORD PTR [rax]
        add     DWORD PTR [rbp-4], eax
        add     DWORD PTR [rbp-8], 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```

# Output options



☐ Compile to binary

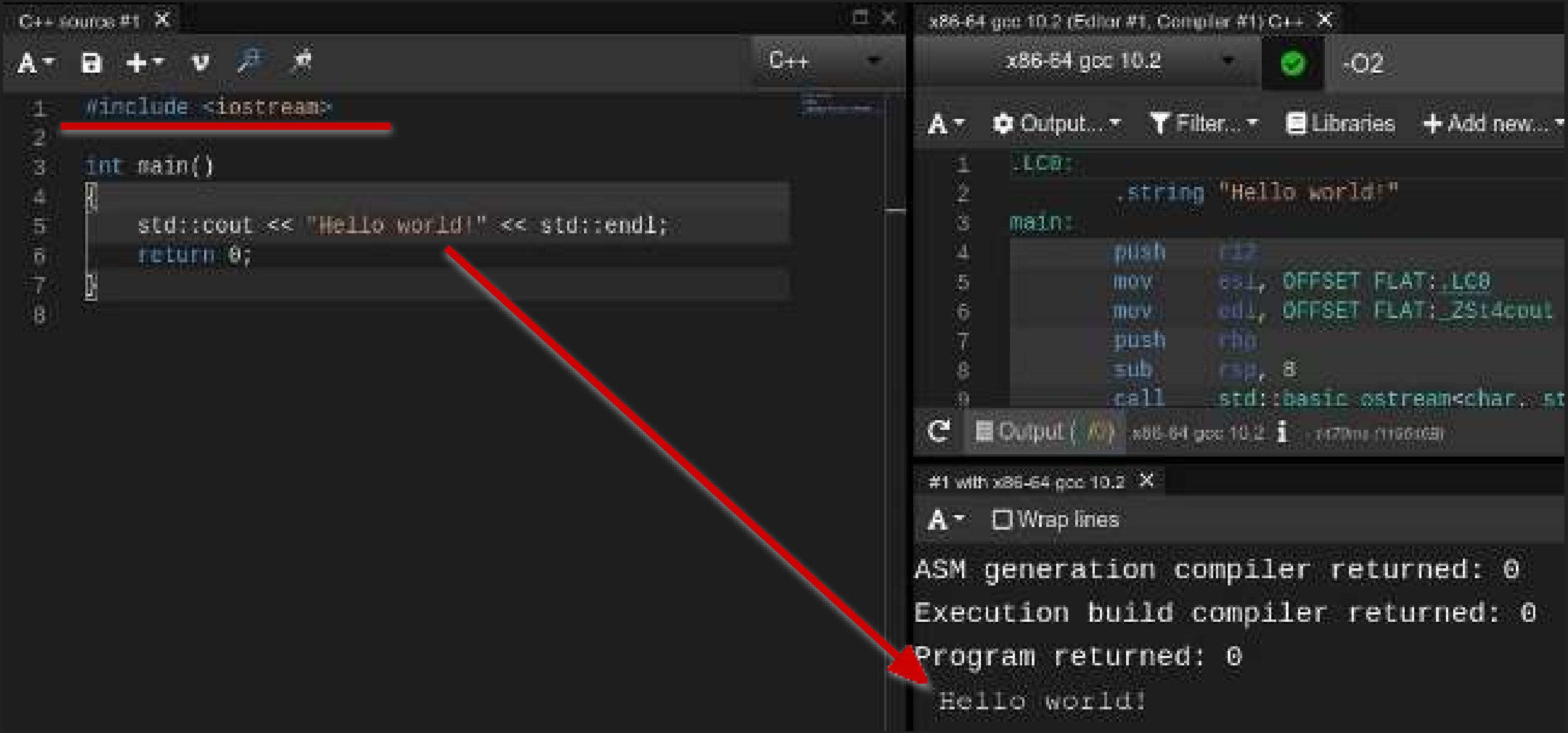
☐ Run the compiled output

☑ Intel asm syntax

☑ Demangle identifiers

```
TR [rbp-24], rdi
TR [rbp-28], esi
TR [rbp-4], 0
TR [rbp-8], 0
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cmp     eax, DWORD PTR [rbp-28]
```

# Code execution

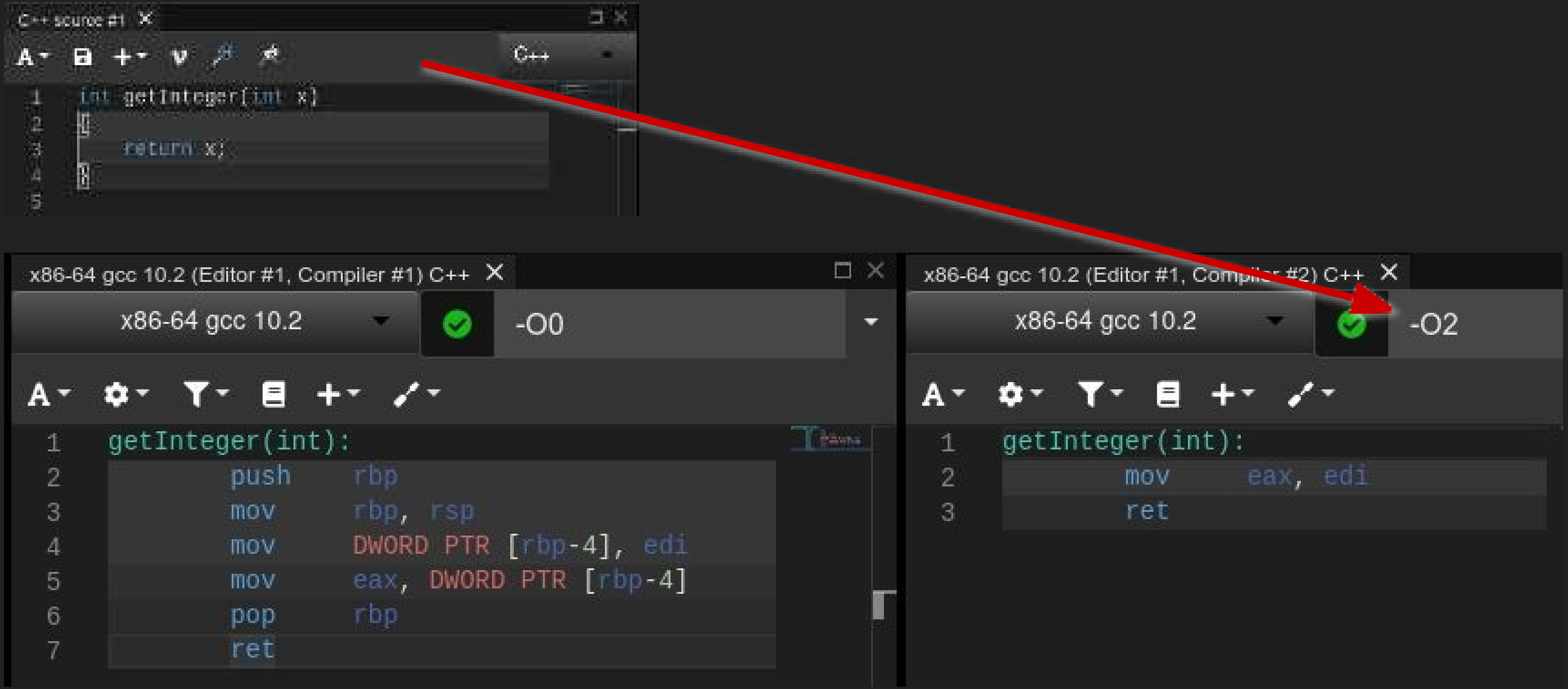


```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

```
.LC0:
        .string "Hello world!"
main:
        push    r12
        mov     esi, OFFSET FLAT:.LC0
        mov     edi, OFFSET FLAT:_ZSt4cout
        push    rbp
        sub     rsp, 8
        call    std::basic_ostream<char, st
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 Hello world!
```

# Compiler options



```
int getInteger(int x)
{
    return x;
}
```

x86-64 gcc 10.2 (Editor #1, Compiler #1) C++

x86-64 gcc 10.2 -O0

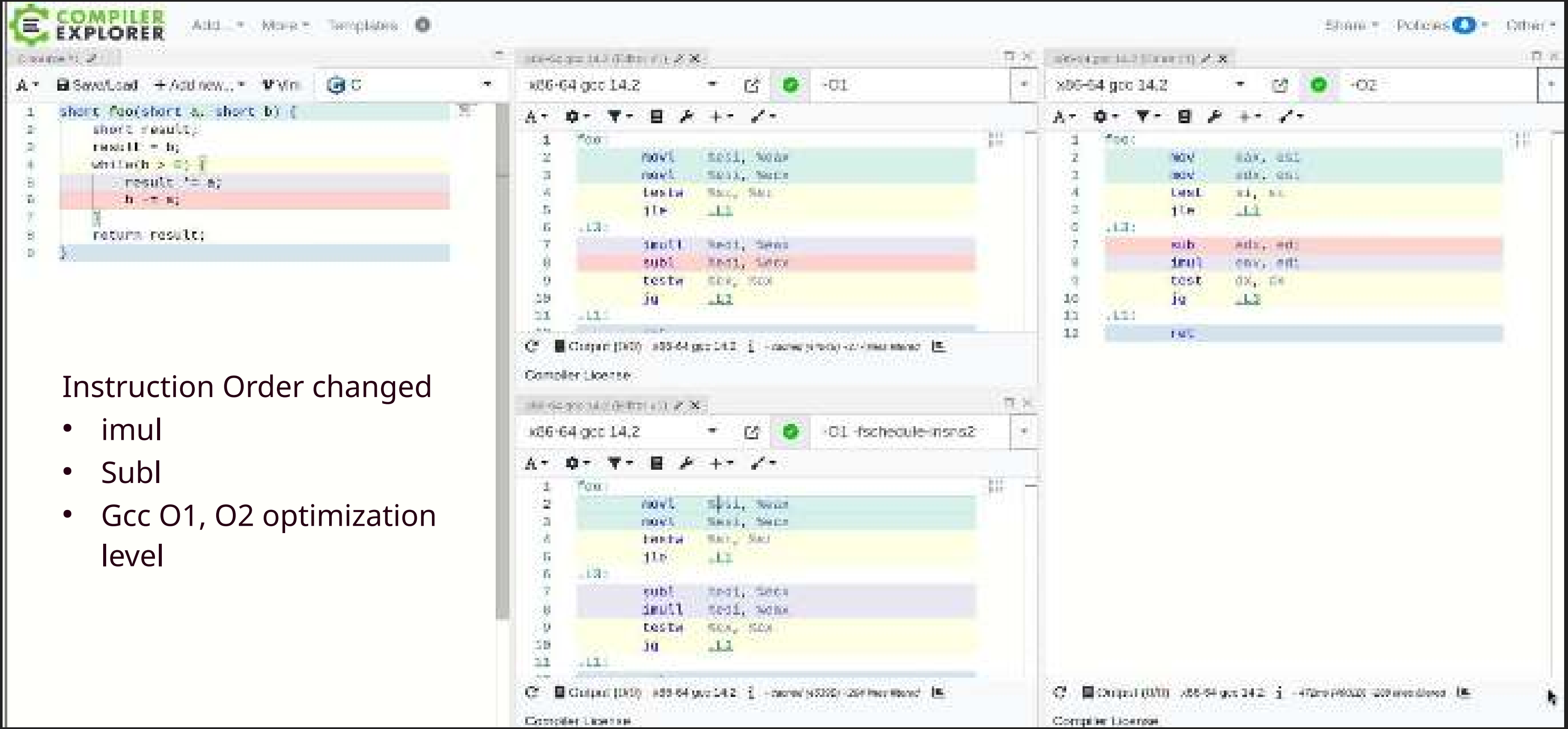
```
getInteger(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```

x86-64 gcc 10.2 (Editor #1, Compiler #2) C++

x86-64 gcc 10.2 -O2

```
getInteger(int):
        mov     eax, edi
        ret
```

# Compiler options



```c
short foo(short a, short b) {
    short result;
    result = b;
    while(b > 0) {
        result *= a;
        b -= a;
    }
    return result;
}
```

x86-64 gcc 14.2 -O1

```
foo:
        movl    %esi, %eax
        movl    %esi, %ecx
        testw   %si, %si
        jle     .L1
.L3:
        imull   %edi, %eax
        subl    %edi, %ecx
        testw   %cx, %cx
        jg      .L3
.L1:
```

x86-64 gcc 14.2 -O1 -fschedule-insns2

```
foo:
        movl    %esi, %eax
        movl    %esi, %ecx
        testw   %si, %si
        jle     .L1
.L3:
        subl    %edi, %ecx
        imull   %edi, %eax
        testw   %cx, %cx
        jg      .L3
.L1:
```

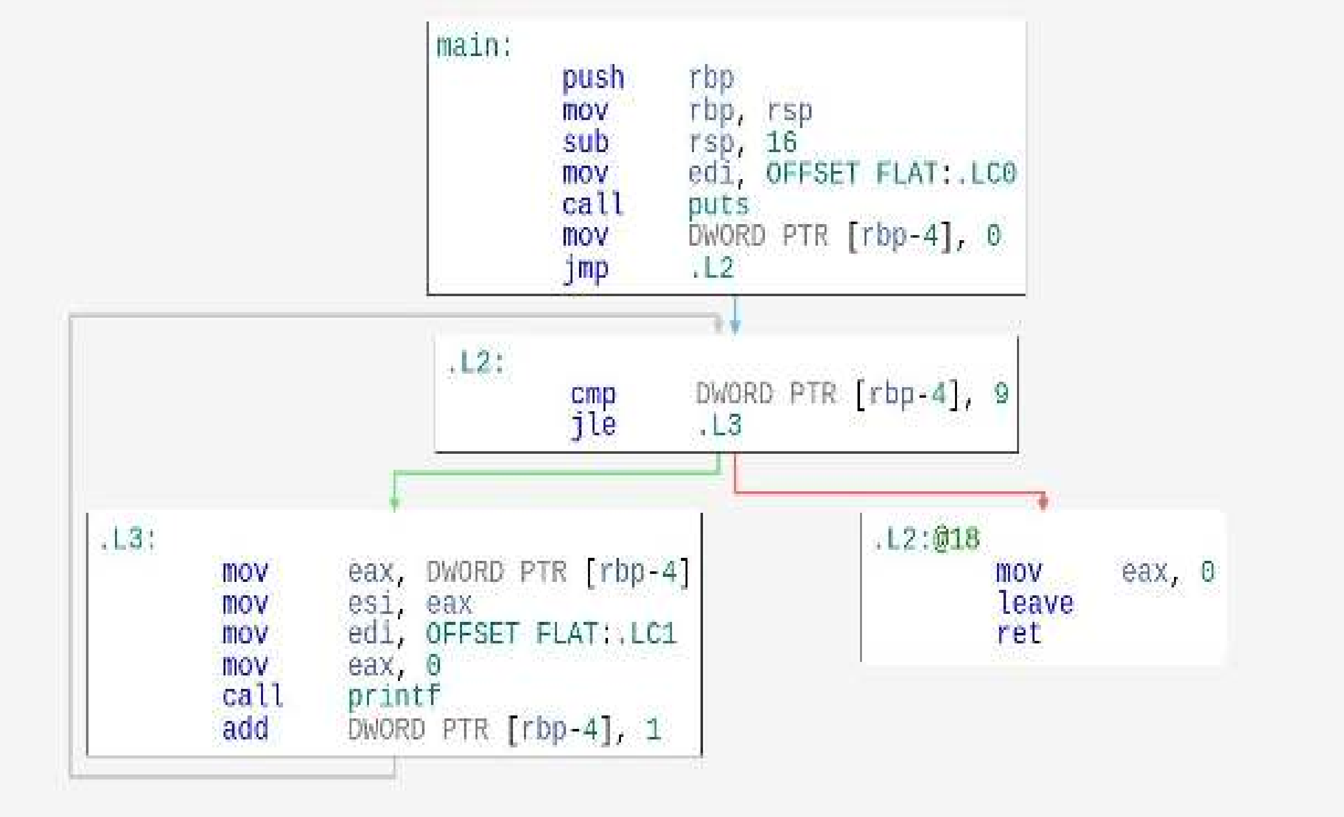
x86-64 gcc 14.2 -O2

```
foo:
        mov     eax, esi
        mov     edx, esi
        test    si, si
        jle     .L1
.L3:
        sub     edx, edi
        imul    eax, edi
        test    dx, dx
        jg      .L3
.L1:
        ret
```

Instruction Order changed

- imul
- Subl
- Gcc O1, O2 optimization level

# Control Flow Graph



```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     DWORD PTR [rbp-4], 0
        jmp     .L2
```

```
.L2:
        cmp     DWORD PTR [rbp-4], 9
        jle     .L3
```

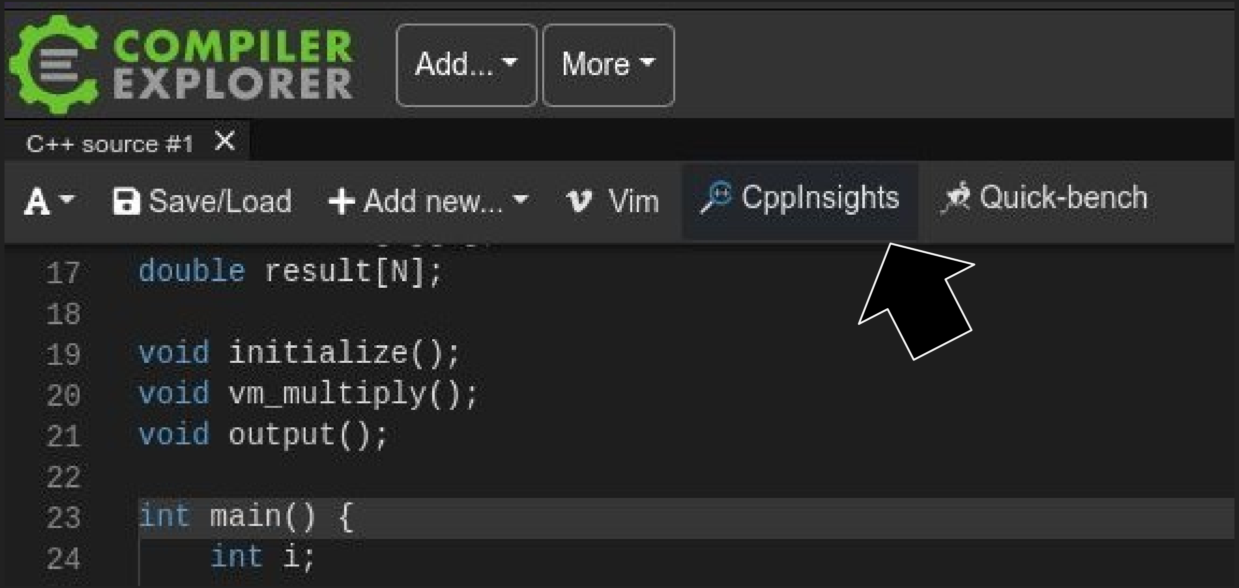
```
.L3:
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    printf
        add     DWORD PTR [rbp-4], 1
```

```
.L2:@18
        mov     eax, 0
        leave
        ret
```

# Other Tools

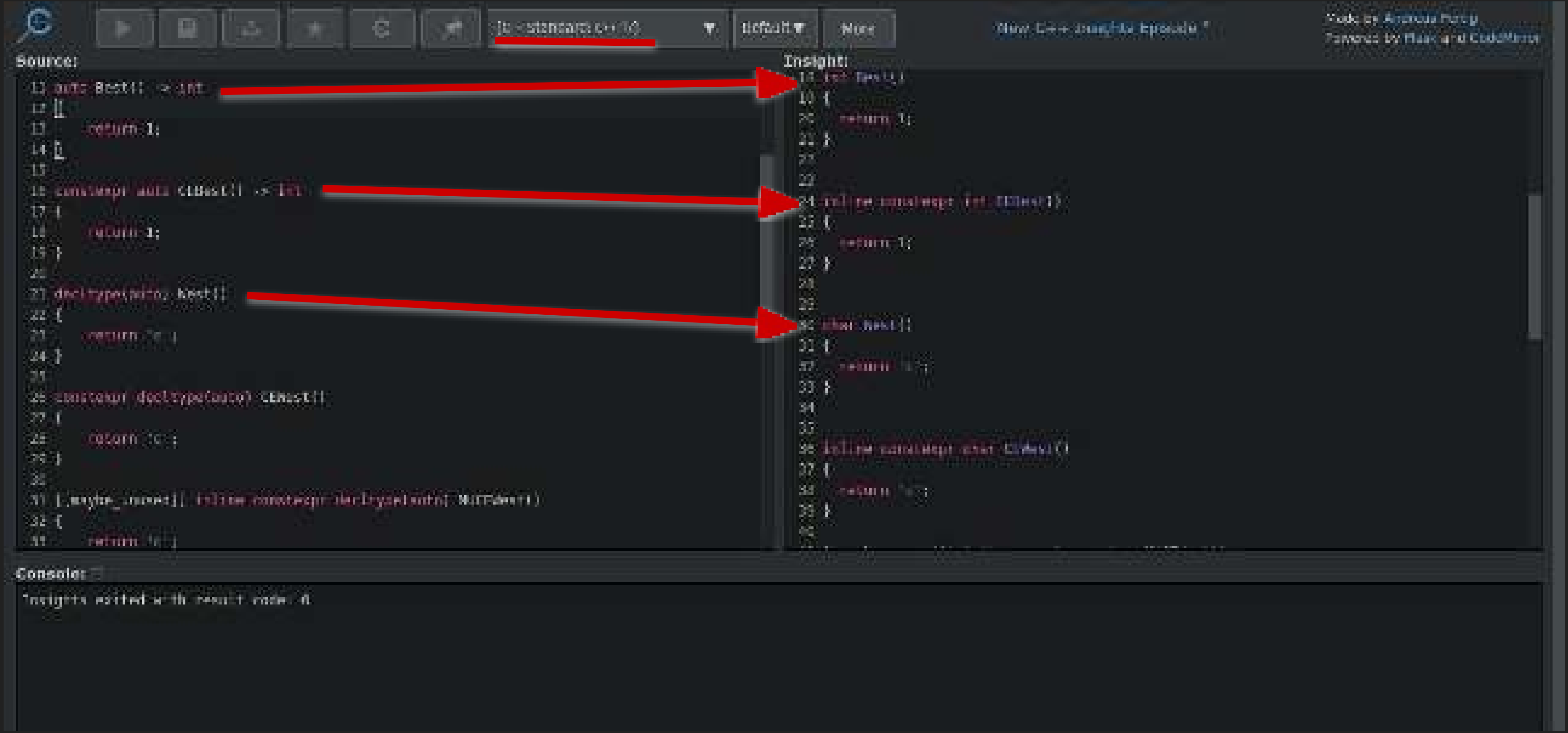
# C++
# Insights

# C++ Insights



```
double result[N];

void initialize();
void vm_multiply();
void output();

int main() {
    int i;
```

# C++ Insights



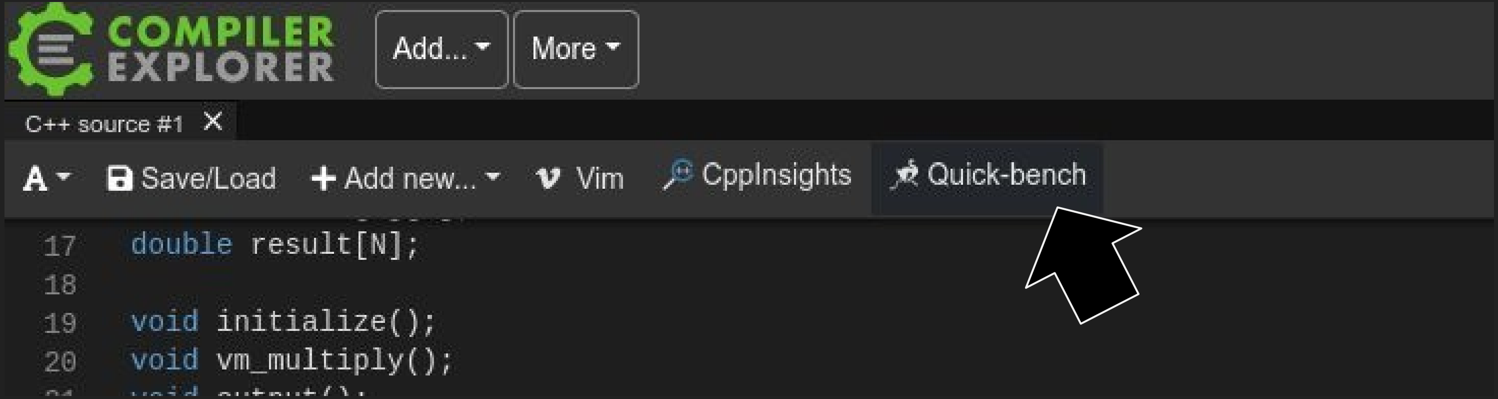
```
auto Best() -> int
{
    return 1;
}

constexpr auto CBest() -> int
{
    return 1;
}

decltype(auto) Best()
{
    return 'c';
}

constexpr decltype(auto) CBest()
{
    return 'c';
}

[[maybe_unused]] inline constexpr decltype(auto) MUTBest()
{
    return 'c';
```

```
int Best()
{
  return 1;
}


inline constexpr int CBest()
{
  return 1;
}


char Best()
{
  return 'c';
}


inline constexpr char CBest()
{
  return 'c';
}
```

```
Insights exited with result code: 0
```

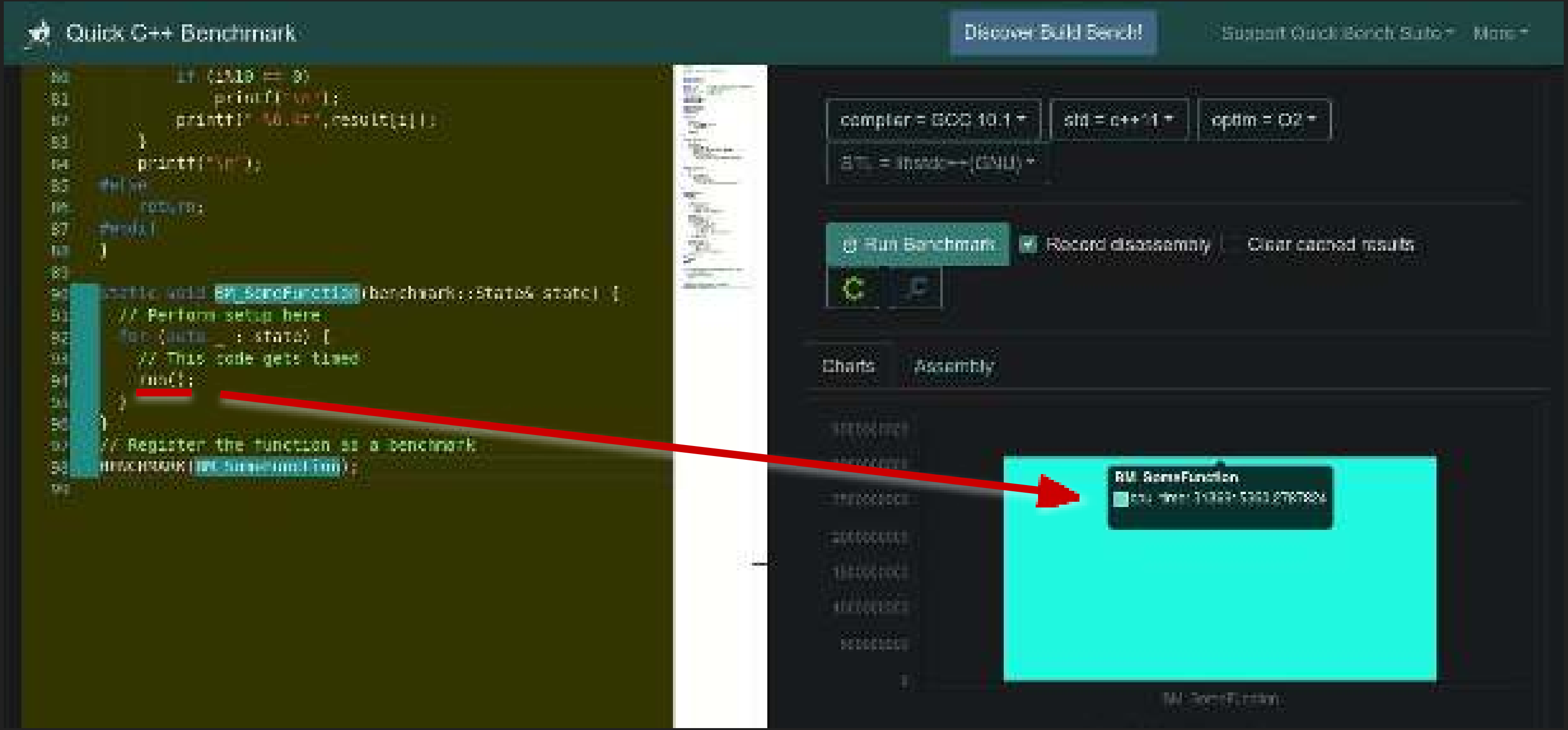
# Quick-bench

# Quick-bench



```
double result[N];

void initialize();
void vm_multiply();
void output();
```

# Quick C++ Benchmark



```
            if (i%10 == 9)
                printf("\n");
            printf("%6.4f",result[i]);
        }
        printf("\n");
#else
        return;
#endif
}

static void BM_SomeFunction(benchmark::State& state) {
  // Perform setup here
  for (auto _ : state) {
    // This code gets timed
    run();
  }
}
// Register the function as a benchmark
BENCHMARK(BM_SomeFunction);
```

compiler = GCC 10.1

std = c++11

optim = O2

STL = libstdc++(GNU)

BM_SomeFunction

cpu_time: [illegible]

# Quick C++ Benchmark

- Relies on Google Benchmark
  - github.com/google/benchmark

## Summary

- It’s in the browser!
- Edit, Compile, View Disassembly, benchmarking and execution.
- Many Languages, Compilers, architectures and configuration options
- C, C++, Rust, C#, GO, FORTRAN, Python, Ruby, Java, …
- Armv7, AVR, X86, MIPS, RISC-V, SPARC, VAX, and many more.

# Questions?

# Thanks