

Forcing the Compiler to Generate **Jump Tables**

Lennox Shou Hao Ho
lennox.ho@intel.com

A Trivial Example

A Save/Load + Add new... Vim CppInsights C++

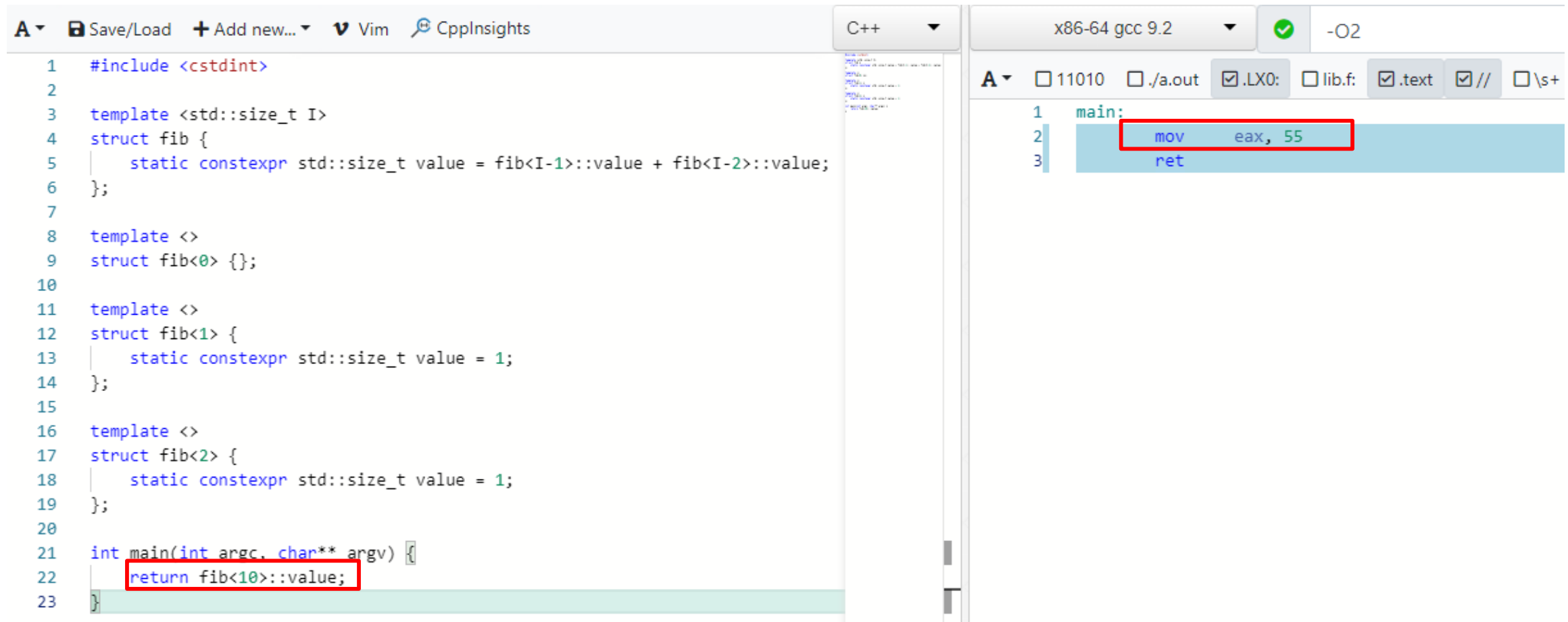
```
1 #include <cstdlib>
2
3 int foo(std::size_t index) {
4     int ret = 0;
5
6     if (index == 1) {
7         ret = 7;
8     }
9     else if (index == 2) {
10        ret = 1;
11    }
12    else if (index == 3) {
13        ret = 6;
14    }
15
16    return ret;
17 }
```

x86-64 gcc 9.2 -O2

A ☐ 11010 ☐ ./a.out ☒ .LX0: ☐ lib.f: ☒ .text ☒ //

```
1 foo(unsigned long):
2     mov     eax, 7
3     cmp     rdi, 1
4     je      .L1
5     mov     eax, 1
6     cmp     rdi, 2
7     je      .L1
8     cmp     rdi, 3
9     mov     eax, 0
10    mov     edx, 6
11    cmov     eax, edx
12 .L1:
13    ret
```

But First, Automatically Generate if-else cases



The screenshot displays the CppInsights web interface. On the left, the C++ source code for a Fibonacci sequence is shown. The code uses a template-based struct to calculate the 10th Fibonacci number. Two instances of the code are highlighted with red boxes: the `return fib<10>::value;` statement in the `main` function and the recursive calculation `static constexpr std::size_t value = fib<I-1>::value + fib<I-2>::value;` in the `fib` struct template. On the right, the assembly output for the `main` function is shown. The assembly consists of two instructions: `mov eax, 55` and `ret`. The `mov` instruction is highlighted with a red box, indicating that the compiler has optimized the entire Fibonacci calculation into a single constant move operation.

```
1 #include <stdint>
2
3 template <std::size_t I>
4 struct fib {
5     static constexpr std::size_t value = fib<I-1>::value + fib<I-2>::value;
6 };
7
8 template <>
9 struct fib<0> {};
10
11 template <>
12 struct fib<1> {
13     static constexpr std::size_t value = 1;
14 };
15
16 template <>
17 struct fib<2> {
18     static constexpr std::size_t value = 1;
19 };
20
21 int main(int argc, char** argv) {
22     return fib<10>::value;
23 }
```

Assembly output for `main`:

```
1 main:
2     mov     eax, 55
3     ret
```

But First, Automatically Generate if-else cases

```
1  #include <stdint>
2
3  template <std::size_t I>
4  struct fib {
5      static constexpr std::size_t value = fib<I-1>::value + fib<I-2>::value;
6
7      constexpr std::size_t operator()(std::size_t index) const {
8          if (index == I) return value;
9          else return fib<I-1>()(index);
10     }
11 };
12
13 template <>
14 struct fib<0> {};
15
16 template <>
17 struct fib<1> {
18     static constexpr std::size_t value = 1;
19     constexpr std::size_t operator()(std::size_t index) const { return value; }
20 };
21
22 template <>
23 struct fib<2> {
24     static constexpr std::size_t value = 1;
25     constexpr std::size_t operator()(std::size_t index) const { return value; }
26 };
27
28 int main(int argc, char** argv) {
29     fib<5> fib_lookup;
30     return fib_lookup(argc);
31 }
```

Fibonacci : 5 numbers

```
x86-64 gcc 9.2  -O2
A 11010 ./a.out ☒ .LX0: ☐ lib.f: ☒ .text ☒ // ☐ \s+
1  main:
2      movsx    rax, edi
3      cmp     edi, 5
4      je      .L3
5      cmp     rax, 4
6      je      .L4
7      cmp     rax, 3
8      sete    al
9      movzx   eax, al
10     add     eax, 1
11     ret
12  .L3:
13     mov     eax, 5
14     ret
15  .L4:
16     mov     eax, 3
17     ret
```

Fibonacci : 500 numbers

```
27  
28 int main(int argc, char** argv) {  
29     fib<500> fib_lookup;  
30     return fib_lookup(argc);  
31 }  
32  
33
```

x86-64 gcc 9.2 -O3

A ▾ ☐ 11010 ☐ ./a.out ☒ .LX0: ☐ lib.f: ☒ .text ☒ // ☐ \s+

```
1  main:  
2      movsx    rax, edi  
3      cmp     edi, 500  
4      je      .L3  
5      cmp     rax, 499  
6      je      .L4  
7      cmp     rax, 498  
8      je      .L5  
9      cmp     rax, 497  
10     je      .L6  
11     cmp     rax, 496  
12     je      .L7  
13     cmp     rax, 495  
14     je      .L8  
15     cmp     rax, 494  
16     je      .L9  
17     cmp     rax, 493  
18     je      .L10  
19     cmp     rax, 492  
20     je      .L11  
21     cmp     rax, 491  
22     je      .L12  
23     cmp     rax, 490  
24     je      .L13  
25     cmp     rax, 489  
26     je      .L14  
27     cmp     rax, 488  
28     je      .L15  
29     cmp     rax, 487  
30     je      .L16  
31     cmp     rax, 486  
32     je      .L17  
33     cmp     rax, 485  
34     je      .L18  
35     cmp     rax, 484  
36     je      .L19  
37     cmp     rax, 483  
38     je      .L20
```

Fibonacci : 500 numbers

975	cmp	rax, 14
976	je	.L489
977	cmp	rax, 13
978	je	.L490
979	cmp	rax, 12
980	je	.L491
981	cmp	rax, 11
982	je	.L492
983	cmp	rax, 10
984	je	.L493
985	cmp	rax, 9
986	je	.L494
987	cmp	rax, 8
988	je	.L495
989	cmp	rax, 7
990	je	.L496
991	cmp	rax, 6
992	je	.L497
993	cmp	rax, 5
994	je	.L2
995	cmp	rax, 4
996	je	.L498
997	cmp	rax, 3
998	je	.L504
999	mov	eax, 1

More Practical Problem : Jump Table + Variant Visit

- Currently all major implementations of Variant classes (`std::variant`, `boost::variant`, `boost::variant2`) perform linear lookup
- This is generally fine, until N is large and/or the visitor function is non-trivial
- Want the choice to use jump tables if we so choose

How to Generate Jump Table? Basic Observation

```
1  #include <cstdlib>
2
3  int foo_1() { return 7; }
4  int foo_2() { return 1; }
5  int foo_3() { return 6; }
6
7  int foo(std::size_t index) {
8      static constexpr decltype(&foo_1) jmp_table[] = { foo_1, foo_2, foo_3 };
9      return jmp_table[index]();
10 }
```

x86-64 gcc 9.2 (Editor #1, Compiler #1) C++ X

x86-64 gcc 9.2 -O2

A ▾ ☐ 11010 ☐ ./a.out ☒ .LX0: ☐ lib.f: ☒ .text ☒ // ☐ \s+ ☒ Intel ☒ Demangle ☐ Li

```
1  foo_1():
2      mov     eax, 7
3      ret
4  foo_2():
5      mov     eax, 1
6      ret
7  foo_3():
8      mov     eax, 6
9      ret
10 foo(unsigned long):
11     jmp     [QWORD PTR foo(unsigned long)::jmp_table[0+rdi*8]]
12 foo(unsigned long)::jmp_table:
13     .quad   foo_1()
14     .quad   foo_2()
15     .quad   foo_3()
```

Use Function Template to Generate Callbacks

```
template <std::size_t Index, typename Visitor, typename Variant>
decltype(auto) visit_callback(Visitor visitor, Variant variant) {
    return visitor(std::get<Index>(variant));
}
```

std::get can take an index



Use Variadic Template to Generate Indices

```
template <std::size_t... I>
struct visit_impl<std::index_sequence<I...>> {

    template <typename Visitor, typename Variant>
    decltype(auto) operator()(Visitor &&visitor, Variant &&variant) const {
        using callback_type = decltype(&visit_callback<0u, decltype(visitor), decltype(variant)>);

        static constexpr callback_type jmp_table[] = { visit_callback<I, decltype(visitor), decltype(variant)>... };

        return jmp_table[variant.index()](std::forward<Visitor>(visitor),
                                           std::forward<Variant>(variant));
    }
};
```

Get Variant Size

```
template <typename Visitor, typename Variant>  
inline decltype(auto) visit(Visitor &&visitor, Variant &&variant) {  
    static constexpr std::size_t N = std::variant_size_v<std::decay_t<Variant>>;  
    return detail::visit_impl<std::make_index_sequence<N>>>()(std::forward<Visitor>(visitor),  
                                                                std::forward<Variant>(variant));  
}
```

Generated ASM

```
struct A {};  
struct B {};  
struct C {};  
  
struct func {  
    int operator()(A) const { return 4; }  
    int operator()(B) const { return 5; }  
    int operator()(C) const { return 6; }  
};  
  
int foo(const std::variant<A, B, C> &var) {  
    return visit(func(), var);  
}
```

```
1  decltype(auto) detail::visit_callback<0ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):  
2      cmp     BYTE PTR [rsi+1], 0  
3      jne     .L7  
4      mov     eax, 4  
5      ret  
6  .L7:  
7      push    rax  
8      call    abort  
9  decltype(auto) detail::visit_callback<1ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):  
10     cmp     BYTE PTR [rsi+1], 1  
11     jne     .L13  
12     mov     eax, 5  
13     ret  
14 .L13:  
15     push    rax  
16     call    abort  
17 decltype(auto) detail::visit_callback<2ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):  
18     cmp     BYTE PTR [rsi+1], 2  
19     jne     .L19  
20     mov     eax, 6  
21     ret  
22 .L19:  
23     push    rax  
24     call    abort  
  
.quad decltype(auto) detail::visit_callback<0ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)  
.quad decltype(auto) detail::visit_callback<1ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)  
.quad decltype(auto) detail::visit_callback<2ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)
```

```
call [QWORD PTR decltype(auto) detail::visit_impl<std::integer_s
```

Sneak Peek – constexpr hash map/set

```
bool is_magic_number(int num) {  
    constexpr std::array values = { 33, 23, 532, 32, 10, 55, 74, 101, 64 };  
    constexpr std::size_t num_buckets = 4;  
  
    constexpr auto set = make_constexpr_hash_set<num_buckets, int_hash>(values);  
    return set.contains(num);  
}
```

x86-64 clang 9.0.0 -O2 -std=c++17

A 11010 /a.out .LX0: lib.f: .text // \s+ Intel Demangle

```
1 is_magic_number(int):                # @is_magic_number(int)  
2     mov     eax, edi  
3     and     eax, 3  
4     mov     rcx, qword ptr [8*rax + .L__const.is_magic_number(int).set]  
5     mov     rdx, qword ptr [8*rax + .L__const.is_magic_number(int).set+8]  
6     mov     eax, 9  
7     cmp     rcx, rdx  
8     jae     .LBB0_4  
9 .LBB0_1:                            # =>This Inner Loop Header: Depth=1  
10    cmp     dword ptr [4*rcx + .L__const.is_magic_number(int).set+32], edi  
11    je      .LBB0_2  
12    add     rcx, 1  
13    cmp     rdx, rcx  
14    jne     .LBB0_1  
15    jmp     .LBB0_4  
16 .LBB0_2:  
17     mov     rax, rcx  
18 .LBB0_4:  
19     cmp     rax, 9  
20     setne  al  
21     ret  
22 .L__const.is_magic_number(int).set:  
23     .quad  0                # 0x0  
24     .quad  3                # 0x3  
25     .quad  5                # 0x5  
26     .quad  7                # 0x7  
27     .long  532              # 0x214  
28     .long  32               # 0x20  
29     .long  64               # 0x40  
30     .long  33               # 0x21  
31     .long  101              # 0x65  
32     .long  10               # 0xa  
33     .long  74               # 0x4a  
34     .long  23               # 0x17  
35     .long  55               # 0x37  
36     .zero  4
```