COOL C++ COMPILER TIDBITS

MATT GODBOLT @MATTGODBOLT

JUNE 5TH 2021

HELLO!

HELLO!

I’m accidentally a verb

COMPILER
EXPLORER

COMPILERS ARE AMAZING!!

```
void LightObject(Renderable *object)
{
  auto world(object->GetMatrix());
  SetWorldMatrix(world);
  auto lMat = world * worldToLight;
  auto boundLs = TransformBounds(
      object->GetBounds(), lMat);
  if (!boundLs.Overlap(lightSphere))
    return;
  auto ms = pos * world.Invert();
  ...
```

```
void LightObject(Renderable *object)
{
  auto world(object->GetMatrix());
  SetWorldMatrix(world);
  auto lMat = world * worldToLight;
  auto boundLs = TransformBounds(
      object->GetBounds(), lMat);
  if (!boundLs.Overlap(lightSphere))
    return;
  auto ms = pos * world.Invert();
  ...
```

```
template<typename T>
auto thingy(T &&t)
{
  auto bongo = t.fronk();
  if (bongo < T::min())
  {
    return t.badger();
  } else {
    return t.smeagol();
  }
}
```

```
template<typename T>
auto thingy(T &&t)
{
  auto bongo = t.fronk();
  if (bongo < T::min())
  {
    return t.badger();
  } else {
    return t.smeagol();
  }
}
```

```
In instantiation of 'auto thingy(T&&)
  [with T = Marmoset]':
error: 'struct Marmoset' has no member
    named 'fronk'
   auto bongo = t.fronk();
                  ~~^~~~~
error: 'min' is not a member of 'Marmoset'
   if (bongo < T::min())
                 ~~~~~~^~
error: 'struct Marmoset' has no member
    named 'badger'
     return t.badger();
            ~~^~~~~~
...
```

```
int sumSquared(
    const vector<int> &v) {
  int res = 0;
  for (auto i : v)
    res += i * i;
  return res;
}
```

```
int sumSquared(
    const vector<int> &v) {
  int res = 0;
  for (auto i : v)
    res += i * i;
  return res;
}
```

```
.L4:
  vmovdqu ymm2, YMMWORD PTR [rax]
  add rax, 32
  vpmulld ymm0, ymm2, ymm2
  vpaddd ymm1, ymm1, ymm0
  cmp rax, rdx
  jne .L4
```

ASSEMBLY!?

ASSEMBLY!?

Ssolbergj, Wikimedia Commons CC SA 4.0

X86-64 ASSEMBLY

```
instr
instr dest_operand
instr dest_operand, source_operand
instr dest_operand, source_operand, source_operand
```

X86-64 ASSEMBLY

```
instr
instr dest_operand
instr dest_operand, source_operand
instr dest_operand, source_operand, source_operand
```

```
ret                                 ; return
inc rax                             ; increment "rax"
mov edx, 1234                       ; set "edx" to the value 1234
add rsi, rdi                        ; "rsi" += "rdi"
vpaddd ymm1, ymm2, ymm0             ; "ymm1" = "ymm2" + "ymm0"
```

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- mov
- movzx
- movsxd
- lea
- call
- ret
- jmp

- push
- pop
- cmp
- test
- je
- jne

- and
- xor
- add
- sub
- shl
- shr
- sar

INSTRUCTIONS

- `mov`
- `movzx`
- `movsxd`
- `lea`
- `call`
- `ret`
- `jmp`

- `push`
- `pop`
- `cmp`
- `test`
- `je`
- `jne`

- `and`
- `xor`
- `add`
- `sub`
- `shl`
- `shr`
- `sar`

And many, many more...

OPERANDS

```
register                                    ; e.g. rax, rbx, ecx...
constant                                    ; e.g. 1234
<size> ptr [register]                       ; e.g. DWORD PTR [rax]
<size> ptr [register + offset]              ; e.g. BYTE PTR [rcd + rsi]
<size> ptr [register + offset + register2 * (1,2,4,8)]
```

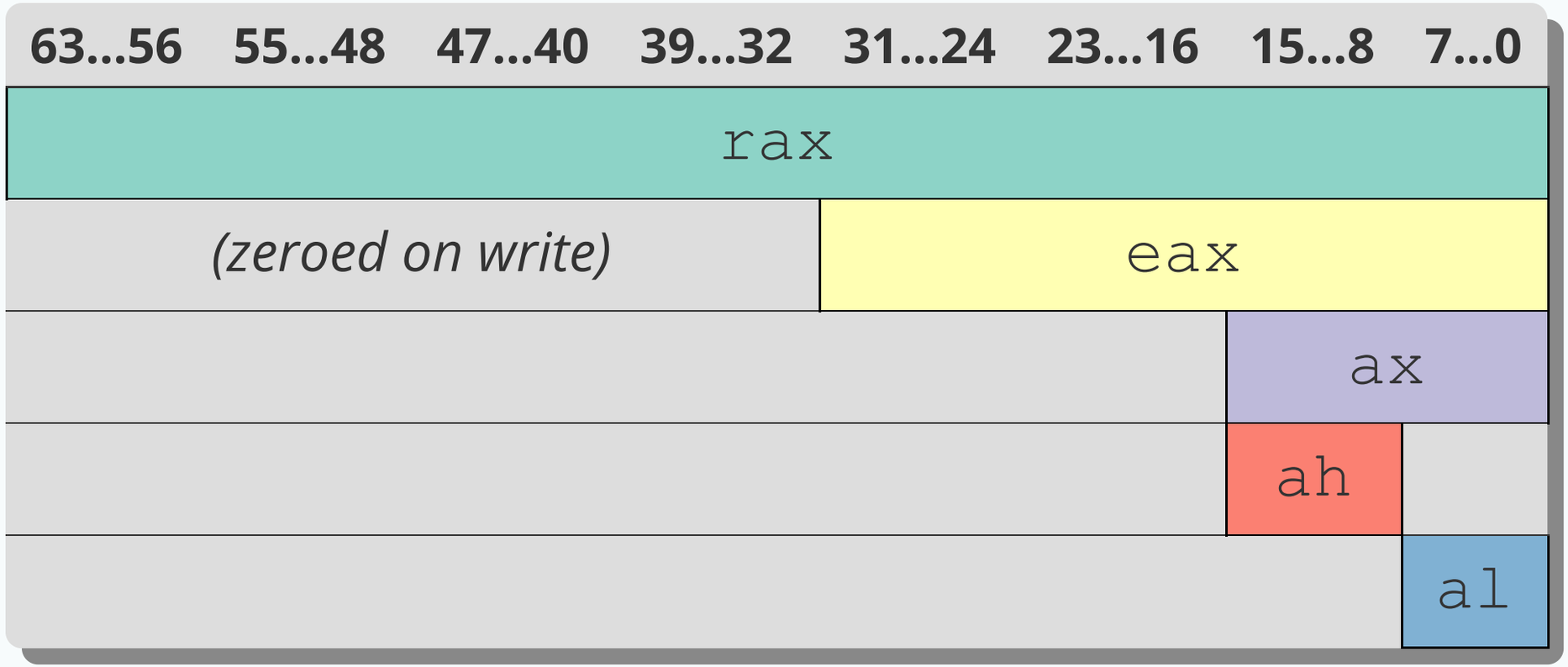
OPERANDS

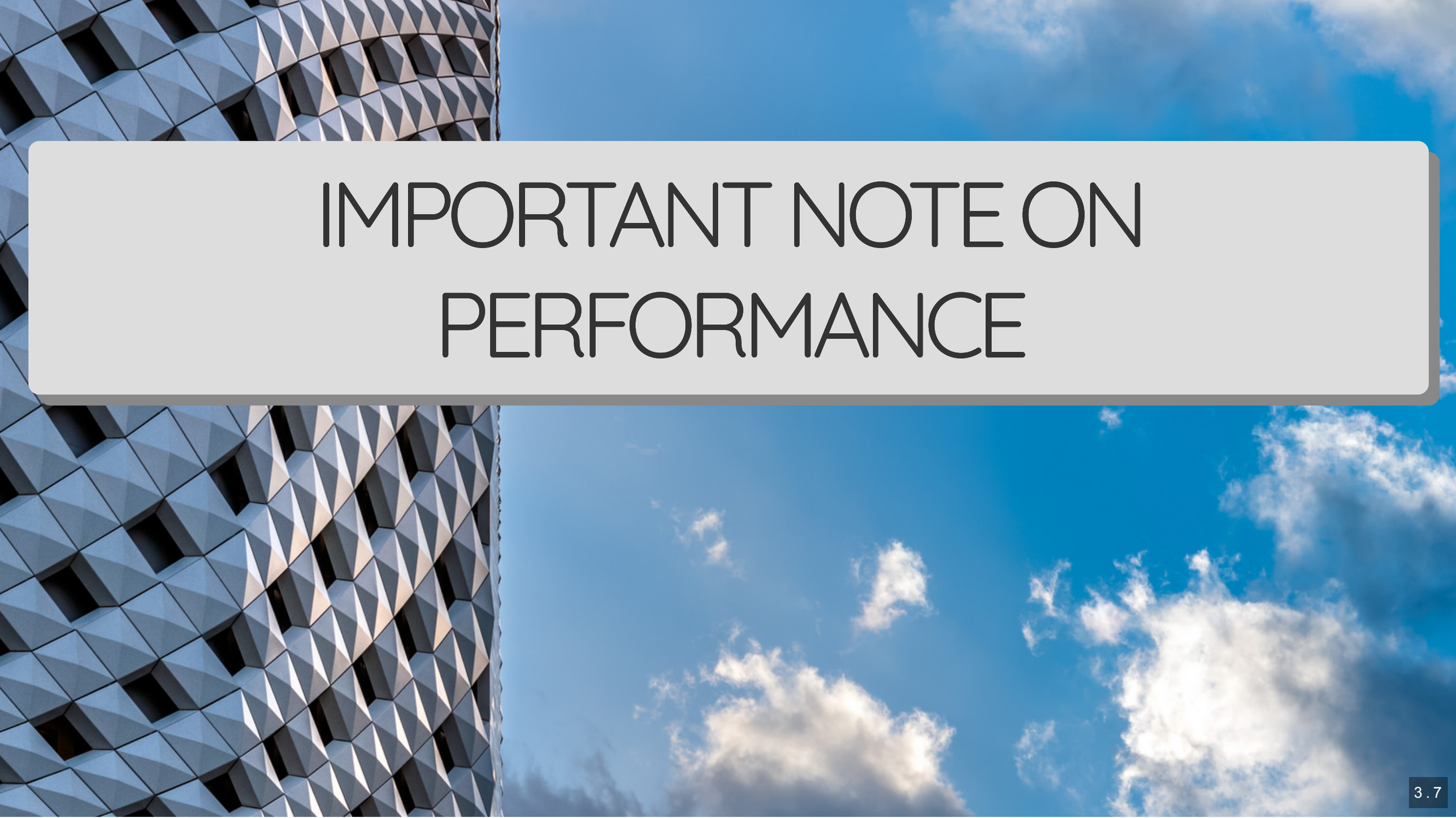
```
register                                    ; e.g. rax, rbx, ecx...
constant                                    ; e.g. 1234
<size> ptr [register]                       ; e.g. DWORD PTR [rax]
<size> ptr [register + offset]              ; e.g. BYTE PTR [rcd + rsi]
<size> ptr [register + offset + register2 * (1,2,4,8)]
```

```
add eax, dword ptr [rdi + 12 + rsi * 4]
                                  ; eax += *(int *)(rdi + 12 + rsi * 4)
```

REGISTERS

- `rax` (return value)
- `rdi` (1st param)
- `rsi` (2nd param)
- `rdx` (3rd param)
- `rbx` `rcx` `rbp` `r8-r15` `rsp`
- `xmm0-15` (`ymm0-15`… `zmm0-31`… `k0-7`)


63...56
55...48
47...40
39...32
31...24
23...16
15...8
7...0
rax
(zeroed on write)
eax
ax
ah
al



IMPORTANT NOTE ON PERFORMANCE

IMPORTANT NOTE ON PERFORMANCE

- Google Benchmark
- hayai
- Nonius
- Quick Bench

TIDBITS!

MATH TIDBITS!

```
int test(int x, int y)
{
  return x - y;
}
```

```
int mulBy65599(int a)
{
  return (a << 16) + (a << 6) - a;
  //        ^           ^
  //     a * 65536      |
  //                 a * 64
  // 65536a + 64a - 1a = 65599a
}
```

```
int divideBy16(int a)
{
  return a >> 4;  // clearly can't trust the compiler to do this for me
}
```

```
unsigned sumUpTo(unsigned x)
{
  auto total = 0u;
  for (auto i = 0u; i < x; ++i)
  {
    total += i;
  }
  return total;
}
```

```
unsigned sumUpTo(unsigned x)
{
  // equivalent to x(x+1)/2
  return ((x - 1) * (x - 2) / 2) + x - 1;
}
```

VECTORIZATION TIDBITS!

```
int sumSquared(const vector<int> &v)
{
  int res = 0;
  for (auto i : v)
  {
    res += i * i;
  }
  return res;
}
```

```
int res_[] = {0,0,0,0,0,0,0,0};
for (; index < v.size();
     index += 8)
{
  for (size_t j = 0; j < 8; ++j)
  {  // but this happens in one instruction...
    auto val = v[index + j];
    res_[j] += val * val;
  }
}
res = res_[0] + res_[1]
    + res_[2] + res_[3]
    + res_[4] + res_[5]
    + res_[6] + res_[7];
```

```
template<typename T>
auto sumSquared(const T &v)
{
  return std::accumulate(
      begin(v), end(v),
      typename T::value_type(),
      [](auto x, auto y)
      {
        return x + y * y;
      });
}
```

ARCHITECTURAL TIDBITS!

```
int countSetBits(unsigned a)
{
  int count = 0;
  while (a != 0)
  {
    count++;
    // clear bottom set bit
    a &= (a - 1);
  }
  return count;
}
```

```
uint32_t switchBits(uint32_t x)
{
  auto first = x & 0xff;
  auto second = (x >> 8) & 0xff;
  auto third = (x >> 16) & 0xff;
  auto fourth = (x >> 24) & 0xff;
  return
      (first << 24)
    | (second << 16)
    | (third << 8)
    | fourth;
}
```

```
bool isspc(char c)
{
    return c == ' '
        || c == '\r'
        || c == '\n'
        || c == '\t';
}
```

```
movabs rax, 0x100002600  ; 0b1000000000000000001001100000000
                         ;   ^                   ^  ^^
                         ;   +- bit 32           |  ++- b10 & 9
                         ;                 bit 13 -+
shrx rax, rax, rdi       ; shift right 'rdi' times
and eax, 1               ; pick the lowest set bit
```

FUNCTION CALL TIDBITS!

```
unsigned identity(unsigned num); // just returns "num"

unsigned sumUpTo(unsigned x)
{
  auto total = 0u;
  for (auto i = 0u; i < x; ++i)
  {
    total += identity(i);
  }
  return total;
}
```

```
struct Func
{
  int operator()(int x)
  {
    return x;
  }
};

unsigned sumUpTo(unsigned x, Func &func)
{
  auto total = 0u;
  for (auto i = 0u; i < x; ++i)
    total += func(i);
  return total;
}
```

CONCLUSION

CONCLUSION

- Compilers are cleverer than we are

CONCLUSION

- Compilers are cleverer than we are
  - if they have the right information

CONCLUSION

- Compilers are cleverer than we are
  - if they have the right information
- Assembly isn't *that* scary

CONCLUSION

- Trust your compiler
- Don’t compromise readability

CONCLUSION

- Trust your compiler
- Don’t compromise readability
- Be aware of compiler limitations:
  - Visibility
  - (and aliasing)

THANKS

Compiler Explorer is at https://godbolt.org/

Thanks for listening!

Thanks to the Compiler Explorer implementors.

THANKS

Compiler Explorer is at https://godbolt.org/

Thanks for listening!

Thanks to the Compiler Explorer implementors.

GO READ SOME ASSEMBLY!