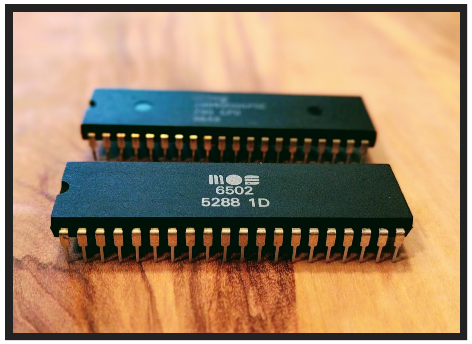
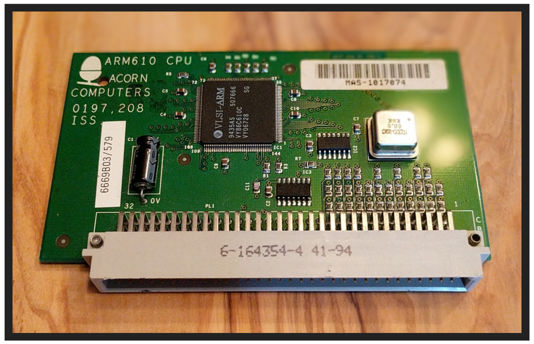
WHAT HAS MY COMPILER DONE FOR ME LATELY?

UNBOLTING THE COMPILER'S LID

Matt Godbolt, DRW Trading
@mattgodbolt matt@godbolt.org

ABOUT ME


6502
5288 1D

ARM610 CPU
ACORN
COMPUTERS
0197,208
ISS
VLSI ARM
6-164354-4 41-94

RED DOG
SUPERIOR FIREPOWER
SEGA

MY GOALS

- Un-scary-fy assembler
- Appreciate your compiler!

OUTLINE

- Compiler Explorer story
- Assembly 101
- What has my compiler done for me lately?
- Behind the scenes of Compiler Explorer

BACKSTORY

```
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0; i < v.size(); ++i)
    result += v[i];
  return result;
}
```

```
int sum(const vector<int> &v) {
  int result = 0;
  for (int x : v) result += x;
  return result;
}
```

Is one better than the other?

WARNING

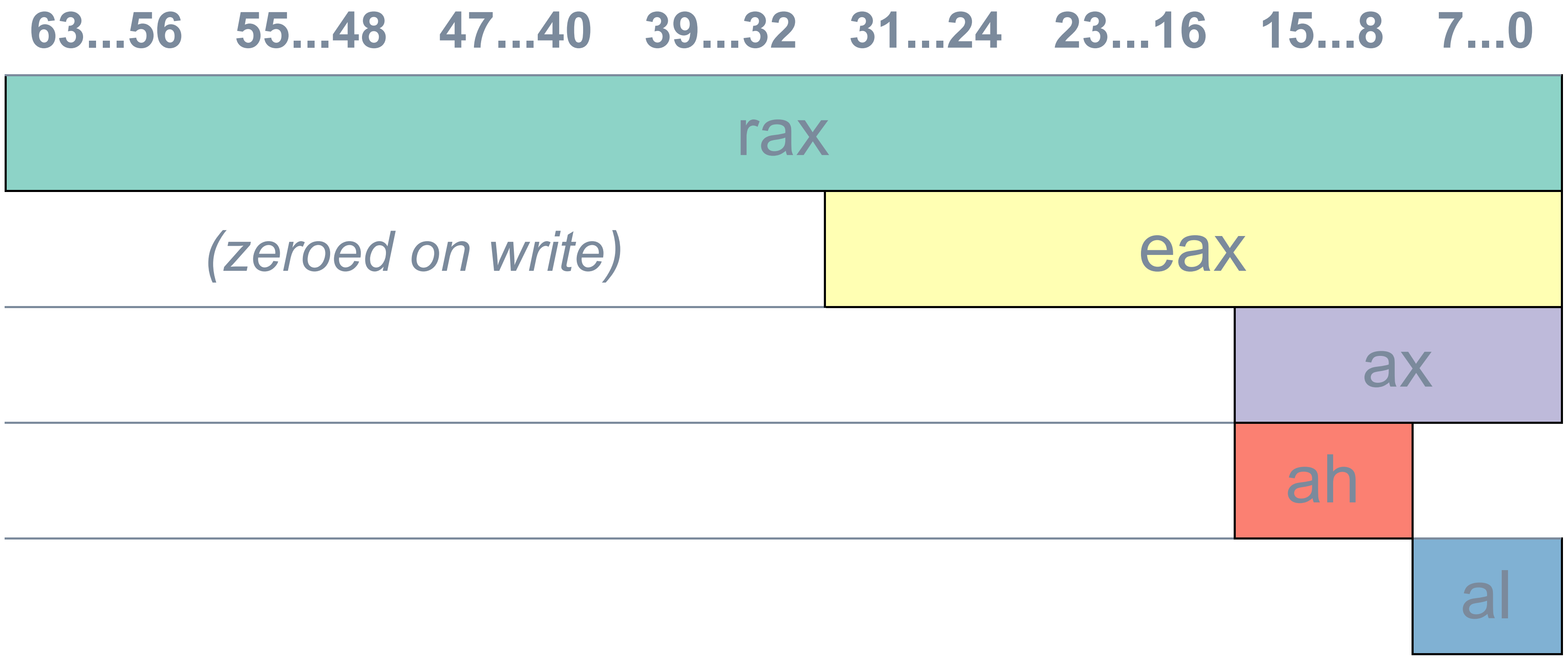
- Reading assembly alone can be misleading
- *Always* measure too
- Google Benchmark
- quick-bench.com

X86 ASSEMBLY 101

REGISTERS

- `rax`, `rbx`, `rcx`, `rdx`, `rsp`, `rbp`, `rsi`, `rdi`, `r8`-`r15`
- `xmm0-xmm15`
- `rdi`, `rsi`, `rdx`... arguments
- `rax` is return value

REGISTERS


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

INSTRUCTIONS

```
op
op dest
op dest, src
op dest, src1, src2
```

- op is e.g. `call`, `ret`, `add`, `sub`, `cmp`...
- `dest`, `src` is register or memory reference:

  `[base + reg1`$_{opt}$ `+ reg2*(1, 2, 4 or 8)`$_{opt}$`]`

(Intel asm syntax)

INSTRUCTIONS

```
mov eax, DWORD PTR [r14]
add rax, rdi
add eax, DWORD PTR [r14+4]
sub eax, DWORD PTR [r14+4*rbx]
lea rax, [r14+4*rbx]
xor edx, edx
```

```
int eax = *r14;    // int *r14;
rax += rdi;
eax += r14[1];
eax -= r14[rbx];
int *rax = &r14[rbx];
edx = 0;
```

SUMMARY

- Registers: `rax`, `rbx`, `rcx` ...
- Size: `rax`, `eax`, `ax` ...
- Params: `rdi`, `rsi`, `rdx`, `rcx` ...
- Result: `rax`
- `op dest, src`
- `dest`, `src` are registers or memory

WHERE WERE WE?

```
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0; i < v.size(); ++i)
    result += v[i];
  return result;
}
```

```
int sum(const vector<int> &v) {
  int result = 0;
  for (int x : v) result += x;
  return result;
}
```

Which is better?

COMPILER EXPLORER V0.1

```
$ g++ /tmp/test.cc -O2 -c -S -o - -masm=intel \
    | c++filt \
    | grep -vE '\s+\.'
```

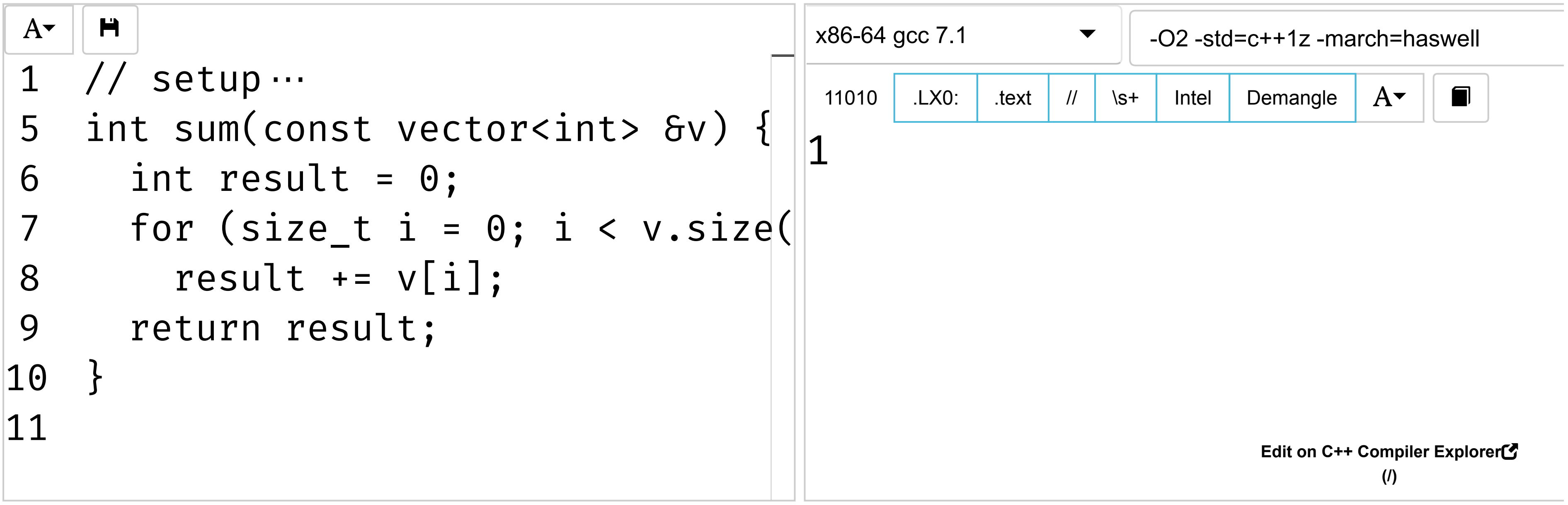
```
sum(std::vector<int, std::allocator<int> > const&):
.LFB786:
    mov rcx, QWORD PTR [rdi]
    mov rax, QWORD PTR 8[rdi]
    sub rax, rcx
    shr rax, 2
    mov rsi, rax
    ...
```

COMPILER EXPLORER V0.1

Not very pretty

TO THE WEB!

DEMO



```
// setup…
int sum(const vector<int> &v) {
  int result = 0;
  for (size_t i = 0; i < v.size(
    result += v[i];
  return result;
}

```

x86-64 gcc 7.1

-O2 -std=c++1z -march=haswell

```
1
```

Edit on C++ Compiler Explorer
(/)

WALKTHROUGH

```
; rdi = const vector<int> *
mov rdx, QWORD PTR [rdi]   ; rdx = *rdi       ≡ begin()
mov rcx, QWORD PTR [rdi+8] ; rcx = *(rdi+8)  ≡ end()
```

```
template<typename T> struct _Vector_impl {
  T *_M_start;
  T *_M_finish;
  T *_M_end_of_storage;
};
```

TRADITIONAL

```
sub rcx, rdx ; rcx = end-begin
mov rax, rcx
shr rax, 2    ; (end-begin)/4
je .L4
add rcx, rdx
xor eax, eax
```

```
size_t size() const noexcept {
  return _M_finish - _M_start;
}
```

RANGE

```
xor eax, eax
cmp rdx, rcx ; begin==end?
je .L4
```

```
auto __begin = begin(v);
auto __end = end(v);
for (auto __it = __begin;
     __it != __end;
     ++it)
```

WALKTHROUGH

```
; rcx ≡ end, rdx = begin, eax = 0
.L3:
  add eax, DWORD PTR [rdx]    ; eax += *rdx
  add rdx, 4                  ; rdx += sizeof(int)
  cmp rdx, rcx                ; is rdx == end?
  jne .L3                     ;   if not, loop
  ret                         ; we're done
```

BACKSTORY

SO, WHICH APPROACH IS BEST?

ALSO

- Optimizer settings are important
- `std::accumulate`

WHAT HAS MY COMPILER DONE FOR ME LATELY?

MULTIPLICATION

```c
int mulByY(int x, int y) {
  return x * y;
}
```

View

```asm
mulByY(int, int):
  mov eax, edi
  imul eax, esi
  ret
```

MULTIPLICATION

```
        1101    (13)
      x 0101     (5)
    ---------
        1101
       0000
      1101
   + 0000
    ---------
    01000001    (65)
```

That's a lot of additions!

Haswell 32-bit multiply - 4 cycles

MULTIPLICATION

```
int mulByConstant(int x) { return x * 2; }
```

View

MULTIPLICATION

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

View

DIVISION

```
int divByY(int x, int y) {
  return x / y;
}
int modByY(int x, int y) {
  return x % y;
}
```

View

Haswell 32-bit divide - 22-29 cycles!

```
divByY(int, int):
  mov eax, edi
  cdq
  idiv esi
  ret
modByY(int, int):
  mov eax, edi
  cdq
  idiv esi
  mov eax, edx
  ret
```

DIVISION

```
unsigned divByConstant(unsigned x) { return x / 2; }
```

View

DIVISION

```
mov eax, edi          ; eax = x
mov edx, 0xaaaaaaab
mul edx               ; eax:edx = x * 0xaaaaaaab
mov eax, edx          ;   (x * 0xaaaaaaab) >> 32
                      ; ≡ (x * 0xaaaaaaab) / 0x10000000
                      ; ≡ x * 0.6666666667
shr eax               ; x * 0.333333333
ret
```

MODULUS

```
int modBy3(unsigned x) {
  return x % 3;
}
```

View

```
mov eax, edi
mov edx, 0xaaaaaaab
mul edx
mov eax, edx
shr eax
lea eax, [rdx+rdx*2]
sub edi, eax
mov eax, edi
ret
```

WHY MODULUS?

- Bucket selection in hash maps
- libc++ special-cases power-of-two
- boost multi_index

COUNTING BITS

```
int countSetBits(int a) {
  int count = 0;
  while (a) {
    count++;
    a &= (a-1);
  }
  return count;
}
```

View

SUMMATION

```cpp
constexpr int sumTo(int x) {
  int sum = 0;
  for (int i = 0; i <= x; ++i)
    sum += i;
  return sum;
}
int main(int argc, const char *argv[]) {
  return sumTo(20);
}
```

View

SUM(X)

$$\sum x \equiv \frac{x(x+1)}{2}$$

$$\equiv x + \frac{x(x-1)}{2}$$

WHAT HAS MY COMPILER DONE FOR ME LATELY?

A lot!

HOW IT WORKS

HOW IT WORKS - BACKEND

- Written in `node.js`
- Runs on Amazon

NODE.JS

```
function compile(req, res, next) {
  // exec compiler, feed it req.body, parse output
}
var webServer = express();
var apiHandler = express.Router();
apiHandler.param('compiler',
    function (req, res, next, compiler) {
  req.compiler = compiler;
  next();
});
apiHandler.post('/compiler/:compiler/compile', compile);
webServer.use('/api', apiHandler);
webServer.listen(10240);
```

AMAZON EC2

- Edge cache
- Load balancer
- Virtual machines
- Docker images
- Shared compiler storage

THE COMPILERS

- Built through docker images
- Compilers stored on S3
- OSS ones publically available
- MS compilers via WINE

HOW IT WORKS - SECURITY

- Compilers: huge attack vector
- Principle of "what's the worst could happen"
- Docker
- `LD_PRELOAD`

HOW IT WORKS - FRONTEND

- Microsoft's Monaco
- GoldenLayout

THE CODE

- github.com/mattgodbolt/compiler-explorer
- github.com/mattgodbolt/compiler-explorer-image
- Running locally is easy!

```
$ make
```

- More in next C++ Weekly

OTHER USES

- Code pastebin
- Compiler development
- C++ REPL
- Training resource

COMING SOON...

- CFG viewer
- Unified languages
- Execution support

THANKS

- Thanks to contributors:
  - Rubén Rincón
  - Gabriel Devillers
  - Simon Brand, Johan Engelen, Jared Wyles, Chedy Najjar
  - ...and the rest!
- Thanks to Patreon folks
- Thanks to awesome C++ community
- Thanks to you!

GO READ SOME ASSEMBLY!

{ gcc , cppx , d , swift , haskell , go , ispc }.godbolt.org

(AND THANK A COMPILER DEVELOPER)