John McFarlane

Signed Integers:

John McFarlane

Signed Integers:

Faster *and* Correcter

Background

Background

- `unsigned`: A Guideline for Better Code (Jon Kalb, CppCon 2016)

Background

- `unsigned`: A Guideline for Better Code (Jon Kalb, CppCon 2016)
- Garbage In, Garbage Out: Arguing about Undefined Behavior... (Chandler Carruth, CppCon 2016)

Background

- `unsigned`: A Guideline for Better Code (Jon Kalb, CppCon 2016)
- Garbage In, Garbage Out: Arguing about Undefined Behavior... (Chandler Carruth, CppCon 2016)
- Random thoughts on programming languages, compilers, operating systems, etc. (Krister Walfridsson's blog, 2016-02-21)

Krister Walfridsson's blog

Random thoughts on programming languages, compilers, operating systems, etc.

Sunday, February 21, 2016

How undefined signed overflow enables optimizations in GCC

Signed integers are not allowed to overflow in C and C++, and this helps compilers generate better code. I was interested in how GCC is taking advantage of this, and here are my findings.[1]

Signed integer expression simplification

The nice property of overflow being undefined is that signed integer operations works as in normal mathematics — you can cancel out values so that `(x*10)/5` simplifies to `x*2`, or `(x+1)<(y+3)` simplifies to `x<(y+2)`. Increasing a value always makes it larger, so `x<(x+1)` is always true.

GCC iterates over the IR (the compiler's Internal Representation of the program), and does the following transformations (`x`, and `y` are signed integers, `c`, `c1`, and `c2` are positive constants, and `cmp` is a comparison operator. I have only listed the transformations for positive constants, but GCC handles negative constants too in the obvious way)

- Eliminate multiplication in comparison with 0

  `(x * c) cmp 0 -> x cmp 0`

- Eliminate division after multiplication

  `(x * c1) / c2 -> x * (c1 / c2)` if `c1` is divisible by `c2`

- Eliminate negation

  `(-x) / (-y) -> x / y`

- Simplify comparisons that are always true or false

Contracts in C++

Contracts in C++

```
void do_something(Object const& thing);
```

Contracts in C++

```
void do_something(Object const& thing);
```

Question : Is `const` liberating or restrictive?

Contracts in C++

```
void do_something(Object const& thing);
```

Question : Is `const` liberating or restrictive?

Answer : It depends.

Contracts in C++

```
void do_something(Object const& thing);
```

Question : Is `const` liberating or restrictive?

Answer : It depends.

Implementor : "It's restrictive!"

Contracts in C++

```
void do_something(Object const& thing);
```

Question : Is `const` liberating or restrictive?

Answer : It depends.

Implementor : "It's restrictive!"

User : "It's liberating!"

Narrow Contracts in C++

```
void visit_something(std::function<void(Object const&)>);
```

Narrow Contracts in C++

```
void visit_something(std::function<void(Object const&)>);
```

```
visit_something([](Object const& thing) {
    const_cast<Object&>(thing) = Object{};  // bad idea
});
```

Narrow Contracts in C++

```
void visit_something(std::function<void(Object const&)>);
```

```
visit_something([](Object const& thing) {
    const_cast<Object&>(thing) = Object{};  // bad idea
});
```

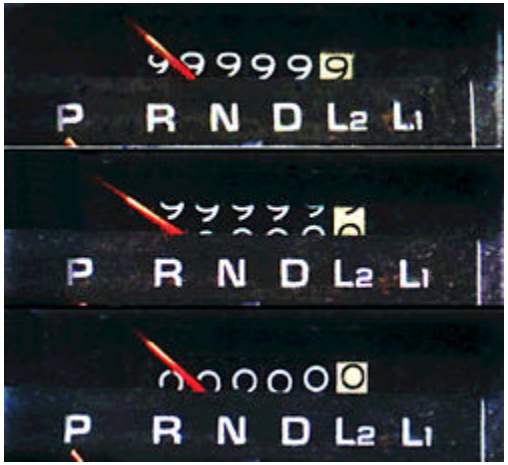
How does ↑ differ from ↓?

```
auto i = numeric_limits<int>::max();
++ i;   // very bad idea
```

Defined but Wrong

```
// error: static_assert expression is not an integral constant expression
static_assert(numeric_limits<signed>::max()+2);

// totally fine
static_assert(numeric_limits<unsigned>::max()+2);
```



If you're arguing...

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```cpp
bool example4a(T x) {
    T c = 1;
    return x + c < x;
}

bool example4b(T x) {
    T c = 1;
    return x + c <= x;
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 | -O3 -DT=signed

11010 .LX0: .text // \s+ Intel Demangle

```asm
example4a(int):
        xor     eax, eax
        ret
example4b(int):
        xor     eax, eax
        ret
```

https://gcc.godbolt.org/# Explorer-Build) 7.1.0- cached

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 | -O3 -DT=unsigned

11010 .LX0: .text // \s+ Intel Demangle

```asm
example4a(unsigned int):
        cmp     edi, -1
        sete    al
        ret
example4b(unsigned int):
        cmp     edi, -1
        sete    al
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- cached

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```
T example2(T x) {
  return (x * 14) / 7;
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 | -O3 -DT=signed

11010 .LX0: .text // \s+ Intel Demangle

```
example2(int):
        lea     eax, [rdi+rdi]
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 | -O3 -DT=unsigned

11010 .LX0: .text // \s+ Intel Demangle

```
example2(unsigned int):
        imul    edi, edi, 14
        mov     edx, 613566757
        mov     eax, edi
        mul     edx
        sub     edi, edx
        shr     edi
        lea     eax, [rdx+rdi]
        shr     eax, 2
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```cpp
bool example1(T x) {
  T c = 5;
  return (x * c) > 0;
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 | -O3 -DT=signed

11010 .LX0: .text // \s+ Intel Demangle

```asm
example1(int):
        test    edi, edi
        setg    al
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 | -O3 -DT=unsigned

11010 .LX0: .text // \s+ Intel Demangle

```asm
example1(unsigned int):
        lea     eax, [rdi+rdi*4]
        test    eax, eax
        setne   al
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```cpp
#include <cstdlib>
T example9d(T x) {
  if (x > 0) {
    return abs(x);
  }
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 | -O3 -DT=signed

11010 .LX0: .text // \s+ Intel Demangle

```asm
example9d(int):
        test    edi, edi
        jle     .L2
        mov     eax, edi
        ret
.L2:
        rep ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 | -O3 -DT=unsigned

11010 .LX0: .text // \s+ Intel Demangle

```asm
example9d(unsigned int):
        test    edi, edi
        je      .L2
        mov     eax, edi
        sar     eax, 31
        xor     edi, eax
        sub     edi, eax
        mov     eax, edi
        ret
.L2:
        rep ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```cpp
bool example8(float * a, T i) {
  float * e0 = a+i;
  float * e1 = a+(i+1);
  return e1 - e0 == 1;
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 | -O3 -DT=signed

11010 .LX0: .text // \s+ Intel Demangle

```asm
example8(float*, int):
        mov     eax, 1
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 | -O3 -DT=unsigned

11010 .LX0: .text // \s+ Intel Demangle

```asm
example8(float*, unsigned int):
        lea     eax, [rsi+1]
        mov     esi, esi
        sub     rax, rsi
        sal     rax, 2
        cmp     rax, 4
        sete    al
        ret
```

g++ (GCC-Explorer-Build) 7.1.0- *cached*

Compiler Explorer C++ Editor Diff View More Share Other

C++ source #1

```
T example10(T m) {
  int sum = 0;
  for (short T i = 0; i <= m; ++ i) {
    sum += i;
  }
  return sum;
}
```

x86-64 gcc 7.1 (Editor #1, Compiler #1)

x86-64 gcc 7.1 -O3 -DT=signed

```
example10(int):
        test    edi, edi
        js      .L4
        xor     ecx, ecx
        xor     eax, eax
        xor     edx, edx
.L3:
        add     ecx, 1
        add     eax, edx
        movsx   edx, cx
        cmp     edx, edi
        jle     .L3
        rep ret
.L4:
        xor     eax, eax
        ret
```

g++ (GCC-Explorer-Build) 7.1.0-cached

x86-64 gcc 7.1 (Editor #1, Compiler #2)

x86-64 gcc 7.1 -O3 -DT=unsigned

```
example10(unsigned int):
        lea     eax, [rdi-12]
        cmp     eax, 65522
        ja      .L6
        lea     ecx, [rdi+1]
        pxor    xmm0, xmm0
        pxor    xmm2, xmm2
        xor     eax, eax
        mov     edx, ecx
        movdqa  xmm1, XMMWORD PTR .LC0[rip]
        shr     edx, 3
        movdqa  xmm4, XMMWORD PTR .LC1[rip]
.L3:
        movdqa  xmm3, xmm1
        add     eax, 1
        cmp     eax, edx
        punpcklwd       xmm3, xmm2
        paddd   xmm0, xmm3
        movdqa  xmm3, xmm1
        paddw   xmm1, xmm4
        punpckhwd       xmm3, xmm2
        paddd   xmm0, xmm3
        jb      .L3
        movdqa  xmm1, xmm0
        mov     edx, ecx
        and     edx, -8
        psrldq  xmm1, 8
        paddd   xmm0, xmm1
        movdqa  xmm1, xmm0
        cmp     ecx, edx
        psrldq  xmm1, 4
        paddd   xmm0, xmm1
        movd    eax, xmm0
        je      .L5
        lea     ecx, [rdx+1]
        add     eax, edx
        movzx   ecx, cx
        cmp     edi, ecx
        jb      .L5
        add     eax, ecx
        lea     ecx, [rdx+2]
        movzx   ecx, cx
        cmp     edi, ecx
        jb      .L5
        add     eax, ecx
        lea     ecx, [rdx+3]
        movzx   ecx, cx
        cmp     edi, ecx
        jb      .L5
        add     eax, ecx
        lea     ecx, [rdx+4]
        movzx   ecx, cx
        cmp     edi, ecx
        jb      .L5
        add     eax, ecx
        lea     ecx, [rdx+5]
        movzx   ecx, cx
        cmp     edi, ecx
        jb      .L5
```

g++ (GCC-Explorer-Build) 7.1.0-cached

Notes

- My knowledge of assembler gets a little hazy after 6502.
- Fewer instructions does *not* mean faster code.
- Some 'fine tuning' of details to make some of the results differ between `signed`/`usigned`.
- I'll bet some `unsigned` optimizations are pretty impressive too so...
- Go and find some good counter-examples!

Thank You!

Special Thanks to Krister Walfridsson & Bay Area ACCU

`c/c++ == 1`

```
(signed) John McFarlane
-----------------------
```

@JSAMcFarlane