Writing LLVM Optimization

SWPP

Apr. 9th

Juneyoung Lee

Doing Your Homework

- As you have noticed, materials of LLVM isn't very available on the Internet.
- For your homework, using the methods described in 3.materials will be sufficient
- But.. you may want to:
  - Find a better way to do the homework
  - Find what to do when I cannot help you anymore (e.g. project..!)

Doing Your Homework

- This is similar to the situation when you're working at company as well.
  - Scenario: you went to Microsoft & are given to fix a bug in Windows
- What you need is to quickly understand what the software looks like
  - Overall structure + Detailed structure for the particular subject (the bug)
- That is why you're taking courses from CSE!
  - If you took OS, you'll understand the structure of Windows much faster.
- Being familiar with tools is important too
  - You can run the sanitizer & see whether the bug reproduces! (it shows line number)

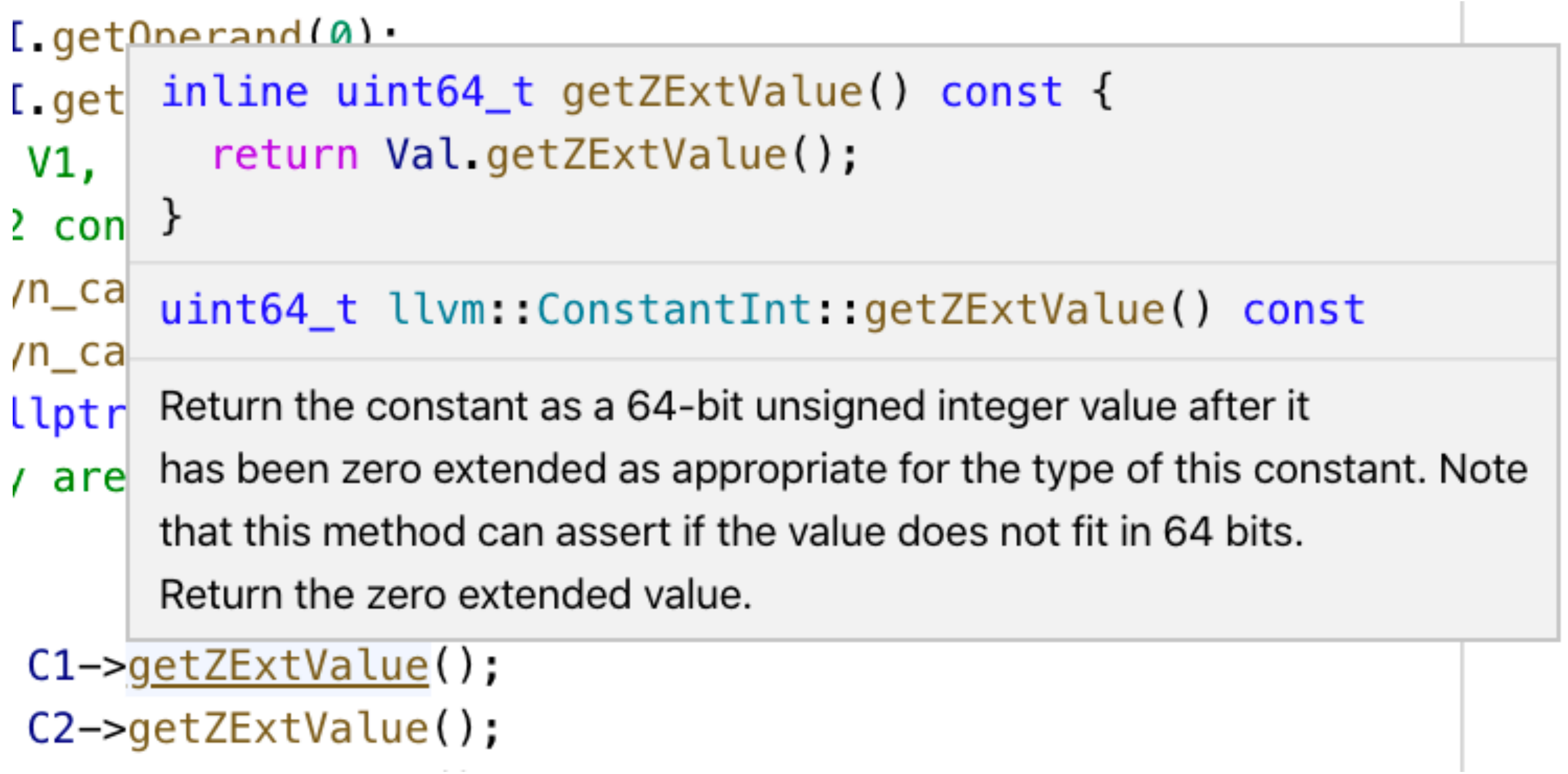
Why LLVM? (in my opinion)

- You can grab the feeling of a ‘well-written program’.
- In parallel, you can understand how to write a safe C/C++ program
- Also, LLVM is used by both software + hardware companies
  - Software company: the speed of application is critical
    - There are three ways to make application faster:
      (1) Use faster algorithm, (2) Use better hardware, (3) Enhance compiler
  - Hardware company: needs a compiler that supports their new chips

How to Understand LLVM

1. Practical Advices

- Use Visual Studio Code's autocompletion & link
  - Put a mouse cursor over methods/classes/etc, It will show you the description of the function!
  - If you press Command Key (on Mac) / Ctrl key (on Linux), it will jump to its definition



```
inline uint64_t getZExtValue() const {
  return Val.getZExtValue();
}
```

```
uint64_t llvm::ConstantInt::getZExtValue() const
```

Return the constant as a 64-bit unsigned integer value after it has been zero extended as appropriate for the type of this constant. Note that this method can assert if the value does not fit in 64 bits.
Return the zero extended value.

```
C1->getZExtValue();
C2->getZExtValue();
```

- Refer to online Doxygen : https://llvm.org/doxygen/classllvm_1_1ConstantInt.html
- Run simple examples using opt : `opt -passes="instcombine,gvn" a.ll -S -o output.ll`
- Buy a book, if you'd like: "Getting Started with LLVM Core Libraries"

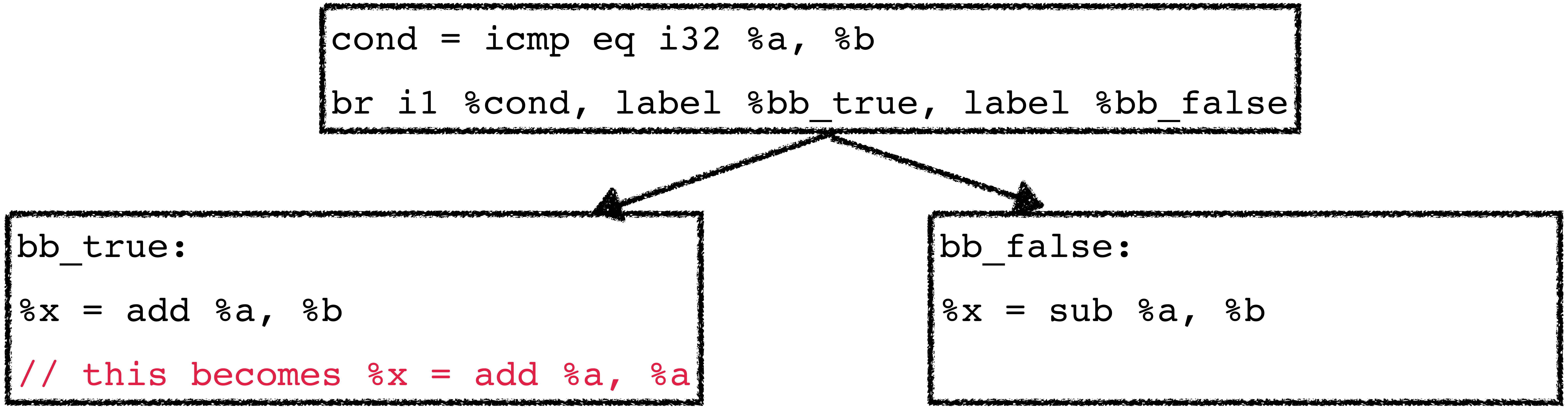
How to Understand LLVM

2. High-level Advices

- Feel free to *explore* LLVM files like InstCombineAddSub.cpp
  - Actually, this is the most powerful solution in the long run
- Try to match the knowledge Professor taught in the class with LLVM code
  - You'll see 'poison' or 'undefined behavior' in many places
- (After starting your project) Freely share your knowledge with teammates
  - Everyone's starting from scratch
  - Good teamwork will make you realize that 1 + 1 = 3 can happen

Assignment 4

- Write a pass that propagates integer equality:



```
cond = icmp eq i32 %a, %b
br i1 %cond, label %bb_true, label %bb_false
```

```
bb_true:
%x = add %a, %b
// this becomes %x = add %a, %a
```

```
bb_false:
%x = sub %a, %b
```

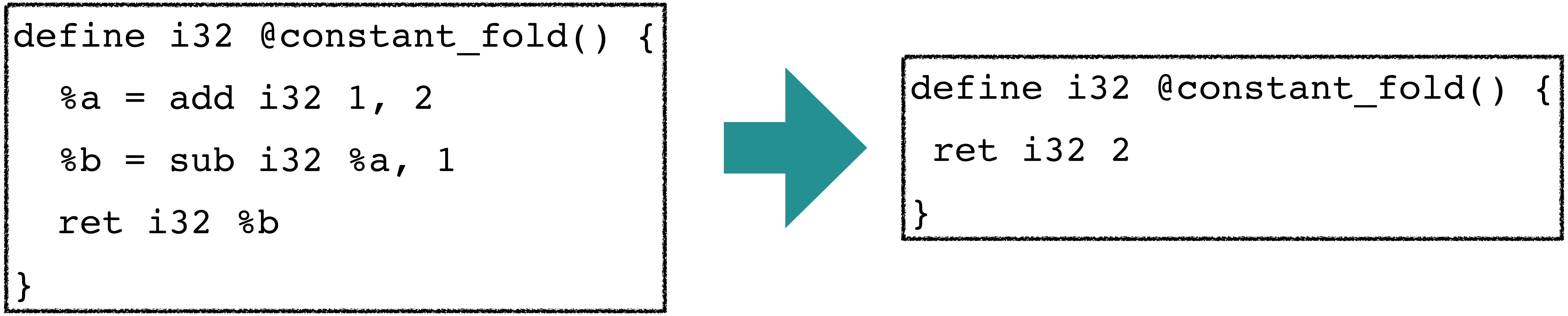
- You will be given many examples so you can refer to
- You'll need to write a test using *FileCheck*.

Assignment 4

- In order to do assn 4, you'll need to build LLVM 10.0.
- Please clone & build it using https://github.com/aqjune/llvmscript
- JSON file: 4.materials/llvm-10.0.json

Simple Optimization: Constant Folding

- my-opt.cpp

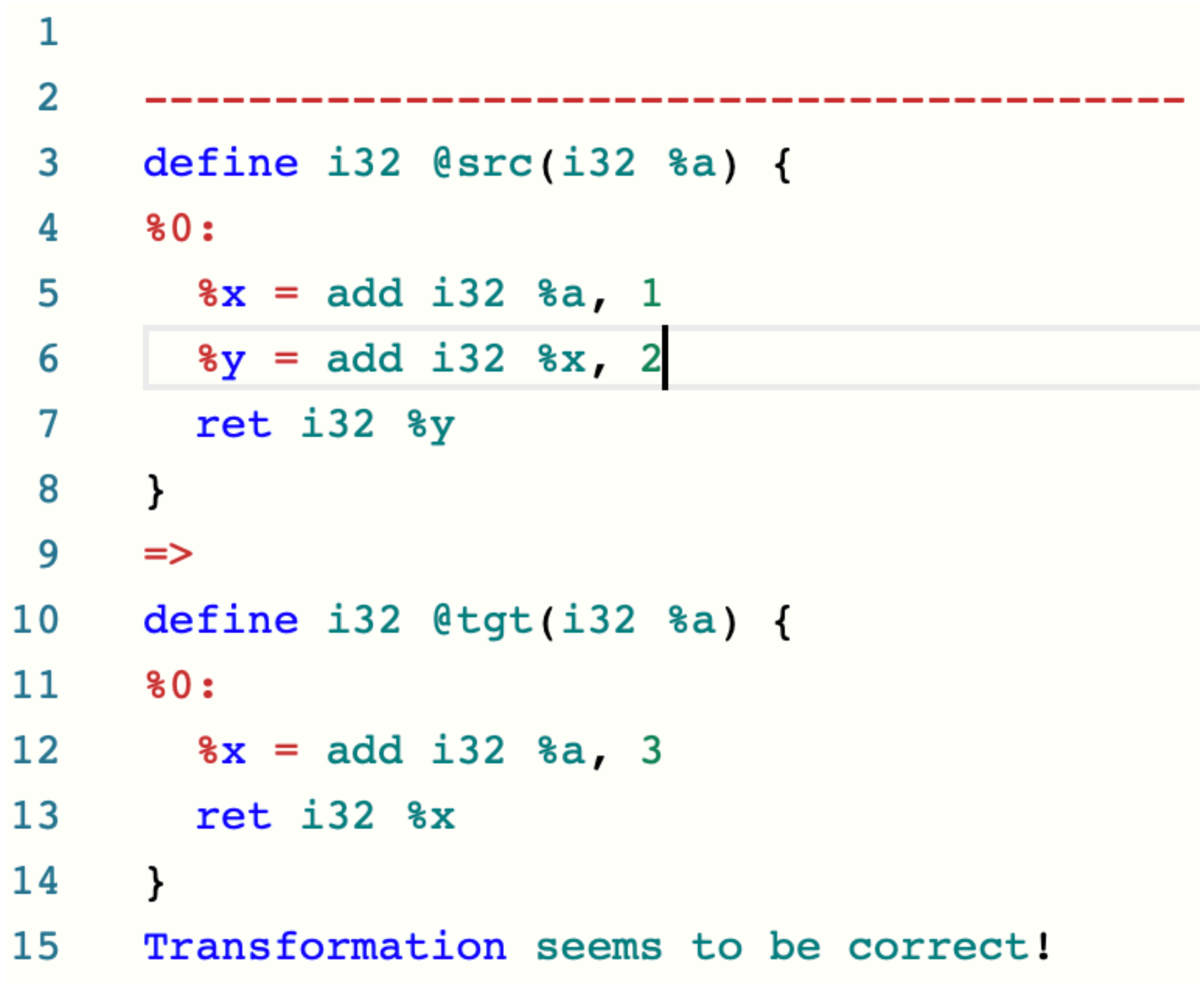


```
define i32 @constant_fold() {
  %a = add i32 1, 2
  %b = sub i32 %a, 1
  ret i32 %b
}
```

```
define i32 @constant_fold() {
 ret i32 2
}
```

How To Test? - 1. Alive2

- Alive2: an automatic LLVM optimization verifier
- http://volta.cs.utah.edu:8080/
- Check whether your input is correctly optimized, using this tool! (not mandatory)



```

----------------------------------------
define i32 @src(i32 %a) {
%0:
  %x = add i32 %a, 1
  %y = add i32 %x, 2
  ret i32 %y
}
=>
define i32 @tgt(i32 %a) {
%0:
  %x = add i32 %a, 3
  ret i32 %x
}
Transformation seems to be correct!
```

How To Test? - 2. FileCheck

- Syntactic check!
- `opt -passes=“my-opt” test.ll -S -o result.ll`
- `FileCheck test.ll < result.ll`

test.ll

```
define i32 @negated_operand(i32 %x) {
; CHECK-LABEL: @negated_operand(
; CHECK-NEXT:    ret i32 0
  %negx = sub i32 0, %x
  %r = add i32 %negx, %x
  ret i32 %r
}
```

Only manually built LLVM will have FileCheck!

./run.sh build <llvm/bin dir>

./run.sh test <llvm/bin dir>