Using LLVM IR

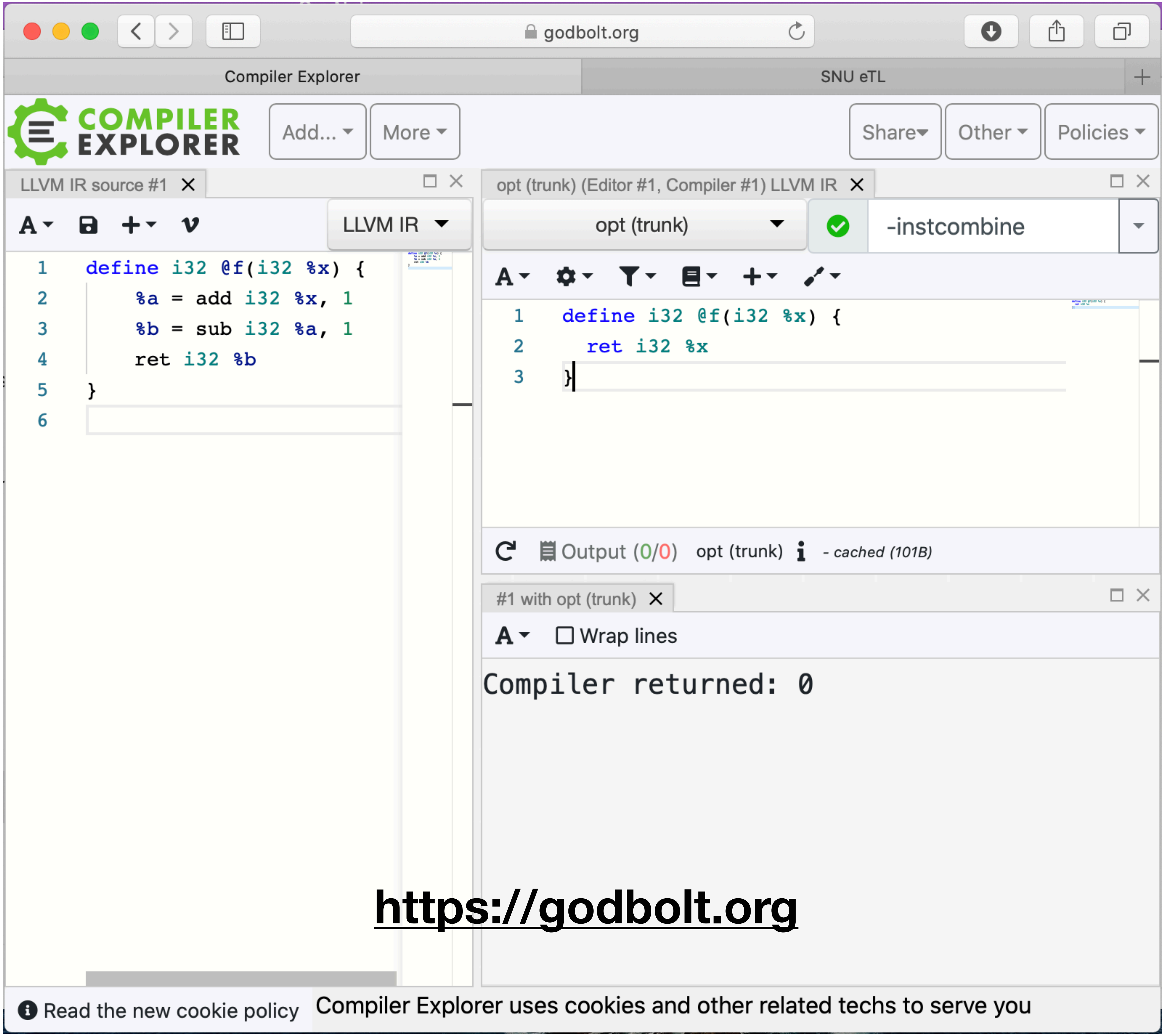
2019. 3. 26

SWPP Practice Session

Juneyoung Lee

Building LLVM

- We’re going to announce Assignment 2 this weekend
- Please build LLVM by following BuildLLVM.md at the class GitHub repo!
- Let us know if there are any issues



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

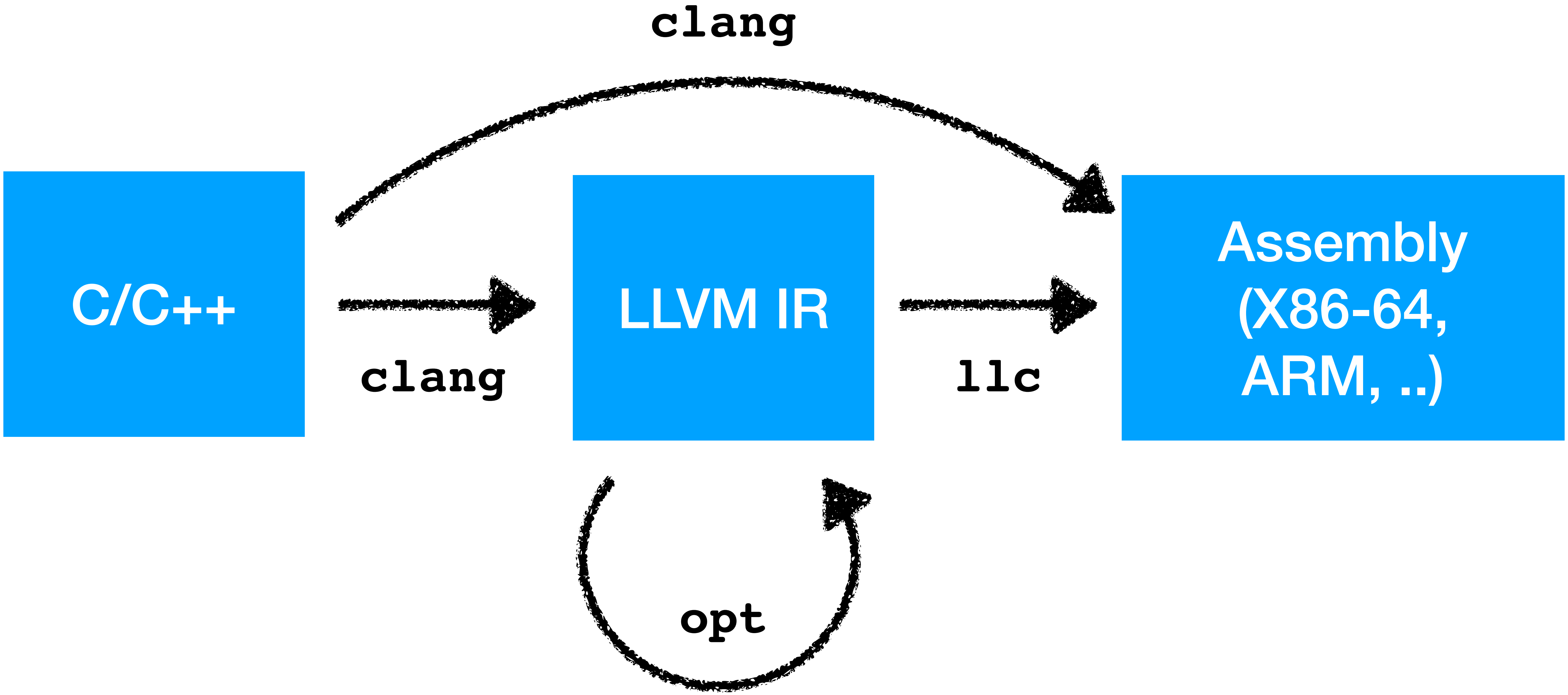
-instcombine

```
define i32 @f(i32 %x) {
  ret i32 %x
}
```

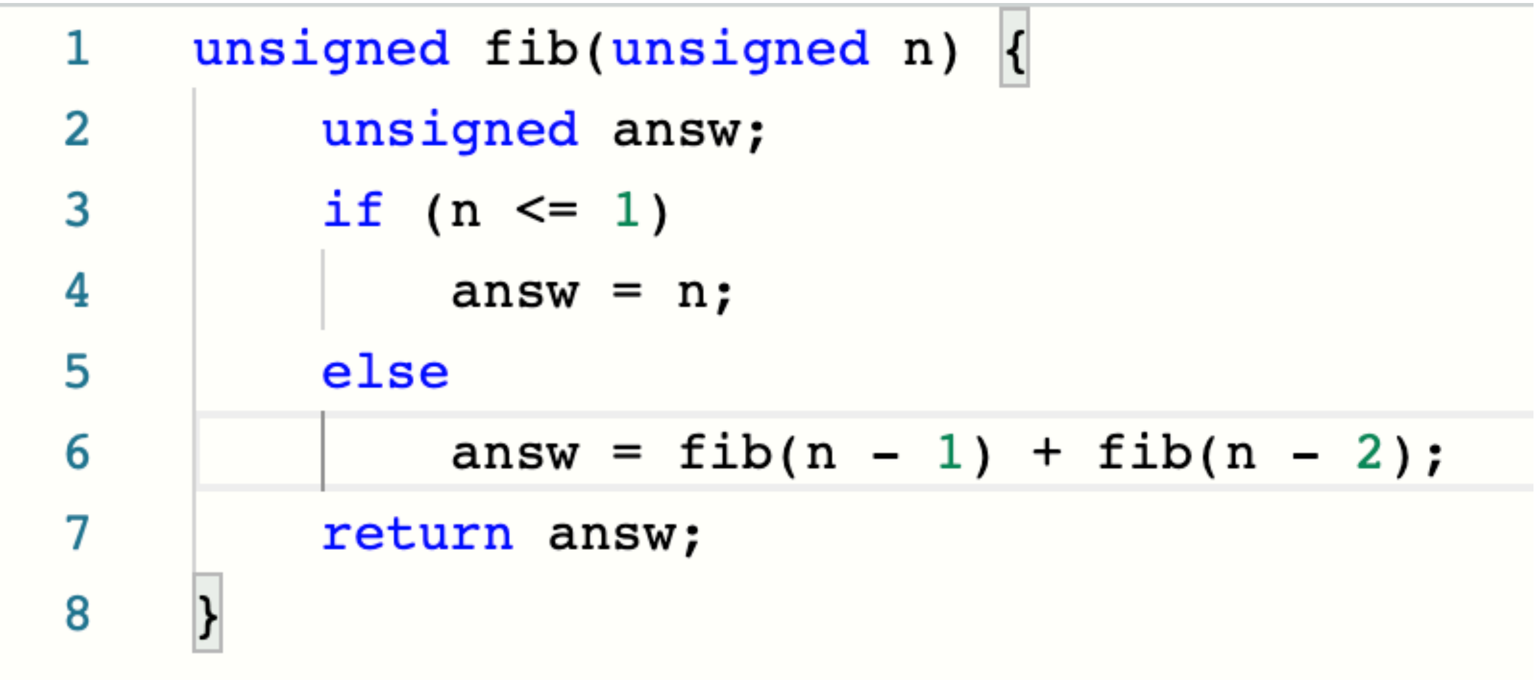
```
Compiler returned: 0
```

https://godbolt.org

Converting LLVM IR from/to *



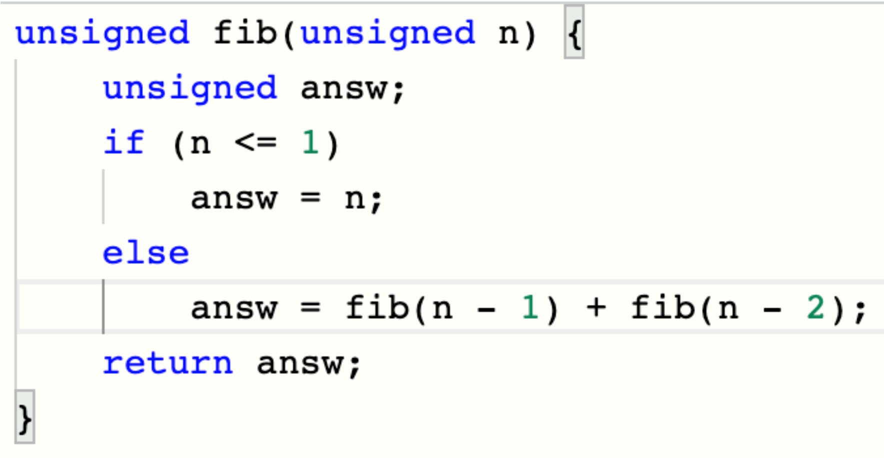
Example - fibonacci



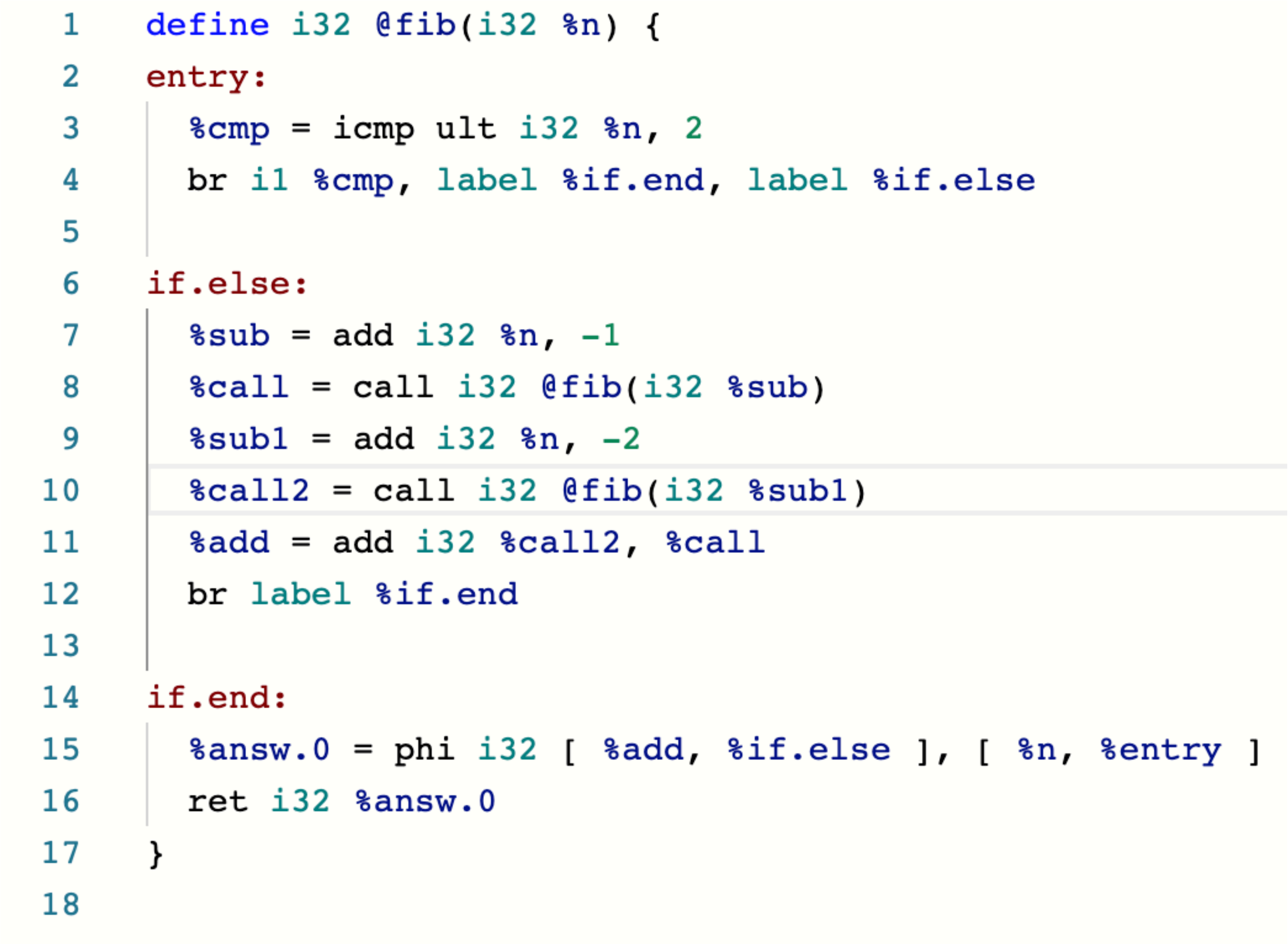
```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

https://godbolt.org/z/QkuFvZ

Example - fibonacci



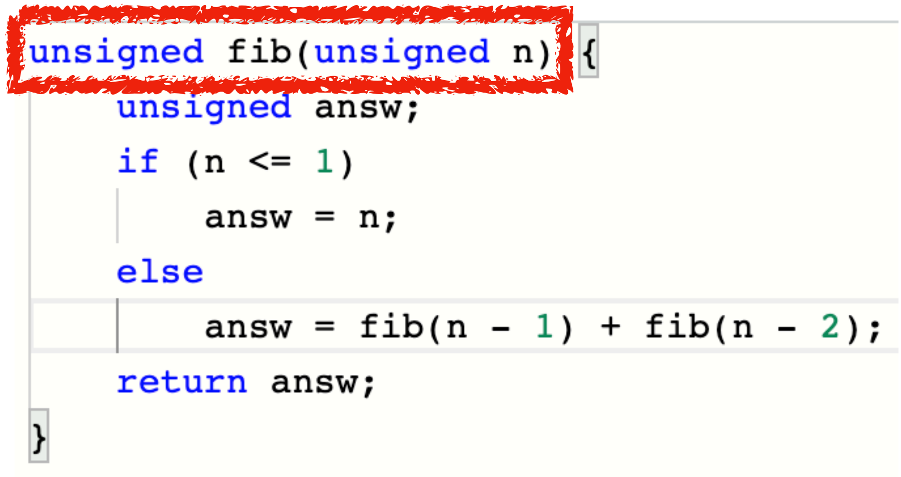
```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```



```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

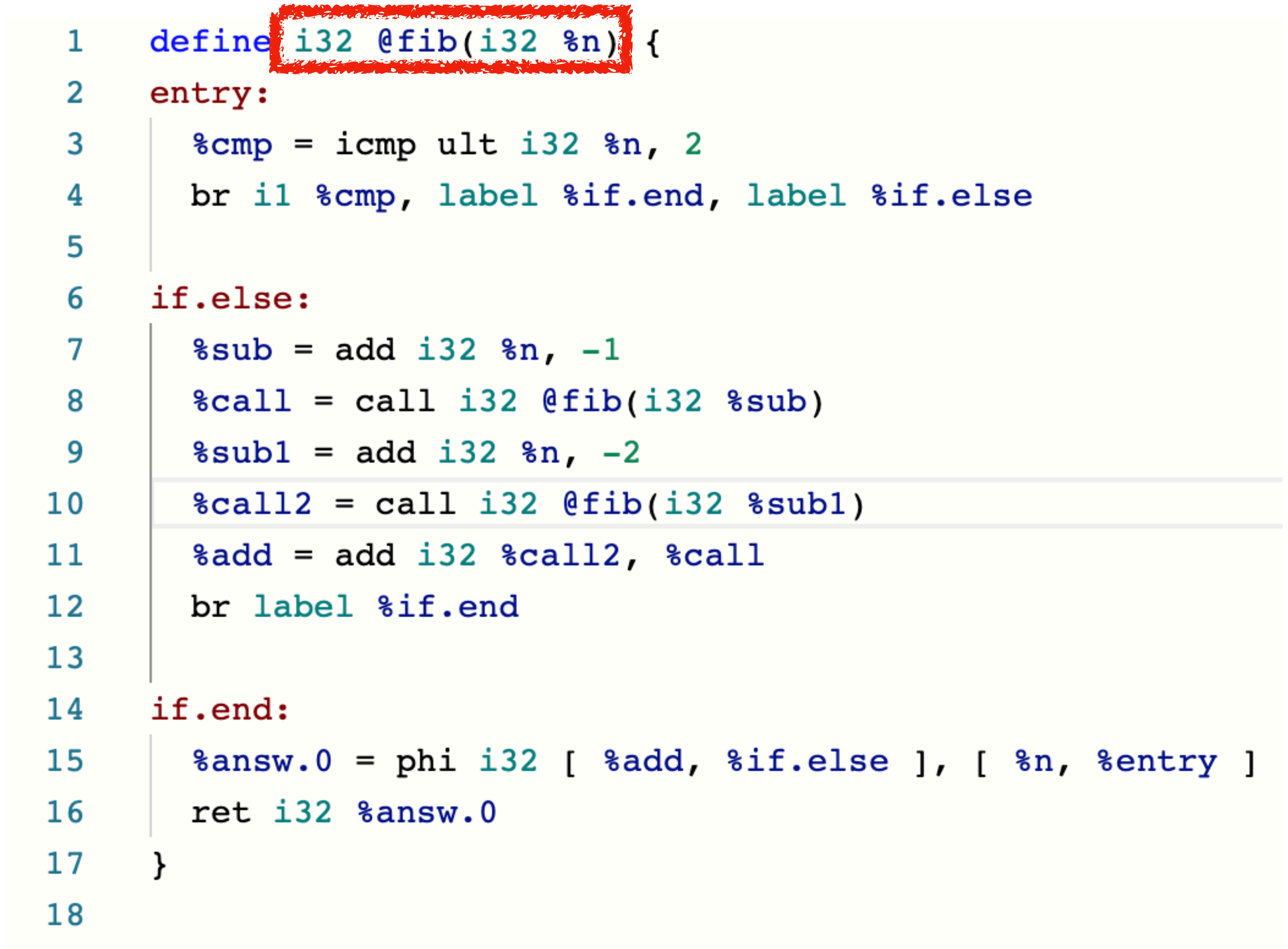
3 Basic blocks: entry, if.else, if.end

Example - fibonacci



```c
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

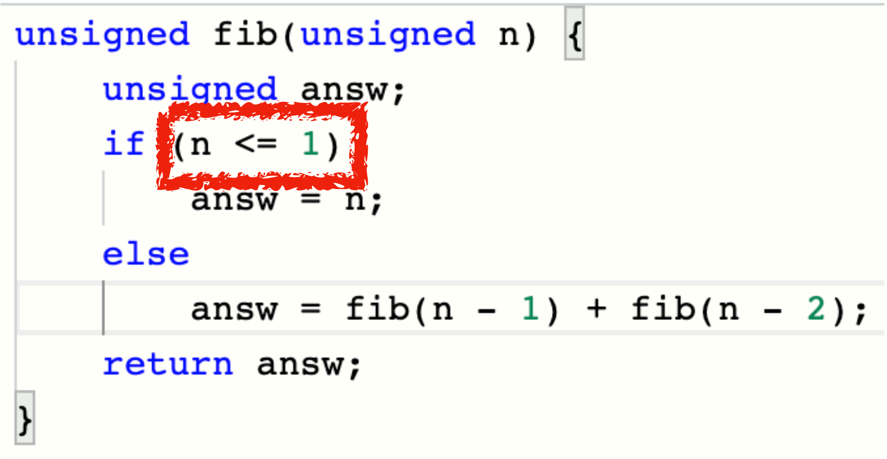
i32: integer, 32 bits



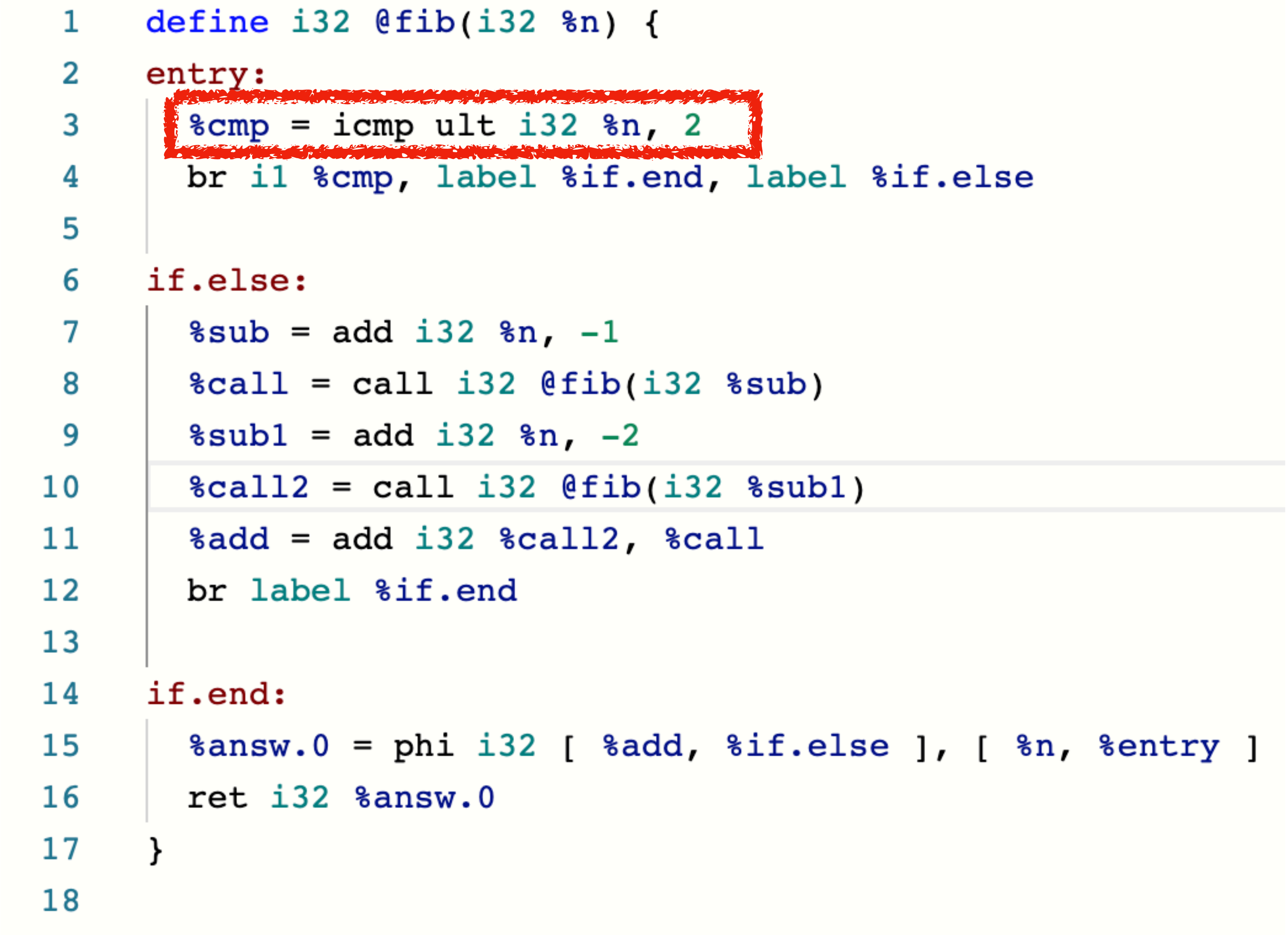
```llvm
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

Example - fibonacci

icmp: integer comparison
ult: unsigned comparison, less than



```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

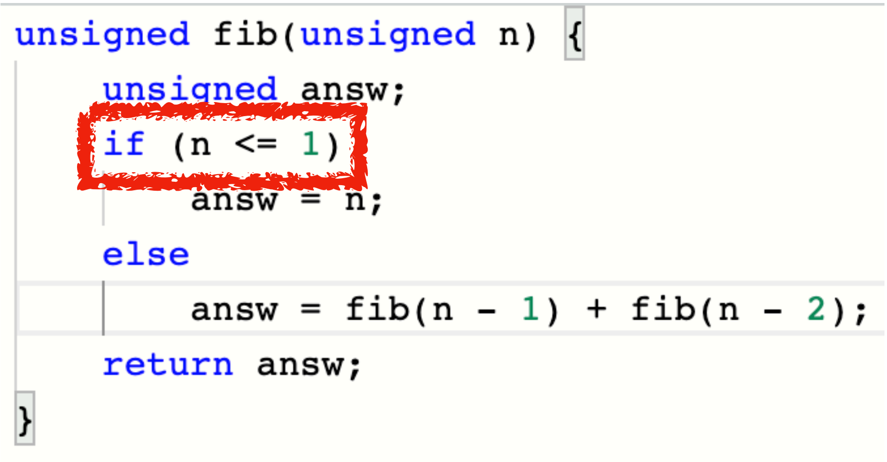


```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}
```

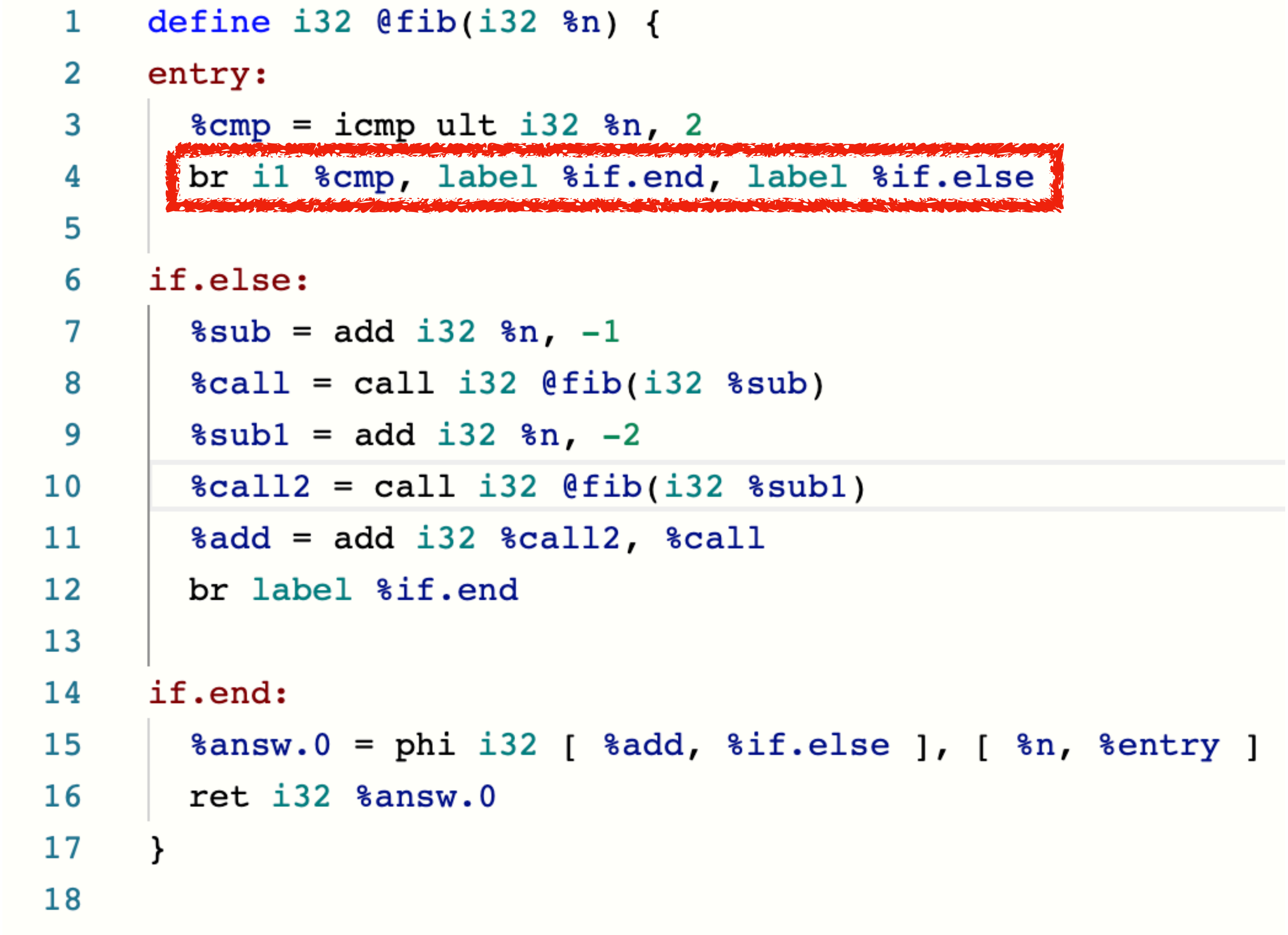
LLVM Language Reference Manual: https://llvm.org/docs/LangRef.html

Example - fibonacci

Note that the branch condition is inversed
There is no special reason in this case.. :/

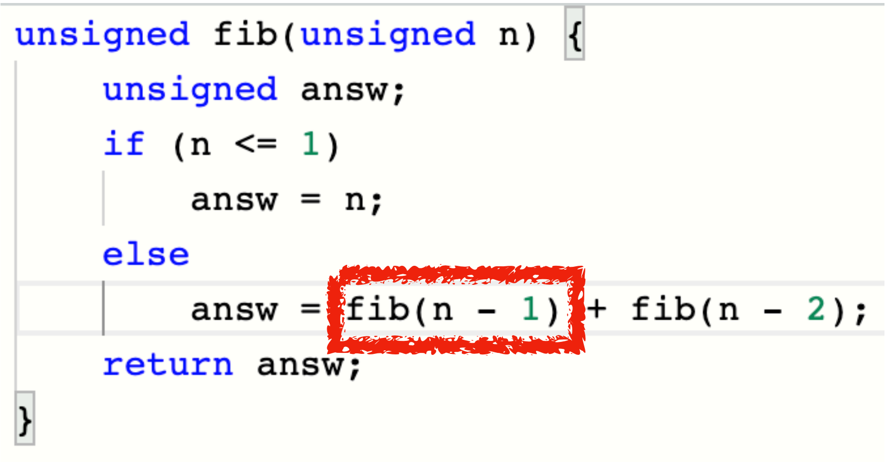


```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

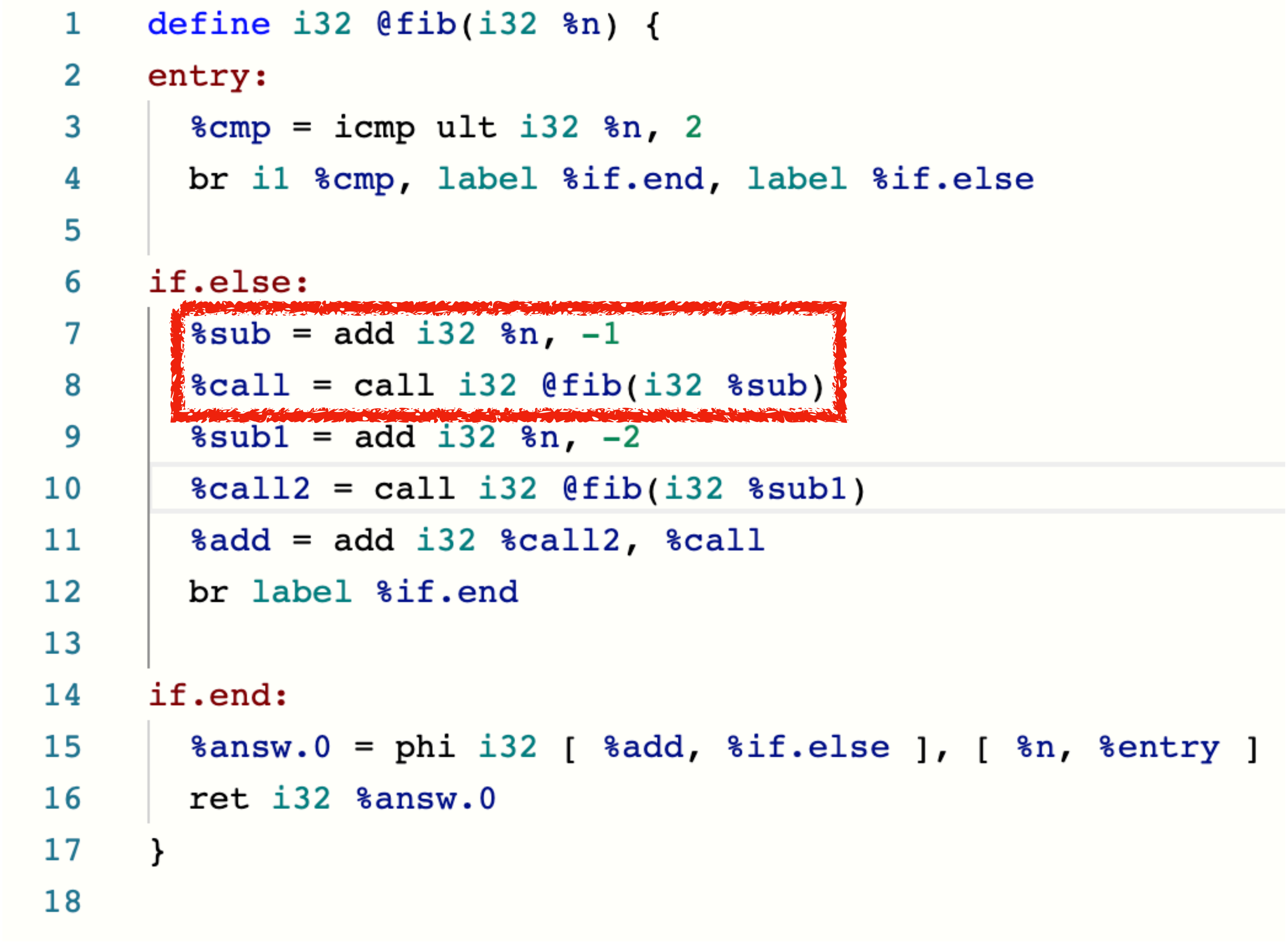


```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

Example - fibonacci

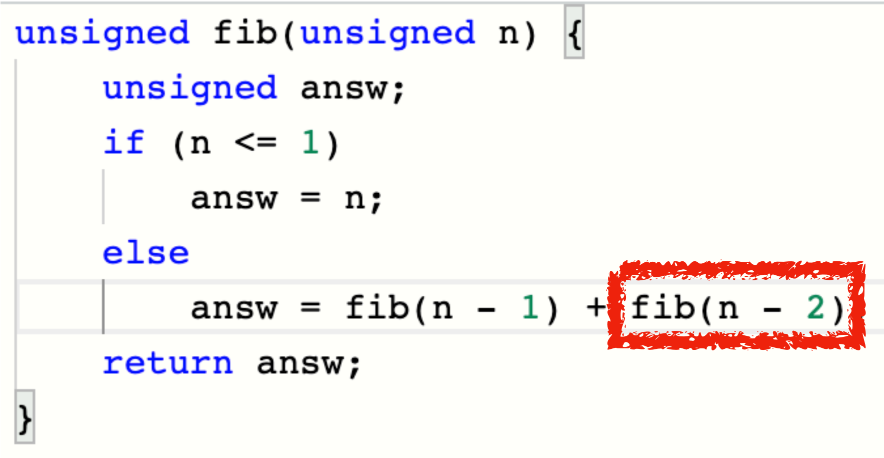


```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

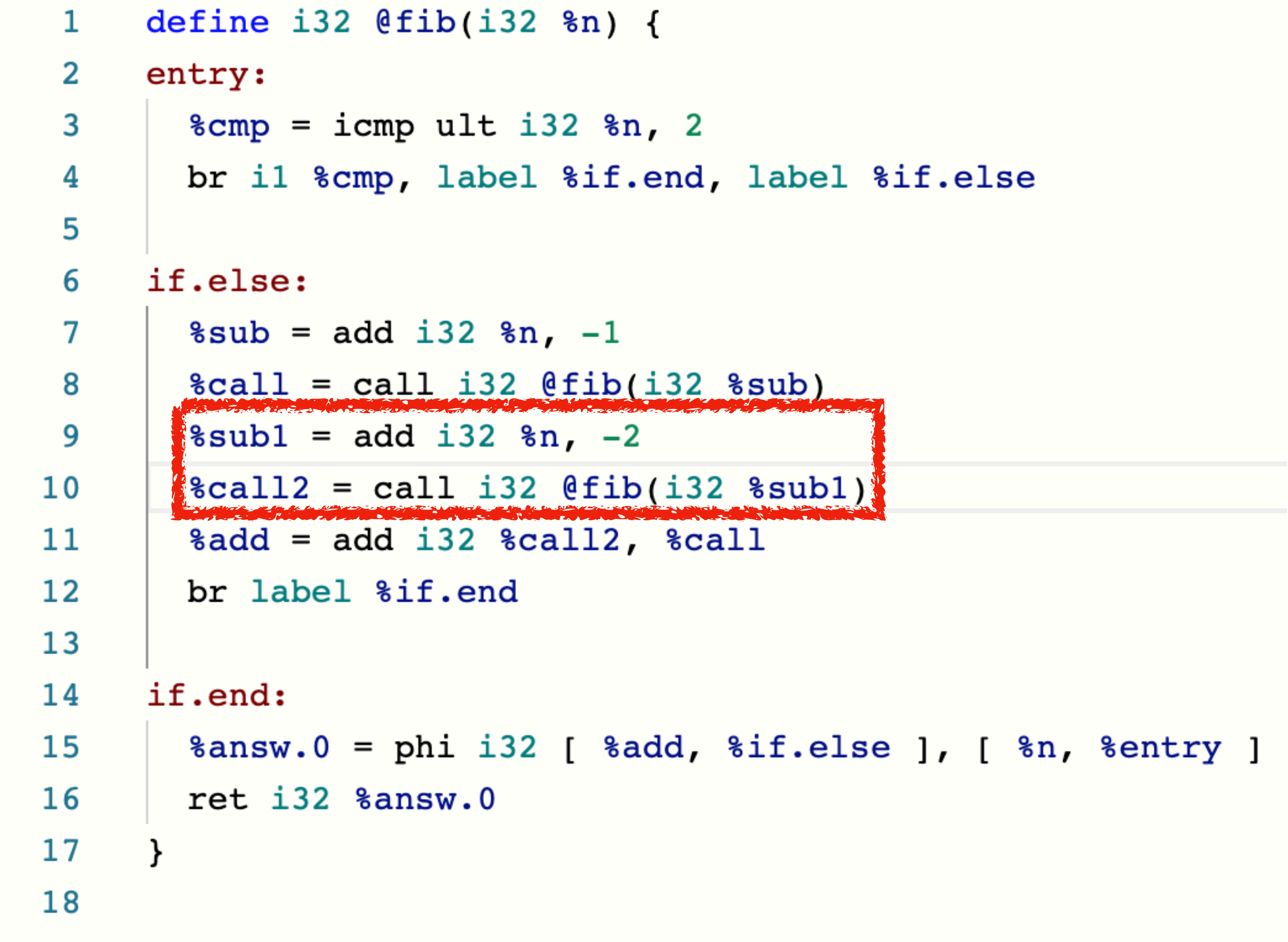


```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

Example - fibonacci

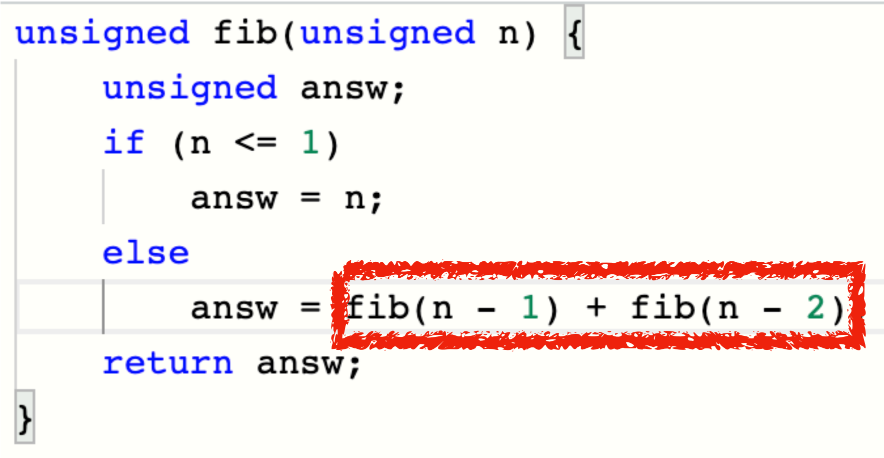


```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

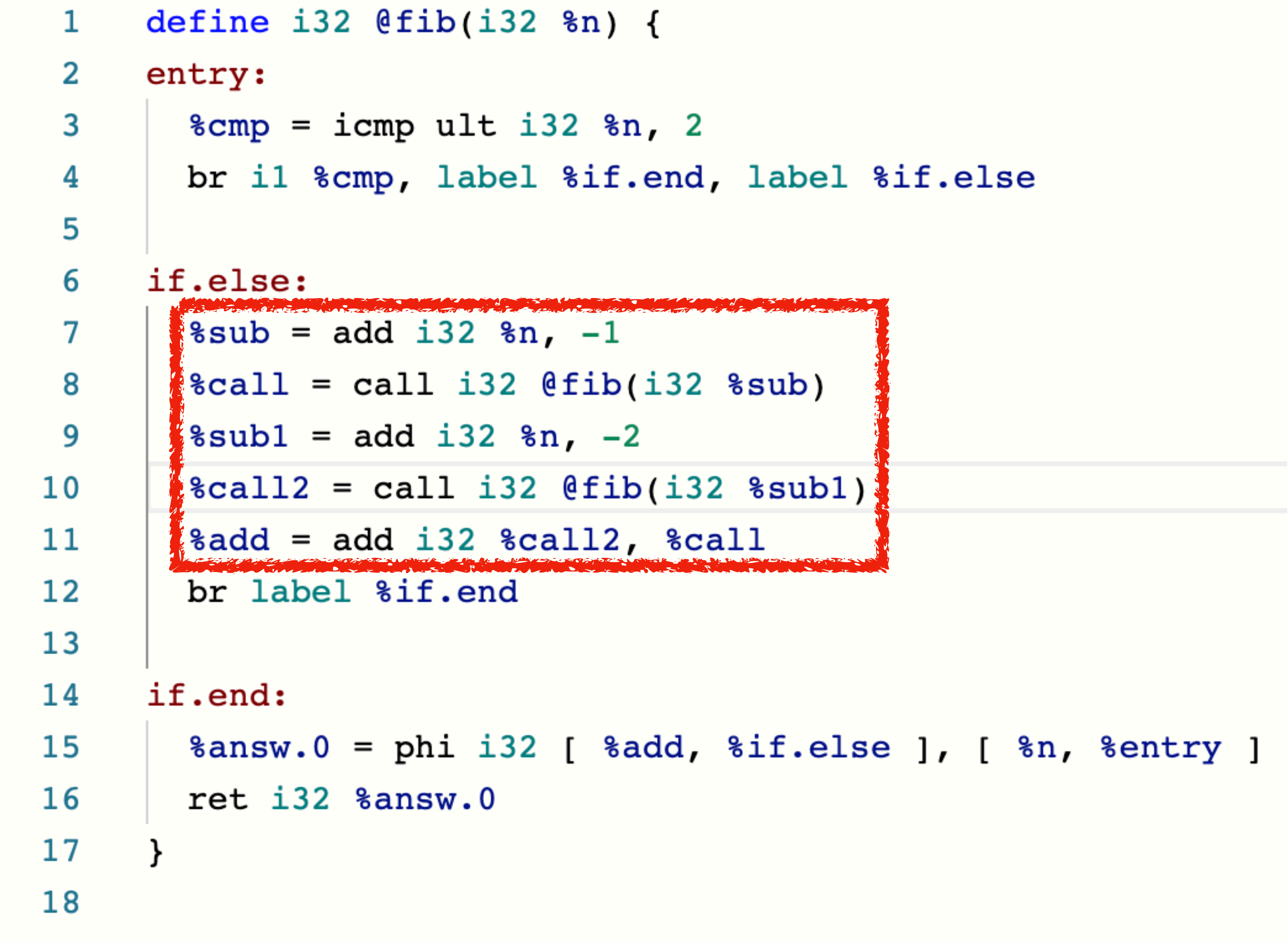


```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

Example - fibonacci



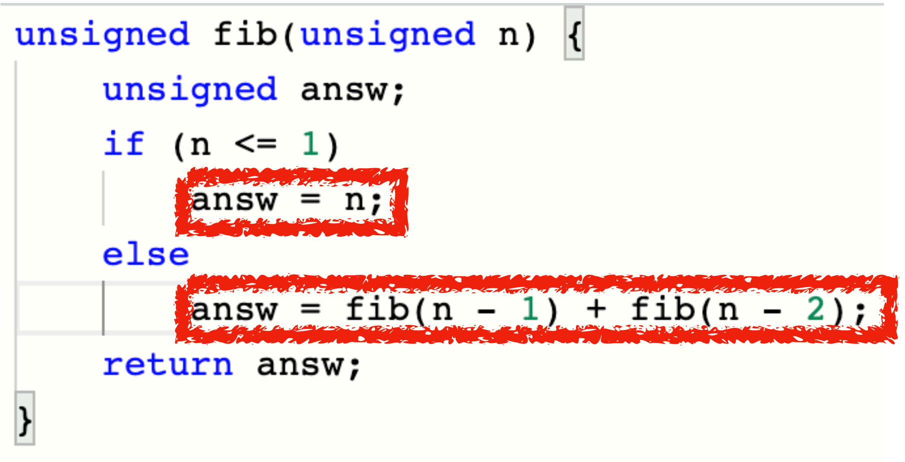
```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2)
    return answ;
}
```



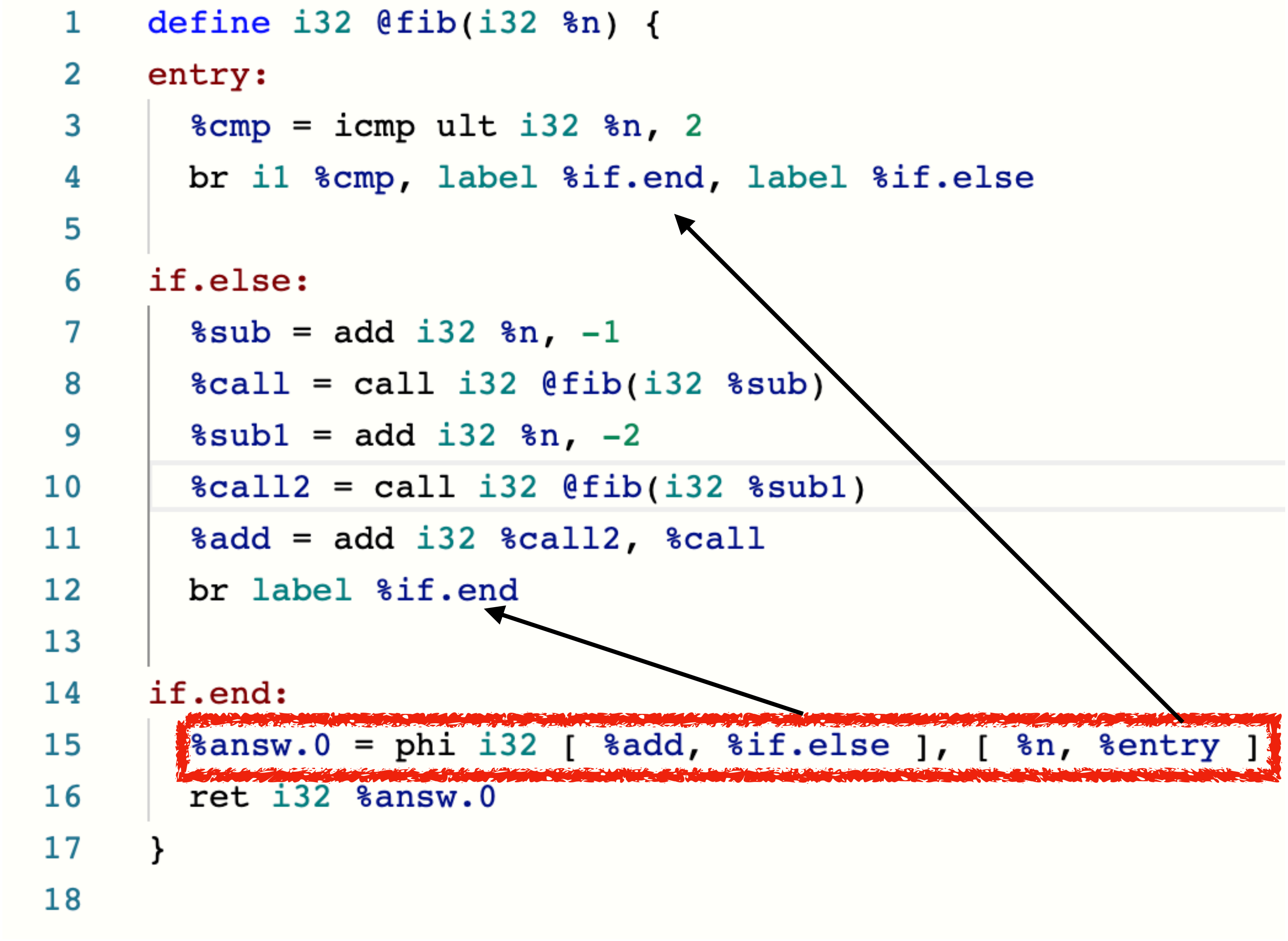
```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}
```

Example - fibonacci

Multiple definitions of variables in different blocks are merged with a phi node.

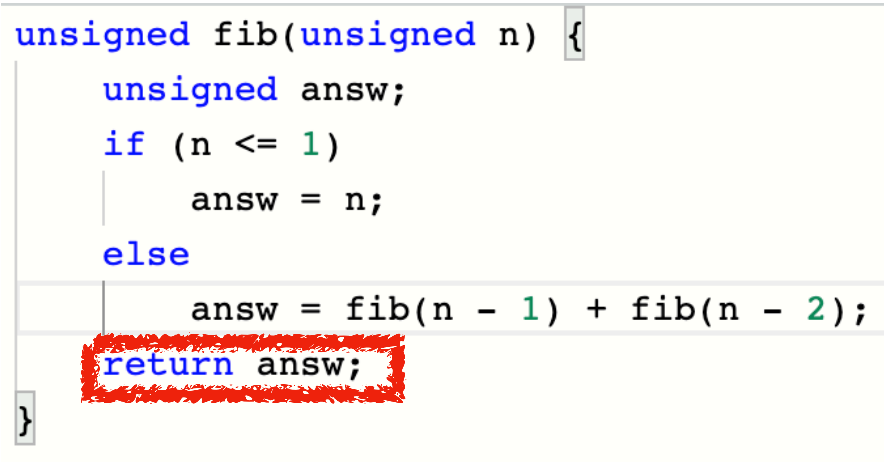


```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```

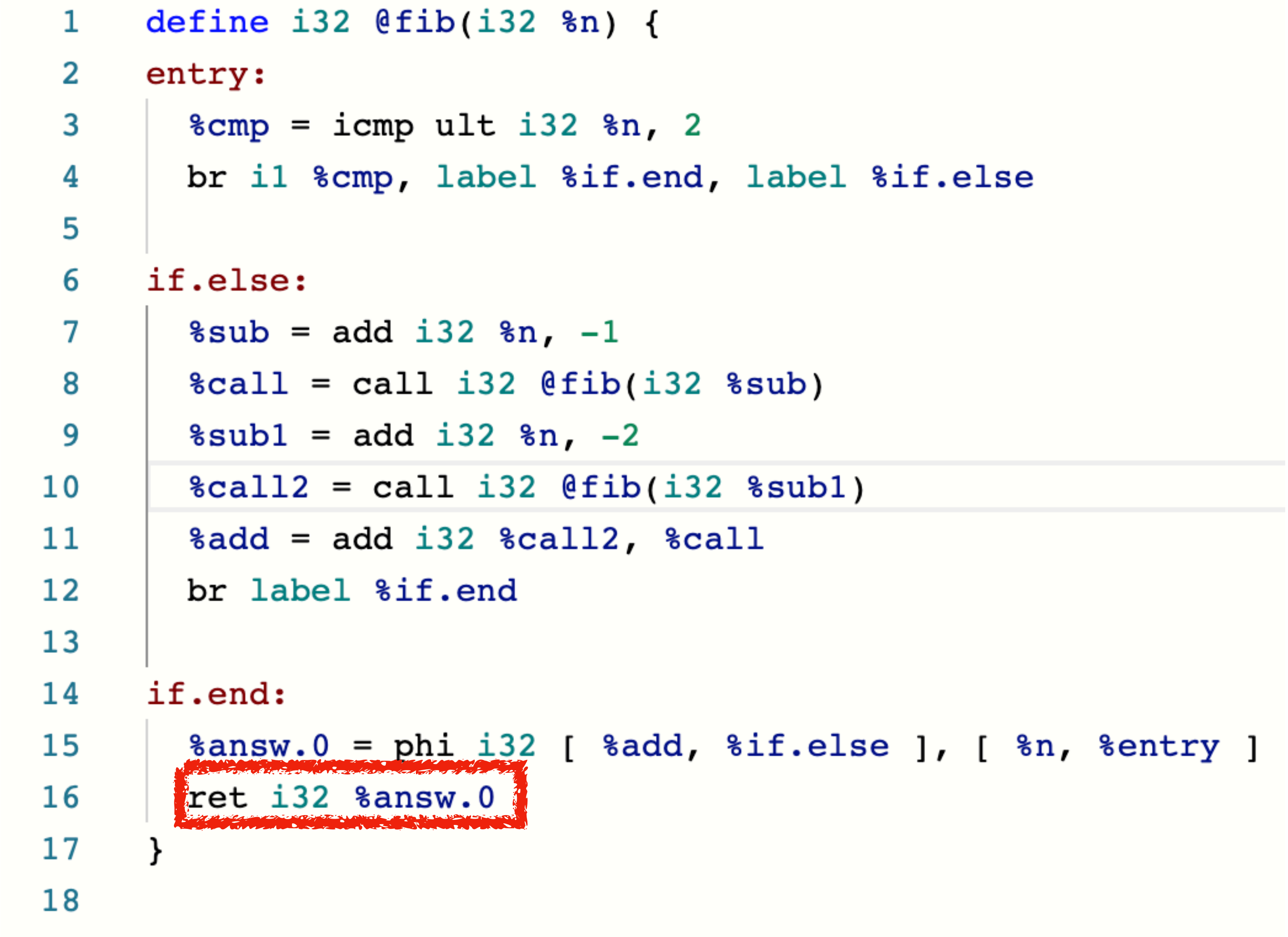


```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}
```

Example - fibonacci



```
unsigned fib(unsigned n) {
    unsigned answ;
    if (n <= 1)
        answ = n;
    else
        answ = fib(n - 1) + fib(n - 2);
    return answ;
}
```



```
define i32 @fib(i32 %n) {
entry:
  %cmp = icmp ult i32 %n, 2
  br i1 %cmp, label %if.end, label %if.else

if.else:
  %sub = add i32 %n, -1
  %call = call i32 @fib(i32 %sub)
  %sub1 = add i32 %n, -2
  %call2 = call i32 @fib(i32 %sub1)
  %add = add i32 %call2, %call
  br label %if.end

if.end:
  %answ.0 = phi i32 [ %add, %if.else ], [ %n, %entry ]
  ret i32 %answ.0
}

```

Play with fibonacci

- Please store the C program as fib.c
- C -> IR:

```
bin/clang -S -emit-llvm -O1 -g0 \
          -fno-discard-value-names fib.c -o -
```

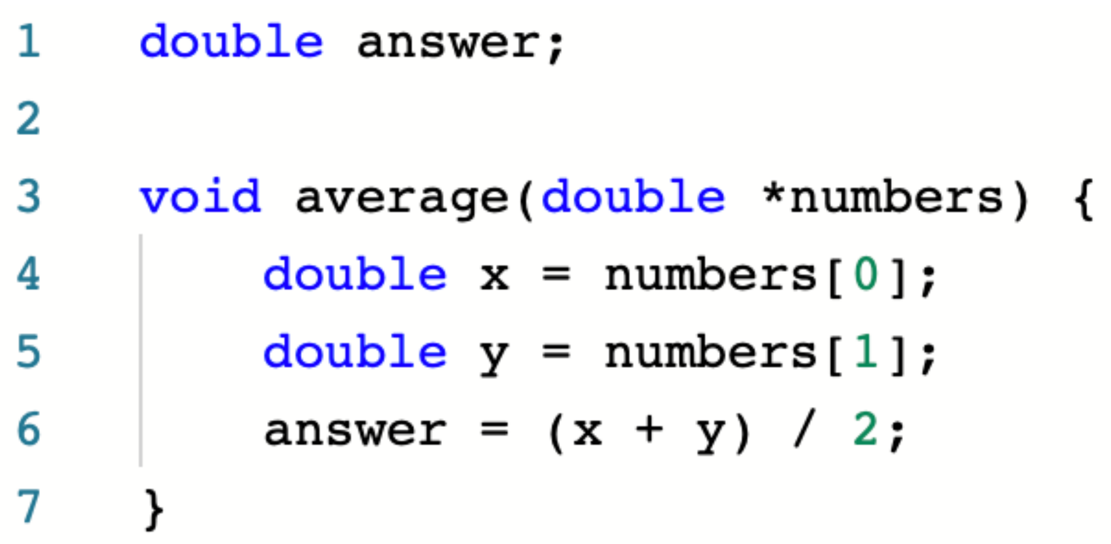
- IR -> Assembly:

```
bin/llc -o fib.s fib.ll
```

fib.ll vs. fib.bc

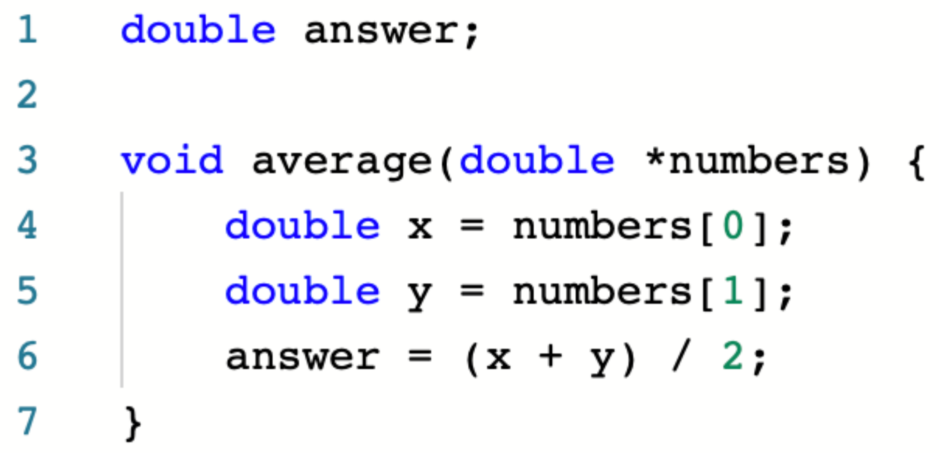
- .ll file: textual form (human understandable form)
- .bc file: binary form (compact, faster for a machine to read)
- `bin/llvm-as  fib.ll -o fib.bc`
- `bin/llvm-dis fib.bc -o fib.ll`

Example 2 - average

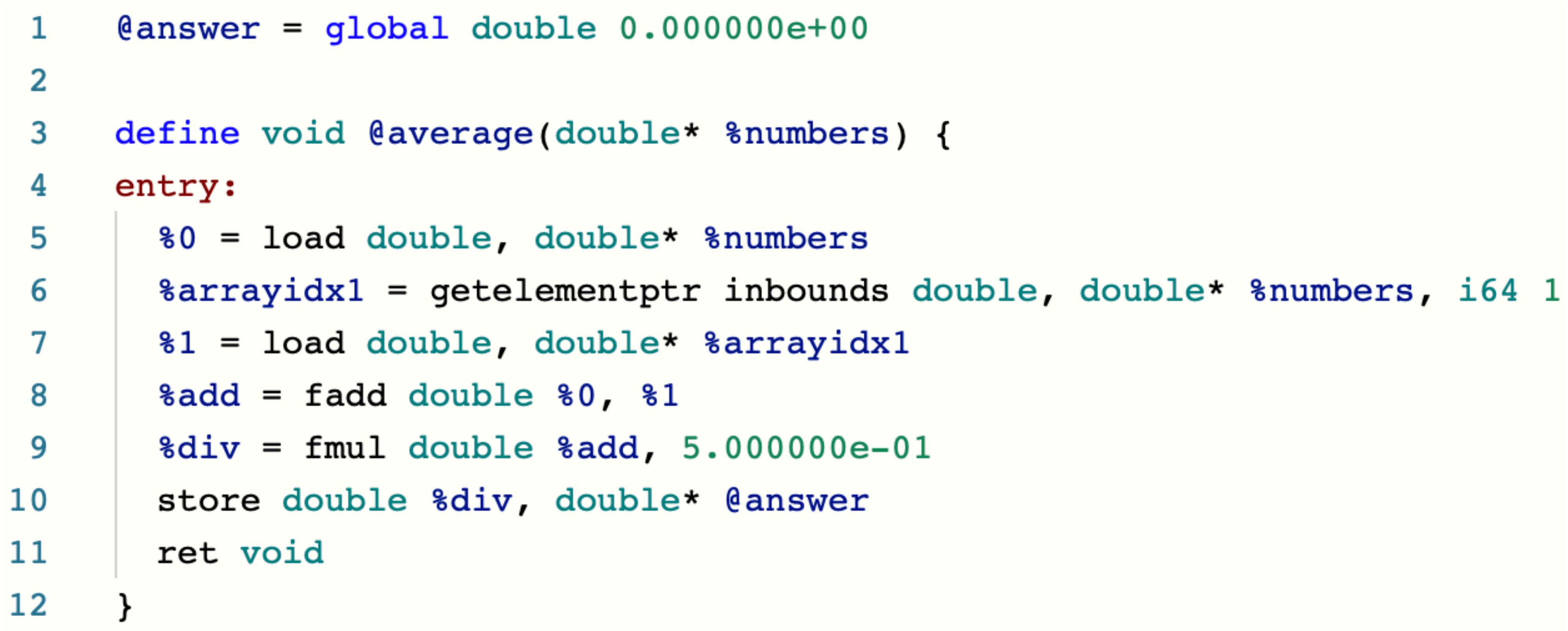


```
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```

Example 2 - average

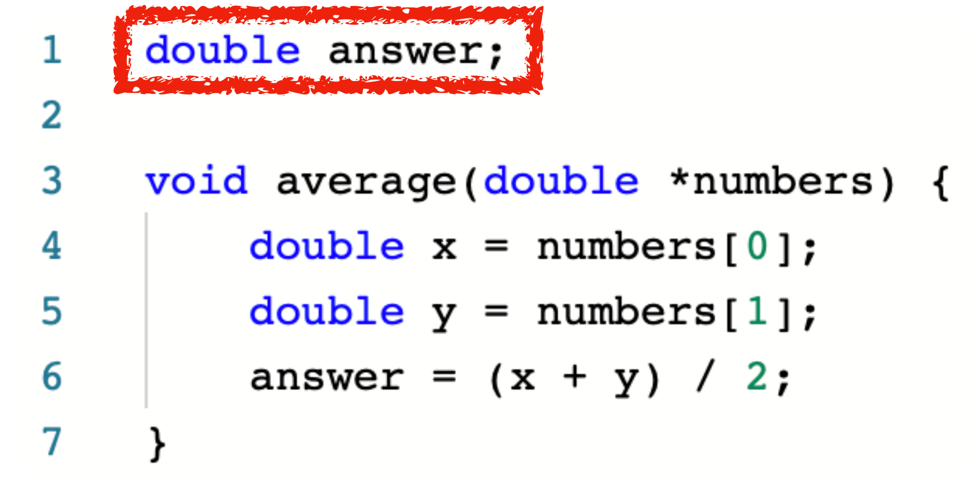


```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```



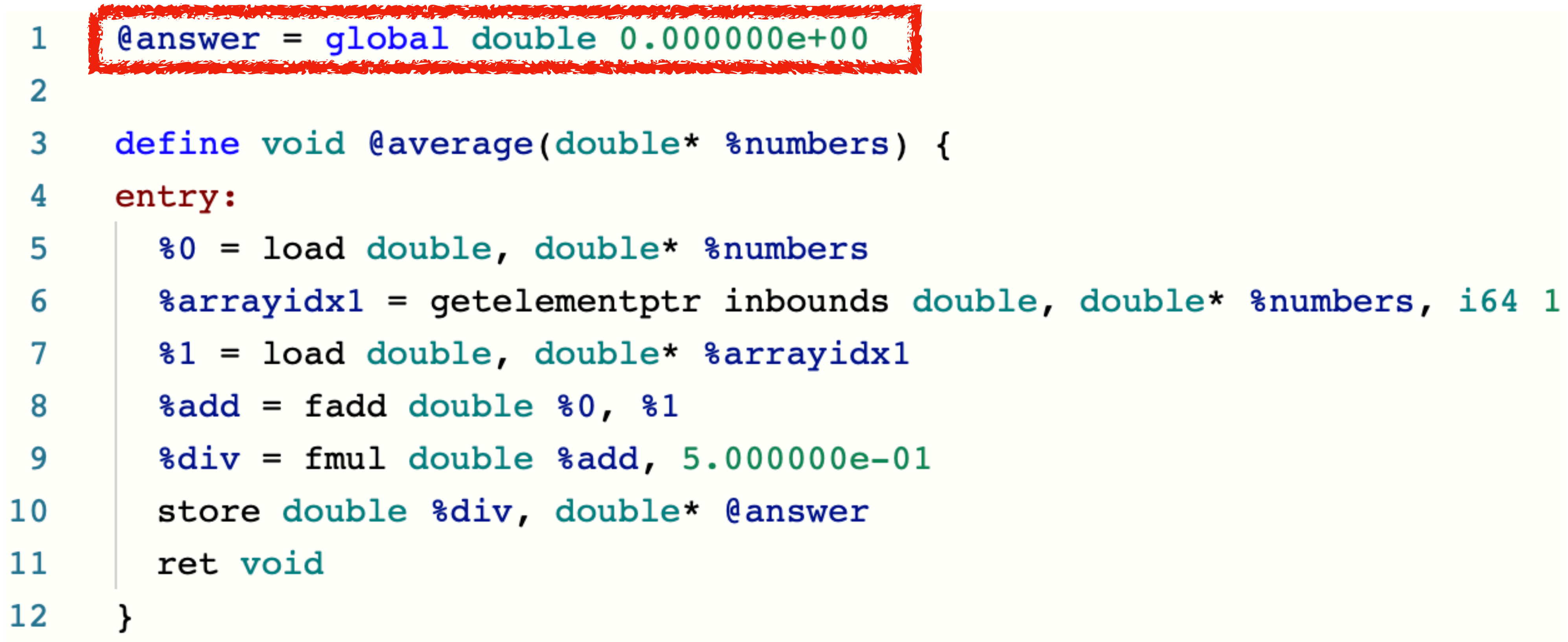
```llvm
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

Example 2 - average



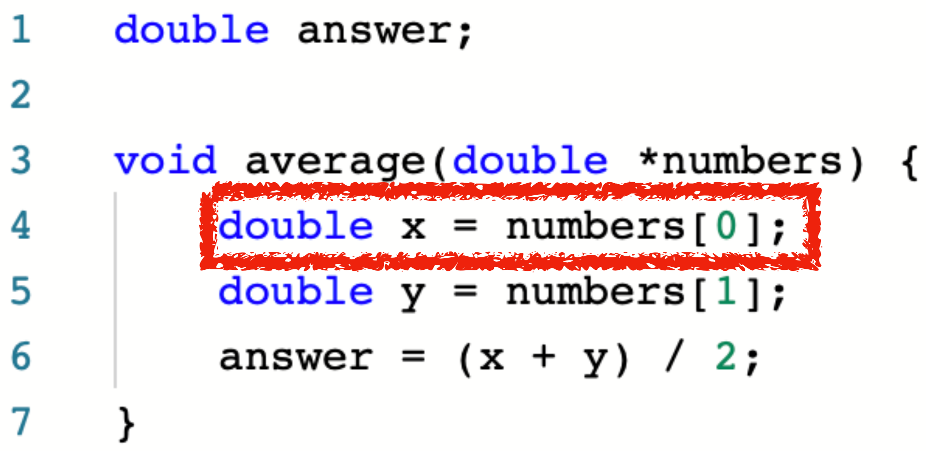
```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```

Global variables have prefix @



```llvm
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

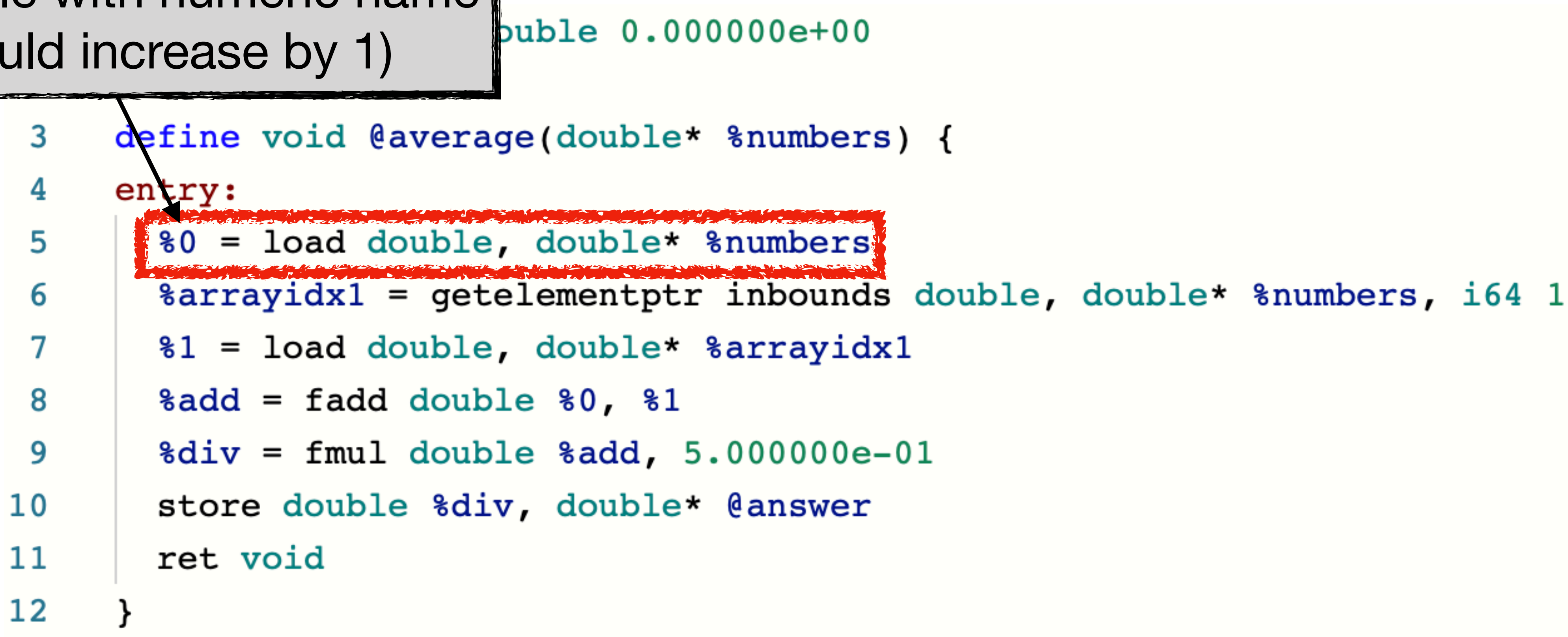
Example 2 - average



```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```

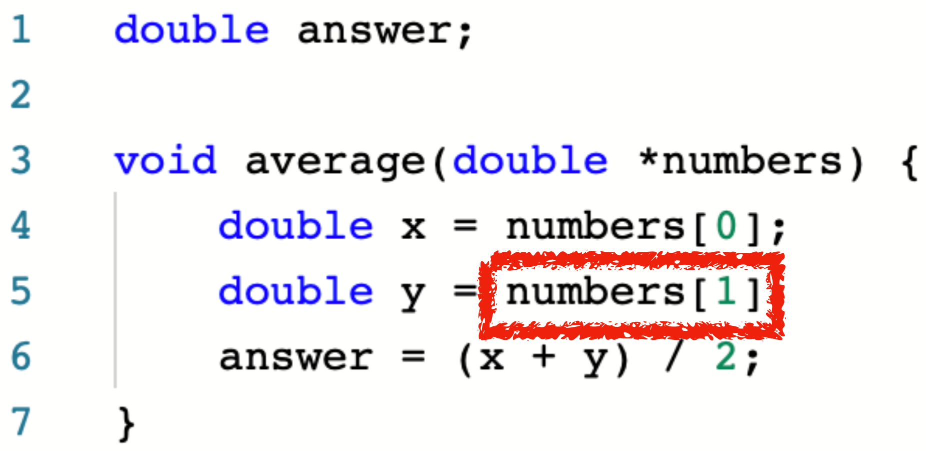
Dereference `%numbers`

A variable with numeric name (should increase by 1)



```llvm
                    uble 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

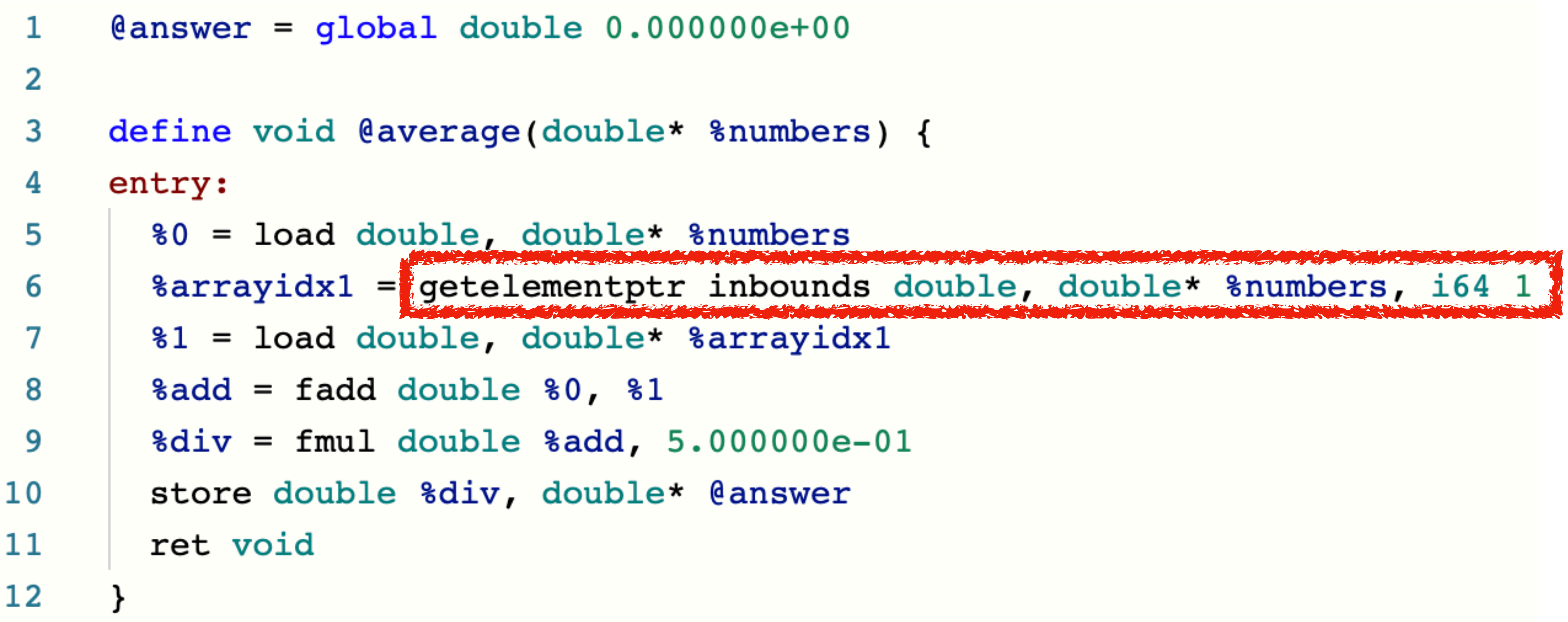
Example 2 - average



```
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```

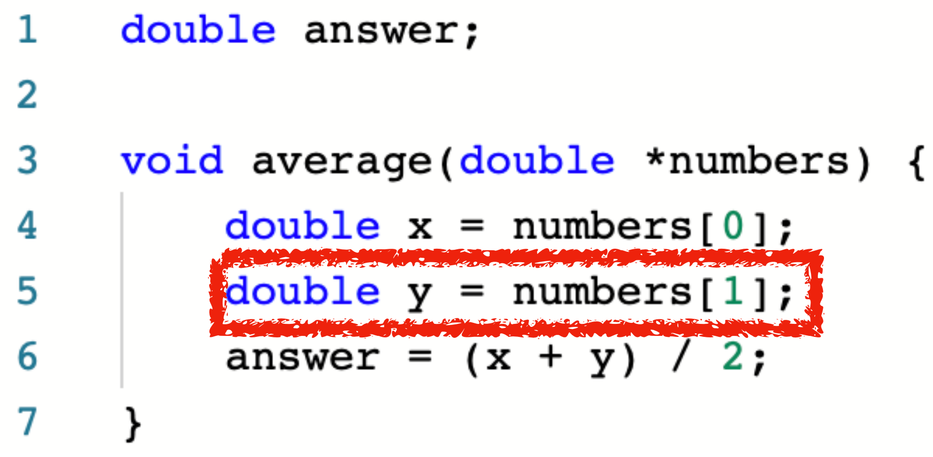
`numbers[1]` **is** `*(numbers + 1)`

Let's calculate `(numbers + 1)` **first**



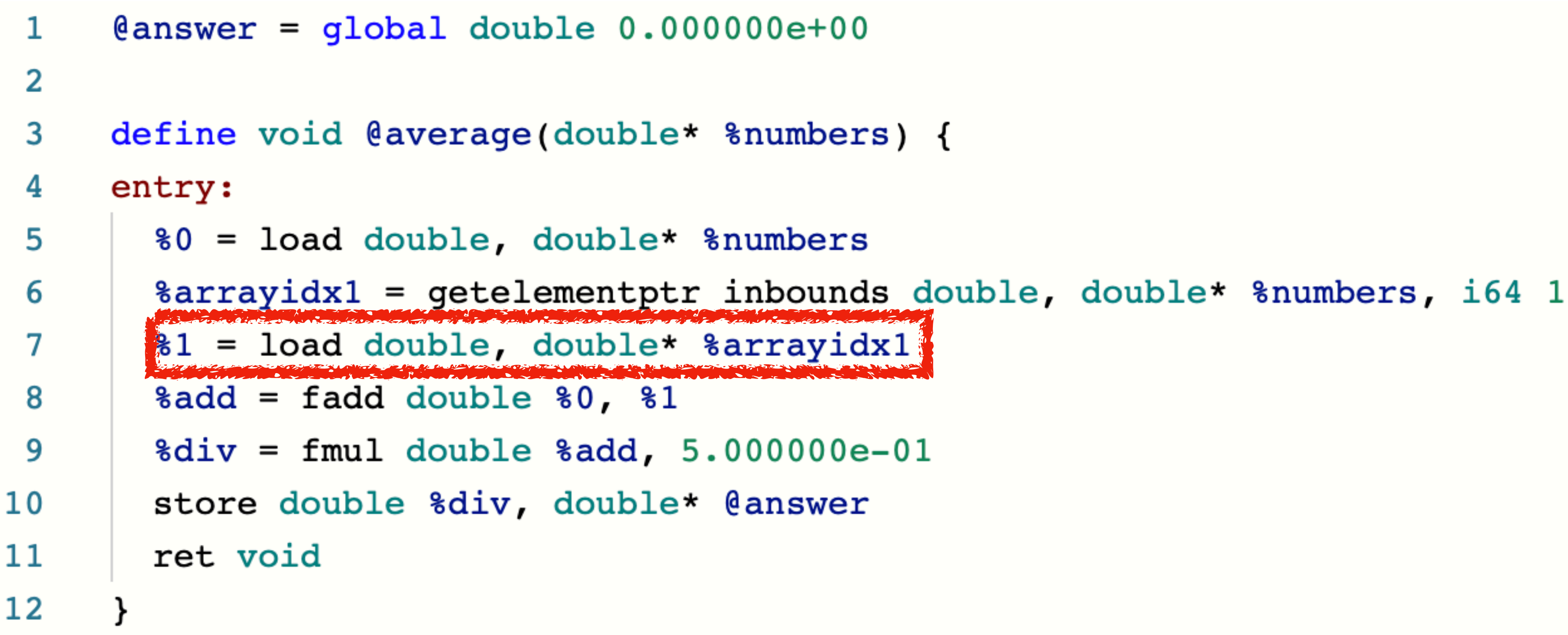
```
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

Example 2 - average



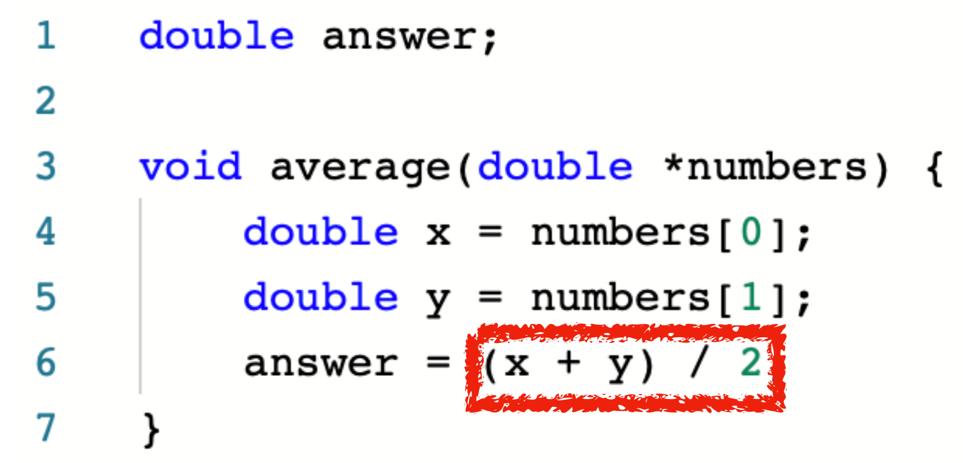
```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2;
}
```

Dereference `(numbers+1)`



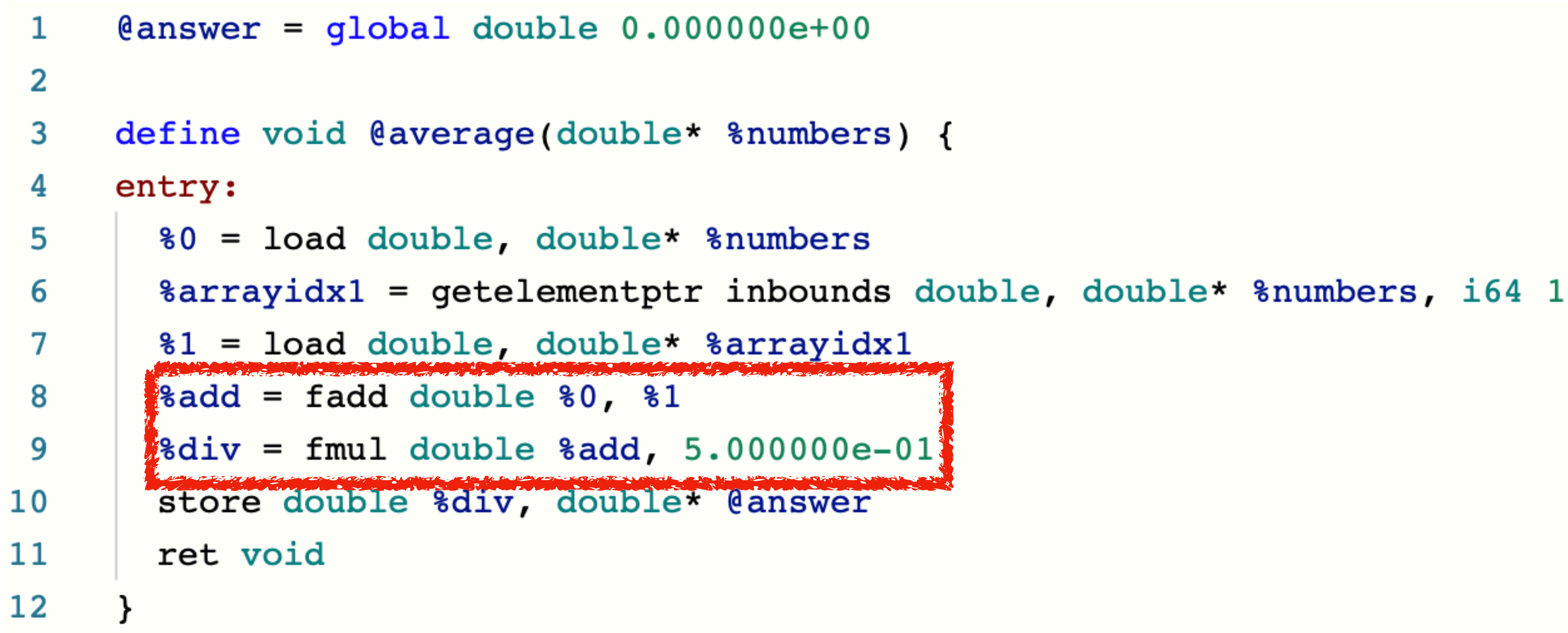
```llvm
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

Example 2 - average



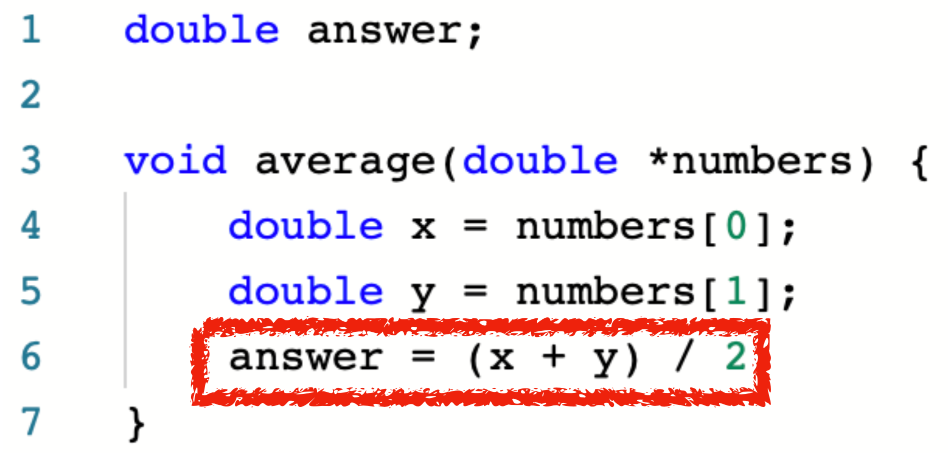
```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2
}
```

Calculate its average



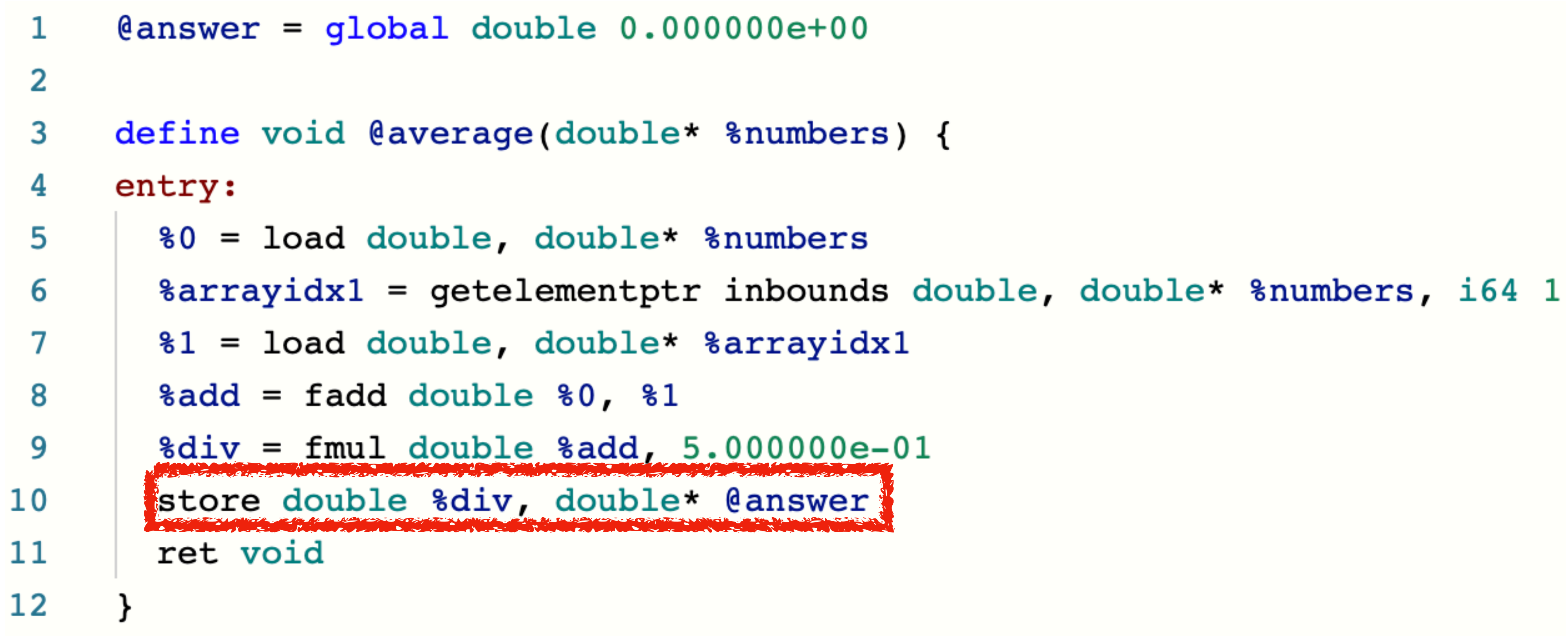
```llvm
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

Example 2 - average



```c
double answer;

void average(double *numbers) {
    double x = numbers[0];
    double y = numbers[1];
    answer = (x + y) / 2
}
```

Store the result to a global variable



```llvm
@answer = global double 0.000000e+00

define void @average(double* %numbers) {
entry:
  %0 = load double, double* %numbers
  %arrayidx1 = getelementptr inbounds double, double* %numbers, i64 1
  %1 = load double, double* %arrayidx1
  %add = fadd double %0, %1
  %div = fmul double %add, 5.000000e-01
  store double %div, double* @answer
  ret void
}
```

Command

```
bin/clang -S -emit-llvm -O1 -g0 \
          -fno-discard-value-names \
          -fno-strict-aliasing average.c -o -
```

Q: What is strict aliasing?