ACM Summer School on Compilers for AI/ML for Women 2024

Examining the LLVM IR

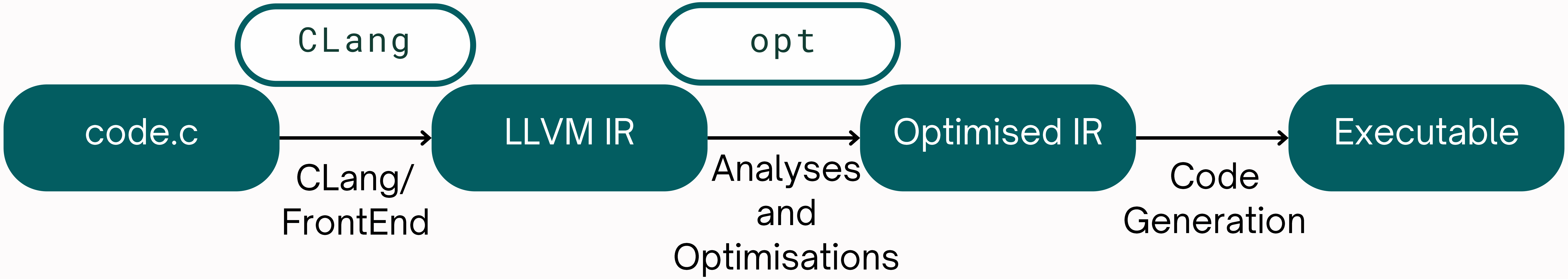
Supriya Bhide,
Doctoral Research Student,
IIT Bombay

What is an LLVM IR?

- **L**ow **L**evel **V**irtual **M**achine
- **I**ntermediate **R**epresentation
- Created under the direction of Chris Lattner, Vikram Adve at University of Illinois
- First Release: 2003

🍎 *Interesting fact: Apple uses LLVM for its macOS, iOS*

LLVM Compiler Pipeline



How do we get there?

```
clang -fno-discard-value-names -emit-llvm -O0 input_src_file.c -o <llvm_ir.ll>
```

-fno-discard-value-names: Tells clang to not remove the variable names

-emit-llvm: Stops the process after generation of LLVM IR

-O0: Specifies the level of optimization
`input_src_file.c`: Source code
`llvm_ir.ll`: A name for the file to which the IR will be saved (you may give any name).

```
opt -mem2reg llvm_ir.ll > opt_llvm_ir.ll
```

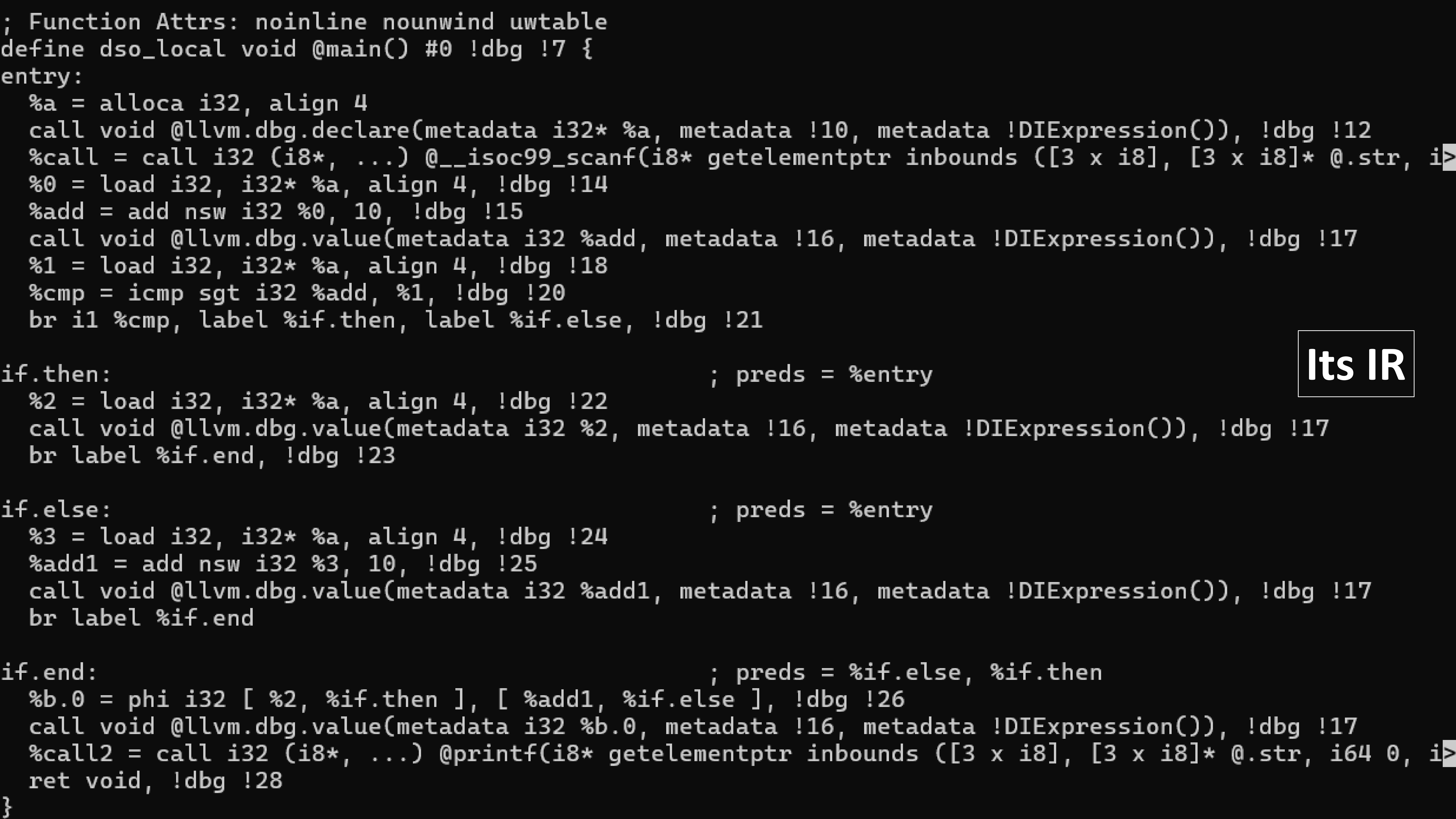
-mem2reg: Promotes registers to SSA variables
`opt_llvm_ir.ll`: A name for the file to which the IR modified after running passes with opt would be saved (you may give any name).

"An example is worth a thousand explanations"

CODE

```
#include <stdio.h>
void main()
{
    int a, b;

    scanf("%d", &a);

    if (a > 10)
        b = a;
    else
        b = a + 10;

    printf("%d", b);
}
```

Its IR



```
; Function Attrs: noinline nounwind uwtable
define dso_local void @main() #0 !dbg !7 {
entry:
  %a = alloca i32, align 4
  call void @llvm.dbg.declare(metadata i32* %a, metadata !10, metadata !DIExpression()), !dbg !12
  %call = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i>
  %0 = load i32, i32* %a, align 4, !dbg !14
  %add = add nsw i32 %0, 10, !dbg !15
  call void @llvm.dbg.value(metadata i32 %add, metadata !16, metadata !DIExpression()), !dbg !17
  %1 = load i32, i32* %a, align 4, !dbg !18
  %cmp = icmp sgt i32 %add, %1, !dbg !20
  br i1 %cmp, label %if.then, label %if.else, !dbg !21

if.then:                                          ; preds = %entry
  %2 = load i32, i32* %a, align 4, !dbg !22
  call void @llvm.dbg.value(metadata i32 %2, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end, !dbg !23

if.else:                                          ; preds = %entry
  %3 = load i32, i32* %a, align 4, !dbg !24
  %add1 = add nsw i32 %3, 10, !dbg !25
  call void @llvm.dbg.value(metadata i32 %add1, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %b.0 = phi i32 [ %2, %if.then ], [ %add1, %if.else ], !dbg !26
  call void @llvm.dbg.value(metadata i32 %b.0, metadata !16, metadata !DIExpression()), !dbg !17
  %call2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i>
  ret void, !dbg !28
}
```

A glimpse at 3 address code

Code

```
b = a;
```

```
b = a + 10;
```

3 Address Code

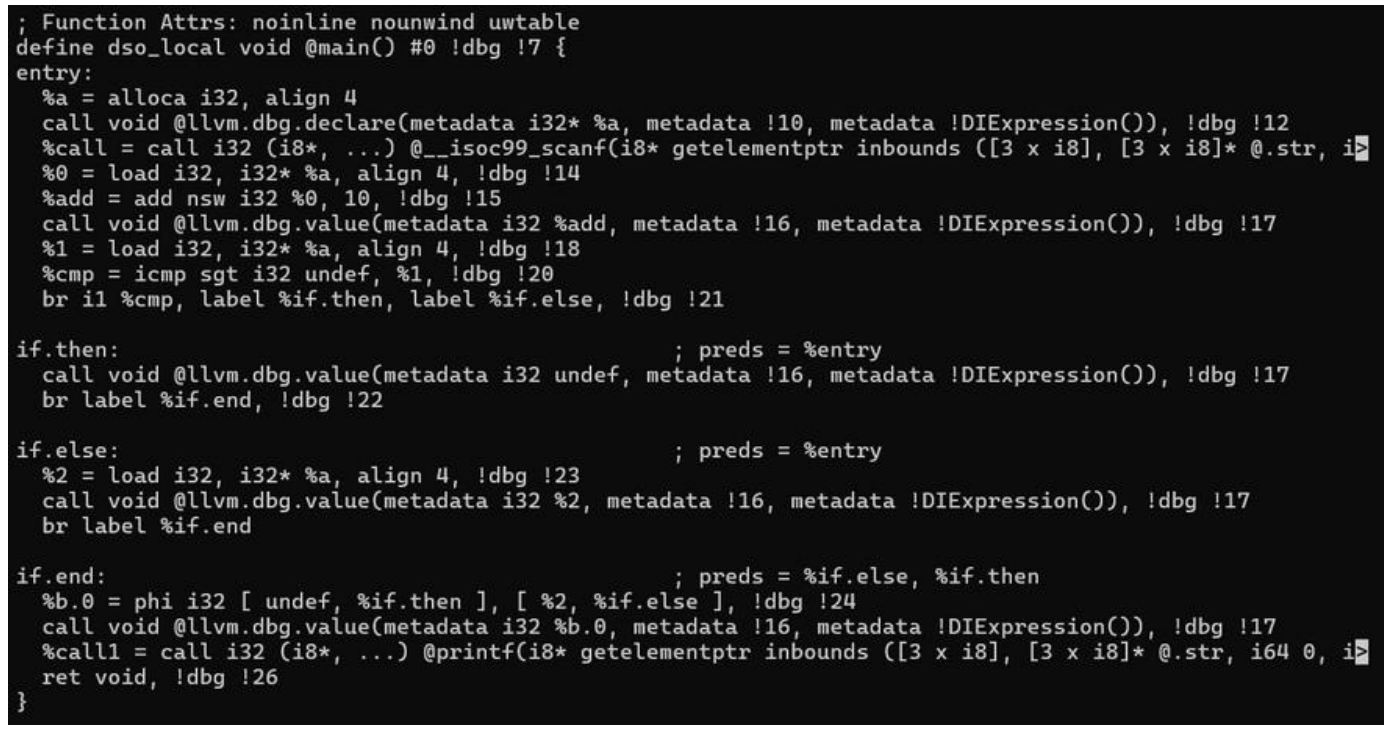
```
load t0, a
store b, t0
```

```
load t1, a
add = t1, 10
store b, add
```

CODE

```c
#include <stdio.h>
void main()
{
    int a, b;

    scanf("%d", &a);

    if (a > 10)
        b = a;
    else
        b = a + 10;

    printf("%d", b);
}
```

LLVM IR

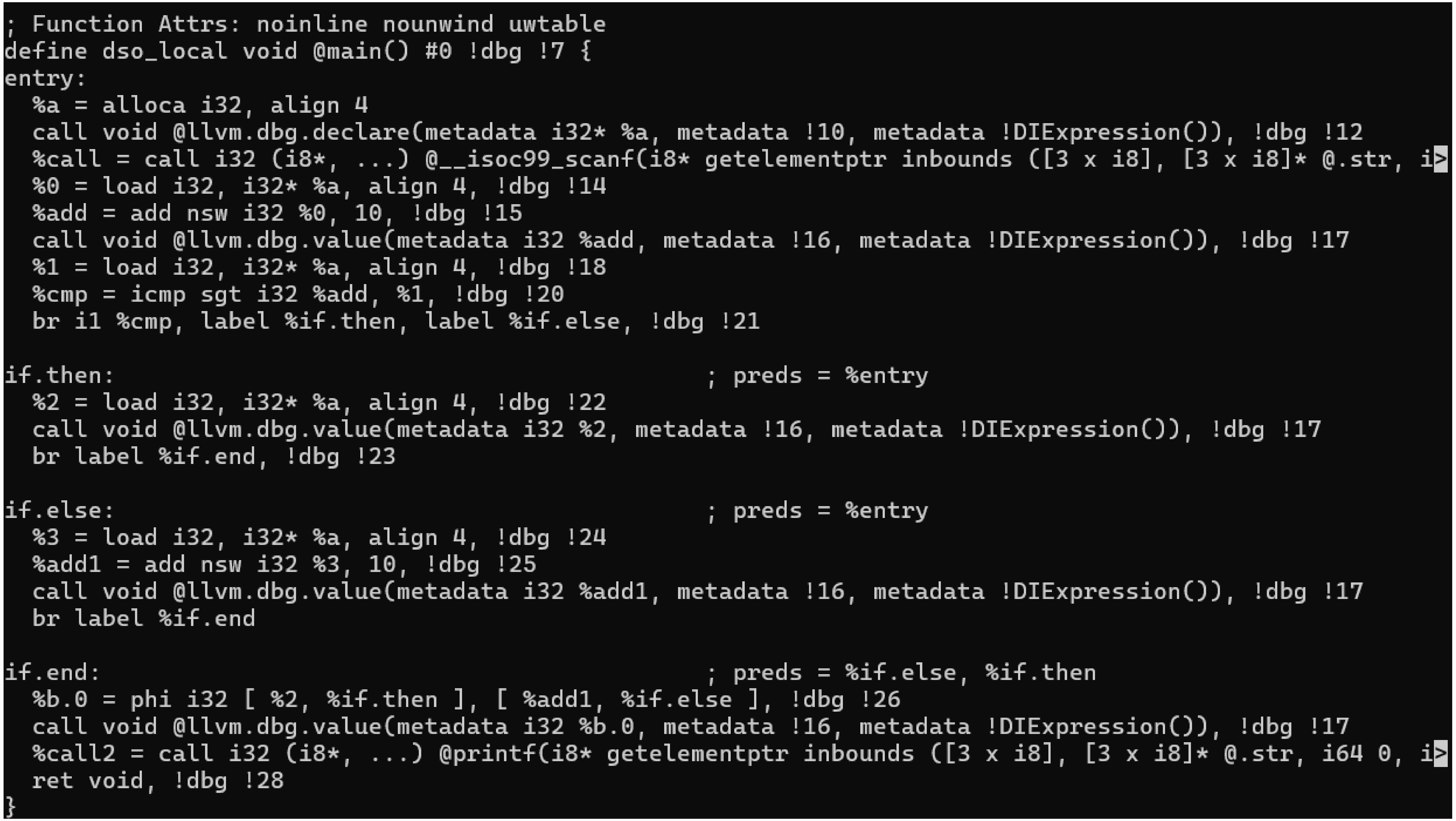


```
; Function Attrs: noinline nounwind uwtable
define dso_local void @main() #0 !dbg !7 {
entry:
  %a = alloca i32, align 4
  call void @llvm.dbg.declare(metadata i32* %a, metadata !10, metadata !DIExpression()), !dbg !12
  %call = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i>
  %0 = load i32, i32* %a, align 4, !dbg !14
  %add = add nsw i32 %0, 10, !dbg !15
  call void @llvm.dbg.value(metadata i32 %add, metadata !16, metadata !DIExpression()), !dbg !17
  %1 = load i32, i32* %a, align 4, !dbg !18
  %cmp = icmp sgt i32 undef, %1, !dbg !20
  br i1 %cmp, label %if.then, label %if.else, !dbg !21

if.then:                                          ; preds = %entry
  call void @llvm.dbg.value(metadata i32 undef, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end, !dbg !22

if.else:                                          ; preds = %entry
  %2 = load i32, i32* %a, align 4, !dbg !23
  call void @llvm.dbg.value(metadata i32 %2, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %b.0 = phi i32 [ undef, %if.then ], [ %2, %if.else ], !dbg !24
  call void @llvm.dbg.value(metadata i32 %b.0, metadata !16, metadata !DIExpression()), !dbg !17
  %call1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i>
  ret void, !dbg !26
}
```

CODE

```c
#include <stdio.h>
void main()
{
 int a, b;

 scanf("%d", &a);

 if (a > 10)
 b = a;
 else
 b = a + 10;

 printf("%d", b);
}
```

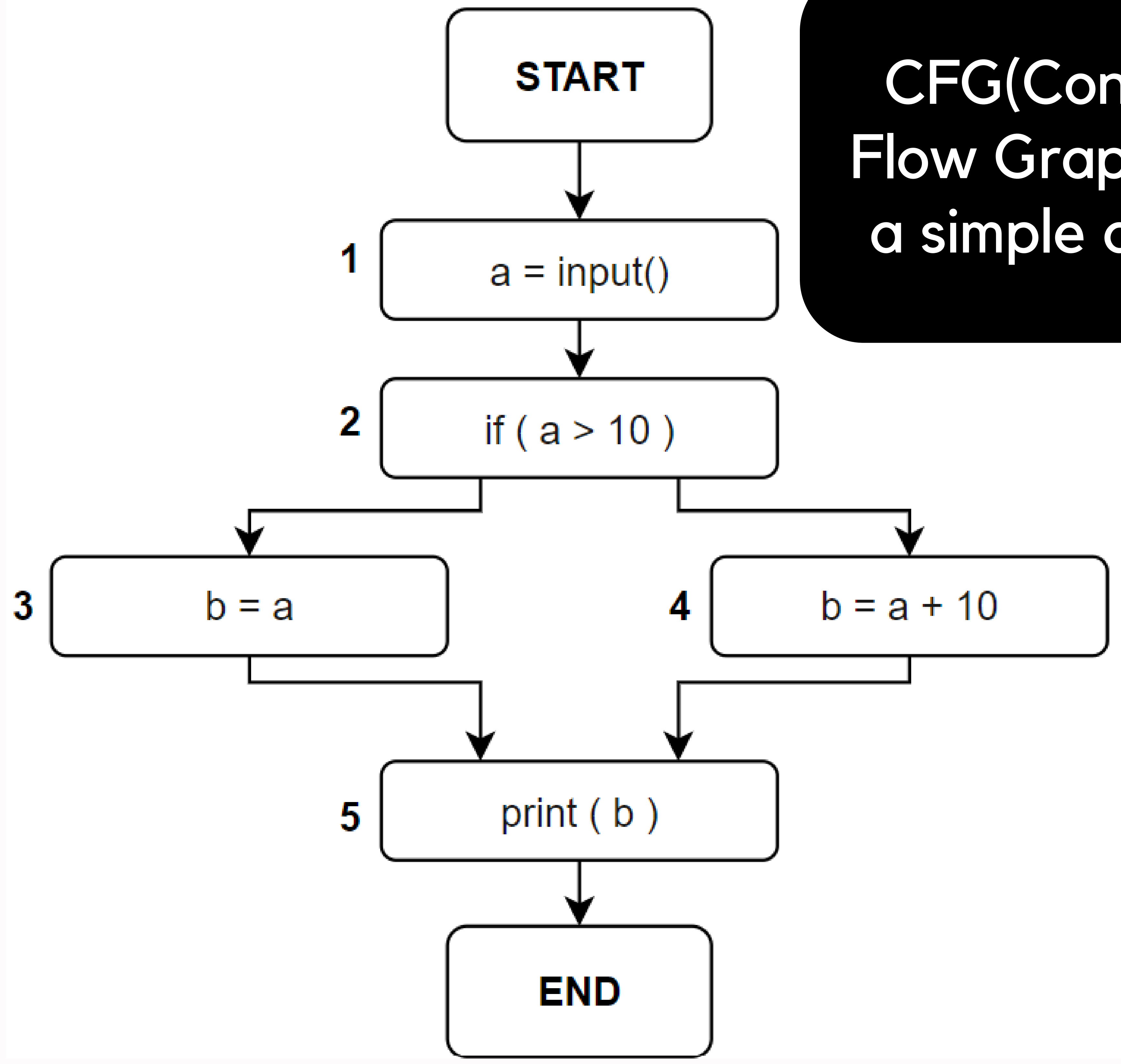
LLVM IR



```
; Function Attrs: noinline nounwind uwtable
define dso_local void @main() #0 !dbg !7 {
entry:
  %a = alloca i32, align 4
  call void @llvm.dbg.declare(metadata i32* %a, metadata !10, metadata !DIExpression()), !dbg !12
  %call = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i>
  %0 = load i32, i32* %a, align 4, !dbg !14
  %add = add nsw i32 %0, 10, !dbg !15
  call void @llvm.dbg.value(metadata i32 %add, metadata !16, metadata !DIExpression()), !dbg !17
  %1 = load i32, i32* %a, align 4, !dbg !18
  %cmp = icmp sgt i32 %add, %1, !dbg !20
  br i1 %cmp, label %if.then, label %if.else, !dbg !21

if.then:                                          ; preds = %entry
  %2 = load i32, i32* %a, align 4, !dbg !22
  call void @llvm.dbg.value(metadata i32 %2, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end, !dbg !23

if.else:                                          ; preds = %entry
  %3 = load i32, i32* %a, align 4, !dbg !24
  %add1 = add nsw i32 %3, 10, !dbg !25
  call void @llvm.dbg.value(metadata i32 %add1, metadata !16, metadata !DIExpression()), !dbg !17
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %b.0 = phi i32 [ %2, %if.then ], [ %add1, %if.else ], !dbg !26
  call void @llvm.dbg.value(metadata i32 %b.0, metadata !16, metadata !DIExpression()), !dbg !17
  %call2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0, i>
  ret void, !dbg !28
}
```

SSA-Static Single Assignment

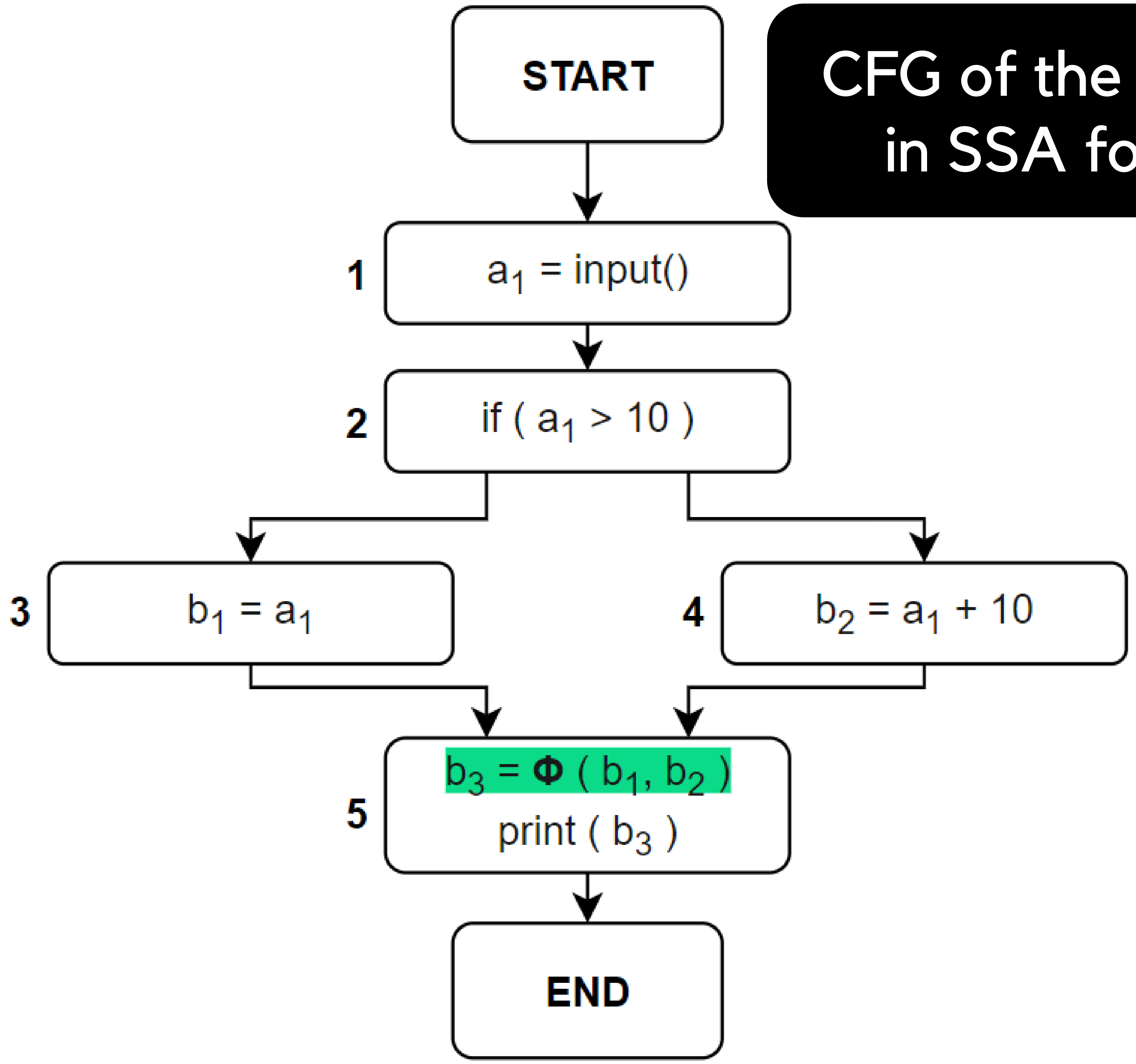
```
#include <stdio.h>
void main()
{
    int a, b;

    scanf("%d", &a);

    if (a > 10)
        b = a;
    else
        b = a + 10;

    printf("%d", b);
}
```



CFG(Control Flow Graph) of a simple code

SSA-Static Single Assignment

```
#include <stdio.h>
void main()
{
 int a, b;

 scanf("%d", &a);

 if (a > 10)
 b = a;
 else
 b = a + 10;

 printf("%d", b);
}
```

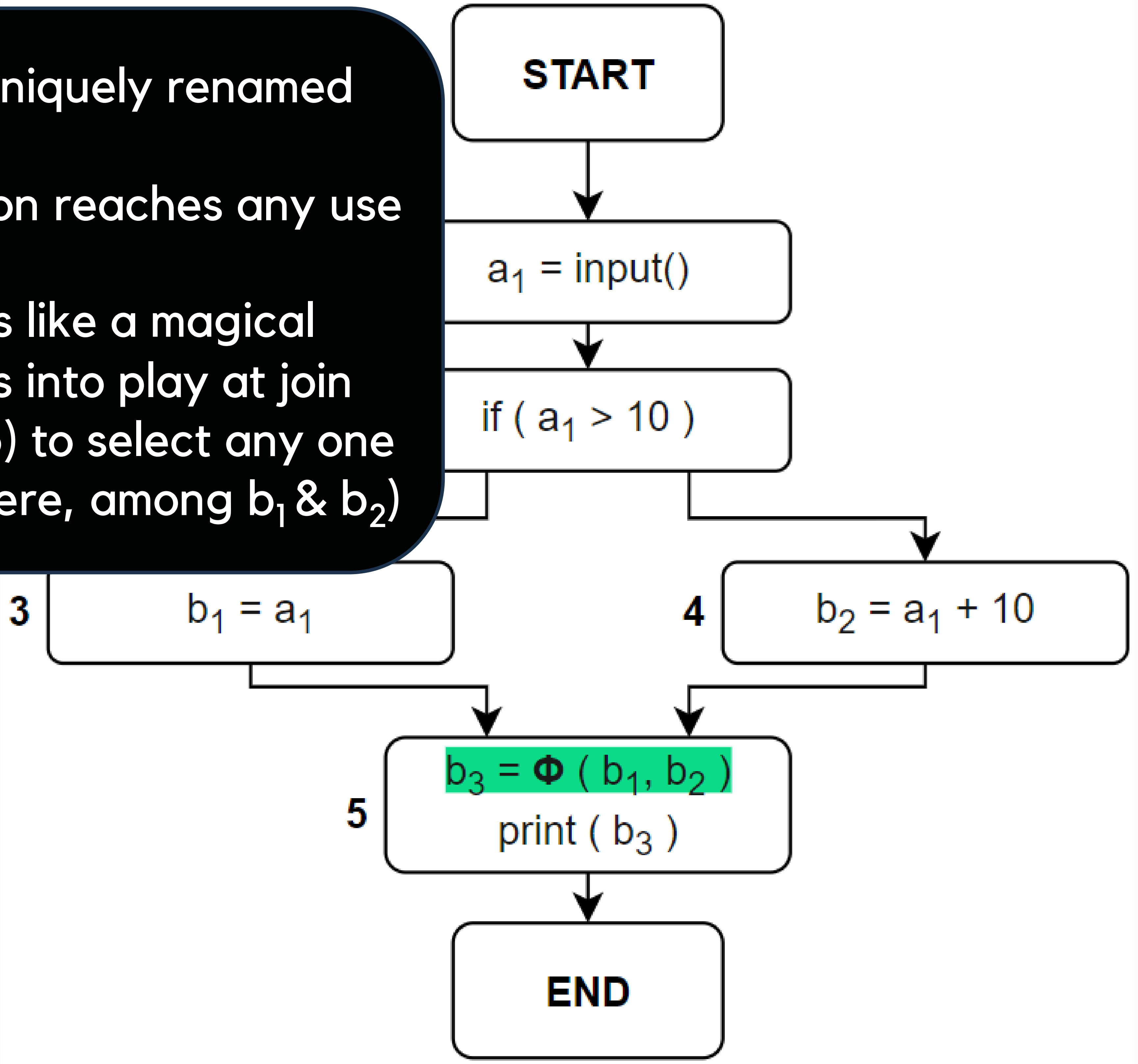


CFG of the code in SSA form

SSA-Static Single Assignment

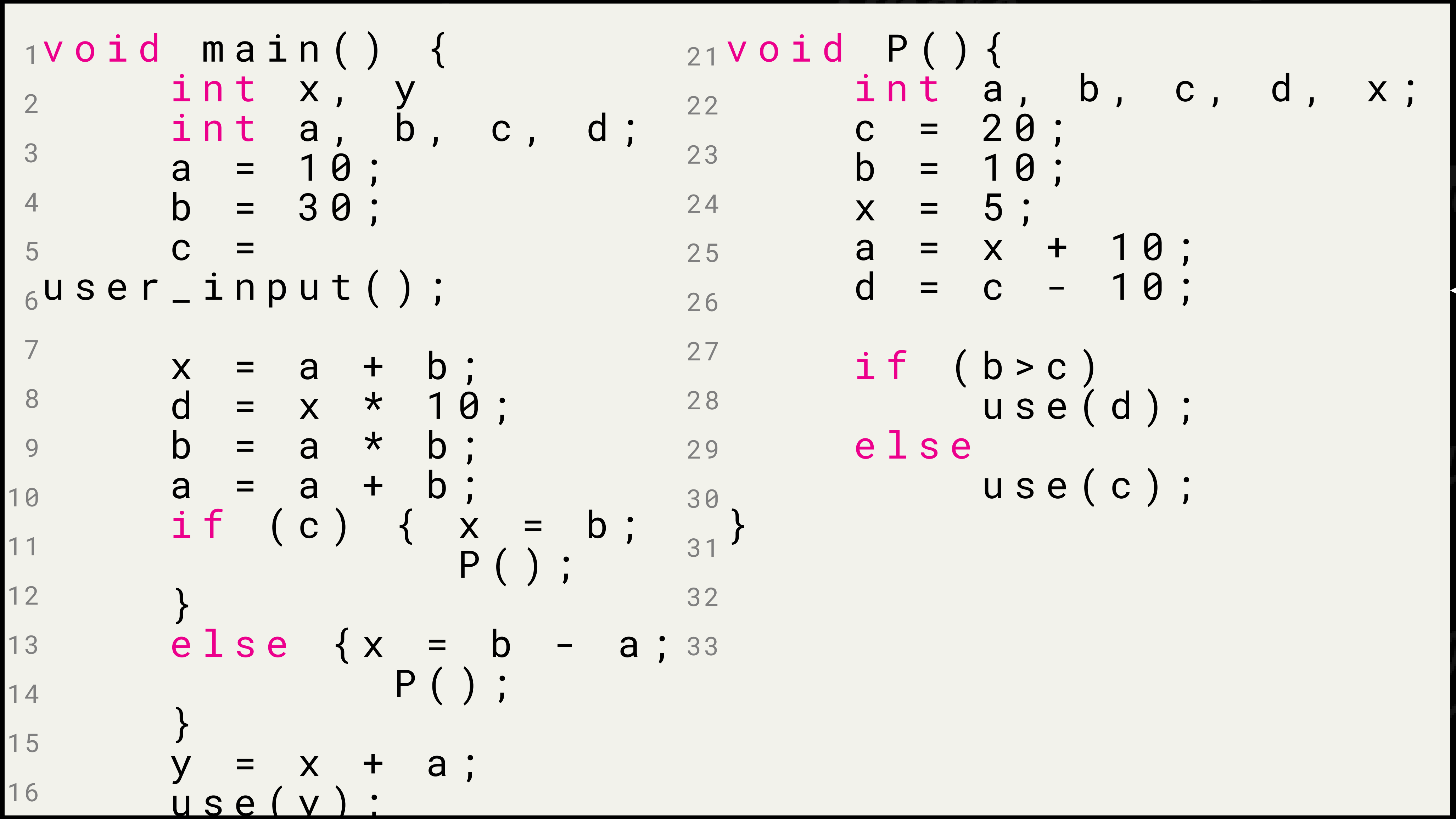
```
#include <stdio.h>
void main()
{
 int a, b;

 scanf("%d", &a);

 if (a > 10)
 b = a;
 else
 b = a + 10;

 printf("%d", b);
}
```

1. Every definition is uniquely renamed
2. Exactly one definition reaches any use of the variable
3. Φ (phi) function acts like a magical operator that comes into play at join nodes (here, node 5) to select any one of the definitions (here, among b_1 & b_2)



- SSA is not as trivial as you might feel it to be.
- It's more than just renaming.
- It makes optimisations and analyses easier, efficient and possible!

- SSA is not as trivial as you might feel it to be.
- It's more than just renaming.
- It makes optimisations and analysis easier, efficient and possible!



```
void main() {
    int x, y
    int a, b, c, d;
    a = 10;
    b = 30;
    c =
user_input();

    x = a + b;
    d = x * 10;
    b = a * b;
    a = a + b;
    if (c) { x = b;
             P();
    }
    else {x = b - a;
           P();
    }
    y = x + a;
    use(y);
```

```
void P(){
    int a, b, c, d, x;
    c = 20;
    b = 10;
    x = 5;
    a = x + 10;
    d = c - 10;

    if (b>c)
        use(d);
    else
        use(c);
}
```

It can get a code of this size

- SSA is not as trivial as you might feel it to be.
- It's more than just renaming.
- It makes optimisations and analysis easier, efficient and possible!

```
void main() {
    int x, y
    int a, b, c, d;
    a = 10;
    b = 30;
    c =
user_input();
    x = a + b;
    d = x * 10;
    b = a * b;
    a = a + b;
    if (c) { x = b;
             P();
    }
    else {x = b - a;
           P();
    }
    y = x + a;
    use(y);
}
```

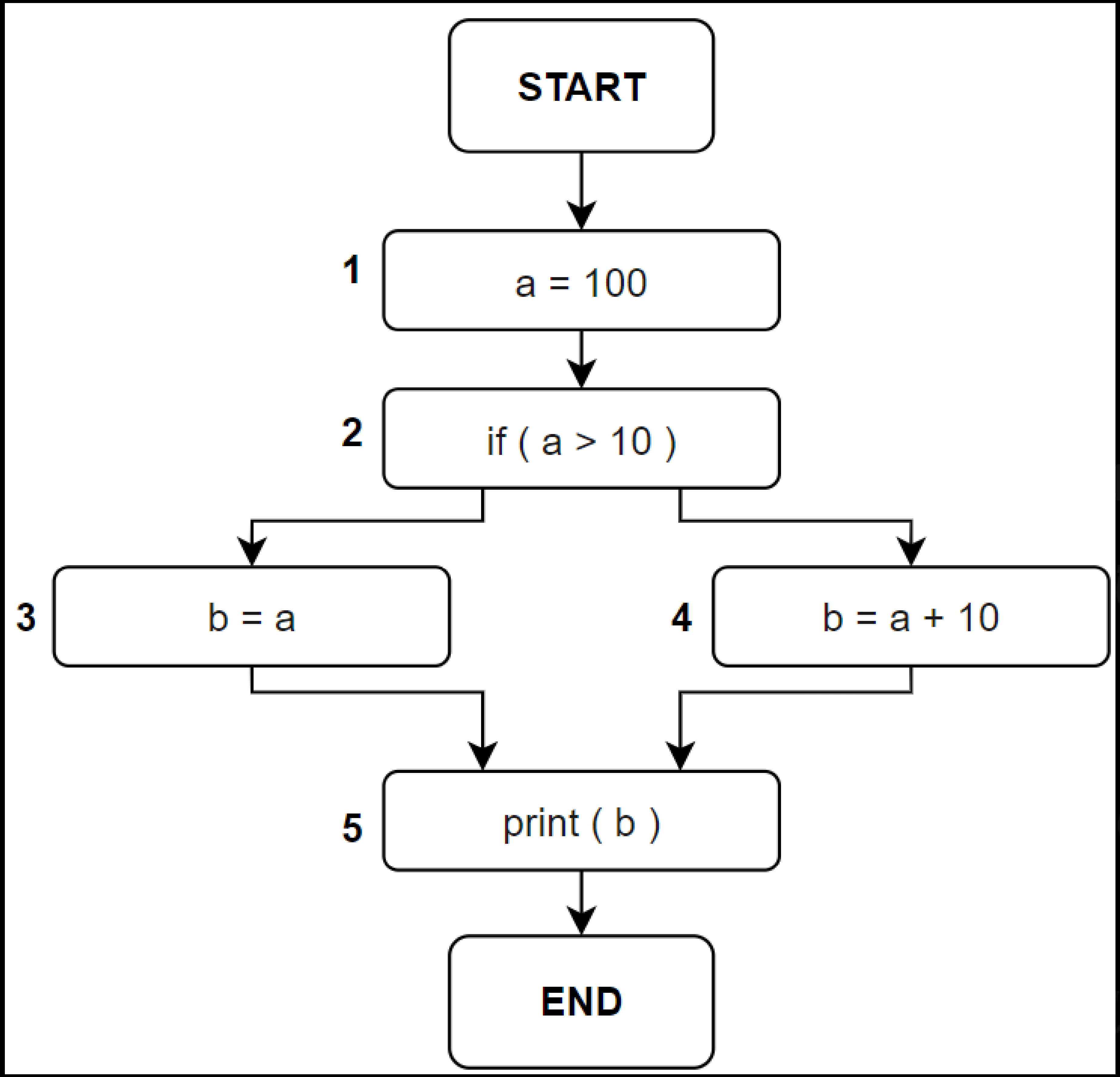
```
void P(){
    int a, b, c, d, x;
    c = 20;
    b = 10;
    x = 5;
    a = x + 10;
    d = c - 10;

    if (b>c)
        use(d);
    else
        use(c);
}
```

Down to this size!

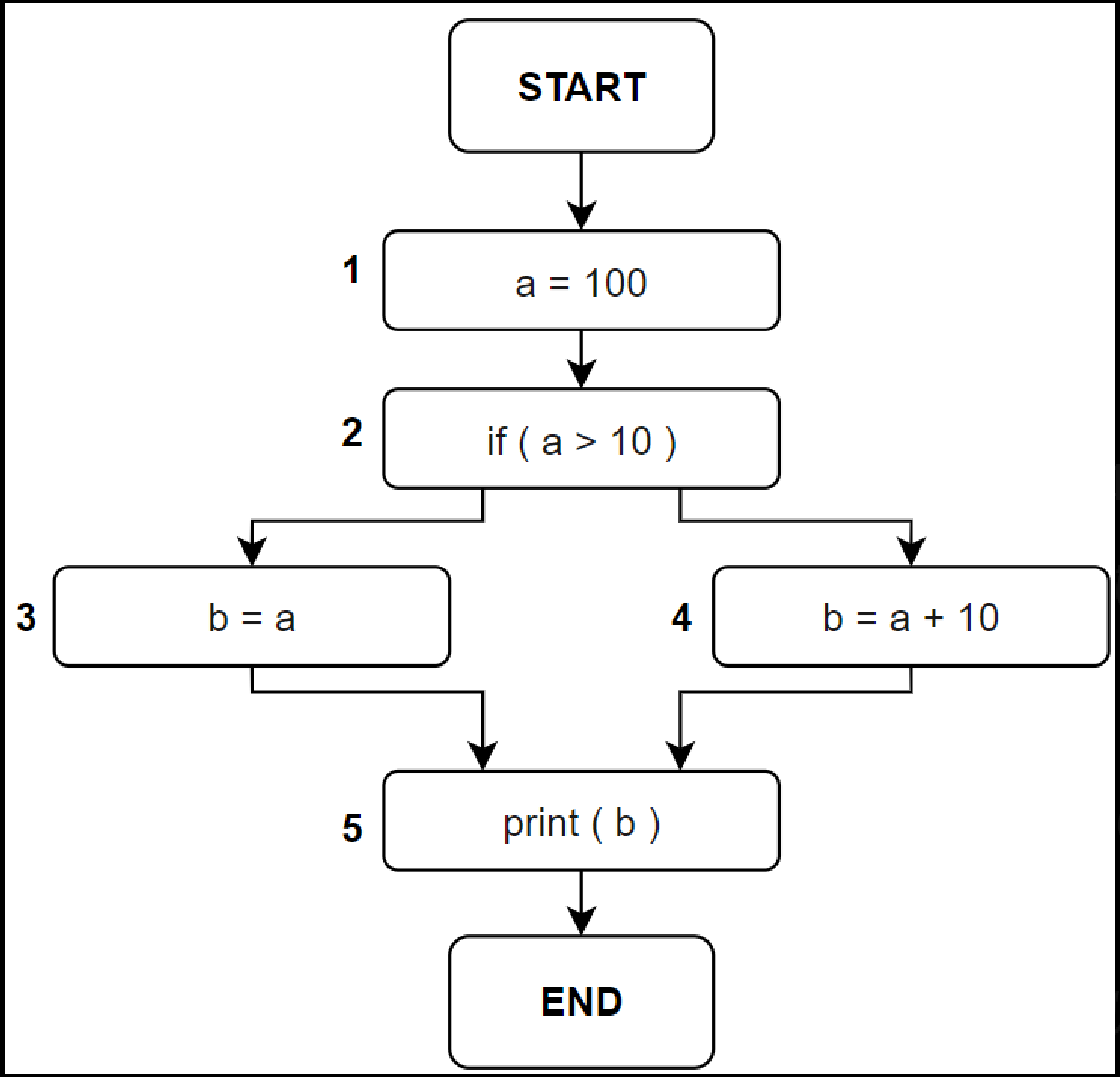
```
void main()
{
    int x, y;
    int a, b, c, d;
    c = user_input();
    if (c){x = 300;
        P();
     }
    else{x = -10;
        P();
    }
    y = x + 310;
    use(y);
}
```

```
void P()
{
    int a,b,c,d,x;
    use(20);
}
```

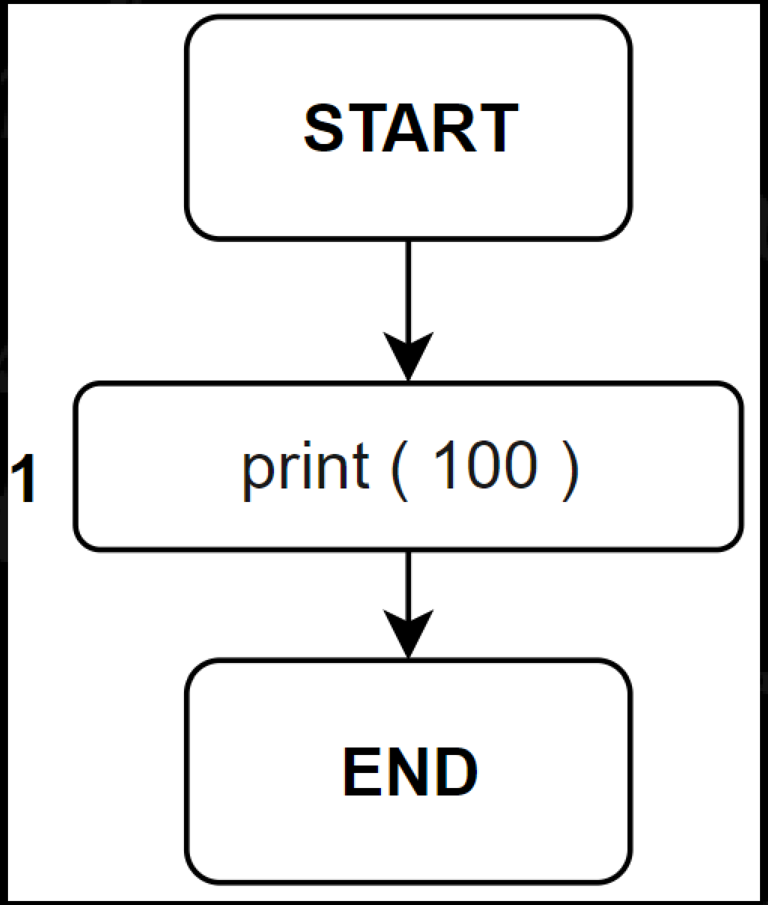
Or a code of this size,



Or a code of this size,



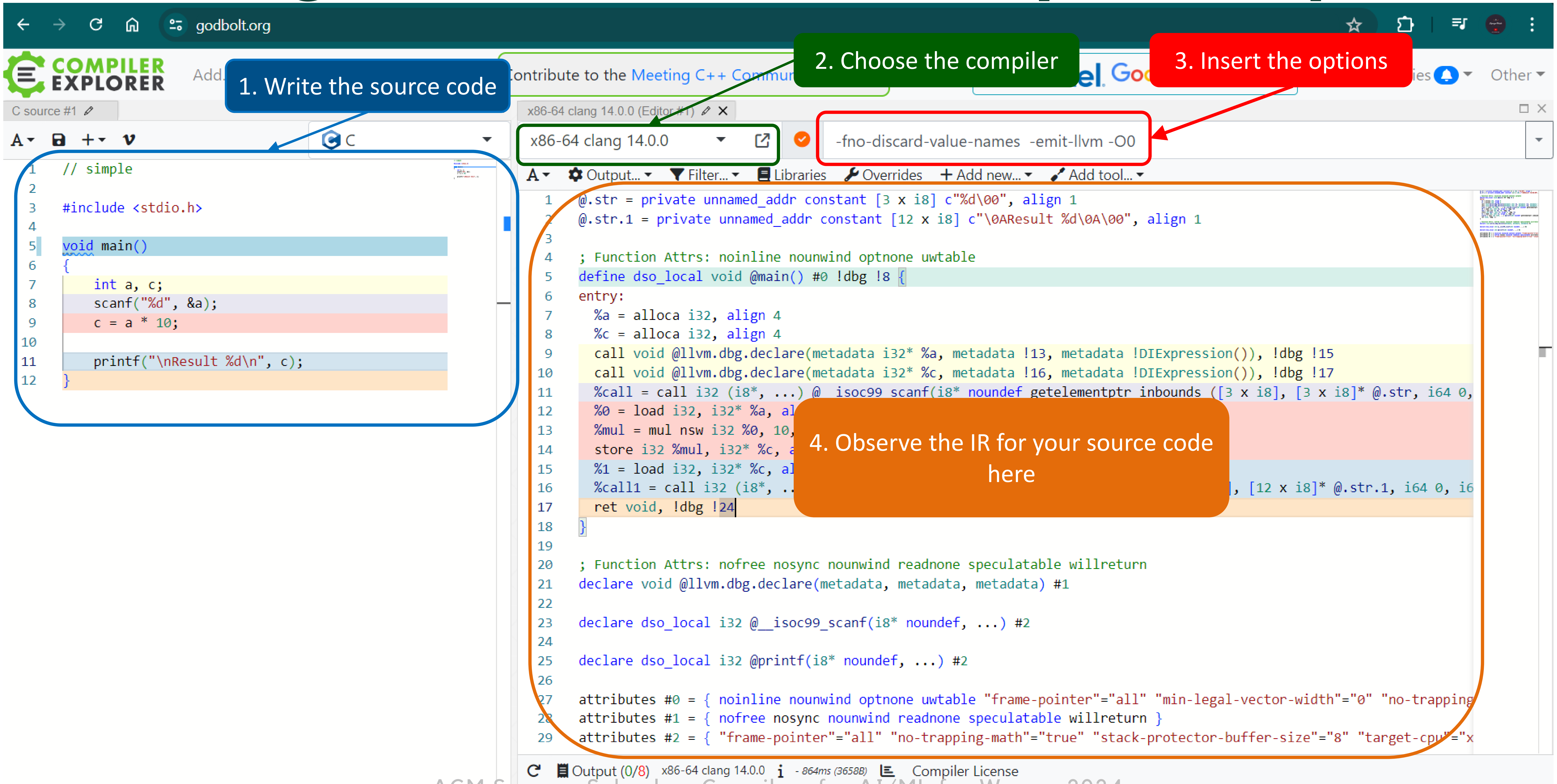
Down to this size!



Lets get back to examining the IR

compiler explorer

Viewing the LLVM IR in compiler explorer



1. Write the source code

2. Choose the compiler

3. Insert the options

4. Observe the IR for your source code here

godbolt.org

C source #1

```c
// simple

#include <stdio.h>

void main()
{
    int a, c;
    scanf("%d", &a);
    c = a * 10;

    printf("\nResult %d\n", c);
}
```

x86-64 clang 14.0.0 (Editor #1)

x86-64 clang 14.0.0

-fno-discard-value-names -emit-llvm -O0

```
@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [12 x i8] c"\0AResult %d\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @main() #0 !dbg !8 {
entry:
  %a = alloca i32, align 4
  %c = alloca i32, align 4
  call void @llvm.dbg.declare(metadata i32* %a, metadata !13, metadata !DIExpression()), !dbg !15
  call void @llvm.dbg.declare(metadata i32* %c, metadata !16, metadata !DIExpression()), !dbg !17
  %call = call i32 (i8*, ...) @__isoc99_scanf(i8* noundef getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i64 0,
  %0 = load i32, i32* %a, al
  %mul = mul nsw i32 %0, 10,
  store i32 %mul, i32* %c, a
  %1 = load i32, i32* %c, al
  %call1 = call i32 (i8*, ..                                        ], [12 x i8]* @.str.1, i64 0, i6
  ret void, !dbg !24
}

; Function Attrs: nofree nosync nounwind readnone speculatable willreturn
declare void @llvm.dbg.declare(metadata, metadata, metadata) #1

declare dso_local i32 @__isoc99_scanf(i8* noundef, ...) #2

declare dso_local i32 @printf(i8* noundef, ...) #2

attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping
attributes #1 = { nofree nosync nounwind readnone speculatable willreturn }
attributes #2 = { "frame-pointer"="all" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x
```

Output (0/8) x86-64 clang 14.0.0 - 864ms (3658B) Compiler License

Resources

- **About LLVM:**
  - https://llvm.org/docs
  - https://llvm.org/docs/LangRef.html
- **Tools to generate and study LLVM IR**
  - A Gentle Introduction to LLVM IR *(a blog)*
  - Compiler Explorer
- **Writing your own pass on LLVM IR**
  - https://llvm.org/docs/WritingAnLLVMNewPMPass.html
  - Tutorial: Writing an LLVM Pass