Compilers 101

LLVM IR

Previously…

Preprocessing

Lexical analysis

Syntax analysis

Semantic analysis

IR Generation

IR Optimization

Optimization

Code generation

Frontend

Middle-end

Backend

Today

- Three address code
- Intermediate representations
- LLVM IR

How to represent computation?

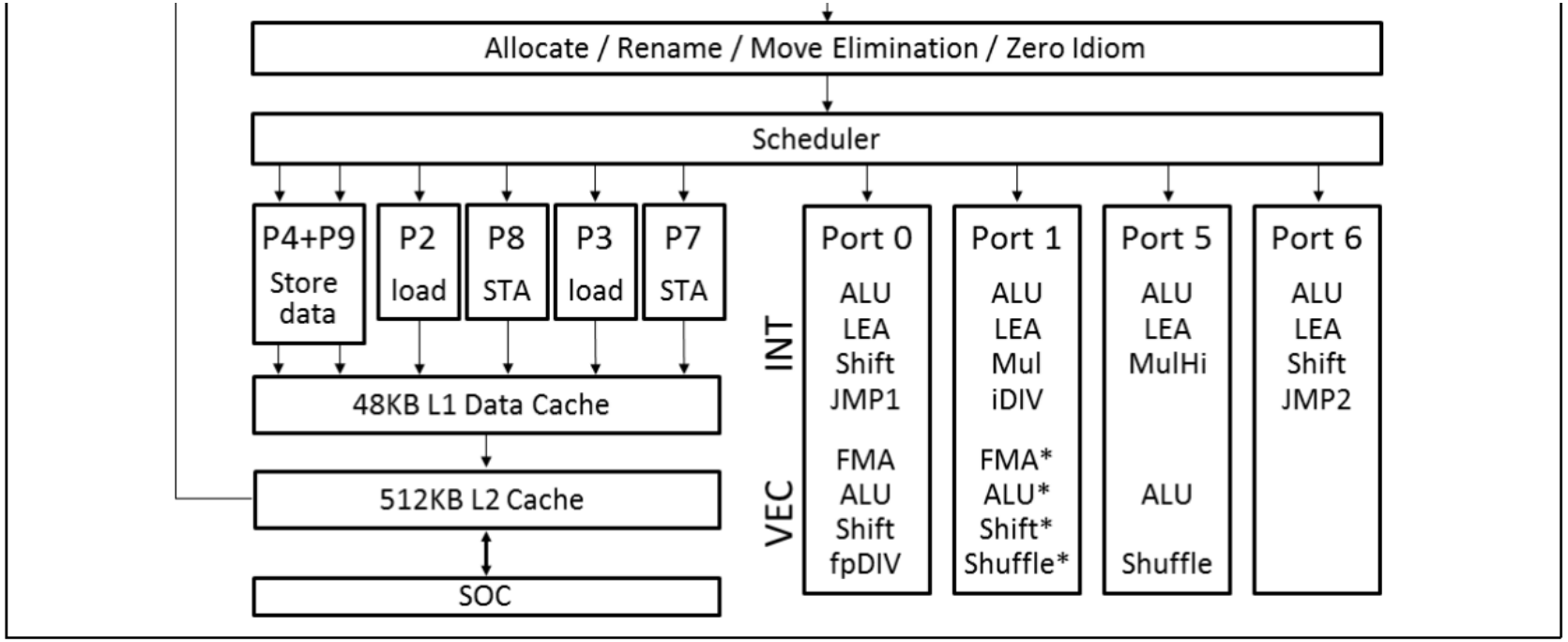


Figure 2-1. Processor Core Pipeline Functionality of the Ice Lake Client Microarchitecture[1]

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Three-address code

- Three-address code is a representation of the computation in the form of a sequence of expressions A := B op C
- Example:

```
A = B + C + D
```

```
t1 = B + C
t2 = t1 + D
A = t2
```

TAC instructions

- The following forms are allowed

- A := constant
- A := B
- A := B op C
- A := constant op B
- A := B op constant
- A := constant op constant

TAC control flow

```
int x, y;
while (x < y) {
    x = x * 2;
}
y = x;
```

```
L0:
    t0 := x >= y;
    if t0 goto L1;
    x := x * 2;
    goto L0;
L1:
    y := x;
```

Three-address code example

```
...

for (i = 0; i < 10; ++i) {
    b[i] = i*i;
}

...
```

```
     t1 := 0                ; initialize i
L1:  if t1 >= 10 goto L2    ; conditional jump
     t2 := t1 * t1          ; square of i
     t3 := t1 * 4           ; word-align address
     t4 := b + t3           ; address to store i*i
     *t4 := t2              ; store through
pointer
     t1 := t1 + 1           ; increase i
     goto L1                ; repeat loop
L2:
```

https://en.wikipedia.org/wiki/Three-address_code

Function calls

```
void bar(int A, int B) {
    …
}

void foo() {
  bar(10, 42);
}
```

```
bar:
    B = pop;
    A = pop;
  …
foo:
    push 10;
    push 42;
    goto bar;
    …
```

Single static assignment

- SSA is a property of intermediate representation, which requires that each variable to be assigned exactly once
- Benefits

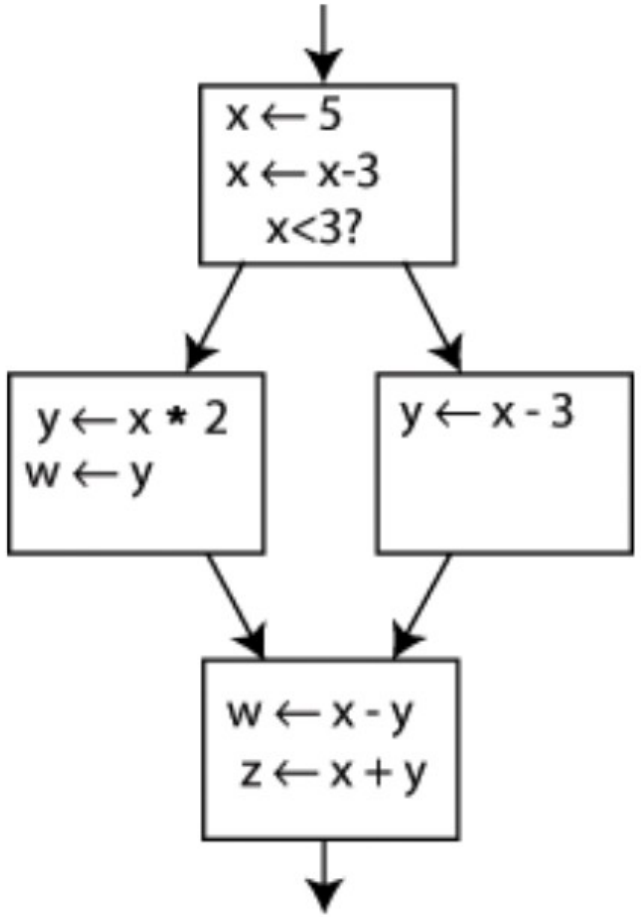
```
y := 1
y := 2
x := y
```

```
y1 := 1
y2 := 2
x1 := y2
```

https://en.wikipedia.org/wiki/Static_single_assignment_form

Converting to SSA

- Basically, you replace each assignment target with a new variable
- Problem: branching
- Solution: phi instructions
- $z = \varphi(x, y)$



Computing minimal SSA

- Dominance frontier: a node B is in the dominance frontier of a node A if A does not strictly dominate B, but does dominate some immediate predecessor of B, or if node A is an immediate predecessor of B.
- If the node A defines a certain variable, then that definition will reach every node A dominates. When we leave those nodes and enter a dominance frontier, we must account for other flows.

Def-use analysis

Def-use analysis is a technique used in compilers to determine the points in the program where variables are defined (def) and used (use).

- Definition (Def): A point in the program where a variable is assigned a value.
- Use (Use): A point in the program where the value of a variable is read.

This analysis helps in:

- Identifying dead code (code that is defined but never used).
- Optimizing away redundant computations.
- Facilitating more complex analyses like reaching definitions, live variable analysis, and more.

Intermediate representations (IRs)

LLVM

- MLIR
- LLVM IR
- (g)MIR

GCC

- GENERIC
- GIMPLE
- RTL

Intermediate representations of LLVM infrastructure

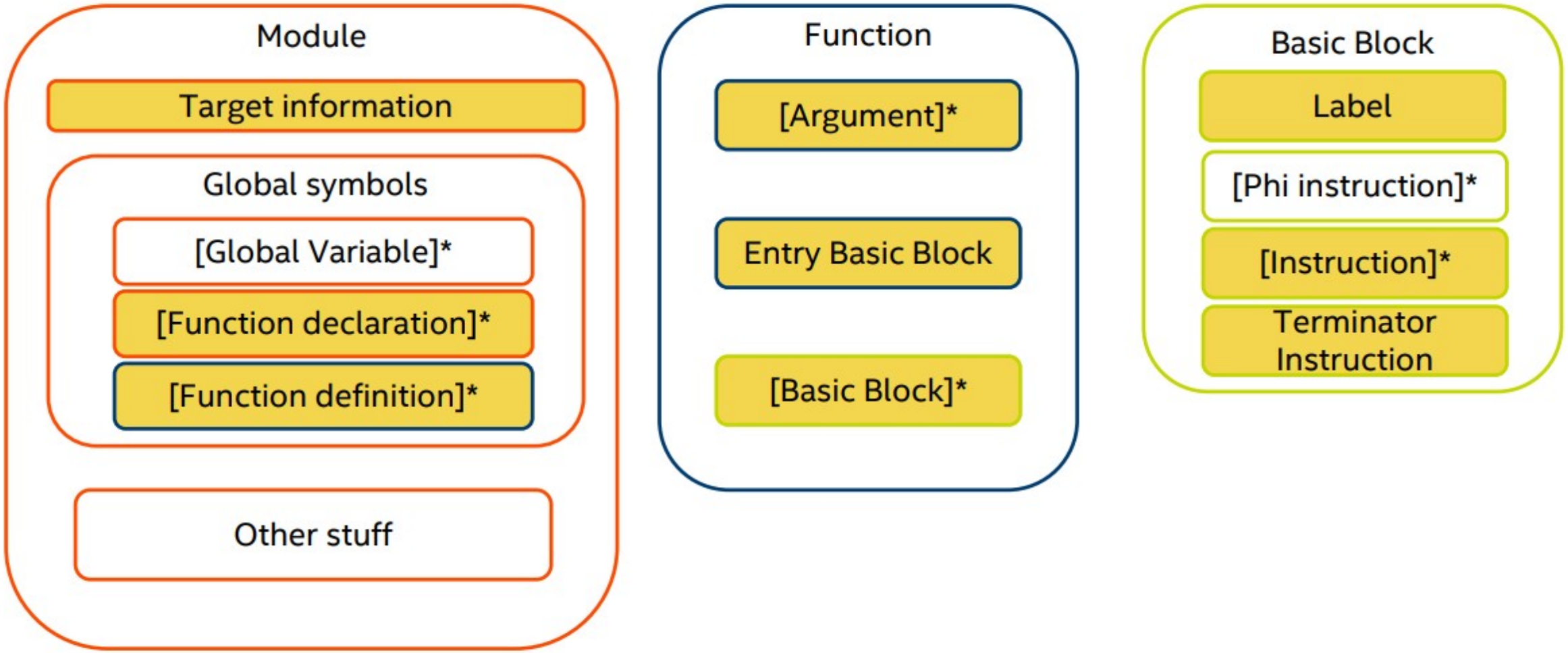
- MLIR
- **LLVM IR**
- (g)MIR

Other IRs will be covered later as a part of this course as well

LLVM IR

- SSA, infinite registers
- Functions
- Metadata
- Textual and binary representation
- Full reference: https://llvm.org/docs/LangRef.html

Simplified IR layout



https://llvm.org/devmtg/2019-04/slides/Tutorial-Bridgers-LLVM_IR_tutorial.pdf

LLVM IR Example

```
int square(int x) {
    return x * x;
}
```

```
; ModuleID = '/app/example.cpp'
source_filename = "/app/example.cpp"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: mustprogress nofree norecurse nosync nounwind readnone willreturn uwtable
define dso_local noundef i32 @_Z6squarei(i32 noundef %0) local_unnamed_addr #0 !dbg !7 {
  call void @llvm.dbg.value(metadata i32 %0, metadata !13, metadata !DIExpression()), !dbg
  %2 = mul nsw i32 %0, %0, !dbg !15
  ret i32 %2, !dbg !16
}

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

attributes #0 = { mustprogress nofree norecurse nosync nounwind readnone willreturn uwtable
attributes #1 = { nofree nosync nounwind readnone speculatable willreturn }

!llvm.dbg.cu = !{!0}
!llvm.module.flags = !{!2, !3, !4, !5}
!llvm.ident = !{!6}

!0 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !1, producer: "clang v
!1 = !DIFile(filename: "/app/example.cpp", directory: "/app", checksumkind: CSK_MD5, checks
!2 = !{i32 7, !"Dwarf Version", i32 5}
!3 = !{i32 2, !"Debug Info Version", i32 3}
!4 = !{i32 1, !"wchar_size", i32 4}
!5 = !{i32 7, !"uwtable", i32 2}
!6 = !{!"clang version 15.0.0 (https://github.com/llvm/llvm-project.git 1c0fc1f074ea5231235
!7 = distinct !DISubprogram(name: "square", linkageName: "_Z6squarei", scope: !8, file: !8,
!8 = !DIFile(filename: "example.cpp", directory: "/app", checksumkind: CSK_MD5, checksum: "
!9 = !DISubroutineType(types: !10)
!10 = !{!11, !11}
!11 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!12 = !{!13}
!13 = !DILocalVariable(name: "x", arg: 1, scope: !7, file: !8, line: 1, type: !11)
!DILocation(line: 0, scope: !7)
```

https://godbolt.org/z/sqo4aG7Gd

Global variables

```
1   int global = 1;
```

```
; ModuleID = '/app/example.cpp'
source_filename = "/app/example.cpp"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@global = dso_local local_unnamed_addr global i32 1, align 4, !dbg !0

!llvm.dbg.cu = !{!2}
!llvm.module.flags = !{!7, !8, !9, !10}
!llvm.ident = !{!11}

!0 = !DIGlobalVariableExpression(var: !1, expr: !DIExpression())
!1 = distinct !DIGlobalVariable(name: "global", scope: !2, file: !5, line: 1, type: !6, isLoc
!2 = distinct !DICompileUnit(language: DW_LANG_C_plus_plus_14, file: !3, producer: "clang ver
!3 = !DIFile(filename: "/app/example.cpp", directory: "/app", checksumkind: CSK_MD5, checksum
!4 = !{!0}
!5 = !DIFile(filename: "example.cpp", directory: "/app", checksumkind: CSK_MD5, checksum: "02
!6 = !DIBasicType(name: "int", size: 32, encoding: DW_ATE_signed)
!7 = !{i32 7, !"Dwarf Version", i32 5}
!8 = !{i32 2, !"Debug Info Version", i32 3}
!9 = !{i32 1, !"wchar_size", i32 4}
!10 = !{i32 7, !"uwtable", i32 2}
!11 = !{!"clang version 15.0.0 (https://github.com/llvm/llvm-project.git 1c0fc1f074ea52312356
```

https://godbolt.org/z/4hebzfe6s

Structures

```
struct Foo {
    int a;
    float b;
    double c;
};

void foo(Foo &f) {
    f.a = 10;
}
```

```
%struct.Foo = type { i32, float, double }

; Function Attrs: mustprogress nofree norecurse nosync nounwind willreturn writeonly
define dso_local void @_Z3fooR3Foo(%struct.Foo* nocapture noundef nonnull writeonly
  call void @llvm.dbg.value(metadata %struct.Foo* %0, metadata !21, metadata !DIExpr
  %2 = getelementptr inbounds %struct.Foo, %struct.Foo* %0, i64 0, i32 0, !dbg !23
  store i32 10, i32* %2, align 8, !dbg !24, !tbaa !25
  ret void, !dbg !32
}
```

https://godbolt.org/z/898E47xGf

Nested structures

```
1  struct Bar {
2      int a;
3  };
4
5  struct Foo {
6      int a;
7      float b;
8      double c;
9      Bar d;
10 };
```

```
%struct.Foo = type { i32, float, double, %struct.Bar }
%struct.Bar = type { i32 }
```

https://godbolt.org/z/jxcvxM57f

Functions

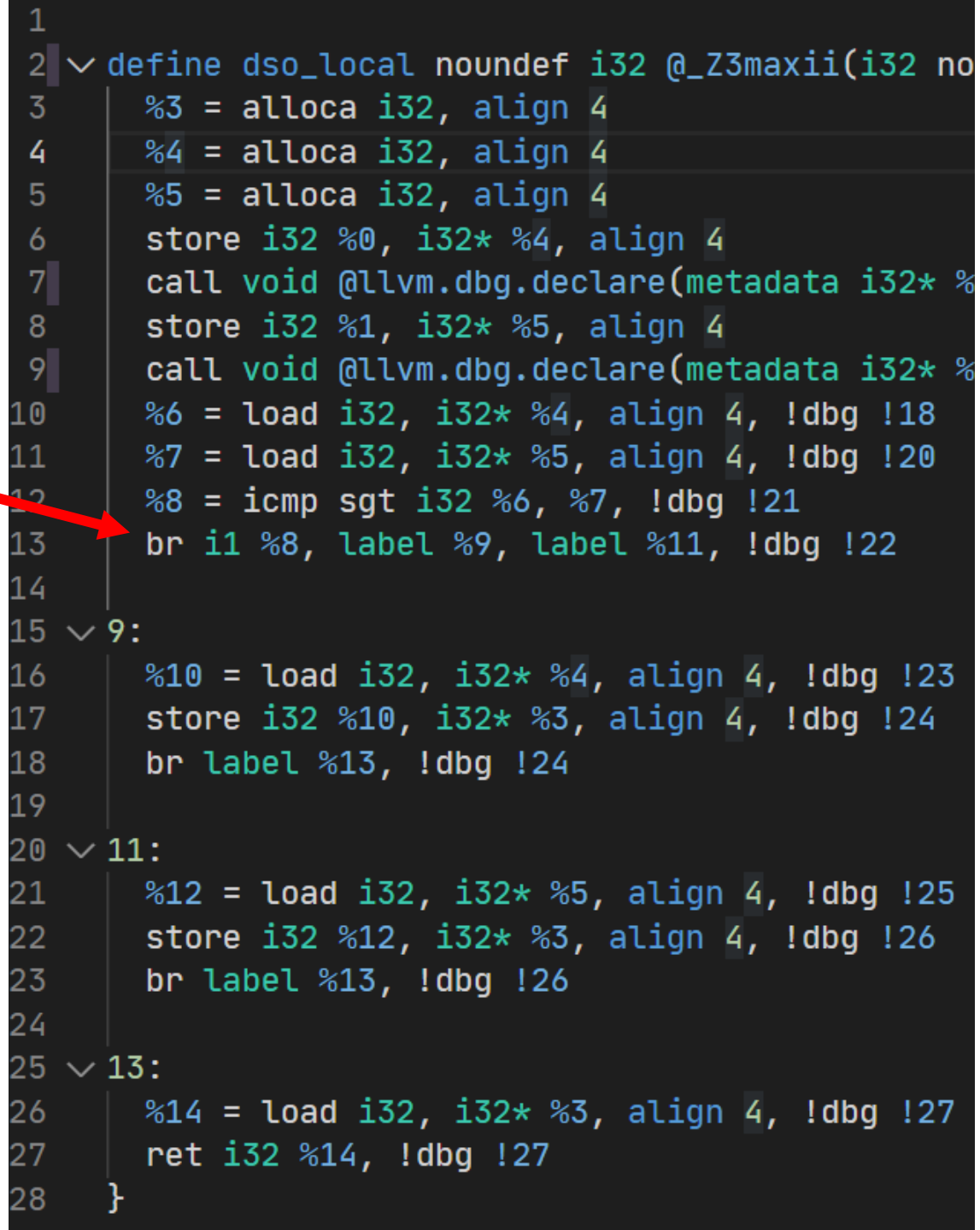
```
11  void foo(Foo &f) {
12  }
13
14  void foo(Bar &b) {
15  }
```

```
; Function Attrs: mustprogress nofree norecurse nosync nounw
define dso_local void @_Z3fooR3Foo(%struct.Foo* nocapture no
  call void @llvm.dbg.value(metadata %struct.Foo* %0, metada
  ret void, !dbg !16
}

; Function Attrs: mustprogress nofree norecurse nosync nounw
define dso_local void @_Z3fooR3Bar(%struct.Bar* nocapture no
  call void @llvm.dbg.value(metadata %struct.Bar* %0, metada
  ret void, !dbg !25
}
```

https://godbolt.org/z/jxcvxM57f

Branches

```
int max(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```



```
1
2  define dso_local noundef i32 @_Z3maxii(i32 no
3    %3 = alloca i32, align 4
4    %4 = alloca i32, align 4
5    %5 = alloca i32, align 4
6    store i32 %0, i32* %4, align 4
7    call void @llvm.dbg.declare(metadata i32* %
8    store i32 %1, i32* %5, align 4
9    call void @llvm.dbg.declare(metadata i32* %
10   %6 = load i32, i32* %4, align 4, !dbg !18
11   %7 = load i32, i32* %5, align 4, !dbg !20
12   %8 = icmp sgt i32 %6, %7, !dbg !21
13   br i1 %8, label %9, label %11, !dbg !22
14
15 9:
16   %10 = load i32, i32* %4, align 4, !dbg !23
17   store i32 %10, i32* %3, align 4, !dbg !24
18   br label %13, !dbg !24
19
20 11:
21   %12 = load i32, i32* %5, align 4, !dbg !25
22   store i32 %12, i32* %3, align 4, !dbg !26
23   br label %13, !dbg !26
24
25 13:
26   %14 = load i32, i32* %3, align 4, !dbg !27
27   ret i32 %14, !dbg !27
28 }
```

https://godbolt.org/z/GbqhabvM8

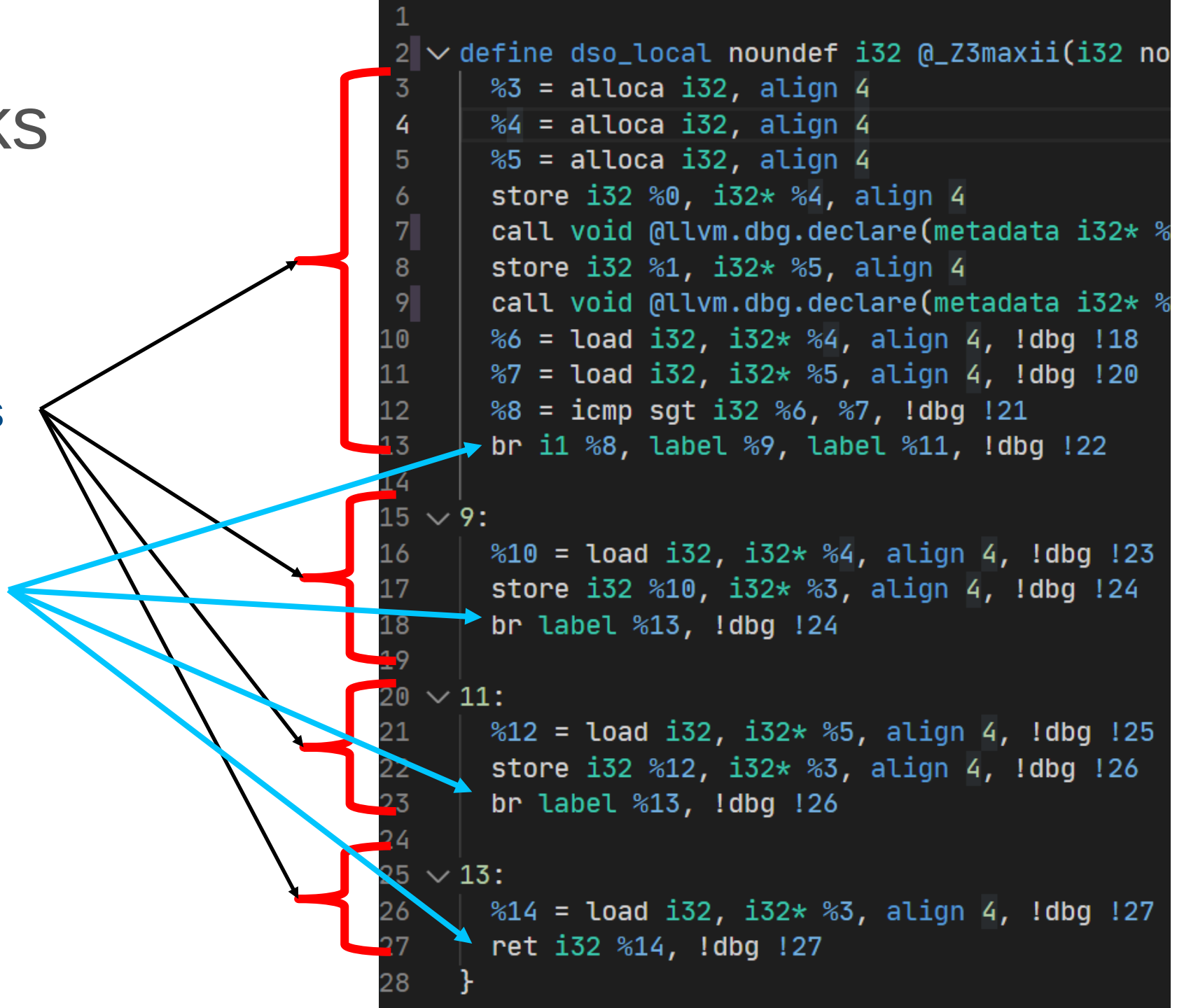
Basic blocks

- A basic block is a sequence of consecutive statements in a program.
- Execution in a basic block flows linearly (no jumps, branches, or labels except at the start and the end).
- Basic block has a single entry and a single exit.

Basic blocks

Basic blocks

Terminators



```
1
2  define dso_local noundef i32 @_Z3maxii(i32 no
3    %3 = alloca i32, align 4
4    %4 = alloca i32, align 4
5    %5 = alloca i32, align 4
6    store i32 %0, i32* %4, align 4
7    call void @llvm.dbg.declare(metadata i32* %
8    store i32 %1, i32* %5, align 4
9    call void @llvm.dbg.declare(metadata i32* %
10   %6 = load i32, i32* %4, align 4, !dbg !18
11   %7 = load i32, i32* %5, align 4, !dbg !20
12   %8 = icmp sgt i32 %6, %7, !dbg !21
13   br i1 %8, label %9, label %11, !dbg !22
14
15 9:
16   %10 = load i32, i32* %4, align 4, !dbg !23
17   store i32 %10, i32* %3, align 4, !dbg !24
18   br label %13, !dbg !24
19
20 11:
21   %12 = load i32, i32* %5, align 4, !dbg !25
22   store i32 %12, i32* %3, align 4, !dbg !26
23   br label %13, !dbg !26
24
25 13:
26   %14 = load i32, i32* %3, align 4, !dbg !27
27   ret i32 %14, !dbg !27
28 }
```

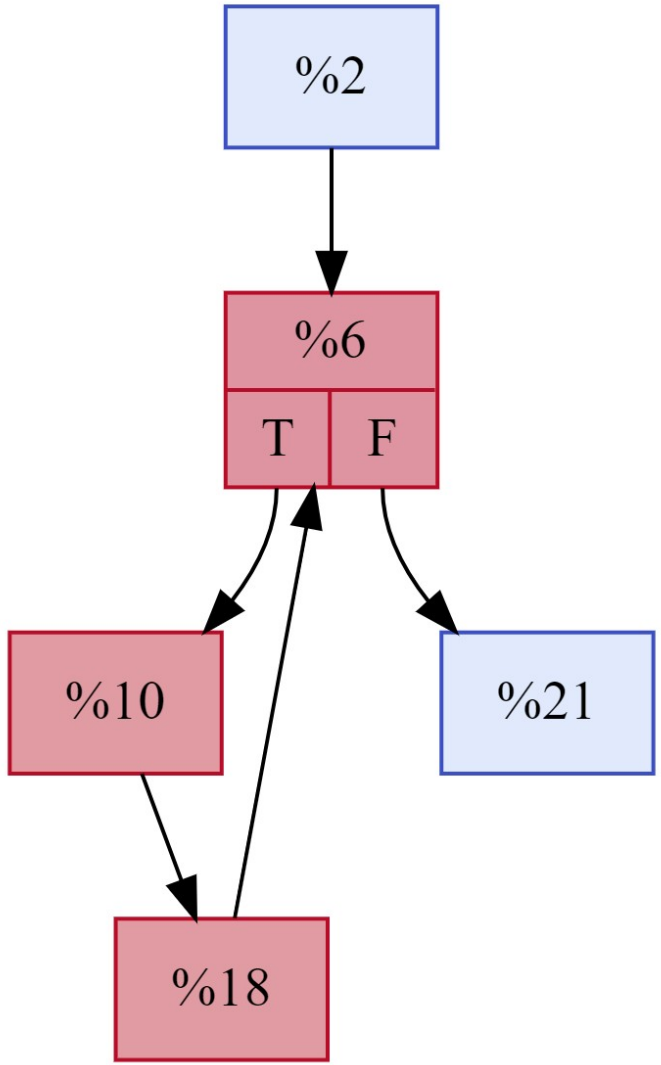
Loops

```
1  void foo(int *i, int N) {
2      for (int j = 0; j < N; j++) {
3          *i += *i * j;
4      }
5  }
```

```
define dso_local void @_Z3fooPii(i32* noundef %
  %3 = alloca i32*, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32* %0, i32** %3, align 8
  call void @llvm.dbg.declare(metadata i32** %3
  store i32 %1, i32* %4, align 4
  call void @llvm.dbg.declare(metadata i32* %4,
  call void @llvm.dbg.declare(metadata i32* %5,
  store i32 0, i32* %5, align 4, !dbg !21
  br label %6, !dbg !22

6:
  %7 = load i32, i32* %5, align 4, !dbg !23
  %8 = load i32, i32* %4, align 4, !dbg !25
  %9 = icmp slt i32 %7, %8, !dbg !26
  br i1 %9, label %10, label %21, !dbg !27

10:
  %11 = load i32*, i32** %3, align 8, !dbg !28
  %12 = load i32, i32* %11, align 4, !dbg !30
  %13 = load i32, i32* %5, align 4, !dbg !31
  %14 = mul nsw i32 %12, %13, !dbg !32
  %15 = load i32*, i32** %3, align 8, !dbg !33
  %16 = load i32, i32* %15, align 4, !dbg !34
  %17 = add nsw i32 %16, %14, !dbg !34
  store i32 %17, i32* %15, align 4, !dbg !34
  br label %18, !dbg !35

18:
  %19 = load i32, i32* %5, align 4, !dbg !36
  %20 = add nsw i32 %19, 1, !dbg !36
  store i32 %20, i32* %5, align 4, !dbg !36
  br label %6, !dbg !37, !llvm.loop !38

21:
  ret void, !dbg !41
}
```

Loops. Control flow graph

```
1  void foo(int *i, int N) {
2      for (int j = 0; j < N; j++) {
3          *i += *i * j;
4      }
5  }
```



CFG for '_Z3fooPii' function

```
define dso_local void @_Z3fooPii(i32* noundef %
  %3 = alloca i32*, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32* %0, i32** %3, align 8
  call void @llvm.dbg.declare(metadata i32** %3
  store i32 %1, i32* %4, align 4
  call void @llvm.dbg.declare(metadata i32* %4,
  call void @llvm.dbg.declare(metadata i32* %5,
  store i32 0, i32* %5, align 4, !dbg !21
  br label %6, !dbg !22

6:
  %7 = load i32, i32* %5, align 4, !dbg !23
  %8 = load i32, i32* %4, align 4, !dbg !25
  %9 = icmp slt i32 %7, %8, !dbg !26
  br i1 %9, label %10, label %21, !dbg !27

10:
  %11 = load i32*, i32** %3, align 8, !dbg !28
  %12 = load i32, i32* %11, align 4, !dbg !30
  %13 = load i32, i32* %5, align 4, !dbg !31
  %14 = mul nsw i32 %12, %13, !dbg !32
  %15 = load i32*, i32** %3, align 8, !dbg !33
  %16 = load i32, i32* %15, align 4, !dbg !34
  %17 = add nsw i32 %16, %14, !dbg !34
  store i32 %17, i32* %15, align 4, !dbg !34
  br label %18, !dbg !35

18:
  %19 = load i32, i32* %5, align 4, !dbg !36
  %20 = add nsw i32 %19, 1, !dbg !36
  store i32 %20, i32* %5, align 4, !dbg !36
  br label %6, !dbg !37, !llvm.loop !38

21:
  ret void, !dbg !41
}
```

Single assignment

```
1   void bar(int);
2   void foo(int k) {
3       int a;
4       if (k > 10) {
5           a = 100;
6       } else {
7           a = k / 2;
8       }
9       bar(a);
10  }
```

https://godbolt.org/z/x6bf1fxxv

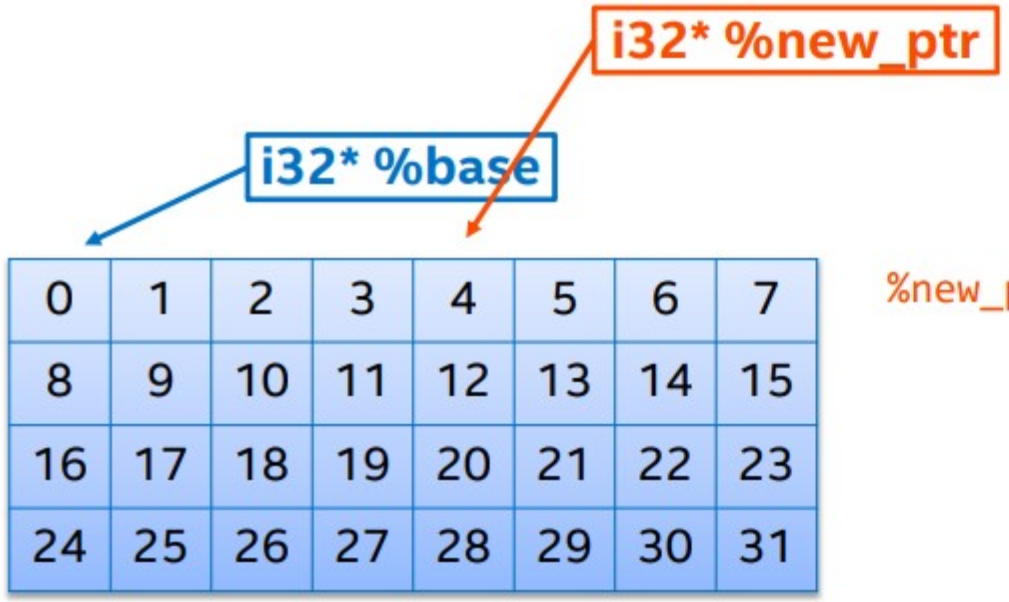
Memory SSA

```
1   void bar(int);
2   void foo(int k) {
3       int a;
4       if (k > 10) {
5           a = 100;
6       } else {
7           a = k / 2;
8       }
9       bar(a);
10  }
```

```
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 %0, i32* %2, align 4
  call void @llvm.dbg.declare(metadata i32* %2,
  call void @llvm.dbg.declare(metadata i32* %3,
  %4 = load i32, i32* %2, align 4, !dbg !18
  %5 = icmp sgt i32 %4, 10, !dbg !20
  br i1 %5, label %6, label %7, !dbg !21

6:
  store i32 100, i32* %3, align 4, !dbg !22
  br label %10, !dbg !24

7:
  %8 = load i32, i32* %2, align 4, !dbg !25
  %9 = sdiv i32 %8, 2, !dbg !27
  store i32 %9, i32* %3, align 4, !dbg !28
  br label %10

10:
  %11 = load i32, i32* %3, align 4, !dbg !29
  call void @_Z3bari(i32 noundef %11), !dbg !30
  ret void, !dbg !31
```

https://godbolt.org/z/x6bf1fxxv

Manipulating pointers




```
%new_ptr = getelementptr i32, i32* %base, i32 4
```

"Offset by 4 **elements of the base type**"

aka GEP

LLVM IR bitcode

Binary Representation of LLVM IR

Use Cases:

- distribution (for usage or inspection by other LLVM based tools).
- intermediate step in a build process that involves further optimization or cross-compilation.
- Just-In-Time (JIT) compilation scenarios.

LLVM IR bitcode: tooling

- Emit bitcode:
  - `clang -emit-llvm -c source.c -o source.bc`
- Converting LLVM IR Bitcode to Human-Readable Format
  - `llvm-dis source.bc -o source.ll`
- Converting Human-Readable LLVM IR to Bitcode
  - `llvm-as source.ll -o source.bc`
- Linking Multiple LLVM Bitcode Files
  - `llvm-link file1.bc file2.bc -o combined.bc`
- Execute LLVM IR bitcode
  - `lli source.bc`

Inspecting LLVM IR

- Inspecting LLVM IR bitcode from clang
  - `clang++ -emit-llvm -o file.bc file.cpp`
- Inspecting LLVM IR from clang
  - `clang++ -emit-llvm -o file.ll file.cpp -S`
- Displaying CFG
  - `opt -analyze -dot-cfg-only`

Processing LLVM IR

- Run optimizations
  - `opt -O2 source.bc -o optimized.bc`
  - `opt -passes='<pass-pipeline>' source.bc -o optimized.bc`
- Compile down to assembler
  - `llc -filetype=obj source.bc -o source.s -S`
- Compile down to binary (object file)
  - `llc -filetype=obj source.bc -o source.o`

Note: both `.bc` (binary) and `.ll` (text) are accepted here

Comparing LLVM modules

```
❯ llvm-diff test.ll out.ll
function @llvm.dbg.value exists only in right module
in function _Z3fooi:
  in block %1 / %1:
    >   call void @llvm.dbg.value(metadata i32 %0, metadata !14, metadata !DIExpression()), !dbg !15
    >   %2 = icmp sgt i32 %0, 10, !dbg !16
    >   br i1 %2, label %3, label %4, !dbg !18
    <   %2 = alloca i32, align 4
    <   %3 = alloca i32, align 4
    <   store i32 %0, i32* %2, align 4
    <   call void @llvm.dbg.declare(metadata i32* %2, metadata !14, metadata !DIExpression()), !dbg !15
    <   call void @llvm.dbg.declare(metadata i32* %3, metadata !16, metadata !DIExpression()), !dbg !17
    <   %4 = load i32, i32* %2, align 4, !dbg !18
    <   %5 = icmp sgt i32 %4, 10, !dbg !20
    <   br i1 %5, label %6, label %7, !dbg !21
```

Debugging LLVM IR

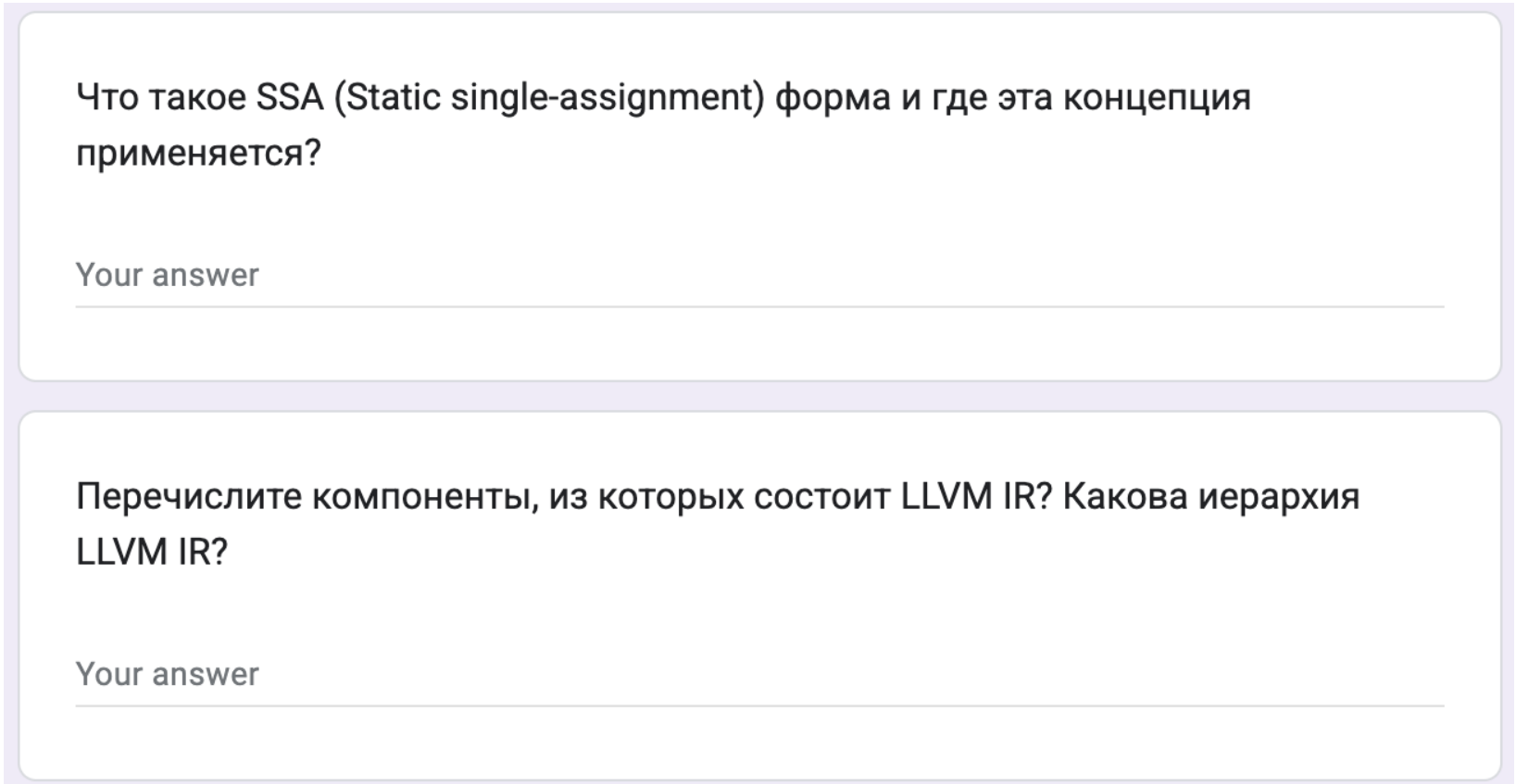
- bugpoint
- https://www.llvm.org/docs/CommandGuide/bugpoint.html
- llvm-reduce
- https://llvm.org/docs/CommandGuide/llvm-reduce.html
- https://www.youtube.com/watch?v=n1jDj7J9N8c

Next time…

- LLVM Passes
- IR Optimizations

Test

https://forms.gle/KNKvREkPhq3QNHj18



Backup: me@gooddoog.ru

Extra materials

- Mapping high-level constructs to LLVM IR - https://github.com/f0rki/mapping-high-level-constructs-to-llvm-ir
- The often misunderstood GEP (GetElementPtr) instruction - https://www.llvm.org/docs/GetElementPtr.html
- LLVM IR Tutorial - Phis, GEPs and other things, oh my! - Vince Bridgers (Intel Corporation), Felipe de Azevedo Piovezan (Intel Corporation) - https://www.youtube.com/watch?v=m8G_S5LwlTo