

# A Look Inside the Go Compiler

*... what the hell is actually happening under the hood?*



# The (Standard) Compiler Pipeline

What's the point?

Compiler's goal is to...

transform source language into *semantically equivalent* target language.

In the case of Go compiler...

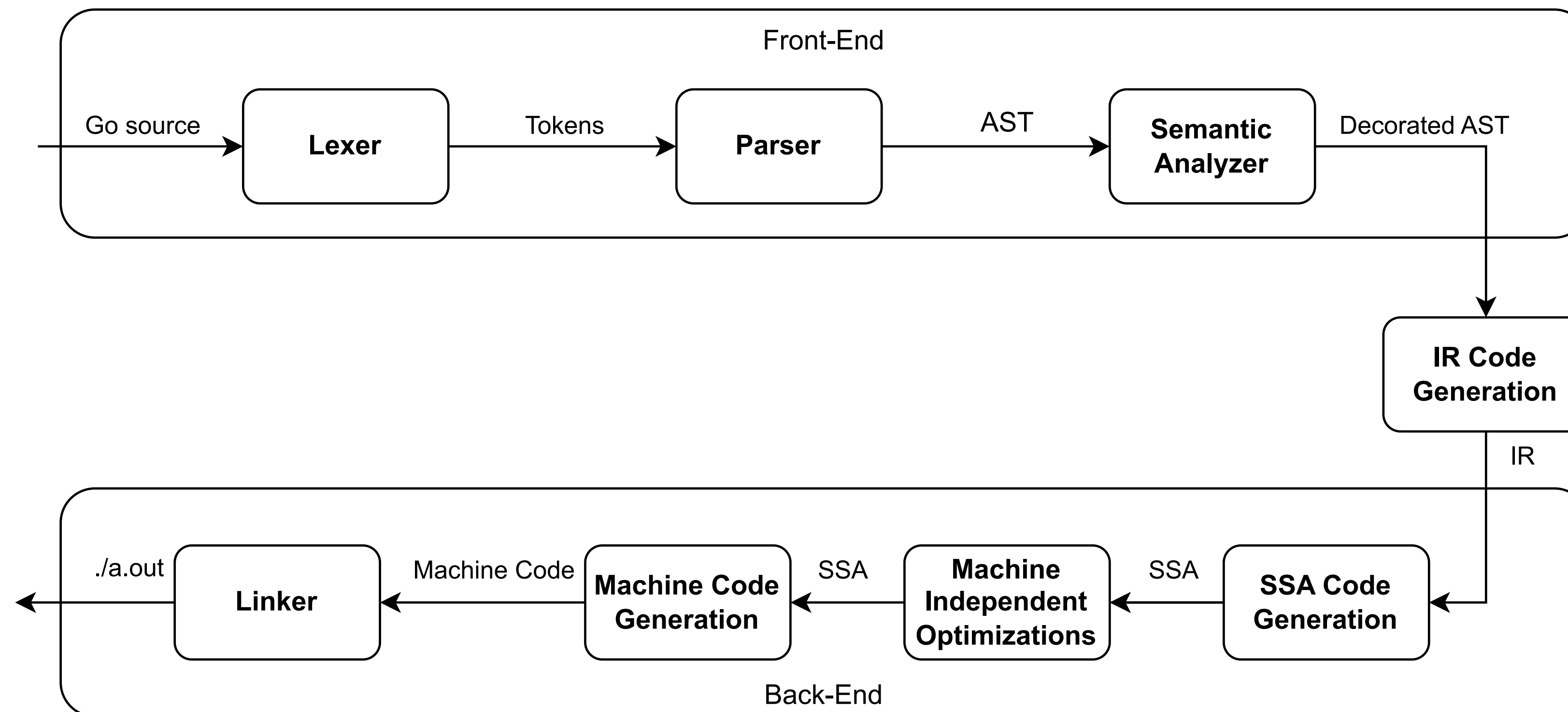
source language is the Go source code,

target language is the assembly.



# The (Standard) Compiler Pipeline

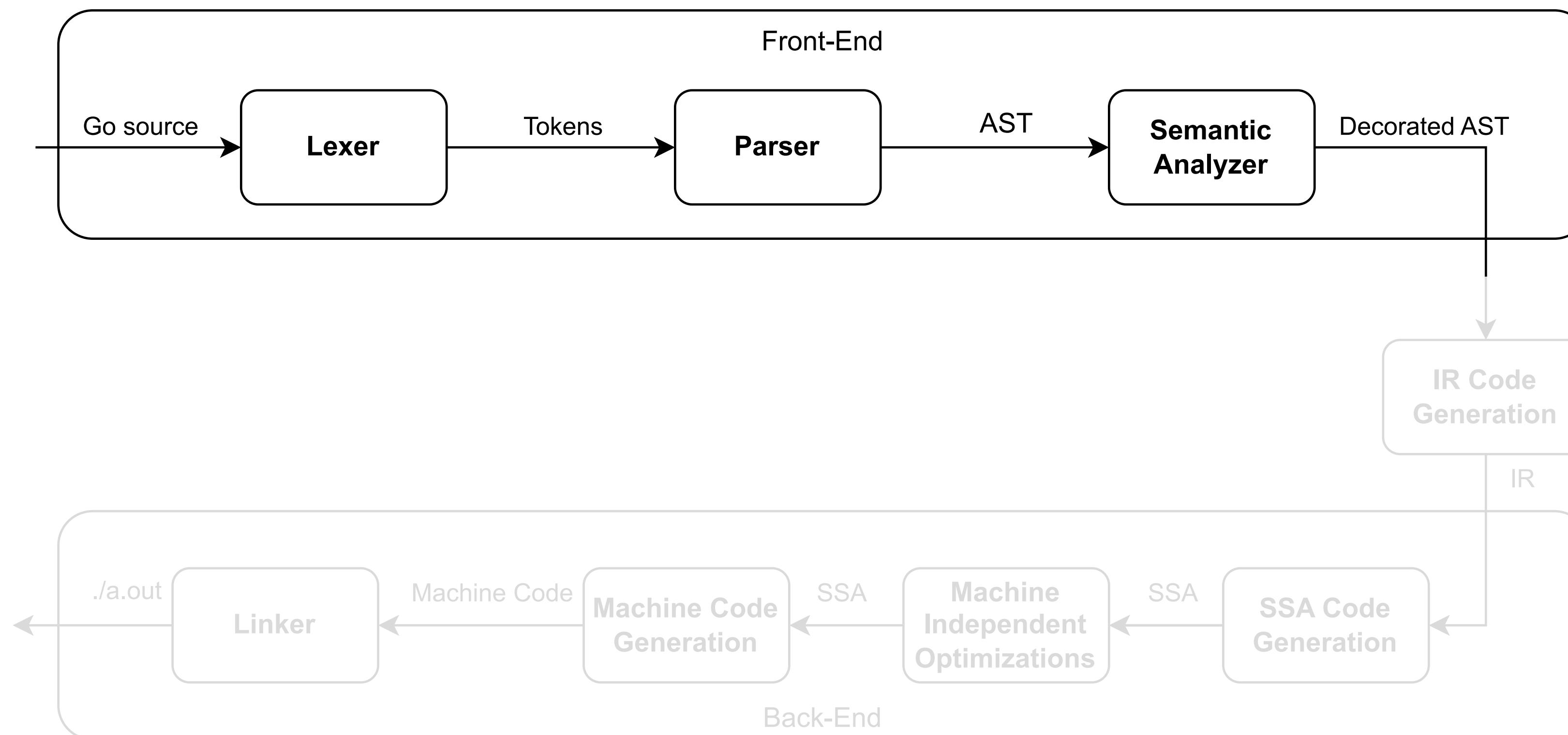
## High-Level Overview





# Frontend

Ensures your code makes sense. Then molds it into backend-ready form.





# Frontend

## Lexing (aka tokenizing aka scanning)

Scans the code, slices it into stream of *tokens*, and keeps track of where everything came from.

- Token is a smallest meaningful unit in the source code.
- It has a *type*, *value*, and *position info*.



# Frontend

## Token Definiton

```
1 // go/src/go/token/token.go
2
3 // Token is the set of lexical tokens of the Go programming language.
4 type Token int
5
6 // The list of tokens.
7 const (
8     // Special tokens
9     ILLEGAL Token = iota
10    EOF
11    COMMENT
12
13    literal_beg
14    // Identifiers and basic type literals
15    // (these tokens stand for classes of literals)
16    IDENT    // main
17    INT      // 12345
18    FLOAT    // 123.45
19    IMAG     // 123.45i
20    CHAR     // 'a'
21    STRING   // "abc"
22    literal_end
```

```
1 // go/src/go/token/position.go
2
3 // Position describes an arbitrary source position
4 // including the file, line, and column location.
5 // A Position is valid if the line number is > 0.
6 type Position struct {
7     Filename string // filename, if any
8     Offset   int    // offset, starting at 0
9     Line     int    // line number, starting at 1
10    Column    int    // column number, starting at 1 (byte count)
11 }
```

```
1 // go/src/go/scanner/scanner.go
2
3 func (s *Scanner) Scan() (pos token.Pos, tok token.Token, lit string)
```



# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```

*s.Scan()* → ("main.go:1:1", PACKAGE, "package")





# Frontend

## Example: Scanning

```
1 package main                s.Scan()      ("main.go:1:9", IDENT, "main")
2
3 const ANSWER = 42
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```





# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```

`s.Scan()` → `("main.go:1:13", SEMICOLON, "\n")`



# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42 s.Scan() ("main.go:3:1", CONST, "const")
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```



# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42      s.Scan() ("main.go:3:7", IDENT, "ANSWER")
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```



# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42          s.Scan() ("main.go:3:14", ASSIGN, "=")
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x = ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```



# Frontend

## Example: Scanning

```
1 package main
2
3 const ANSWER = 42
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13
14     return false
15 }
16
17 func main() {
18     greet(42)
19 }
20
```

`s.Scan()` → `("main.go:3:16", INT, "42")`



# Frontend Parser

Builds structure from tokens, turning flat code into an *abstract syntax tree* according to the grammar.

- Hand-written recursive descent parser with limited lookahead.
- Parser closely mirrors the formal grammar of the language.





# Frontend

## Sample Parser Code

### Function declarations

A function declaration binds an identifier, the *function name*, to a function.

```
FunctionDecl = "func" FunctionName [ TypeParameters ] Signature [ FunctionBody ] .  
FunctionName = identifier .  
FunctionBody = Block .
```

[https://go.dev/ref/spec#Function\\_declarations](https://go.dev/ref/spec#Function_declarations)

```
1 // go/src/go/parser/parser.go  
2  
3 func (p *parser) parseFuncDecl() *ast.FuncDecl {  
4     // ...  
5     pos := p.expect(token.FUNC)  
6     // ...  
7     ident := p.parseIdent()  
8     // ...  
9     params := p.parseParameters(false)  
10    results := p.parseParameters(true)  
11  
12    var body *ast.BlockStmt  
13    switch p.tok {  
14    case token.LBRACE:  
15        body = p.parseBody()  
16        p.expectSemi()  
17    case token.SEMICOLON:  
18        p.next()  
19        // ...  
20    default:  
21        p.expectSemi()  
22    }  
23  
24    decl := &ast.FuncDecl{  
25        Doc:  doc,  
26        Recv: recv,  
27        Name: ident,  
28        Type: &ast.FuncType{  
29            Func:      pos,  
30            TypeParams: tparams,  
31            Params:    params,  
32            Results:   results,  
33        },  
34        Body: body,  
35    }  
36    return decl  
37 }
```






# Frontend Parsed AST


```
1 package main.go:111
2 - NamePos: main.go:111
3 - Name: "main"
4 - Name: "main"
5
6 Decl: []ast.Decl (len = 3) {
7   0: *ast.GenDecl {
8     TokPos: main.go:311
9     Tok: const
10    Lparen: -
11    Specs: []ast.Spec (len = 3) {
12      0: *ast.ValueSpec {
13        Names: []*ast.Ident (len = 1) {
14          0: *ast.Ident {
15            NamePos: main.go:377
16            Name: "ANSWER"
17            Obj: *ast.Object {
18              Kind: const
19              Name: "ANSWER"
20              Decl: *Obj @ 22
21              Data: 0
22            }
23          }
24        }
25        Values: []ast.Expr (len = 1) {
26          0: *ast.BasicLit {
27            ValuePos: main.go:316
28            Kind: INT
29            Value: "42"
30          }
31        }
32      }
33    }
34    Rparen: -
35  }
36  1: *ast.FuncDecl {
37    Decl: *ast.CommentGroup {
38      List: []ast.Comment {
39        0: *ast.Comment {
40          Pos: main.go:511
41          Text: "// Greet the user depending on the parameter `x`."
42        }
43      }
44      Name: *ast.Ident {
45        NamePos: main.go:616
46        Name: "greet"
47        Obj: *ast.Object {
48          Kind: func
49          Name: "greet"
50          Decl: *Obj @ 36
51        }
52      }
53      Type: *ast.FuncType {
54        Params: *ast.FieldList {
55          Opening: main.go:611
56          List: []ast.Field {
57            0: *ast.Field {
58              Names: []*ast.Ident (len = 1) {
59                0: *ast.Ident {
60                  NamePos: main.go:612
61                  Name: "x"
62                  Obj: *ast.Object {
63                    Kind: var
64                    Name: "x"
65                    Decl: *Obj @ 59
66                  }
67                }
68              }
69            }
70          }
71          Type: *ast.Ident {
72            NamePos: main.go:614
73            Name: "int"
74          }
75        }
76        Results: *ast.FieldList {
77          Opening: -
78          List: []ast.Field {
79            0: *ast.Field {
80              Type: *ast.Ident {
81                NamePos: main.go:619
82                Name: "bool"
83              }
84            }
85          }
86        }
87      }
88    }
89    Body: *ast.BlockStmt {
90      Lbrace: main.go:624
91      List: []ast.Stmt {
92        0: *ast.IfStmt {
93          Cond: *ast.BinaryExpr {
94            X: *ast.Ident {
95              NamePos: main.go:915
96              Name: "x"
97              Obj: *Obj @ 64
98            }
99            OpPos: main.go:917
100           Op: ==
101           Y: *ast.Ident {
102             NamePos: main.go:918
103             Name: "ANSWER"
104             Obj: *Obj @ 37
105           }
106         }
107         Body: *ast.BlockStmt {
108           Lbrace: main.go:917
109           List: []ast.Stmt {
110             0: *ast.ExprStmt {
111               Expr: *ast.CallExpr {
112                 Fun: *ast.Ident {
113                   NamePos: main.go:1813
114                   Name: "println"
115                 }
116                 Lparen: main.go:1818
117                 Args: []ast.Expr {
118                   0: *ast.BasicLit {
119                     ValuePos: main.go:1811
120                     Kind: STRING
121                     Value: "\"hello, gophercamp!\""
122                   }
123                 }
124               }
125             }
126             1: *ast.ReturnStmt {
127               Result: *ast.BasicLit {
128                 ValuePos: main.go:1811
129                 Kind: STRING
130                 Value: "\"hello, gophercamp!\""
131               }
132             }
133           }
134         }
135       }
136     }
137     Rbrace: main.go:1811
138   }
139   2: *ast.FuncDecl {
140     Name: *ast.Ident {
141       NamePos: main.go:1716
142       Name: "main"
143       Obj: *ast.Object {
144         Kind: func
145         Name: "main"
146         Decl: *Obj @ 158
147       }
148       Type: *ast.FuncType {
149         Params: *ast.FieldList {
150           Opening: main.go:1718
151           List: []ast.Field {
152             0: *ast.Field {
153               Name: "x"
154               Obj: *ast.Object {
155                 Kind: var
156                 Name: "x"
157                 Decl: *Obj @ 159
158               }
159             }
160           }
161           Results: *ast.FieldList {
162             Opening: main.go:1718
163             List: []ast.Field {
164               0: *ast.Field {
165                 Type: *ast.Ident {
166                   NamePos: main.go:1813
167                   Name: "bool"
168                 }
169               }
170             }
171           }
172         }
173         Body: *ast.BlockStmt {
174           Lbrace: main.go:1723
175           List: []ast.Stmt {
176             0: *ast.ExprStmt {
177               Expr: *ast.CallExpr {
178                 Fun: *ast.Ident {
179                   NamePos: main.go:1812
180                   Name: "greet"
181                 }
182                 Lparen: main.go:1817
183                 Args: []ast.Expr {
184                   0: *ast.BasicLit {
185                     ValuePos: main.go:1818
186                     Kind: INT
187                     Value: "42"
188                   }
189                 }
190               }
191             }
192             1: *ast.ReturnStmt {
193               Result: *ast.BasicLit {
194                 ValuePos: main.go:1811
195                 Kind: STRING
196                 Value: "\"hello, gophercamp!\""
197               }
198             }
199           }
200         }
201       }
202     }
203     Rbrace: main.go:1811
204   }
205   3: *ast.FuncDecl {
206     Name: *ast.Ident {
207       NamePos: main.go:1716
208       Name: "main"
209       Obj: *ast.Object {
210         Kind: func
211         Name: "main"
212         Decl: *Obj @ 158
213       }
214       Type: *ast.FuncType {
215         Params: *ast.FieldList {
216           Opening: main.go:1718
217           List: []ast.Field {
218             0: *ast.Field {
219               Name: "x"
220               Obj: *ast.Object {
221                 Kind: var
222                 Name: "x"
223                 Decl: *Obj @ 159
224               }
225             }
226           }
227           Results: *ast.FieldList {
228             Opening: main.go:1718
229             List: []ast.Field {
230               0: *ast.Field {
231                 Type: *ast.Ident {
232                   NamePos: main.go:1813
233                   Name: "bool"
234                 }
235               }
236             }
237           }
238         }
239         Body: *ast.BlockStmt {
240           Lbrace: main.go:1723
241           List: []ast.Stmt {
242             0: *ast.ExprStmt {
243               Expr: *ast.CallExpr {
244                 Fun: *ast.Ident {
245                   NamePos: main.go:1812
246                   Name: "greet"
247                 }
248                 Lparen: main.go:1817
249                 Args: []ast.Expr {
250                   0: *ast.BasicLit {
251                     ValuePos: main.go:1818
252                     Kind: INT
253                     Value: "42"
254                   }
255                 }
256               }
257             }
258             1: *ast.ReturnStmt {
259               Result: *ast.BasicLit {
260                 ValuePos: main.go:1811
261                 Kind: STRING
262                 Value: "\"hello, gophercamp!\""
263               }
264             }
265           }
266         }
267       }
268     }
269     Rbrace: main.go:1811
270   }
271 }
272
273 FileStart: main.go:111
274 FileEnd: main.go:282
275 Scope: *ast.Scope {
276   Objects: map[string]*ast.Object {
277     "ANSWER": *Obj @ 22
278     "greet": *Obj @ 48
279     "main": *Obj @ 162
280   }
281   Variables: []*ast.Ident {
282     0: *Obj @ 71
283     1: *Obj @ 83
284     2: *Obj @ 116
285     3: *Obj @ 133
286     4: *Obj @ 148
287   }
288   Comments: []ast.CommentGroup {
289     0: *Obj @ 37
290     1: *ast.CommentGroup {
291       List: []ast.Comment {
292         0: *ast.Comment {
293           Pos: main.go:712
294           Text: "// Greet the user only if the argument is"
295         }
296         1: *ast.Comment {
297           Pos: main.go:712
298           Text: "// the answer to everything."
299         }
300       }
301     }
302   }
303   Version: ""
304 }
```

```
1 package main
2
3 const ANSWER = 42
4
5 // Greet the user depending on the parameter `x`.
6 func greet(x int) bool {
7     // Greet the user only if the argument is
8     // the answer to everything.
9     if x == ANSWER {
10         println("hello, gophercamp!")
11         return true
12     }
13     return false
14 }
15
16 func main() {
17     greet(42)
18 }
19
```

```
0 *ast.File {
1   . Package: main.go:1:1
2   . Name: *ast.Ident {
3     . NamePos: main.go:1:9
4     . Name: "main"
5   }
6   . Decl: []ast.Decl (len = 3) {
7     . 0: *ast.GenDecl {
8       . TokPos: main.go:3:1
9       . Tok: const
10      . Lparen: -
11      . Specs: []ast.Spec (len = 1) {
12        . 0: *ast.ValueSpec {
13          . Names: []*ast.Ident (len = 1) {
14            . 0: *ast.Ident {
15              . NamePos: main.go:3:7
16              . Name: "ANSWER"
17              . Obj: *ast.Object {
18                . Kind: const
19                . Name: "ANSWER"
20                . Decl: *(obj @ 12)
21                . Data: 0
22              }
23            }
24          }
25          . Values: []ast.Expr (len = 1) {
26            . 0: *ast.BasicLit {
27              . ValuePos: main.go:3:16
28              . Kind: INT
29              . Value: "42"
30            }
31          }
32        }
33      }
34      . Rparen: -
35    }
36    . 1: *ast.FuncDecl {
37      . // ...
38    }
39  }
40}
```

 **Microsoft** Josef Podaný

16

25th April '25 @ Gophercamp 



# Frontend

## Semantic Analyzer

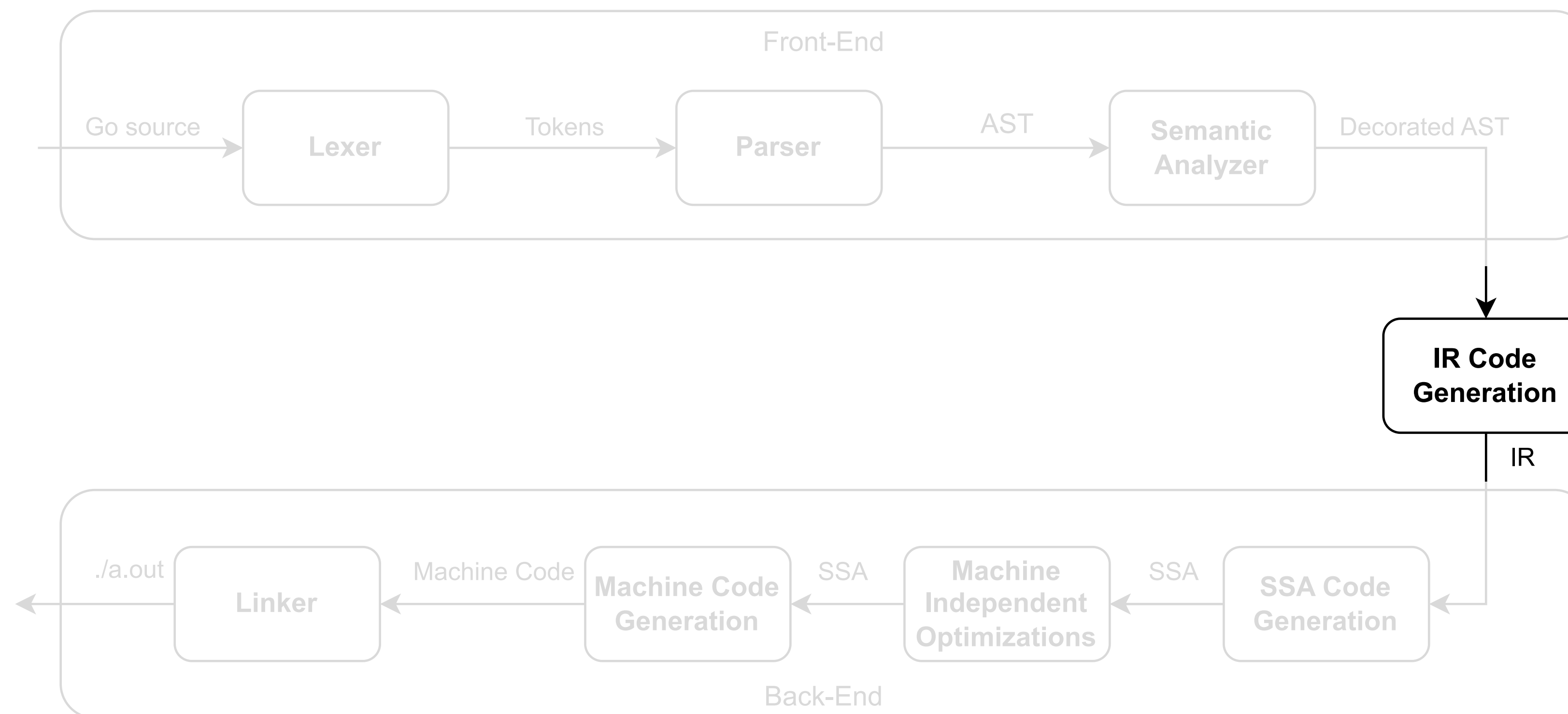
The code may look right—this phase makes sure it *is* right.

- Type inference, type checking and monomorphization of generics
- Scope, definition, name resolution
- Return statement checking
- noding: lowering to *Intermediate Representation*



# Middle-end

Perform Go specific optimizations & desugar Go constructs.





# Middle-end IR Optimization

- Function call inlining
- Devirtualization of known interface method calls
- Escape analysis



# Middle-end

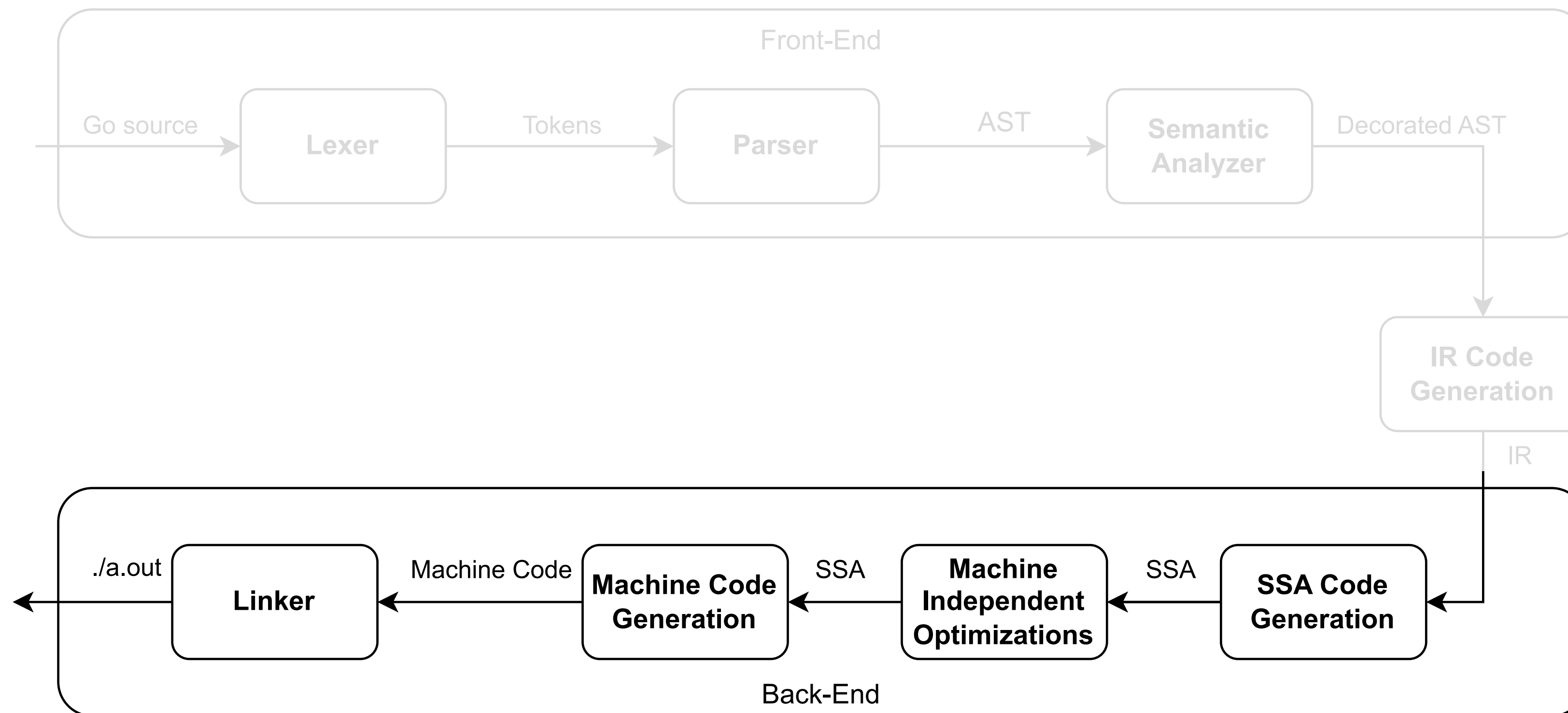
## Walking the IR

- Decompose complex statements into simpler ones.
- Desugar higher-level constructs:
  - Switch statements into binary search or jump tables.
  - Operations on maps and channels replaced with runtime code.



# Backend

Lowers, optimizes, and assembles your code for the target architecture.







# Backend

## Why Custom Backend?

*"It's a small toolchain that we can keep in our heads and make arbitrary changes to, quickly and easily. Honestly, if we'd built on GCC or LLVM, we'd be moving so slowly I'd probably have left the project years ago."*

*[...] Certainly there is a room for improvement, but the custom toolchain is one of the key reasons we've accomplished so much in so little time."*

- Russ Cox in Hacker News thread<sup>1</sup>

<sup>1</sup><https://news.ycombinator.com/item?id=8817990>





# Backend

## Lowering to Static Single Assignment

Static Single Assignment: IR in which each variable is assigned to *exactly once*.

```
1 func example() int {
2     a := 10
3     b := a * 10
4
5     c := 0
6     if b == 100 {
7         c = b
8     } else {
9         c = 42
10    }
11
12    return c
13 }
```

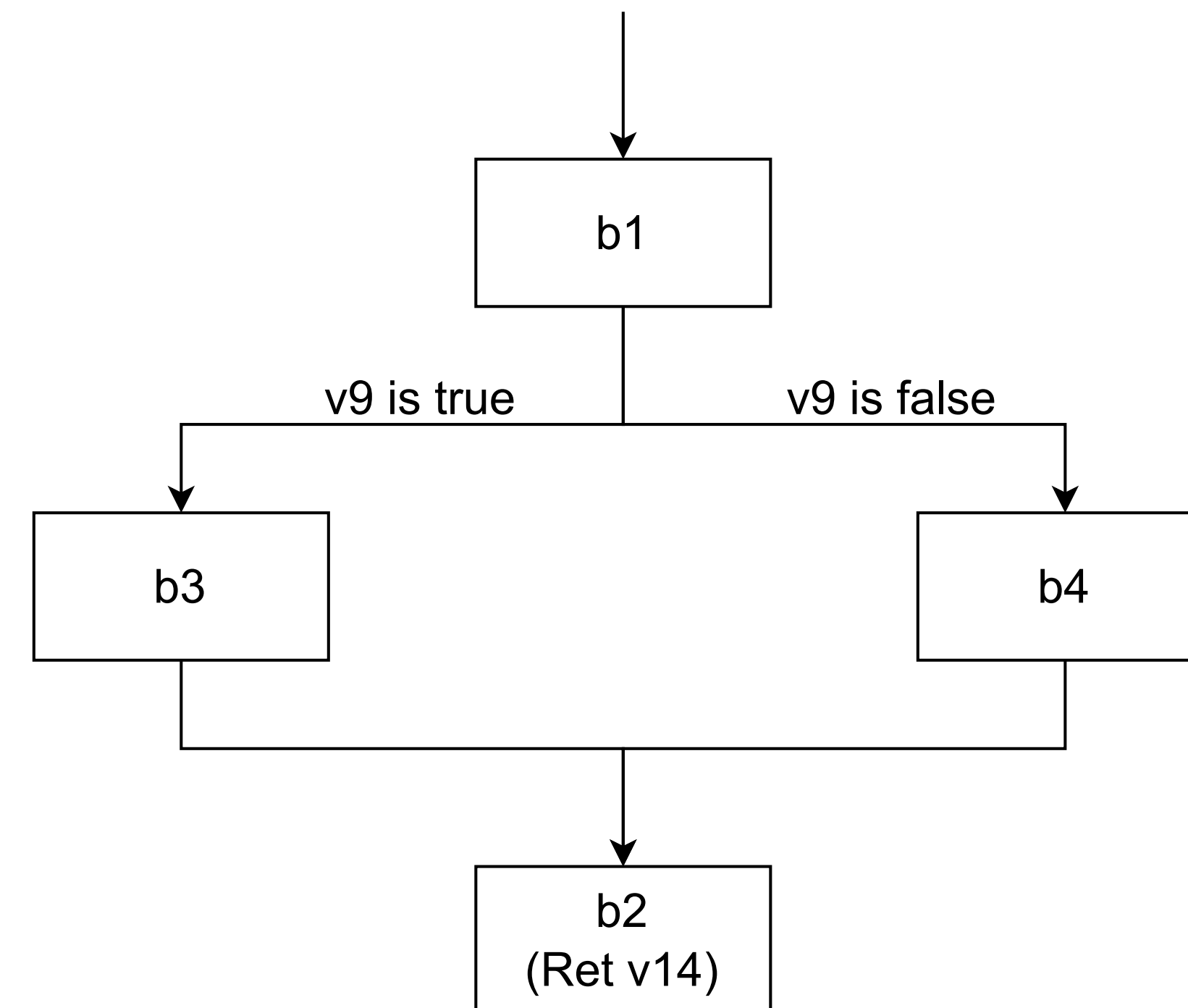
```
1 b1:
2     v1 (?) = InitMem <mem>
3     v2 (?) = SP <uintptr>
4     v3 (?) = SB <uintptr>
5     v4 (?) = LocalAddr <*int> {~r0} v2 v1
6     v5 (?) = Const64 <int> [0] (c[int])
7     v6 (?) = Const64 <int> [10] (a[int])
8     v7 (24) = Mul64 <int> v6 v6 (b[int])
9     v8 (?) = Const64 <int> [100]
10    v9 (27) = Eq64 <bool> v7 v8
11    v11 (?) = Const64 <int> [42] (c[int])
12    If v9 → b3 b4 (27)
13
14    b2: ← b3 b4
15    v12 (33) = Phi <int> v10 v11 (c[int])
16    v13 (33) = Copy <mem> v1
17    v14 (33) = MakeResult <int,mem> v12 v13
18    Ret v14 (+33)
19
20    b3: ← b1
21    v10 (28) = Copy <int> v7 (b[int], c[int])
22
23    Plain → b2 (28)
24
25    b4: ← b1
26
27    Plain → b2 (33)
```



# Backend

## SSA & Control-Flow Graph

```
1 b1:
2   v1 (?) = InitMem <mem>
3   v2 (?) = SP <uintptr>
4   v3 (?) = SB <uintptr>
5   v4 (?) = LocalAddr <*int> {~r0} v2 v1
6   v5 (?) = Const64 <int> [0] (c[int])
7   v6 (?) = Const64 <int> [10] (a[int])
8   v7 (24) = Mul64 <int> v6 v6 (b[int])
9   v8 (?) = Const64 <int> [100]
10  v9 (27) = Eq64 <bool> v7 v8
11  v11 (?) = Const64 <int> [42] (c[int])
12 If v9 → b3 b4 (27)
13
14 b2: ← b3 b4
15   v12 (33) = Phi <int> v10 v11 (c[int])
16   v13 (33) = Copy <mem> v1
17   v14 (33) = MakeResult <int,mem> v12 v13
18 Ret v14 (+33)
19
20 b3: ← b1
21   v10 (28) = Copy <int> v7 (b[int], c[int])
22
23 Plain → b2 (28)
24
25 b4: ← b1
26
27 Plain → b2 (33)
```





# Backend

## Machine-Independent Optimization

- Optimizations performed in multiple (ordered) passes
  - Dead code elimination
  - Removal of unneeded nil checks
  - Constant folding
  - Common sub-expression elimination
- Application of function intrinsics



# Backend

## Definition of Passes

```
1 // src/cmd/compile/internal/ssa/compile.go
2
3 // list of passes for the compiler
4 var passes = [...]pass{
5     {name: "number lines", fn: numberLines, required: true},
6     {name: "early phielim and copyelim", fn: copyelim},
7     {name: "early deadcode", fn: deadcode},
8     {name: "short circuit", fn: shortcircuit},
9     {name: "decompose user", fn: decomposeUser, required: true},
```

```
1 // src/cmd/compile/internal/ssa/compile.go
2
3 // Double-check phase ordering constraints.
4 // This code is intended to document the ordering requirements
5 // between different phases. It does not override the passes
6 // list above.
7
8 type constraint struct {
9     a, b string // a must come before b
10 }
11
12 var passOrder = [...]constraint{
13     {"dse", "insert resched checks"},
14     {"insert resched checks", "lower"},
15     {"insert resched checks", "tighten"},
```



# Backend

## Definition of Passes via Rewrite Rules

```
1 // src/cmd/compile/internal/ssa/_gen/generic.rules
2
3 // constant folding
4 (Add8 (Const8 [c]) (Const8 [d])) ⇒ (Const8 [c+d])
5
6 // Convert x * 1 to x.
7 (Mul(8|16|32|64) (Const(8|16|32|64) [1]) x) ⇒ x
8
9 // constant floating point comparisons
10 (Eq32F (Const32F [c]) (Const32F [d])) ⇒ (ConstBool [c = d])
```



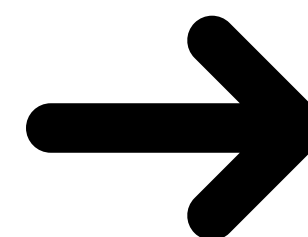


# Backend

## Optimization of SSA

```
1 func example() int {
2     a := 10
3     b := a * 10
4
5     c := 0
6     if b == 100 {
7         c = b
8     } else {
9         c = 42
10    }
11
12    return c
13 }
```

```
1 b1:
2     v1 (?) = InitMem <mem>
3     v2 (?) = SP <uintptr>
4     v3 (?) = SB <uintptr>
5     v4 (?) = LocalAddr <*int> {~r0} v2 v1
6     v5 (?) = Const64 <int> [0] (c[int])
7     v6 (?) = Const64 <int> [10] (a[int])
8     v7 (24) = Mul64 <int> v6 v6 (b[int])
9     v8 (?) = Const64 <int> [100]
10    v9 (27) = Eq64 <bool> v7 v8
11    v11 (?) = Const64 <int> [42] (c[int])
12    If v9 → b3 b4 (27)
13
14    b2: ← b3 b4
15        v12 (33) = Phi <int> v10 v11 (c[int])
16        v13 (33) = Copy <mem> v1
17        v14 (33) = MakeResult <int,mem> v12 v13
18    Ret v14 (+33)
19
20    b3: ← b1
21        v10 (28) = Copy <int> v7 (b[int], c[int])
22
23    Plain → b2 (28)
24
25    b4: ← b1
26
27    Plain → b2 (33)
```



```
1 b2:
2     v1 (?) = InitMem <mem>
3     v7 (+24) = Const64 <int> [100] (c[int], b[int])
4     v14 (+33) = MakeResult <int,mem> v7 v1
5    Ret v14 (+33)
```



# Backend

## Machine-Specific Optimizations & Machine Code

After machine-independent opts.:

```
1 b2:
2     v1 (?) = InitMem <mem>
3     v7 (+24) = Const64 <int> [100] (c[int], b[int])
4     v14 (+33) = MakeResult <int,mem> v7 v1
5 Ret v14 (+33)
```

After lowering:

```
1 b2:
2     v1 (?) = InitMem <mem>
3     v7 (+24) = MOVDconst <int> [100] (b[int], c[int])
4     v14 (+33) = MakeResult <int,mem> v7 v1
5 Ret v14 (+33)
```

Final machine code:

```
1     00000 (22) TEXT main.simple(SB), ABIInternal
2     00001 (22) FUNCDATA $0, glocals·FzY36IO2mY0y4dZ1+Izd/w=(SB)
3     00002 (22) FUNCDATA $1, glocals·FzY36IO2mY0y4dZ1+Izd/w=(SB)
4 v9  00003 (+33) MOVD $100, R0
5 b2  00004 (33) RET
6     00005 (?) END
```





```
go build -gcflags="that's all!"
```

# Helpful Resources

- <https://go.dev/src/cmd/compile/README>
- <https://go.dev/src/cmd/compile/internal/ssa/README>
- <https://eli.thegreenplace.net/2019/go-compiler-internals-adding-a-new-statement-to-go-part-1/>
- <https://eli.thegreenplace.net/2019/go-compiler-internals-adding-a-new-statement-to-go-part-2/>
- [https://www.quasilyte.dev/blog/post/go\\_ssa\\_rules/](https://www.quasilyte.dev/blog/post/go_ssa_rules/)
- <https://dave.cheney.net/2019/08/20/go-compiler-intrinsics>