How to write a self hosted Go compiler from scratch

Daisuke Kashiwagi
Gophercon 2020
November 12

About me

Daisuke Kashiwagi   https://github.com/DQNEO

- Software engineer at Mercari
- Living in Japan
- Longtime PHP user mostly on web
- Wrote some compilers for fun

Today’s Goal

Convince you that

- You can write your own Go compiler
- It’s really fun !!

Agenda

- Demo
- Writing a C compiler in Go
- Writing a Go compiler in Go
- Contribution to the official Go compiler
- Writing another Go compiler in Go

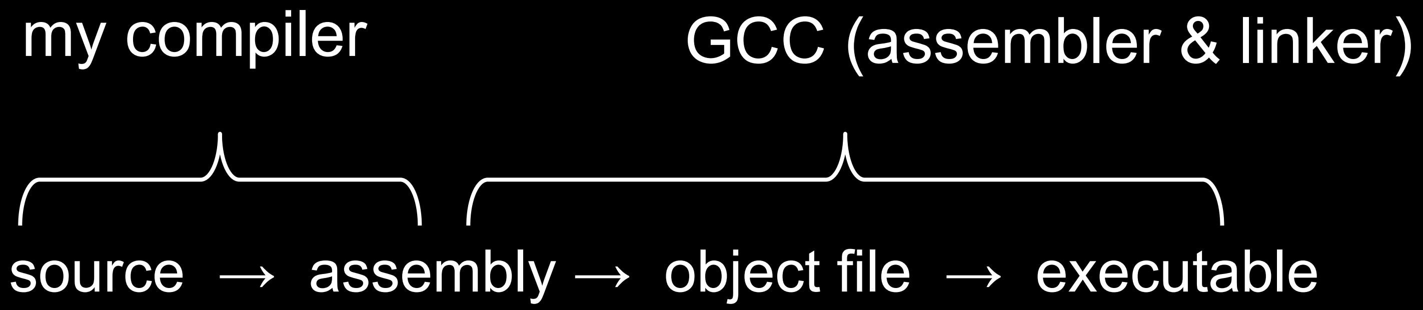
My compilers

1. 8cc.go
   C compiler in Go

2. minigo
   Go compiler in Go ← can compile itself

3. babygo
   Go compiler in Go ← can compile itself

Architecture of my compilers

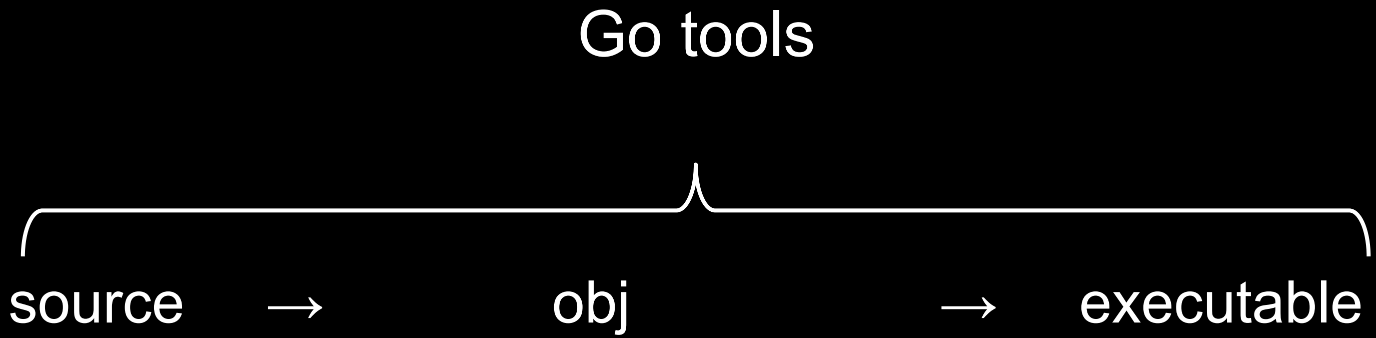


my compiler

GCC (assembler & linker)

source → assembly → object file → executable

Architecture of the official Go compiler



Go tools

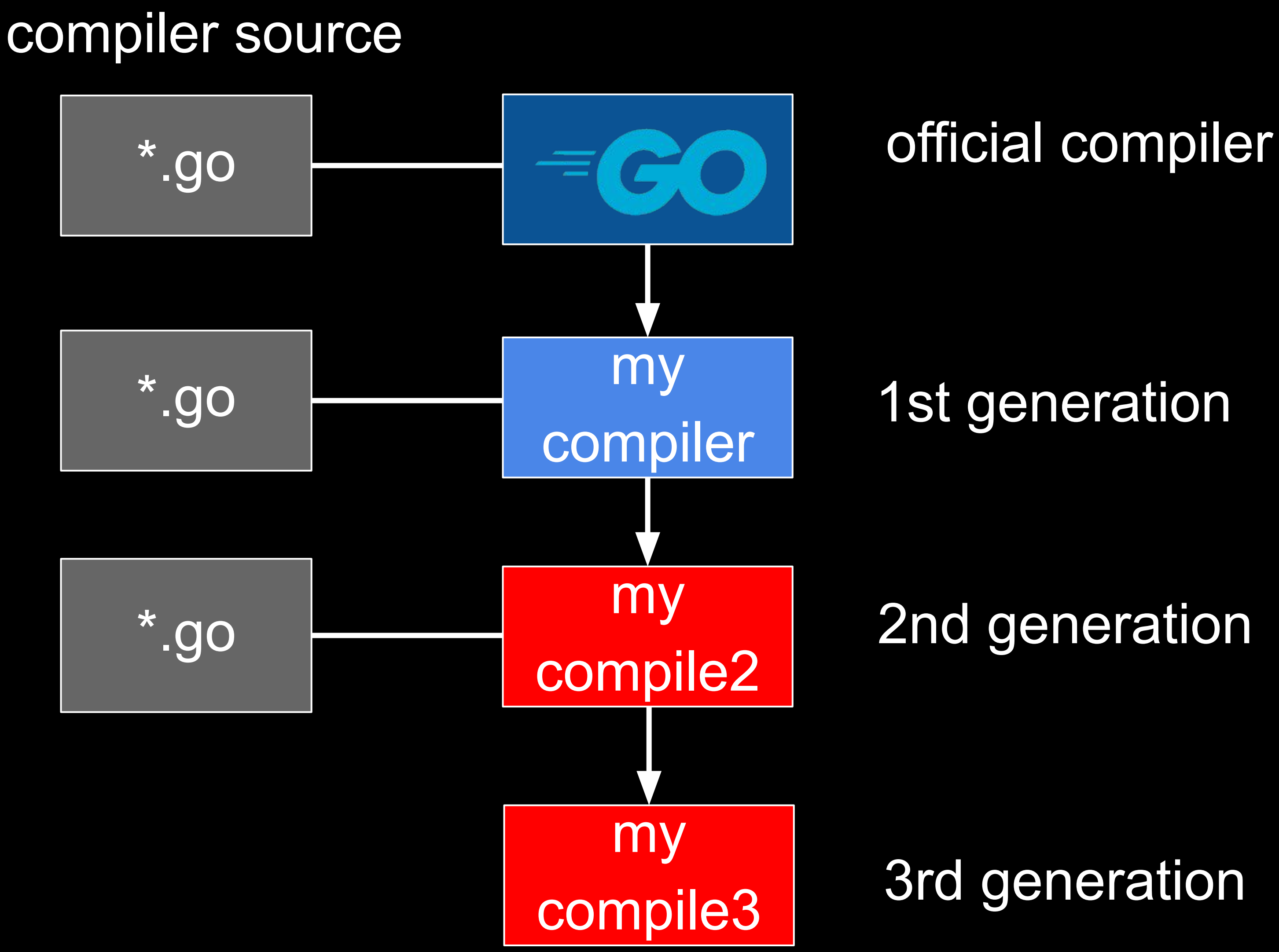
source → obj → executable

minigo & babygo

- Targeting x86-64 Linux only
- Lexer and parser are handwritten
- Standard libs are made from scratch
- Stack machine
- Far from production quality (for now)
    - No garbage collection
    - No concurrency
    - Minimal error check

Demo
hello world

Self hosting Go compiler



compiler source

*.go

official compiler

*.go

my compiler

1st generation

*.go

my compile2

2nd generation

my compile3

3rd generation

Demo: self hosting

Me before the journey

- Zero knowledge about compilers
    - Did not major in CS
- Not very good at Go
    - Mostly a PHP programmer
    - Gave up on “Tour of Go” twice
- Wanted to be better at Go
- Interested in low level programming

C compiler

Encounter with 8cc

https://github.com/rui314/8cc

made by Mr. Rui Ueyama

Encounter with 8cc

- self hosting C compiler
- written from scratch
- 9,000 lines of code

Diary:

https://www.sigbus.info/how-i-wrote-a-self-hosting-c-compiler-in-40-days

8cc: First commit

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
 int val;
 if (scanf("%d", &val) == EOF) {
   perror("scanf");
   exit(1);
 }
 printf("\t.text\n\t"
        ".global mymain\n"
        "mymain:\n\t"
        "mov $%d, %%eax\n\t"
        "ret\n", val);
 return 0;
}
```

```
#include <stdio.h>

extern int mymain(void);

int main(int argc, char **argv) {
 int val = mymain();
 printf("%d\n", val);
 return 0;
}
```

https://github.com/rui314/8cc/commit/3764b2071b9601067b81976d80175a0851d0f209

8cc.go: Porting 8cc to Go

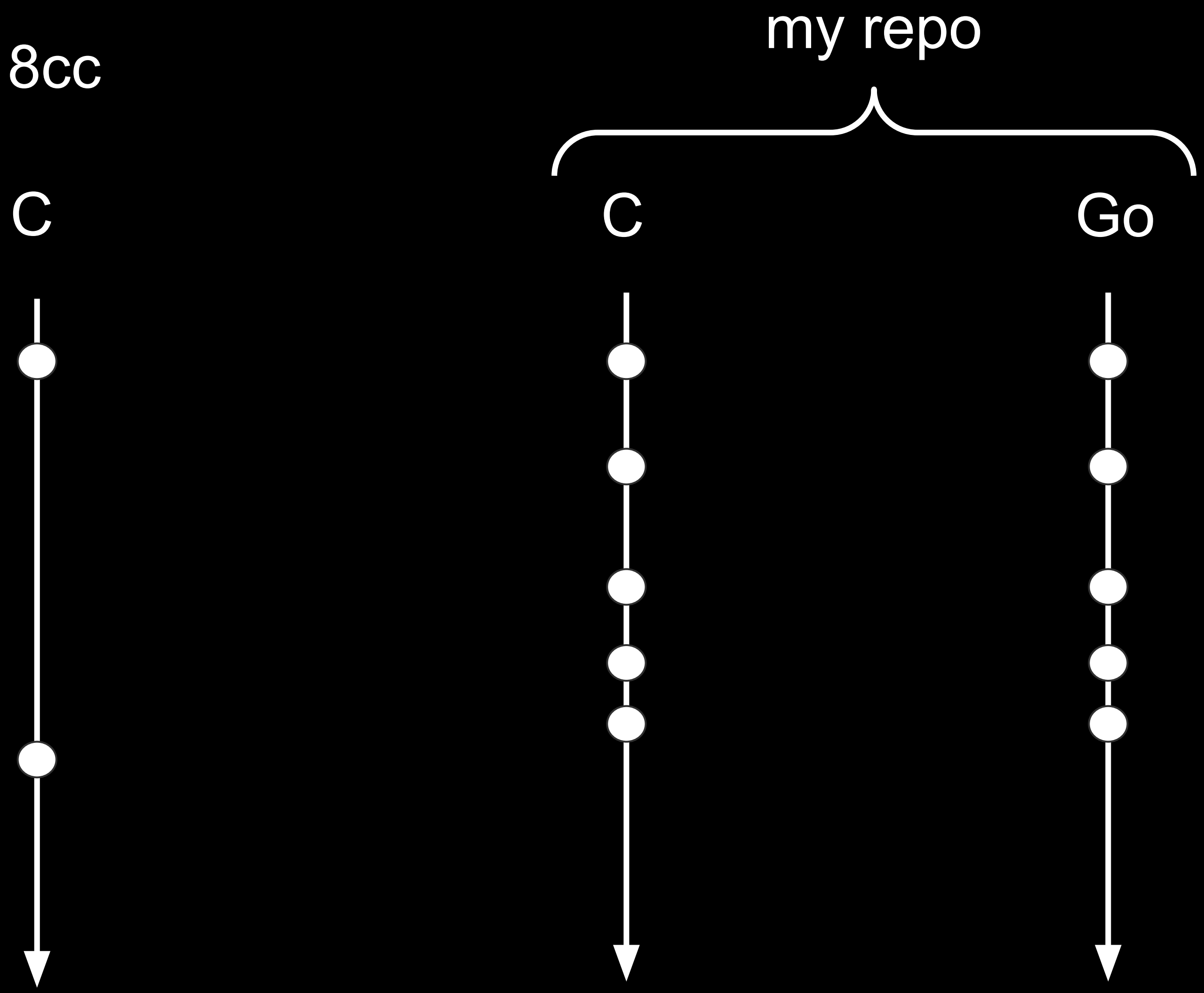
https://github.com/DQNEO/8cc.go

	My work	Inspired by
▶️ 1	8cc.go	8cc
2	minigo	8cc
3	babygo	chibicc, go/parser

8cc.go: Porting 8cc to Go

- C compiler written in Go
- Ported commits from the beginning from C to Go

Porting commits from C to C to Go



Porting commits from C to C to Go

- Continued for 5 months
- Ported over 100 commits
- Covered most major syntax

Porting 8cc : I learned

- How to write C and Go
- How the C language works internally
- How to read/write assembly code
- What stack machines are like
- How similar is C to Go

Learn C and Go at the same time

C

```c
static char *REGS[] = {"rdi", "rsi", "rdx", "rcx", "r8", "r9"};
```

Go

```go
var REGS = [...]string{"rdi", "rsi", "rdx","rcx", "r8", "r9"}
```

Learn C and Go at the same time

C

```c
static Ast *ast_uop(int type, Ctype *ctype, Ast *operand)
{
   Ast *r = malloc(sizeof(Ast));
   r->type = type;
   r->ctype = ctype;
   r->operand = operand;
   return r;
}
```

Go

```go
func ast_uop(typ int, ctype *Ctype, operand *Ast) *Ast {
  r := &Ast{}
  r.typ = typ
  r.ctype = ctype
  r.operand = operand
  return r
}
```

Can I write a Go compiler by simply using this knowledge ?

a Go compiler in go

Tried writing a Go compiler

https://github.com/DQNEO/minigo

	My work	Inspired by
1	8cc.go	8cc
▶️ 2	minigo	8cc
3	babygo	chibicc, go/parser

minigo: My first go compiler

First commit

```
   .globl main
main:
  movl   $0, %eax
  ret
```

a program which exits with status 0

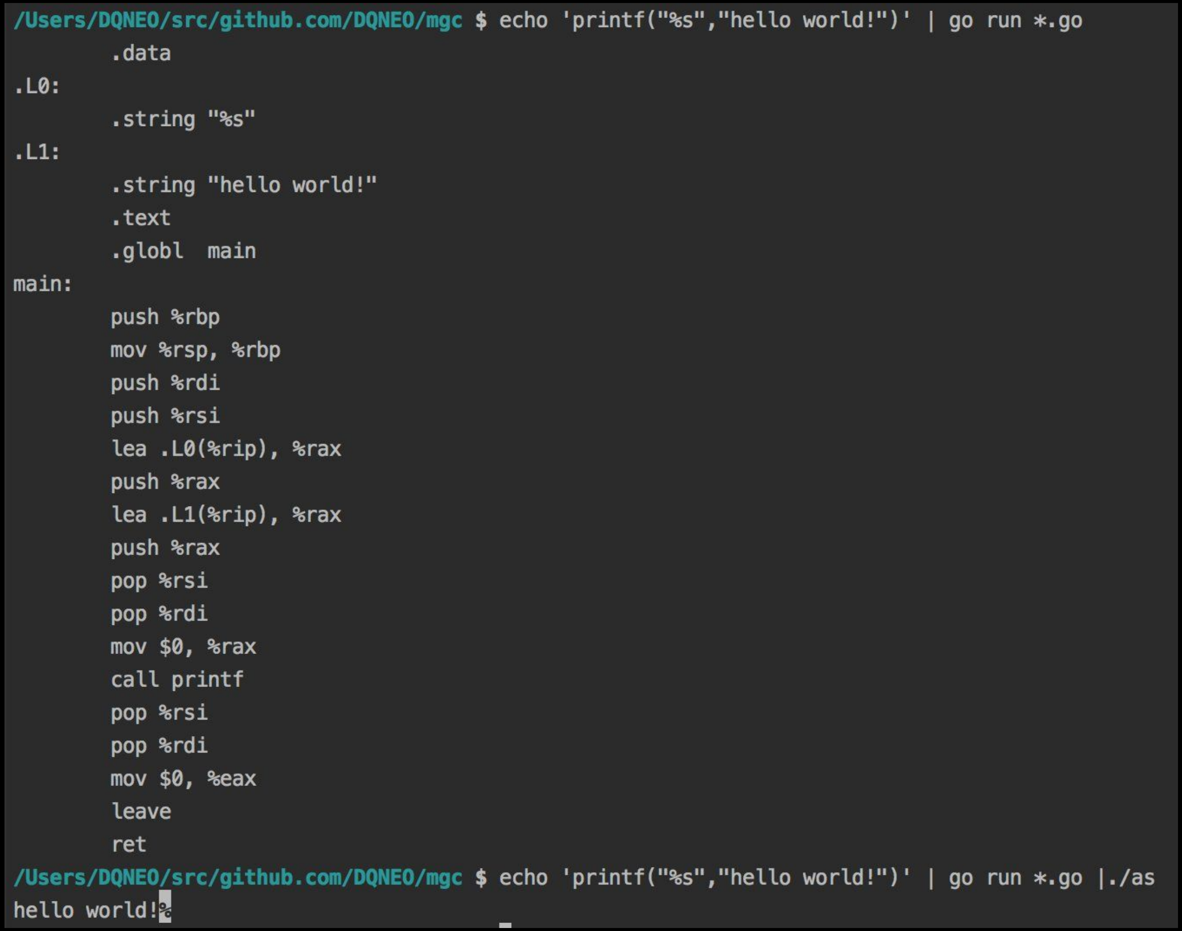
minigo: My first go compiler

- Day 1: Arithmetic addition worked

```
$ echo '2 + 5' | go run main.go

# ==== Start Dump Tokens ===
2  +  5
# ==== End Dump Tokens ===
# right=5
# ==== Dump Ast ===
# ast.binop=binop
# left=2
# right=5
        .globl  main
main:
        movl    $2, %ebx
        movl    $5, %eax
        addl    %ebx, %eax
        ret
```

minigo: My first go compiler

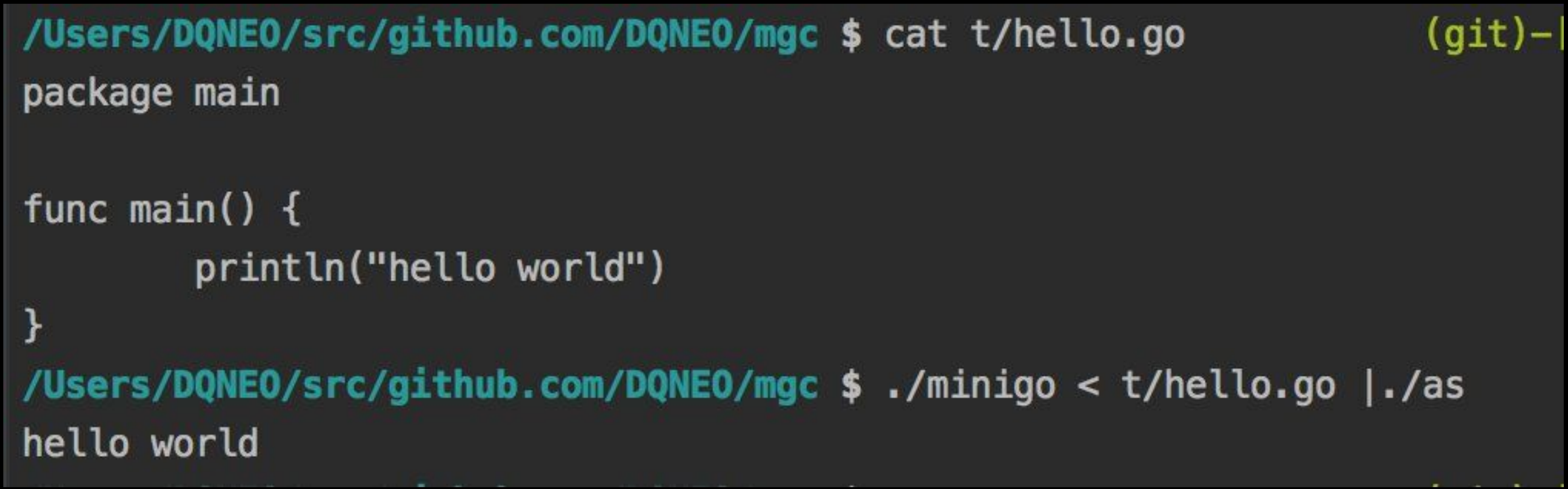
- Day 2: Function call worked



```
/Users/DQNEO/src/github.com/DQNEO/mgc $ echo 'printf("%s","hello world!")' | go run *.go
        .data
.L0:
        .string "%s"
.L1:
        .string "hello world!"
        .text
        .globl  main
main:
        push %rbp
        mov %rsp, %rbp
        push %rdi
        push %rsi
        lea .L0(%rip), %rax
        push %rax
        lea .L1(%rip), %rax
        push %rax
        pop %rsi
        pop %rdi
        mov $0, %rax
        call printf
        pop %rsi
        pop %rdi
        mov $0, %eax
        leave
        ret
/Users/DQNEO/src/github.com/DQNEO/mgc $ echo 'printf("%s","hello world!")' | go run *.go |./as
hello world!%
```

minigo: My first go compiler

- Day 5: an entire “Hello world” file worked



```
/Users/DQNEO/src/github.com/DQNEO/mgc $ cat t/hello.go          (git)-
package main

func main() {
        println("hello world")
}
/Users/DQNEO/src/github.com/DQNEO/mgc $ ./minigo < t/hello.go |./as
hello world
```

minigo: rapid progress in the first half

- Month 1: FizzBuzz worked
- Month 2: It was able to parse itself

Go language is easy to scan and parse

- Designed as such
- Parser can determine mode only by looking at one token in top level
  - ”type” ,“var”, “func”
- types can be read from left to right
  - e.g. []*int
- few historical twists and turns in its syntax

Writing a Go compiler in Go: the easy parts

- Lexer and parser can be easily implemented
- You can use powerful tools like slice, map, for-range

Writing a Go compiler in Go: the hard parts

- You must implement powerful tools like slice, map, for-range
- Some data types are larger than a single register
    - string (16 bytes), slice (24 bytes)
        - handling them on a stack machine is not trivial
- Runtime features
    - Goroutine
    - Memory management

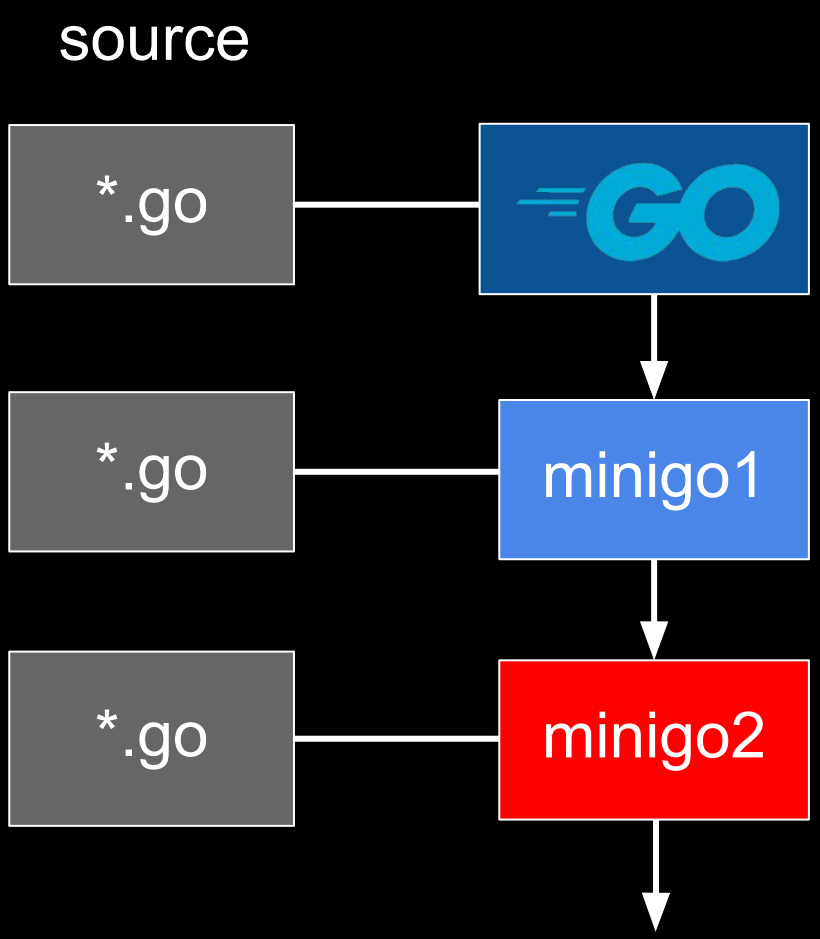
Learning Go spec by writing its compiler

- Assignment is not an expression (x = 1)
- Increment is not an expression (x++)
- How iota works
- How identifiers are “resolved”
- Role of the universe block
- etc.

minigo: Struggles in the last half

- Month 3: implement append, map, interface
- Month 4: SEGV in 2nd generation compiler
- Month 5: SEGV in 2nd generation compiler

bugs in the 2nd gen compilation

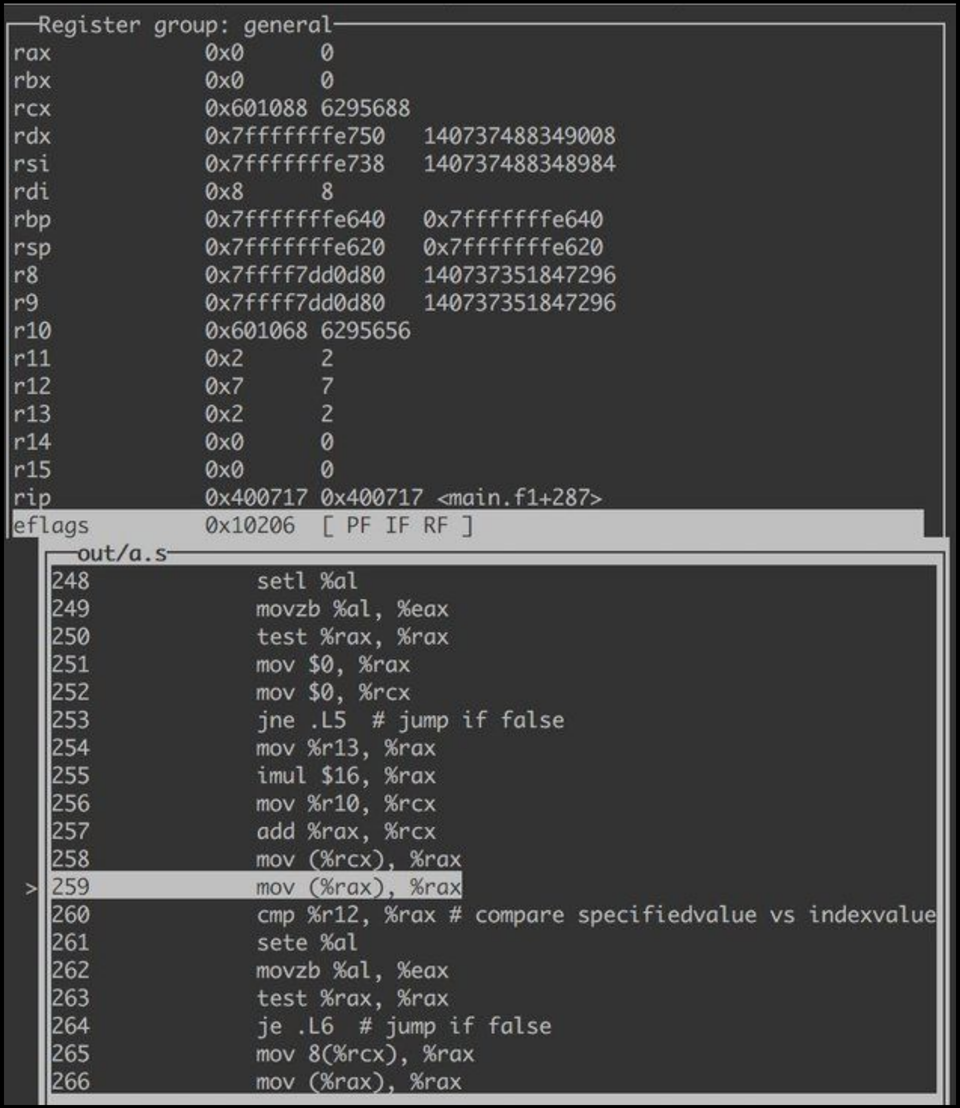


Official go

1 generation:
an ordinary go program

2 generation:
my assembly
with a lot of mistakes

minigo: Fought with SEGV by gdb



```
Register group: general
rax            0x0      0
rbx            0x0      0
rcx            0x601088 6295688
rdx            0x7fffffffe750   140737488349008
rsi            0x7fffffffe738   140737488348984
rdi            0x8      8
rbp            0x7fffffffe640   0x7fffffffe640
rsp            0x7fffffffe620   0x7fffffffe620
r8             0x7ffff7dd0d80   140737351847296
r9             0x7ffff7dd0d80   140737351847296
r10            0x601068 6295656
r11            0x2      2
r12            0x7      7
r13            0x2      2
r14            0x0      0
r15            0x0      0
rip            0x400717 0x400717 <main.f1+287>
eflags         0x10206  [ PF IF RF ]
```

```
out/a.s
                setl %al
                movzb %al, %eax
                test %rax, %rax
                mov $0, %rax
                mov $0, %rcx
                jne .L5  # jump if false
                mov %r13, %rax
                imul $16, %rax
                mov %r10, %rcx
                add %rax, %rcx
                mov (%rcx), %rax
>               mov (%rax), %rax
                cmp %r12, %rax # compare specifiedvalue vs indexvalue
                sete %al
                movzb %al, %eax
                test %rax, %rax
                je .L6  # jump if false
                mov 8(%rcx), %rax
                mov (%rax), %rax
```

minigo: Won

- Month 6: Successfully compiled itself 🎉

minigo: self hosted

- 10,000 lines of code
- Without taking any look at the official compiler
- Supports
  - slice, array, struct
  - map, interface, method call
  - type assertion, type switch
  - etc.

minigo: Added more features

- Environment variables
- GOPATH
- importing of 3rd party libraries
- Eliminated libc dependency

Implementation of “append”

Borrowed from the “Go programming language”

```
func append1(x []byte, elm byte) []byte {
  var z []byte
  xlen := len(x)
  zlen := xlen + 1

  if cap(x) >= zlen {
    z = x[:zlen]
  } else {
    var newcap int
    if xlen == 0 {
      newcap = 1
    } else {
      newcap = xlen * 2
    }
    z = makeSlice(zlen, newcap, 1)
    for i:=0;i<xlen;i++ {
      z[i] = x[i]
    }
  }

  z[xlen] = elm
  return z
}
```

Implementation of malloc (1st ver)

- Using a static area (pseudo heap)
- each malloc() consumes a piece of segment

```
var heap [640485760]byte
var heapTail *int
func malloc(size int) *int {
  if heapTail+ size > len(heap) + heap  {
     panic("malloc failed")
  }
  r := heapTail
  heapTail += size
  return r
}
```

Implementation of “map”

- array of pairs of key and value
- “map get” is just a linear search
- Mostly written in assembly code

Implementation of "interface"

- Serialize string representation of a type on assignment
    - e.g.

      ```
      var x *T
      var i interface{} = x
      *T  →  “*G_NAMED(main.T)”
      ```
- type switch / type assertion compares those string representations
- Lookup of method call is like “map get”

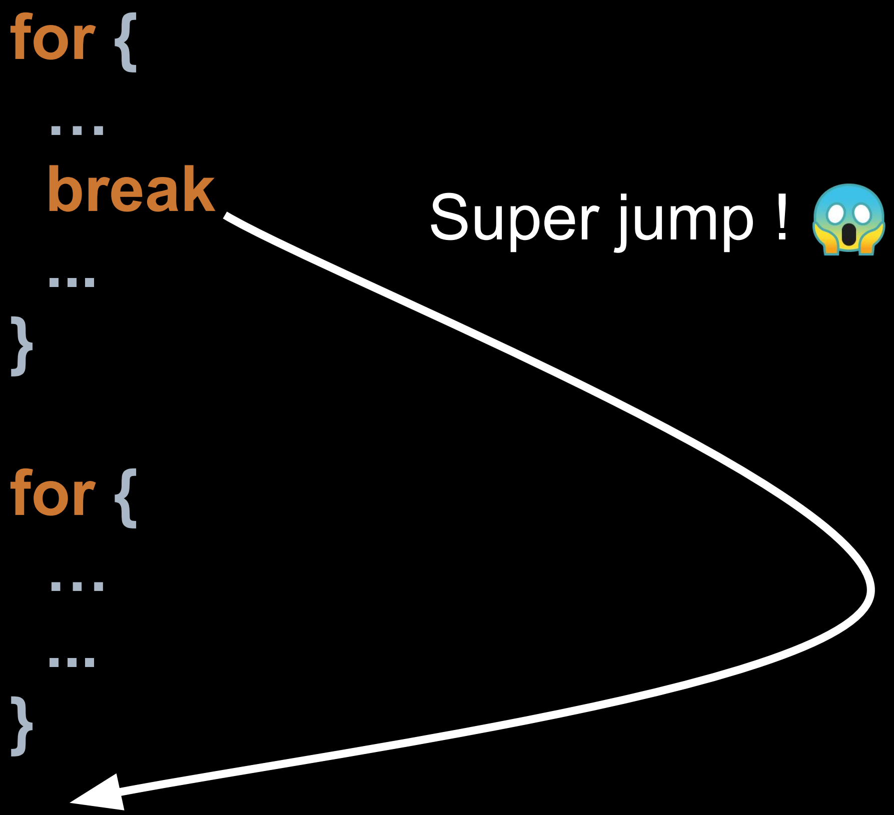
minigo lacks...

- Garbage collection
- Go routine
  - extremely difficult
- Floating point numbers
- Multiplatform (OS,CPU)
- etc

Funny bug: break

```
for {
  ...
  break
  ...
}
```

Funny bug: break



```
for {
  ...
  break
  ...
}

for {
  ...
  ...
}
```

Super jump ! 😱

minigo : Room for improvement - Not Go-ish

- Internal ABI (Application Binary Interface) is very close to that of C compilers
  - e.g. registers assignment in function call
- Started with null-terminated string and libc dependency
  - Changed the fundamental design in the end
    - null-terminated string → slice-like struct
    - Eliminated libc dependency
  - I wish I had done it from the beginning

minigo: Room for improvement

- Code generation is a chaos
    - Assignment is super complicated due to my poor understanding of stack machine

Contributing to
the official Go compiler

Tried reading the official Go compiler

- After minigo, I started to look at the official compiler
- Found myself being able to understand some parts
  - I had an overall map in my mind about what compilers look like
- Could read code by thinking “What’s different between mine and theirs?”

Tried reading the official Go compiler

- How size of each embedded type is designed ?

src/cmd/compile/internal/gc/align.go

src/cmd/compile/internal/gc/go.go

Official compiler: size of slice

```
case TSLICE:
  if t.Elem() == nil {
     break
  }
  w = int64(sizeof_Array)
```

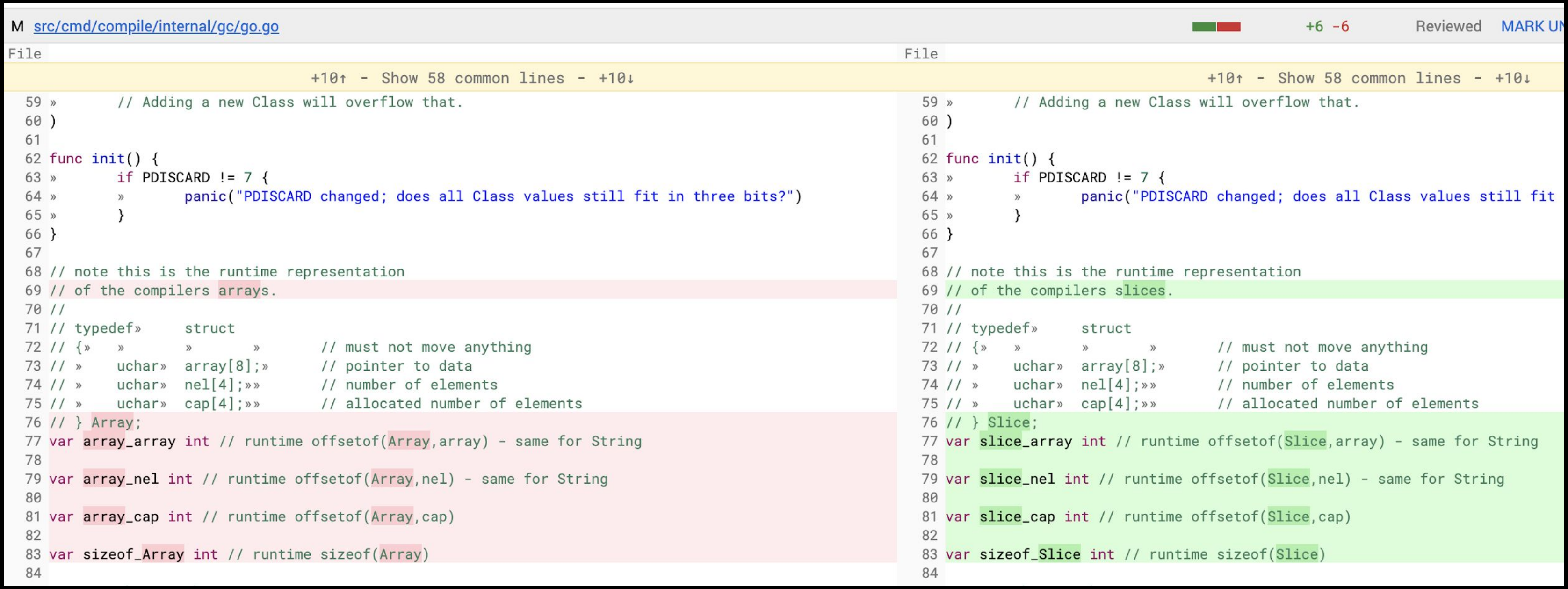
Why is the size of slice named “sizeof_Array” ?

Official compiler: variable names for slice

```
// note this is the runtime representation
// of the compilers arrays.
//
// typedef struct
// {
//     uchar  array[8];  // pointer to data
//     uchar  nel[4];       // number of elements
//     uchar  cap[4];       // allocated number of elements
// } Array;
var array_array int // runtime offsetof(Array,array) - same for String
var array_nel int // runtime offsetof(Array,nel) - same for String
var array_cap int // runtime offsetof(Array,cap)
var sizeof_Array int // runtime sizeof(Array)
```

Could we improve these ?

Tried submitting a patch



M src/cmd/compile/internal/gc/go.go   🟩🟥   +6 -6   Reviewed   MARK UN

File

+10↑ - Show 58 common lines - +10↓

```
»       // Adding a new Class will overflow that.
)

func init() {
»       if PDISCARD != 7 {
»       »       panic("PDISCARD changed; does all Class values still fit in three bits?")
»       }
}

// note this is the runtime representation
// of the compilers arrays.
//
// typedef»     struct
// {»   »       »       »       // must not move anything
// »    uchar»  array[8];»      // pointer to data
// »    uchar»  nel[4];»»       // number of elements
// »    uchar»  cap[4];»»       // allocated number of elements
// } Array;
var array_array int // runtime offsetof(Array,array) - same for String

var array_nel int // runtime offsetof(Array,nel) - same for String

var array_cap int // runtime offsetof(Array,cap)

var sizeof_Array int // runtime sizeof(Array)

```

File

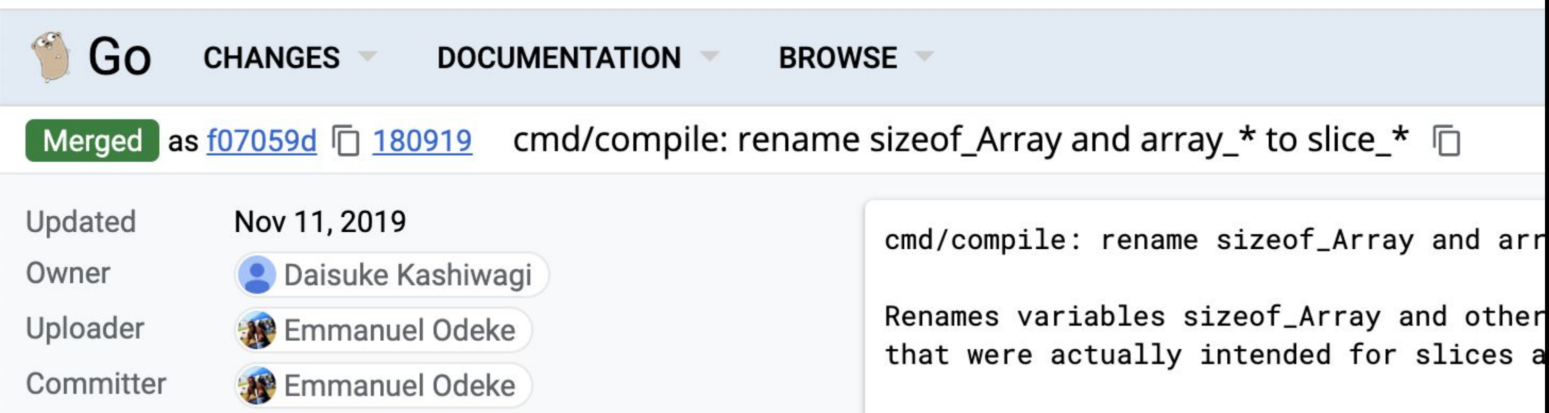
+10↑ - Show 58 common lines - +10↓

```
»       // Adding a new Class will overflow that.
)

func init() {
»       if PDISCARD != 7 {
»       »       panic("PDISCARD changed; does all Class values still fit
»       }
}

// note this is the runtime representation
// of the compilers slices.
//
// typedef»     struct
// {»   »       »       »       // must not move anything
// »    uchar»  array[8];»      // pointer to data
// »    uchar»  nel[4];»»       // number of elements
// »    uchar»  cap[4];»»       // allocated number of elements
// } Slice;
var slice_array int // runtime offsetof(Slice,array) - same for String

var slice_nel int // runtime offsetof(Slice,nel) - same for String

var slice_cap int // runtime offsetof(Slice,cap)

var sizeof_Slice int // runtime sizeof(Slice)

```

- “array” → “slice”
- Tried Gerrit

  https://go-review.googlesource.com/c/go/+/180919

Merged 🎉



Merged as f07059d 180919 cmd/compile: rename sizeof_Array and array_* to slice_*

Updated Nov 11, 2019
Owner Daisuke Kashiwagi
Uploader Emmanuel Odeke
Committer Emmanuel Odeke

```
cmd/compile: rename sizeof_Array and arr

Renames variables sizeof_Array and other
that were actually intended for slices a
```

- It’s in Go 1.4

  https://github.com/golang/go/commit/f07059d949057f414dd0f8303f93ca727d716c62

Took a rest

- Took a rest from compilers for half a year

Lingering questions

- Could I do self-host much more easily if I try another one… ?
- What would it be like to take a different approach … ?
  - If I started without libc from the beginning ?
  - if I used go/parser ?
  - What is the ideal stack machine … ?

chibicc was born

https://github.com/rui314/chibicc

made by Mr. Rui Ueyama

with much simpler stack machine

another Go compiler in go

Started writing another Go compiler

https://github.com/DQNEO/babygo

	My work	Inspired by
1	8cc.go	8cc
2	minigo	8cc
▶️ 3	babygo	chibicc, go/parser

babygo: Theme

- How do I achieve self-hosting with less code ?

babygo: First commit

```
// runtime
.text
.global _start
_start:
  movq $42, %rdi
  movq $60, %rax
  syscall
```

a program which exits with status 42

First commit: minigo vs babygo

minigo

```
    .global main
main:
  movl $42, %eax
  ret
```

babygo

```
    .global _start
_start:
  movq $42, %rdi
  movq $60, %rax
  syscall
```

(apple to apple comparison)

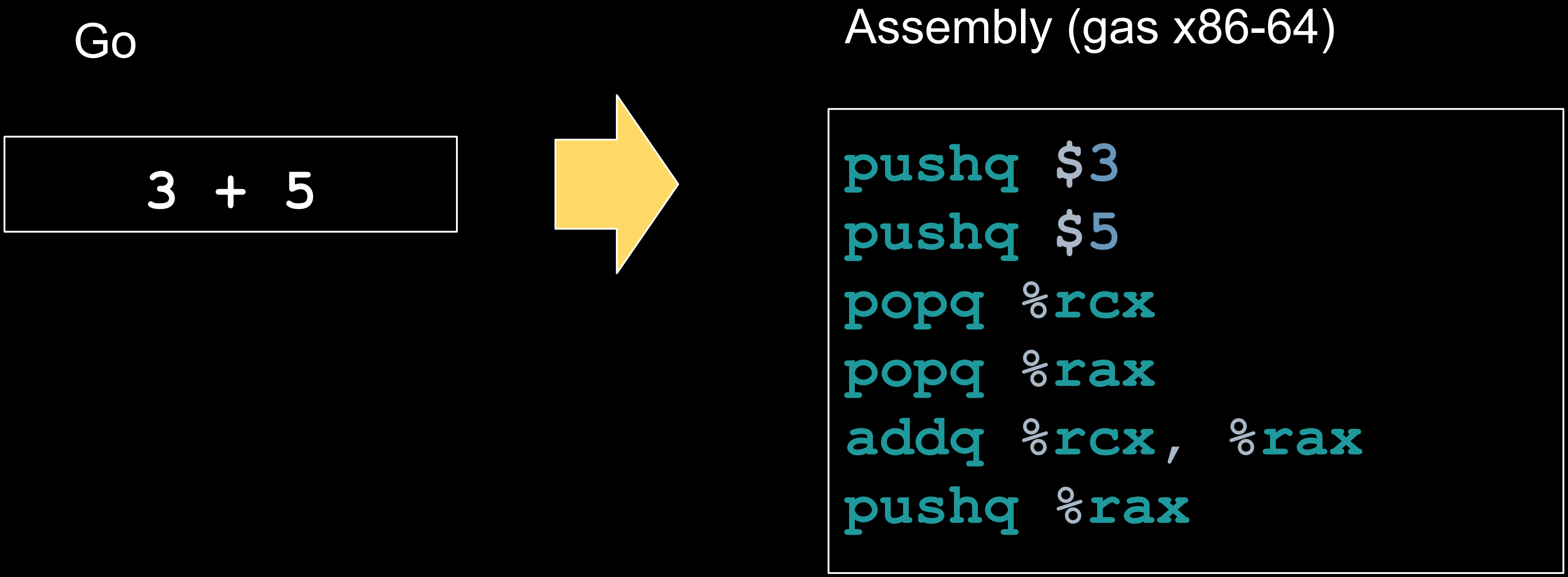
babygo: different approaches

- less features
- better stack machine
- more Go-like
- the order of implementation

babygo: less features

- as small as possible
- omitted
  - map, interface, method
  - packaging system
  - etc.

Stack machine (chibicc style)



Go

```
3 + 5
```

Assembly (gas x86-64)

```
pushq $3
pushq $5
popq %rcx
popq %rax
addq %rcx, %rax
pushq %rax
```

babygo: stack machine (chibicc-like)

Go

```
x = y
```

Assembly (gas x86-64)

```
leaq -16(%rbp), %rax
pushq %rax
```

address of x

```
leaq -8(%rbp), %rax
pushq %rax
```

address of y

```
popq %rax
movq 0(%rax), %rax
pushq %rax
```

value of y

```
popq %rdi
popq %rax
movq %rdi, (%rax)
```

assign value to x

babygo: stack machine (chibicc-like)

Source

```
a.b[c].d

 = e[f].g[h]
```

Assembly (gas x86-64)

```
<calc address>
pushq %rax
```

address of left expr

```
<calc address>
pushq %rax
```

address of right expr

```
popq %rax
movq 0(%rax), %rax
pushq %rax
```

value of right

```
popq %rdi
popq %rax
movq %rdi, (%rax)
```

assign value to left

babygo: being more Go-like

- Independent from libc
- string is a combination of a pointer and a length
- make ABI (Application Binary Interface)

  more similar to that of the official Go

babygo: Handwritten syscall

runtime.s (callee)

```
syscall.Syscall:
 movq   8(%rsp), %rax # syscall number
 movq  16(%rsp), %rdi # arg0
 movq  24(%rsp), %rsi # arg1
 movq  32(%rsp), %rdx # arg2
 syscall
 ret
```

runtime.go (caller)

```
syscall.Syscall(
 uintptr(SYS_BRK),
 addr,
 uintptr(0),
 uintptr(0)
)
```

ABI of official Go

source

```
func sum(a int, b int) int {
    return a + b
}
```

Go's Assembler

```
TEXT     "".sum(SB), ..., $0-24
MOVQ    $0, "".~r2+24(SP)
MOVQ    "".a+8(SP), AX
ADDQ    "".b+16(SP), AX
MOVQ    AX, "".~r2+24(SP)
RET
```

ABI of babygo

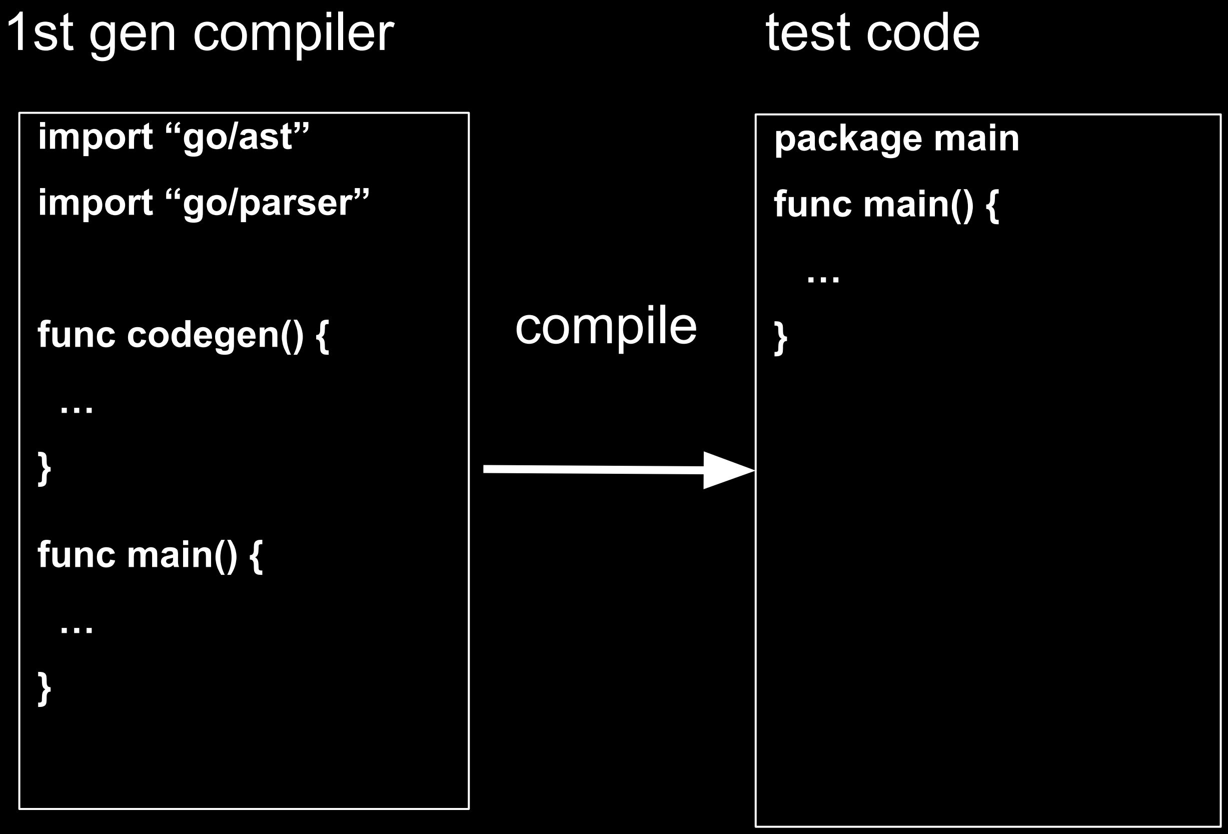
source

```
func sum(a int, b int) int {
    return a + b
}
```

GNU assembler

```
main.sum:
 pushq %rbp
 movq %rsp, %rbp
 leaq 16(%rbp), %rax # address of a
 pushq %rax
 popq %rax
 movq 0(%rax), %rax # load value
 pushq %rax
 leaq 24(%rbp), %rax # address of b
 pushq %rax
 popq %rax
 movq 0(%rax), %rax # load value
 pushq %rax
 popq %rcx # right
 popq %rax # left
 addq %rcx, %rax
 pushq %rax
 popq %rax # returned value
 leave
 ret
```

babygo: Order of implementation



1st gen compiler

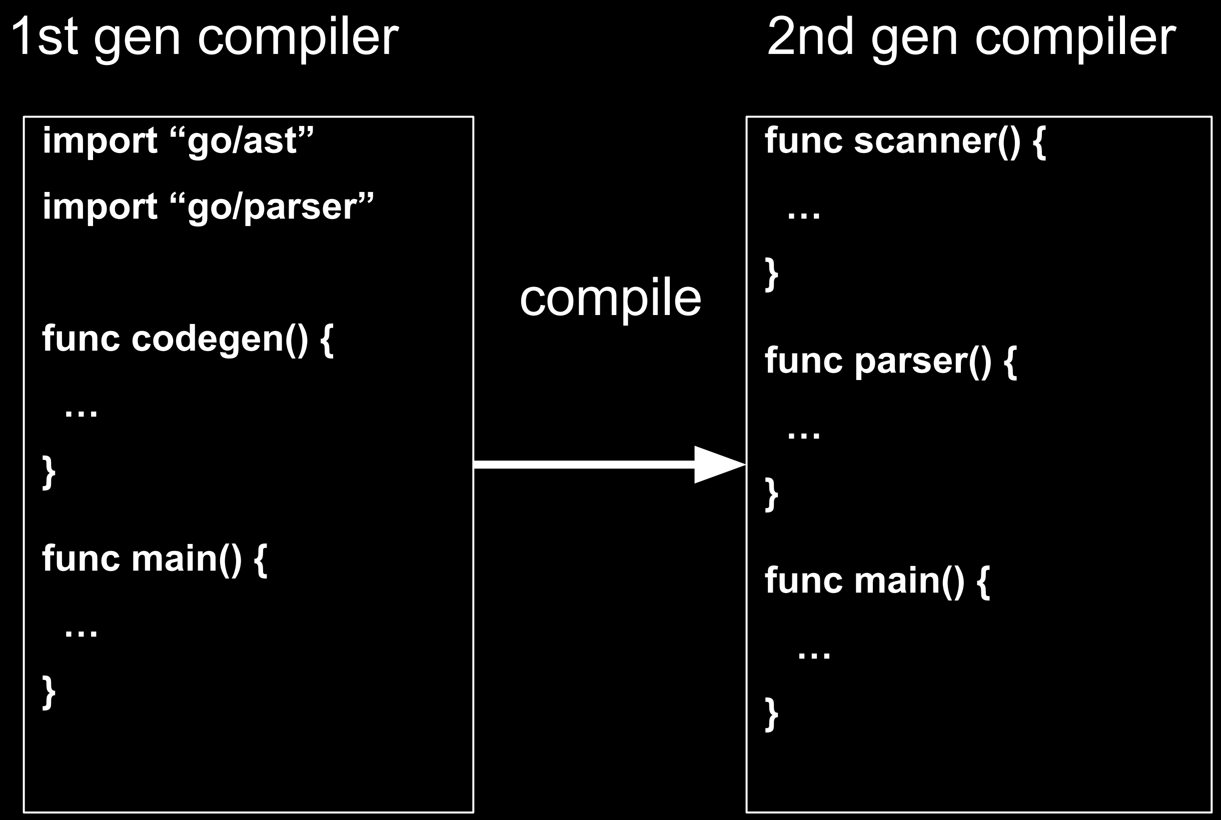
```
import “go/ast”

import “go/parser”


func codegen() {

  ...

}

func main() {

  ...

}
```

test code

```
package main

func main() {

   ...

}
```

- Write codegen first using go/parser, go/ast
- Evaluate codegen design first

babygo: Order of implementation



1st gen compiler

```
import "go/ast"
import "go/parser"


func codegen() {
  ...
}
func main() {
  ...
}
```

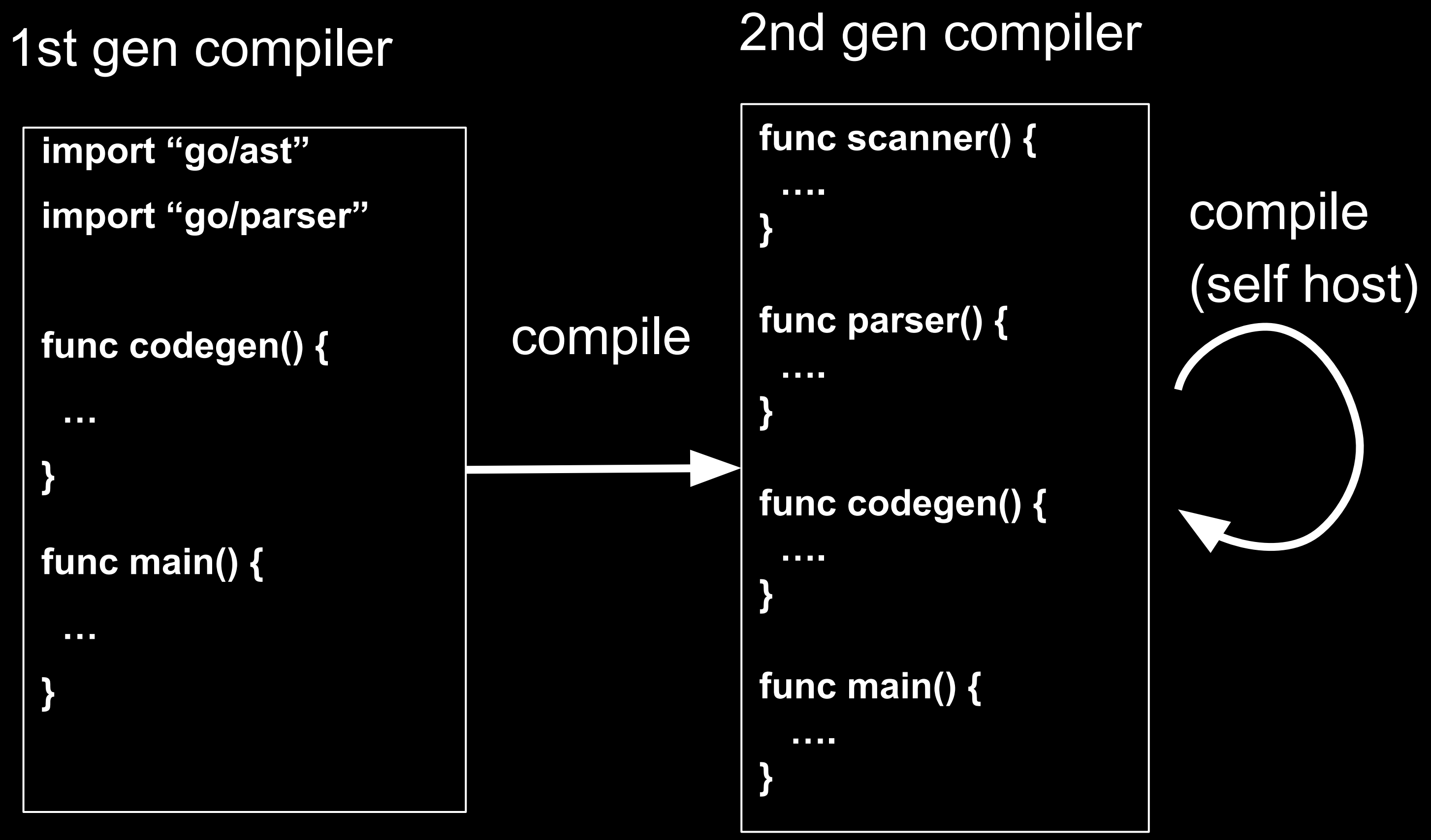
compile

2nd gen compiler

```
func scanner() {
  ...
}
func parser() {
  ...
}
func main() {
   ...
}
```

- Write 2nd gen compiler with the minimum grammar that 1st gen supports
- Re-invent go/* packages
- Easy to debug codegen

babygo: Order of implementation



1st gen compiler

```
import “go/ast”
import “go/parser”


func codegen() {
  …
}
func main() {
  …
}
```

compile

2nd gen compiler

```
func scanner() {
  ….
}

func parser() {
  ….
}

func codegen() {
  ….
}

func main() {
   ….
}
```

compile
(self host)

- 2nd gen can compile itself
- 1st gen is not needed any more

Achieved self-host again 🎉

- with half time
- with half lines of code (4,900 lines)
  - Composed of only 3 files
    - main.go
    - runtime.go
    - runtime.s
- with much higher readability

Conclusion

Conclusion

- Writing a Go compiler is not that hard
    - as long as you don’t pursue a perfect one
- Making something is the best way to understand it
- This experience helped me understand and contribute to the official compiler

Conclusion

- If you want to learn compilers,
  - I’d recommend babygo or chibicc as materials
    - https://github.com/DQNEO/babygo
    - https://github.com/rui314/chibicc
  - Replaying the commit history is a good way

Conclusion

- No need to be a computer science expert beforehand
- You can just get started

Let’s make
your own Go compiler !

Thank you:
Rui
my colleagues

Thank you for listening

Appendix

About chibicc versions

chibicc was renewed while I was working on this presentation.

The old version I was referring to is here.

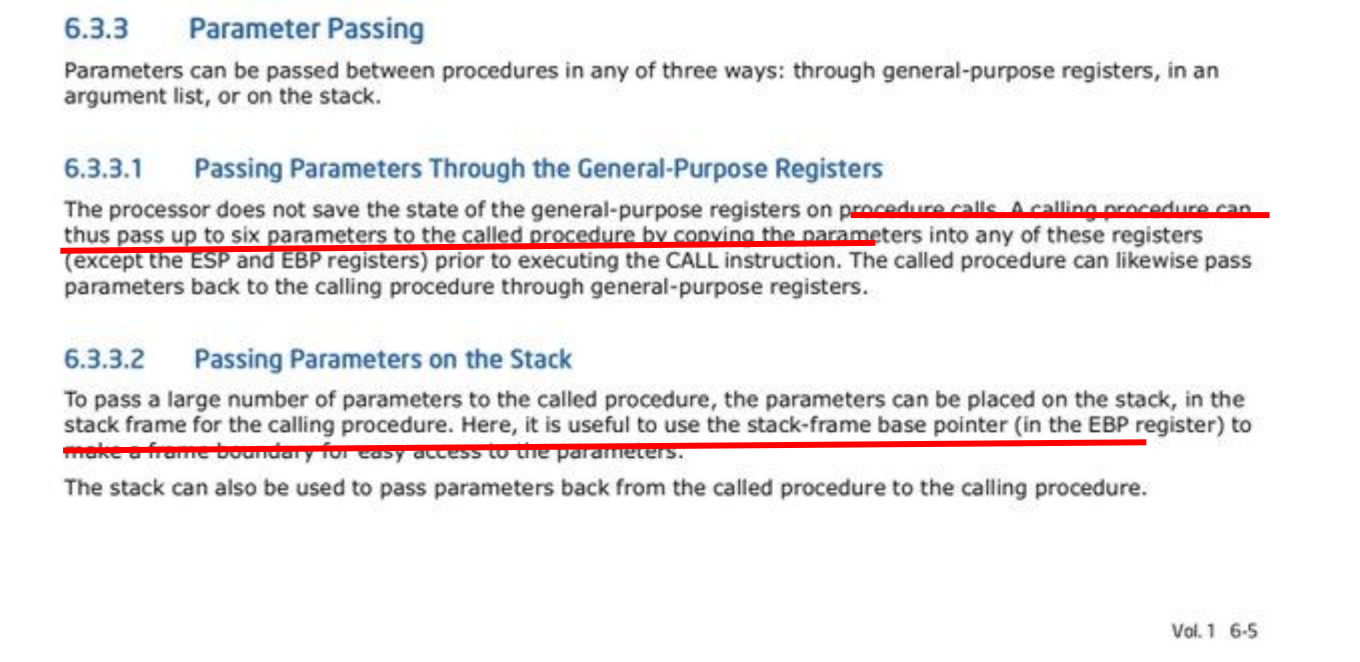
https://github.com/rui314/chibicc/tree/historical/old

How I learned assembly language

- I didn’t read any book about assembly.
- Googled
- StackOverfolwed
- Fed chibicc or gcc with small pieces of C code, and read the output assembly code
- Official documentation (GAS, Intel CPU) are sometimes useful after you’ve got some knowledge

Intel’s manual can be helpful

e.g. How to realize multiple returned values



6.3.3 Parameter Passing

Parameters can be passed between procedures in any of three ways: through general-purpose registers, in an argument list, or on the stack.

6.3.3.1 Passing Parameters Through the General-Purpose Registers

The processor does not save the state of the general-purpose registers on procedure calls. A calling procedure can thus pass up to six parameters to the called procedure by copying the parameters into any of these registers (except the ESP and EBP registers) prior to executing the CALL instruction. The called procedure can likewise pass parameters back to the calling procedure through general-purpose registers.

6.3.3.2 Passing Parameters on the Stack

To pass a large number of parameters to the called procedure, the parameters can be placed on the stack, in the stack frame for the calling procedure. Here, it is useful to use the stack-frame base pointer (in the EBP register) to make a frame boundary for easy access to the parameters.

The stack can also be used to pass parameters back from the called procedure to the calling procedure.

Intel® 64 and IA-32 Architectures Software Developer’s Manual

Refs

- GNU Assembler
    - https://sourceware.org/binutils/docs/as/
- Intel Software Developer Manuals
    - https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html#combined