


DISASSEMBLING \Rightarrow GO

EYAL POST | MAY 2018



```
INCQ AX
MOVQ 0x30(SP), CX
CMPQ CX, AX
JL 0x104c411
MOVQ 0x18(SP), BP
ADDQ $0x20, SP
```



ABOUT ME

POST /Eyal

GETT /Architect

WHY...?

```
INCQ AX  
MOVQ 0x30(SP), CX  
CMPQ CX, AX  
JL 0x104c411  
MOVQ 0x18(SP), BP  
ADDQ $0x20, SP
```



DISASSEMBLING GO

WHY...?

DEEPER UNDERSTANDING OF GO

WHY...?

DEEPER UNDERSTANDING OF GO

MYTH BUSTING

WHY...?

DEEPER UNDERSTANDING OF GO

MYTH BUSTING

BECAUSE I CAN

Before we start...

WARNING



The micro-optimizations shown in this presentation were performed by professionals.

Do not try them in your code!

A close-up photograph of two hands holding lit sparklers. The sparklers are positioned vertically, with their tips glowing and emitting a shower of bright sparks. The background is dark and out of focus, featuring several circular bokeh light spots in warm tones. The overall mood is celebratory and festive.

HOW DID IT START?

UNDERSTANDING EMBEDDING

(Deeper understanding)

```
type Base struct {  
  
}  
  
func (b Base) Print() {  
    fmt.Println("Base")  
}  
  
type Sub struct {  
    Base  
}
```

```
type Base struct {  
    }  
  
func (b Base) Print() {  
    fmt.Println("Base")  
}  
  
type Sub struct {  
    Base  
}  
  
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    b.Print()  
    s.Print()  
}
```

```
ty
```

```
}
```

```
fu
```

HOW DO YOU SEE THE GENERATED ASSEMBLY?

```
}
```

```
type Sub struct {
```

```
    Base
```

```
}
```

```
func Test() {
```

```
    b := Base{}
```

```
    s := Sub{}
```

```
    b.Print()
```

```
    s.Print()
```

```
}
```

HOW DO YOU SEE THE GENERATED ASSEMBLY?

```
> go tool objdump -S -s embed.Test meetup
```

```
TEXT meetup/embed.Test(SB)
```

```
func Test() {  
    0x108e7d0      65488b0c25a0080000  MOVQ GS:0x8a0, CX  
    0x108e7d9      483b6110           CMPQ 0x10(CX), SP  
    0x108e7dd      761f              JBE 0x108e7fe  
    0x108e7df      4883ec08          SUBQ $0x8, SP  
    0x108e7e3      48892c24          MOVQ BP, 0(SP)  
    0x108e7e7      488d2c24          LEAQ 0(SP), BP  
    b.Print()  
    0x108e7eb      e870ffffff        CALL meetup/embed.Base.Print(SB)  
}  
    0x108e7f5      488b2c24          MOVQ 0(SP), BP  
    0x108e7f9      4883c408          ADDQ $0x8, SP  
    0x108e7fd      c3               RET
```

```
type Base struct {  
}
```

```
func (b Base) Print() {  
    fmt.Println("Base")  
}
```

```
type Sub struct {  
    Base  
}
```

```
func Test() {  
    b := Base{  
        s := Sub{  
  
        b.Print()  
        s.Print()  
}
```

TEXT meetup/embed.Test(SB)

func Test() {

0x108e7d0 65488b0c25a0080000 MOVQ GS:0x8a0, CX

0x108e7d9 483b6110 CMPQ 0x10(CX), SP

0x108e7dd 761f JBE 0x108e7fe

0x108e7df 4883ec08 SUBQ \$0x8, SP

0x108e7e3 48892c24 MOVQ BP, 0(SP)

0x108e7e7 488d2c24 LEAQ 0(SP), BP

b.Print()

0x108e7eb e870ffffff CALL meetup/embed.Base.Print(SB)

...

...

}

```

type Base struct {
}

func (b Base) Print() {
    fmt.Println("Base")
}

type Sub struct {
    Base
}

func Test() {
    b := Base{}
    s := Sub{}

    b.Print()
    s.Print()
}

```

```

TEXT meetup/embed.Test(SB)
func Test() {
    0x108e7d0      65488b0c25a0080000    MOVQ GS:0x8a0, CX
    0x108e7d9      483b6110              CMPQ 0x10(CX), SP
    0x108e7dd      761f                 JBE 0x108e7fe
    0x108e7df      4883ec08             SUBQ $0x8, SP
    0x108e7e3      48892c24             MOVQ BP, 0(SP)
    0x108e7e7      488d2c24             LEAQ 0(SP), BP
    b.Print()
    0x108e7eb      e870ffffff          CALL meetup/embed.Base.Print(SB)
    ...
    ...
}

```



```
type Base struct {  
}
```

```
func (b Base) Print() {  
    fmt.Println("Base")  
}
```

```
type Sub struct {  
    Base  
}
```

```
func Test() {  
    b := Base{  
        s := Sub{  
  
        b.Print()  
        s.Print()  
}
```

TEXT meetup/embed.Test(SB)

func Test() {

0x108e7d0 65488b0c25a0080000 MOVQ GS:0x8a0, CX

0x108e7d9 483b6110 CMPQ 0x10(CX), SP

0x108e7dd 761f JBE 0x108e7fe

0x108e7df 4883ec08 SUBQ \$0x8, SP

0x108e7e3 48892c24 MOVQ BP, 0(SP)

0x108e7e7 488d2c24 LEAQ 0(SP), BP

b.Print()

0x108e7eb e870ffffff CALL meetup/embed.Base.Print(SB)

s.Print()

}

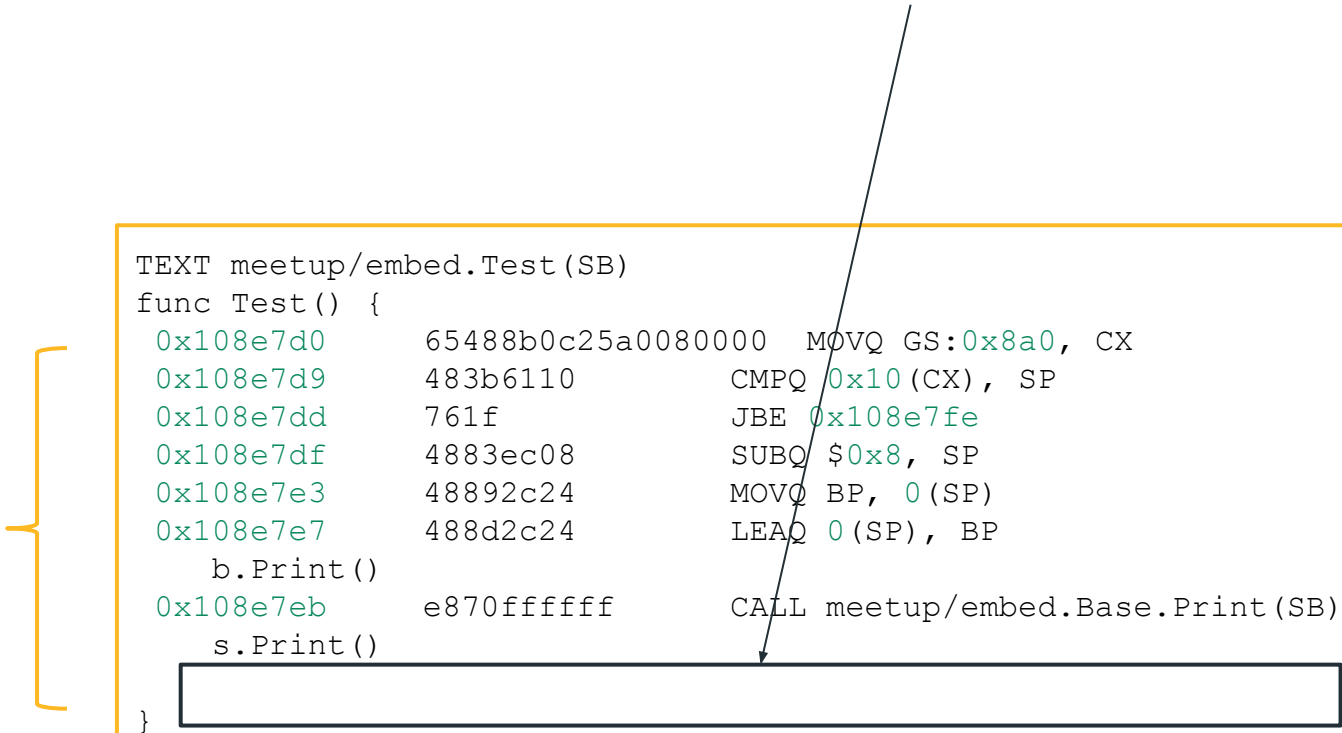
```
type Base struct {  
}
```

```
func (b Base) Print() {  
    fmt.Println("Base")  
}
```

```
type Sub struct {  
    Base  
}
```

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    b.Print()  
    s.Print()  
}
```

What do you expect
to see here?



```
TEXT meetup/embed.Test(SB)  
func Test() {  
    0x108e7d0      65488b0c25a0080000    MOVQ GS:0x8a0, CX  
    0x108e7d9      483b6110              CMPQ 0x10(CX), SP  
    0x108e7dd      761f                 JBE 0x108e7fe  
    0x108e7df      4883ec08              SUBQ $0x8, SP  
    0x108e7e3      48892c24              MOVQ BP, 0(SP)  
    0x108e7e7      488d2c24              LEAQ 0(SP), BP  
    b.Print()  
    0x108e7eb      e870ffffff           CALL meetup/embed.Base.Print(SB)  
    s.Print()  
}
```

```
type Base struct {  
}
```

```
func (b Base) Print() {  
    fmt.Println("Base")  
}
```

```
type Sub struct {  
    Base  
}
```

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    b.Print()  
    s.Print()  
}
```

TEXT meetup/embed.Test(SB)

func Test() {

0x108e7d0 65488b0c25a0080000 MOVQ GS:0x8a0, CX

0x108e7d9 483b6110 CMPQ 0x10(CX), SP

0x108e7dd 761f JBE 0x108e7fe

0x108e7df 4883ec08 SUBQ \$0x8, SP

0x108e7e3 48892c24 MOVQ BP, 0(SP)

0x108e7e7 488d2c24 LEAQ 0(SP), BP

b.Print()

0x108e7eb e870ffffff CALL meetup/embed.Base.Print(SB)

s.Print()

0x108e7f0 e86bffffff CALL meetup/embed.Base.Print(SB)

}

```
> go tool objdump -S -s embed meetup | grep TEXT
```

```
> go tool objdump -S -s embed meetup | grep TEXT
```

**Show all functions in the
generated code**



```
> go tool objdump -S -s embed meetup | grep TEXT
```

```
TEXT meetup/embed.Base.Print(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.Test(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.init(SB) <autogenerated>
```

No Sub.Print method...

```
type Base struct {  
}
```

```
func (b Base) Print() {  
    fmt.Println("Base")  
}
```

```
type Sub struct {  
    Base  
}
```

Wait, But...

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    b.Print()  
    s.Print()  
}
```

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    b.Print()  
    s.Print()  
}
```

Wait, But...
Let's say I do this


```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    basePrint := Base.Print  
}
```

Wait, But...
Let's say I do this

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    basePrint := Base.Print  
}
```

What is the type of basePrint?

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
}
```

It's a function that accepts Base as first parameter

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
}
```

Which means we can pass 'b' as the first parameter

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b) /* == b.Print() */  
}
```

They are equivalent

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
}
```

But what will happen if we do the same with Sub?

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
  
    subPrint := Sub.Print  
}
```

What is the type of subPrint?

```
func Test() {  
    b := Base{  
    s := Sub{  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
  
    subPrint := Sub.Print  
    subPrint(?)  
}
```

Should we pass 'b' or 's' to it?


```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
  
    subPrint := Sub.Print  
    subPrint(s)  
}
```

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
  
    subPrint := Sub.Print  
    subPrint(s)  
}
```

How does it work?

```
func Test() {  
    b := Base{}  
    s := Sub{}  
  
    var basePrint func(Base)  
    basePrint = Base.Print  
    basePrint(b)  
  
    subPrint := Sub.Print  
    subPrint(s)  
}
```

How does it work?

```
> go tool objdump -S -s embed meetup | grep TEXT
```

```
TEXT meetup/embed.Base.Print(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.Test(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.init(SB) <autogenerated>
```

```
TEXT meetup/embed.Sub.Print(SB) <autogenerated>
```

```
> go tool objdump -S -s embed meetup | grep TEXT
```

```
TEXT meetup/embed.Base.Print(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.Test(SB) /Users/eyalp/go/src/meetup/embed/embed.go
```

```
TEXT meetup/embed.init(SB) <autogenerated>
```

```
TEXT meetup/embed.Sub.Print(SB) <autogenerated>
```

Hello Sub.Print!

BOTTOM LINE?

BOTTOM LINE?

**The compiler does not always do stuff
the way we expect it to do it**

SWITCH VS MAP

(Myth busting)

Enums in Go

```
type Animal int32
```

```
const (  
    Dog Animal = iota  
    Cat  
    Lemur  
)
```

Enums in Go

```
func (i Animal) String() string {  
    switch i {  
    case Dog:  
        return "Dog"  
    case Cat:  
        return "Cat"  
    case Lemur:  
        return "Lemur"  
    default:  
        return fmt.Sprintf("Animal_%d", i)  
    }  
}
```

```
func ParseAnimal(name string) (Animal, error) {
    switch name {
    case "Dog":
        return Dog, nil
    case "Cat":
        return Cat, nil
    case "Lemur":
        return Lemur, nil
    default:
        return Animal(0), fmt.Errorf("Invalid %s", name)
    }
}
```

Enums in Go

```
func ParseAnimal(name string) (Animal, error) {  
    switch name {  
    case "Dog":  
        return Dog, nil  
    case "Cat":  
        return Cat, nil  
    case "Lemur":  
        return Lemur, nil  
    default:  
        return Animal, error  
    }  
}
```

WritePreview

AA B i “ < > ↻ ☰ ☷ ☹ ↶ @ 🚩

Suggest to use a **map** instead of a **switch** case.
If the enum will grow to more than 20 values, the
difference will become noticable

MD Styling with Markdown is supported

CancelAdd single commentStart a review

REALLY?

REALLY?

Let's benchmark

Enums in Go

```
var animals = map[string]Animal{
    "Dog":    Dog,
    "Cat":    Cat,
    "Lemur":  Lemur,
}

func ParseAnimalViaMap(name string) (Animal, error) {
    if x, ok := animals[name]; ok {
        return x, nil
    }
    return Animal(0), fmt.Errorf("Invalid value %s", name)
}
```

```
func ParseAnimalViaSwitch(name string) (Animal, error) {  
    switch name {  
    case "Dog":  
        return Dog, nil  
    case "Cat":  
        return Cat, nil  
    case "Lemur":  
        return Lemur, nil  
    default:  
        return Animal(0), fmt.Errorf("Invalid value %s", name)  
    }  
}
```


Enums in Go

3 enums:

BenchmarkParseX-4

BenchmarkParseY-4

3000000

3000000

488 ns/op

451 ns/op



Which is which?

Enums in Go

3 enums:

BenchmarkParseViaMap-4	3000000	488 ns/op
BenchmarkParseViaSwitch-4	3000000	451 ns/op

Enums in Go

3 enums:

BenchmarkParseViaMap-4	3000000	488 ns/op
BenchmarkParseViaSwitch-4	3000000	451 ns/op

6 enums

BenchmarkParseViaMap-4	3000000	555 ns/op
BenchmarkParseViaSwitch-4	3000000	523 ns/op

Enums in Go

3 enums:

BenchmarkParseViaMap-4	3000000	488 ns/op
BenchmarkParseViaSwitch-4	3000000	451 ns/op

6 enums

BenchmarkParseViaMap-4	3000000	555 ns/op
BenchmarkParseViaSwitch-4	3000000	523 ns/op

24 enums

BenchmarkParseViaMap-4	1000000	1387 ns/op
BenchmarkParseViaSwitch-4	1000000	1288 ns/op

Enums in Go

3 enums:

BenchmarkParseViaMap-4	3000000	488 ns/op
BenchmarkParseViaSwitch-4	3000000	451 ns/op

6 enums

BenchmarkParseViaMap-4	3000000	555 ns/op
BenchmarkParseViaSwitch-4	3000000	523 ns/op

24 enums

BenchmarkParseViaMap-4	1000000	1387 ns/op
BenchmarkParseViaSwitch-4	1000000	1288 ns/op

100 enums

BenchmarkParseViaMap-4	300000	4141 ns/op
BenchmarkParseViaSwitch-4	300000	4065 ns/op

Enums in Go

6 enums

BenchmarkParseViaMap-4	3000000	555 ns/op
BenchmarkParseViaSwitch-4	3000000	523 ns/op

24 enums

BenchmarkParseViaMap-4	1000000	1387 ns/op
BenchmarkParseViaSwitch-4	1000000	1288 ns/op

100 enums

BenchmarkParseViaMap-4	300000	4141 ns/op
BenchmarkParseViaSwitch-4	300000	4065 ns/op

150 enums

BenchmarkParseViaMap-4	300000	5515 ns/op
BenchmarkParseViaSwitch-4	300000	5570 ns/op

HOW COME?

HOW COME?

Let's try to understand how a switch case is implemented...


```
func intToString(num int) string {  
    switch num {  
        case 1:  
            return "1"  
        case 2:  
            return "2"  
        case 3:  
            return "3"  
        case 4:  
            return "4"  
        case 5:  
            return "5"  
        case 6:  
            return "6"  
        case 7:  
            return "7"  
        case 8:  
            return "8"  
        case 9:  
            return "9"  
        case 10:  
            return "10"  
        case 11:  
            return "11"  
        case 12:  
            return "12"  
        default:  
            return "N/A"  
    }  
}
```

```

TEXT meetup/swtch.intToString(SB)
func intToString(num int) string {
    0x10913e0 488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5 4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9 0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef 4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3 7f6a        JG 0x109145f
    case 1:
    0x10913f5 4883f801    CMPQ $0x1, AX
    0x10913f9 744e        JE 0x1091449
    case 2:
    0x10913fb 4883f802    CMPQ $0x2, AX
    0x10913ff 7432        JE 0x1091433
    switch num {
    0x1091401 4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405 7416        JE 0x109141d
    return "N/A"
    0x1091407 488d05db290300    LEAQ go.string.*+249(SB), AX
    0x109140e 4889442410    MOVQ AX, 0x10(SP)
    0x1091413 48c744241803000000    MOVQ $0x3, 0x18(SP)
    0x109141c c3          RET
    return "3"
    0x109141d 488d05d8280300    LEAQ go.string.*+12(SB), AX
    0x1091424 4889442410    MOVQ AX, 0x10(SP)
    0x1091429 48c744241801000000    MOVQ $0x1, 0x18(SP)
    0x1091432 c3          RET

```

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a        JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e        JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432        JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416        JE 0x109141d
    return "N/A"

```

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
        case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
        switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
            case 3:
    0x10913f3    7f6a        JG 0x109145f
            case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e        JE 0x1091449
            case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432        JE 0x1091433
        switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
            case 3:
    0x1091405    7416        JE 0x109141d
        return "N/A"
    }
}

```

Compare num to 6

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

Jump if Greater

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

In not...

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

Compare to 3

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a        JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e        JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432        JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416        JE 0x109141d
    return "N/A"

```

Jump if Greater

...


```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

If not...

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

Compare to 1

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

Compare to 2

...

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

Compare to 3

CMPQ \$0x3, AX

```

TEXT meetup/switch.intToString(SB)
func intToString(num int) string {
    0x10913e0    488b442408    MOVQ 0x8(SP), AX
    switch num {
    0x10913e5    4883f806    CMPQ $0x6, AX
    case 6:
    0x10913e9    0f8fbe000000    JG 0x10914ad
    switch num {
    0x10913ef    4883f803    CMPQ $0x3, AX
    case 3:
    0x10913f3    7f6a    JG 0x109145f
    case 1:
    0x10913f5    4883f801    CMPQ $0x1, AX
    0x10913f9    744e    JE 0x1091449
    case 2:
    0x10913fb    4883f802    CMPQ $0x2, AX
    0x10913ff    7432    JE 0x1091433
    switch num {
    0x1091401    4883f803    CMPQ $0x3, AX
    case 3:
    0x1091405    7416    JE 0x109141d
    return "N/A"

```

These are the actual
switch cases

...

```

...
switch num {
0x10914ad ← 4883f809    CMPQ $0x9, AX
    case 9:
0x10914b1    7f4e        JG 0x1091501
    case 7:
0x10914b3    4883f807    CMPQ $0x7, AX
0x10914b7    7432        JE 0x10914eb
    case 8:
0x10914b9    4883f808    CMPQ $0x8, AX
0x10914bd    7516        JNE 0x10914d5
    return "8"
0x10914bf    488d053b280300    LEAQ go.string.*+17(SB), AX
0x10914c6    4889442410    MOVQ AX, 0x10(SP)
0x10914cb    48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914d4    c3          RET
    return "9"
0x10914d5    488d0526280300    LEAQ go.string.*+18(SB), AX
0x10914dc    4889442410    MOVQ AX, 0x10(SP)
0x10914e1    48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914ea    c3          RET
    return "7"
...

```

```

...
switch num {
0x10914ad 4883f809    CMPQ $0x9, AX
case 9:
0x10914b1 7f4e      JG 0x1091501
case 7:
0x10914b3 4883f807    CMPQ $0x7, AX
0x10914b7 7432      JE 0x10914eb
case 8:
0x10914b9 4883f808    CMPQ $0x8, AX
0x10914bd 7516      JNE 0x10914d5
return "8"
0x10914bf 488d053b280300    LEAQ go.string.*+17(SB), AX
0x10914c6 4889442410    MOVQ AX, 0x10(SP)
0x10914cb 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914d4 c3      RET
return "9"
0x10914d5 488d0526280300    LEAQ go.string.*+18(SB), AX
0x10914dc 4889442410    MOVQ AX, 0x10(SP)
0x10914e1 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914ea c3      RET
return "7"
...

```

Compare num to 9

```

...
switch num {
0x10914ad 4883f809    CMPQ $0x9, AX
case 9:
0x10914b1 7f4e      JG 0x1091501
case 7:
0x10914b3 4883f807    CMPQ $0x7, AX
0x10914b7 7432      JE 0x10914eb
case 8:
0x10914b9 4883f808    CMPQ $0x8, AX
0x10914bd 7516      JNE 0x10914d5
return "8"
0x10914bf 488d053b280300    LEAQ go.string.*+17(SB), AX
0x10914c6 4889442410    MOVQ AX, 0x10(SP)
0x10914cb 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914d4 c3      RET
return "9"
0x10914d5 488d0526280300    LEAQ go.string.*+18(SB), AX
0x10914dc 4889442410    MOVQ AX, 0x10(SP)
0x10914e1 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914ea c3      RET
return "7"
...

```

Jump if greater


```

...
switch num {
0x10914ad 4883f809    CMPQ $0x9, AX
case 9:
0x10914b1 7f4e      JG 0x1091501
case 7:
0x10914b3 4883f807    CMPQ $0x7, AX
0x10914b7 7432      JE 0x10914eb
case 8:
0x10914b9 4883f808    CMPQ $0x8, AX
0x10914bd 7516      JNE 0x10914d5
return "8"
0x10914bf 488d053b280300    LEAQ go.string.*+17(SB), AX
0x10914c6 4889442410    MOVQ AX, 0x10(SP)
0x10914cb 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914d4 c3      RET
return "9"
0x10914d5 488d0526280300    LEAQ go.string.*+18(SB), AX
0x10914dc 4889442410    MOVQ AX, 0x10(SP)
0x10914e1 48c744241801000000    MOVQ $0x1, 0x18(SP)
0x10914ea c3      RET
return "7"
...

```

Switch cases

HOW IS A SWITCH CASE IMPLEMENTED?

HOW IS A SWITCH CASE IMPLEMENTED?

Binary Search!

**HOW IS IT IMPLEMENTED WHEN
WORKING ON STRINGS?**

HOW IS IT IMPLEMENTED WHEN WORKING ON STRINGS?

Binary search over the string length

HOW IS IT IMPLEMENTED WHEN WORKING ON STRINGS?

And then over the content

BOTTOM LINE?

BOTTOM LINE?

Don't assume - benchmark!

BOTTOM LINE?

Don't assume - benchmark!
Don't waste time on micro-optimizations

CONCATENATING STRINGS

(Myth busting)

```
func YoureDoingItWrong() {  
    s := "Some string"  
    for i := 0; i < 100; i++ {  
        s = s + strconv.Itoa(i)  
    }  
    fmt.Println(s)  
}
```

```
func YoureDoingItWrong() {  
    s := "Some string"  
    for i := 0; i < 100; i++ {  
        s = s + strconv.Itoa(i)  
    }  
    fmt.Println(s)  
}
```

WHAT'S WRONG WITH THIS CODE?

```
func YoureDoingItWrong() {  
    s := "Some string"  
    for i := 0; i < 100; i++ {  
        s = s + strconv.Itoa(i)  
    }  
    fmt.Println(s)  
}
```

WHAT'S WRONG WITH THIS CODE?

STRING REALLOCATION

```
func YoureDoingItRight() {  
    b := bytes.Buffer{}  
    b.WriteString("Some string")  
    for i := 0; i < 100; i++ {  
        b.WriteString(strconv.Itoa(i))  
    }  
    fmt.Println(b.String())  
}
```

```
func YoureDoingItRight_1_10() {  
    s := strings.Builder{}  
    s.WriteString("Some string")  
    for i := 0; i < 100; i++ {  
        s.WriteString(strconv.Itoa(i))  
    }  
    fmt.Println(s.String())  
}
```

```
func getAnimalLatLng(animalId string, country string) LatLng {  
    cacheKey := buildCacheKey(animalId, country)  
    return getLocationFromCache(cacheKey)  
}
```



```
func getAnimalLatLng(animalId string, country string) LatLng {  
    cacheKey := buildCacheKey(animalId, country)  
    return getLocationFromCache(cacheKey)  
}
```

```
func buildCacheKey(animalId string, country string) string {  
    return "location:" + animalId + ":" + country  
}
```

```
func getAnimalLatLng(animalId string, country string) LatLng {  
    cacheKey := buildCacheKey(animalId, country)  
    return getLocationFromCache(cacheKey)  
}
```

```
func buildCacheKey(animalId string, country string) string {  
    return "location:" + animalId + ":" + country  
}
```



Write Preview

AA B i “ <> 🔗 ☰ ☷ ☹ ↶ @ 🚩

Concatenating strings this way causes redundant memory allocations. Use **strings.Join()** or **strings.Builder** instead

📄 Styling with Markdown is supported

Cancel Add single comment Start a review

REALLY?



REALLY?

Let's benchmark

REALLY?

~~Let's benchmark~~
Let's disassemble

```
func addStrings(s1 string, s2 string, s3 string) string {  
    return strings.Join([]string{s1, s2, s3}, ":")  
}
```

```
func addStrings(s1 string, s2 string, s3 string) string {
    return strings.Join([]string{s1, s2, s3}, ":")
}
```

```
strconc.go:23 0x108ed91 0f57c0      XORPS X0, X0
strconc.go:23 0x108ed94 0f11442438  MOVUPS X0, 0x38(SP)
strconc.go:23 0x108ed99 0f11442448  MOVUPS X0, 0x48(SP)
strconc.go:23 0x108ed9e 0f11442458  MOVUPS X0, 0x58(SP)
strconc.go:22 0x108eda3 488b442478  MOVQ 0x78(SP), AX
strconc.go:23 0x108eda8 4889442438  MOVQ AX, 0x38(SP)
strconc.go:22 0x108edad 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:23 0x108edb5 4889442440  MOVQ AX, 0x40(SP)
strconc.go:22 0x108edba 488b842488000000 MOVQ 0x88(SP), AX
strconc.go:23 0x108edc2 4889442448  MOVQ AX, 0x48(SP)
strconc.go:22 0x108edc7 488b842490000000 MOVQ 0x90(SP), AX
strconc.go:23 0x108edcf 4889442450  MOVQ AX, 0x50(SP)
strconc.go:22 0x108edd4 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:23 0x108eddc 4889442458  MOVQ AX, 0x58(SP)
strconc.go:22 0x108edel 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:23 0x108ede9 4889442460  MOVQ AX, 0x60(SP)
strconc.go:23 0x108edee 488d442438  LEAQ 0x38(SP), AX
strconc.go:23 0x108edf3 48890424    MOVQ AX, 0(SP)
strconc.go:23 0x108edf7 48c744240803000000 MOVQ $0x3, 0x8(SP)
strconc.go:23 0x108ee00 48c744241003000000 MOVQ $0x3, 0x10(SP)
strconc.go:23 0x108ee09 488d052b130300  LEAQ go.string.*t1(SB), AX
strconc.go:23 0x108ee10 4889442418  MOVQ AX, 0x18(SP)
strconc.go:23 0x108ee15 48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:23 0x108ee1e e83df9ffff  CALL strings.Join(SB)
```

```

strconc.go:23      0x108ed91      0f57c0      XORPS X0, X0
strconc.go:23      return strings.Join([]string{s1, s2, s3}, ":")

```

```

strconc.go:23      0x108ed9e      0f11442458      MOVUPS X0, 0x58(SP)
strconc.go:22      0x108eda3      488b442478      MOVQ 0x78(SP), AX
strconc.go:23      0x108eda8      4889442438      MOVQ AX, 0x38(SP)
strconc.go:22      0x108edad      488b842480000000 MOVQ 0x80(SP), AX
strconc.go:23      0x108edb5      4889442440      MOVQ AX, 0x40(SP)
strconc.go:22      0x108edba      488b842488000000 MOVQ 0x88(SP), AX
strconc.go:23      0x108edc2      4889442448      MOVQ AX, 0x48(SP)
strconc.go:22      0x108edc7      488b842490000000 MOVQ 0x90(SP), AX
strconc.go:23      0x108edcf      4889442450      MOVQ AX, 0x50(SP)
strconc.go:22      0x108edd4      488b842498000000 MOVQ 0x98(SP), AX
strconc.go:23      0x108eddc      4889442458      MOVQ AX, 0x58(SP)
strconc.go:22      0x108ede1      488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:23      0x108ede9      4889442460      MOVQ AX, 0x60(SP)
strconc.go:23      0x108edee      488d442438      LEAQ 0x38(SP), AX
strconc.go:23      0x108edf3      48890424      MOVQ AX, 0(SP)
strconc.go:23      0x108edf7      48c744240803000000 MOVQ $0x3, 0x8(SP)
strconc.go:23      0x108ee00      48c744241003000000 MOVQ $0x3, 0x10(SP)
strconc.go:23      0x108ee09      488d052b130300      LEAQ go.string.*+ 11(SB), AX
strconc.go:23      0x108ee10      4889442418      MOVQ AX, 0x18(SP)
strconc.go:23      0x108ee15      48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:23      0x108ee1e      e83df9ffff      CALL strings.Join(SB)

```



```
strconc.go:23 0x108ed91 0f57c0 XORPS X0, X0
str return strings.Join([]string{s1, s2, s3}, ":")
str
```

```
strconc.go:23 0x108ed9e 0f11442458 MOVUPS X0, 0x58(SP)
strconc.go:22 0x108eda3 488b442478 MOVQ 0x78(SP), AX
strconc.go:23 0x108eda8 4889442438 MOVQ AX, 0x38(SP)
strconc.go:22 0x108edad 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:23 0x108edb5 4889442440 MOVQ AX, 0x40(SP)
strconc.go:22 0x108edba 488b842488000000 MOVQ 0x88(SP), AX
strconc.go:23 0x108edc2 4889442448 MOVQ AX, 0x48(SP)
```

The ":" parameter

```
strconc.go:22 0x108edd4 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:23 0x108eddc 4889442458 MOVQ AX, 0x58(SP)
strconc.go:22 0x108ede1 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:23 0x108ede9 4889442460 MOVQ AX, 0x60(SP)
strconc.go:23 0x108edee 488d442438 LEAQ 0x38(SP), AX
strconc.go:23 0x108edf3 48890424 MOVQ AX, 0(SP)
strconc.go:23 0x108edf7 48c744240803000000 MOVQ $0x3, 0x8(SP)
strconc.go:23 0x108ee00 48c744241003000000 MOVQ $0x3, 0x10(SP)
strconc.go:23 0x108ee09 488d052b130300 LEAQ go.string.*+ 11(SB), AX
strconc.go:23 0x108ee10 4889442418 MOVQ AX, 0x18(SP)
strconc.go:23 0x108ee15 48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:23 0x108ee1e e83df9ffff CALL strings.Join(SB)
```

```

strconc.go:23 0x108ed91 0f57c0 XORPS X0, X0
str return strings.Join([]string{s1, s2, s3}, ":")
str

```

```

strconc.go:23 0x108ed9e 0f11442458 MOVUPS X0, 0x58(SP)
strconc.go:22 0x108eda3 488b442478 MOVQ 0x78(SP), AX
strconc.go:23 0x108eda8 4889442438 MOVQ AX, 0x38(SP)
strconc.go:22 0x108edad 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:23 0x108edb5 4889442440 MOVQ AX, 0x40(SP)
strconc.go:22 0x108edba 488b842488000000 MOVQ 0x88(SP), AX
strconc.go:23 0x108edc2 4889442448 MOVQ AX, 0x48(SP)

```

Call the strings.Join() function

```

strconc.go:22 0x108edd4 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:23 0x108eddc 4889442458 MOVQ AX, 0x58(SP)
strconc.go:22 0x108ede1 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:23 0x108ede9 4889442460 MOVQ AX, 0x60(SP)
strconc.go:23 0x108edee 488d442438 LEAQ 0x38(SP), AX
strconc.go:23 0x108edf3 48890424 MOVQ AX, 0(SP)
strconc.go:23 0x108edf7 48c744240803000000 MOVQ $0x3, 0x8(SP)
strconc.go:23 0x108ee00 48c744241003000000 MOVQ $0x3, 0x10(SP)
strconc.go:23 0x108ee09 488d052b130300 LEAQ go.string.*+11(SB), AX
strconc.go:23 0x108ee10 4889442418 MOVQ AX, 0x18(SP)
strconc.go:23 0x108ee15 48c744242001000000 MOVO $0x1, 0x20(SP)
strconc.go:23 0x108ee1e e83df9ffff CALL strings.Join(SB)

```

```
func addStrings(s1 string, s2 string, s3 string) string {  
    return s1 + ":" + s2 + ":" + s3  
}
```

```
func addStrings(s1 string, s2 string, s3 string) string {  
    return s1 + ":" + s2 + ":" + s3  
}
```

WHAT DO YOU EXPECT TO SEE HERE?

```
func addStrings(s1 string, s2 string, s3 string) string {
    return s1 + ":" + s2 + ":" + s3
}
```

```
strconc.go:17 0x108e8fd 0f86ad000000 JBE 0x108e9b0
strconc.go:17 0x108e903 4883ec70 SUBQ $0x70, SP
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
strconc.go:18 0x108e937 4889442418 MOVQ AX, 0x18(SP)
strconc.go:18 0x108e93c 48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e918 48896c2468 MOVQ 0x78(SP), AX
strconc.go:17 0x108e91d 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:17 0x108e922 48896c2468 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e929 488b8c2488000000 MOVQ 0x80(SP), AX
strconc.go:17 0x108e930 48896c2468 MOVQ AX, 0x10(SP)
strconc.go:17 0x108e937 48896c2468 LEAQ go.string.*+ 11(SB), AX
strconc.go:18 0x108e93c 48896c2468 MOVQ AX, 0x18(SP)
strconc.go:18 0x108e943 48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48896c2468 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48896c2468 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 48896c2468 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b8c2498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 48896c2468 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8c24a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 48896c2468 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

**Just moving a lot of parameters
from the stack to the next
function**

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
strconc.go:18 0x108e937 4889442418 MOVQ AX, 0x18(SP)
strconc.go:18 0x108e93c 48c744242001000000 MOVQ $0x1, 0x20(SP)
strconc.go:17 0x108e945 488b042488000000 MOVQ 0x88(SP), CX
strconc.go:17 0x108e94b 488b042428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b042490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

Call runtime.concatstring5 func

CALL runtime.concatstring5(SB)


```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
strconc.go:18 0x108e937 4889442408 MOVQ AX, 0x18(SP)
strconc.go:18 0x108e93e 4889442408 MOVQ AX, 0x20(SP)
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
```

Yes!

```
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
```

And there's also concatstring2

```
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
```

And there's also concatstring2,3

```
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
```

And there's also concatstring2,3,4

```
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

```
return s1 + ":" + s2 + ":" + s3
```

```
strconc.go:17 0x108e907 48896c2468 MOVQ BP, 0x68(SP)
strconc.go:17 0x108e90c 488d6c2468 LEAQ 0x68(SP), BP
strconc.go:18 0x108e911 48c7042400000000 MOVQ $0x0, 0(SP)
strconc.go:17 0x108e919 488b442478 MOVQ 0x78(SP), AX
strconc.go:18 0x108e91e 4889442408 MOVQ AX, 0x8(SP)
strconc.go:17 0x108e923 488b842480000000 MOVQ 0x80(SP), AX
strconc.go:18 0x108e92b 4889442410 MOVQ AX, 0x10(SP)
strconc.go:18 0x108e930 488d0564130300 LEAQ go.string.*+11(SB), AX
```

And 'concatstring' which takes a slice of strings

```
strconc.go:17 0x108e945 488b8c2488000000 MOVQ 0x88(SP), CX
strconc.go:18 0x108e94d 48894c2428 MOVQ CX, 0x28(SP)
strconc.go:17 0x108e952 488b8c2490000000 MOVQ 0x90(SP), CX
strconc.go:18 0x108e95a 48894c2430 MOVQ CX, 0x30(SP)
strconc.go:18 0x108e95f 4889442438 MOVQ AX, 0x38(SP)
strconc.go:18 0x108e964 48c744244001000000 MOVQ $0x1, 0x40(SP)
strconc.go:17 0x108e96d 488b842498000000 MOVQ 0x98(SP), AX
strconc.go:18 0x108e975 4889442448 MOVQ AX, 0x48(SP)
strconc.go:17 0x108e97a 488b8424a0000000 MOVQ 0xa0(SP), AX
strconc.go:18 0x108e982 4889442450 MOVQ AX, 0x50(SP)
strconc.go:18 0x108e987 e8a4ebfaff CALL runtime.concatstring5(SB)
```

WHAT DOES `runtime.concatstringX` DO?

WHAT DOES `runtime.concatstringX` DO?

Basically the same as `Strings.Join`

IN FACT...

```
package strings
...
...
func Join(a []string, sep string) string {
    switch len(a) {
    case 0:
        return ""
    case 1:
        return a[0]
    case 2:
        return a[0] + sep + a[1]
    case 3:
        return a[0] + sep + a[1] + sep + a[2]
    }
    ...
    //rest of strings.join code here
}
```

```
package strings
```

```
...
```

```
...
```

```
func Join(a []string, sep string) string {
```

```
    switch len(a) {
```

```
    case 0:
```

```
        return ""
```

```
    case 1:
```

```
        return a[0]
```

```
    case 2:
```

```
        return a[0] + sep + a[1]
```

```
    case 3:
```

```
        return a[0] + sep + a[1] + sep + a[2]
```

```
    }
```

```
...
```

```
//rest of strings.join code here
```

```
}
```

runtime.concatstring3



runtime.concatstring5

BOTTOM LINE?

BOTTOM LINE?

Don't assume - disassemble!

BOTTOM LINE?

**Abandon myths from past experience
since they might not be relevant in Go**

SLICES, RANGES ETC.

(Because I can)

```
func GetVal(ia []int, index int) int {  
    return ia[index]  
}
```



```
func GetVal(ia []int, index int) int {  
    return ia[index]  
}
```

WHICH ONE GENERATES “MORE” ASSEMBLY CODE?

```
func GetVal(ia []int, index int) int {  
    if index >= 0 && index < len(ia) {  
        return ia[index]  
    }  
    return 0  
}
```

```
func GetVal(ia []int, index int) int {  
    return ia[index]  
}
```

LET'S COMPARE

```
func GetVal(ia []int, index int) int {  
    if index >= 0 && index < len(ia) {  
        return ia[index]  
    }  
    return 0  
}
```

range.go:79	0x104c450	4883ec08	SUBQ \$ 0x8, SP
range.go:79	0x104c454	48892c24	MOVQ BP, 0(SP)
range.go:79	0x104c458	488d2c24	LEAQ 0(SP), BP
range.go:79	0x104c45c	488b442428	MOVQ 0x28(SP), AX
range.go:79	0x104c461	488b4c2418	MOVQ 0x18(SP), CX
range.go:80	0x104c466	4839c8	CMPQ CX, AX
range.go:80	0x104c469	7317	JAE 0x104c482
range.go:80	0x104c46b	488b4c2410	MOVQ 0x10(SP), CX
range.go:80	0x104c470	488b04c1	MOVQ 0(CX)(AX*8), AX
range.go:80	0x104c474	4889442430	MOVQ AX, 0x30(SP)
range.go:80	0x104c479	488b2c24	MOVQ 0(SP), BP
range.go:80	0x104c47d	4883c408	ADDQ \$ 0x8, SP
range.go:80	0x104c481	c3	RET
range.go:80	0x104c482	e8f93bfdff	CALL runtime.panicindex(SB)
range.go:80	0x104c487	0f0b	UD2

```
func GetVal(ia []int, index int) int {
    return ia[index]
}
```

range.go:79	0x104c450	4883ec08	SUBQ \$ 0x8, SP
range.go:79	0x104c454	48892c24	MOVQ BP, 0(SP)
range.go:79	0x104c458	488d2c24	LEAQ 0(SP), BP
range.go:79	0x104c45c	488b442428	MOVQ 0x28(SP), AX
range.go:79	0x104c461	488b4c2418	MOVO 0x18(SP), CX
range.go:80	0x104c466	4839c8	CMPQ CX, AX
range.go:80	0x104c469	7317	JAE 0x104c482
range.go:80	0x104c46b	488b4c2410	MOVQ 0x10(SP), CX
		488b04c1	MOVQ 0(CX) (AX*8), AX
		4889442430	MOVQ AX, 0x30(SP)
		488b2c24	MOVQ 0(SP), BP
		4883c408	ADDQ \$ 0x8, SP
		c3	RET
		e8f93bfdf	CALL runtime.panicindex(SB)
		0f0b	UD2

Bounds checking

```
func GetVal(ia []int, index int) int {
    return ia[index]
}
```

range.go:71	0x104c420	488b442420	MOVQ 0x20(SP), AX
range.go:72	0x104c425	4885c0	TESTQ AX, AX
range.go:72	0x104c428	7c0a	JL 0x104c434
range.go:72	0x104c42a	488b4c2410	MOVQ 0x10(SP), CX
range.go:72	0x104c42f	4839c8	CMPQ CX, AX
range.go:72	0x104c432	7c0a	JL 0x104c43e
range.go:75	0x104c434	48c744242800000000	MOVQ \$ 0x0, 0x28(SP)
range.go:75	0x104c43d	c3	RET
range.go:75	0x104c43e	488b4c2408	MOVQ 0x8(SP), CX
range.go:73	0x104c443	488b04c1	MOVQ 0(CX)(AX*8), AX
range.go:73	0x104c447	4889442428	MOVQ AX, 0x28(SP)
range.go:73	0x104c44c	c3	RET

```
func GetVal(ia []int, index int) int {
    if index >= 0 && index < len(ia) {
        return ia[index]
    }
    return 0
}
```

range.go:71	0x104c420	488b442420	MOVQ 0x20(SP), AX
range.go:72	0x104c425	4885c0	TESTQ AX, AX
range.go:72	0x104c428	7c0a	JL 0x104c434
range.go:72	0x104c42a	488b4c2410	MOVQ 0x10(SP), CX
range.go:72	0x104c42f	4839c8	CMPQ CX, AX
range.go:72	0x104c432	7c0a	JL 0x104c43e
range.go:75	0x104c434	48c744242800000000	MOVQ \$ 0x0, 0x28(SP)
range.go:75	0x104c43d	c3	RET
Manual bounds checking			
range.go:73	0x104c447	488b4c2408	MOVQ 0x8(SP), CX
range.go:73	0x104c44c	488b04c1	MOVQ 0(CX)(AX*8), AX
		4889442428	MOVQ AX, 0x28(SP)
		c3	RET

```
func GetVal(ia []int, index int) int {
    if index >= 0 && index < len(ia) {
        return ia[index]
    }
    return 0
}
```

BCE

BCE

Bounds Check Elimination

BCE

Bounds Check Elimination

LET'S LOOK AT ANOTHER EXAMPLE...

```
func Sum(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

```
func Sum(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

Let's look at the generated code

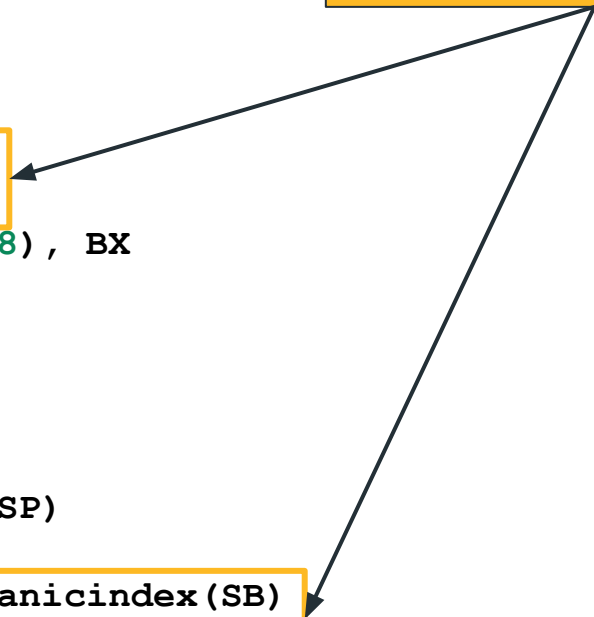
TEXT	meetup/rng.Sum(SB)	
0x309b6	488b7c2408	MOVQ 0x8(SP), DI
0x309bb	488b742410	MOVQ 0x10(SP), SI
0x309c0	31c9	XORL CX, CX
0x309c2	31c0	XORL AX, AX
0x309c4	4839f0	CMPQ SI, AX
0x309c7	7d17	JGE 0x309e0
0x309c9	4839f0	CMPQ SI, AX
0x309cc	7318	JAE 0x309e6
0x309ce	488d1cc7	LEAQ 0(DI)(AX*8), BX
0x309d2	488b2b	MOVQ 0(BX), BP
0x309d5	4801e9	ADDQ BP, CX
0x309d8	48ffc0	INCQ AX
0x309db	4839f0	CMPQ SI, AX
0x309de	7ce9	JL 0x309c9
0x309e0	48894c2420	MOVQ CX, 0x20(SP)
0x309e5	c3	RET
0x309e6	e8f5a0fdff	CALL runtime.panicindex(SB)
0x309eb	0f0b	UD2

TEXT meetup/rng.Sum(SB)

0x309b6	488b7c2408
0x309bb	488b742410
0x309c0	31c9
0x309c2	31c0
0x309c4	4839f0
0x309c7	7d17
0x309c9	4839f0
0x309cc	7318
0x309ce	488d1cc7
0x309d2	488b2b
0x309d5	4801e9
0x309d8	48ffc0
0x309db	4839f0
0x309de	7ce9
0x309e0	48894c2420
0x309e5	c3
0x309e6	e8f5a0fdff
0x309eb	0f0b

```
MOVQ 0x8(SP), DI
MOVQ 0x10(SP), SI
XORL CX, CX
XORL AX, AX
CMPQ SI, AX
JGE 0x309e0
CMPQ SI, AX
JAE 0x309e6
LEAQ 0(DI)(AX*8), BX
MOVQ 0(BX), BP
ADDQ BP, CX
INCQ AX
CMPQ SI, AX
JL 0x309c9
MOVQ CX, 0x20(SP)
RET
CALL runtime.panicindex(SB)
UD2
```

Bounds checking



TEXT meetup/rng.Sum(SB)

0x309b6	488b7c2408
0x309bb	488b742410
0x309c0	31c9
0x309c2	31c0
0x309c4	4839f0
0x309c7	7d17
0x309c9	4839f0
0x309cc	7318
0x309ce	488d1cc7
0x309d2	488b2b
0x309d5	4801e9
0x309d8	48ffc0
0x309db	4839f0
0x309de	7ce9
0x309e0	48894c2420
0x309e5	c3
0x309e6	e8f5a0fdff
0x309eb	0f0b

MOVQ 0x8(SP), DI
MOVQ 0x10(SP), SI
XORL CX, CX
XORL AX, AX
CMPQ SI, AX
JGE 0x309e0
CMPQ SI, AX
JAE 0x309e6
LEAQ 0(DI)(AX*8), BX
MOVQ 0(BX), BP
ADDQ BP, CX
INCQ AX
CMPQ SI, AX
JL 0x309c9
MOVQ CX, 0x20(SP)
RET
CALL runtime.panicindex(SB)
UD2

Bounds checking

But actually.. This is Go 1.4 assembly

```
func Sum(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

Let's look at the generated code today

TEXT	meetup/rng.Sum(SB)	
0x104c390	488b442410	MOVQ 0x10(SP), AX
0x104c395	488b4c2408	MOVQ 0x8(SP), CX
0x104c39a	31d2	XORL DX, DX
0x104c39c	4889d3	MOVQ DX, BX
0x104c39f	eb0e	JMP 0x104c3af
0x104c3a1	488d7201	LEAQ 0x1(DX), SI
0x104c3a5	488b3cd1	MOVQ 0(CX)(DX*8), DI
0x104c3a9	4801fb	ADDQ DI, BX
0x104c3ac	4889f2	MOVQ SI, DX
0x104c3af	4839c2	CMPQ AX, DX
0x104c3b2	7ced	JL 0x104c3a1
0x104c3b4	48895c2420	MOVQ BX, 0x20(SP)
0x104c3b9	c3	RET

Bounds checking completely removed


```
func Sum(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

And it makes sense

```
func Sum1(ia []int) int {  
    sum := 0  
    for i := range ia {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

```
func Sum2(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

WHAT'S THE DIFFERENCE?

```
func Sum1(ia []int) int {  
    sum := 0  
    for i := range ia {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

```
func Sum2(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

```

TEXT meetup/rng.Sum1(SB)
  range.go:13  0x104c390  488b442410  MOVQ 0x10(SP), AX
  range.go:13  0x104c395  488b4c2408  MOVQ 0x8(SP), CX
  range.go:13  0x104c39a  31d2       XORL DX, DX
  range.go:13  0x104c39c  4889d3     MOVQ DX, BX
  range.go:15  0x104c39f  eb0e      JMP 0x104c3af
  range.go:15  0x104c3a1  488d7201  LEAQ 0x1(DX), SI
  range.go:16  0x104c3a5  488b3cd1  MOVQ 0(CX)(DX*8), DI
  range.go:17  0x104c3a9  4801fb    ADDQ DI, BX
  range.go:17  0x104c3ac  4889f2     MOVQ SI, DX
  range.go:15  0x104c3af  4839c2     CMPQ AX, DX
  range.go:15  0x104c3b2  7ced      JL 0x104c3a1
  range.go:19  0x104c3b4  48895c2420 MOVQ BX, 0x20(SP)
  range.go:19  0x104c3b9  c3        RET

```

```

TEXT meetup/rng.Sum2(SB)
  range.go:30  0x104c390  488b442410  MOVQ 0x10(SP), AX
  range.go:30  0x104c395  488b4c2408  MOVQ 0x8(SP), CX
  range.go:30  0x104c39a  31d2       XORL DX, DX
  range.go:30  0x104c39c  4889d3     MOVQ DX, BX
  range.go:32  0x104c39f  eb0e      JMP 0x104c3af
  range.go:32  0x104c3a1  488d7201  LEAQ 0x1(DX), SI
  range.go:33  0x104c3a5  488b3cd1  MOVQ 0(CX)(DX*8), DI
  range.go:34  0x104c3a9  4801fb    ADDQ DI, BX
  range.go:34  0x104c3ac  4889f2     MOVQ SI, DX
  range.go:32  0x104c3af  4839c2     CMPQ AX, DX
  range.go:32  0x104c3b2  7ced      JL 0x104c3a1
  range.go:36  0x104c3b4  48895c2420 MOVQ BX, 0x20(SP)
  range.go:36  0x104c3b9  c3        RET

```

Sum1

```
MOVQ 0x10(SP), AX
MOVQ 0x8(SP), CX
XORL DX, DX
MOVQ DX, BX
JMP 0x104c3af
LEAQ 0x1(DX), SI
MOVQ 0(CX)(DX*8), DI
ADDQ DI, BX
MOVQ SI, DX
CMPQ AX, DX
JL 0x104c3a1
MOVQ BX, 0x20(SP)
```

Sum2

```
MOVQ 0x10(SP), AX
MOVQ 0x8(SP), CX
XORL DX, DX
MOVQ DX, BX
JMP 0x104c3af
LEAQ 0x1(DX), SI
MOVQ 0(CX)(DX*8), DI
ADDQ DI, BX
MOVQ SI, DX
CMPQ AX, DX
JL 0x104c3a1
MOVQ BX, 0x20(SP)
```

```
func Sum1(ia []int) int {  
    sum := 0  
    for i := range ia {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

```
func Sum2(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

Exactly the same code

```
func Sum2(ia []int) int {  
    sum := 0  
    for i := 0; i < len(ia); i++ {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

One last thing...

```
func Sum2(ia []int) int {  
    sum := 0  
    for i := len(ia) - 1; i >= 0; i-- {  
        v := ia[i]  
        sum += v  
    }  
    return sum  
}
```

What about this?

TEXT meetup/rng.SumReverse(SB)

range.go:37	0x104c390	4883ec08	SUBQ \$0x8, SP
range.go:37	0x104c394	48892c24	MOVQ BP, 0(SP)
range.go:37	0x104c398	488d2c24	LEAQ 0(SP), BP
range.go:37	0x104c39c	488b442418	MOVQ 0x18(SP), AX
range.go:39	0x104c3a1	488d48ff	LEAQ -0x1(AX), CX
range.go:39	0x104c3a5	488b542410	MOVQ 0x10(SP), DX
range.go:39	0x104c3aa	31db	XORL BX, BX
range.go:39	0x104c3ac	eb0e	JMP 0x104c3bc
range.go:39	0x104c3ae	488d71ff	LEAQ -0x1(CX), SI
range.go:40	0x104c3b2	488b3cca	MOVQ 0(DX)(CX*8), DI
range.go:41	0x104c3b6	4801fb	ADDQ DI, BX
range.go:41	0x104c3b9	4889f1	MOVQ SI, CX
range.go:39	0x104c3bc	4885c9	TESTQ CX, CX
range.go:39	0x104c3bf	7c07	JL 0x104c3c8
range.go:40	0x104c3c1	4839c1	CMPQ AX, CX
range.go:40	0x104c3c4	72e8	JB 0x104c3ae
range.go:40	0x104c3c6	eb0e	JMP 0x104c3d6
range.go:43	0x104c3c8	48895c2428	MOVQ BX, 0x28(SP)
range.go:43	0x104c3cd	488b2c24	MOVQ 0(SP), BP
range.go:43	0x104c3d1	4883c408	ADDQ \$0x8, SP
range.go:43	0x104c3d5	c3	RET
range.go:40	0x104c3d6	e8a53cfdff	CALL runtime.panicindex(SB)
range.go:40	0x104c3db	0f0b	UD2

TEXT meetup/rng.SumReverse(SB)

range.go:37	0x104c390	4883ec08
range.go:37	0x104c394	48892c24
range.go:37	0x104c398	488d2c24
range.go:37	0x104c39c	488b442418
range.go:39	0x104c3a1	488d48ff
range.go:39	0x104c3a5	488b542410
range.go:39	0x104c3aa	31db
range.go:39	0x104c3ac	eb0e
range.go:39	0x104c3ae	488d71ff
range.go:40	0x104c3b2	488b3cca
range.go:41	0x104c3b6	4801fb
range.go:41	0x104c3b9	4889f1
range.go:39	0x104c3bc	4885c9
range.go:39	0x104c3bf	7c07
range.go:40	0x104c3c1	4839c1
range.go:40	0x104c3c4	72e8
range.go:40	0x104c3c6	eb0e
range.go:43	0x104c3c8	48895c2428
range.go:43	0x104c3cd	488b2c24
range.go:43	0x104c3d1	4883c408
range.go:43	0x104c3d5	c3
range.go:40	0x104c3d6	e8a53cfdff
range.go:40	0x104c3db	0f0b

```
SUBQ $0x8, SP
MOVQ BP, 0(SP)
LEAQ 0(SP), BP
MOVQ 0x18(SP), AX
LEAQ -0x1(AX), CX
MOVQ 0x10(SP), DX
XORL BX, BX
JMP 0x104c3bc
LEAQ -0x1(CX), SI
MOVQ 0(DX)(CX*8), SI
ADDQ DI, BX
MOVQ SI, CX
TESTQ CX, CX
JL 0x104c3c8
CMPQ AX, CX
JB 0x104c3ae
JMP 0x104c3d6
MOVQ BX, 0x28(SP)
MOVQ 0(SP), BP
ADDQ $0x8, SP
RET
```

Bounds checking

CALL runtime.panicindex(SB)

BOTTOM LINE?

BOTTOM LINE?

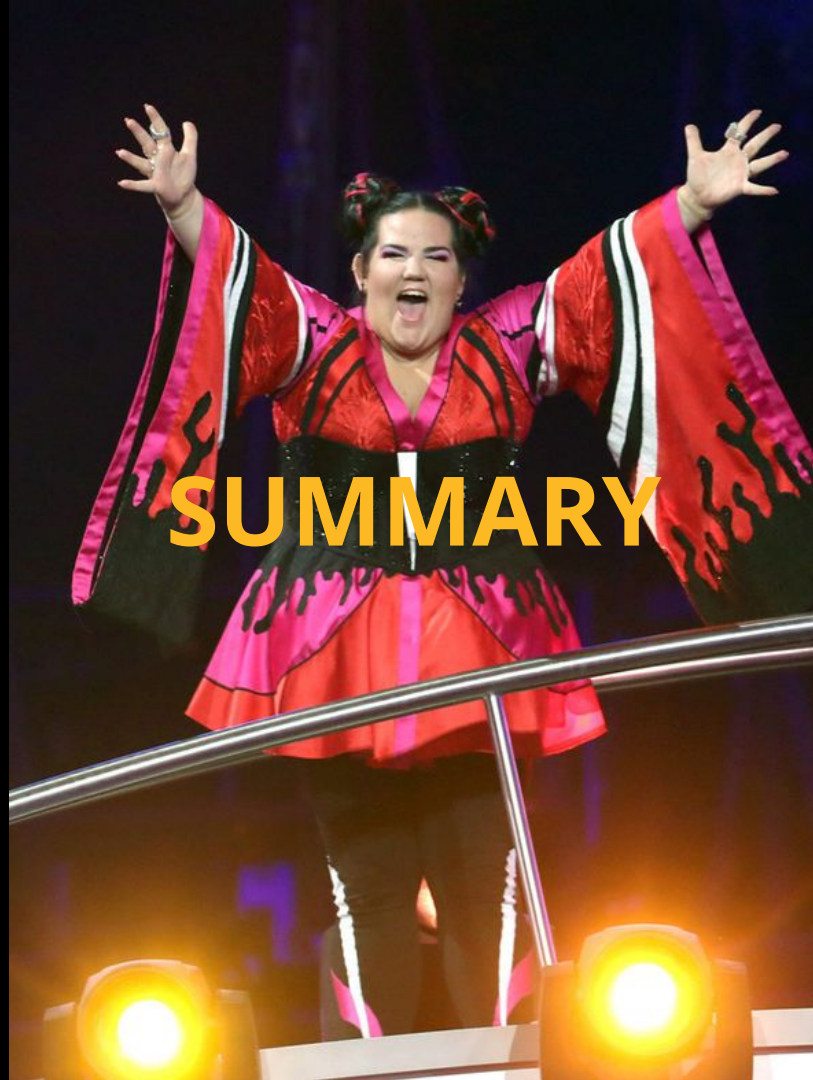
None - I just showed it because I can!

BOTTOM LINE?

The Go compiler is always improving

BOTTOM LINE?

**If your code is simple
it has better chances to be optimized**



DON'T MICRO-OPTIMIZE

**DON'T MICRO-OPTIMIZE
LEAVE THAT TO THE COMPILER**

**DON'T MICRO-OPTIMIZE
LEAVE THAT TO THE COMPILER
PREFER CLEAR CODE**

THE COMPILER ALWAYS IMPROVES

BENCHMARK

VALIDATE YOUR MYTHS

THANK YOU

Icons

↳ Tip: you can change the color of the icons in the toolbar

General



Arrows



More Icons

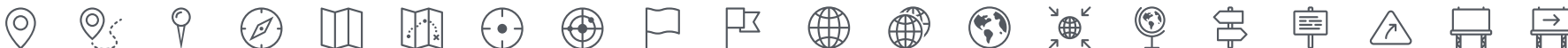
E-Commerce



Web



Location



Weather



More Icons

Electronics



Miscellaneous

