Golang汇编命令解读

2016-12-05 10:12 by 轩脉刃, 3235 阅读, 2 评论, 收藏, 编辑

我们可以很容易将一个golang程序转变成汇编语言。

比如我写了一个main.go:

```
package main


func g(p int) int {
     return p+1;
}

func main() {
     c := g(4) + 1
     _ = c
}
```

使用命令:

```
GOOS=linux GOARCH=386 go tool compile -S main.go >> main.S
```

我们就获取了main.S是main.go的汇编版本。

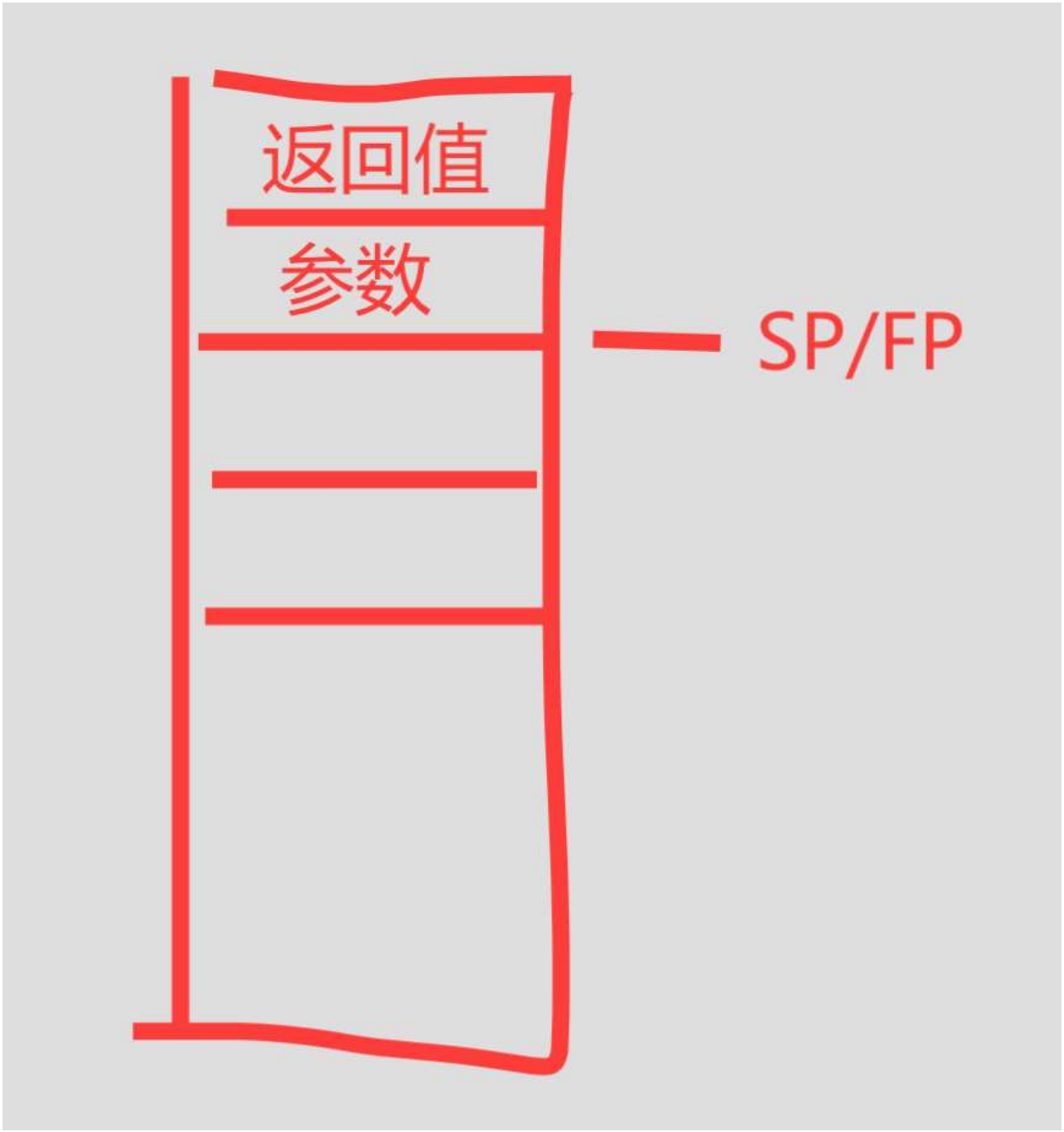
```
"".g t=1 size=16 value=0 args=0x10 locals=0x0
     0x0000 00000 (main.go:4)     TEXT     "".g(SB), $0-16
     0x0000 00000 (main.go:4)     NOP
     0x0000 00000 (main.go:4)     NOP
     0x0000 00000 (main.go:4)     FUNCDATA     $0, gclocals·23e8278e2b69a3a75fa59b23c49ed6ad(SB)
     0x0000 00000 (main.go:4)     FUNCDATA     $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x0000 00000 (main.go:5)     MOVQ     "".p+8(FP), BX
     0x0005 00005 (main.go:5)     INCQ     BX
     0x0008 00008 (main.go:5)     MOVQ     BX, "".~r1+16(FP)
     0x000d 00013 (main.go:5)     RET
     0x0000 48 8b 5c 24 08 48 ff c3 48 89 5c 24 10 c3        H.\$.H..H.\$..
"".main t=1 size=16 value=0 args=0x0 locals=0x0
     0x0000 00000 (main.go:8)     TEXT     "".main(SB), $0-0
     0x0000 00000 (main.go:8)     NOP
     0x0000 00000 (main.go:8)     NOP
     0x0000 00000 (main.go:8)     FUNCDATA     $0, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x0000 00000 (main.go:8)     FUNCDATA     $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x0000 00000 (main.go:9)     MOVQ     $4, BX
     0x0007 00007 (main.go:9)     INCQ     BX
     0x000a 00010 (main.go:9)     INCQ     BX
     0x000d 00013 (main.go:11)     RET
     0x0000 48 c7 c3 04 00 00 00 48 ff c3 48 ff c3 c3        H......H..H...
"".init t=1 size=80 value=0 args=0x0 locals=0x0
     0x0000 00000 (main.go:11)     TEXT     "".init(SB), $0-0
     0x0000 00000 (main.go:11)     MOVQ     (TLS), CX
     0x0009 00009 (main.go:11)     CMPQ     SP, 16(CX)
     0x000d 00013 (main.go:11)     JLS     62
     0x000f 00015 (main.go:11)     NOP
     0x000f 00015 (main.go:11)     NOP
     0x000f 00015 (main.go:11)     FUNCDATA     $0, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x000f 00015 (main.go:11)     FUNCDATA     $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x000f 00015 (main.go:11)     MOVBQZX     "".initdone·(SB), BX
     0x0016 00022 (main.go:11)     CMPB     BL, $0
     0x0019 00025 (main.go:11)     JEQ     47
     0x001b 00027 (main.go:11)     MOVBQZX     "".initdone·(SB), BX
     0x0022 00034 (main.go:11)     CMPB     BL, $2
     0x0025 00037 (main.go:11)     JNE     40
     0x0027 00039 (main.go:11)     RET
     0x0028 00040 (main.go:11)     PCDATA     $0, $0
     0x0028 00040 (main.go:11)     CALL     runtime.throwinit(SB)
     0x002d 00045 (main.go:11)     UNDEF
     0x002f 00047 (main.go:11)     MOVB     $1, "".initdone·(SB)
     0x0036 00054 (main.go:11)     MOVB     $2, "".initdone·(SB)
     0x003d 00061 (main.go:11)     RET
     0x003e 00062 (main.go:11)     CALL     runtime.morestack_noctxt(SB)
     0x0043 00067 (main.go:11)     JMP     0
```

首先这个程序根据TEXT是定义函数的，分为3个部分

```
0x0000 00000 (main.go:4)     TEXT     "".g(SB), $0-16
0x0000 00000 (main.go:8)     TEXT     "".main(SB), $0-0
0x0000 00000 (main.go:11)     TEXT     "".init(SB), $0-0
```

这个"". 代表的是这个函数的命名空间。

g(SB) 这里就有个SB的伪寄存器。全名未Static Base，代表g这个函数地址，0 – 16中的0代表局部变量字节数总和，0表示不存在局部变量。-16代表在0的地址基础上空出16的长度作为传入和返回对象。这个也就是golang如何实现函数的多返回值的方法了。它在定义函数的时候，开辟了一定空间存储传入和传出对象。



NOP命令是作为占位符使用，提供给编译器使用的。可以忽略不看。

下面是

```
    0x0000 00000 (main.go:4)    FUNCDATA    $0, gclocals·23e8278e2b69a3a75fa59b23c49ed6ad(SB)
    0x0000 00000 (main.go:4)    FUNCDATA    $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
```

这里的FUNCDATA是golang编译器自带的指令，plan9和x86的指令集都是没有的。它用来给gc收集进行提示。提示0和1是用于局部函数调用参数，需要进行回收。

下面是

```
    0x0000 00000 (main.go:5)    MOVQ    "".p+8(FP), BX
    0x0005 00005 (main.go:5)    INCQ    BX
    0x0008 00008 (main.go:5)    MOVQ    BX, "".~r1+16(FP)
```

这里有一个FP寄存器，FP是frame pointer，是指向栈底，SP是指向栈顶。BX是一个临时寄存器，那么上面的句子是代表把FP + 8这个位置的数据（参数p），保存到BX中。FP+8代表什么呢，按照上面的图，代表的是参数

INCQ是自增算法，BX里面的数自加1，然后把BX里面的数存储到FP+16，代表的是返回值。
下面就是RET，直接返回。

下面再看看main，我们会发现，main函数里面并没有call g函数，这是由于go汇编编译器会把一些短的函数变成内嵌函数，减少函数调用。

if

我们在main里面增加一个判断逻辑。代码为：

```
package main


func g(p int) int {
     return p+1;
}

func main() {
     c := g(4) + 1
     var d bool
     if (c > 4) {
          d = true
     } else {
          d = false
     }
     _ = d
     return
}
```

对应的main的汇编为：

```
     0x0000 00000 (main.go:8)     TEXT     "".main(SB), $0-0
     0x0000 00000 (main.go:8)     NOP
     0x0000 00000 (main.go:8)     NOP
     0x0000 00000 (main.go:8)     FUNCDATA     $0, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x0000 00000 (main.go:8)     FUNCDATA     $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
     0x0000 00000 (main.go:9)     MOVQ     $4, BX
     0x0007 00007 (main.go:9)     INCQ     BX
     0x000a 00010 (main.go:9)     INCQ     BX
     0x000d 00013 (main.go:11)     CMPQ     BX, $4
     0x0011 00017 (main.go:11)     JLE     27
     0x0013 00019 (main.go:12)     MOVQ     $1, AX
     0x001a 00026 (main.go:17)     RET
     0x001b 00027 (main.go:14)     MOVQ     $0, AX
     0x001d 00029 (main.go:17)     JMP     26
```

可以看是试用CMPQ来进行比较的，JLE代表CMP比较之后的结果，如果BX小于等于4，那么就跳到27指令，就是MOVQ $0, AX，把AX赋值为0，就是false，否则赋值为1，true

这里的JLE是条件转移指令：http://baike.baidu.com/link?url=pdfEpBZ-c-0wW0YJrAL71MUi1nIlEpBqljD3agfLtB5KrAaFembK6yKKUNycaSUWTTTyynKhNIfw2LXvGFx4euTIgyXQcVLz5HPxS4AO-kVT9wQxlL6_O-1ygTwfshEgas91S14FU3CRU7a1jImTSK

for

程序改为：

```
package main


func g(p int) int {
     var sum int
     for i := 0; i < p; i++ {
          sum = sum + i
     }
     return sum
}

func main() {
     c := g(4) + 1
     _ = c
}
```

这里面有一个for循环，产生的汇编为：

```
    0x0000 00000 (main.go:4)    TEXT    "".g(SB), $0-16
    0x0000 00000 (main.go:4)    NOP
    0x0000 00000 (main.go:4)    NOP
    0x0000 00000 (main.go:4)    MOVQ    "".p+8(FP), DX
    0x0005 00005 (main.go:4)    FUNCDATA    $0, gclocals·23e8278e2b69a3a75fa59b23c49ed6ad(SB)
    0x0005 00005 (main.go:4)    FUNCDATA    $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)
    0x0005 00005 (main.go:5)    MOVQ    $0, CX
    0x0007 00007 (main.go:6)    MOVQ    $0, AX
    0x0009 00009 (main.go:6)    CMPQ    AX, DX
    0x000c 00012 (main.go:6)    JGE     $0, 25
    0x000e 00014 (main.go:7)    ADDQ    AX, CX
    0x0011 00017 (main.go:6)    INCQ    AX
    0x0014 00020 (main.go:6)    NOP
    0x0014 00020 (main.go:6)    CMPQ    AX, DX
    0x0017 00023 (main.go:6)    JLT     $0, 14
    0x0019 00025 (main.go:9)    MOVQ    CX, "".~r1+16(FP)
    0x001e 00030 (main.go:9)    RET
```

AX存的是变量i，DX存的是参数p，CX存的是变量sum，下面的几个命令：

```
0x0009 00009 (main.go:6)    CMPQ    AX, DX
    0x000c 00012 (main.go:6)    JGE     $0, 25
    0x000e 00014 (main.go:7)    ADDQ    AX, CX
    0x0011 00017 (main.go:6)    INCQ    AX
    0x0014 00020 (main.go:6)    NOP
    0x0014 00020 (main.go:6)    CMPQ    AX, DX
    0x0017 00023 (main.go:6)    JLT     $0, 14
```

实际上是使用CMP，JGE，JLT来不断控制循环过程。