plan9

教程1
教程2

伪寄存器

```
FP: Frame pointer: arguments and locals.
PC: Program counter: jumps and branches.
SB: Static base pointer: global symbols.
SP: Stack pointer: top of stack.
```

x64寄存器

X64	rax	rbx	rcx	rdx	rdi	rsi	rbp	rsp	r8	r9	r10
Plan9	AX	BX	CX	DX	DI	SI	BP	SP	R8	R9	R10

后缀

```
B: 1 byte
W: 2 bytes
D: 4 bytes
Q: 8 bytes
```

栈扩大、缩小

```
SUBQ $0x18, SP //push,为函数分配函数栈帧
ADDQ $0x18, SP //pop,清除函数栈帧
```

数据copy

```
MOVB $1, DI // 1 byte
MOVW $0x10, BX // 2bytes
MOVD $1, DX // 4 bytes
MOVQ $-10, AX // 8 bytes
```

计算

```
ADDQ AX, BX // BX += AX
SUBQ AX, BX // BX -= AX
IMULQ AX, BX // BX *= AX
```

比较

```
CMPQ $2,$1
```

跳转

```
//无条件跳转
JMP addr // 跳转到地址，地址可为代码中的地址 不过实际上手写不会出现这种东西
JMP label // 跳转到标签 可以跳转到同一函数内的标签位置
JMP 2(PC) // 以当前置顶为基础，向前/后跳转x行
JMP -2(PC) //同上
//有条件跳转
CMPQ $2,$1
JG label
```

指令	描述
JMP	直接跳转
JE/JZ	相等/零
JNE/JNZ	不相等/非零
JS	负数
JNS	非负数
JG/JNLE	大于（有符号）
JGE/JNL	大于或等于（有符号）
JL/JNGE	小于（有符号）
JLE/JNG	小于或等于（有符号）
JA/JNBE	超过（无符号）
JAE/JNB	超过或相等（无符号）
JB/JNAE	低于（无符号）
JBE/JNA	低于或等于（无符号）

标志寄存器		
CF	进位标志	最近的操作使最高位产生了进位。用于检查无符号溢出的操作
ZF	零标志	最近的操作得出的结果为0
SF	符号标志	最近的操作得出的结果为负数
OF	溢出标志	最近的操作导致了一个补码溢出（正溢出 / 负溢出）

声明变量

用法：

```
//GLOBL在DATA之后
DATA symbol+offset(SB)/width,value
GLOBL symbol, (NOPTR+RODATA), $4
```

GOLBL选项：

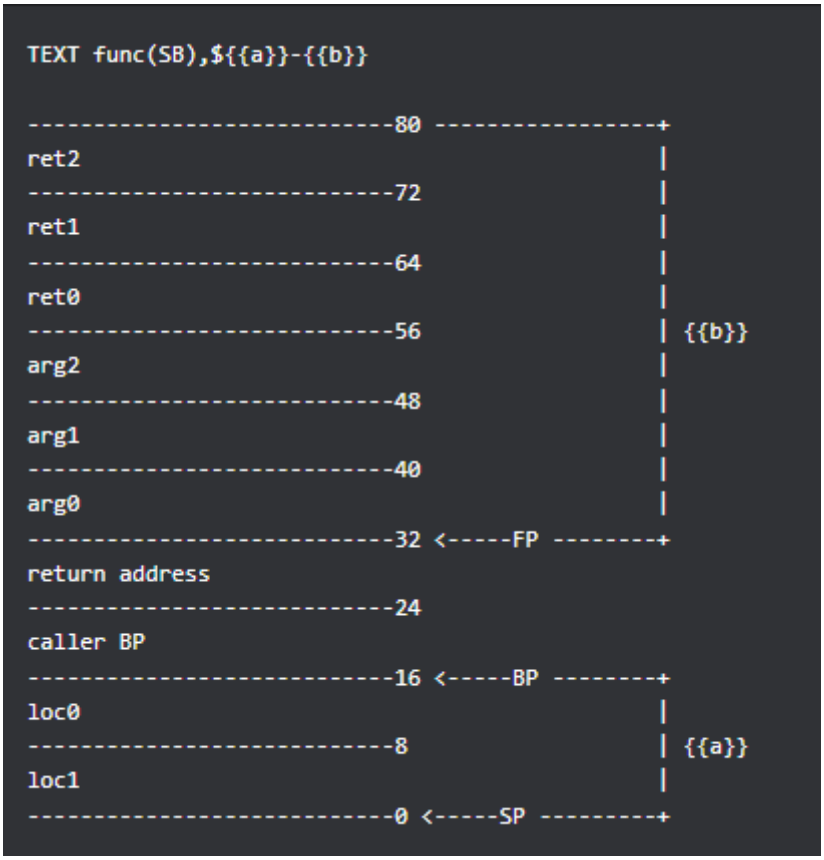
```
NOPTR: 没有指针，不被GC扫描
RODATA: 只读数据
```

示例：

```
//定义
DATA ·Id+0(SB),$100
GLOBL ·Id(SB),NOPTR,$8

//         只在当前文件文件中生效
//         |
DATA ·Name<>+0(SB)/8,$·Name+16(SB)
DATA ·Name<>+8(SB)/8,$6
DATA ·Name<>+16(SB)/8,$"gopher"
GLOBL ·Name<>(SB),RODATA,$24

//使用
MOVQ ·Id(SB),AX
```

函数

```
//导入函数的选项
#include "textflag.h"

//告诉汇编器该数据放到TEXT区
// |
// |  告诉汇编器这是基于静态地址的数据(static base)
// |         |
// |         |  栈帧大小(caller BP+局部变量+调用其他函数参数及返回值,但不包括调用其它函数时的ret address的大小)
// |         |            |
// |         |            |  参数及返回值大小
// |         |            |  |
// |         |            |  |
TEXT ·Swap(SB),NOSPLIT,$0-32
//      参数a
//        |
    MOVQ a+0(FP), AX  // FP(Frame pointer)栈帧指针 这里指向栈帧最低位
    MOVQ b+8(FP), BX
//         无名返回值0
//               |
    MOVQ BX ,ret0+16(FP)
    MOVQ AX ,ret1+24(FP)
    RET
```



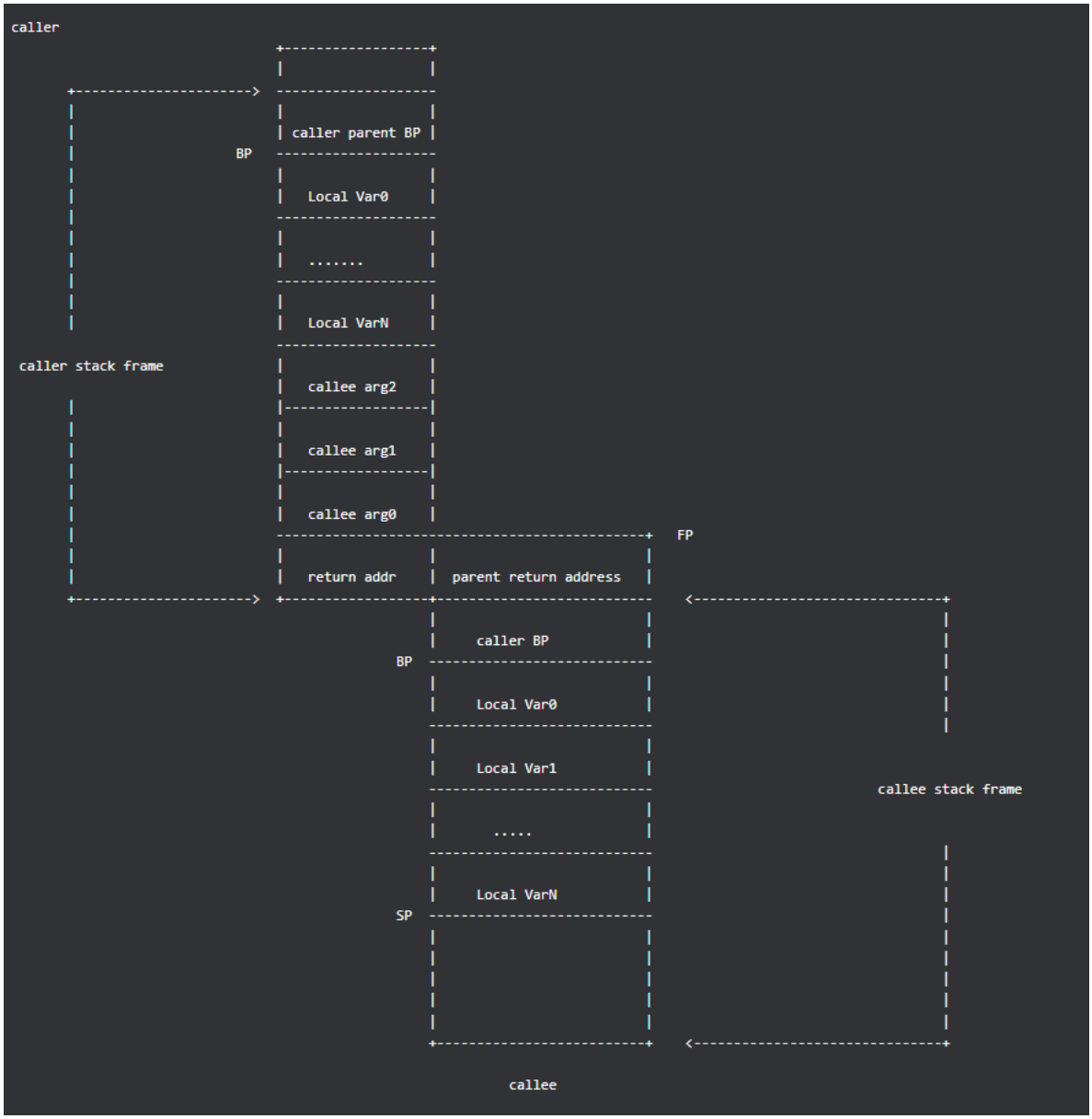
函数参数摆放方向

```
FP
high ----------------------> low
argN, ... arg3, arg2, arg1, arg0
```

栈操作

```
TEXT func(SB),${{a}}-{{b}}
    //分配栈
    MOVQ SP,BP
    SUBQ ${{a}},SP

    //释放栈
    ADDQ ${{a}},SP
```

栈结构



地址运算

```
LEAQ (BX)(AX*8), CX
// 上面代码中的 8 代表 scale
// scale 只能是 0、2、4、8
// 用 LEAQ 的话，即使是两个寄存器值直接相加，也必须提供 scale

// 在寄存器运算的基础上，可以加上额外的 offset
LEAQ 16(BX)(AX*1), CX
```

模拟32位

```
MODE $32
```

放入指令流

```
LONG $123456
WORD $123456
```

其他

```
//自增
INCQ AX
//自减
DECQ AX
```

示例

斐波那契数列

```
TEXT ·Fab(SB),$0-16
    MOVQ n+0(FP), AX
    CMPQ AX, $1
    JBE end
    SUBQ $16, SP
    MOVQ AX, 0(SP)
    DECQ 0(SP)
    CALL ·Fab(SB)
    MOVQ 8(SP), AX
    MOVQ AX, 40(SP)
    DECQ 0(SP)
    CALL ·Fab(SB)
    MOVQ 8(SP), AX
    ADDQ AX, 40(SP)
    ADDQ $16, SP
    RET
end:
    MOVQ $1, ret+8(FP)
    RET
```