X86

1 汇编语言入门教程 - 知乎 (zhihu.com)

1. 首先 x86 中对地址的操作，比如 esp+ox12，指的是栈移动 16+2=18 个字节，所以地址均为字节（一个 int 为四个字节）

   比如： `sub` `0x18`，`%rsp` 表示开辟 24 字节的内存空间

2. rsp 只是指针，指向栈顶（栈的生长方向是从高地址向低地址， rsp 指向最后一个值，也就是地址最小处）

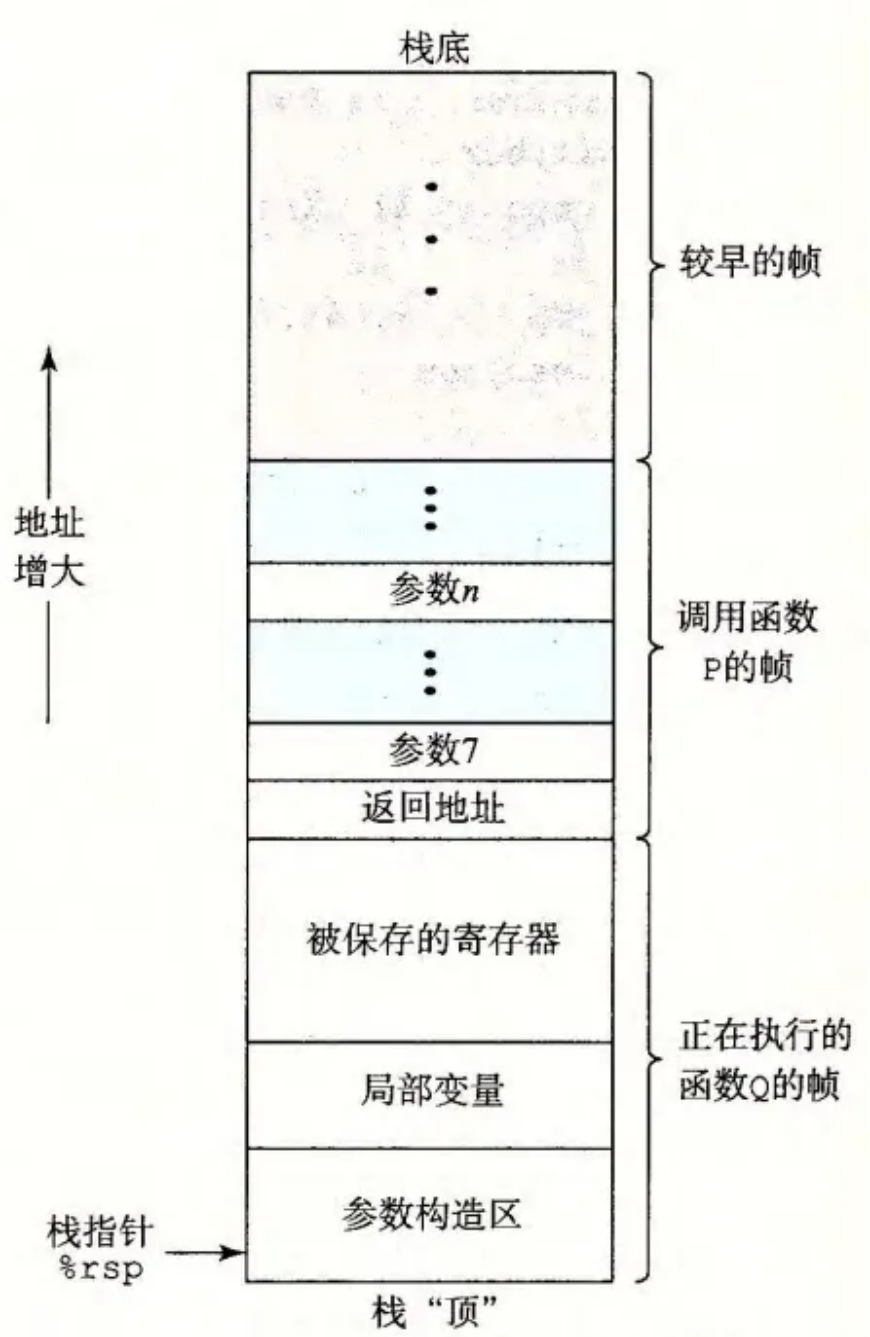

图 3-25 通用的栈帧结构（栈用来传递参数、存储返回信息、保存寄存器，以及局部存储。省略了不必要的部分）

如果我在一个函数 a 处设置了断点，触发断点后，rsp 指向 ret，

3. 函数的参数传递不看顺序，只看寄存器名称：
   func：edi esi 两个寄存器
   则在输入时无需考虑哪一个寄存器先赋值，对应的功能在函数中不会改变。
4. edi esi 一般作为函数传参所用的寄存器，若只有一个参数，一般使用 edi。
5. r 打头的寄存器包含了 e 打头的寄存器；e 打头的寄存器包含了对应的原始寄存器。
   比如： rdi 寄存器是 edi 寄存器在 64位模式下的扩展，它包含了 edi。
   所以若一个数据被存入了 rax，那么在指令中使用 eax 等同于使用 rax 访问数据。
6. sscanf 会依次将读取到的数值压入栈中，比如"%d %d"的输入形式，输入"1 2"，此时栈中为：
   1
   2 <- rsp
7. **`x86` 采用小端存储，要注意输入字节的顺序（倒着存放字节，注意是以字节为单位）**
8. 0x4(%rsp) 指的是栈指针位置向上走一个 int 值的大小，这意味着此时获取到的元素为倒数第二个值
9. jbe (if 【操作数 2】 below or equal【操作数 1】)：com a b，当 a>=b 时跳转.
10. test　%edi,%edi ： 检测 edi 中的值是否为 0
11. 汇编代码中，一个 0 表示 4 个 bit，也就是 0.5 字节，两个 0 为一个字节长度。
    比如下面这个就是 8 个字节（16 个 0）：

```Bash
00 00 00 00 00 00 00 00
```

1. SF，
   指令执行后，其结果是否为负。若结果为负，SF=1；如果非负，SF=0。
   两个数相加，结果转换成二进制，看最高位，若为1，SF=1，若为0，SF=0。
2. *ZF*
   指令执行后，其结果是否为0，若结果为0，那么ZF=1；如果不为0，那么ZF=0。
3. PF
   指令执行后，其结果的二进制表示中1的个数是否为偶数，若1的个数为偶数，PF=1；若1的个数为奇数，PF=0。

```Shell
如果SF = 0，则跳转，反之不跳转
jns address
```

gdb

Shell

```shell
# 进入函数内部
disas function_name

# 运行文件（断点处会停止）
r

# 继续运行
c

# 在第n行设置断点
b n

# 在函数处设置断点
b function_name

# 获取断点信息
info b

# 删除断点
delete 断点号n
```

Plain Text

```
GDB 命令摘要，适用于 IA32 系统

命令                    效果

启动:
  gdb
  gdb <文件>

运行和停止

  quit                      退出 gdb
  run                       运行程序
  run 1 2 3                 使用命令行参数 1 2 3 运行程序
  kill                      停止程序
  quit                      退出 gdb
  Ctrl-d                    退出 gdb
    注意：Ctrl-C 不会退出 gdb，但会中断当前
    gdb 命令

断点

  break sum                 在函数 sum 的入口处设置断点
  break *0x40046b            在地址 0x40046b 处设置断点
  disable 1                 禁用断点 1
                  (gdb 会为每个你创建的断点编号)
  enable 1                  启用断点 1
  delete 1                  删除断点 1
  delete                    删除所有断点
  clear sum                 清除函数 sum 入口处的任何断点

执行

  stepi                     执行一条指令
  stepi 4                   执行四条指令
  nexti                     类似于 stepi，但继续
                  通过函数调用而不停止
  step                      执行一条 C 语句
  continue                  恢复执行直到下一个断点
  until 3                   继续执行直到程序触发断点 3
  finish                    恢复执行直到当前函数返回
  call sum(1, 2)            调用 sum(1,2) 并打印返回值

查看代码

  disas                     反汇编当前函数
  disas sum                  反汇编函数 sum
  disas 0x8048335            反汇编 0x8048335 附近的函数
  disas 0x8048335 0x8048343 反汇编指定地址范围内的代码

  print /x $eip             以十六进制打印程序计数器
  print /d $eip             以十进制打印程序计数器
  print /t $eip             以二进制打印程序计数器

查看数据
```

```
    print /d $eax             以十进制打印 %eax 的内容
    print /x $eax             以十六进制打印 %eax 的内容
    print /t $eax             以二进制打印 %eax 的内容
    print 0x100               打印 0x100 的十进制表示
    print /x 555              打印 555 的十六进制表示
    print /x ($esp+8)         打印 (%esp 的内容) + 8 的十六进制
    print *(int *) 0xffffcca8  打印地址 0xffffcca8 处的整数
    print *(int *) ($esp+8)   打印地址 %esp + 8 处的整数
    print (char *) 0xbfff890  查看存储在 0xffffcca8 处的字符串

    x/w   0xffffcca8          查看从地址 0xffffcca8 开始的 (4 字节) 字
                    x/w   $esp                查看从 $esp 地址开始的 (4 字节) 字
    x/wd  $esp                查看从 $esp 地址开始的 (4 字节) 字。
                    以十进制打印
    x/2w  $esp                查看从 $esp 地址开始的两个 (4 字节) 字
    x/2wd $esp                查看从 $esp 地址开始的两个 (4 字节) 字。
                    以十进制打印
    x/g   $esp                查看从 $esp 地址开始的 (8 字节) 字
    x/gd  $esp                查看从 $esp 地址开始的 (8 字节) 字。
                    以十进制打印
    x/a   $esp                查看 $esp 地址。以相对于
                    上一个全局符号的偏移量打印。
    x/s   0xffffcca8          查看存储在 0xffffcca8 处的字符串
    x/20b sum                 查看函数 sum 的前 20 个字节的操作码
    x/10i sum                 查看函数 sum 的前 10 条指令

    (注意: `x' 命令的格式字符串的一般形式为
      x/[NUM][SIZE][FORMAT] 其中

      NUM  = 要显示的对象数量
      SIZE = 每个对象的大小 (b=字节, h=半字, w=字,
                          g=巨人 (四字) )
      FORMAT = 每个对象的显示方式 (d=十进制, x=十六进制, o=八进制, 等)

      如果你没有指定 SIZE 或 FORMAT, 将使用默认值, 或者你
      在之前的 `print' 或 `x' 命令中指定的最后一个值。)

  有用的信息

    backtrace               打印当前地址和堆栈回溯
    where                   打印当前地址和堆栈回溯

    info program            打印程序的当前状态
    info functions          打印程序中的函数
    info stack              打印堆栈的回溯
    info frame              打印当前堆栈帧的信息
    info registers          打印寄存器及其内容
    info breakpoints        打印用户可设置断点的状态

    display /FMT EXPR       使用格式 FMT 打印表达式 EXPR
                  每次 GDB 停止时
    undisplay               关闭显示模式
    help                    获取关于 gdb 的信息
```