1．用堆栈传递参数有 3 种方式：

(1) __cdecl

参数从右到左顺序压入堆栈，由调用者清理堆栈；

是 C 语言参数传递规范。

__cdecl 的例子：

```
f:
push bp; (4)
mov bp, sp
mov ax, [bp+4]; arg0
add ax, [bp+6]; arg1
pop bp; (5)
ret; (6)
main:
mov ax, 20; arg1
push ax; (1)
mov ax, 10; arg0
push ax; (2)
call f; (3)
here:
add sp, 4; (7)
```

__cdecl 堆栈布局：

```
ss:1FF8 old bp ← bp (4)
ss:1FFA here ← (3)(5)
ss:1FFC 10 ← (2)(6)
ss:1FFE 20 ← (1)
ss:2000    ← (7)
```

(2) __pascal

参数从左到右顺序压入堆栈，由被调用者清理堆栈；

是 Pascal 语言参数传递规范。

__pascal 的例子：

```
f:
push bp; (4)
mov bp, sp
mov ax, [bp+6]; arg0
add ax, [bp+4]; arg1
pop bp; (5)
ret 4; (6)
main:
mov ax, 10
```

```
push ax; (1) arg0
mov ax, 20
push ax; (2) arg1
call f; (3)
here:
```

__pascal 的堆栈布局：

```
ss:1FF8 old bp<- bp (4)
ss:1FFA here <- (3)(5)
ss:1FFC 20 <- (2)
ss:1FFE 10 <- (1)
ss:2000   <-(6)
```

(3) __stdcall

参数从右到左顺序压入堆栈，由被调用者清理堆栈；是 Windows API 函数的参数传递规范。

__stdcall 的例子：

```
f:
push bp; (4)
mov bp, sp
mov ax, [bp+4]; arg0
add ax, [bp+6]; arg1
pop bp; (5)
ret 4; (6)
main:
mov ax, 20
push ax; (1) arg1
mov ax, 10
push ax; (2) arg0
call f; (3)
here:
```

__stdcall 的堆栈布局：

```
ss:1FF8 old bp<- bp (4)
ss:1FFA here <- (3)(5)
ss:1FFC 10 <- (2)
ss:1FFE 20 <- (1)
ss:2000   <-(6)
```

32 位__stdcall 的例子：

```
f:
push ebp;(4)
```

```
mov ebp, esp
mov eax,[ebp+8];arg0
add eax,[ebp+0Ch];arg1
pop ebp;(5)
ret 8;(6)
main:
push 20;(1) arg1
push 10;(2) arg0
call f;(3)
here:
```

32 位__stdcall 的堆栈布局：

```
ss:1FF0 old ebp<- ebp (4)
ss:1FF4 here <- (3)(5)
ss:1FF8 10 <- (2)
ss:1FFC 20 <- (1)
ss:2000   <-(6)
```

2．动态变量

```
int f(int a, int b)
{
   int c; /* c 是局部动态变量 */
   c = a+b;
   return c;
}
```

上述 C 语言函数可翻译成以下汇编代码：

```
f:
push bp; (4)
mov bp, sp
sub sp, 2; (5) 这里挖的坑就是给变量 c 的
mov ax, [bp+4]
add ax, [bp+6]
mov [bp-2], ax
mov ax, [bp-2]
mov sp, bp; (6)此时变量 c 死亡
pop bp; (7)
ret; (8)
main:
mov ax, 20
push ax; (1)
mov ax, 10
push ax; (2)
call f; (3)
```

```
here:
add sp, 4;(9)此时参数 a,b 死亡
```
执行上述代码时，堆栈布局如下：
```
ss:1FF6 [30] (5) 变量 c
ss:1FF8 old bp<- bp(4)(6)
ss:1FFA here <- (3)(7)
ss:1FFC 10 <- (2)(8)
ss:1FFE 20 <- (1)
ss:2000   <-(9)
```

3．C 语言函数中需要保护 bp，bx，si，di

C 语言的函数里面除了不能破坏 bp 外，还要保护 bx,si,di 的值：

```
f:
push bp
mov bp, sp
sub sp, n; 其中 n 一个常数,用来为动态变量分配空间
push bx
push si
push di
...
pop di
pop si
pop bx
mov sp, bp
pop bp
ret
```

4．递归

```
int f(int n)
{
   if(n==1)
      return 1;
   return n+f(n-1);
}
```
上述 C 语言递归函数可翻译成以下汇编代码：
```
f:
```

```
push bp;(3)(6)(9)
mov bp, sp
mov ax, [bp+4]
cmp ax, 1
je done
dec ax
push ax;(4)(7)
call f;(5)(8)
there:
add sp, 2;(12)(15)
add ax, [bp+4]
done:
pop bp;(10)(13)(16)
ret;(11)(14)(17)
main:
mov ax, 3
push ax;(1)
call f;(2)
here:
add sp, 2;(18)
```

执行上述代码时的堆栈布局如下：

```
ss:1FEE oldbp<-bp(9)
ss:1FF0 there<-(8)(10)
ss:1FF2 1<-(7)(11)
ss:1FF4 oldbp<-bp(6)12
ss:1FF6 there<-(5)(13)
ss:1FF8 2<-(4)(14)
ss:1FFA oldbp<-bp(3)(15)
ss:1FFC here <-(2)(16)
ss:1FFE 3 <-(1)(17)
ss:2000  <-(18)
```

5．混合语言编程

http://www.cc98.org/topic/4591481

6．int、iret

int 21h 对应的函数首地址保存在 0:84h 处，该地址是一个远指针。

0:84 78h

0:85 56h

0:86 34h

0:87 12h

dword ptr 0:[84h]称为 int 21h 的中断向量(其实是它的函数首地址)

int n 对应的中断向量的存储位置一定是 0:n*4

n 的取值范围是[00, FF], 所以 256 个中断向量会占用 0:0~0:3FF 之间共 400h 个字节的内存，这块区域称为中断向量表。

显然, int 00h 的中断向量保存在 dword ptr 0:[0];

int 01h 的中断向量保存在 dword ptr 0:[4];

int 03h 的中断向量保存在 dword ptr 0:[0Ch];

int 08h 的中断向量保存在 dword ptr 0:[20h];

int 0FFh 的中断向量保存在 dword ptr 0:[3FCh];

BIOS 会完成部分中断向量的填写，如 int10h、int16h、int 13h 这几个 BIOS 中断的向量在 DOS 启动前就已经填好了; DOS 启动完成后，会填入 int 21h 的中断向量。

```
1000:2000 mov ah, 2
1000:2002 mov dl, 'A'
1000:2004 int 21h
1000:2006 mov ah,4Ch
```

上面这条 int 21h 指令执行时, cpu 做了以下 4 件事:

pushf

push cs 即 1000h

push 下条指令的偏移地址即 2006h

jmp dword ptr 0:[84h]; jmp 1234h:5678h

上面这条 jmp 会跳转到 int 21h 的中断服务函数内部：

```
1234:5678  ...
           ...
1234:56FF  iret; 中断返回
```

iret 在执行时，cpu 做了以下 3 件事情：

```
pop ip 即 ip=2006h
pop cs 即 cs=1000h
popf
```

```
1000:2000 call 1234:5678; 远调用
;此时会 push 1000h 再 push 2005h
;最后 jmp 1234:5678
1000:2005 mov ah, 4Ch
1000:2007 int 21h
...
1234:5678 ...
          retf; 此时执行 pop ip 再 pop cs
```

中断程序例子：

http://10.71.45.100/bhh/int80.asm

http://10.71.45.100/bhh/int00.asm

http://10.71.45.100/bhh/int8.asm

7．缓冲溢出

http://10.71.45.100/bhh/overflow.rar

8．期末考试重点

https://www.cc98.org/topic/4537011