

本节内容

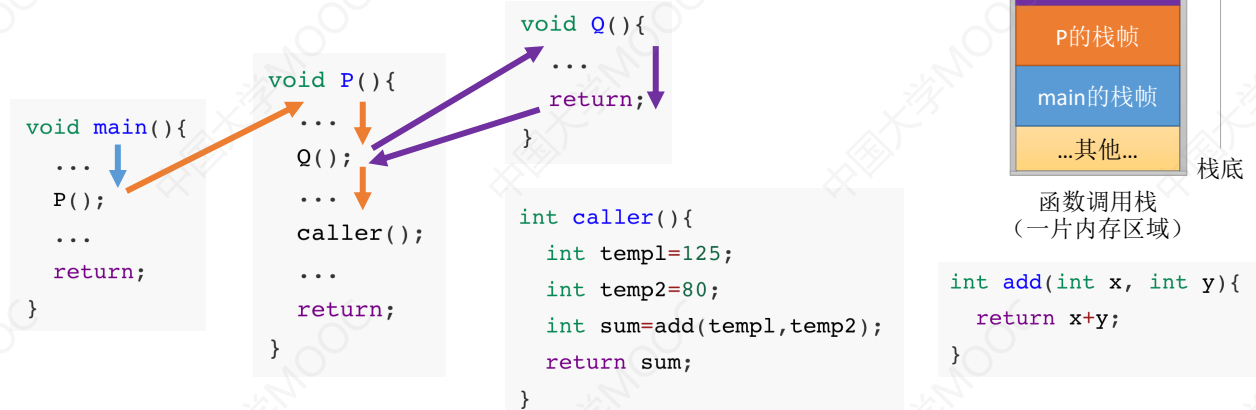
函数调用 机器级表示

—— call、ret 指令

王道考研/CSKAOYAN.COM

1

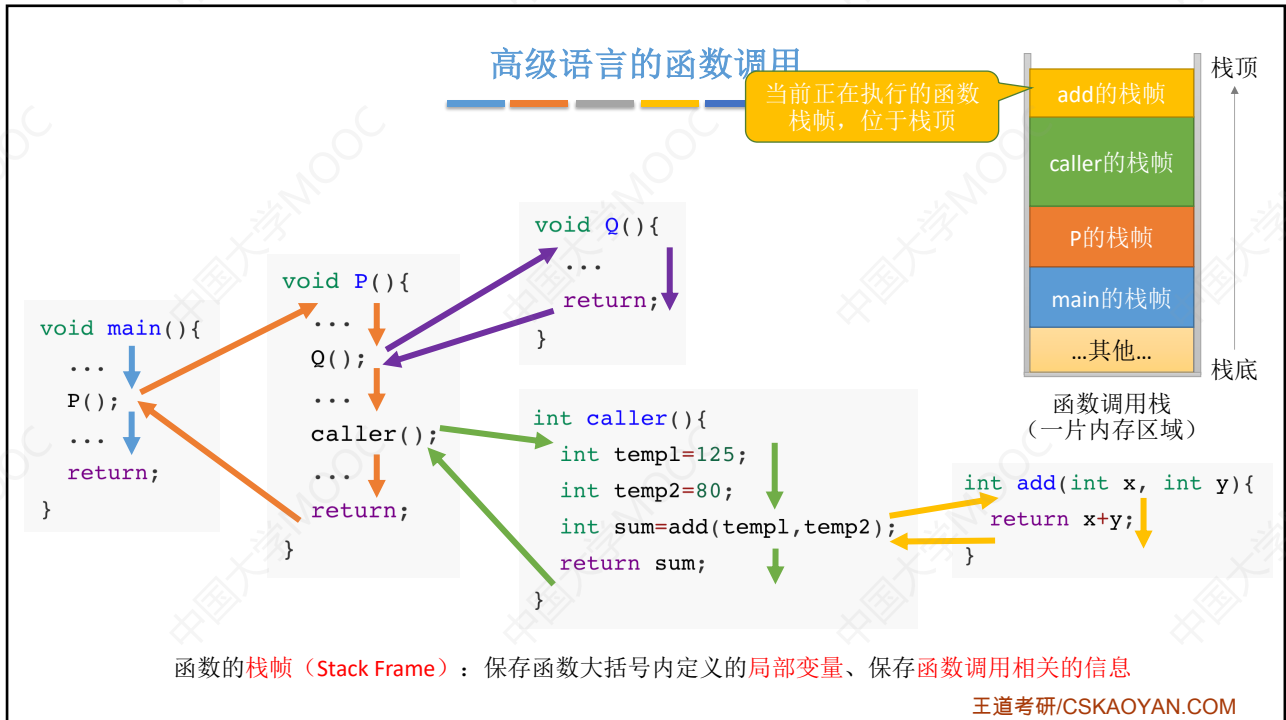
高级语言的函数调用



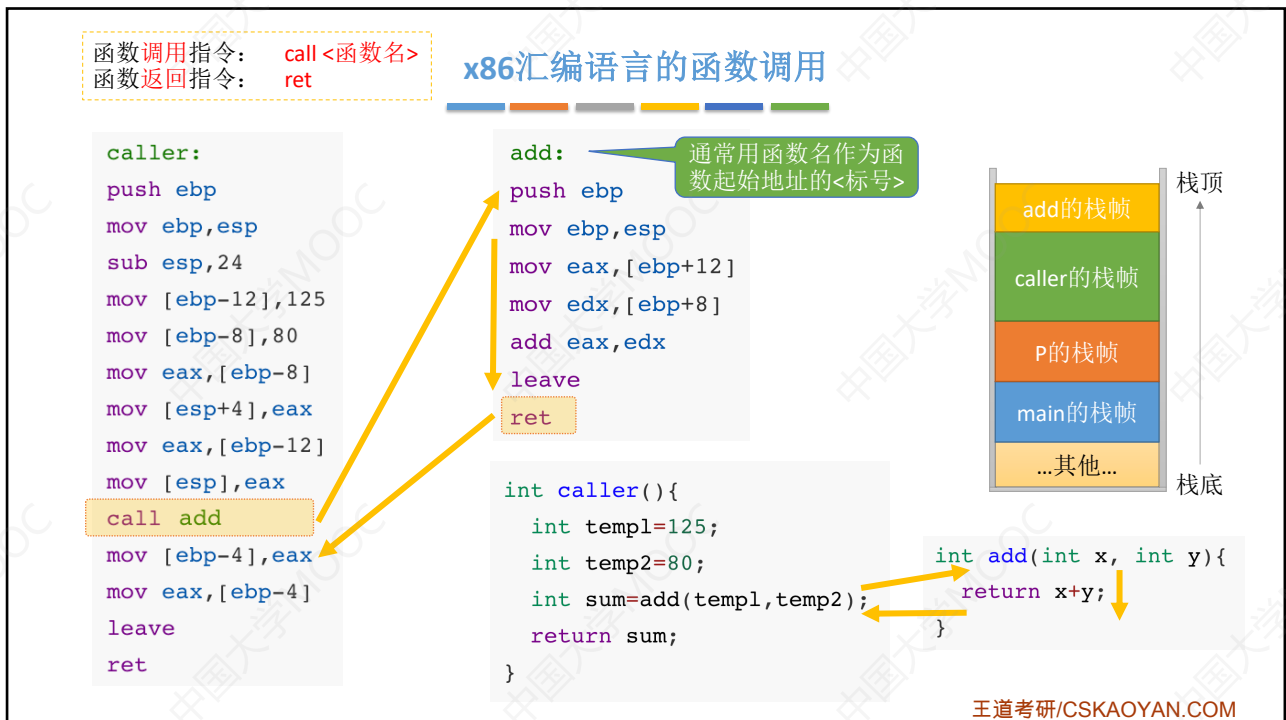
函数的栈帧 (Stack Frame)：保存函数大括号内定义的局部变量、保存函数调用相关的信息

王道考研/CSKAOYAN.COM

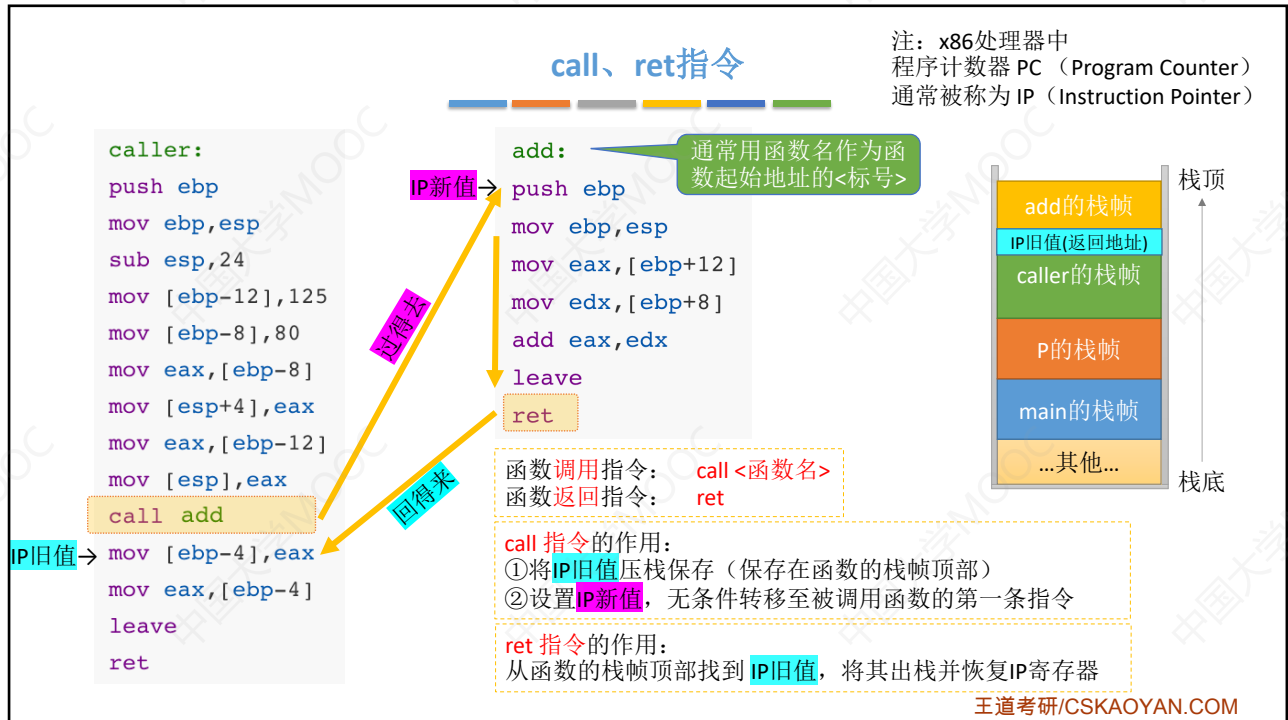
2



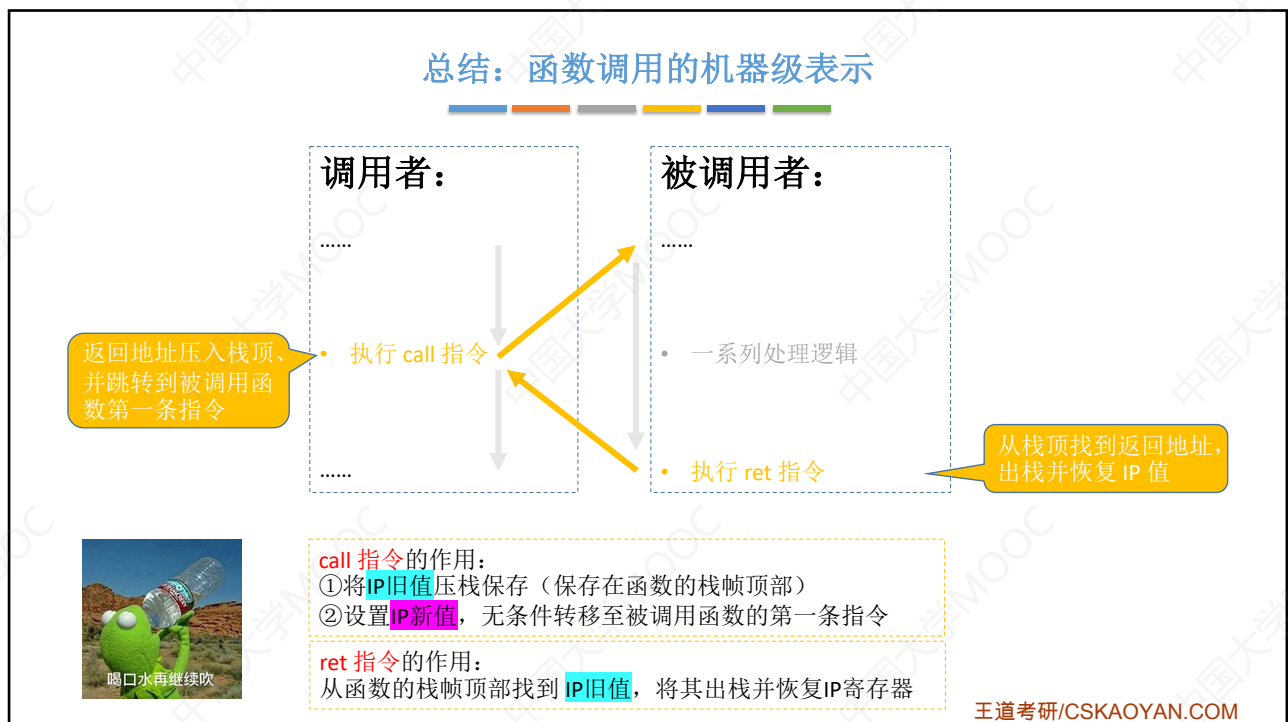
3



4



5



6

小朋友，你是否有很多问号？

```
int caller(){  
    int temp1=125;  
    int temp2=80;  
    int sum=add(temp1,temp2);  
    return sum;  
}  
  
int add(int x, int y){  
    return x+y;  
}
```



如何传递调用参数、返回值？

如何访问栈帧里的数据？

栈帧内可能包含哪些内容？

为什么倒过来了？



王道考研/CSKAOYAN.COM

7

本节内容

函数调用 机器级表示

——如何访问栈帧？

王道考研/CSKAOYAN.COM

8

小朋友，你是否有很多问号？

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}

int add(int x, int y){
    return x+y;
}
        
```

栈底

栈顶

栈为什么倒过来了？

别说我有问题
我觉得你有问题

如何传递调用参数、返回值？

如何访问栈帧里的数据？

栈帧内可能包含哪些内容？

王道考研/CSKAOYAN.COM

9

函数调用栈在内存中的位置

32位系统，进程虚拟地址空间为4GB

(高地址) 栈底

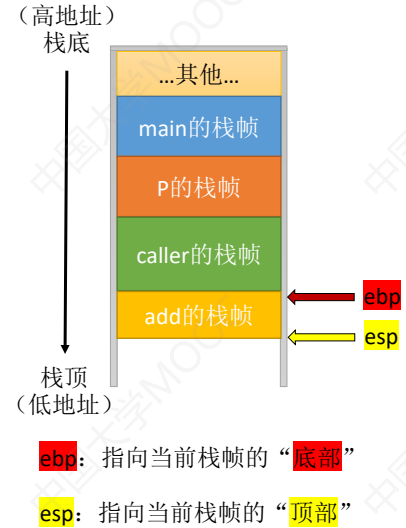
栈顶 (低地址)

注：大多数教材的图示，通常栈底在上，栈顶在下

王道考研/CSKAOYAN.COM

10

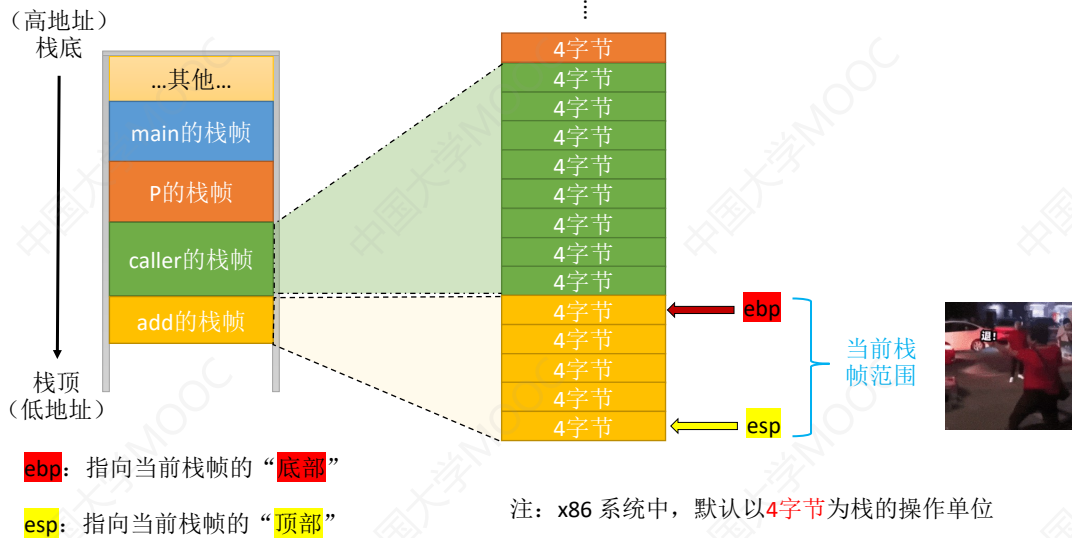
标记栈帧范围：EBP、ESP寄存器



王道考研/CSKAOYAN.COM

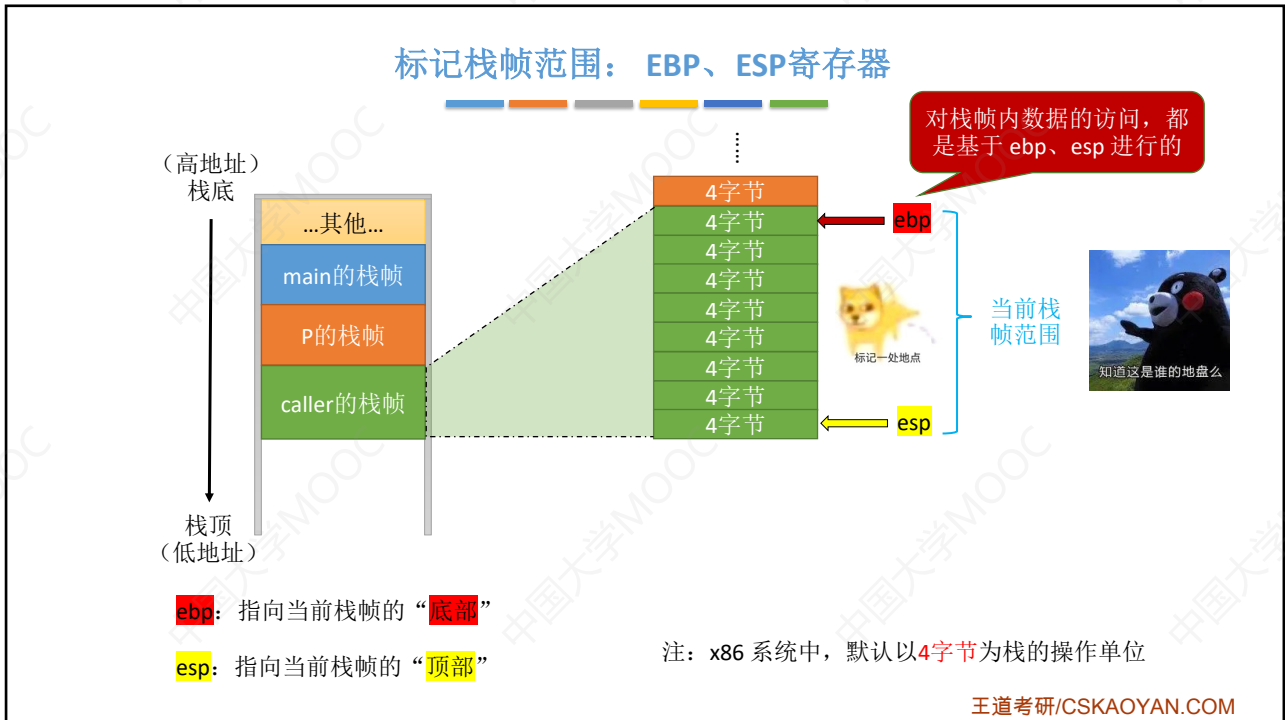
11

标记栈帧范围：EBP、ESP寄存器

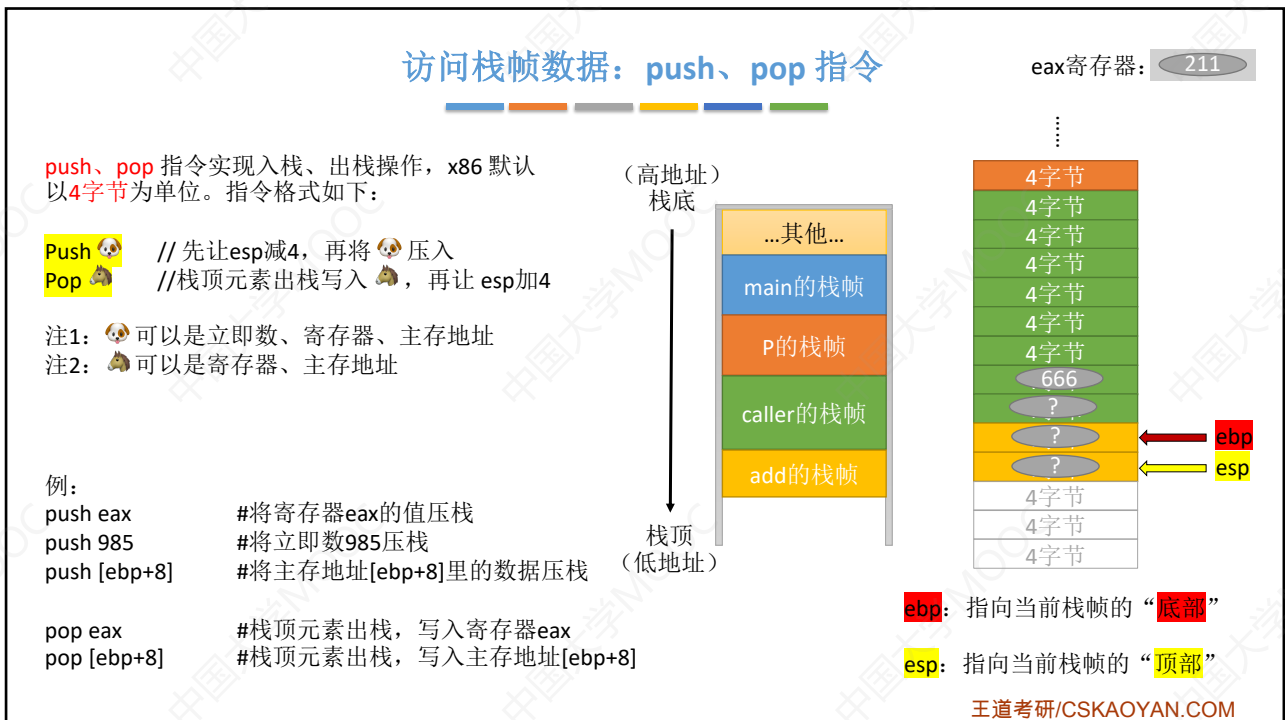


王道考研/CSKAOYAN.COM

12



13



14

访问栈帧数据: push、pop 指令

push、pop 指令实现入栈、出栈操作，x86 默认以4字节为单位。指令格式如下：

Push 🐼 // 先让esp减4，再将🐼压入
Pop 🐼 // 栈顶元素出栈写入🐼，再让 esp加4

注1: 🐼可以是立即数、寄存器、主存地址

注2: 🐼可以是寄存器、主存地址

例：

→ push eax #将寄存器eax的值压栈
push 985 #将立即数985压栈
push [ebp+8] #将主存地址[ebp+8]里的数据压栈

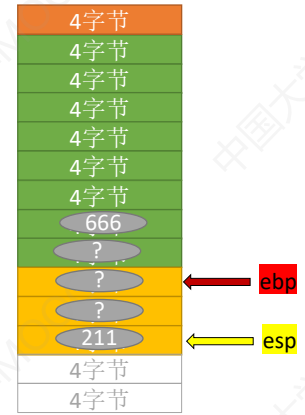
pop eax #栈顶元素出栈，写入寄存器eax
pop [ebp+8] #栈顶元素出栈，写入主存地址[ebp+8]

(高地址)
栈底

栈顶
(低地址)



eax寄存器: 211



ebp: 指向当前栈帧的“底部”

esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

15

访问栈帧数据: push、pop 指令

push、pop 指令实现入栈、出栈操作，x86 默认以4字节为单位。指令格式如下：

Push 🐼 // 先让esp减4，再将🐼压入
Pop 🐼 // 栈顶元素出栈写入🐼，再让 esp加4

注1: 🐼可以是立即数、寄存器、主存地址

注2: 🐼可以是寄存器、主存地址

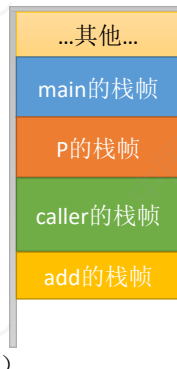
例：

→ push eax #将寄存器eax的值压栈
push 985 #将立即数985压栈
push [ebp+8] #将主存地址[ebp+8]里的数据压栈

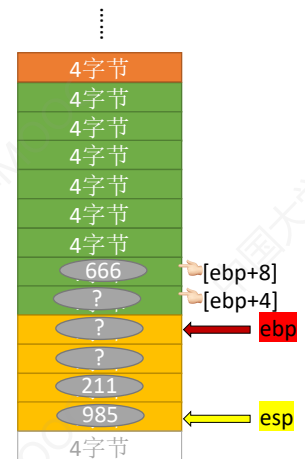
pop eax #栈顶元素出栈，写入寄存器eax
pop [ebp+8] #栈顶元素出栈，写入主存地址[ebp+8]

(高地址)
栈底

栈顶
(低地址)



eax寄存器: 211



ebp: 指向当前栈帧的“底部”

esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

16

访问栈帧数据: push、pop 指令

push、pop 指令实现入栈、出栈操作，x86 默认以4字节为单位。指令格式如下：

Push // 先让esp减4，再将 压入
Pop // 栈顶元素出栈写入 ，再让 esp加4

注1: 可以是立即数、寄存器、主存地址
 注2: 可以是寄存器、主存地址

例：
 push eax #将寄存器eax的值压栈
 push 985 #将立即数985压栈
 push [ebp+8] #将主存地址[ebp+8]里的数据压栈

pop eax #栈顶元素出栈，写入寄存器eax
 pop [ebp+8] #栈顶元素出栈，写入主存地址[ebp+8]

ebp: 指向当前栈帧的“底部”
esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

17

访问栈帧数据: push、pop 指令

push、pop 指令实现入栈、出栈操作，x86 默认以4字节为单位。指令格式如下：

Push // 先让esp减4，再将 压入
Pop // 栈顶元素出栈写入 ，再让 esp加4

注1: 可以是立即数、寄存器、主存地址
 注2: 可以是寄存器、主存地址

例：
 push eax #将寄存器eax的值压栈
 push 985 #将立即数985压栈
 push [ebp+8] #将主存地址[ebp+8]里的数据压栈

pop eax #栈顶元素出栈，写入寄存器eax
 pop [ebp+8] #栈顶元素出栈，写入主存地址[ebp+8]

ebp: 指向当前栈帧的“底部”
esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

18

访问栈帧数据: push、pop 指令

push、pop 指令实现入栈、出栈操作，x86 默认以**4字节**为单位。指令格式如下：

Push 🐼 // 先让esp减4，再将🐼压入
Pop 🐼 // 栈顶元素出栈写入🐼，再让 esp加4

注1: 🐼可以是立即数、寄存器、主存地址
 注2: 🐼可以是寄存器、主存地址

例:

```
push eax      #将寄存器eax的值压栈
push 985      #将立即数985压栈
push [ebp+8]  #将主存地址[ebp+8]里的数据压栈
```

pop eax #栈顶元素出栈，写入寄存器eax
 pop [ebp+8] #栈顶元素出栈，写入主存地址[ebp+8]

eax寄存器: 666

ebp: 指向当前栈帧的“底部”
esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

19

访问栈帧数据: mov 指令

例:

```
sub esp, 12    #栈顶指针-12
mov [esp+8], eax #将eax的值复制到主存[esp+8]
mov [esp+4], 958 #将958复制到主存[esp+4]
mov eax, [ebp+8] #将主存[ebp+8]的值复制到eax
mov [esp], eax  #将eax的值复制到主存[esp]
add esp, 8      #栈顶指针+8
```

eax寄存器: 211

ebp: 指向当前栈帧的“底部”
esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

20

访问栈帧数据：mov 指令

例：

```

sub esp, 12    #栈顶指针-12
mov [esp+8], eax #将eax的值复制到主存[esp+8]
mov [esp+4], 958 #将958复制到主存[esp+4]
mov eax, [ebp+8] #将主存[ebp+8]的值复制到eax
mov [esp], eax   #将eax的值复制到主存[esp]
add esp, 8      #栈顶指针+8
        
```

ebp: 指向当前栈帧的“底部”

esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

21

访问栈帧数据：mov 指令

例：

```

sub esp, 12    #栈顶指针-12
mov [esp+8], eax #将eax的值复制到主存[esp+8]
mov [esp+4], 958 #将958复制到主存[esp+4]
mov eax, [ebp+8] #将主存[ebp+8]的值复制到eax
mov [esp], eax   #将eax的值复制到主存[esp]
add esp, 8      #栈顶指针+8
        
```

ebp: 指向当前栈帧的“底部”

esp: 指向当前栈帧的“顶部”

王道考研/CSKAOYAN.COM

- 可以用 **mov** 指令，结合 **esp**、**ebp** 指针访问栈帧数据
- 可以用减法/加法指令，即 **sub/add** 修改栈顶指针 **esp** 的值

22

总结：如何访问栈帧？

方法一：

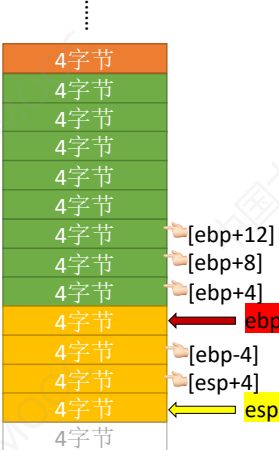
Push 🐼 // 先让esp减4，再将 🐼 压入
Pop 🐼 // 栈顶元素出栈写入 🐼，再让 esp加4

注1: 🐼 可以是立即数、寄存器、主存地址
注2: 🐼 可以是寄存器、主存地址

方法二：

mov 指令，结合 esp、ebp 指针访问栈帧数据

注：可以用减法/加法指令，即 sub/add 修改栈顶指针 esp 的值



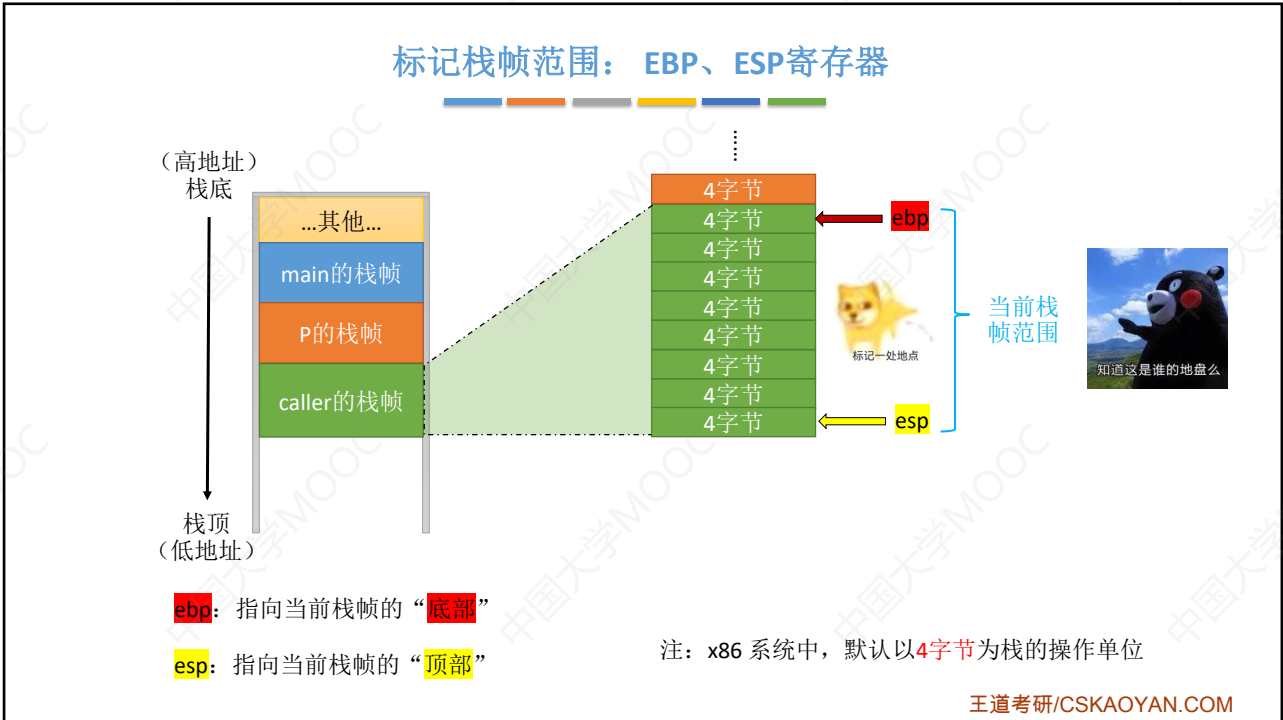
ebp: 指向当前栈帧的“底部”

esp: 指向当前栈帧的“顶部”

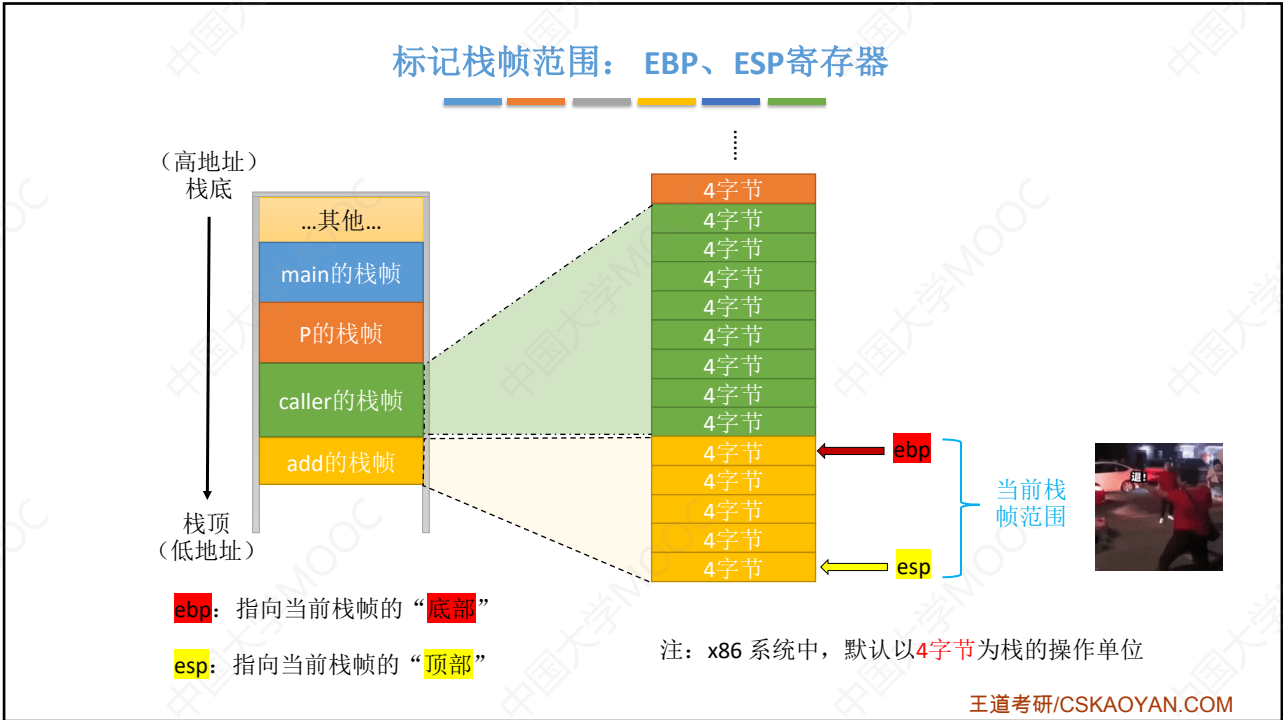
本节内容

函数调用
机器级表示

——如何切换栈帧？

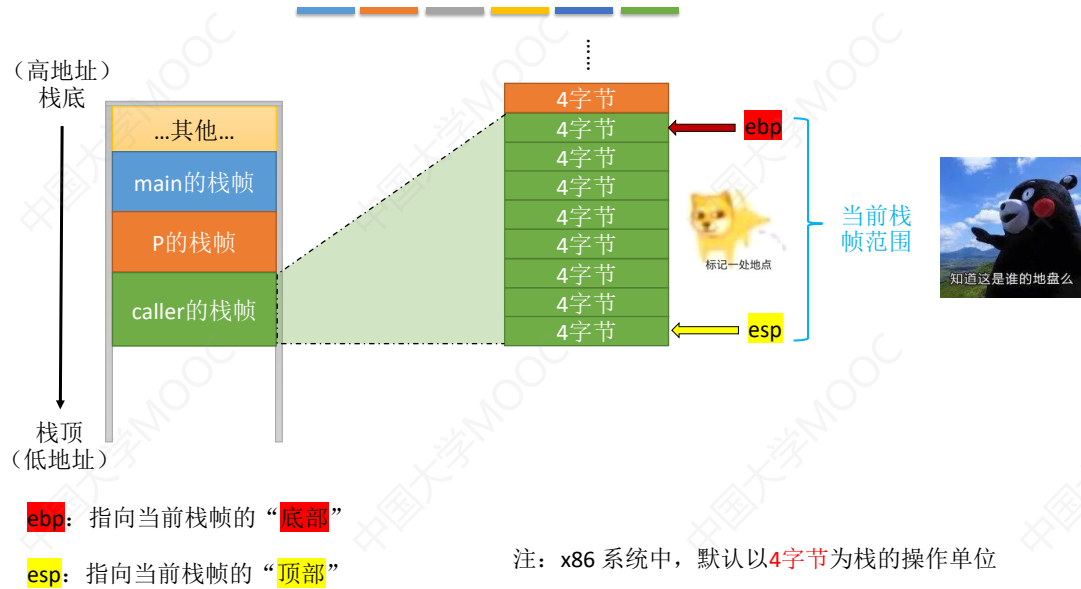


25



26

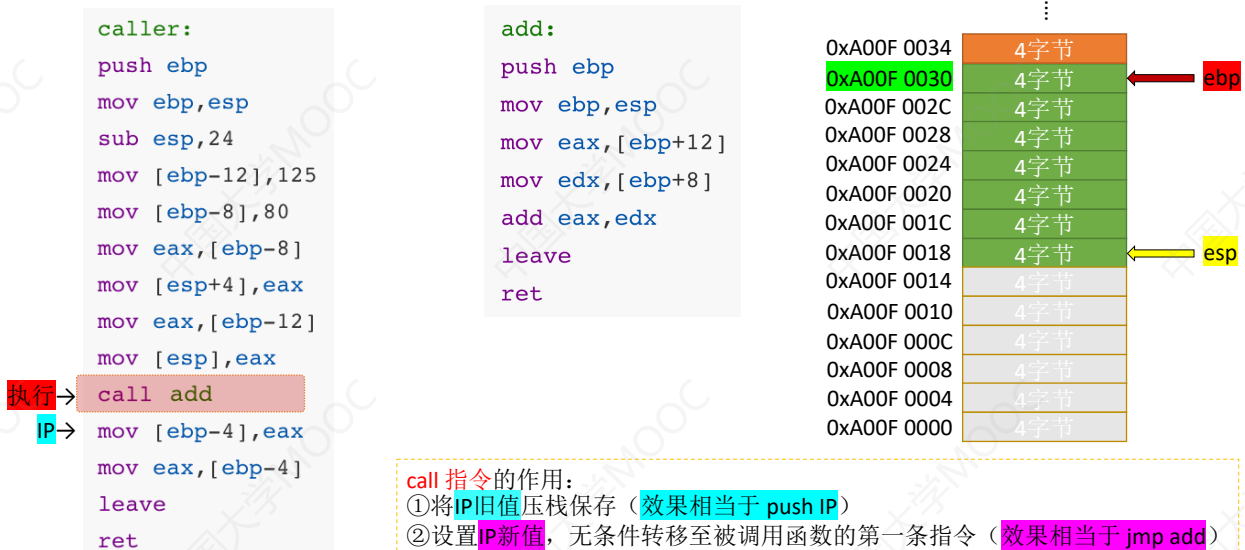
标记栈帧范围：EBP、ESP寄存器



王道考研/CSKAOYAN.COM

27

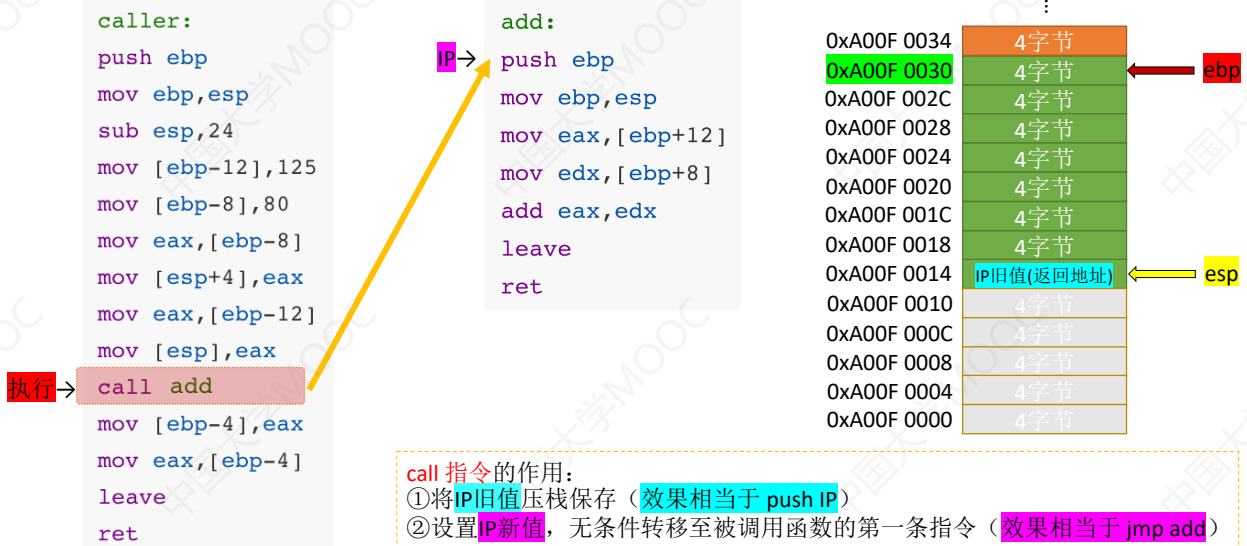
函数调用时, 如何切换栈帧?



王道考研/CSKAOYAN.COM

28

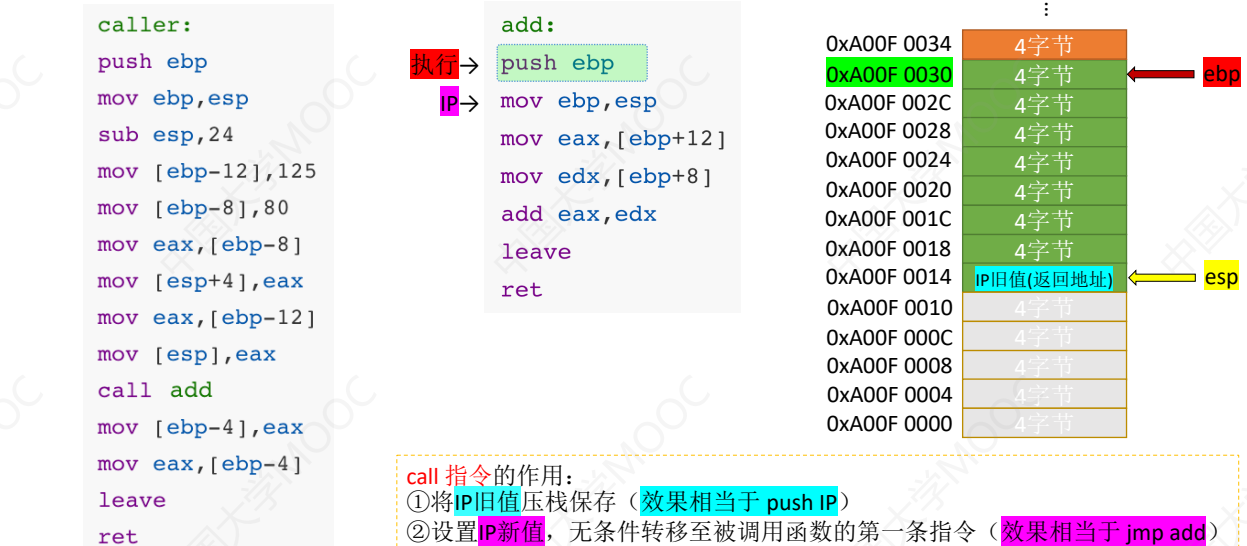
函数调用时，如何切换栈帧？



王道考研/CSKAOYAN.COM

29

函数调用时，如何切换栈帧？



王道考研/CSKAOYAN.COM

30

函数调用时，如何切换栈帧？

```

caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

```

执行 →

```

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret

```

IP →

0xA00F 0034	4字节	
0xA00F 0030	4字节	← ebp
0xA00F 002C	4字节	
0xA00F 0028	4字节	
0xA00F 0024	4字节	
0xA00F 0020	4字节	
0xA00F 001C	4字节	
0xA00F 0018	4字节	
0xA00F 0014	IP旧值(返回地址)	
0xA00F 0010	0xA00F 0030	← esp
0xA00F 000C	4字节	
0xA00F 0008	4字节	
0xA00F 0004	4字节	
0xA00F 0000	4字节	

王道考研/CSKAOYAN.COM

31

函数调用时，如何切换栈帧？

```

caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

```

执行 →

```

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret

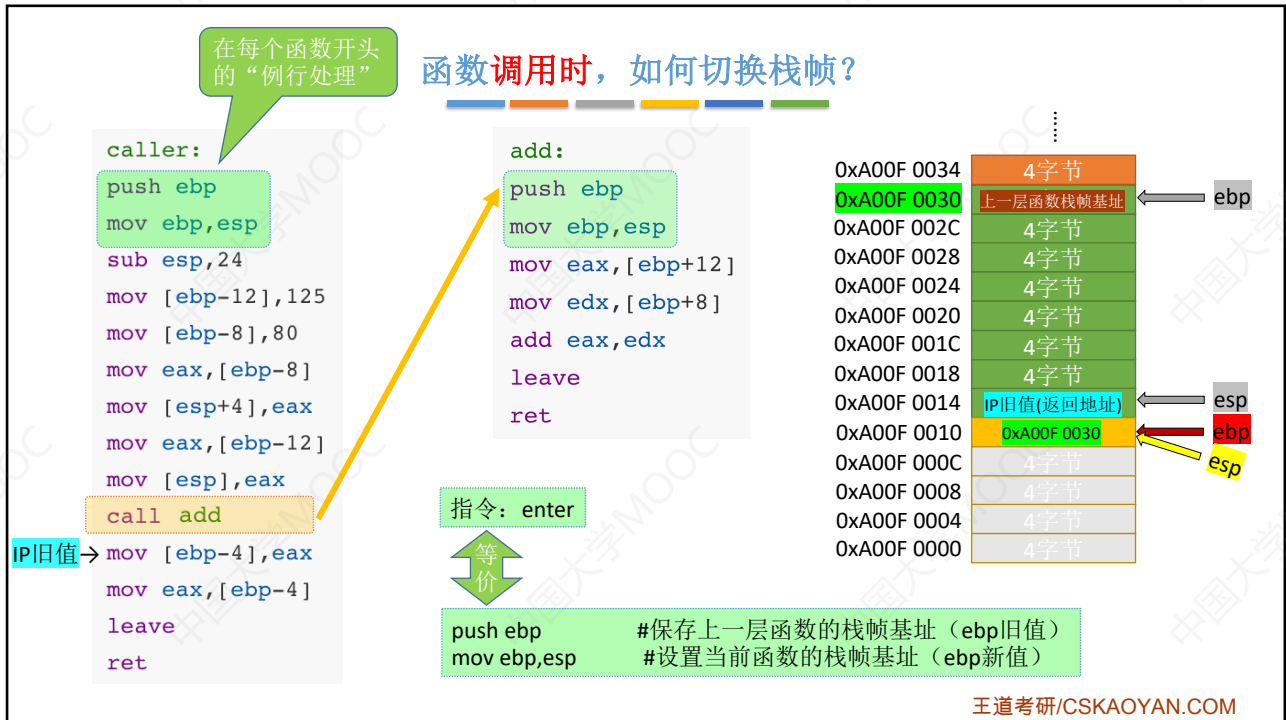
```

IP →

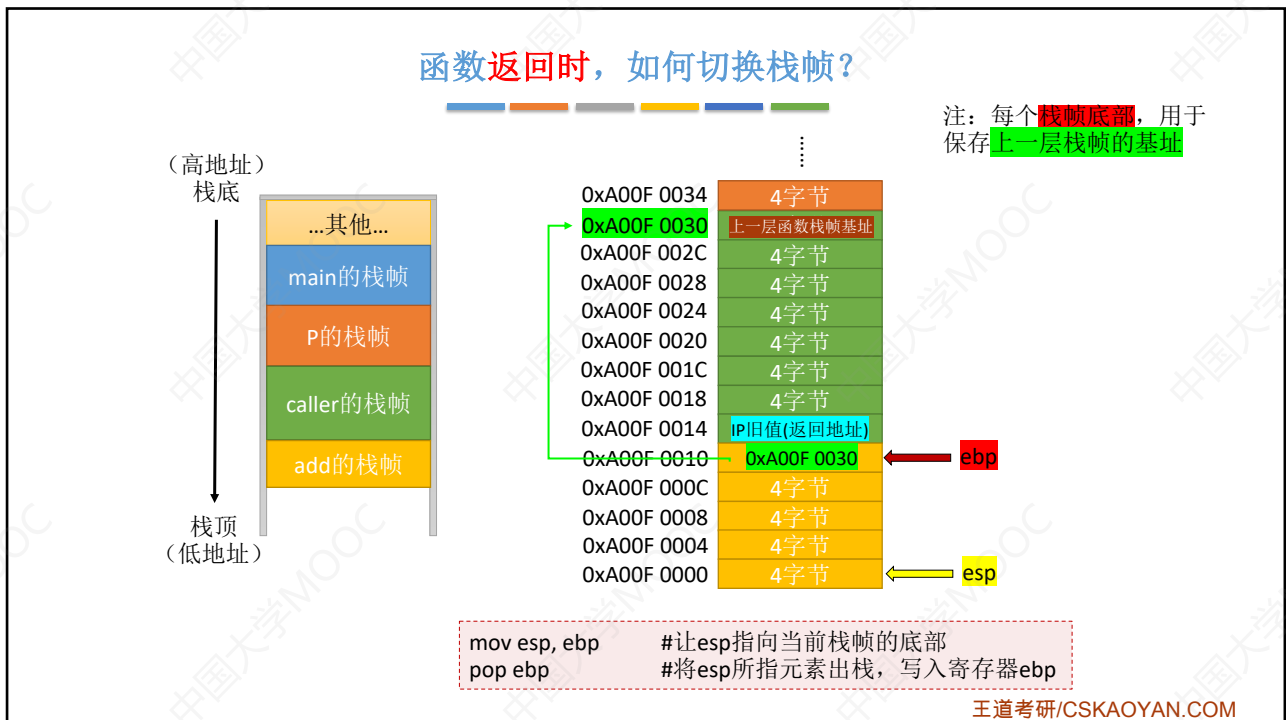
0xA00F 0034	4字节	
0xA00F 0030	4字节	
0xA00F 002C	4字节	
0xA00F 0028	4字节	
0xA00F 0024	4字节	
0xA00F 0020	4字节	
0xA00F 001C	4字节	
0xA00F 0018	4字节	
0xA00F 0014	IP旧值(返回地址)	
0xA00F 0010	0xA00F 0030	← ebp
0xA00F 000C	4字节	← esp
0xA00F 0008	4字节	
0xA00F 0004	4字节	
0xA00F 0000	4字节	

王道考研/CSKAOYAN.COM

32



33



34

栈顶
(低地址)



0xA00F 0034	4字节
0xA00F 0030	上一层函数栈帧基址
0xA00F 002C	4字节
0xA00F 0028	4字节
0xA00F 0024	4字节
0xA00F 0020	4字节
0xA00F 001C	4字节
0xA00F 0018	4字节
0xA00F 0014	IP旧值(返回地址)
0xA00F 0010	0xA00F 0030
0xA00F 000C	4字节
0xA00F 0008	4字节
0xA00F 0004	4字节
0xA00F 0000	4字节

➡ `mov esp, ebp` #让esp指向当前栈帧的底部
`pop ebp` #将esp所指元素出栈，写入寄存器ebp

35

栈顶
(低地址)

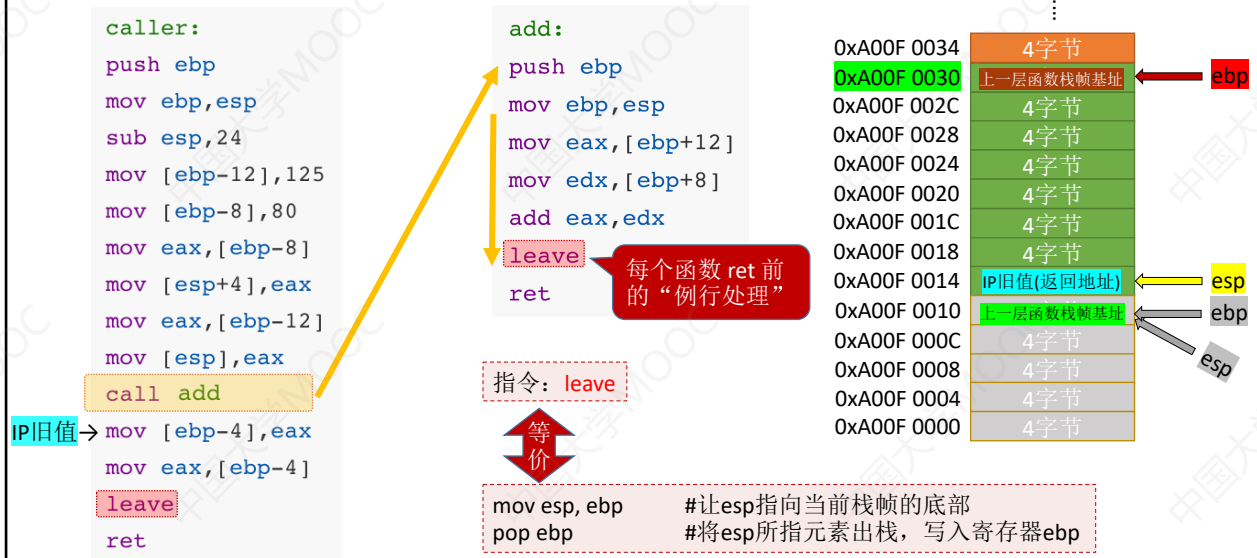


0xA00F 0034	4字节
0xA00F 0030	上一层函数栈帧基址
0xA00F 002C	4字节
0xA00F 0028	4字节
0xA00F 0024	4字节
0xA00F 0020	4字节
0xA00F 001C	4字节
0xA00F 0018	4字节
0xA00F 0014	IP旧值(返回地址)
0xA00F 0010	4字节
0xA00F 000C	4字节
0xA00F 0008	4字节
0xA00F 0004	4字节
0xA00F 0000	4字节

指令: `leave` 效果等价 `mov esp, ebp` #让esp指向当前栈帧的底部
 `pop ebp` #将esp所指元素出栈, 写入寄存器ebp

36

函数返回时，如何切换栈帧？



37

函数返回时，如何切换栈帧？



38

总结：函数调用的机器级表示

除了main函数，其他所有函数的汇编代码结构都一样！



调用者：

.....

返回地址压入栈顶，
并跳转到被调用函数
第一条指令

• 执行 call 指令

被调用者：

- 保存上一层函数栈帧，
设置当前函数栈帧

push ebp
mov ebp, esp

或：enter 指令

- 一系列处理逻辑

- 恢复上一层函数的栈帧

mov esp, ebp
pop ebp

或：leave 指令

- 执行 ret 指令

从栈顶找到返回地址，
出栈并恢复 IP 值

王道考研/CSKAOYAN.COM

39

一个函数的汇编代码框架：

例：2019年真题

被调用者：

- 保存上一层函数栈帧，
设置当前函数栈帧

- 一系列处理逻辑

- 恢复上一层函数的栈帧

- 执行 ret 指令

45. (16分) 已知 $f(n) = n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$ ，计算 $f(n)$ 的 C 语言函数 $f1$ 的源程序（阴影部分）及其在 32 位计算机 M 上的部分机器级代码如下：

行号	汇编代码	机器码	指令
1	00401000	55	push ebp
...
11	00401018	83 7D 08 01	cmp dword ptr [ebp+8], 1
12	0040101C	7E 17	jle f1+35h (00401035)
...
13	0040101E	8B 45 08	mov eax, dword ptr [ebp+8]
14	00401021	83 E8 01	sub eax, 1
15	00401024	50	push eax
16	00401025	E8 D6 FF FF FF	call f1 (00401000)
...
19	00401030	0F AF C1	imul eax, ecx
20	00401033	EB 05	jmp f1+3Ah (0040103a)
...
21	00401035	B8 01 00 00 00	mov eax, 1
...
26	00401040	3B EC	cmp ebp, esp
...
30	0040104A	C3	ret

push ebp
mov ebp, esp

mov esp, ebp
pop ebp

或：leave 指令

其中，机器级代码行包括行号、虚拟地址、机器指令和汇编指令，计算机 M 按字节编址，int 型数据占 32 位。请回答下列问题：

王道考研/CSKAOYAN.COM

40

一个函数的汇编代码框架：

被调用者：

- 保存上一层函数栈帧，设置当前函数栈帧
- 一系列处理逻辑
- 恢复上一层函数的栈帧
- 执行 ret 指令

例：2017年真题

44. (10分) 在按字节编址的计算机 M 上，题 43 中 f1 的部分源程序（阴影部分）与对应的机器级代码（包括指令的虚拟地址）如下：

其中，机器级代码行包括行号、虚拟地址、机器指令和汇编指令。请回答下列问题。

	int f1(unsigned n)	
1	00401020 55	push ebp
	mov ebp,esp
	for(unsigned i=0; i<= n-1; i++)	
	
20	0040105E 39 4D F4	cmp dword ptr [ebp-0Ch],ecx
	
	power *= 2;	
	
23	00401066 D1 E2	shl edx,1
	
	return sum;	
	
35	0040107F C3	ret

王道考研/CSKAOYAN.COM

41

本节内容

函数调用 机器级表示

- 栈帧内包含哪些内容？
- 参数、返回值传递

王道考研/CSKAOYAN.COM

42


问题回顾

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
        
```

```

int add(int x, int y){
    return x+y;
}
        
```




别说我有问题
我觉得你有问题

如何传递调用参数、返回值？

如何访问栈帧里的数据？


栈帧内可能包含哪些内容？

栈为什么倒过来了？



王道考研/CSKAOYAN.COM

43



非常优秀

GNU/Linux家的好东西，
开源、自由、非常流行


一个栈帧内可能包含哪些内容？

- gcc 编译器将每个栈帧大小设置为 16B 的整数倍（当前函数的栈帧除外），因此栈帧内可能出现空闲未使用的区域。
- 通常将局部变量集中存储在栈帧底部区域
- 通常将调用参数集中存储在栈帧顶部区域
- 栈帧最底部一定是上一层栈帧基址（ebp旧值）
- 栈帧最顶部一定是返回地址（当前函数的栈帧除外）

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}

int add(int x, int y){
    return x+y;
}
        
```



4字节

上一层栈帧基址

sum

temp2

temp1

空闲未使用

y

x

IP (返回地址)

4字节

（高地址）
栈底

↓

栈顶
（低地址）

...其他...

main的栈帧

P的栈帧

caller的栈帧

add的栈帧

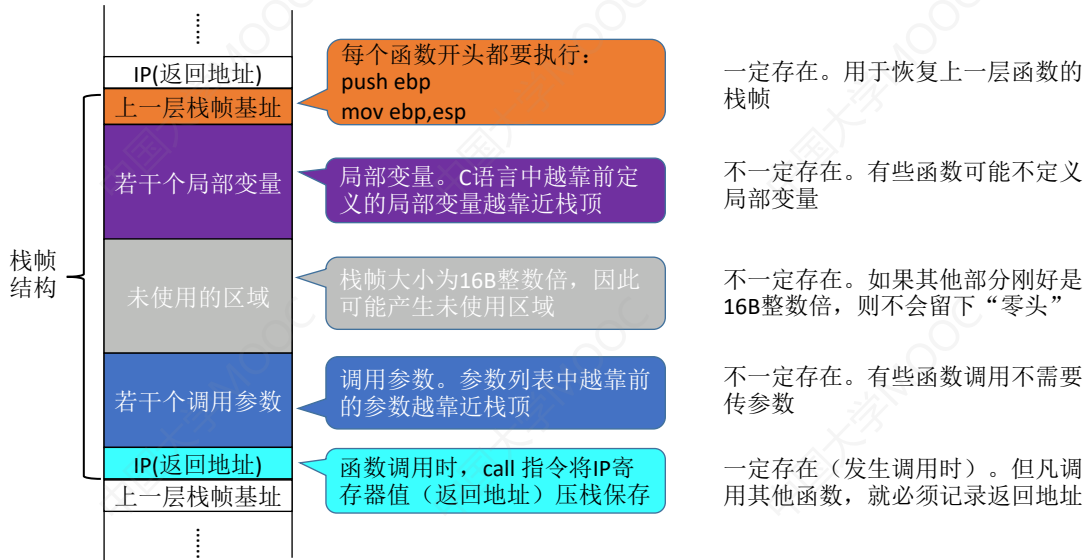
局部变量。C语言中越靠前定义的局部变量越靠近栈顶

调用参数。参数列表中越靠前的参数越靠近栈顶

王道考研/CSKAOYAN.COM

44

一个栈帧内可能包含哪些内容？



王道考研/CSKAOYAN.COM

45

汇编代码实战



王道考研/CSKAOYAN.COM

46

访问当前函数的局部变量：
[ebp-4]、[ebp-8]...

汇编代码实战：访问局部变量

栈帧结构

```

caller:
push ebp
mov ebp, esp
sub esp, 24
mov [ebp-12], 125
mov [ebp-8], 80
mov eax, [ebp-8]
mov [esp+4], eax
mov eax, [ebp-12]
mov [esp], eax
call add
mov [ebp-4], eax
mov eax, [ebp-4]
leave
ret

add:
push ebp
mov ebp, esp
mov eax, [ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret

```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}

```

内存地址与内容

0xA00F 0058	4字节	...
0xA00F 0034	4字节	...
0xA00F 0030	4字节	0xA00F 0058 ← ebp
0xA00F 002C	4字节	未初始化 (sum) ← [ebp-4]
0xA00F 0028	4字节	80 (temp2) ← [ebp-8]
0xA00F 0024	4字节	125 (temp1) ← [ebp-12]
0xA00F 0020	4字节	...
0xA00F 001C	4字节	...
0xA00F 0018	4字节	...
0xA00F 0014	4字节	...
0xA00F 0010	4字节	...
0xA00F 000C	4字节	...
0xA00F 0008	4字节	...

寄存器: eax: edx:

```

int add(int x, int y){
    return x+y;
}

```

王道考研/CSKAOYAN.COM

47

汇编代码实战：传递参数

栈帧结构

```

caller:
push ebp
mov ebp, esp
sub esp, 24
mov [ebp-12], 125
mov [ebp-8], 80
mov eax, [ebp-8]
mov [esp+4], eax
mov eax, [ebp-12]
mov [esp], eax
call add
mov [ebp-4], eax
mov eax, [ebp-4]
leave
ret

add:
push ebp
mov ebp, esp
mov eax, [ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret

```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}

```

内存地址与内容

0xA00F 0058	4字节	...
0xA00F 0034	4字节	...
0xA00F 0030	4字节	0xA00F 0058 ← ebp
0xA00F 002C	4字节	未初始化 (sum) ← [ebp-4]
0xA00F 0028	4字节	80 (temp2) ← [ebp-8]
0xA00F 0024	4字节	125 (temp1) ← [ebp-12]
0xA00F 0020	4字节	...
0xA00F 001C	4字节	80 (y) ← [esp+4]
0xA00F 0018	4字节	125 (x) ← esp
0xA00F 0014	4字节	...
0xA00F 0010	4字节	...
0xA00F 000C	4字节	...
0xA00F 0008	4字节	...

寄存器: eax: edx:

```

int add(int x, int y){
    return x+y;
}

```

王道考研/CSKAOYAN.COM

48

汇编代码实战：函数调用

栈帧结构

```

caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret
                    
```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
                    
```

内存地址表

0xA00F 0058	4字节
...	...
0xA00F 0034	4字节
0xA00F 0030	0xA00F 0058
0xA00F 002C	未初始化 (sum)
0xA00F 0028	80 (temp2)
0xA00F 0024	125 (temp1)
0xA00F 0020	4字节
0xA00F 001C	80 (y)
0xA00F 0018	125 (x)
0xA00F 0014	IP (返回地址)
0xA00F 0010	4字节
0xA00F 000C	4字节
0xA00F 0008	4字节

寄存器: eax: 125, edx:

王道考研/CSKAOYAN.COM

49

汇编代码实战：切换栈帧

栈帧结构

```

caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret
                    
```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
                    
```

内存地址表

0xA00F 0058	4字节
...	...
0xA00F 0034	4字节
0xA00F 0030	0xA00F 0058
0xA00F 002C	未初始化 (sum)
0xA00F 0028	80 (temp2)
0xA00F 0024	125 (temp1)
0xA00F 0020	4字节
0xA00F 001C	80 (y)
0xA00F 0018	125 (x)
0xA00F 0014	IP (返回地址)
0xA00F 0010	0xA00F 0030
0xA00F 000C	4字节
0xA00F 0008	4字节

寄存器: eax: 125, edx:

王道考研/CSKAOYAN.COM

50

访问上一层函数传过来的参数：
[ebp+8]、[ebp+12]...

汇编代码实战：访问参数

栈帧结构

栈帧结构

- IP(返回地址)
- 上一层栈帧基址
- 若干个局部变量
- 未使用的区域
- 若干个调用参数
- IP(返回地址)
- 上一层栈帧基址

```
caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret
```

```
int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
```

内存地址表

0xA00F 0058	4字节	P函数栈帧
...	...	
0xA00F 0034	4字节	caller栈帧
0xA00F 0030	0xA00F 0058	
0xA00F 002C	未初始化 (sum)	
0xA00F 0028	80 (temp2)	
0xA00F 0024	125 (temp1)	
0xA00F 0020	4字节	
0xA00F 001C	80 (y)	
0xA00F 0018	125 (x)	
0xA00F 0014	IP (返回地址)	
0xA00F 0010	0xA00F 0030	
0xA00F 000C	4字节	ebp esp
0xA00F 0008	4字节	

eax: 80 edx: 125

```
int add(int x, int y){
    return x+y;
}
```

王道考研/CSKAOYAN.COM

51

通常用eax寄存器返回结果

汇编代码实战：传递返回值

栈帧结构

栈帧结构

- IP(返回地址)
- 上一层栈帧基址
- 若干个局部变量
- 未使用的区域
- 若干个调用参数
- IP(返回地址)
- 上一层栈帧基址

```
caller:
push ebp
mov ebp,esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax,[ebp-8]
mov [esp+4],eax
mov eax,[ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax,[ebp-4]
leave
ret

add:
push ebp
mov ebp,esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax,edx
leave
ret
```

```
int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
```

内存地址表

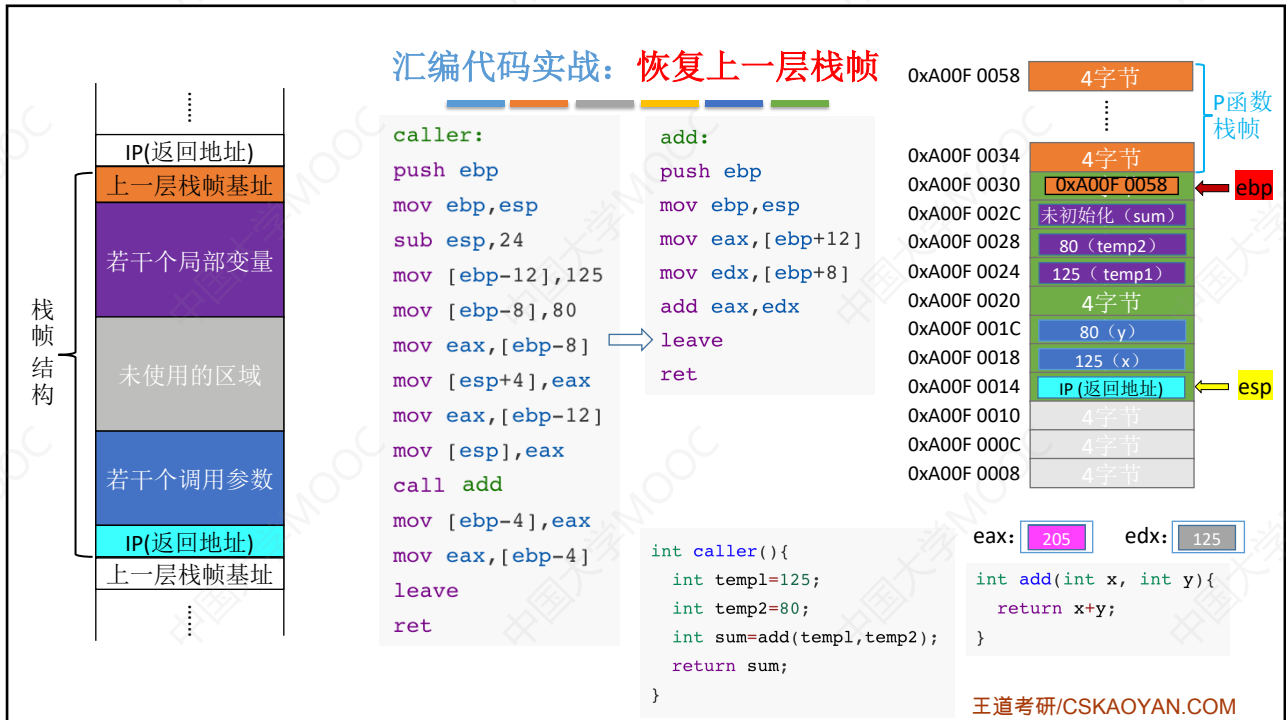
0xA00F 0058	4字节	P函数栈帧
...	...	
0xA00F 0034	4字节	caller栈帧
0xA00F 0030	0xA00F 0058	
0xA00F 002C	未初始化 (sum)	
0xA00F 0028	80 (temp2)	
0xA00F 0024	125 (temp1)	
0xA00F 0020	4字节	
0xA00F 001C	80 (y)	
0xA00F 0018	125 (x)	
0xA00F 0014	IP (返回地址)	
0xA00F 0010	0xA00F 0030	
0xA00F 000C	4字节	ebp esp
0xA00F 0008	4字节	

eax: 205 edx: 125

```
int add(int x, int y){
    return x+y;
}
```

王道考研/CSKAOYAN.COM

52



53



54

通常用eax寄存器返回结果

汇编代码实战：使用返回值

栈帧结构

```

caller:
push ebp
mov ebp, esp
sub esp, 24
mov [ebp-12], 125
mov [ebp-8], 80
mov eax, [ebp-8]
mov [esp+4], eax
mov eax, [ebp-12]
mov [esp], eax
call add
mov [ebp-4], eax
mov eax, [ebp-4]
leave
ret

add:
push ebp
mov ebp, esp
mov eax, [ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret
                    
```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
                    
```

内存地址表

0xA00F 0058	4字节
...	...
0xA00F 0034	4字节
0xA00F 0030	0xA00F 0058 ← ebp
0xA00F 002C	205 (sum) ← [ebp-4]
0xA00F 0028	80 (temp2)
0xA00F 0024	125 (temp1)
0xA00F 0020	4字节
0xA00F 001C	80 (y)
0xA00F 0018	125 (x) ← esp
0xA00F 0014	4字节
0xA00F 0010	4字节
0xA00F 000C	4字节
0xA00F 0008	4字节

eax: 205 edx: 125

```

int add(int x, int y){
    return x+y;
}
                    
```

王道考研/CSKAOYAN.COM

55

通常用eax寄存器返回结果

汇编代码实战：传递返回值

栈帧结构

```

caller:
push ebp
mov ebp, esp
sub esp, 24
mov [ebp-12], 125
mov [ebp-8], 80
mov eax, [ebp-8]
mov [esp+4], eax
mov eax, [ebp-12]
mov [esp], eax
call add
mov [ebp-4], eax
mov eax, [ebp-4]
leave
ret

add:
push ebp
mov ebp, esp
mov eax, [ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret
                    
```

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
                    
```

内存地址表

0xA00F 0058	4字节
...	...
0xA00F 0034	4字节
0xA00F 0030	0xA00F 0058 ← ebp
0xA00F 002C	205 (sum) ← [ebp-4]
0xA00F 0028	80 (temp2)
0xA00F 0024	125 (temp1)
0xA00F 0020	4字节
0xA00F 001C	80 (y)
0xA00F 0018	125 (x) ← esp
0xA00F 0014	4字节
0xA00F 0010	4字节
0xA00F 000C	4字节
0xA00F 0008	4字节

eax: 205 edx: 125

```

int add(int x, int y){
    return x+y;
}
                    
```

王道考研/CSKAOYAN.COM

56

总结：函数调用的机器级表示



继续说，我在听

调用者：

.....

可用 push 或 mov 指令实现

返回地址压入栈顶，并跳转到被调用函数第一条指令

通过eax寄存器

- 将调用参数写入当前栈帧的顶部区域
- 执行 call 指令
- 使用返回值

.....

被调用者：

- 保存上一层函数栈帧，设置当前函数栈帧
- 初始化局部变量
- 一系列处理逻辑
- 向上层函数传递返回值
- 恢复上一层函数的栈帧
- 执行 ret 指令

push ebp
mov ebp,esp
或：enter指令

[ebp-4]、[ebp-8]...

通过eax寄存器

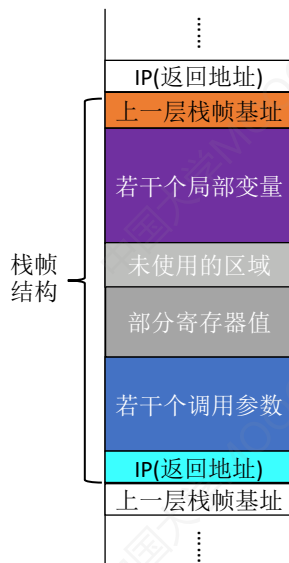
mov esp, ebp
pop ebp
或：leave指令

从栈顶找到返回地址，出栈并恢复 IP 值

王道考研/CSKAOYAN.COM

57

拓展：一个栈帧内可能包含哪些内容？



栈帧结构

调用其他函数前，如果有必要，可将某些寄存器（如：eax、edx、ecx）的值入栈保存，防止中间结果被破坏

不一定存在。如果这些寄存器值不是运算的中间结果，则可以不保存

王道考研/CSKAOYAN.COM

58

总结：函数调用的机器级表示



如 `eax`、`edx`、`ecx`

可用 `push` 或 `mov` 指令实现

返回地址压入栈顶，并跳转到被调用函数第一条指令

通过 `eax` 寄存器

调用者：

.....

保存必要的寄存器

• 将调用参数写入当前栈帧的顶部区域

• 执行 `call` 指令

• 使用返回值

恢复必要的寄存器

.....

被调用者：

• 保存上一层函数栈帧，设置当前函数栈帧

`push ebp`
`mov ebp, esp`
或： `enter` 指令

• 初始化局部变量

`[ebp-4]`、`[ebp-8]`...

• 一系列处理逻辑

• 向上层函数传递返回值

通过 `eax` 寄存器

• 恢复上一层函数的栈帧

`mov esp, ebp`
`pop ebp`
或： `leave` 指令

• 执行 `ret` 指令

从栈顶找到返回地址，出栈并恢复 IP 值

王道考研/CSKAOYAN.COM

59

函数调用 机器级代码

重要概念

栈帧 — 在函数调用栈中，一个栈帧对应一层函数，用于存储每层函数相关的一些信息
栈底在高地址方向、栈顶在低地址方向。以4字节为单位操作栈帧

ebp、esp 寄存器 — 标记了当前正在执行的函数栈帧范围
ebp 指向当前函数栈帧底部4字节、esp 指向当前函数栈帧顶部4字节

push、pop 指令 — `push` 指令将数据压入栈顶，并将 esp 减4
`pop` 指令将栈顶元素出栈，并将 esp 加4

如何切换栈帧 — 在每个函数开头，需例行执行 `enter` 指令 — 等价于 `push ebp` `mov ebp, esp`
每个函数 `ret` 之前，需例行执行 `leave` 指令 — 等价于 `mov esp, ebp` `pop ebp`

发起调用、返回 — `call` 指令 — ①将IP旧值压栈保存（保存在函数的栈帧顶部）
②设置IP新值，无条件转移至被调用函数的第一条指令
`ret` 指令 — 从函数的栈帧顶部找到 IP 旧值，将其出栈并恢复 IP 寄存器

如何传递参数、返回值 — 在 `call` 指令前，将调用参数写入栈帧顶部区域
在 `ret` 指令前，将函数返回值写入 `eax` 寄存器

如何访问栈帧内数据 — 访问当前函数的局部变量 — `[ebp-4]`、`[ebp-8]`...
访问上一层函数传过来的参数 — `[ebp+8]`、`[ebp+12]`...

一个栈帧包含哪些内容（自底向顶） — ①上一层栈帧的基地址（ebp旧值）；②若干个局部变量；③未使用区；
④部分寄存器值；⑤若干个调用参数；⑥返回地址（IP旧值）

王道考研/CSKAOYAN.COM

60