x86 Call/Return Protocol

1. Save "caller-save" registers (%eax, %ecx, %edx) onto stack (opt)

```
push %ecx
```

2. Push arguments onto stack, in reverse order (argN,…, arg1)

```
push 0x10 // argument 2
push 0x20 // argument 1
```

3. Call function (which pushes return address onto stack)

```
call 0x80400000
```

4. Establish new base pointer (saving old one)

```
push %ebp
movl %esp, %ebp
```

5. Save "callee-save" registers (%ebx, %esi, %edi) onto stack

```
push %ebx
```

6. Allocation: Make room for local variables on stack

```
sub 0x08, %esp
```

7. Execute body of routine (use base pointer to access arguments)

```
movl 0x8(%ebp), %ecx // argument 1
movl 0xc(%ebp), %edx // argument 2
```

8. Deallocation: Free local stack space

```
add 0x08, %esp
```

9. Restore "callee-save" registers

```
pop %ebx
```

10. Restore old base pointer

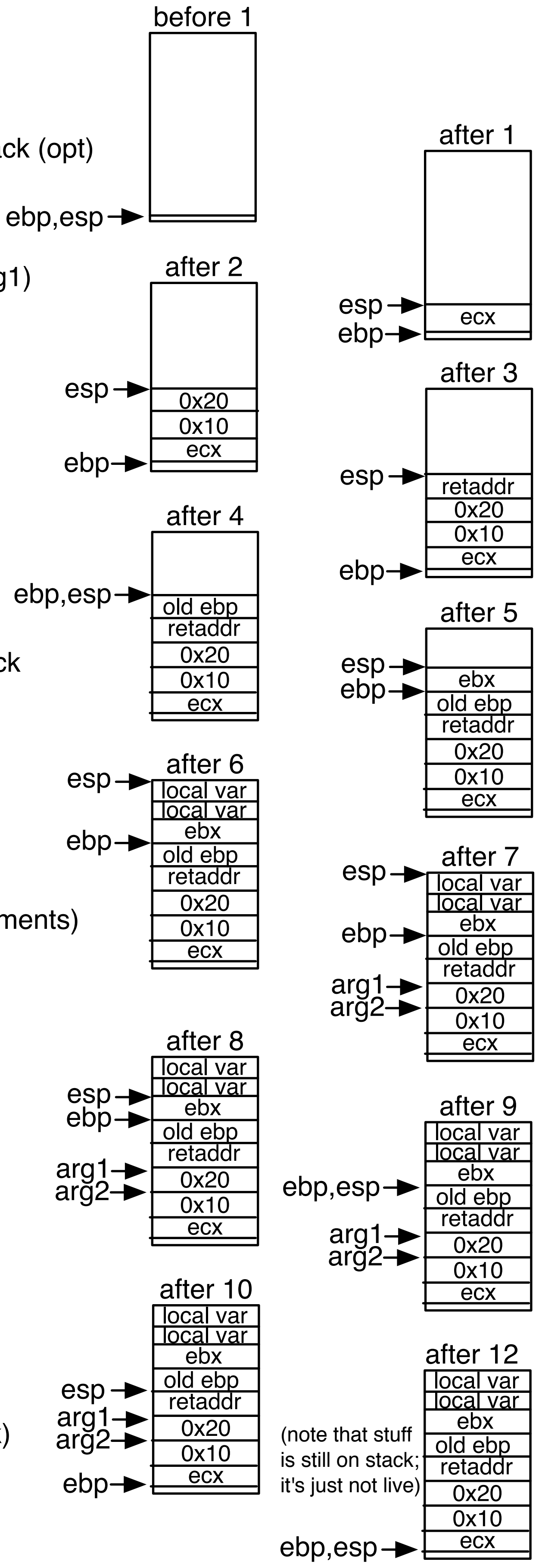
```
movl %ebp, %esp
pop %ebp
```

11. Return from function (popping return address off of stack)

```
ret
```

12. Restore "caller-save" registers

```
pop %ecx
```



before 1
ebp,esp
after 1
esp
ebp
ecx
after 2
esp
ebp
0x20
0x10
ecx
after 3
esp
ebp
retaddr
0x20
0x10
ecx
after 4
ebp,esp
old ebp
retaddr
0x20
0x10
ecx
after 5
esp
ebp
ebx
old ebp
retaddr
0x20
0x10
ecx
after 6
esp
ebp
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx
after 7
esp
ebp
arg1
arg2
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx
after 8
esp
ebp
arg1
arg2
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx
after 9
ebp,esp
arg1
arg2
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx
after 10
esp
arg1
arg2
ebp
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx
after 12
(note that stuff is still on stack; it's just not live)
ebp,esp
local var
local var
ebx
old ebp
retaddr
0x20
0x10
ecx