Announcement

- Assignment 3
  - Due Monday, October 20
  - Simulating an Accumulator Machine – Part II
  - Assignment 2 to be returned this afternoon.

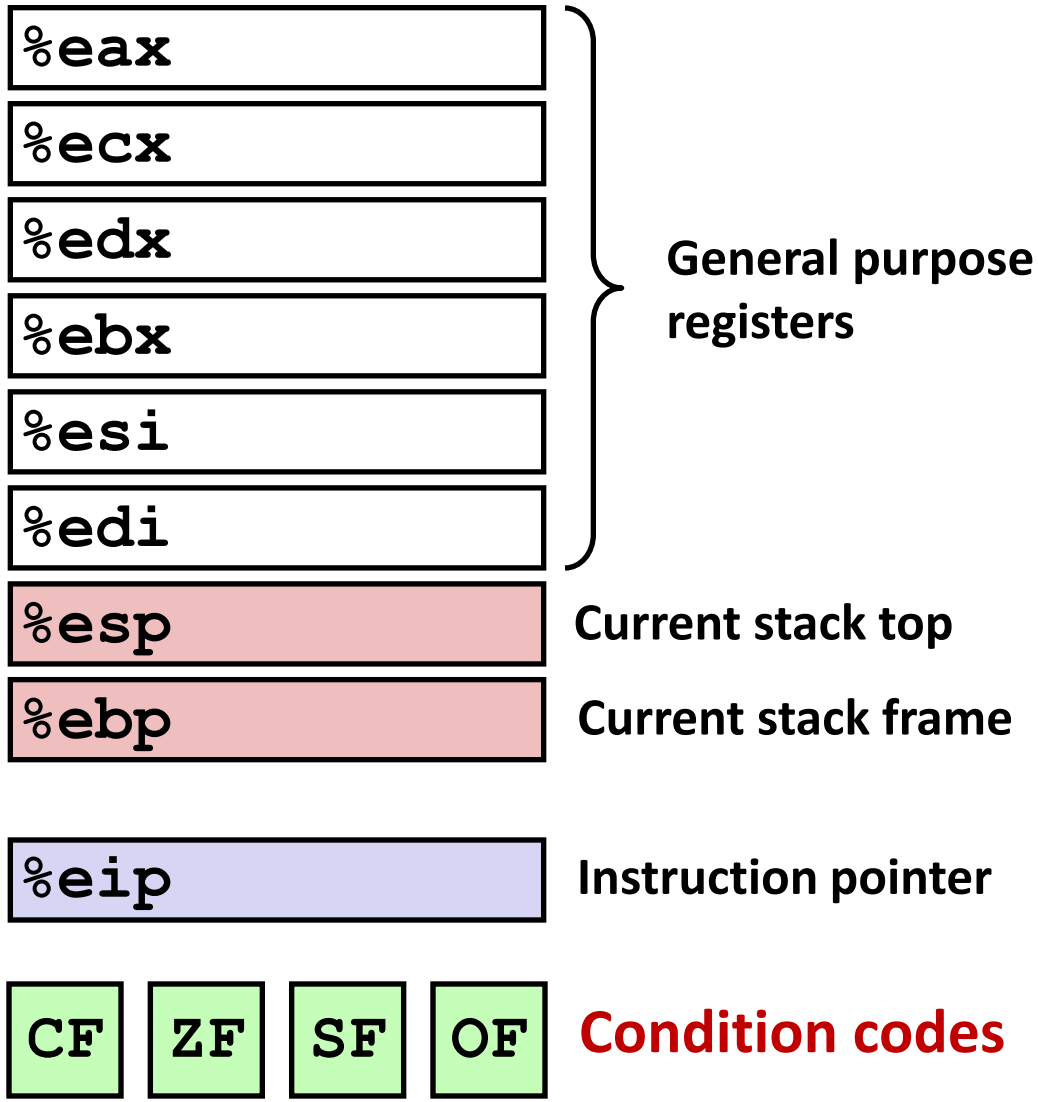
Lecture 17

Control: Iteration

CPSC 275
Introduction to Computer Systems

Processor State

- Information about currently executing program
  - Temporary data (**%eax**, …)
  - Location of runtime stack (**%ebp**,**%esp**)
  - Location of current code control point (**%eip**, …)
  - Status of recent tests (**CF, ZF, SF, OF**)



Setting Condition Codes

- Implicit setting by arithmetic instructions

```
      addl %ebx,%eax
      je .exit
        ...
.exit:
```

- Explicit setting by compare instruction

```
     cmpl   %eax,%edx
     jle    .L6
      ...
.L6:
```

Jumping

- `j`*x* Instructions
  - Jump to a different part of the code depending on the condition codes

j*x*	Description	Condition
`jmp`	Unconditional	`1`
`je`	Equal / Zero	`ZF`
`jne`	Not Equal / Not Zero	`~ZF`
`js`	Negative	`SF`
`jns`	Nonnegative	`~SF`
`jg`	Greater (Signed)	`~(SF^OF) & ~ZF`
`jge`	Greater or Equal (Signed)	`~(SF^OF)`
`jl`	Less (Signed)	`(SF^OF)`
`jle`	Less or Equal (Signed)	`(SF^OF) | ZF`
`ja`	Above (unsigned)	`~CF & ~ZF`
`jb`	Below (unsigned)	`CF`

“Do-While” Loop Example

C Code

```
int pcount_do(unsigned x) {
  int result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

Goto Version

```
int pcount_do(unsigned x) {
  int result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
  return result;
}
```

What does this function do?

“Do-While” Loop Compilation

Goto Version

```
int pcount_do(unsigned x)  {
  int result = 0;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
  return result;
}
```

```
%edx     x
%eax     result
```

```
  movl   $0,%eax       #   result = 0
.L2:                   # loop:
  movl   %edx,%ecx
  andl   $1,%ecx       #   t = x & 1
  addl   %ecx,%eax     #   result += t
  shrl   %edx          #   x >>= 1
  jne    .L2           #   If !0, goto loop
```

“While” Loop Example

C Code

```
int pcount_while(unsigned x) {
  int result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Goto Version

```
int pcount_do(unsigned x) {
  int result = 0;
  if (!x) goto done;
loop:
  result += x & 0x1;
  x >>= 1;
  if (x)
    goto loop;
done:
  return result;
}
```

General "While" Translation

While version

```
while (Test)
  Body
```

Do-While Version

```
  if (!Test)
    goto done;
  do
    Body
    while(Test);
done:
```

Goto Version

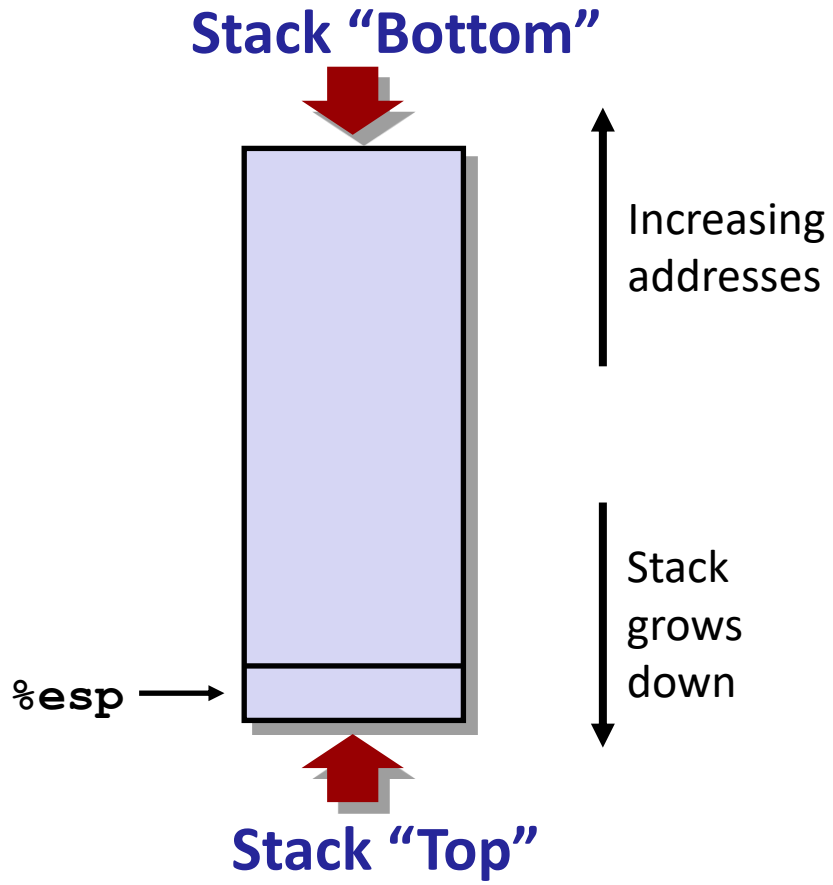
```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

What happens when a function is called?

Control: Procedures

IA-32 Stack

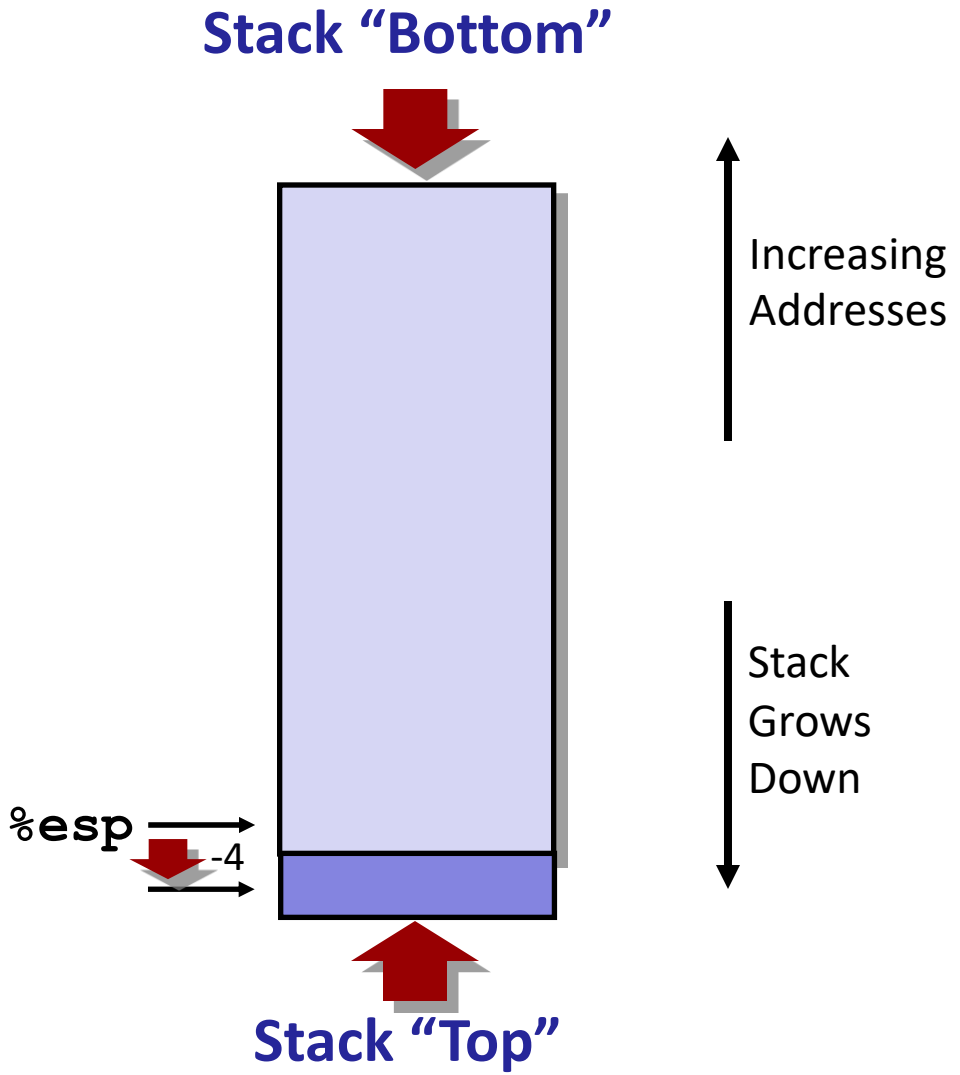
- Grows toward lower addresses
- Register **`%esp`** contains lowest stack address
  - address of "top" element
- Two operations on stack:
  - **push**: adding a new item on the stack
  - **pop**: removing the top element from the stack



IA32 Stack: `push`

`pushl` *src*

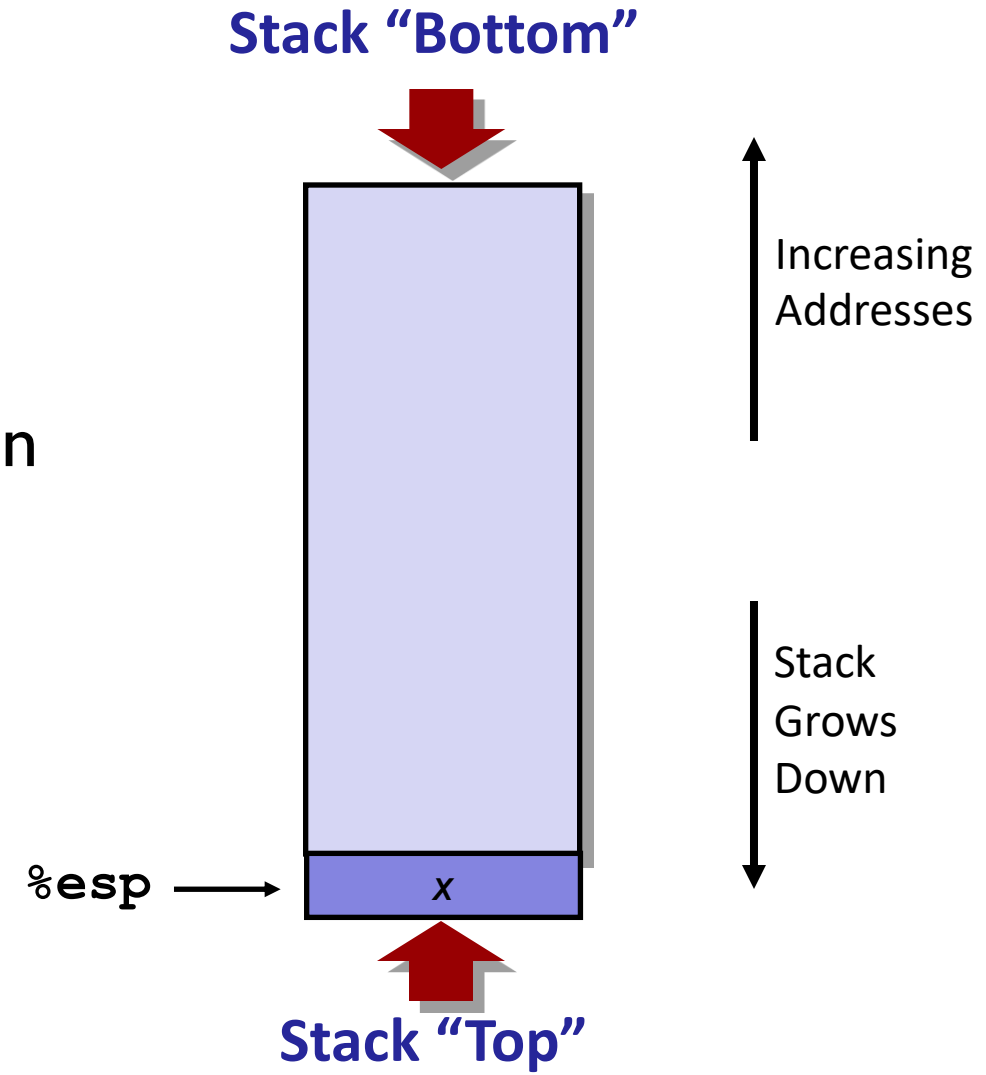
- Fetch operand at *src*
- Decrement **`%esp`** by 4



IA32 Stack: `push`

`pushl` *src*

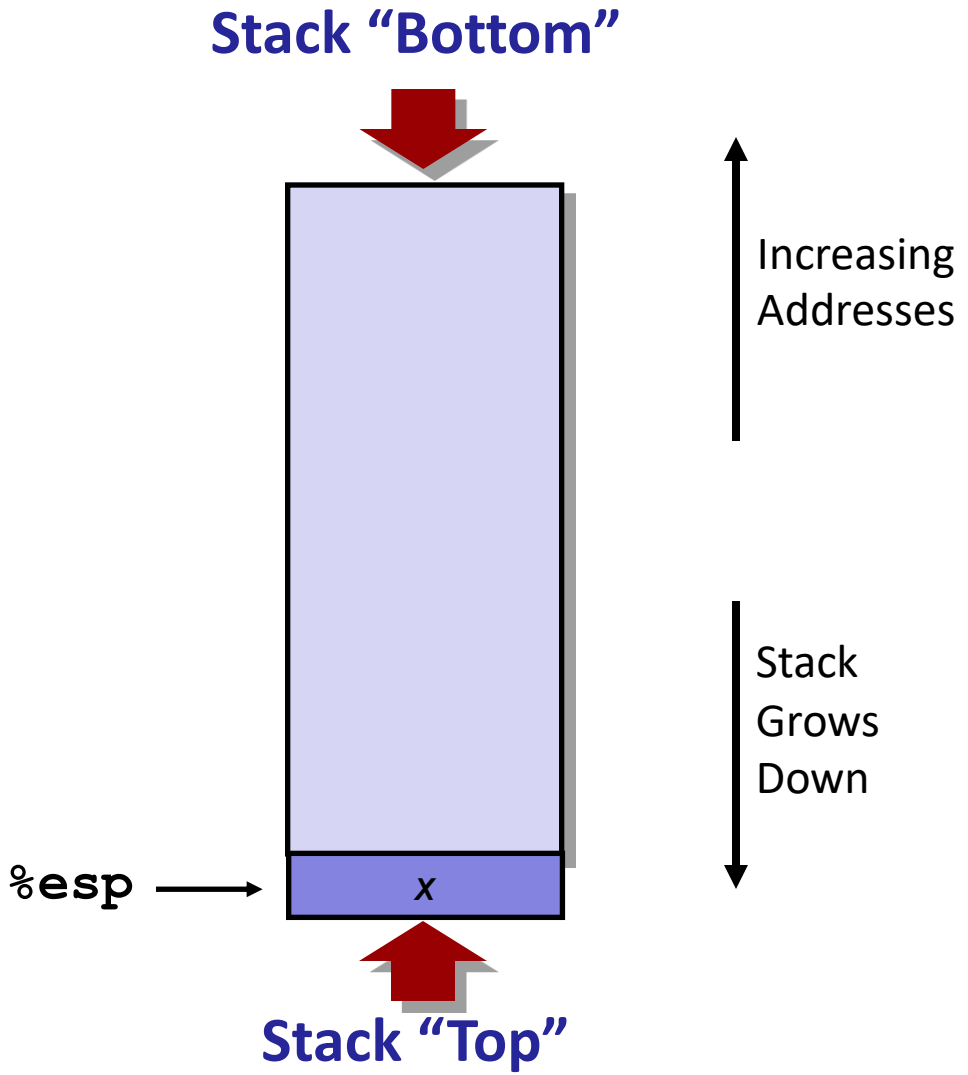
- Fetch operand at *src*
- Decrement **`%esp`** by 4
- Write operand at address given by **`%esp`**



IA32 Stack: **pop**

`popl` *dest*

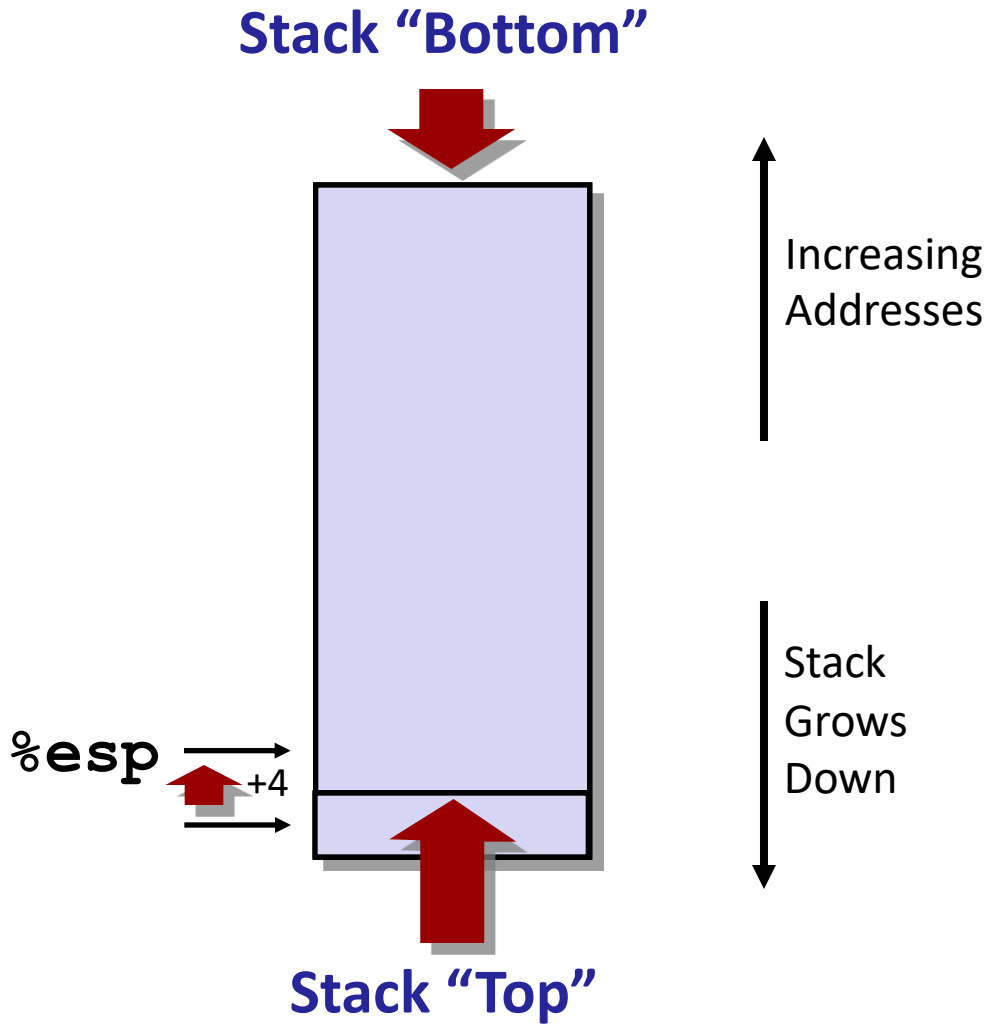
– Copy the stack top item to *dest*.



IA32 Stack: **pop**, cont'd

`popl` *dest*

- Copy the stack top item to *dest*.
- Increment **`%esp`** by 4

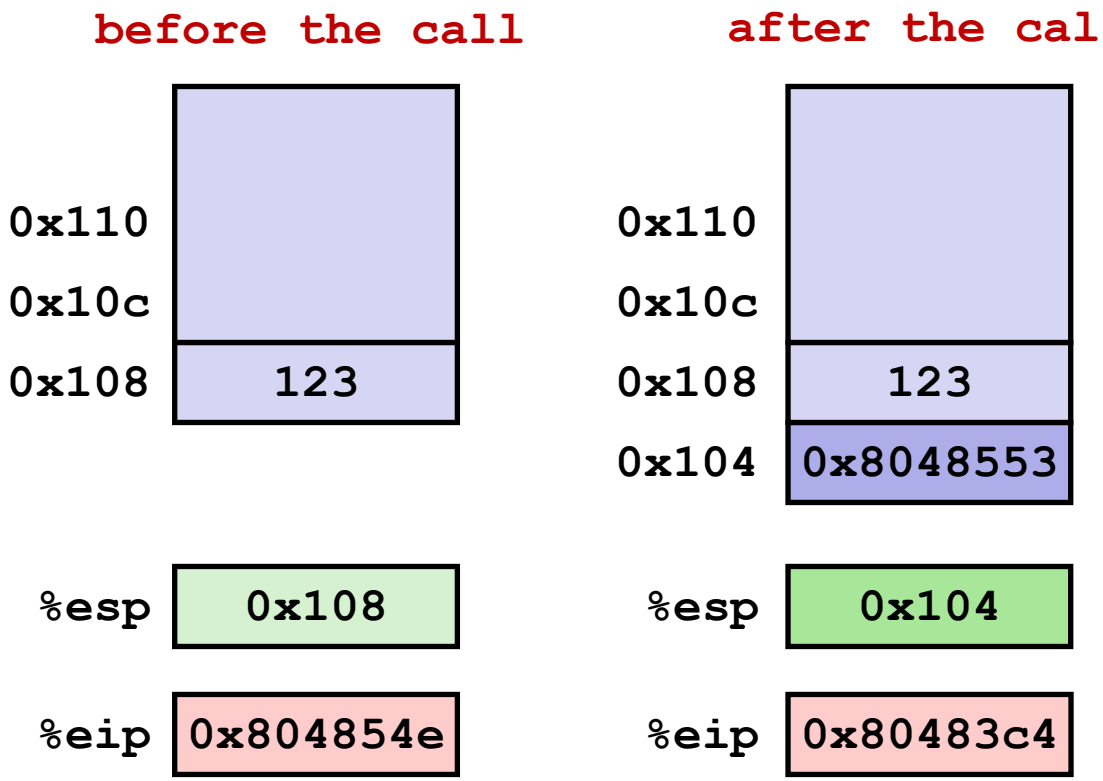


Procedure Control Flow

- Use stack to support procedure (function) calls and return
- Procedure call: **`call`** *`label`*
  - Push *return address* on stack
  - Jump to *`label`*
- Return address: address of the *next* instruction right after call
- Procedure return: **`ret`**
  - Pop return address from stack
  - Jump to return address

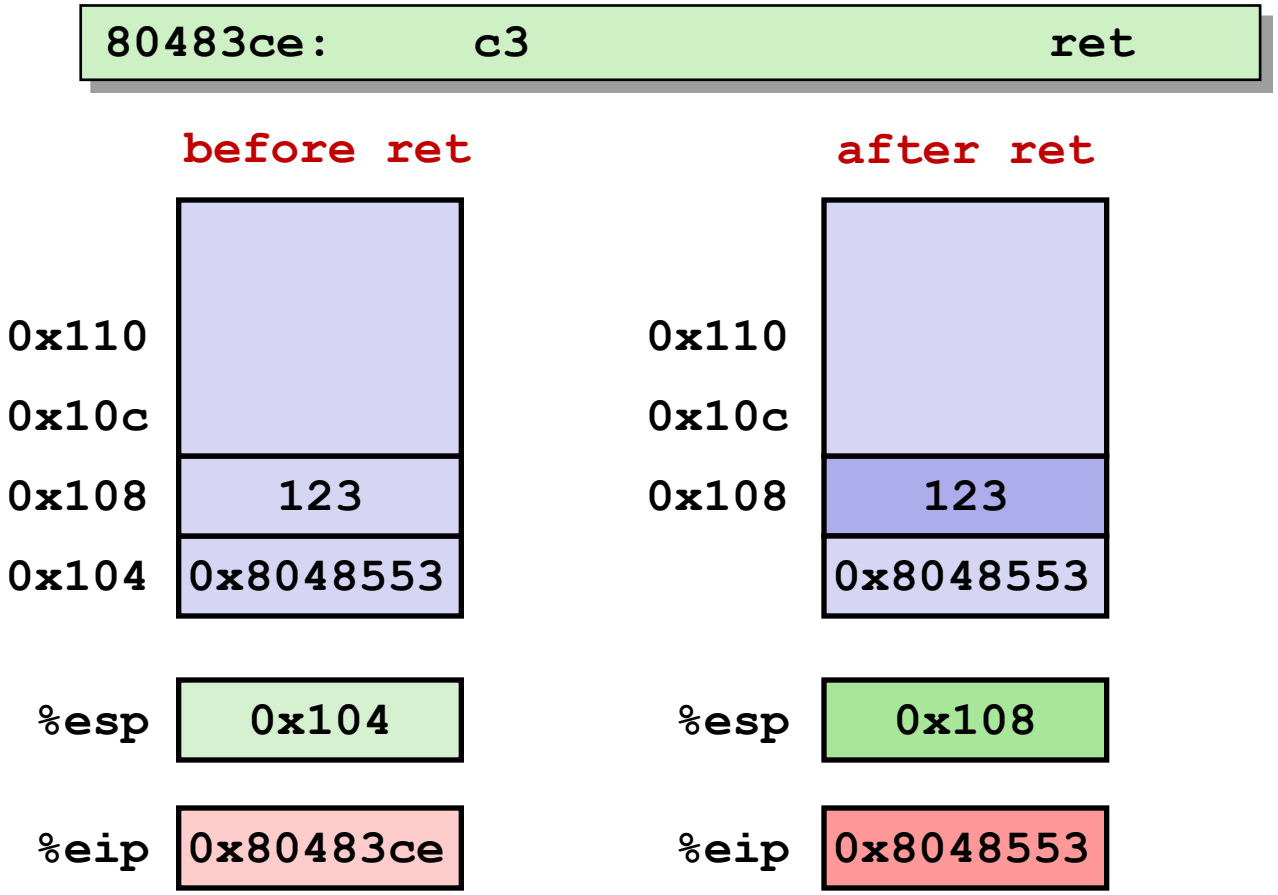
Procedure Call Example

```
804854e:    e8 3d 06 00 00        call   80483c4 <sum>
8048553:    50                    pushl  %eax
```



%eip: program counter

Procedure Return Example



%eip: program counter