Slide 106

How would we translate this to IA32? What should be on the stack?

```
int func(int a, int b, int c) {
  return b+c;
}

int main() {
  func(1, 2, 3);
}
```

Assume the stack initially looks like:

Slide 107

How would we translate this to IA32? What should be on the stack?

main:

func:

Stack

Slide 108

How would we translate this to IA32? What should be on the stack?

main:

1. push $3
2. push $2
3. push $1
4. call func

func:

Slide 109

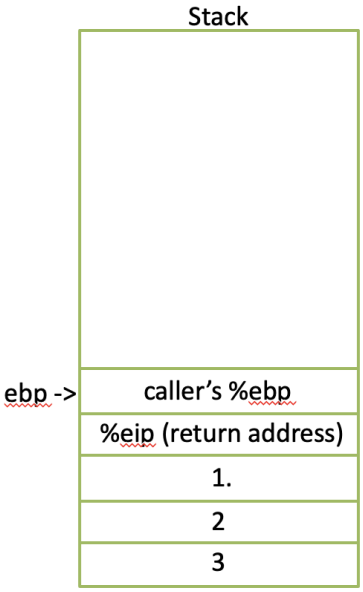
How would we translate this to IA32? What should be on the stack?

main:

1. push $3
2. push $2
3. push $1
4. call func

func:

1. push %ebp
2. movl %esp, %ebp (move %ebp up)
3. subl $N, %esp (if we needed space)



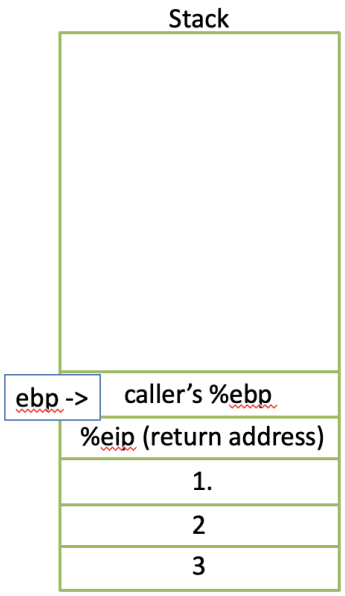
How would we translate this to IA32? What should be on the stack?

main:

1. push $3
2. push $2
3. push $1
4. call func

func:

1. push %ebp
2. movl %esp, %ebp (move %ebp up)
3. subl $N, %esp (if we needed space)
4. movl 12(%ebp), %eax
5. add 16(%ebp), %eax
6. leave
7. ret



Register Usage Conventions

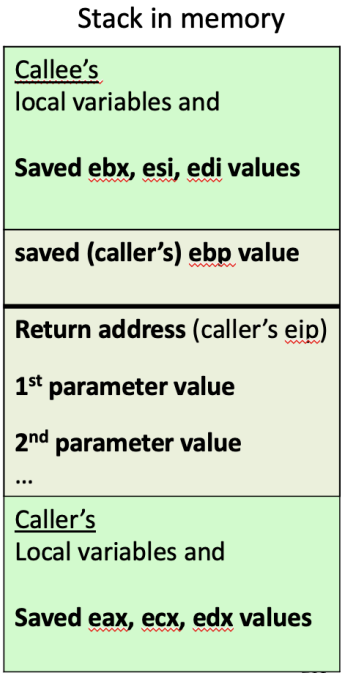
eax, edx, ecx: caller saved registers:

if values needed by caller after call, caller must save them to its frame prior to call

ebx, esi, edi: callee saved registers:

callee must save these resisters values to its frame before use, and restore the saved values prior to returning to caller

- This is why you see functions use eax, ecx, and edx (it doesn't have to save them to use them)



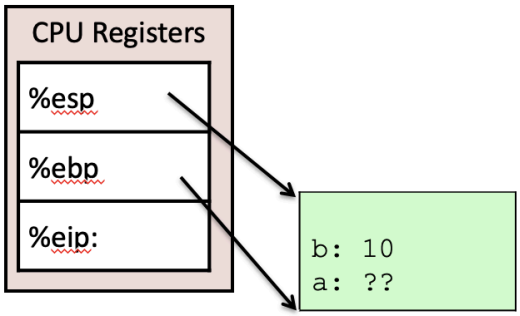
Example: translate to IA32

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

```
int sum(int x, int y) {
    int res;
    res = x+y;
    return res;
}
```

Start with IA32 code to call to sum

```
main:
  # assume some main code
  # and a at %ebp-8,  b at %ebp-12
```



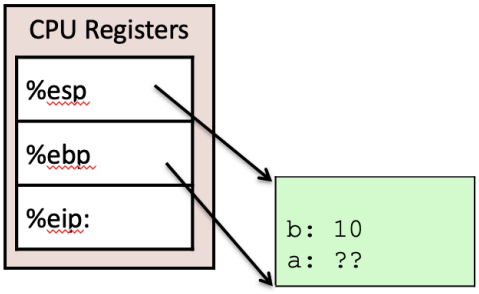
Example: translate to IA32

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

```
int sum(int x, int y) {
    int res;
    res = x+y;
    return res;
}
```

(1) Push argument values on stack: last arg value pushed first

```
main:
  # assume some main code
  # and a at %ebp-8,  b at %ebp-12
  push $3
  push -12(%ebp)
```



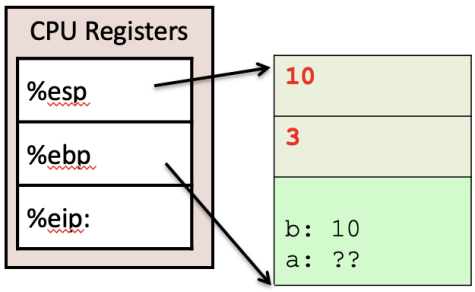
Example: translate to IA32

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

```
int sum(int x, int y) {
    int res;
    res = x+y;
    return res;
}
```

(2) call sum function
(saves %eip, jmps to start of sum)

```
main:
  # assume some main code
  # and a at %ebp-8, b at %ebp-12
  push $3
  push -12(%ebp)
  call sum
```

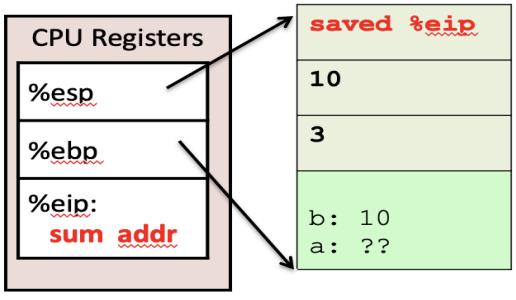


Example: translate to IA32

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

```
int sum(int x, int y) {
    int res;
    res = x+y;
    return res;
}
```

```
main:
  # assume some main code
  # and a at %ebp-8, b at %ebp-12
  push $3
  push -12(%ebp)
  call sum
```



Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
     int res;
     res = x+y;
     return res;
}
```

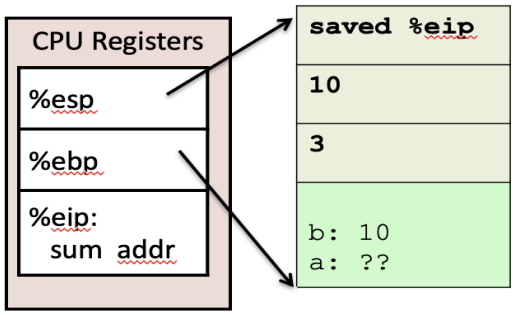
```
sum:
  # func preamble
  # instructions

  # then sum function
  # body instructions
```

Now at 1st instruction in sum but sum's stack still needs set-up

Function Preamble Code

- finishes the job of setting up the callee's stack frame
- Comes before any instrs in the function body



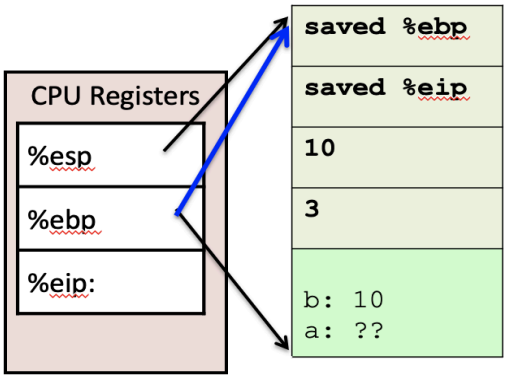
Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
     int res;
     res = x+y;
     return res;
}
```

Function Preamble Code

(4) Change %ebp to point to sum's bottom of stack

```
sum:
  pushl %ebp
  movl %esp, %ebp
```



Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
    int res;
    res = x+y;
    return res;
}
```

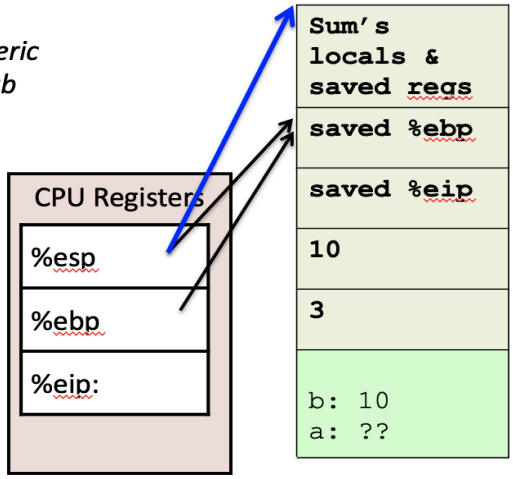
Function Preamble Code

(5) Make space on the stack for sum's local variables (and spilled registers)

Function preamble code is often very generic every function beginning is: push, mov, sub

```
sum:
  pushl %ebp
  movl %esp, %ebp
  subl $20, %esp
```

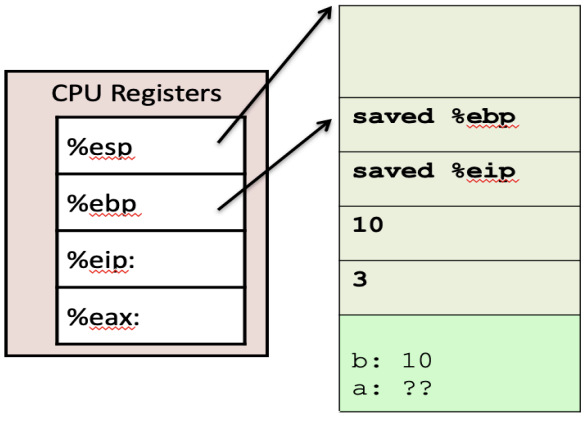
Why $20?
Why not: enough space for local variable and some saved register values



Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
    int res;
    res = x+y;
    return res;
}
sum:
  pushl %ebp
  movl %esp, %ebp
  subl $20, %esp
  movl 8(%ebp), %eax
  addl 12(%ebp), %eax
  movl %eax, -4(%ebp)
```

(6) Next, translates sum's function body code and put return values in %eax (let's say res is at %ebp -4)



Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
     int res;
     res = x+y;
     return res;
}
```

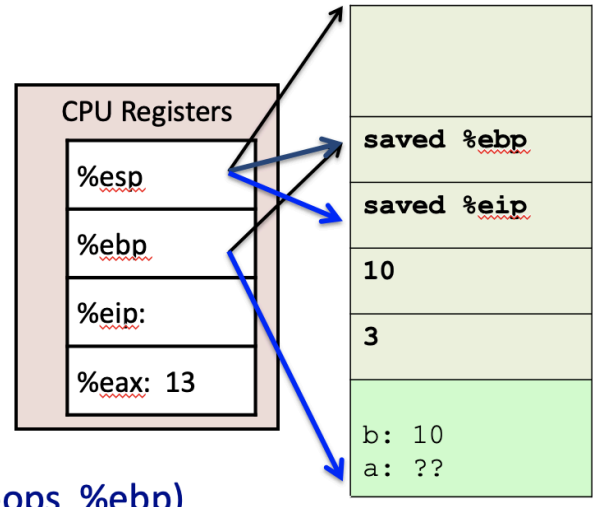
```
sum:
   pushl %ebp
   movl %esp, %ebp
   subl $20, %esp
   movl 8(%ebp), %eax
   addl 12(%ebp), %eax
   movl %eax, -4(%ebp)
   leave
```

(leave: %esp ← %ebp and pops %ebp)

Next, translates return from sum:

(7) put return value in %eax
(it is already there)

(8) restore caller's frame (mostly)



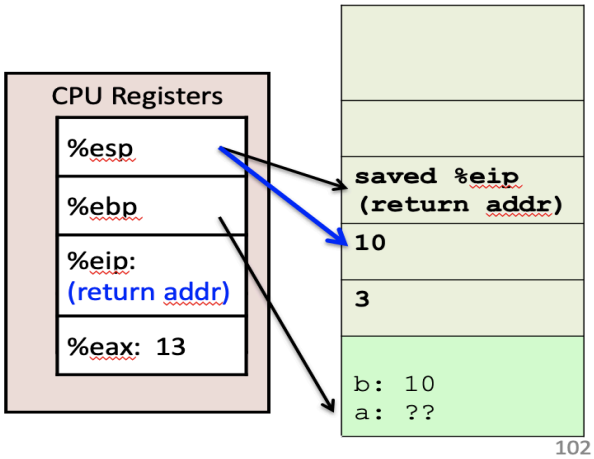
Example: translate to IA32 (cont)

```
int sum(int x,int y)
{
     int res;
     res = x+y;
     return res;
}
```

```
sum:
   pushl %ebp
   movl %esp, %ebp
   subl $20, %esp
   movl 8(%ebp), %eax
   addl 12(%ebp), %eax
   movl %eax, -4(%ebp)
   leave
   ret
```

Next, translates return from sum:

(9) return to caller:
Pop the return addres (saved %eip) into %eip



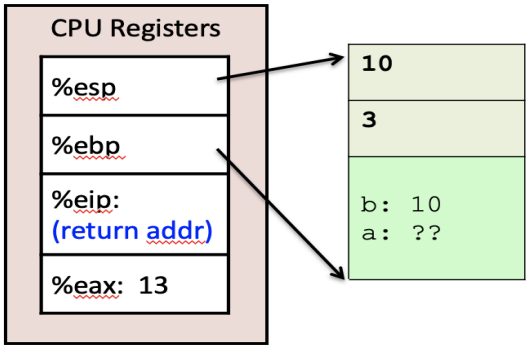
Example: translate to IA32 (cont)

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

Now we are back in main, what do we need to do?

(10) Get rid of parameter space on top of stack
(11) Store return value in a

```
main:
  # … assume some main code
  # and a at %ebp-8, b at %ebp-12
  pushl $3
  pushl -12(%ebp)
  call sum
```



Example: translate to IA32 (cont)

```
int main() {
    int a, b;
    b = 10;
    a = sum(b,3);
    printf("%d",a);
}
```

Now we are back in main, what do we need to do?

(10) Get rid of parameter space on top of stack
(11) Store return value in a

```
main:
  # … assume some main code
  # and a at %ebp-8, b at %ebp-12
  pushl $3
  pushl -12(%ebp)
  call sum
  addl $8, %esp
  movl %eax, -8(%ebp)
```