X86 Stack
Calling Function POV

Computer Systems Section 3.7

Stack Frame

Reg	Value
ebp	xFFFF FFF0
esp	xFFFF FFE0
eax	x0000 000E

What's In a Stack Frame?

- Caller's frame info (pointer to top of caller's frame)
- Space for saved state
- Space for Local Variable Values
- Space for arguments to lower level functions
- Return address (when calling functions)

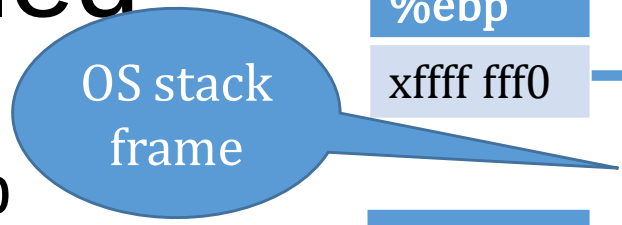
When I call someone…

- My stack frame is active
- I need to copy argument values to my stack frame
- I need to save state in my stack frame (more later)
- I need to store my return address in my stack frame
- I need to branch to called function's first instruction

Stack when main called

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```



%ebp
xffff fff0

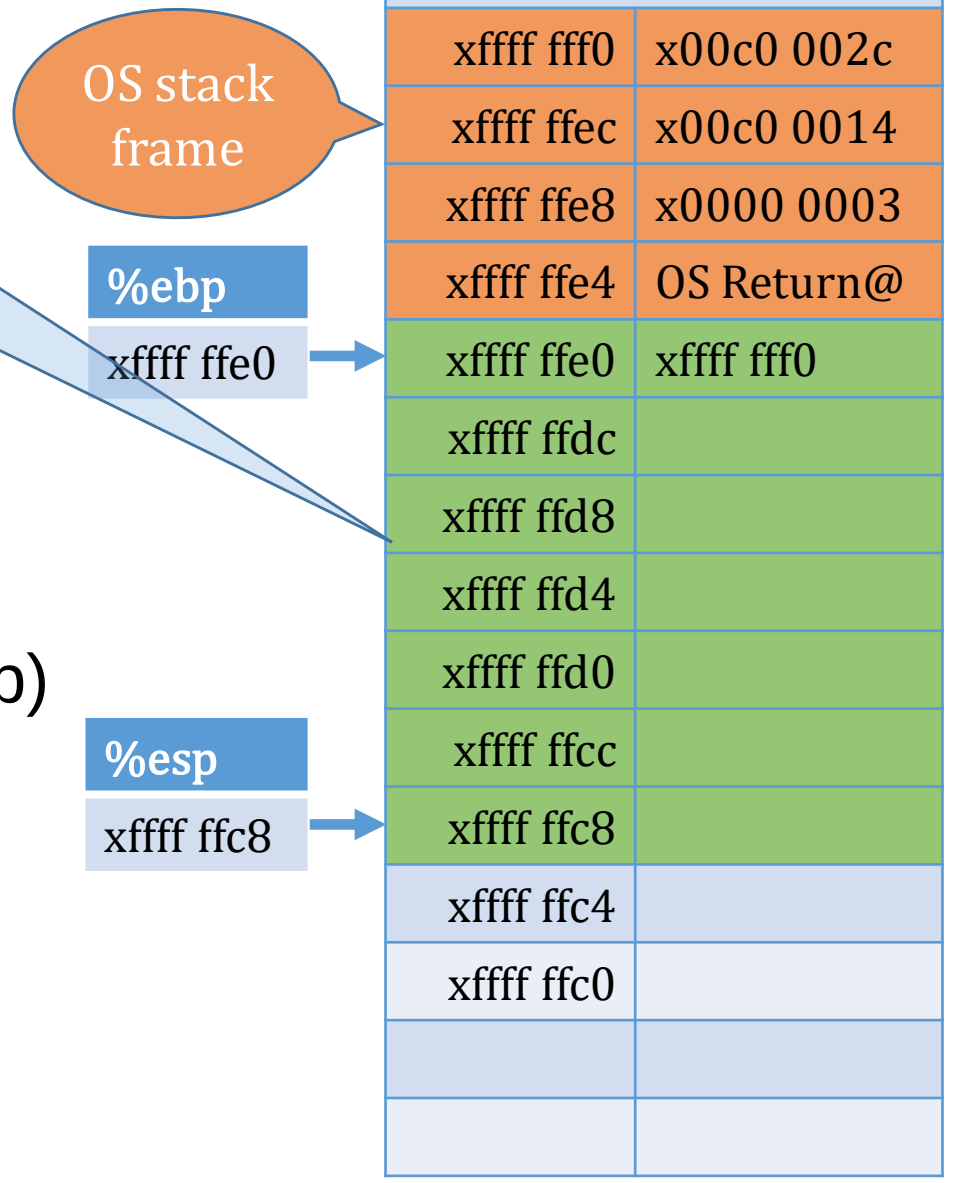
%esp
xffff ffe4

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	
xffff ffdc	
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	
xffff ffc8	
xffff ffc4	
xffff ffc0	

Main creates it's stack frame

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

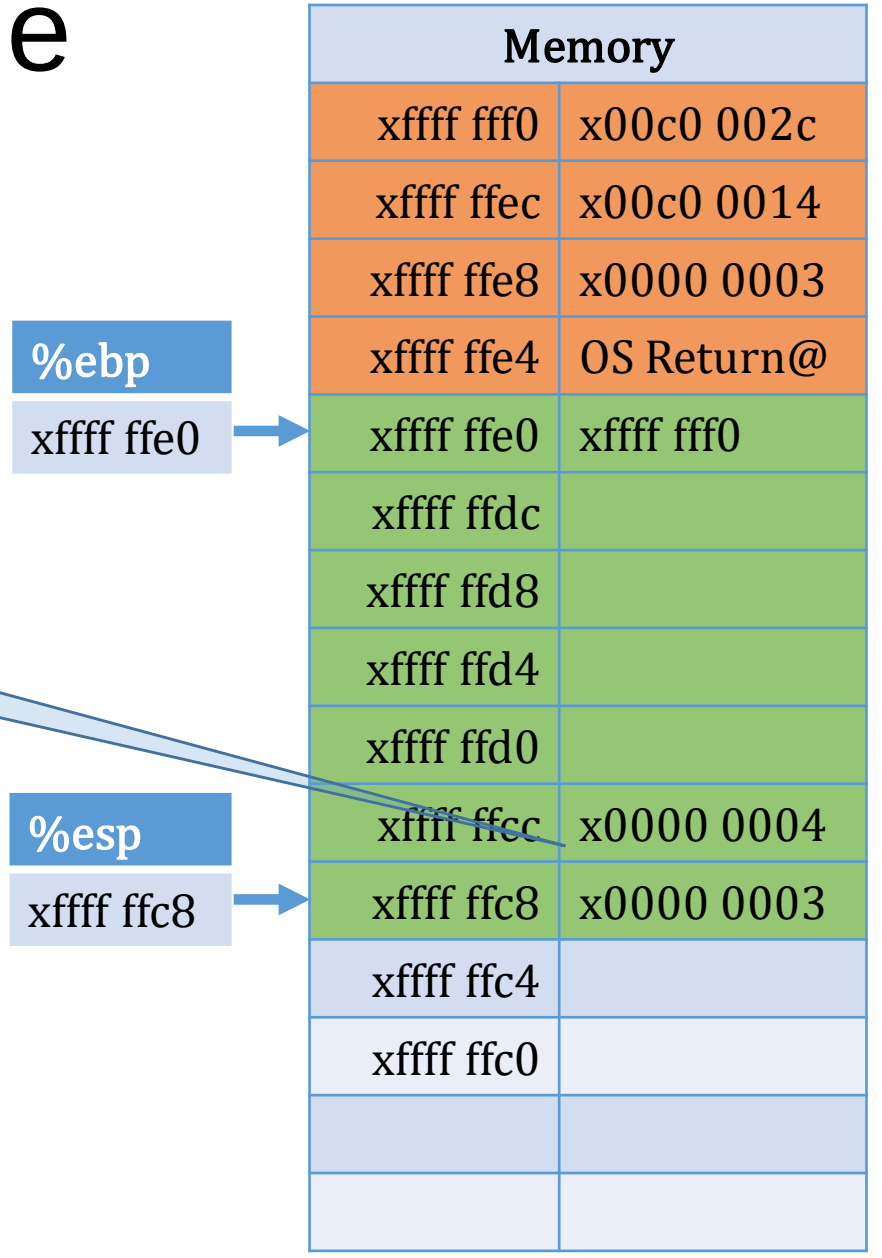
```
main:
pushl  %ebp
movl   %esp, %ebp
subl   $24, %esp
movl   $4, 4(%esp)
movl   $3, (%esp)
call   myfunc
movl   %eax, -4(%ebp)
movl   $0, %eax
leave
```



Copy args to bottom of frame

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```



x86 Argument Conventions

- Argument 1 goes at the bottom of the stack frame
- Argument 2 goes above argument 1
- Argument 3 goes above argument 2
- …

Question

- Why write arguments upside down? (1 on the bottom)

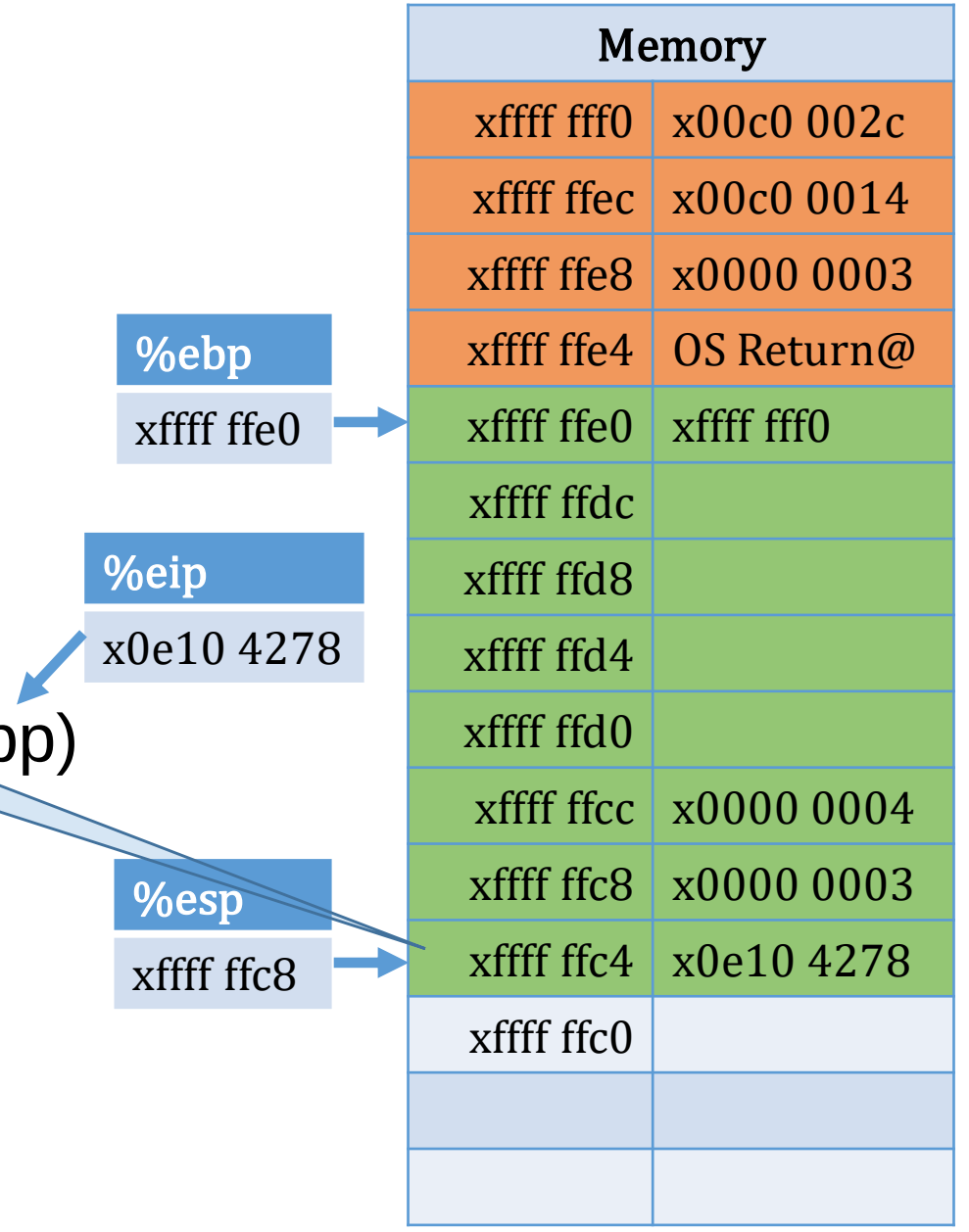
- Hint… Think about:
  printf("Student %s grade %d\n",sname,grade);

Save Return Address

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl %ebp
        movl  %esp, %ebp
        subl  $24, %esp
        movl  $4, 4(%esp)
        movl  $3, (%esp)
        call  myfunc
        movl  %eax, -4(%ebp)
        movl  $0, %eax
        leave
```

call myfunc:
push %eip
movl myfunc,%eip

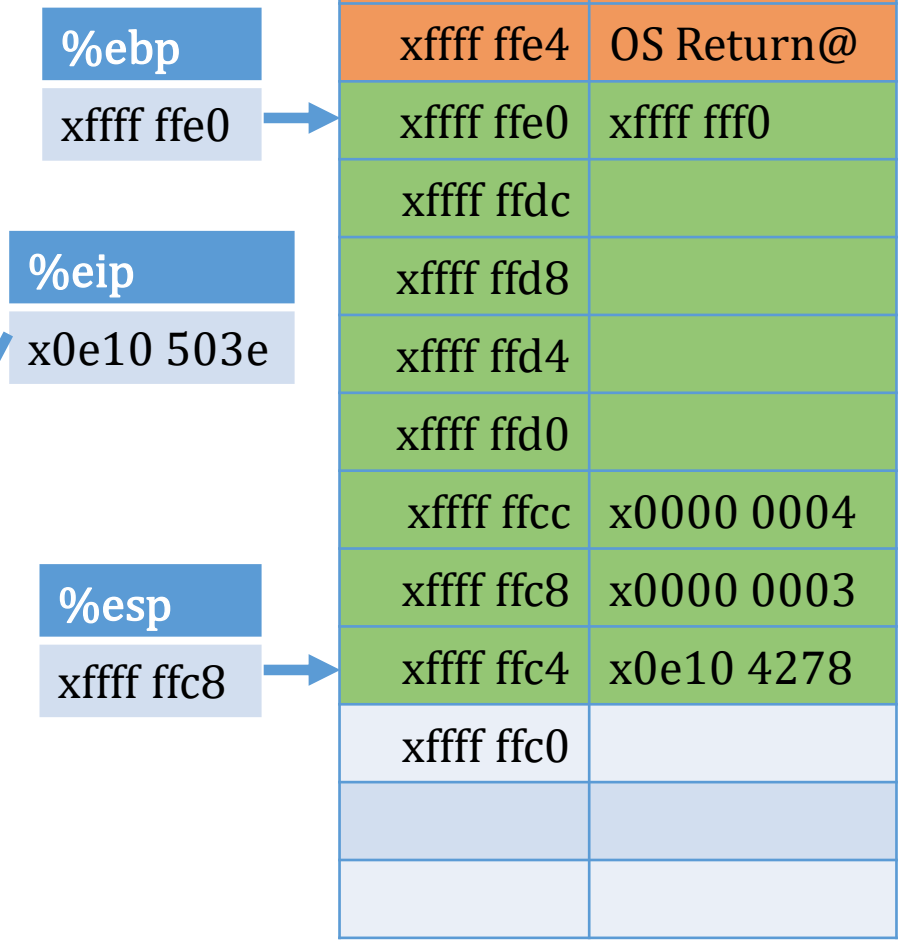


Branch to Called Function

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```

call myfunc:
push %eip
movl myfunc,%eip



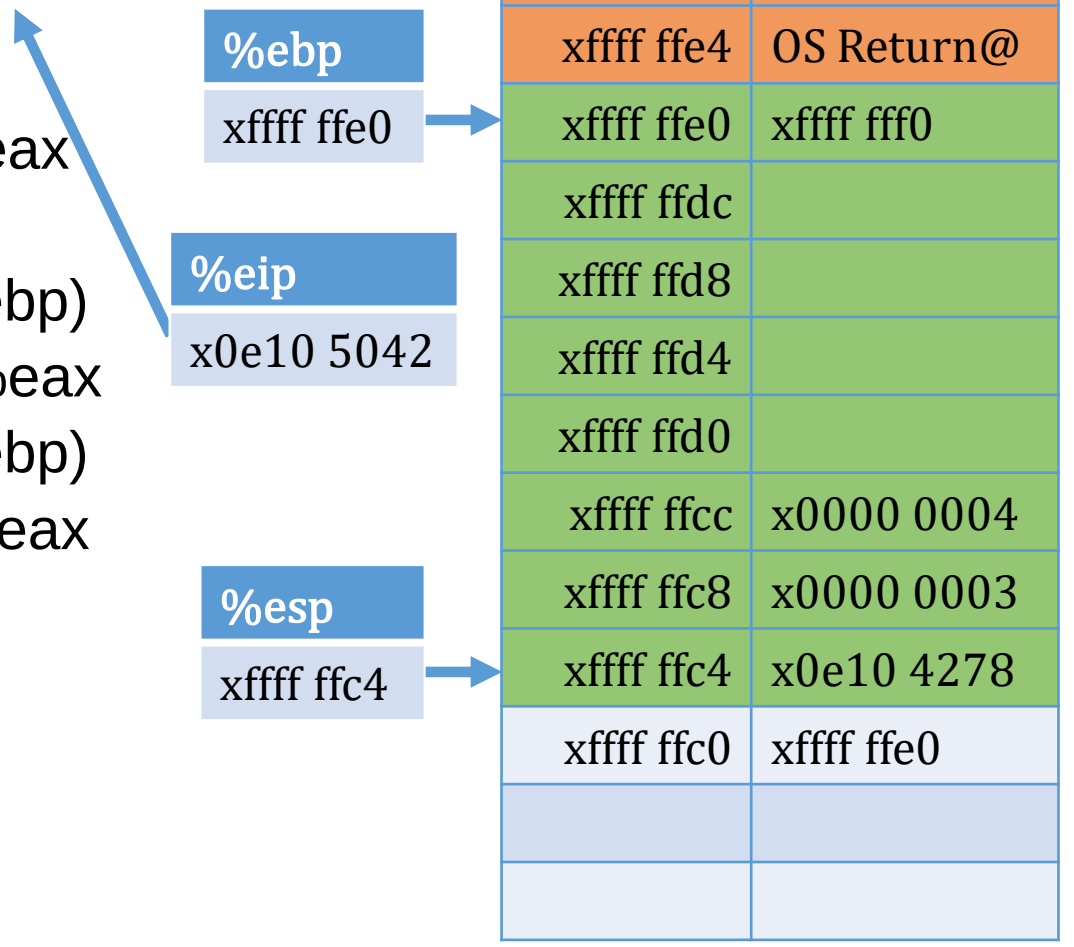
When I am called…

- My caller's stack frame is still active
- I need to save information about my callers frame
- I need to create my own stack frame

Branch to Caller Complete

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```



Myfunc Execution here.

Save return value

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

%eax
x0000 000a

%eax holds return value

%eip
x0e10 5042

%ebp
xffff ffc0

%esp
xffff ff??

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	xffff ffe0

Unwind Frame Stack

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

%eax holds return value

%eax
x0000 000a

%ebp
xffff ffe0

%eip
x0e10 505c

%esp
xffff ffc4

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	xffff ffe0

Return

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

%eax holds return value

%eax
x0000 000a

%ebp
xffff ffe0

%eip
x0e10 4278

%esp
xffff ffc8

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	xffff ffe0

After Return…

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl %ebp
        movl  %esp, %ebp
        subl  $24, %esp
        movl  $4, 4(%esp)
        movl  $3, (%esp)
        call  myfunc
        movl  %eax, -4(%ebp)
        movl  $0, %eax
        leave
```

%eax holds return value

%eax
x0000 000a

%ebp
xffff ffe0

%eip
x0e10 4278

%esp
xffff ffc8

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	

Handle Return Value

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```

%eax holds return value

%eax
x0000 000a

%ebp
xffff ffe0

%eip
x0e10 427a

%esp
xffff ffc8

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	x0000 000a
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	

Set Main's Return Value

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```

%eax holds return value

%eax
x0000 0000

%ebp
xffff ffe0

%eip
x0e10 427E

%esp
xffff ffc8

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	x0000 000a
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	

Unwind Main's Stack Frame

```
int main() {
  int ma;
  ma=myfunc(3,4);
  return 0;
}
```

```
main:
        pushl  %ebp
        movl   %esp, %ebp
        subl   $24, %esp
        movl   $4, 4(%esp)
        movl   $3, (%esp)
        call   myfunc
        movl   %eax, -4(%ebp)
        movl   $0, %eax
        leave
```

%ebp
xffff fff0

%esp
xffff ffe4

%eax
x0000 0000

Memory	
xffff fff0	x00c0 002c
xffff ffec	x00c0 0014
xffff ffe8	x0000 0003
xffff ffe4	OS Return@
xffff ffe0	xffff fff0
xffff ffdc	x0000 000a
xffff ffd8	
xffff ffd4	
xffff ffd0	
xffff ffcc	x0000 0004
xffff ffc8	x0000 0003
xffff ffc4	x0e10 4278
xffff ffc0	

X86 Calling Conventions

On Entry

- Save caller's base pointer
- Create my stack frame
- Initialize local variables

On Call

- Save Arguments
- Save Return Address
- Branch to Callee

On Return

- Use return value in %eax

On Exit

- Save return value in %eax
- Return my stack frame
- Restore caller's stack frame

Register Resources

- Single set of registers used by all functions
- When a caller invokes a callee…
  - Should it assume the called function will not change registers?
  - If callee can change registers, caller cannot assume values won't be changed.
  - IBM conventions...
    - Callee stores all register values on entry
    - Callee restores all register values before return
    - Wasteful – store all registers whether you need them or not
  - X86 conventions… some registers are managed by caller, some by callee…

Managing Registers

<table>
<tr><th>REG</th><th>Can I change?</th><th>Keeps Value over Calls?</th></tr>
<tr><td>%eax</td><td>Yes – used for return value</td><td>No – used for return value</td></tr>
<tr><td>%ecx</td><td rowspan="2">Yes – caller assumes these may change</td><td rowspan="2">No – called function may change these</td></tr>
<tr><td>%edx</td></tr>
<tr><td>%edi</td><td rowspan="3">No – Caller expects these to be unchanged</td><td rowspan="3">Yes – called function must keep these values</td></tr>
<tr><td>%esi</td></tr>
<tr><td>%ebx</td></tr>
<tr><td>%ebp</td><td colspan="2" rowspan="2">Managed by calling conventions which save/restore stack frames</td></tr>
<tr><td>%esp</td></tr>
</table>

Managing Registers

REG	Can I change?	Keeps Value over Calls?	Classification
%eax	Yes – used for return value	If I need these I must save before I call and restore on return	Caller-saved
%ecx	Yes – caller assumes these may change		
%edx			
%edi	If I use these, I must save caller's values on entry and restore on exit	Yes – called function must keep these values	Callee-Saved
%esi			
%ebx			
%ebp	Managed by calling conventions which save/restore stack frames		
%esp			

Managing Registers

- Use %eax, %ecx, %edx for localized values
- Use %edi, %esi, %ebx if you need values over calls

- If you use %edi, %esi, %ebx (callee-saved regs) save on entry, restore on exit

- If you need to save more reg values over calls, save whichever of %eax, %ecx, %edx you need (caller-saved regs) before you call, and restore after you call

X86 Calling Conventions (complete)

On Entry
- Push caller's ebp
- Push callee-saved regs if req.
- Create (data part) of my stack frame
- Initialize local variables

On Call
- Push caller-saved regs if req.
- Save Arguments
- Push Return Address
- Branch to Callee

On Return
- Pop caller-saved regs if req.
- Use return value in %eax

On Exit
- Save return value in %eax
- Return (data part of) stack frame
- Pop callee save regs if req.
- Pop caller's ebp
- Pop return address

Complete Stack Frame Contents

(%ebp)	**Caller's %ebp**	required!
	Caller's %esi	if I'm going to use %esi
	Caller's %ebx	if I'm going to use %ebx
	Caller's %edi	if I'm going to use %edi
	padding	So local vars start on address % by 16
	Local Variables	if needed
	….	
	My %eax	if I'm calling and need old value
	My %ecx	if I'm calling and need old value
	My %edx	if I'm calling and need old value
	Parameter n	if needed
	…	
	Parameter 1	
(%esp)	**Return Address**	if I'm calling

and $-16,%esp
-16=xFFFF FFF0