The examples and discussion in the following slides have been adapted from a variety of sources, including:

Chapter 3 of Computer Systems 2nd Edition by Bryant and O'Hallaron
x86 Assembly/GAS Syntax on WikiBooks
(http://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax)
Using Assembly Language in Linux by Phillip ??
(http://asm.sourceforge.net/articles/linasm.html)

The C code was compiled to assembly with gcc version 4.5.2 on Ubuntu Linux.

Unless noted otherwise, the assembly code was generated using the following command line:

```
gcc -S -m32 -O0 file.c
```

AT&T assembly syntax is used, rather than Intel syntax, since that is what the gcc tools use.
Stack Operations: push

The `push` instruction:
- decrements the stack pointer `esp` by 4, making room on the stack
- copies the value of its operand to the top of the stack

```
main:
    pushl %ebp
    ...
```

before

<table>
<thead>
<tr>
<th>esp</th>
<th>argv</th>
<th>argc</th>
<th>??</th>
</tr>
</thead>
</table>

not in the stack

current stack

after

<table>
<thead>
<tr>
<th>esp</th>
<th>value in ebp</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>esp</th>
<th>argv</th>
<th>argc</th>
<th>??</th>
</tr>
</thead>
</table>

current stack

not in the stack

value in ebp
The `pop` instruction:
- copies the item at the top of the stack into its operand
- increments the stack pointer `esp` by 4, removing the old top item

```
main:

... popl %ebp ...
```

**before**

```
esp

<table>
<thead>
<tr>
<th>argv</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
</tr>
<tr>
<td>??</td>
</tr>
<tr>
<td>value of ebp</td>
</tr>
</tbody>
</table>
```

**after**

```
esp

<table>
<thead>
<tr>
<th>argv</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
</tr>
<tr>
<td>??</td>
</tr>
</tbody>
</table>
```
static int max(int A, int B);

int main() {  // caller

    int x = 7;
    int y = 12;

    int retval = max(x, y);
    return 0;
}

int max(int A, int B) {  // called procedure

    int Bigger = A;
    if (A < B)
        Bigger = B;
    return Bigger;
}
Logical Steps in a Procedure Call

Calling a procedure (function) in C would seem to involve four steps:

1. set up the parameters that will be passed into the called procedure
2. cause execution to jump to the first instruction within the procedure
3. when the procedure is done, cause execution to jump back to the next instruction in the caller
4. access the value returned by the called procedure, if any

```c
// caller
int x, y;

... 
int retval = max(x, y);

... 

// called
int max(int A, int B) {
    int Bigger;
    ...
    return Bigger;
}
```
Preparing Parameters

The caller's stack frame (prior to the call) will logically reflect the current state of the caller's execution.

```
// caller
int x, y;
...
int retval = max(x, y);
...
```

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>x</td>
</tr>
<tr>
<td>ebp - 8</td>
<td>y</td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>esp</td>
<td>end caller frame</td>
</tr>
</tbody>
</table>

caller's frame before parameters are set
Preparing Parameters

Naturally we'll put the parameters being passed in onto the stack before making the call, something like this (note order of parameters on the stack):

```c
// caller
int x, y;
...
int retval = max(x, y);
...
```

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>x</td>
</tr>
<tr>
<td>ebp - 8</td>
<td>y</td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>esp + 4</td>
<td>passed value: y</td>
</tr>
<tr>
<td>esp</td>
<td>passed value: x</td>
</tr>
</tbody>
</table>

*caller's frame after parameters are set*
main:

... # usual ebp/esp contortions
andl $-16, %esp # make room on stack for
subl $32, %esp # locals and passed values
movl $7, 28(%esp) # x = 7
movl $12, 24(%esp) # y = 12
movl 24(%esp), %eax # eax = y
movl %eax, 4(%esp) # (esp + 4) = y
movl 28(%esp), %eax # eax = x
movl %eax, (%esp) # (esp) = x

...
We jump into the called procedure by using the assembly instruction `call`:

```plaintext
main:
    ...                   # this follows the code
    call max              # shown previously
    movl %eax, 20(%esp)
    ...                   # this follows the code
```

The `call` instruction has two effects:

1. Push the address of the next instruction in the caller onto the stack
2. Put the address represented by the symbol `max` into the PC
 Called Procedure Overview

max:

pushl %ebp    # stack setup within proc
movl %esp, %ebp
subl $16, %esp

movl 8(%ebp), %eax    # body of computation
movl %eax, -4(%ebp)
movl 8(%ebp), %eax
cmp 12(%ebp), %eax
jge .L3
movl 12(%ebp), %eax
movl %eax, -4(%ebp)

.L3:

movl -4(%ebp), %eax  # set return value

leave               # restore stack pointers
ret                 # jump back to caller
Local Stack Setup

```
max:
pushl %ebp          # save old frame ptr
movl %esp, %ebp    # set ebp to this frame
subl $16, %esp     # make room on stack for locals
...
```

The stack layout for `max()` shows:
- `ebp + 16` for caller
- `ebp + 12` passed value: B
- `ebp + 8` passed value: A
- `ebp + 4` return-to address
- `ebp` old frame ptr
- `ebp - 4` local: Bigger
- `esp` ???

The stack layout for `main()` shows:
- `ebp + 16` for caller
- `ebp + 12` passed value: B
- `ebp + 8` passed value: A
- `ebp + 4` return-to address
- `ebp` old frame ptr
- `ebp - 4` local: Bigger
- `esp` ???

The stack layout for `max()` shows:
- `ebp + 16` for caller
- `ebp + 12` passed value: B
- `ebp + 8` passed value: A
- `ebp + 4` return-to address
- `ebp` old frame ptr
- `ebp - 4` local: Bigger
- `esp` ???

The stack layout for the main program shows:
- `ebp + 16` for caller
- `ebp + 12` passed value: B
- `ebp + 8` passed value: A
- `ebp + 4` return-to address
- `ebp` old frame ptr
- `ebp - 4` local: Bigger
- `esp` ???
The Computations

max:
  ...
  movl 8(%ebp), %eax  # retrieve A from stack
  movl %eax, -4(%ebp) # set Bigger to A
  movl 8(% ebp), %eax
  cmpl 12(%ebp), %eax # compare A to B (on stack)
  jge .L3             # A >= B, ready to return
  movl 12(%ebp), %eax # else, get B from stack
  movl %eax, -4(%ebp) # set Bigger to B

.L3:
  ...

The Stack

ebp + 12 passed value: B
ebp + 8 passed value: A
ebp + 4 return-to address
ebp old frame ptr
ebp - 4 local: Bigger
Returning a Value

We use the `eax` register to hold the return value:

```assembly
max:
    ...  
    movl  -4(%ebp), %eax # set return value to Bigger 
    ...  
```
Preparing to Leave

The `leave` instruction resets the stack and frame pointers prior to returning:

```
max:
  ...
  leave
  ...
```

- `ebp` must be reset to point to the beginning of the stack frame of the caller.
- `esp` must be reset to point to where the top of the stack was when the call instruction was made.

**The Stack**

```
        ebp  local
  ebp - 4    stuff
  ebp - 8    for
  ebp - 12   caller
          ...
  esp + 8    passed value: y
  esp + 4    passed value: x
  esp        return-to address
```
Preparing to Leave

---

**Before**

- **ebp + 16**
  - *older frame ptr*
  - `x`
  - ...
- **ebp + 12**
  - *passed value: B*
- **ebp + 8**
  - *passed value: A*
- **ebp + 4**
  - *return-to address*
- **ebp**
  - *old frame ptr*
- **ebp - 4**
  - *local: Bigger*
- **esp**
  - ??
  - ??

---

**After**

- **ebp - 4**
  - *older frame ptr*
- **ebp - 8**
  - `y`
- **ebp - 12**
  - `x`

---

Assembly:

```
1 movl %ebp, %esp  # 1
2 popl %ebp        # 2
```
We jump back by using the assembly instruction `ret`:

```assembly
max:
    ... ret ...
```

### Before

<table>
<thead>
<tr>
<th>ebp</th>
<th>older frame ptr</th>
<th>ebp - 4</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 8</td>
<td>y</td>
<td>esp + 12</td>
<td></td>
</tr>
<tr>
<td>esp + 8</td>
<td>passed value: y</td>
<td>esp + 8</td>
<td></td>
</tr>
<tr>
<td>esp + 4</td>
<td>passed value: x</td>
<td>esp + 4</td>
<td></td>
</tr>
<tr>
<td>esp</td>
<td>return-to address</td>
<td>esp</td>
<td>passed value: x</td>
</tr>
</tbody>
</table>

### After

<table>
<thead>
<tr>
<th>ebp</th>
<th>older frame ptr</th>
<th>ebp - 4</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 8</td>
<td>y</td>
<td>esp + 8</td>
<td></td>
</tr>
<tr>
<td>esp + 8</td>
<td>passed value: y</td>
<td>esp</td>
<td></td>
</tr>
<tr>
<td>esp + 4</td>
<td>passed value: x</td>
<td>esp</td>
<td>passed value: x</td>
</tr>
<tr>
<td>esp</td>
<td>return-to address</td>
<td>esp</td>
<td>passed value: x</td>
</tr>
</tbody>
</table>

---

*CS@VT Computer Organization I ©2005-2012 McQuain*
## Stack Summary

**Stack before parameters are set, frame for `main()`**

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>x</td>
</tr>
<tr>
<td>ebp - 8</td>
<td>y</td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>ebp - 16</td>
<td></td>
</tr>
<tr>
<td>ebp - 20</td>
<td></td>
</tr>
<tr>
<td>ebp - 24</td>
<td></td>
</tr>
<tr>
<td>ebp - 28</td>
<td></td>
</tr>
<tr>
<td>esp</td>
<td></td>
</tr>
</tbody>
</table>

**Passed value:** y

**Stack after parameters are set, frame for `main()`**

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>x</td>
</tr>
<tr>
<td>ebp - 8</td>
<td>y</td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>ebp - 16</td>
<td></td>
</tr>
<tr>
<td>ebp - 20</td>
<td></td>
</tr>
<tr>
<td>ebp - 24</td>
<td></td>
</tr>
<tr>
<td>ebp - 28</td>
<td>passed value: y</td>
</tr>
<tr>
<td>esp</td>
<td>passed value: x</td>
</tr>
</tbody>
</table>
### Stack Summary

#### Stack after execution of the call instruction

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame ptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>x</td>
</tr>
<tr>
<td>ebp - 8</td>
<td>y</td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>ebp - 16</td>
<td></td>
</tr>
<tr>
<td>ebp - 20</td>
<td></td>
</tr>
<tr>
<td>ebp - 24</td>
<td></td>
</tr>
<tr>
<td>ebp - 28</td>
<td>passed value: y</td>
</tr>
<tr>
<td>esp</td>
<td>return-to address</td>
</tr>
</tbody>
</table>

#### Stack after execution of stack setup code in `max()`

<table>
<thead>
<tr>
<th>ebp</th>
<th>old frame pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp - 4</td>
<td>local: Bigger</td>
</tr>
<tr>
<td>ebp - 8</td>
<td></td>
</tr>
<tr>
<td>ebp - 12</td>
<td></td>
</tr>
<tr>
<td>esp</td>
<td></td>
</tr>
</tbody>
</table>
## Stack Summary

<table>
<thead>
<tr>
<th>ebp</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>older frame ptr</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>y</td>
</tr>
<tr>
<td>esp</td>
<td>passed value: y</td>
</tr>
<tr>
<td></td>
<td>passed value: x</td>
</tr>
<tr>
<td></td>
<td>return-to address</td>
</tr>
</tbody>
</table>

Stack after execution of `leave` instruction in `max()`
### Stack Summary

<table>
<thead>
<tr>
<th>ebp</th>
<th></th>
<th></th>
<th></th>
<th>stack after execution of ret instruction in <code>max()</code>, execution now proceeds in <code>main()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>passed value: y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>passed value: x</td>
</tr>
<tr>
<td>older frame ptr</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>y</td>
</tr>
</tbody>
</table>

Stack after execution of `ret` instruction in `max()`, execution now proceeds in `main()`.