Credits and Disclaimers

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

Chapter 3 of Computer Systems 3rd 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 4.8.3 on CentOS 7.

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

```
 gcc -S -m64 -fno-asynchronous-unwind-tables -O0 file.c
```

AT&T assembly syntax is used, rather than Intel syntax, since that is what the gcc tools use.
The **push** instruction:
- decrements the stack pointer `rsp` by 8, making room on the stack
- copies the value of its operand to the top of the stack

```assembly
main:
        pushq   %rbp
        ...
```

**before**

```
 rsp
 ??
 ??
 ??
```

**current stack**

```
 ??
 ??
 ??
```

**not in the stack**

```
 ??
 ??
 ??
```

**after**

```
 rsp
 "value in rbp"
 ??
 ??
 ??
```
The `pop` instruction:
- copies the item at the top of the stack into its operand
- increments the stack pointer `rsp` by 8, removing the old top item
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

---

// 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.

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

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

caller's frame before parameters are set
Preparing Parameters

The first 6 parameters are passed in registers, parameters 7 and up are passed on the stack. While there’s are only 2 parameters in this example, note order of (unneeded) parameters on the stack:

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

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

### Registers

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rdi</td>
<td>x</td>
</tr>
<tr>
<td>rsi</td>
<td>y</td>
</tr>
<tr>
<td>rdx</td>
<td>??</td>
</tr>
<tr>
<td>rcx</td>
<td>??</td>
</tr>
<tr>
<td>r8</td>
<td>??</td>
</tr>
<tr>
<td>r9</td>
<td>??</td>
</tr>
</tbody>
</table>

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

```
rbp   old frame ptr
rbp - 4 x:  7
rbp - 8 y: 12
rbp - 12

passed value:
argument n

rsp   passed value:
argument 7

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

    # usual rbp/rsp contortions make room on
    # stack for locals and passed values

    subq  $16, %rsp

    movl  $7, -4(%rbp)
    movl  $12, -8(%rbp)

    movl  -8(%rbp), %edx
    movl  -4(%rbp), %eax

movl  %edx, %esi
movl  %eax, %edi

...
Jumping In

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

```assembly
main:
    ...                   # this follows the code
    # shown previously
    movl    %edx, %esi
    movl    %eax, %edi
    call    max
    movl    %eax, -12(%rbp))
    ...
```

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

The Stack:
- `rbp` - 4 local
- `rbp` - 8 stuff
- `rbp` - 12 for
- caller
- ... ??
- `rsp` + 8 ??
- `rsp` return-to address
Called Procedure Overview

max:
    pushq  %rbp                      # stack setup
    movq  %rsp, %rbp
    subq  $24, %rsp

    movl  %edi, -20(%rbp)           # function body
    movl  %esi, -24(%rbp)
    movl  -20(%rbp), %eax
    movl  %eax, -4(%rbp)
    movl  -20(%rbp), %eax
    cmpl  -24(%rbp), %eax
    jge   .L4
    movl  -24(%rbp), %eax
    movl  %eax, -4(%rbp)

.L4:
    movl  -4(%rbp), %eax            # set return value

    leave                           # jump back to caller
    ret
Local Stack Setup

\[
\text{max:} \quad \begin{align*}
\text{pushq} & \quad \%\text{rbp} & \quad \# \text{ save old frame ptr} \\
\text{movq} & \quad \%\text{rsp}, \%\text{rbp} & \quad \# \text{ set rbp to this frame} \\
\text{subq} & \quad $24, \%\text{rsp} & \quad \# \text{ make room for locals} \\
\text{movl} & \quad \%\text{edi}, -20(\%\text{rbp}) & \quad \# \text{ move parameters} \\
\text{movl} & \quad \%\text{esi}, -24(\%\text{rbp}) & \\
\ldots
\end{align*}
\]
The Computations

max:

...  
movl  -20(%rbp), %eax
movl  %eax, -4(%rbp)
movl  -20(%rbp), %eax
cmpl  -24(%rbp), %eax
jge   .L4
movl  -24(%rbp), %eax
movl  %eax, -4(%rbp)

.L4:

<table>
<thead>
<tr>
<th>rbp + 4</th>
<th>return-to address</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp</td>
<td>old frame ptr</td>
</tr>
<tr>
<td>rbp - 4</td>
<td>local: Bigger</td>
</tr>
<tr>
<td>rbp - 20</td>
<td></td>
</tr>
<tr>
<td>rbp - 24</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>y</td>
</tr>
</tbody>
</table>

the Stack
Returning a Value

We use the eax (or %rax) register to hold the return value:

max:

```
  . . .
  movl  -4(%rbp), %eax  # set return value to Bigger
  . . .
```
Preparing to Leave

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

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

- `rbp` must be reset to the beginning of the stack frame of the caller.

- `rsp` must be reset to where the top of the stack was when the call instruction was made.

```
<table>
<thead>
<tr>
<th>rbp</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp - 4</td>
<td>stuff</td>
</tr>
<tr>
<td>rbp - 8</td>
<td>for</td>
</tr>
<tr>
<td>rbp - 12</td>
<td>caller</td>
</tr>
<tr>
<td>rsp</td>
<td>return-to address</td>
</tr>
</tbody>
</table>
```

the Stack
Preparing to Leave

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp + 8</td>
<td>return-to address</td>
</tr>
<tr>
<td>ebp</td>
<td>old frame ptr</td>
</tr>
<tr>
<td>ebp - 4</td>
<td>local: Bigger</td>
</tr>
<tr>
<td>rsp</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp</td>
<td>x</td>
</tr>
<tr>
<td>rbp - 4</td>
<td>y</td>
</tr>
<tr>
<td>rbp - 8</td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>??</td>
</tr>
</tbody>
</table>

movl %rbp, %rsp  # 1
popl %rbp        # 2
Jumping Back

We jump back by using the assembly instruction `ret`:

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

```
popq %eip
```

<table>
<thead>
<tr>
<th>rbp</th>
<th>older frame ptr</th>
<th>return-to address</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp - 4</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>rbp - 8</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rsp</td>
<td>return-to address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rbp</th>
<th>older frame ptr</th>
<th>return-to address</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp - 4</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>rbp - 8</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rsp + 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rsp + 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rsp</td>
<td>return-to address</td>
</tr>
</tbody>
</table>

before | after
Stack Summary

Stack before execution of the call instruction
### Stack Summary

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp</td>
<td>old frame ptr</td>
</tr>
<tr>
<td>rbp - 4</td>
<td>x</td>
</tr>
<tr>
<td>rbp - 8</td>
<td>y</td>
</tr>
<tr>
<td>rbp - 12</td>
<td>??</td>
</tr>
<tr>
<td>rbp - 16</td>
<td>??</td>
</tr>
<tr>
<td>rbp - 20</td>
<td>??</td>
</tr>
<tr>
<td>rbp - 24</td>
<td>??</td>
</tr>
<tr>
<td>rbp - 28</td>
<td>??</td>
</tr>
<tr>
<td>rsp</td>
<td>return-to address</td>
</tr>
</tbody>
</table>

**Stack after execution of the call instruction**

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbp</td>
<td>old frame pointer</td>
</tr>
<tr>
<td>rbp - 4</td>
<td>local: Bigger</td>
</tr>
<tr>
<td>rbp - 20</td>
<td>x</td>
</tr>
<tr>
<td>rbp - 24</td>
<td>y</td>
</tr>
<tr>
<td>rbp</td>
<td>. . .</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>older frame ptr</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stack Summary

stack after execution of leave instruction in max()
Stack Summary

<table>
<thead>
<tr>
<th>rbp</th>
<th>. . .</th>
<th>stack after execution of ret instruction in max(), execution now proceeds in main()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>older frame ptr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>rsp</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>