Assembly Language – Calling Conventions

• Learning Objectives
  • Define stack frame
  • Explain how the assembler sets up the stack for execution of a function.
  • Locate parameters and local variables on the stack.
Invoking Functions

• In certain very simple cases, you can just jump to a function address (but this is quite unusual).

• Consider the function:
  
  ```c
  extern void g(void);
  
  void f(void) {
    g();
  }
  ```

• After we execute g, there is nothing left to be done in function f; therefore, transferring control to g via a simple jump instruction works.
Use of jmp function of context

- Note that the ability to use a jmp to invoke a function is a product of the context, not the function being called.

```c
extern void g(void);

void f(void) {
    g();
    g();
    g();
    g();
}
```

- The first two instances of calls to g require that control return to a specific point in function f.
Given the sum function below, it would it be OK to `jmp` to `sum` rather than invoking a regular call and return?

- True
- False

```
extern int sum(int a, int b);

int f(int a, int b) {
    return sum(a, b);
}
```
What if we turn off the optimizer?

```
pushl  %ebp
movl  %esp, %ebp
subl  $24, %esp
movl  12(%ebp), %eax
movl  8(%ebp), %ecx
movl  %ecx, -4(%ebp)
movl  %eax, -8(%ebp)
movl  -4(%ebp), %eax
movl  -8(%ebp), %ecx
movl  %eax, (%esp)
movl  %ecx, 4(%esp)
calll  sum
addl  $24, %esp
popl  %ebp
ret
```
Scribble

```
main
  p2
  p1
  n args
  Return addr
  ebp

esp

Call f

Parameters

f, takes over

stack frame for f

locals
```
Calling Conventions

• The way the compiler has agreed to use the stack, registers and functions to enable functional decomposition (and separate compilation).

• Registers are divided into two sets:
  - Callee saved: the caller assumes that the contents of these registers will be unchanged when the called functions return.
    • Implication: If the callee uses the registers, the callee must save them and restore them.
    • esp, ebx, ebp, esi, edi
  - Caller saved: the caller assumes that these registers could be lost in the called function.
    • Implication: The callee can use these registers any way it wants without having to restore them.
    • eax, ecx, edx
The Caller Side

- Save any registers necessary.
- Push arguments on the stack.
- Call the function
  - Push the return address on the stack
  - Jump to the function
The Callee Side

- Save the frame pointer (ebp)
- Set the frame pointer to the current top of stack.
- Adjust stack pointer to make space for the stack frame
  - Leave space for all the local variables.
  - Maintain required alignment of stack frames.
- Inside the function:
  - Parameters are positive offsets from ebp.
  - Locals are typically negative offsets from the ebp.
Summing Up

- Caller must save caller-saved registers it is using.
- Callee must save callee-saved registers it intends to use.
- Caller pushes arguments and return address.
- Callee creates (aligned) stack frame.
- Arguments are positive offsets from frame pointer.
- Locals are negative offsets from frame pointer.