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 in registers and 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:

```
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.

Screen Capture

- Tailcall.c: see how the compiler turns a function call into a jump statement.

Use of `jmp` is a function of context

- Note that the ability to use a `jmp` to invoke a function is a product of the context in which the function is being called.

```
extern void g(void);

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

Use of `jmp` is a function of context

- Note that the ability to use a `jmp` to invoke a function is a product of the context in which the function is being called.

```
extern void g(void);

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

- The first two instances of calls to g require that control return to a specific point in function `f`.

Screen Capture

- Tailcall1.c: only the last instance of g is a tailcall.
- Tailcall2.c: What if we have printf instead of g?
- Tailcall3.c: What if we call a function with the same parameters?
- Tailcall4.c: Let's turn optimization off.

Screen Capture

- Tailcall2.c

Screen Capture

- We examined several variants of Tailcall3.c:
  - What if we cast the return value of sum to a long and return it?
  - What if we cast the return value of sum to a long, but don't return it.
  - Takeaway: be able to look at C and determine if the code can use a tailcall.

What if we turn off the optimizer?

```
f:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
        movl    %edx, %esi
        movl    %eax, %edi
        call    sum
        leave
        ret
```

Calling Conventions

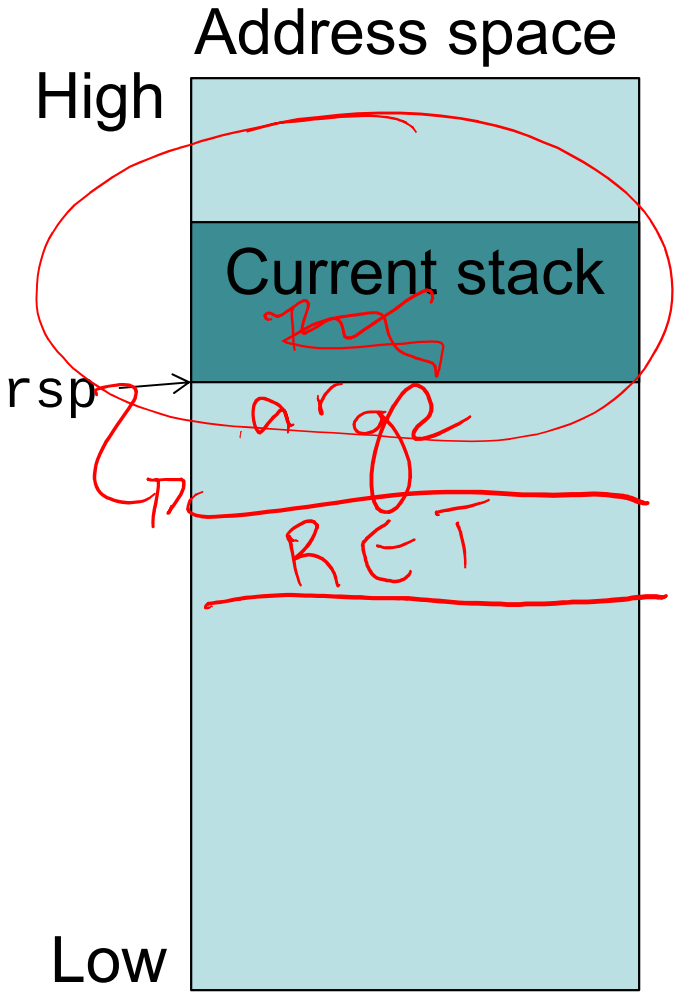
- The way the compiler has agreed to use the stack, registers and functions to enable functional decomposition (and separate compilation).
- Registers are divided into two sets:
  - Callee saved: the caller assumes that the contents of these registers will be unchanged when the called functions return.
    - Implication: If the callee uses the registers, the callee must save them and restore them.
    - `%rbx, %rbp, %r12-%r15`
  - 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.
    - `(the rest): %rax, %rcx, %rdx, %rdi, %rsi, %r8-%r11`

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.
    - `%rbx, %rbp, %r12-%r15, %rsp`
  - 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.
    - `(the rest): %rax, %rcx, %rdx, %rdi, %rsi, %r8-%r11`

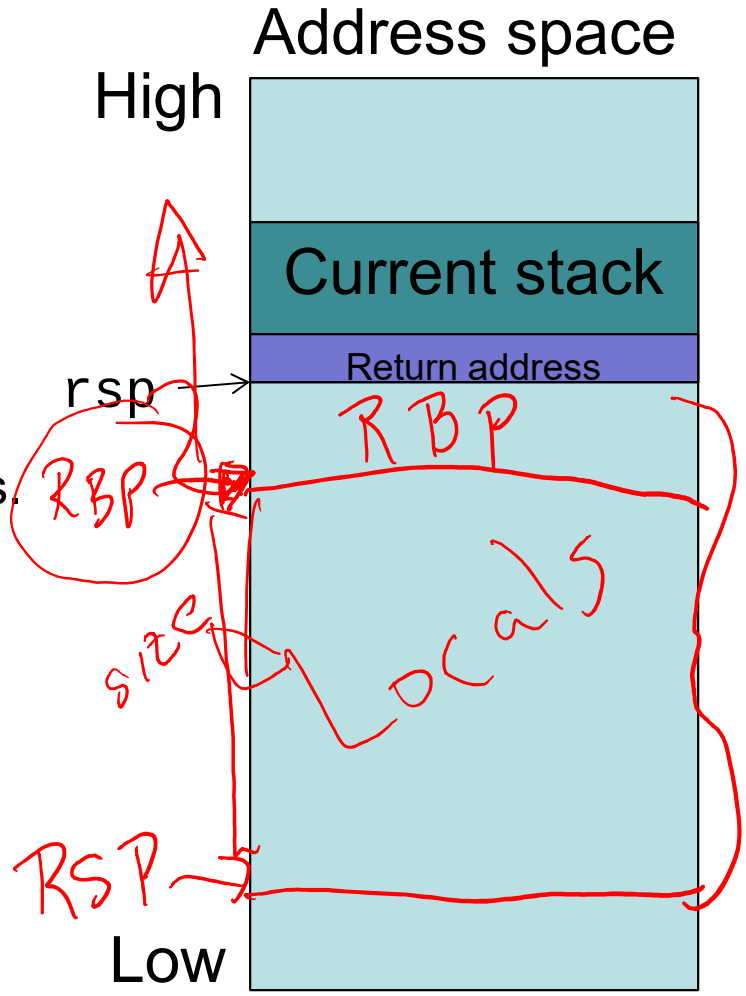
The Caller Side

- Save any registers necessary.
- Put arguments in registers (or on the stack).
- Call the function
  - Put the return address on the stack
  - Jump to the function



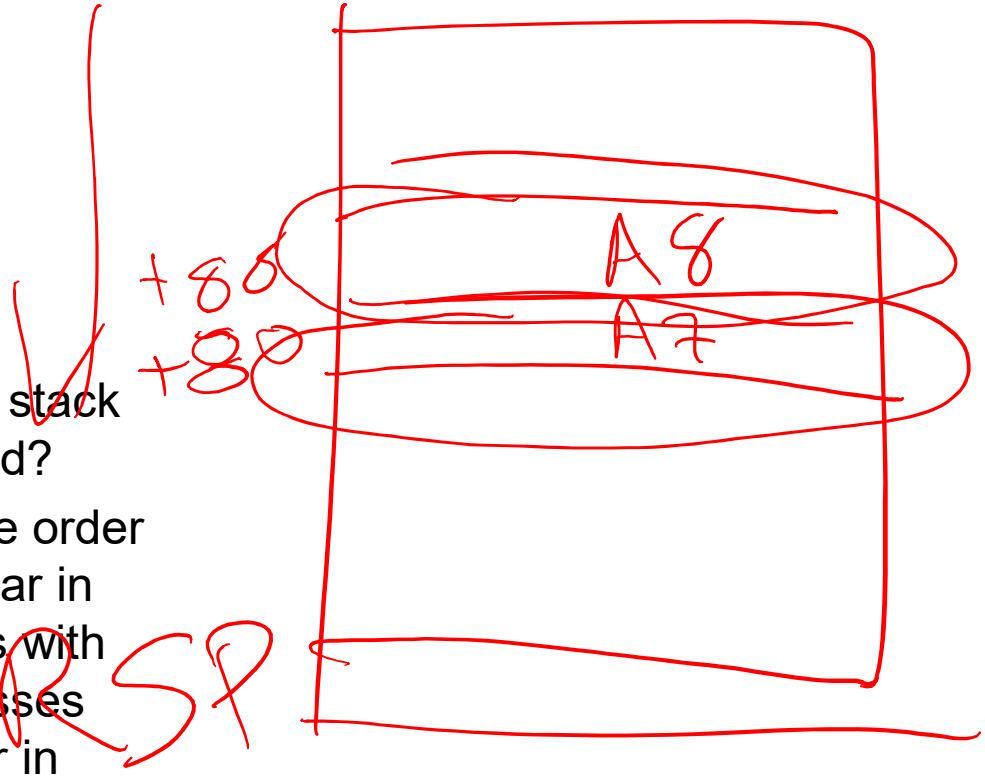
The Callee Side

- Save the frame pointer (`rbp`)
- 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:
  - Stack parameters are positive offsets from `rbp`.
  - Locals are typically negative offsets from the `rbp`.



Screen Capture

- fib.[cs]: -O0
- fib1.[cs] –O1
- fib.2.[cs] –O3
- manyargs.[cS]:
  - In what order are stack arguments passed?
  - Pushed in reverse order so that they appear in memory locations with increasing addresses (e.g., they appear in order)



Summing Up

- Caller must save caller-saved registers it is using.
- Callee must save callee-saved registers it intends to use.
- Caller places arguments in registers/on stack, calls procedure, placing return address on stack.
- Callee creates (aligned) stack frame.
- Arguments on the stack are positive offsets from frame pointer.
- Locals are negative offsets from frame pointer.