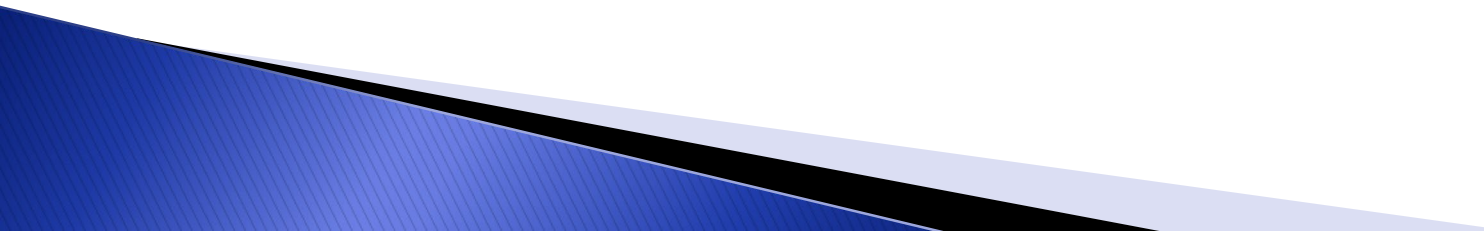


Fibonacci in assembler

Recursion and register allocation

Recall register allocation rules

- ▶ Leaf functions have free use of parameter registers, r10, and r11 (all caller saved) and should use them preferentially
 - ▶ Functions that call other functions, particularly those that have more than one call, tend towards callee-saved registers (rbx, r12–r15)
 - ▶ All functions tend to use rbp and rsp for their stack frame
 - ▶ All functions tend to use rax for their return value
- 

Fibonacci

- ▶ The general case computation is:
 - $F(n) = F(n-1) + F(n-2)$
- ▶ This puts pressure on rax: both recursive calls will set it and the function itself needs to set it.
- ▶ This puts pressure on rdi:
 - The function doesn't use rdi directly in the general case
 - The first recursive call (n-1) will trash rdi
 - We need the value in rdi to set the parameter of the second recursive call (n-2)

Recursion in general

- ▶ Recursive methods typically *cannot* rely on the parameter and caller saved registers because their recursive child instances will also rely on those very same registers – it's the same code!
- ▶ So we need to selectively use callee-saved registers

Back to Fibonacci – general case

▶ RAX:

- We need to save 1 partial result
- We can add the saved partial result to the second result to get the function's return value

▶ RDI:

- We don't need n in the general case, but we do need $n-1$ and $n-2$
- Save $n-1$ before we make the first recursive call
- Decrement and use to make the second recursive call

Fibonacci – end case

- ▶ If n is less than 2, we return n
- ▶ No need to save anything at all

Fibonacci in C with assembler leanings

```
unsigned long f(unsigned long n)
{
    unsigned long rv;

    rv = n;
    if(n >= 2)
    {
        unsigned long partial, param;
        param = n-1;
        partial = f(param);
        param -=1;
        rv = f(param);
        rv += partial;
    }
    return rv;
}
```