

# Register Allocation


What can be used?

When do you save?

# Linux uses System V ABI

- ▶ The first six integer or pointer arguments are passed in registers RDI, RSI, RDX, RCX, R8, R9
- ▶ RAX is used for return values
- ▶ RSP (stack pointer) must be restored when control is returned
- ▶ If the callee wishes to use registers RBX, RBP, and R12–R15, it must restore their original values before returning control to the caller.
- ▶ All other registers must be saved by the caller if it wishes to preserve their values.

# So which registers do we use?

- ▶ Depends on what our function is passed
  - ▶ Depends on what registers our function uses
  - ▶ Depends on what our function calls
- 

# Goal – Performance!

Maximize performance by minimizing data motion

Maximize performance by minimizing memory accesses

- ▶ Only when doing a copy is data motion useful; otherwise it is overhead that enables doing the useful work
- ▶ The most common way of minimizing memory accesses is to work in registers as much as we can
  - Registers are the highest level of cache and the fastest storage on the machine
  - Data your code references repeatedly belongs in a register
  - If it only gets touched one time, maybe leave it in memory and use the addressing modes effectively

# All functions

Have to save and restore any of these if it modifies them:

- ▶ RBP: we probably were using as part of the stack frame and planned to restore it anyway
- ▶ RSP: if we don't restore it, the program will probably crash. Assume all functions deal with RSP correctly.
- ▶ RBX, R12 – R15: Have to save these before we use them

# Leaf routines


- ▶ Leaf routines make no calls
- ▶ Can freely use RDI, RSI, RDX, RCX, R8, R9 even when passed fewer than 6 parameters
- ▶ Can use freely use R10 and R11, since they are caller-saved
- ▶ Can use freely use RAX as long as it fills in the return value when done
- ▶ *Should only resort to using callee-saved registers when it runs out of caller-saved registers*

# Functions that call other functions

Trade-offs to be made:

- ▶ If we make function calls, we have to save and restore RAX and any of the parameter registers as well as R10–R11 before and after *each* call if we still want to use the values they had prior to the call. This is expensive if we do it repeatedly in a loop.
- ▶ If we use RBX, RBP, R12–R15 we only have to save them one time (at the beginning of our function) and restore them one time (at the end of our function). This is cheaper than repeatedly saving the caller-saved registers.

# Those were guidelines

- ▶ The actual measure is going to be total number of times memory is accessed (lower is better)
  - ▶ The guidelines for leaf-level functions are always correct; use cheap registers before using expensive registers
  - ▶ The guidelines for functions that call other functions are predicated on certain assumptions that need to be verified
- 



# What are the costs?

- ▶ Using RBX and R12–R15 costs a function two memory accesses per register (save the old value and restore it).
  - *Memory accesses are expensive (at least 4x more than register transfers)*
- ▶ Giving RBX or any of R12–15 a new value, say from a parameter register, costs a register to register transfer.
  - *Register to register transfers are cheap, but not free (1 cycle)*
  - *Register to register transfers are far, far cheaper than touching memory*

# Don't pay unless you have to

- ▶ If a function only makes a single function call one time, it is slightly cheaper to save and restore the parameter registers before and after the call than it is to save a callee-saved register, overwrite it, and later on restore it.
  - ▶ If a function no longer cares about the caller-saved registers when it starts calling other functions, it might not have any need to save their values in the callee-saved registers
- 