Lecture notes on compiling C with Clang

November 10, 2015

[15/10/28]
Clang: compiler LLVM: infrastructure
Read intro to LLVM
In x86 we say base pointer, but the general term is frame pointer
Automatic variables vs local variables?
static link works as an extra parameter for languages that have it
One frame per function call, not per function
Clang stack frame:
parameters slots are further down; we only use them if we need to local vars allocated when running and placed at the end
[15/11/02]
Stack frame in x86: - Why is return addr passed first? Because we use registers for parameters sometimes (only if necessary) - Local variables furthest down/most recent because they are allocated by the function we call
Read LLVM calling conventions link
Clang function idiom: L1. Push base pointer L2. Adjust stack pointer (if used) L3. Pop BP L4. RET pops return addr
rax: accumulator
Stack frame are very cheap. - just pushing, popping, moving to registers / manipulating pointers - heap/garbage collection is much more costly
Computing the index in the frame - How do we map (variable) names to indices in the stack frame? - Symbol table
Keeping names for runtime slows things down, we prefer integer offsets
Compiled with clang -S - Why does y have two values? – Parameters go into registers. If necessary we put them into the frame. – Parameters have a slot in the stack frame and are moved there if necessary. – Slots are always reserved for parameters. (ignoring some optimisation cases)
Leaf procedure: a procedure that doesn't call any other procedures - they turn up as leaves if you draw the call tree - leaf procedures don't need to use the stack pointer
[15/11/04]
Many arguments =¿ spill into the stack - We only have about 32 registers at most. - Use them when we can but use stack if necessary.
printn/printf exercise

Last lecture we looked at leaf functions calls - but what about calling functions?

The order for computing argi is unspecified. Sometimes we swap the order for optimisation. f(x++, x) - no guarantee that x++ will be done first

Sometimes we have to have parameters/values inside the frame. (i.e. when using ampersand/address operator) - you can't point at a register using - "escaping variables"

lea = load effective address (we want address, not value)

*p = x =¿ movq ("indirect addressing")

callq can use * callq *-8(rbp)