Generating Stack Machine Code Using LLVM

Alan Li

See also: [Technical Report]

Introduction

- Software stack machines re-emerge as blockchain and web streaming become popular
  - Stack machines boast better **code density** than register machines
  - Stack machine is much **simpler** than register machines
  - Examples: WebAssembly, TelegramVM, EthereumVM …
- Traditional stack machine codegens are AST-based
  - Missing aggressive optimizations
- LLVM is designed for register machine code generation
  - LLVM is missing some key infrastructures for stack machine codegen
  - Repurposing LLVM to emit stack code is intuitively challenging (but doable)
  - Public WASM backend is an attempt but its methodology is rudimentary and not canonical
  - LLVM is not traditionally good with code size optimization -- need a lot of tunings.
- Open question: can we come up with a general-purpose LLVM codegen plan for stack machines?

Experiment Platform: EthereumVM (EVM)

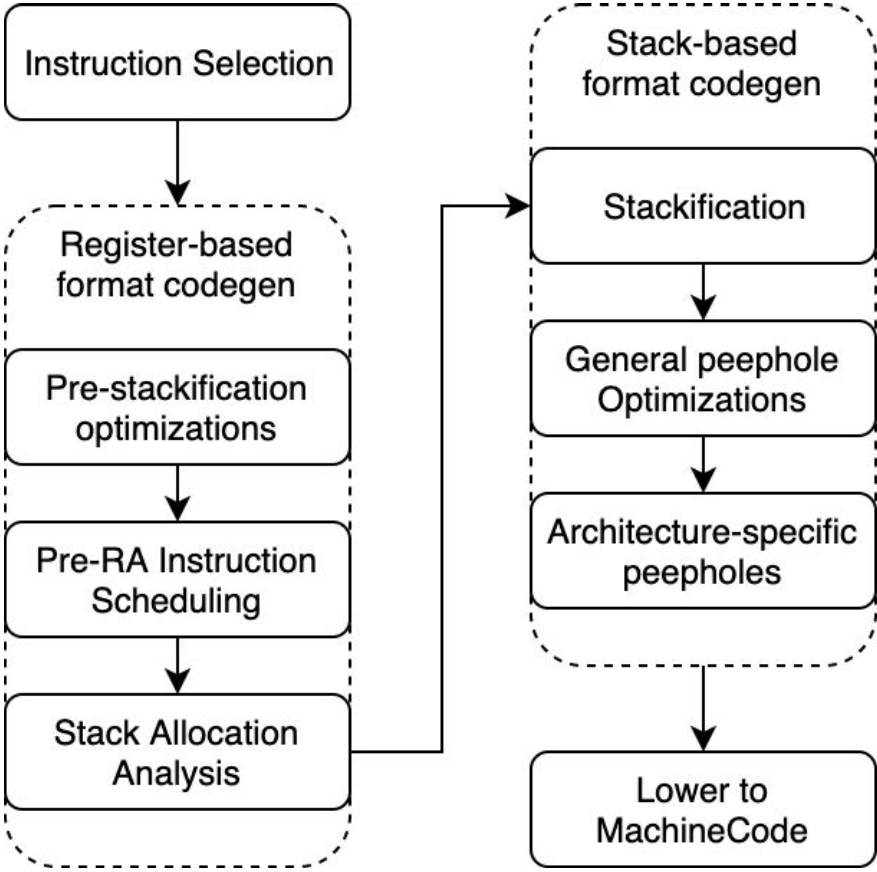
- Most widely used smart contract platform in the blockchain world
- Simplistic/archaic stack machine with 256-bit word length
  - Deterministic: no heap, gc, multithreading, floating point, external variables, dynamic jumps
  - Has blockchain-specific instructions
  - Executing each bytecode instruction costs “gas”
- Over 1 billion worth of DeFi assets are running on EVM
- Solidity and Vyper are the only two smart contract languages:
  - Solidity is criticized for its bad and unsafe design, and its compiler is not-standardized
  - Vyper is simplistic and Python-based
  - After 5 years we are still not seeing other DSLs emerging -- mostly because the platform needs something like LLVM

Optimization Goal for stack machine:

- Reduce non-computing instructions
  - SWAP, DUPLICATE, POP …
    - Stack manipulation instructions are not contributing to actual computing
    - Affected by variables’ location and order on stack or memory
  - Memory spilling instructions (putlocal/getlocal)
- Reduce overall code size
  - Most stack machines are designed for size-sensitive scenarios: (eg. blockchain)
- (blockchain-specific) Reduce overall gas consumption
  - Compromises with code size
- (potentially) software-hardware (software-VM arch) co-design

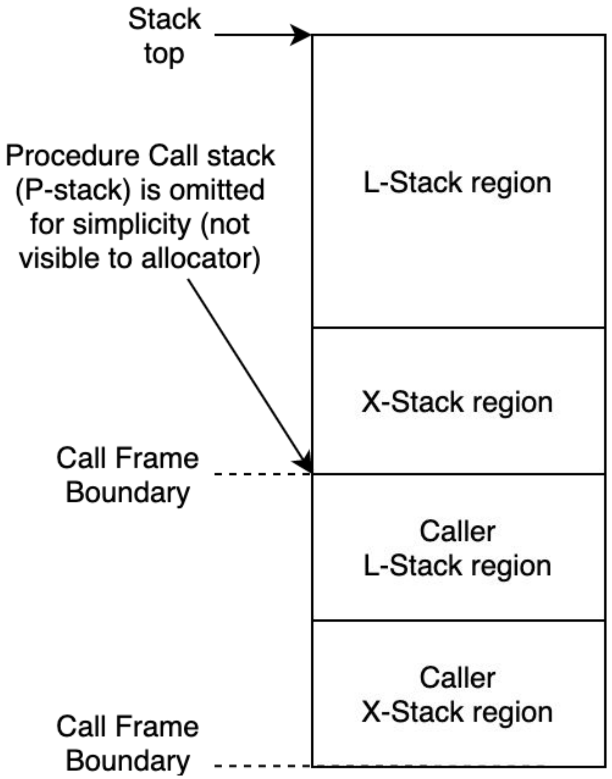
A Stack Machine Codegen Pipeline

- Define two sets of instructions:
  - Register-based Instr
  - Stack-based Instr
  - Mapping (reg -> stack)
  - Borrowed from WASM backend
- Codegen is split into two parts
- Register allocation pass is replaced by Stackification pass
  - Stack-based codegen are non-canonical so a lot of infrastructure tools are unavailable



Stack allocation Pass -- From VReg to Stack format

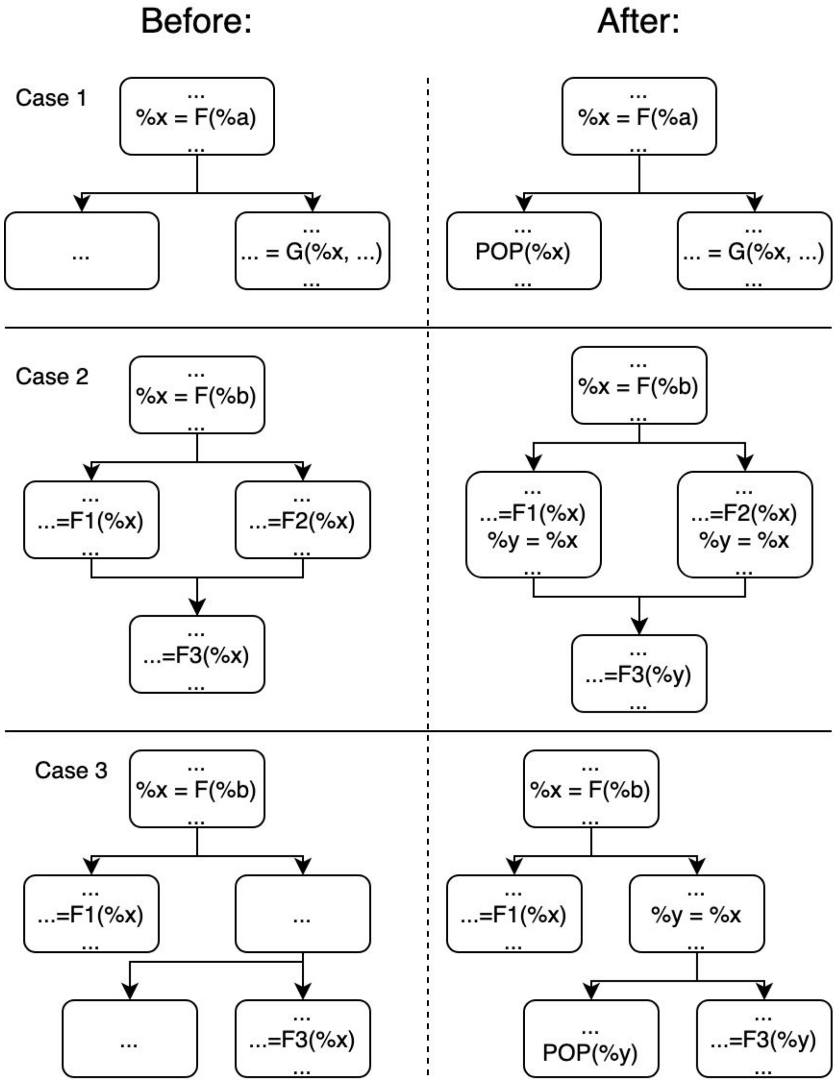
- Assumption: Non-SSA, legal (def dominates all uses)
- Replaces register allocation pass
- Allocate virtual registers to either:
  - Memory slot (spilling) -- All unanalyzable cases
  - intra-MachineBasicBlock stack slot (L-stack) -- All MBB-local regs
  - inter-MachineBasicBlock stack slot (X-stack) -- Some cross-MBB regs
- Pre-RA instruction scheduling helps to reduce manipulation overhead for MBB-local registers
- Properties:
  - L-stacks are empty at entry and exit of MBB
  - X-stacks are empty at entry and exit of MF
  - All successors of a MBB have exactly same incoming X-stack (shape, order)



Shannon, Mark, and Chris Bailey. "Global Stack Allocation–." In *22nd EuroForth Conference*, p. 13. 2006.

Optimizations to improve stack allocation performance

- Goal: Try to avoid spilling regs to memory
  - Instead move regs to inter-MBB stack slots
- Still need to make sure each dominated MBB has at least one use
  - Or a POP must be inserted to keep stack balanced
- Optimization observations:
  - Heuristic: Less active vregs on stack → less stack manipulation overhead
  - Solution: Should **avoid** register coalescing, value numbering



Pre-RA Instruction Scheduling

- Goal: reduce stack manipulation overhead
  - **Local, depth-first** scheduling algorithms show good empirical performance
- Pushing it further:
  - a **global** instruction scheduler might benefit by reducing cross-basicblock vreg pressure

Shannon, Mark, and Chris Bailey. "Global Stack Allocation–." In *22nd EuroForth Conference*, p. 13. 2006.

Park, J., Park, J., Song, W., Yoon, S., Burgstaller, B., & Scholz, B. (2011). Treegraph-based instruction scheduling for stack-based virtual machines. *Electronic Notes in Theoretical Computer Science*, *279*(1), 33-45.

Other Optimization ideas for stack machines

- Function Outlining
  - To reduce overall code size
- Profile-guided/trace-oriented inlining
  - For blockchain use cases: inline most accessed execution paths.
- TreeGraph scheduling -- another stack allocation scheme *
- Rematerialization
  - To reduce virtual register pressure

- Park, J., Park, J., Song, W., Yoon, S., Burgstaller, B., & Scholz, B. (2011). Treegraph-based instruction scheduling for stack-based virtual machines. *Electronic Notes in Theoretical Computer Science*, *279*(1), 33-45.

The End