LLVM-MOS 6502 Backend

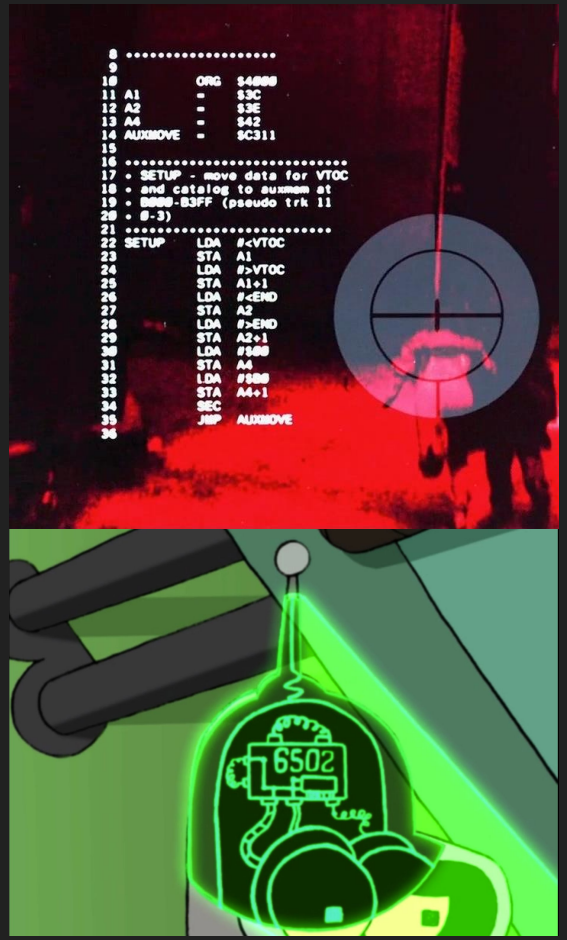
Having a Blast in the Past

Daniel Thornburgh
LLVM-MOS Team
EuroLLVM 2022

The 6502

The MOS Technologies 6502 is the 1975 8-bit CPU behind:

- Nintendo Entertainment System
- Apple I, II, IIe
- Atari 2600-7800, 400-130XE
- Commodore PET, VIC-20, 64, 128
- BBC Micro, Master
- Tamagotchi
- The Terminator
- Bender (Futurama)



Why build a 6502 LLVM backend?

- Machine-generating 6502 code is easy.
- Machine-generating *good* 6502 code is notoriously hard.
- It's easy (if tedious) for a human to write good 6502 code.
- How well do LLVM codegen techniques work here?
- If 6502, why not PIC? Intel 8051? Z80?

The 6502: Chief Difficulties

- 256-byte stack
- Three heterogenous 8-bit registers:
  - A: an 8-bit accumulator
  - X,Y: 8-bit index registers
- Zero Page (first 256 bytes of RAM)
  - Only place to put pointers
  - Faster than general memory

LLVM-MOS: How'd it Go?

- Clang/Rust frontend
  - C99/C++11 freestanding
  - No floating point/exceptions
- Full LLVM and LLD ELF backend.
- On average, generates sorta okay code.
- 0.0813 CoreMark/s for 1 Mhz 6502.
- On occasion, generates great code!
- Backend is young; polishing takes time.


Hello Eur

Demo! Rust on Atari 800

511 lines Rust +

1449 lines ASM (music player) +

14KiB binary assets +

Atari 800 SDK

---

30 KiB Atari 800 Executable

Thanks mrk!

Music: “Noisy Pillars” by miker, orig. Jeroen Tel

LLVM-MOS: Lots of Weird Optimization

- IV Index Extraction
- Zero Extension in LSR Addressing Modes
- Logical Pseudo-Instruction Set
- Static Stack Allocation
- Early G_SELECT Lowering
- RMW RegClass Widening
- Global NZ flag Invariant
- Imaginary Registers
- Light Spilling in Greedy Regalloc
- Target-Specific CSR Slots
- Custom Output Formats
- Post-RA-Pseudo-Expansion Register Scavenging
- Opportunistic NZ Flag Optimization

LLVM-MOS: The Big Ones

1. Static Stack Allocation
2. Imaginary Registers

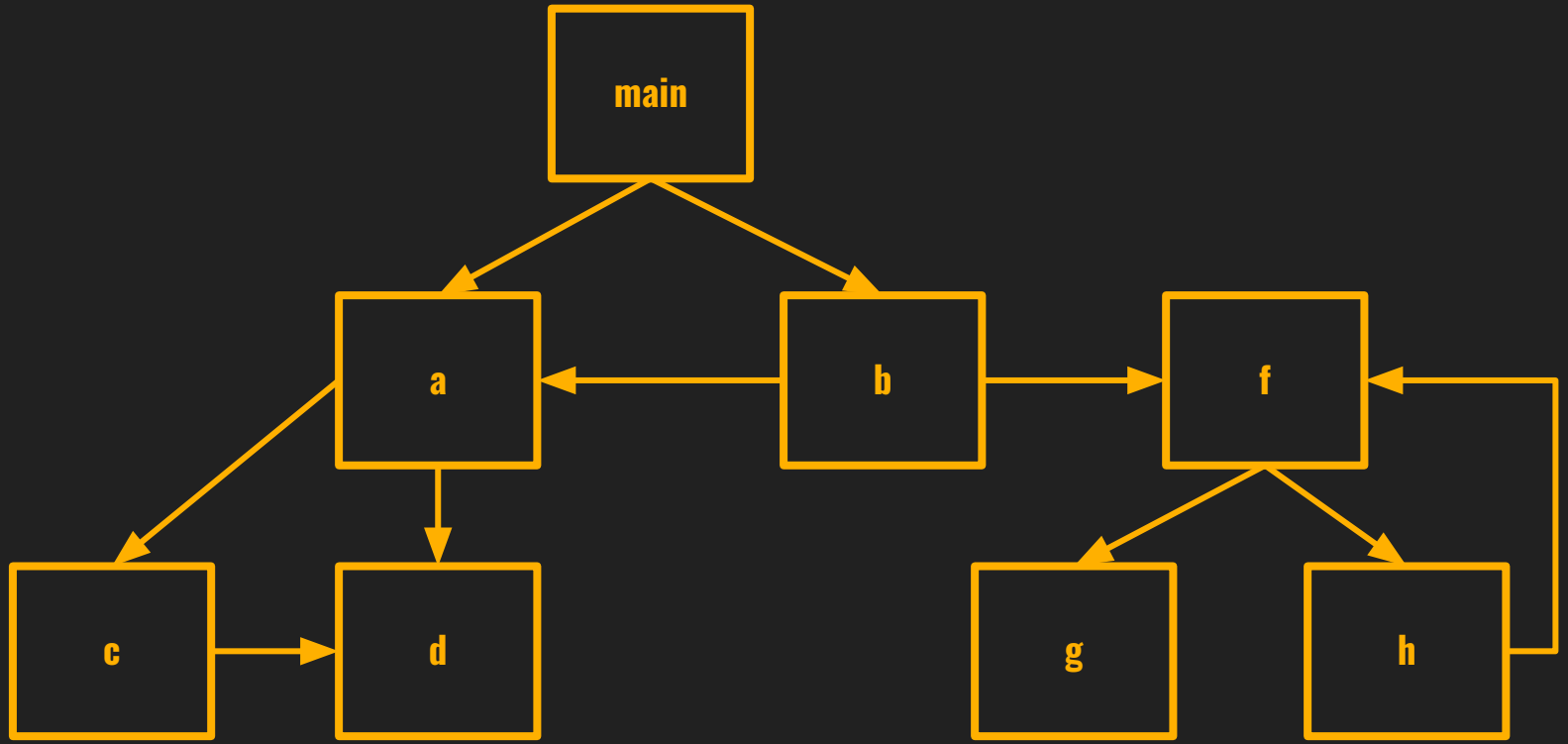
Problem: Tiny HW Stack, Slow SW Stack

The 6502's 256-byte hardware stack pointer is too small for C!

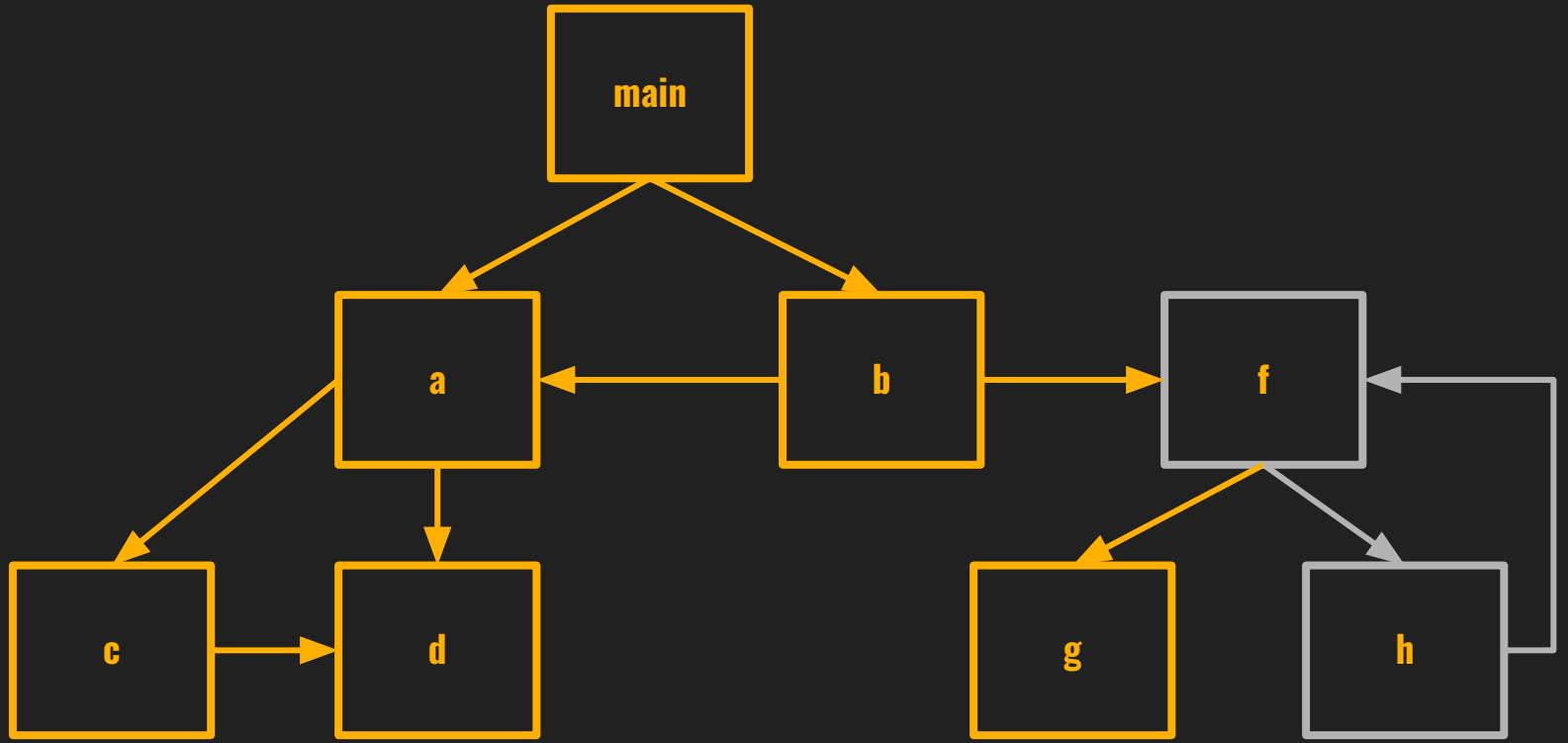
Maintaining/using a 16-bit software stack pointer is slow:

Operation	Auto (Cycles)	Static (Cycles)
Function Prologue	10	0
Function Epilogue	10	0
Variable access	8	4
Array offset access	18	5

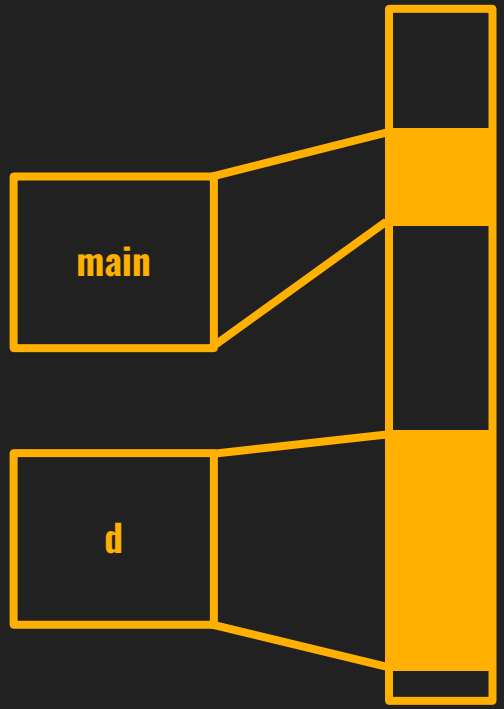
Solution: Static Stack Allocation



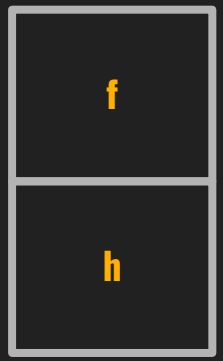
Solution: Static Stack Allocation



Stack Frame Placement

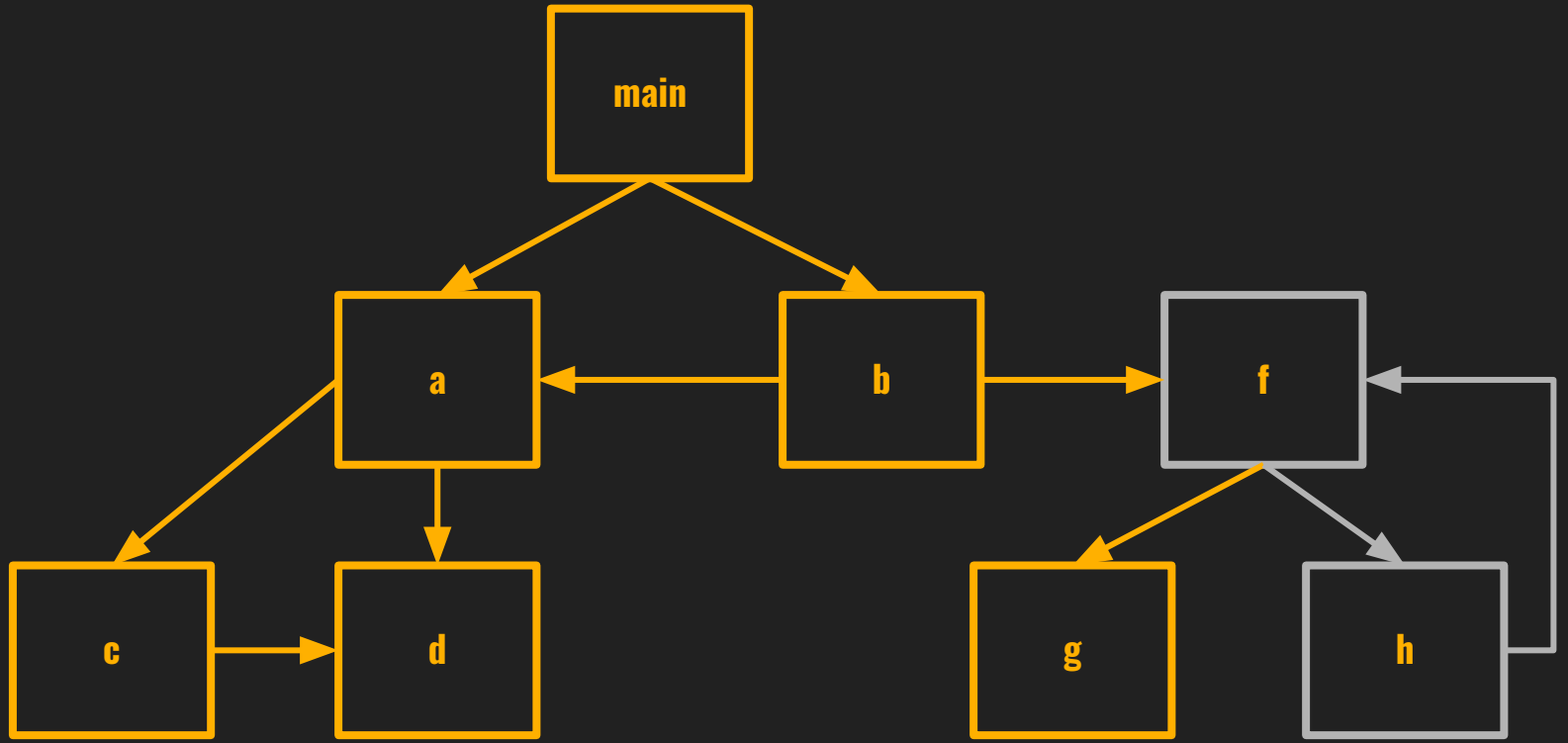


.bss

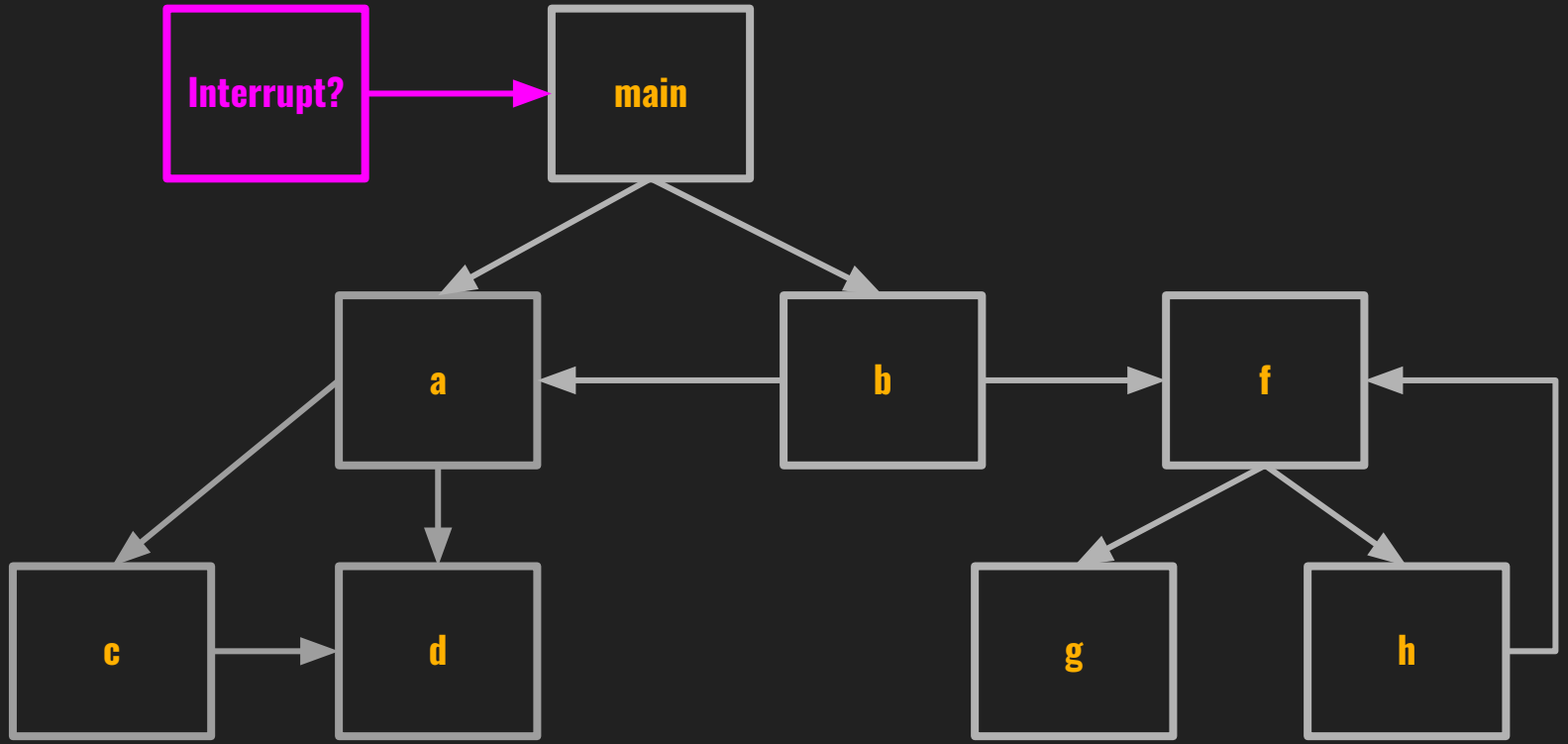


Dynamic
Stack

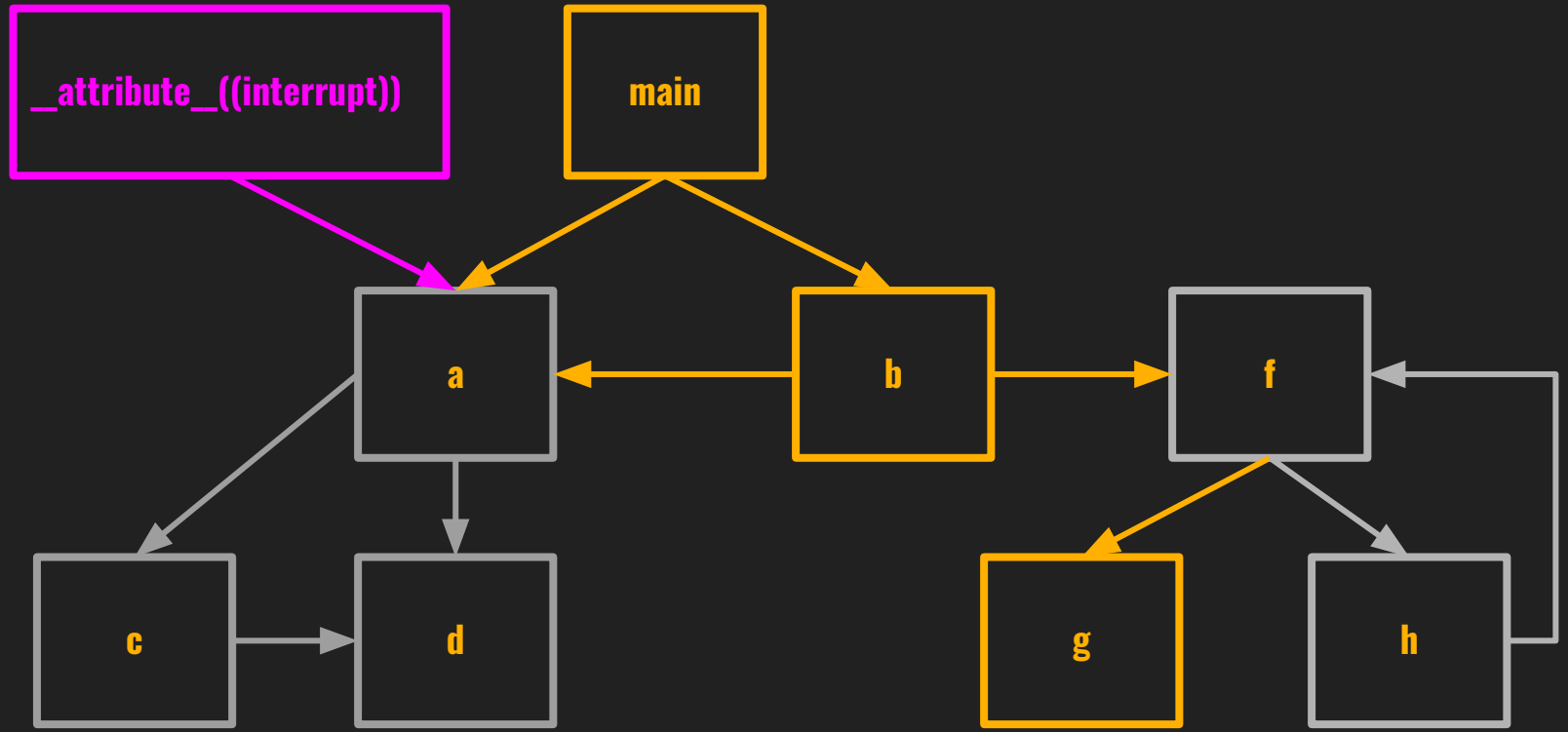
Solution: Static Stack Allocation



Solution: Static Stack Allocation



Solution: Static Stack Allocation



Problem: Tiny Register Set

- Just 3 8-bit registers.
- No register class has >3 elements.
- Greedy Regalloc: “error: ran out of registers during register allocation”

Solution: Imaginary Registers

- The zero page is register-like.
- Reserve 32 bytes of zero page as imaginary registers.
- Greedy regalloc is most pleased.
- Lower to __rcXX symbols after codegen; satisfied by linker scripts.

Calling Convention

Register	Description	Saver
RS0	Stack Pointer	Callee
A,X,Y,RS1-RS7	Argument/Return	Caller
RS8–RS9	Temporaries	Caller
RS10–RS14	Saved Registers	Callee
RS15	Frame Pointer / Saved Register	Callee

Upstream?

- *Outside of our target*, we have a 22,421 line diff from upstream.
  - Much is test changes for other targets due to little tweaks here and there.
  - Major surgery was done to Loop Strength Reduction.
- Near term:
  - Decrease the diff
  - Hacks -> Abstractions
- Long term:
  - Get to near human-level performance
  - Evaluate cost/value of changes

Thanks for Listening!

- Wiki: https://llvm-mos.org/
- Github: https://github.com/llvm-mos
- Early Development Log
- Slack