TL-45

HW/SW FULLSTACK

Overview

HW Design

- ISA Design
- uArch Design
  - Implementation [Single Cycle / Pipeline / OoO]
- Bus Design (Wishbone)
- Peripherals [UART, LCD, VGA Framebuffer]

SW Design

- Assembler (Python+Antlr => llvm-mc)
- Compiler (LLVM clang)
  - Rust
- Simple OS (Shell, simple FAT32 FS)

ISA Design

ISA is the soul of a processor. Collection of Instructions, Registers etc.

Feature of Processor

- **[RISC]** / CISC
- # of bits: 32
- **Register Machine** / Mixed Machine
- Addressing Model? RRR, RRI, RR(IH)
- Addressing Size: dword-> byte
- Special Usecase? Not really here. Using it as a uController

uArch

uArch is the implementation details of a processor. This is completely transparent to SW

Single Cycle Machine

- Simple
- Slow

2 stage pipeine

- Common in low cost uProc
- Basically, Single Cycle w/ prefetch

4/5 stage pipeline

- Small processors
- Stuff like those MIPS in your routers

uArch Cont’d

Superscalar Pipeline

- Basically, multiple pipelines in parallel
- execute code together

Superscalar Out of Order

- Issue in order
- Execute out of order
- Commit in order (ReOrder Buffer ROB)
- Desktop / Mobile processors now.

FPGAs

How are we going to make it?

- Introducing FPGAs
  - Programmable Logic
- Made of “Logic Elements” (Cells). Each “Cell” is usually 1-2 register or a 4-8 input arbitrary gate.
- Programmed using HDL such as
  - (System) Verilog
  - VHDL
- Or High-Level Synthesis tools like
  - Chisel
  - System C

Peripherals

Just a CPU is boring…. I want to see stuff

Peripherals and a Bus

- A bus is how CPU can talk to multiple stuff using only one port.
  - With MMIO in this case
  - Implementing the Wishbone B4 standard
    - Non-Open standard including AXI, AHB (ARM)
- Peripherals
  - This really just depends on your FPGA dev board
    - VGA, GPIO, LCD, 7-segment disp etc…

Software

Okay, I know you guys are mostly software people :D

Assembler

- Well, we need a simple assembler to test
- Initially written by hand using Antlr in Python
- Simple Compiler :D
- Source code has been lost ☹ Sorry, used to be bad at keeping my stuff together

- llvm-mc – More on this later

Compiler

Options for compiler

- Start from scratch
  - Pros:
    - Probably the simplest unless you’ve done this before
    - You know every bit of code you wrote
    - Easier to debug
  - Cons:
    - Less optimization
- Port LLVM / GCC
  - Pros:
    - Optimization
    - Language Support
  - Cons:
    - Complexity – Lack of documentation
    - Compile time. LLVM with only 1 backend, takes about 15 minutes to compile on my machine. (Threadripper 1950x + 64G)

LLVM – a New Backend

Describe Registers and Instructions in Table Gen

- LLVM DSL for Describing ISA
  - defm AND : ALUTypeInstr<0x8, "and", and>;

Iterate this with llvm-mc, the LLVM machine code playground.

- I strongly suggest you start with an assembler. Yes, LLVM can work without one but good place to start.

Work on Target Lowering

- ISEL -> GlobalISEL or SelectionDAG
  - I used selectionDAG at the time. GlobalISEL seems to be taking off now?
- Iterate this with llc
  - –show-isel-dags, show-*-dags are super helpful in debugging

At last Clang and Rustc

When you are confident enough, you can add in support for clang.

Slowly test against larger and larger code base. Iterate.

Rustc is almost the “final” piece.

- To compile anything useful in rust, you will need to compile the “core” crate.
- This generated a (iirc) 30+ MB .ll file
- You would want to avoid having to debug here. LoL