Lab 1 Overview and RISC-V Tools

Section 2
2/1/2013
Yunsup Lee
Find the common case(s), and make sure you can support them efficiently
  - Sometimes at the cost of supporting the uncommon case inefficiently

Examples
  - Found 20% of VAX instructions responsible for 60% of microcode, but only account for 0.2% of execution time!
  - Reg-Immediate instructions

How?
  - Intuition, Simulation, Building, ...
Although there are properties that you can prove mathematically, computer architecture is often based on empirical studies.

This means that your data will be application specific.
- Pick your applications carefully!

“It depends”: Always think about both sides of the argument.
- Keep asking questions to yourself.
• Endianness
• Combinational Path
• Critical Path
• Setup Time
• C-Q (Clock-to-Q) Delay
Agenda

• PSet 1
• How to prepare for the quiz
• Lab 1: What are we asking you to do?
• RISC-V Infrastructure
  – What is available?
  – How to use it?
• Chisel
  – What is it?
  – Some basic Chisel Examples
• Great preparation for the test
• Each question is graded on 0, 1, 2 in terms of effort shown
• 15% of your final grade
How to prepare for the quiz?

• Start early!
• Review the material carefully
  – Read the book
• Make sure you can justify decisions of past architects
• Think about why some ideas might be good and bad
• Go through PSets carefully
• Go through past exams
Lab 1

• Provided RISC-V Processors
  – Implemented in Chisel
• Build C++ simulators
• Run benchmarks on simulators, gather numbers
  – CPI, instruction mix
• Answer, make recommendations, propose new designs
Lab 1

• Provided RISC-V 32-bit processors
  – 1-stage
  – 2-stage
  – 5-stage
    • Fully bypassed
    • Interlocked (stalls to resolve all hazards)
  – Micro-coded

• Only use 1-stage, 5-stage in the lab
  – 2-stage, micro-code are there for your own edification (or open-ended portion, if you choose)
A Lab 1 Chisel Processor

Tile

CPU

CPath  DPath

RAM  ROM
A Lab 1 Chisel Processor

Top

Tile

CPU

CPATH

DPath

I$

D$

ROM

Uncore

RAM
• Add tools to your path
  
  $ source ~cs152/sp13/cs152.bashrc

• Copy Lab Files
  
  $ cp –R ~cs152/sp13/lab1 .

• Build a Chisel processor, Compile simulator, run all tests & benchmarks
  
  $ make run-emulator
Build different variants

$ export MK_TARGET_PROC=rv32_ucode
$ make run-emulator

(other options)
$ export MK_TARGET_PROC=rv32_2stage
$ export MK_TARGET_PROC=rv32_5stage
Chisel

• A new HCL language
  – designed for GENERATING hardware, not simulating it!
  – Verilog, others, were designed to simulate hardware, NOT to build hardware.
    • Thus Verilog can be very difficult to use in creating hardware

• Chisel is “embedded in Scala”
  – A Chisel processor is actually a legal Scala program
    • output of Chisel “program” is either Verilog or C++ code, that describe the processor at the cycle-accurate level
Chisel

*.scala files

Run our Scala “program” using SBT

Heavily templated C++ code

g++

binary called “emulator”

ONLY going to use C++ simulation for Lab 1
RISC-V Toolchain

• riscv-gcc/riscv-g++
  – cross-compiles binaries to RISC-V
  – C and assembly code

• riscv-objdump
  – disassembles binaries into assembly code
  – gives you program address information, register allocations, etc.!

• Fesvr
  – front-end server, call this to run RISC-V binaries
Front-end server

Tile

CPU

CPath

DPath

I$

D$

ROM

Uncore

Top

fesvr

RAM