Quiz 2

• Should be returned late next week
  – (I’ve been working 100% on Lab 3...)
Agenda

- The Mystery OOO Processor
- Lab 3
  - Simics & MAI
  - Chisel

Saturday, March 10, 2012
DEC Alpha 21264

- 1996/1997
- single-core
  - 4-way
  - out-of-order
  - highly speculative
  - 7-stage
  - up to 80 instructions in flight
  - tournament branch predictor
- 15.2M transistors
  - 6M for logic
  - rest is caching, history tables
- 350 nm
- 600 MHz
- 64KB I$, 64KB D$ (on-chip)
  - 1 to 16MB L2$ (off-chip)
- 314mm² die (fairly large)
DEC Alpha 21264

- 1996/1997
- single-core
  - 4-way
  - out-of-order
  - highly speculative
  - 7-stage
  - up to 80 instructions in flight
  - tournament branch predictor
- 15.2M transistors
  - 6M for logic
  - rest is caching, history tables
- 350 nm
- 600 MHz
- 64KB I$, 64KB D$ (on-chip)
  - 1 to 16MB L2$ (off-chip)
- 314mm² die (fairly large)
Lab 3

- Directed Portion
  - Simics
  - Chisel
- Open-ended Portion
  - Your pick of either Simics, Chisel, or C++

- I’m still writing it... I apologize for any difficulties that arise from this
Lab 3: Simics

• Will use Micro-Architectural Interface to simulate speculative Out-of-Order Processor

• Question 1
  – gather CPI of multiple benchmarks

• Question 2
  – compare pipeline width, ROB size with CPI
# my_ma_script.simics
ma_cpu0->fetches_per_cycle = 16
ma_cpu0->execute_per_cycle = 16
ma_cpu0->retires_per_cycle = 16
ma_cpu0->commits_per_cycle = 16
cpu0->reorder_buffer_size = 32
run-cycles 11_000_000
Lab 3: Chisel
DEC Alpha 21264

- 1996/1997
- single-core
  - 4-way
  - out-of-order
  - highly speculative
  - 7-stage
  - up to 80 instructions in flight
  - tournament branch predictor
- 15.2M transistors
  - 6M for logic
  - rest is caching, history tables
- 350 nm
- 600 MHz
- 64KB I$, 64KB D$ (on-chip)
  - 1 to 16MB L2$ (off-chip)
- 314mm² die (fairly large)
DEC Alpha 21264

Diagram of the DEC Alpha 21264 architecture.
Example Machine: MIPS R10k

- 0.35 micron process
- 16.6 x 17.9 mm chip
- 298 mm²
- 6.8 million transistors
  - 4.4 million cache
  - 2.4 million logic
- Full-custom design for datapaths and control logic
- Semi-custom design for less critical control logic

Jan 1996 (150-200 MHz) $3,000 for 200 MHz
1997 -> 250nm @ 250 MHz

R10k Pipeline Timing Diagram

Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.
The Berkeley Out of Order Machine (BOOM)

- 2012 (March 7th Vintage)
- single issue
- 6-stage
- full branch speculation (BHT)
- magic, 1-cycle memory (no caches)
- no bypasses
- no floating point
- no exceptions

default parameters:
- 4 issue slots
- 8 ROB entries
- 4 LD/ST entries
- 4 entries in fetch buffer
- up to 4 branches
The BOOM Processor

- Entire “Tile” is described in Chisel code

Tile (Target System)

- ROM 128 kB
- RAM 128 kB
- CPU
- DPath

Saturday, March 10, 2012
• You will write code in `emulator.cpp` that will probe the state of BOOM.
Running BOOM

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

• Copy Lab Files
  
  $ cp -R ~cs152/Lab3 ./Lab3

• Build a BOOM Processor, Compile Simulator, Run all Tests & Benchmarks
  
  $ cd Lab3/
  
  $./runall.sh

• Takes ~3 minutes
BOOM Parameters

• # of entries in Issue Window
• # of entries in ROB
• # of entries in LD/ST queues
• # of entries in BHT
• can turn on/off branch prediction

• unfortunately, can’t guarantee that all combinations will pass all tests!
Caveats

• BOOM is *brand new*
  – and thus buggy! (maybe)
  – email me if you have tests/benchmarks fail

• BOOM is somewhat artificial
  – uses magic, 1-cycle memory (no caches)
  – has no bypasses (load-use is 3 cycles), which means out-of-order issue is advantageous
Lab 3: Chisel - Directed Portion

• Question 1
  – collect CPI with BHT, without, compare to 5-stage in-order

• Question 2
  – Probe the Instruction Window to potential benefit of dual issue

• Question 3
  – Probe IW for dual issue of ALU/Mem ops
BOOM: A Single Issue Slot

Question 1 – collect CPI with BHT, without, compare to 5-stage in-order

Question 2 – Probe the Instruction Window to potential benefit of dual issue

Question 3 – Probe IW for dual issue of ALU/Mem ops

(UOP Code) 

BrMask 

Ctrl... 

Val 

RDst 

RS1 

p1 

RS2 

p2 

Br Logic 

Resolve or Kill 

(WDest0, WDest1) 

(UOP Code) 

BrMask 

Ctrl... 

Val 

RDst 

RS1 

p1 

RS2 

p2 

Br Logic 

Resolve or Kill 

(WDest0, WDest1) 

Control Signals 

Physical Destination Register 

Physical Source Registers 

Issued to the Register Read stage

Saturday, March 10, 2012
each instruction gets a br mask... allows us to kill instructions

(From the register file's two write ports)

the register file has two write-ports, so watch both ports' write addresses

each slot asserts request when ready to fire

one slot gets the “issue”

(note: I show a bus implementation, but it's actually implemented with a bunch of muxes)
Probing Chisel from C++

- `dat_t<1> Tile_cpu_d_IntegerIssueSlot__slot_valid`
- `dat_t<1> Tile_cpu_d_IntegerIssueSlot_1__slot_valid`
- `dat_t<1> Tile_cpu_d_IntegerIssueSlot_2__slot_valid`
- `dat_t<1> Tile_cpu_d_IntegerIssueSlot_3__slot_valid`

- These are Chisel `dat_t` types... to get to `int64_t`
- `dat_t<1> Tile_cpu_d_IntegerIssueSlot__slot_valid.lo_word()`

- Grep emulator/rv32_boom/generated-src/Tile.h for Chisel signals

Saturday, March 10, 2012
Lab 3: Chisel - Open-ended

• Option 1
  – Probe LD/ST Unit (C++)
    • currently does not bypass load data out of SDQ
    • measure performance loss

• Option 2
  – Build a better branch predictor for BOOM (Chisel)
  – compete against your classmates (and your TA...)
  – +2 points for best predictor

• Option 3
  – Analyze other pieces of BOOM?
Branch Predictors
Branch Prediction

- Two kinds of correlating branch predictors:

  - **Local**
    - Local History Table
    - Branch History Table
    - PC
  
  - **Global**
    - Branch History Table
    - Global History
• 21264 uses both! (tournament predictor)
Tournament Branch Predictor (Alpha 21264)

- Choice predictor learns whether best to use local or global branch history in predicting next branch
- Global history is speculatively updated but restored on mispredict
- Claim 90-100% success on range of applications
Reading Printouts
Reading Printouts
Reading Printouts

- **FetchBuffer**
- **ROB**
- **LAQ**
- **SAQ**
- **SDQ**

- freelist (bit-array of free/notfree bools)
- new physical register dest given out
- preregister valid?
- BHT Prediction?
- Branch Resolution

- Issued Inst?
- Inst in Decode
- ISA registers
- Translated Physical Registers

- PCs

- Saturday, March 10, 2012
comment out "#define DEBUG" in emulator.cpp to get the "short form" debugging
Analyzing the Waveforms

• Although not necessary to complete the lab, it is possible to generate waveform files from Chisel runs (stored in cpu.vcd file).
• START EARLY
  – compute time will become valuable
• Everything you need to run the Lab locally is provided
  – except Scala Built Tool (SBT)
    • easy to install, read the README
  – caveat
    • BOOM generates a LOT of C++ code
    • use gcc 4.4 or newer
    • should take no more than 5 minutes
    • (probably need to change emulator/common/Makefile.include variable CPP=g++44 to CPP=g++)
If you like Chisel/BOOM...

- This is just the beginning of BOOM
  - I’m looking for excited undergrads who want to hack processors
  - laundry list of features to add
    - wider issue widths
    - caches/memory hierarchy
    - higher performance LD/ST unit
    - floating point support
Questions?

- 2012 (March 7th Vintage)
- single issue
- 6-stage
- full branch speculation (BHT)
- magic, 1-cycle memory (no caches)
- no bypasses
- no floating point
- no exceptions