Computer Architecture Review

or

CSE141’s Greatest Hits

Instruction Set Architecture

• What we learned
  – ISA types
  – ISA formats and tradeoffs
  – addressing modes
  – branch types
  – MIPS ISA

Instruction Set Architecture

• What we can do
  – identify and write simple code for various ISA types
  – identify and use several addressing modes (MIPS type, particularly)
  – write MIPS code (with cheat sheet)
  – encode or decode MIPS code to/from machine language
  – evaluate tradeoffs between ISAs

Performance

• What we learned
  – speedup
  – execution time
  – benchmarks
  – Amdahl’s law
Performance
• What we can do
  – calculate CPI, ET, clock-rate, etc.
  – calculate speedup
  – apply Amdahl’s law

Computer Arithmetic
• What we learned
  – the computer uses binary numbers
  – number systems
  – negative numbers
  – addition, subtraction, multiplication, division
  – ALU design
  – fast adders
  – floating point numbers, operations

Computer Arithmetic
• What we can do
  – manipulate binary numbers
  – do arithmetic on binary numbers
  – do arithmetic on fp numbers
  – simple adder, ALU design

CPU Architecture
• What we learned
  – single-cycle cpu, multiple cycle cpu
  – datapaths
  – control logic
  – multiple-cycle control
    • FSM
    • microprogramming
  – exceptions
CPU Architecture

- What we can do
  - construct datapath for new instructions
  - generate control logic (or FSM or microprogram) for new datapath or new instruction (don’t memorize microprogram format, just learn concepts)
  - incorporate exceptions into datapath and control

Pipelining

- What we can do
  - design a slightly different pipelined machine
  - generate control for it
  - understand implications of all kinds of data hazards
  - understand implications of all kinds of branch hazards
  - reason about instruction schedules for pipelined, superscalar, out-of-order, or VLIW machines

Pipelining

- What we learned
  - pipelined machine design, including
    - use of intermediate registers
    - pipelined control
  - data hazards, bubbles, and forwarding
  - branch hazards, bubbles/flushing, and simple branch prediction
  - advanced architectural concepts including branch prediction, superscalar execution, superpipelining, and out-of-order execution.

This pipeline is designed to eliminate the ld-use data hazard of the MIPS pipeline. Does it? Assume address calculation is done in addr, and all other alu operations are done in ex. What other hazards does it create? Show all bubbles and forwarding for the following code:

```
lw R3, 100(R6)  if id addr mem ex wb
add R5, R3, R2  if id
sw R5, 200(R6)
sub R7, R1, R3
lw R4, 500(R7)
```
Memory

- What we learned
  - locality
  - memory hierarchies
  - hits, misses, miss penalties
  - cache performance (miss rates, MCPI, ET)
  - write-through, write-back, write-allocate, write-around
  - associativity
  - virtual memory, page tables
  - TLBs
  - Compulsory, capacity, conflict misses

- What we can do
  - identify types of locality, types of misses (compulsory, capacity, conflict)
  - identify hits and misses and manipulate cache structures for all types of caches
  - evaluate or predict cache performance
  - do LRU replacement
  - manipulate page table and TLB structures, identify TLB misses and page faults

11. (5 points) A 256 KB cache interprets addresses this way. What is the associativity of this cache? (Addresses are byte addresses)

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
<th>block offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>10 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

Multiprocessors

- What we learned
  - bus-based, network-based
  - UMA, NUMA
  - shared-memory, message-passing
  - cache coherency
  - multithreading