Welcome to CSE 502

Introduction & Review
Today’s Lecture

• Course Overview
• Course Topics
• Grading
• Logistics
• Academic Integrity Policy
• Homework #1 (of 1)
• Quiz #1 (of 1)
• Key concepts from Undergrad Computer Architecture
Course Overview (1/3)

• Caveat 1: I’m new here.
• Caveat 2: This is a new course.

• Computer Architecture is
  ... the science and art of selecting and interconnecting hardware and software components to create computers ...
Course Overview (2/3)

• Ever wonder what’s inside that box, anyway?

• Computer Architecture is an umbrella term
  – *Architecture*: software-visible interface
  – *Micro-architecture*: internal organization of components

• This course is mostly about *micro-architecture*
  – What’s inside the processor (CPU)
  – What implications this has on software
Course Overview (3/3)

• This course is hard, roughly like CSE 506
  – In CSE 506, you learn what’s inside an OS
  – In CSE 502, you learn what’s inside the CPU

• This is a project course
  – Learn why things are the way they are, first hand
  – We will “build” emulators of CPU components
  – If you don’t know C++, you need to learn it quickly
Course Topics

- Intro/Review
- Memory Hierarchy
- Pipelining
- Instruction-Level Parallelism
- Processor Front-end
- Execution Core
- Multi-[socket(SMP,DSM) | thread(SMT,CMT) | core(CMP)]
- Prefetching
- Vector Processing and GPUs

Will devote most time to items in **bold**
## Grading

<table>
<thead>
<tr>
<th>kk</th>
<th>Due Date</th>
<th>Points</th>
<th>Grading</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Quiz</td>
<td>Today</td>
<td>0</td>
<td>Curve 0 to 100</td>
<td>Yes</td>
</tr>
<tr>
<td>1 Homework</td>
<td>Last class</td>
<td>10</td>
<td>Curve 0 to 100</td>
<td>No</td>
</tr>
<tr>
<td>1 Warm-up Project</td>
<td>Feb 25</td>
<td>20</td>
<td>See below</td>
<td>No</td>
</tr>
<tr>
<td>1 Course Project</td>
<td>Last class</td>
<td>100</td>
<td>See below</td>
<td>Yes</td>
</tr>
<tr>
<td>1 Final</td>
<td></td>
<td>30</td>
<td>Absolute value</td>
<td>No</td>
</tr>
<tr>
<td>Participation</td>
<td></td>
<td>10</td>
<td>Curve 0 to 100</td>
<td>No</td>
</tr>
</tbody>
</table>

### Warm-up Project Points

<table>
<thead>
<tr>
<th>Warm-up Project</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Port, Direct-Mapped, 16K</td>
<td>10</td>
</tr>
<tr>
<td>2 Port, Direct-Mapped, 16K</td>
<td>12</td>
</tr>
<tr>
<td>1 Port, Set-associative, 16K</td>
<td>14</td>
</tr>
<tr>
<td>2 Port, DM, 64K, Non-Block</td>
<td>16</td>
</tr>
<tr>
<td>2 Port, SA, 64K, Non-Block</td>
<td>18</td>
</tr>
<tr>
<td>2 Port, SA, 64K, NB, Way-pred.</td>
<td>20</td>
</tr>
</tbody>
</table>

### Course Project Points

<table>
<thead>
<tr>
<th>Course Project</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-cycle</td>
<td>50</td>
</tr>
<tr>
<td>5+ stage pipeline</td>
<td>60</td>
</tr>
<tr>
<td>Super-scalar</td>
<td>70</td>
</tr>
<tr>
<td>Out-of-order</td>
<td>80</td>
</tr>
<tr>
<td>Out-of-order, Branch-Pred</td>
<td>90</td>
</tr>
<tr>
<td>Out-of-order, BP, SMT</td>
<td>100</td>
</tr>
</tbody>
</table>

Without curve, need 100 points to get an A
Logistics (1/3)

• Project milestones
  – There are *no* official project milestones
  – If *you* need milestones, send me a milestone schedule
    • I will deduct 5 points for each milestone you miss

• Book
  – Hennessy & Patterson, use it as a reference
  – Exam questions will mostly come from the book
  – Appendices as important as the book chapters
Logistics (2/3)

• Working in groups
  – Permitted on everything except Quiz and Final
  – Groups may range in size from 1 to 72 people

• Attendance
  – Optional (but highly advised)
  – No laptop, tablet, or phone use in class
    • Don’t test me - I will deduct grade points
Logistics (3/3)

• Blackboard
  – This will be my first time using it, don’t expect much there

• Course Mailing List
  – Subscription is **required**

• Quiz
  – Completion is **required**
  – If you missed the 1\textsuperscript{st} class, come to office hours for it
Academic Integrity Policy

• Probably different from other classes
  – Much more open, but much more strict
  – Resembles the “real world”

• Actual Policy:
  – All submitted work must have an explicit Copyright label containing your name.
  – All submitted work must have an explicit license.
  – All Copyright laws of the United States must be respected.
Copyright Example 1

• Copyright © 2013 by Mike Ferdman. All rights reserved.
Copyright Example 2

- Copyright © 2013 by Mike Ferdman. Permission to copy and distribute verbatim copies permitted.

Copying is OK, but without ANY modifications
Copyright Example 3

- Copyright © 2013 by Mike Ferdman. This work is licensed under GPLv3, details in accompanying COPYING file.
Questions?
Homework #1 (of 1) part 1

• Start with
  volatile int x = 0;
  int main(int argc, char* argv[]) {
    for(int y=0;y<1234567890;++y) ++x;
    return x;
  }

• Implement the most accurate way to measure time
  — How long does the for() loop take to run?
Quiz #1 (of 1)
Review

- Understanding and Measuring Performance
- Memory Locality
- Power and Energy
- Parallelism and Critical Paths
- Instruction Set Architecture
- Basic Processor Organization

This is intended to be a review, not an introduction!
**Amdahl’s Law**

\[ \text{Speedup} = \frac{\text{time}_{\text{without enhancement}}}{\text{time}_{\text{with enhancement}}} \]

An enhancement speeds up fraction \( f \) of a task by factor \( S \)

\[ \text{time}_{\text{new}} = \text{time}_{\text{orig}} \cdot \left( (1-f) + \frac{f}{S} \right) \]

\[ S_{\text{overall}} = \frac{1}{(1-f) + \frac{f}{S}} \]
The **Iron Law** of Processor Performance

\[
\frac{\text{Time}_{\text{Program}}}{\text{Instructions}_{\text{Program}}} = \frac{\text{Cycles}_{\text{Instruction}}}{\text{Time}_{\text{Cycle}}}
\]

Architects target CPI, but **must** understand the others.
Performance

- **Latency** (execution time): time to finish one task
- **Throughput** (bandwidth): number of tasks/unit time
  - Throughput can exploit parallelism, latency can’t
  - Sometimes complimentary, often contradictory

Example: move people from A to B, 10 miles
- Car: capacity = 5, speed = 60 miles/hour
- Bus: capacity = 60, speed = 20 miles/hour
- Latency: car = 10 min, bus = 30 min
- Throughput: car = 15 PPH (count return trip), bus = 60 PPH

No right answer: pick metric for your goals
Performance Improvement

• Processor A is X times faster than processor B if
  – Latency(P,A) = Latency(P,B) / X
  – Throughput(P,A) = Throughput(P,B) * X

• Processor A is X% faster than processor B if
  – Latency(P,A) = Latency(P,B) / (1+X/100)
  – Throughput(P,A) = Throughput(P,B) * (1+X/100)

• Car/bus example
  – Latency? Car is 3 times (200%) faster than bus
  – Throughput? Bus is 4 times (300%) faster than car
Partial Performance Metrics Pitfalls

• Which processor would you buy?
  – Processor A: CPI = 2, clock = 2.8 GHz
  – Processor B: CPI = 1, clock = 1.8 GHz
  – Probably A, but B is faster (assuming same ISA/compiler)

• Classic example
  – 800 MHz Pentium III faster than 1 GHz Pentium 4
  – Same ISA and compiler
Averaging Performance Numbers (1/2)

- Latencies are additive, throughput is not
  \[ \text{Latency}(P1+P2,A) = \text{Latency}(P1,A) + \text{Latency}(P2,A) \]
  \[ \text{Throughput}(P1+P2,A) \neq \text{Throughput}(P1,A) + \text{Throughput}(P2,A) \]

- Example:
  - 1 mile @ 30 miles/hour + 1 mile @ 90 miles/hour
    - Average is \textit{not} 60 miles/hour
  - 0.033 hours at 30 miles/hour + 0.01 hours at 90 miles/hour
    - Average is \textit{only} 47 miles/hour! (2 miles / (0.033 + 0.01 hours))
Averaging Performance Numbers (2/2)

- **Arithmetic**: times
  - proportional to time
  - e.g., latency

- **Harmonic**: rates
  - inversely proportional to time
  - e.g., throughput

- **Geometric**: ratios
  - unit-less quantities
  - e.g., speedups

\[ \frac{1}{n} \sum_{i=1}^{n} Time_i \]

\[ \frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}} \]

\[ \sqrt[n]{\prod_{i=1}^{n} Ratio_i} \]

Memorize these to avoid looking them up later
Locality Principle

- Recent past is a good indication of near future

*Temporal Locality*: If you looked something up, it is very likely that you will look it up again soon

*Spatial Locality*: If you looked something up, it is very likely you will look up something nearby soon
• **Power**: instantaneous rate of energy transfer
  – Expressed in Watts
  – In Architecture, implies conversion of electricity to heat
  – $\text{Power(Comp1+Comp2)} = \text{Power(Comp1)} + \text{Power(Comp2)}$

• **Energy**: measure of using power for some time
  – Expressed in Joules
  – power $\times$ time (joules = watts $\times$ seconds)
  – $\text{Energy(OP1+OP2)} = \text{Energy(OP1)} + \text{Energy(OP2)}$
Power vs. Energy (2/2)

Power is height of curve

Lower power design could simply be slower

Approach 1

Approach 2

Energy is area under curve

Total energy needed to complete operation

Approach 1

Approach 2

Does this example help or hurt?
Why is energy important?

- Because electricity consumption has costs
  - Impacts battery life for mobile
  - Impacts electricity costs for tethered
    - Delivering power for buildings, countries
    - Gets worse with larger data centers ($7M for 1000 racks)
Why is power important?

• Because Energy is Power*Time

• Because of delivery, packaging, and cooling costs
  – Increased cost of thermal packing
    • $1/W for CPUs > 35W
  – Temperature, noise issues ...

• Because of compaction, density, and reliability
  – Thermal failures
    • 50% server reliability degradation for +10oC
    • 50% decrease in hard disk lifetime for +15oC
Power: The Basics

• Dynamic power vs. Static power
  – Static: “leakage” power
  – Dynamic: “switching” power
  – Dynamic power dominates, but static increasing in importance

• Static power: steady, per-cycle energy cost
• Dynamic power: capacitive and short-circuit
  – Capacitive power: charging/discharging at 0→1 and 1→0
  – Short-circuit power: brief short-circuit during transitions
Dynamic Power Dissipation (Capacitive)

Power $\approx \frac{1}{2} CV^2 Af$

- **Capacitance**: Function of wire length, transistor size
- **Supply Voltage**: Function of technology and operating frequency
- **Activity factor**: Average fraction of all possible transitions (0→1 and 1→0) per cycle?
- **Clock frequency**: Function of desired performance

What uses power in a chip?
Lowering Dynamic Power

- Reducing Voltage (V) has quadratic effect
  - Has a negative (~linear) effect on frequency
  - Limited by technology (insufficient difference of 1 & 0)
- Lowering Capacitance (C) has linear effect
  - May improve frequency
  - Limited by technology (small transistors, short wires)
- Reducing switching Activity (A) has linear effect
  - A function of signal transition stats
  - Turns off idle units to reduce activity
  - Impacted by logic and architecture decisions
Leakage Power (1/3)

[Diagram of a CMOS transistor with labels for Gate, Source, Drain, Applied Voltage, Current, and Threshold Voltage]
Leakage Power (2/3)

- Gate Leakage
- Channel Leakage
- Sub-threshold Conductance
Channel Length keeps Shrinking (faster transistors)

Channel resistance decreases (Leakage increases)

Oxide Thickness keeps Shrinking (faster transistors)

Probability of Quantum Tunneling Increases (Leakage increases)

\[ I_{ox} = K_2 W \left( \frac{V}{T_{ox}} \right)^2 e^{-\alpha T_{ox}/V} \]

\[ I_{sub} = K_1 We^{-V_{th}/\eta V_\theta} (1-e^{-V/V_\theta}) \]

(important take-away is on the next slide)
Thermal Runaway

• Leakage is a function of temperature

• \( \uparrow \text{Temp} \rightarrow \uparrow \text{Leakage} \)

• Which burns more power

• Which leads to \( \uparrow \text{Temp} \), which leads to...

\[
I_{\text{sub}} = K_1 W e^{-\frac{V_{th}}{V_0}} \left(1 - e^{-\frac{V}{V_0}}\right)
\]

Positive feedback loop will melt your chip
Power Management in Processors

• **Clock gating**
  – Stop switching in unused components
  – Done automatically in most designs
  – Near instantaneous on/off behavior

• **Power gating**
  – Turn off power to unused cores/caches
  – High latency for on/off
    • Saving SW state, flushing dirty cache lines, turning off clock tree
    • Carefully done to avoid voltage spikes or memory bottlenecks
  – Issue: Area & power consumption of power gate
  – Opportunity: use thermal headroom for other cores
**DVFS: Dynamic Voltage/Frequency Scaling**

- Set frequency to the lowest needed
  - Execution time = $IC \times CPI \times F$

- Scale back $V$ to lowest for that frequency
  - Lower voltage $\rightarrow$ slower transistors
  - Power $\approx C \times V^2 \times F$

- Provides P states for power management
  - Heavy load: frequency, voltage, power high
  - Light load: frequency, voltage, power low
  - Trade-off: power savings vs overhead of scaling
  - Effectiveness limited by voltage range
Parallelism: Work and Critical Path

- **Parallelism**: number of independent tasks available
- **Work** \((T_1)\): time on sequential system
- **Critical Path** \((T_\infty)\): time on infinitely-parallel system

**Average Parallelism**:

\[ P_{avg} = \frac{T_1}{T_\infty} \]

For a \(p\)-wide system:

\[ T_p \geq \max\{ \frac{T_1}{p}, T_\infty \} \]

\[ P_{avg} \gg p \Rightarrow T_p \approx \frac{T_1}{p} \]

Can trade off frequency for parallelism
ISA: A contract between HW and SW

• **ISA**: Instruction Set Architecture
  – A well-defined hardware/software interface

• The “contract” between software and hardware
  – Functional definition of operations supported by hardware
  – Precise description of how to invoke all features

• No guarantees regarding
  – How operations are implemented
  – Which operations are fast and which are slow (and when)
  – Which operations take more energy (and which take less)
Components of an ISA

• Programmer-visible states
  – Program counter, general purpose registers, memory, control registers

• Programmer-visible behaviors
  – What to do, when to do it

Example “register-transfer-level” description of an instruction

if imem[pc]==“add rd, rs, rt”
then
  pc \leftarrow pc+1
  gpr[rd]=gpr[rs]+gpr[rt]

• A binary encoding

ISAs last forever, don’t add stuff you don’t need
RISC vs. CISC

• Recall Iron Law:
  – $(\text{instructions/program}) \times (\text{cycles/instruction}) \times (\text{seconds/cycle})$

• **CISC** (Complex Instruction Set Computing)
  – Improve “instructions/program” with “complex” instructions
  – Easy for assembly-level programmers, good code density

• **RISC** (Reduced Instruction Set Computing)
  – Improve “cycles/instruction” with many single-cycle instructions
  – Increases “instruction/program”, but hopefully not as much
    • Help from smart compiler
  – Perhaps improve clock cycle time $(\text{seconds/cycle})$
    • via aggressive implementation allowed by simpler instructions

Today’s x86 chips translate CISC into ~RISC
Prototypical Processor Organization

Addr-gen.  Fetch  Decode  Issue  Execute  Memory

PC

Instruction Cache

Register File

ALU

Data Cache

(Write-back)
Conclusion

• Know the topics from the Review section
  – If you don’t, you need to catch up

• Make sure you know C++
  – If you don’t, get a book today

• So far, we had intro + review potpourri

• The rest of this course will be very unlike this lecture
  – Very few new terms
  – Practically no formulas
  – Lots of new material

Questions?