Instruction Set Architecture: Critical Interface

- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

Example: MIPS

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0, r1, r2</td>
<td>Programmable storage</td>
</tr>
<tr>
<td>r31</td>
<td>2^32 x bytes</td>
</tr>
<tr>
<td>PC</td>
<td>32 x 32-bit FP regs (paired DP)</td>
</tr>
<tr>
<td>hi, lo</td>
<td>HI, LO, PC</td>
</tr>
</tbody>
</table>

Arithmetic/Logical
- Add, AddI, Sub, SubI, And, Or, Xor, Nor, Slt, Sltu, Addl, AddIU, SltI, SltIU, AndI, OrI, XorI
- Sll, Srl, Sra, Sllv, Srlv, Sraw

Memory Access
- LB, LBU, LH, LHU, LW, LWL, LWL
- SB, SH, SW, SWL, SWR

Control
- J, JL, JR, JALR, BEQ, BNE, BGEZ, BGTZ, BGTZAL, BGEZAL

32-bit instructions on word boundary

ISA vs. Computer Architecture

- Old definition of computer architecture = instruction set design
  - Other aspects of computer design called implementation
  - Insinuates implementation is uninteresting or less challenging
- New view is computer architecture >> ISA
- Architect’s job much more than instruction set design; technical hurdles today more challenging than those in instruction set design

Comp. Arch. is an Integrated Approach

- What really matters is the functioning of the complete system
  - hardware, runtime system, compiler, operating system, and application
  - In networking, this is called the “End to End argument”
- Computer architecture is not just about transistors, individual instructions, or particular implementations
  - E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions
Computer Architecture is
Design and Analysis
Architecture is an iterative process:
• Searching the space of possible designs
• At all levels of computer systems
Creativity
Mediocre Ideas
Bad Ideas
Good Ideas

What Computer Architecture brings to Table
• Other fields often borrow ideas from architecture
• Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl’s Law
  5. The Processor Performance Equation
• Careful, quantitative comparisons
  – Define, quantify, and summarize relative performance
  – Define and quantify relative cost
  – Define and quantify dependability
  – Define and quantify power

1) Taking Advantage of Parallelism
• Increasing throughput of server computer via multiple processors or multiple disks
  • Detailed HW design
    – Carry lookahead adders uses parallelism to speed up computing
    sums from linear to logarithmic in number of bits per operand
    – Multiple memory banks searched in parallel in set-associative caches
  • Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.
    – Not every instruction depends on immediate predecessor ⇒ executing instructions completely/partially in parallel possible
    – Classic 5-stage pipeline:
      1) Instruction Fetch (Ifetch),
      2) Register Read (Reg),
      3) Execute (ALU),
      4) Data Memory Access (Dmem),
      5) Register Write (Reg)

2) The Principle of Locality
• The Principle of Locality:
  – Program access a relatively small portion of the address space at any instant of time.
• Two Different Types of Locality:
  – Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  – Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
• Last 30 years, HW relied on locality for memory perf.
### Levels of the Memory Hierarchy

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Access Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Registers</td>
<td>100s Bytes</td>
<td>300 - 500 ps (0.3-0.5 ns)</td>
</tr>
<tr>
<td>L1 and L2 Cache</td>
<td>10s-100s K Bytes</td>
<td>~1 ns - ~10 ns</td>
</tr>
<tr>
<td>Main Memory</td>
<td>G Bytes</td>
<td>80ns - 200ns</td>
</tr>
<tr>
<td>Disk</td>
<td>T Bytes</td>
<td>10 ms (10,000,000 ns)</td>
</tr>
</tbody>
</table>

#### CPU Registers
- **Registers**: 100s Bytes, 300-500 ps
- **Instruction Operands**: 1-8 bytes

#### L1 and L2 Cache
- **Blocks**: 32-64 bytes
- **Cache**: 1-8 bytes

#### Main Memory
- **Pages**: 4K-8K bytes
- **Memory**: G Bytes

#### Disk
- **Files**: Mbytes
- **Tape**: infinite sec-min

#### Costs
- CPU Registers: ~$1000s per GByte
- L1 and L2 Cache: ~$1000s per GByte
- Main Memory: ~$100 per GByte
- Disk: ~$1 per GByte

### 3) Focus on the Common Case

- **Common sense guides computer design**
  - Since its engineering, common sense is valuable
- **In making a design trade-off, favor the frequent case over the infrequent case**
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- **Frequent case is often simpler and can be done faster than the infrequent case**
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- **What is frequent case and how much performance improved by making case faster** >> Amdahl’s Law

### 4) Amdahl’s Law

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}\right)
\]

\[
\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]

**Best you could ever hope to do:**

\[
\text{Speedup}_{\text{minimum}} = \frac{1}{1 - \text{Fraction}_{\text{enhanced}}}
\]

**Amdahl’s Law example**

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

\[
\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \frac{0.4}{10}\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56
\]

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

### 5) Processor performance equation

\[
\text{CPU time} = \text{Seconds} = \text{Instructions} \times \text{Cycles} \times \text{Seconds}
\]

<table>
<thead>
<tr>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Inst. Set.</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

### CSE 490/590 Administrivia

- Jangyoung Kim’s Office Hours
  - Office: 232 Bell
  - Office Hours: MWF, 1pm – 2pm
- Please check the web page: [http://www.cse.buffalo.edu/~stevko/courses/cse490/spring11](http://www.cse.buffalo.edu/~stevko/courses/cse490/spring11)
- Don’t forget: Recitations start from next week!
- Disclaimer: Lecture notes are heavily based on UC Berkeley’s materials.
Technology Trends

Moore's Law: 2X transistors / “year”

- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors / cost-effective integrated circuit double every N months (12 ≤ N ≤ 24)

Tracking Technology Performance Trends

- Drill down into 4 technologies:
  - Disks,
  - Memory,
  - Network,
  - Processors
- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
  - Bandwidth: number of events per unit time
    - E.g., M bits / second over network, M bytes / second from disk
  - Latency: elapsed time for a single event
    - E.g., one-way network delay in microseconds, average disk access time in milliseconds

Latency Lags Bandwidth (for last ~20 years)

- Performance Milestones
  - Disk: 3600, 5400, 7200, 10000, 15000 RPM (dx, 16x)
    - Latency = simple operation w/o contention BW = best case

Disks: Archaic(Nostalgic) v. Modern(Newfangled)

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9500
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003
  - 15000 RPM
  - 73.4 GBytes
  - Tracks/Inch: 800
  - Bits/Inch: 9500
  - Four 2.5” platters
    - Bandwidth: 86 MBytes/sec
    - Latency: 5.7 ms
    - Cache: 8 MBytes

Memory: Archaic (Nostalgic) v. Modern (Newfangled)

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 xtors, 35 mm²
  - 16-bit data bus per module, 16 pins/chip
  - 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
  - 256.00 Mbits/chip
  - 256,000,000 xtors, 204 mm²
  - 64-bit data bus per DIMM, 66 pins/chip
  - 1600 Mbytes/sec
  - Latency: 52 ns
  - Block transfers (page mode)
**Latency Lags Bandwidth (last ~20 years)**

- **Performance Milestones**
  - **Memory Module**: 16-bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM
  - **Disk**: 3600, 5400, 7200, 10000, 15000 RPM

- **LANs**: Archaic (Nostalgic) v. Modern (Newfangled)
  - **Ethernet** 802.3
  - **Year of Standard**: 1978
  - **10 Mbits/s** link speed
  - **Latency**: 3000 µsec
  - **Coaxial cable**: Copper core, 1 mm thick, twisted to avoid antenna effect
  - **Ethernet 802.3ae**
  - **Year of Standard**: 2003
  - **10,000 Mbits/s** (1000X) link speed
  - **Latency**: 190 µsec (15X)

- **CPUs**: Archaic (Nostalgic) v. Modern (Newfangled)
  - **1982 Intel 80286**
  - **12.5 MHz**
  - **2 MIPS (peak)**
  - **Latency**: 320 ns
  - **Microcode interpreter, separate FPU chip**
  - **2001 Intel Pentium 4**
  - **1500 MHz** (120X)
  - **4500 MIPS (peak)** (2250X)
  - **Latency**: 15 ns (20X)

- **Rule of Thumb for Latency Lagging BW**
  - In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4 (and capacity improves faster than bandwidth)
  - Stated alternatively: Bandwidth improves by more than the square of the improvement in Latency
6 Reasons Latency Lags Bandwidth

1. Moore’s Law helps BW more than latency
   • Faster transistors, more transistors, more pins help Bandwidth
     » MPU Transistors: 0.130 vs. 42 M transistors (300X)
     » DRAM Transistors: 0.064 vs. 256 M transistors (4000X)
     » MPU Pins: 68 vs. 423 pins (6X)
     » DRAM Pins: 16 vs. 96 pins (4X)
   • Smaller, faster transistors but communicate over (relatively) longer lines: limits latency
     » Feature size: 1.5 to 3 vs. 0.18 micron (8X, 17X)
     » MPU Die Size: 35 mm² vs. 204 mm² (ratio √ ⇒ 2X)
     » DRAM Die Size: 47 mm² vs. 217 mm² (ratio √ ⇒ 2X)

2. Distance limits latency
   • Size of DRAM block ⇒ long bit and word lines ⇒ most of DRAM access time
   • Speed of light and computers on network
   • 1. & 2. explains linear latency vs. square BW?

3. Bandwidth easier to sell (“bigger=“better”)
   • E.g., 10 Gbits/s Ethernet (“10 Gig”) vs. 10 µsec latency Ethernet
   • 4400 MB/s DIMM (“PC4400”) vs. 50 ns latency
   • Even if just marketing, customers now trained
   • Since bandwidth sells, more resources thrown at bandwidth, which further tips the balance

4. Latency helps BW, but not vice versa
   • Spinning disk faster improves both bandwidth and rotational latency
     » 3600 RPM ⇒ 15000 RPM = 4.2X
     » Average rotational latency: 8.3 ms ⇒ 2.0 ms
     » Things being equal, also helps BW by 4.2X
   • Lower DRAM latency ⇒ More access/second (higher bandwidth)
   • Higher linear density helps disk BW (and capacity), but not disk Latency
     » 9,550 BPI ⇒ 533,000 BPI ⇒ 60X in BW

5. Bandwidth hurts latency
   • Queues help Bandwidth, hurt Latency (Queueing Theory)
   • Adding chips to widen a memory module increases Bandwidth but higher fan-out on address lines may increase Latency

6. Operating System overhead hurts Latency more than Bandwidth
   • Long messages amortize overhead; overhead bigger part of short messages

Summary of Technology Trends

• For disk, LAN, memory, and microprocessor, bandwidth improves by square of latency improvement
  – In the time that bandwidth doubles, latency improves by no more than 1.2X to 1.4X

• Lag probably even larger in real systems, as bandwidth gains multiplied by replicated components
  – Multiple processors in a cluster or even in a chip
  – Multiple disks in a disk array
  – Multiple memory modules in a large memory
  – Simultaneous communication in switched LAN

• HW and SW developers should innovate assuming Latency Lags Bandwidth
  – If everything improves at the same rate, then nothing really changes
  – When rates vary, require real innovation

Acknowledgements

• These slides heavily contain material developed and copyrighted by
  – Krste Asanovic (MIT/UCB)
  – David Patterson (UCB)

• And also by:
  – Arvind (MIT)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)

• MIT material derived from course 6.823
• UCB material derived from course CS252