CDA 5106 Advanced Computer Architecture 1

Module 1 | Fundamentals of Computer Design

Course Overview

- Prerequisite: CDA 4150/EEL 4768: Introduction to Computer Architecture/Computer System Design
- Credits: 3
- Semester: Spring 2009
- Schedule: Tue Thu 4:30pm - 5:45pm
- Room: HEC 0111
- Instructor: Dr. Oleg Kachirski (okachirski@yahoo.com)

2

Textbook

- The required text for the course:

  John L. Hennessy and David A. Patterson, *Computer Architecture - A Quantitative Approach*, 4th Edition, Morgan Kaufmann Publishers Inc., 2006, ISBN: 0-12-370490-1

- A reference text:

  John P. Shen and Mikko H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, McGrawHill, ISBN: 0-07-057064-7

3

At a Glance

Objective: to present in detail how modern computer systems work and are built.

Topics on architecture and organization of modern computing systems: CPU design, instruction sets, superscalar processors and multiprocessors.

Emphasis: pipelining, instruction level parallelism, thread-level parallelism, memory hierarchies, input/output, and network-oriented interconnections.

4

Information Sources

- Textbook material
- Class Notes (Adopted from Prof. David Patterson)
- Current research papers
- In-class 'infosessions' (submit via e-mail or USB flash)
- Class website: http://www.eecs.ucf.edu/courses/cda5106/spr2009/
- The Internet
- Ask questions by e-mail

5

Grading

- Homework assignments: 15%
- In-class Infosessions: 5%
- Mid-term Exam: 25%
- Project/Presentation: 25%
- Final Exam (cumulative): 30%

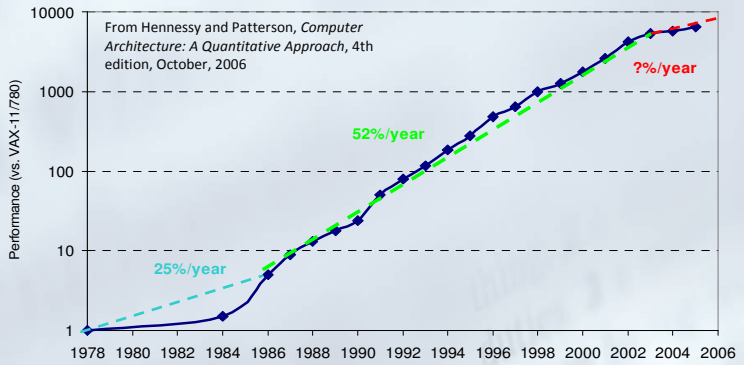
6

Overview

- Computer Design Trends
- Design Characteristics
- Instruction Set Architecture
- Design Trends
- Power Trends
- Cost Trends

7

Crossroads: Uniprocessor Performance



8

Computer Design Facts (Old and New)

- Old: Power is free, transistors expensive
- New: "Power wall" - Power expensive, transistors cheap
- Old: Sufficiently increasing ILP via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New: "ILP wall" - law of diminishing returns on more HW for ILP
- Old: Multiplies are slow, Memory access is fast
- New: "Memory wall" - Memory slow, multiplies fast
  (120 clock cycles to DRAM memory, 4 clocks for multiply)
- Old: Uniprocessor performance 2X / 1.5 yrs
- New: Power Wall + ILP Wall + Memory Wall = ?
  - Uniprocessor performance now 2X / maybe 5 yrs
- Change in chip design: multiple "cores" (2X processors/chip in ~ 2 years)
  - Simpler processors are more power efficient

9

Déjà vu all over again?

- Multiprocessors imminent in 1970s, '80s, '90s, …
- "… today's processors … are nearing an impasse as technologies approach the speed of light.."
  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature
  ⇒ Custom multiprocessors strove to lead uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"
  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies switch to multiprocessors (AMD, Intel, IBM, Sun)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: 1 to 2 CPUs

10

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved without participation of computer architects
- The 4th Edition of textbook (Computer Architecture: A Quantitative Approach) explores shift from Instruction Level Parallelism to Thread Level Parallelism / Data Level Parallelism

11

Design Characteristics

- Depends on an application
- Different computer classes => different design goals
- Flexibility, general performance and affordability
- Low power usage and compact size
- High performance and availability
- Future upgradeability
- Maximum compatibility
- Special purpose applications

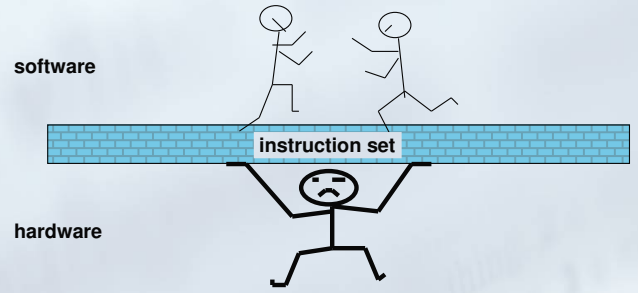
12

Server Availability Effects

Application	Cost of downtime per hour (thousands of $)	Annual losses (millions of $) with downtime of 1% (87.6 hrs/yr)	0.5% (43.8 hrs/yr)	0.1% (8.8 hrs/yr)
Brokerage operations	$6450	$303	$283	$56.5
Credit card authorization	$2600	$228	$114	$22.8
Package shipping services	$150	$13	$6.6	$1.3
Home shopping channel	$113	$9.9	$4.9	$1.0
Catalog sales center	$90	$7.9	$3.9	$0.8
Airline reservation center	$89	$7.9	$3.9	$0.8
Cellular service activation	$41	$3.6	$1.8	$0.4
Online network fees	$25	$2.2	$1.1	$0.2
ATM service fees	$14	$1.2	$0.6	$0.1

13

ISA: 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

14

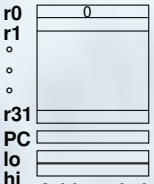
Instruction Set Architecture Dimensions

"... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation." – Amdahl, Blaauw, and Brooks, 1964

- Organization of Programmable Storage
- Data Types & Data Structures: Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions

15

Example ISA: MIPS



Programmable storage

2^32 x bytes

31 x 32-bit GPRs (R0=0)

32 x 32-bit FP regs (paired DP)

HI, LO, PC

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control 32-bit instructions on word boundary

J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

16

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
- Computer architecture >> ISA
- Architect's job is much more than instruction set design; technical hurdles today more challenging than those in IS design

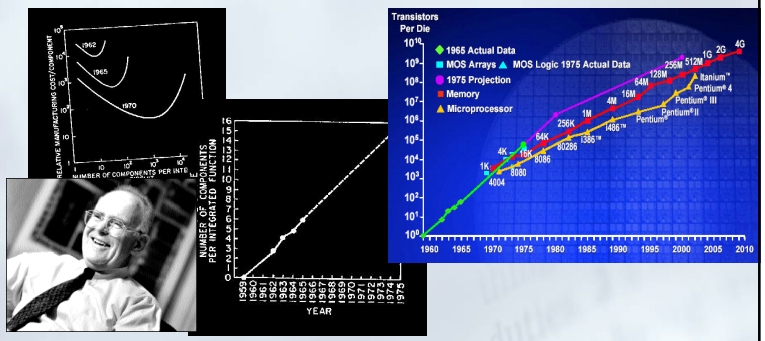
17

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

18

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 ($12 \leq N \leq 24$)

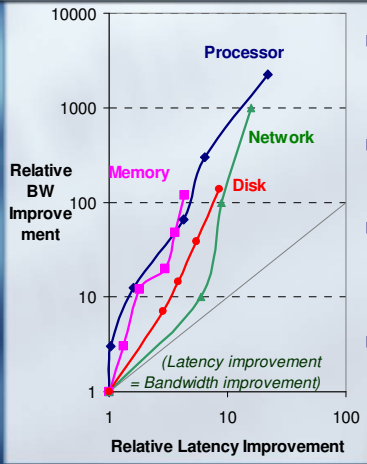
19

Tracking Technology Performance Trends

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

20

Latency Lags Bandwidth (last ~20 years)



- Processor: '286, '386, '486, Pentium, Pentium Pro, Pentium 4 (21x,2250x)
- Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s (16x,1000x)
- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk : 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

21

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 xtors (300X)
  - DRAM Transistors: 0.064 vs. 256 M xtors (4000X)
  - MPU Pins: 68 vs. 423 pins (6X)
  - DRAM Pins: 16 vs. 66 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 vs. 204 mm^2 (ratio sqrt ⇒ 2X)
  - DRAM Die Size: 47 vs. 217 mm^2 (ratio sqrt ⇒ 2X)

22

6 Reasons Latency Lags Bandwidth (cont'd)

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 msec 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

23

6 Reasons Latency Lags Bandwidth (cont'd)

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

24

6 Reasons Latency Lags Bandwidth (cont'd)

5. Bandwidth hurts latency

- Queues help Bandwidth, hurt Latency (Queuing 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
  - 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

Define and quantify power (1 / 2)

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*

$$Power_{dynamic} = 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

- For mobile devices, energy is a better metric

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5V to 1V
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Example of quantifying power

- Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?

$$\begin{aligned} Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched \\ &= 1/2 \times .85 \times CapacitiveLoad \times (.85 \times Voltage)^2 \times FrequencySwitched \\ &= (.85)^3 \times OldPower_{dynamic} \\ &\approx 0.6 \times OldPower_{dynamic} \end{aligned}$$

Define and quantify power (2 / 2)

- Because leakage current flows even when a transistor is off, now *static power* important too

$$Power_{static} = Current_{static} \times Voltage$$

- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In 2006, goal for leakage was 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

29

Cost Trends

- Impacted by R&D costs, time, volume, competition
- IC costs – most variance

$$ICCost = \frac{DieCost + TestingCost + PackagingCost}{FinalTestYield}$$

$$DieCost = \frac{WaferCost}{DiesPerWafer \times DieYield}$$

$$DiesPerWafer = \frac{WaferArea}{DieArea} - \frac{WaferCircumference}{DieDiagonalSize}$$

30

Die Yield

- Empirical
- Based on fabrication process & experience

$$DieYield = WaferYield \times \left(1 + \frac{DefectsPerUnitArea \times DieArea}{\alpha}\right)^{-\alpha}$$

- Wafer yield assumed 100%
- α = 4.0 (as of 2006)

31

Next Class

- Continue with Module 1
- Processor performance
- Amdahl's Law
- Benchmarking
- Principle of Locality

32

Infosessions (Lecture 1)

- Processor fabrication
  - Why wafers are made round?
  - What is 90nm/65nm/45nm fabrication process?
- "Green Computing" primer
  - Eco-friendly materials
  - Power consumption
  - Virtualization technology
- CPU overclocking
  - CPU multiplier / FSB speed; example
  - Memory latency; cooling

33

Review

- Computer Design Trends
- Design Characteristics
- Instruction Set Architecture
- Design Trends
- Power Trends
- Cost Trends

34

In This Lecture (Lecture 2)

- Dependability, MTTF
- Measuring Performance
- Principle of Locality
- Amdahl's Law
- Performance vs. Price-Performance
- Fallacies & Pitfalls in Computer Design

35

Define and Quantify Dependability

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:
  1. Service accomplishment, where the service is delivered as specified in SLA
  2. Service interruption, where the delivered service is different from the SLA
- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

36

Define and Quantity Dependability

- *Module reliability* = measure of continuous service accomplishment (or time to failure).
  2 metrics
1. *Mean Time To Failure* (*MTTF*) measures Reliability
2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation
- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Example Calculating Reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$\begin{aligned} FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\ &= 10 + 2 + 5/1{,}000{,}000 \\ &= 17/1{,}000{,}000 \\ &= 17{,}000\,FIT \\ MTTF &= 1{,}000{,}000{,}000/17{,}000 \\ &\approx 59{,}000\,hours \end{aligned}$$

Measuring Performance

- Benchmarking various components of a computer
- Summarizing performance as a score

$$n = \frac{Performance_X}{Performance_Y}$$

- Desktop Benchmarks
  - FLOPS, MIPS, Graphics subsystem
  - SPEC 2006
- Server Benchmarks
  - Transactions, throughput
  - TPC

Summarizing Performance Results

- SPECRatio - geometric mean of scores:

$$GeometricMean = \sqrt[n]{\prod_{i=1}^{n} sample_i} = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln(sample_i)\right)$$

$$GeometricStDev = \exp\left(\sqrt{\frac{\sum_{i=1}^{n} (\ln(sample_i) - \ln(GeometricMean))^2}{n}}\right)$$

- GeometricStDev shows lognormal distribution compatibility

Computer Architecture Realm

- 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, quantity, and summarize relative performance
  - Define and quantity relative cost
  - Define and quantity dependability
  - Define and quantity power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked

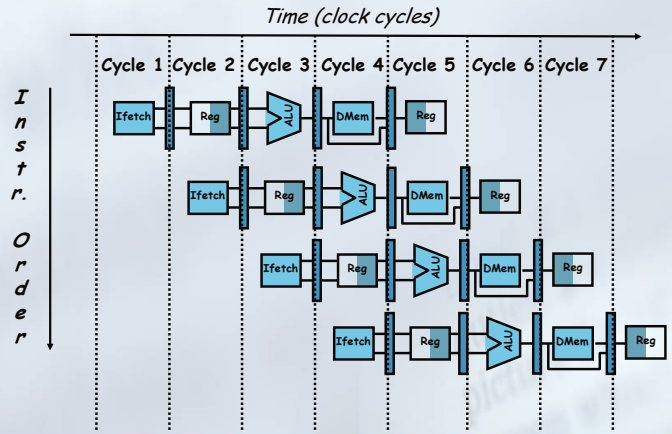
41

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)

42

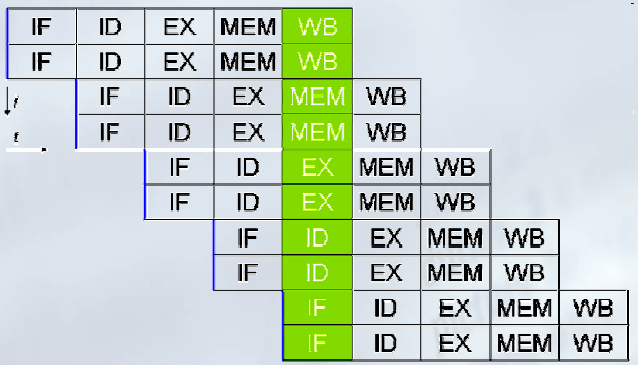
Pipelined Instruction Execution



43

Superscalar Processors

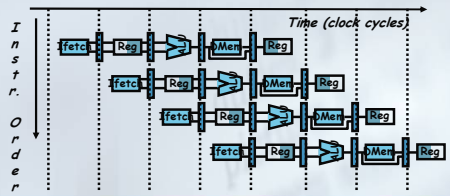
- Simultaneously dispatch multiple instructions to redundant functional units on the processor



44

Limits to Pipelining

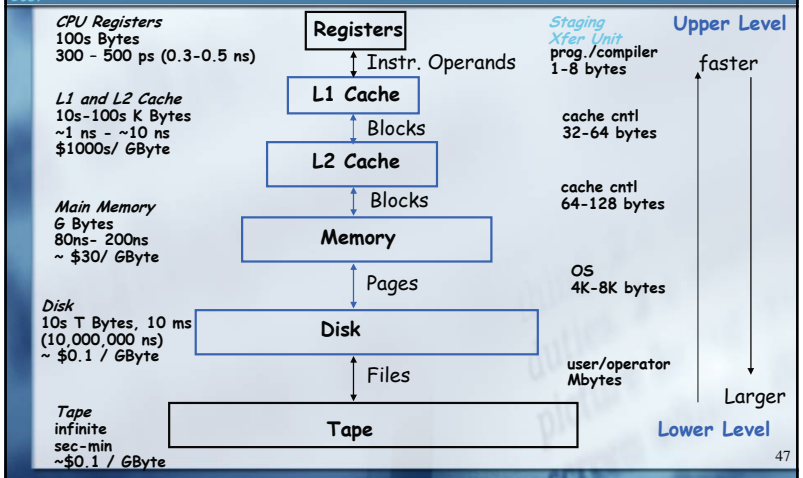
- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: attempt to use the same hardware to do two different things at once
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).



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



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

Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{(1 - Fraction_{enhanced})}$$

Amdahl's Law Example

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

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

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

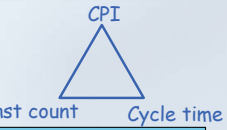
Processor Performance Equation

$$CPUtime = \frac{CPUClockCycles}{ClockRate}$$

$$CPI = \frac{CPUClockCycles}{IC}$$

$$CPUClockCycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

Processor Performance Equation



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization	X		X
Technology			X

Key Points

- Choose features to optimize accurately
- Reproduce *every* component for fault tolerance
- Cost-effective design accounts for *all* system components
- Compare “apples to apples” when benchmarking
- Beware of marketing definitions (e.g., MFFT)

53

Infosessions

- FPGA Technology
  - Architecture
  - Applications
  - Programming/HDL
- Cloud Computing
  - Concepts
  - Compare with SaaS?
  - Example (Amazon EC2)

54