CSE 3038 - Computer Organization

Prof. Haluk Topcuoglu

Chapter 1 – Computer Abstractions and Technology

[Adapted from:
Morgan Kaufmann Slides,
Prof. Dan Garcia (@Berkeley)
Prof. M. J. Irwin (@PennState)]

Spring 2022

Course Information

- Instructor: (Prof. Haluk Topcuoglu)

  http://mimoza.marmara.edu.tr/~haluk/

- Teaching Assistant: (Lokman Altın)

- Textbook: Patterson & Hennessy, *Computer Organization and Design*, 6th Edition (MIPS version)

  (5th or 6th edition required. Do not get the previous editions)

  Average 35-40 pages of reading/week !!

Course Organization

- Grading (revised)
  - Attendance (Pop-up Quizzes) (5%)
  - Homework (Related Quizzes) (15 %)
  - Projects (20%) (*2 projects planned*)
  - Midterm (20%)
  - Final (40%)

- Separate project quizzes will be scheduled.
- Attend to all lectures/problem sessions on time !!!

Policies on Projects and Exams

- There will be a set of homeworks throughout the semester; but they will not be graded.
  - *At least **3** quizzes (in place of HWs) planned throughout the semester.*
- All projects will be done with a group of **3** students.
  - You will select your partners and the group will not be changed throughout the semester.
- Copying (partially or full) from other students *is a form of cheating.*
- Copying (partially or full) solutions from Web including Github (and similar sites) *is another form of cheating.*
- In case of any forms of cheating in projects/quizzes/exams, the penalties will be severe.
- ***At the minimum: F in the course, and a letter to your university record documenting the incidence of cheating.***
- ***Both Giver and Receiver are equally culpable and suffer equal penalties***

Submitting Projects

- Projects will be submitted electronically from the Canvas system.
- You can access all class material from the Canvas account.
  - Canvas – A Learning Management System
  - Registration required
  - Registration email will be sent by the end of this week !!.

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Computers used for running larger programs for multiple users, often simultaneously
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

Classes of Computers

- Supercomputers
  - Consists of thousands of processors and many terrabytes of memory
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

Supercomputers in the World

TOP 500 List (November 2021)

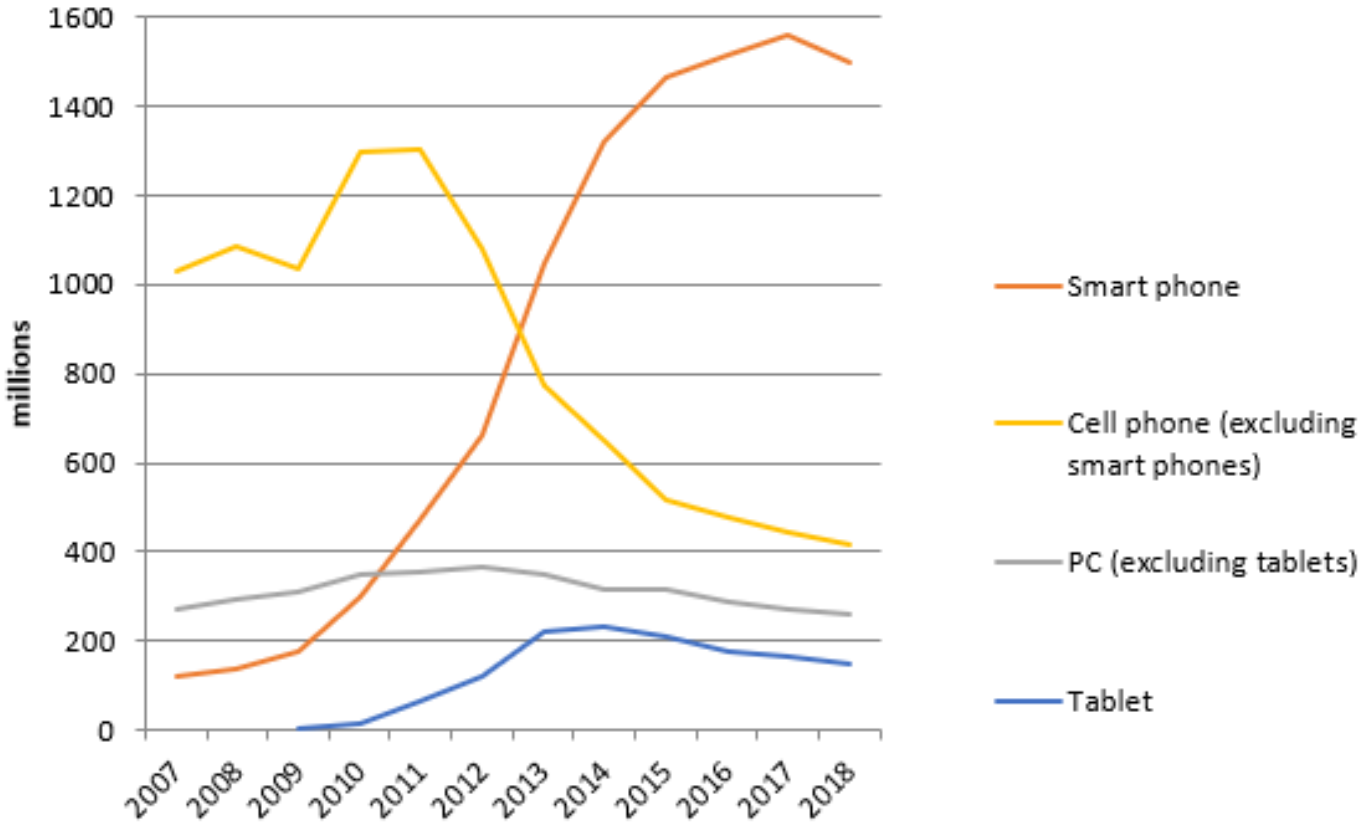
Fugaku is a supercomputer developed by Fujitsu for use at Riken Center for Computationl Science (Japan), which as of November 2021 is the fastest supercomputer in the world, capable of 442 petaFLOPS.

There is also a Green List
Power Efficiency (Gflops / watts)

Green 500 List (November 2021)

Name	Unit	Value
kiloFLOPS	kFLOPS	10^3
megaFLOPS	MFLOPS	10^6
gigaFLOPS	GFLOPS	10^9
teraFLOPS	TFLOPS	10^{12}
petaFLOPS	PFLOPS	10^{15}
exaFLOPS	EFLOPS	10^{18}
zettaFLOPS	ZFLOPS	10^{21}
yottaFLOPS	YFLOPS	10^{24}

The PostPC Era



The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses

- Cloud computing
  - Portion of software run on a PMD and a portion run in the Cloud
  - Warehouse Scale Computers (WSC)
    - Data centers containing 100 000 servers
    - Amazon and Google cloud vendors

What You Will Learn

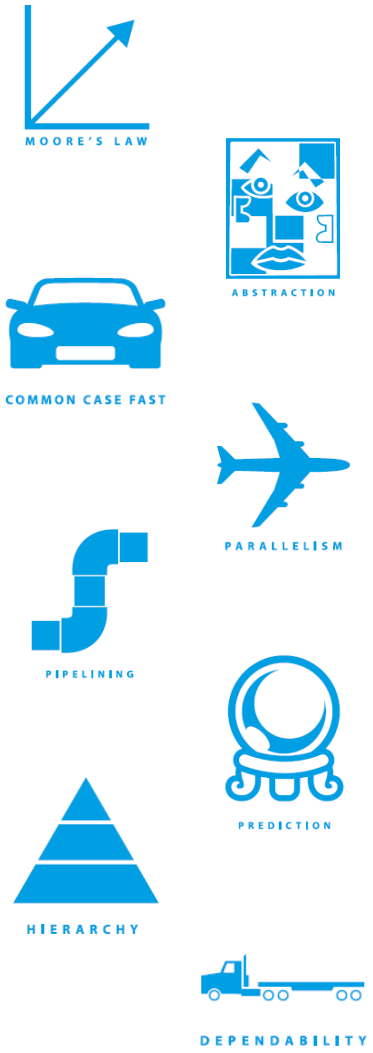
- How programs are translated into the machine language
  - and how the hardware executes them
- The hardware/software interface
- What determines program performance
  - and how it can be improved
- How hardware designers improve performance (caches, pipelines)
- What is parallel processing (multicore)

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Eight Great Ideas

1. Use ***abstraction*** to simplify design
2. Design for ***Moore's Law***
3. Make the ***common case fast***
4. Performance *via* ***parallelism***
5. Performance *via* ***pipelining***
6. Performance *via* ***prediction***
7. ***Hierarchy*** of memories
8. ***Dependability*** *via* redundancy



#1: Abstraction

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

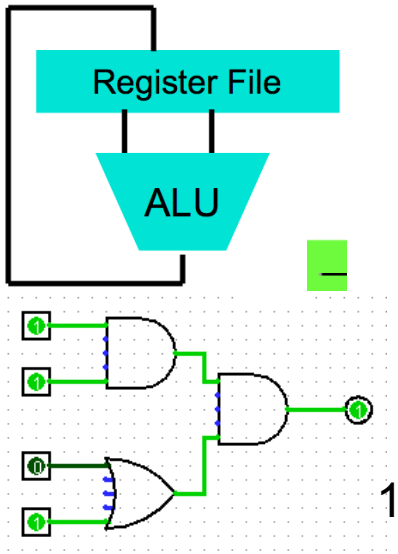
Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

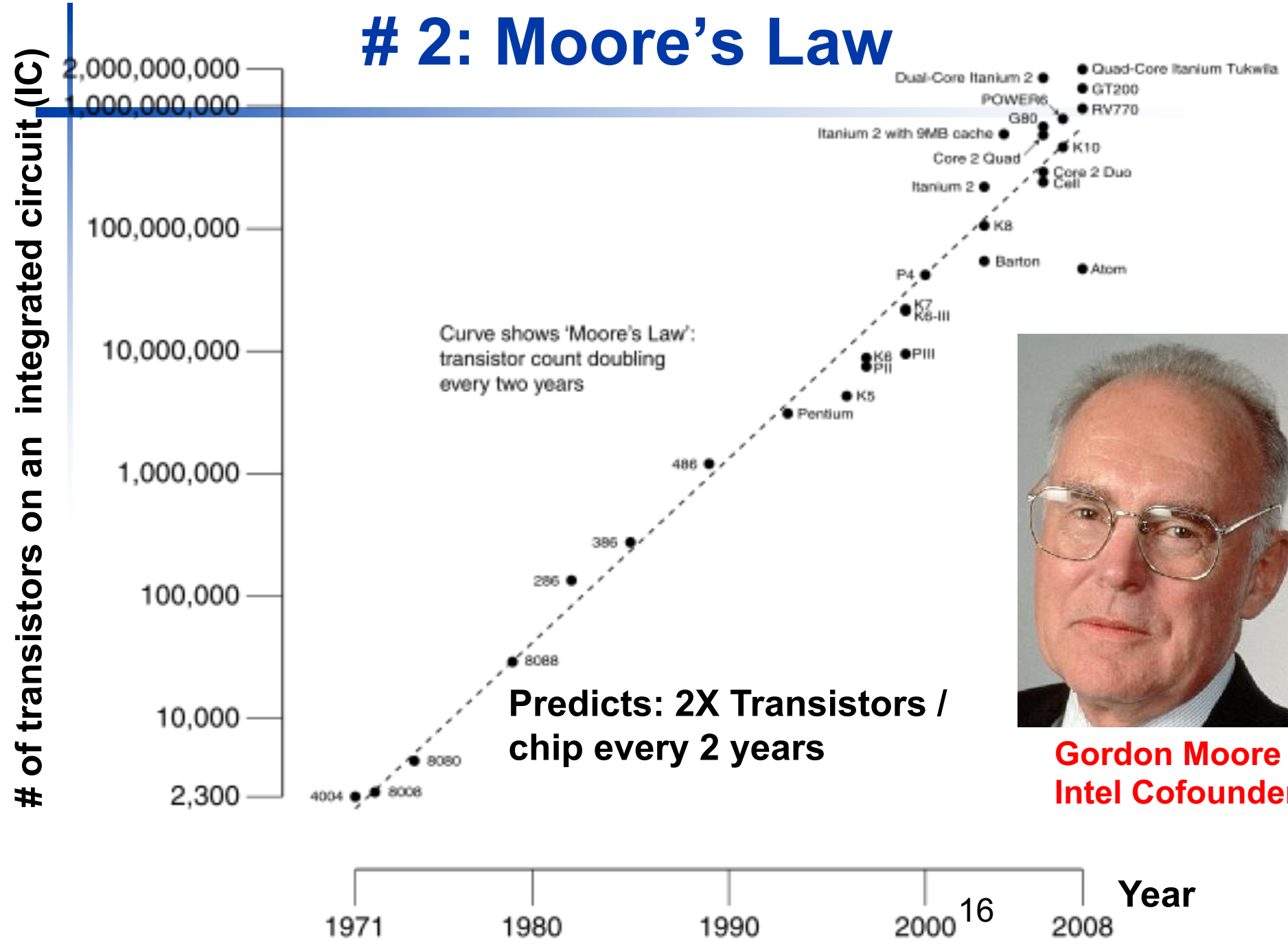
```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



2: Moore's Law



Gordon Moore
Intel Cofounder

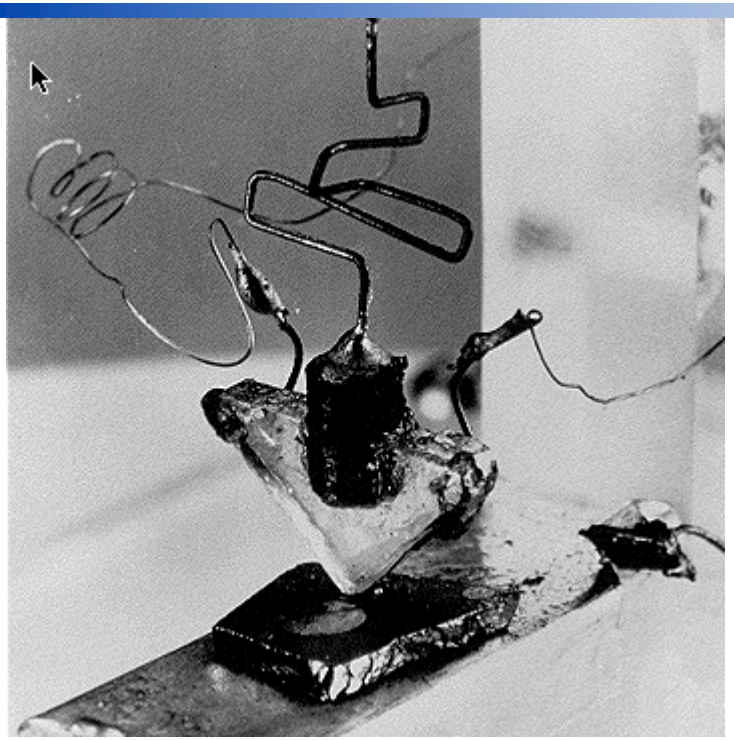
Moore's Law

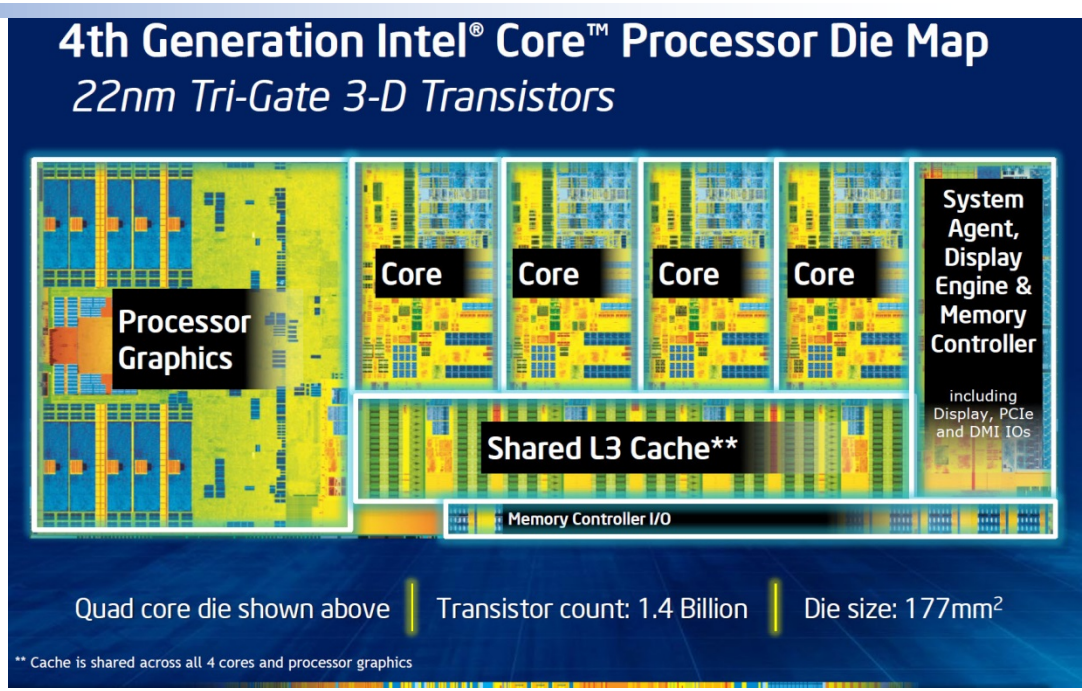
in 1965 (By Gordon Moore)

- number of transistors that can be integrated on a die would double every 18 to 24 months (i.e., grow exponentially with time)

- **Amazingly visionary**
  - 2300 transistors, 1 MHz clock (Intel 4004) - 1971
  - 16 Million transistors (Ultra Sparc III)
  - 42 Million transistors, 2 GHz clock (Intel Xeon) – 2001
  - 55 Million transistors, 3 GHz, 130nm technology, 250mm^2 die (Intel Pentium 4) – 2004
  - 290+ Million transistors, 3 GHz (Intel Core 2 Duo) – 2007
  - 721 Million transistors, 2 GHz (Nehalem) - 2009
  - 1.4 Billion t, 3.4 GHz Intel Haswell (Quad core) – 2013

Then and Now





http://techguru3d.com/4th-gen-intel-haswell-processors-architecture-and-lineup/

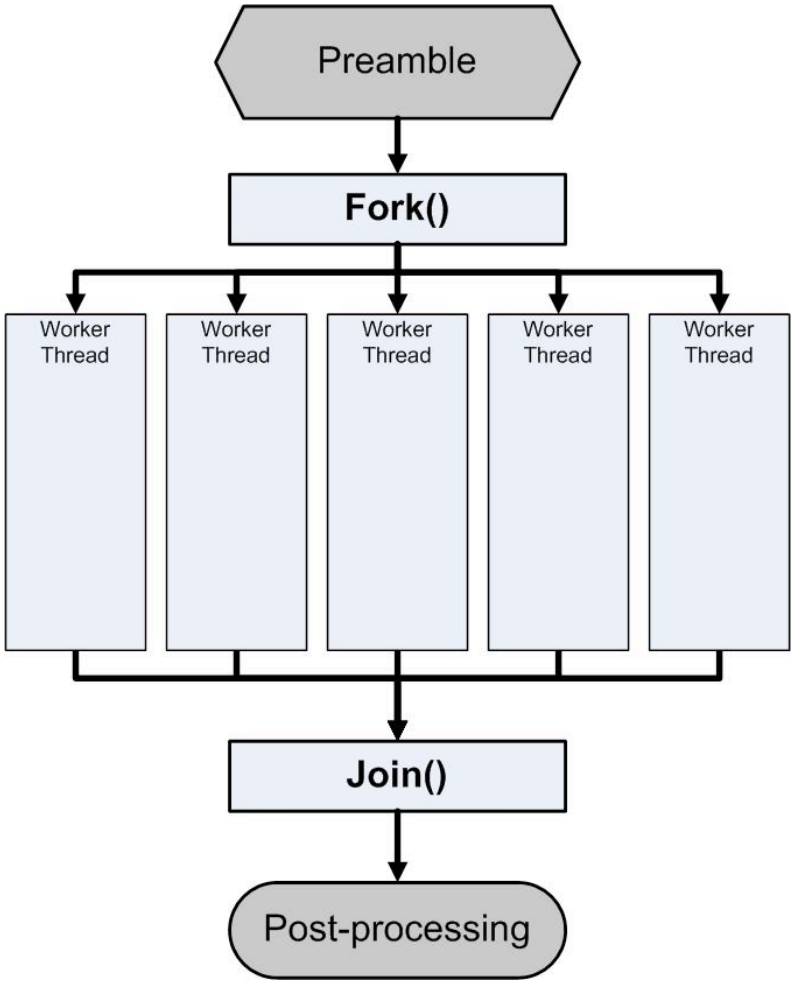
- The first transistor
  - One workbench at AT&T Bell Labs
  - 1947
  - Bardeen, Brattain, and Shockley
- An Intel Haswell
  - 1.4 billion transistors
  - 177 square millimeters
  - Four processing cores

#3 : Make the Common Case Fast

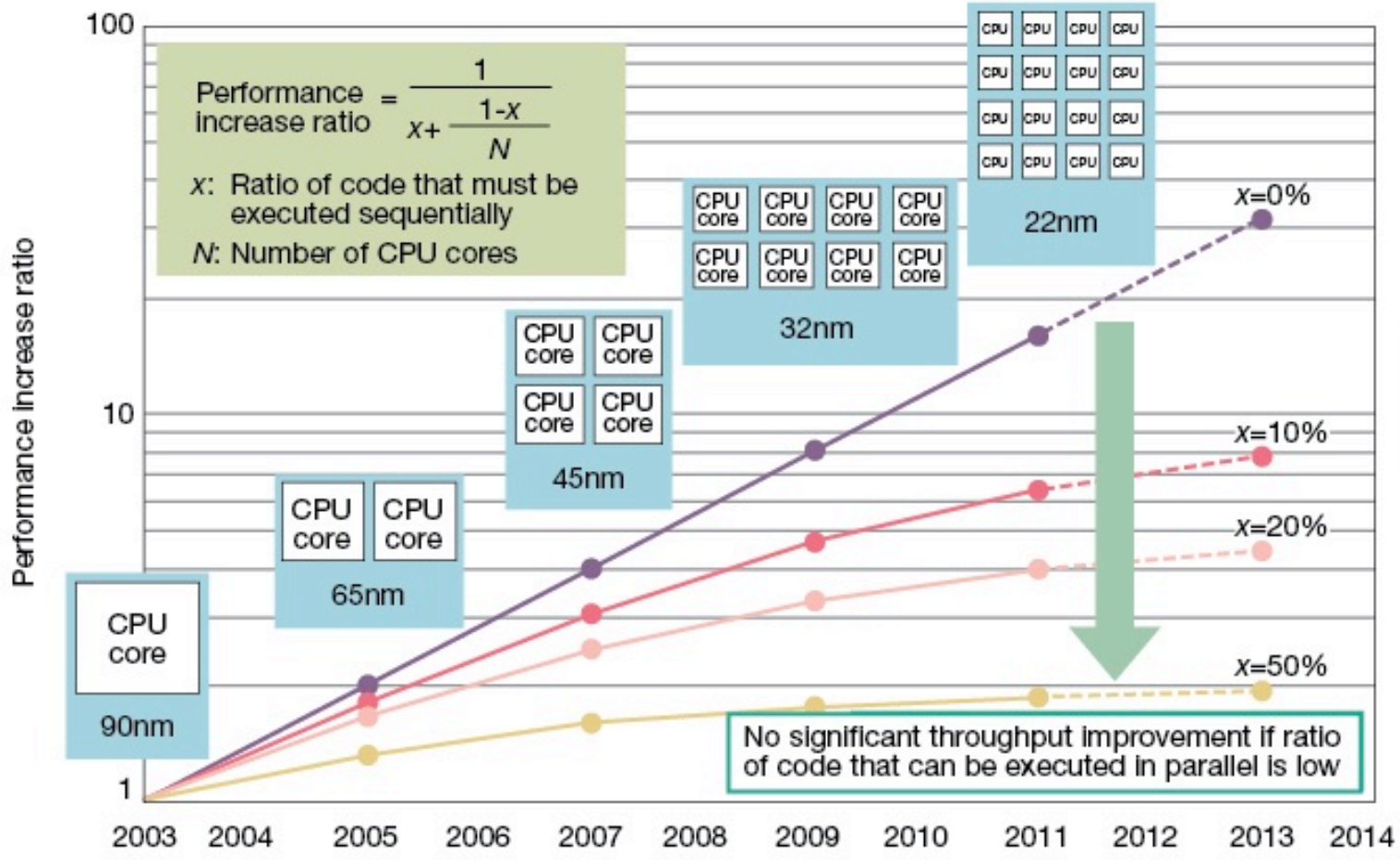
- Designing proper instruction formats
- Designing proper addressing modes

4: Parallelism

- Instruction-Level Parallelism
- Data-Level Parallelism
- Thread-Level Parallelism



*** Amdahl's Law

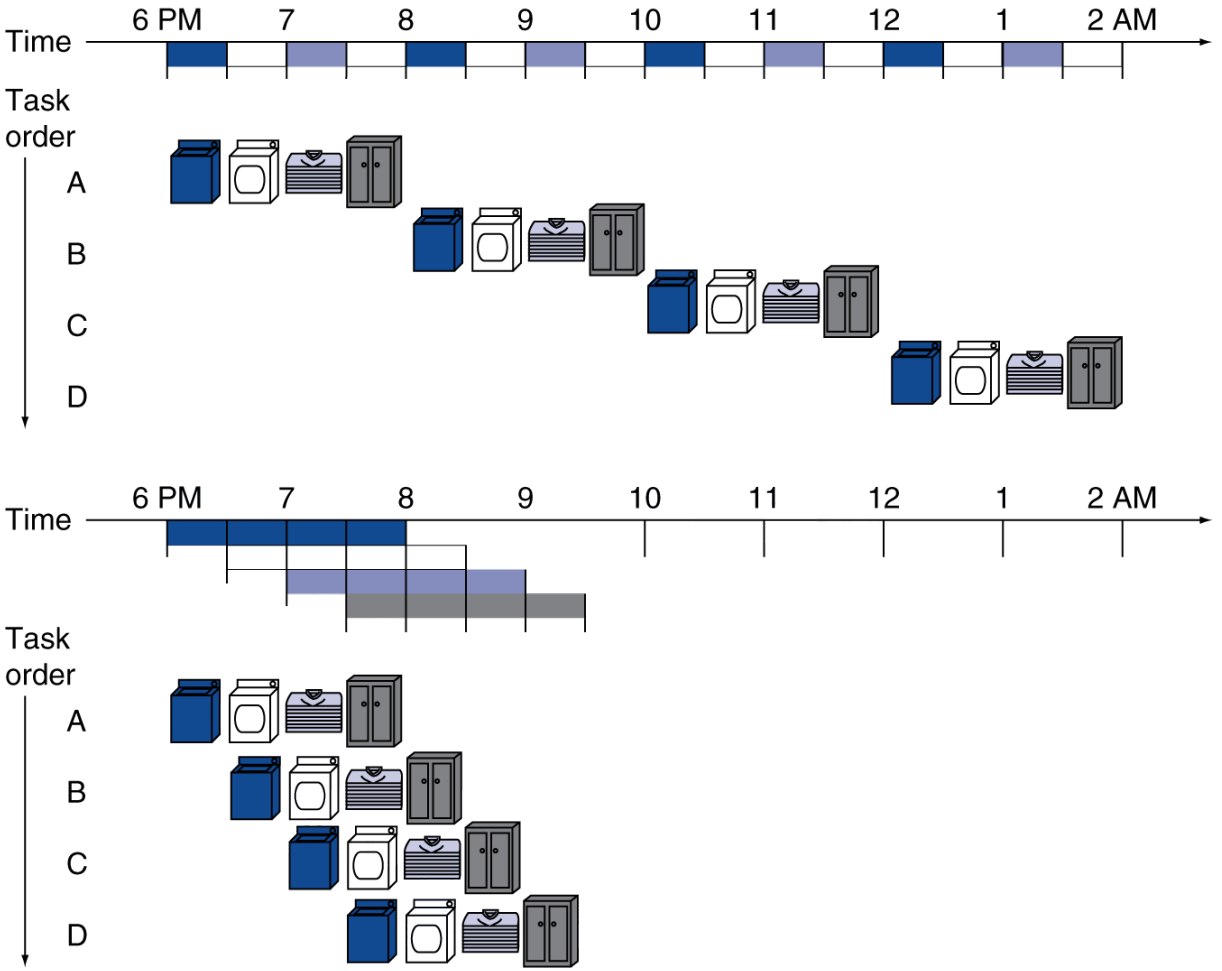


Gene Amdahl

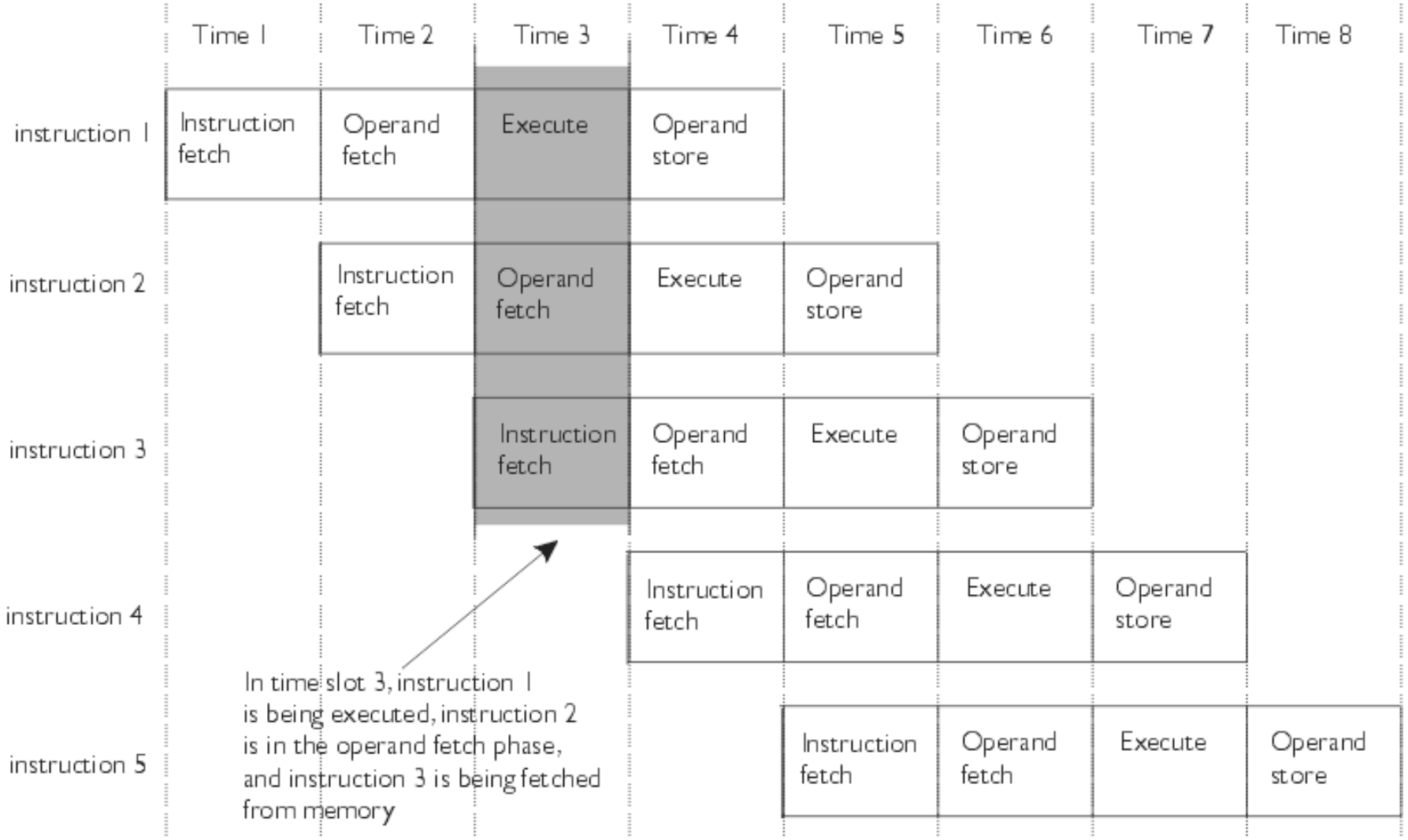
Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

#5 Pipelining (A Laundry Analogy)

- Pipelined laundry: (wash,dry, fold, store)
- overlap execution - Parallelism improves performance



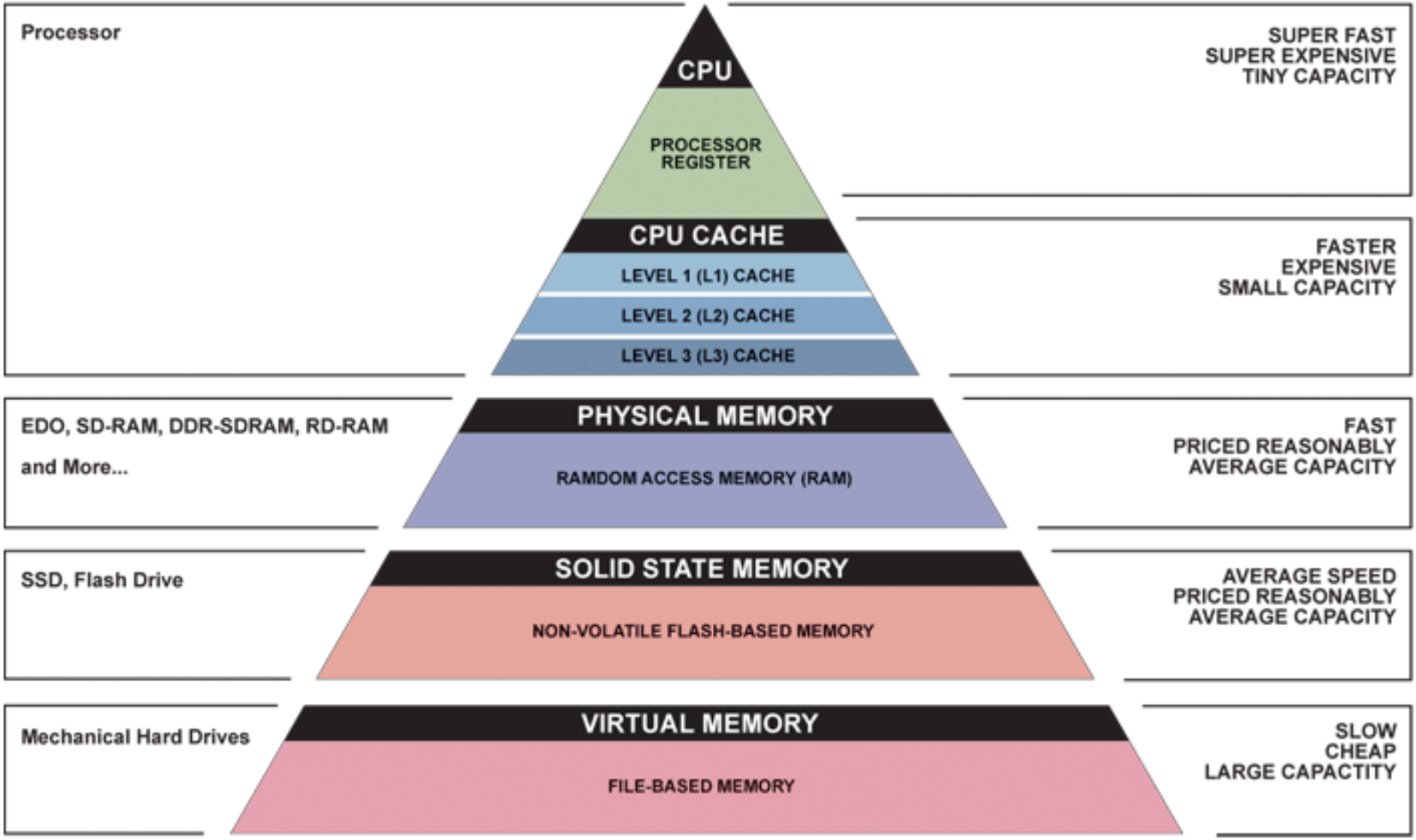
#5: Pipelining



#6 : Prediction

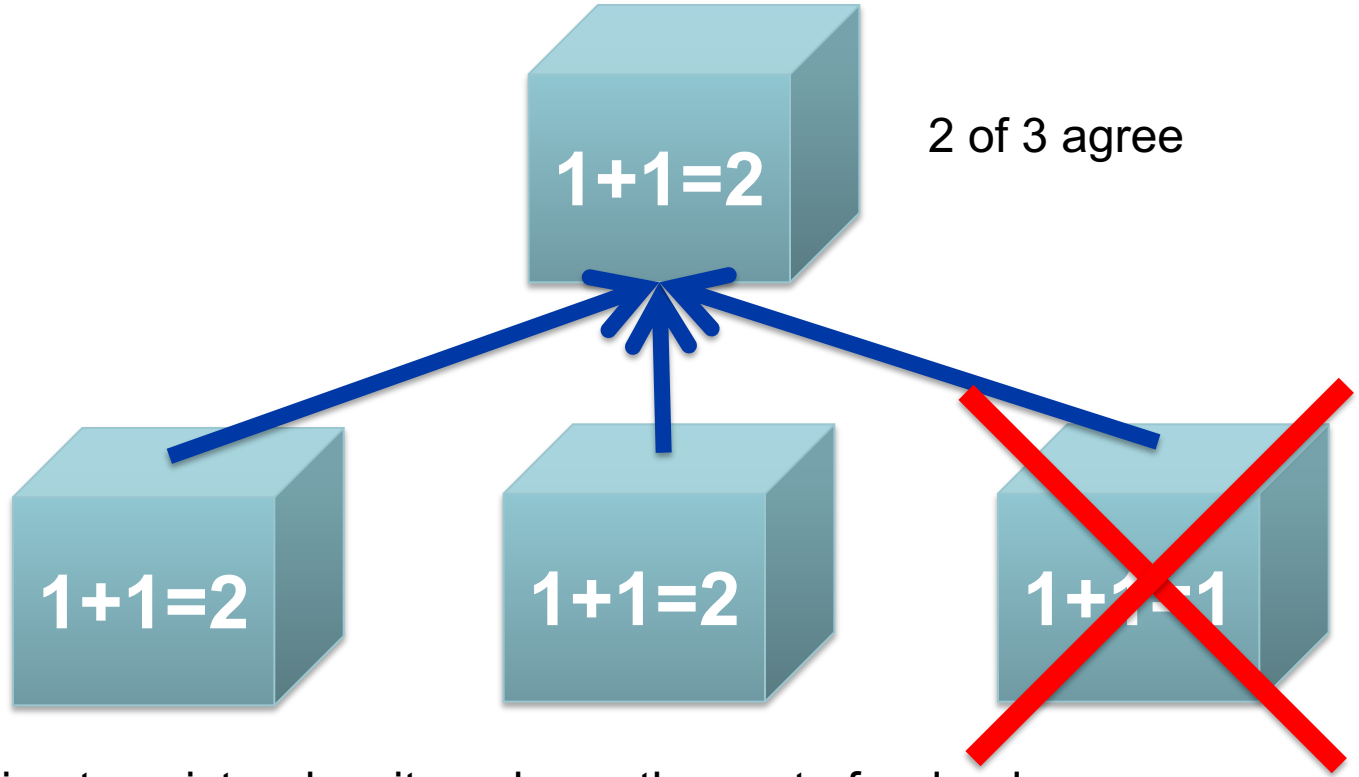
- Executing Instructions speculatively using prediction
- Branch Instructions (Predict to be taken)

#7: Hierarchy of Memories



#8: Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail

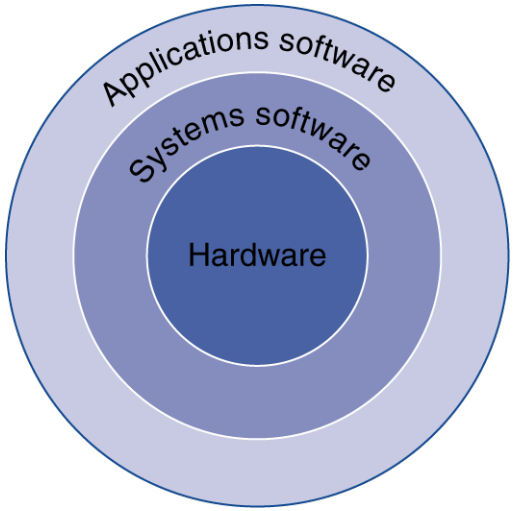


Increasing transistor density reduces the cost of redundancy

#8: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory to instructors
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

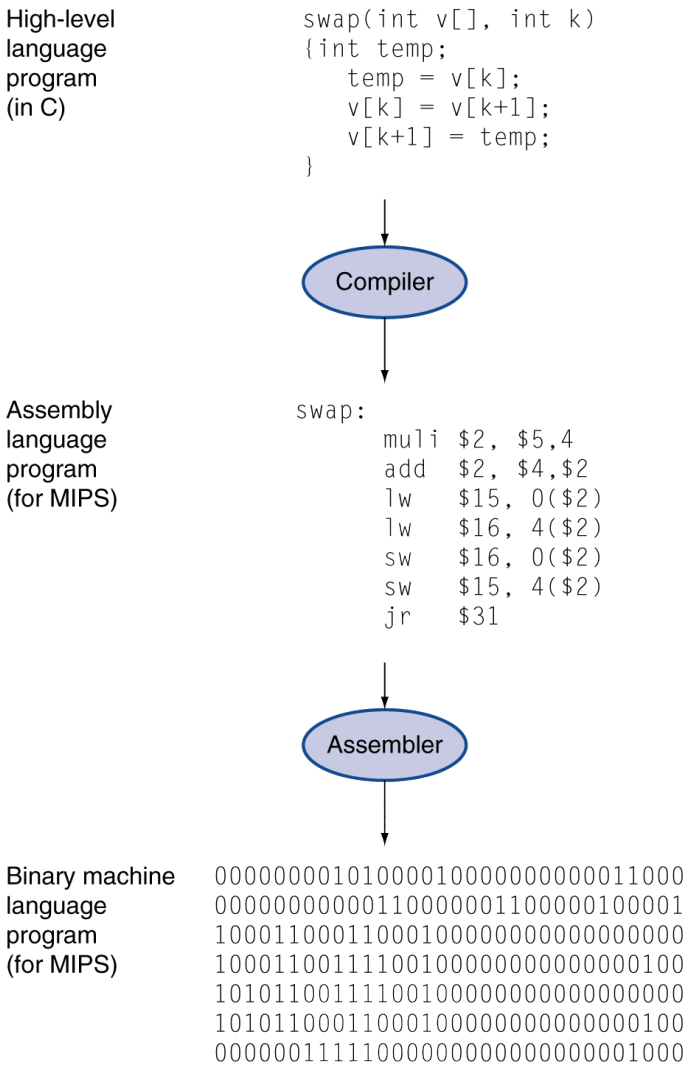
Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

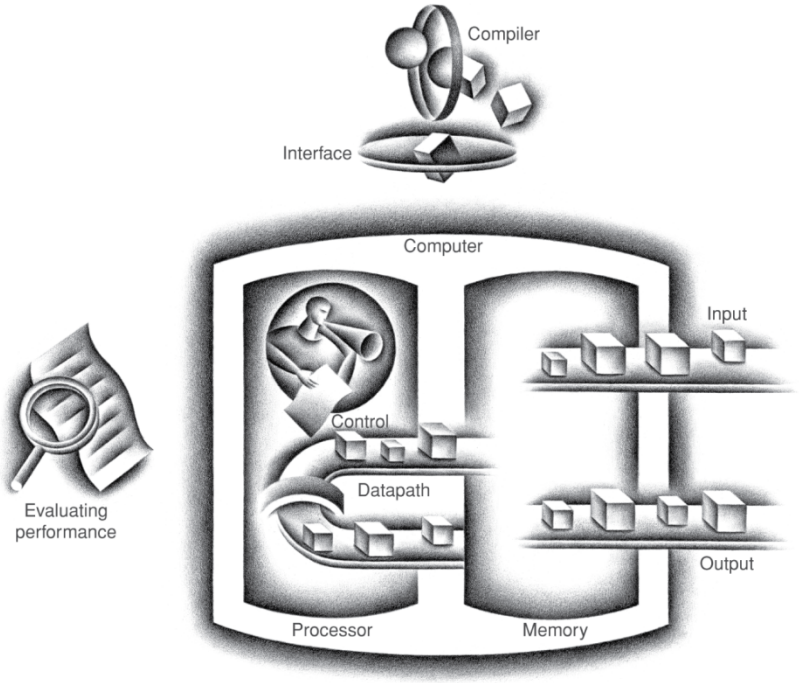


Advantages of Higher-Level Languages

- Allow the programmer to think in a **more natural language** and for their intended use (Fortran for scientific computation, Lisp for symbol manipulation, Java for web programming, …)
- Improve **programmer productivity** – more understandable code that is easier to debug and validate
- Improve **program maintainability**
- Allow programs to be **independent of the computer** on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
- Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine
- *As a result, very little programming is done today at the assembler level*

Components of a Computer

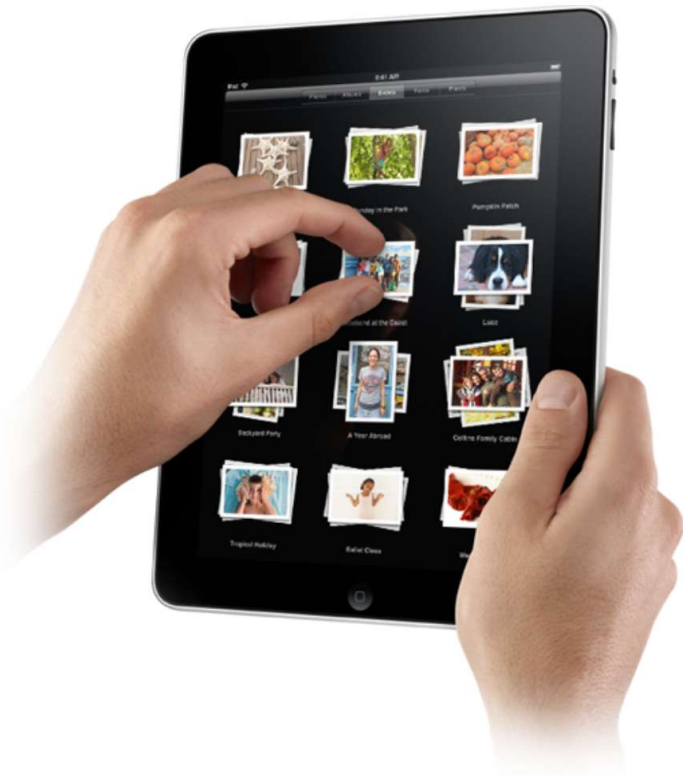
The BIG Picture



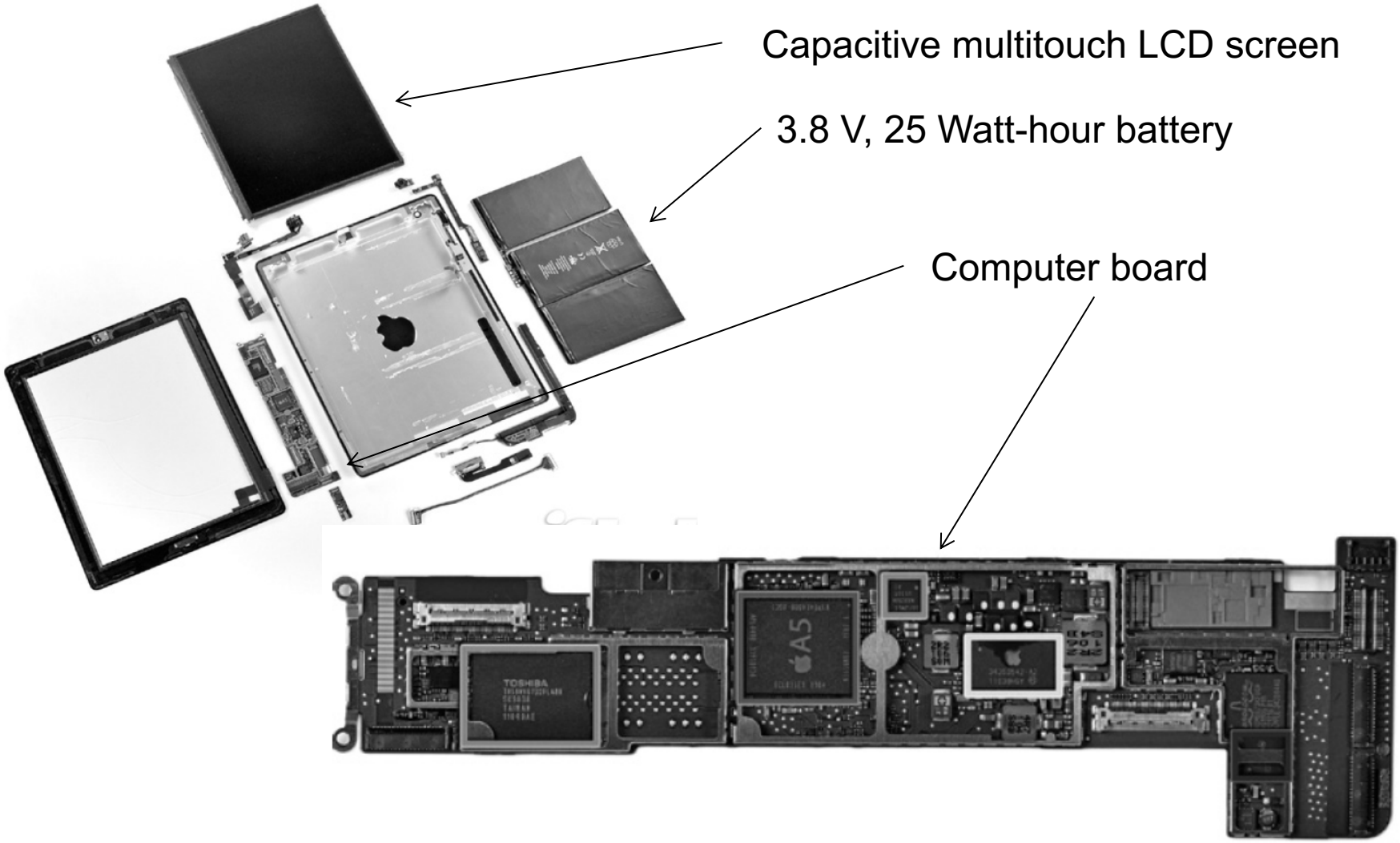
- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



Opening the Box



Logic Board of Apple iPad 2

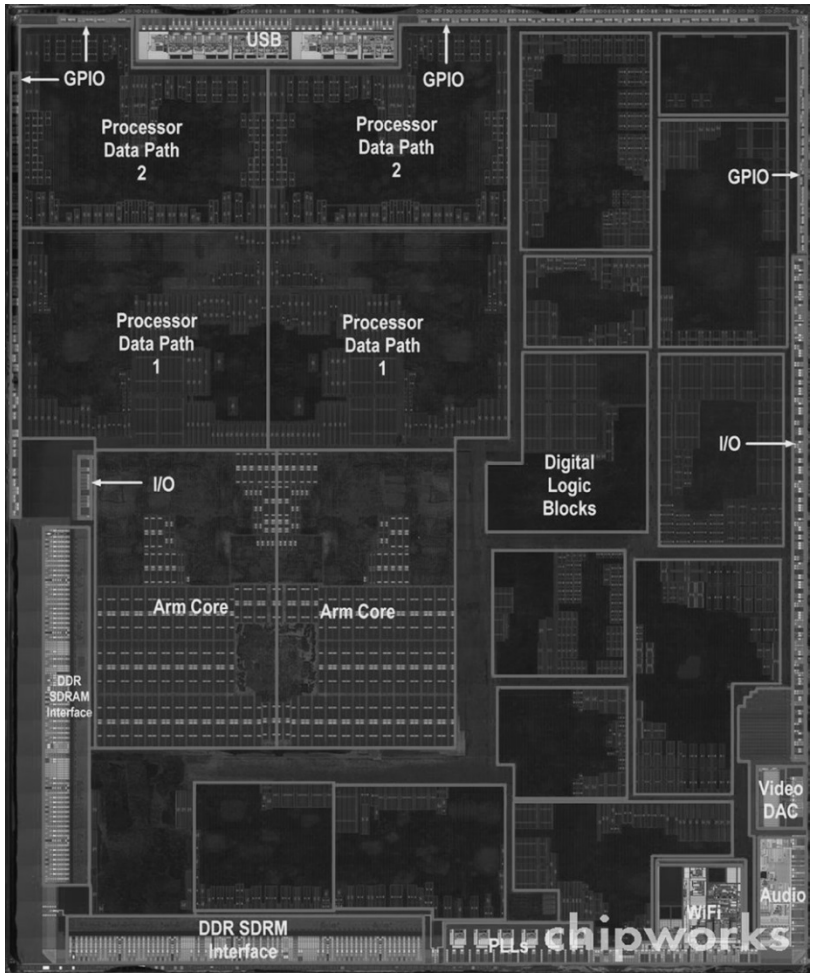
- – Apple A5 – Dual-core Processor
- – Samsung K9PFG08U5A –Multi-Level Cell (2 bit-per-cell) NAND Flash Memory
- – Broadcom BCM5973 - I/O controller
- – Texas Instruments CD3240B - Touchscreen Line Driver
- – Broadcom BCM5974- Touchscreen Controller
- – Apple 3430542 - Dialog Semiconductor D1946A - Power Management
- – Apple 338SC940 - Cirrus Logic CLI1S546A0 - Audio Codec

Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

Inside the Processor

- Apple A5



Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

Computer Architecture

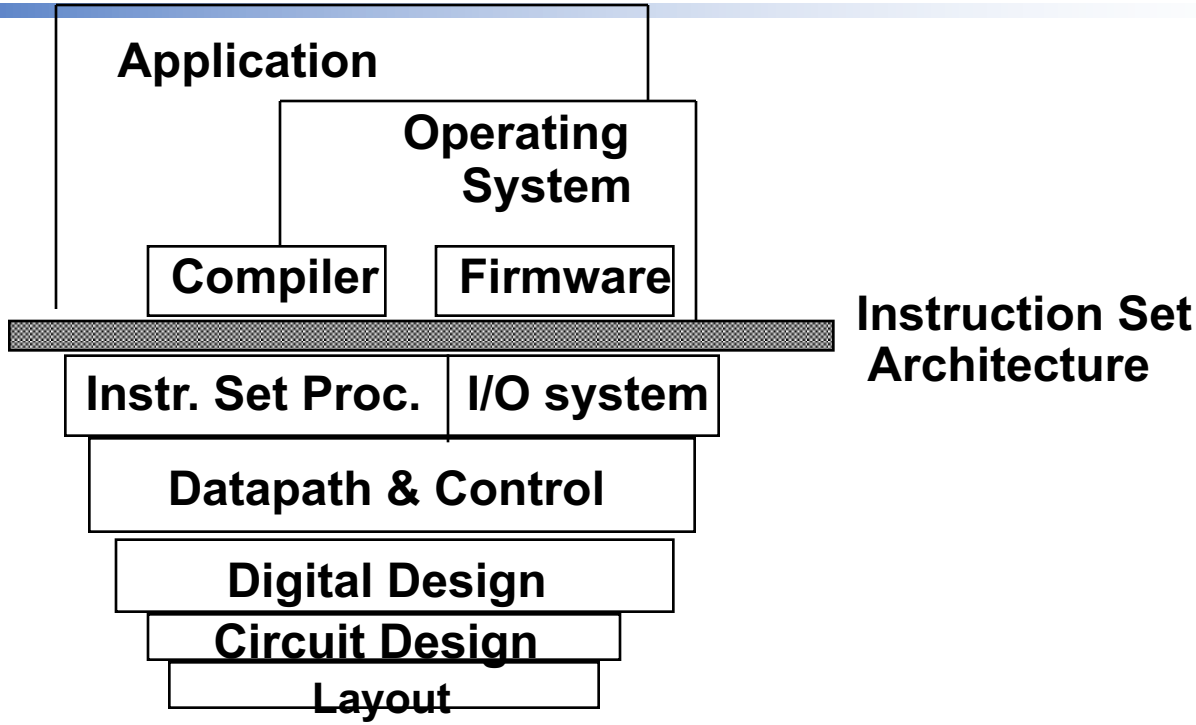
Computer Architecture =

Instruction Set Architecture + Machine Organization

what the machine looks like

How you talk to machine

Instruction Set Architecture



- A very important abstraction
  - interface between hardware and low-level software
  - standardizes instructions, machine language bit patterns, etc.
  - advantage: *different implementations of the same architecture*

Instruction Set Architecture

An abstract interface between the hardware and the lowest level software of a machine that encompasses all information necessary to write a machine language program that will run correctly

-- Organization of Programmable Storage

-- Data Types & Data Structures: Encodings & Representations

-- Instruction Set

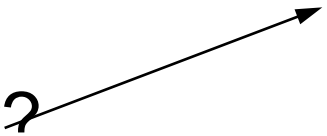
-- Instruction Formats

-- Modes of Addressing and Accessing Data Items and Instructions

Key ISA Decisions

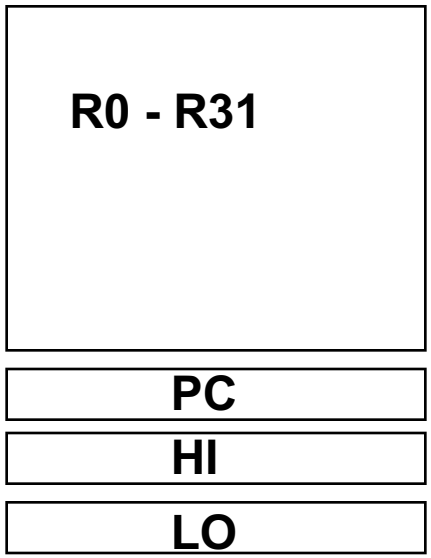
- Operations
  - How many and which ones
- Operands
  - How many
  - Location
  - Types
  - How to specify?
- Instruction format
  - size
  - How many formats?

How does the computer know what 0001 0100 1101 1101 means



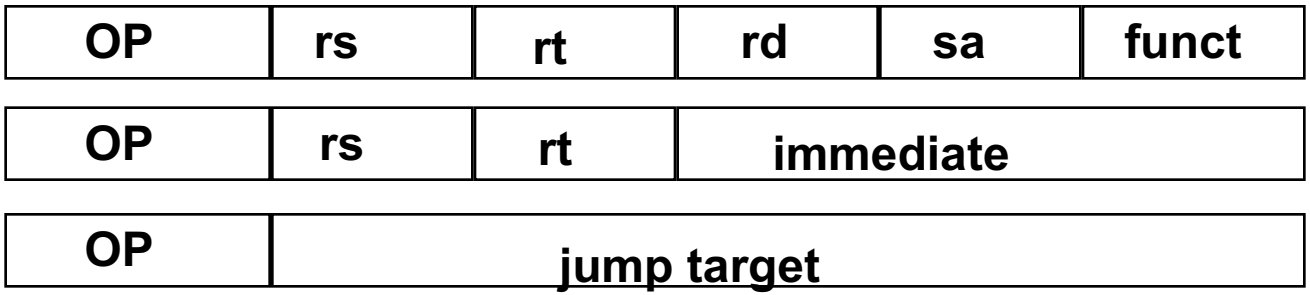
MIPS Instruction Set Architecture

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
  - Memory Management
  - Special



Registers

3 Instruction Formats: all 32 bits wide



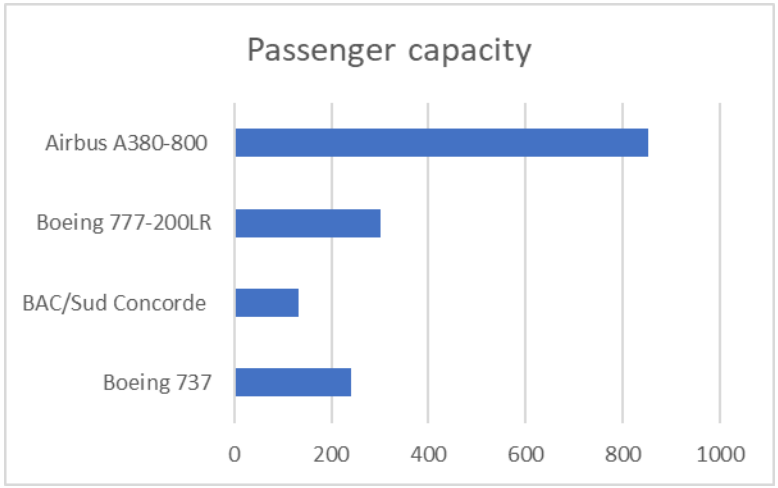
Computer Organization

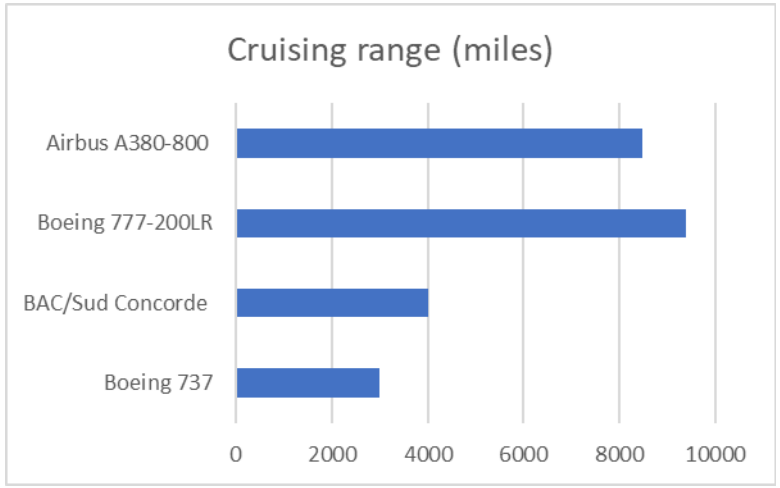
- **Capabilities & Performance Characteristics of Principal Functional Units**
  - **(e.g., Registers, ALU, Shifters, Logic Units, ...)**
- **Ways in which these components are interconnected**
- **Information flows between components**
- **Logic and means by which such information flow is controlled.**

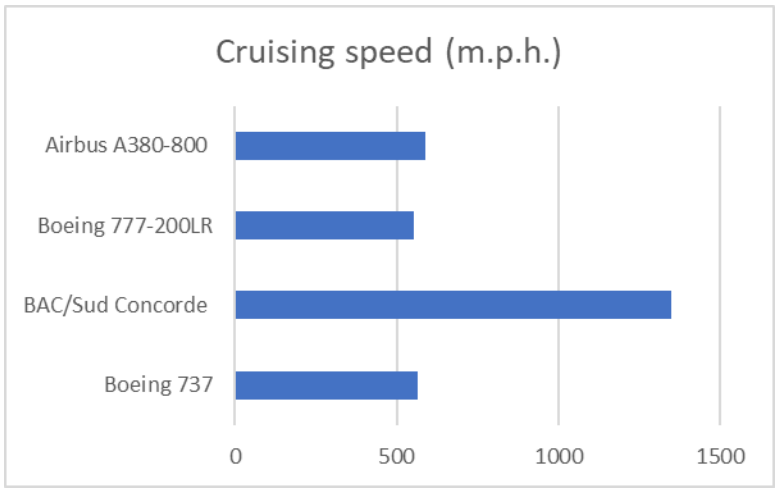
Once you have decided on an ISA, you must decide how to design the hardware to execute those programs written in ISA as fast as possible (or as cheaply as possible or using as little power as possible?)

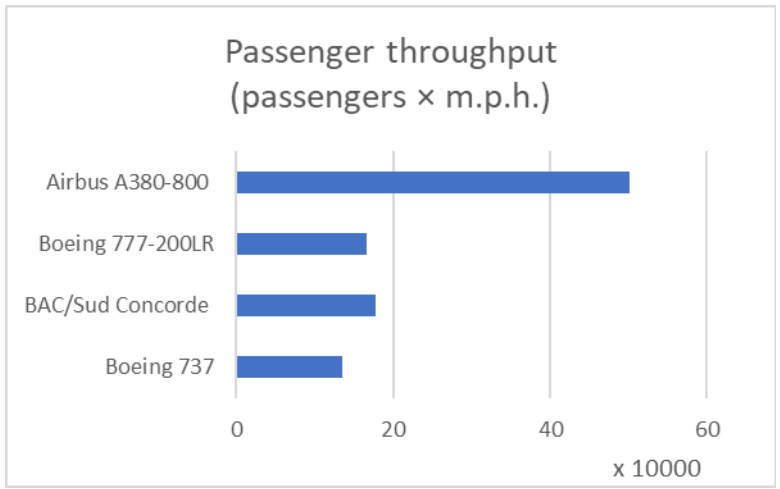
Defining Performance

- Which airplane has the best performance?









Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding a new machine or more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

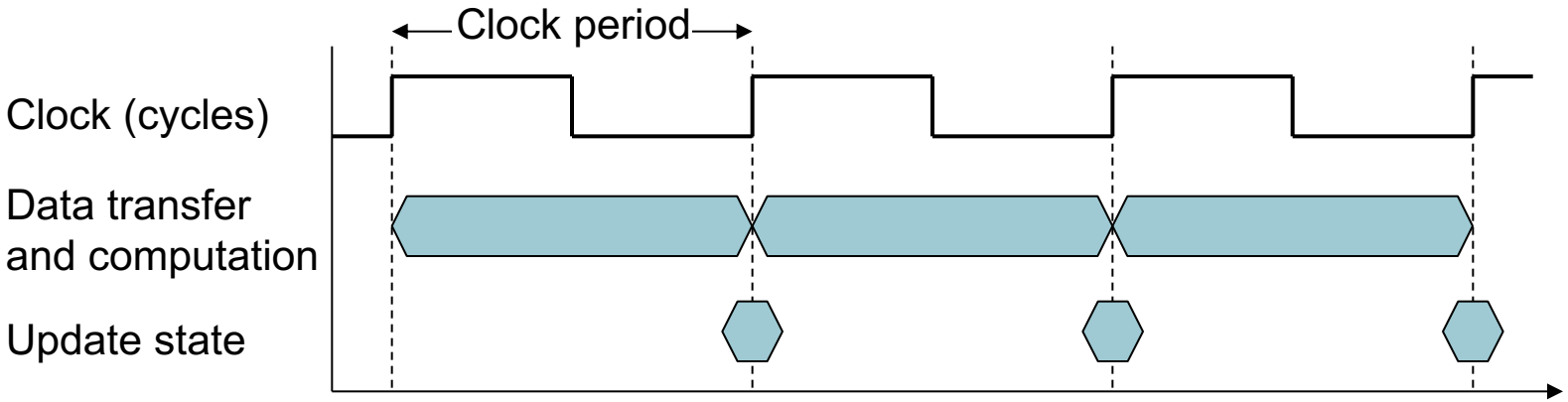
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



  - Clock period: duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Clock frequency (rate): cycles per second
    - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example (page 34)

- A program runs 10s on Computer A (2GHz clock)
- Designing Computer B
  - Aim for 6s CPU time for the same program
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

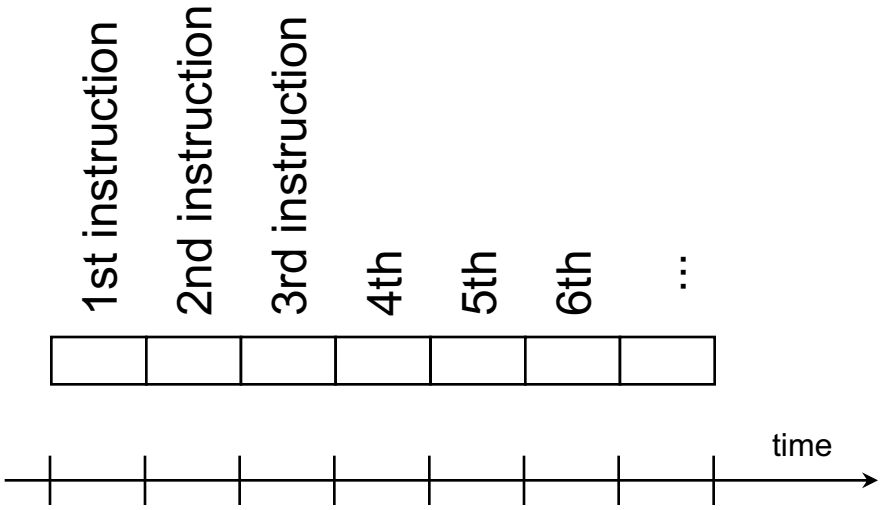
$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

How many cycles are required for a program?

- Could assume that number of cycles equals number of instructions

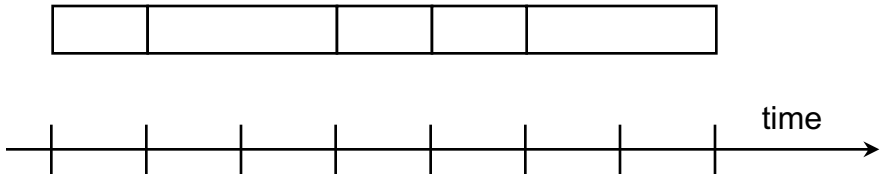


This assumption is incorrect

different instructions take different amounts of time on different machines.

Why? these are machine instructions, not lines of C code

Different numbers of cycles for different instructions



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers

- *Important point: changing the cycle time often changes the number of cycles required for various instructions*

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example (page 35)

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example (page 37)

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

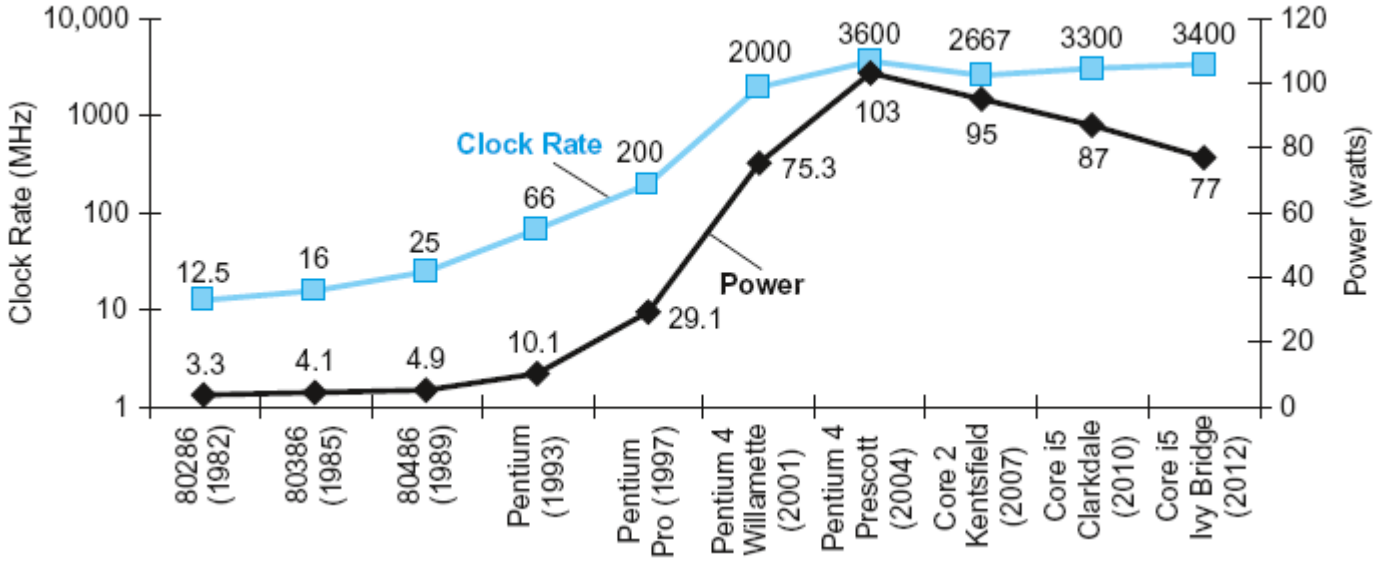
Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI (*ex. more divides*)
  - Programming language: affects IC, CPI (*ex. Java has indirect calls; higher CPI instr.*)
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000

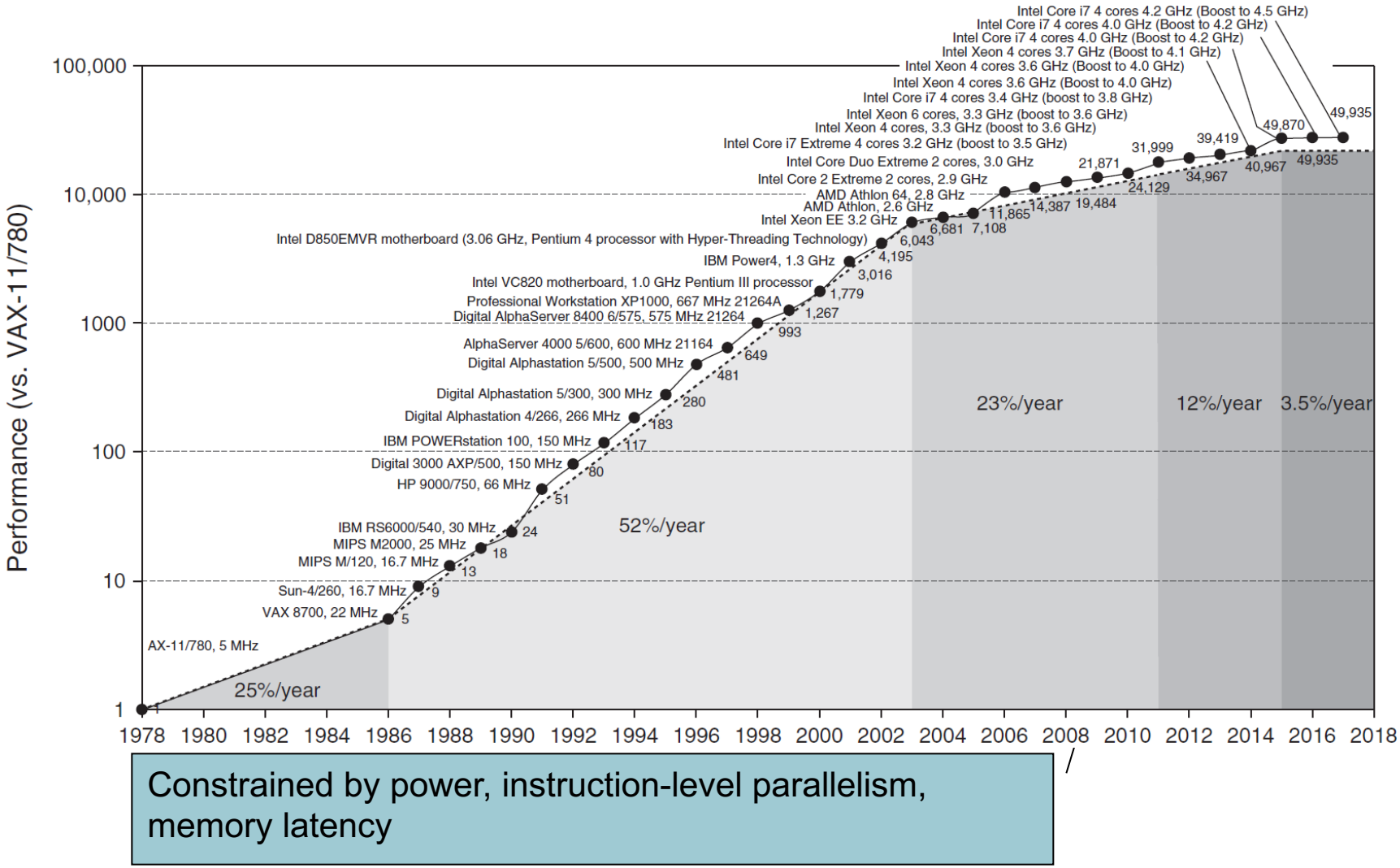
Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

A Sea Change is at Hand

- **The power challenge has forced a change in the design of microprocessors**
  - Since 2002 the rate of improvement in the response time of programs on desktop computers has slowed from a factor of 1.5 per year to less than a factor of 1.2 per year
- As of 2006 all desktop and server companies are shipping microprocessors with multiple processors – cores – per chip

Product	AMD Barcelona	Intel Nehalem	IBM Power 6	Sun Niagara 2
Cores per chip	4	4	2	8
Clock rate	2.5 GHz	~2.5 GHz?	4.7 GHz	1.4 GHz
Power	120 W	~100 W?	~100 W?	94 W

- *Plan of record: double the number of cores per chip per generation (about every two years)*

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- System Performace Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPECspeed 2017 Integer benchmarks on a 1.8 GHz Intel Xeon E5-2650L

Description	*Name*	*Instruction Count x 10^9*	*CPI*	*Clock cycle time (seconds x 10^–9)*	*Execution Time (seconds)*	*Reference Time (seconds)*	*SPECratio*
Perl interpreter	perlbench	2684	0.42	0.556	627	1774	2.83
GNU C compiler	gcc	2322	0.67	0.556	863	3976	4.61
Route planning	mcf	1786	1.22	0.556	1215	4721	3.89
Discrete Event simulation - computer network	omnetpp	1107	0.82	0.556	507	1630	3.21
XML to HTML conversion via XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Video compression	x264	4488	0.32	0.556	813	1763	2.17
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Artificial Intelligence: Monte Carlo tree search (Go)	leela	2236	0.79	0.556	987	1703	1.73
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
General data compression	xz	8533	1.32	0.556	6290	6182	0.98
Geometric mean							2.36

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

Example

- Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

- We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Fallacy: Computers at low utilization use little power

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - CPI varies between programs on a given CPU

MIPS example

- Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

  The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

Midterm Question

We are thinking about adding a new addressing mode to our instruction set. The addressing mode adds two integers and an offset to get the effective address in the proposed machine. We have to change our compiler to use this new feature. Therefore our new compiler is going to replace the code sequence:

```
add $s1, $s2, $s3
lw $s5, 0($s1)
```

or

```
add $s1, $s2, $s3
sw $s5, 0($s1)
```

with a **load** or a **store** using the new addressing mode.

Midterm Question (cont.)

Our measurements give us the following instruction distribution on this machine before we apply the new addressing mode.

Instruction Class	Frequency
Loads	24%
Stores	16%
ALU Ops	44%
Branches	16%

a) Assume that the new addressing mode can be used for **25%** of the loads and stores (in other words **25%** of memory access for data). What is the ratio of the instruction count on the proposed machine and the previous machine?

b) If the new addressing mode lengthens the clock cycle time 6% and cycle per instruction (i.e., CPI) remains the same, which machine is faster? How much faster?