Computer Structure 234267

Lecturers:

Lihu Rappoport

Adi Yoaz

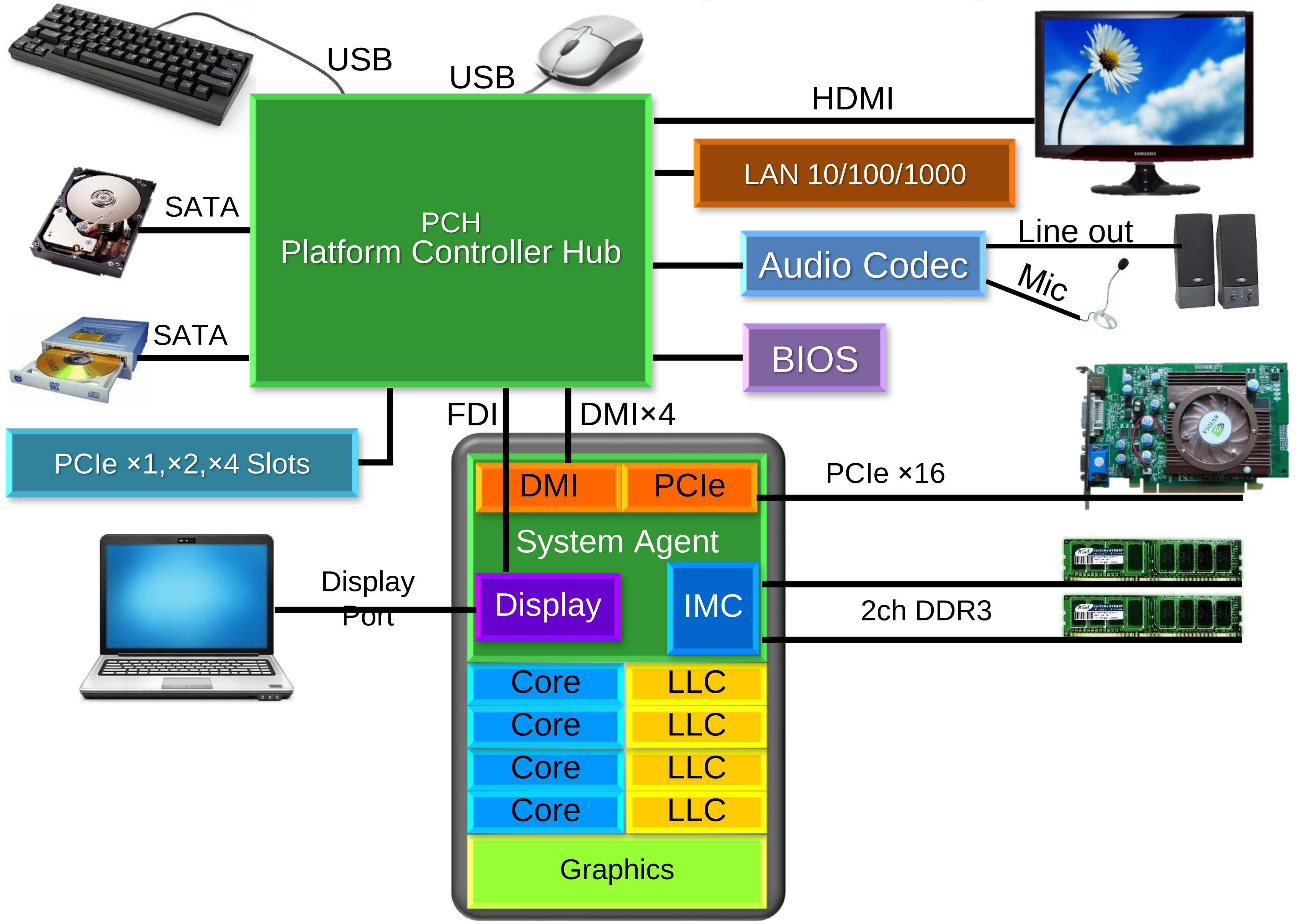
General Course Information

- **Grade**
  - 20% : 4 exercises (all mandatory) תקף – submission in pairs
  - 80% final exam
- **Come to the lectures and to the tutorials !**
  - The material for the exam includes all that is taught during the lectures and the tutorials – the foils do not contain everything
  - While in the classroom wear masks properly over the nose, and keep distance
- **Course web site**
  - http://webcourse.cs.technion.ac.il/236267/
  - Foils will be on the web several days before each lecture

Class Focus

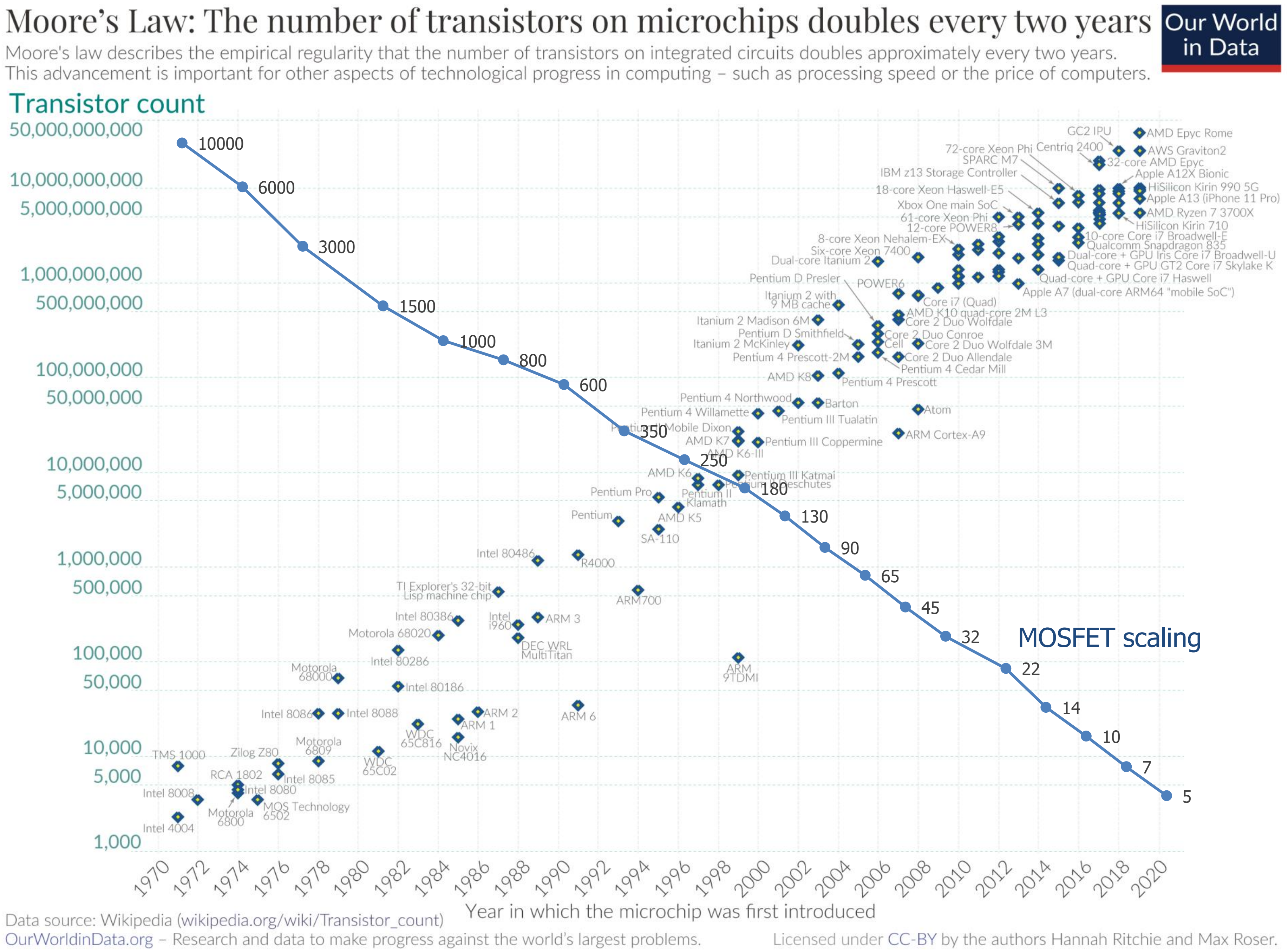
- **CPU**
  - Introduction: performance, instruction set (RISC vs. CISC)
  - Pipeline, hazards
  - Branch prediction
  - Out-of-order execution
- **Memory Hierarchy**
  - Cache and cache coherency
  - Main memory
  - Virtual Memory
- **More topics**
  - Multi-threading
  - System
  - Power considerations

Personal Computer System



Architecture & Microarchitecture

- **Architecture**
  The processor features seen by the "user"
  - Instruction set, addressing modes, data width, …
- **Micro-architecture**
  The internal implementation of a processor
  - Caches size and structure, number of execution units, …
- **Processors with different µArch can support the same architecture**
- **Compatibility**
  - A new processor can run existing software
  - The processor µArch is new, but it supports the Architecture of past generations, possibly adding to it



MOSFET scaling

10 μm	1971
6 μm	1974
3 μm	1977
1.5 μm	1981
1 μm	1984
800 nm	1987
600 nm	1990
350 nm	1993
250 nm	1996
180 nm	1999
130 nm	2001
90 nm	2003
65 nm	2005
45 nm	2007
32 nm	2009
22 nm	2012
14 nm	2014
10 nm	2016
7 nm	2018
5 nm	2020

Future

3 nm	~ 2022
2 nm	~ 2024

CPU Performance

- **CPUs work according to a clock signal**
  - Clock cycle is measured in nsec (10^{-9} of a second)
  - Clock frequency (= 1/clock cycle) measured in GHz (10^9 cyc/sec)

- **CPI – Cycles Per Instruction**
  - Average #cycles per Instruction (in a given program)

$$\text{CPI} = \frac{\text{Total number of cycles required to execute the program}}{\text{Total number of instructions executed in the program}}$$

  - IPC (= 1/CPI) : Instructions per cycles

Calculating the CPI of a Program

- ICi: #times instruction of type i is executed in the program
- IC: #instruction executed in the program: $IC = \sum_{i=1}^{n} IC_i$
- Fi: relative frequency of instruction of type i : $Fi = ICi/IC$

- CPI_i – #cycles to execute instruction of type i
  - e.g.: $CPI_{add} = 1$, $CPI_{mul} = 3$
- #cycles required to execute the entire program:

$$\#cyc = \sum_{i=1}^{n} CPI_i * IC_i = CPI * IC$$

- CPI:

$$CPI = \frac{\#cyc}{IC} = \frac{\sum_{i=1}^{n} CPI_i * IC_i}{IC} = \sum_{i=1}^{n} CPI_i * \frac{IC_i}{IC} = \sum_{i=1}^{n} CPI_i * F_i$$

CPU Time

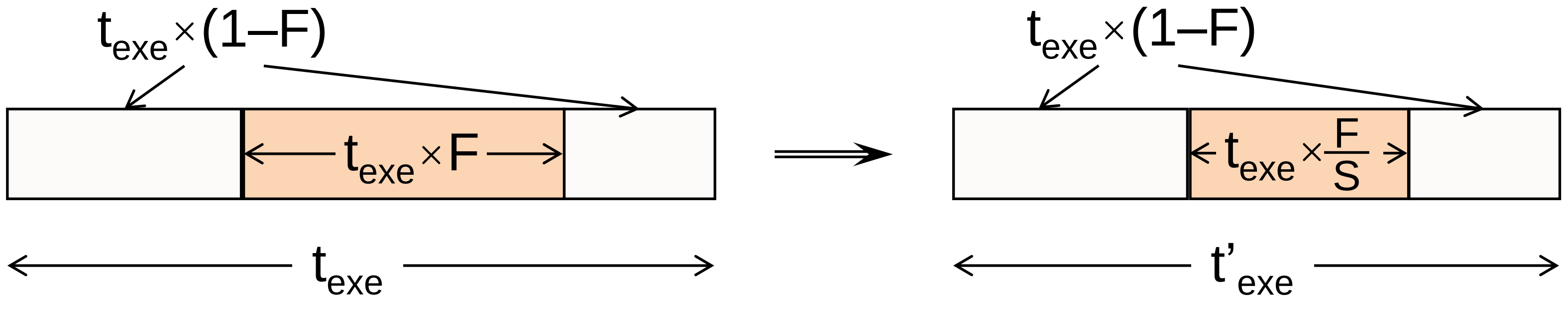
- **CPU Time - time required to execute a program**

$$\text{CPU Time} = \text{IC} \times \text{CPI} \times \text{clock cycle}$$

- **Our goal: minimize CPU Time**
  - Minimize clock cycle: more GHz (process, circuit, uArch)
  - Minimize CPI: uArch (e.g.: more execution units)
  - Minimize IC: architecture (e.g.: vector instructions)

Amdahl's Law

Suppose enhancement E accelerates a fraction *F* of the task by a factor *S*, and the remainder of the task is unaffected, then:



$$t'_{exe} = t_{exe} \times \left[(1-F) + \frac{F}{S} \right]$$

$$\text{Speedup}_{overall} = \frac{t_{exe} - t'_{exe}}{t_{exe}} = 1 - \frac{t'_{exe}}{t_{exe}} = 1 - \left[(1-F) + \frac{F}{S} \right] = F - \frac{F}{S}$$

Amdahl's Law: Example

- **Floating point instructions improved to run 2× faster**
- **10% of the executed instructions are FP instruction**

$$t'_{exe} = t_{exe} \times (0.9 + 0.1 / 2) = 0.95 \times t_{exe}$$

$$\text{Speedup}_{overall} = 1 - 0.95 = 0.05 = 5\%$$

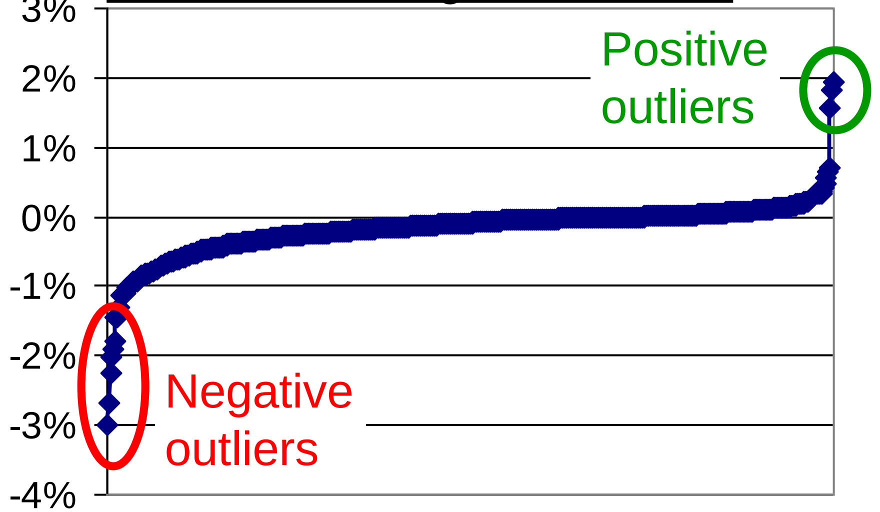
Corollary:

Make The Common Case Fast

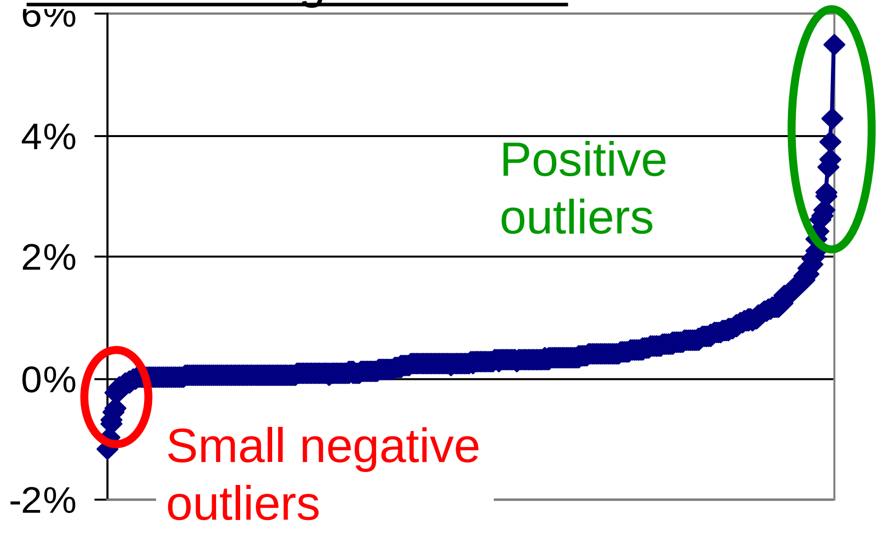
Evaluating Performance of future CPUs

- **Use a performance simulator to evaluate the performance of a new feature / algorithm**
  - Models the uarch to a great detail
  - Run 100's of representative applications
- **Produce the performance s-curve**
  - Sort the applications according to the IPC increase
  - Baseline (0) is the processor without the new feature

S-curve with negative results



S-curve of a good feature

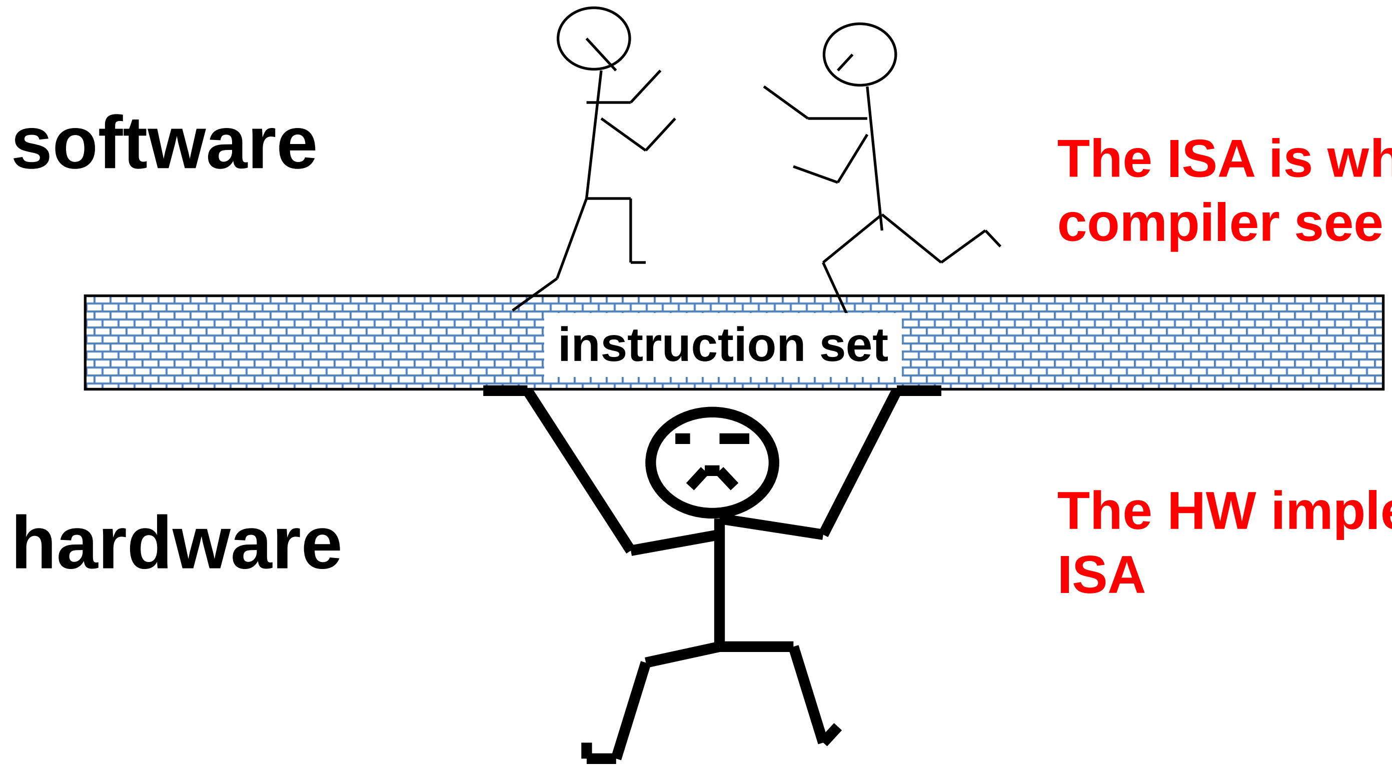


Using Benchmarks for Comparing Performance

- **Benchmark types**
  - Real applications, or representative parts of real apps
  - Synthetic benchmarks which represent typical workloads

- **Benchmarks are targeted at the specific system usages**
  - SPEC INT – integer applications: data compression, C complier, Perl interpreter, database system, chess-playing, Text-processing, …
  - SPEC FP – floating point applications: fluid dynamics, quantum chemistry, image ray-tracing, speech recognition, …
  - SYSMark – reflects usage patterns of business users
    - office productivity: Acrobat, Excel, PowerPoint, Word
    - media creation: Photoshop, Premiere, Sketchup
    - data/financial analysis: Excel, WinZip
  - TPC Benchmarks - transaction-processing throughput

Instruction Set Design



The ISA is what the user / compiler see

The HW implements the ISA

ISA Considerations

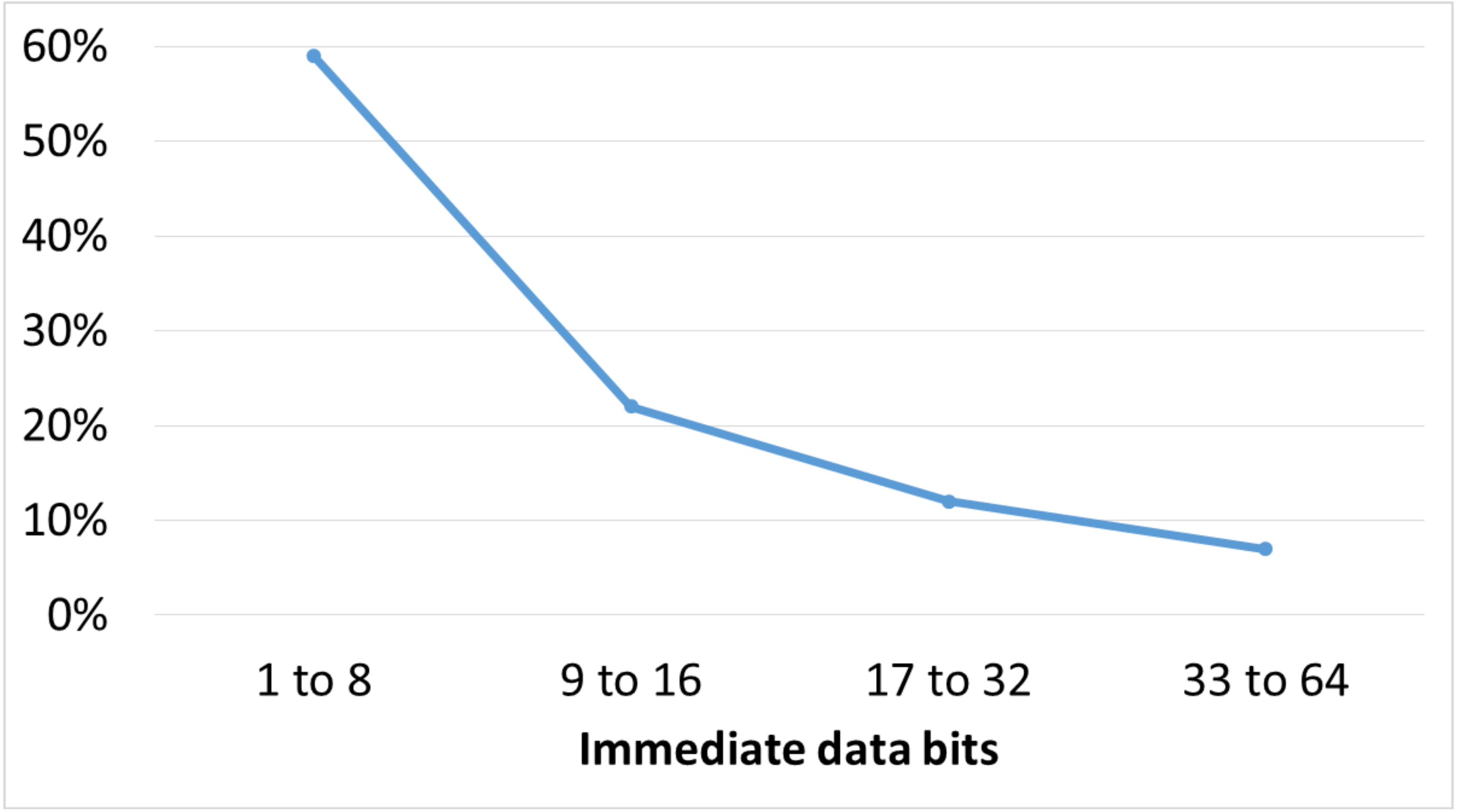
- **Reduce the IC to reduce execution time**
  - A single vector instruction performs the work of multiple scalar instructions

- **Simple instructions ⇒ simpler HW implementation**
  - Higher frequency, lower power, lower cost

- **Code size**
  - Long instructions take more time to fetch and require a larger memory
    - Important for large code footprint workloads

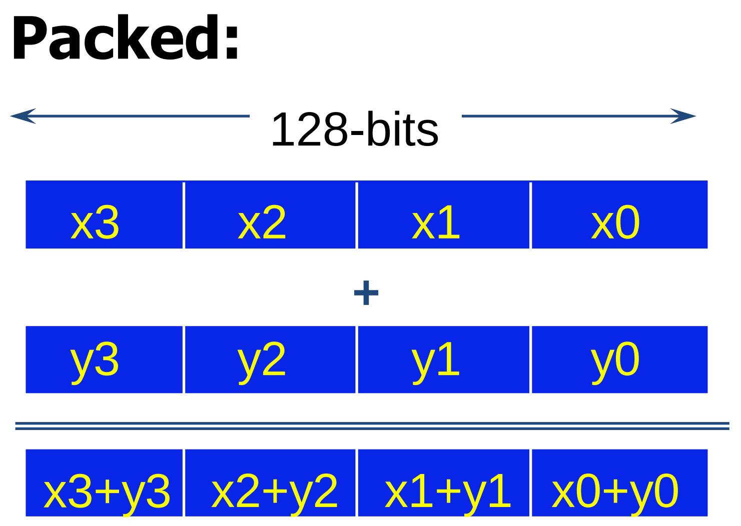
Architectural Consideration Example

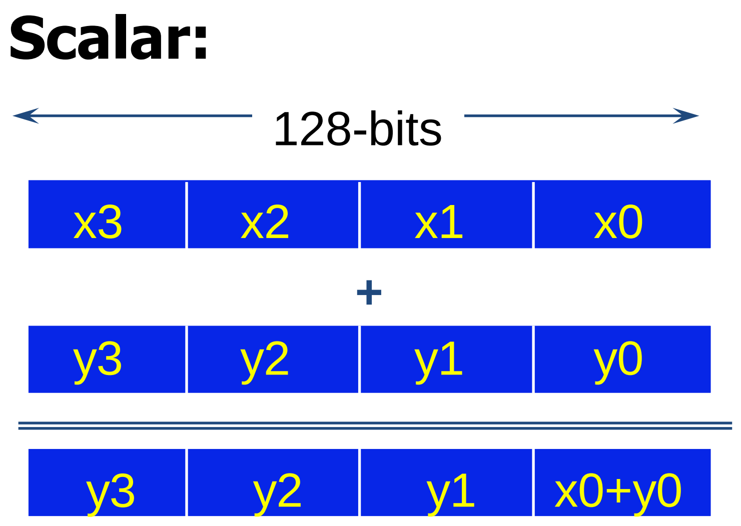
- **Most instructions use a short immediate value**
  - Having all instructions support the maximum size needed would increase code size



ISA Extensions for Reducing IC

- **Example**
  - 128-bit packed (vector) / scalar single precision FP (4×32)
  - 128-bit registers (XMM0 – XMM7)





  - A single “packed add” instruction replaces 4 “add” instructions
  - In order to fully gain the performance, need also new HW
    - execution unit that performs “packed add” in a single cycle

CISC Processors

- **CISC – Complex Instruction Set Computer** – a high-level machine language
  - Example: x86
- **Complex instructions and complex addressing modes**

  ⇒ complicates the processor and slows down the simple, common instructions

  ⇒ contradicts Make The Common Case Fast
- **Small number of registers** ⇒ requires many memory accesses
- **Non-orthogonal registers:** some operations supported only on specific registers

  ⇒ Not compiler friendly
- **ALU operations directly on memory**

```
    add   eax, 7680[ecx]  ;eax ← MEM[7680+ecx]+ eax
```

- **Variable length instructions**
  - Common instructions get short codes ⇒ save code length
  - Difficult to decode few instructions in parallel: only after an instruction is decoded, its length (and when the next instruction stats) is known
  - An instruction may cross a cache line or a page

Top 10 Instructions

Rank	instruction	% of total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	**96%**

Simple instructions dominate instruction frequency

RISC Processors

- **RISC – Reduced Instruction Set Computer**
  - Simple instructions enable fast hardware: make the common case fast
  - Example: ARM
- **Characteristics**
  - A small instruction set, with few instruction formats
  - Simple instructions that execute simple tasks, most of them single-cycle (with pipeline)
  - A few addressing methods, with ALU operations on registers only
    - Memory is accessed using Load and Store instructions only
  - Many orthogonal registers – all instructions available with all registers
  - Three address machine: Add dst, src1, src2
  - Fixed length instructions
  - Complier friendly – easier to utilize the ISA: orthogonal, simple

RISC Processors (Cont.)

- **Simple architecture ⇒ Simple micro-architecture**
  - Simple, small and fast control logic
  - Simpler to design and validate
  - Leave space for large on die caches
  - Shorter time-to-market
- **Existing RISC processor are not "pure" RISC**
  - E.g., support division which takes many cycles
- **Modern CISC processors use RISC µ-arch ideas**
  - Internally translate the CISC instructions into RISC-like operations
    - the inside core looks much like that of a RISC processor