4190.308: Computer Architecture

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

Spring 2019

Course Information

- Schedule
  - 9:30 – 10:45 (Tuesday & Thursday)
  - Lecture room: Engineering Bldg. #301-203
  - 3 credits
  - Official language: English

- TA: Jae-Hoon Shim (x7296)

- SNU eTL system for exam/project scores
- http://csl.snu.ac.kr for announcements and lecture slides
- http://sys.snu.ac.kr for project submissions and automatic grading

About Me

- Jin-Soo Kim (김진수)
  - Professor @ CSE Dept.
  - Systems Software & Architecture Laboratory
  - Operating systems, storage systems, parallel and distributed computing, embedded systems, …

- E-mail: jinsoo.kim@snu.ac.kr
- Tel: 02-880-7302
- Office: Engineering Bldg. #301-520 (office hours: Tuesday & Thursday)
- The best way to contact me is by email

Prerequisites

- Prerequisites
  - Programming Practice (4190.103A) – C programming
  - Logic Design (M1522.000700) – Must!
  - Data Structure (M1522.000900) – Recommended
- You should be familiar with the followings:
  - Shells and basic Linux commands
  - C programming & debugging skills (on Linux)
  - Basic knowledge on digital circuits and systems
- Accessible x86-64/Linux (Ubuntu 18.04.2 LTS or similar) machine

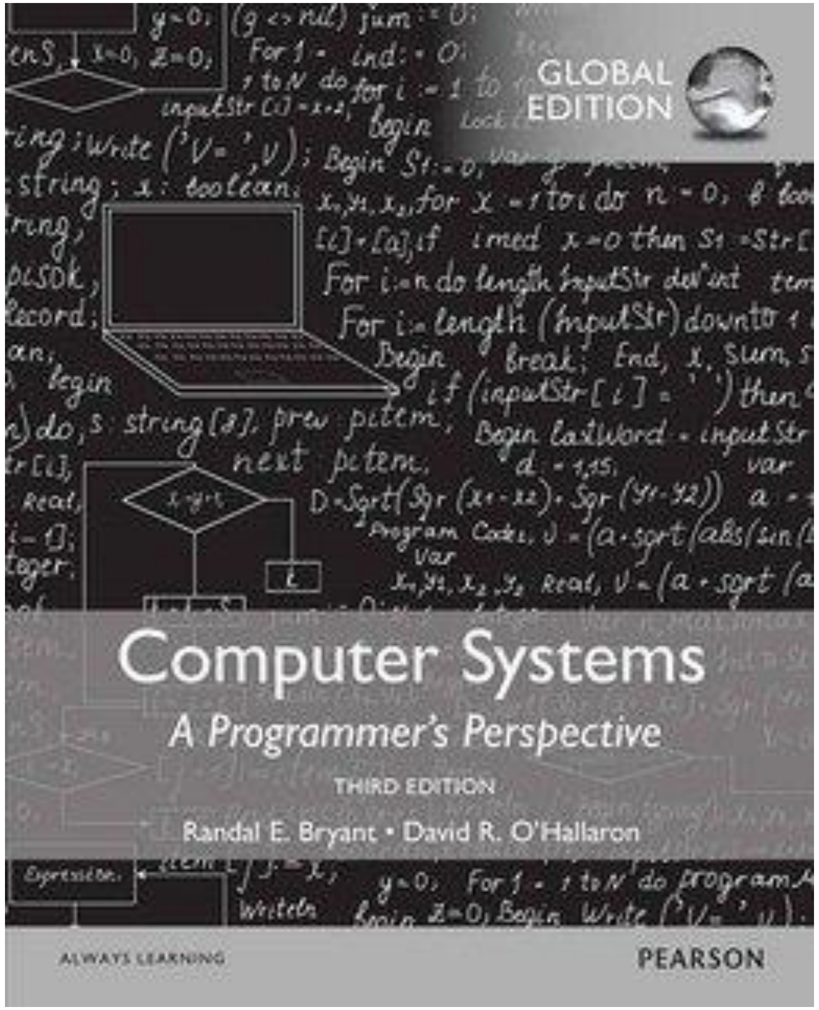
Policies for Non-major Students

- Your course registration form (“초안지“) will be accepted only if …
  - You have an experience on C programming and debugging on Linux (gcc/gdb) and
  - You have already taken the “Logic Design” course

- Other introductory CSE courses for non-major students:
  - M1522.000600 Computer Programming
  - M1522.000700 Logic Design
  - M1522.000900 Data structures
  - 4190.101 Discrete Mathematics
  - 4190.103 Programming Practice

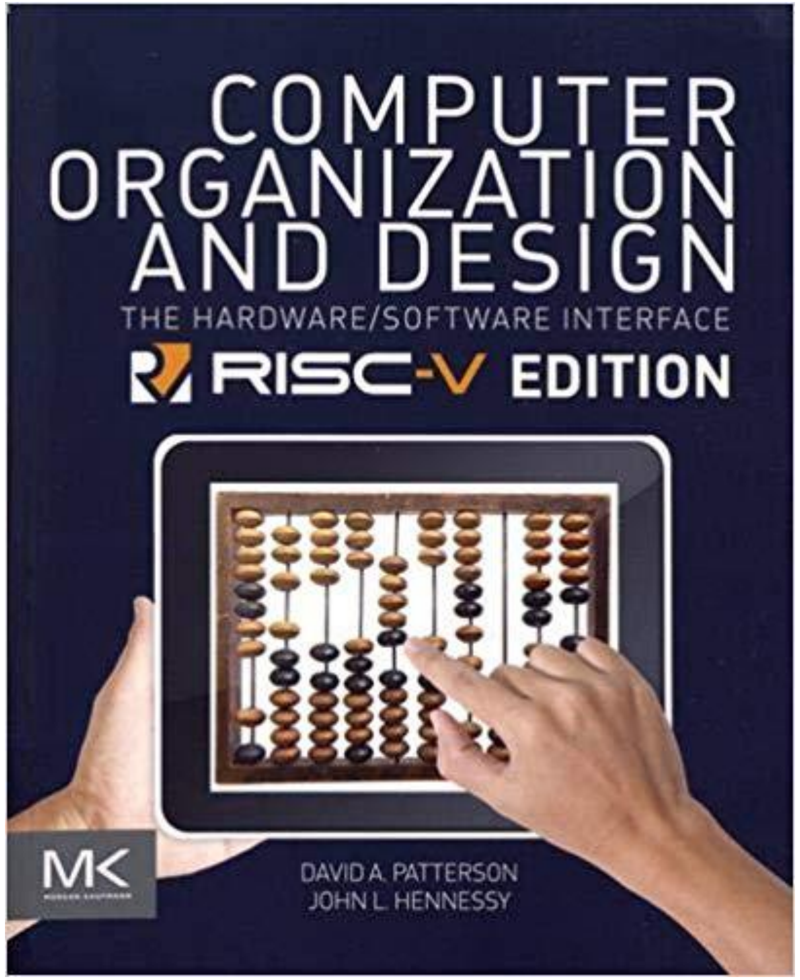
Textbook

- Computer Systems:
  A Programmer's Perspective
  - Randal E. Bryant and David R. O'Hallaron
  - Third Edition
  - Pearson Education Limited, 2016
  - http://csapp.cs.cmu.edu



References

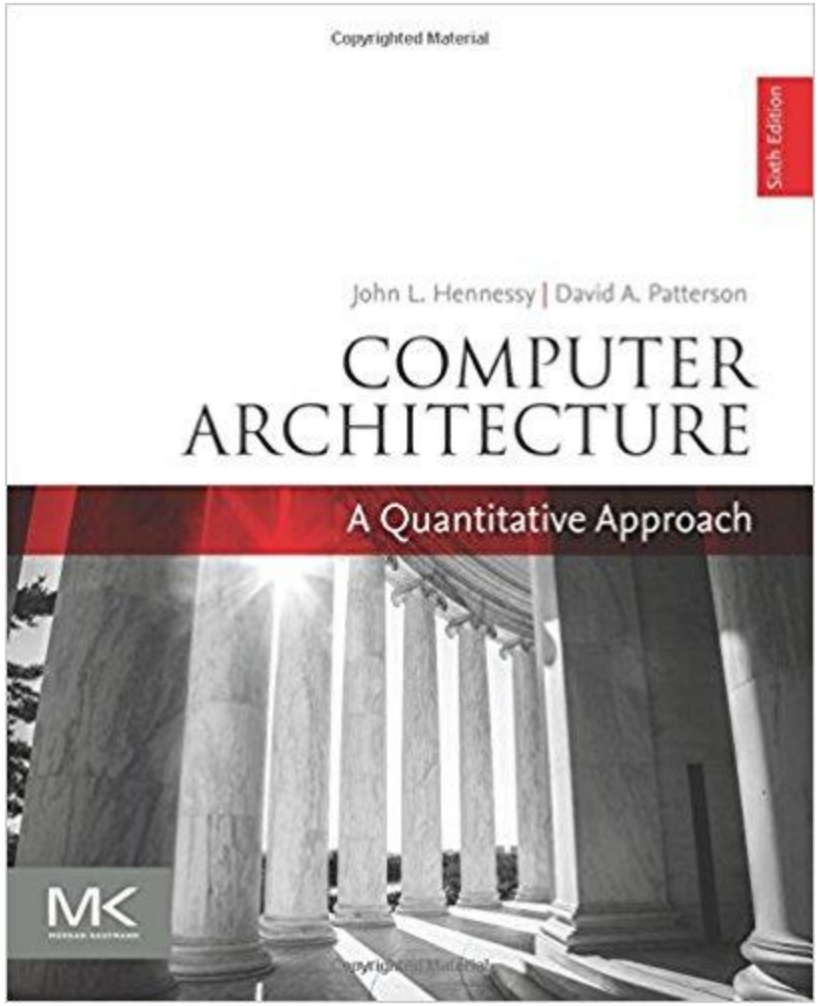
- Computer Organization and Design: The Hardware/Software Interface (RISC-V Edition)
  - David A. Patterson and John L. Hennessy (Turing Award Recipients in 2017)
  - First Edition
  - Morgan Kaufmann, 2017
  - http://booksite.elsevier.com/9780128122754/



References

- Computer Architecture:
  A Quantitative Approach
  - John L. Hennessy and David A. Patterson
  - Sixth Edition
  - Morgan Kaufmann, 2017

  - http://booksite.elsevier.com/9780128119051

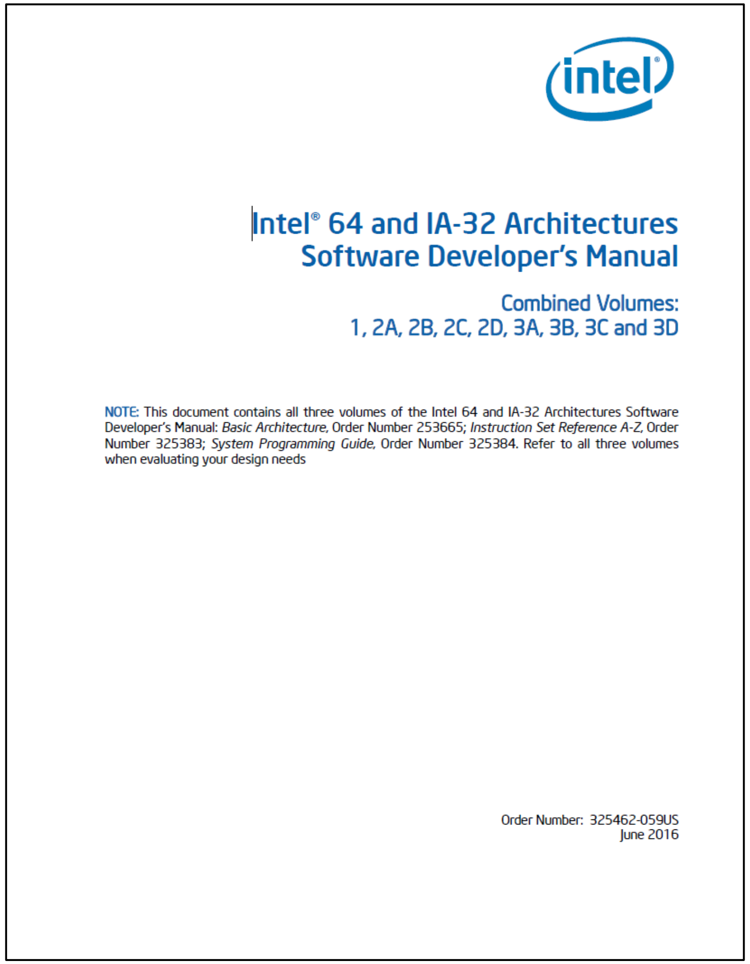


References

- Intel 64 and IA-32 Architectures Software Developer's Manual

  - Volume 1: Basic Architecture
  - Volume 2: Instruction Set Reference
  - Volume 3: System Programming Guide

  - https://software.intel.com/en-us/articles/intel-sdm



Intel® 64 and IA-32 Architectures Software Developer's Manual

Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C and 3D

NOTE: This document contains all three volumes of the Intel 64 and IA-32 Architectures Software Developer's Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384. Refer to all three volumes when evaluating your design needs

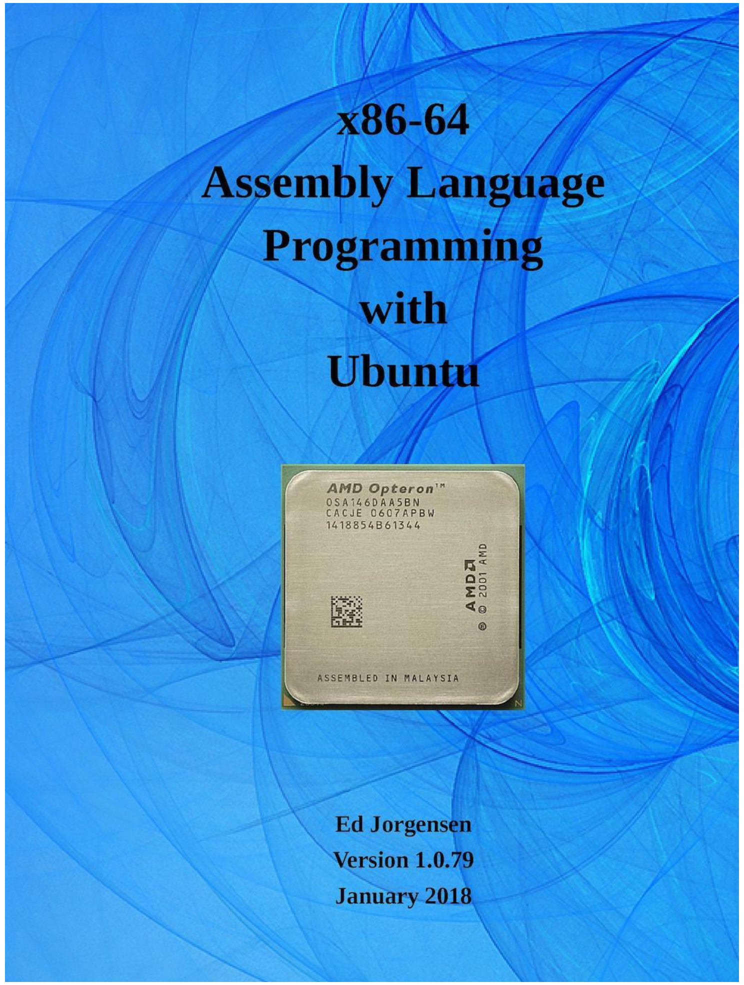
Order Number: 325462-059US
June 2016

References

- x86-64 Assembly Language Programming with Ubuntu

  - Ed Jorgensen
  - Version 1.1.13
  - September 2018

  - http://www.egr.unlv.edu/~ed/x86



Topics

- Introduction to Computer Architecture
- Integers and Floating Points
- x86-64 Instruction Set Architecture
- Sequential Architecture
- Pipelined Architecture
- Cache
- Virtual memory
- I/O and Storage
- Parallel Computer Architecture

Projects (subject to change)

- C programming
- x86-64 assembly programming
- y86-64 assembly programming
  - y86-64 is a simplified instruction set used in this course based on Intel x86-64 architecture
- Optimizing y86-64 assembly programs for pipelined processor

Grading Policy (subject to change)

- Exams: 60%
  - Midterm: 25%
  - Final: 35%
- Projects: 40%

- University policy requires students to attend at least 2/3 of the scheduled classes. Otherwise, you'll fail this course.
- Also, if you miss one of the exams, you'll fail this course.

Cheating Policy

- What is cheating?
  - Copying another student's solution (or one from the Internet) and submitting it as your own
  - Allowing another student to copy your solution
- What is NOT cheating?
  - Helping others use systems or tools
  - Helping others with high-level design issues
  - Helping others debug their code
- Penalty for cheating
  - Severe penalty on the grade (F) and report to dept. chair
  - Ask helps to your TA or instructor if you experience any difficulty!

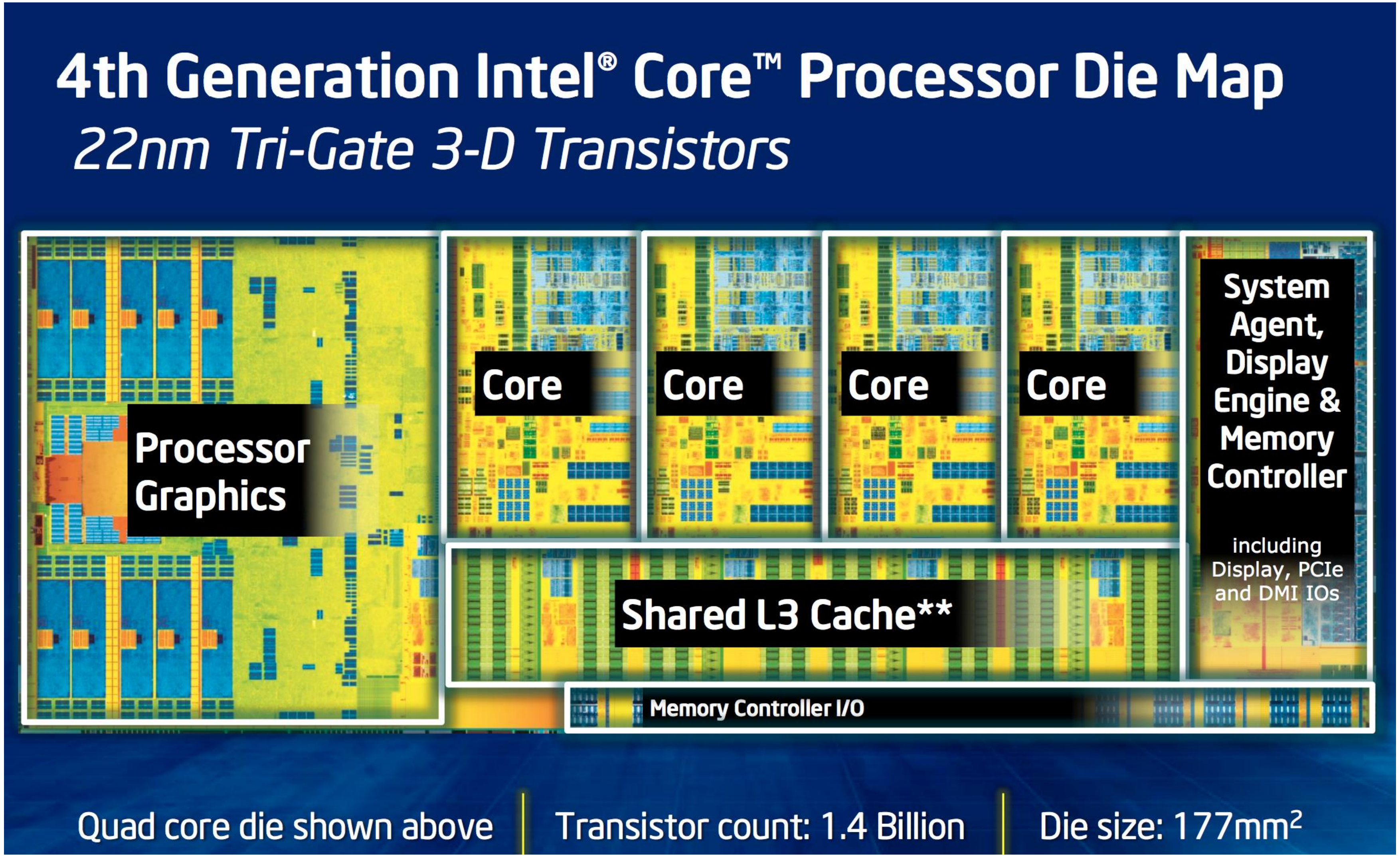
World's Tallest Lego Tower

- Omer Tower @ Tel Aviv, Israel
  - In memory of Omer Sayag, an 8-year-old boy who was a Lego enthusiast and died of cancer in 2014
  - Completed in Dec. 2017

- 118ft (~ 36m)
- > 500,000 Lego bricks

Source: http:// www.dailymail.co.uk/news/article-5215235/Tel-Aviv-toy-towers-world-record.html

Intel Core i7-4770K (Haswell, 2013)

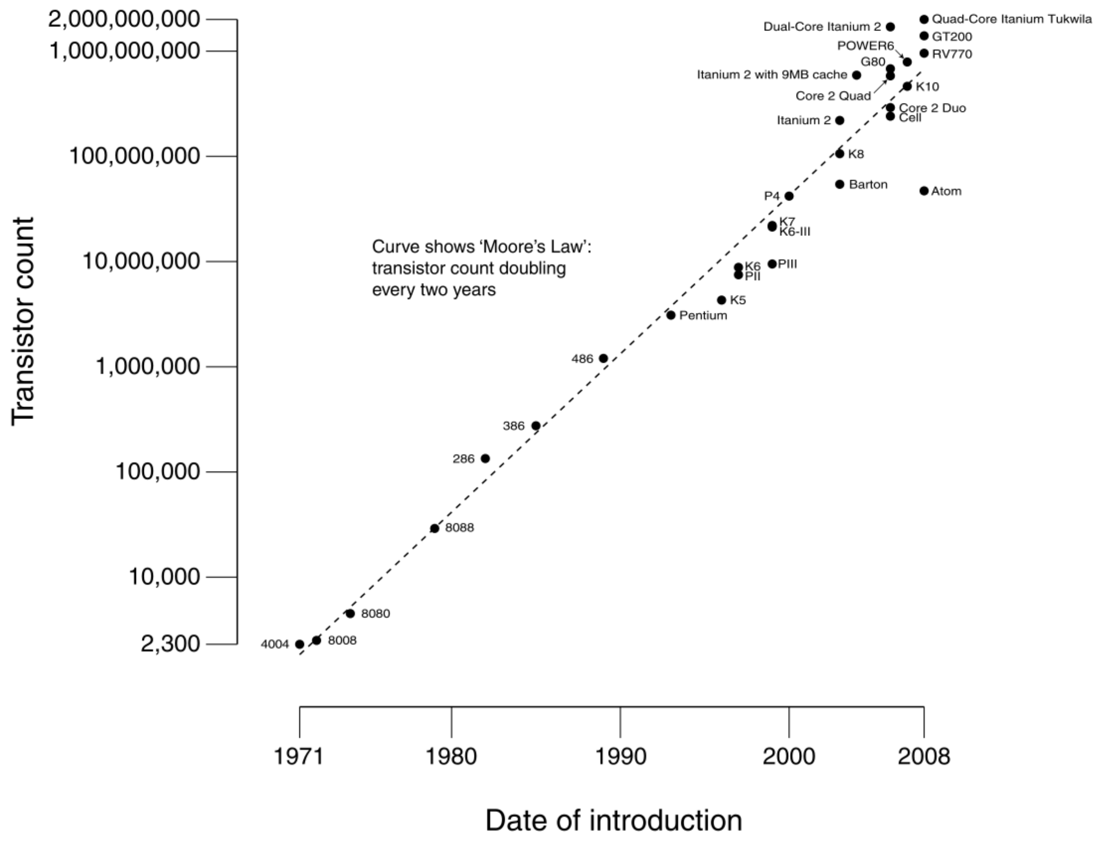


Source: https://www.anandtech.com/show/7003/the-haswell-review-intel-core-i74770k-i54560k-tested

Moore's Law

- By Gordon Moore @ Intel (1965)

CPU Transistor Counts 1971-2008 & Moore's Law



"The number of transistors incorporated in a chip will approximately double every 24 months."

Gordon Moore, Intel Co-founder

What Happened:

104 cabinets
(76 computes,
8 switches,
20 disks)

9298 cores

$150m^2$

1997

ASCI Red at Sandia

1.3 TF/s, 850 KW

2017

Cavium ThunderX2

~ 1.1 TF/s, ~ 0.2 KW

3.5 orders of magnitude

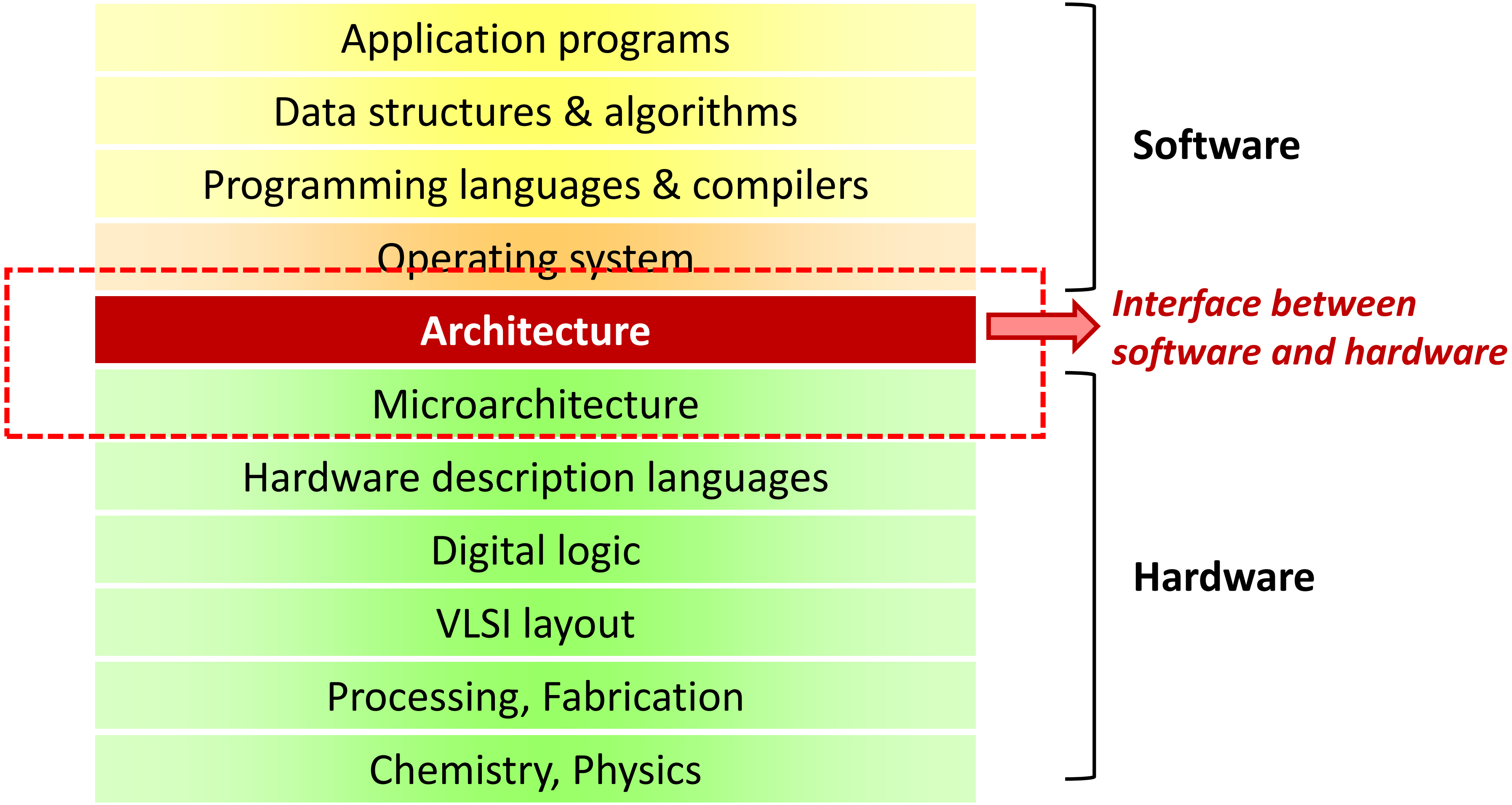
Source: David Keyes, "Algorithmic Adaptations to Extreme Scale Computing," ATPESC, 2018.

World's Tallest Lego Tower (Revisited)

Source: http:// www.dailymail.co.uk/news/article-5215235/Tel-Aviv-toy-towers-world-record.html
http://opanoticias.com/noticias/construyen-en-israel-la-torre-de-lego-mas-grande-del-mundo-36-metros-de-altura/
https://www.mirror.co.uk/news/uk-news/worlds-tallest-lego-tower-built-11763183

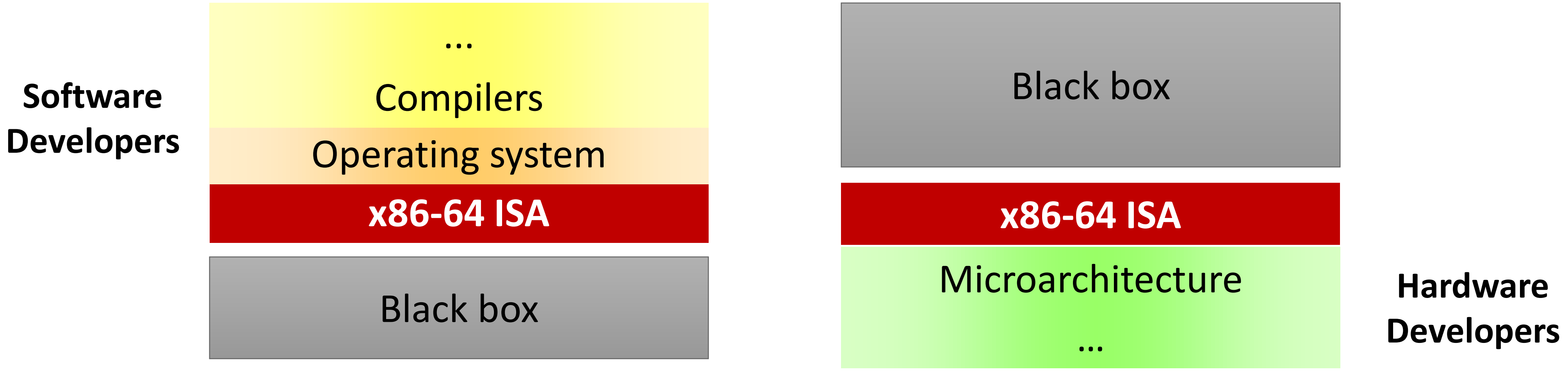
Taming Complexity

- Levels of abstractions



Instruction Set Architecture (ISA)

- The hardware/software interface
  - Hardware abstraction visible to software (OS, compilers, …)
  - Instructions and their encodings, registers, data types, addressing modes, etc.
  - Written documents about how the CPU behaves
  - e.g. All 64-bit Intel CPUs follow the same x86-64 (or Intel 64) ISA



Machine Code Example

- C code: add two signed integers
- Assembly code
  - Add two 8-byte integers
    - "quad" words in x86-64 parlance
    - Same instruction whether signed or unsigned
  - Operands
    - x: Register `%rdi`
    - y: Register `%rsi`
    - t: Register `%rax`
- Machine code
  - 4-byte instruction
  - Stored at memory address `0x4004d6`

```
long t = x + y;
```

```
leaq (%rdi,%rsi),%rax
```

```
0x4004d6: 48 8d 04 37
```

Abstraction is Good, But …

- Abstraction helps us deal with complexity
  - Hide lower-level details

- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations

- This is why you should take this course seriously even if you don't want to be a computer architect!

Example #1: Int's ≠ Integers, Float's ≠ Reals

- Is $x^2 \geq 0$?
  - Float's: ??
  - Int's: ??

```
int x = 50000;
printf (“%s\n”, (x*x >= 0)? “Yes” : “No”);
```

- Is $(x + y) + z == x + (y + z)$?
  - Unsigned & Signed Int's: ??
  - Float's: ??

```
float x = 1e20, y = -1e20, z = 3.14;
printf (“%s\n”, (x+y)+z==x+(y+z)? “Yes” : “No”);
```

Example #2: Memory Matters

- Memory referencing bug example

```
/* Echo Line */
void echo()
{
    // Way too small!
    char buf[4];
    gets(buf);
    puts(buf);
}

int main()
{
    printf("Type: ");
    echo();
    return 0;
}
```

```
$ ./bufdemo
Type:012
012

$ ./bufdemo
Type: 01234567890123456789012
01234567890123456789012

$ ./bufdemo
Type: 012345678901234567890123
Segmentation fault (core dumped)
```

Example #3: Constant Factors Matter

- There's more to performance than asymptotic complexity
- Array copy example

```
void copyij (int src[2048][2048],
             int dst[2048][2048])
{
  int i, j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji (int src[2048][2048],
             int dst[2048][2048])
{
  int i, j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3 ms (copyij) **81.8 ms** (copyji)

`copyji()` is 20x slower on 2.0GHz Intel Core i7 Haswell. Why?

Example #4: I/O Matters

- Computers do more than execute programs
- They need to get data in and out
  - I/O system critical to program reliability and performance
- Many system-level issues arise in presence of I/O
  - Concurrent operations by autonomous processes
  - Coping with unreliable media
  - Cross platform compatibility
  - Complex performance issues

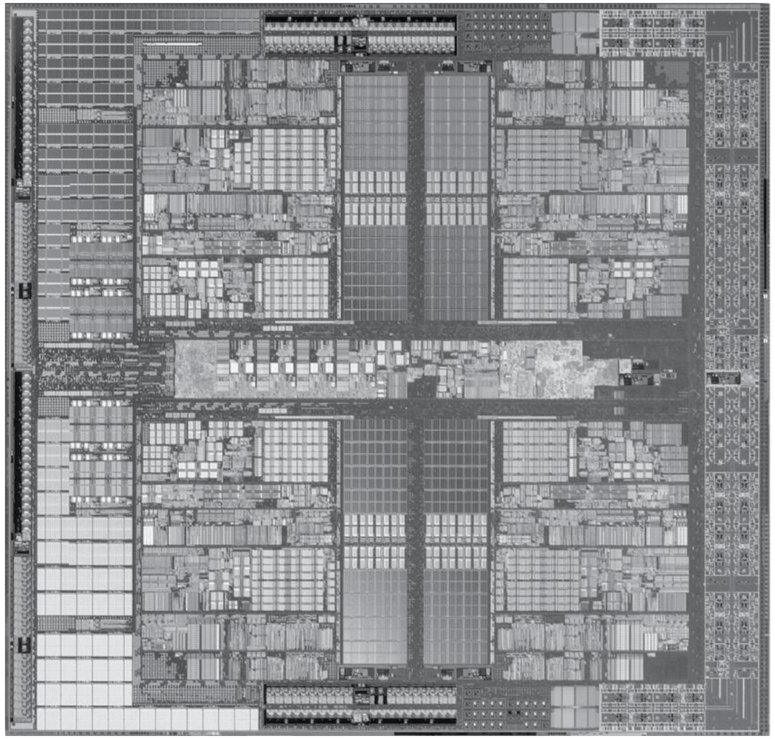
Example #5: You'll Need Assembly

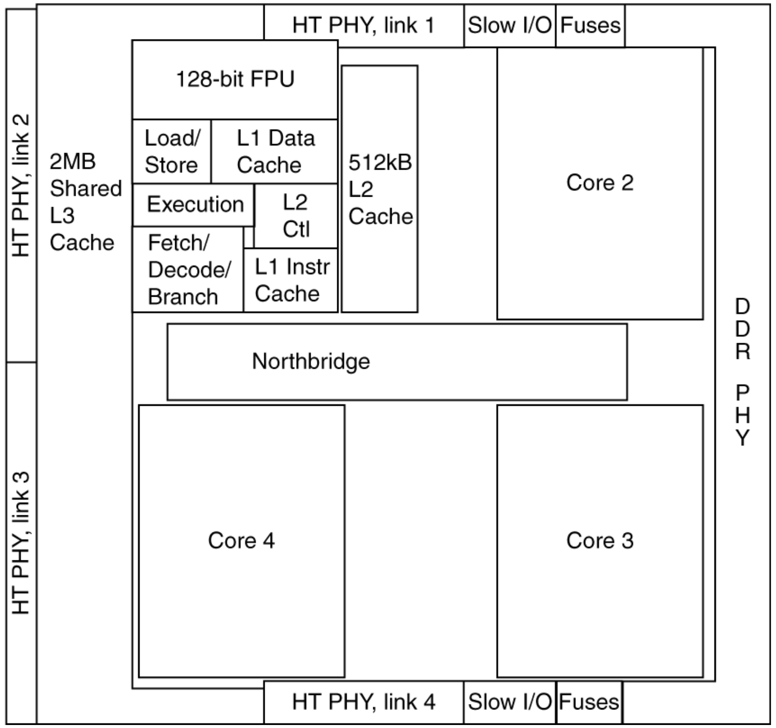
- Chances are, you'll never write programs in assembly
- But: Understanding assembly is key to machine-level execution model

- Behavior of programs in presence of bugs
  - High-level language models break down
- Tuning program performance
  - Understand optimizations done / not done by the compiler
  - Understanding sources of program inefficiency
- Implementing systems software (e.g. Compiler, OS, Boot loader, …)
- Creating / fighting malware

What we learn in this course: Overview

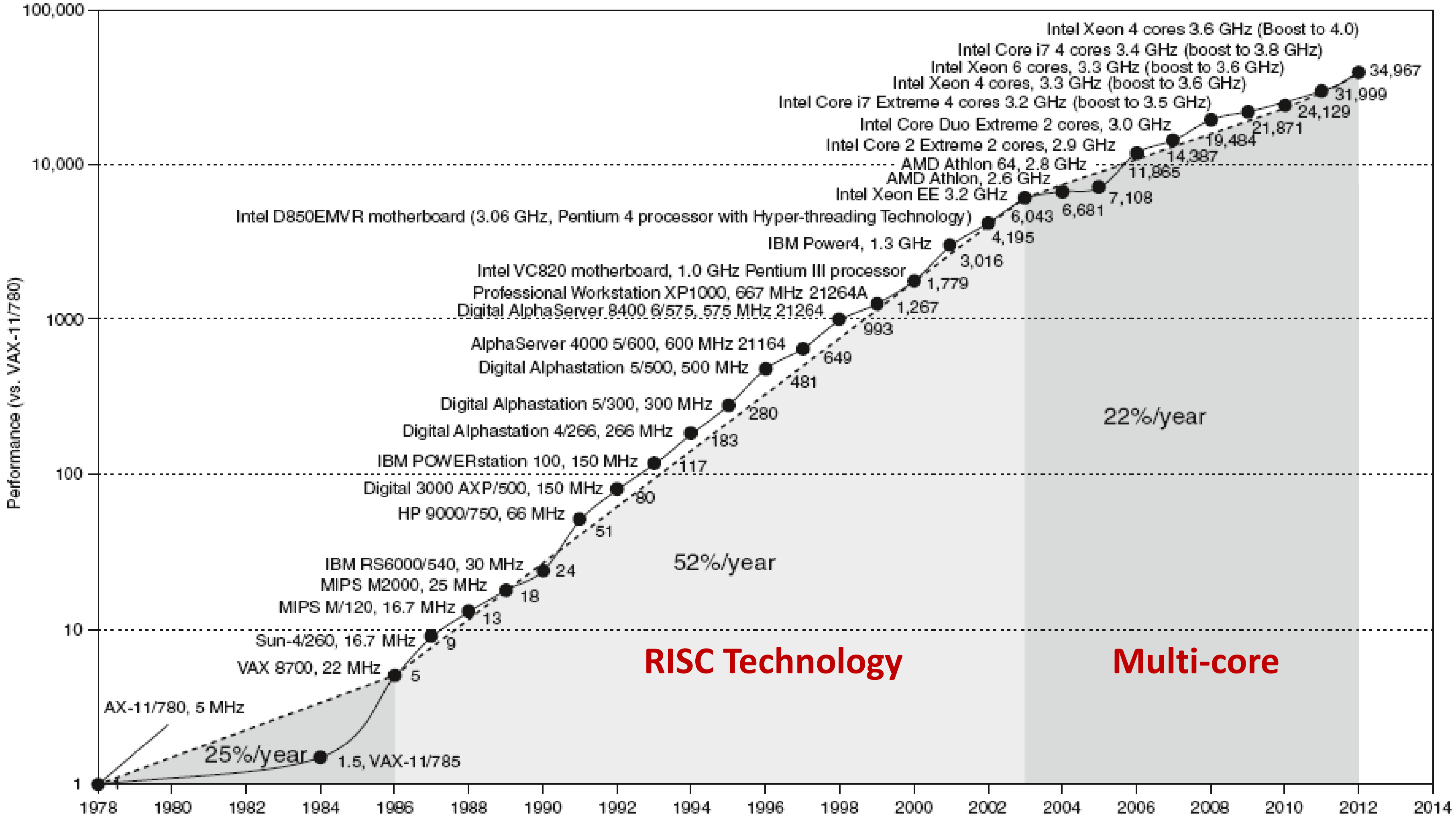
- You will understand what each block does by the end of this term!
- You will also learn how to program the CPU and write efficient code





AMD Barcelona: 4 processor cores

Uniprocessor Performance



End of Moore's law?

What we learn in this course: Specifics

- How data are represented?
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface – Instruction Set Architecture (ISA)
- What determines program performance
- How hardware designers / software developers improve performance
- What is parallel processing

Why Take This Course?

- To graduate!
- To design the next great instruction set? Well…
  - ISA has largely converged, especially in desktop / server / laptop / mobile space
  - Dictated by powerful market forces (Intel/ARM)
- To get a job in Intel, NVIDIA, ARM, Apple, Qualcomm, Google, …
  - Tremendous organizational innovations relative to established ISA abstractions
- Design, analysis, and implementation concepts that you'll learn are vital to all aspects of computer science and engineering
- This course will equip you with an intellectual toolbox for dealing with a host of systems design challenges
- And finally, just for fun!

Partially borrowed from David Culler's CS252 lecture slides

Summary

- Modern Computer Architecture is about managing and optimizing across several levels of abstraction w.r.t. dramatically changing technology and application load

- This course focuses on
  - x86-64 Instruction Set Architecture (ISA) – what interface is supported in Intel CPUs?
  - An implementation based on Pipelining (Microarchitecture) – how to make it faster?

- Understanding Computer Architecture is vital to other "systems" courses:
  - Operating systems, Compilers, Programming languages, Embedded systems, Storage systems, Computer networks, Parallel processing, Distributed systems, etc.