4190.308: Computer Architecture

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software & Architecture Lab.

Seoul National University

Fall 2020

Course Information

- Schedule
  - 11:00 – 12:15 (Monday & Wednesday)
  - Lecture room: ~~Engineering Bldg. #302-208~~ (Online lecture using Zoom)
  - 3 credits
  - Official language: English

- TA: Injae Kang (abcinje@snu), Sunmin Jeong (sunnyday0208@snu)
- 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: Monday & Wednesday)
- The best way to contact me is by email

Myths About This Course

- It's an introductory course
  - Introduction to Computers?
  - About 20% of students have dropped every semester
- It's all about hardware
  - It's about how to separate work between software and hardware, and about how to design the interface between them
- It's not relevant for software engineers
  - Writing good software requires understanding details of underlying implementation
- Who needs to know the assembly language these days?
  - Well, you'll see

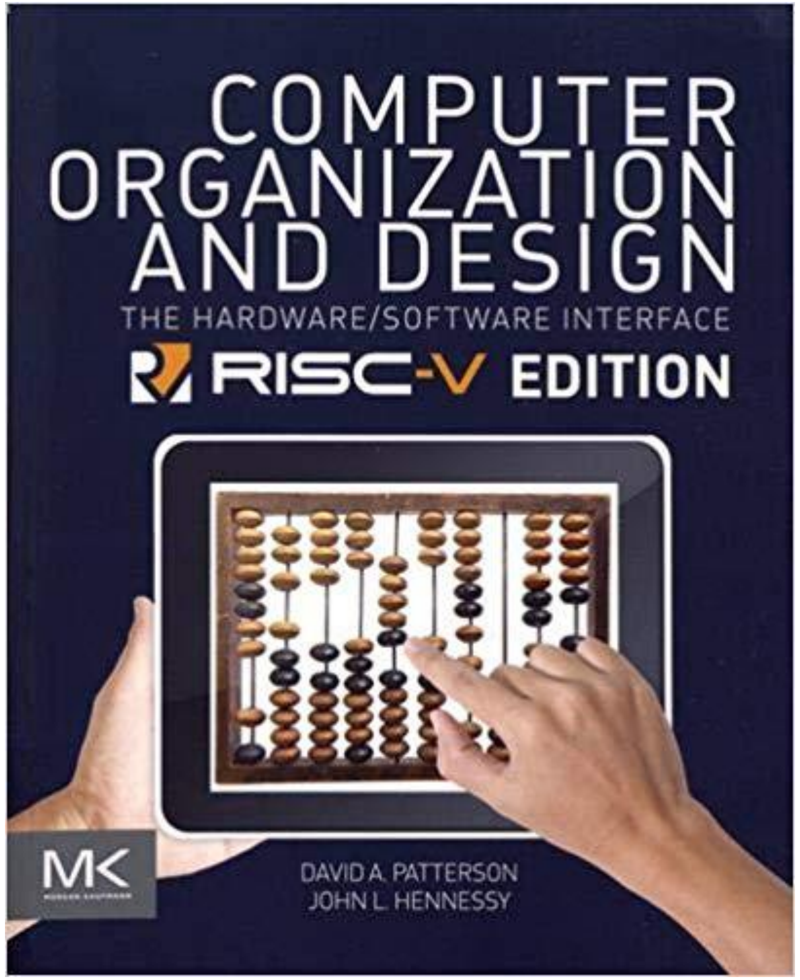
Prerequisites

- 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 (and Python!) programming skills
  - Basic knowledge on digital circuits and systems
- Accessible Linux (Ubuntu 18.04.3 LTS or similar) or MacOS machine

Textbook

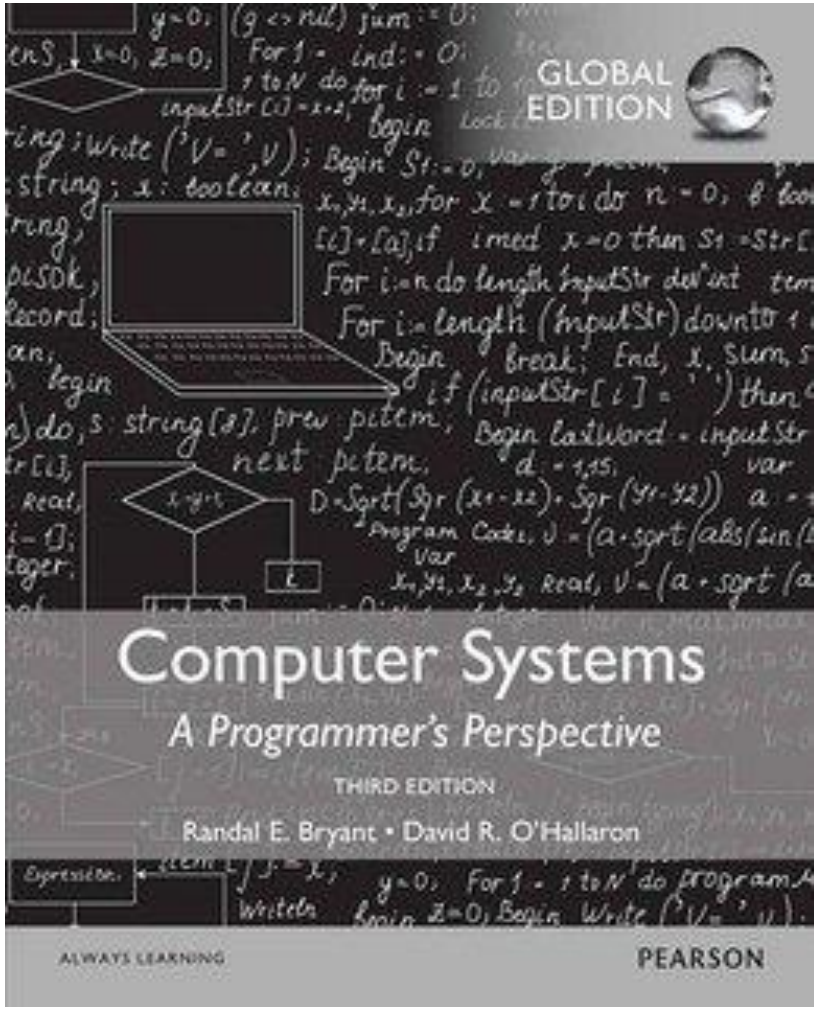
- Computer Organization and Design: The Hardware/Software Interface (RISC-V Edition)
  - David A. Patterson and John L. Hennessy (Turing Award Recipients in 2017)
  - First Edition
  - Morgan Kaufmann, 2017
  - http://booksite.elsevier.com/9780128122754/

  - Note: There are also MIPS and ARM editions



Previous Textbook

- Computer Systems:
  A Programmer's Perspective
  - Randal E. Bryant and David R. O'Hallaron
  - Third Edition
  - Pearson Education Limited, 2016
  - Based on x86-64
  - http://csapp.cs.cmu.edu

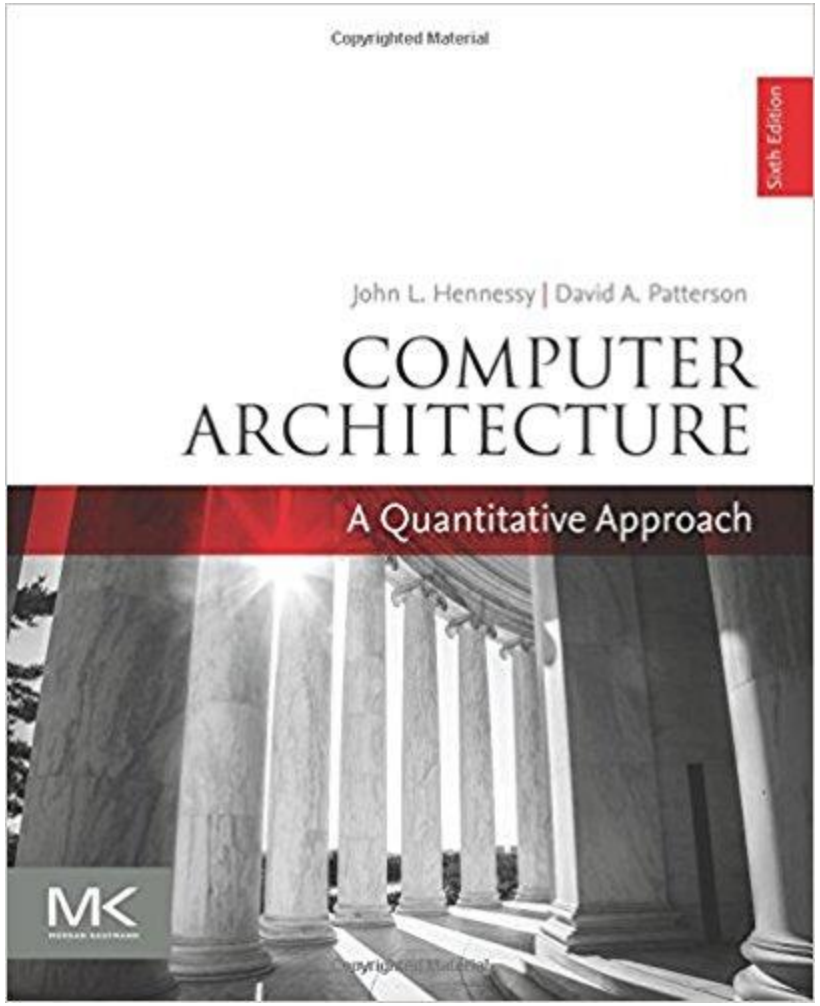


Reference

- Computer Architecture:
  A Quantitative Approach

  - John L. Hennessy and David A. Patterson
  - Sixth Edition
  - Morgan Kaufmann, 2017

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



Topics

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

Project Topics (subject to change)

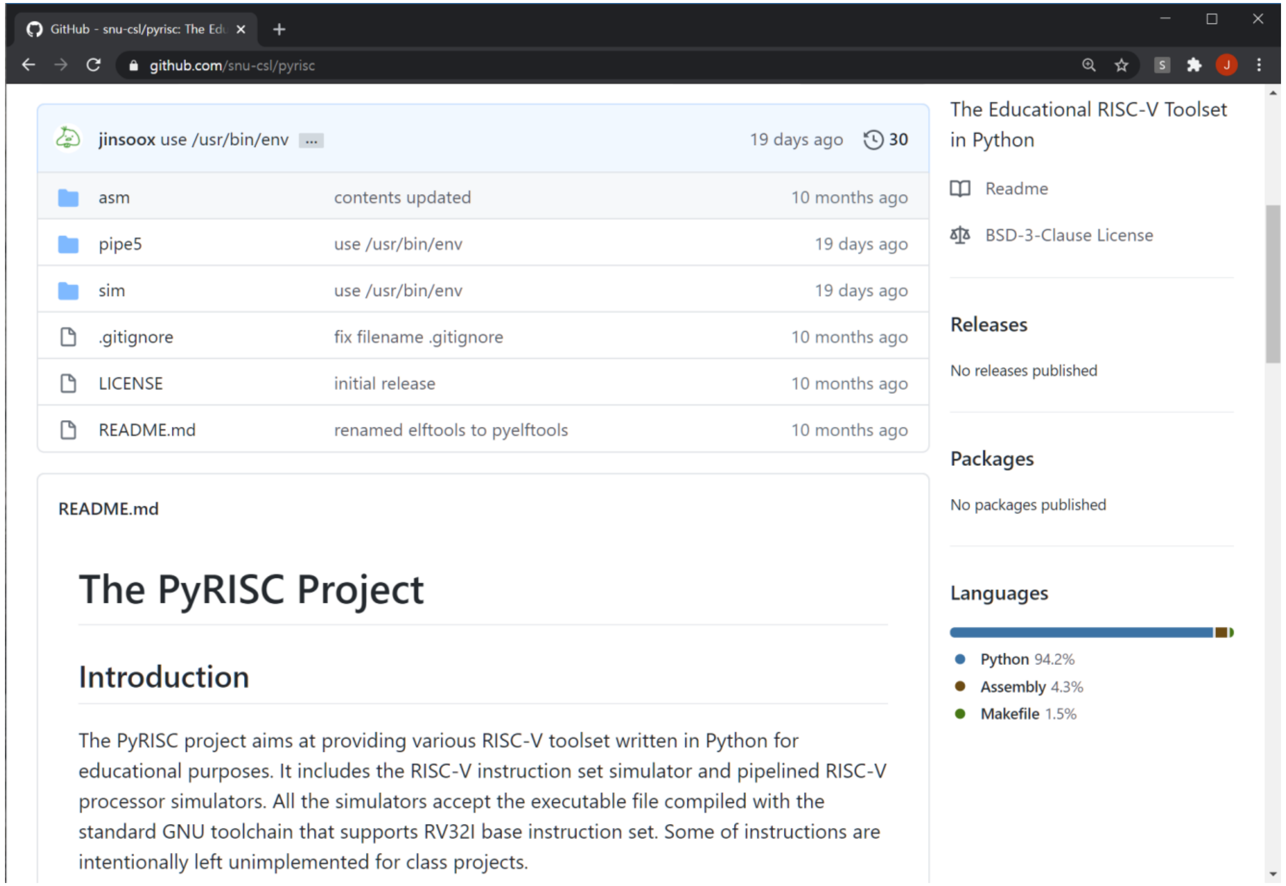
- C programming
- RISC-V assembly programming
- Designing pipelined processor
- Optimizing RISC-V assembly programs for pipelined processor
- Cache simulation

Why Python?

- We will use pyrisc, a RISC-V simulator written in Python

- You are required to modify the simulator

- Available at https://github.com/snu-csl/pyrisc



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.
- We are using the electronic attendance system via eTL.
- Also, if you miss one of the exams, you'll fail this course.

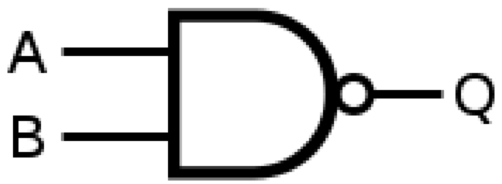
Cheating Policy

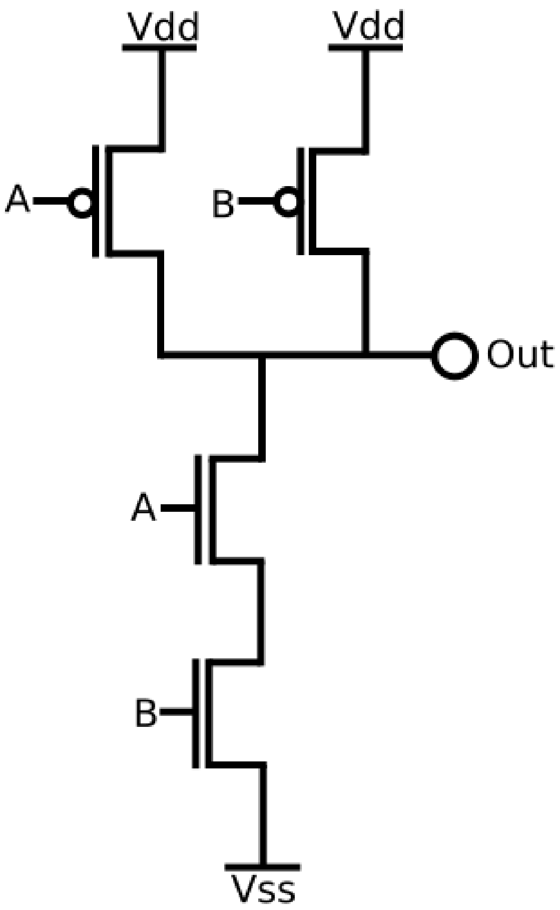
- What is cheating?
  - Copying another student's solution (or one from the Internet) and submitting it as your own
  - Allowing another student to copy your solution
- What is NOT cheating?
  - Helping others use systems or tools
  - Helping others with high-level design issues
  - Helping others debug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

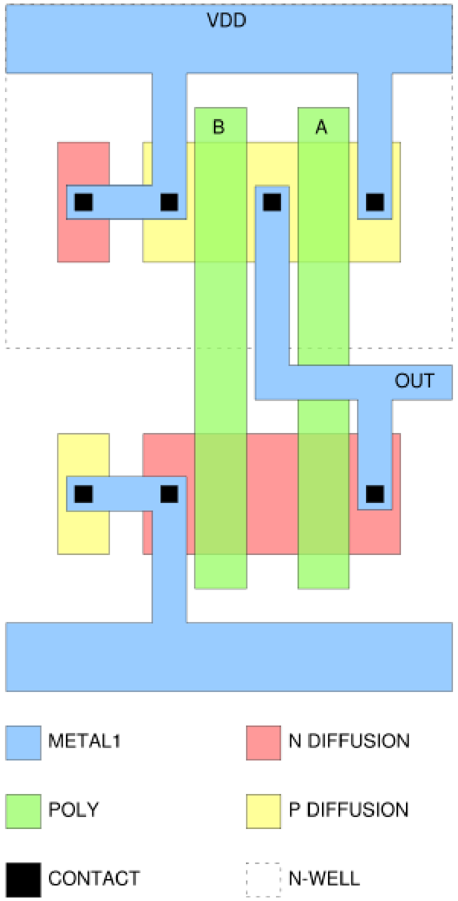
What and Why?

Transistors and Logic Gates

- NAND logic built with CMOS technology







Source: https://en.wikipedia.org/wiki/NAND_gate

How To Run Your Program?

Software
- Application programs
- Data structures & algorithms
- Programming languages (e.g., C)

?

Hardware
- Digital logic
- Transistors

Architecture

Application programs

Data structures & algorithms

Programming languages (e.g., C)

Software

Architecture (or Instruction Set Architecture) → ***Interface between software and hardware***

Digital logic

Transistors

Hardware

Topic 1: How To Design Interface?

- Choices critically affect both the software programmer and hardware designer
- Example: Copying *n* bytes from address *A* to *B*

x86_64 (CISC)

```
movq    A, %rsi
movq    B, %rdi
movq    n, %rcx
REP MOVS
```

RISC-V (RISC)

```
la      a0, A
la      a1, B
li      a2, n
add     a3, a0, a2
```

```
L0:
  lbu     a4, 0(a0)
  sbu     a4, 0(a1)
  addi    a0, a0, 1
  addi    a1, a1, 1
  bne     a0, a3, L0
```

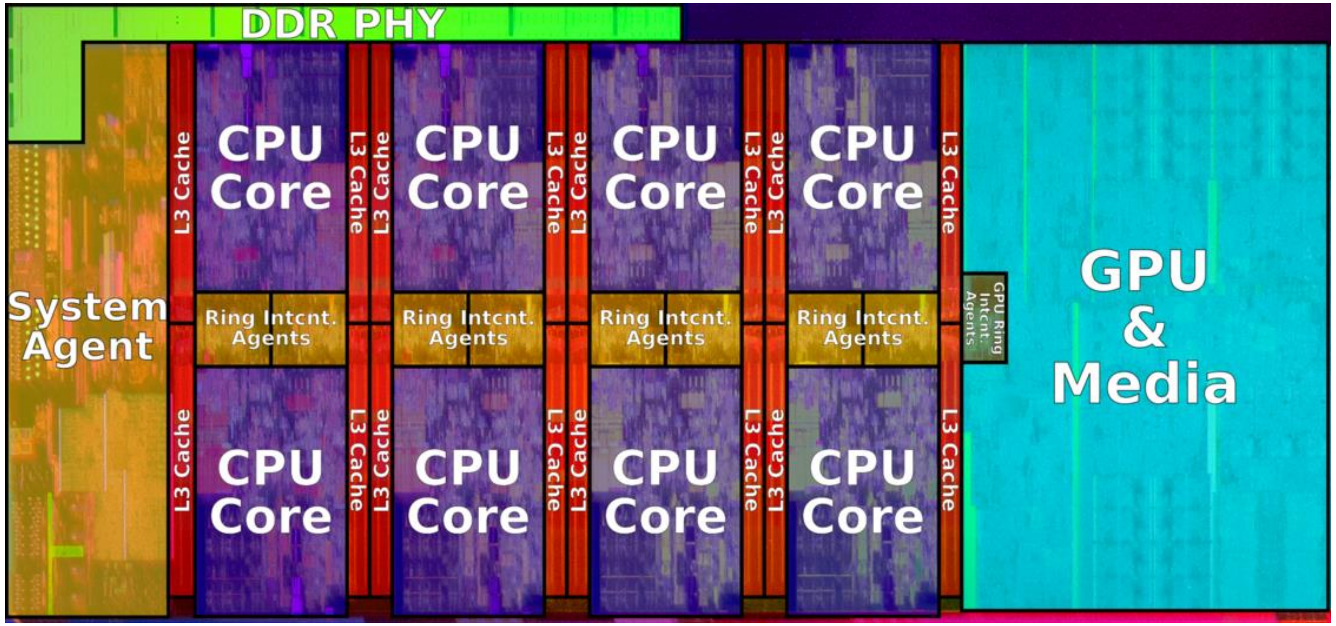
- Trade-offs: code size, compiler complexity, operating frequency, number of cycles to execute, hardware complexity, energy consumption, etc.

Topic 2: How To Implement?

- Microarchitectures: Where should you spend transistors to run your program faster with conforming to the given interface?

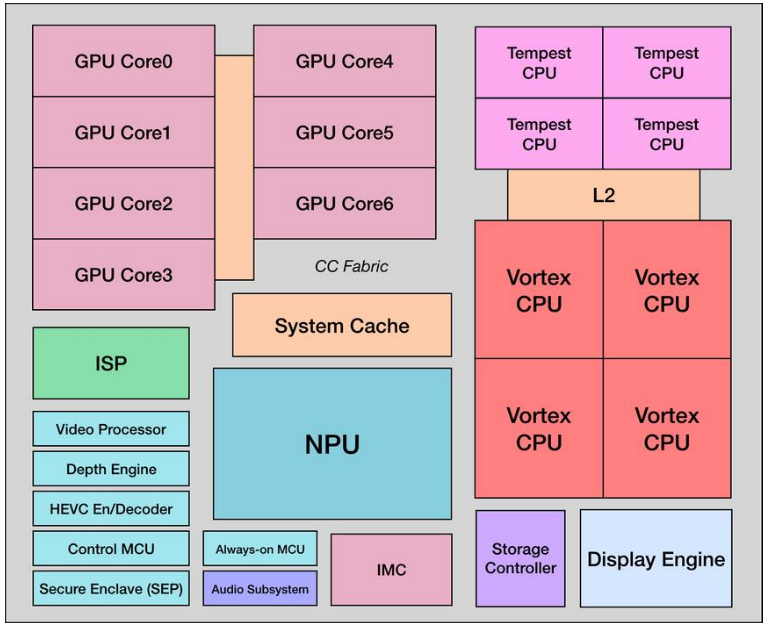
Intel Core i9-9900K (Coffee Lake, 2018)

Transistors: ~ 3B (14nm), Die size: ~ 177mm^2



Apple A12X Bionic (2018)

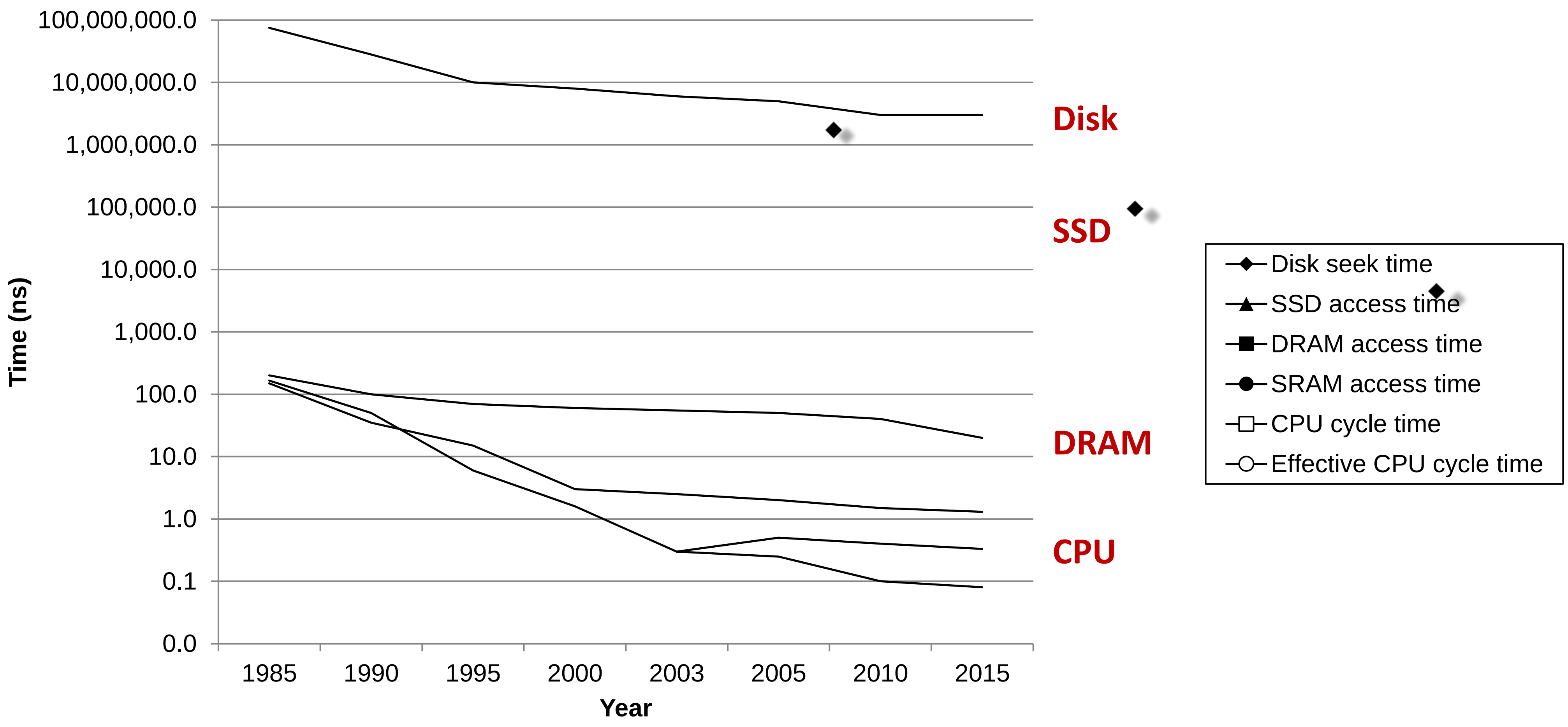
Transistors: ~ 10B (7nm), Die size: ~ 122mm^2



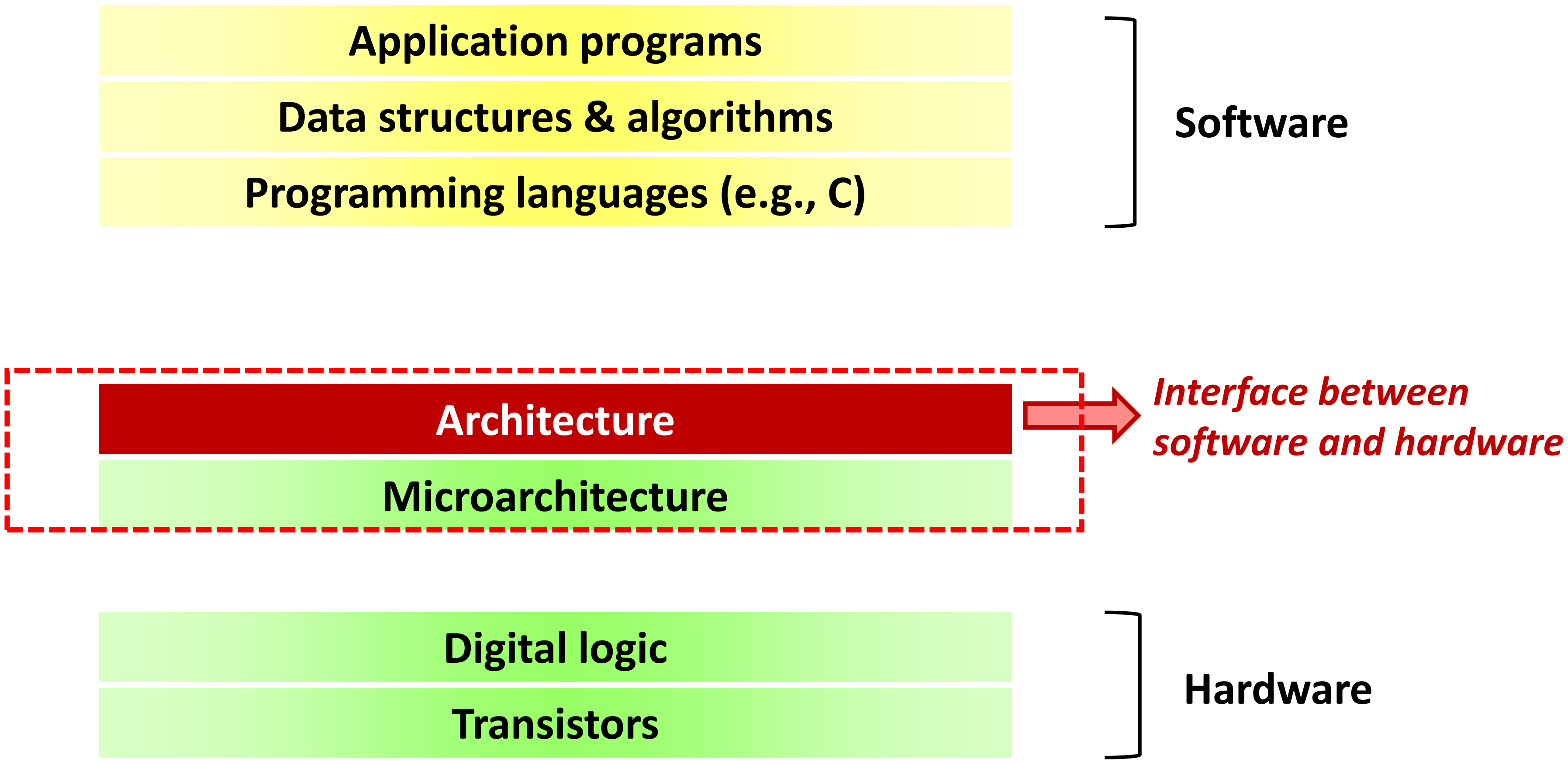
Source: https://en.wikichip.org/wiki/intel/core_i9/i9-9900k, https://en.wikichip.org/wiki/apple/ax/a12x

Topic 3: What About the Memory?

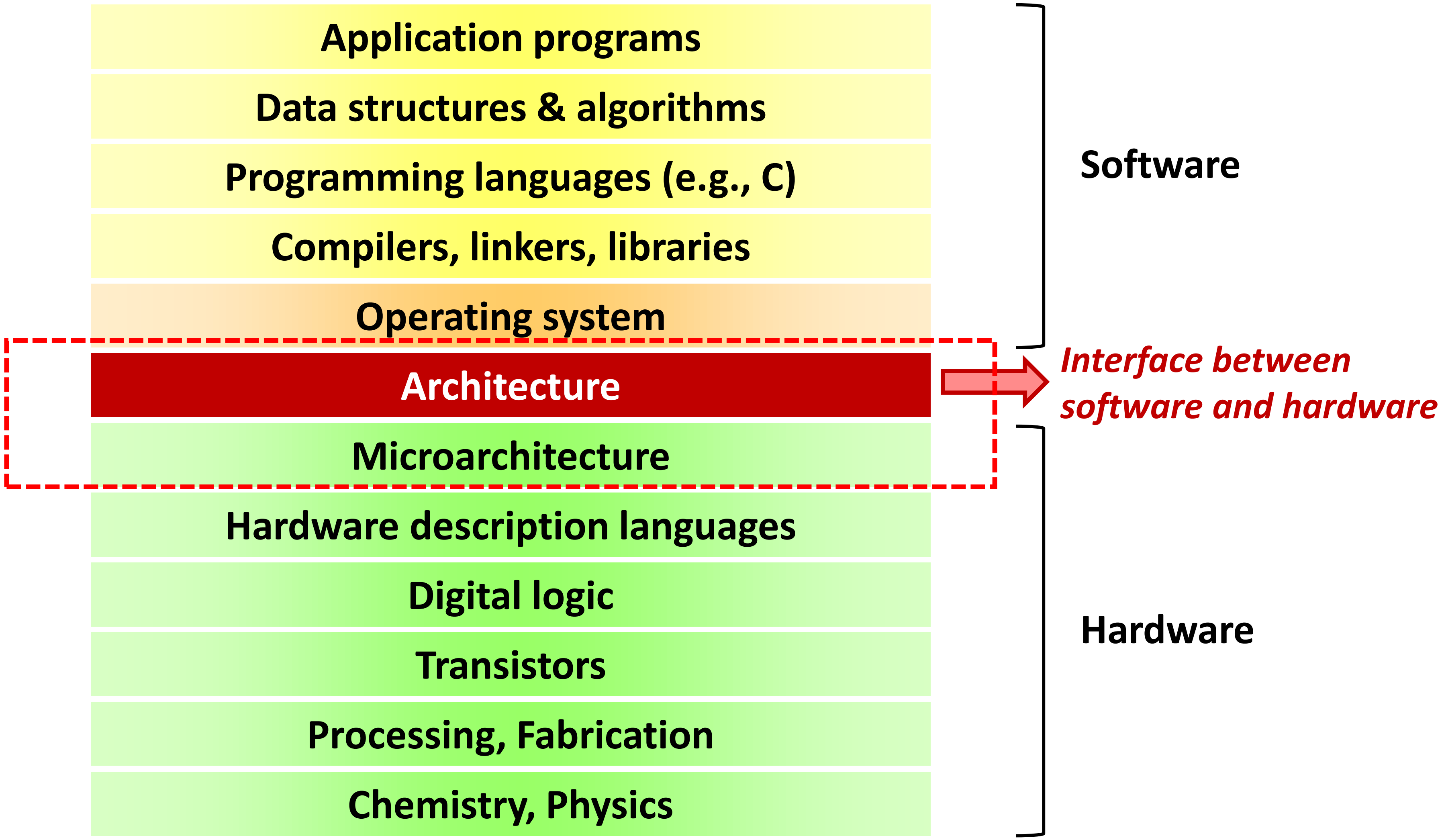
- It's just too slow!



The Scope of This Course

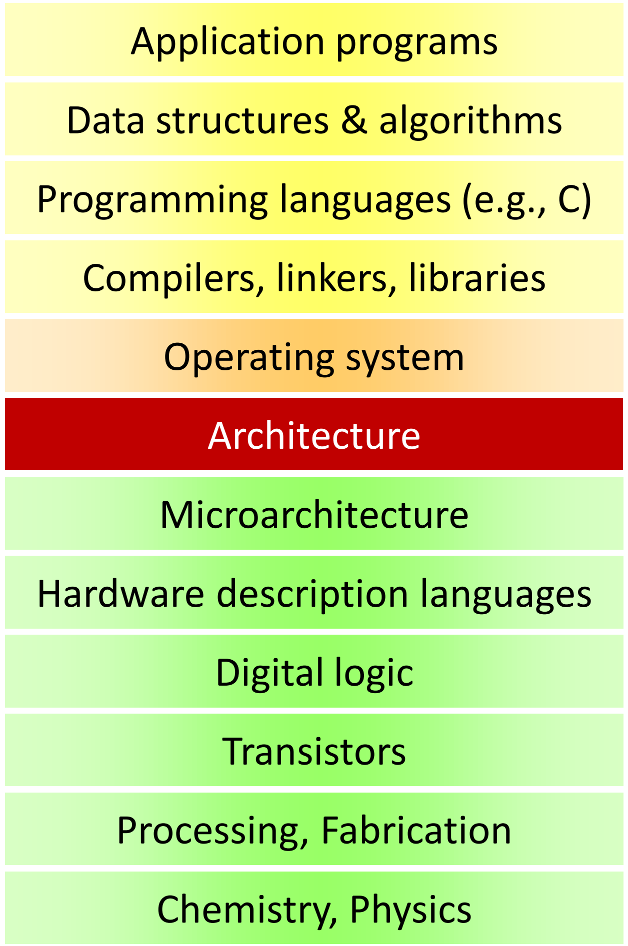


Full Levels of Abstraction



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
- What is the right place to solve the problem?
- 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: More Than Just GHz

CPU	Clock Speed	SPECint2000	SPECfp2000
Athlon 64 FX-55	2.6GHz	1854	1782
Pentium 4 Extreme Edition	3.46GHz	1772	1724
Pentium 4 Prescott	3.8GHz	1671	1842
Opteron 150	2.4GHz	1655	1644
Itanium 2 9MB	1.6GHz	1590	2712
Pentium M 755	2.0GHz	1541	1088
POWER5	1.9GHz	1452	2702
SPARC64 V	1.89GHz	1345	1803
Athlon 64 3200+	2.2GHz	1080	1250
Alpha 21264C	1.25GHz	928	1019

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];
}
```

4.3 ms

```
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];
}
```

81.8 ms

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

Example #4: 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;
}
```

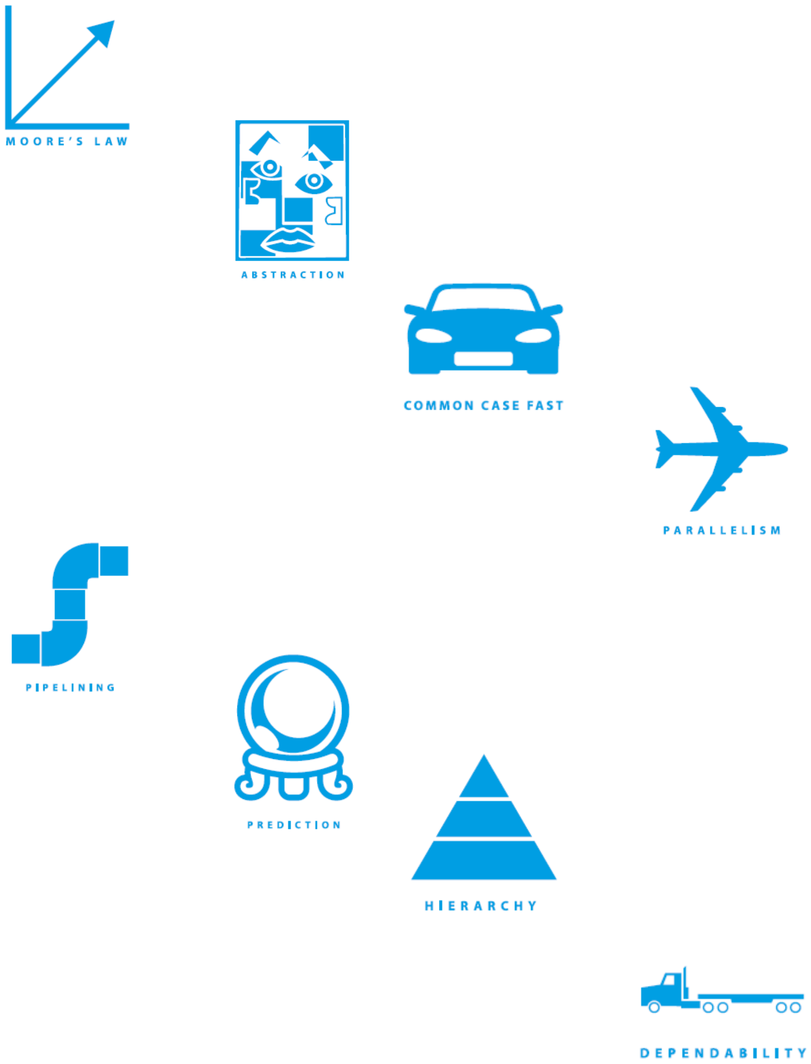
```
$ ./bufdemo
Type:012
012

$ ./bufdemo
Type: 01234567890123456789012
01234567890123456789012

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

What You Will Learn

- 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

Eight Great Ideas in Computer Architecture

- Design for Moore's Law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy



Role of The (Computer) Architect

- Look backward (to the past)
  - Understand tradeoffs and designs, upsides/downsides, past workloads
  - Analyze and evaluate the past
- Look forward (to the future)
  - Be the dreamer and create new designs. Listen to dreamers
  - Push the state of the art. Evaluate new design choices
- Look up (towards problems in the computing stack)
  - Understand important problems and their nature
  - Develop architectures and ideas to solve important problems
- Look down (towards device/circuit technology)
  - Understand the capabilities of the underlying technology
  - Predict and adapt to the future of technology. Enable the future technology

Borrowed from Onur Mutlu's 18-447 lecture Slides

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 and RISC-V?)
- 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
  - RISC-V Instruction Set Architecture (ISA) – A new open interface
  - An implementation based on Pipelining (Microarchitecture) – how to make it faster?
  - Memory hierarchy – How to make trade-offs between performance and cost?
- Understanding Computer Architecture is vital to other "systems" courses:
  - System programming, Operating systems, Compilers, Embedded systems, Computer networks, Multicore computing, Distributed systems, Mobile computing, Security, Machine learning, Quantum computing, etc.