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

Systems Software &
Architecture Lab.

Seoul National University

Fall 2021

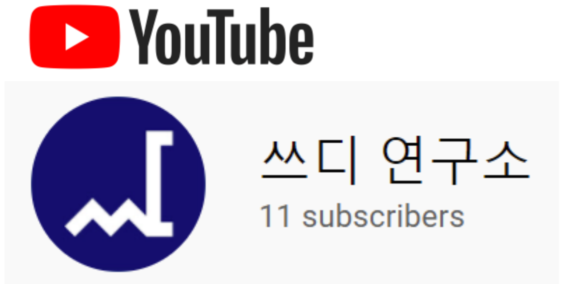
4190.308:
Computer Architecture

Course Information

- Schedule
  - 14:00 – 15:15 (Tuesday & Thursday)
  - Lecture room: ~~Engineering Bldg. #302-208~~ (Online lecture using Zoom)
  - 3 credits
  - Official language: English

- TAs: Jaehoon Shim, Ikjoon Son, Seongyeop Jeong
- SNU eTL system for exam/project scores
- http://csl.snu.ac.kr/courses/4190.308/2021-2/ for announcements and lecture slides
- http://sys.snu.ac.kr for project submissions and automatic grading

About Me



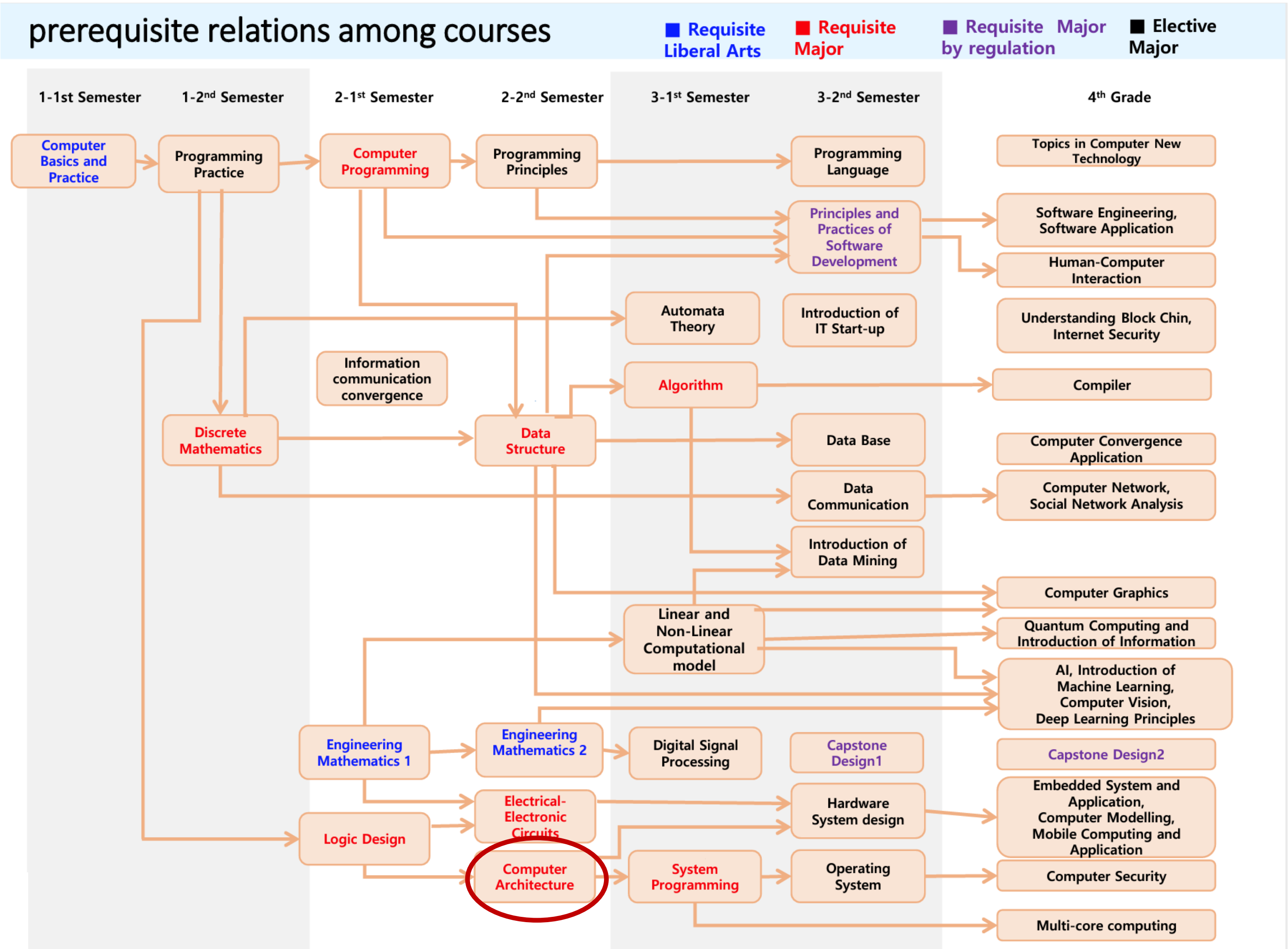
- Jin-Soo Kim (김진수)
  - Professor @ CSE Dept.
  - Systems Software & Architecture Laboratory
  - Operating systems, storage systems, parallel and distributed computing, embedded systems, …

- E-mail: jinsoo.kim@snu.ac.kr
- Tel: 02-880-7302
- Office: Engineering Bldg. #301-504 (office hours: Tuesday & Thursday)
- The best way to contact me is by email

Myths About This Course

- It's an introductory course
  - Introduction to Computers? NO!
  - Past records show that about 20% of students have dropped every semester
- It's all about hardware
  - NO! 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
  - NO! Writing good software requires understanding details of underlying implementation
- Who needs to know the assembly language these days?
  - Well, you'll see...

Where Are We?



Source: https://cse.snu.ac.kr/en/page/degree-requirements

Prerequisites

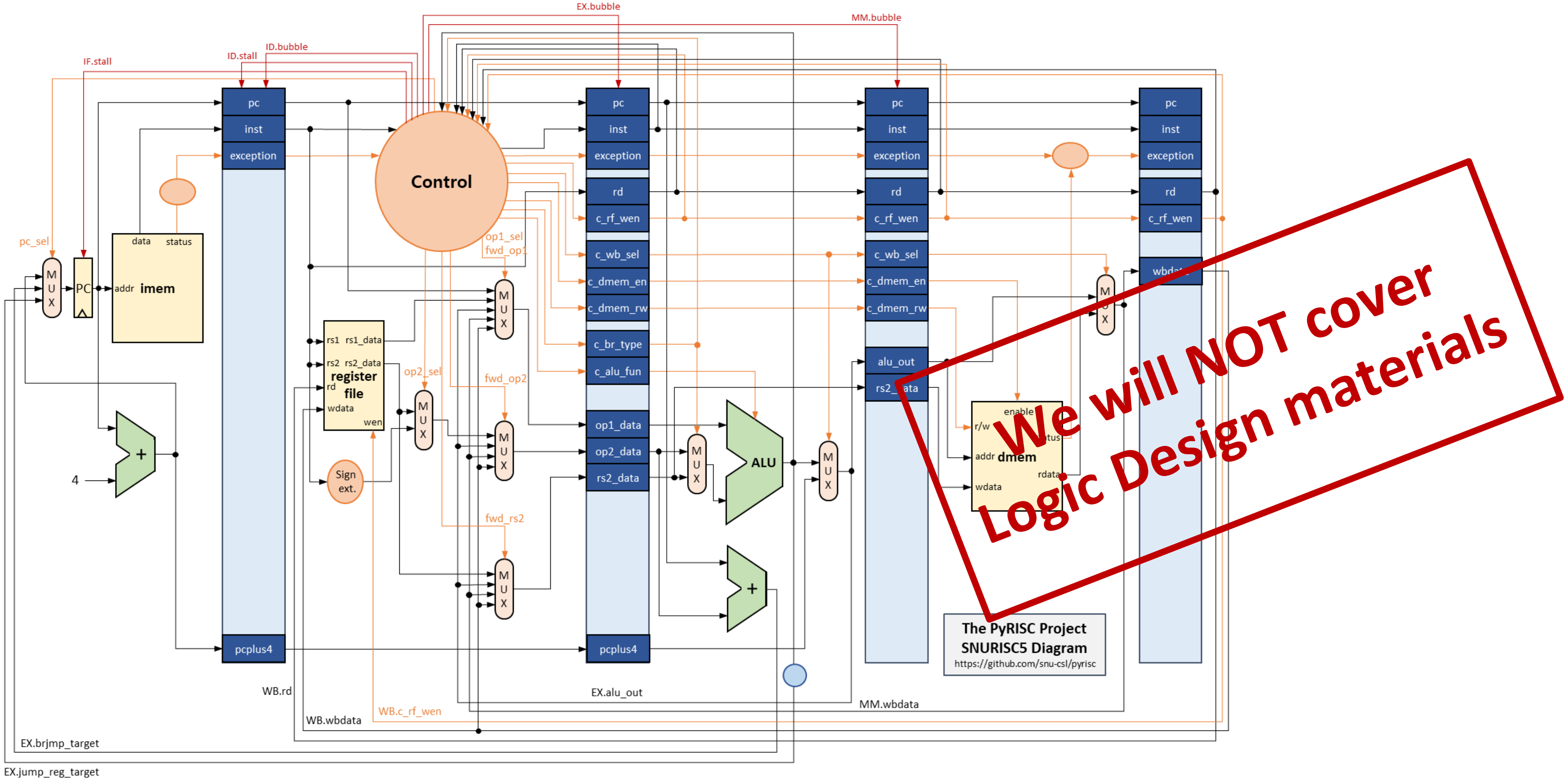
- Prerequisites
  - Programming Practice (4190.103A) – C programming
  - Logic Design (M1522.000700) – Must!
  - Data Structure (M1522.000900) – Recommended

- 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 later) or MacOS machine

Check Yourself: Logic Design

- You should be able to understand how the following circuit works



Check Yourself: C & Linux

- Bit manipulations

```
int f(int x) {
  return (x >> 8) & 0x1f;
}
```

- Pointers

```
int g(float x) {
  return *((int *) &x);
}
```

- Linux development tools

```
$ make
$ gcc -O2 -g -o abc abc.c
```

We will NOT cover C & Linux

Check Yourself: Python

- We are using a CPU simulator (called pyrisc) written in Python
  - Why? Because it is a lot easier than Verilog...
  - If you haven't heard of Verilog, then think again...
  - Pyrisc is available at https://github.com/snu-csl/pyrisc
- You need to change the internals of the simulator in one or two project assignments
  - Lists?
  - Dictionaries?
  - Tuples?

We will NOT teach you Python

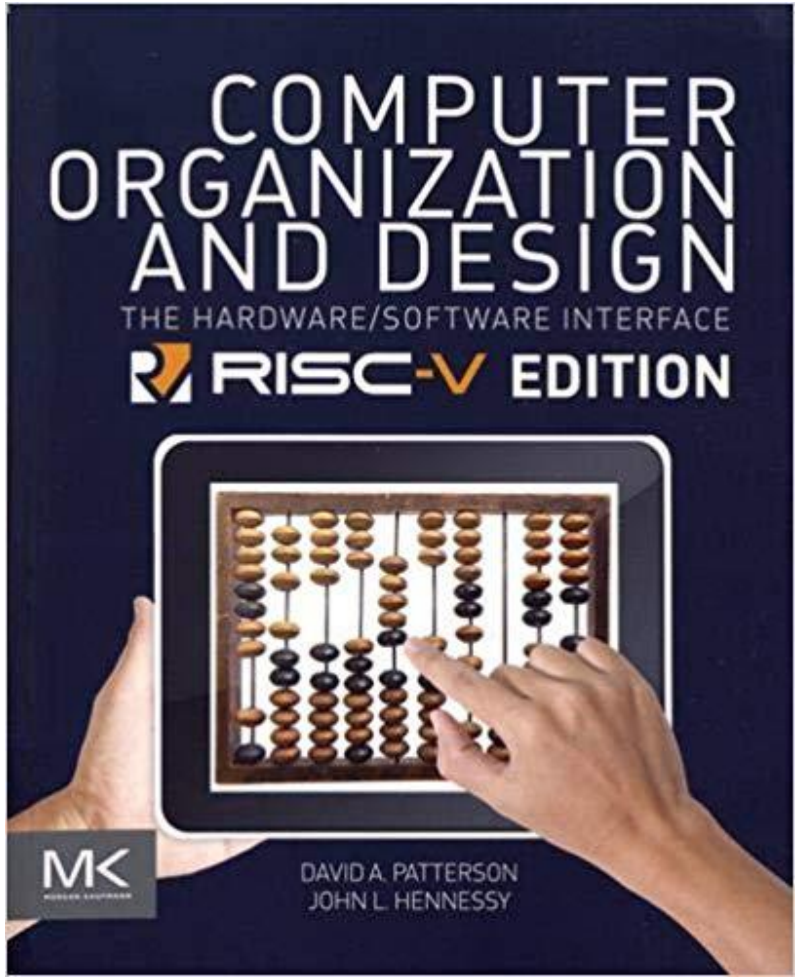
```
def gen(self, inst):
    from datapath import Pipe, EX, MM, WB

    opcode = RISCV.opcode(inst)
    if opcode in [ EBREAK, ECALL ]:
        Pipe.ID.exception |= EXC_EBREAK
    elif opcode == ILLEGAL:
        Pipe.ID.exception |= EXC_ILLEGAL_INST
        inst = BUBBLE
        opcode = RISCV.opcode(inst)
```

A Gentle Reminder

- If you feel that you are not ready yet, then take this course later. Again, remember this is NOT an introductory course!
- It's CSE department's policy that all major/minor students can take the required course whenever you want!
- So, there is no need to rush

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



Topics

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

Project Topics (subject to change)

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

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?

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

Higher is better

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];
}
```

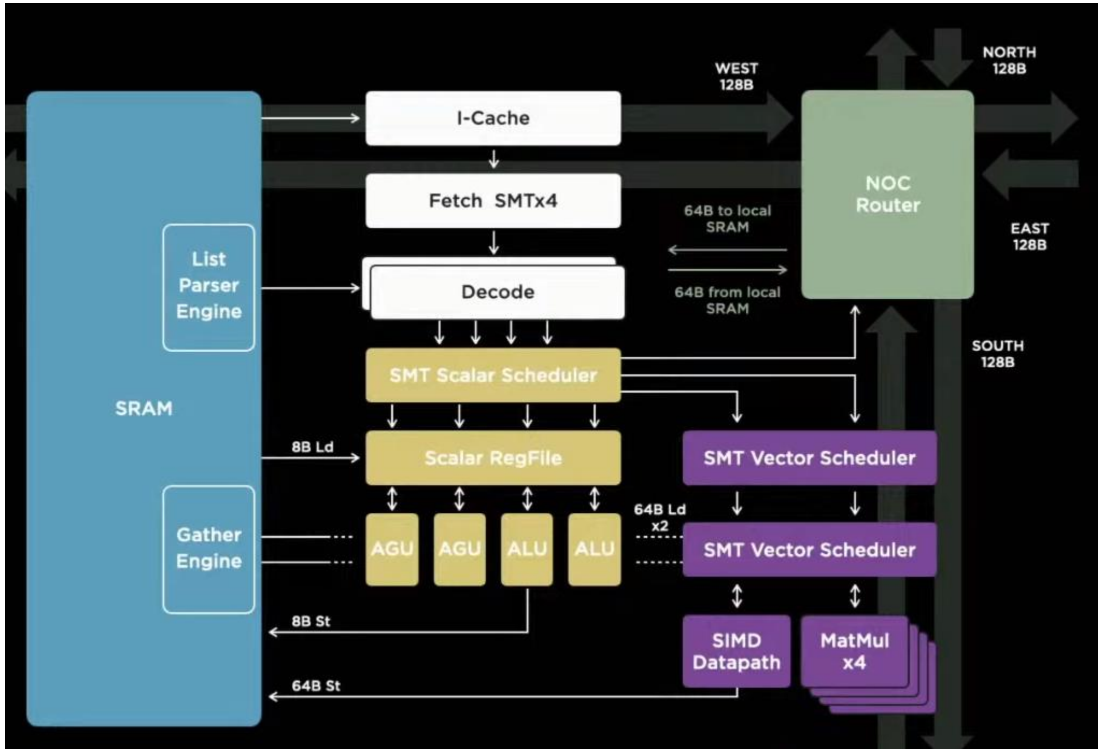
81.8 ms

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

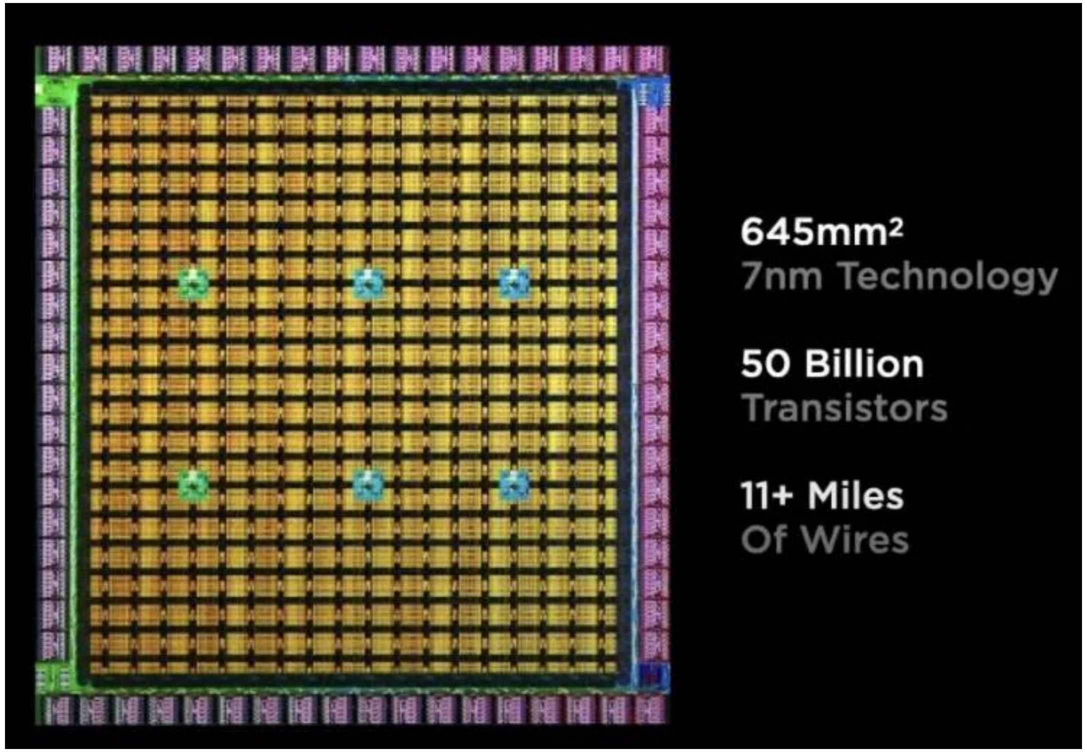
What You Will Learn

- How to represent data
- 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

Tesla AI Chip (2021)



Training Node (1 TFLOPS – BF16)



D1 Chip (354 nodes)

Source: Tesla AI Day 2021 (https://www.youtube.com/watch?v=DSw3IwsgNnc)

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, Tesla, …
  - Still tremendous innovations!
- 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.