Computer Architecture, Lecture 1: What is Computer Architecture?

Hossam A. H. Fahmy

Cairo University

Electronics and Communications Engineering

Building at the nanoscale

According to the 1913 Webster, architecture is:

- the art or science of building;. . . or
- construction, in a more general sense.

Now, we get:

```
$ dict architecture
From wn [wn]:
  architecture

:

   4: (computer science) the structure and organization of a
      computer's hardware or system software; "the architecture
      of a computer's system software" [syn: {computer
      architecture}]
```

The two architectures

Goals	function, cost, safety, ease of construction, energy efficiency, fast build time, aesthetics, ...	⇒ Plans
Materials	steel, concrete, brick, wood, glass, ...	
Buildings	houses, mosques, offices, museums, schools, ...	

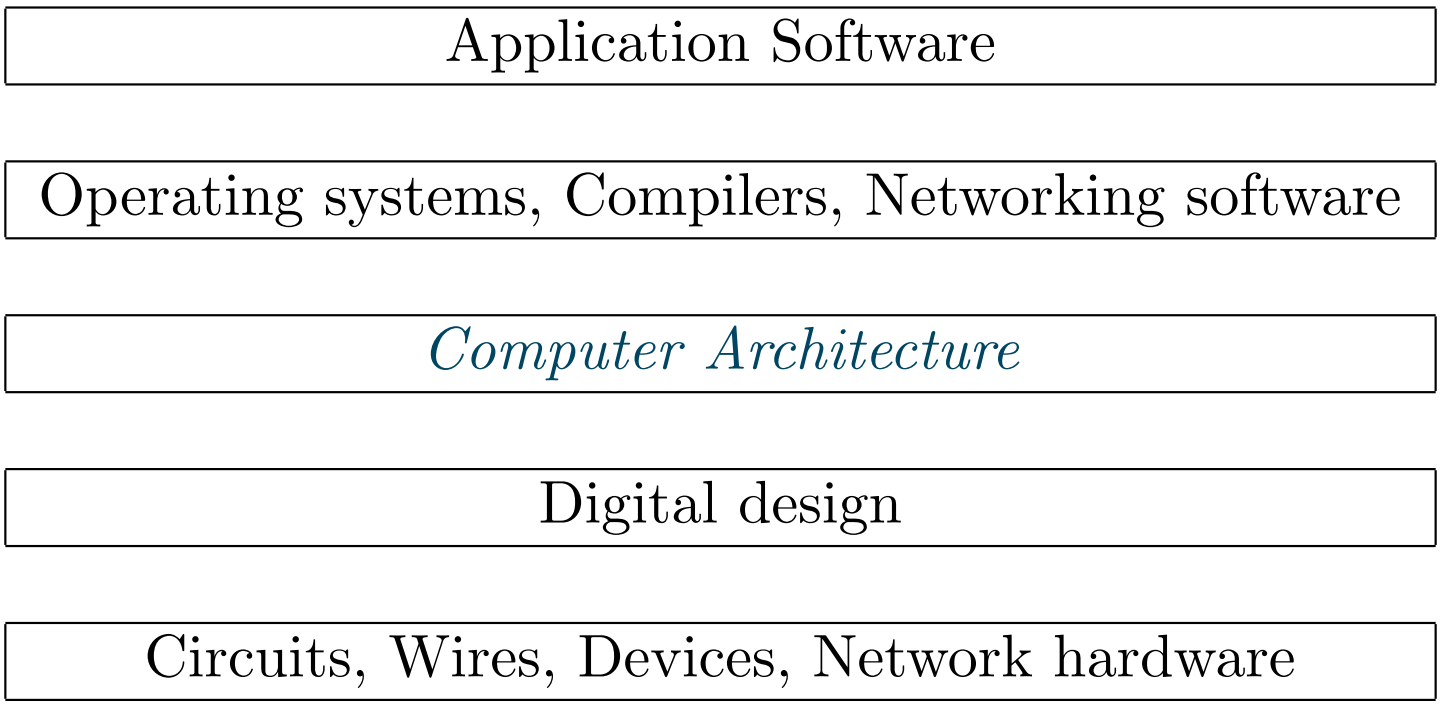
Goals	function, performance, reliability, cost, ease of manufacture, energy effieciency, time to market, ...	⇒ Plans
Technology	logic gates, SRAM, DRAM, circuit techniques, packaging, magnetic storage, ...	
Computers	PCs, servers, PDAs, supercomputers, embedded, ...	

What are the differences?

There are at least three important differences:

1. the age (thousands of years versus about 70),
2. the rate of change, and
3. the automated mass production.

Where does computer architecture fit?



The computer architecture interacts with many fields. It cannot be studied in vaccum.

Design goals

Domain Is it a general purpose processor or domain specific (Graphics, Neural network, Encryption, ...)?

Functional Should be correct! What functions should it support?

Reliable A spacecraft is different from a PC. *Is it really?*

Performance It is not just the frequency but the speed of real tasks. You cannot please everyone all the time.

Low cost design cost (*how big are the teams? How long do they take?*), manufacturing cost, testing cost, ...

Energy efficiency this is the "running cost". Energy is drawn from *various sources*. The cooling is a big issue.

Your job as an architect is to balance all of that given the technologies available and the specific target computer.
The goals, the technologies, and the targets are all changing!

Computer architecture: Structure

Within the processor: Registers, Operational Units (Integer, Floating Point, special purpose, ...)

Outside the processor: Memory, I/O, ...

Instruction Set Architecture (ISA): What is the best for the target application? Why?

Examples: Sun SPARC, MIPS, Intel x86 (IA32), IBM S/390.

Defines: data (types, storage, and addressing modes), instruction (operation code) set, and instruction formats.

Computer architecture: Organization

Within the processor: Pipeline(s), Control Unit, Instruction Cache, Data Cache, Branch Prediction, ...

Outside the processor: Secondary Caches, Memory Interleaving, Redundant Disk Arrays, Multi-Processors, ...

Which implementation is better? How do you define better?

We always optimize according to some purpose (application) that sets the conditions of the problem.

Same architecture and multiple circuits

Realization of the computer: This is the physical fabrication and assembly.
It is possible to implement an architecture (structure and organization) using

- the $130nm$ CMOS technology or the $5nm$ technology,
- 4 metalization layers or 11 metalization layers,
- Aluminum or Copper for the wires, ...

ISA types

The addressing modes define some types:

Load/Store Only the `LOAD` and `STORE` instructions refer to data in the memory. The ALU instructions operate on data in registers. Used by many processors: PowerPC, MIPS, HPPA, SPARC.

R/M The ALU instructions have one source or the destination in memory. Used in mainframes and older microprocessors: IBM S/390, Intel x86 (IA32).

R+M The ALU instructions may have any (or all) the arguments in memory. Not commonly used now but was in the Dec VAX machines.

Which type has a variable instruction size? Why? Is this good?

Data types

1. Integers. How do we represent negative numbers?
2. Floating point. Why not fixed?
3. Decimal digits. Who needs those? Why not floating decimal?
4. Characters. In what encoding? Is it portable?
5. Bit strings. Why use individual bits?

What is the size of each?

Operation types

In a program, we *operate* on the data and *depending* on the results we *choose* among the alternative paths in the algorithm. The program interacts with the world to *get* the initial data and *produce* the final result.

1. Arithmetic or Logical operations handle integer, floating point, decimal, or binary data. Can we operate on characters?
2. Comparison instructions check a condition and produce a TRUE or FALSE result.
3. Control transfer instructions branch (or jump) to another part of the program.
4. Load and Store instructions deal with the memory while I/O instructions interact with other devices. Must they be separate?

Beyond the basics

Computers serve multiple programs and users simultaneously:

- There are instructions to ask for services from another program module or from the operating system and then to return back. How do we pass the arguments and the results?
- To provide some security measures, the instructions that handle critical tasks are used only by the OS. The processor may be in different modes (for example a user mode or a system mode).

Usually a *Program Status Word (PSW)* holds many pieces of information regarding the state of the processor including: condition results, user id, current instruction address, user/system mode, enable/disable traps, ...

The journey from the processor to the memory

There are three distinct levels:

1. The program sees an address space provided at the user level. This is usually a flat linear space going from zero to the maximum memory location allowed for a process.
2. The system sees a full system address space. To accomodate multiple users and programs, the system relocates the programs and provides protection against unauthorized memory access.
3. The hardware deals with the reality. With the use of caches and virtual memory, the hardware resolves the virtual address to get the required information from its actual location. The primary concern is to shorten the average access time.

As a computer architect, you design whatever is required at each level. *What do you think is needed?*

Moore's law

"Cramming More Components onto Integrated Circuits" by Gordon E. Moore, Electronics, 1965.

- Observation: transistor density doubles annually. He was slightly off. Since then the density doubled almost every 18 months (he had only four data points).
- Corollaries:
  - the cost per transistor halves at the same rate,
  - the speed increases with scaling, and
  - the performance doubles almost every two years.

Do the periods (creation–2022) and (2022–2024) have the same absolute increase in computing power per processor core?
Ans: This *was true* in previous periods. Now, parallel cores are the trend.

A look at the trends

Year	Model	CPU cores (Perf+Eff)	GPU cores	Neural cores	Memory	Transistors count
2020	M1	8 (4+4)	8	16	8-16 GB	16 G
2021	M1 Pro	10 (8+2)	16	16	16-32 GB	34 G
2021	M1 Max	10 (8+2)	32	16	32-64 GB	57 G
2022	M1 Ultra	20 (16+4)	64	32	64-128 GB	114 G

Key issues

- Current architectures have multiple pipelines and caches. They include several cores per chip.
- It is important to evaluate the various options that become available with improvements in technology.
- A good (cost-effective) allocation of resources is necessary for the design's success in the market.
- A high production volume is critical to recover costs.

In this class, you will learn to analyze, model, and look at design targets. Are you up to the challenge of keeping (or improving!) the historical trends?