DSC-204A: Scalable Data Systems, Fall 2025

2: Computer Organization & Data Representation

Lecturer: Hao Zhang *Scribe: Yuheng Zha, Yuyuan Wu, Brandon Hsu*

Introduction and Logistics

- **Outline:** High-level introduction to the fundamentals of computer organization, followed by technical content on computer data representation.
- **Class recap:** We emphasized how computers *store, process, and retrieve* data, the trade-offs across the *memory hierarchy* (latency, bandwidth, capacity, cost), and how *floating-point representations* impact numerical precision and performance (including mixed-precision in ML).
- **Survey + extra credit detail:** Please finish the *Beginning of Quarter* survey by **10/10**; if $\geq 80\%$ complete it, everyone receives **+1 point**. TAs will post completion percentages.
- **Enrollment/waitlist:** Instructor approves anyone in **EASy**. If you're on the list, wait until the **end of this week**; new slots usually open because some students drop after seeing **PA1**.
- **Course map (high level):** Foundations of Data Systems → Cloud → ML Systems → Big Data (historical arc touches 1980–2000 and onward).

1 Basics of Computer Organization

Computers are the foundation of modern data systems.

Question: What is a computer?
Answer: ***Programmable*** *electronic device that can* ***store, retrieve,*** *and* ***process*** *digital* ***data.***—Peter Naur

Question: What are computers composed of?
Answer: **Hardware**: Electronics (wires, circuits, transistors, capacitors, devices, etc.), **Software**: Programs (instructions for how to process data) and data

From a data-centric perspective, our goal in building computers is to store and process data at scale.

- To store and retrieve data, we need **Disk** and **Memory**:
  - Question: Why we need both?
  - Answer: Because disk provides long-term storage but is slow, while memory is faster but only retains data short-term. (We will introduce a core concept called memory hierarchy later in the class to deal with these trade-offs.)
- To process data: Processors (**CPU**, **GPU**, etc.). Why we have different processors? Because different applications have different computational requirements (parallel processing (GPU) is ideal for Deep Learning applications).
- To retrieve data from remote: **Networks** (communication highway between many different computers)

2 Hardware

2.1 Key Parts of Computer Hardware

- **Processor (CPU/GPU, ...):**
  - Orchestrates and executes low-level instructions (machine language defined by an ISA).
  - Each processor architecture has its own finite instruction vocabulary (the ISA).
  - CPUs are general-purpose; GPUs are specialized for massive parallelism (common in DL/ML workloads); increased compute demand motivates reduced-precision arithmetic where appropriate.
- **Main Memory (RAM / DRAM):**
  - Volatile storage for fast, random access to data/instructions during execution.
  - Byte-addressable; all addresses are accessible at low latency (relative to disk/network).
  - Sequential and random data access is low latency with main memory.
- **Disk (Secondary / Persistent Storage):**
  - Non-volatile; higher capacity and lower cost/byte than RAM, but higher latency.
  - *Contrast with memory:* memory is faster but volatile and expensive/byte; secondary storage is slower, persistent, and cheaper/byte. Data must be loaded into memory before computation.
  - Only sequential data access is low latency with disk storage.
- **Network (Remote data):**
  - Enables sending/receiving data to/from remote machines and cloud storage.
  - Typically highest latency but effectively unbounded capacity via remote resources.
  - *NIC (Network Interface Controller):* hardware that transmits/receives data.

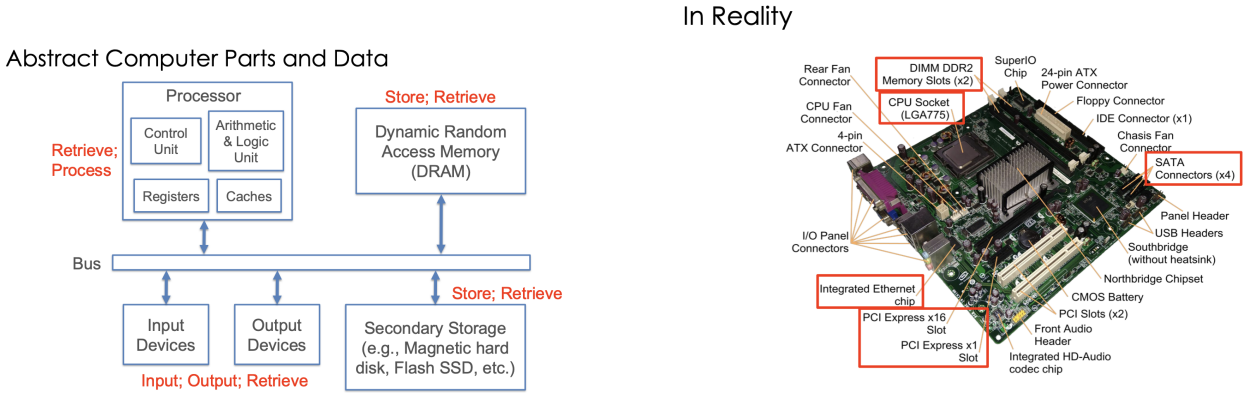
2.2 Abstract vs. Physical View

Putting all these components together results in Figure 1a which is implemented in every modern computer system. All components communicate with each other through the bus/interconnect, which acts as the local data transmission medium. In reality, computers are much more complicated and come with a variety of additional components as depicted in Figure 1b. The board shown is called the motherboard and it contains different components such as SATA connectors (interface for external storage), integrated Ethernet chip, PCI express (PCI-E) slots (interface for external GPU), CPU socket, memory socket, etc.

3 Software

3.1 Key Aspects of Software

- **Instruction:** Command understood by hardware. The **Instruction Set Architecture (ISA)** bridges hardware and software (x86, MIPS, RISC-V, etc.).
- **Program(code):** Collection of instructions for hardware to execute.



(a) Abstraction (b) Reality

Figure 1: Computer organization. (a) Abstract blocks: processors, memory (DRAM), secondary storage, I/O, connected by buses and interconnects (communication highway). (b) Real motherboards/systems are much more complex.

- **Programming Language (PL):** Human-readable language to write programs at a higher abstraction than the ISA.
- **Application Programming Interface (API):** Set of functions/interfaces exposed by software for use by developers or other programs.
- **Data:** Digital representations of information that software stores, processes, displays, retrieves, or transmits.

3.2 Main Kinds of Software

- **Firmware:** Read-only code "baked into" devices for basic control.
- **Operating Systems:** Programs that enable application software to use hardware more efficiently (process, memory, and device management). Famous examples: Linux, Windows, macOS, etc.
- **Applications:** Collection of interrelated programs to manipulate data, typically designed for human use (e.g., Excel, Chrome, PostgreSQL).

4 Digital Representation of Data

4.1 Basic Concepts

- **Bit**: All digital data are sequences of 0 and 1 (binary digits).
- **Data type**: First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the programming language. e.g.: Boolean, Byte, Integer, Float, Character, and String.
- **Data structure**: A second layer of abstraction to organize multiple instances of same or varied data types as a more complex object with specified properties. e.g.: Array, Linked list, Tuple, Graph.

4.2 Bits and Bytes

- All digital data are sequences of **bits** (0 or 1). We "count" in base 2.
  - e.g.: represent 15213_{10} as 0011101101101101_2 (16-bits)
- **Byte** = 8 bits (why? because of historical development and practical standardization).
- A Byte is typically the basic unit of data types, and most CPUs are byte-addressable (cannot address units smaller than a byte).
- Notation: lowercase b for bits, capital B for bytes.
- How to represent negative numbers? (will discuss later)

4.3 Bytes and Data types

- **Data types** assign semantics to bit patterns (e.g., `bool`, `int`, `float`).
- The size and interpretation of a data type depends on programming language.
- Data structures organize multiple values into richer objects (arrays, structs, tuples, strings, etc.).
- Example: Boolean, just 1 bit needed but actual size is almost always 1B; Integer, typically 4 bytes; many variants (short, unsigned, etc.)

Type	Bytes	Notes
`bool`	1	Boolean value
`int32`	4	32-bit integer
`float16`	2	IEEE half precision
`bfloat16`	2	Brain floating point 16-bit
`float32`	4	IEEE single precision
`float64`	8	IEEE double precision

Table 1: Common numeric data types and their sizes.

4.4 Concept Checks (final exam)

- **Q1**: How many unique data items can be represented by 3 bytes?
  - **Answer**:2^{24}
  - Because given k bits, we can represent 2^k unique data items. 3 bytes = 24 bits $\Rightarrow 2^{24}$ items.
  - Common approximation: $2^{10} \approx 10^3$; KiB (1024 B) vs KB (1000 B), often approximated as equal for back-of-the-envelope math.
- **Q2**: How many bits are needed to distinguish 97 data items?
  - **Answer**:7 bits
  - Because for k unique items, invert the exponent to get $\log_2(k)$. But the number of bits must be an integer. So, we only need $\lceil \log_2(k) \rceil$. So, we only need the next higher power of 2. $97 \rightarrow 128 = 2^7$, and the result is 7 bits.

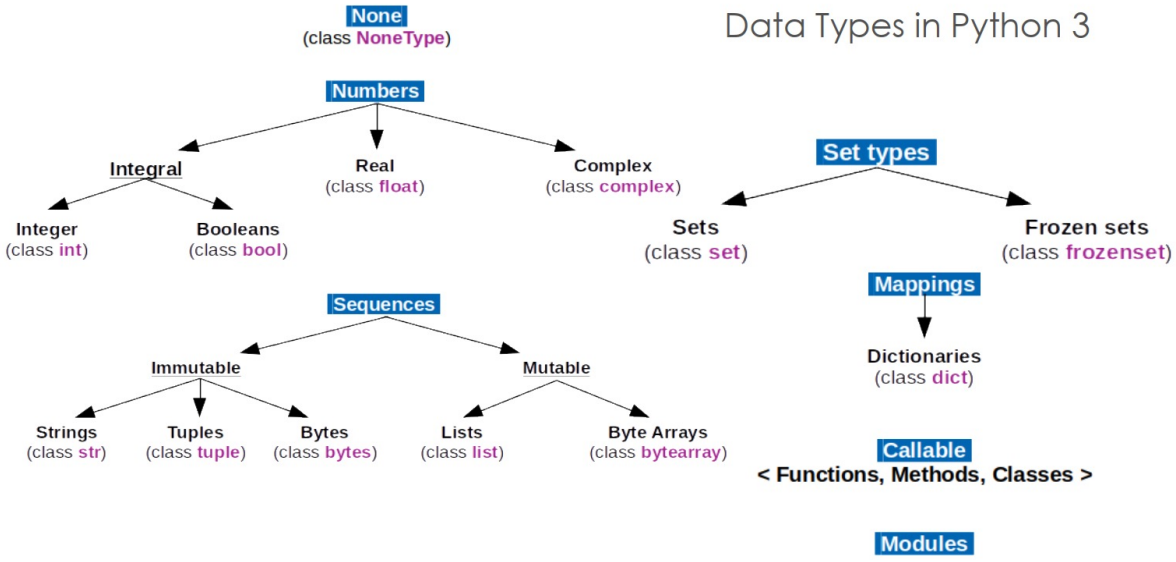


Figure 2: Data Types in Python 3

- **Q3**: How to convert from decimal to binary representation?
  - **Answer**: Repeatedly subtract highest powers of 2 (or use division by 2 with remainders)
  - To convert decimal n to binary: find the largest $2^k \leq n$, set bit k, subtract, and repeat; fill remaining lower bits with 0s. Example values discussed in class: 5_{10}, 47_{10}, 163_{10}, 16_{10} (Figure 3).

	7	6	5	4	3	2	1	0
Decimal	128	64	32	16	8	4	2	1
5_{10}						1	0	1
47_{10}			1	0	1	1	1	1
163_{10}	1	0	1	0	0	0	1	1
16_{10}				1	0	0	0	0

Figure 3: Conversion Examples

  - For fractional decimals like 1.20_{10}, binary may repeat (e.g., repeating 0011 pattern).
- **Q4**: How to convert from binary to decimal?
  - **Answer**: Multiply each binary digit by 2^n according to its position and sum the results.

4.5 Integers

- Encoding Integers:

  - **Unsigned:** standard base-2 representation (if you are pretty sure that you are only going to work on positive Integers or only work on negative Integers, you can use Unsigned).
  - **Signed (two's complement):** negate by inverting bits and adding 1 (if you are going to use positive and negative Integers).

- For n bits, two's-complement range is $[-2^{n-1}, 2^{n-1} - 1]$.
- Example: -5 in 8-bit two's complement is $1111\,1011_2$. `short int x=15213` encodes as $0011\,1011\,0110\,1101_2$; `short int y=-15213` encodes as $1100\,0100\,1001\,0011_2$.
- Worked example (positives/negatives): with bit weights $\{-16, 8, 4, 2, 1\}$: $10 = 01010_2 = 8 + 2$, and $-10 = 10110_2 = -16 + 4 + 2$ (illustrates signed weights in two's complement) (Figure 4).

```
        -16  8  4  2  1
10 =     0   1  0  1  0      8+2 = 10

        -16  8  4  2  1
-10 =    1   0  1  1  0      -16+4+2 = -10
```

Figure 4: Two-complement Example

- **Overflow:** selecting a type with insufficient range can cause overflow and lead to unexpected results (language-dependent), so choose types carefully.

4.6 Floating-Point

A floating-point number encodes: **sign bit**, **exponent** (with bias), and **fraction/mantissa**.

- **Sign**: 0 for positive, 1 for negative
- **Exponent** primarily controls *range* (how large/small values can be).
- **Fraction (mantissa)** primarily controls *precision* (how many significant digits you can represent).

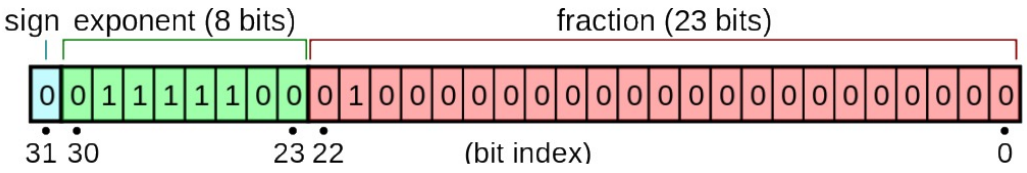


Figure 5: Standard IEEE format for single (aka binary32)

Precision: IEEE-754 single-precision format is 4B long; double-precision format is 8B long.

- Precision is *non-uniform* across magnitudes: large values lose absolute fractional detail; *subnormals* extend range at reduced precision.

$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

$$(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625$$

Figure 6: How to Calculate

- **Precision–performance trade-off (discussion highlight):** lower-precision formats can yield more FLOPs/watt and higher throughput (e.g., mixed-precision training) at the cost of numerical precision; stability depends on task and algorithm.
- **Language note:** Java/C `float` is single precision (4 B); Python `float` is double precision (8 B). **Mixed precision speedups:** industry practice reports ∼2–3× speedups when using mixed precision with appropriate kernels while maintaining similar accuracy (task-dependent).

More Common formats: `float16`, `bfloat16`, `float32`, `float64` (and, in ML contexts, `fp8` variants).

- **bfloat16 vs. float16:** both are 16-bit; bfloat16 keeps an 8-bit exponent (like `float32`) for wide range but fewer mantissa bits (less precision), while IEEE `float16` has a smaller exponent (narrower range) but more mantissa bits (more precision). `float16` is now common for deep learning parameters.

Figure 7: Difference between bf16 and fp16

- **bf16 vs. fp32:** Conversion between fp32 and bf16 is effortless. bf16 has an 8-bit exponent while fp32 also has an 8-bit exponent.
- **Hardware (NVIDIA GPUs)**: By reducing the precision (e.g., from fp64 to fp8), you would observe almost a linear increase in FLOPs. Major deep learning frameworks use mixed-precision training.

4.7 Character (char) and String

- A string is typically just a variable-sized array of char.
- ASCII was originally 7-bit (later 8-bit extensions). UTF-8 (Unicode) is now standard and backward-compatible with ASCII, supporting ∼1.1M code points (often up to 4 B per code point).

4.8 Digital Objects

- All digital objects are collections of basic data types (bytes, integers, floats, and characters)
- Everything from SQL rows to ML model weights is ultimately bits-on-disk.
- Storing/processing large models highlights RAM vs. disk cost/latency differences; reduced precision can lower memory/computation costs in practice (with care).

5 Practice Questions (final exam)

- **Q1**: How many space do I need to store GPT-3?
  - **Answer**: $175 \times 10^9 \times 2$ bytes = **350** B bytes = **350** GB .
  - GPT-3 is an ML model with trained weights, so the question is the same as how much space do I need to store the Python files and weights.
  - The data type of GPT-3 is `bf16`(16-bits, 2 bytes/param) and GPT-3 is a model with 175B parameters. Therefore, the answer is $175 \times 10^9 \times 2$ bytes = **350** GB (weights only).
- **Q2**: What do exponent and fraction control in float point representation?
  - **Answer**: Exponent controls **range/scale** (how large/small values can be). Fraction controls **precision** (detail/resolution of values).
  - Any problem about floating point (compared to fixed point)? Answer: More complex (to both human and computers); Inconsistent precision.
  - Fixed vs. Floating Point (discussion highlight): Fixed point splits integer and fractional parts at a fixed position (simple and fast but inflexible, limited dynamic range). Floating point offers wide dynamic range with magnitude-dependent precision, which better fits many scientific/ML workloads despite added complexity.
- **Q3**: What is the difference between BF16 and FP16? (will discuss in next lecture)
  - `bfloat16`: keeps a **larger exponent** (like `float32`) and a **smaller fraction** ⇒ **wider range**, **lower precision**.
  - `float16`: uses a **smaller exponent** and a **larger fraction** ⇒ **narrower range**, **higher precision**.