Architecture and Memory: A Deep Dive aided by Artificial Intelligence

Alp Kale - Ege Kaya - Bora Demirtola

About the lecture today...

We are conducting research to assess the ability of AI of generating lectures suitable for university-level courses.

Primary objective is to determine if AI-generated material are appropriate for teaching purposes.

All material presented in this presentation have been generated by AI— from text to visuals, utilizing information from **many different** AI agents such as: **ChatGPT, Gemini, Deepseek, and more..**

Additionally, we have written a comprehensive report detailing our observations and the methodology employed. The report, slides, and all texts and visuals generated during the research will be available on our **GitHub page.**

Please pay attention as we conclude with a **quick quiz** measuring retention and a **questionnaire** to gauge the quality of the material generated! :)

Topics tackled today:

- Basic Components of a Computer System
- Memory Organization & Addressing
- Endianness and Byte Ordering
- D Flip-Flop Memory Architecture
- Design Alternatives: Decoder vs. Multiplexer
- ROM-based Logic and Non-Conventional Memory Usage
- Addressing Architectures (2+3 vs. 3+2)
- The Memory Wall Problem
- DDR RAM and Parallelism
- Custom Architectures for Specialized Use Cases

First off, quick recap.

What are the fundamental components of a computer?

A basic computer has:

- **CPU**: Executes instructions and controls operations
- **Memory**: Stores programs and data
- **I/O Devices**: Interfaces for user and external device communication
- **Buses**: Highways that move information between components (Address, Data, Control)
    - **Address Bus**: Carries the address of memory locations (can address 2^n cells)
    - **Data Bus**: Carries data (m bits per cycle)
    - **Control Bus**: Sends read/write/status signals (e.g., Ready, Completed)

Memory Organization & Addressing

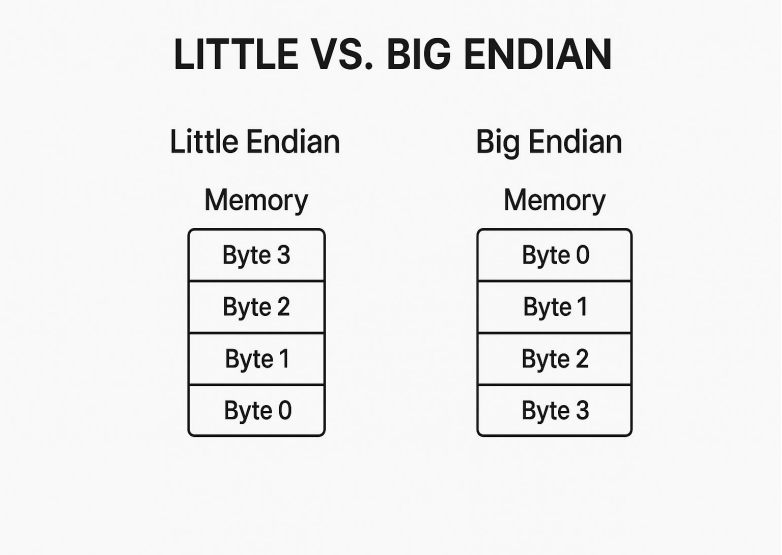
- **Address Space**: The range of unique addresses the CPU can use to access memory cells.
- Most common organization: **byte-addressable memory**
- Data is stored in **groups of bytes**, e.g., a 32-bit word occupies 4 bytes
- The CPU reads/writes these 4 bytes in groups
- Every byte has its own address – this is called **byte-addressable memory**
- For a 32-bit word (4 bytes):
  - The first byte has address X, next is X+1, then X+2, X+3
  - 2 bits are enough to choose 1 byte among the 4 inside a word

Endianness Explained

Endianness defines how multi-byte data is stored:

- **Big-endian**: Stores the most significant byte first (at the lowest address).
- **Little-endian**: Stores the least significant byte first.
- **Why it matters**:
  - Affects data exchange between different systems
  - Important for serialization and network protocols
- **Examples**:
  - Intel CPUs: Little-endian
  - ARM CPUs: Configurable (bi-endian)

Important: Misinterpreting endianness can corrupt data



Prompt: Diagram – Little vs. Big Endian [Illustration: Memory with 4 bytes shown in different orders for little and big endian representations]

Static RAM with D Flip-Flops

1. Flip-Flops Store Bits:

- A **D flip-flop (D-FF)** stores 1 bit of data.
- Controlled by a **clock** and **enable** signal.
- Used in **registers** and **small, fast memories**.

2. Building Multi-Bit Words:

- Combine D-FFs **in parallel** to store multiple bits (e.g., 4 D-FFs = 4-bit memory word).
- All flip-flops in a word share the **same enable/control** logic.

3. Selecting the Right Word:

- Use a **decoder** to select which memory word to access.
- Decoder takes **address bits** as input and activates only one output line.

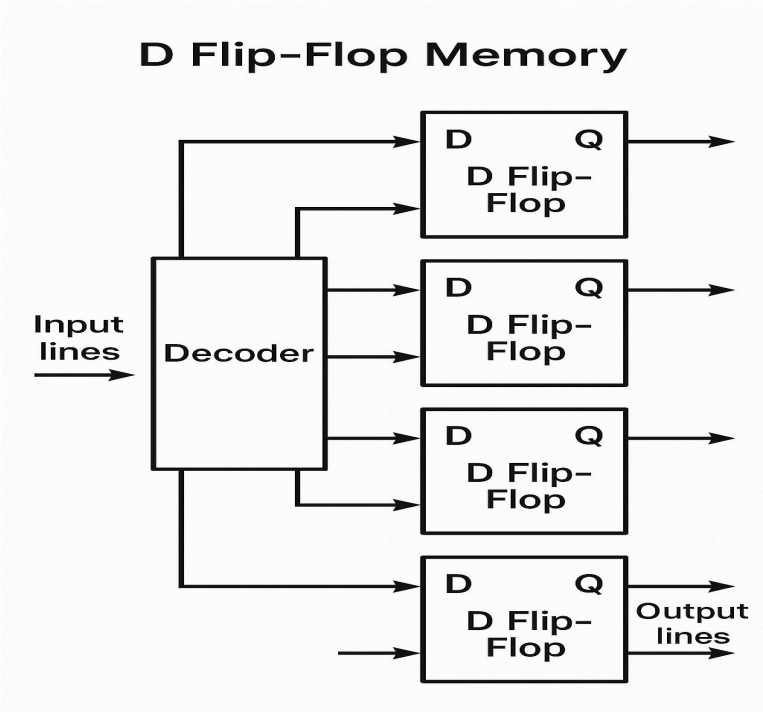
Example: A 4-bit memory block uses 4 D flip-flops to store one word, with a decoder enabling one block at a time.

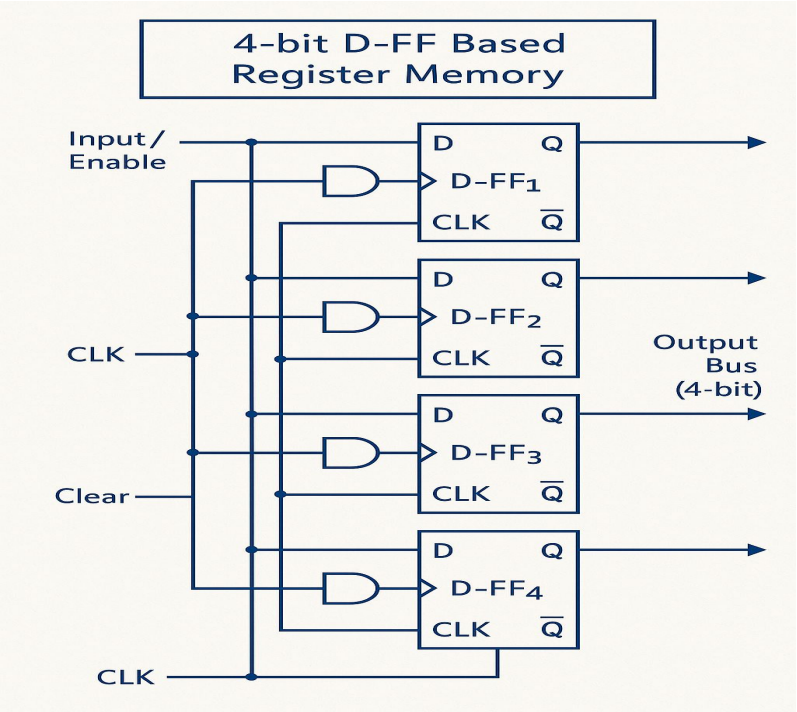
To expand memory, add more address bits and a larger decoder for more words, or add more flip-flops in parallel for wider data without changing the decoder.

This design suits small, fast storage like registers and scales by increasing either word count or data width.

Less Specified Prompt

More Specified Prompt





Prompt: Diagram – D Flip-Flop Memory [Block diagram showing input lines, decoder, D flip-flop registers, and output]

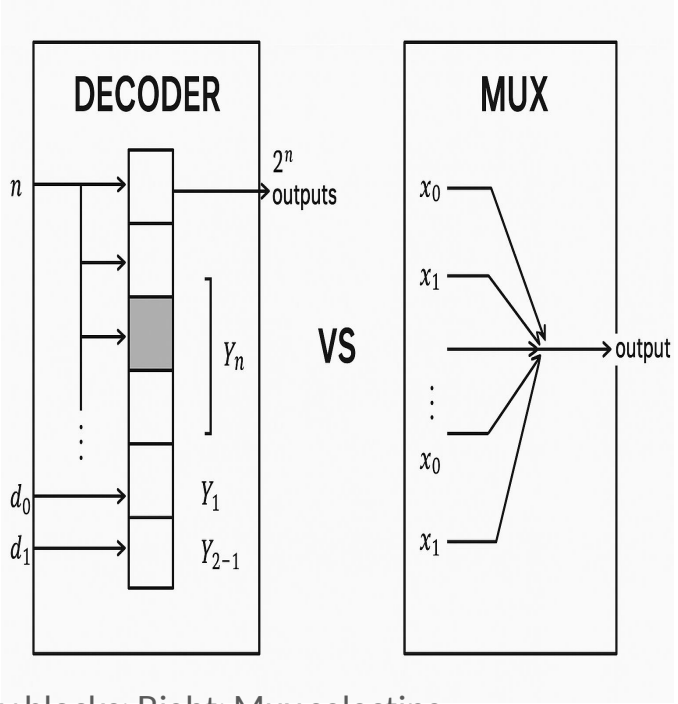
Key question: Decoder vs. Multiplexer in Memory – Key Differences

Decoder-Based:

- Converts address to one output for row selection.
- Scalable and efficient for large DFF memories.

Multiplexer-Based:

- Selects data from multiple inputs.
- Inefficient for DFF memories since each output needs a MUX with inputs equal to input size, causing high fan-in and complexity.



Summary:

- Decoder is preferred for large memories due to better scalability and simpler wiring.
- MUX is only usable for small memories or control logic with few inputs.

Prompt: Diagram – Decoder vs. Mux [Left: Decoder enabling one of many blocks; Right: Mux selecting one of several inputs for output]

ROM-Based Logic Design

Can memory behave like logic? Yes — with ROM!

- Example: **4-bit binary adder** implemented using a ROM(Read-Only Memory).
- ROM stores output values based on input address (inputs = address, output = data)

Advantages

- Very easy to design
- No Boolean minimization required
- Good for **prototypes**

Applications:

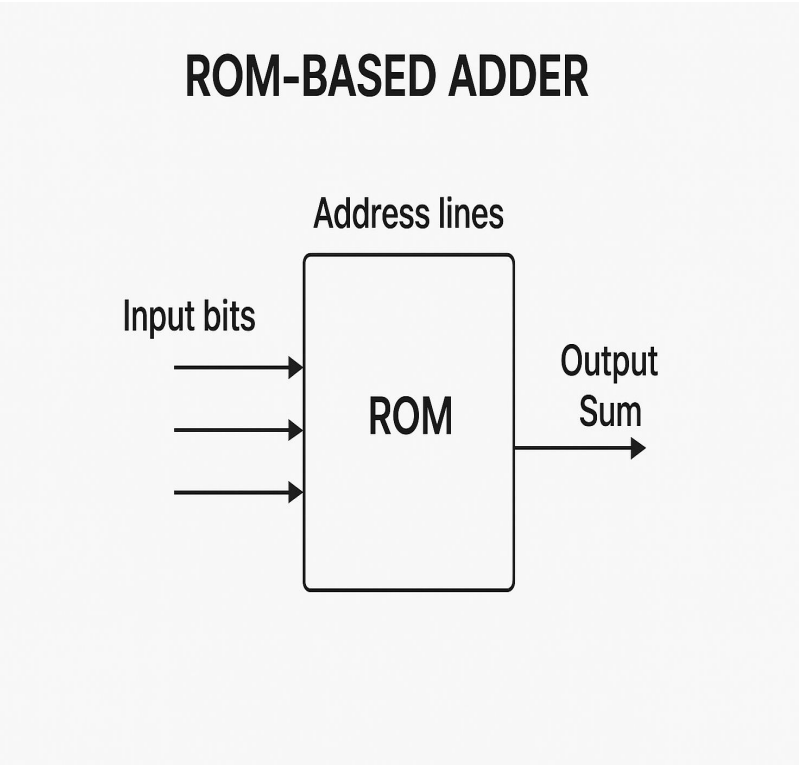
- Bootloaders (BIOS, firmware)
- Constant lookup tables (e.g., sine/cosine values)
- Microcode for instruction decoding

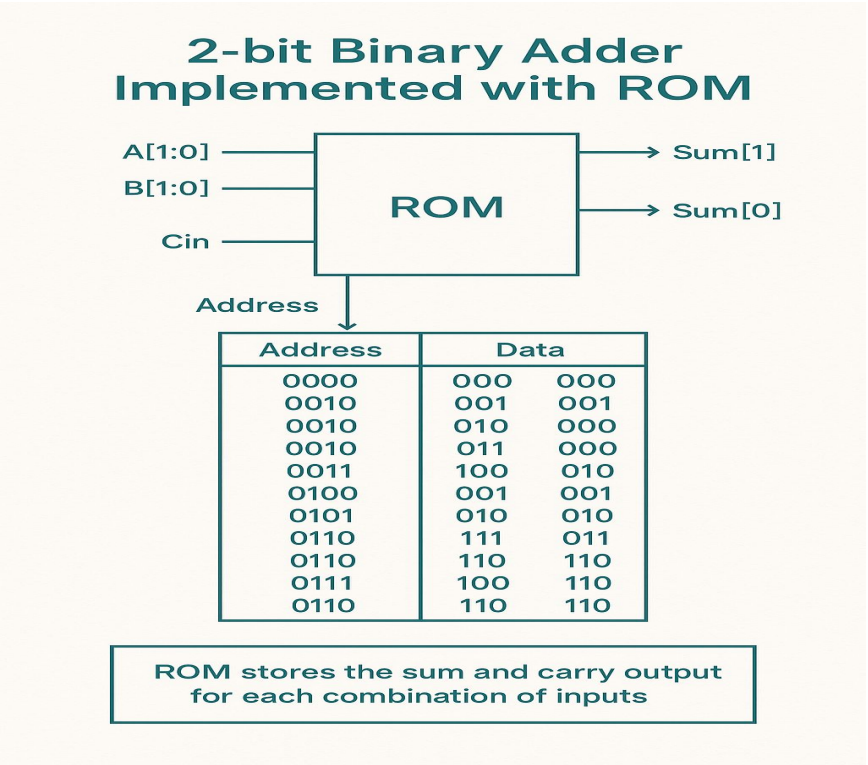
Drawbacks

- **Brute-force**: Exponential size with more inputs
- Not scalable
- Only for **special-purpose** or small-scale designs

Less Specified Prompt

More Specified Prompt





Prompt: Diagram – ROM-Based Adder [Diagram: Input bits → Address lines → ROM → Output Sum stored at each address]

Addressing Memory: 2+3 vs 3+2 Bits

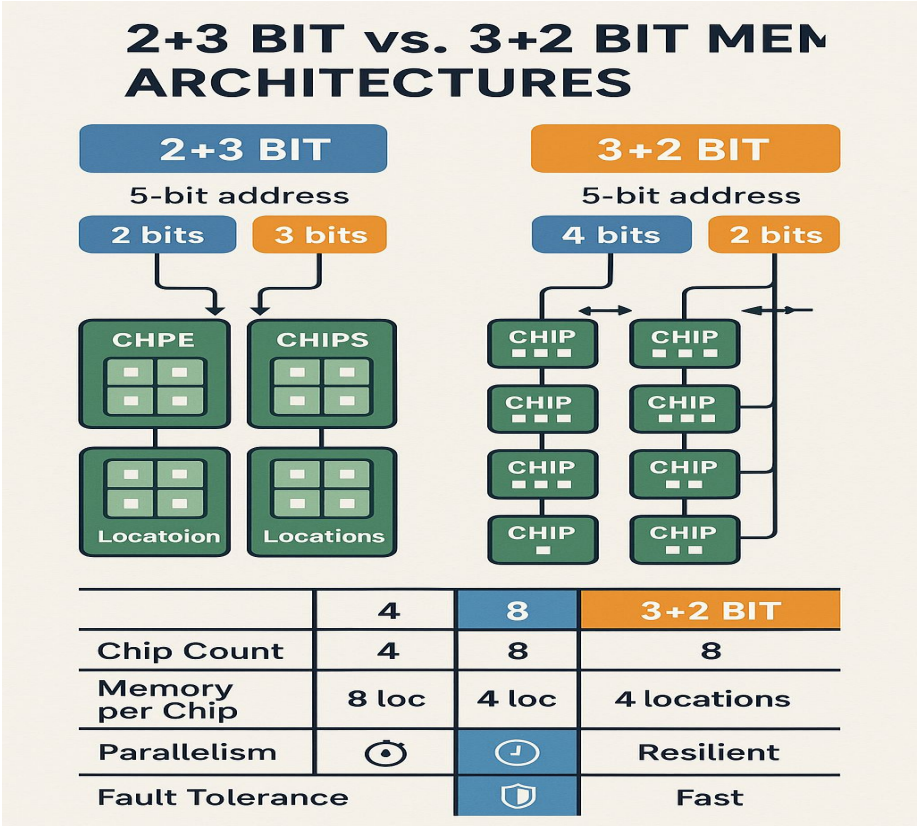
Assume we have 5 address bits at our disposal. How to utilize them? Two options:

- **Notation:**
  - *3+2:* 3-bit decoder selects block; 2-bit decoder selects word inside block
  - *2+3:* 2-bit decoder selects block; 3-bit decoder selects word inside block
- **Addressing Behavior:**
  - *3+2:* First selects a block, then fills all words in that block sequentially (e.g., from `00000` to `00001` fills within the same block first)
  - *2+3:* First selects a smaller block, then fills the corresponding word across multiple blocks simultaneously (fills same bit in all blocks before moving to next)
- **Fault Tolerance:**
  - *3+2:* More fault tolerant — faults in one block don't affect others since blocks are accessed independently
  - *2+3:* Less fault tolerant — simultaneous access across blocks means a fault in one block can impact multiple data lines
- **Parallelism:**
  - *2+3:* Better suited for parallel access — multiple blocks accessed simultaneously at the same word offset
  - *3+2:* Less parallelism — one block accessed at a time, filling words sequentially within that block

Prompt and the Output

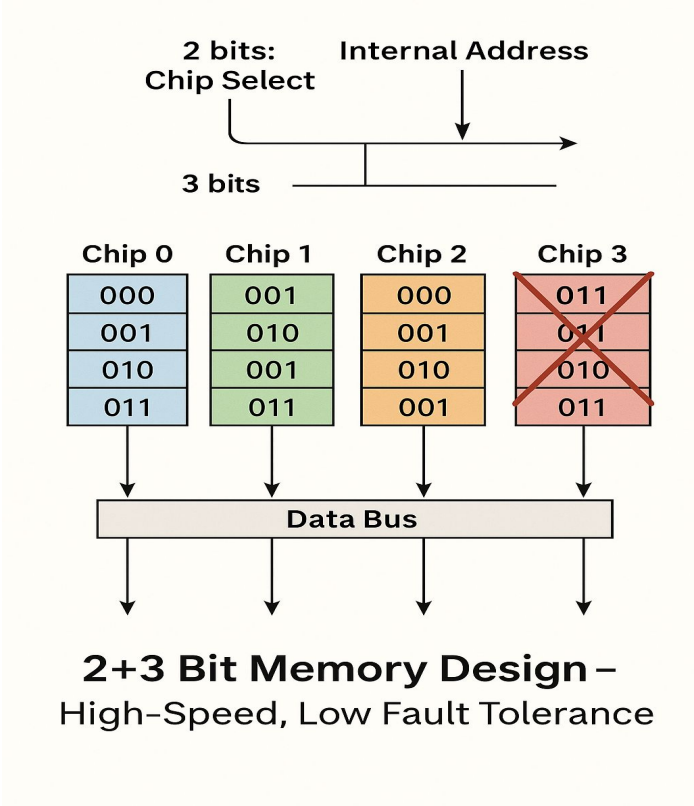
Create a side-by-side technical diagram comparing 2+3 and 3+2 bit memory addressing architectures:

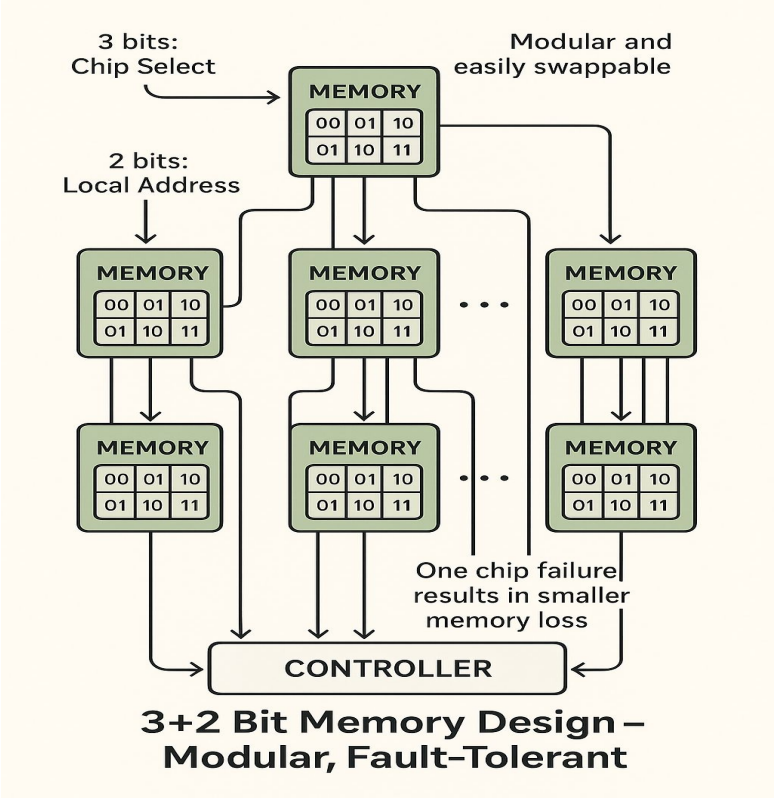
- **Left (2+3 bit):**
  - 4 memory blocks (chips), each with 8 locations
  - 2-bit decoder selects block; 3-bit decoder selects word inside block
  - Parallel access across blocks with shared data bus
  - Same word in all blocks accessed simultaneously
- **Right (3+2 bit):**
  - 8 memory blocks, each with 4 locations
  - 3-bit decoder selects block; 2-bit decoder selects word inside block
  - One block accessed at a time with individual data paths
  - Words accessed sequentially within a block before switching
- **Annotations:**
  - Address breakdown, chip count, locations per chip
  - Access style (parallel vs sequential)
  - Fault tolerance (2+3 less fault tolerant; 3+2 more modular and fault tolerant)
- **Style:**
  - Neutral colors, academic fonts, schematic engineering style
  - Labels like “Chip Select,” “Internal Address,” “Total Address = 32”
- **Title:**
  - *Comparison of 2+3 vs. 3+2 Bit Memory Addressing Architectures*



Less Specified Prompt

More Specified Prompt





Prompt: Diagram – Layout Comparison [Side-by-side visual of how address bits map to chips and local positions in 2+3 and 3+2 schemes]

The Memory Wall Problem

Problem: CPU speeds have grown faster than memory access speeds

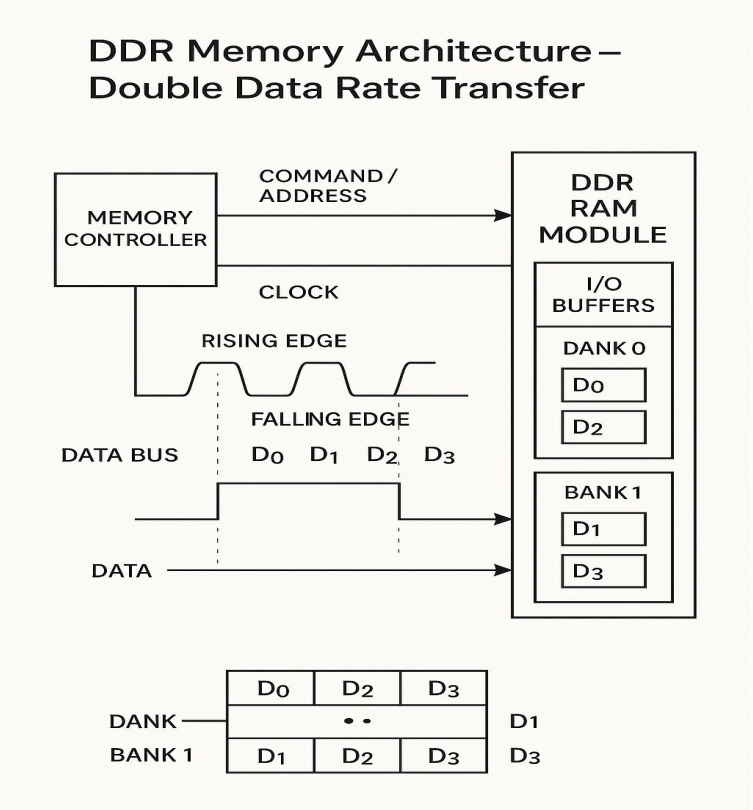
- This creates a **bottleneck** in modern computing

Solution?

- **Architectural solutions**:
  - **DDR RAM**: double the throughput using rising & falling clock edges
  - **Locality-aware design**: store even/odd data on different sides of the memory card
  - **Cache systems** and **parallelism**
- **Software adaptation**

How DDR-RAM Helps?

- DDR = Double Data Rate
- Transfers data on both clock edges (rising and falling)
- Doubles data rate without increasing clock speed
- Still requires efficient software to fully benefit



Prompt: Diagram – DDR RAM [Clock waveform with two transfers per cycle shown, colored for rising/falling edges. Memory cells mapped accordingly.]

Custom Architectures & Optimization

The more you know, the better you design.

Specialized Needs = Specialized Hardware

- **Examples**:
  - Up-counter → custom logic faster than general adder
  - **GPU** optimized for image/data parallelism

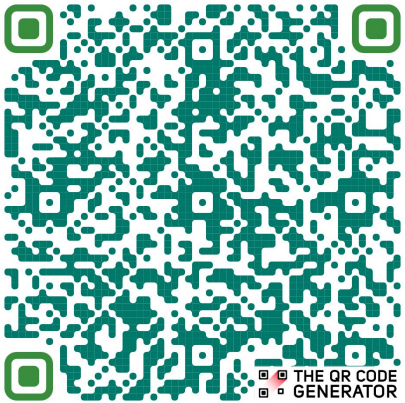
    These systems are:

    - Faster for their job
    - More efficient in energy and area
    - But not flexible

Takeaway: Custom architectures trade **scalability** for **performance and area efficiency**

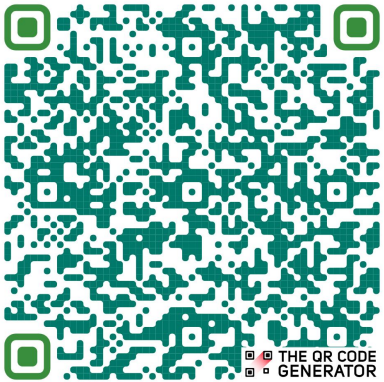
Let's recap.

- **Q1: What does the address bus do?**
- **Q2: In Little Endian format, where is the Least Significant Byte stored?**
- **Q3: Which layout allows parallel data transfer ?**
- **Q4: What makes DDR RAM faster than SDRAM?**



Any questions?

- Let’s have a brief discussion before wrapping up.
- We’re open to any questions about architecture, memory, or the design choices we made.



Sources

Sources Used in This Work

1. **Primary Lecture Material:**
   - Lecture notes by Prof. Montuschi.
2. **Academic Textbooks:**
   - John L. Hennessy & David A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th Edition, Morgan Kaufmann, 2019.
   - Bruce Jacob, Spencer Ng, David Wang, *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann, 2007.
3. **Reference Articles and Illustrations (paraphrased and reinterpreted):**
   - Wikipedia articles on:
     - **Endianness** (en.wikipedia.org/wiki/Endianness)
     - **Bus (computing)** (en.wikipedia.org/wiki/Bus_(computing))
     - **D flip-flop**, **ROM**, **DDR, SDRAM**, and others
   - Visual and conceptual inspiration from:
     - RealPars (realpars.com)
     - GeeksforGeeks and TutorialsPoint (used only for concept inspiration)
4. **AI Tools Used:**
   - ChatGPT-4, Deepseek: Text rewriting, simplification, and concept clarification
   - DALL·E (OpenAI): Custom schematic illustrations based on our descriptions
   - Diagrams edited or labeled by our team where needed for clarity

License Information

- This presentation is licensed under **CC BY-NC-SA 4.0**
  (You may share and adapt for non-commercial purposes, with attribution and share-alike)
- AI-generated content used where appropriate, reviewed by team for accuracy
- For more information visit: https://creativecommons.org/licenses/by/4.0/