Memories

Digital Design and Computer Architecture
Mohammad Sadrosadati
Frank K. Gürkaynak

http://safari.ethz.ch/ddca

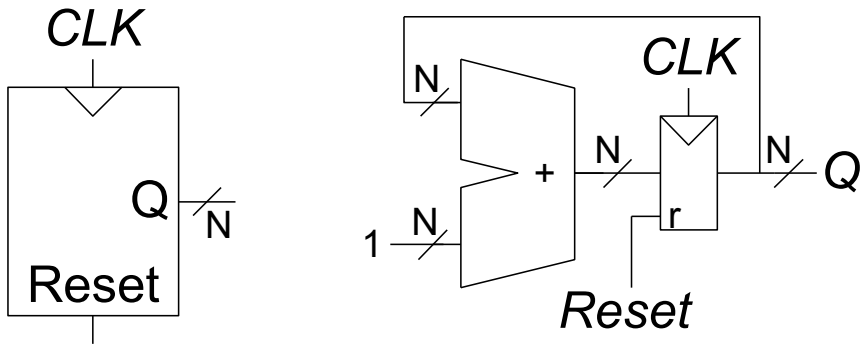
What will we learn today?

- **One more common sequential building block**
  - Counters
- **How can we store data?**
- **Array organization of memories**

Counters

- **Increments on each clock edge.**
  - Used to cycle through numbers. For example,
  - `000, 001, 010, 011, 100, 101, 110, 111, 000, 001…`
- **Example uses:**
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

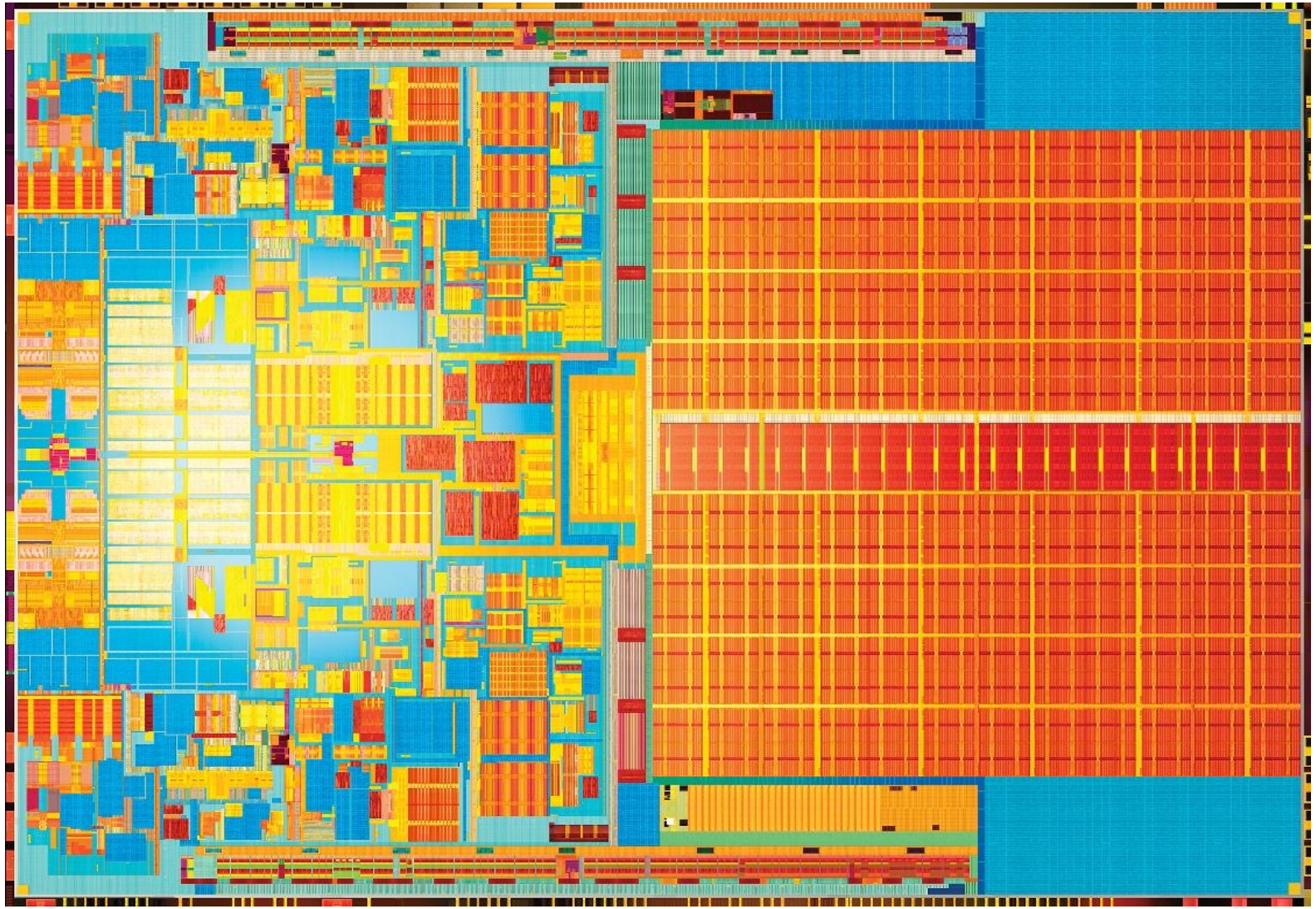
Symbol **Implementation**



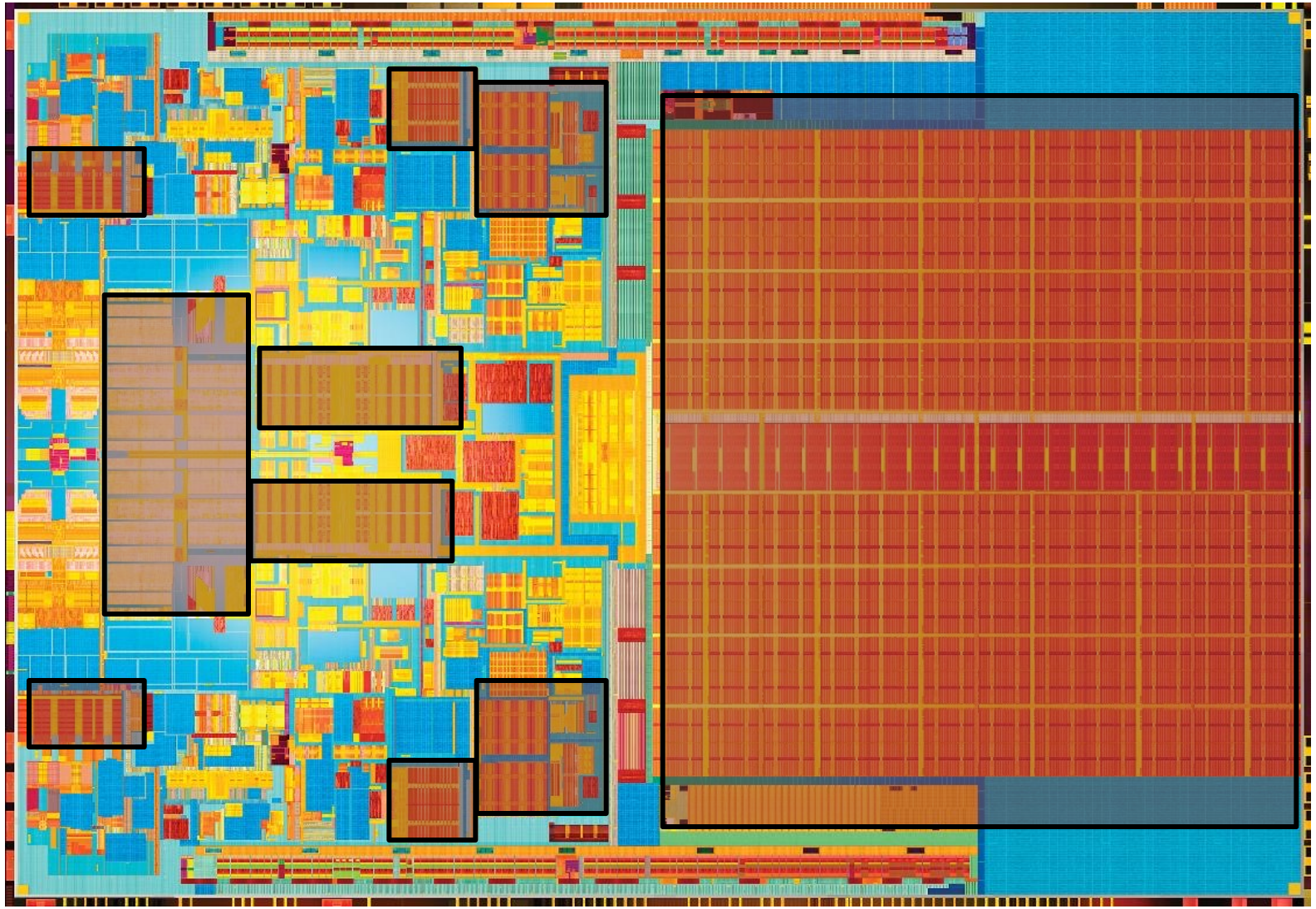
Motivation: Memory Elements

- **Memories are large blocks**
  - A significant portion of a modern circuit is memory.
- **Memories are practical tools for system design**
  - Programmability, reconfigurability all require memory
- **Allows you to store data and work on stored data**
  - Not all algorithms are designed to process data as it comes, some require data to be stored.
  - Data type determines required storage
    - SMS: 160 bytes
    - 1 second normal audio: 64 kbytes
    - 1 HD picture: 7.32 Mbytes

Die photograph of an Intel processor in 45nm



Larger Memory Blocks



How can we store data

- **Flip-Flops (or Latches)**
  - Very fast, parallel access
  - Expensive (one bit costs 20+ transistors)

How can we store data

- **Flip-Flops (or Latches)**
  - Very fast, parallel access
  - Expensive (one bit costs 20+ transistors)
- **Static RAM (we will describe them in a moment)**
  - Relatively fast, only one data word at a time
  - Less expensive (one bit costs 6 transistors)

How can we store data

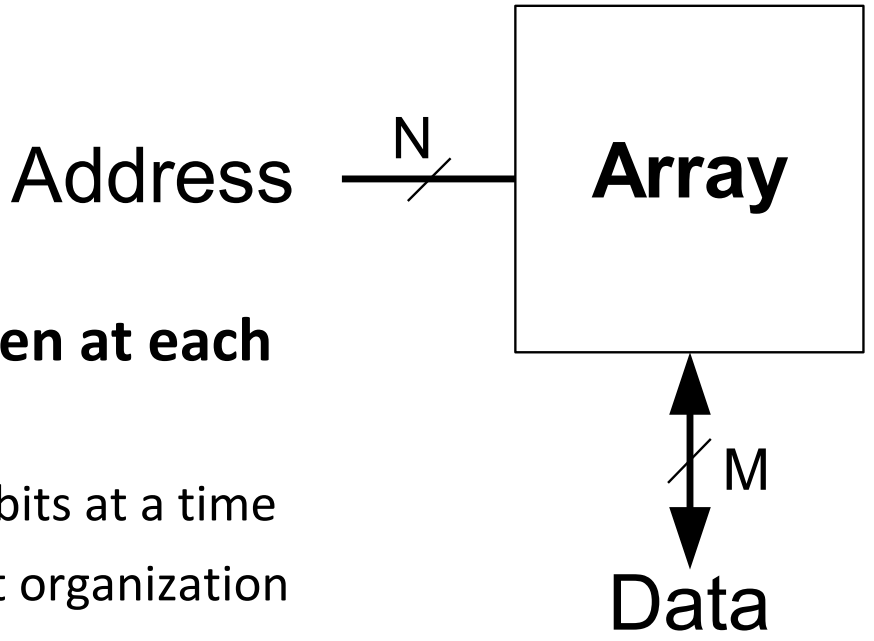
- **Flip-Flops (or Latches)**
  - Very fast, parallel access
  - Expensive (one bit costs 20+ transistors)
- **Static RAM (we will describe them in a moment)**
  - Relatively fast, only one data word at a time
  - Less expensive (one bit costs 6 transistors)
- **Dynamic RAM (we will describe them a bit later)**
  - Slower, reading destroys content (refresh), one data word at a time, needs special process
  - Cheaper (one bit is only a transistor)

How can we store data

- **Flip-Flops (or Latches)**
  - Very fast, parallel access
  - Expensive (one bit costs 20+ transistors)
- **Static RAM (we will describe them in a moment)**
  - Relatively fast, only one data word at a time
  - Less expensive (one bit costs 6 transistors)
- **Dynamic RAM (we will describe them a bit later)**
  - Slower, reading destroys content (refresh), one data word at a time, needs special process
  - Cheaper (one bit is only a transistor)
- **Other storage technology (hard disk, flash)**
  - Much slower, access takes a long time, non-volatile
  - Per bit cost is lower (no transistors directly involved)

Array Organization of Memories

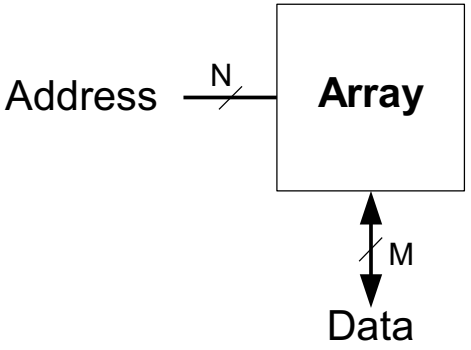
- **Efficiently store large amounts of data**
  - Consists of a memory array (stores data)
  - Address selection logic (selects one row of the array)
  - Readout circuitry (reads data out)

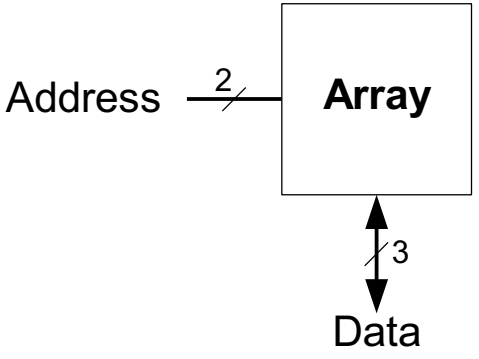


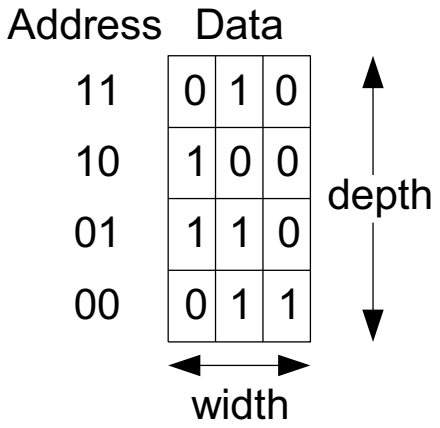
- **An M-bit value can be read or written at each unique N-bit address**
  - All values can be accessed, but only M-bits at a time
  - Access restriction allows more compact organization

Memory Arrays

- **Two-dimensional array of bit cells**
  - Each bit cell stores one bit
- **An array with N address bits and M data bits:**
  - 2^N rows and M columns
  - Depth: number of rows (number of words)
  - Width: number of columns (size of word)
  - Array size: depth × width = 2^N × M

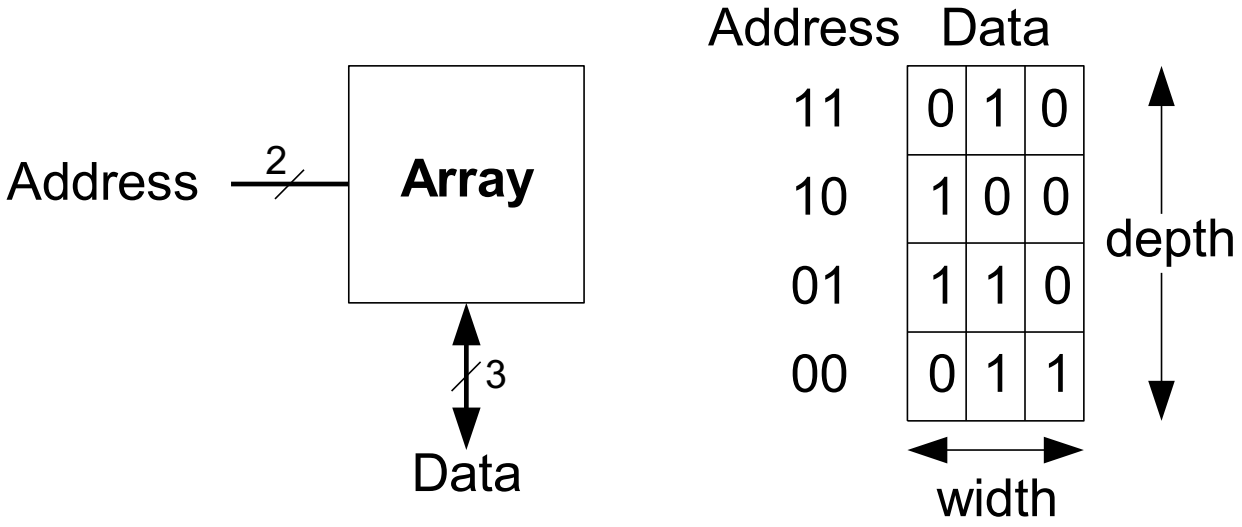




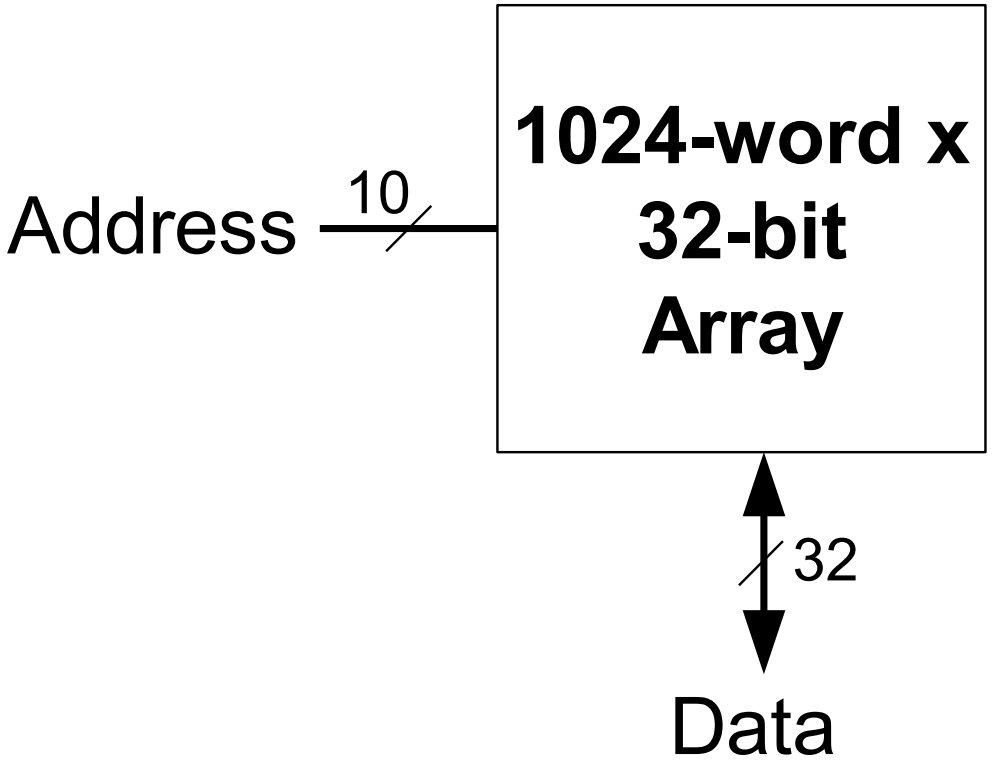


Memory Array: Example

- $2^2 \times 3$-bit array
- Number of words: 4
- Word size: 3-bits
- For example, the 3-bit word stored at address 10 is 100



Memory Arrays

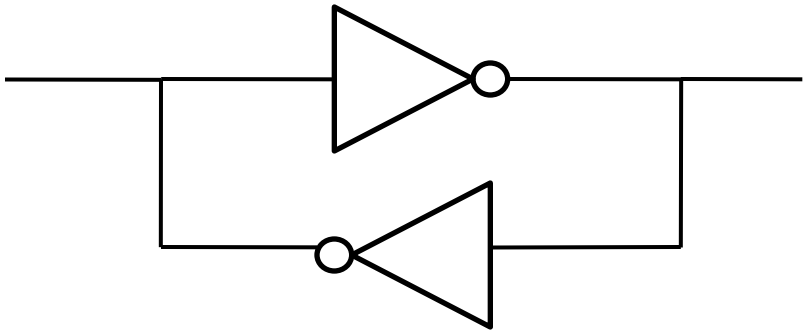


Types of Memories

- **Volatile memories (loses data when power is off)**
  - Static Random Access Memory (SRAM)
  - Dynamic Random Access Memory (DRAM)
- **Non-volatile memories (keeps data even without power)**
  - Read Only Memory (ROM)
  - Various forms of flash memory (i.e. EEPROM)

Static Random Access Memory

- **Volatile:** Stores data by cross coupled inverters, once data is stored the inverters keep the value (therefore static)



- Historically called Random Access Memory, because data can be accessed in any order (unlike magnetic tapes which allowed only serial access)

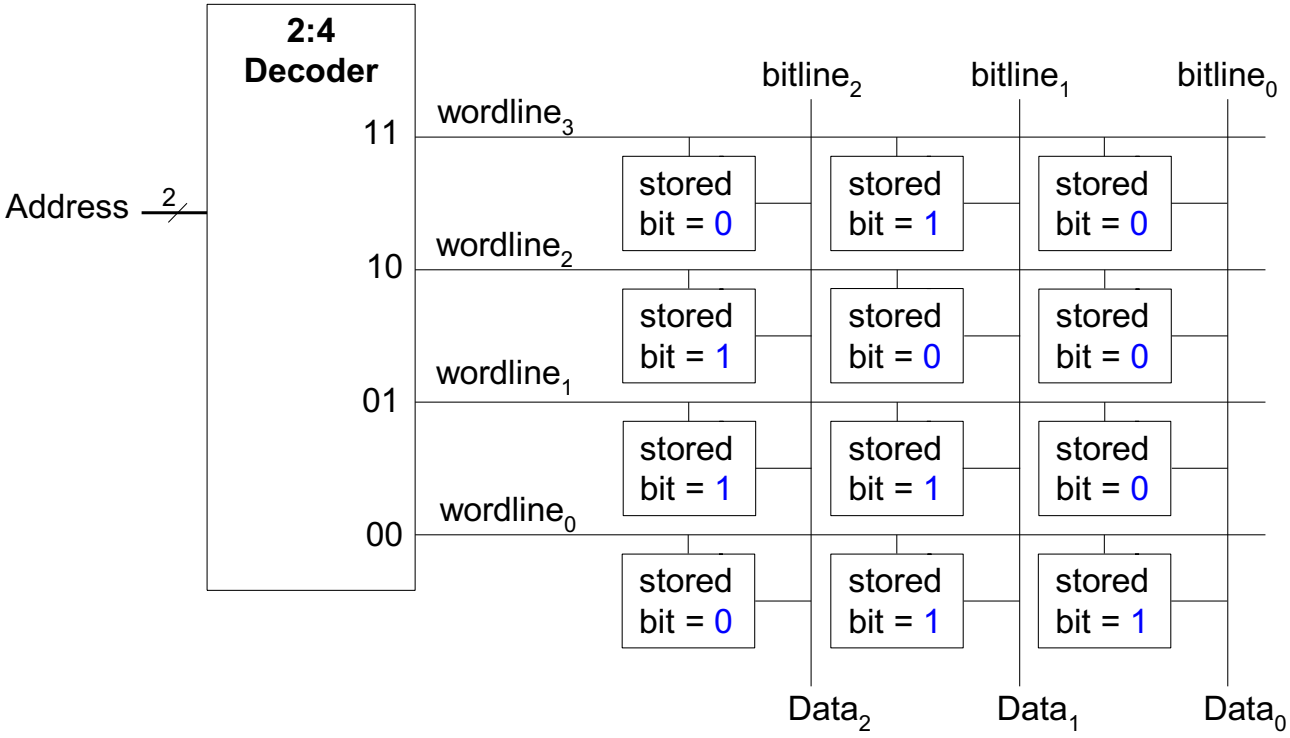
Dynamic Random Access Memory

- **Volatile: Stores data by charging a (small) capacitor**

- **Problem is that the charge on the capacitor will slowly discharge (memory will forget the value) with time.**
- **It is called Dynamic, because we have to refresh the contents before memory forgets what it stored.**
- **The larger the capacitor, the longer it takes to forget**
  - This costs area, ingenious methods are used to increase capacitance

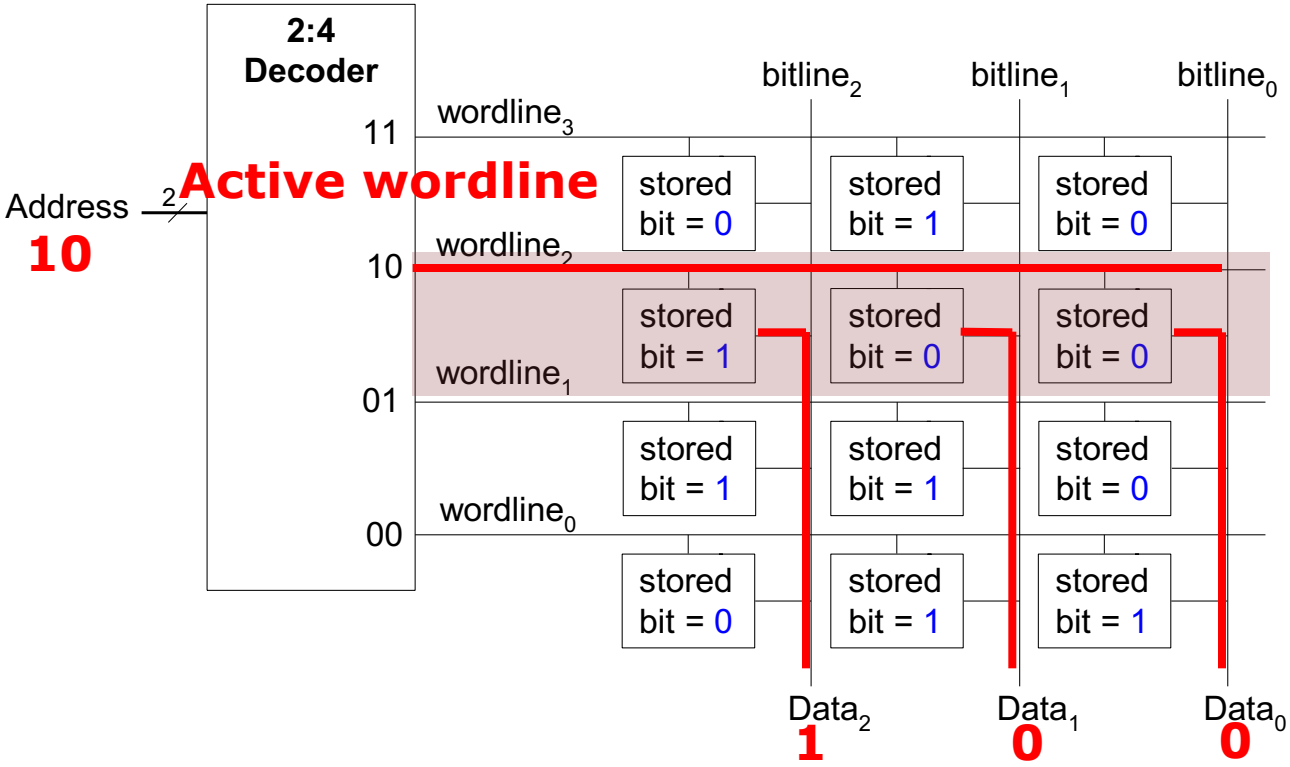
Memory Array Organization

- Storage nodes in one column connected to one bitline
- Address decoder activates only ONE wordline
- Content of one line of storage available at output



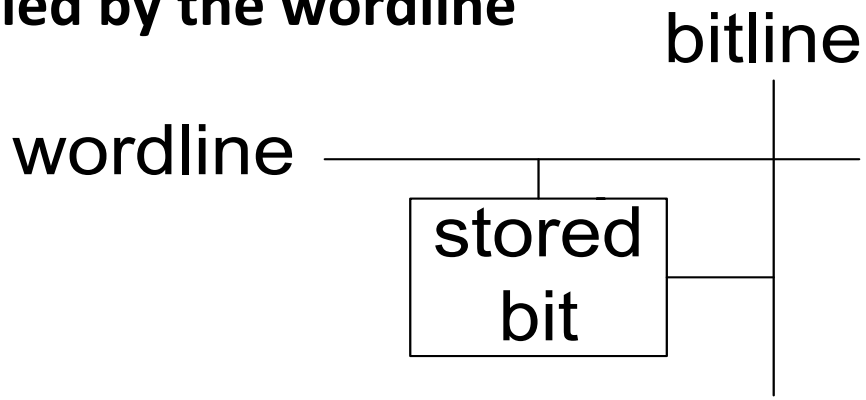
Memory Array Organization

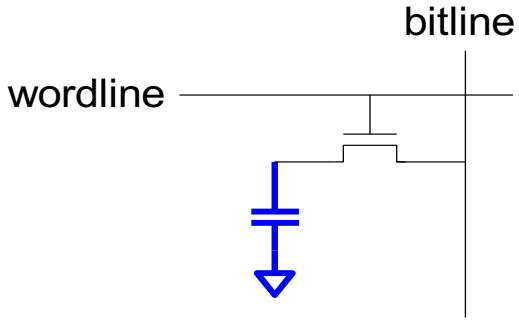
- Storage nodes in one column connected to one bitline
- Address decoder activates only ONE wordline
- Content of one line of storage available at output



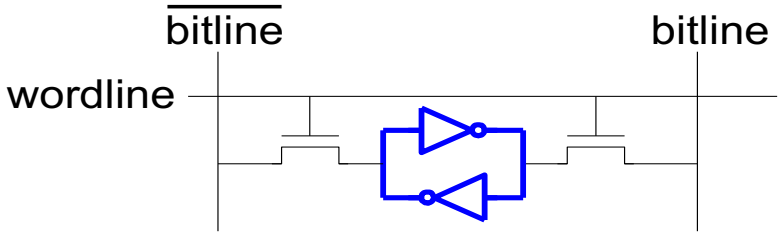
How is Access Controlled ?

- **Access transistors configured as switches connect the bit storage to the bitline.**
- **Access controlled by the wordline**





DRAM

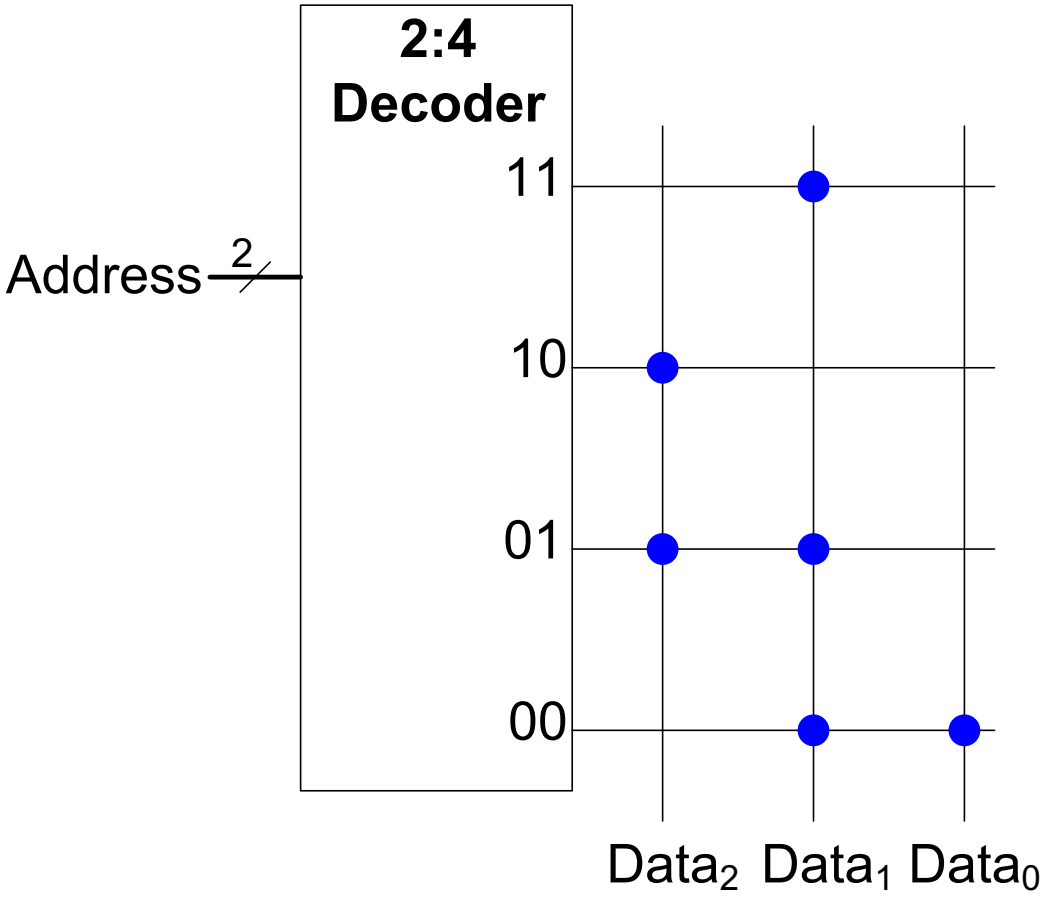


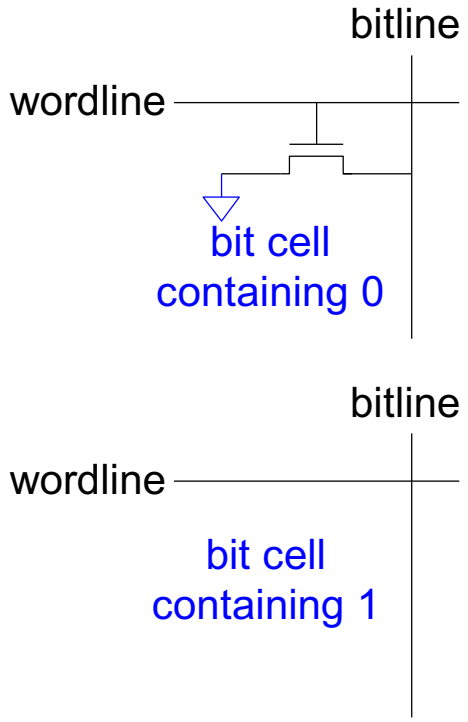
SRAM

Read Only Memories

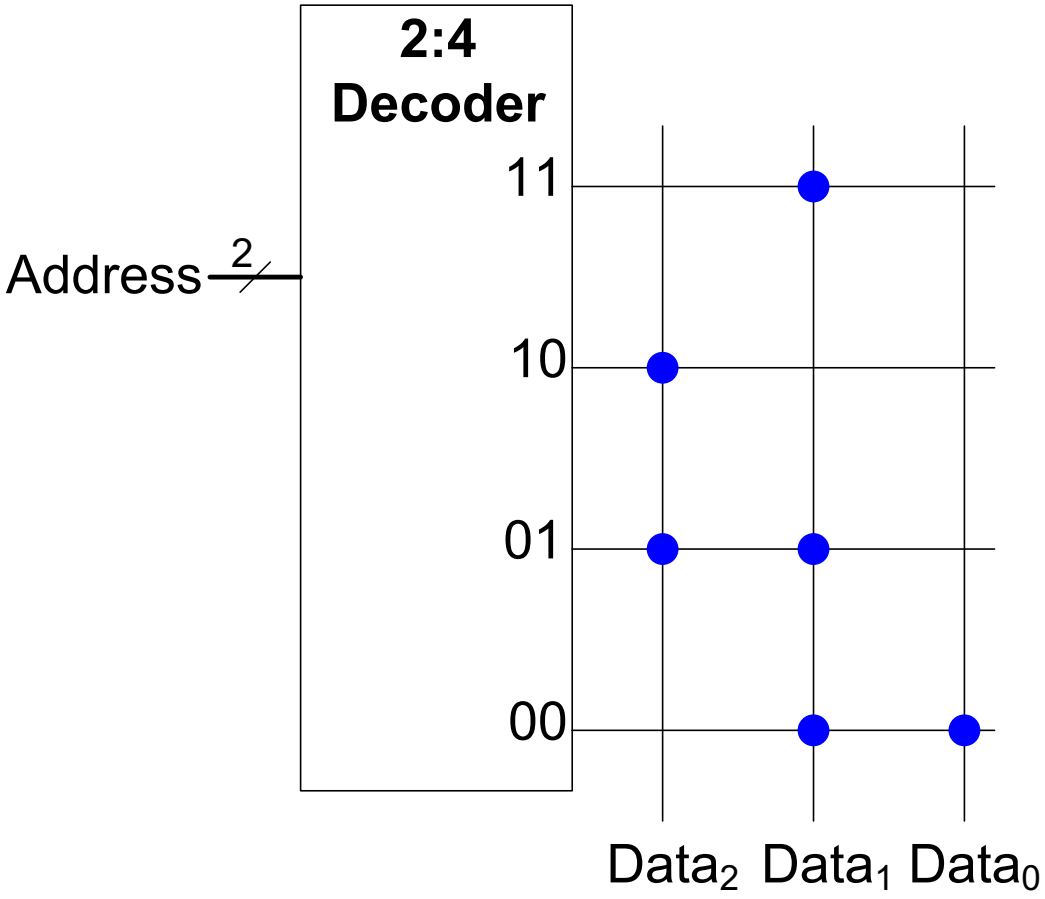
- **Non Volatile: Read Only Memories (ROM) can be made much denser**
  - No need to change the content (no storage transistors)
  - Denser array
- **Used for keeping content that will not change**
  - Program of an embedded system
  - Configuration data
  - Look up tables
- **Re-writable (flash) memories are commonly used**
  - These are actually programmable, but writing is very slow
  - From an application point of view identical to ROMs

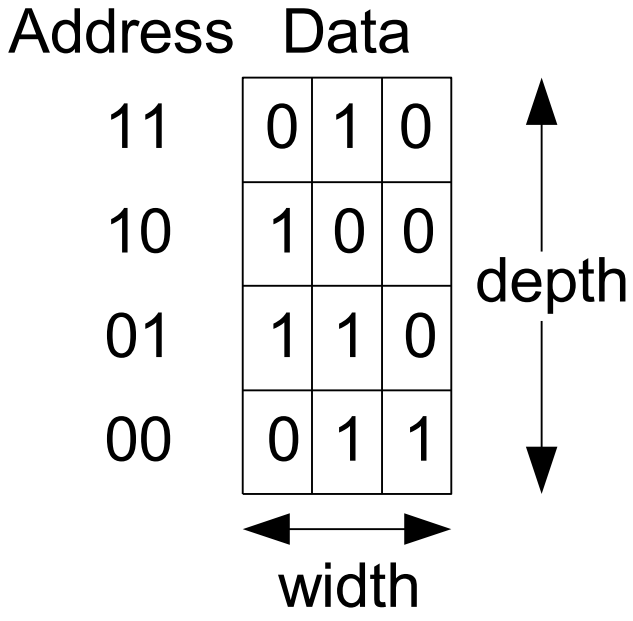
ROMs: Dot Notation



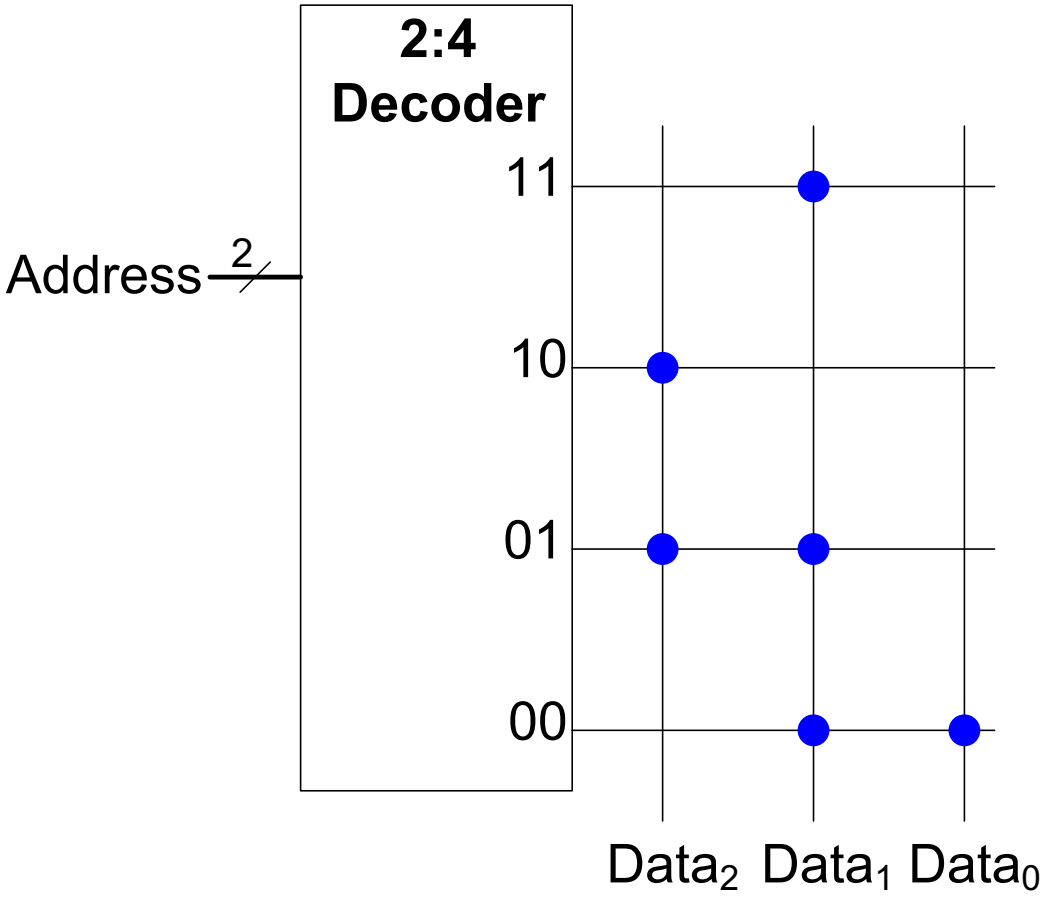


ROM Storage


2:4
Decoder
Address
2
11
10
01
00
Data2 Data1 Data0


Address Data
11 0 1 0
10 1 0 0
01 1 1 0
00 0 1 1
depth
width

ROM Logic



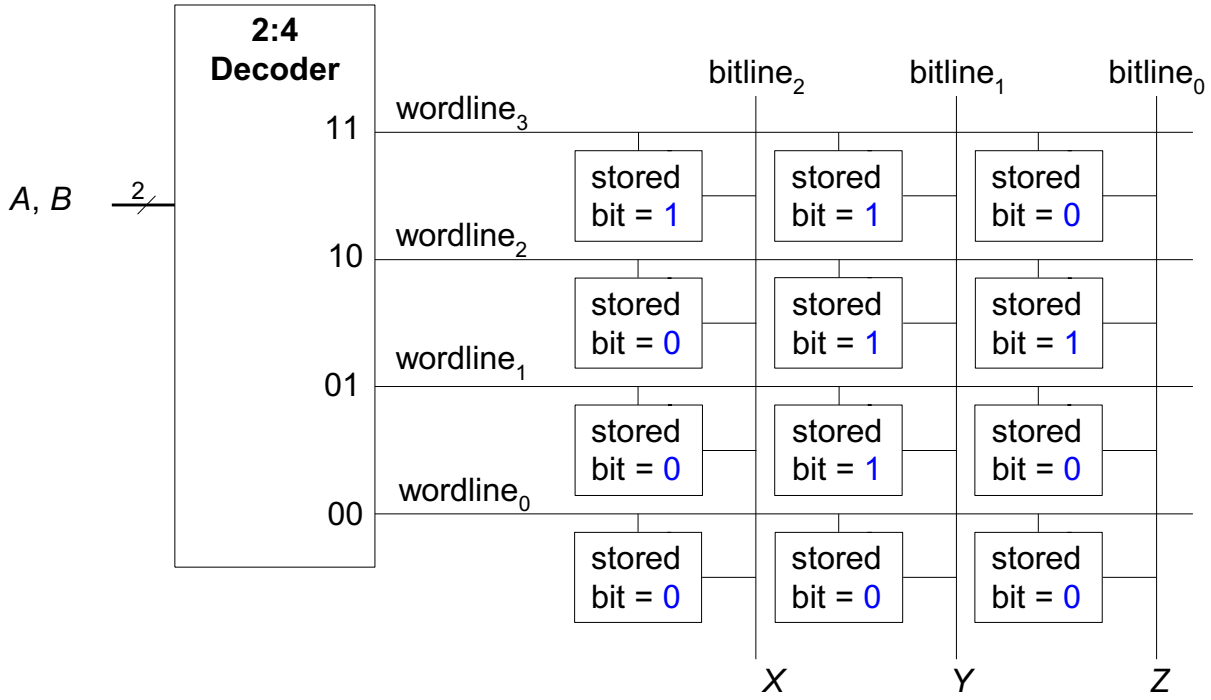
$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A_1} + A_0$$

$$Data_0 = \overline{A_1}\,\overline{A_0}$$

Logic with Memory Arrays

- **Implement the following logic functions using a 22 × 3-bit memory array:**
  - X = AB
  - Y = A + B
  - Z = A $\overline{B}$

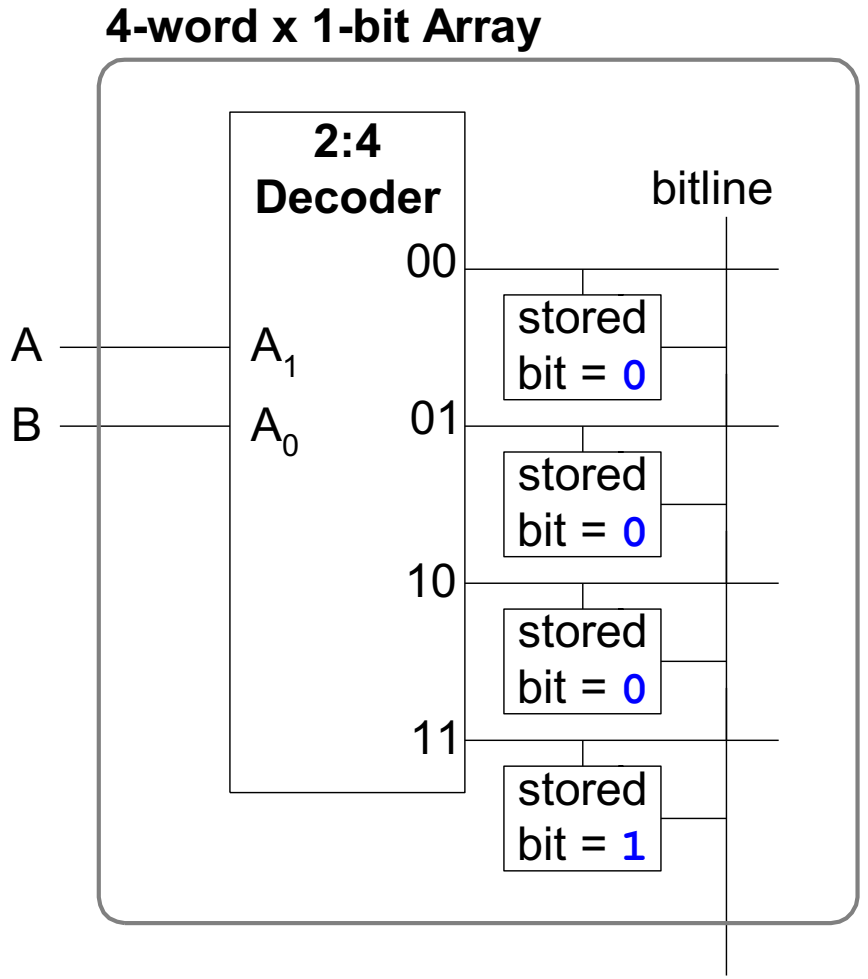


Logic with Memory Arrays

- **Called lookup tables (LUTs): look up output at each input combination (address)**

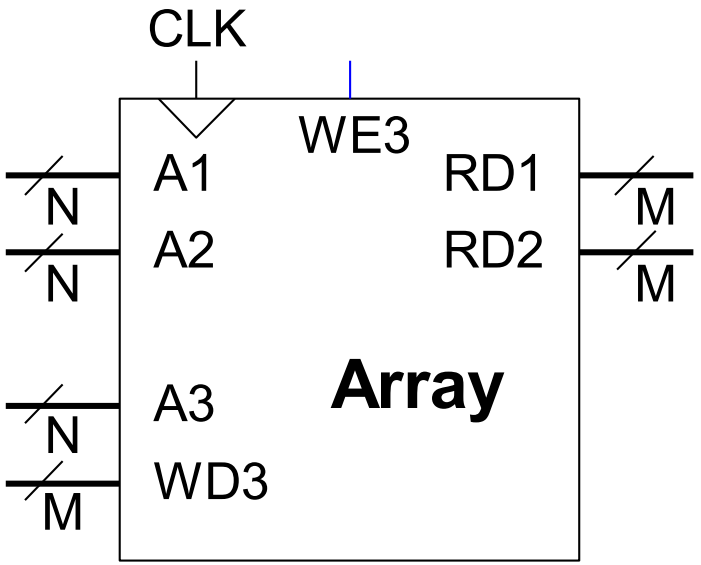
Truth Table

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Multi-ported Memories

- **Port: address/data pair**
- **3-ported memory:**
  - 2 read ports (A_1/RD_1, A_2/RD_2)
  - 1 write port (A_3/WD_3, WE_3 enables writing)
- **Small multi-ported memories are called register files**



Memory Arrays in Verilog

```
// 256 x 3 memory module with one read/write port
module dmem( input           clk, // clock
             input           we,  // write enable
             input  [7:0]    a    // 8-bit address
             input  [2:0]    wd,  // 3-bit write data
             output [2:0]    rd); // 3-bit read data

  reg  [2:0]    RAM[255:0];       // Memory array, holds
                                  // 256 entries each 3 bits wide
  assign rd = RAM[a];             // Read access

  always @(posedge clk)           // with rising clock
      if (we)                     // if write enable
          RAM[a] <= wd;           // write data is stored in array
endmodule
```

What have we learned?

- **Different ways of storing data**
  - Registers
  - Static Memory
  - Dynamic Memory
- **Array organization**
  - Compact form
  - One row active at a time