

Canvas: Upload
Videos

ENGR 210 / CSCI B441
“Digital Design”

Website: PDF Slides

Memory

Missing USB-cable

Andrew Lukefahr

Announcements

- P3 is due ~~Friday~~ Today
 - Adds “demo” requirement. Will need real hardware.
 - Demo: create video and upload to Canvas
- Upload Box Number to Canvas
 - Part of P3’s grade

Post old
Exam?

Exam

→ Wednesday: Review

After Exam, Prof.
Hinnebaash

Monday: Exam

→ comprehensive exam

→ Verilog writing

post Q1 on Canvas → Friday
have out Monday turn in
on Canvas

post Q2 → End on Canvas @
9:25 ET, due back @

11am ET

→ basic syntax issues ⇒ don't care
→ always-ff, always-comb
⇒ inferred latches
⇒ missing defaults,
→ testbench

Always specify
defaults for
always_comb!

Topics

- ROM vs RAM
- ROM/RAM Structure

ROM vs RAM

- ROM → Read-Only Memory
 - typically also "Random-Access"
- RAM → Random-Access Memory

⇒ Rom is a RAM that is Read-Only

RAM is R/W version of ROM.

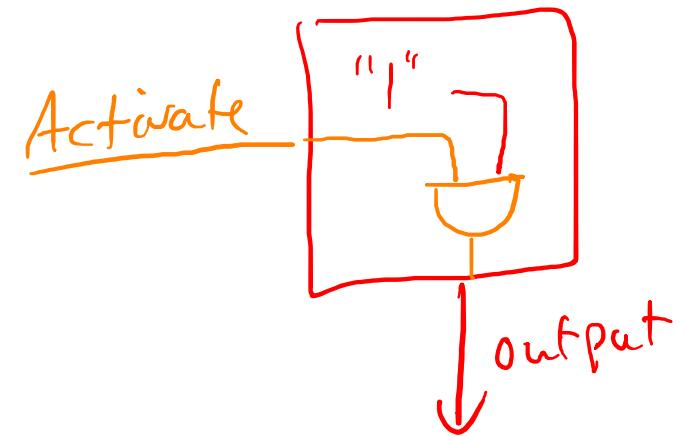
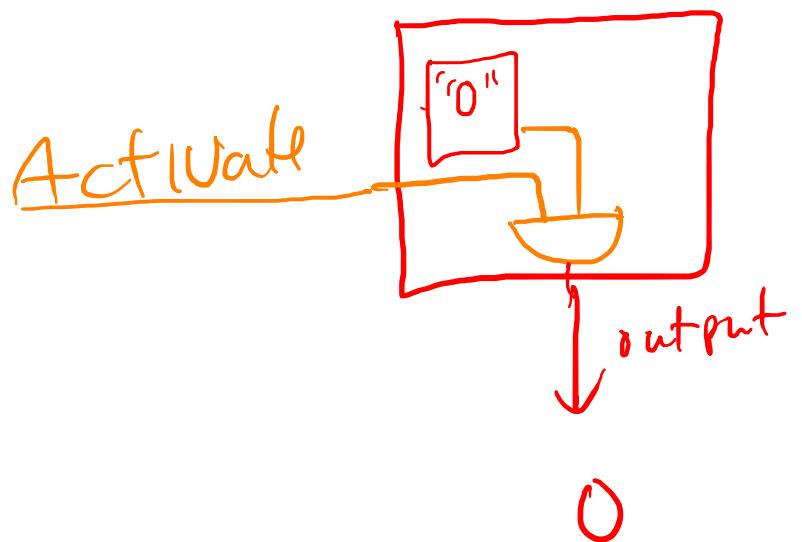
inputs: Address (Addr)

output: Value (data)

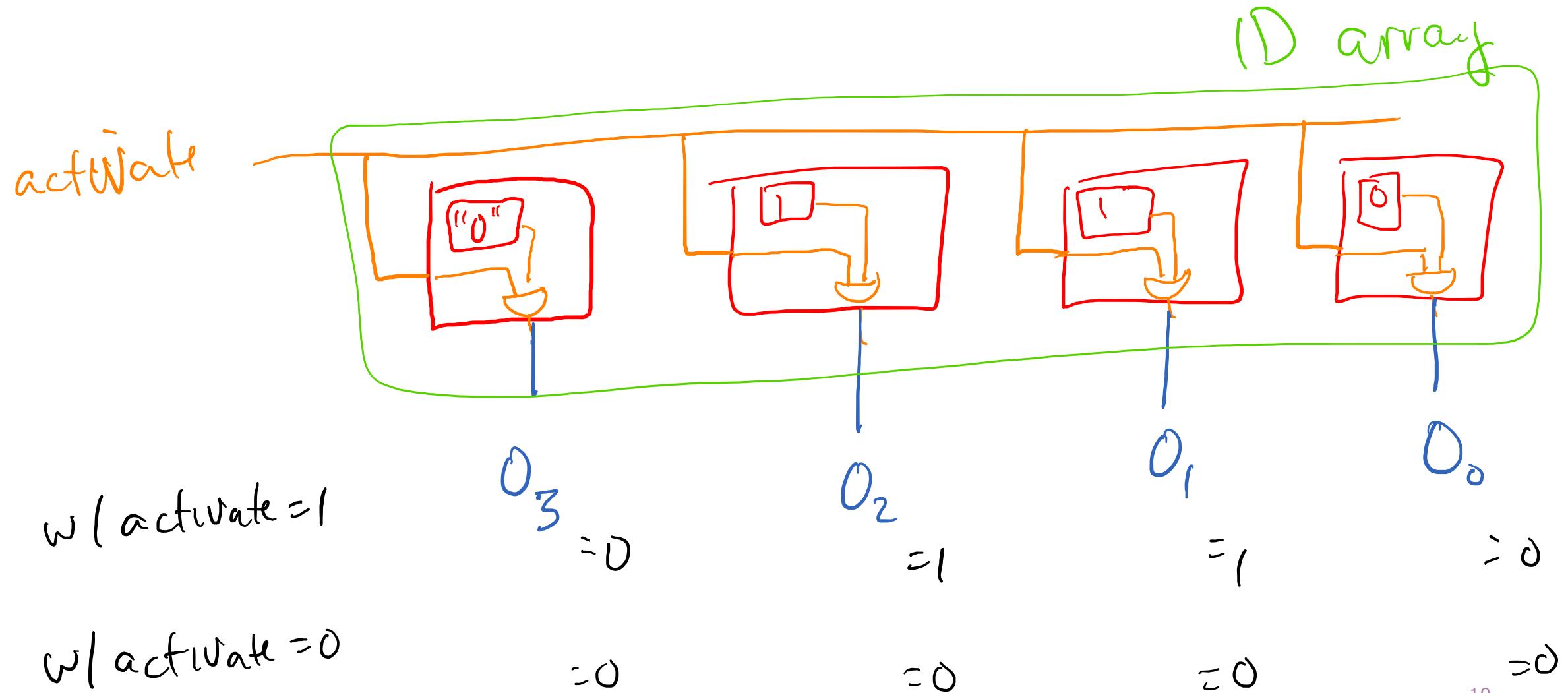
ROM vs RAM

- ROM – Read-Only Memory
 - Input: address
 - Output: fixed value
- RAM – Random-Access Memory
 - Read/Write version of a ROM

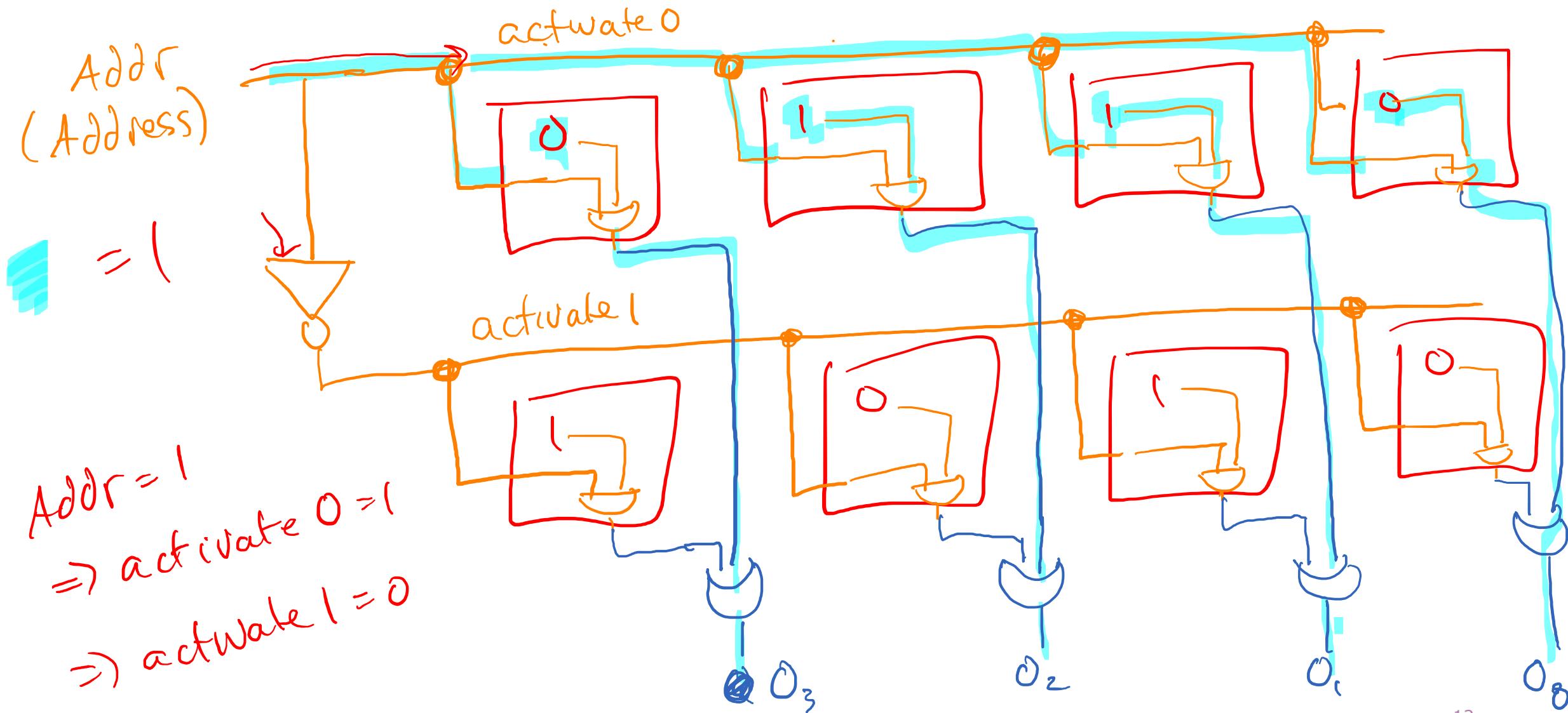
Rom Cell



Multiple ROM Cells

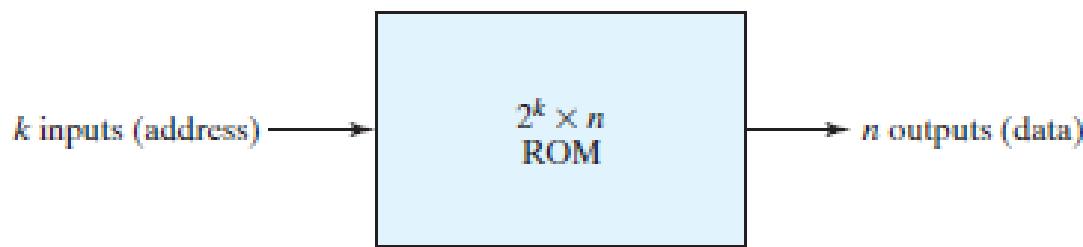


Array of ROM Cells



The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern.

Once the pattern is established, it stays within the unit even when power is turned off and on again.

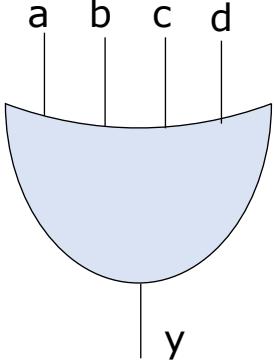


In addition to address inputs and data outputs, integrated circuit ROM chips have:

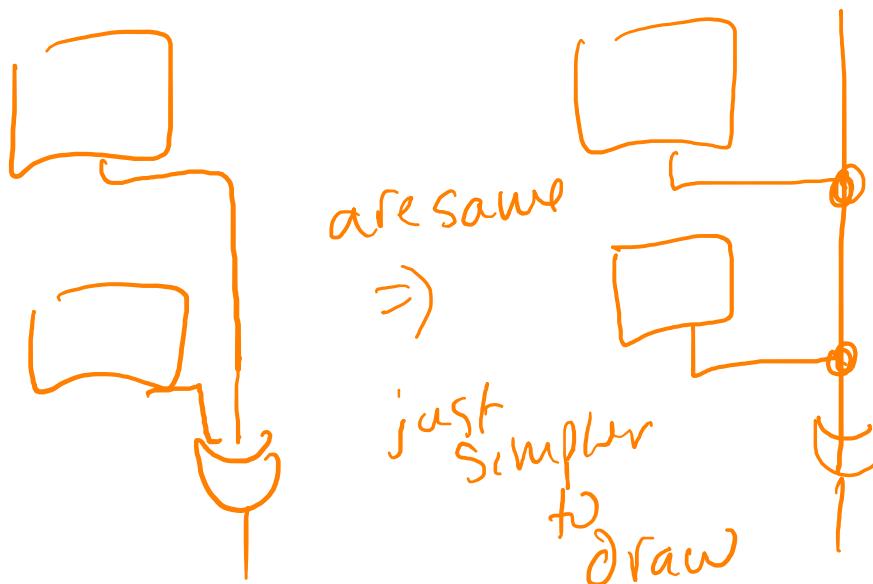
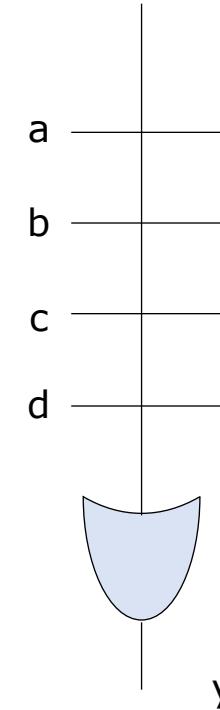
- One or more enable inputs
- Sometimes come with three-state outputs to facilitate the construction of large arrays of ROM.

Notation for the gates

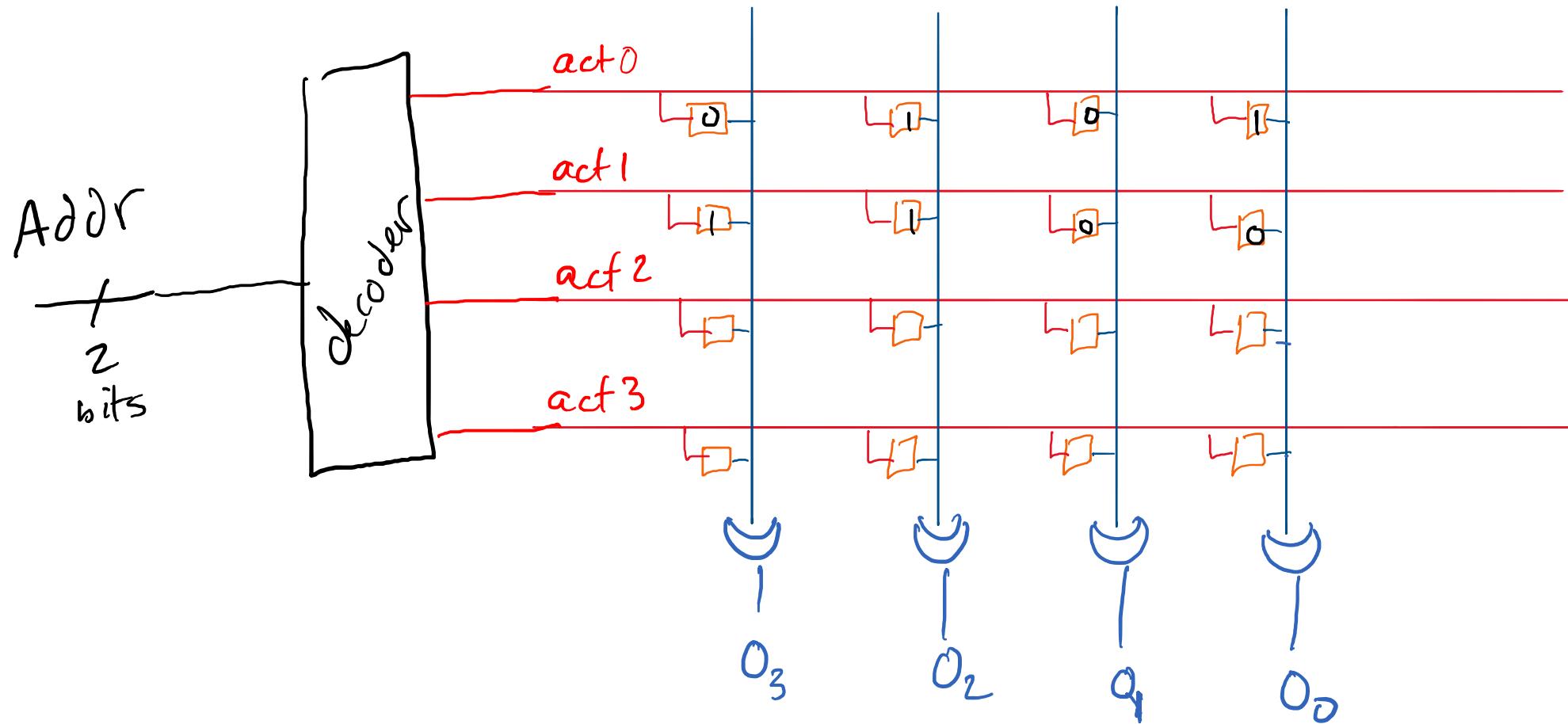
Instead of:



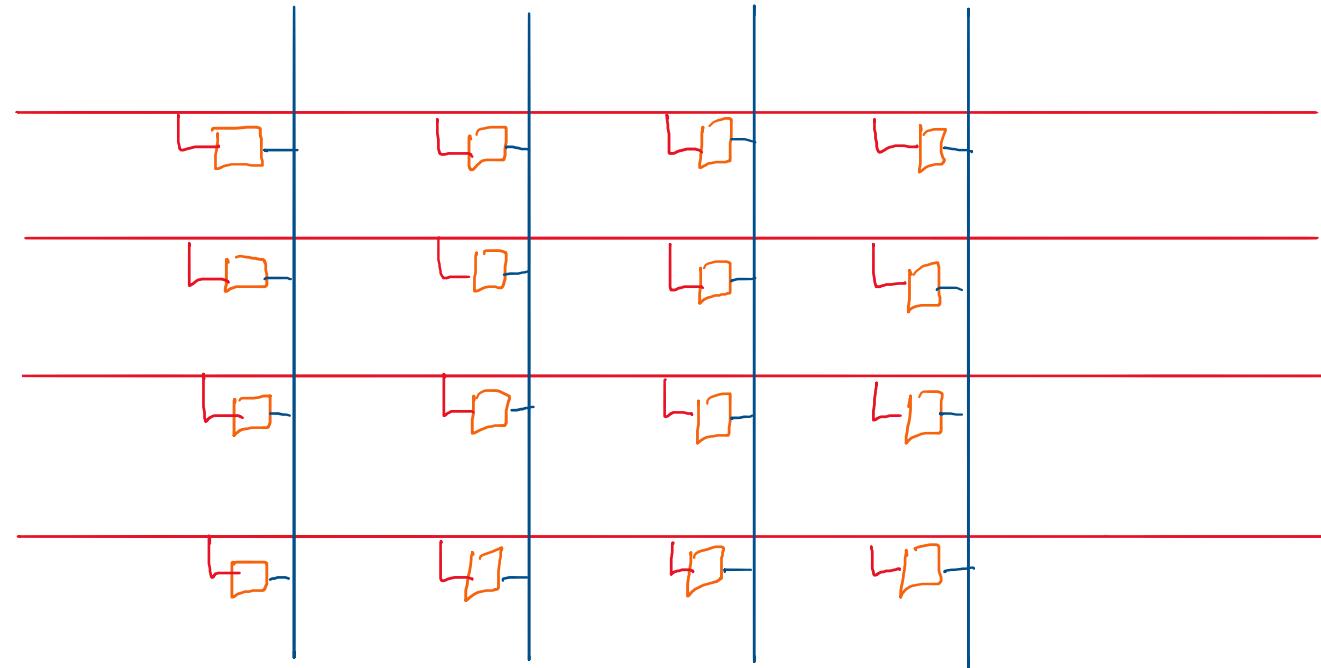
We often use
notation:



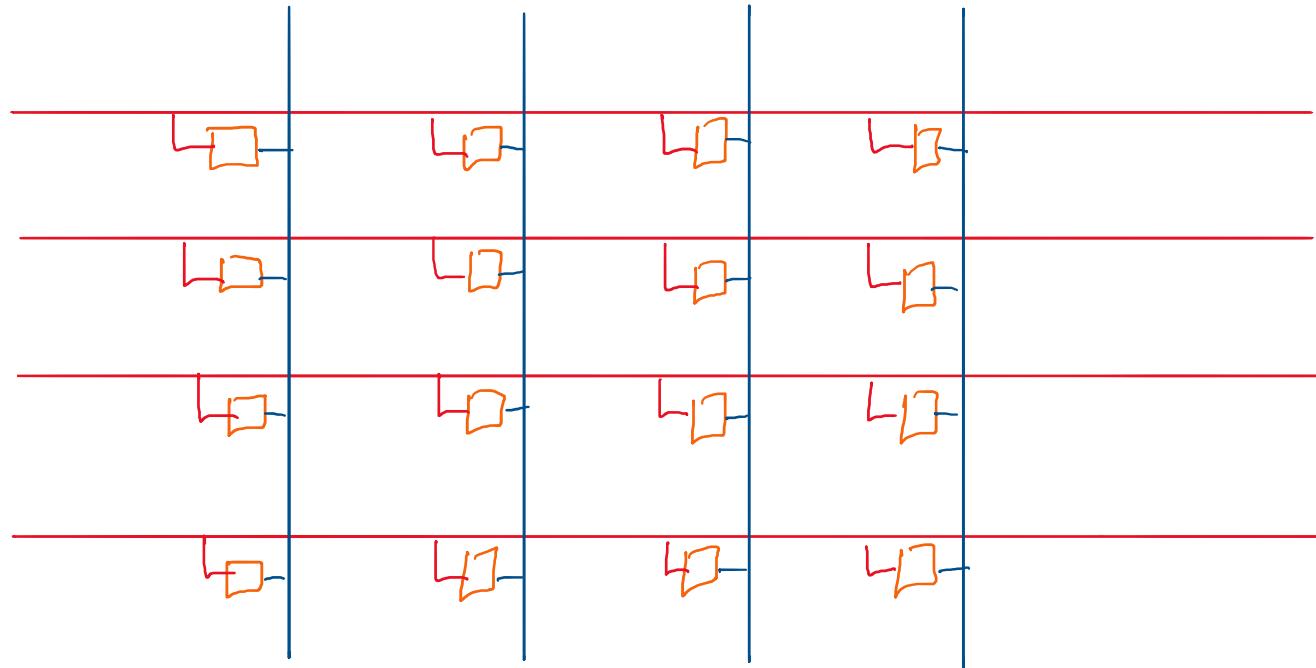
2-bit ROM



2-bit ROM



2-bit ROM



2-bit ROM \Rightarrow $= 0$

$Addr = 00$



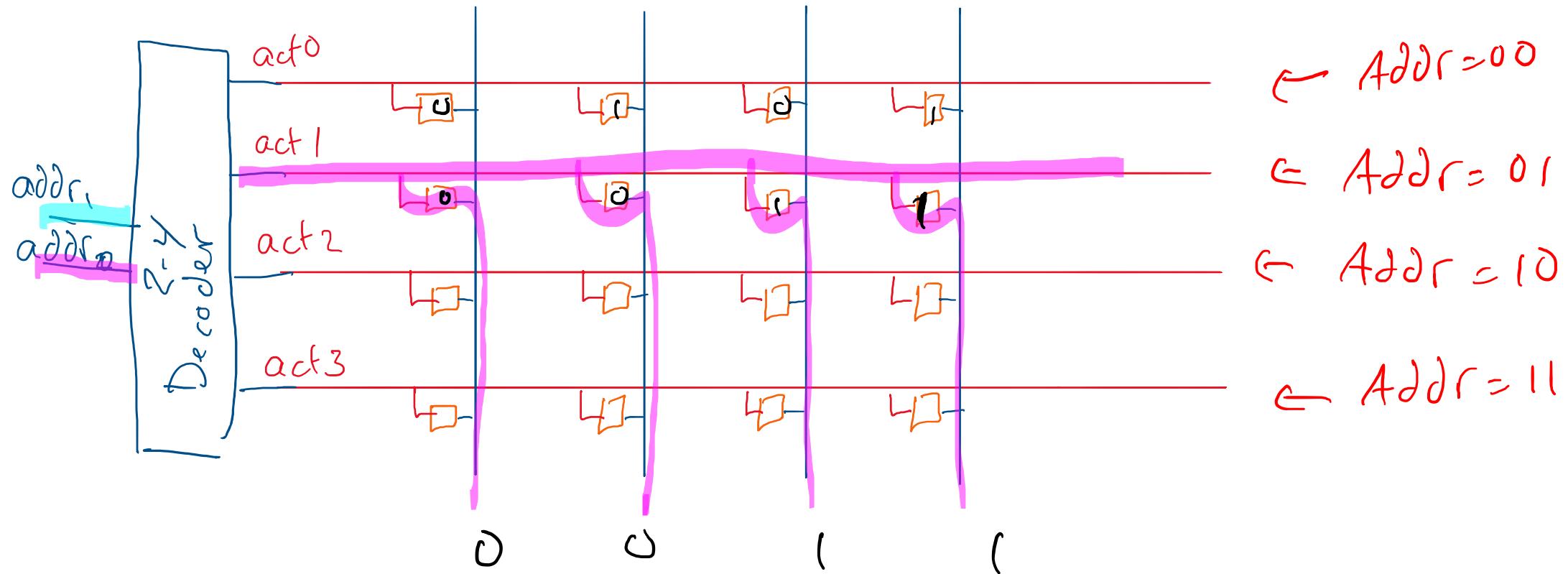
$Addr = 01$



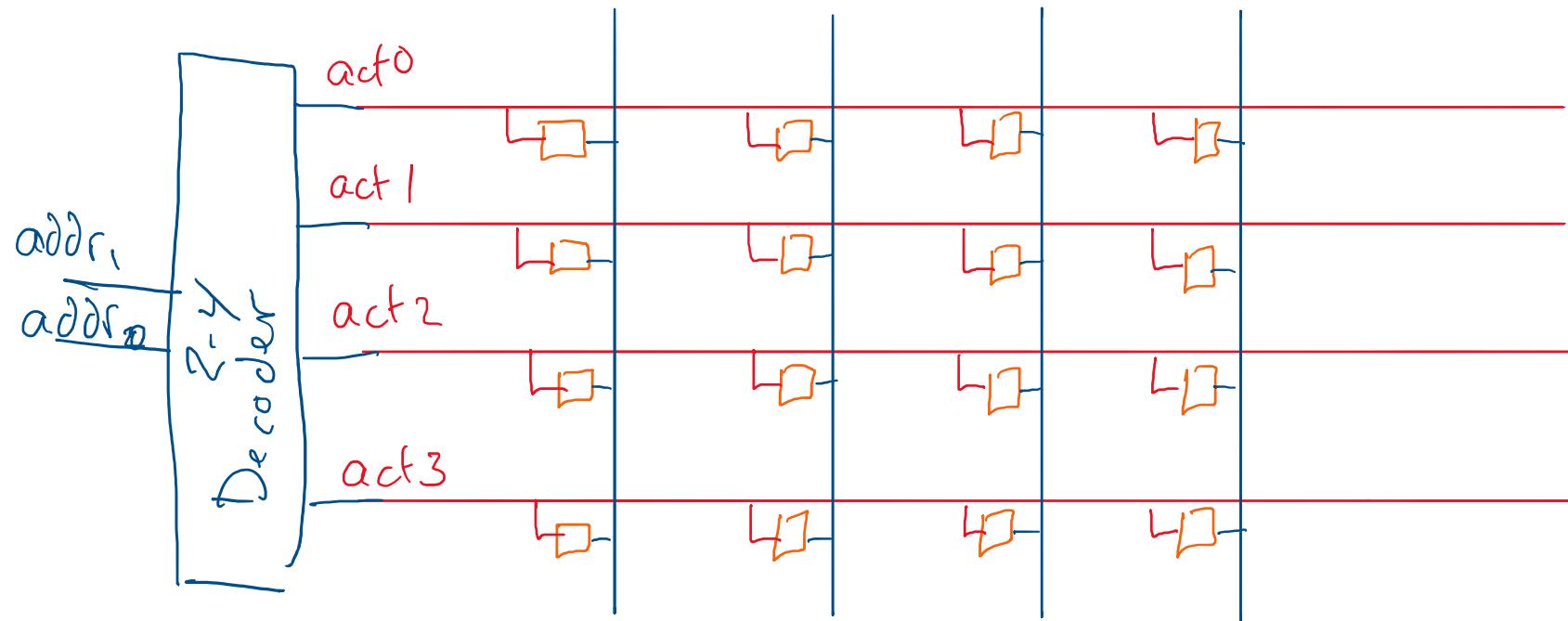
$Addr = 10$



$Addr = 11$

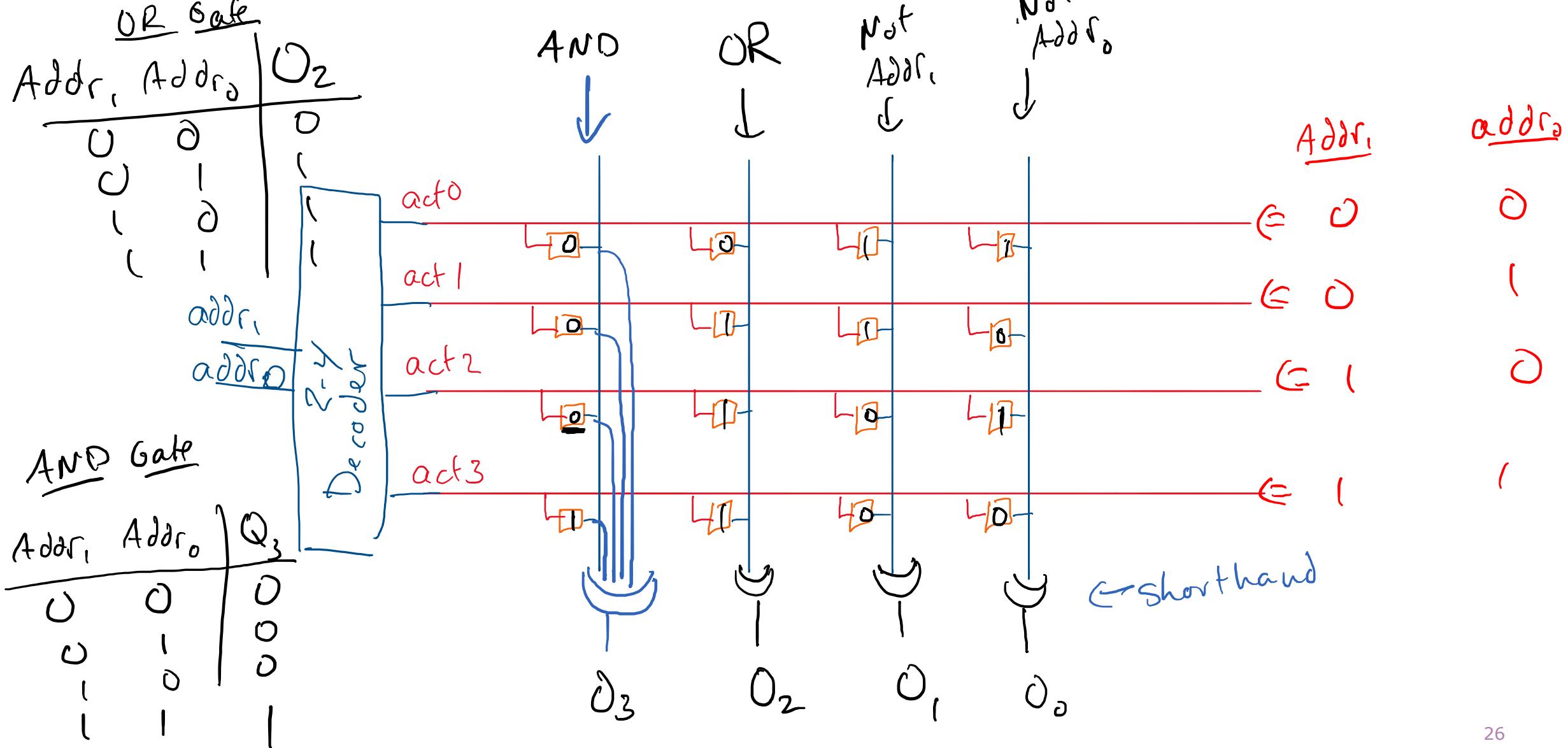


2-bit ROM



2 inputs
 $\Rightarrow 2^2$ outputs
 $= 4$ outputs

2-bit ROM of AND + OR

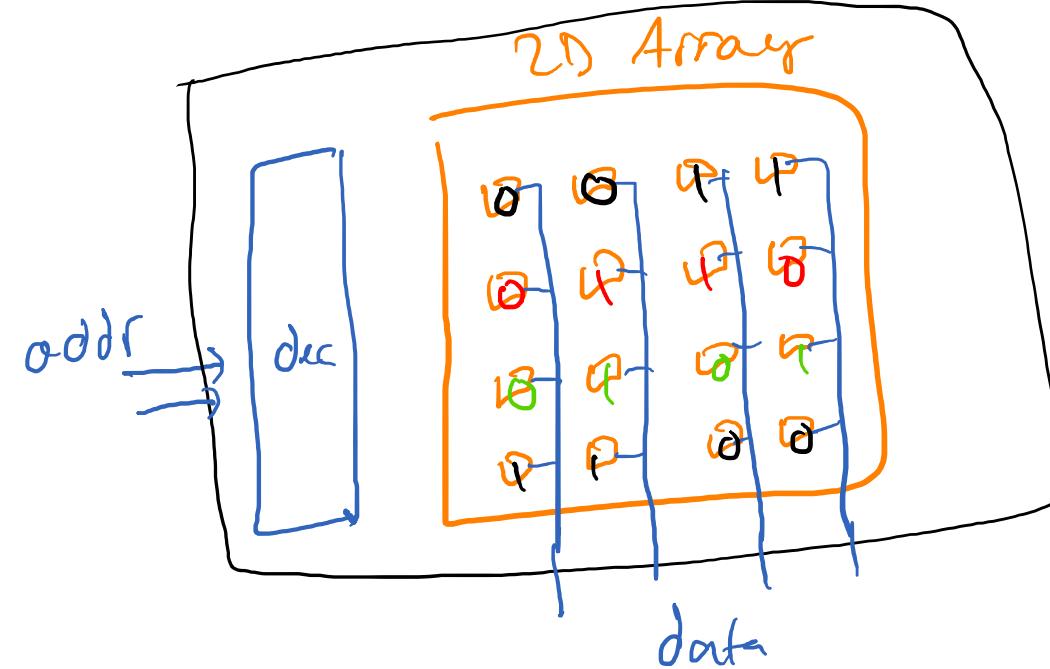


ROM in Verilog

```
module ROM (
    input [1:0] addr, ←
    output [3:0] data ←
)
    logic [3:0] array [0:3]; //2D Array
    assign array = { 4'b0011, 4'b0110, 4'b0101, 4'b1100 }
    assign data = array[addr]; ← Select a row for output
endmodule
```

*size w/in
row → big → small*

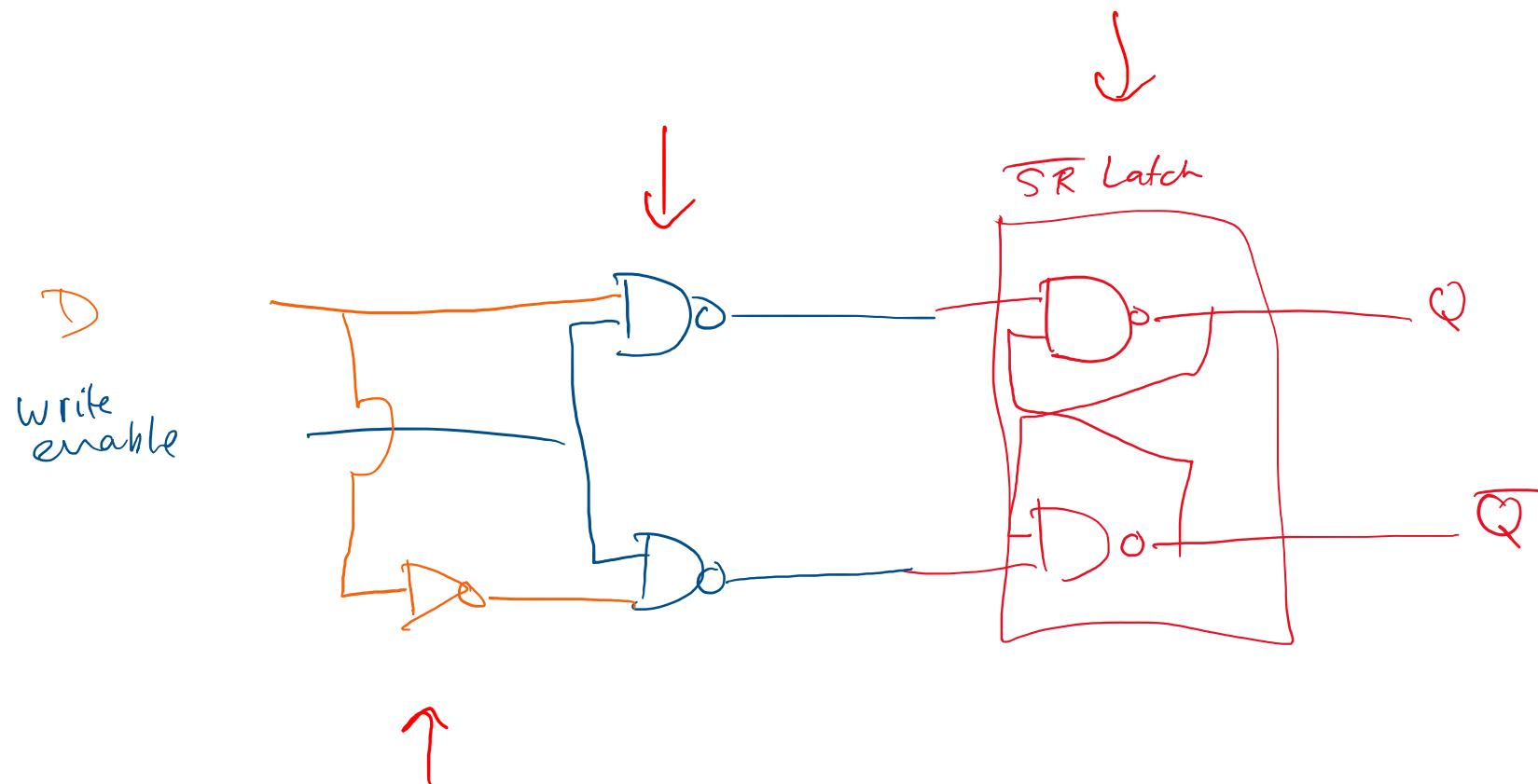
of rows, small → big



RAM

- Similar to ROM
- BUT WRITABLE!

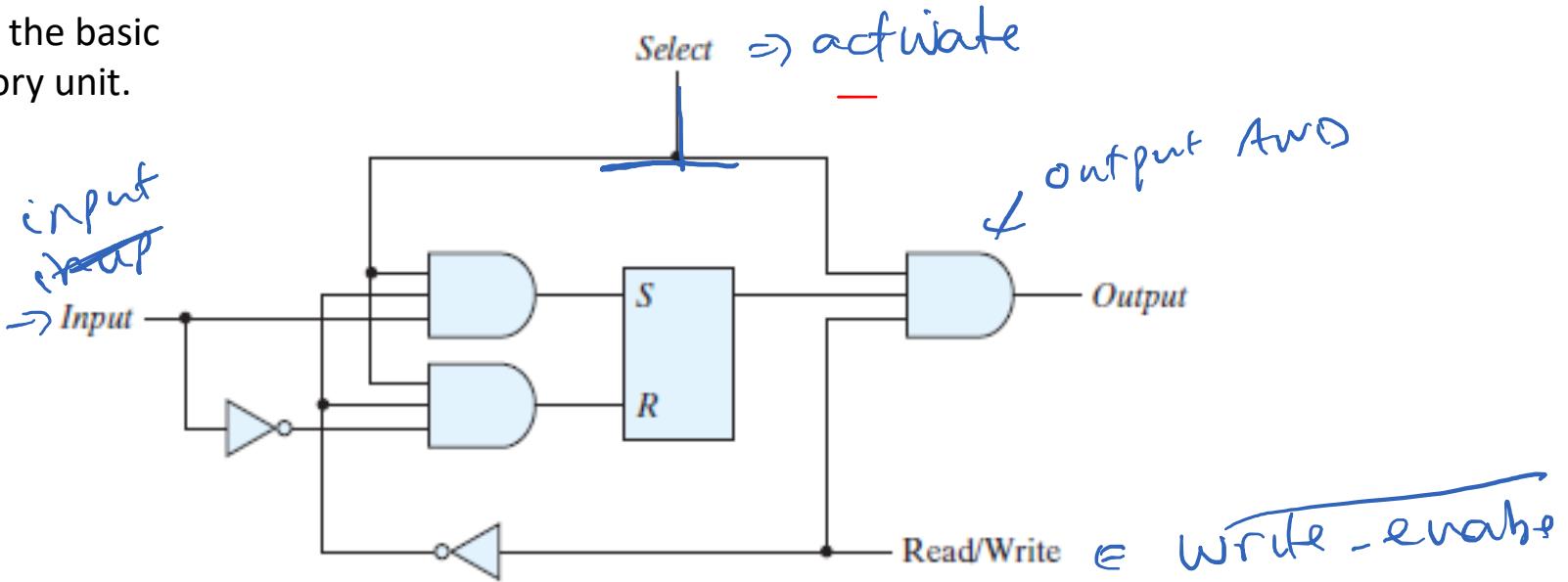
D Latch



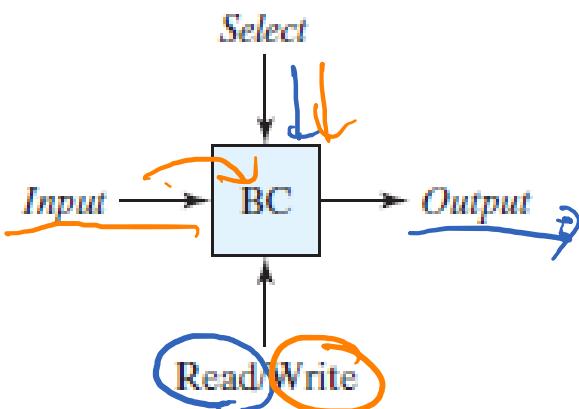
Static RAM cell

The binary storage cell is the basic building block of a memory unit.

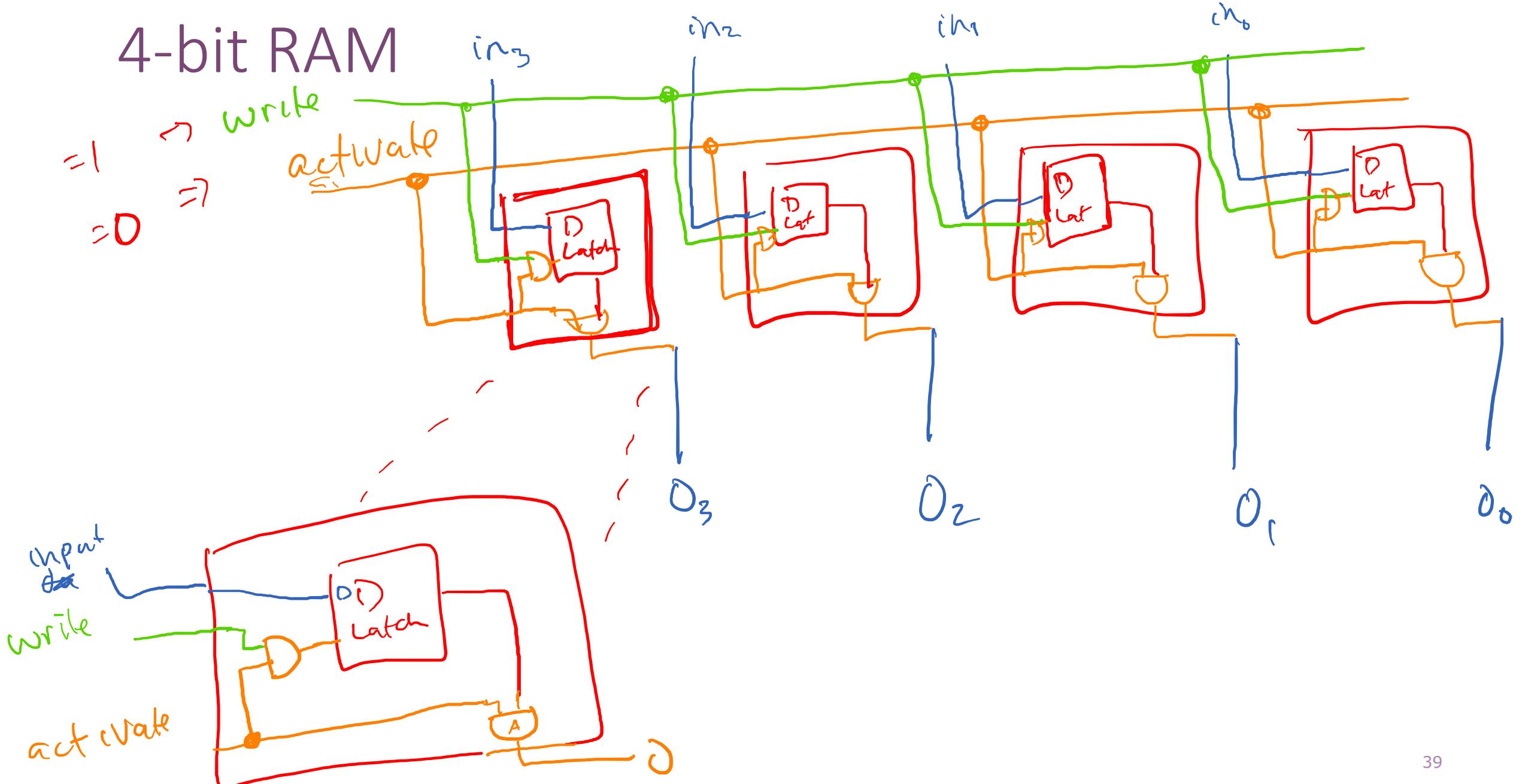
The equivalent logic of a binary cell that stores one bit of information:



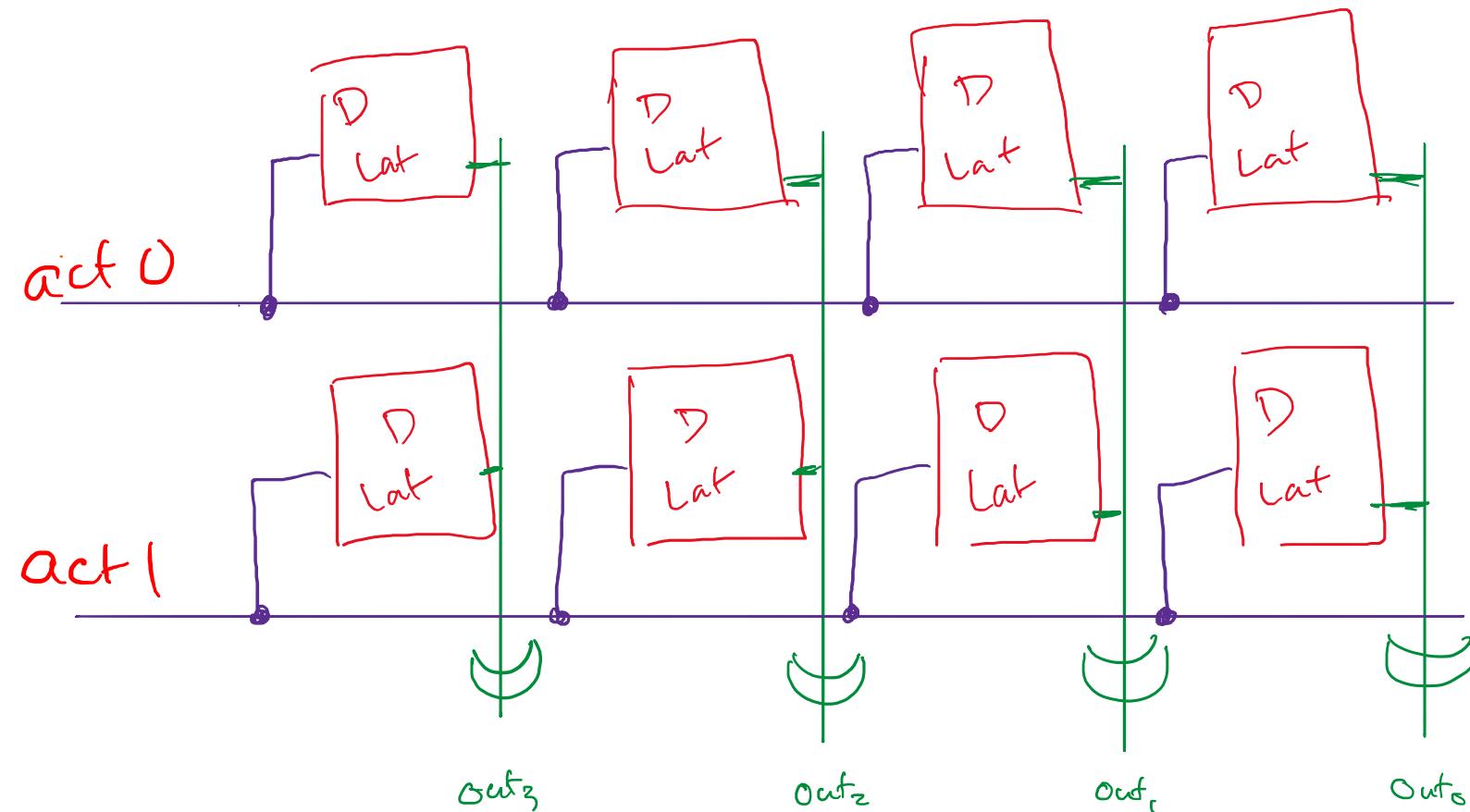
Block diagram:



4-bit RAM

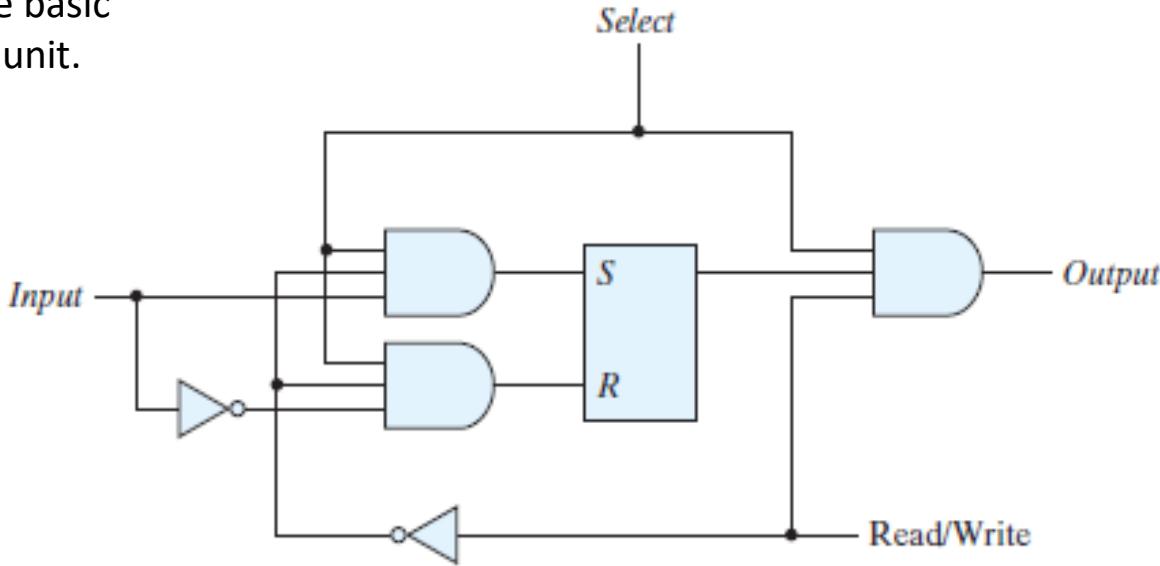


Array of 4-bit RAM

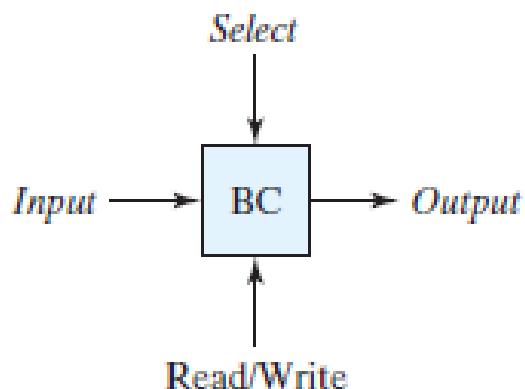


The binary storage cell is the basic building block of a memory unit.

The equivalent logic of a binary cell that stores one bit of information:

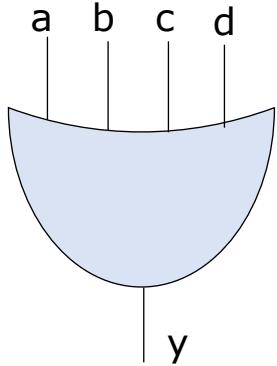


Block diagram:

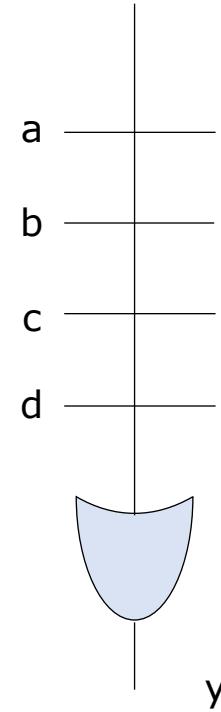


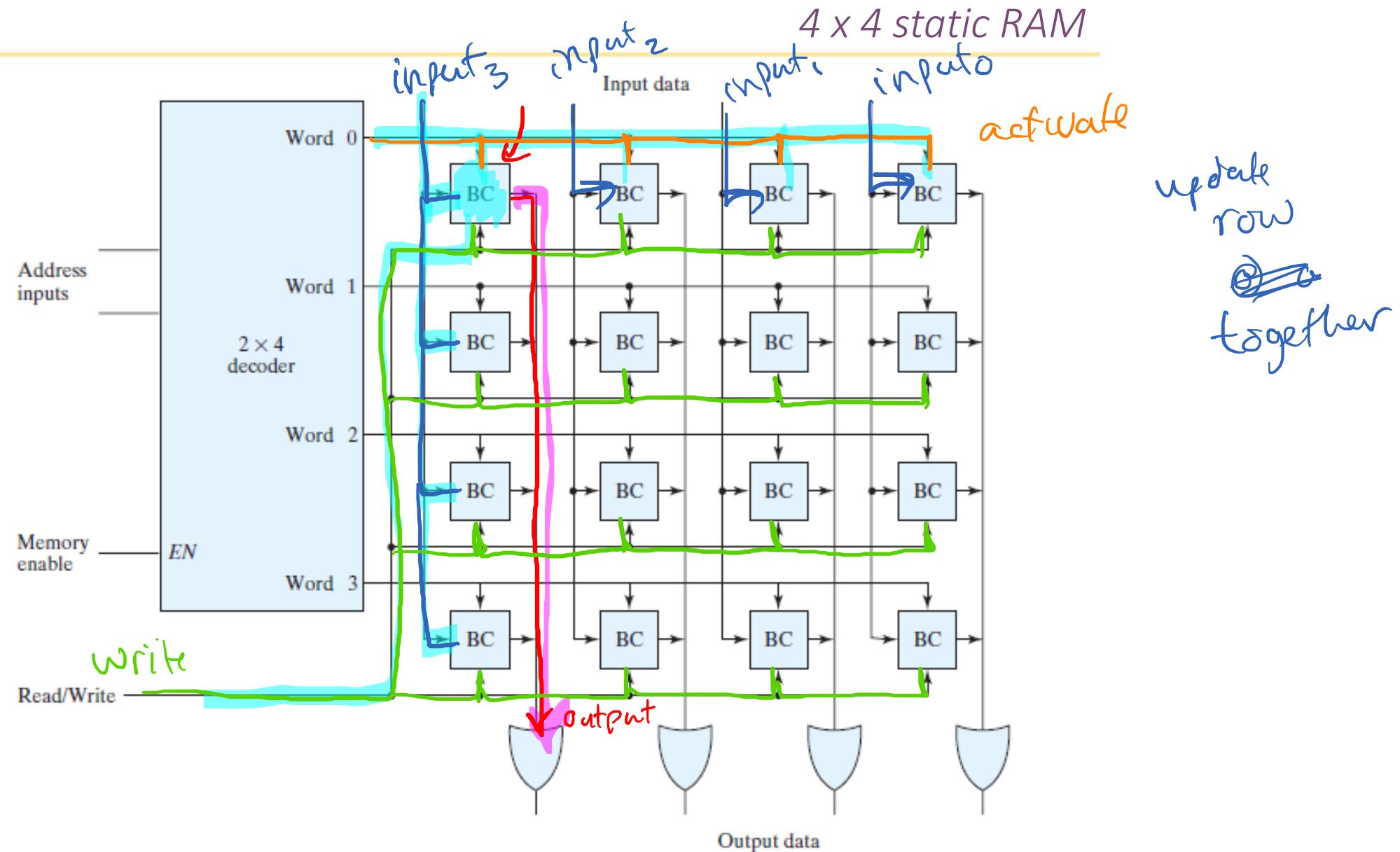
Notation for the gates

Instead of:

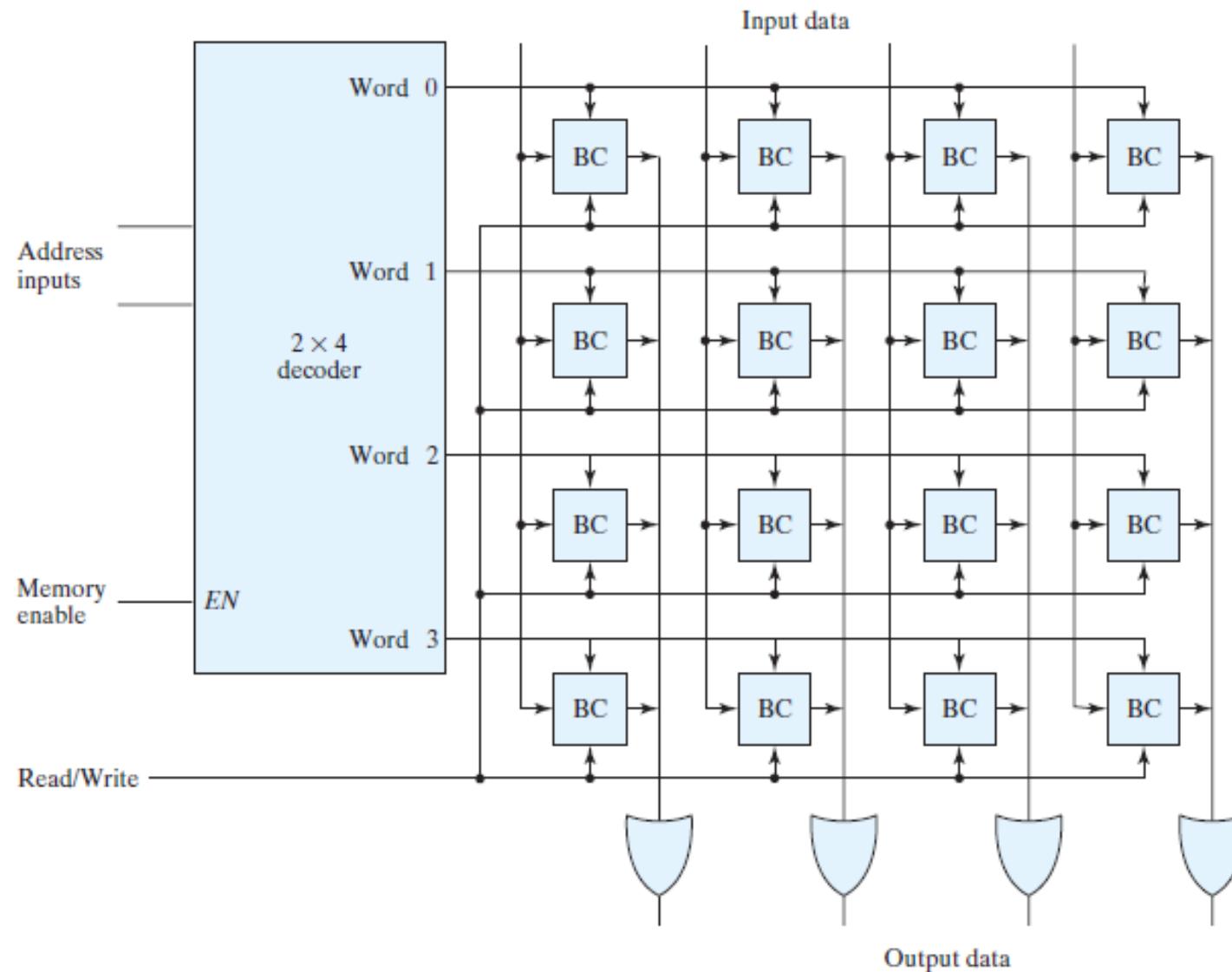


We often use
notation:

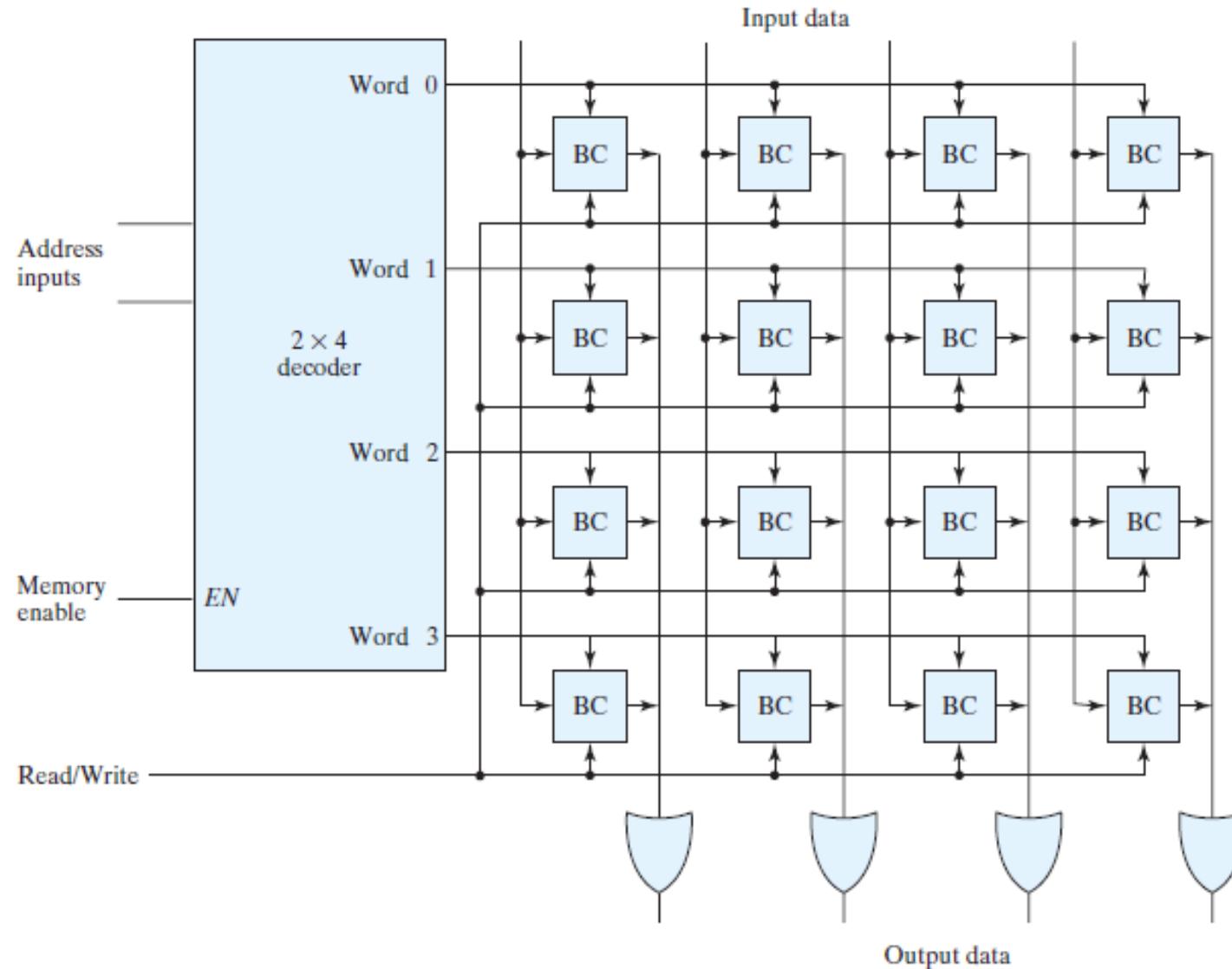




4 x 4 static RAM



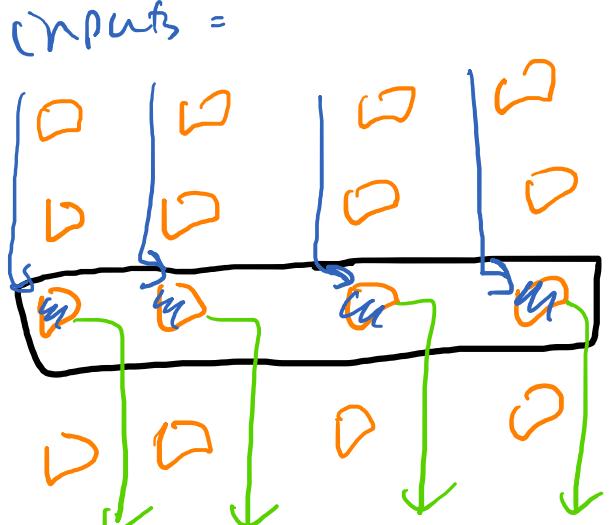
4 x 4 static RAM



Flip-Flop RAM in Verilog

```
module RAM (
    input      clk,
    input [1:0] addr,
    input      set,   ← write in previous slide
    input [3:0] set_data, ← input in previous slide
    output [3:0] read_data
)
    logic [3:0] array [0:3]; //2D Array
    always_ff @ (posedge CLK) begin
        if (set) array[addr] <= set_data;
    end
    assign read_data = array[addr];
endmodule
```

addr →



Flip-Flop RAM in Verilog

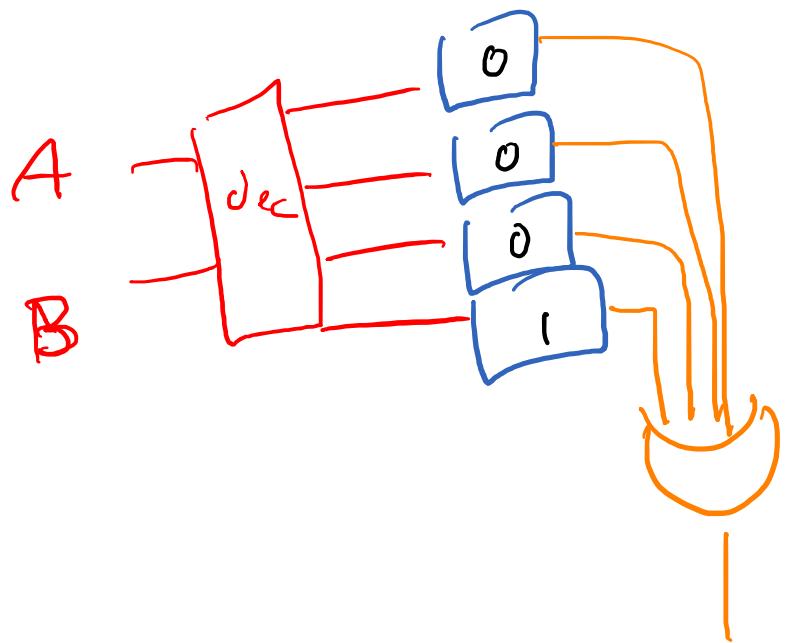
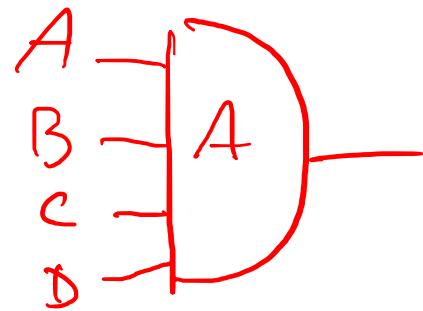
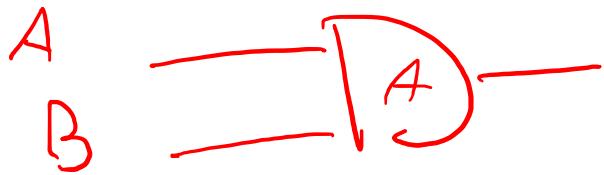
```
module RAM (
    input      clk,
    input [1:0] addr,
    input      set,
    input [3:0] set_data,
    output [3:0] read_data
)
    logic [3:0] array [0:3]; //2D Array
    always_ff @(posedge clk) begin
        if (set) array[addr] <= set_data;
    end
    assign read_data = array[addr];
endmodule
```

Latch RAM in Verilog

```
module RAM (                                ← does not need clk
    input [1:0] addr,
    input        set,
    input [3:0]  set_data,
    output [3:0] read_data
)
    logic [3:0] array [0:3]; //2D Array
    always_latch begin //if you really want a latch
        if (set) array[addr] = set_data; ← not have
    end
    assign read_data = array[addr];
endmodule
```

← does not need clk

→ not have default



$A \otimes B$

