Canvas: Upload Videos

Website: PDF Slides

ENGR 210 / CSCI B441
"Digital Design"

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

Exam

Post old Exam?

After Exam, Prof. Hiranebaugh

→ Wednesday: Review

Monday: Exam

→ comprehensive exam

→ verilog writing

→ basic syntax issues ⇒ don't care

→ always_ff, always_comb

⇒ inferred latches

⇒ missing defaults,

→ testbench

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

post Q2 → End on Canvas @ 9:25 ET, due back @ 11am ET

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.

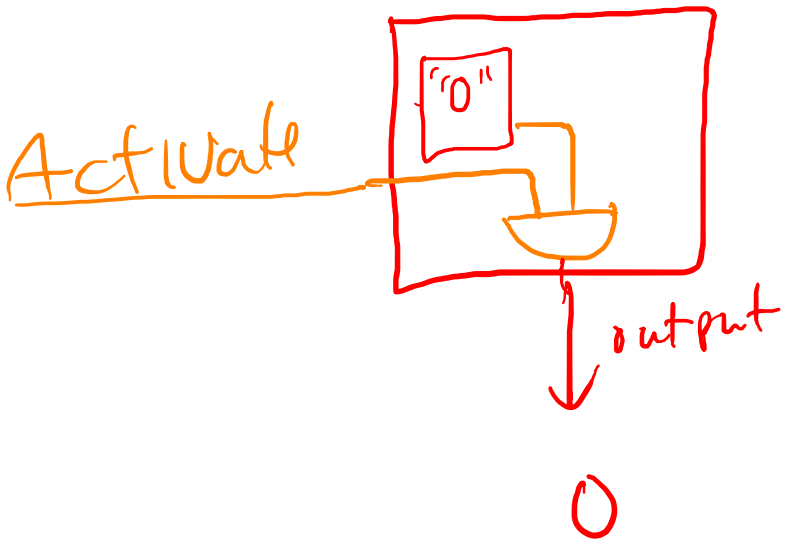
inputs: Address (Addr)
output: Value (data)

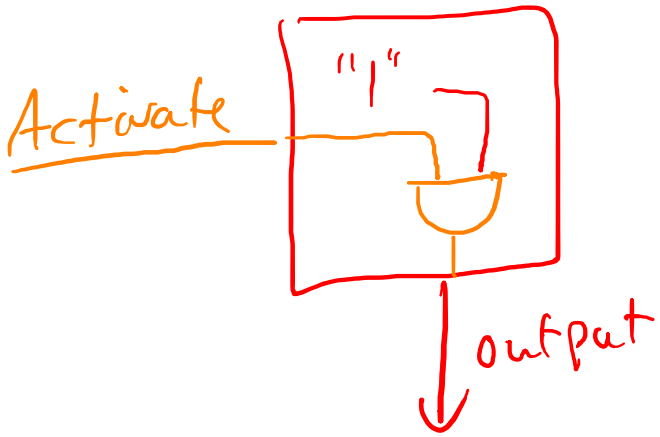
ROM vs RAM

- ROM – Read-Only Memory
  - Input: address
  - Output: fixed value

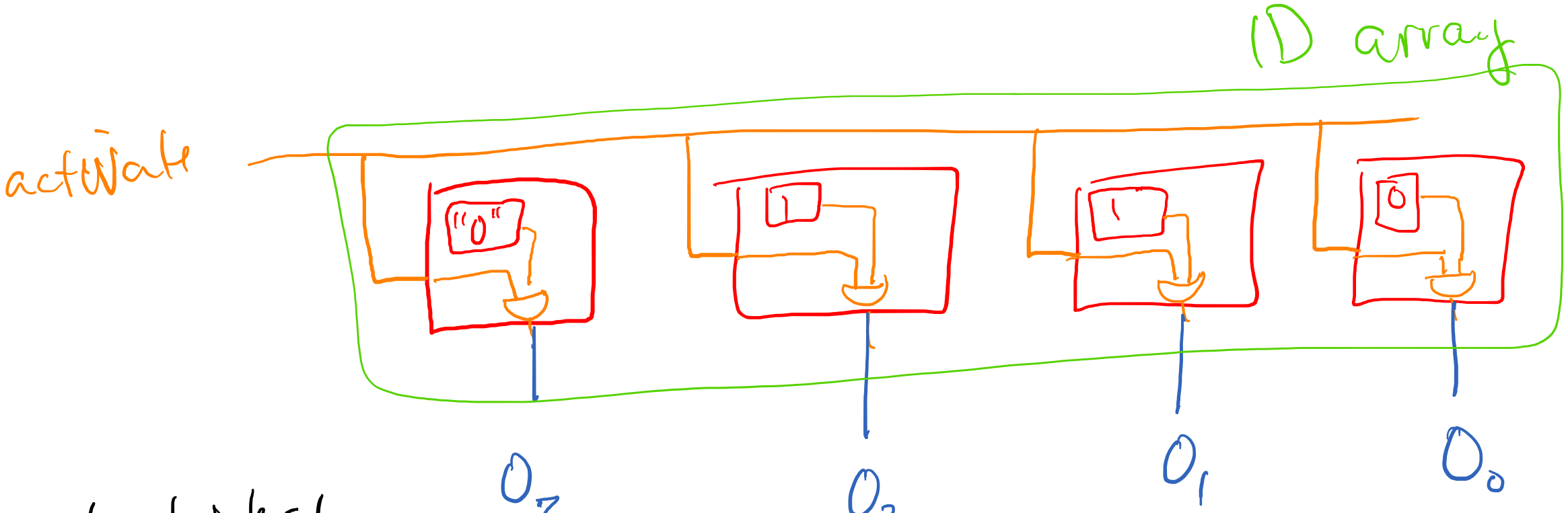
- RAM – Random-Access Memory
  - Read/Write version of a ROM

Rom Cell





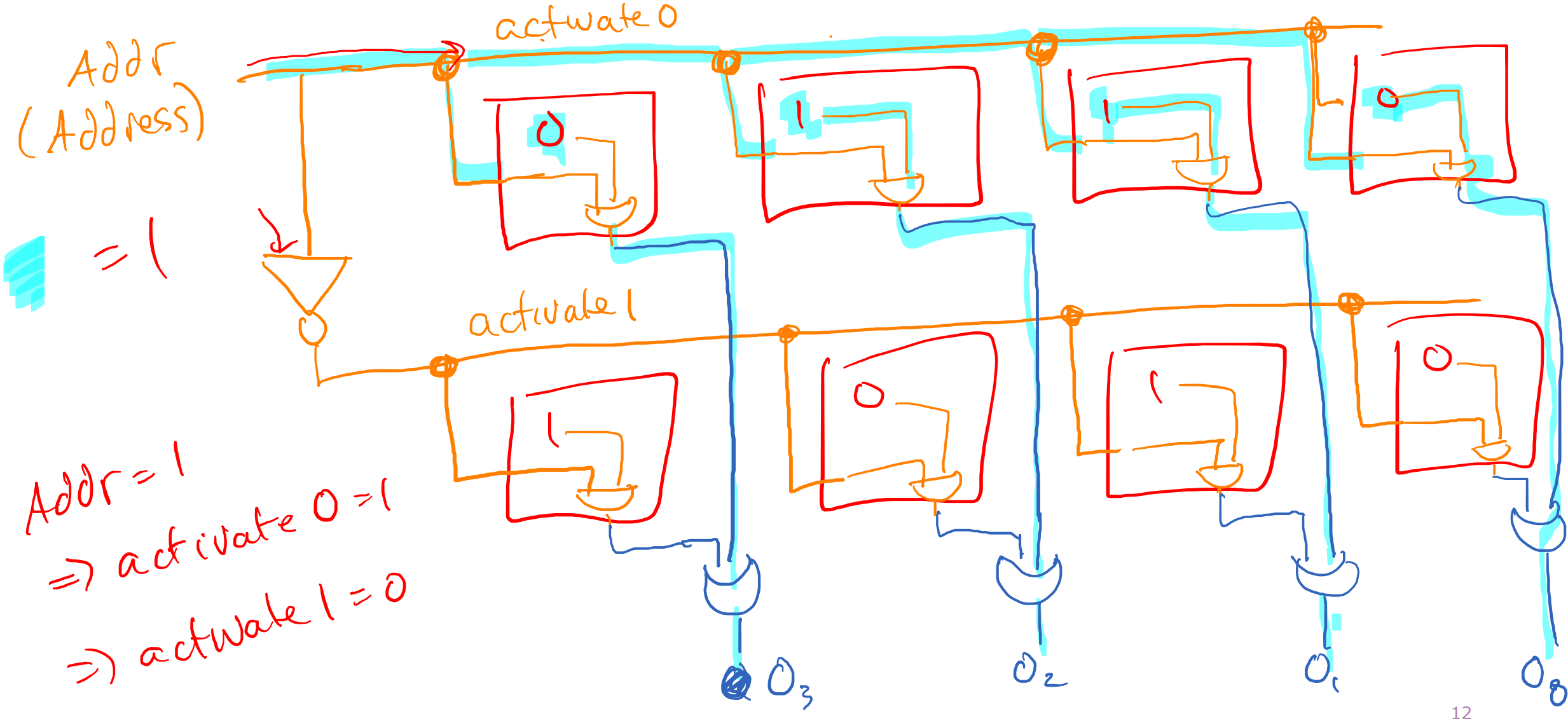
Multiple ROM Cells



w/ activate = 1 $O_3 = 0$ $O_2 = 1$ $O_1 = 1$ $O_0 = 0$

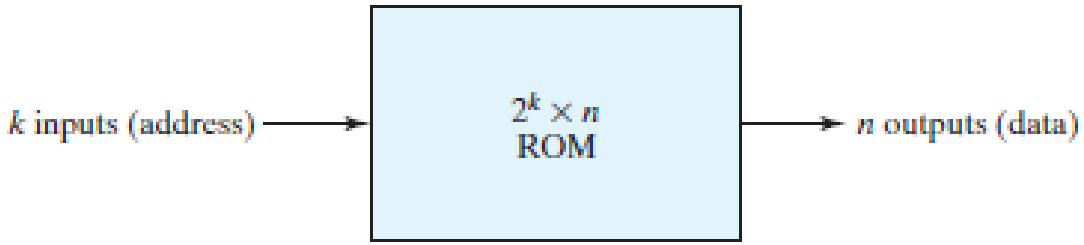
w/ activate = 0 $= 0$ $= 0$ $= 0$ $= 0$

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.

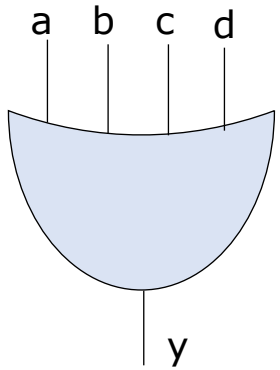


In addition to address inputs and data outputs, integrated circuit ROM chips have:

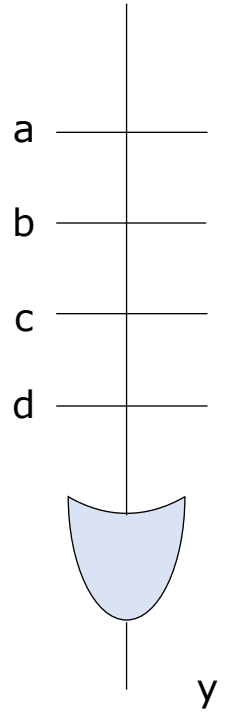
- One or more enable inputs
- Sometimes come with three-state outputs to facilitate the construction of large arrays of ROM.

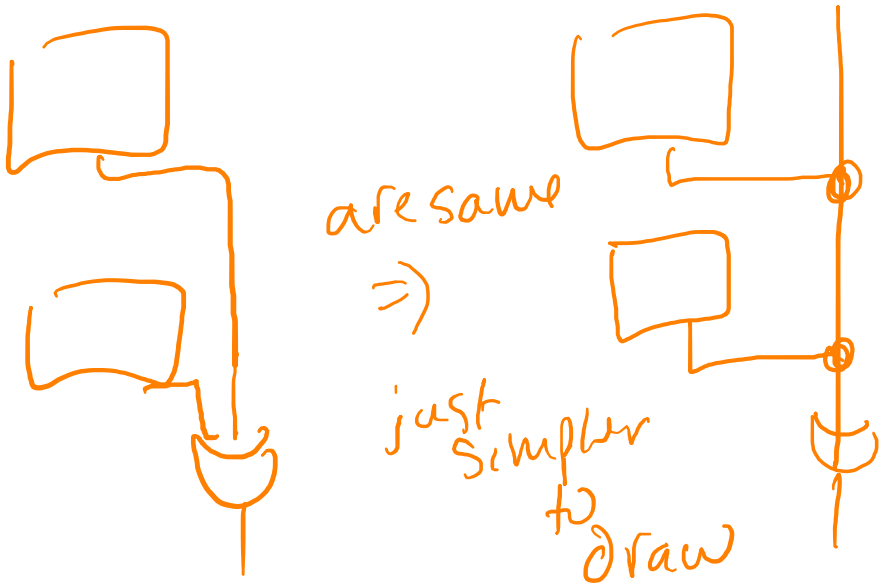
Notation for the gates

Instead of:

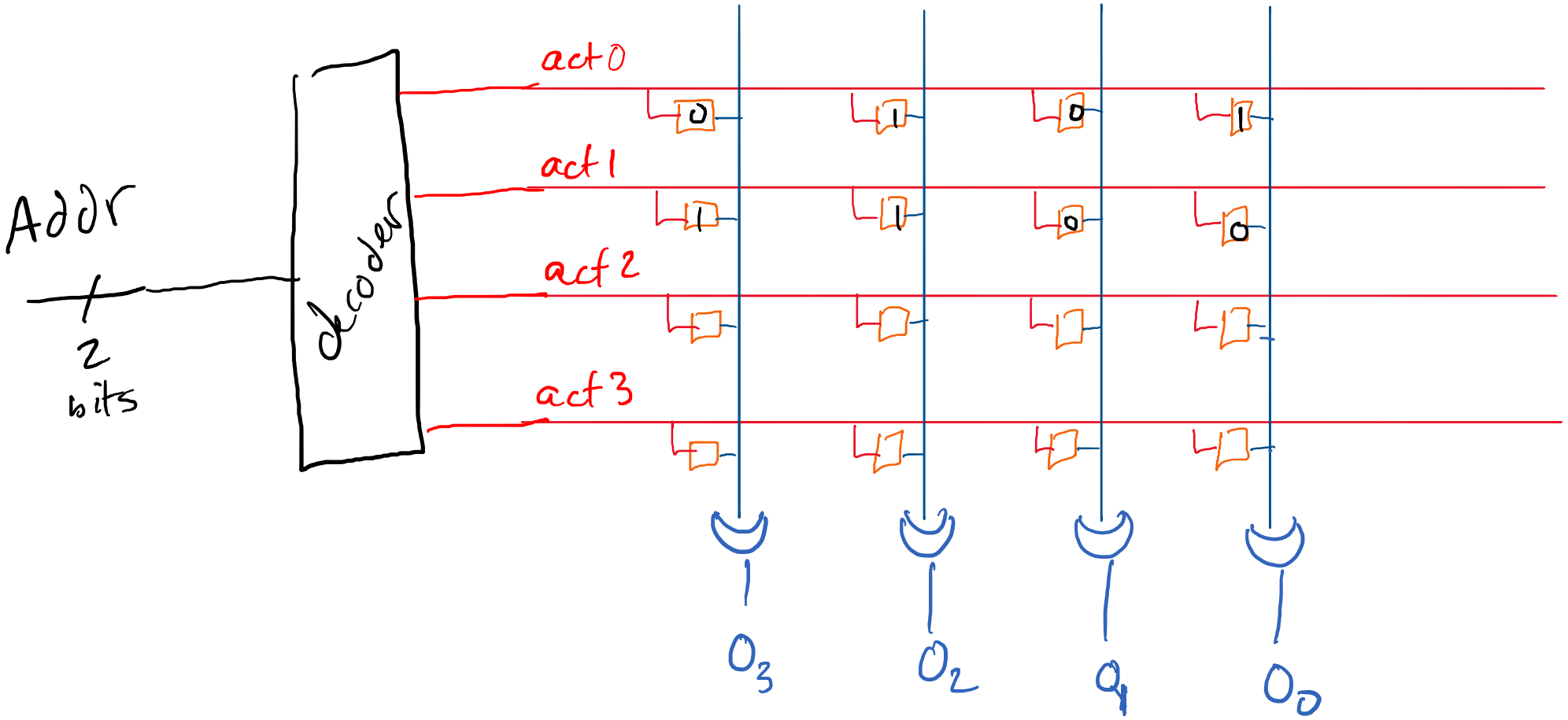


We often use notation:

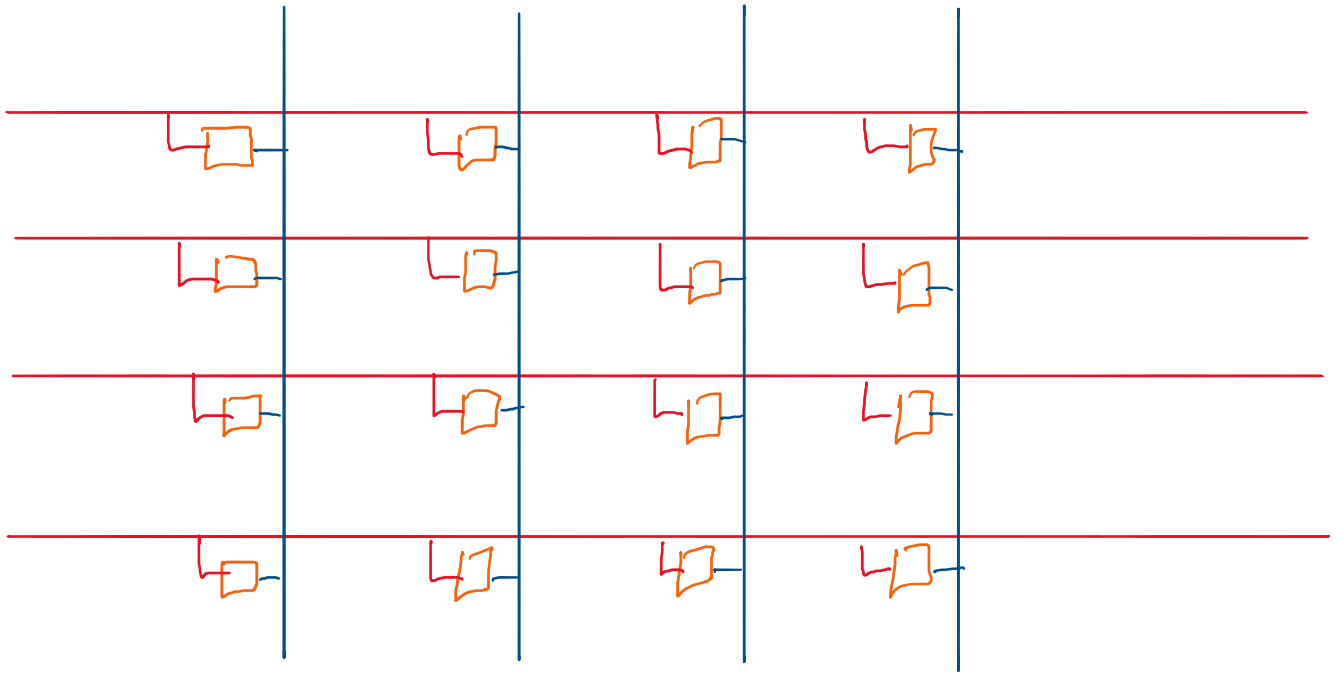




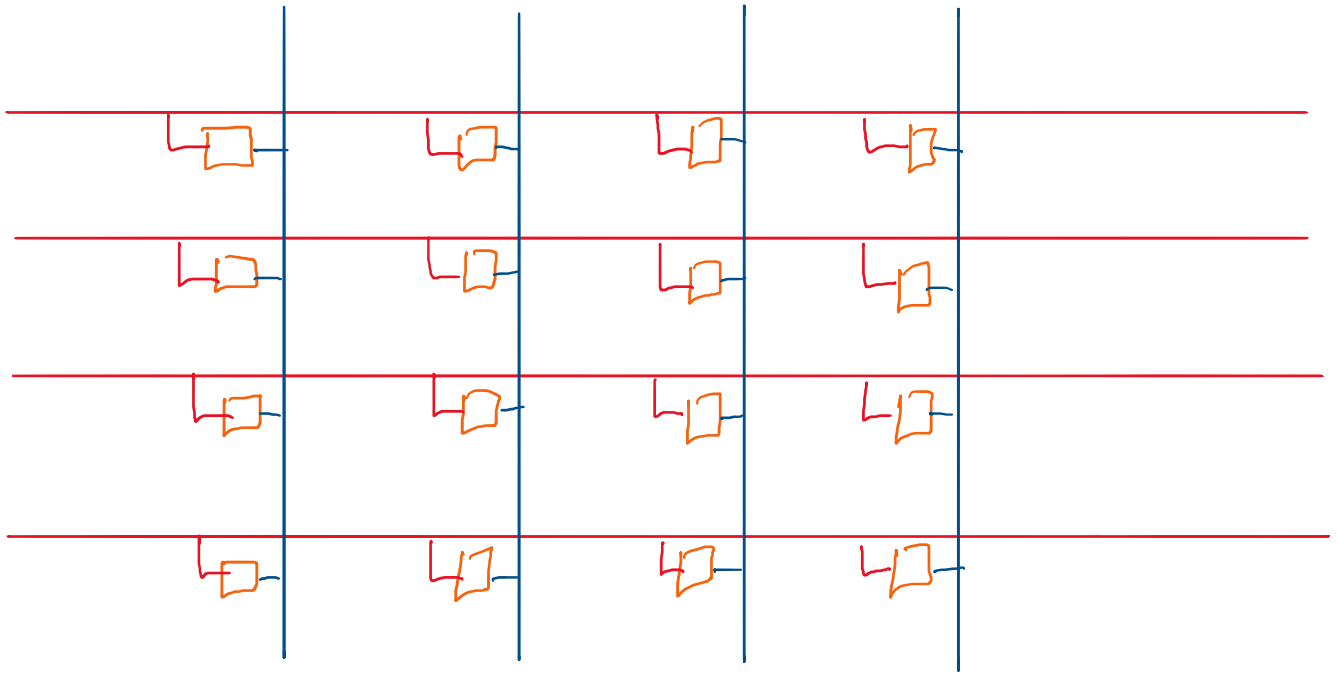
2-bit ROM



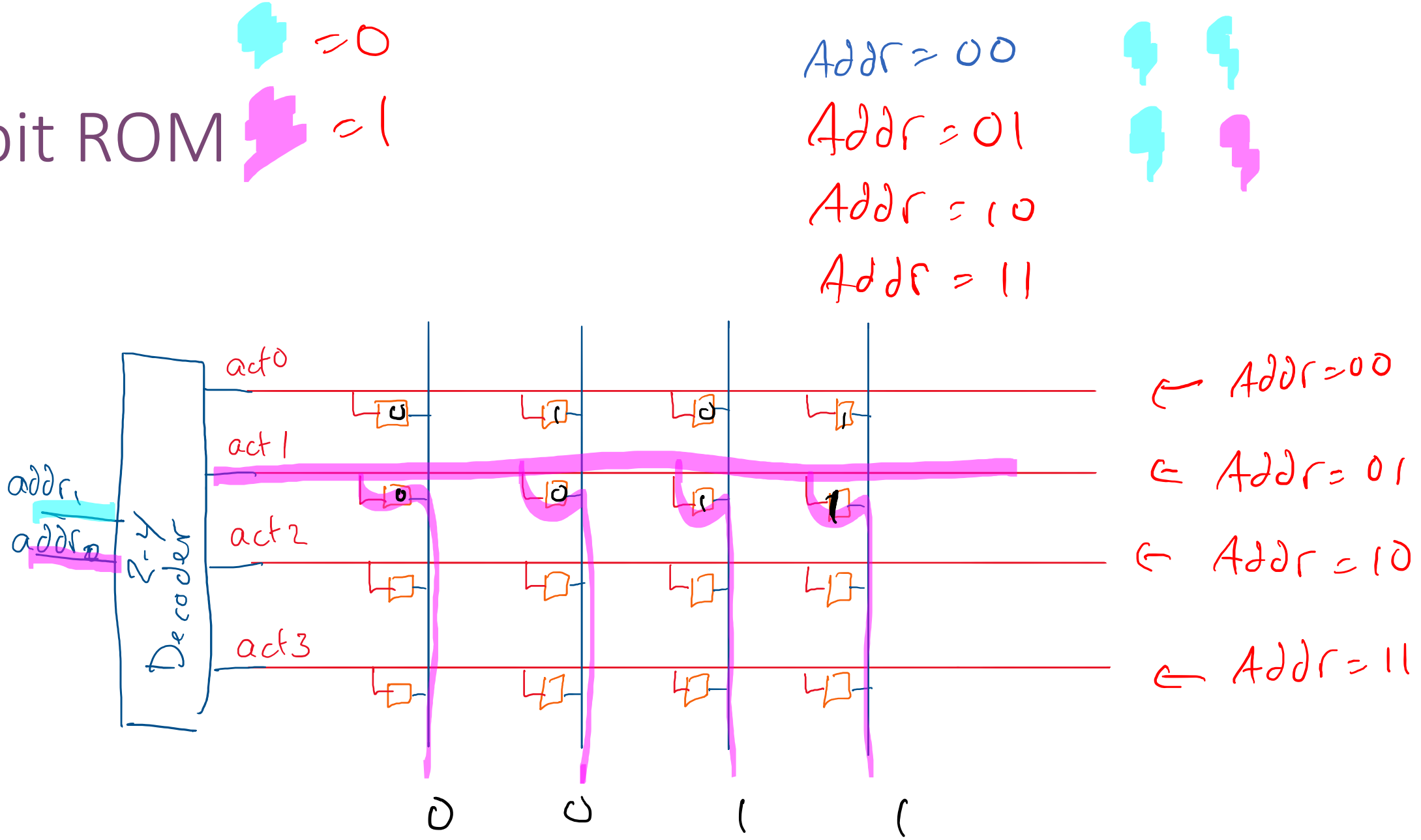
2-bit ROM



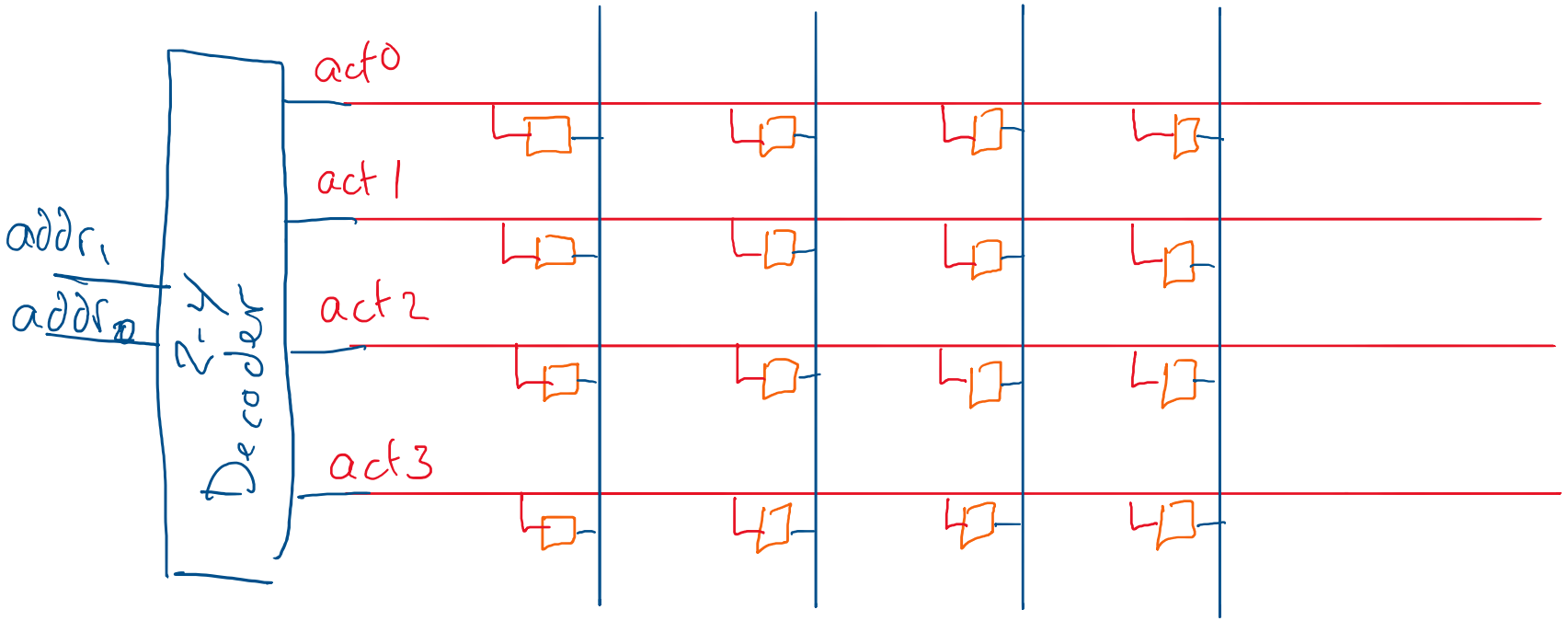
2-bit ROM



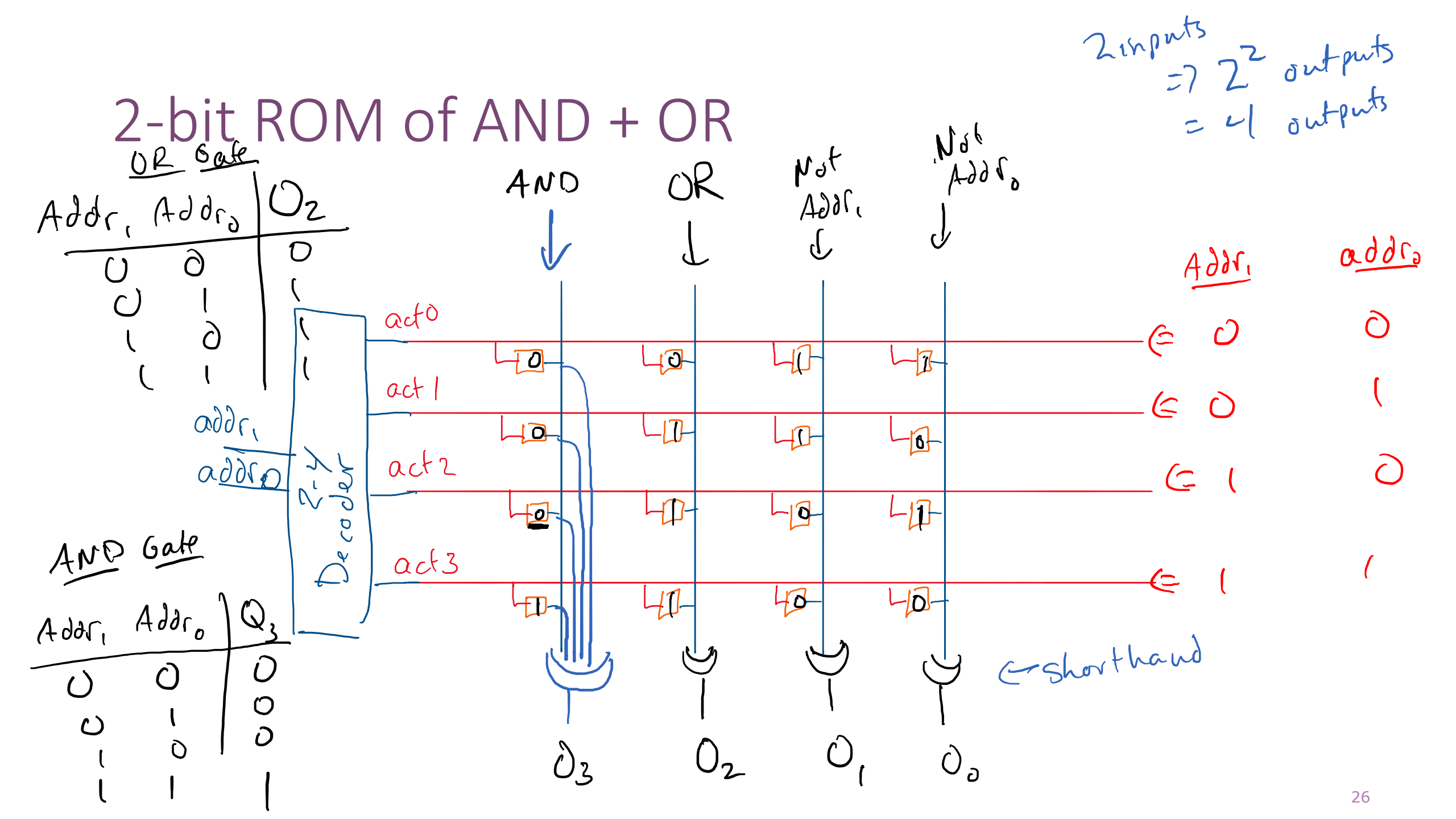
2-bit ROM


=0
=1
Addr=00
Addr=01
Addr=10
Addr=11
act0
act1
act2
act3
addr1
addr0
2-4 Decoder
← Addr=00
← Addr=01
← Addr=10
← Addr=11
0 0 1 1

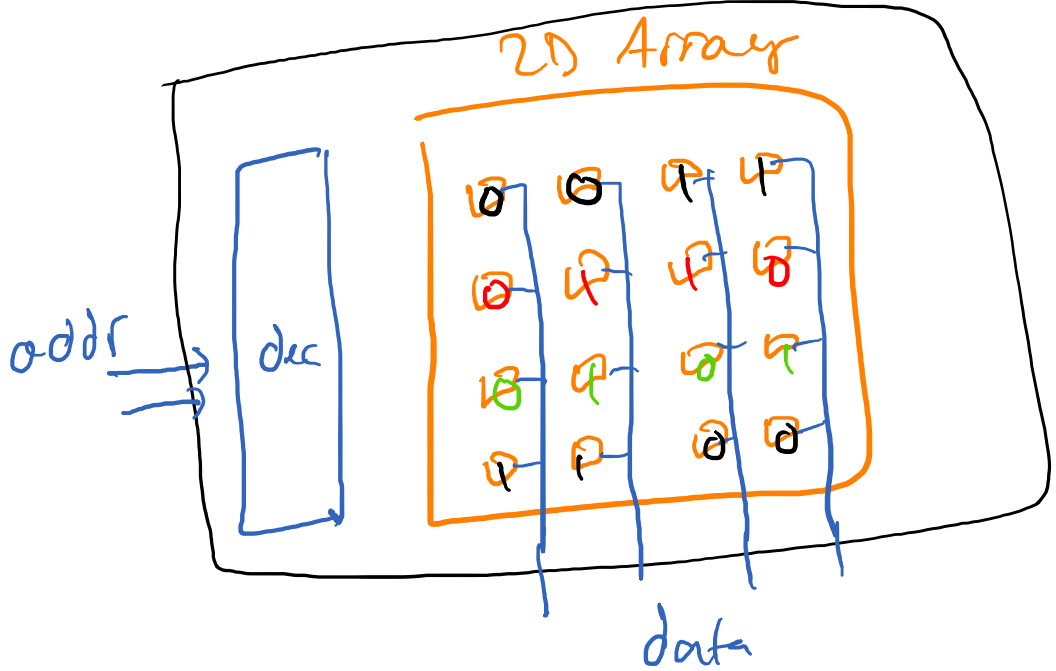
2-bit ROM



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];
endmodule
```

size w/in row → ~~bit~~ big → small

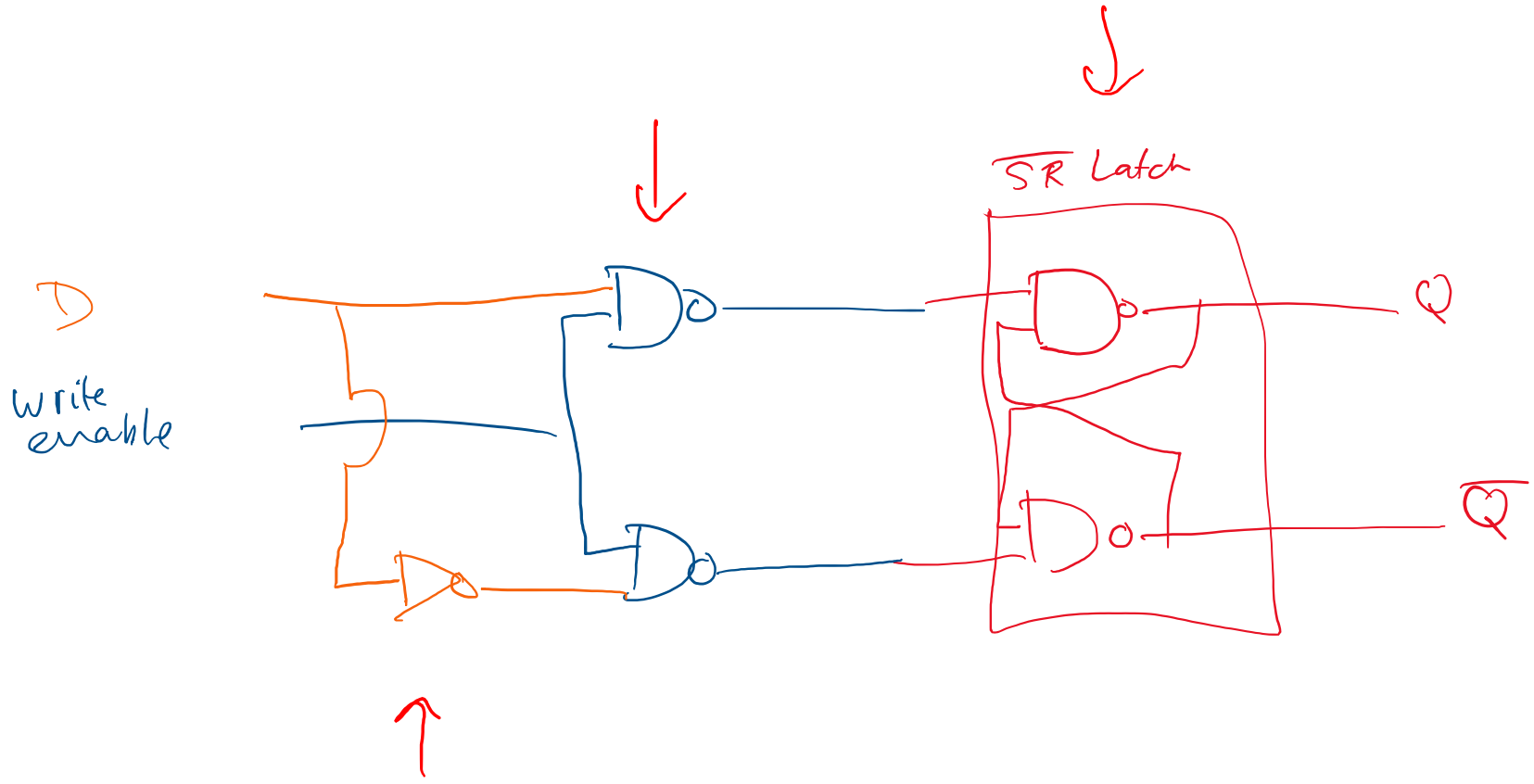
# of rows, small → big

← select a row for output

RAM

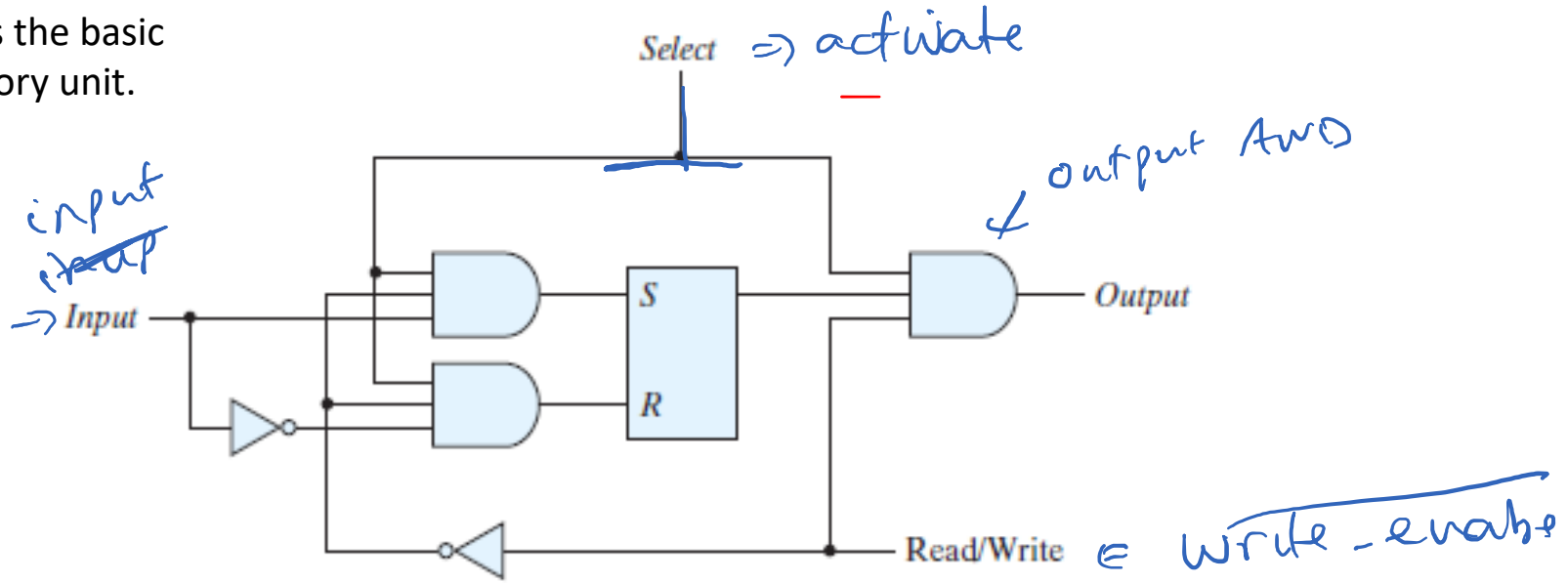
- Similar to ROM
- BUT WRITABLE!

D Latch

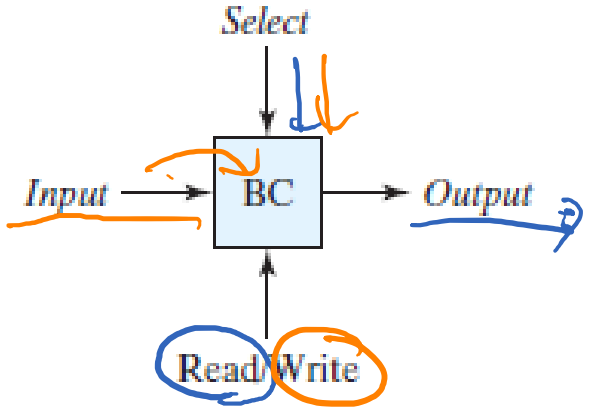


The binary storage cell is the basic building block of a memory unit.

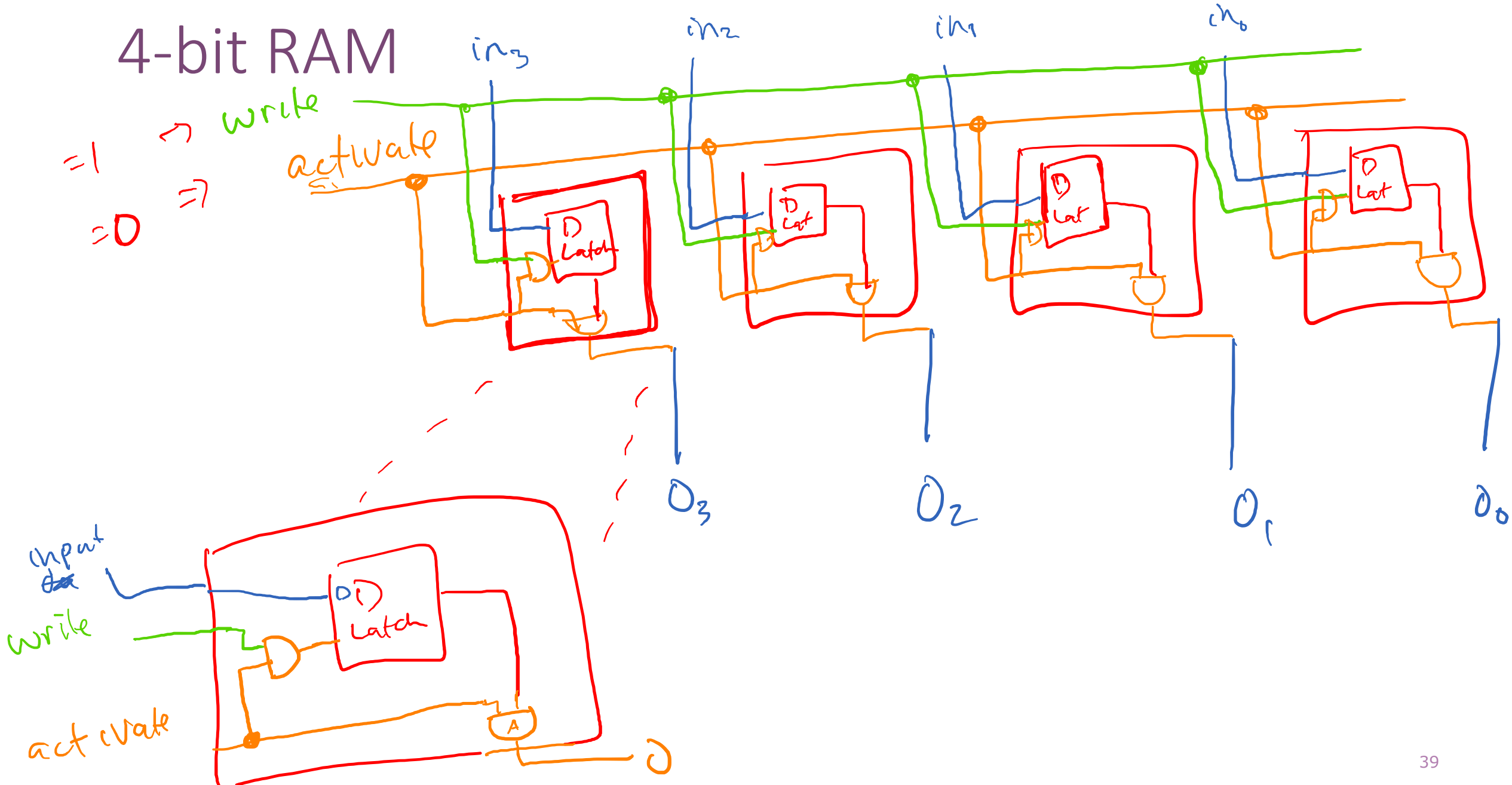
The equivalent logic of a binary cell that stores one bit of information:



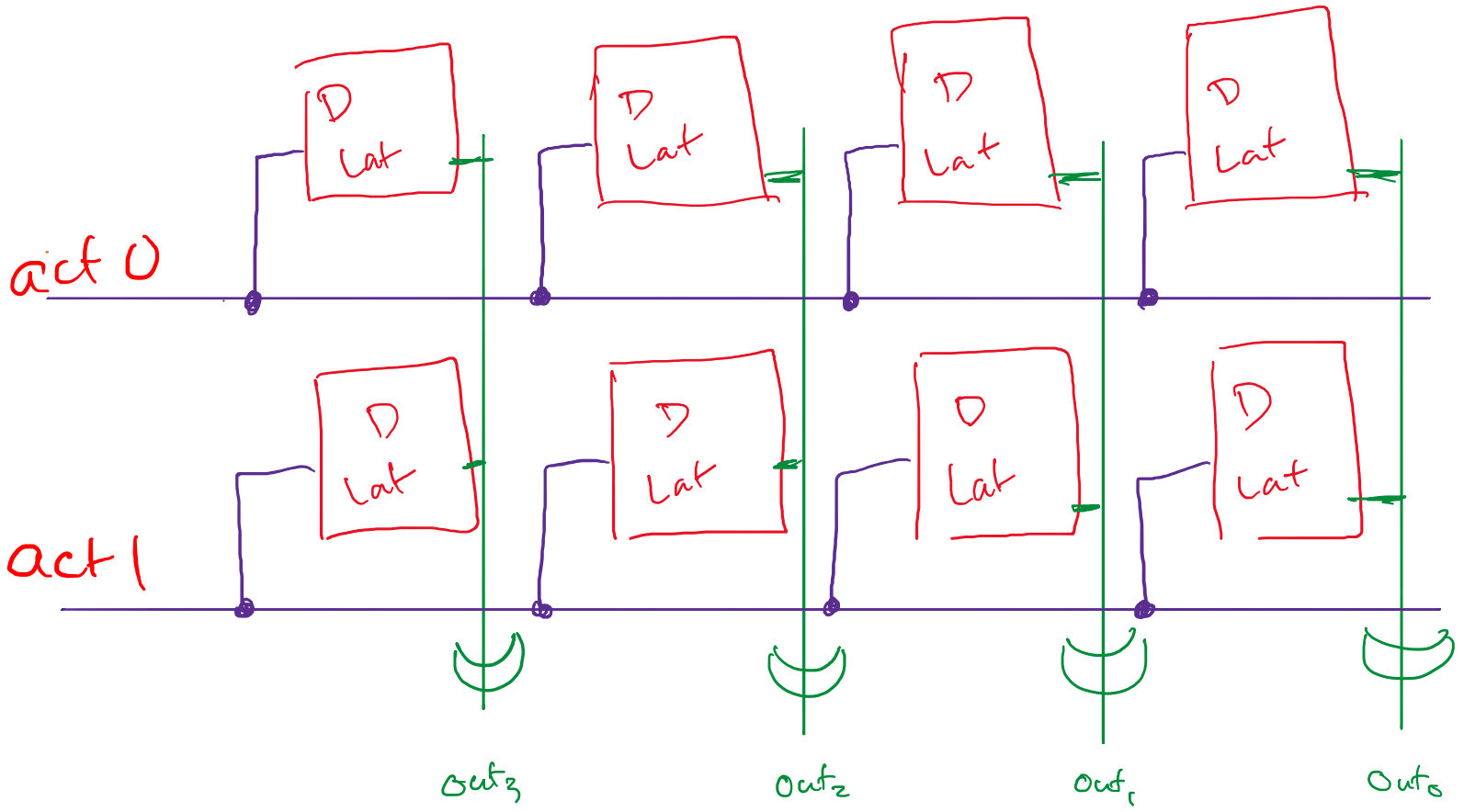
Block diagram:



4-bit RAM

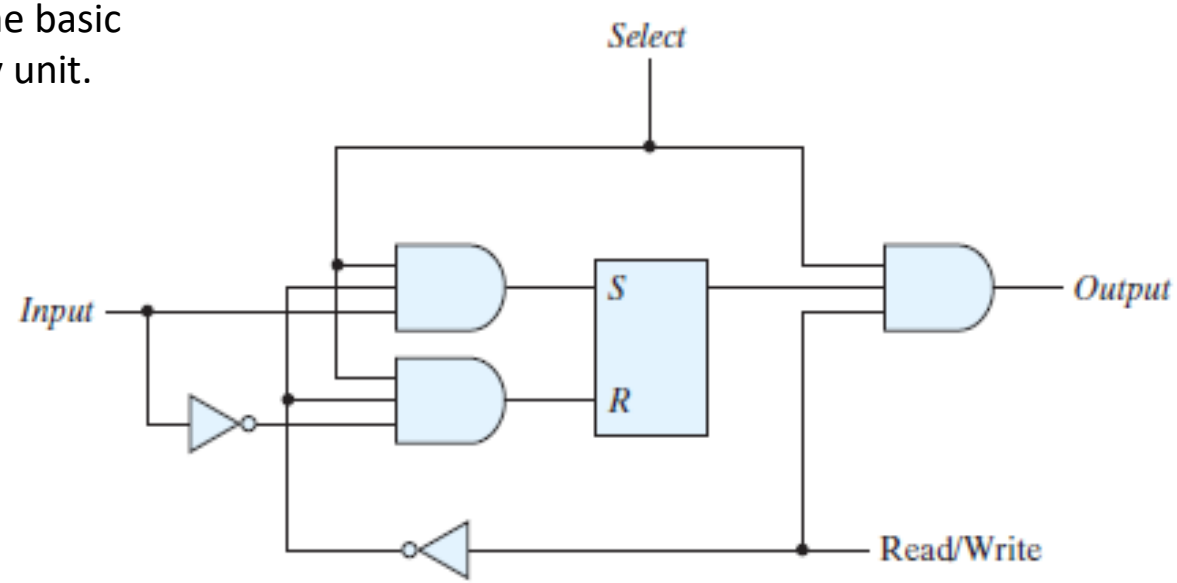

write
activate
=1
=0
in_3
in_2
in_1
in_0
D Latch
O_3
O_2
O_1
O_0
input
write
activate
A

Array of 4-bit RAM

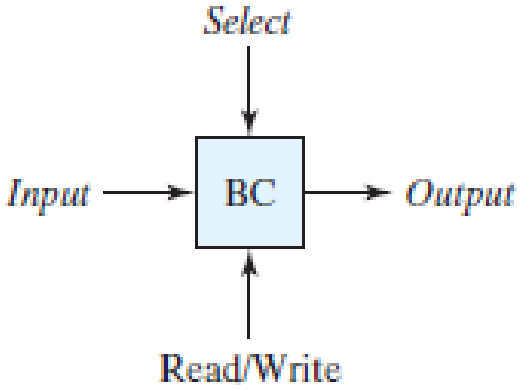


The binary storage cell is the basic building block of a memory unit.

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

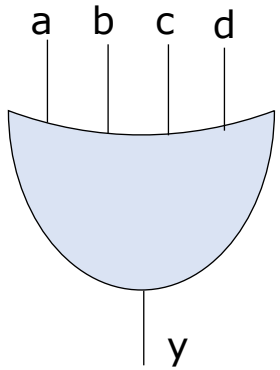


Block diagram:

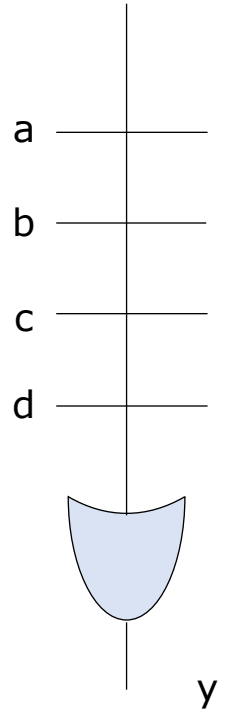


Notation for the gates

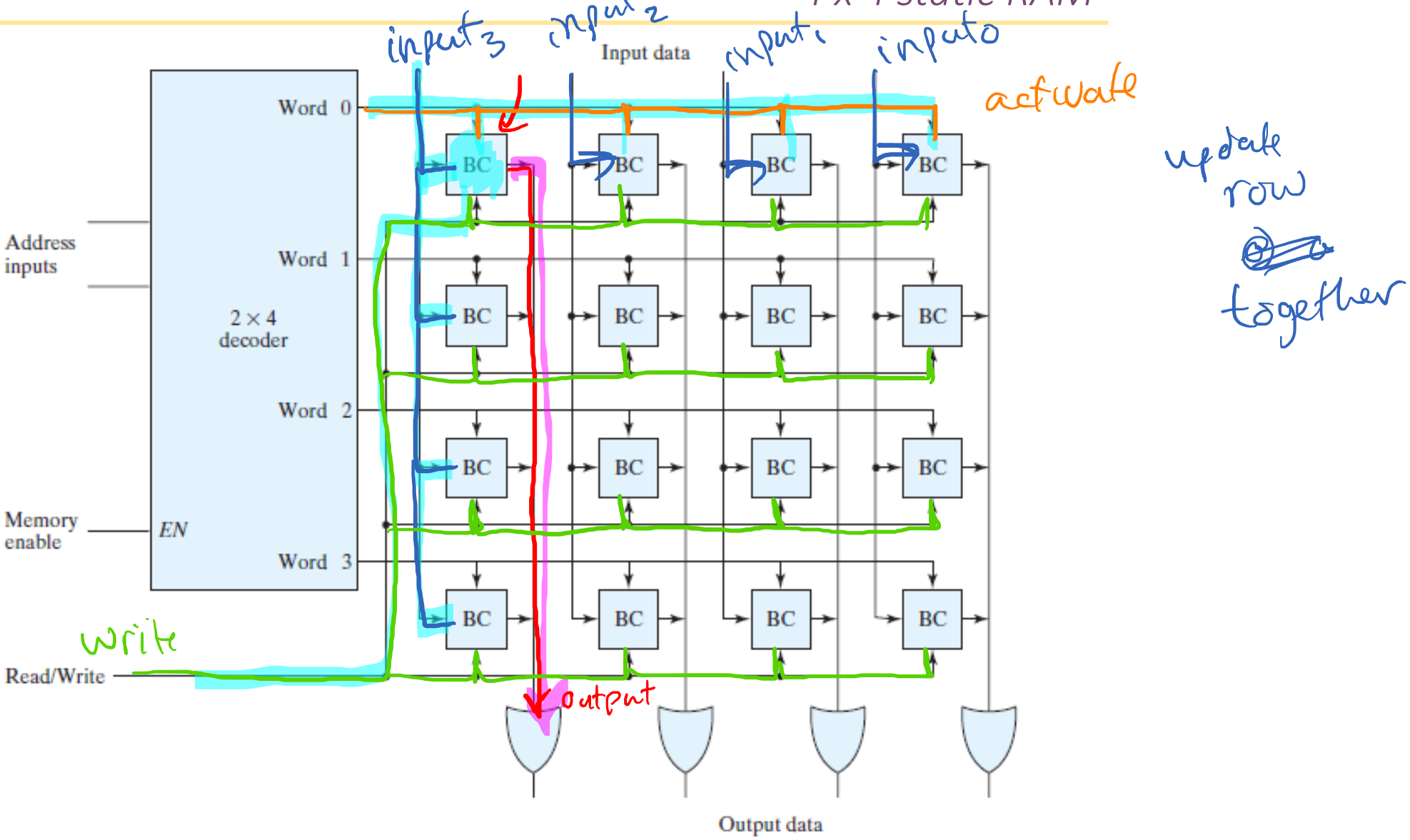
Instead of:



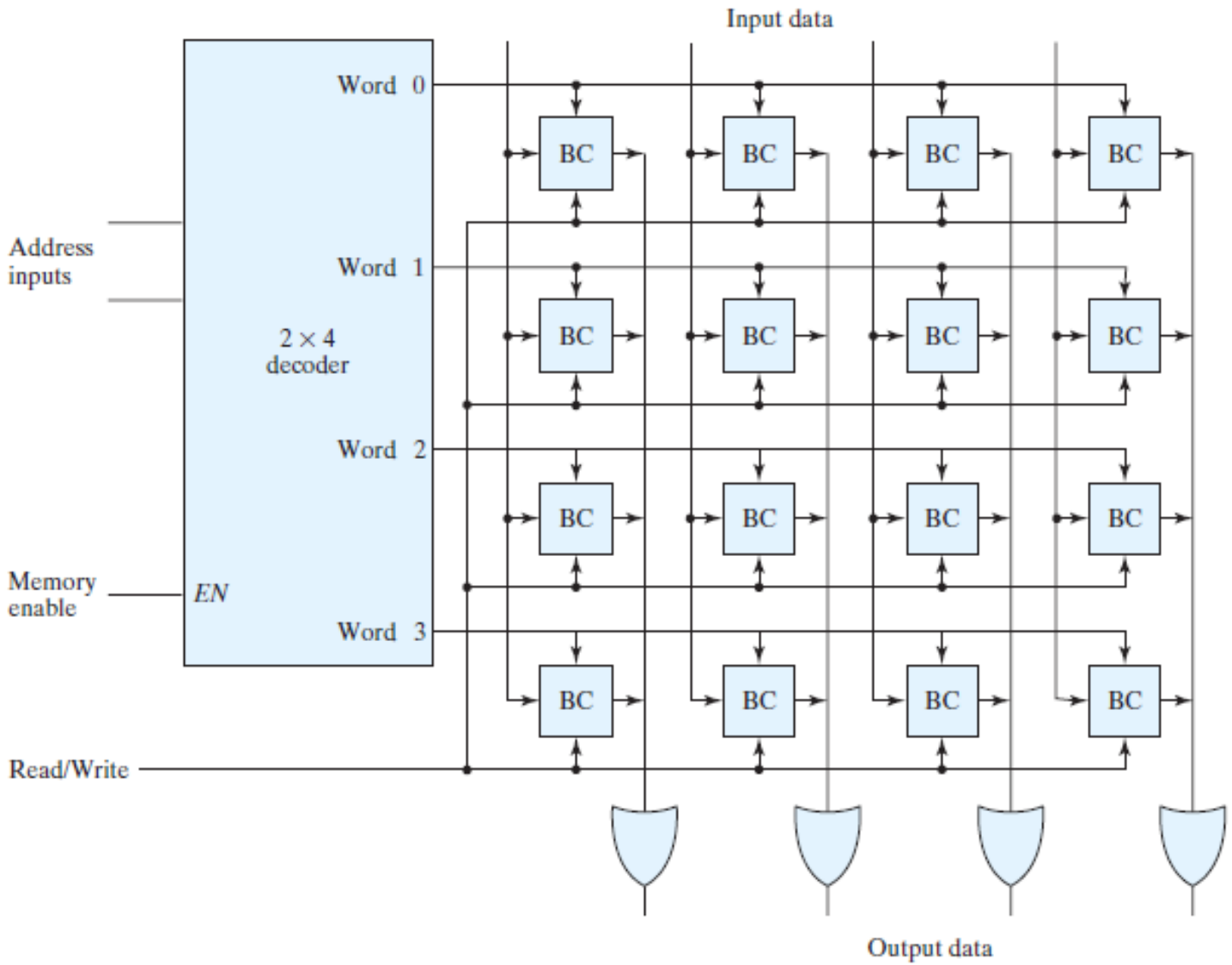
We often use notation:



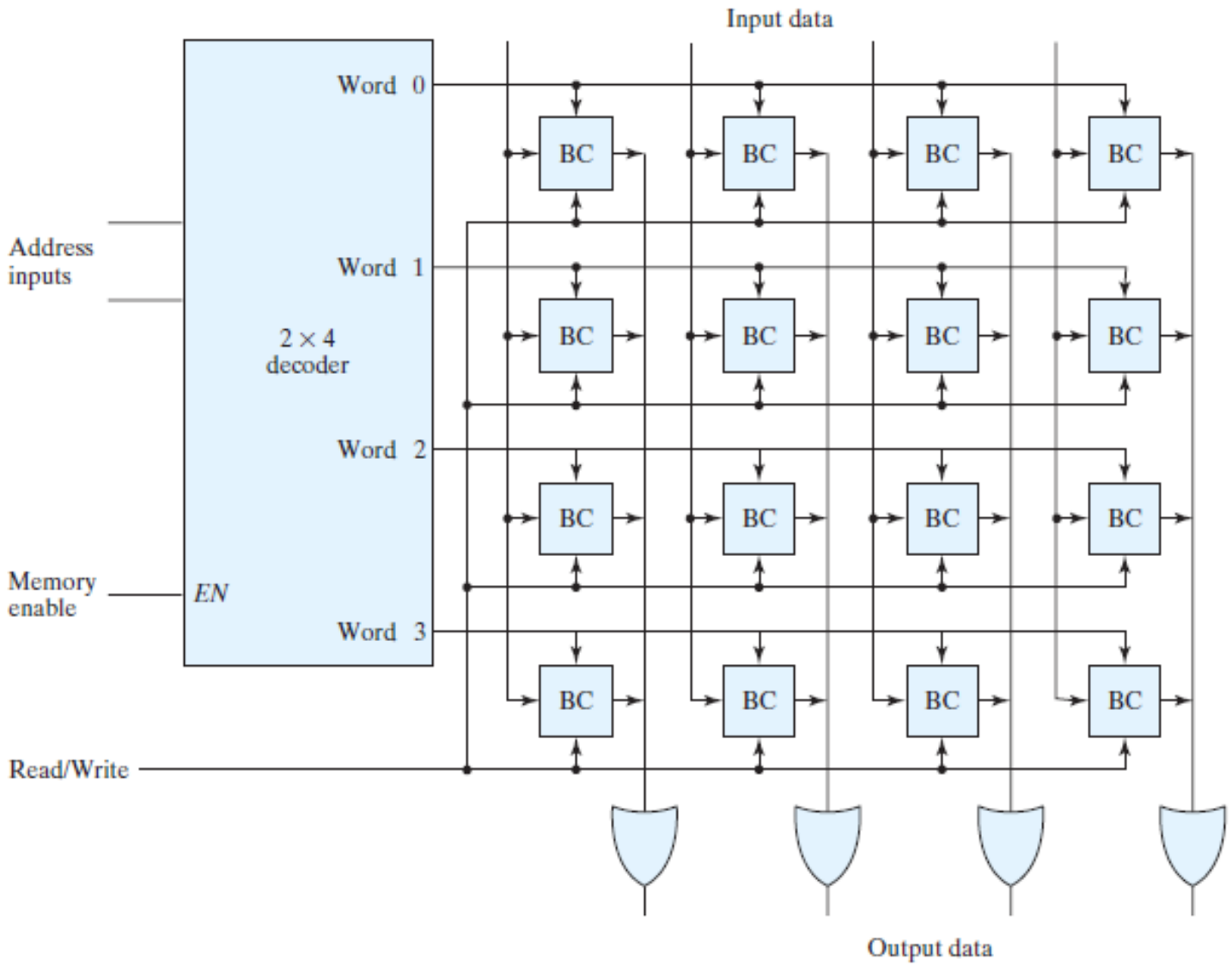
4 x 4 static RAM



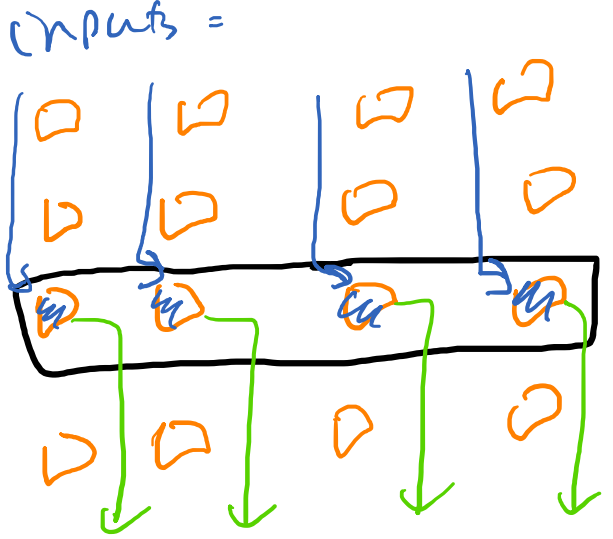
4 x 4 static RAM



4 x 4 static RAM



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
```

write in previous slide

input in previous slide

Flip-Flop RAM in Verilog

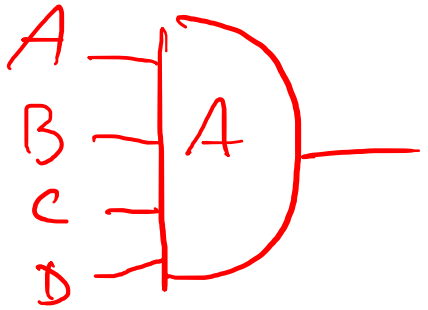
```
module RAM (
  input       clk,
  input [1:0] addr,
  input       set,
  input [3:0]  set_data,
  output [3:0] read_data
)
  logic [3:0] array [0:3]; //2D Array
  always_ff @(posedge clk) begin
       if (set) array[addr] <= set_data;
  end
  assign read_data = array[addr];
endmodule
```

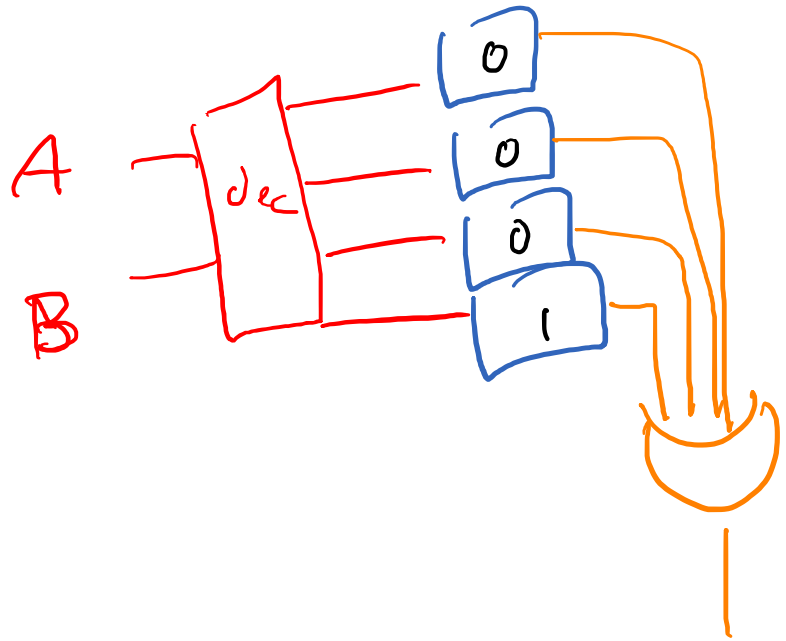
Latch RAM in Verilog

```
module RAM (
  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;
  end
  assign read_data = array[addr];
endmodule
```

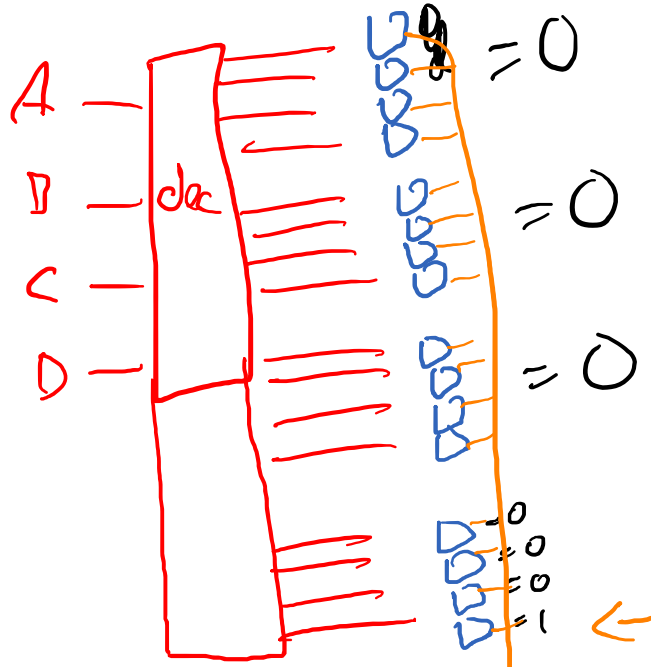
does not need clk

not have default





A & B



A & B & C & D