Memory II

+

Tri-State Busses

Andrew Lukefahr

Topics

- ROMs/RAMs
- Memory + State Machines
- Tri-State Busses

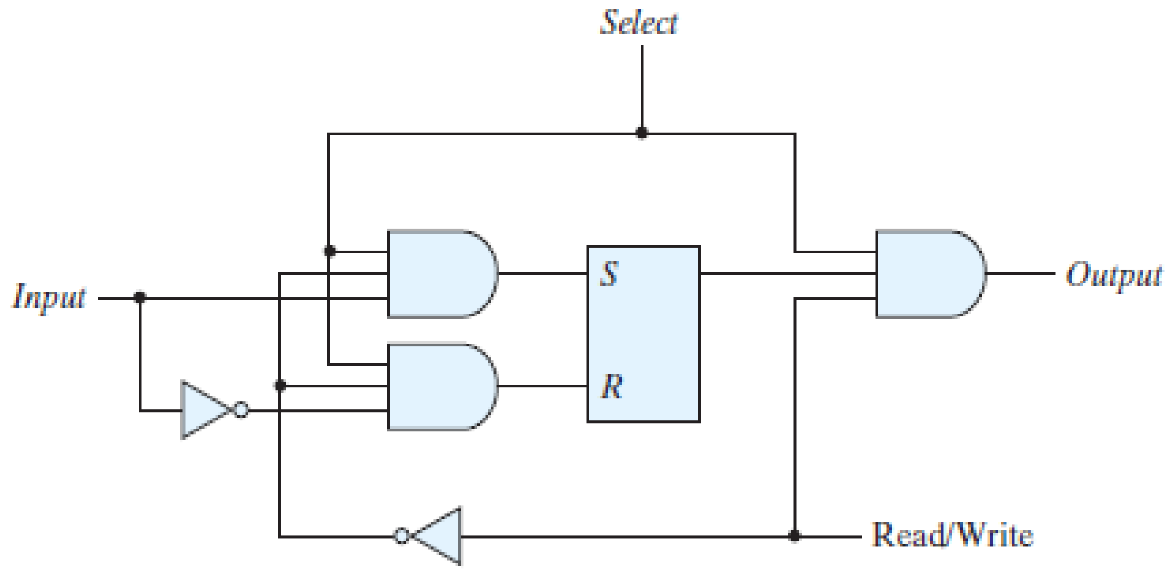
Review: ROM vs RAM

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

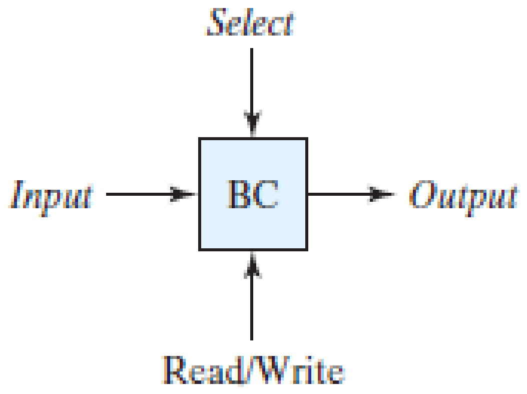
- RAM – Random-Access Memory
    - Read/Write version of a ROM

Review: RAM cell

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



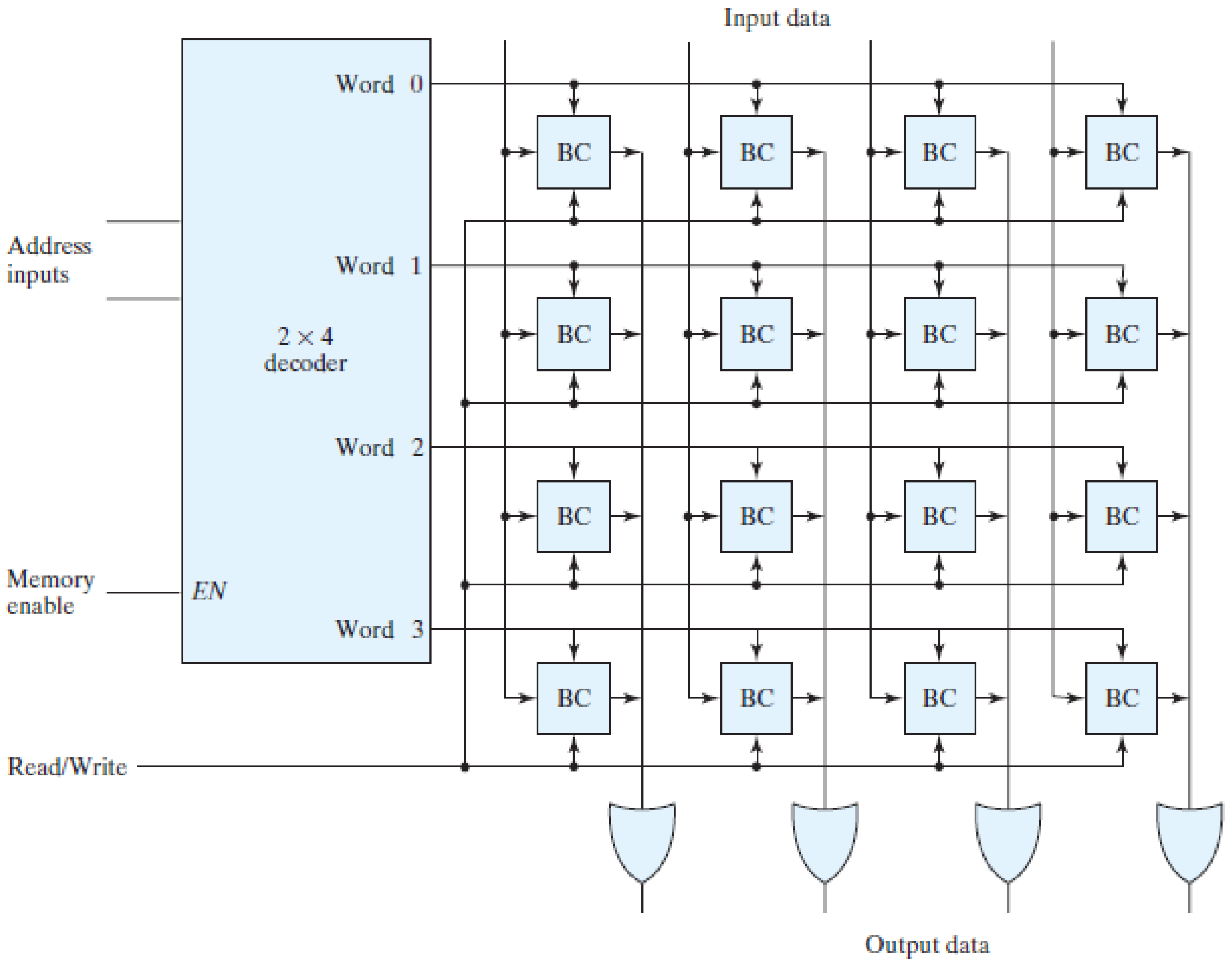
The equivalent logic of a binary cell that stores one bit of information:



Review: RAM

- How many address bits?

- How many data bits?



Aside: SRAM vs DRAM

- SRAM:
  - Static RAM
  - What we've discussed so far
  - Uses full SR Latch to maintain value

- DRAM:
  - Dynamic RAM
  - Charges capacitor to store charge
  - Requires active "refresh" circuit

Aside: SRAM vs DRAM

- SRAM:
  - Uses full SR Latch to maintain value

  + Easier

  - Bigger cells

  - Higher power

- DRAM:
  - Charges capacitor to store charge

  + Smaller cells, higher density

  - Requires sophisticated read and “refresh” circuits

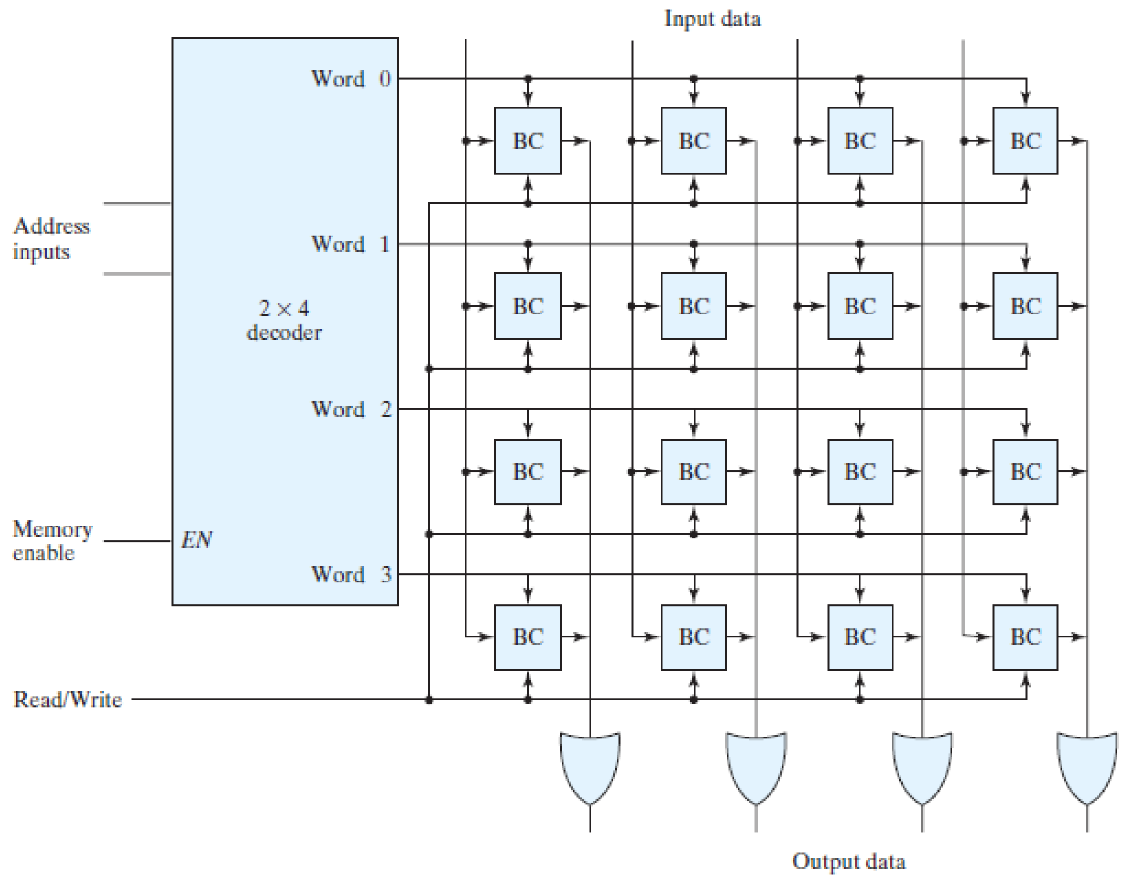
Aside: SRAM vs DRAM

- SRAM:
  - On Exam!

- DRAM:
  - Not on exam!

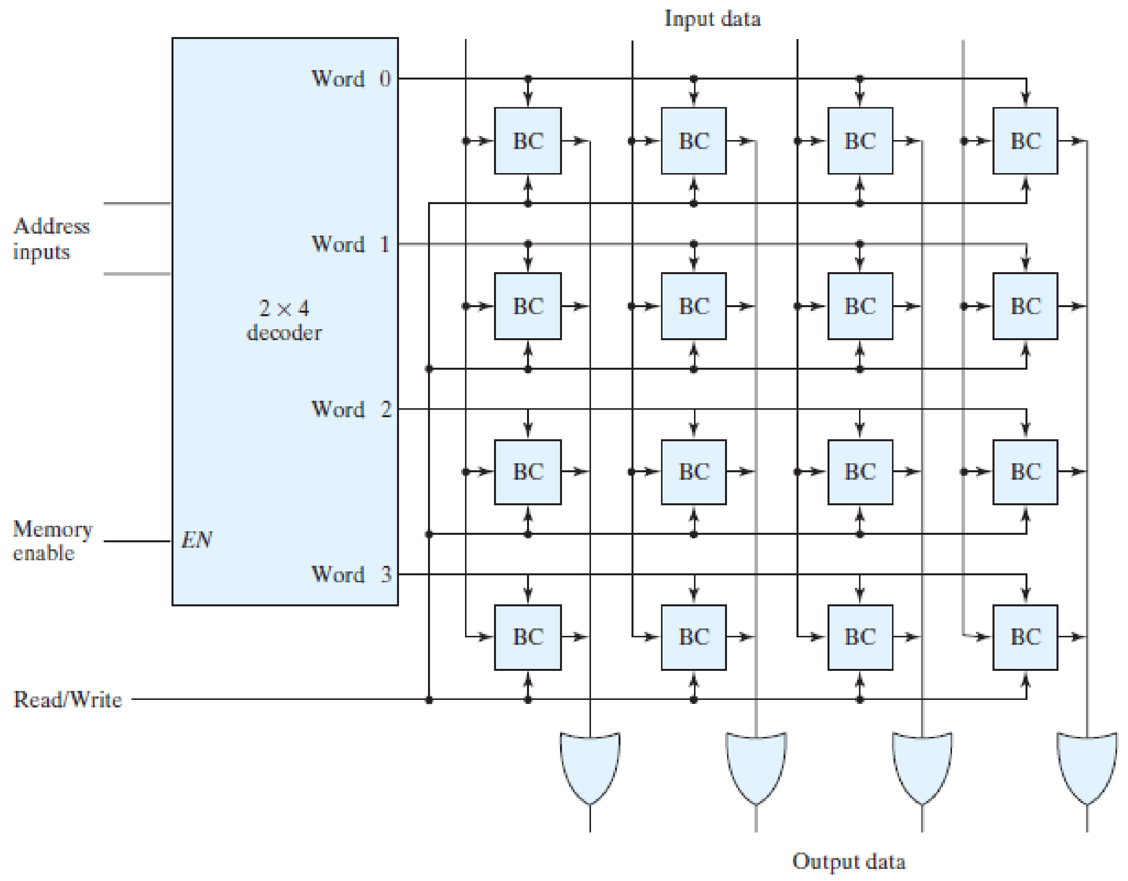
Queues with RAMs

- Given: RAM array (shown)
- Make: 4-element 4-bit **Queue**
  - Recall: First-in-first-out
- Tip: Use a state machine!



Queues with RAMs

- Two queue "functions"
- Enqueue:
    - Adds element to queue
    - enque(4'b XXXX)
- Dequeue:
    - Removes element from queue
    - 4'bXXXX = deque()



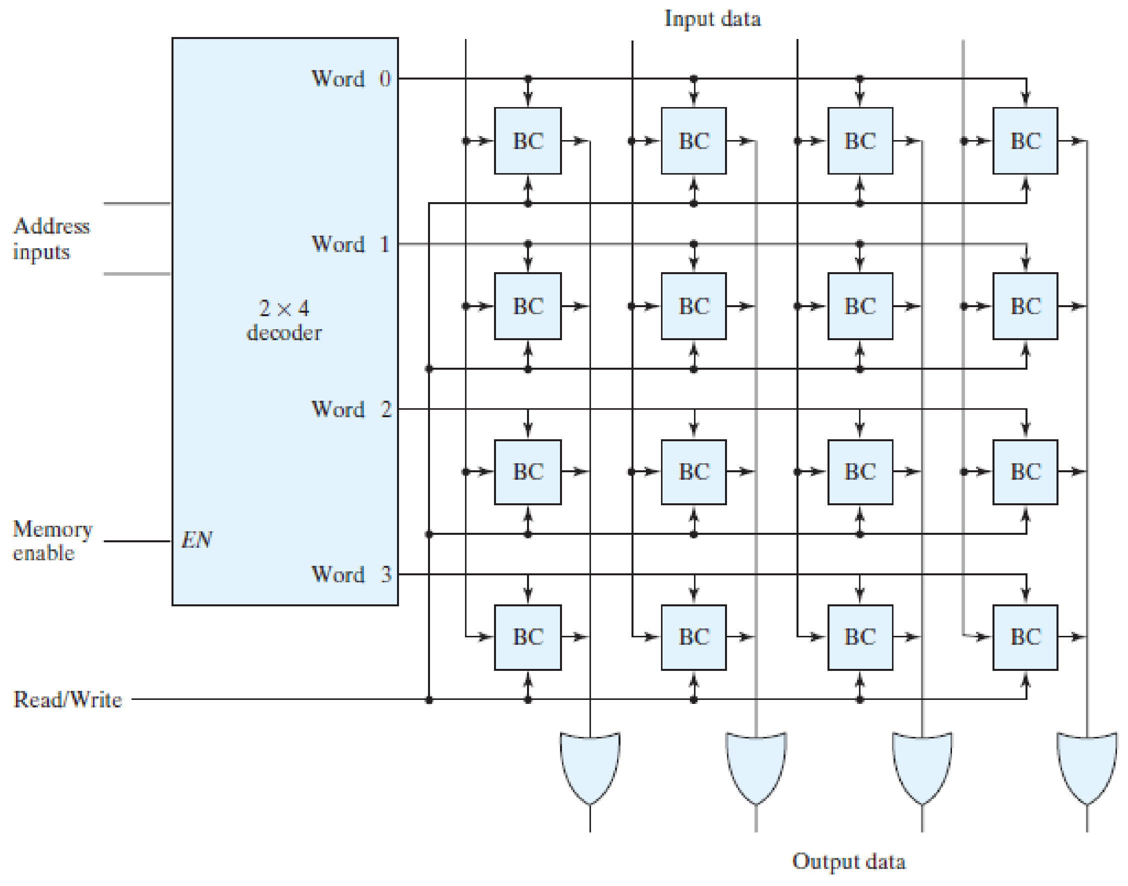
Enque with RAMs

enque(4’b 0001)

enque(4’b 0010)

enque(4’b 0100)

enque(4’b 1000)



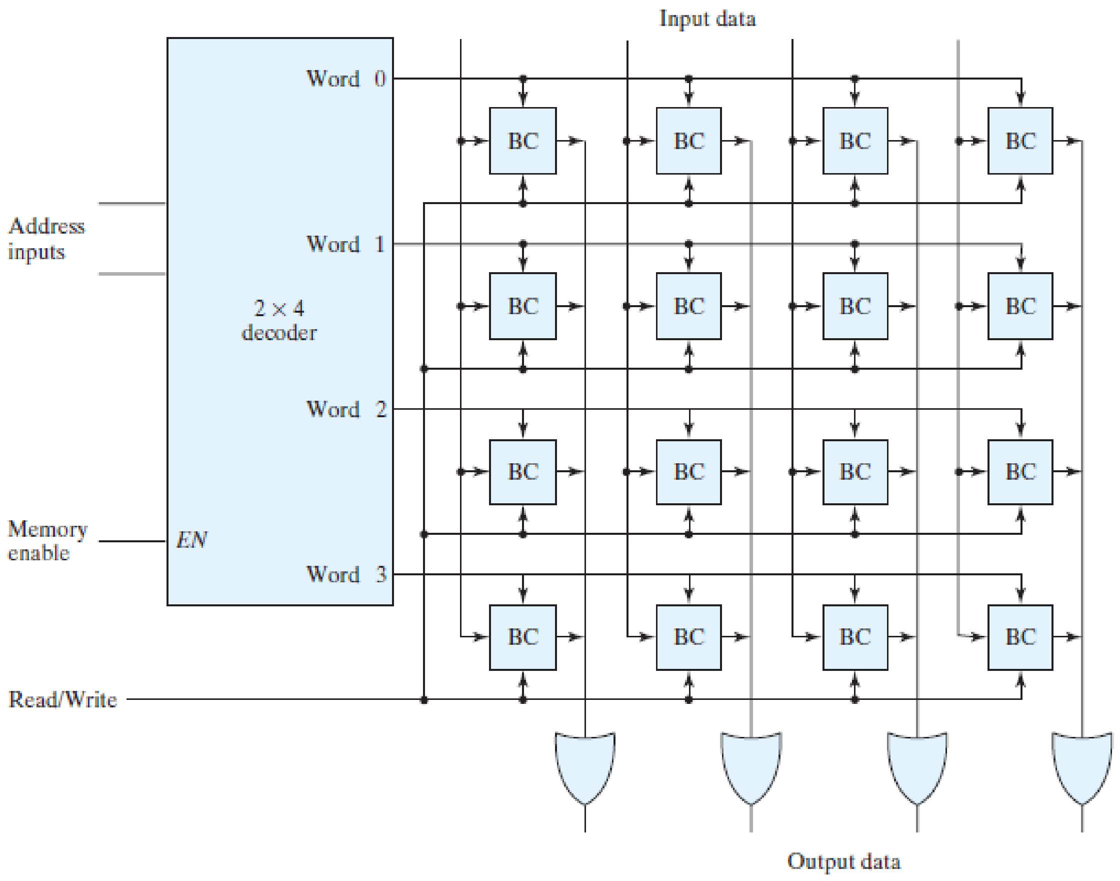
Deque with RAMs

4’b0001 = deque()

4’b0010 = deque()

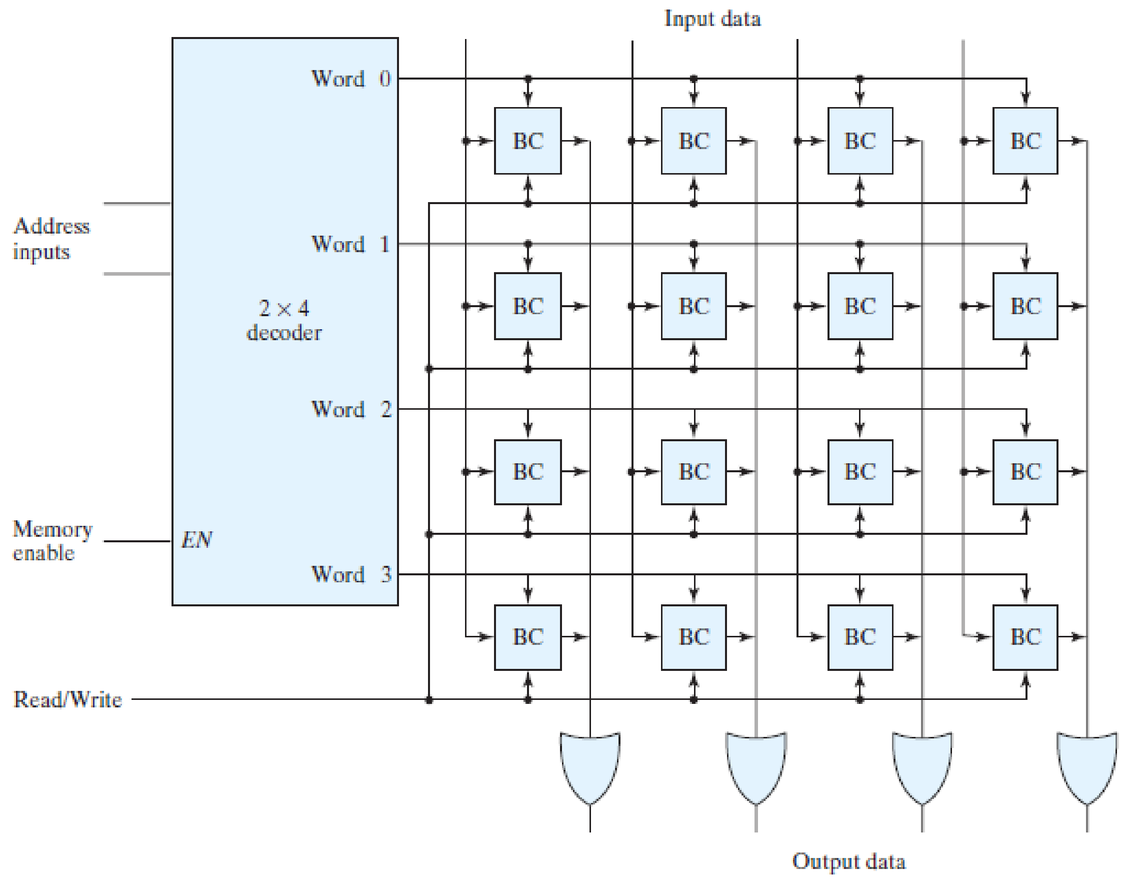
4’b0100 = deque()

4’b1000 = deque()



Enque/Deque with RAMs

enque(4’b 0001)
enque(4’b 0010)
enque(4’b 0100)
4’b0001 = deque()
4’b0010 = deque()
enque(4’b 1000)
enque(4’b 0011)
4’b0100 = deque()
4’b1000 = deque()
enque(4’b 0110)
4’b0011 = deque()



Enque State Machine

- Inputs: `enq_req, [3:0] enq_data`
- To RAM: `mem_en, address, read_writeN,    [3:0] input`

Deque State Machine

- Inputs: `deq_req`
- Outputs: `[3:0] deq_data`
- To RAM: `mem_en, address, read_writeN`
- From RAM: `[3:0] output`

Challenge: Enque/Deque State Machine

- Inputs: `enq_req, deq_req, [3:0] enq_data`
- Outputs: `[3:0] deq_data`
- To RAM: `mem_en, address, read_writeN,    [3:0] input`
- From RAM: `[3:0] output`

Challenge II: Enque/Deque State Machine

- Inputs: `enq_req, deq_req, [3:0] enq_data`
- Outputs: `[3:0] deq_data,` **`enq_err`**`,` **`deq_err`**
- To RAM: `mem_en, address, read_writeN,   [3:0] input`
- From RAM: `[3:0] output`

Challenge III: Enque/Deque State Machine

- Inputs: `enq_req, deq_req, [3:0] enq_data`
- Outputs: `[3:0] deq_data,` **`enq_err`**`,` **`deq_err`**
- To RAM: `mem_en, address, read_writeN, [3:0] input`
- From RAM: `[3:0] output`

- What if `enq_req == 1` AND `deq_req == 1`?

Topic-Shift:

Tri-State Busses

Busses

- Boolean Logic is bi-state:
  - 1: logical true
  - 0: logical false

  - X: The simulation tools don't know if it's 1 or 0

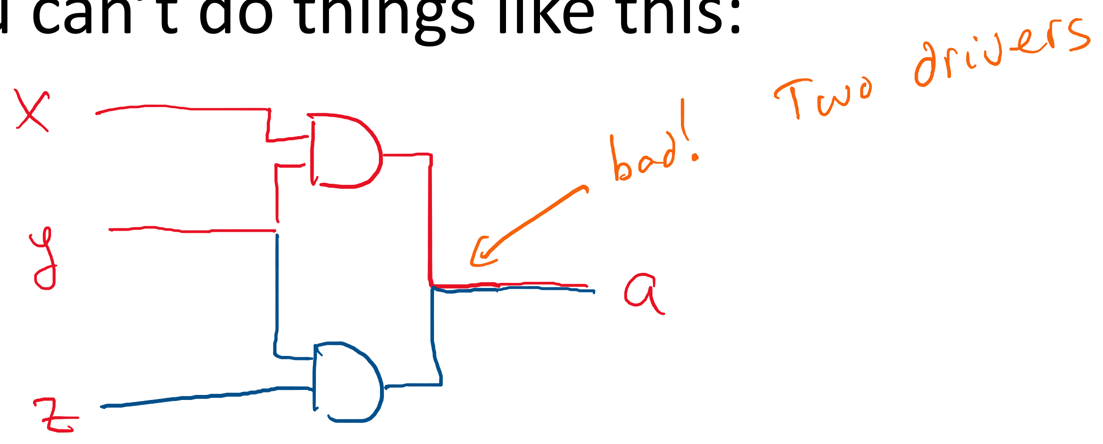
- So you can't do things like this:

Busses

- Boolean Logic is bi-state:
  - 1: logical true
  - 0: logical false

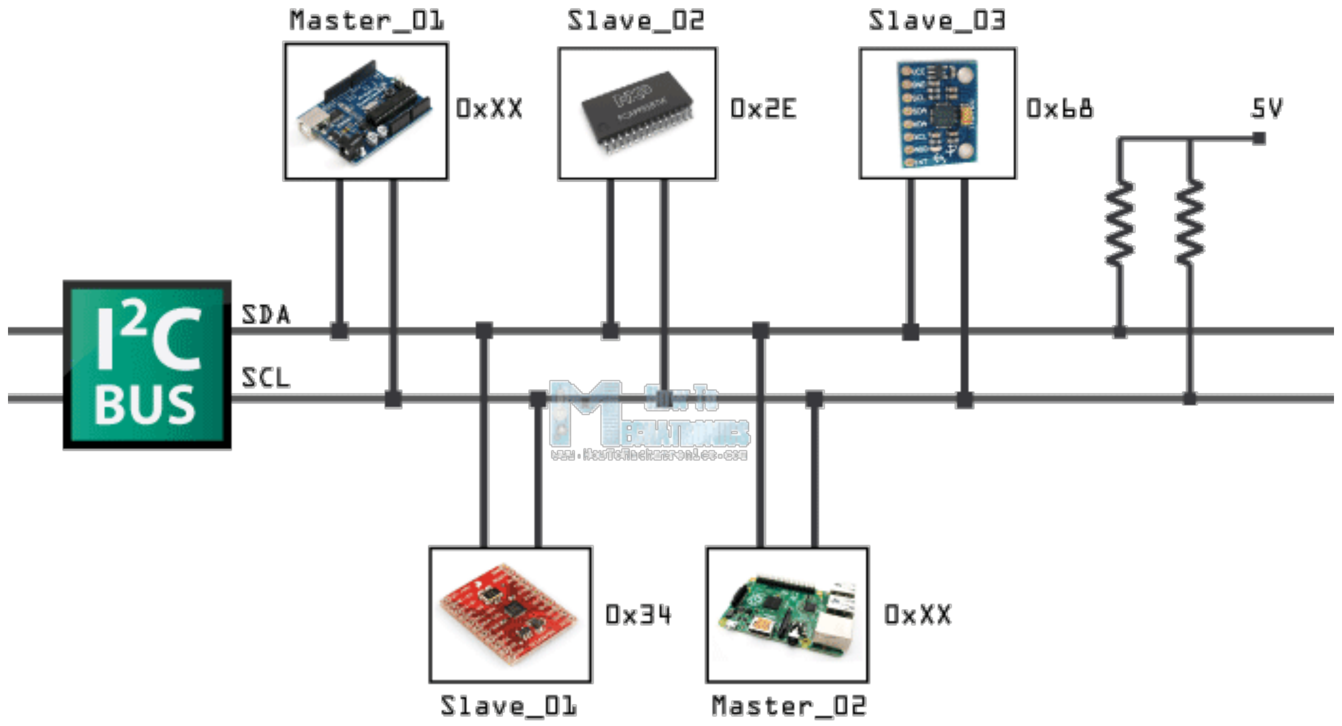
  - X: The simulation tools don't know if it's 1 or 0

- So you can't do things like this:



Question: Then how does I2C work?

- I2C: **Inter** - **Integrated** Circuit **Communication**



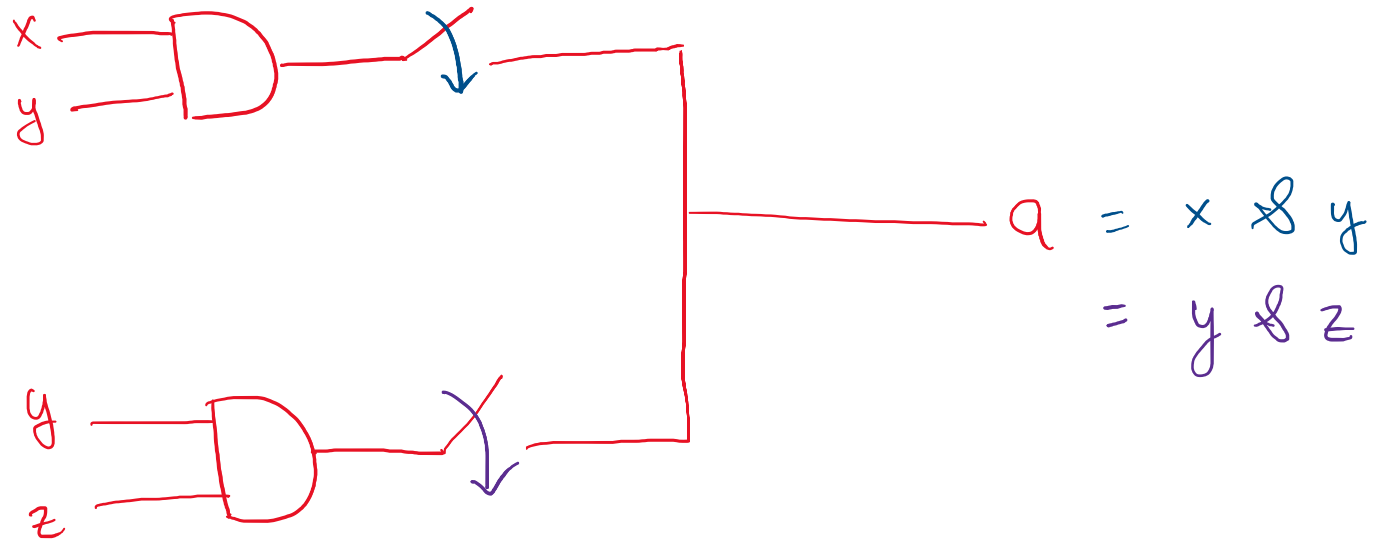
- **Clearly multiple "drivers" for 1 wire?**

Answer: A “Tri-State” Bus

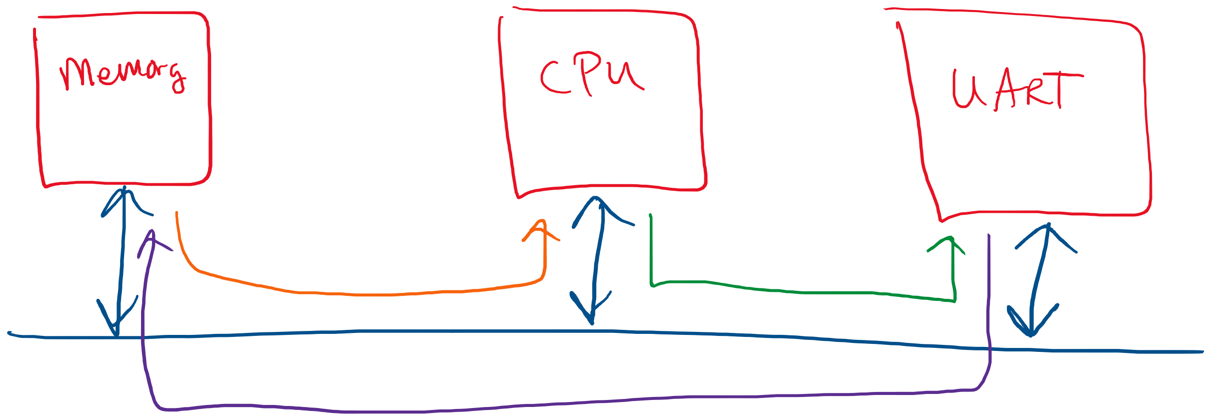
- “Tri-State” signals:
  - 1: this is logical true
  - 0: this is logical false
  - X: The simulation tools don’t know if it’s 1 or 0
  - Z: this is “high impedance”

- Z: High Impedance
  - Stop driving a logical value
  - Pretend I’m not connected

Tri-State logic



Tri-State Bus

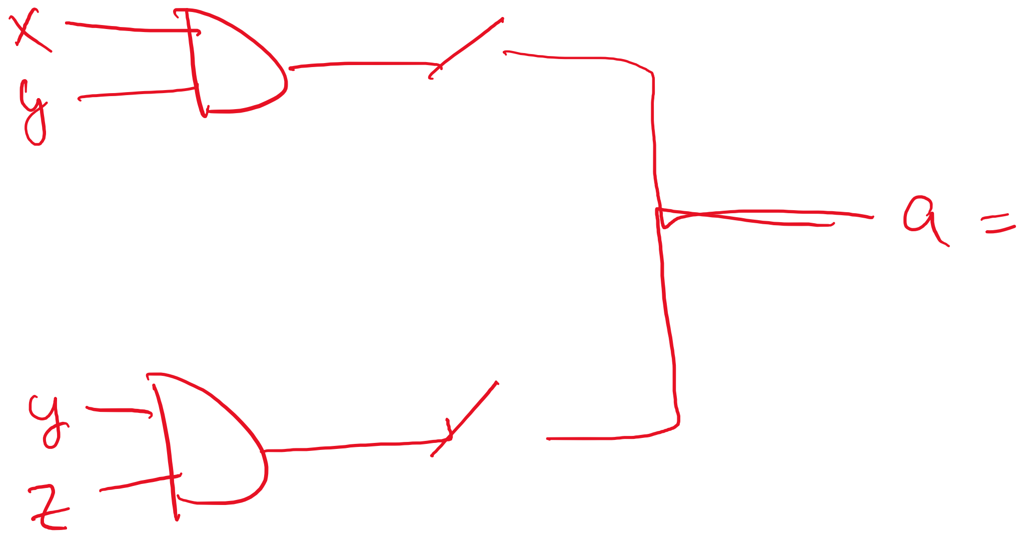


all on same bus

but not at same time

Problems with Tri-State Logic

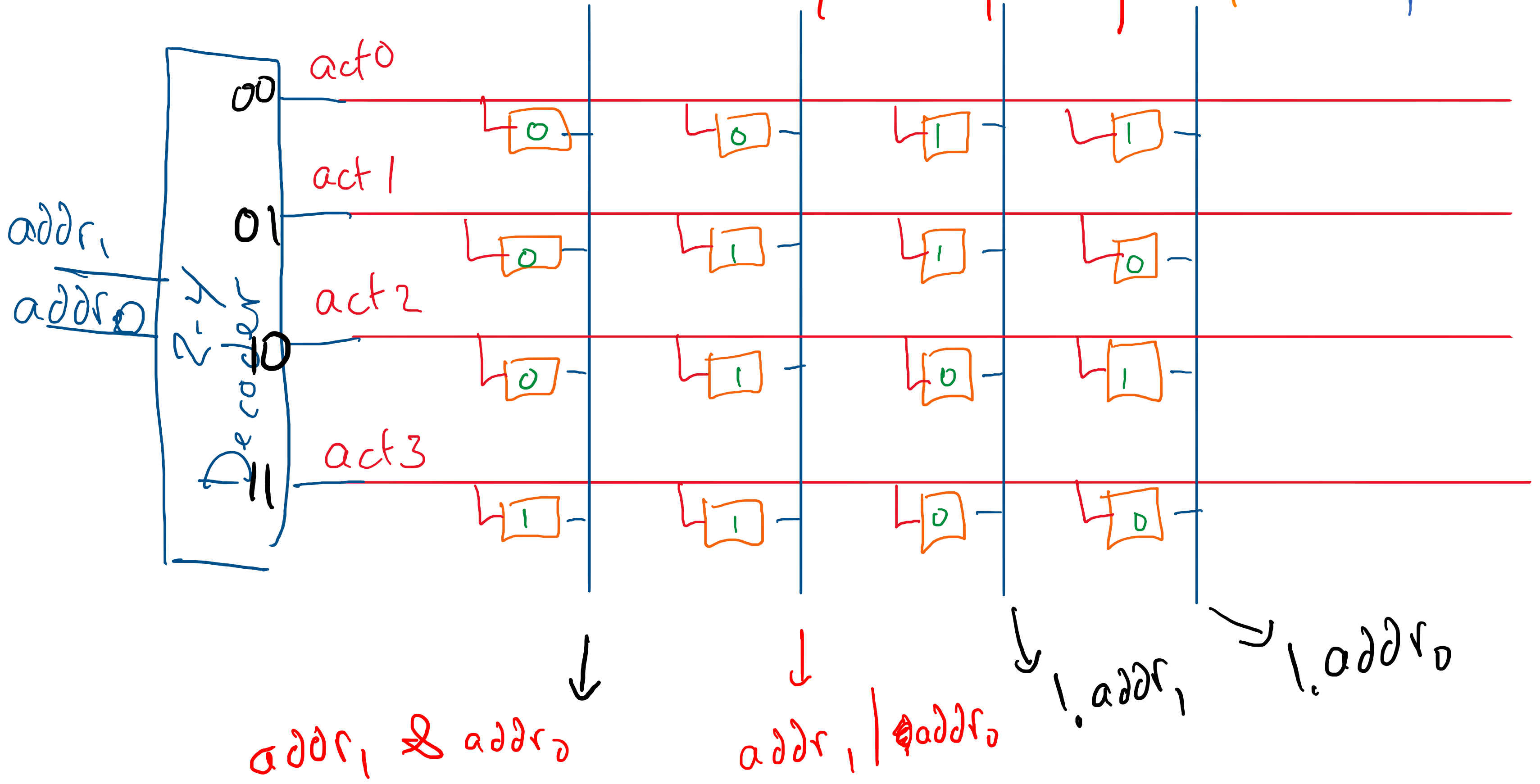
- What if two signals “drive” at once?



Solution: Don’t Do That!

2-bit ROM of AND + OR

$Addr_1$	$Addr_0$	AND	OR	$!Addr_1$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0


$addr_1$
$addr_0$
2:4 Decoder
00
01
10
11
act0
act1
act2
act3
0 0 1 1
0 1 1 0
0 1 0 1
1 1 0 0
$addr_1$ & $addr_0$
$addr_1$ | $addr_0$
$!addr_1$
$!addr_0$