P2 Digital Electronics

Lecture 5: Integration of digital logic components: Multiplexers, ROMs and PLAs

Mark Cannon

mark.cannon@eng.ox.ac.uk

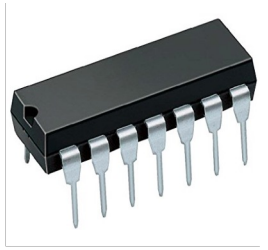
Trinity Term 2026

Overview of lectures

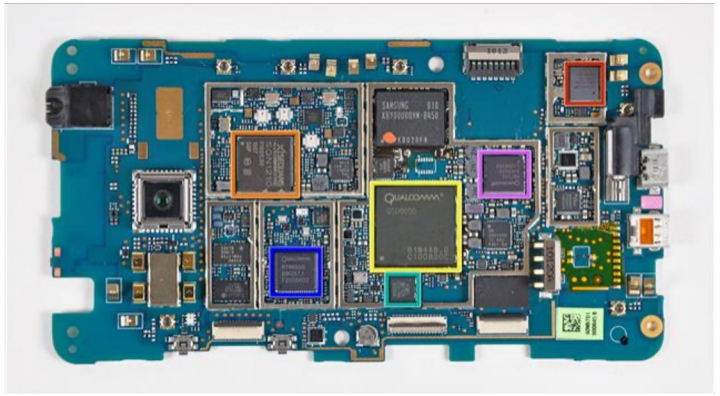
1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. **Integration of digital logic components**
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

From transistor to large scale integration



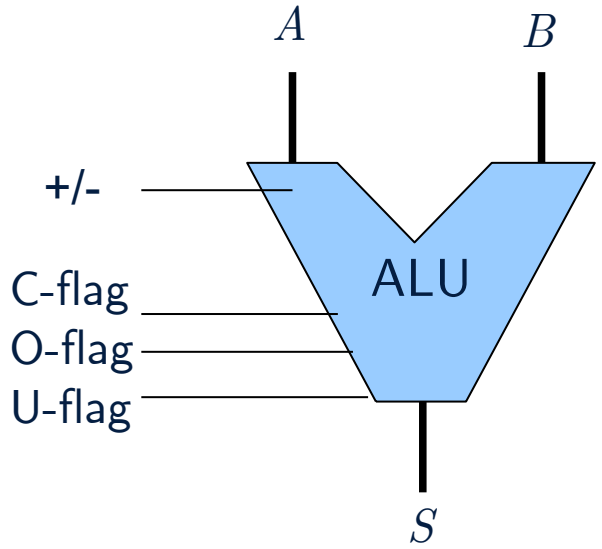
- Transistor
- Logic Gate
- SSI – Small Scale Integration
  chips with a few 10s of transistors
- MSI – Medium Scale Integration
  chips with ~ 100 of transistors
- LSI – Large Scale Integration
  chips with $\sim 10^4$ transistors
- VLSI/ULSI - Very Large Scale Integration
  chips with $\sim 10^9$ transistors

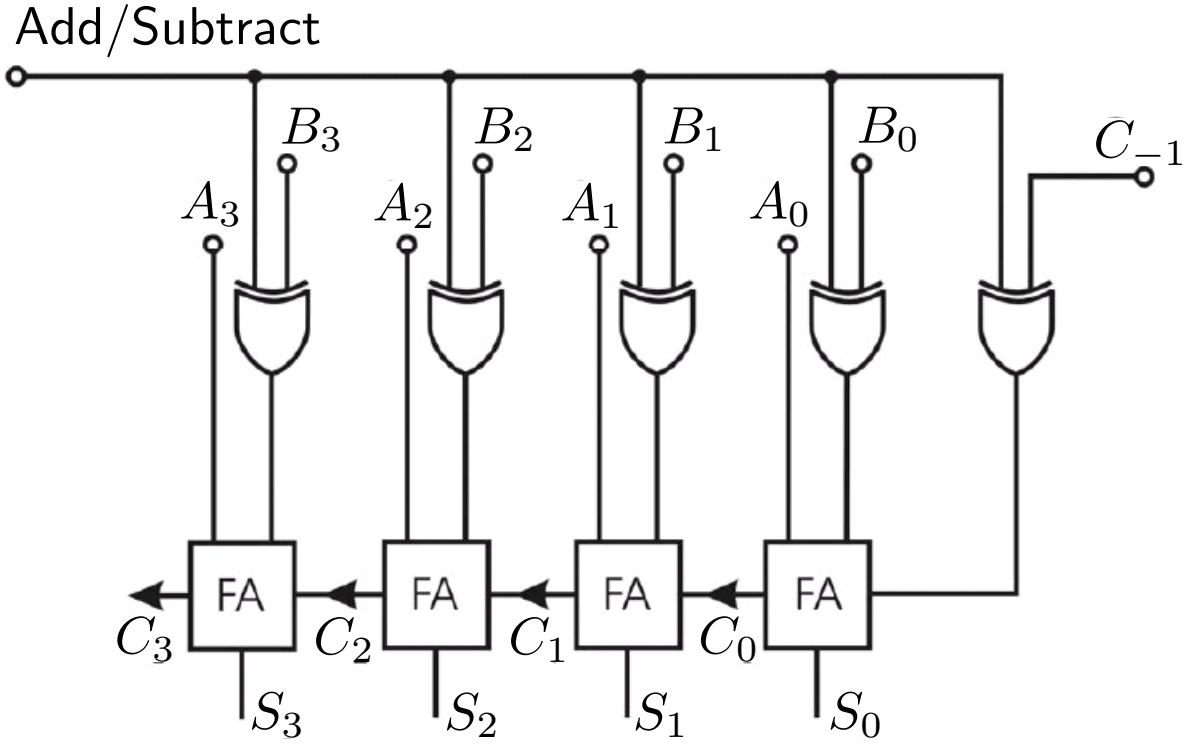


Arithmetic Logic Unit

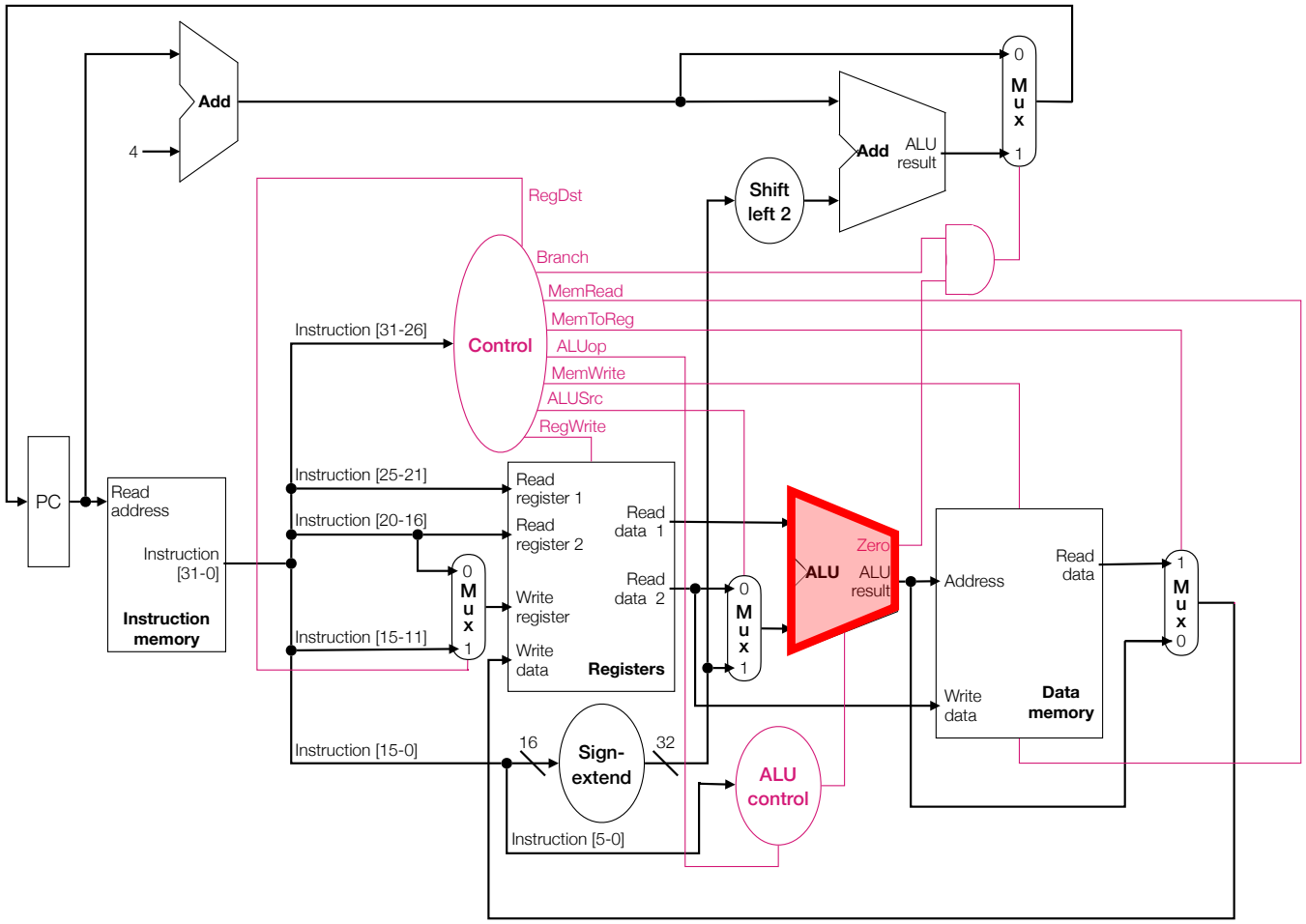
The part of a CPU that does numerical calculations

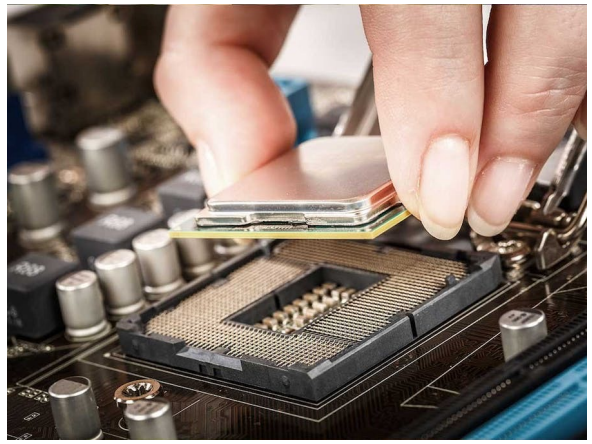
- Contains a full ripple adder (8-bit, 16-bit etc)
- Driven by control lines ($\pm$)
- Two input numbers
- One output
- Flags indicate fault conditions
  - Carry-out
  - Overflow
  - Underflow



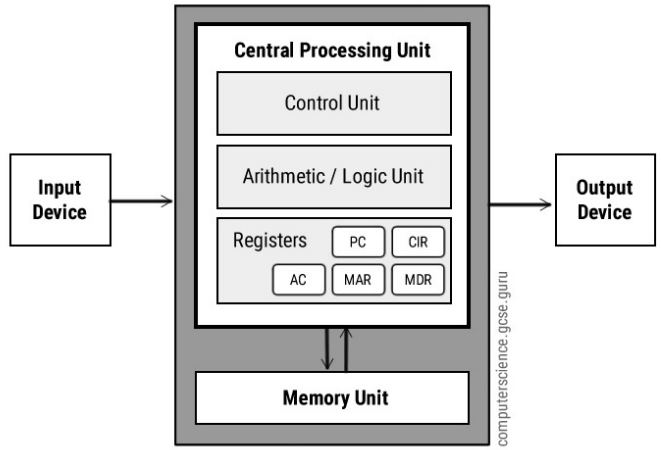


A Simple Central Processing Unit (CPU)





Von Neumann architecture

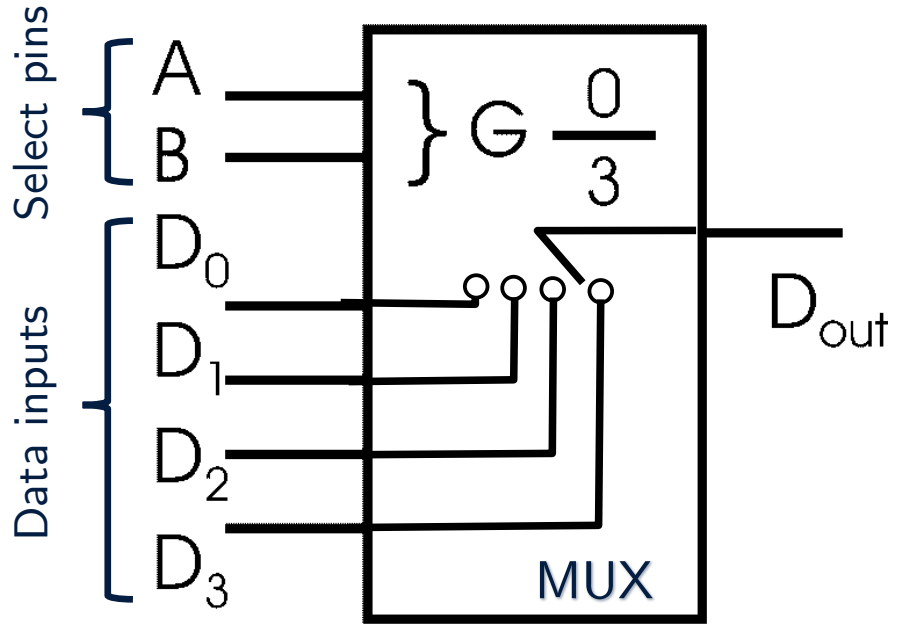


- Details of the CPU will be covered next year in A2 (Computer Architecture)
- This lecture covers essential concepts and components that help us build the CPU

Outline of lecture 5

- Electronic integration
- Multiplexers (MUX)
- Bus and tri-state connections
- Read only Memory (ROM)
  - Decoder
  - Use of ROMs
- Programmable logic array (PLA)

Multiplexers

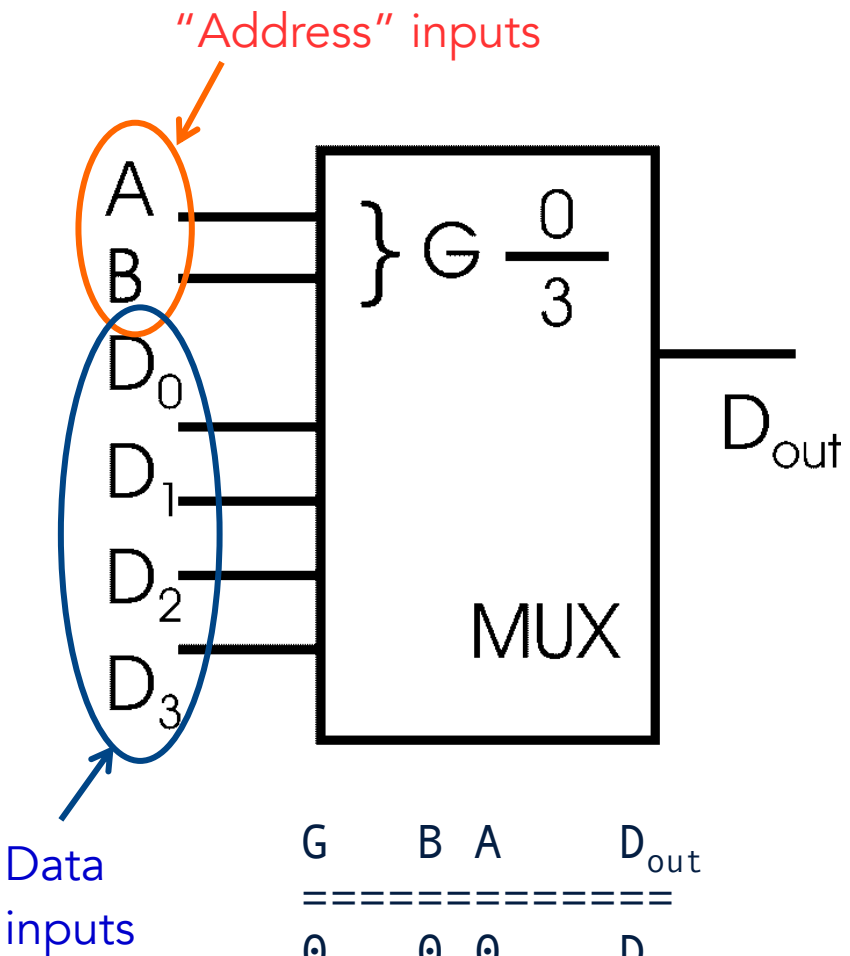


A multiplexer is a digital switch

It selects which data stream or bit is sent to the next section of a circuit

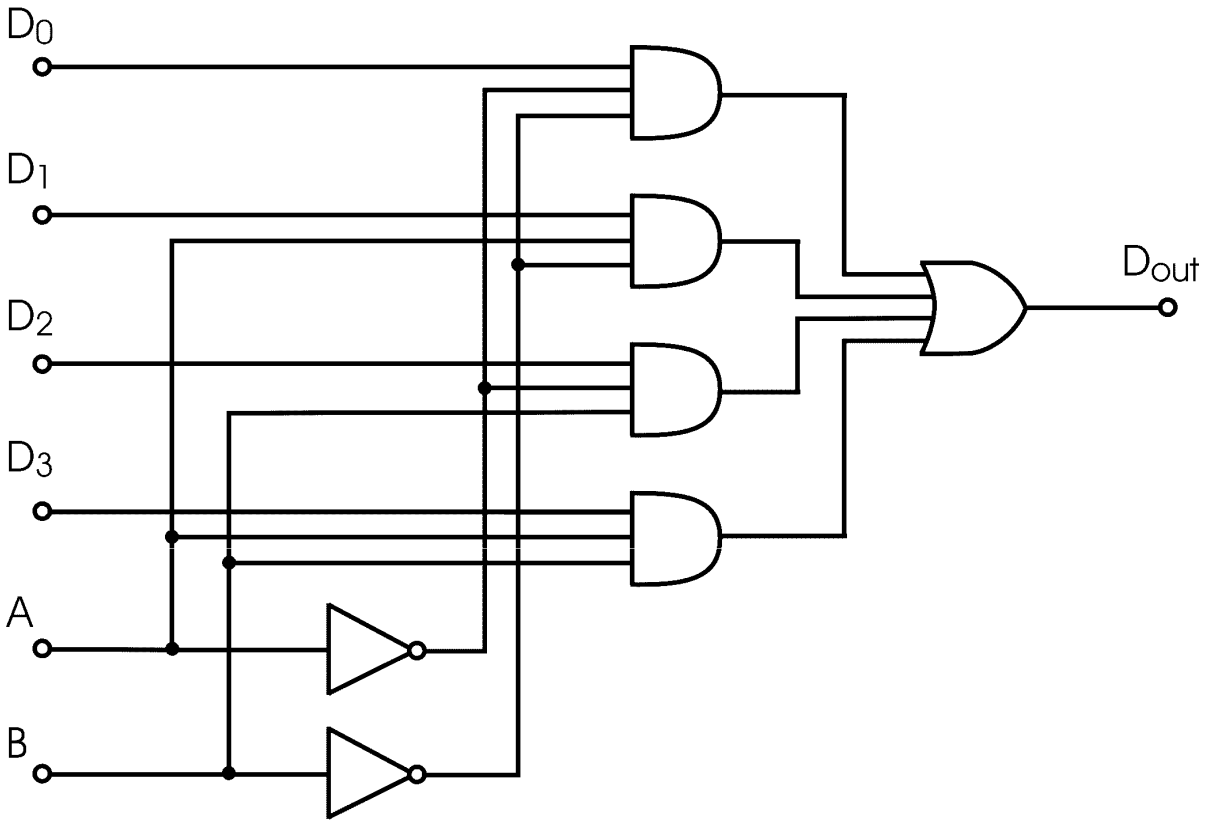
Can be used to perform simple logic programming

MUX logic

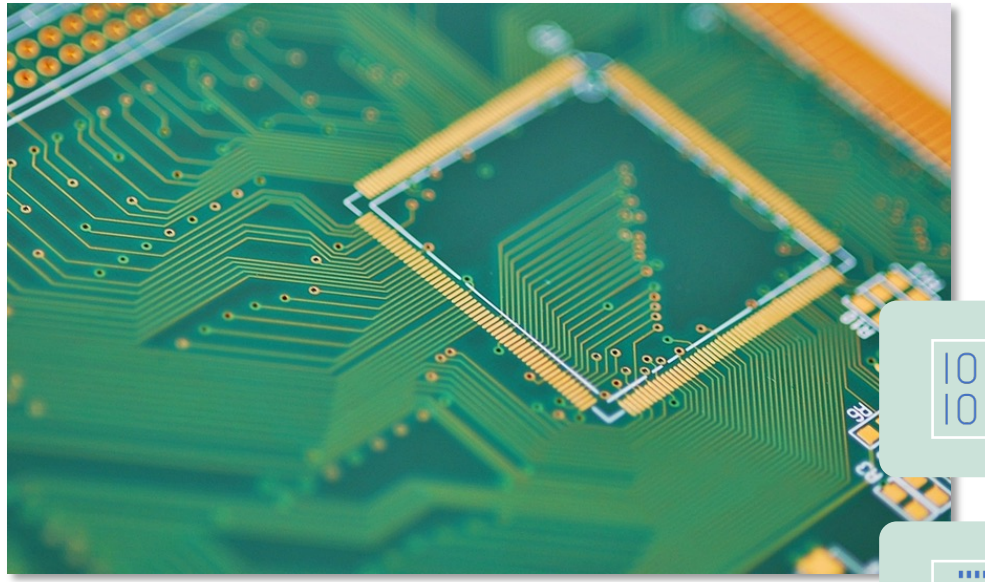


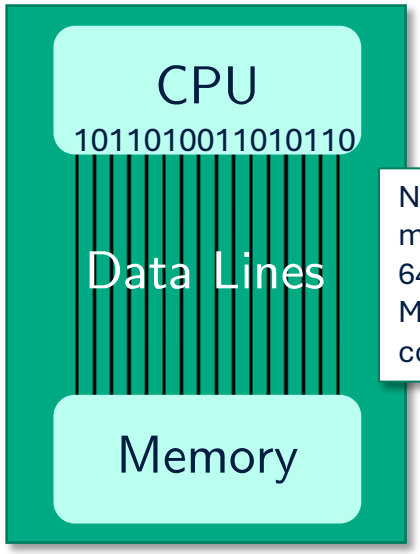
G	B	A	D_{out}
0	0	0	D_0
1	0	1	D_1
2	1	0	D_2
3	1	1	D_3

$$D_{out} = \overline{A}.\overline{B}.D_0 + A.\overline{B}.D_1 + \overline{A}.B.D_2 + A.B.D_3$$



Buses and tri-state outputs





Not very scalable, modern architectures are 64 bits!
Multiple peripherals are connected to the CPU

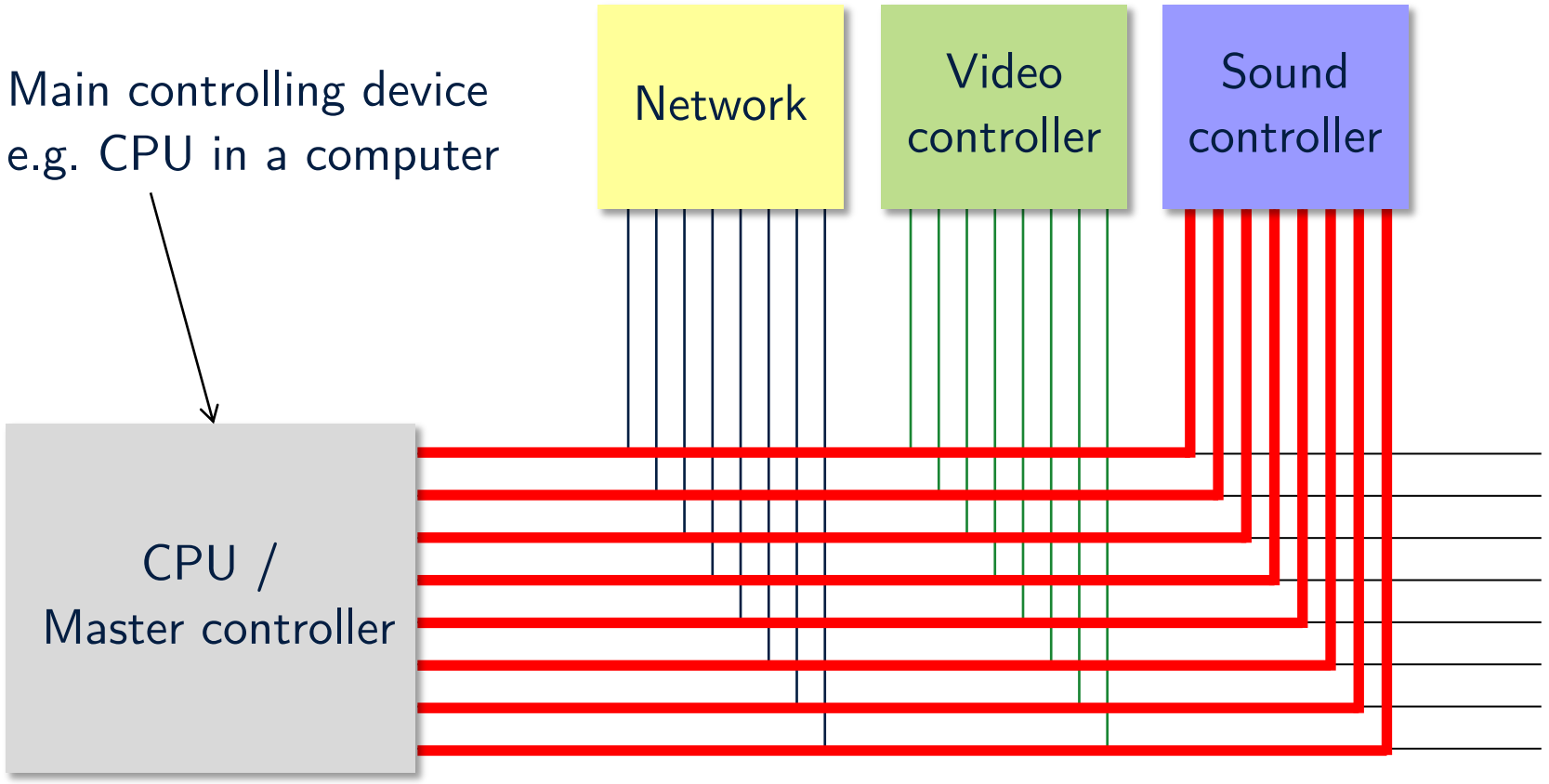
1010
1010

Integrated circuits often have a very large number of inputs and outputs.

Many digital lines uses up a lot of printed circuit board space.

Parallel data lines called **buses** are used to transport digital data around a system

Bus architecture

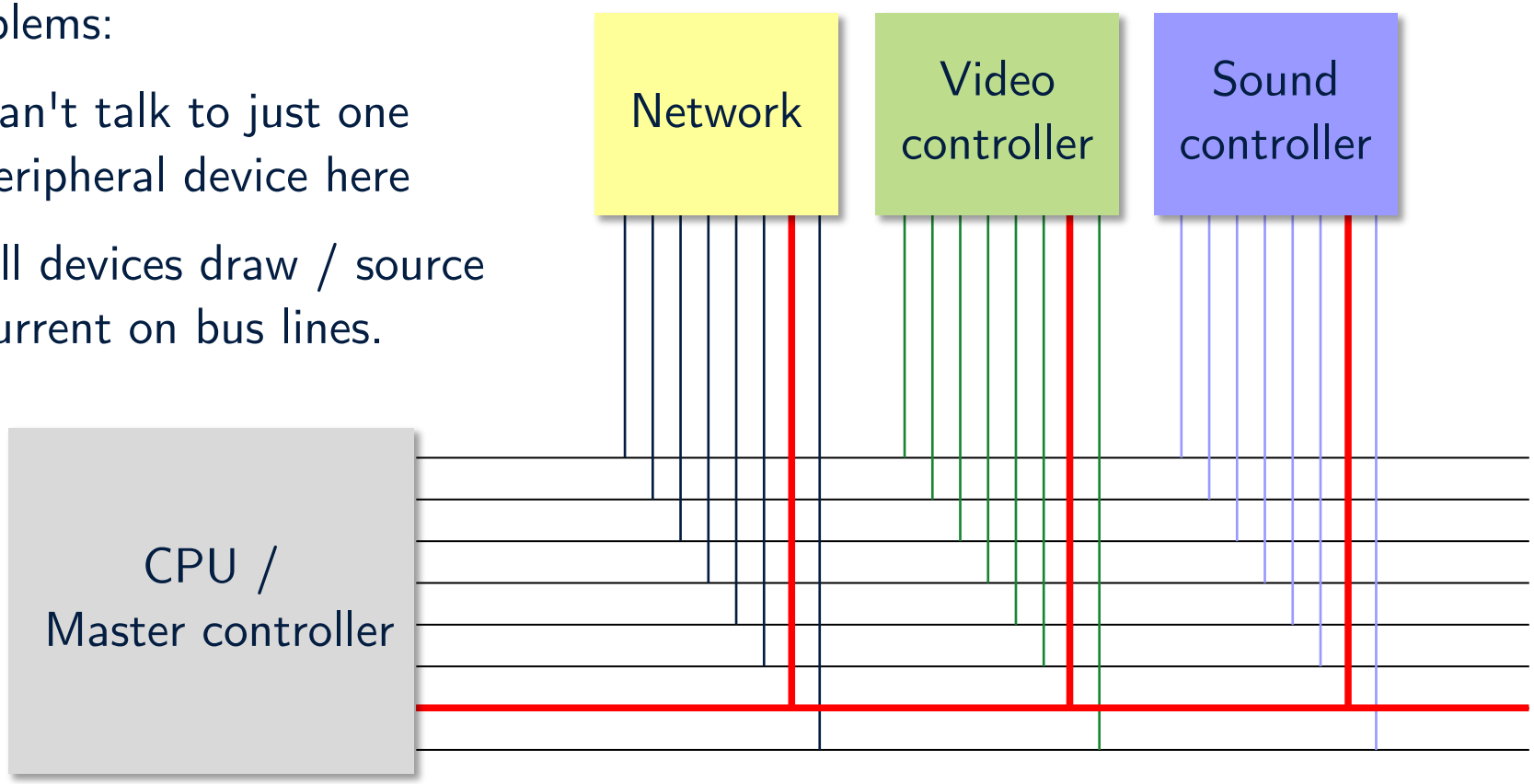


Bus lines (wires) shared between all peripherals
as many as 64 or more in many cases

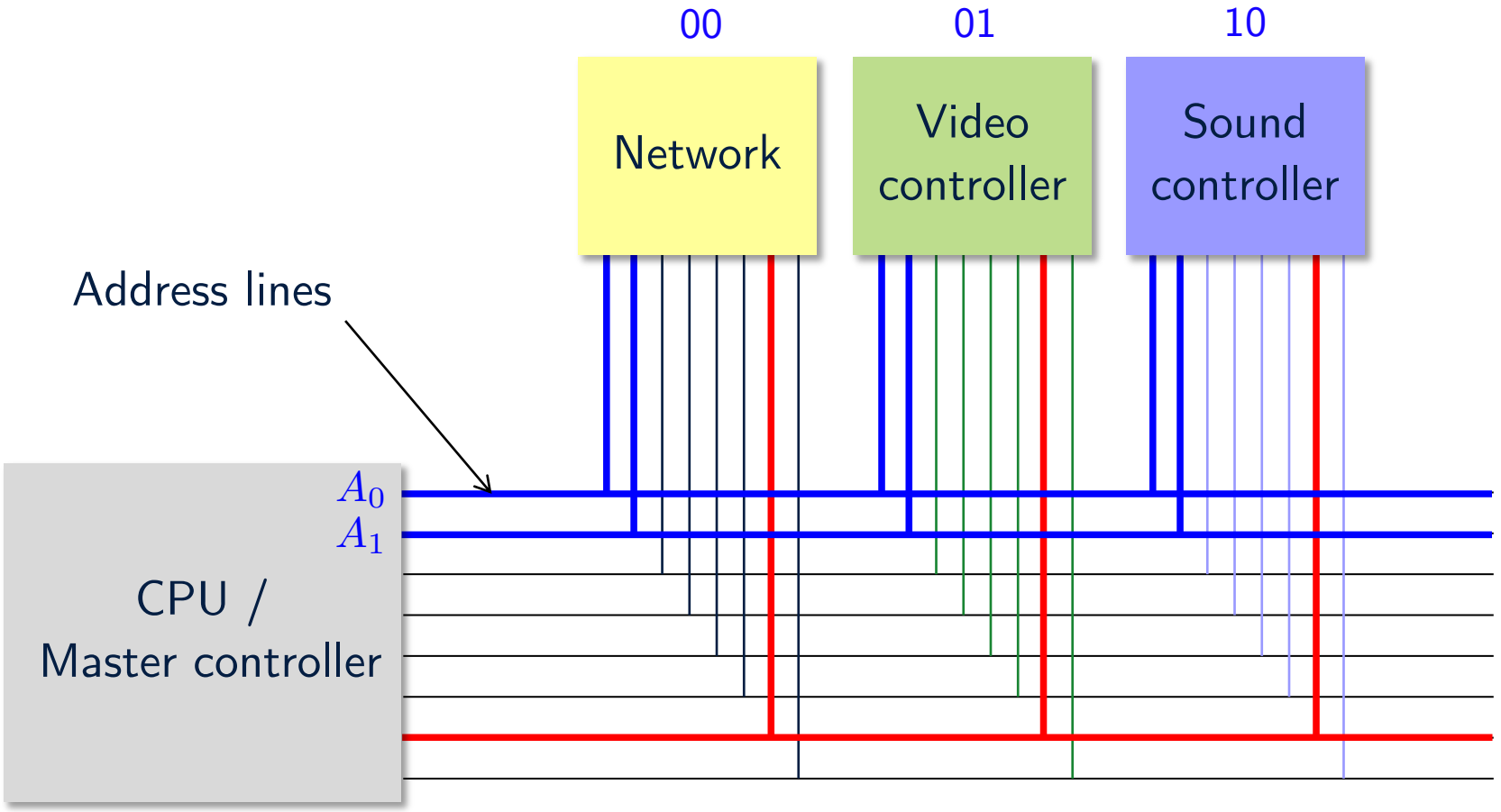
Bus conflicts

Problems:

- Can't talk to just one peripheral device here
- All devices draw / source current on bus lines.



Bus conflicts



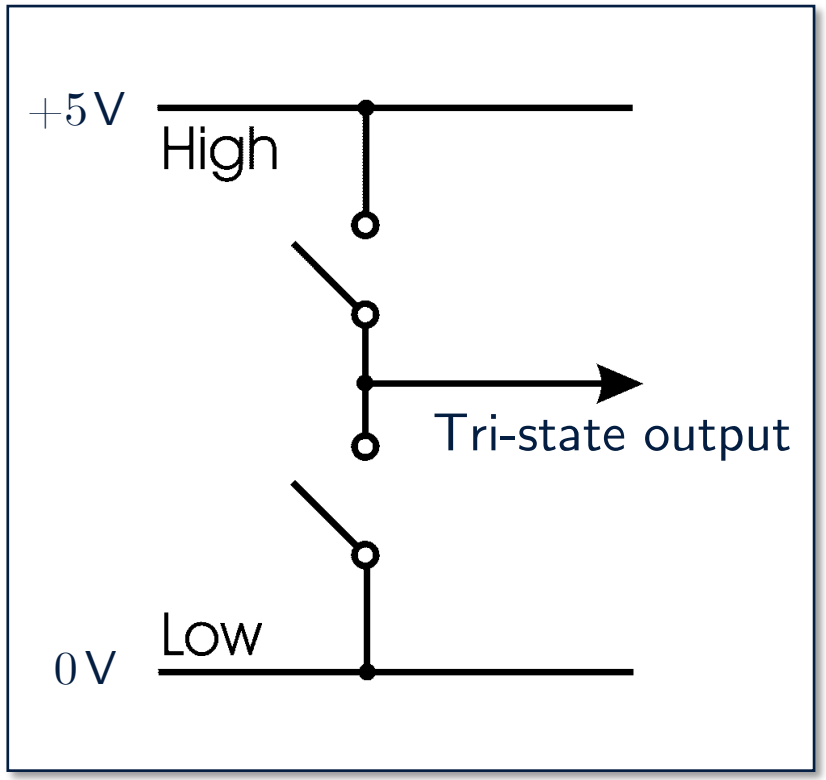
Problem: Although we can now select which peripheral to talk to,
the others are still connected and can draw current from bus lines and confuse data levels...

Solution: ensure unwanted peripherals 'disconnect' from bus lines

Tri-state outputs

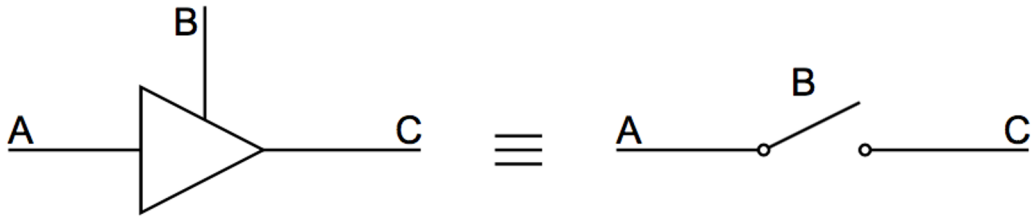
Add an extra logic state to outputs:

- ⋆ High (1)
- ⋆ Low (0)
- ⋆ High impedance (draws no current)



⇓

Allows outputs to become isolated from bus lines



Tri state symbol

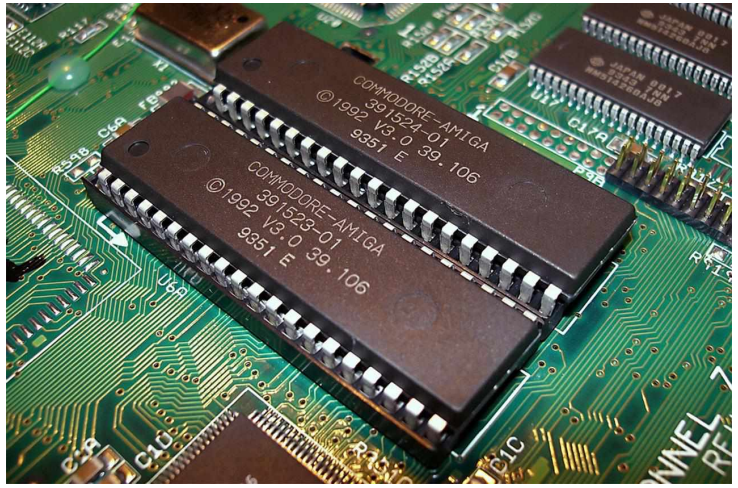
Chip select and output enable

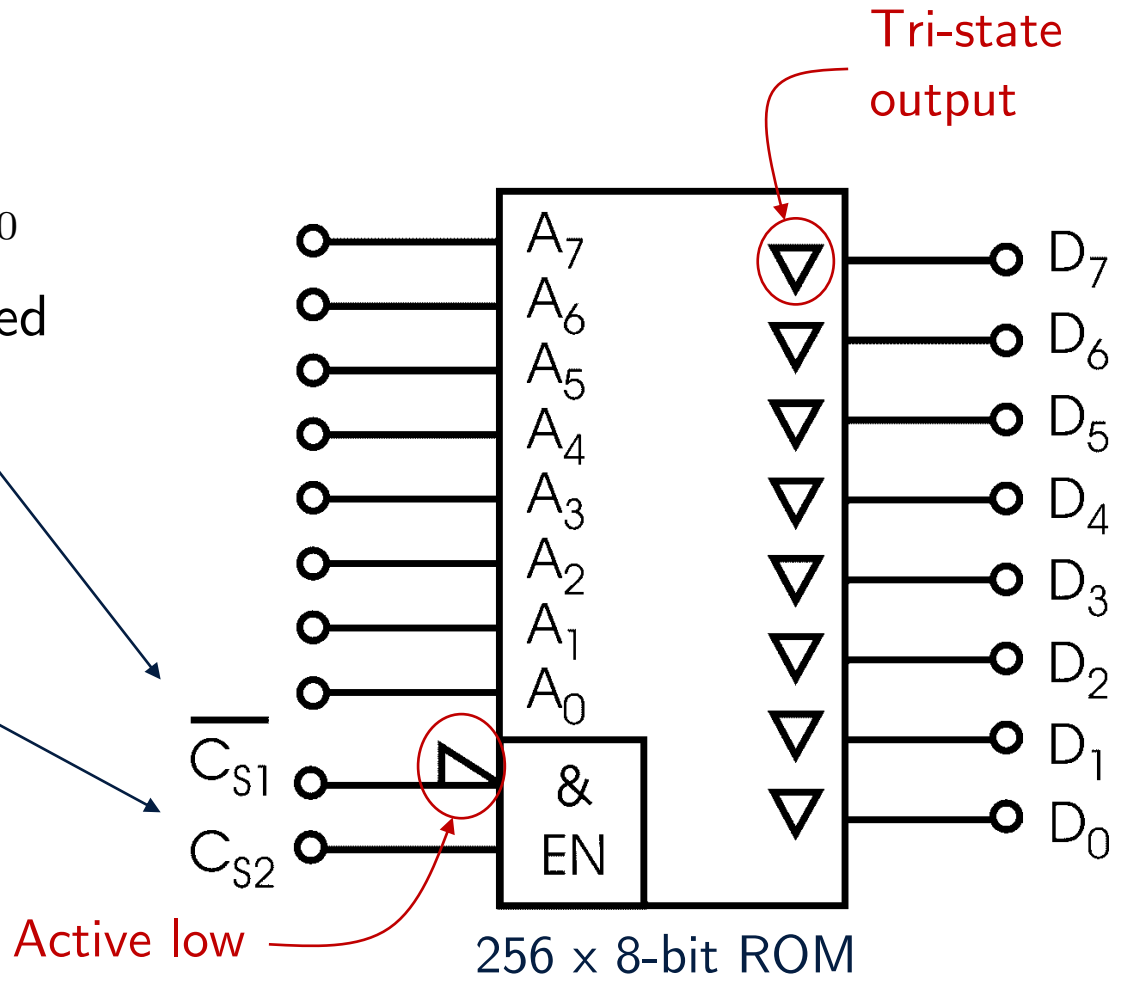
Chip select – switches on chip inputs

- ⋆ makes ROM read input address $A_7 \ldots A_0$
- ⋆ address lines draw no power when disabled

Output enable – connects outputs

- ⋆ allows data output from $D_7 \ldots D_0$
- ⋆ data lines draw no power when disabled



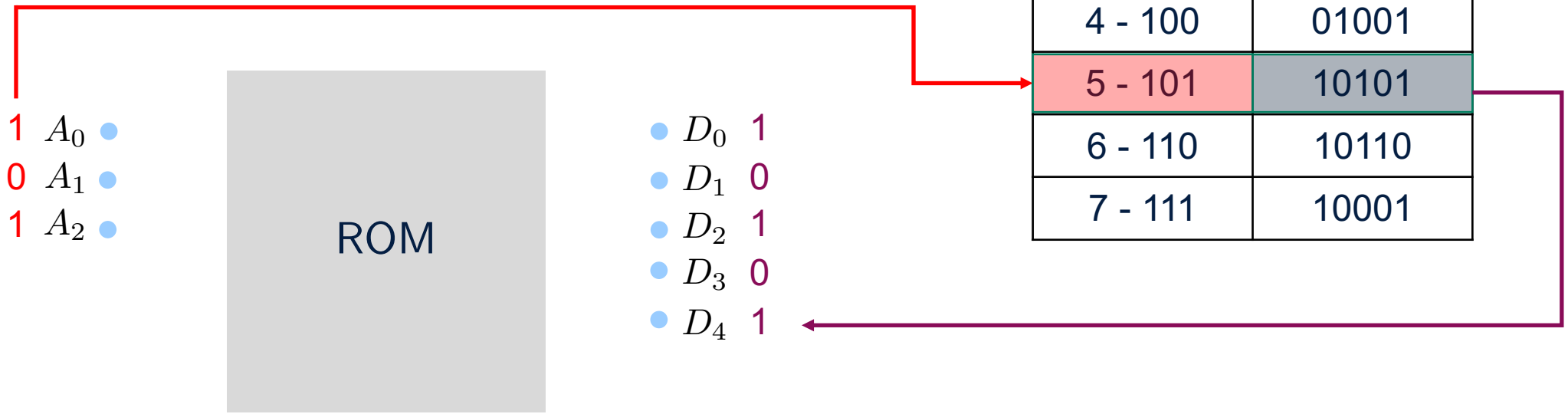


ROM – read only memory

ROM = look-up table

- ⋆ Table contents fixed by design
- ⋆ Can implement any truth table, but relatively inefficient for implementing logic functions

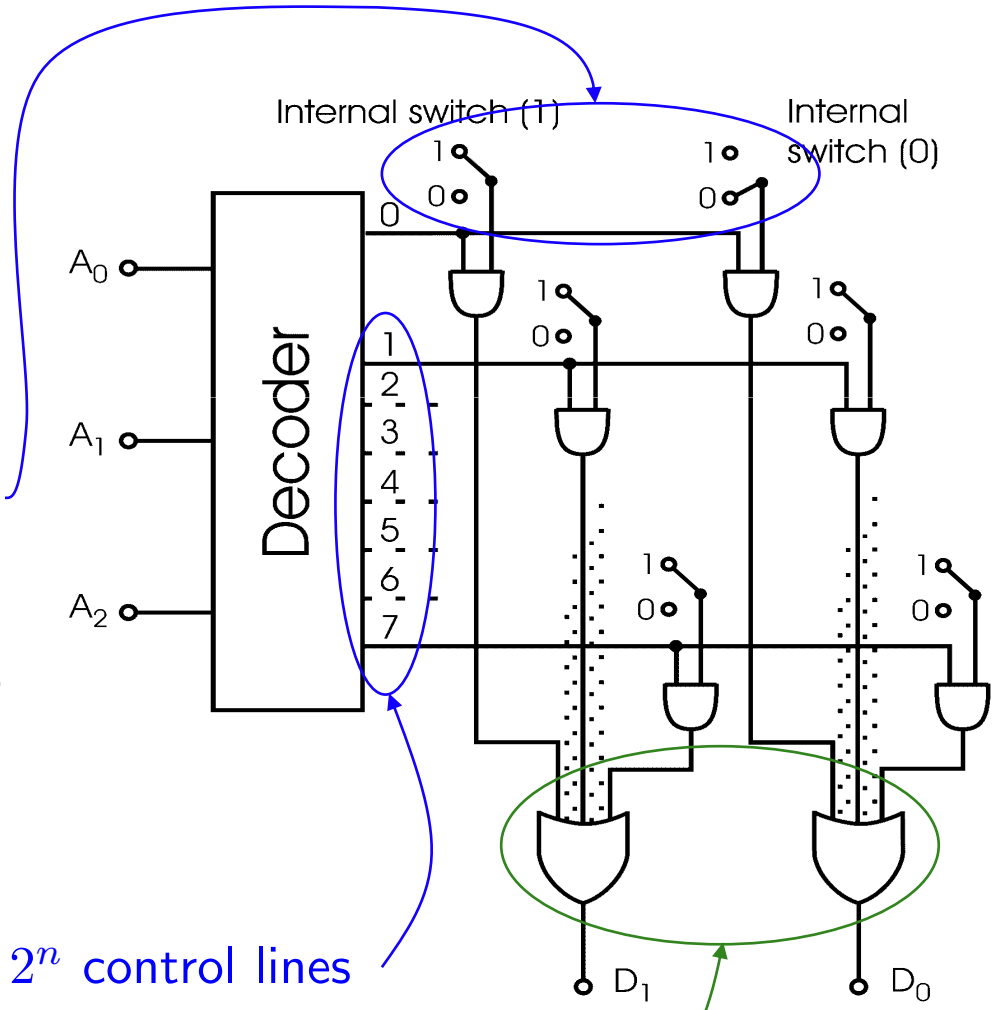
Address	Contents
0 - 000	01011
1 - 001	11001
2 - 010	11000
3 - 011	10000
4 - 100	01001
5 - 101	10101
6 - 110	10110
7 - 111	10001



3-bit wide address space with 5-bit wide data

How a ROM works

- $n \times m$ ROM: n address bits, m data bits
- Decoder: converts input binary number to drive one of 2^n control lines
- $2^n \times m$ matrix of digital switches to store bits of data
- Output stage to OR the outputs from each of the m columns



AND

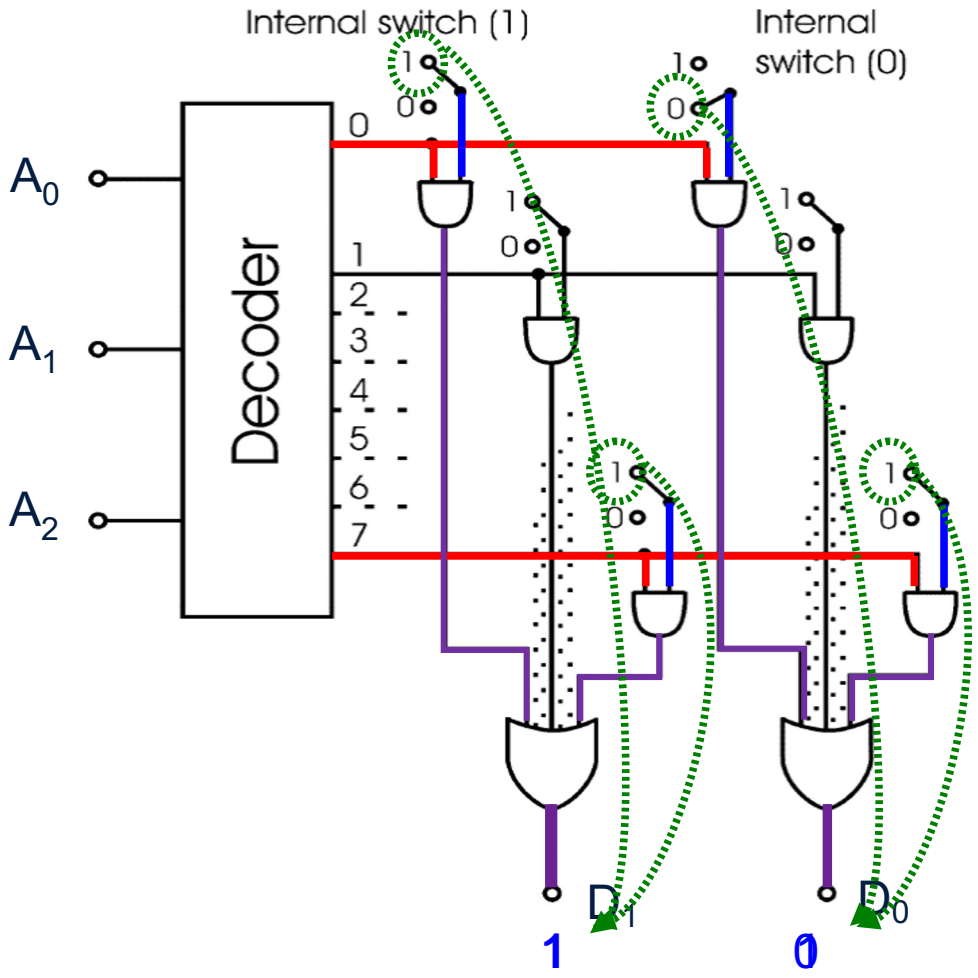
a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	x

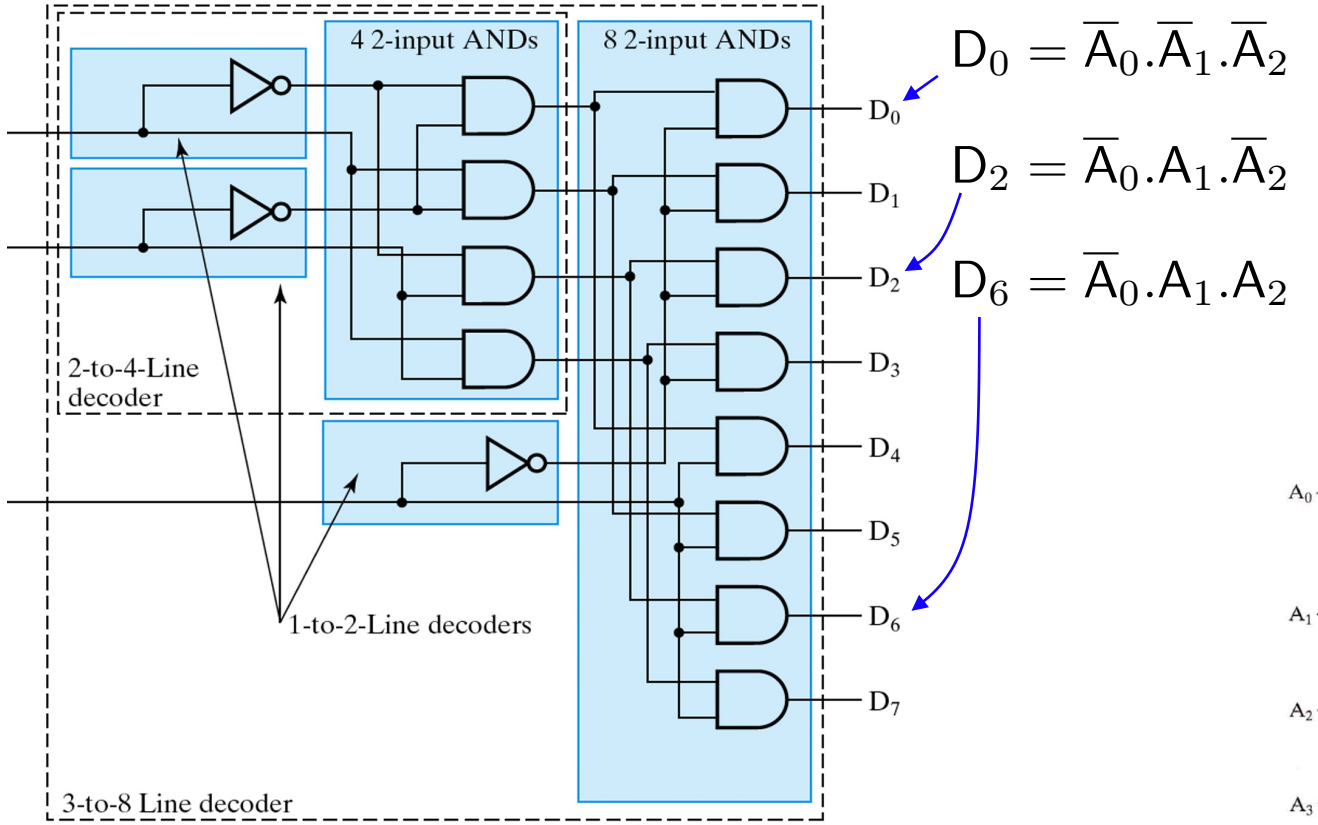
Decoder truth table

A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	**1**	0	0	0	0	0	0	0
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	**1**	0	0	0	0
1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	**1**	0	0
1	1	0	0	0	0	0	0	0	**1**	0
1	1	1	0	0	0	0	0	0	0	**1**



The Decoder ensures that only one control line is activated at any time instant, so only the selected stored data is connected to the output

The decoder



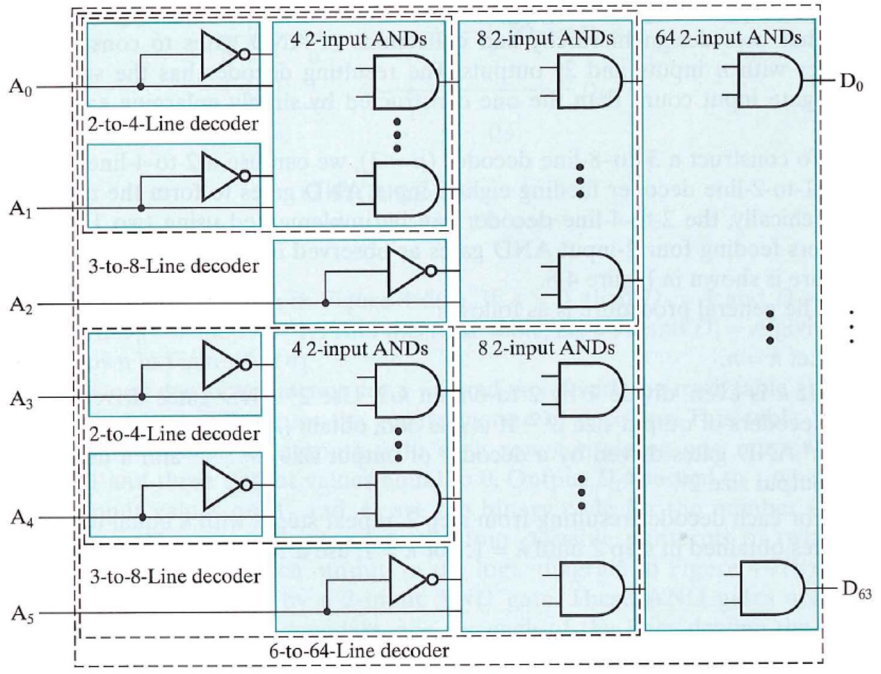
$$D_0 = \overline{A}_0.\overline{A}_1.\overline{A}_2$$

$$D_2 = \overline{A}_0.A_1.\overline{A}_2$$

$$D_6 = \overline{A}_0.A_1.A_2$$

$3 \rightarrow 8$

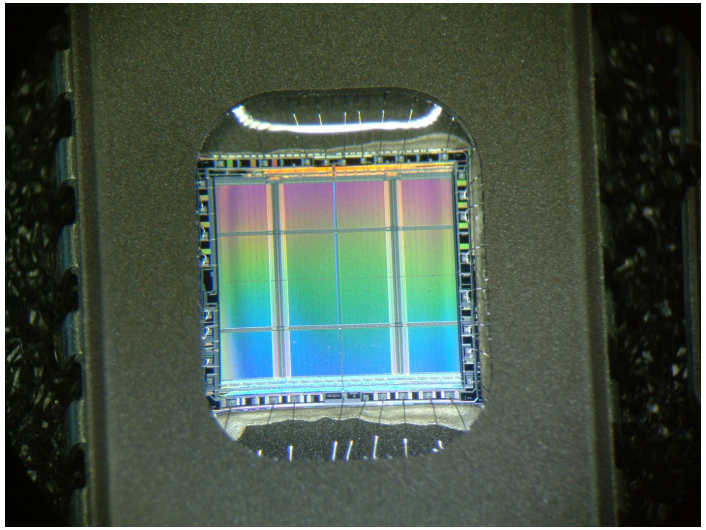
- ⋆ Similar in form to the multiplexer
- ⋆ Can become very complex when there are lots of address bits



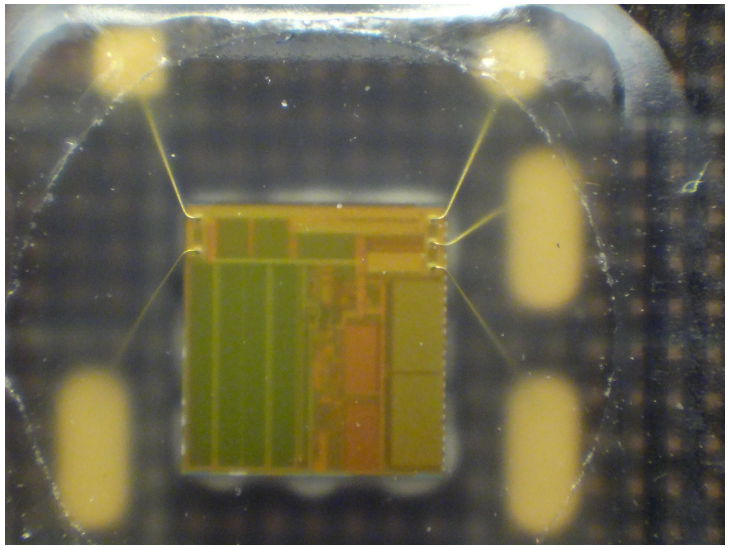
$6 \rightarrow 64$

ROM usage

- ROM is ‘non-volatile’ memory so doesn’t disappear when power is switched off; data is written by manufacturer
- PROM (Programmable ROM) is non-volatile, but can be programmed once using a special device
- Use a ROM (or PROM) to implement a required truth table
- EPROMs and EEPROMs can be erased and reprogrammed multiple times. EPROM is erasable by UV light, EEPROM by electricity

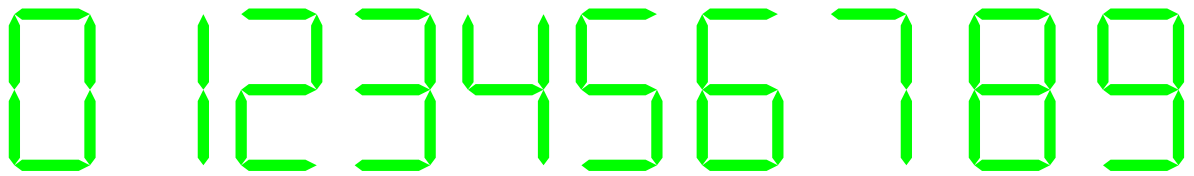


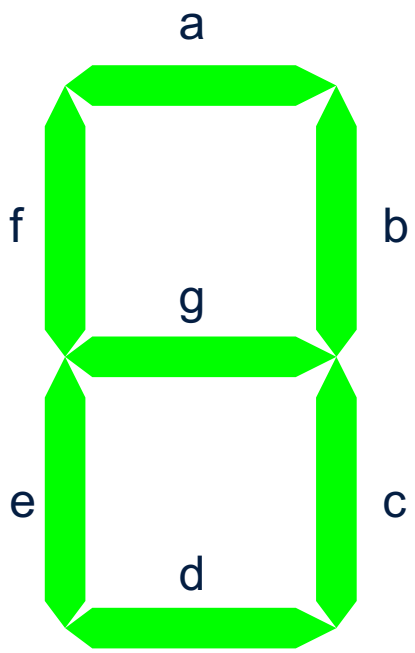
4 Mbit EPROM



EEPROM inside a SIM card

Code converter: 7 segment display

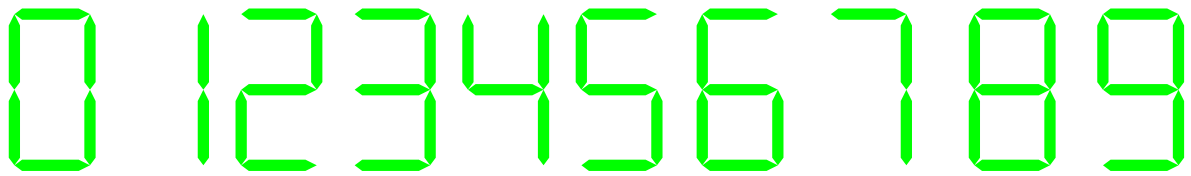


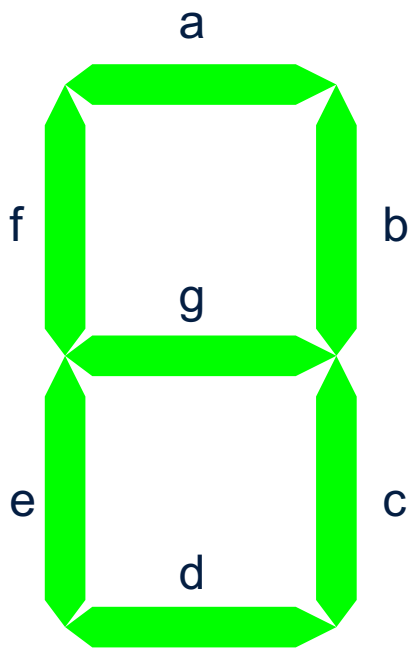


x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

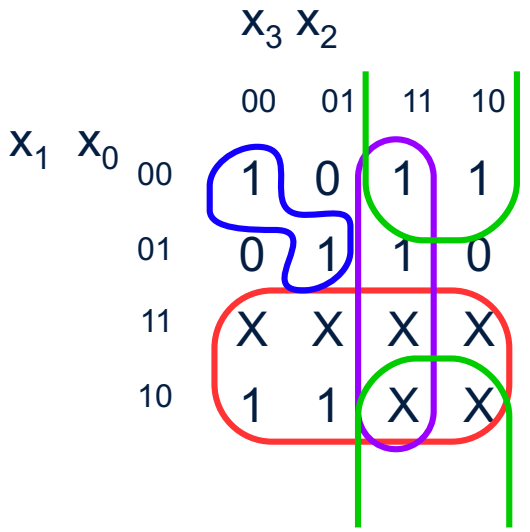
#	outputs
0	abcdef
1	bc
2	abdeg
3	abcdg
4	bcfg
5	acdfg
6	bcdefg
7	abc
8	abcdefg
9	abcdfg

Code converter: 7 segment display





x_3	x_2	x_1	x_0	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1



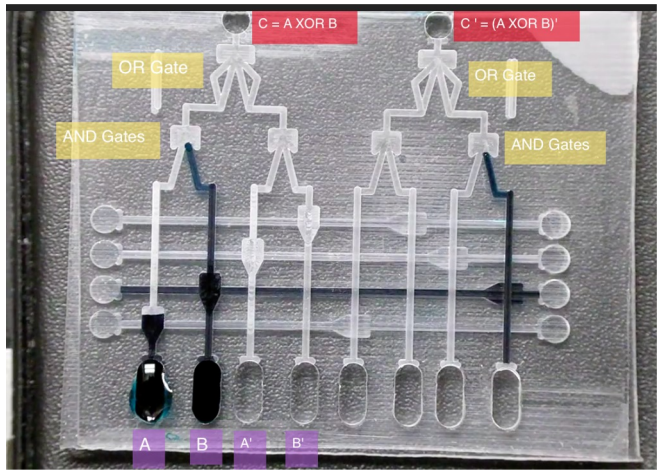
$$a = x_1 + x_3 \cdot x_2 + \overline{x_0} \cdot x_3 + \overline{x_2 \oplus x_0}$$

#	outputs
0	abcdef
1	bc
2	abdeg
3	abcdg
4	bcfg
5	acdfg
6	bcdefg
7	abc
8	abcdefg
9	abcdfg

What is a logic gate?

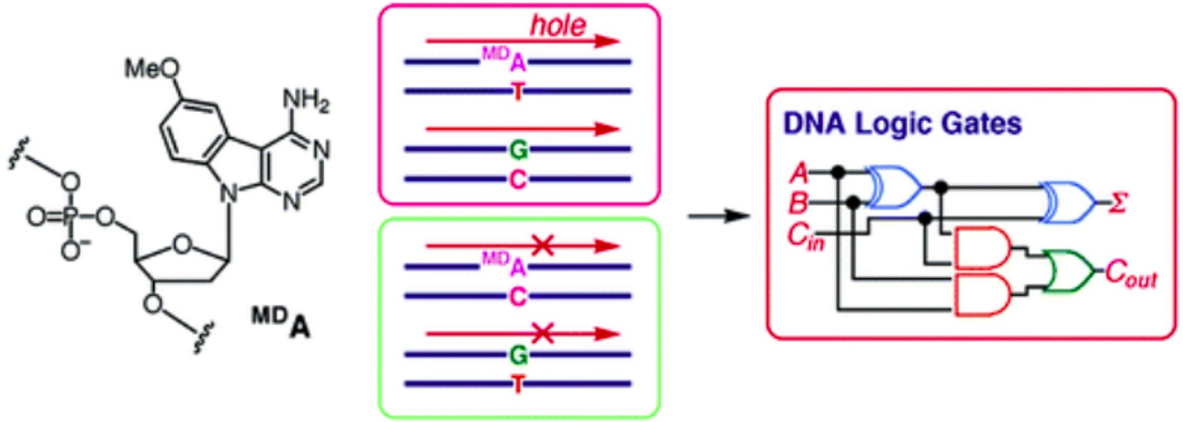
- A circuit constructed from transistors that implements a logic expression?
- Or any device that implements a logic expression?
- Mechanical gates: https://www.youtube.com/watch?v=HGNsLtT2wXw, https://www.youtube.com/watch?v=5X_Ft4YR_wU, https://www.randomwraith.com/logic.html

- Fluidic logic gates



https://www.youtube.com/watch?v=T3-3Y8gDbPA

DNA logic gates



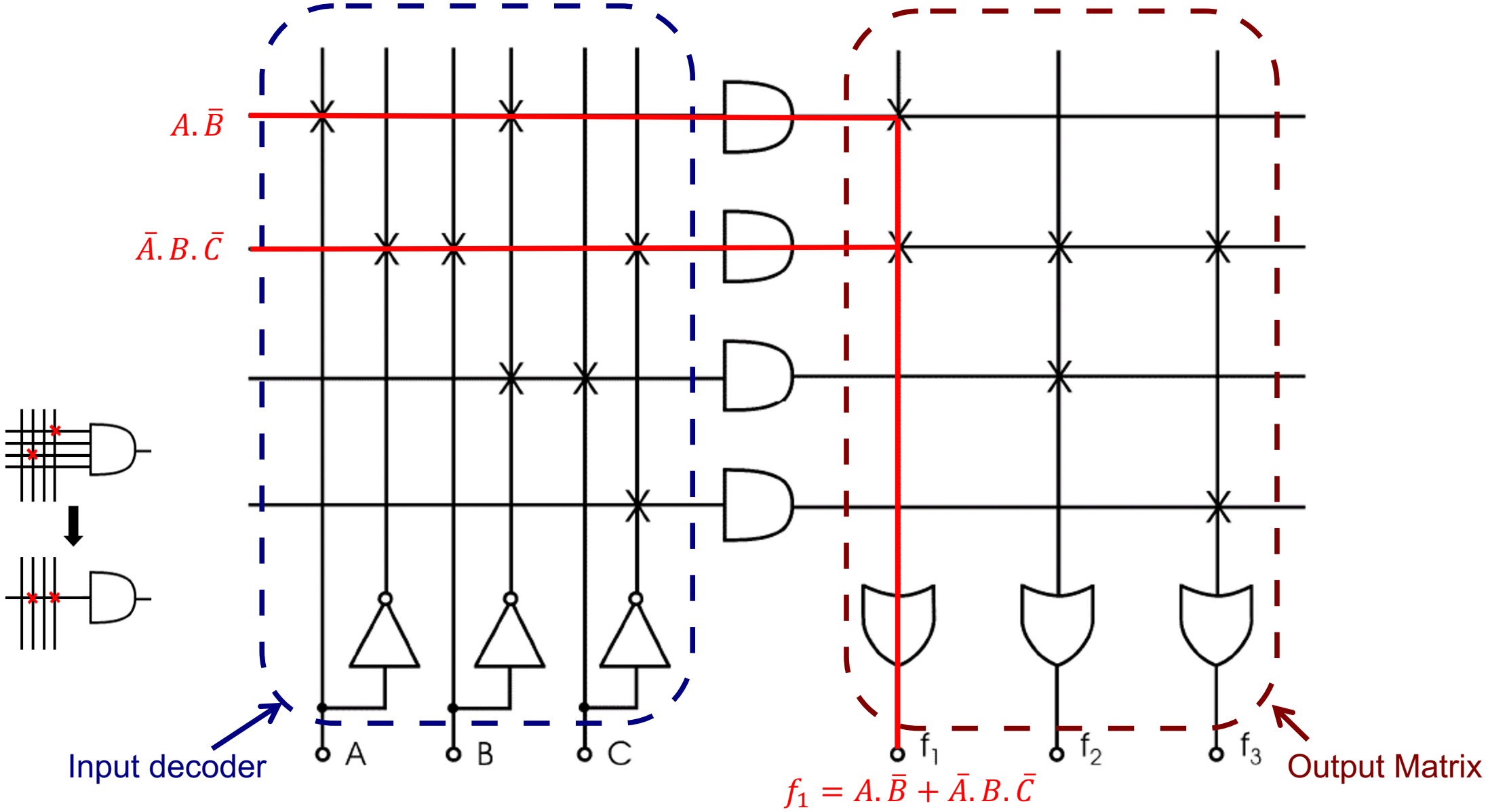
J. Am. Chem. Soc., 2004, 126 (30), pp 9458-9463

Programmable Logic Array (PLA)

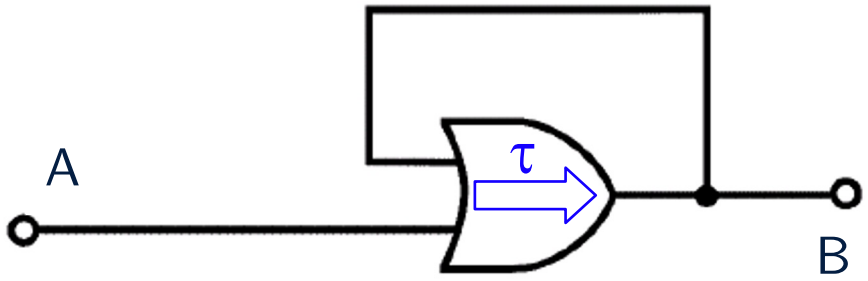
A PLA is a chip capable of implementing any sum-of-products logic function

- Contains OR and AND gates and logic to link them together
- During programming, permanent links are made between inputs and outputs
- More efficient than ROM for simple logic functions
- Generally, can only program once using a special programmer
- Precursors to modern, more sophisticated field programmable gate arrays (FPGAs)

Programmable Logic Array (PLA)

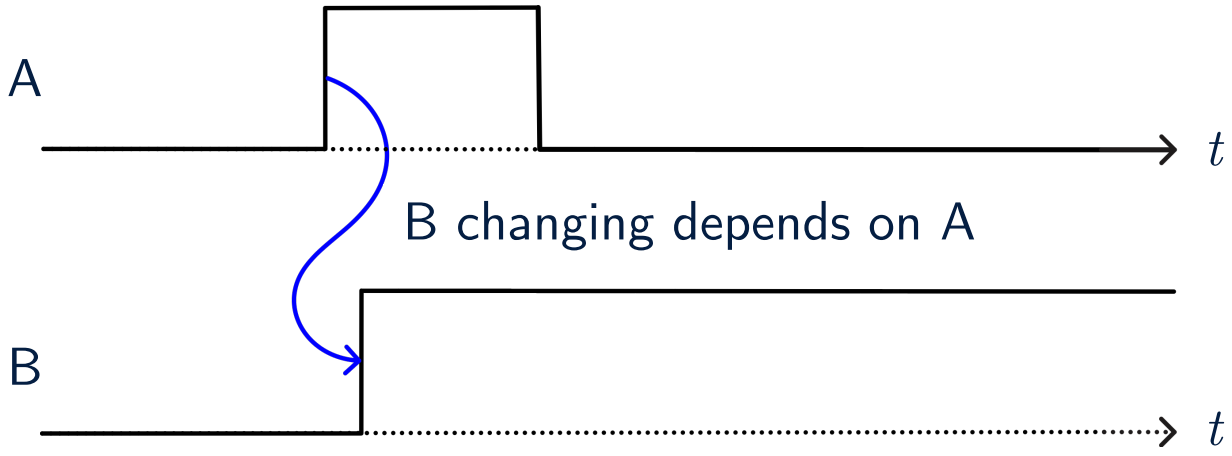


Creating a memory: a simple latch



OR gate with feedback

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

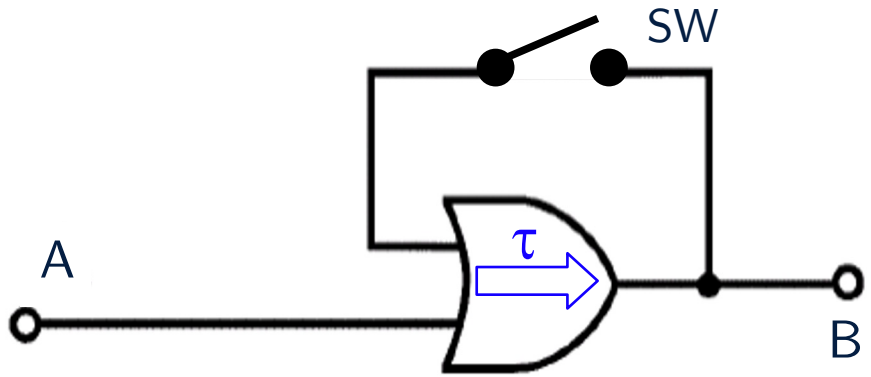


Timing diagram showing edge dependence

The feedback allows us to record an event (A changing from 0 to 1) ...
but we can't set B back to 0!

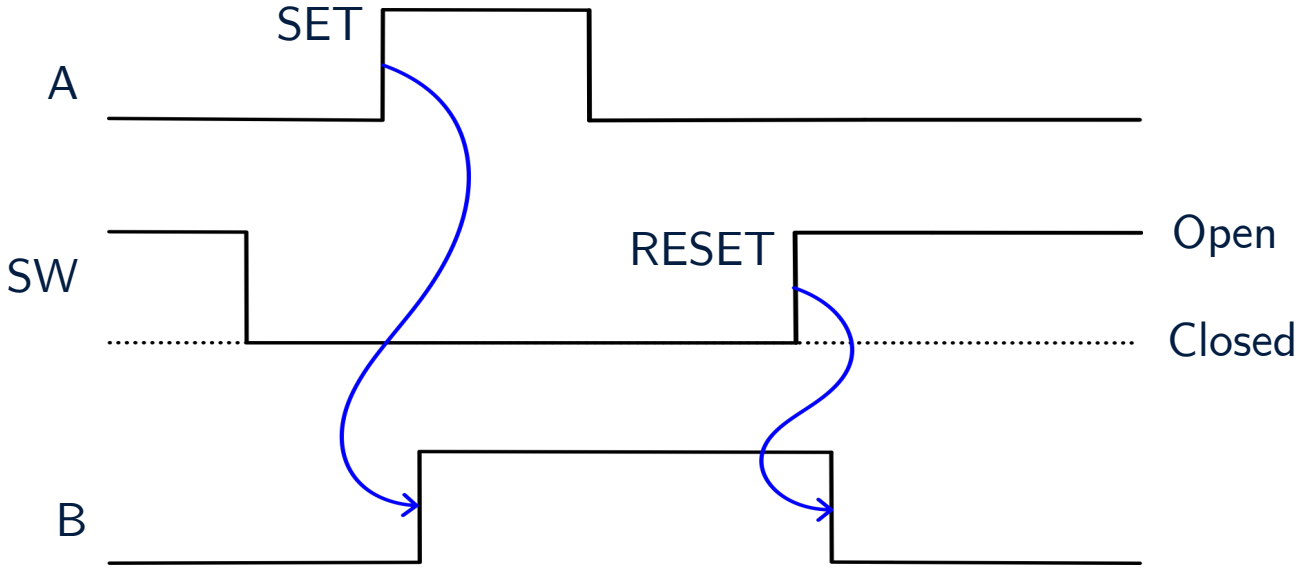
Including a reset

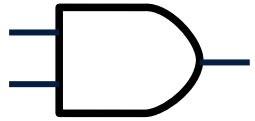
We need a way to make the feedback signal go to 0 when we want to store a 0 bit



Next value of output: $B_{n+1} = B_n + A$

No way to reset, as after setting $B_n = 1 \ \forall n$

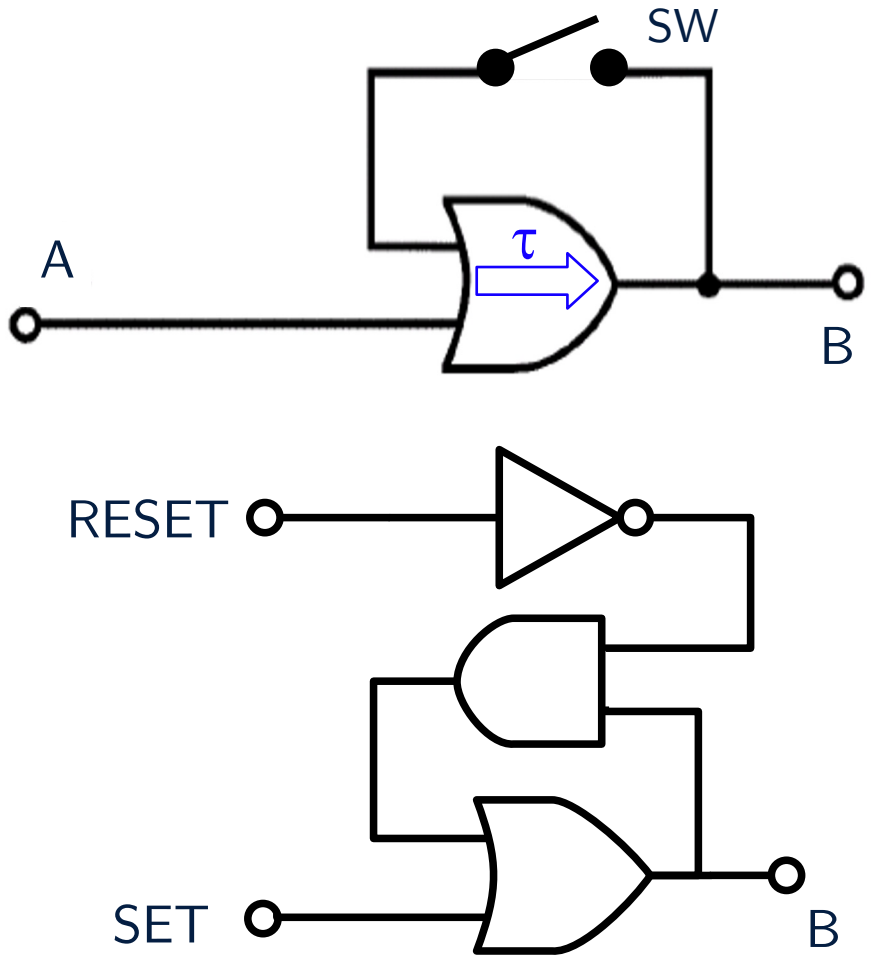




A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Including a reset

We need a way to make the feedback signal go to 0 when we want to store a 0 bit



Next value of output was $B_{n+1} = B_n + A$

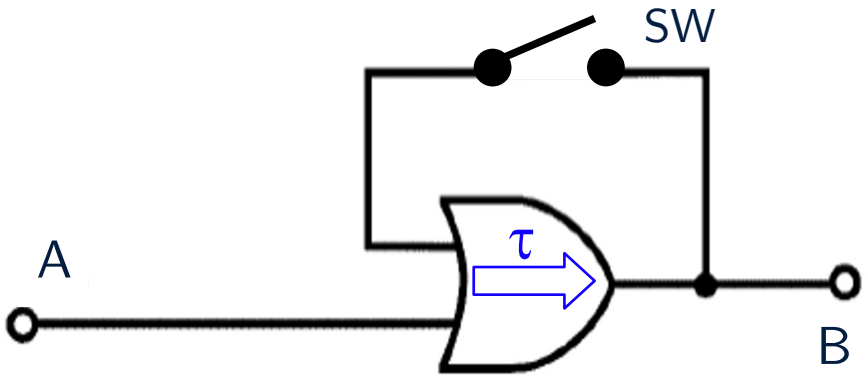
Now reset if RESET $= 1$:

$$B_{n+1} = B_n.\overline{RESET} + A$$
$$= \overline{(\overline{B_n} + RESET)} + A$$
$$\Rightarrow \quad \overline{B}_{n+1} = \overline{(\overline{B_n} + RESET)} + A$$
$$= \overline{\overline{(\overline{B_n} + RESET)} + SET}$$

Can implement with 2 NOR gates

Including a reset

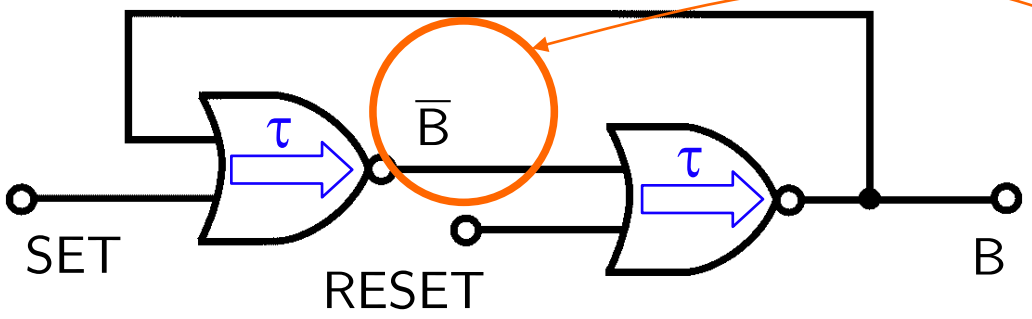
We need a way to make the feedback signal go to 0 when we want to store a 0 bit



Next value of output was $B_{n+1} = B_n + A$

Now reset if RESET $= 1$:

$$B_{n+1} = B_n.\overline{RESET} + A$$
$$= \overline{(\overline{B_n} + RESET)} + A$$
$$\Rightarrow \overline{B}_{n+1} = \overline{\overline{(\overline{B_n} + RESET)} + A}$$
$$= \overline{\overline{\overline{(\overline{B_n} + RESET)} + SET}}$$



Can implement with 2 NOR gates

Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. **Integration of digital logic components**
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk