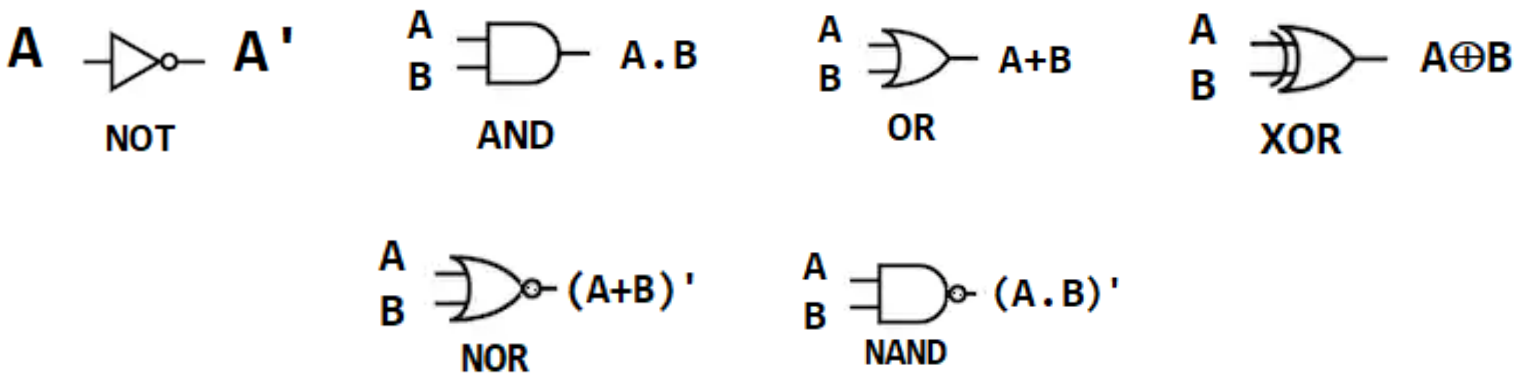
Summary

Boolean variables **A, B, C, . . .**
Boolean functions **F(A, B, C, D, ..)**
Boolean Gates **AND, OR, NOT, XOR, NOR, NAND**



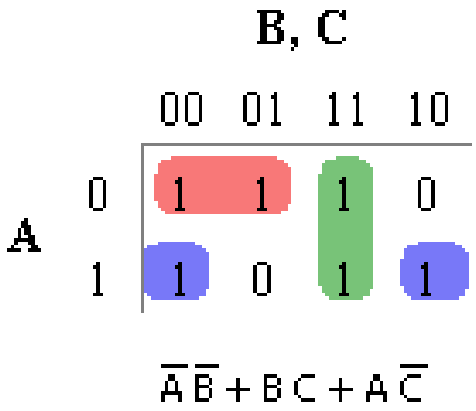
Summary

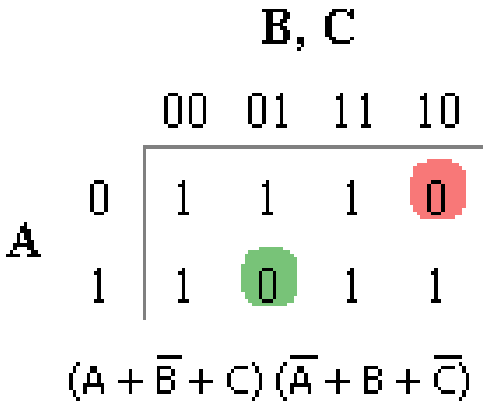
Obtaining F from the truth table

There are two ways to do that:

1. Sum of Product (**SOP**)
2. Product of Sum (**POS**)

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1





Summary

Simplification tools

1) Identities table
2) Karnaugh map

Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$AA' = 0$	$A + A' = 1$
	AB = BA	A + B = B + A
	(AB)C = A(BC)	(A + B) + C = A + (B + C)
	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
	A(A + B) = A	A + AB = A
	$(AB)' = A' + B'$	$(A + B)' = A'B'$

AB\CD	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	0	1	1	0
10	1	0	0	1

Logic gates

Combinational and sequential circuits

Combinational circuit : the outputs are uniquely determined by the current input values

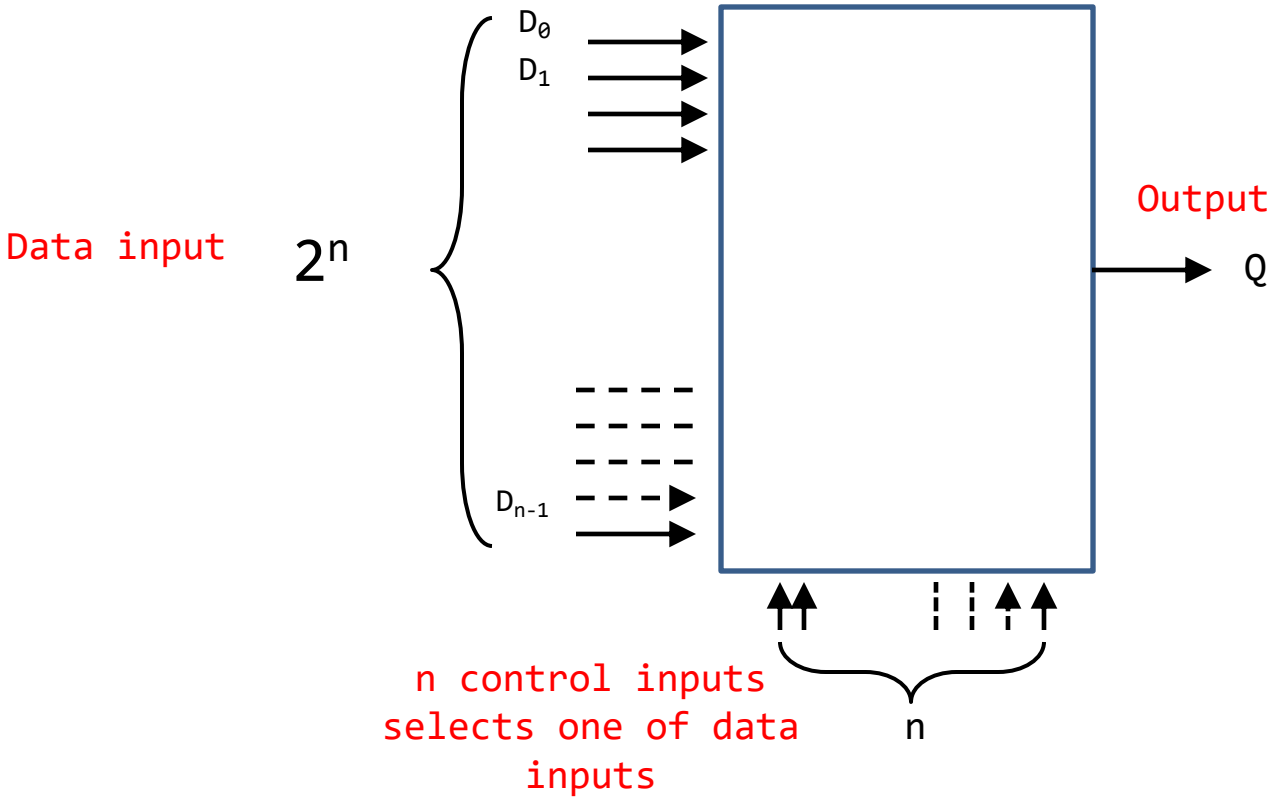
Examples – Encoder, Decoder, Multiplexer

Sequential Circuit : the output depends upon present as well as past input

Examples – Flip-flops

Logic gates

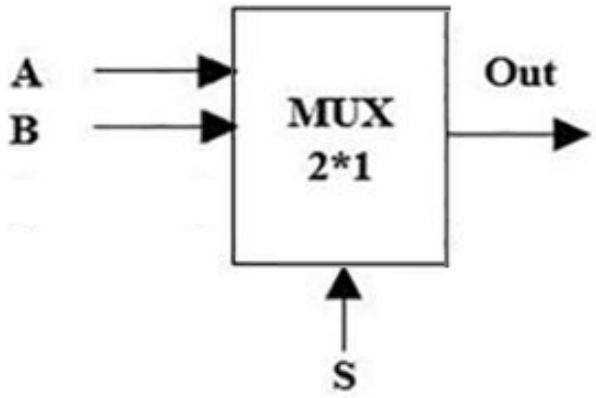
Multiplexer



Logic gates

Multiplexer

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

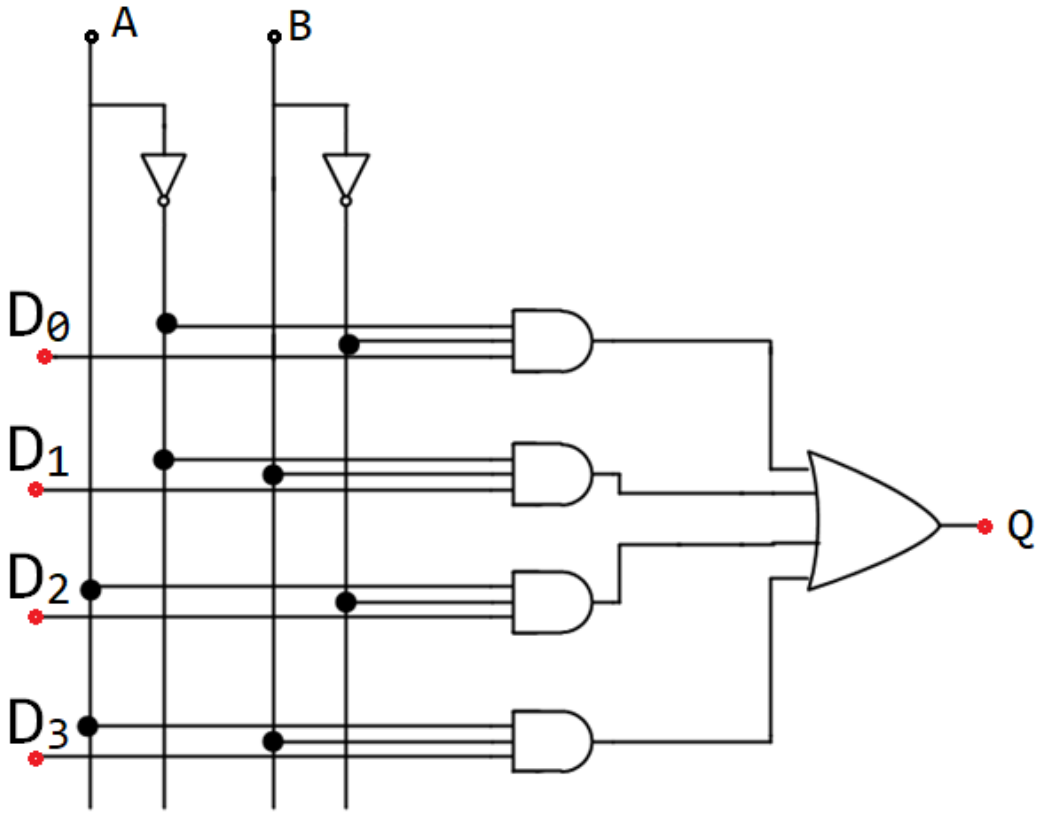


S	Out
0	A
1	B

Logic gates

Multiplexer

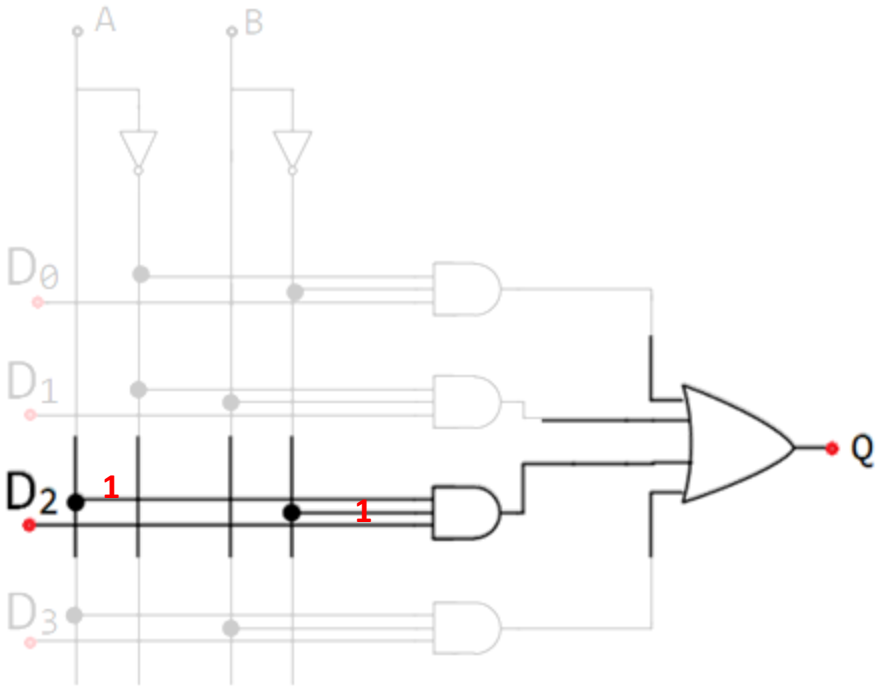
A	B	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



Logic gates

Multiplexer

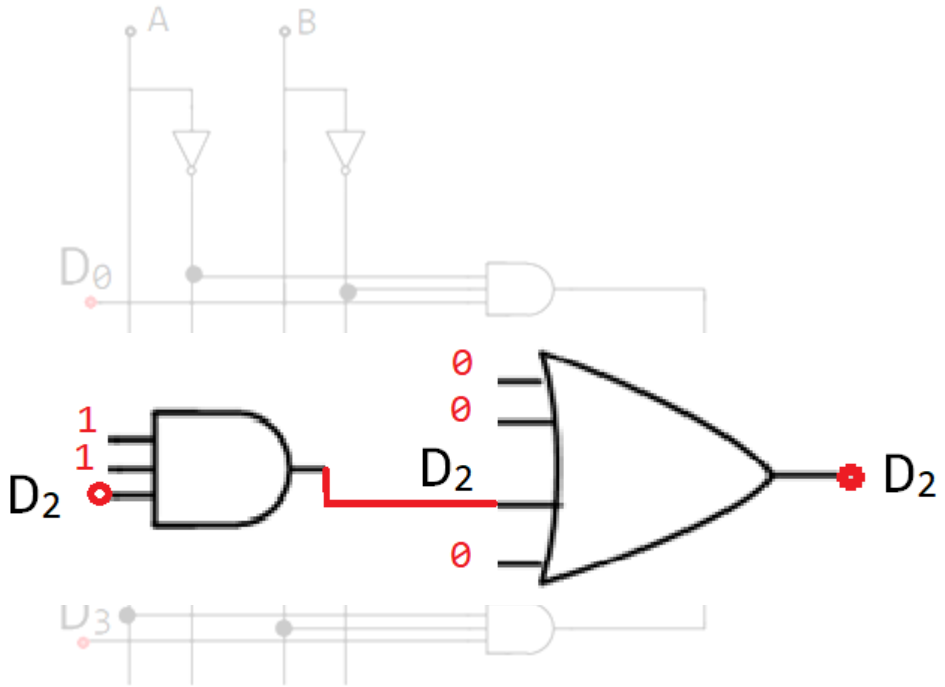
A	B	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



Logic gates

Multiplexer

A	B	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



Identity law	1A = A	0 + A = A

Logic gates

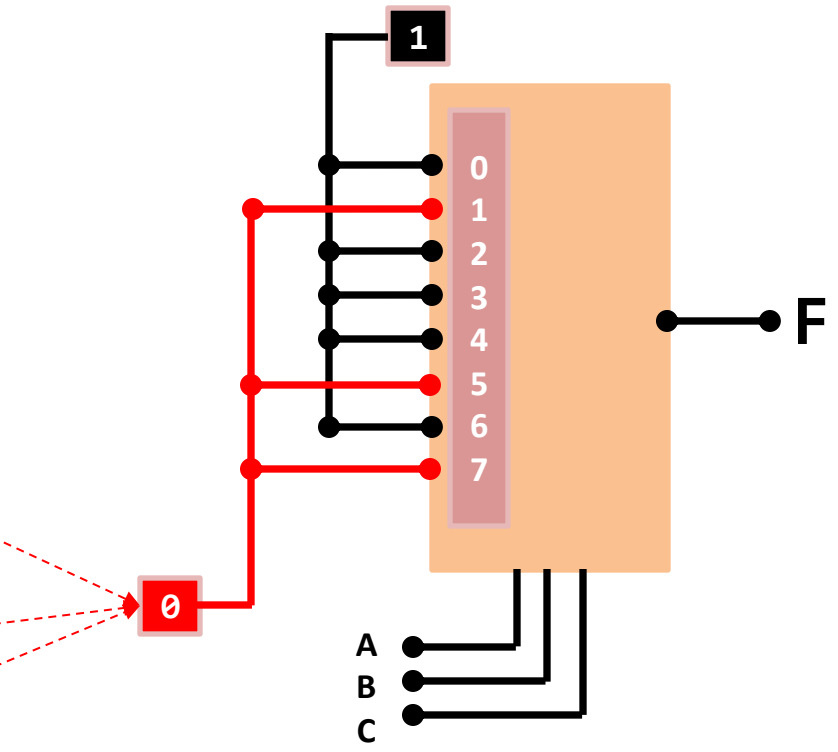
Multiplexer(Application)

Example: MUX(8X1)

n=3

	A	B	C	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

$2^3 = 8$



$F = C' + A'B$

Logic gates

Multiplexer(Application)

Example: MUX(4X1)

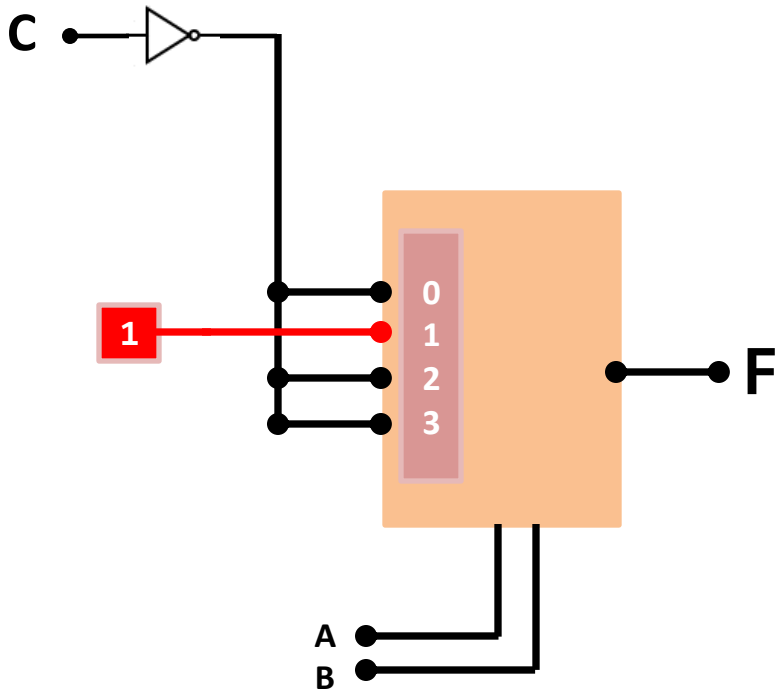
		A	B	C	F
0	0	0	0	0	1
	1	0	0	1	0
1	2	0	1	0	1
	3	0	1	1	1
2	4	1	0	0	1
	5	1	0	1	0
3	6	1	1	0	1
	7	1	1	1	0

F = C'

F = 1

F = C'

F = C'



Logic gates

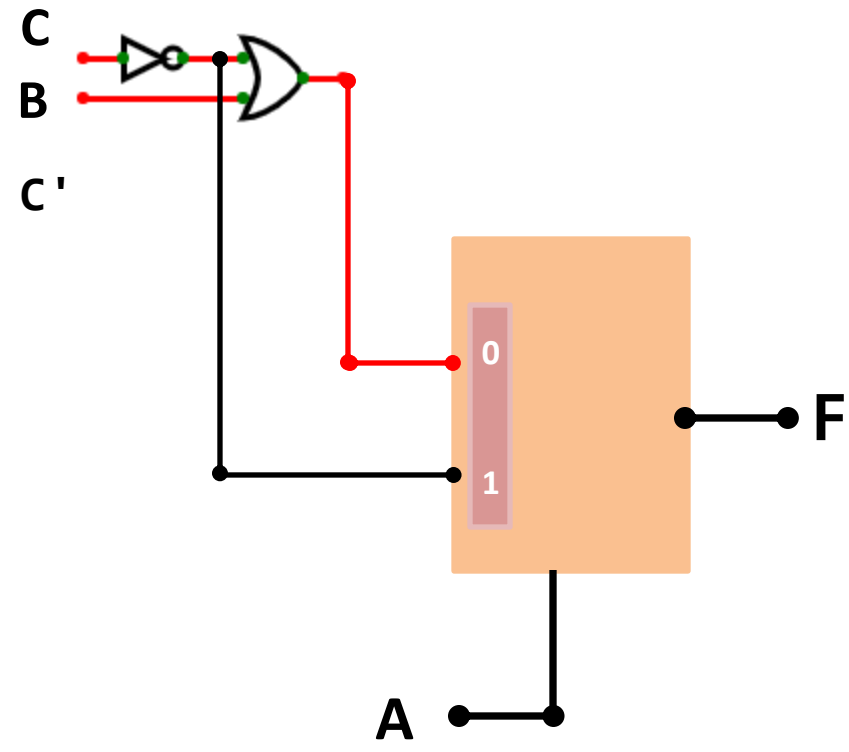
Multiplexer(Application)

Example: MUX(2X1)

		A	B	C	F
0	0	0	0	0	1
	1	0	0	1	0
	2	0	1	0	1
	3	0	1	1	1
1	4	1	0	0	1
	5	1	0	1	0
	6	1	1	0	1
	7	1	1	1	0

F = B + C'

F = C'

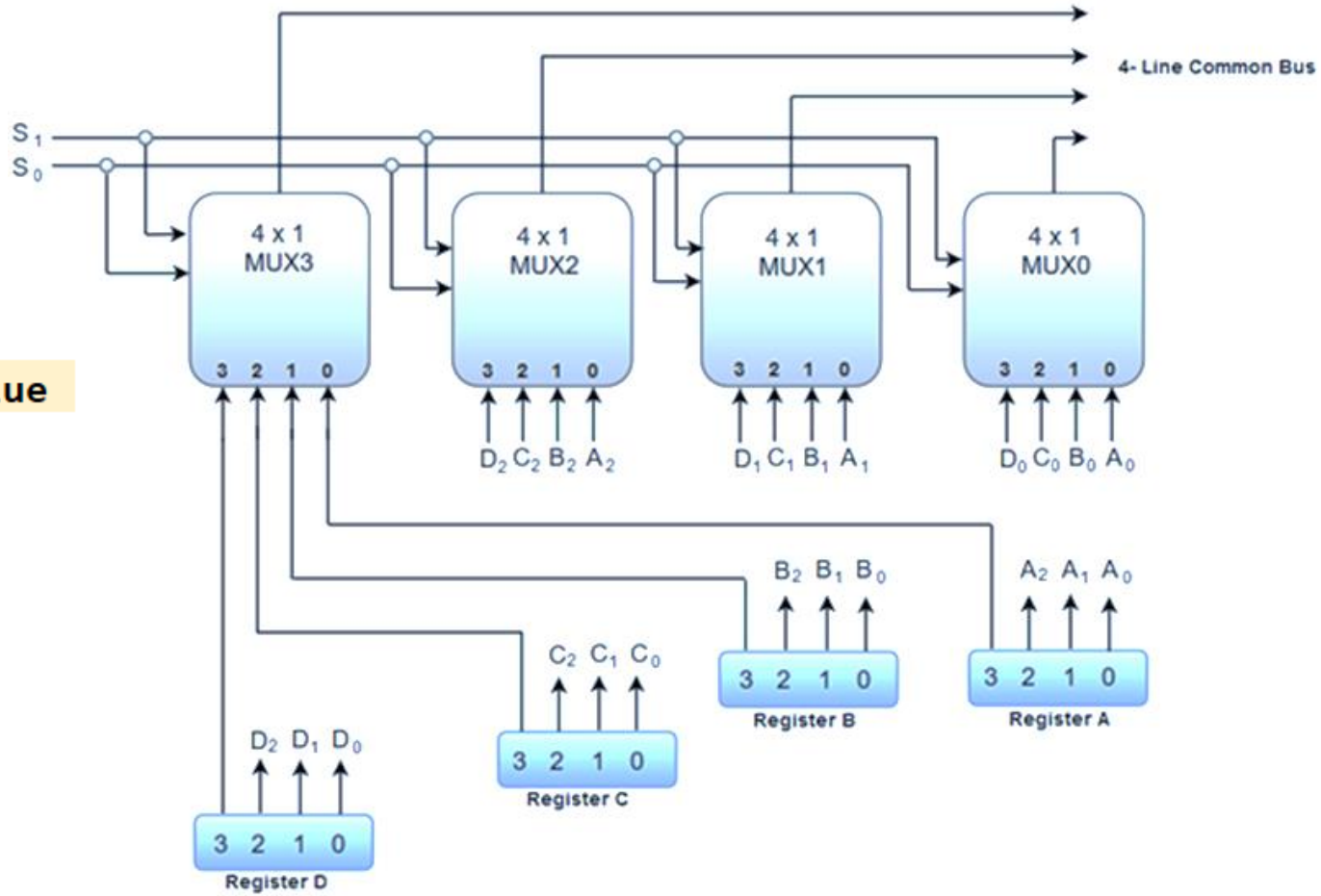


Logic gates

Multiplexer(Application)

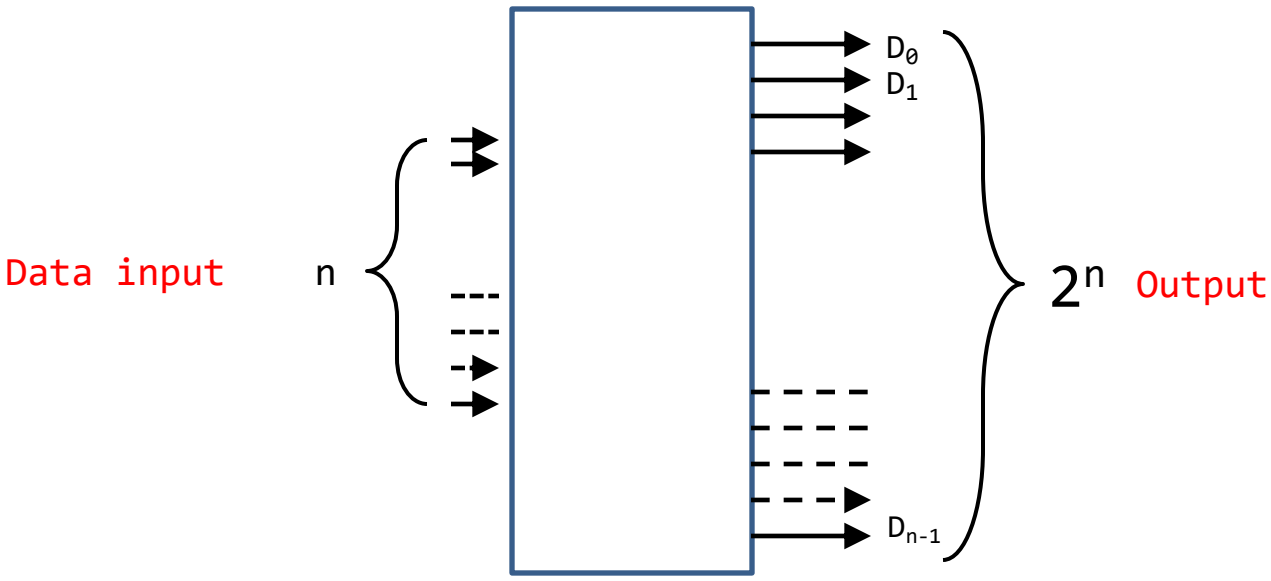
Bus System for 4 Registers

S0	S1	BUS value
0	0	A
1	0	B
0	1	C
1	1	D



Logic gates

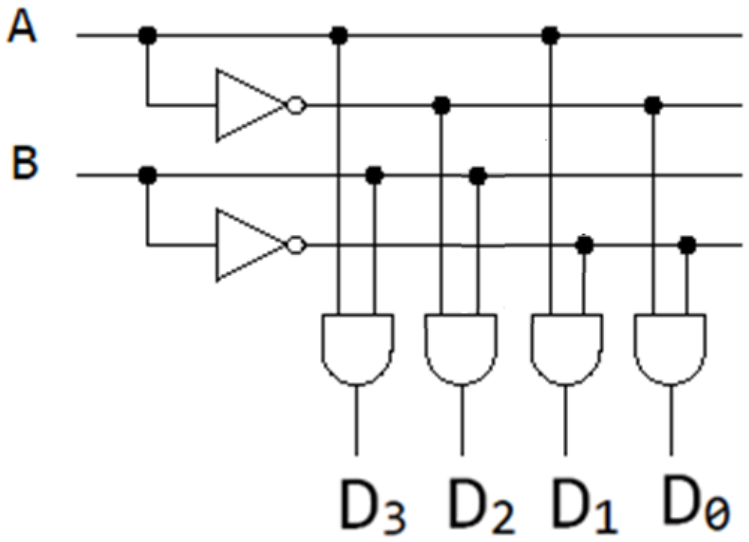
Decoders



Logic gates

Decoders

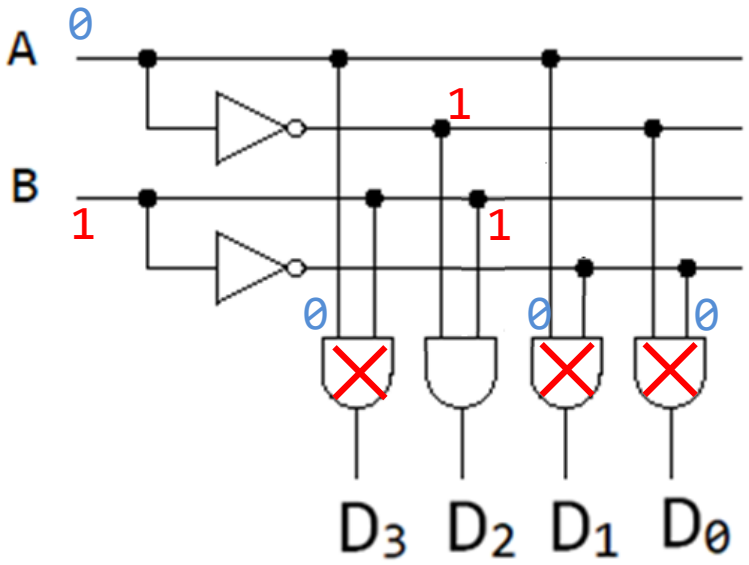
B	A	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Logic gates

Decoders

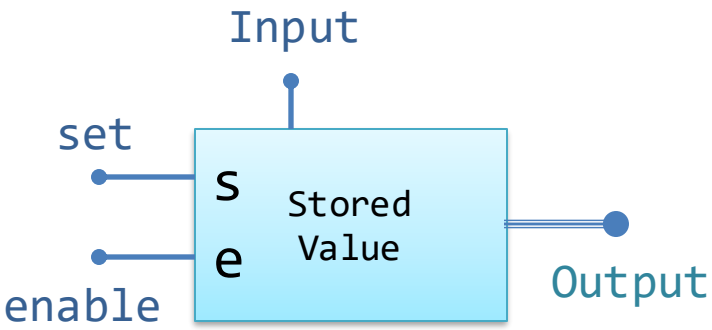
B	A	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



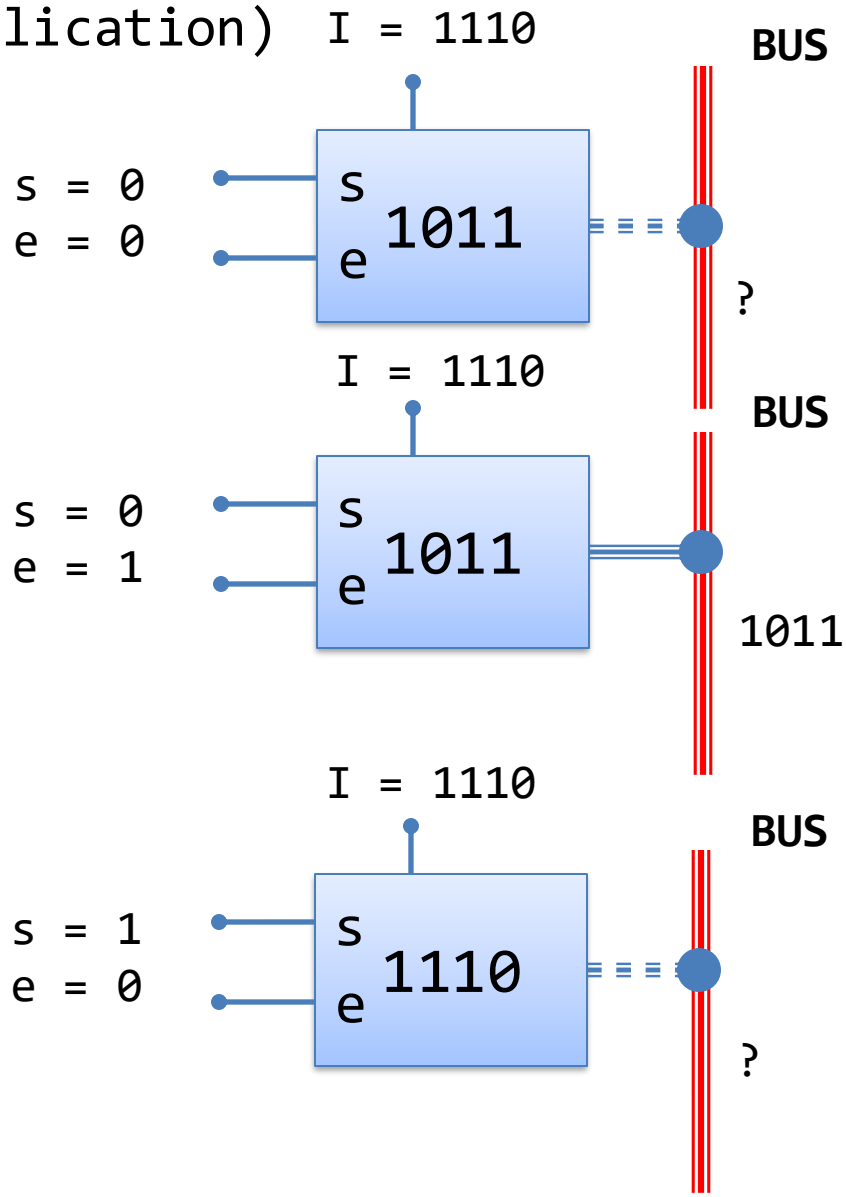
Logic gates

Registers (local fast memories)

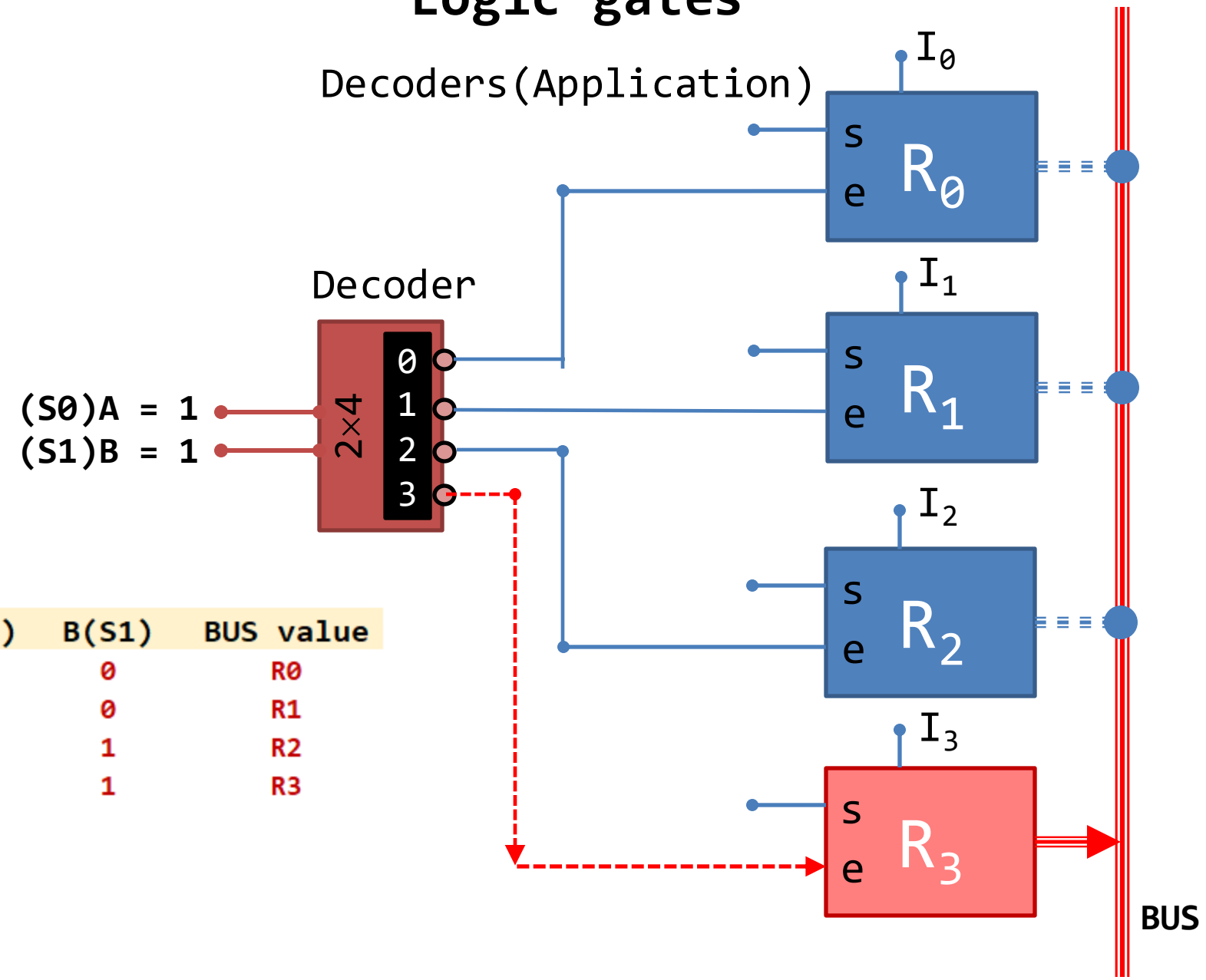
1. Input
2. Output
3. Stored value
4. Control signals
   - e - Enable
   - s - Set



Decoders(Application)



Logic gates



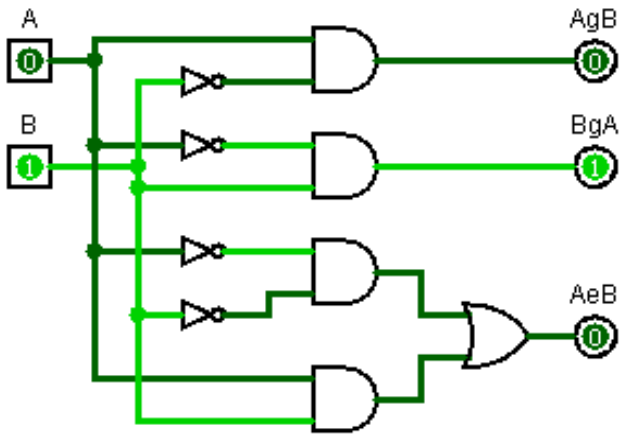
A(S0)	B(S1)	BUS value
0	0	R0
1	0	R1
0	1	R2
1	1	R3

Logic gates

Comparators

A	B	A>B	B>A	A=B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

A=B : A'B' + AB
A>B : AB'
B>A : A'B



A⊕B = A'B + AB'
(A⊕B)' = A'B' + AB

A=B : (A⊕B)'
A>B : AB'
B>A : A'B

Logic gates

Comparators

	B	B1	B0	A1	A0	A	A>B	B>A	A=B
0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	1	1	0	0
2	0	0	0	1	0	2	1	0	0
3	0	0	0	1	1	3	1	0	0
4	1	0	1	0	0	0	0	1	0
5	1	0	1	0	1	1	0	0	1
6	1	0	1	1	0	2	1	0	0
7	1	0	1	1	1	3	1	0	0
8	2	1	0	0	0	0	0	1	0
9	2	1	0	0	1	1	0	1	0
10	2	1	0	1	0	2	0	0	1
11	2	1	0	1	1	3	1	0	0
12	3	1	1	0	0	0	0	1	0
13	3	1	1	0	1	1	0	1	0
14	3	1	1	1	0	2	0	1	0
15	3	1	1	1	1	3	0	0	1

A>B :

$B1'B0'A0 + B1'A1 + B0'A1A0$

B>A :

$A1'A0'B0 + A1'B1 + A0'B1B0$

A=B :

$B1'B0'A1'A0' + B1'B0A1'A0 + B1B0'A1A0' + B1B0A1A0$

A=B :

$= (A1 \oplus B1)'(A0 \oplus B0)'$

Logic gates

Comparators

A>B :

(A1>B1) OR {(A1 = B1) AND (A0>B0)}

= (A1 AND B1') OR (A1⊕B1)' AND (A0 AND B0')

= A1B1' + (A1⊕B1)'.A0B0'

A=B :

(A1 = B1) AND (A0 = B0)

= (A1⊕B1)' AND (A0⊕B0)'

= (A1⊕B1)'(A0⊕B0)'

B>A :

(B1>A1) OR {(B1 = A1) AND (B0>A0)}

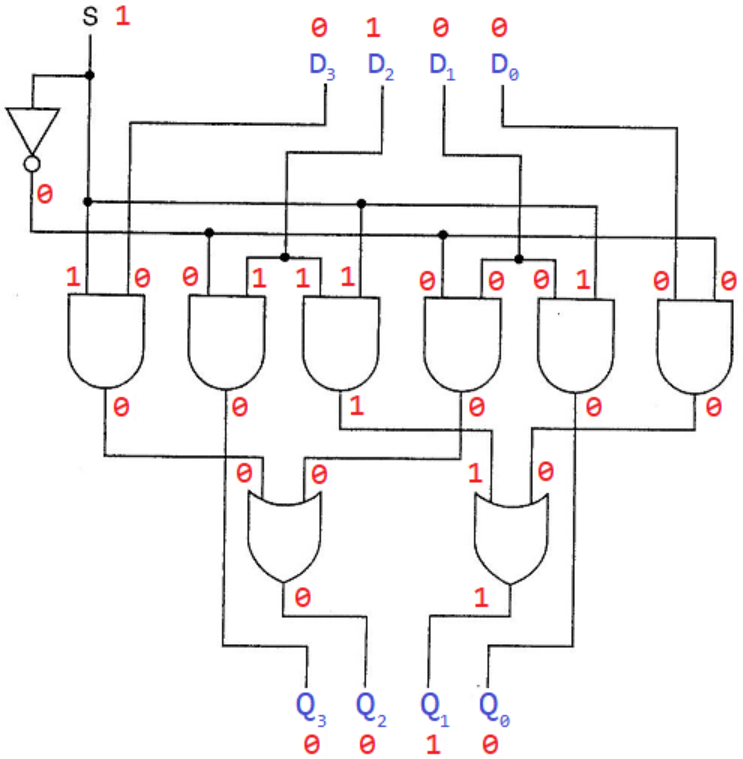
= (B1 AND A1') OR (B1⊕A1)' AND (B0 AND A0')

= B1A1' + (B1⊕A1)'.B0A0'

Logic gates

1-bit left/right shifter

S = 1 shift to right
S = 0 shift to left

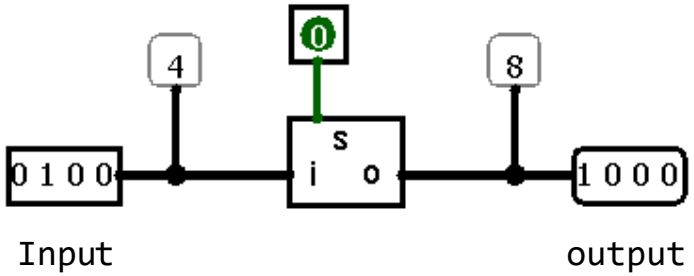


Logic gates

1-bit left/right shifter

S = 1 shift to right >>
S = 0 shift to left <<

Left shift = value × 2
Left shift = 4 × 2 = 8

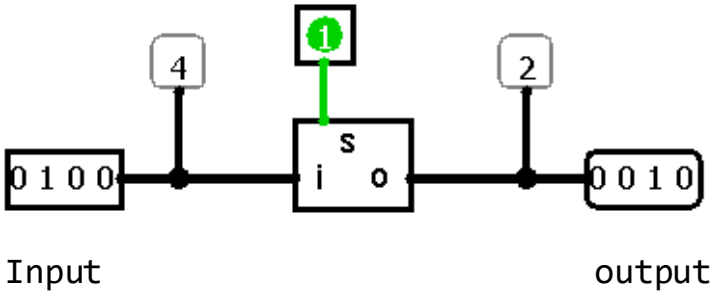


Logic gates

1-bit left/right shifter

S = 1 shift to right >>
S = 0 shift to left <<

Right shift = value/2
Right shift = 4/2 = 2



Logic gates

2-bit left/right shift in C

```c
#include <stdio.h>

int main(void) {
  int x = 16;

  printf("Shift to the  right (16/4): %i \n", x>>2);
  printf("Shift to the  left  (16*4): %i \n", x<<2);
  return 0;
}
```

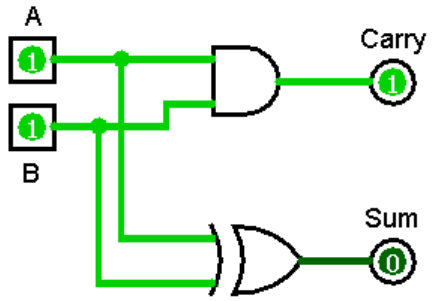
```
> ./main
Shift to the  right (16/4): 4
Shift to the  left  (16*4): 64
>
```

Logic gates

Half adder

A	B	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

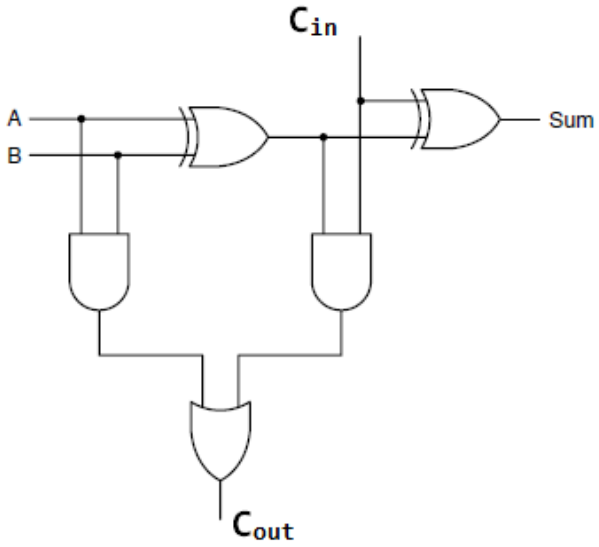
$$\text{sum} = \bar{A}B + A\bar{B} = A \oplus B$$
$$\text{carry} = AB$$



Logic gates

Full adder (2 half adders)

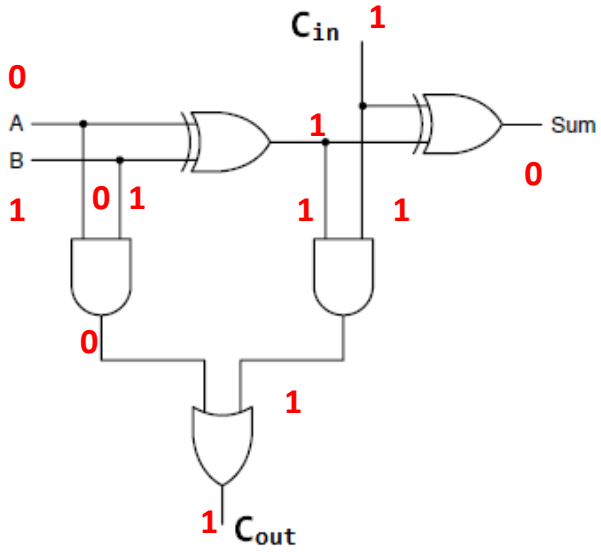
A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Logic gates

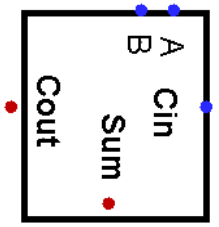
Full adder (2 half adders)

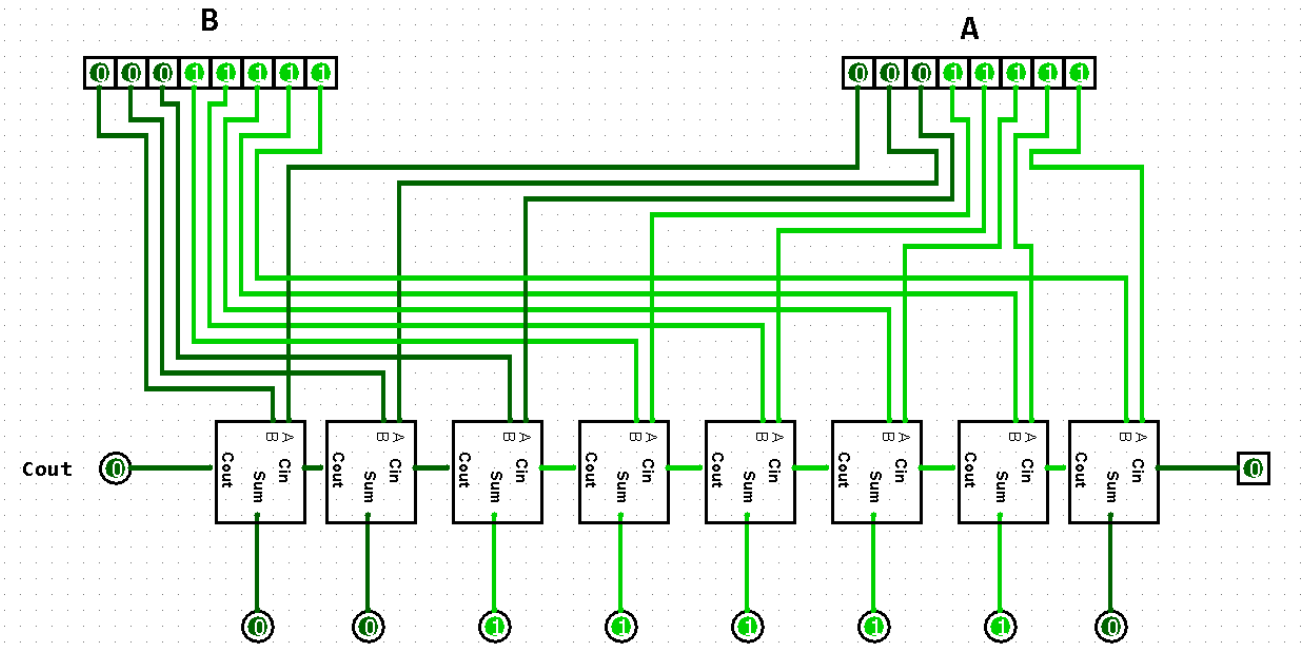
A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Logic gates

8-bit adder (8 full adders)

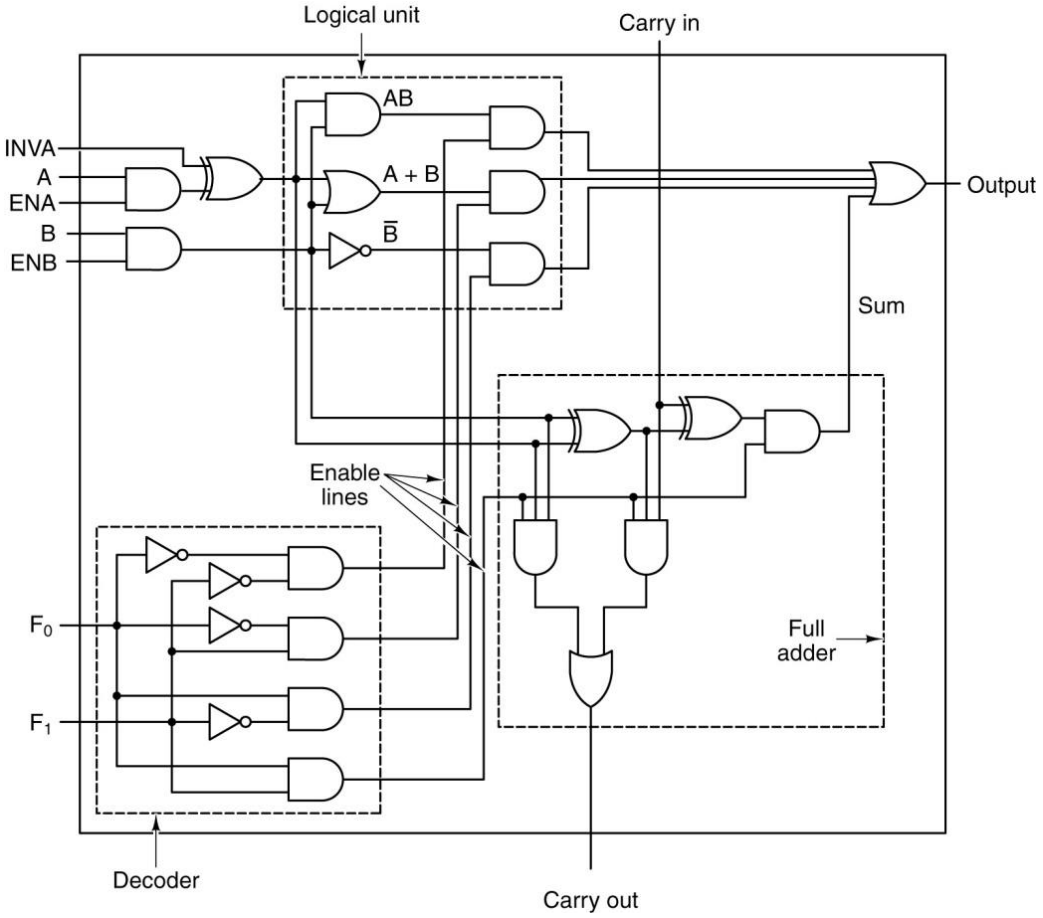




Logic gates

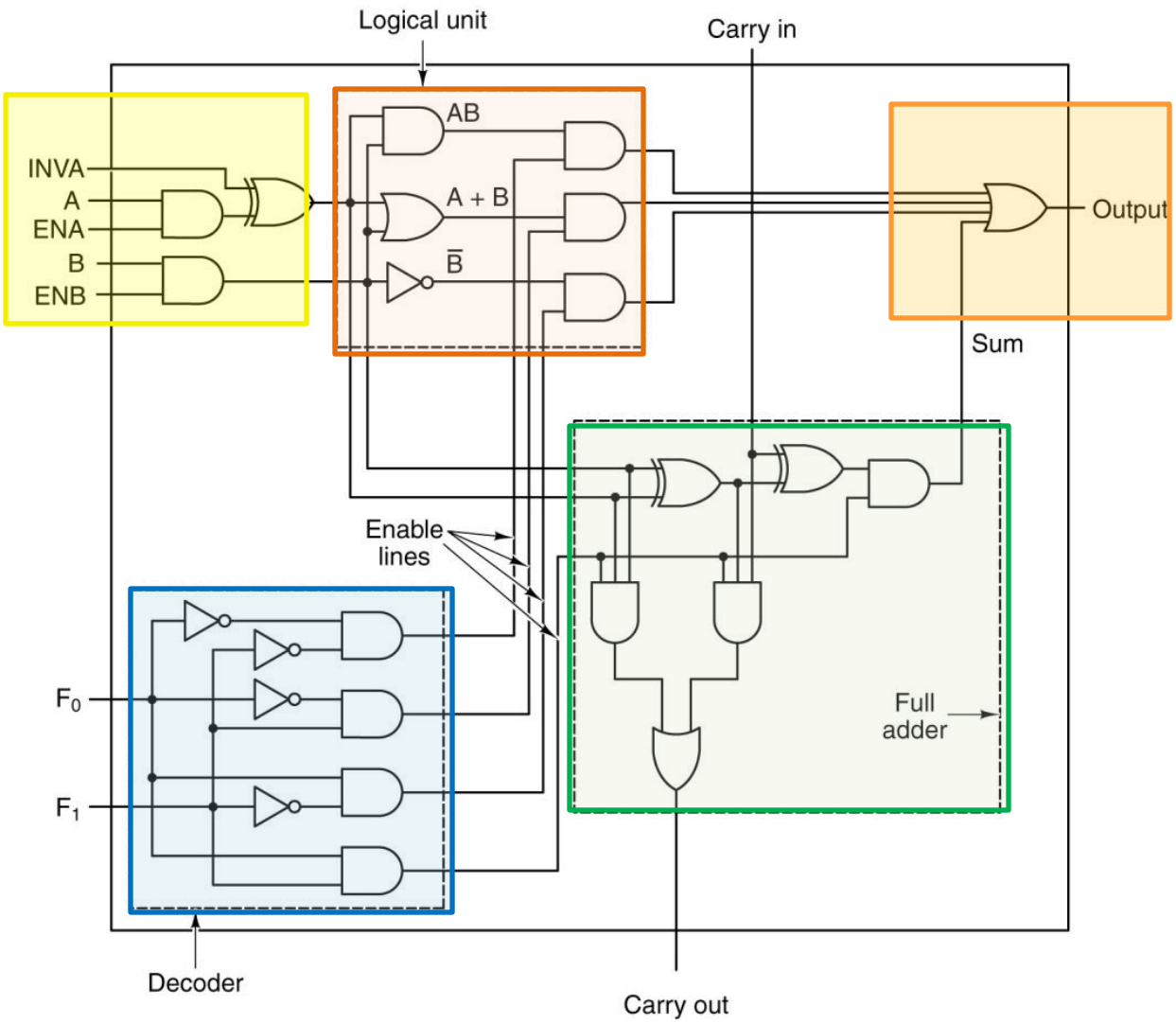
Arithmetic Logic Unit(ALU)

All circuits such as NOT, AND, OR, FULL ADDER, DECODER, COMPARATOR and SHIFTER now are available to build a 8, 16, 32, or 64 bits ALU.



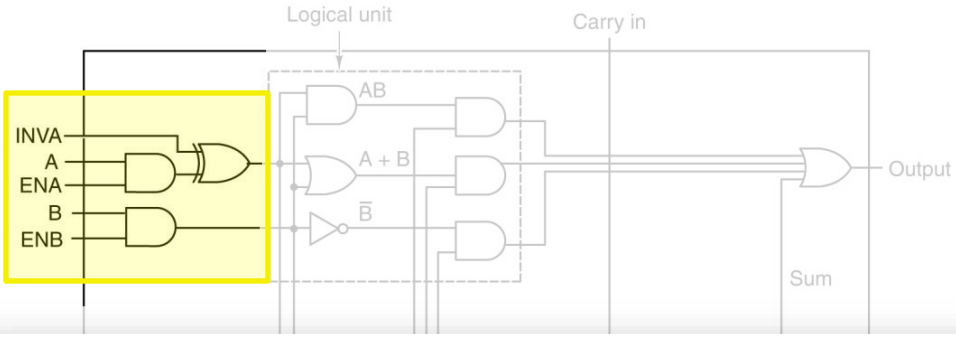
Logic gates

1-bit ALU(arithmetic logic unit)



Logic gates

1-bit ALU(arithmetic logic unit)



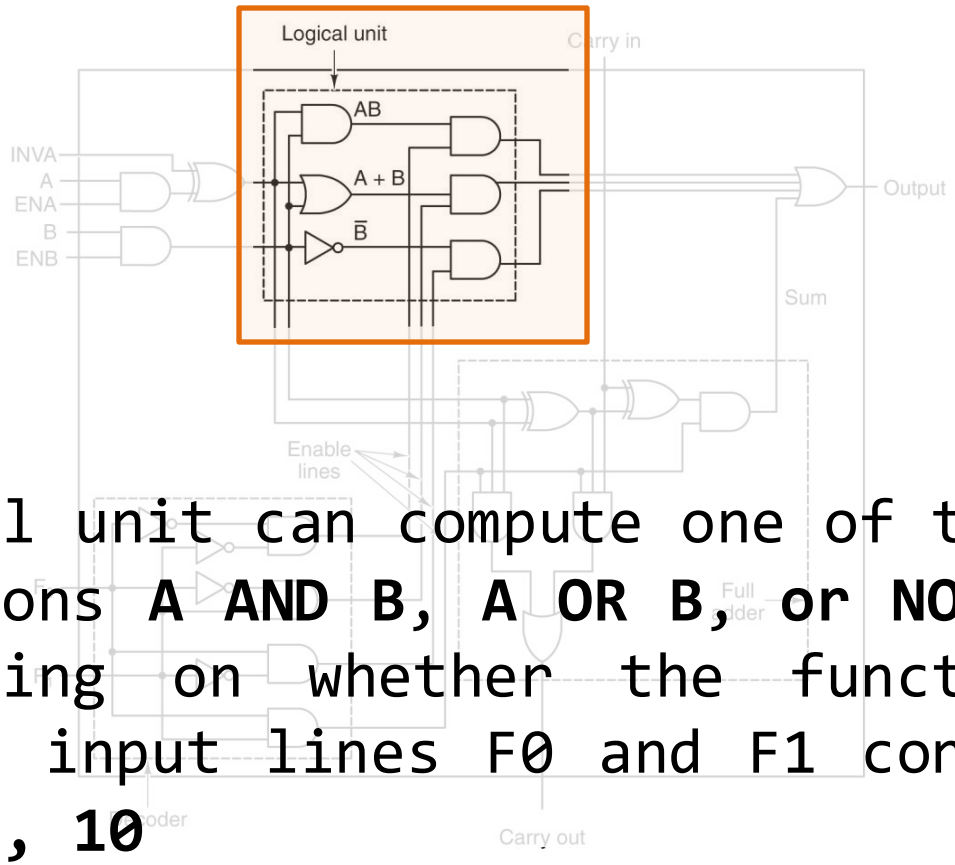
ENB	B **AND** ENB	ENA	A **AND** ENA	INVA	
1	B	1	A	1	**A'**
0	0	0	0	0	**A**

Identity law	1A = A	0 + A = A

Under normal conditions, ENA and ENB are both 1 to enable both inputs and INVA is 0

Logic gates

1-bit ALU(arithmetic logic unit)

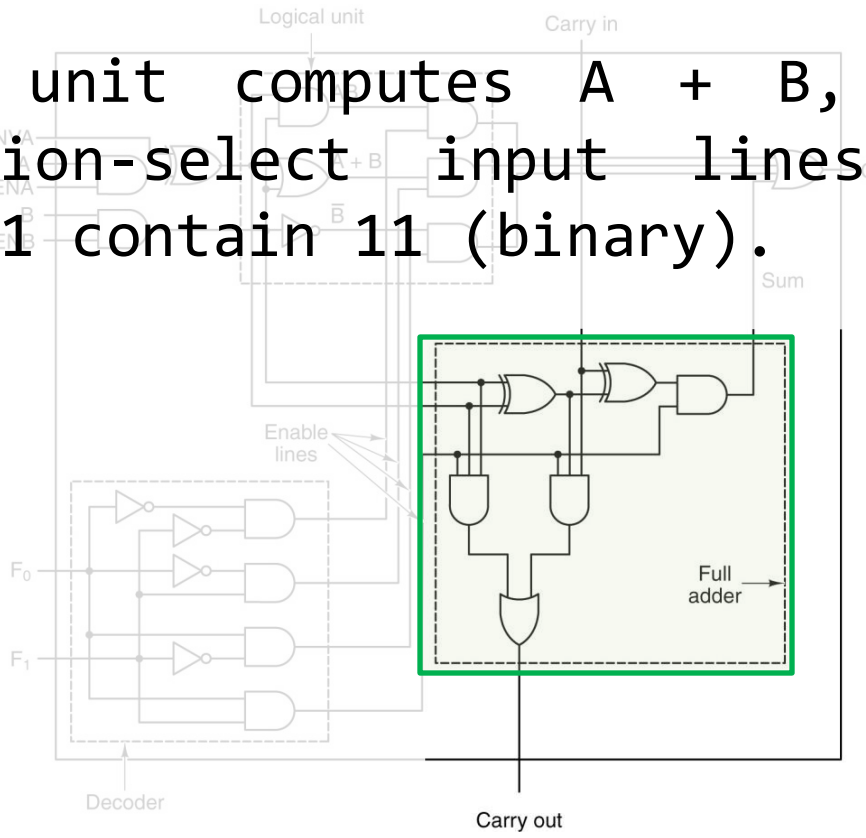


Logical unit can compute one of three functions **A AND B, A OR B, or NOT B,** depending on whether the function-select input lines F0 and F1 contain **00, 01, 10**

Logic gates

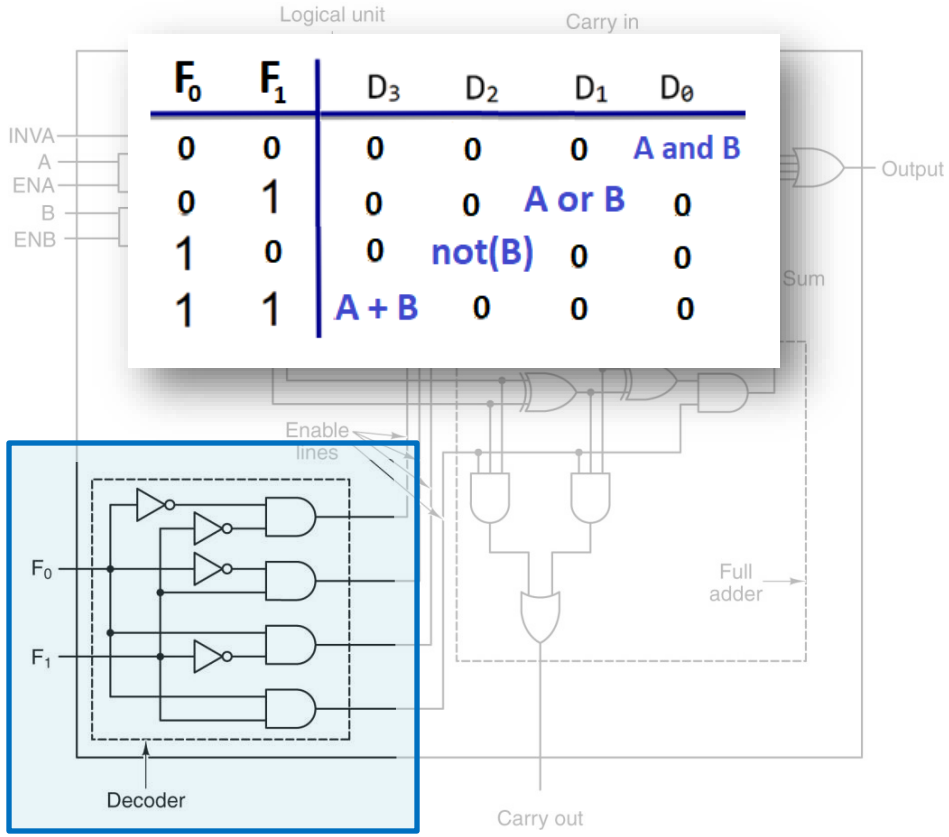
1-bit ALU(arithmetic logic unit)

This unit computes A + B, the function-select input lines F0 and F1 contain 11 (binary).



Logic gates

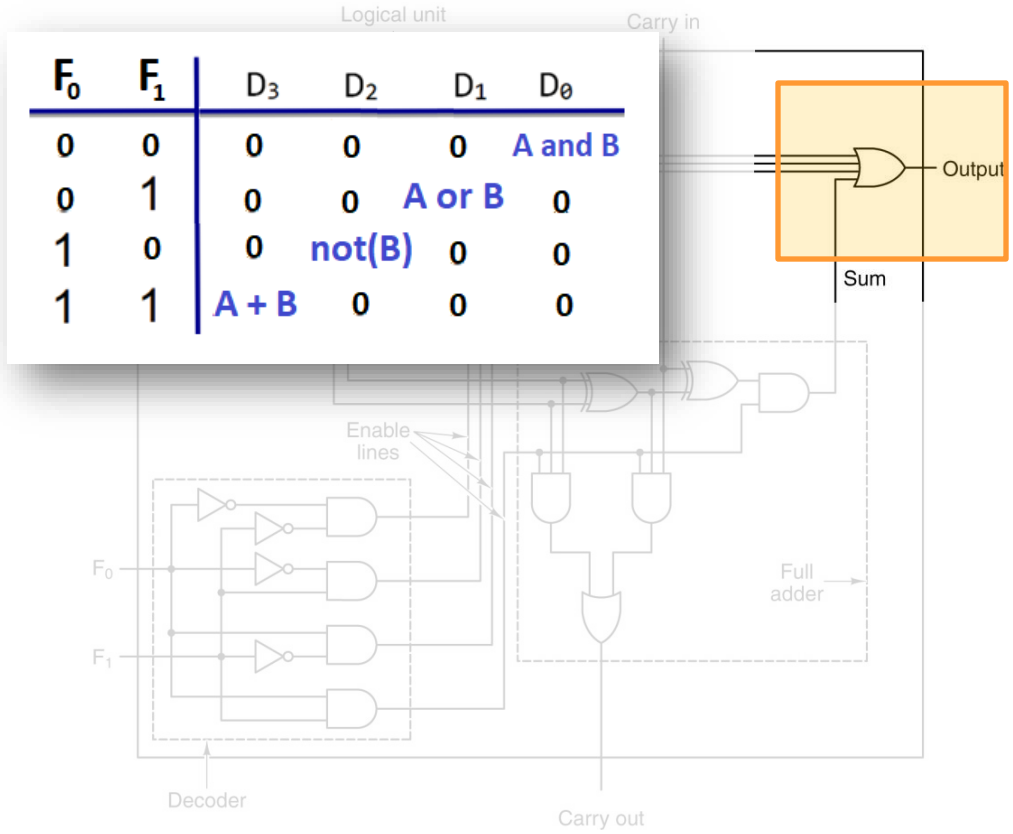
1-bit ALU(arithmetic logic unit)



F_0	F_1	D_3	D_2	D_1	D_0
0	0	0	0	0	A and B
0	1	0	0	A or B	0
1	0	0	not(B)	0	0
1	1	A + B	0	0	0

Logic gates

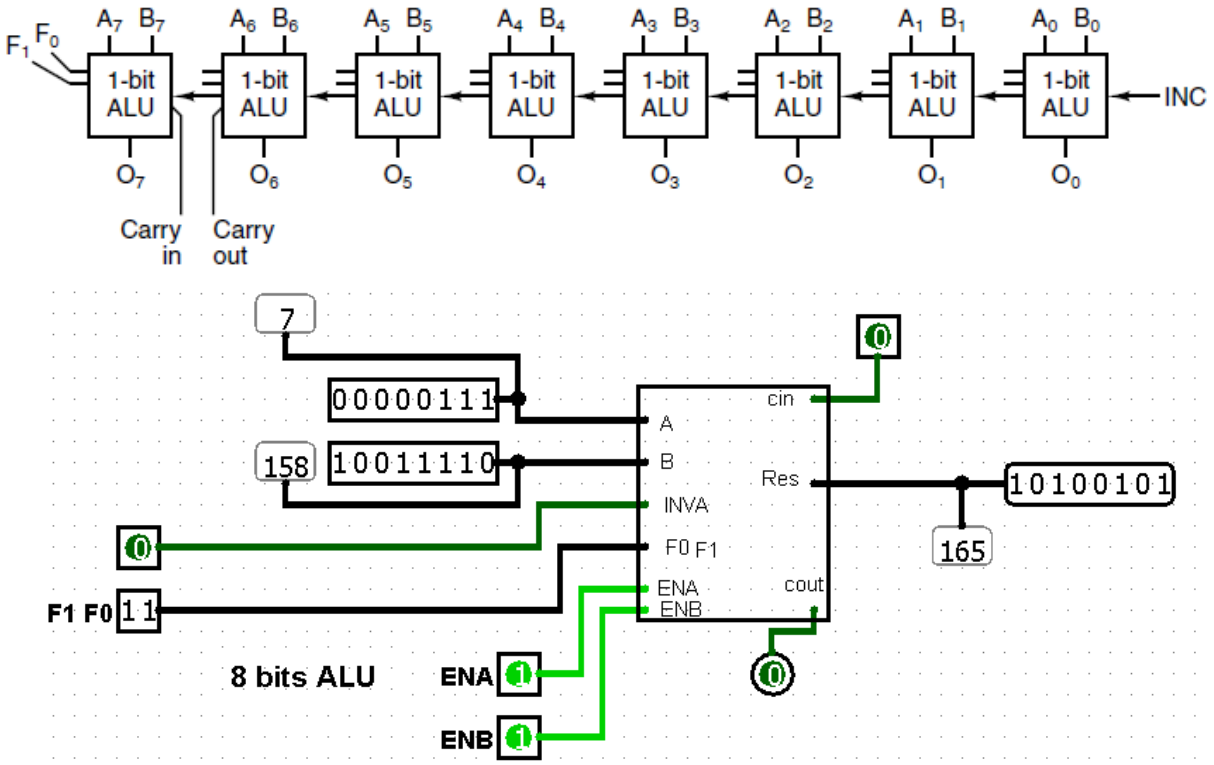
1-bit ALU(arithmetic logic unit)



F_0	F_1	D_3	D_2	D_1	D_0
0	0	0	0	0	A and B
0	1	0	0	A or B	0
1	0	0	not(B)	0	0
1	1	A + B	0	0	0

Logic gates

8-bit ALU



Logic gates

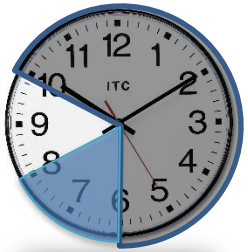
ALU (list of operations)

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

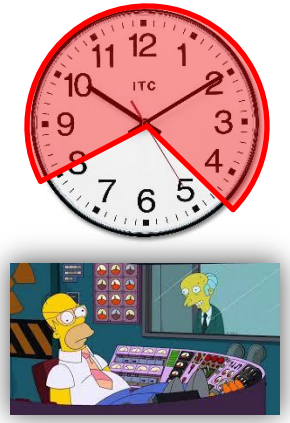
Logic gates

clocks

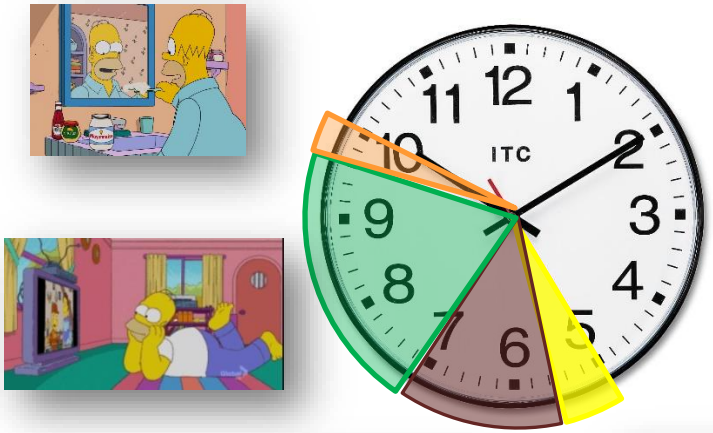
10:00 PM – 6:00 AM



8:00 AM – 4:30 PM



5:00 PM – 10:00 PM



5:00 PM – 10:00 PM

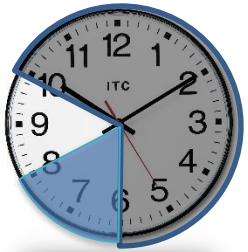
Clock:

- Rotation of Earth around its own axis takes around 24 hours (one cycle)
- Some tasks are completed in one cycle
- Some tasks take many cycles to be completed (ordering from Amazon)

Logic gates

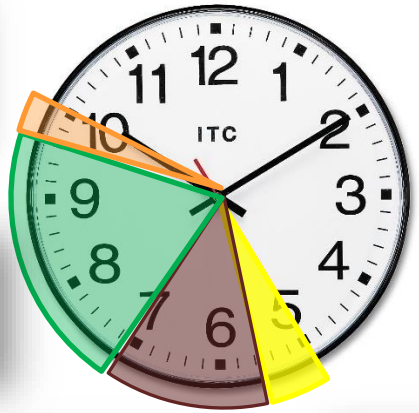
clocks

10:00 PM – 6:00 AM

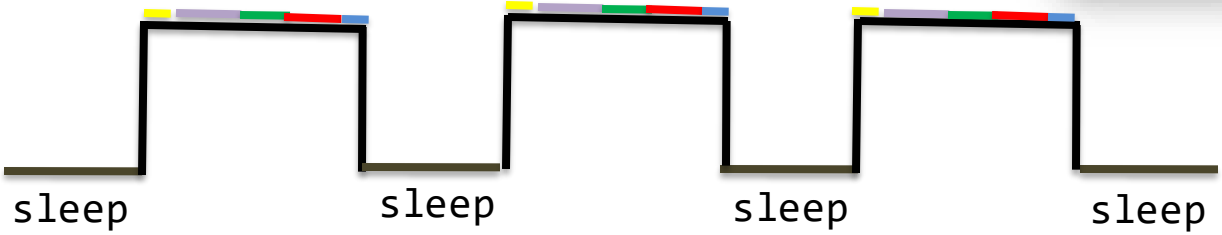


8:00 AM – 4:30 PM

5:00 PM – 10:00 PM



5:00 PM – 10:00 PM



Logic gates

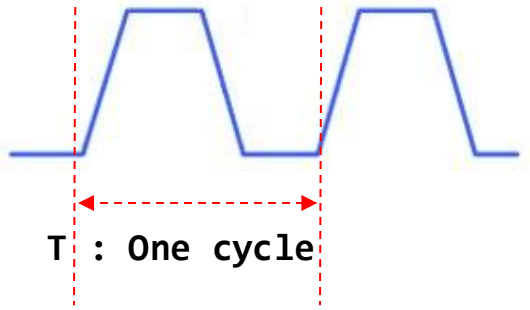
clocks

A clock in digital systems is a signal used to synchronize operations of various components within a digital circuit, such as a CPU or microcontroller.
It ensures that data is processed and transferred in an orderly and predictable manner.
A clock signal is a continuous, periodic waveform (typically a square wave) that oscillates between two levels:

High (1)
Low (0)

The clock signal has a constant **frequency** (f), meaning it completes a certain number of cycles per second, measured in Hertz (Hz). For example, a clock with a frequency of 1 GHz completes 1 billion cycles per second. The **period** (T), measured in seconds (s), represents the duration of one complete cycle.

$$f = \frac{1}{T} \quad \text{or} \quad T = \frac{1}{f}$$



Logic gates

clock

Duty Cycle, the percentage of time the clock signal stays high during one cycle. A typical duty cycle is 50%, meaning the clock is high for half of the cycle and low for the other half.

The clock synchronizes different components of the system, ensuring that operations happen in a coordinated manner. For example:

- **In a CPU**, the clock synchronizes fetching, decoding, and executing instructions.
- **In memory systems**, the clock ensures proper timing of read/write operations.

- **Pipeline Stages**: In pipelined systems (e.g., modern CPUs), the clock ensures that each pipeline stage progresses at the right time.

- **State Changes**: Digital circuits use flip-flops and registers that change state only on a clock edge (either rising edge or falling edge). This ensures predictable state transitions.

Logic gates

clock

There are several ways that digital circuits can be triggered by the clock:

a) Rising Edge Triggering

The circuit responds to the transition from 0 to 1 on the clock signal. Flip-flops or registers triggered by the rising edge change their state at the moment the clock signal goes from low to high. This is the most common method used in synchronous systems.

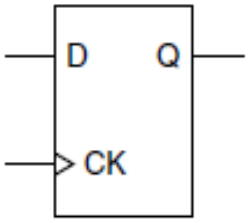
Example:

A positive-edge triggered D flip-flop captures the input value (D) on the rising edge of the clock and stores it until the next rising edge.

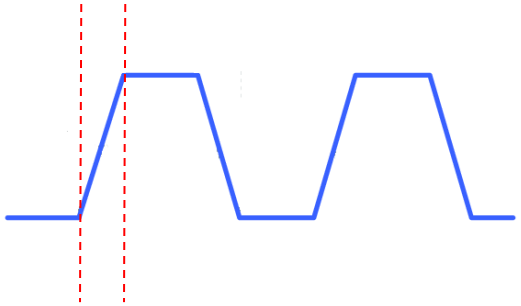
Symbol:

The small triangle on the clock input of a flip-flop symbol indicates positive-edge triggering.

Rising edge triggered flip flop



Rising edge



Logic gates

clock

b) Falling Edge Triggering

The circuit responds to the transition from 1 to 0 on the clock signal.

Flip-flops or registers triggered by the falling edge change their state at the moment the clock signal goes from high to low.

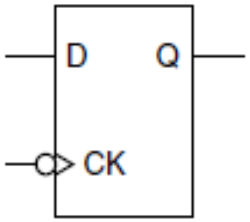
Example:

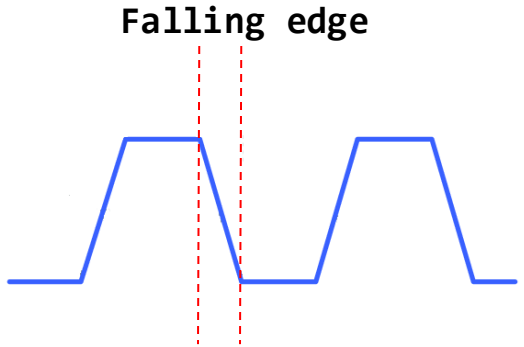
A negative-edge triggered D flip-flop captures the input value on the falling edge of the clock.

Symbol:

A small triangle with a bubble (inversion symbol) on the clock input of a flip-flop symbol indicates negative-edge triggering.

Falling edge triggered flip flop





Logic gates

clock

c) Level Triggering

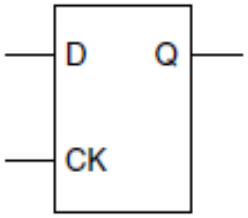
Instead of responding to an edge, level-triggered circuits respond when the clock is at a specific level:

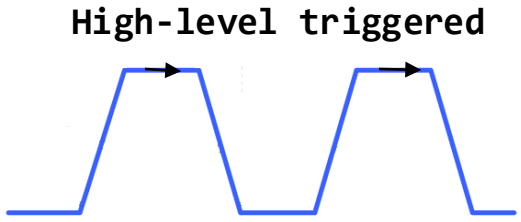
High-Level Triggering:

The circuit responds when the clock is high (logic 1). As long as the clock remains high, the circuit can continuously change state.

Example: Level-sensitive latches respond to a high clock level and store data as long as the clock stays high.

High-level triggered latch

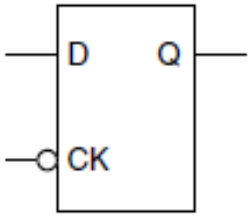


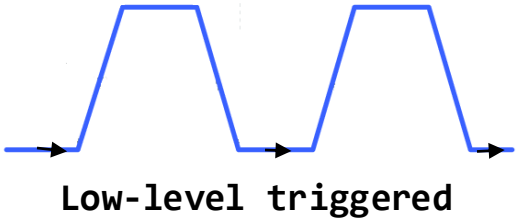


Low-Level Triggering:

The circuit responds when the clock is low (logic 0). As long as the clock remains low, the circuit remains sensitive to input changes.

Low-level triggered latch

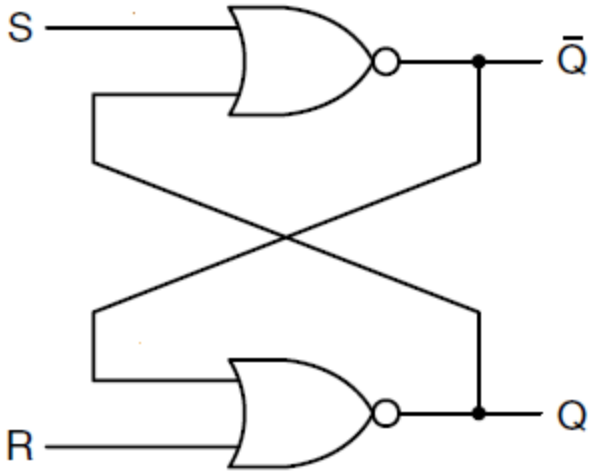




Logic gates

SR Latches

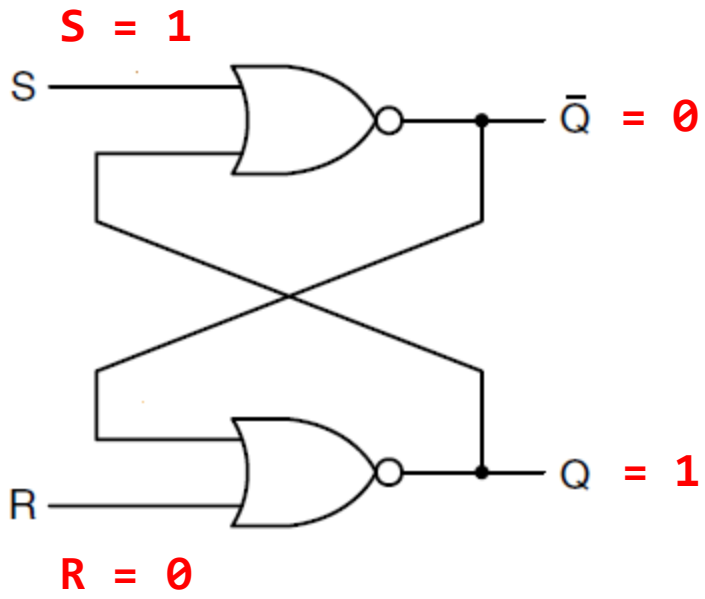
S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	0
Not valid		



SR Latch :
simplest form of memory
S : Set
R : Reset
S = R = 1 enters unstable state

Logic gates

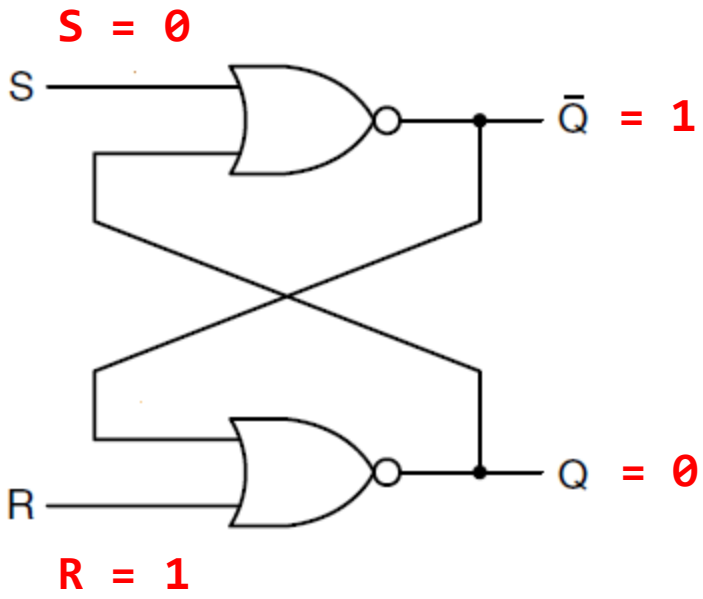
SR Latches



SR Latch (set):
S = 1 and R = 0 set Q = 1

Logic gates

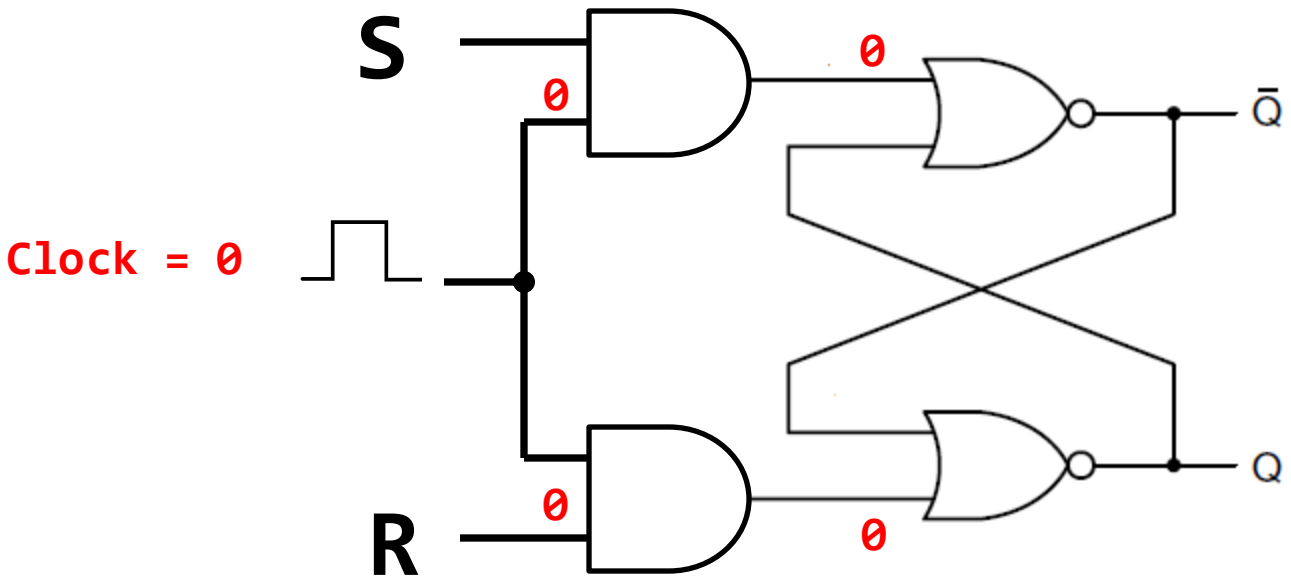
SR Latches



SR Latch (reset):
S = 0 and R = 1 set Q = 0

Logic gates

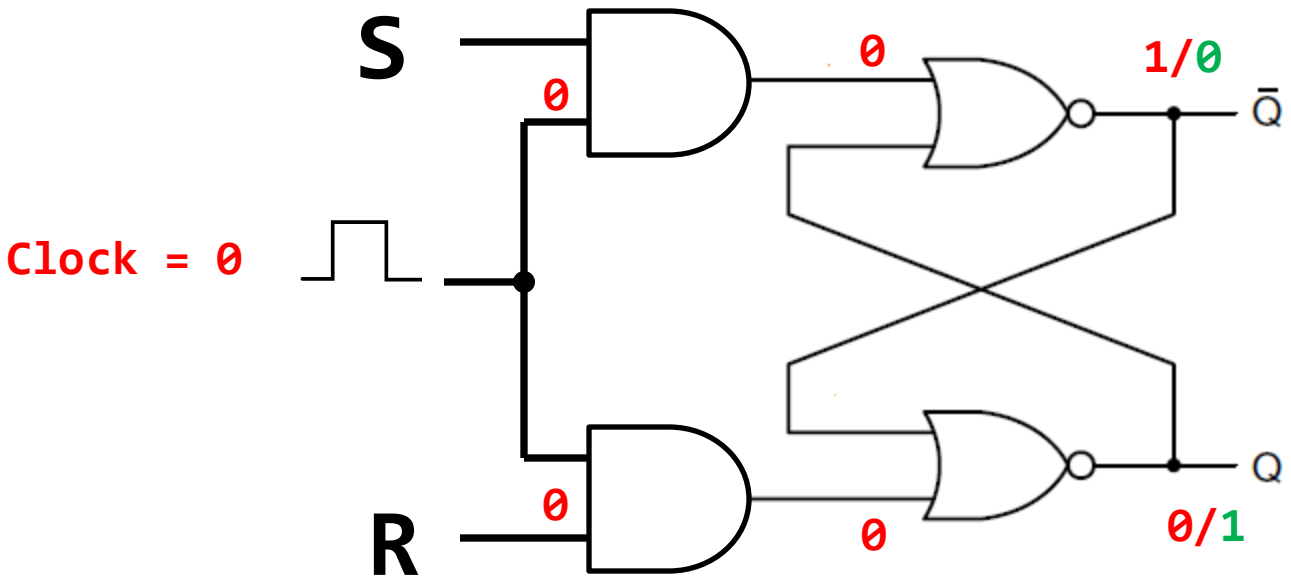
Clocked SR Latches



SR Latch : sensitive to change at any time
Clock is added to control loading of the latch at specific fixed intervals

Logic gates

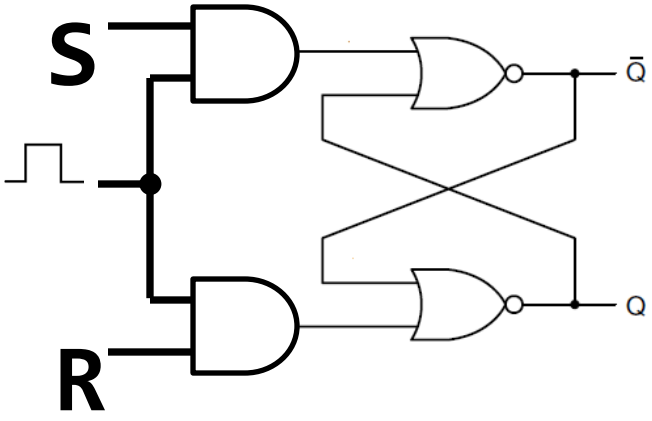
Clocked SR Latches

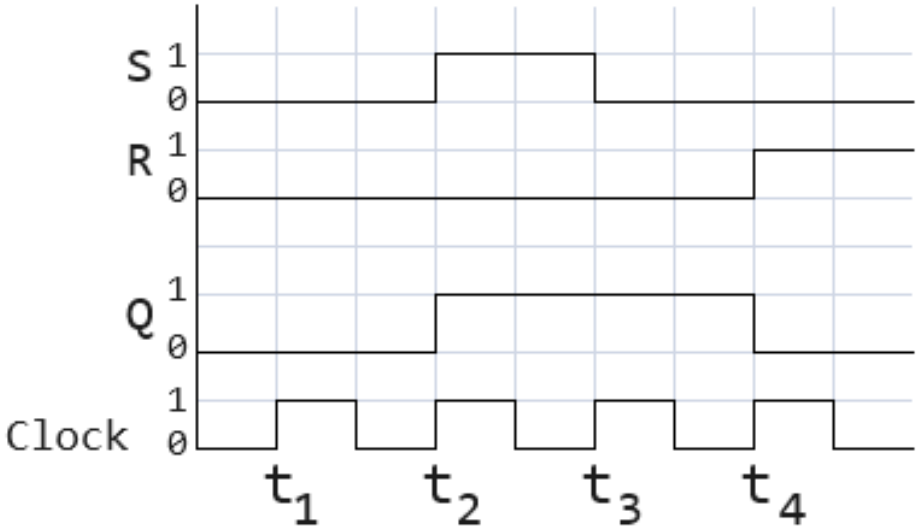


SR Latch : sensitive to change at any time
Clock is added to control loading of the latch at specific fixed intervals

Logic gates

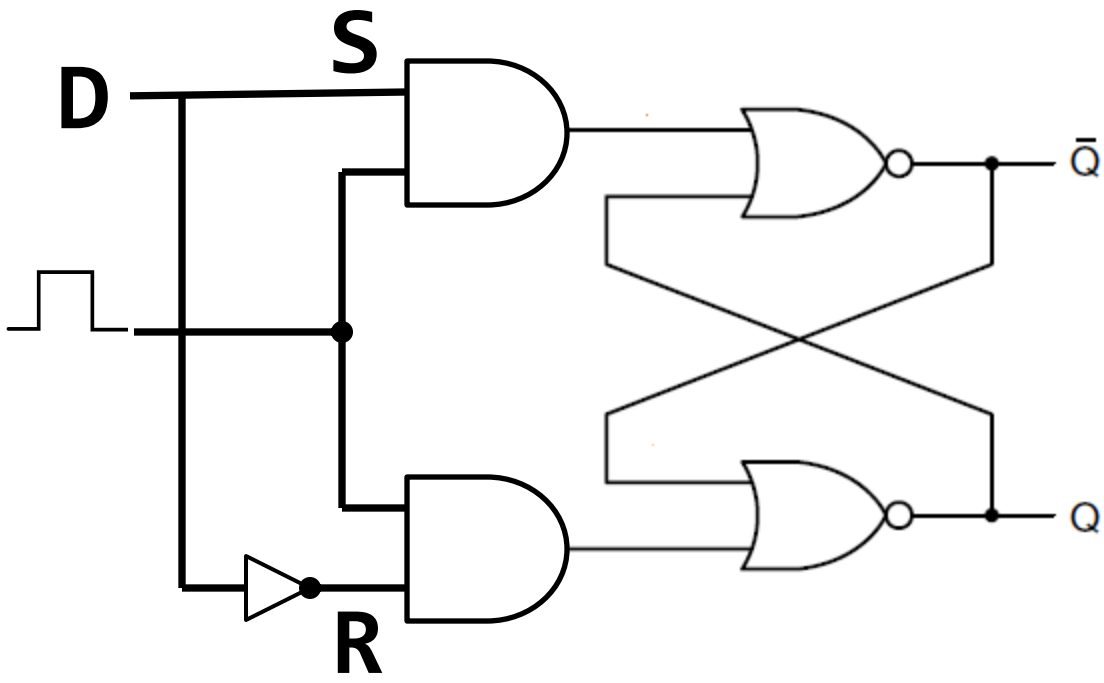
Clocked SR Latches

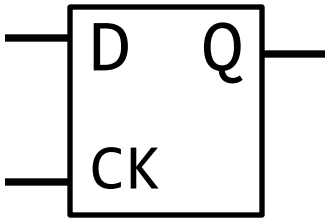




Logic gates

Clocked D Latches

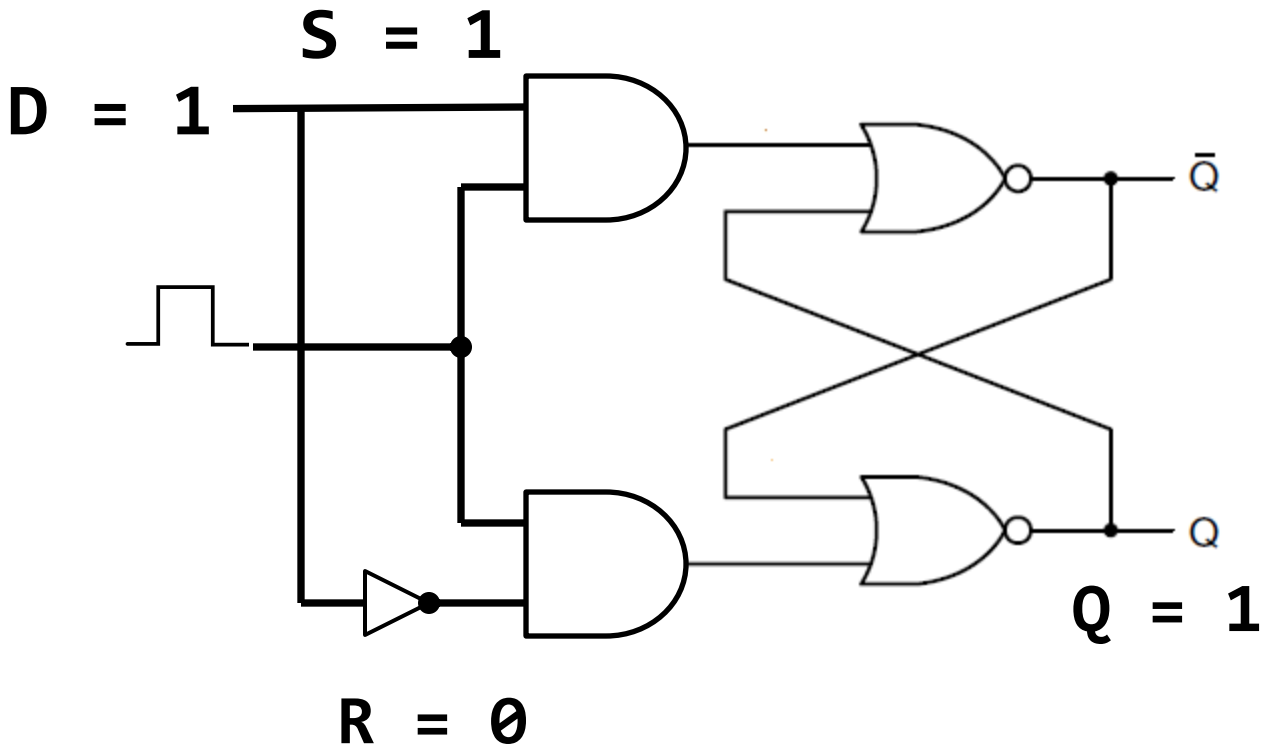




D Latch : to remove invalid state S = 1 and R = 1

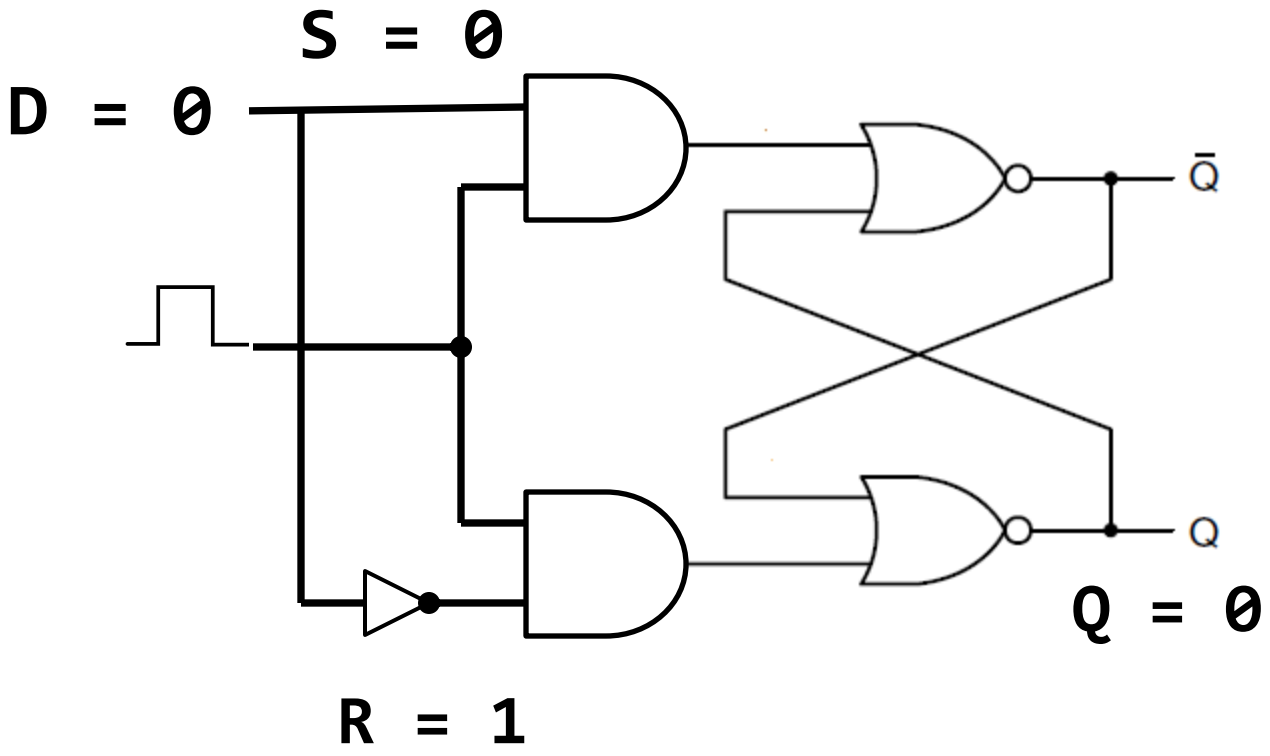
Logic gates

Clocked D Latches



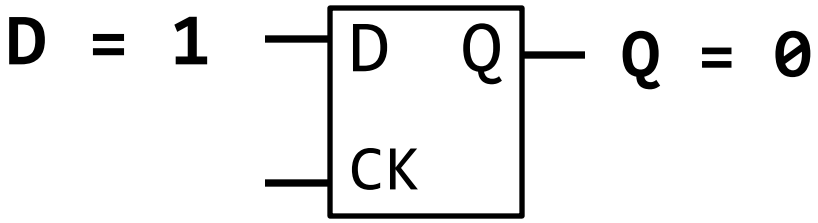
Logic gates

Clocked D Latches



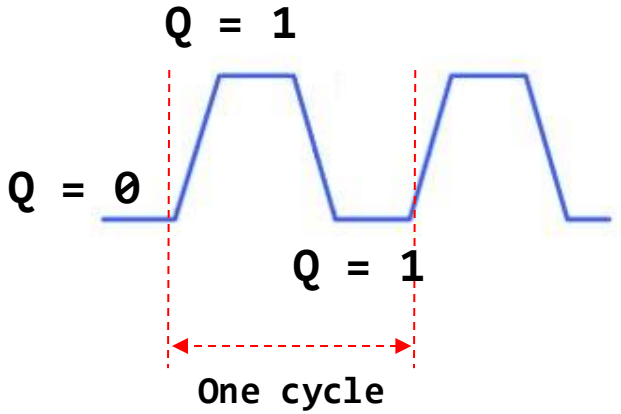
Logic gates

Clocked D Latches



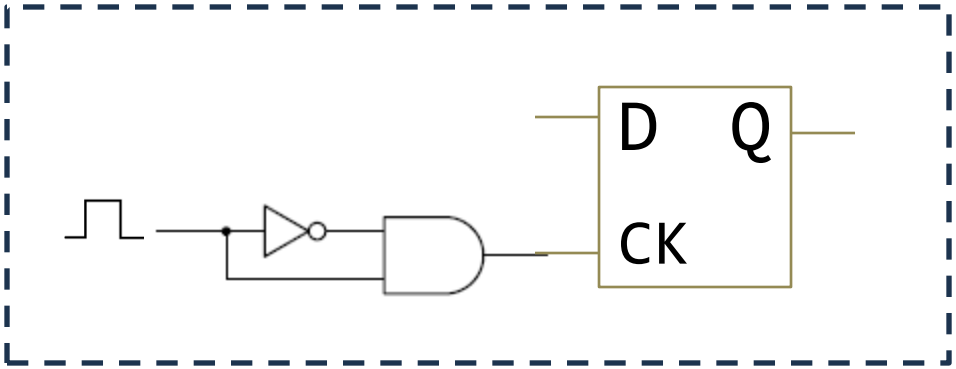
CK : 0 -> 1 (High-level triggered)

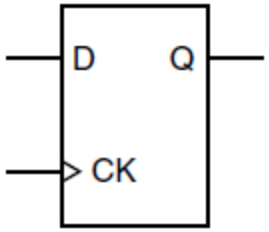
Q = 1



Logic gates

Edge-triggered Flip Flops

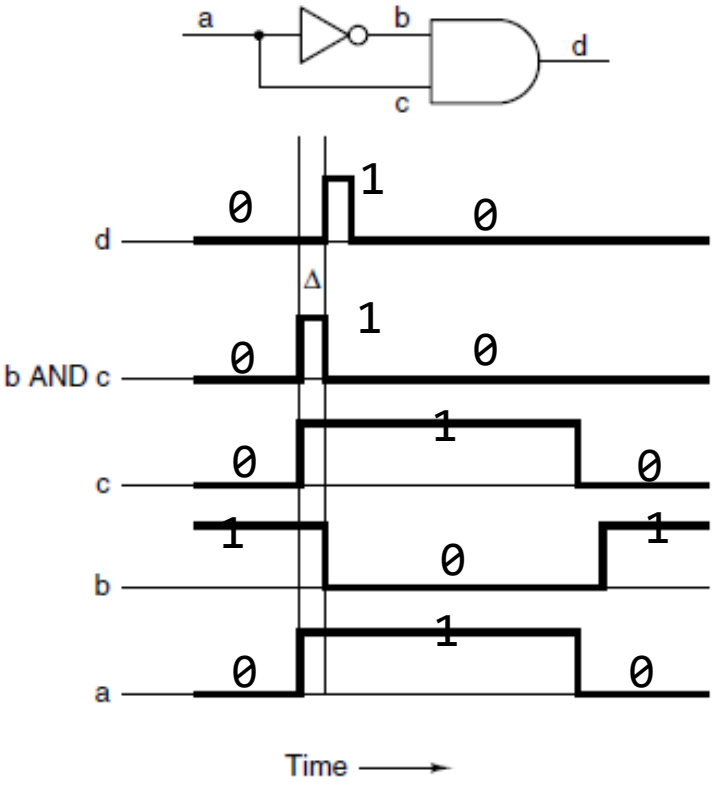


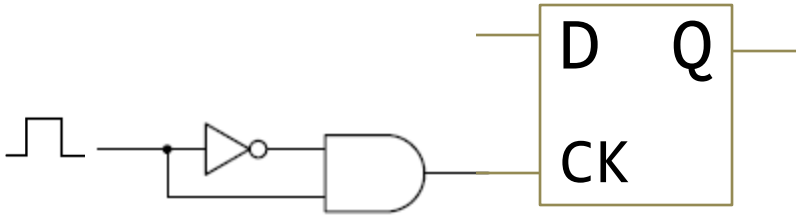


Rising edge triggered D flip-flop

Logic gates

Edge-triggered Flip Flops

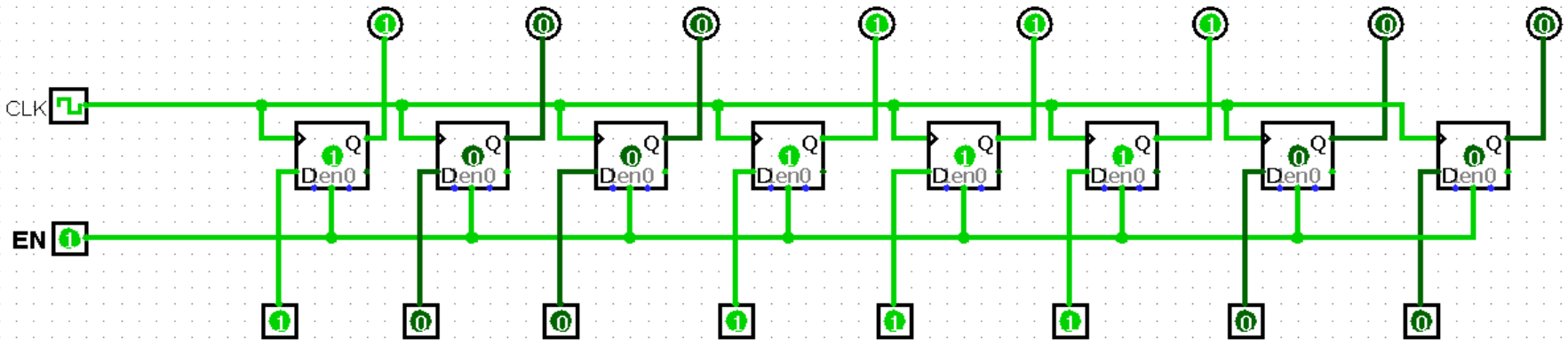


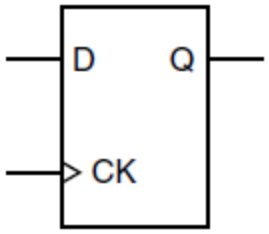


D Latch : loading the D latch at a single point in time

Logic gates

8 bits register



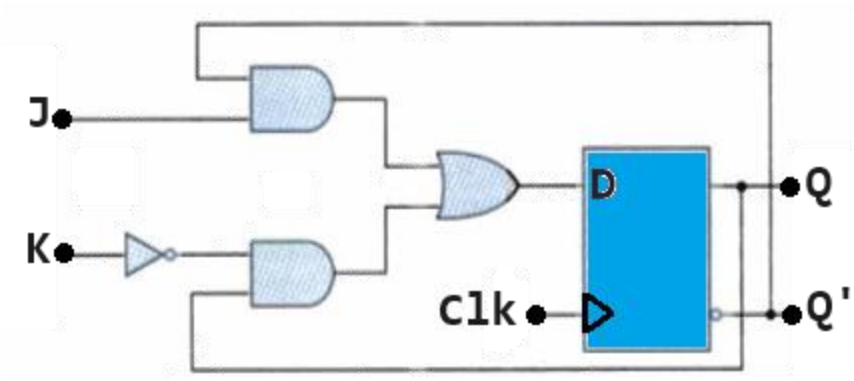


1-bit memory

Logic gates

JK Filp-Flops

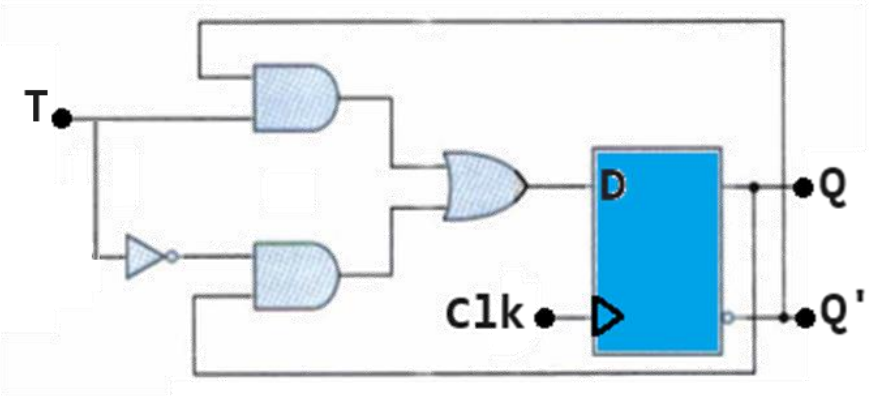
J	K	Q_{t+1}
0	0	Q_t
1	0	1
0	1	0
1	1	$\bar{Q}_t$



Logic gates

T(Toggle) Filp-Flops

J	K	Q_{t+1}
0	0	Q_t
1	1	$\bar{Q}_t$



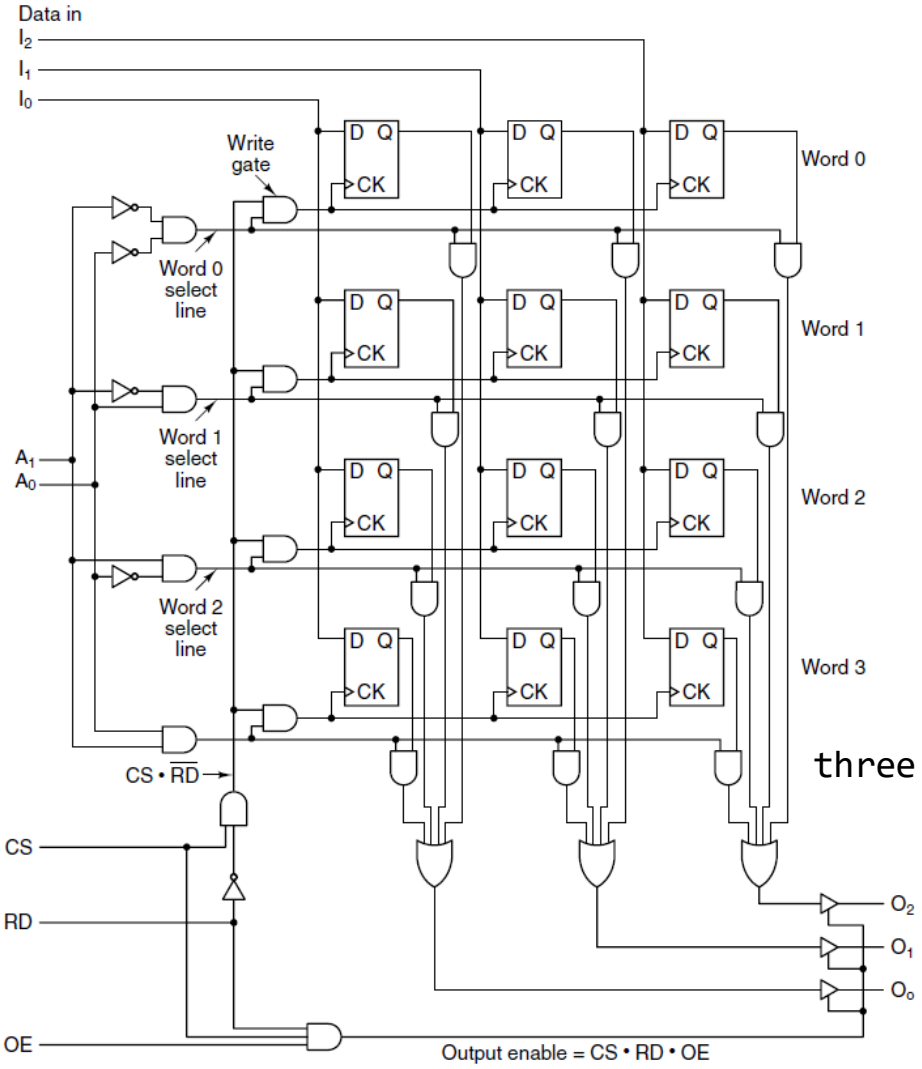
Logic gates

12-bit memory

Cs = 0-> 1 ->0
RD(write) = 0
OE(write) = 0

RD(read) = 1
OE(read) = 1
CS(read) = 1

Three inputs are data



Two are for the address

three outputs are for data

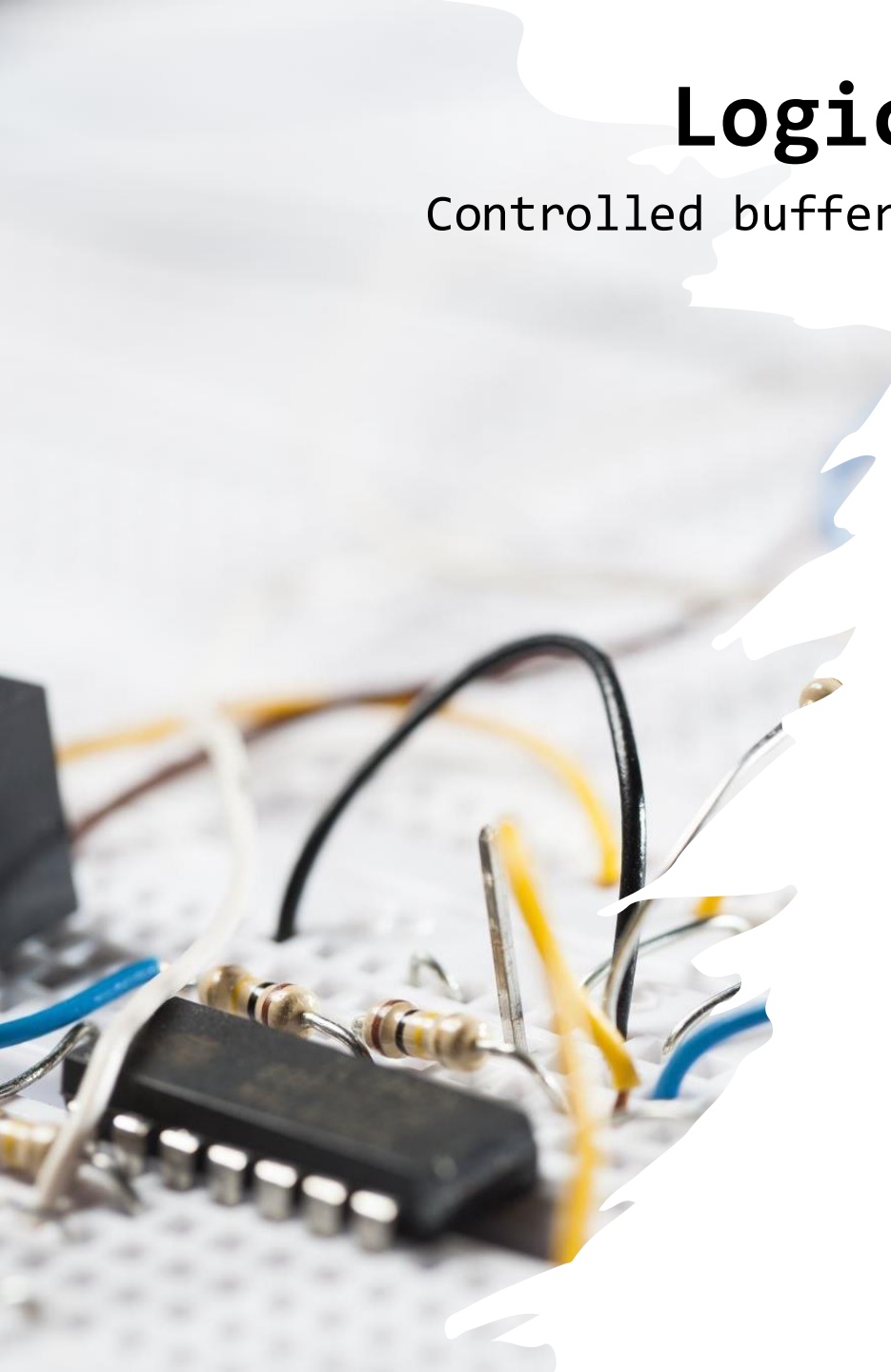
CS : for Chip Select

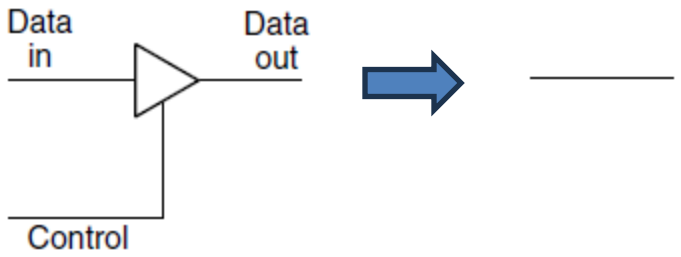
RD :for distinguishing between read and write

OE : Output Enable

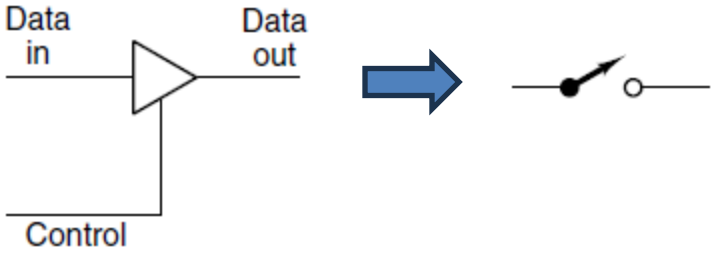
Logic gates

Controlled buffer (tri-state device)





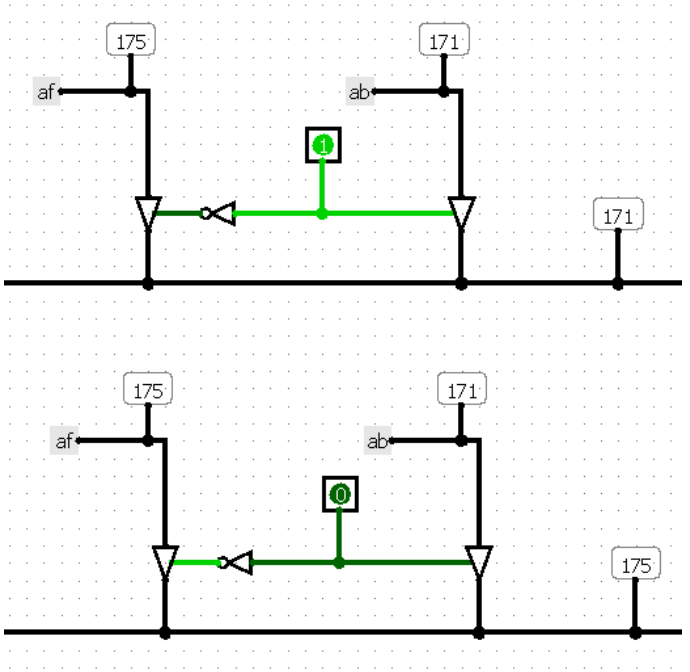
Control = High



Control = Low

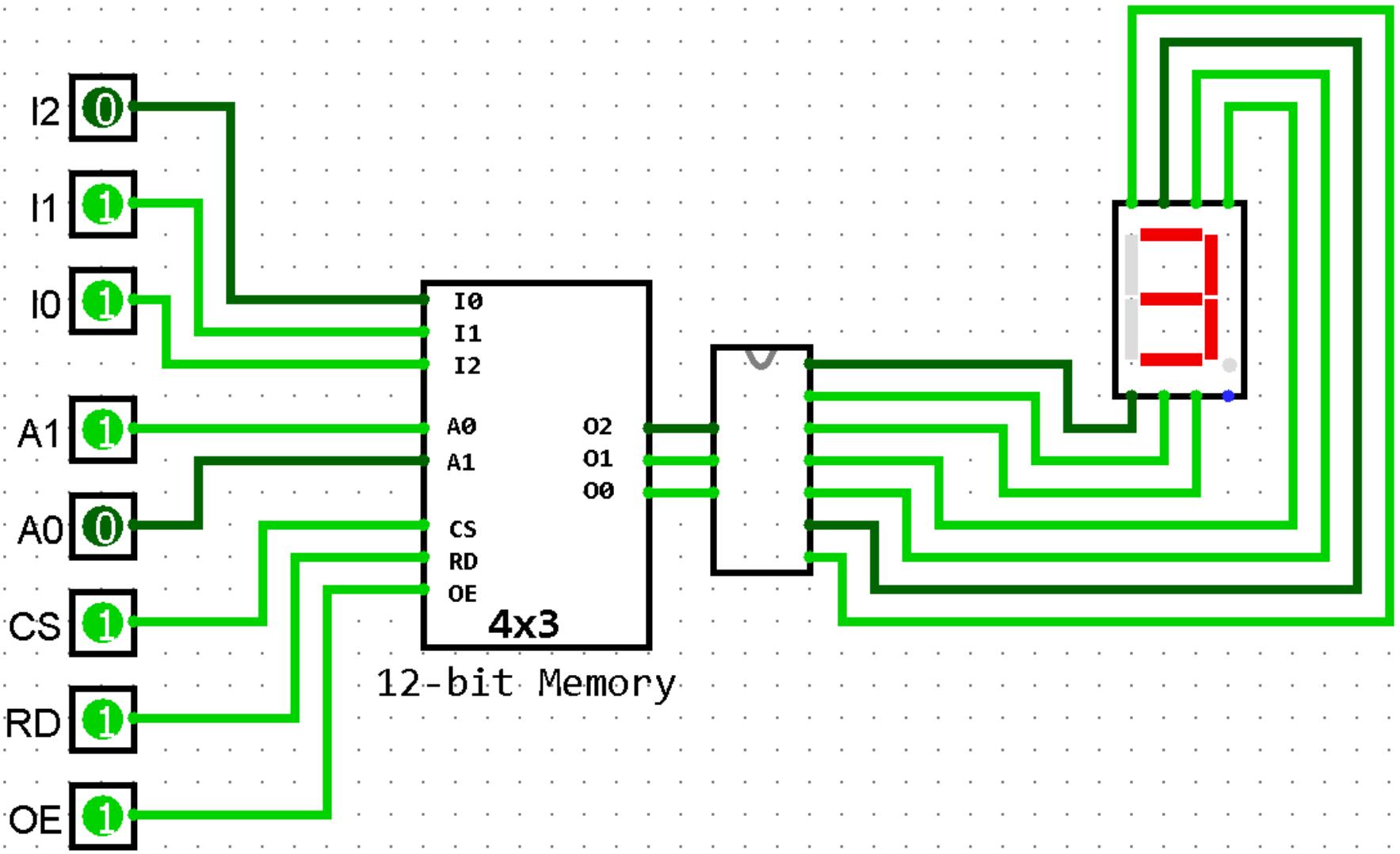
Logic gates

Controlled buffer (tri-state device)



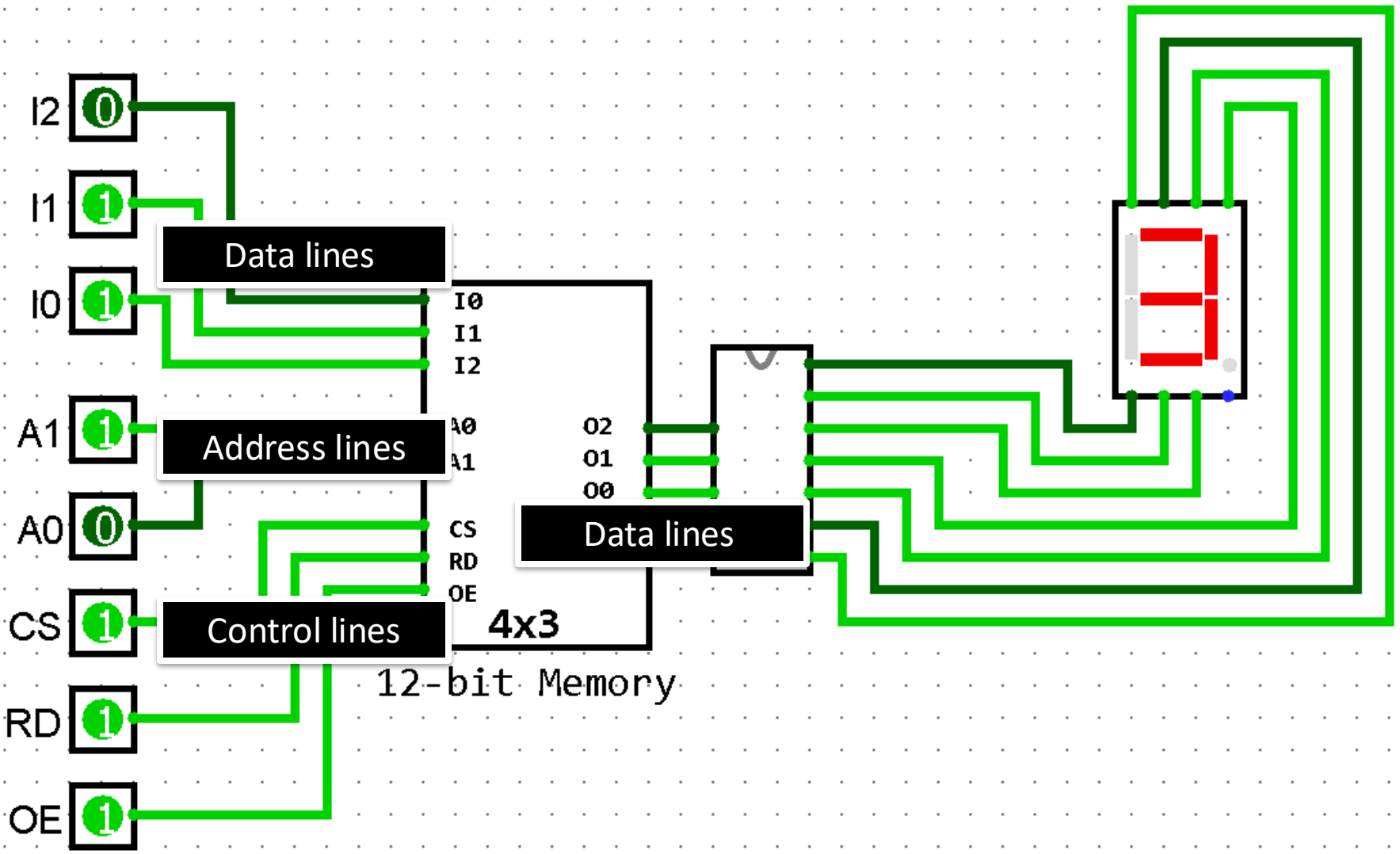
Logic gates

12-bit memory



Logic gates

12-bit memory



Logic gates

Reading materials

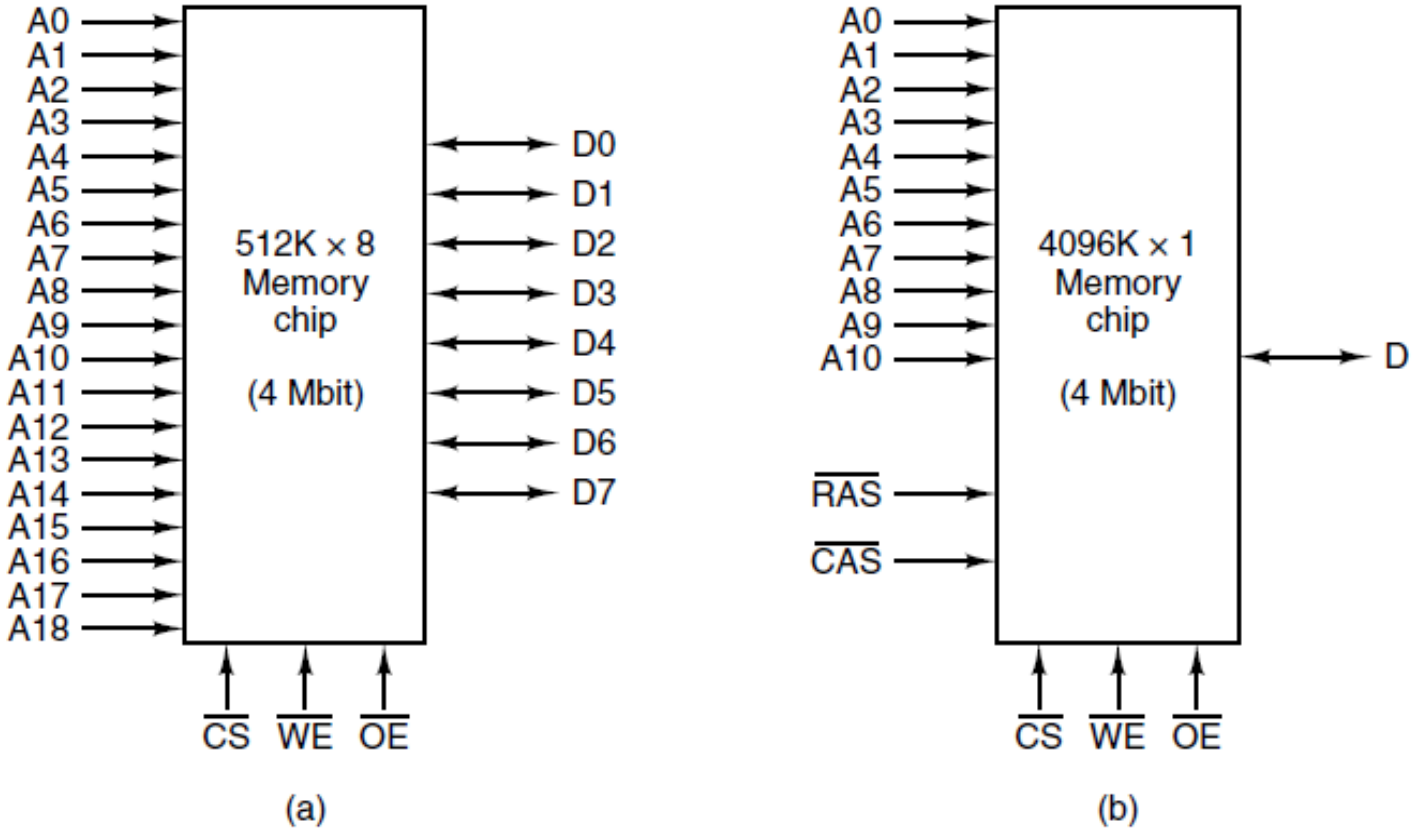
Getting back to the memory circuit, it should now be clear what the three non-inverting buffers on the data output lines are for. When CS, RD, and OE are all high, the output enable signal is also high, enabling the buffers and putting a word onto the output lines. When any one of CS, RD, or OE is low, the data outputs are disconnected from the rest of the circuit.

3.3.5 Memory Chips

Logic gates

Reading materials

4-Mbit memory

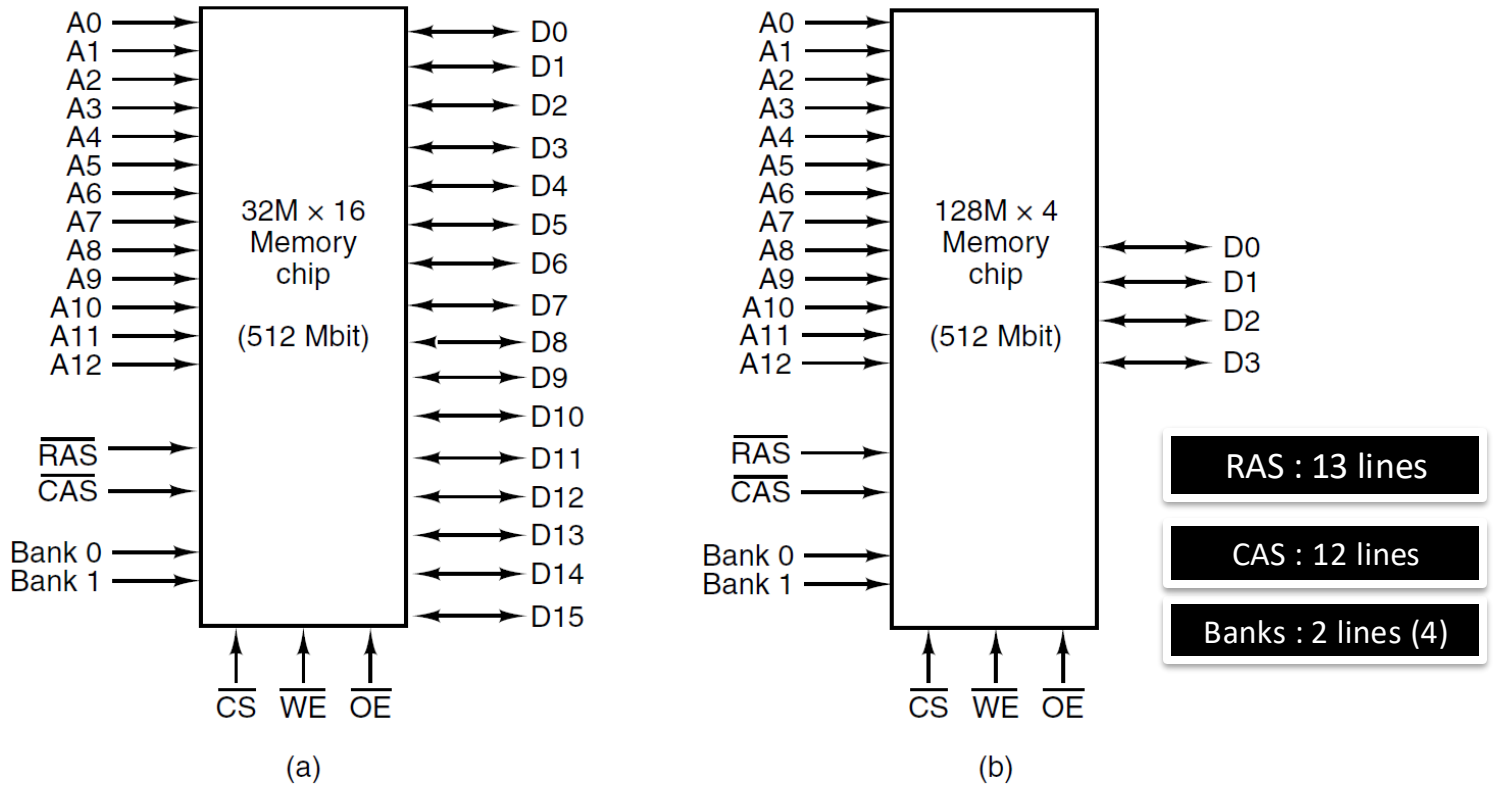


Two ways of organizing a 4-Mbit memory chip.

Logic gates

Reading materials

512-Mbit memory



RAS : 13 lines

CAS : 10 lines

Banks : 2 lines (4)

RAS : 13 lines

CAS : 12 lines

Banks : 2 lines (4)

Two ways of organizing a 512-Mbit memory chip.

Logic gates

RAMs and ROMs

RAM (Random Access Memories):

1. Static(SRAM)
   I. Based on circuits similar to basic D flip-flop
   II. Very fast
   III. A typical access time is on the order of a nanosecond or less, static RAMS are popular as cache memory
   IV. Their contents are retained as long as the power is kept on

2. Dynamic(DRAM)
   I. An array of cells, each cell containing one transistor and a tiny capacitor(charged 1 and discharged 0)
   II. Must be refreshed (reloaded) every few milliseconds to prevent the data from leaking away
   III. Because external logic must take care of the refreshing, dynamic RAMs require more complex interfacing than static ones
   IV. Have a very high density (many bits per chip), e.g. main memories

Logic gates

RAMs and ROMs

ROM (Read Only Memories):

I. The data in a ROM are inserted during its manufacture

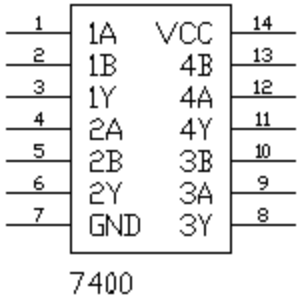
II. ROMs are much cheaper than RAMs

II. Neither the program nor the data are ever changed

Type	Category	Erasure	Byte alterable	Volatile	Typical use
SRAM	Read/write	Electrical	Yes	Yes	Level 2 cache
DRAM	Read/write	Electrical	Yes	Yes	Main memory (old)
SDRAM	Read/write	Electrical	Yes	Yes	Main memory (new)
ROM	Read-only	Not possible	No	No	Large-volume appliances
PROM	Read-only	Not possible	No	No	Small-volume equipment
EPROM	Read-mostly	UV light	No	No	Device prototyping
EEPROM	Read-mostly	Electrical	Yes	No	Device prototyping
Flash	Read/write	Electrical	No	No	Film for digital camera

A comparison of various memory types.

Logic gates
Integrated circuit (IC)



7400