Computer Architecture
Lecture 4

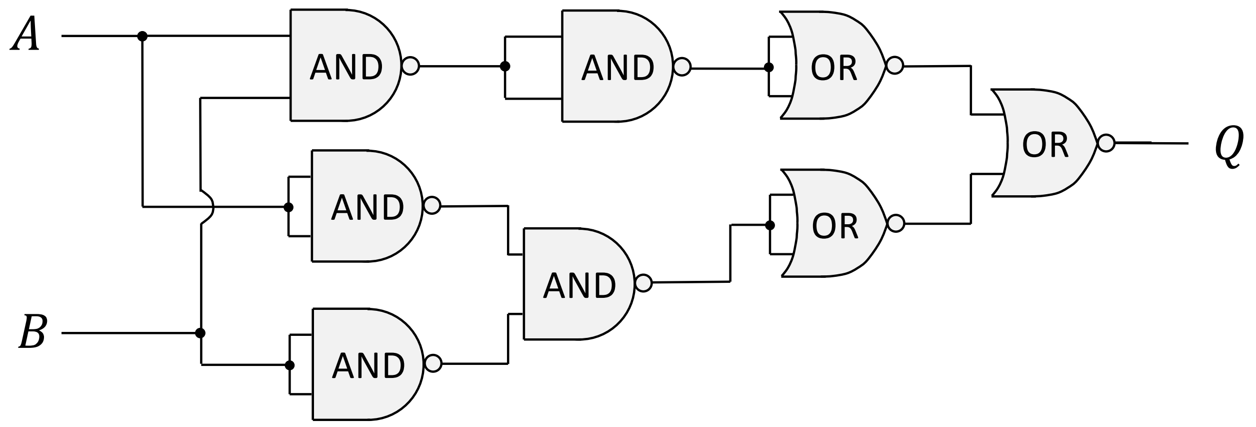
Combinational Logic Circuits (Cont.)

Artem Burmyakov, Alexander Tormasov

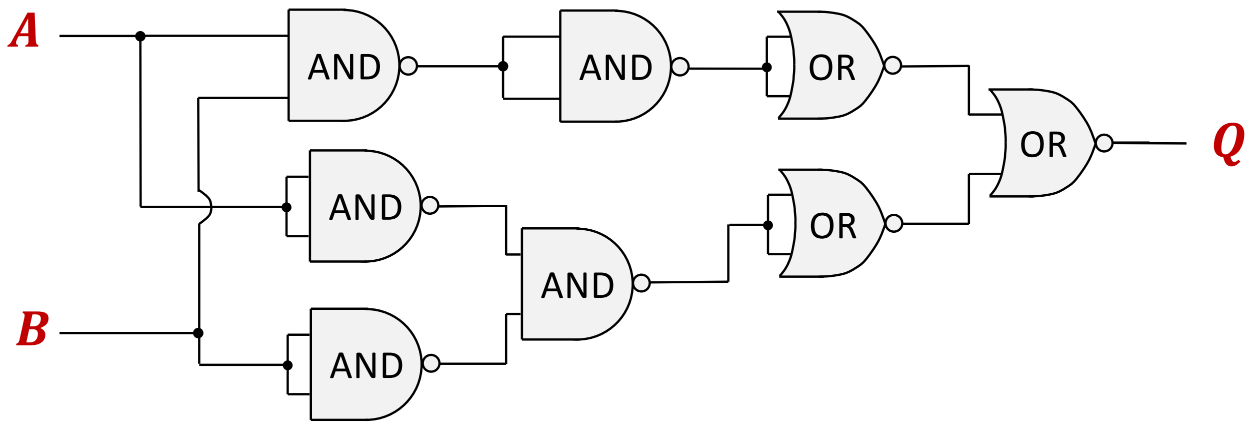
September 16, 2021

Recap: Characteristics of Combinational Logic Circuits

Sample combinational logic circuit:

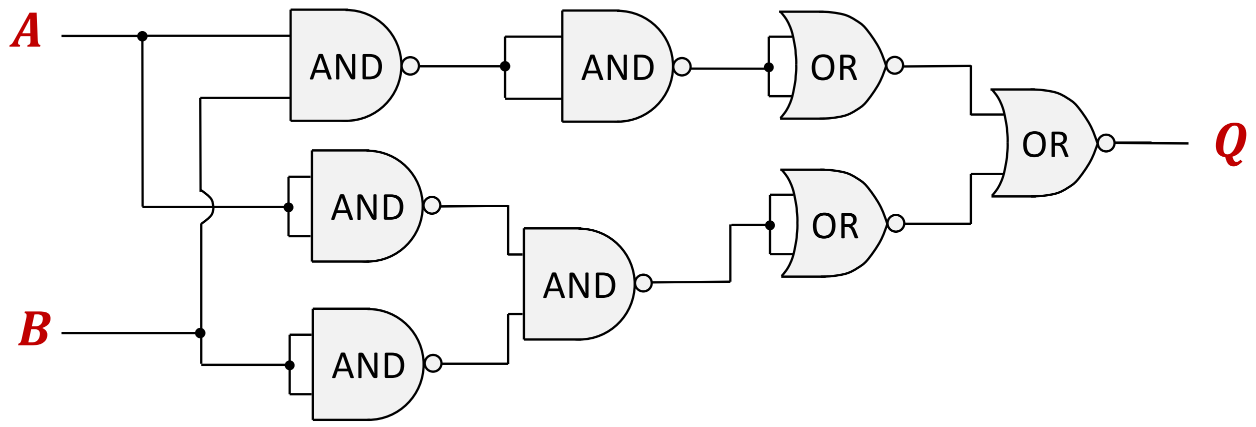


Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on "current" input signals

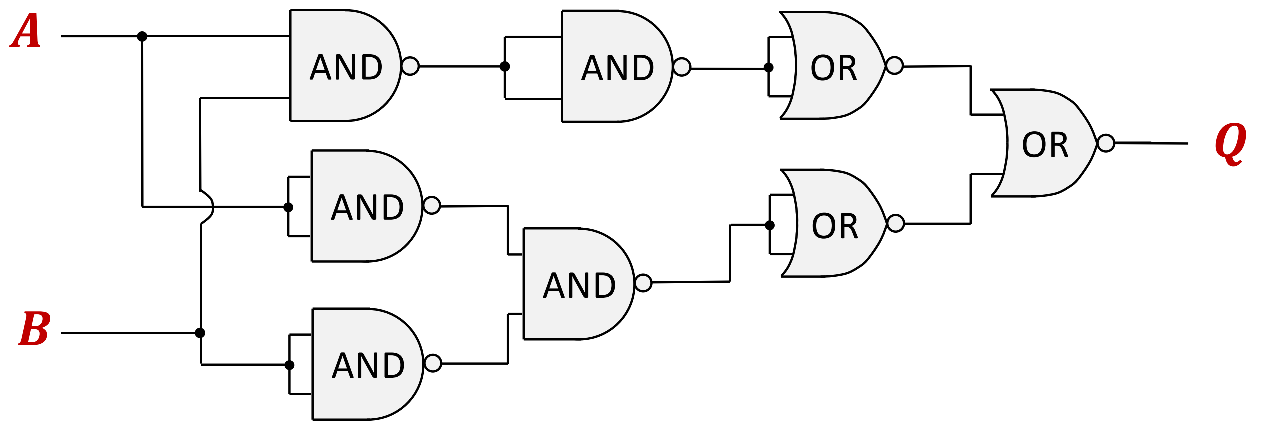
Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on "current" input signals

"Combinational circuit" = combine inputs and get the output

Recap: Characteristics of Combinational Logic Circuits

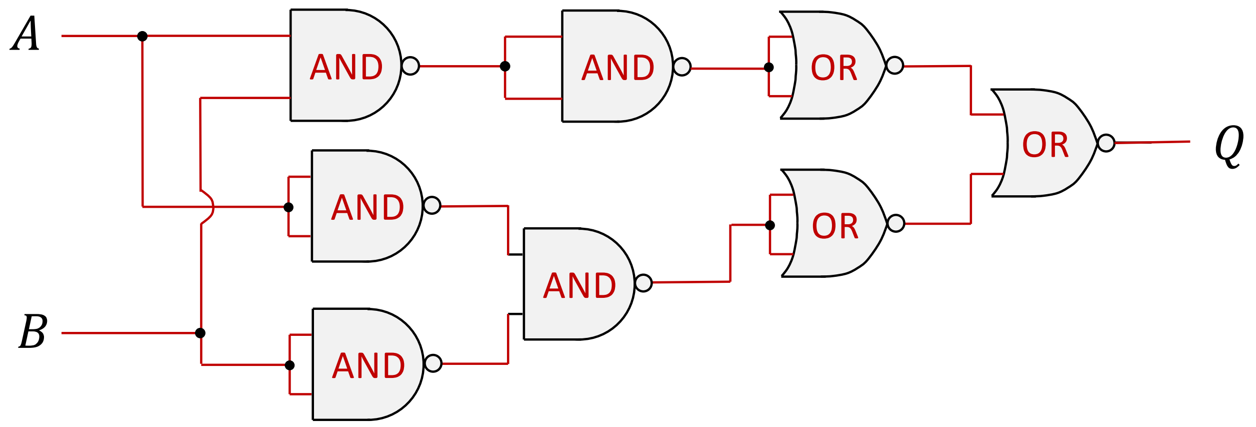


1) An output signal depends just on "current" input signals

"Combinational circuit" = combine inputs and get the output

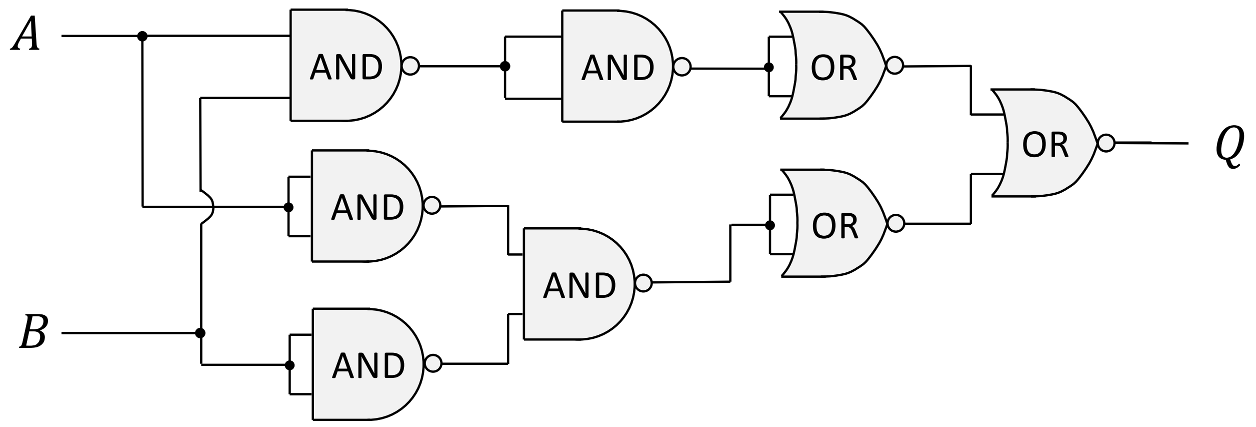
There is no dependency on previous signals or computation results

Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on “current” input signals
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them

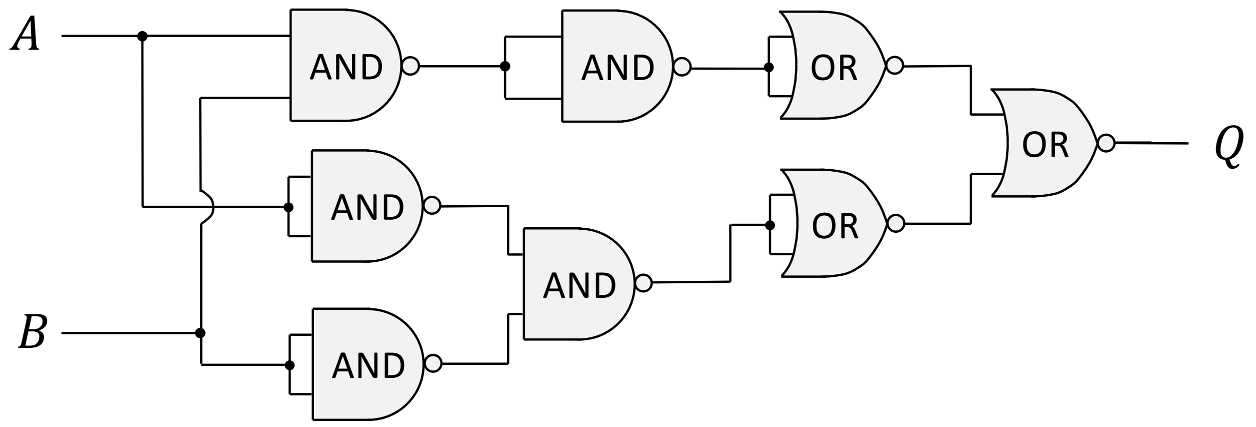
Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on “current” input signals
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them

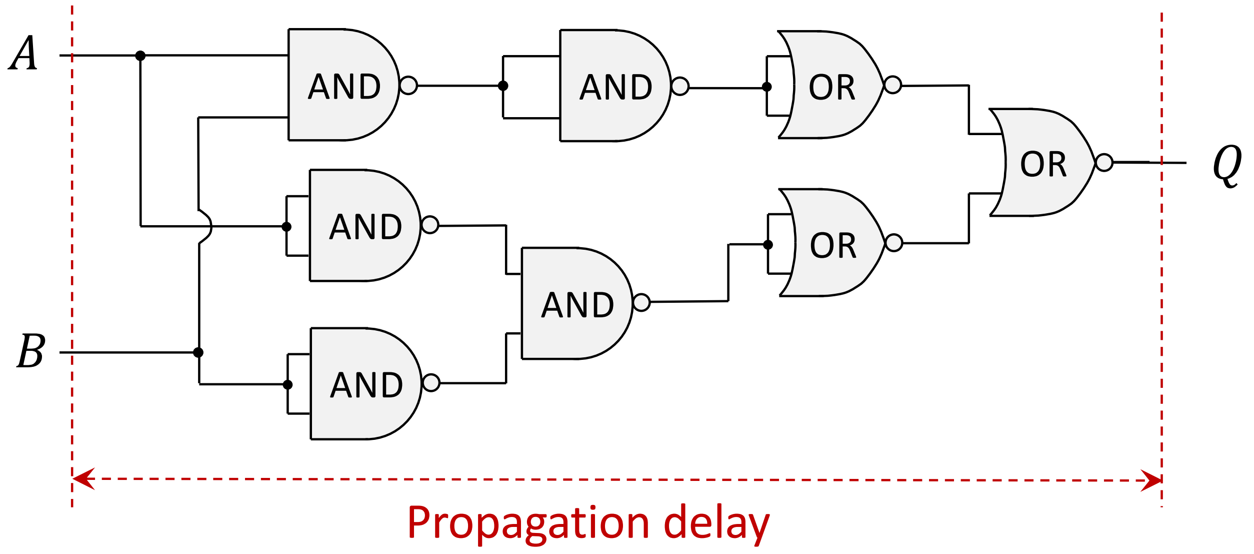
Observation: “shorter” wires are better, e.g. due to lower propagation delays (there is some connection to Moore’s law)

Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on “current” input signals
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them
3) Output signal is updated after the change of input signals, subject to propagation delay

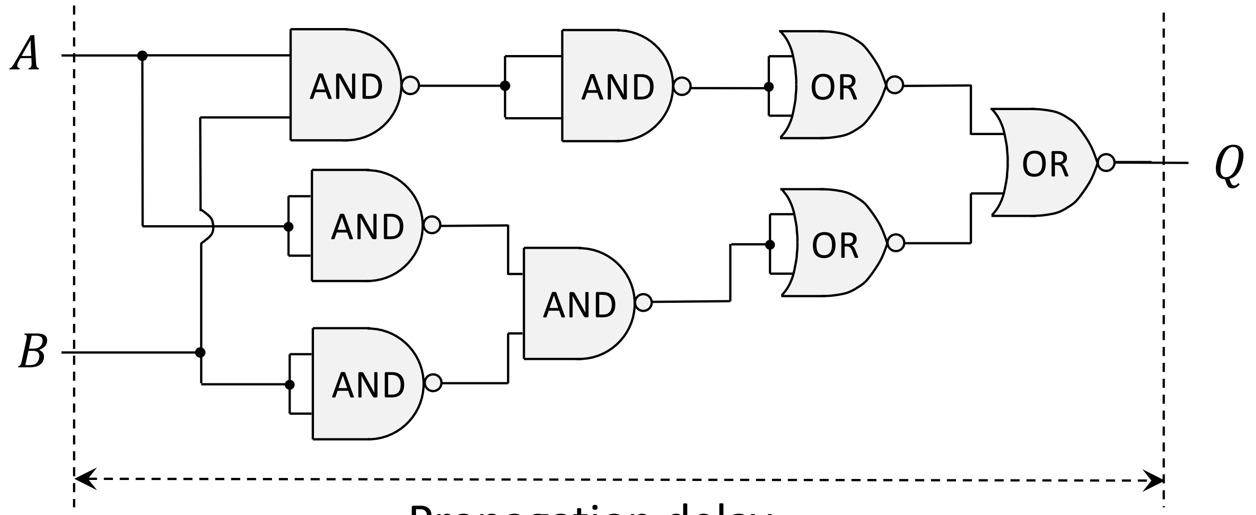
Recap: Characteristics of Combinational Logic Circuits



3) Output signal is updated after the change of input signals, subject to propagation delay

Propagation delay – time delay between the change of inputs and the update of output

Recap: Characteristics of Combinational Logic Circuits



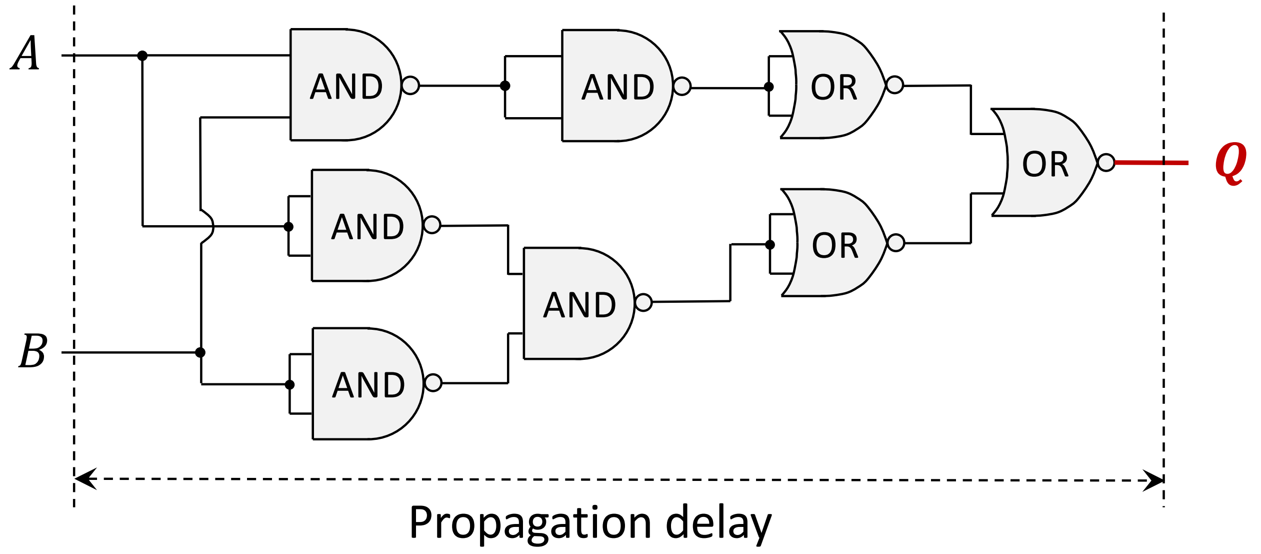
Propagation delay

3) Output signal is updated after the change of input signals, subject to propagation delay

   Propagation delay – time delay between the change of inputs and the update of output

   Critical Path of the Integrated Circuit – the one taking the longest time (results in the "worst case" propagation delay)

Recap: Characteristics of Combinational Logic Circuits



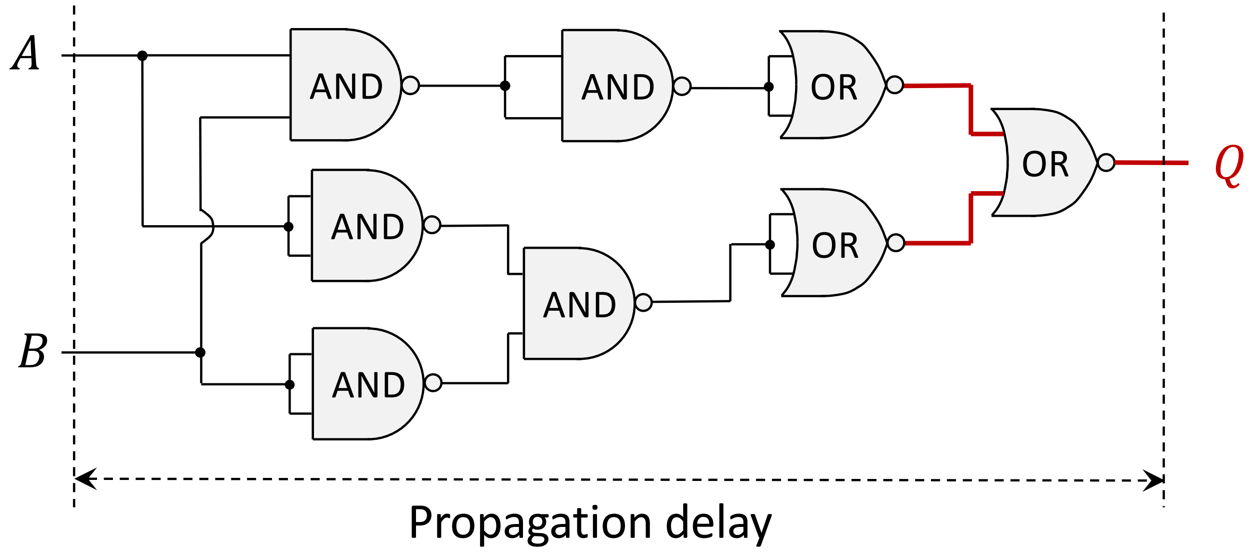
3) Output signal is updated after the change of input signals, subject to propagation delay

   Propagation delay – time delay between the change of inputs and the update of output

   Critical Path of the Integrated Circuit – the one taking the longest time
   (results in the "worst case" propagation delay)

   Computed in the direction starting from the output pin(s)

Recap: Characteristics of Combinational Logic Circuits



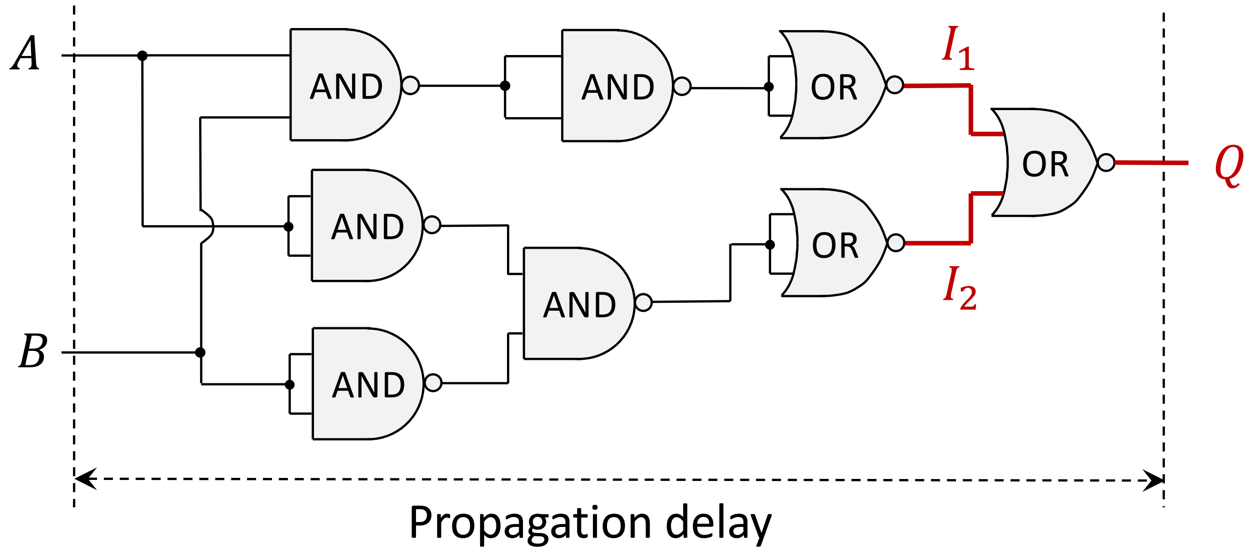
3) Output signal is updated after the change of input signals, subject to propagation delay

Propagation delay – time delay between the change of inputs and the update of output

Critical Path of the Integrated Circuit – the one taking the longest time
(results in the “worst case” propagation delay)

Computed from the output pin(s)

Recap: Characteristics of Combinational Logic Circuits



Q is updated when both inputs, I_1 and I_2, are updated;

Inputs I_1 and I_2 are in turn outputs of other logic gates;

Repeat backtracking, to compute the worst case delay

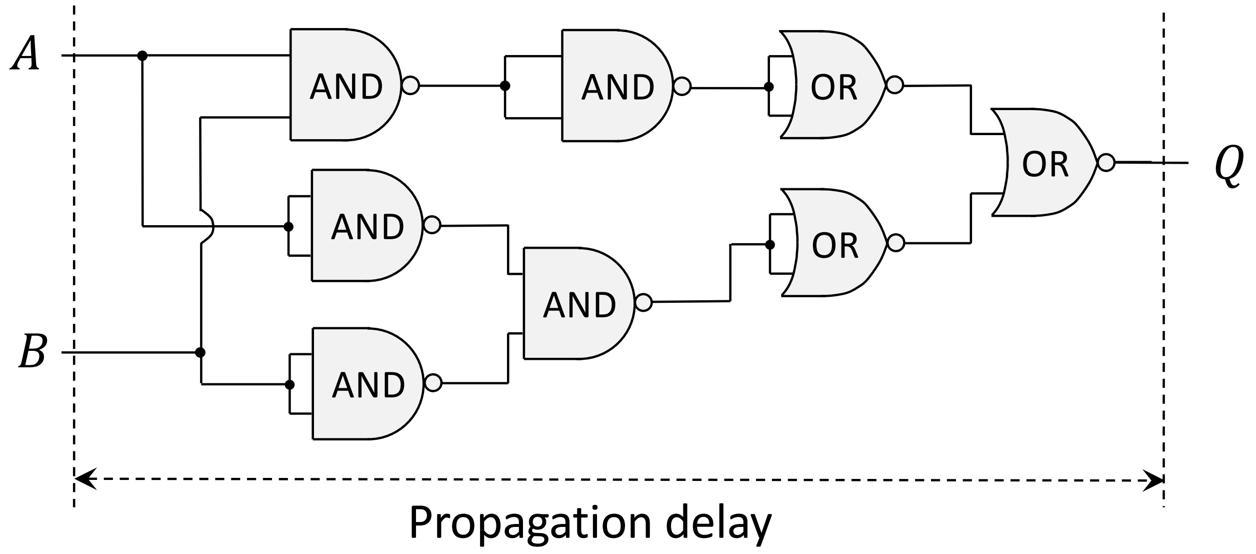
3) Output signal is updated after the change of input signals, subject to propagation delay

Propagation delay – time delay between the change of inputs and the update of output

Critical Path of the Integrated Circuit – the one taking the longest time (results in the “worst case” propagation delay)

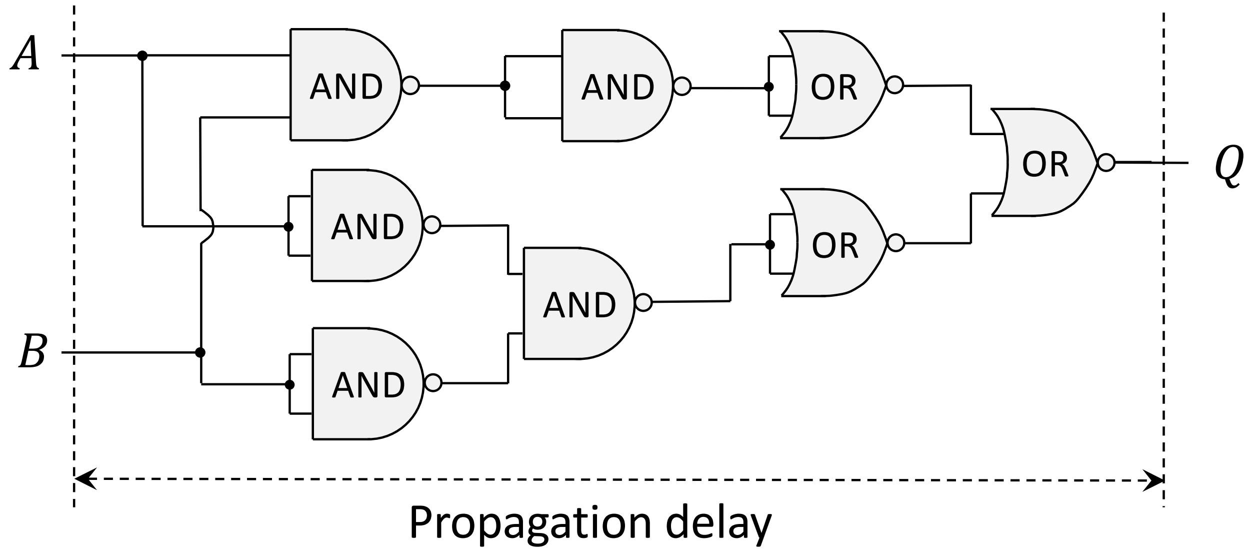
Computed in the direction starting from the output pin(s)

Recap: Characteristics of Combinational Logic Circuits



3) Output signal is updated after the change of input signals, subject to propagation delay

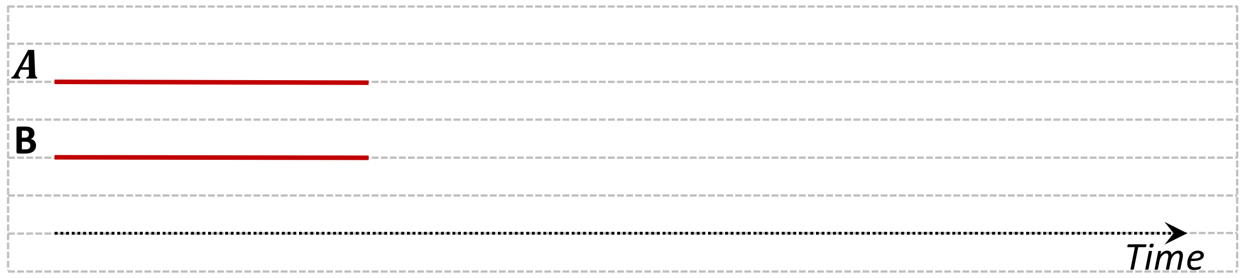
Recap: Characteristics of Combinational Logic Circuits



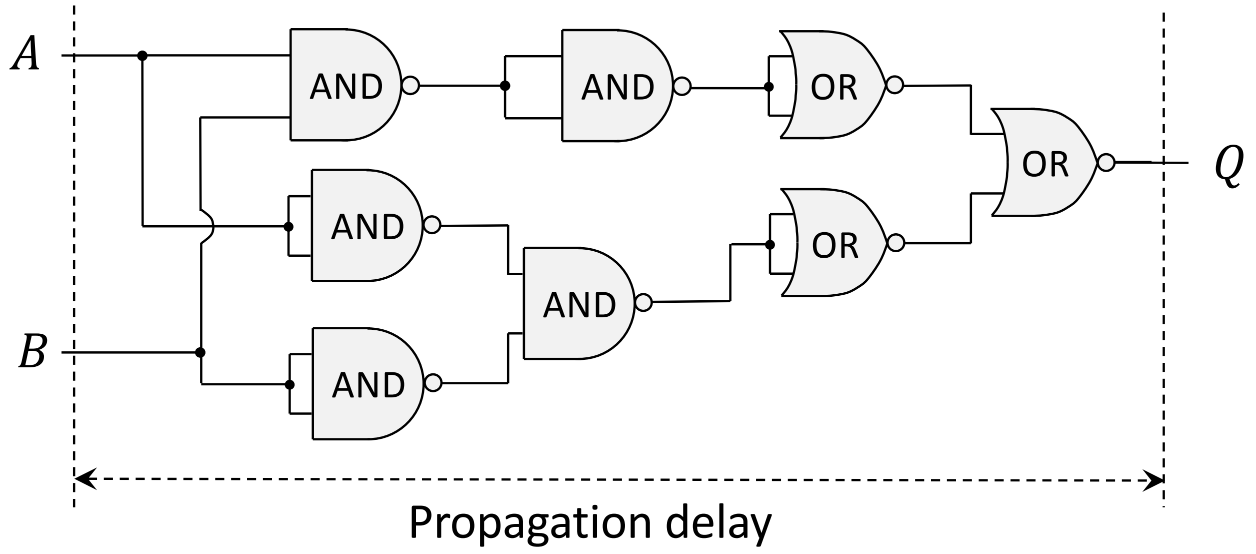
3) Output signal is updated after the change of input signals, subject to propagation delay

Sample timing diagram:

"0" – deasserted state;
"1" - asserted



Recap: Characteristics of Combinational Logic Circuits

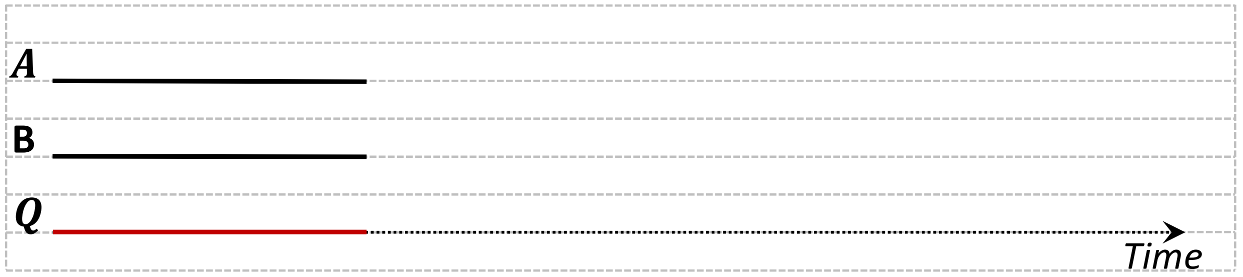


3) Output signal is updated after the change of input signals, subject to propagation delay

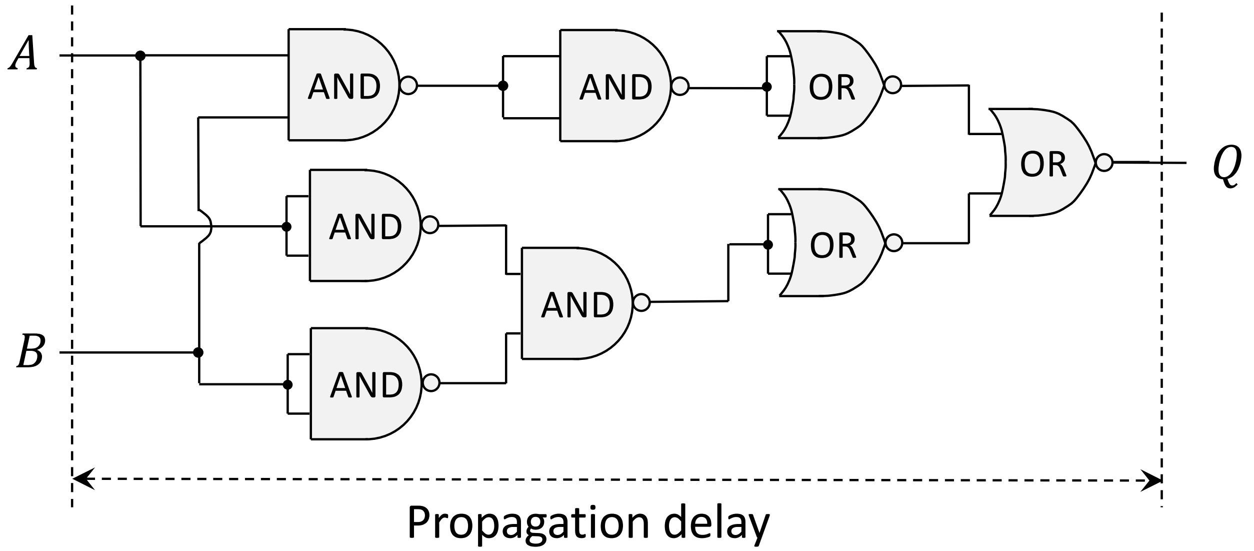
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

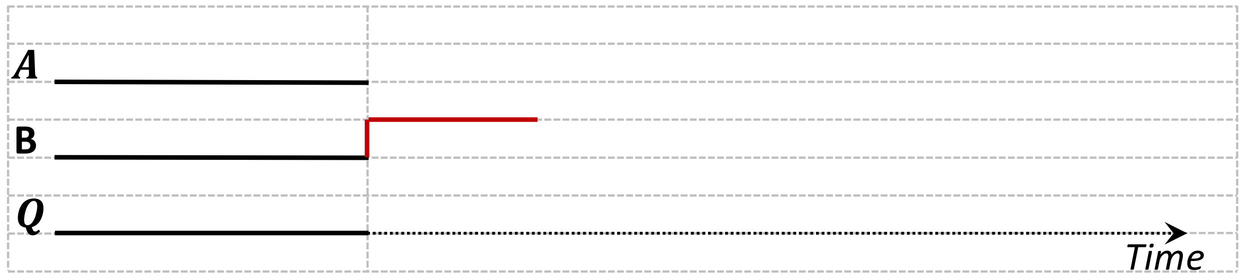


3) Output signal is updated after the change of input signals, subject to propagation delay

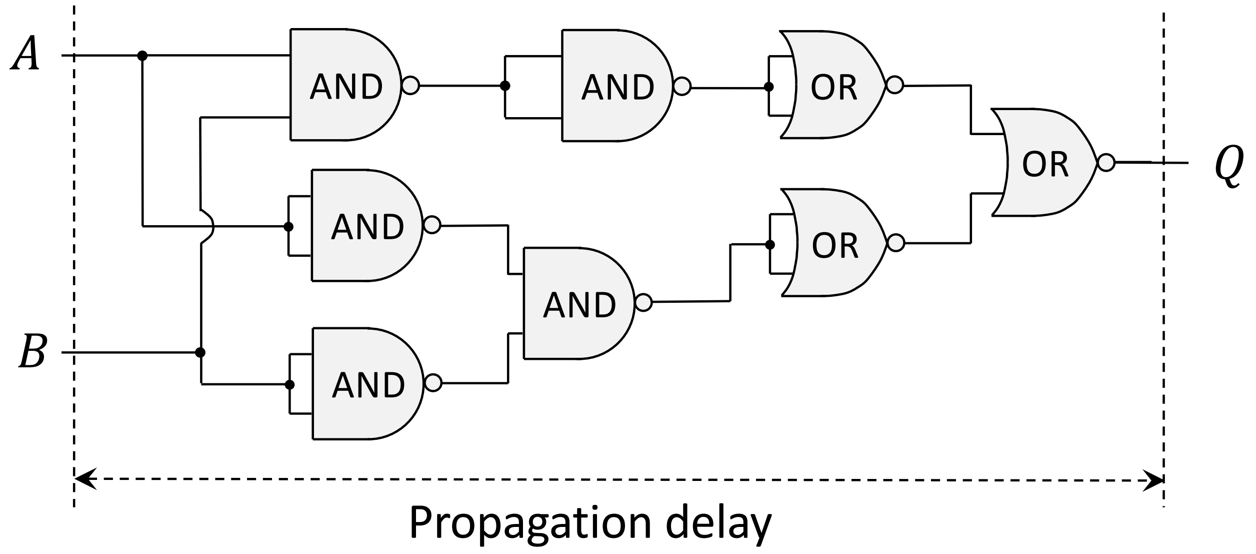
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

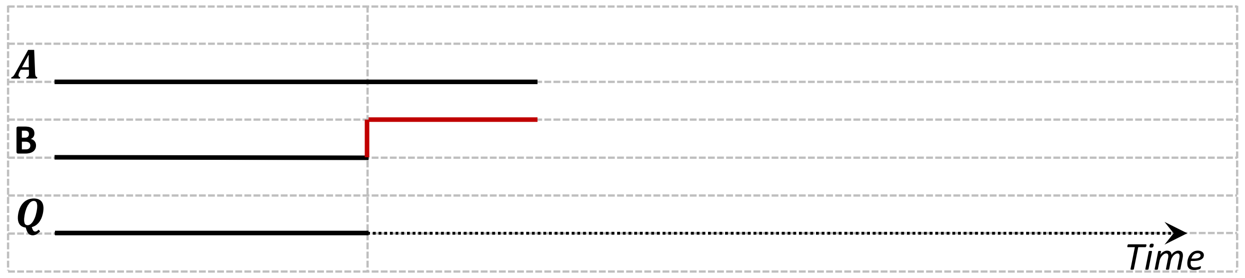


3) Output signal is updated after the change of input signals, subject to propagation delay

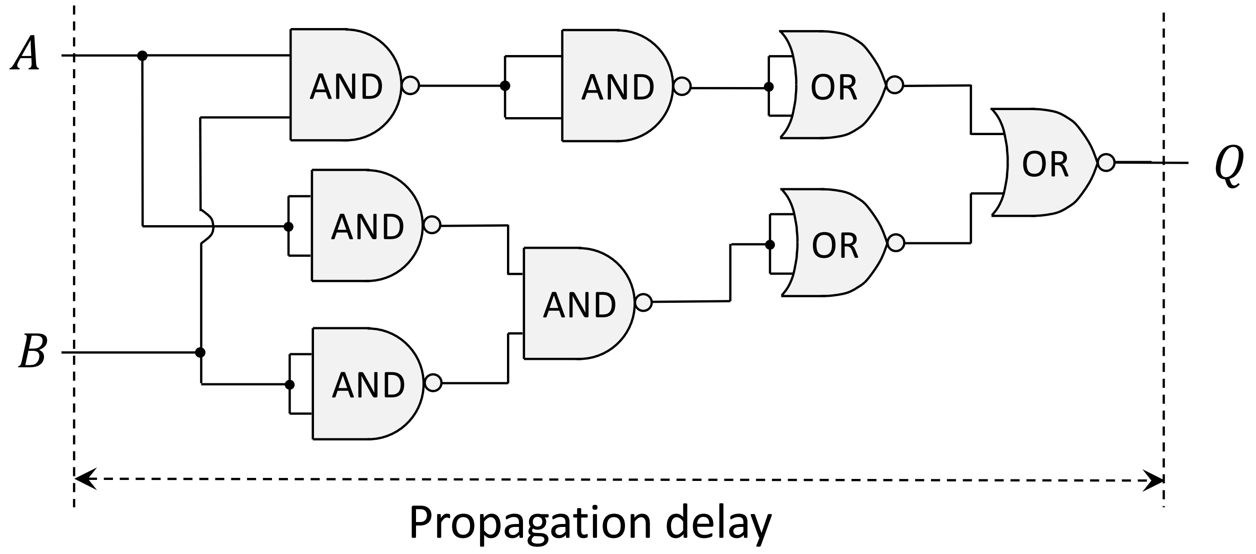
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

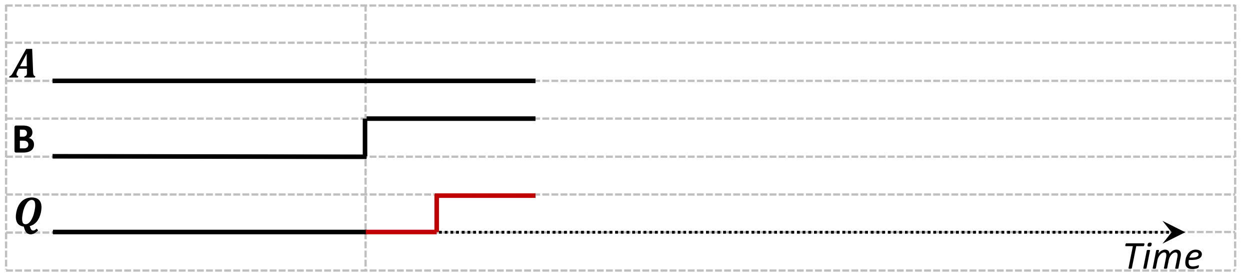


3) Output signal is updated after the change of input signals, subject to propagation delay

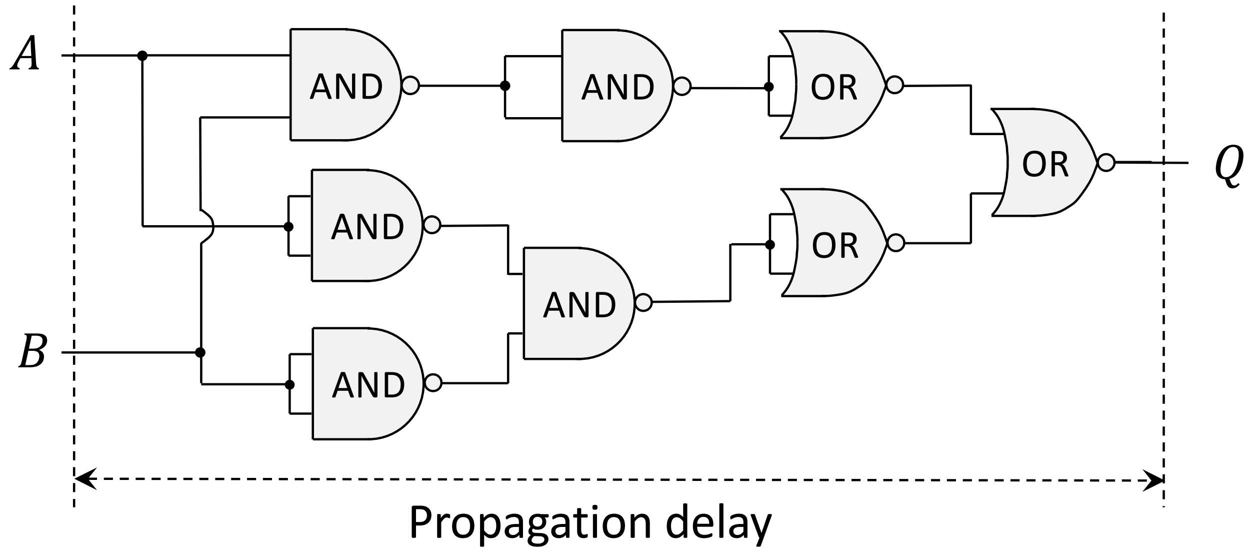
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

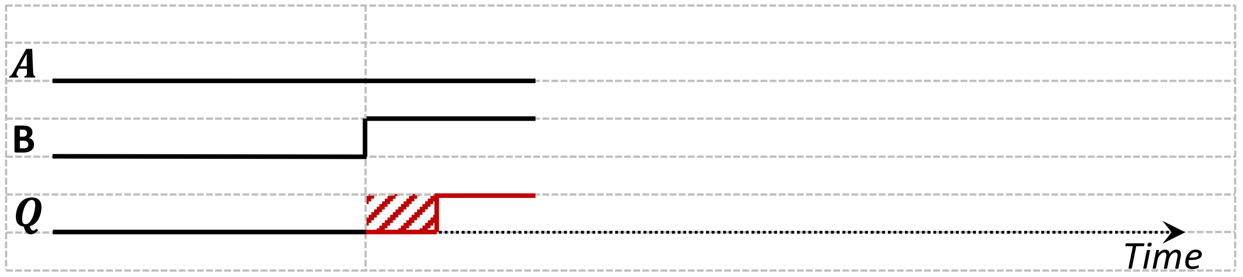


3) Output signal is updated after the change of input signals, subject to propagation delay

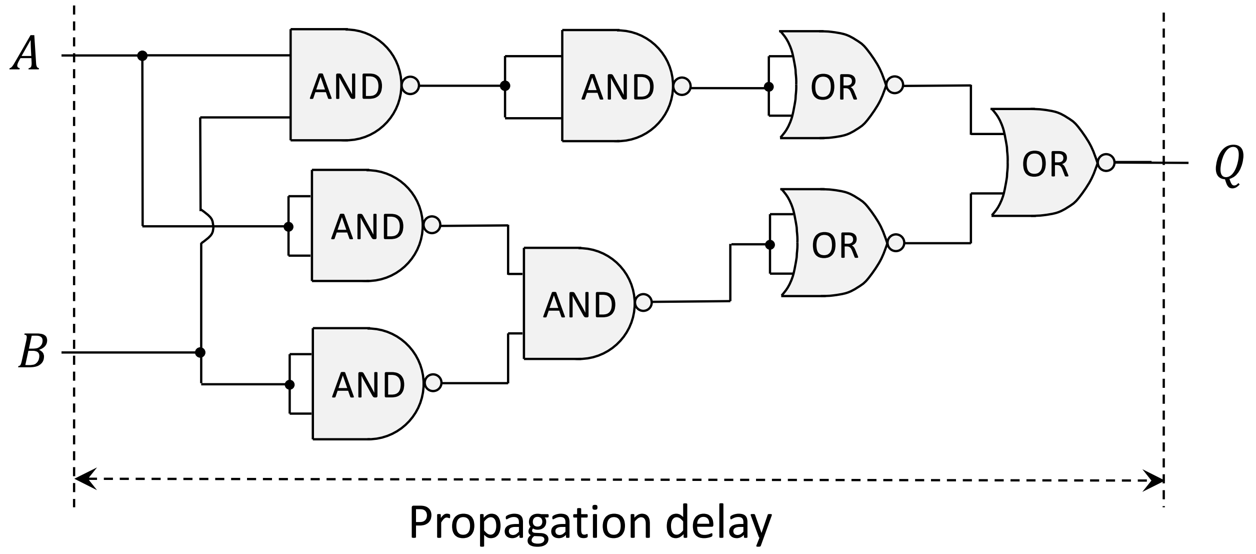
Sample timing diagram:

“0” – deasserted state;
“1” - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

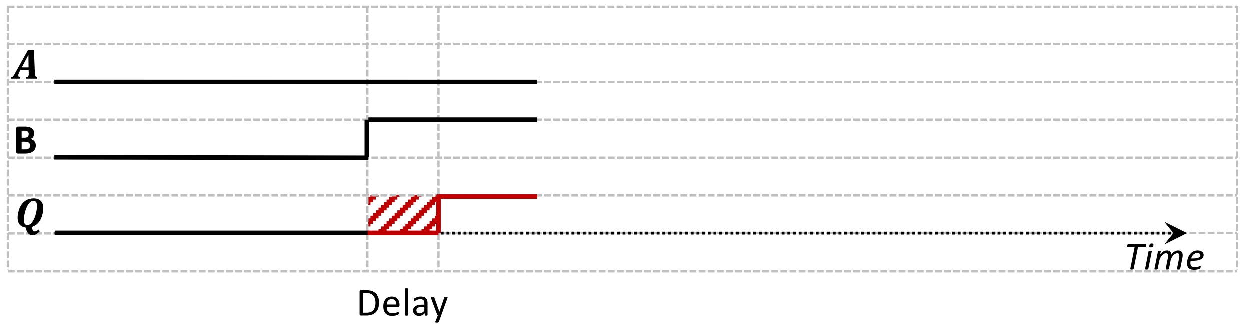


3) Output signal is updated after the change of input signals, subject to propagation delay

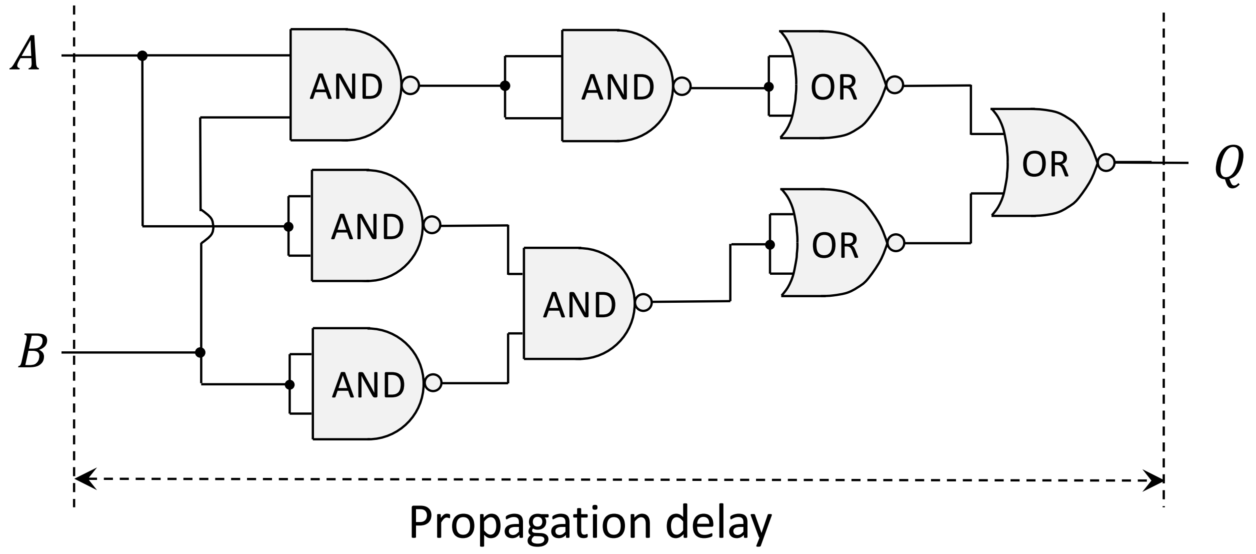
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above



Recap: Characteristics of Combinational Logic Circuits

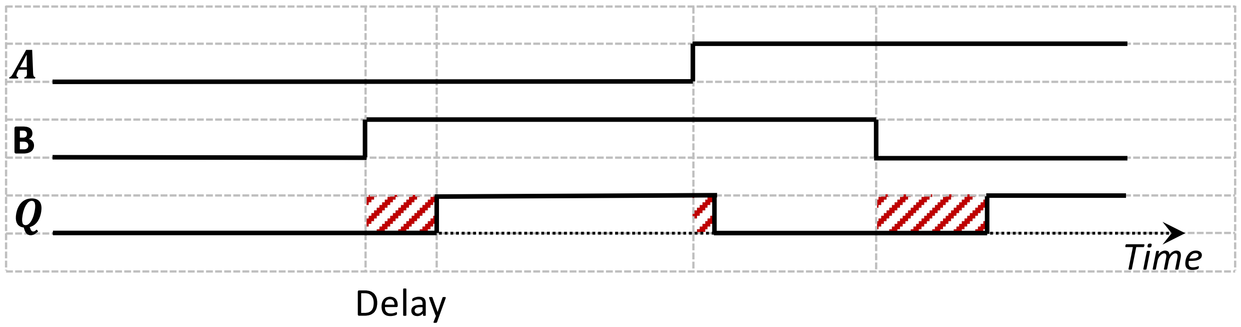


3) Output signal is updated after the change of input signals, subject to propagation delay

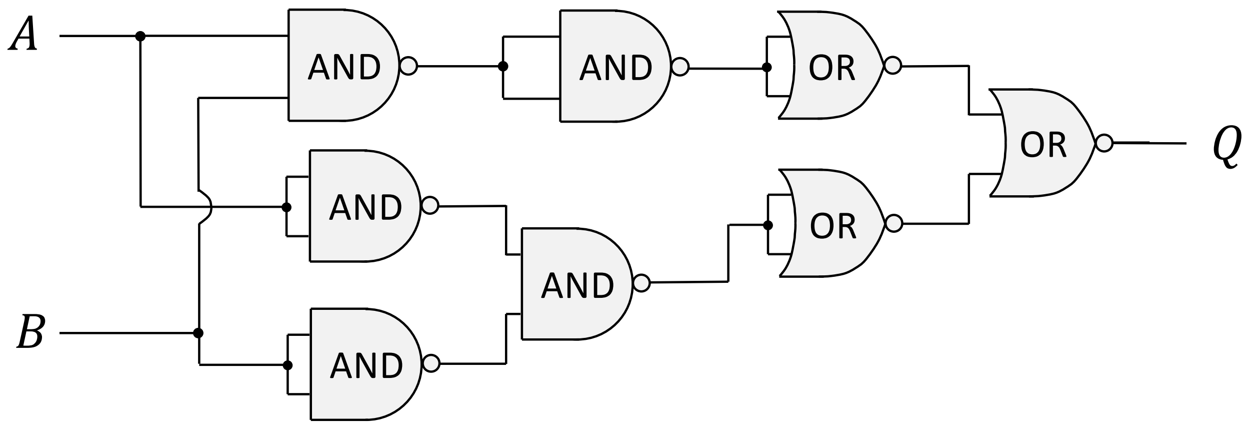
Sample timing diagram:

"0" – deasserted state;
"1" - asserted

Might not correspond to the circuit above

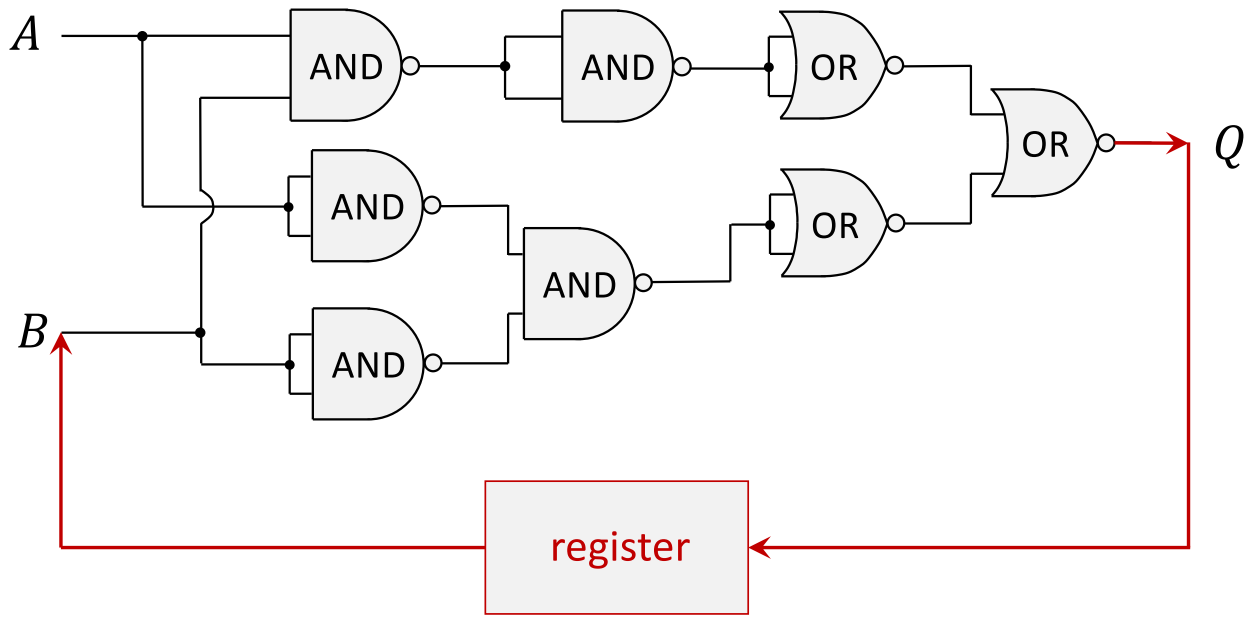


Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on "current" input signals ("combinational" = combine inputs)
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them
3) Output signal is updated after the change of input signals, subject to propagation delay
4) Use no memory units, such as registers (as opposed to "sequential" circuits – to be discussed)

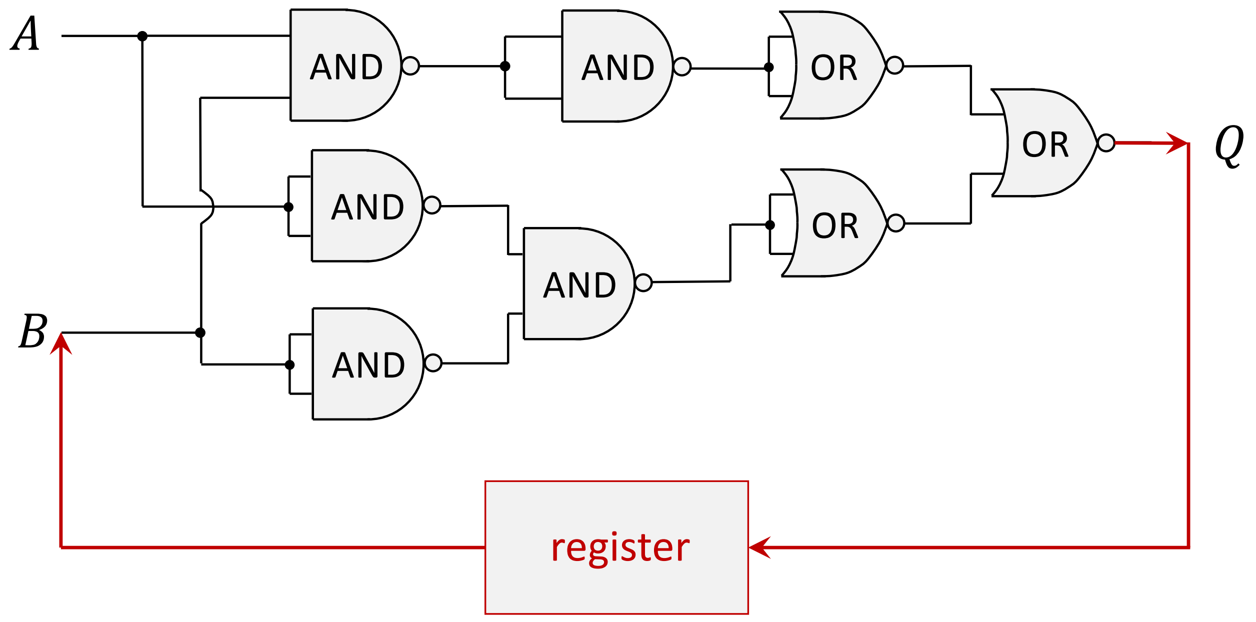
Recap: Characteristics of Combinational Logic Circuits



4) Use no memory units, such as registers (as opposed to “sequential” circuits – to be discussed)

Sequential circuits – an alternative to combinational

Recap: Characteristics of Combinational Logic Circuits

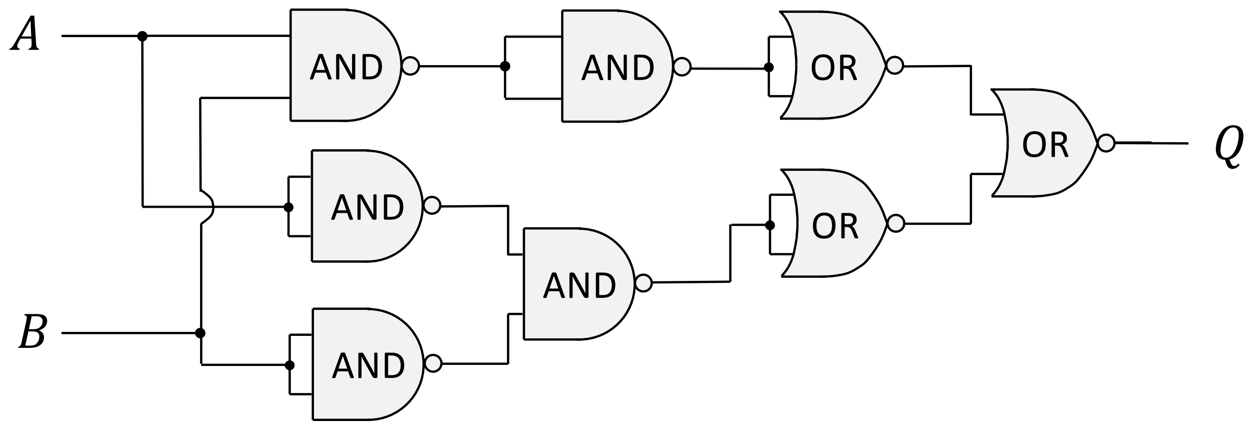


4) Use no memory units, such as registers (as opposed to “sequential” circuits – to be discussed)

Sequential circuits – an alternative to combinational

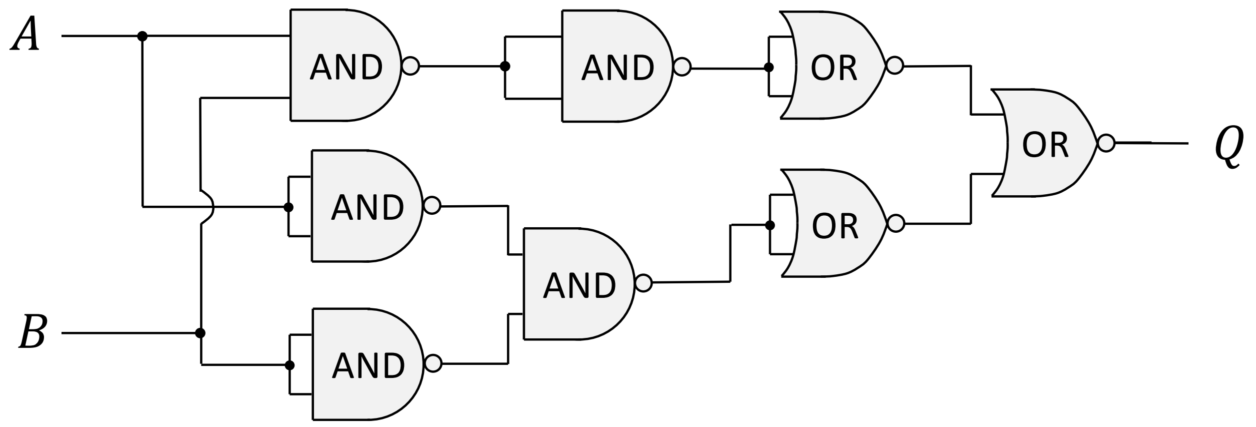
Register (from “to register” = “to remember”) – a memory element, to store electrical signal

Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on “current” input signals (“combinational” = combine inputs)
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them
3) Output signal is updated after the change of input signals, subject to propagation delay
4) Use no memory units, such as registers
5) Typically, no clock signal is present (exceptions might apply)

Recap: Characteristics of Combinational Logic Circuits



1) An output signal depends just on “current” input signals (“combinational” = combine inputs)
2) Comprised of logic gates, such as AND, OR, etc., and wires connecting them
3) Output signal is updated after the change of input signals, subject to propagation delay
4) Use no memory units, such as registers
5) Typically, no clock signal is present (exceptions might apply)

Key advantage: Fast (thus, used in such components, as processor ALU)

Key disadvantage: No support of inputs synchronization (will be discussed later)

Implementation of Arithmetic Operations in Computers

Key principles:

1) Convert decimal number inputs into binary representation

Implementation of Arithmetic Operations in Computers

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using a combinational logic circuit

Implementation of Arithmetic Operations in Computers

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

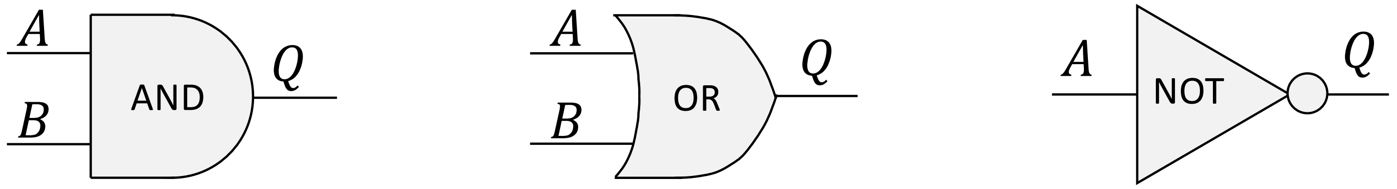
Boolean Function Implementation: Key Results from Discrete Math

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT



Boolean Function Implementation: Key Results from Discrete Math

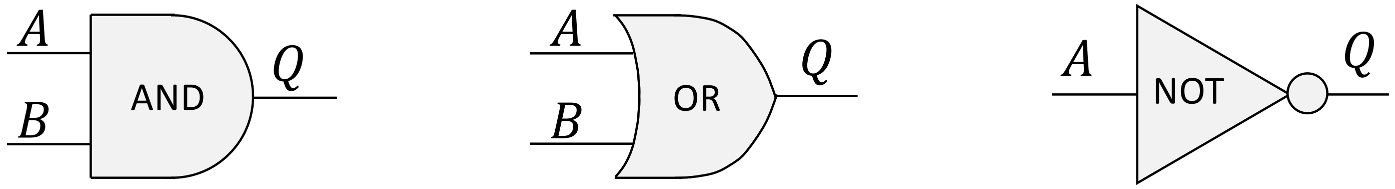
Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT These is the basis:

If some of these logic gates is removed, some functions are not implementable



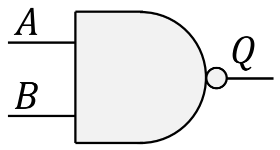
Boolean Function Implementation: Key Results from Discrete Math

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. "+", "-", "*", "/") by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate (no other logic gate is needed)



NAND

Boolean Function Implementation: Key Results from Discrete Math

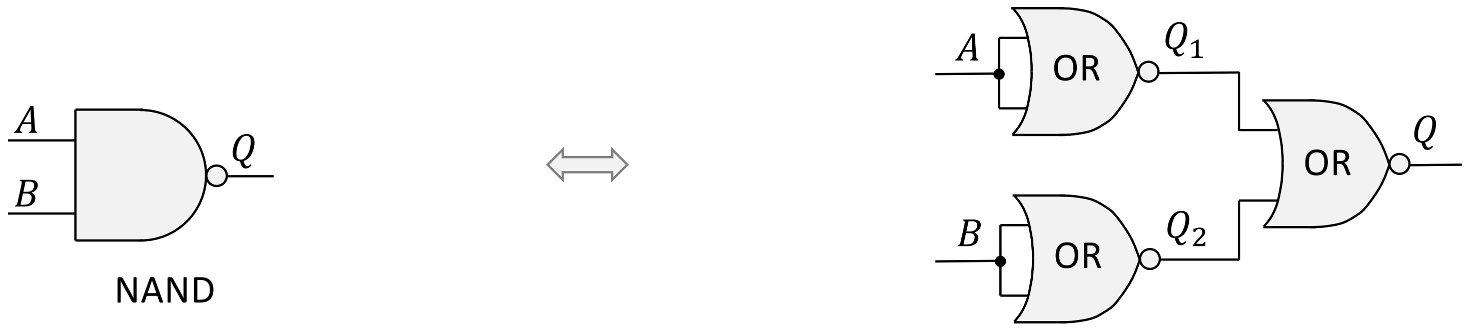
Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate (no other logic gate is needed)

Example:



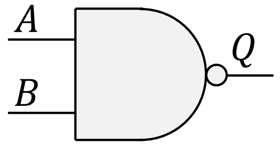
Boolean Function Implementation: Key Results from Discrete Math

Key principles:

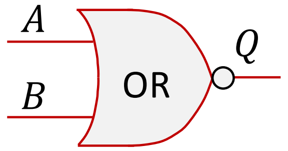
1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate
3) NOR – another logic gate



NAND



NOR

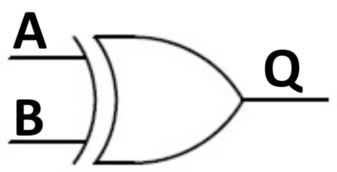
Boolean Function Implementation: Key Results from Discrete Math

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate
3) NOR – another logic gate
4) XOR and boolean constant “1”



+ “1”

There is a notion of hardwired “1” or “0”

Boolean Function Implementation: Key Results from Discrete Math

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate
3) NOR – another logic gate
4) XOR and boolean constant “1”

These results are formally proved in Discrete Math in forms of “Logic Completeness Theorems”

Boolean Function Implementation: Key Results from Discrete Math

Key principles:

1) Convert decimal number inputs into binary representation;
2) Implement an arithmetic operation (e.g. “+”, “-”, “*”, “/”) by using combinational logic circuits;
3) Convert the computation result back into decimal format

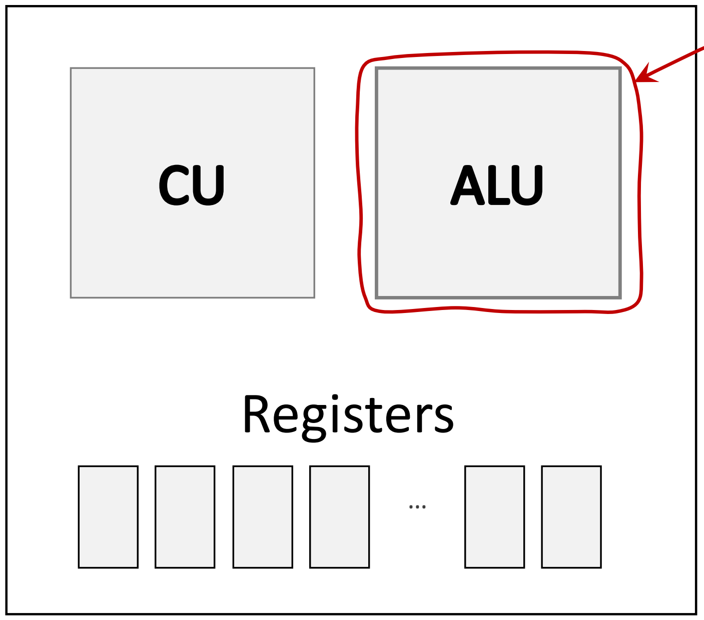
Any Boolean function can be implemented by using only one out of these sets of logic gates:

1) AND, OR, NOT
2) NAND – universal logic gate
3) NOR – another logic gate
4) XOR and boolean constant “1”

These results are formally proved in Discrete Math in forms of “Logic Completeness Theorems”

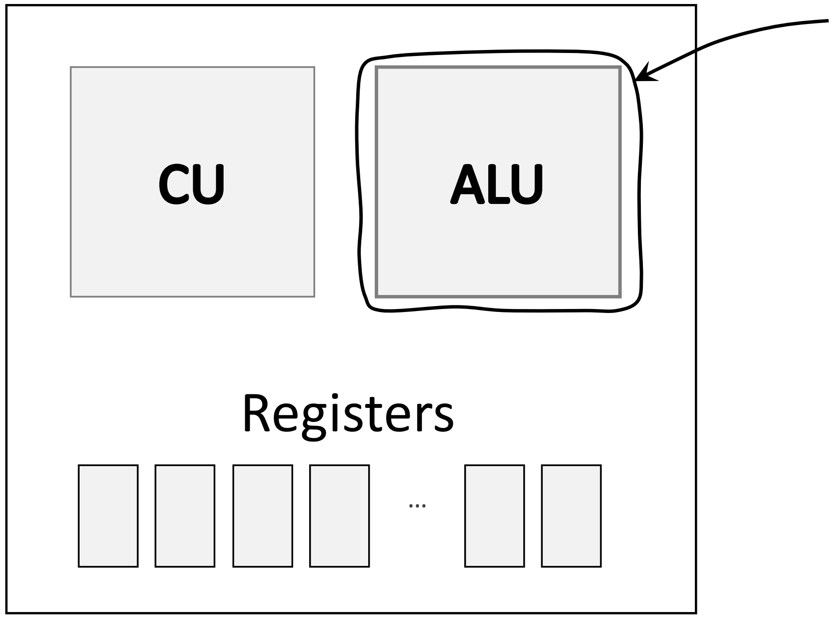
Zhegalkin polynomial – another interesting result

Types of Logic Circuits for CPU Components



Combinational logic circuit for arithmetical and logic operations

Types of Logic Circuits for CPU Components

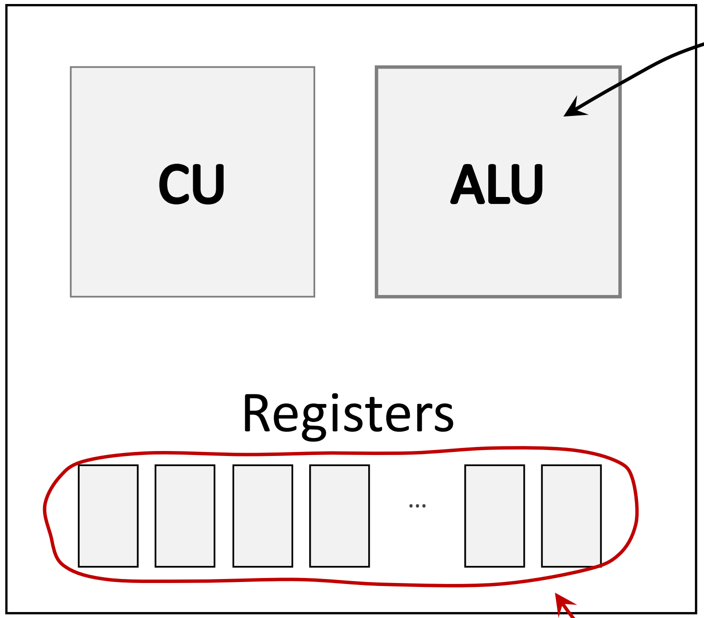


Combinational logic circuit for arithmetical and logic operations

Fast - advantage

No synchronization support - disadvantage

Types of Logic Circuits for CPU Components



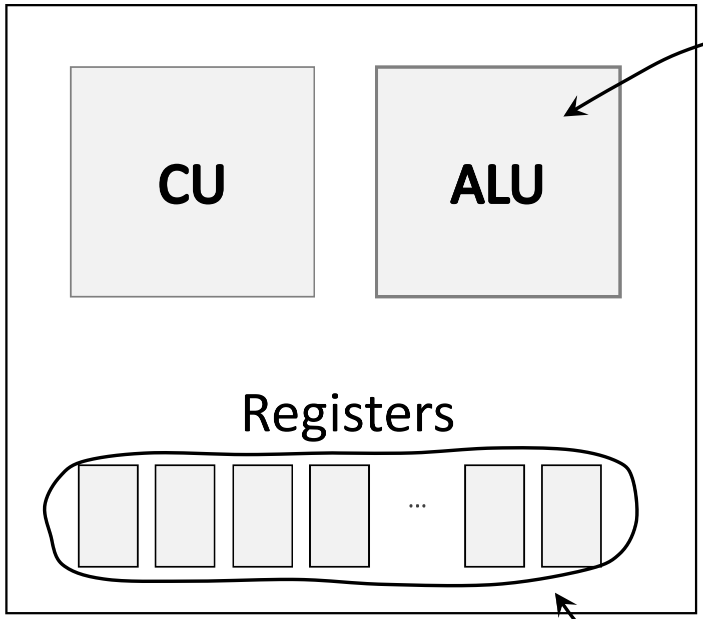
Combinational logic circuit for arithmetical and logic operations

Fast - advantage

No synchronization support - disadvantage

Sequential logic circuits (clocked)

Types of Logic Circuits for CPU Components



Combinational logic circuit for arithmetical and logic operations

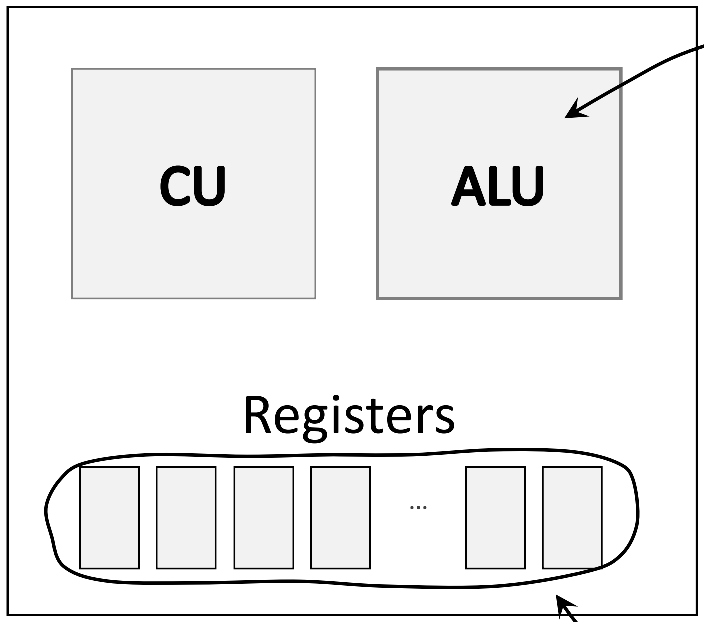
Fast - advantage

No synchronization support - disadvantage

Sequential logic circuits (clocked)

1) Capture inputs (might arrive not simultaneously);

Types of Logic Circuits for CPU Components



Combinational logic circuit for arithmetical and logic operations

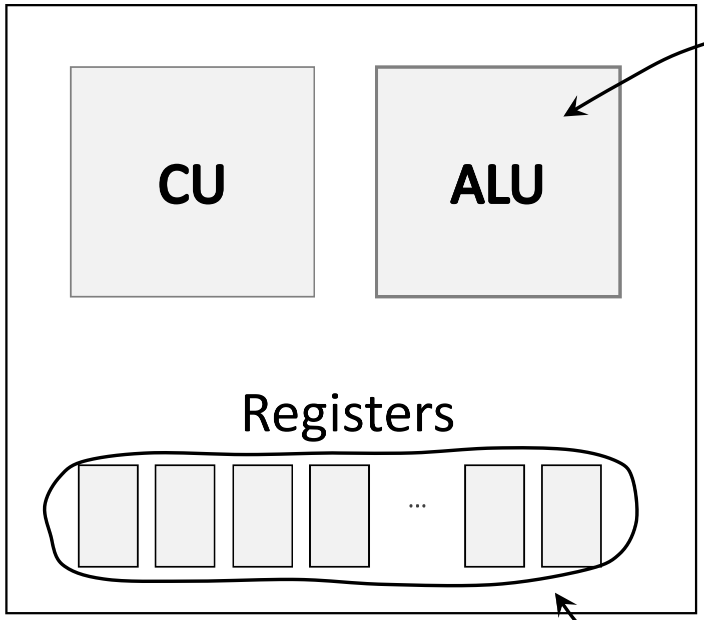
Fast - advantage

No synchronization support - disadvantage

Sequential logic circuits (clocked)

1) Capture inputs (might arrive not simultaneously);
2) Store inputs until a certain time instant;

Types of Logic Circuits for CPU Components



Combinational logic circuit for arithmetical and logic operations

Fast - advantage

No synchronization support - disadvantage

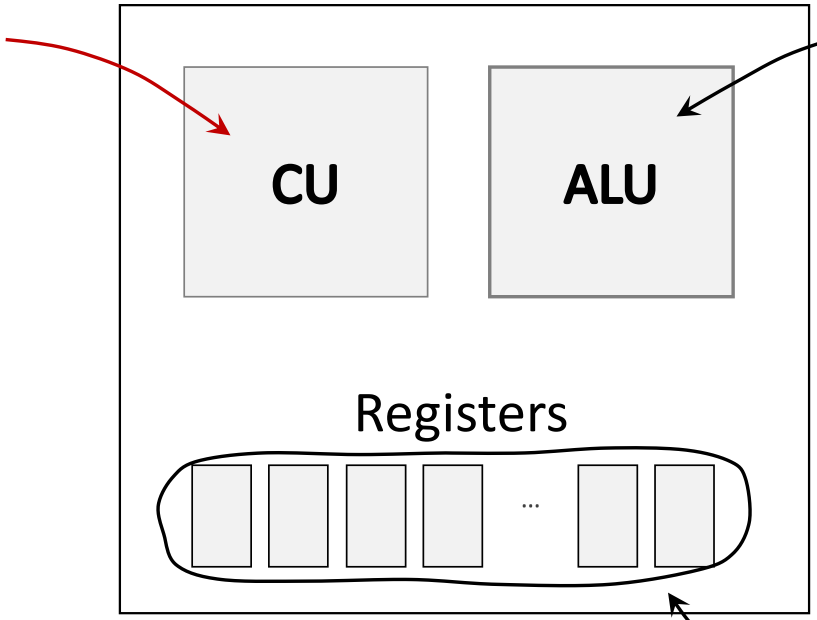
Sequential logic circuits (clocked)

1) Capture inputs (might arrive not simultaneously);
2) Store inputs until a certain time instant;
3) Output all inputs simultaneously to ALU

Types of Logic Circuits for CPU Components

Not a pure combinational circuit

(for example, typically has IR – instruction register)



Combinational logic circuit for arithmetical and logic operations

Fast - advantage

No synchronization support - disadvantage

Sequential logic circuits (clocked)

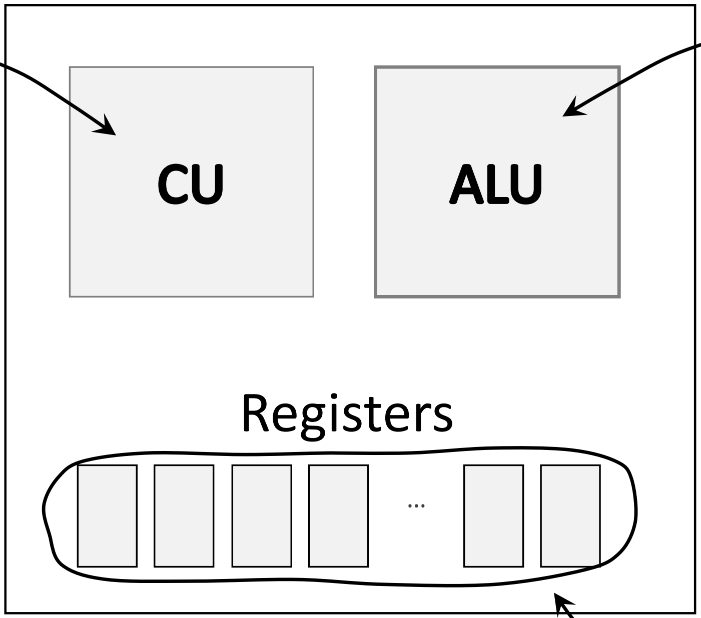
1) Capture inputs (might arrive not simultaneously);
2) Store inputs until a certain time instant;
3) Output all inputs simultaneously to ALU

Types of Logic Circuits for CPU Components

Not a pure combinational circuit

(for example, typically has IR – instruction register)

Recall Instruction Fetch (IF) stage of a pipelined execution for instructions



Combinational logic circuit for arithmetical and logic operations

Fast - advantage

No synchronization support - disadvantage

Sequential logic circuits (clocked)

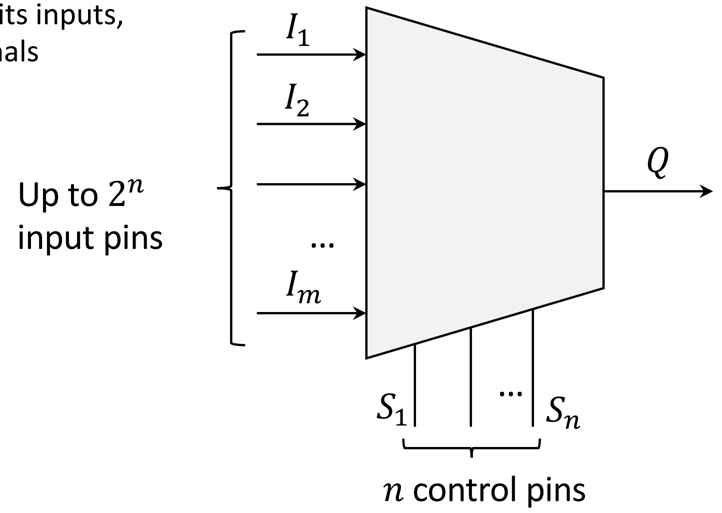
1) Capture inputs (might arrive not simultaneously);
2) Store inputs until a certain time instant;
3) Output all inputs simultaneously to ALU

Key Hardware Building Blocks for Combinational Circuits

1) Multiplexer
2) Demultiplexer
3) Decoder
4) Encoder

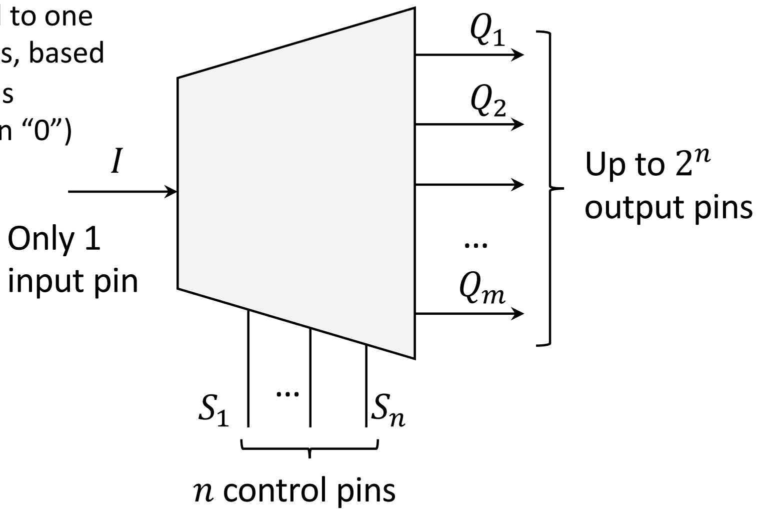
Multiplexor

Sets output to one of its inputs, based on selector signals



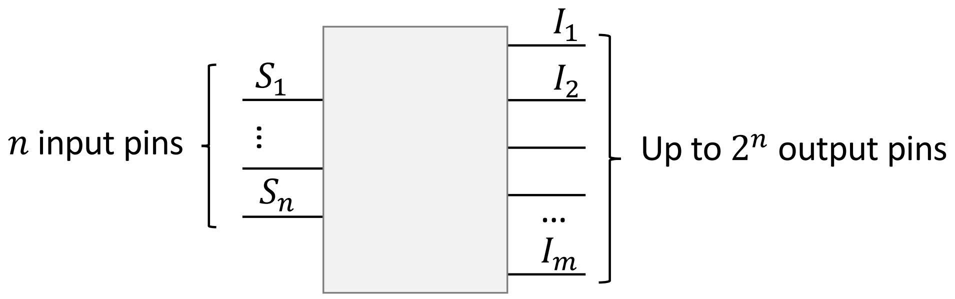
Demultiplexor

Forwards input signal to one of its multiple outputs, based on the selector signals (other outputs remain "0")



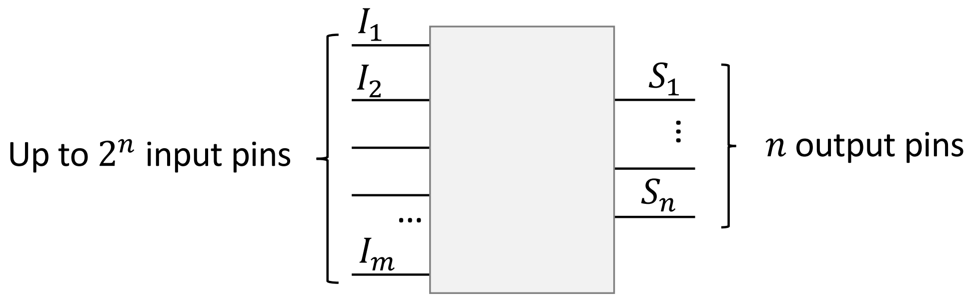
Decoder

Sets to "1" exactly one output pin, which corresponds to the signals of input pins;
All other pins are set to "0"



Encoder

The opposite function of a decoder;
Only one output pin is set to "1", while all others – "0"



Key Hardware Building Blocks for Combinational Circuits

1) Multiplexer
2) Demultiplexer
3) Decoder
4) Encoder

These circuits are generously used in the implementation of real hardware, such as the main system memory, ALU, etc.

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Circuit for “+”

Circuit for “-”

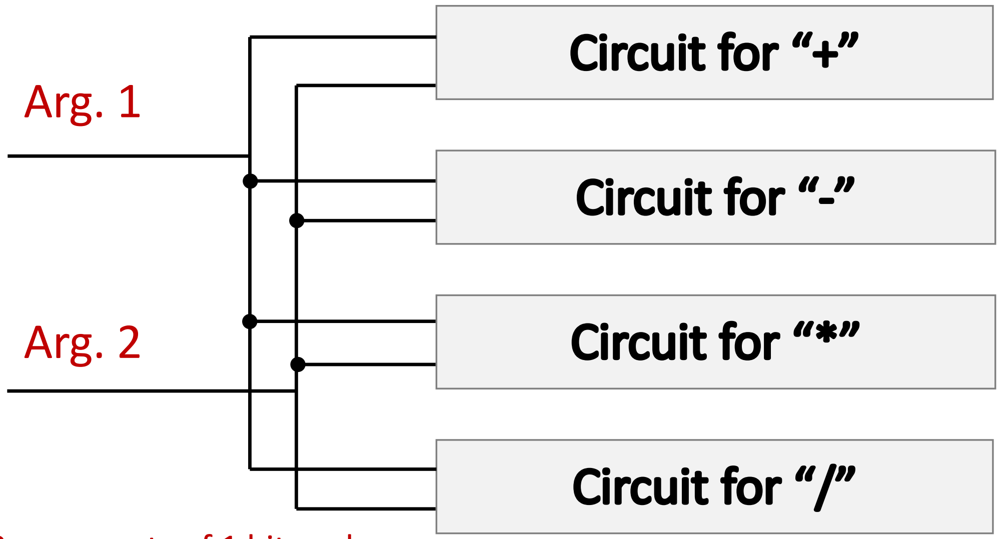
Circuit for “*”

Circuit for “/”

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

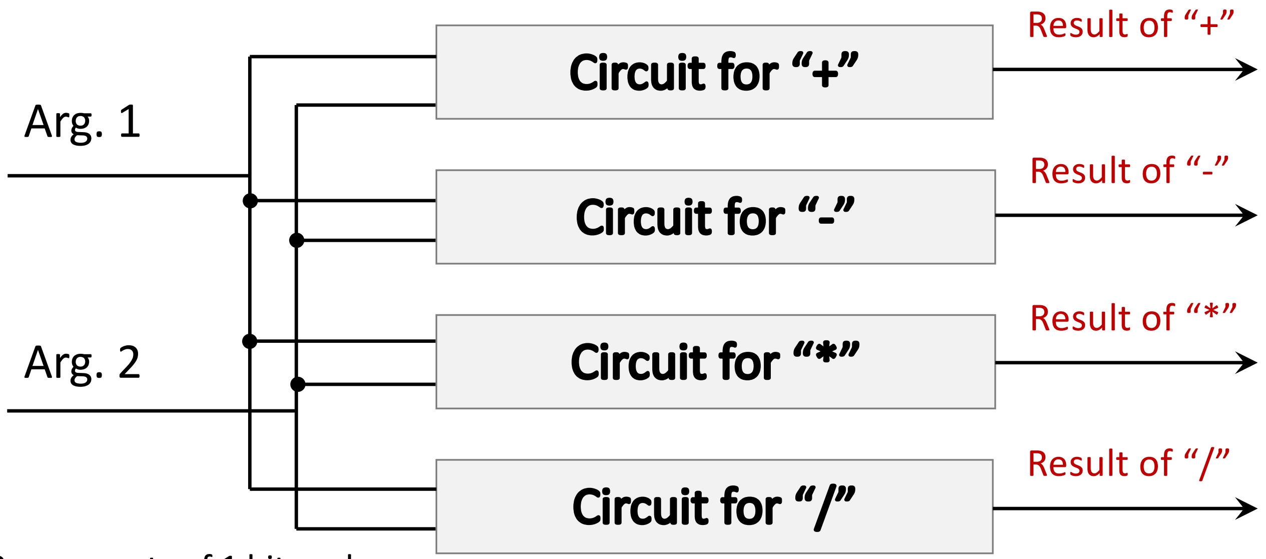


Only 2 arguments of 1 bit each
(for explanation simplicity)

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”



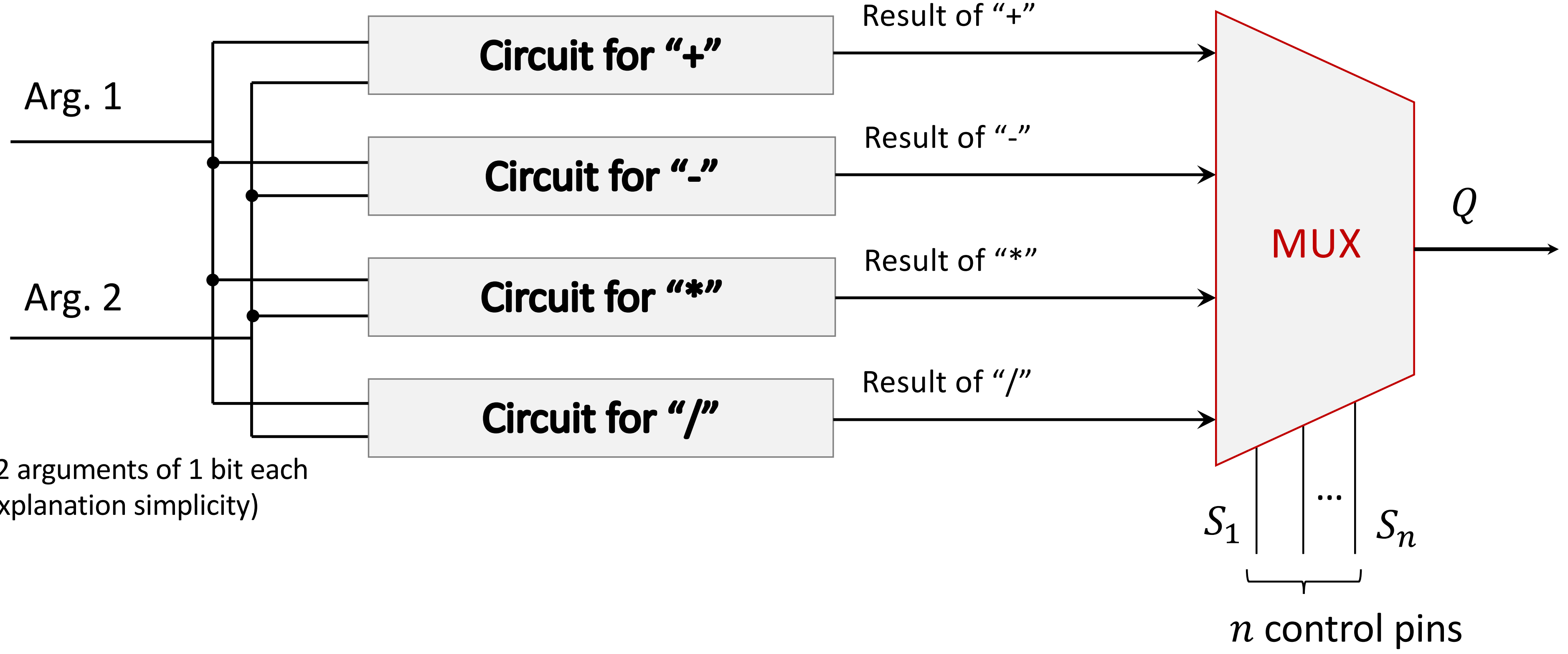
Only 2 arguments of 1 bit each
(for explanation simplicity)

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Multiplexer:
Sets output to one of its inputs, based on selector signals



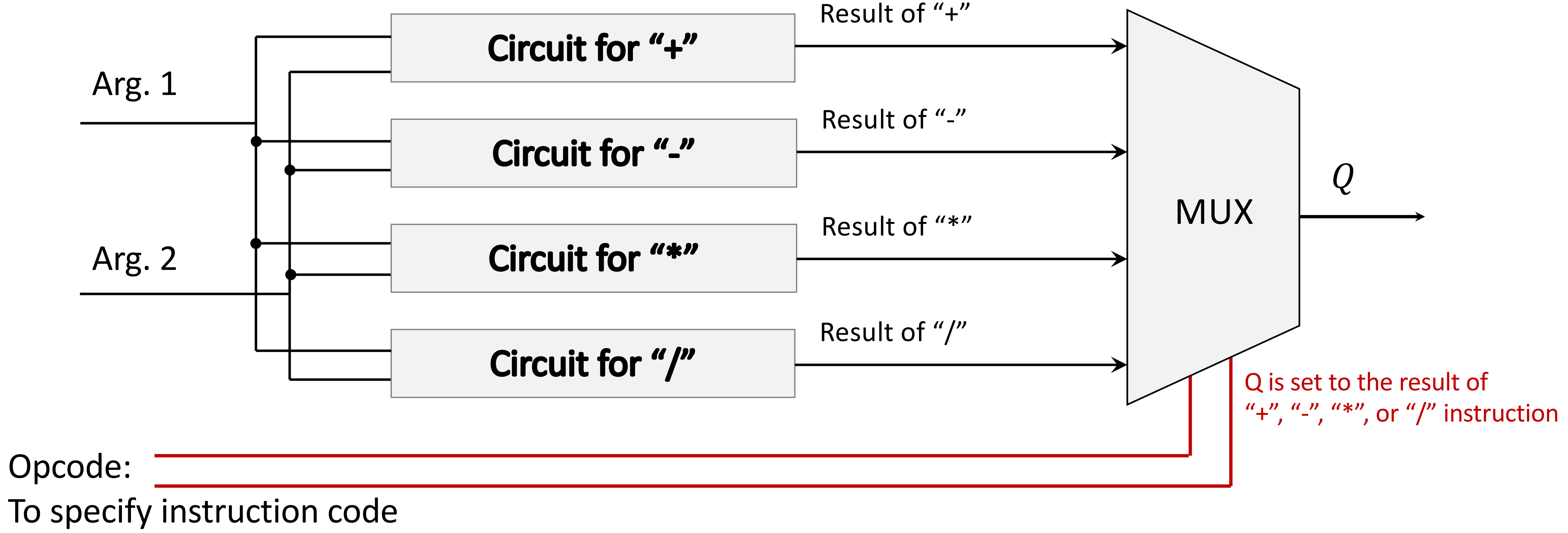
Only 2 arguments of 1 bit each
(for explanation simplicity)

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Multiplexer:
Sets output to one of its inputs, based on selector signals

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

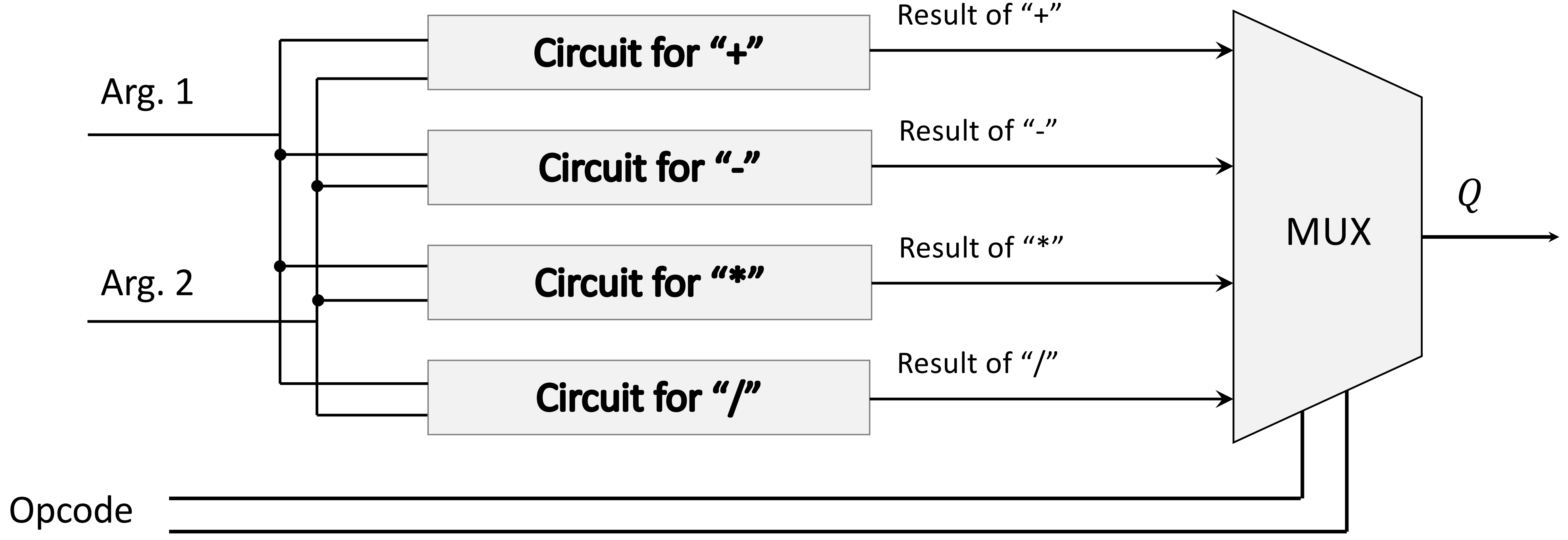


Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Multiplexer:
Sets output to one of its inputs, based on selector signals

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”



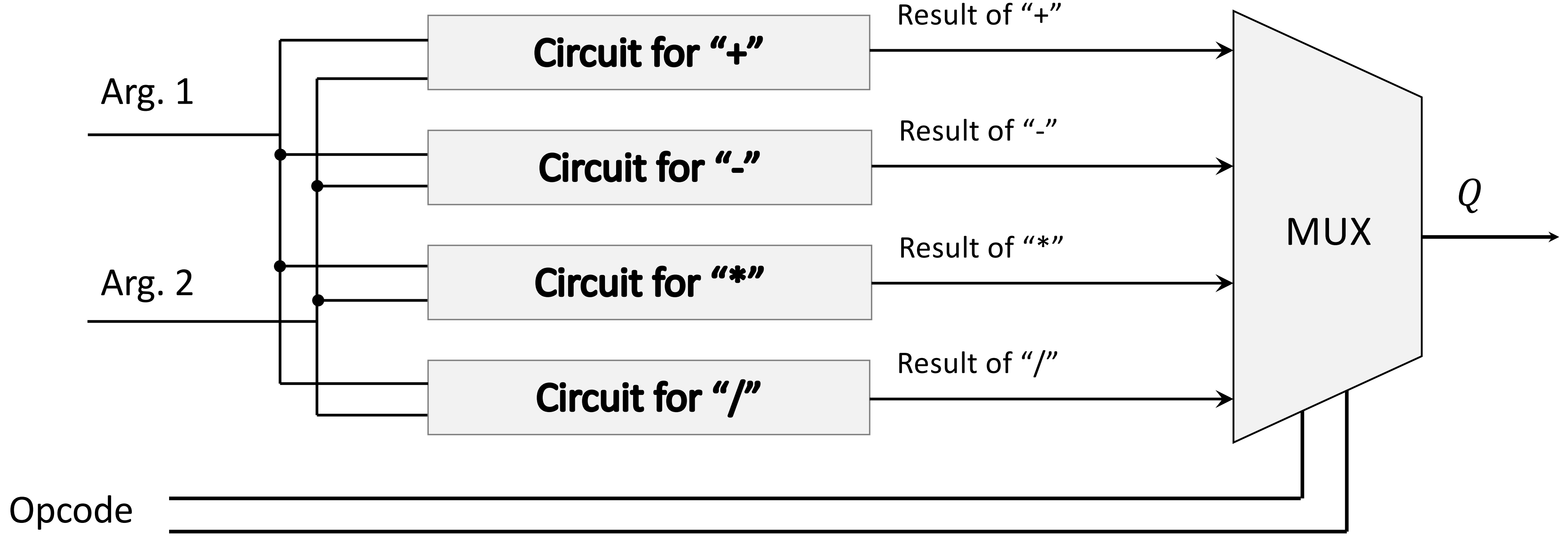
There is a correlation between the execution time of the longest instruction and CPU clock length

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Multiplexer:
Sets output to one of its inputs, based on selector signals

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”



There is a correlation between the execution time of the longest instruction and CPU clock length

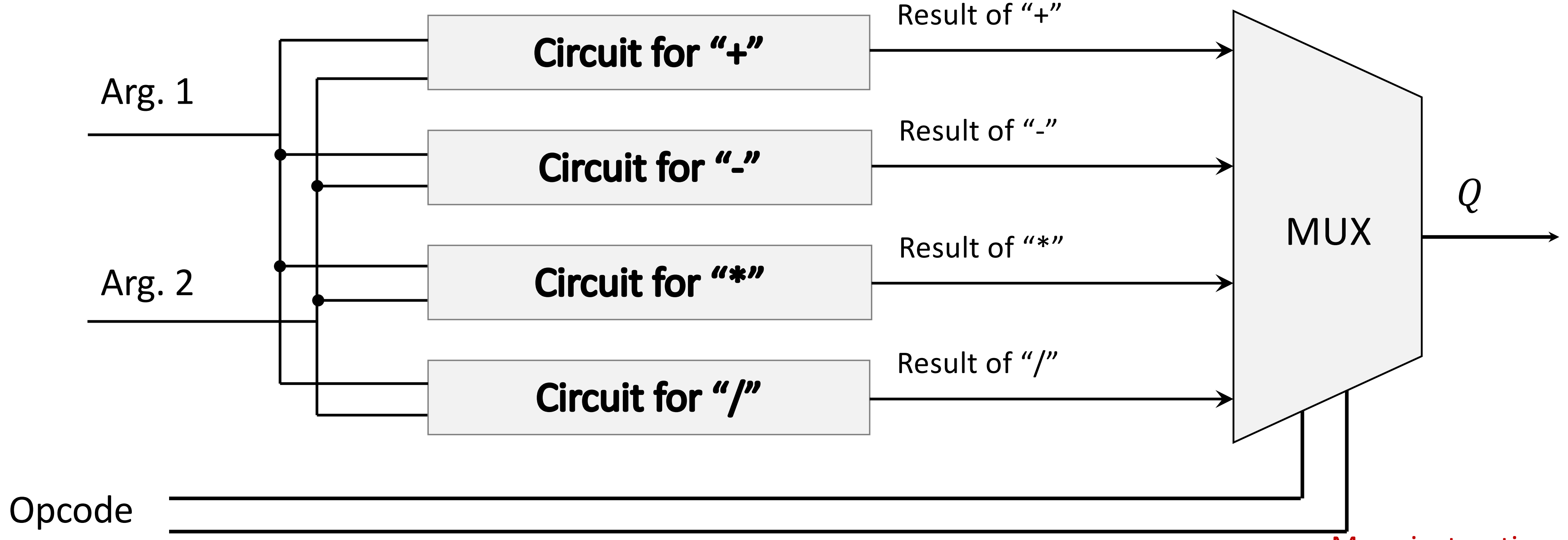
For this example, the longest instruction must complete within 1 CPU clock cycle

Sample Use Case for Multiplexer: ALU Implementation (An Oversimplified Concept)

ALU (Arithmetic Logic Unit) – the part of CPU to implement arithmetic and logic instructions

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Multiplexer:
Sets output to one of its inputs, based on selector signals



There is a correlation between the execution time of the longest instruction and CPU clock length

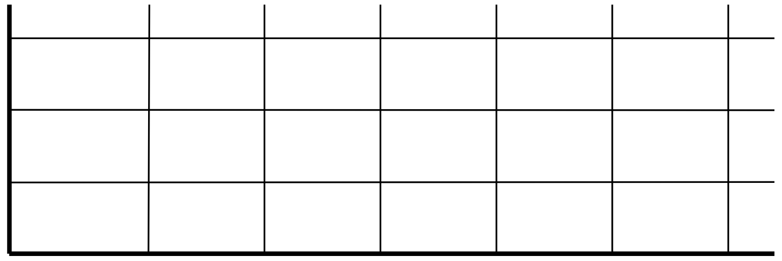
For this example, the longest instruction must complete within 1 CPU clock cycle

More instructions supported is not necessary better (recall RISC vs. CISC designs)

Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

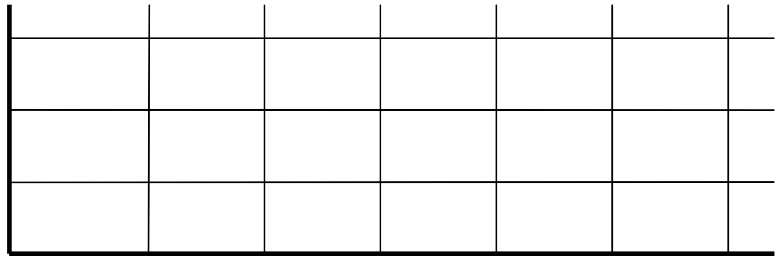
a set of memory cells
(each cell has its address)

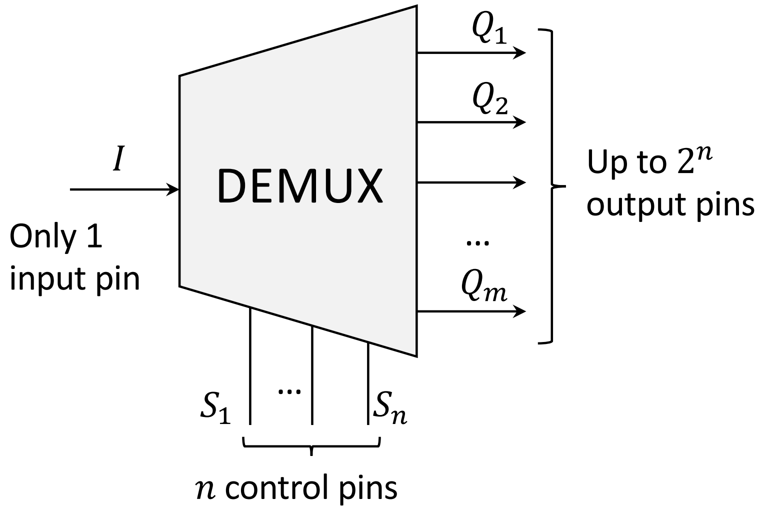


Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

a set of memory cells
(each cell has its address)

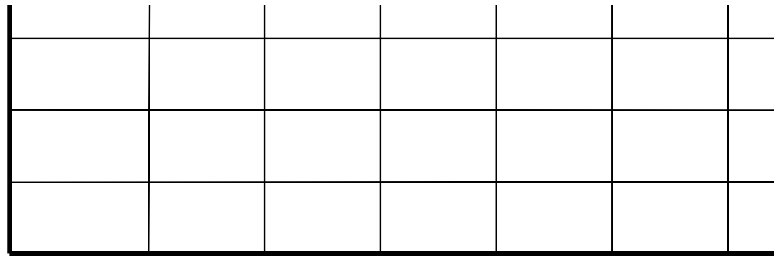


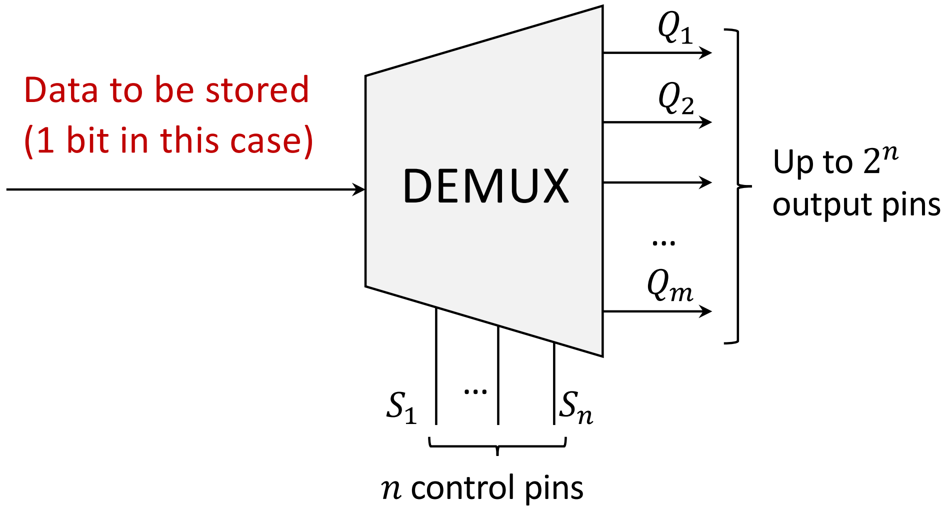


Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

a set of memory cells
(each cell has its address)

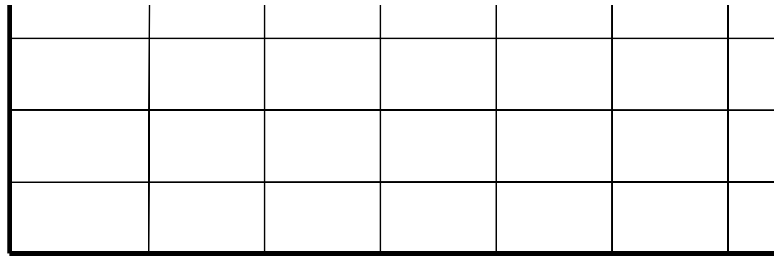


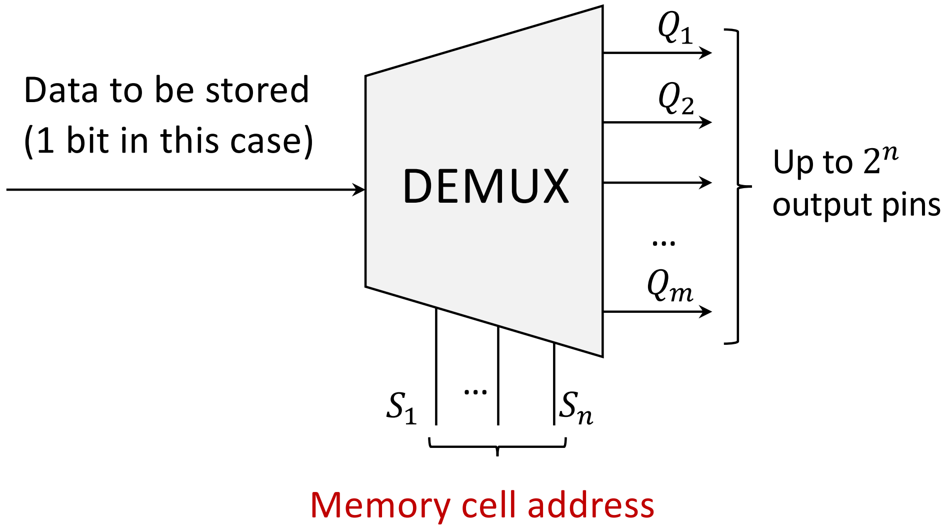


Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

a set of memory cells
(each cell has its address)

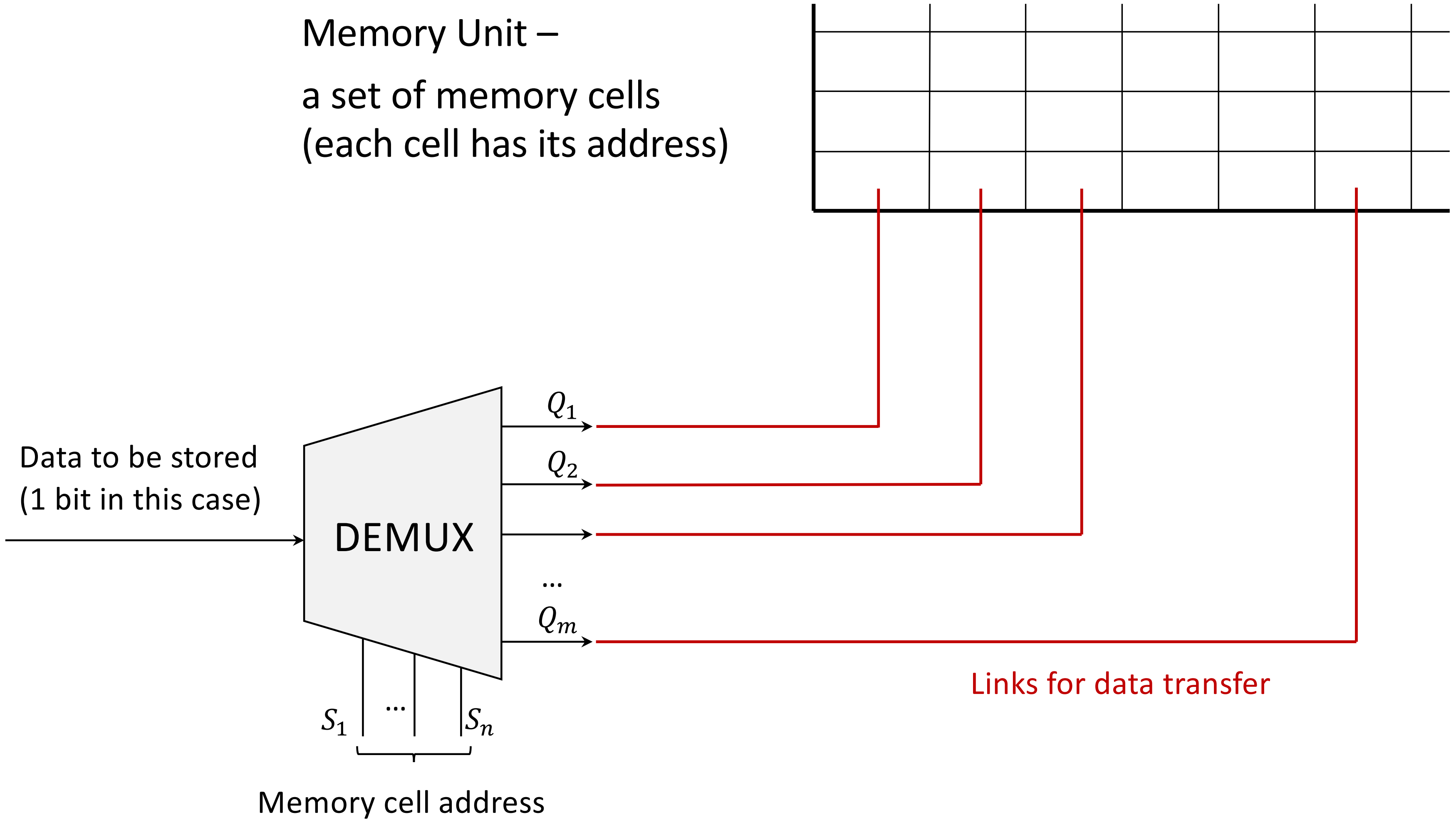




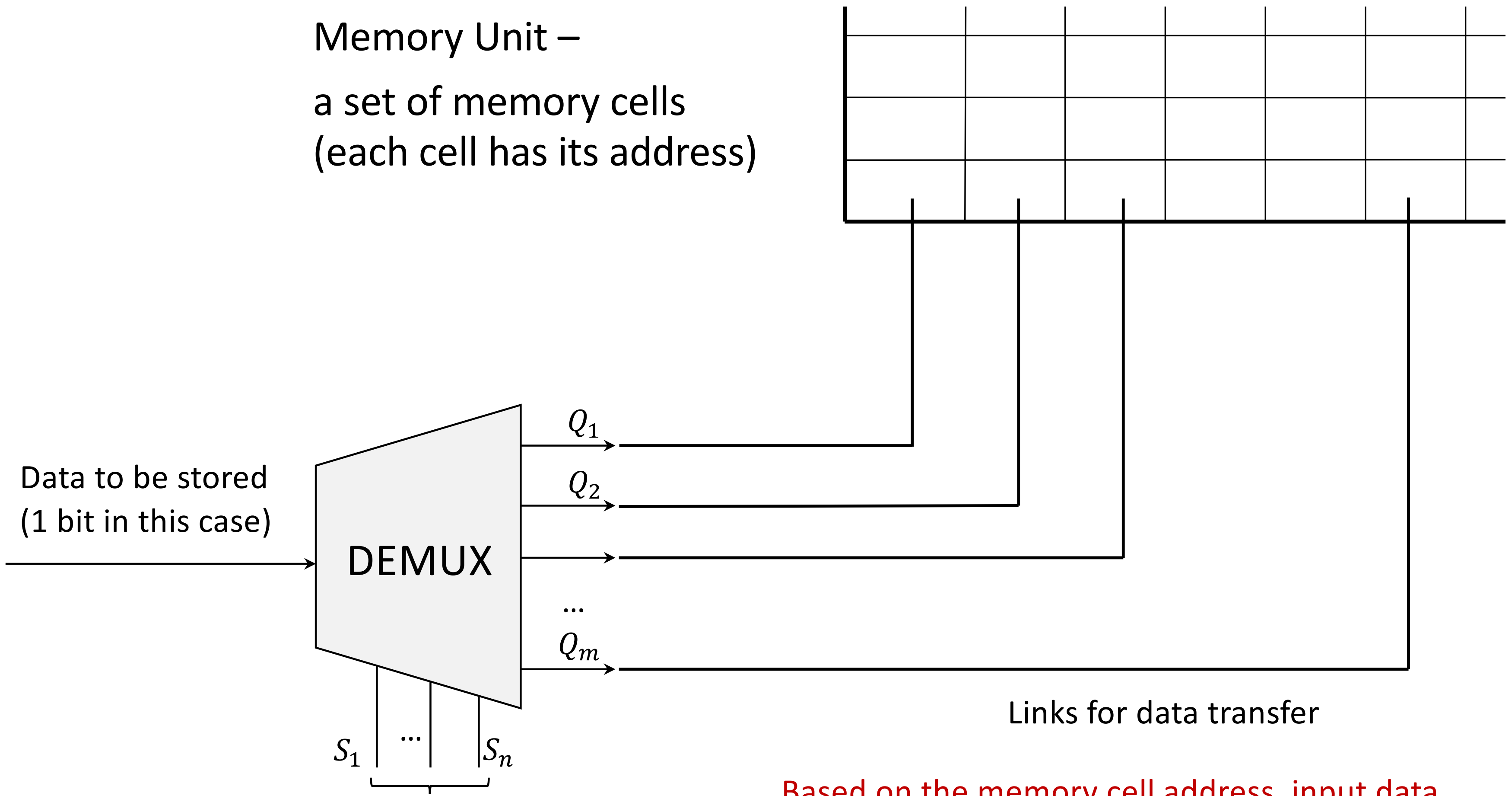
Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

a set of memory cells
(each cell has its address)



Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)



Memory cell address

Based on the memory cell address, input data is propagated to one of the memory cells

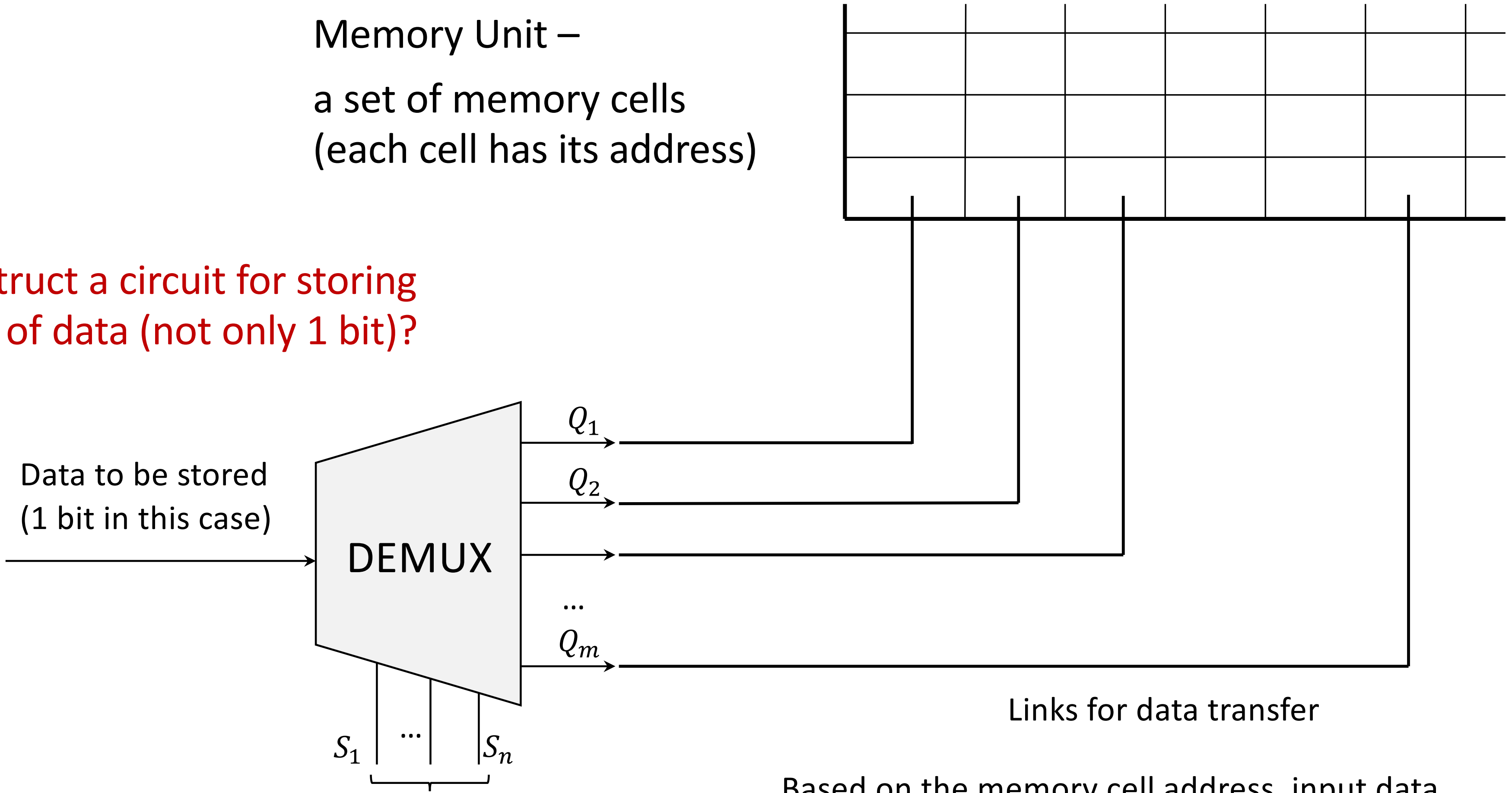
Sample Use Case for Demultiplexer: Memory Addressing (Simplified View)

Memory Unit –

a set of memory cells
(each cell has its address)

Question:

How to construct a circuit for storing multiple bits of data (not only 1 bit)?

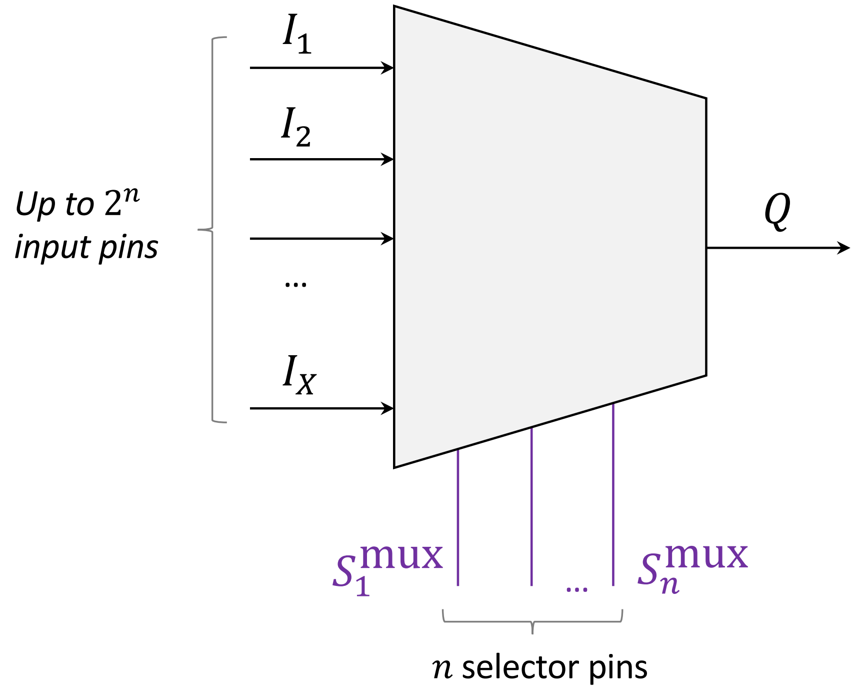


Memory cell address

Based on the memory cell address, input data is propagated to one of the memory cells

X-to-1 Multiplexer

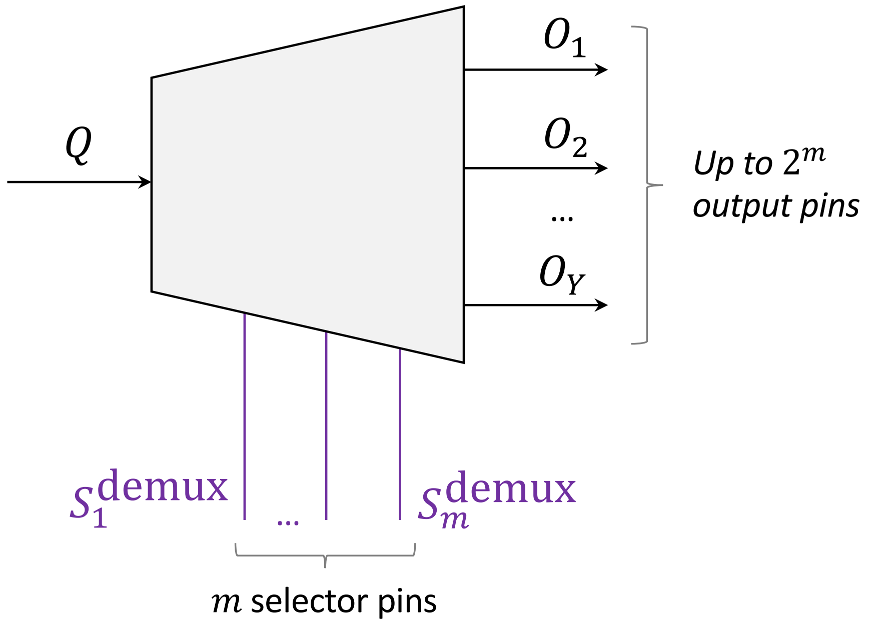
Based on the values of its n selector (control) pins, a multiplexer sets the value of its output Q to the value of one of its $X \leq 2^n$ inputs



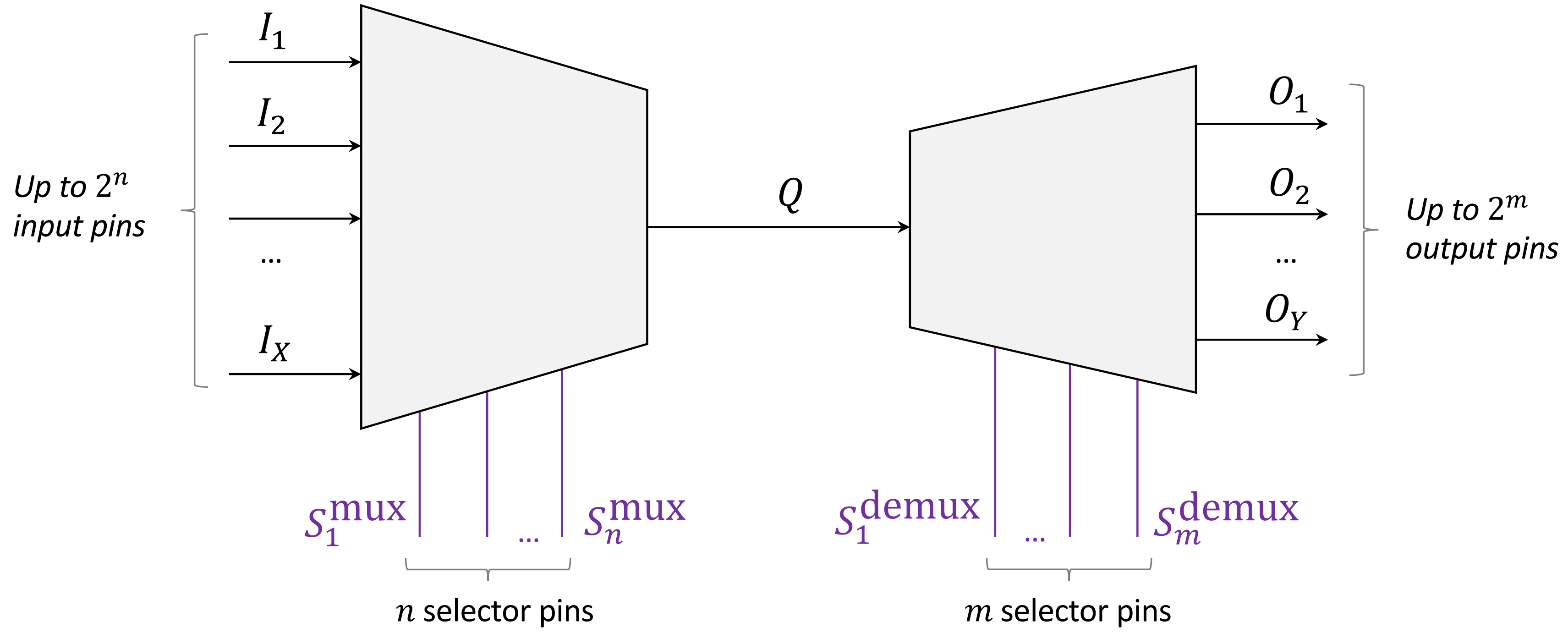
Question: How to construct an X-to-Y Multiplexer?

1-to-Y Demultiplexer

Based on the values of its m selector pins, a demultiplexer outputs its input value Q at one of its $Y \leq 2^m$ output pins, while all other output pins are set to 0s

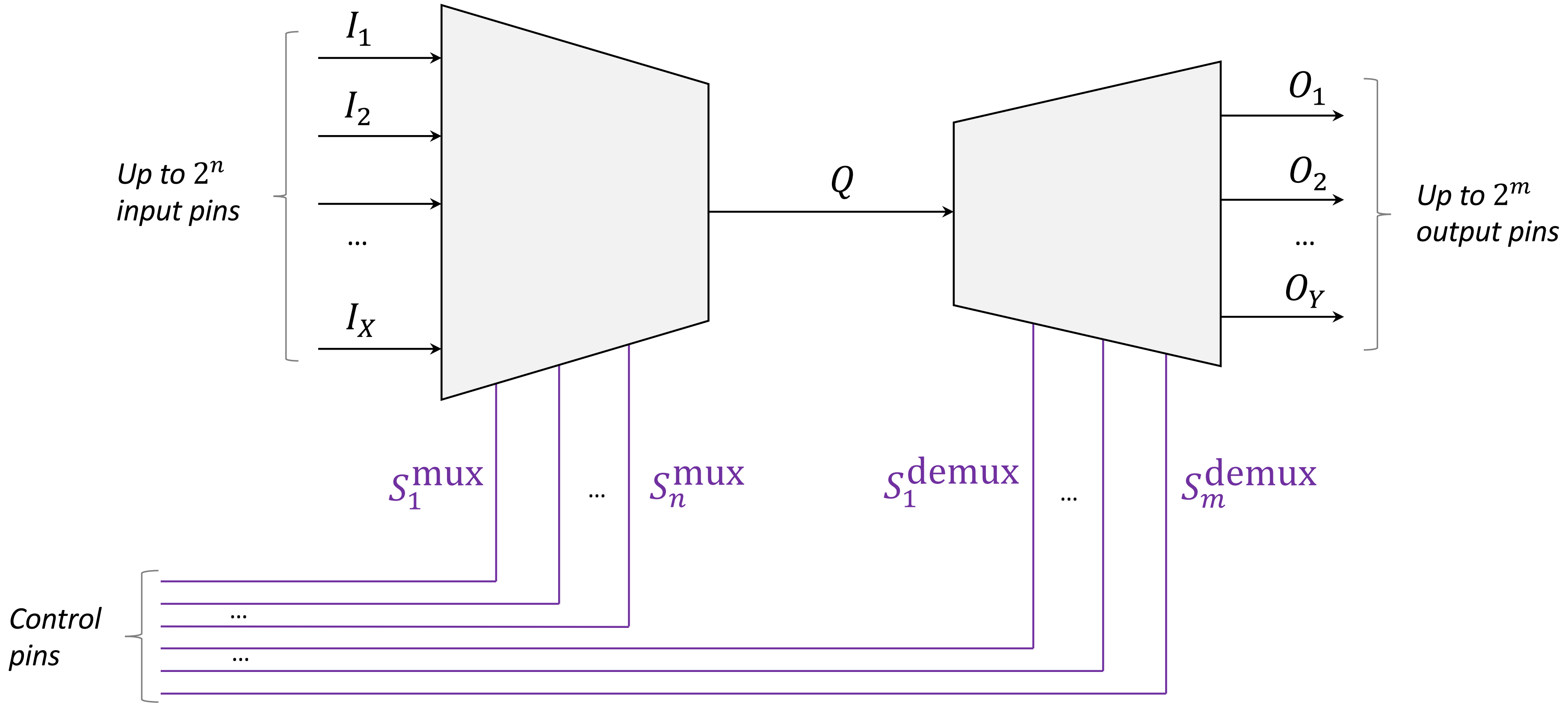


Multiplexer-Demultiplexer Composition

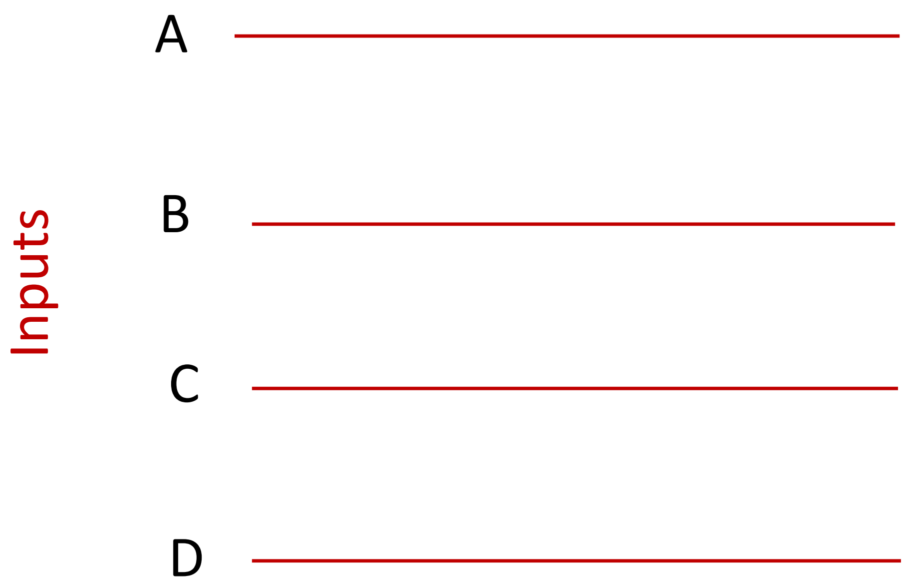


X-to-*Y* Multiplexer:

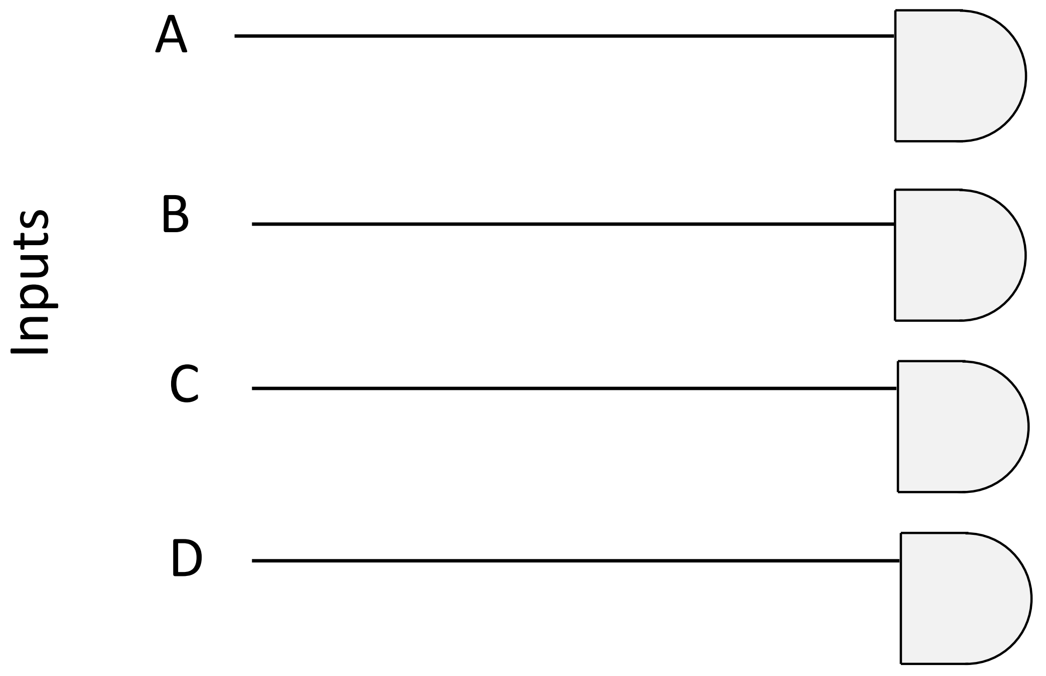
Chooses one of its *X* input pins, and outputs its value at one of its *Y* output pins, while all other output pins are reset to 0s (usually)



4-to-1 Multiplexer Implementation

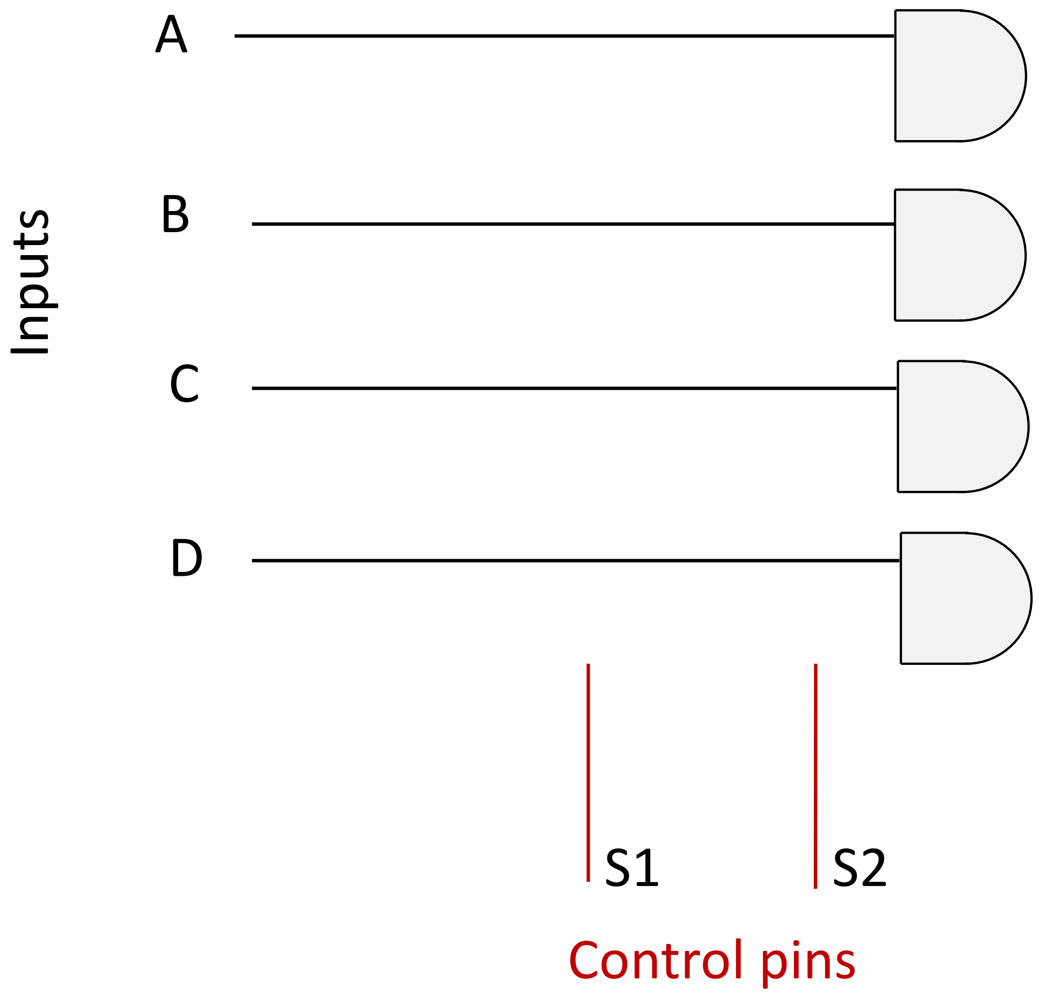


4-to-1 Multiplexer Implementation

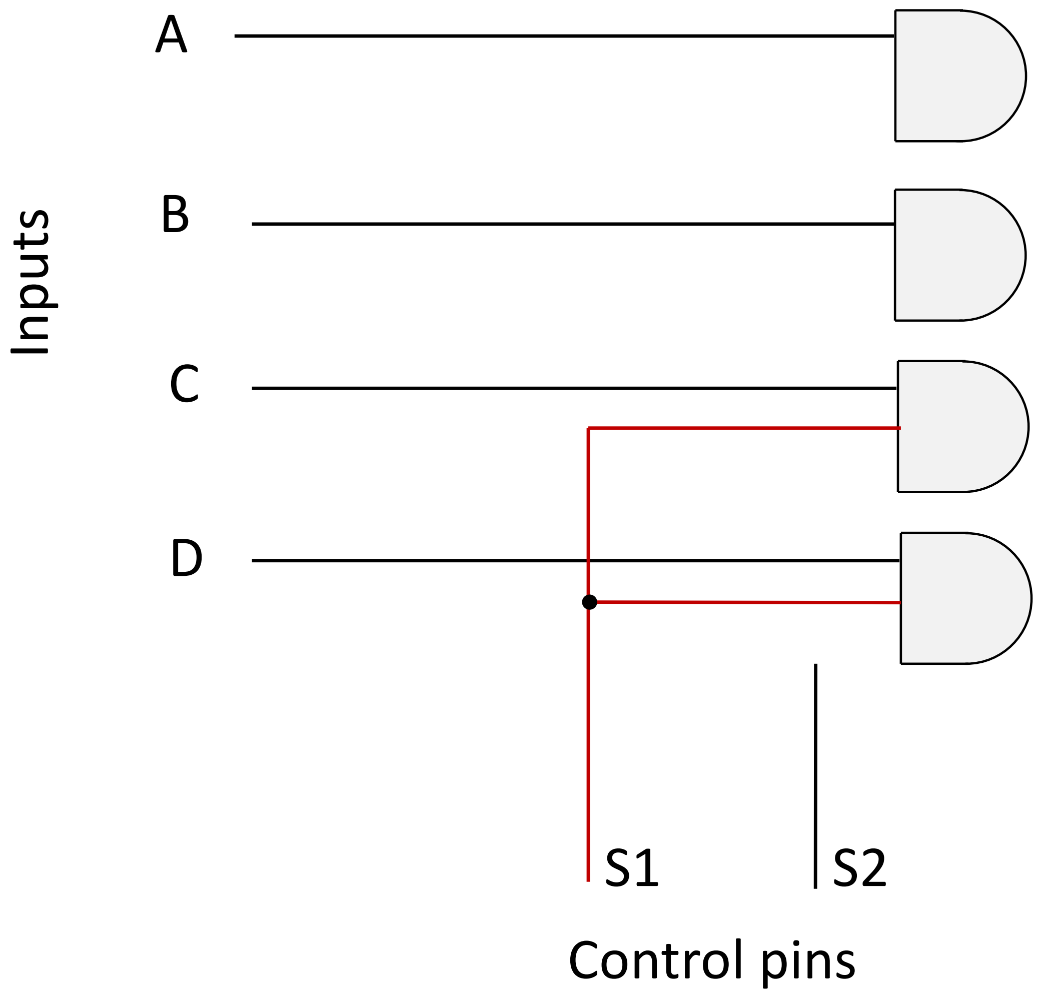


AND logic gate
for each pin

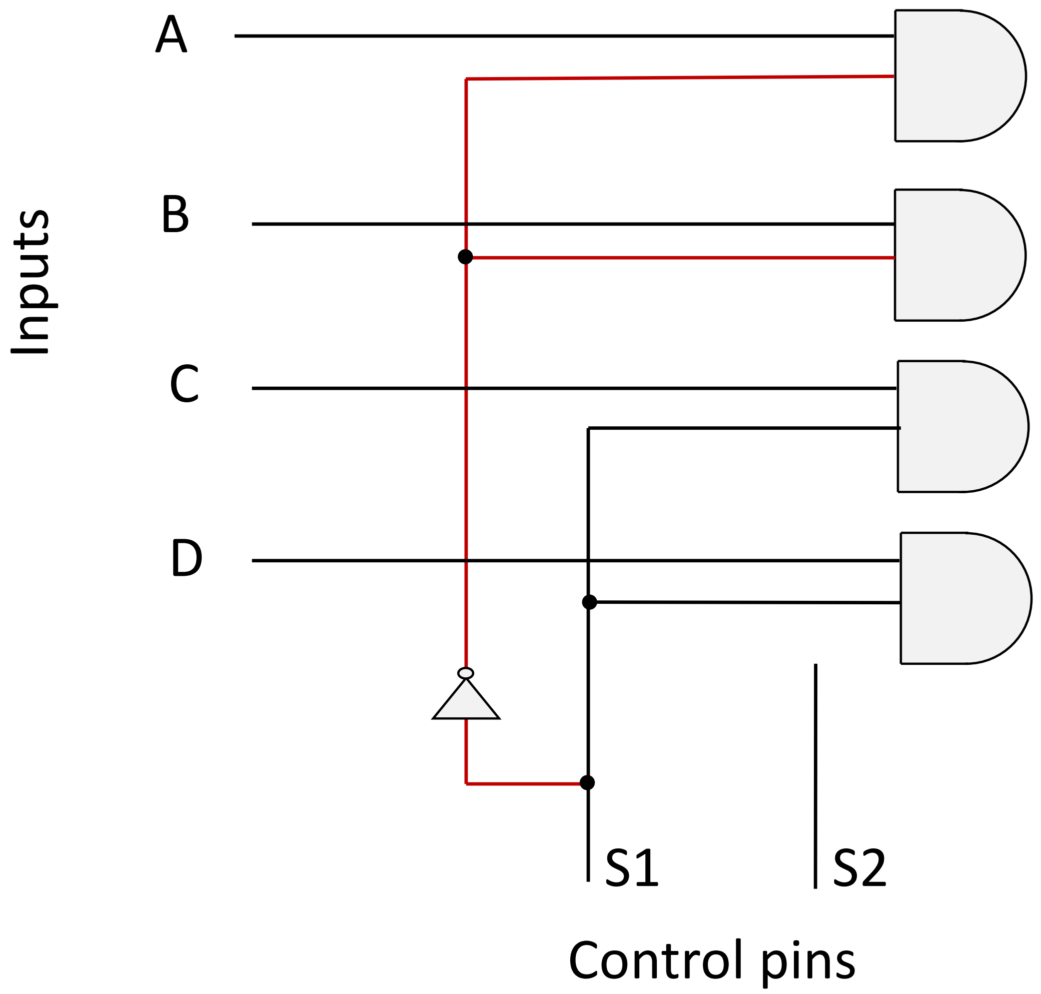
4-to-1 Multiplexer Implementation



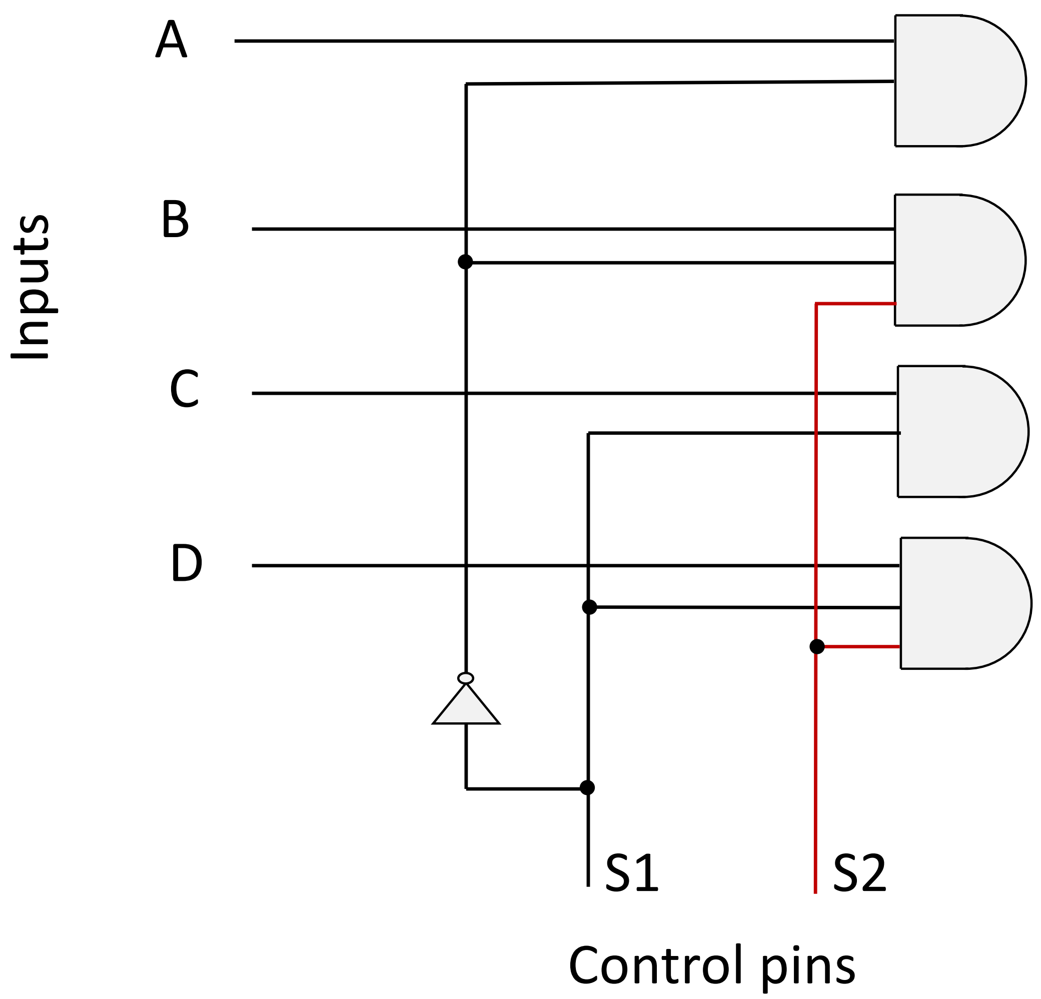
4-to-1 Multiplexer Implementation



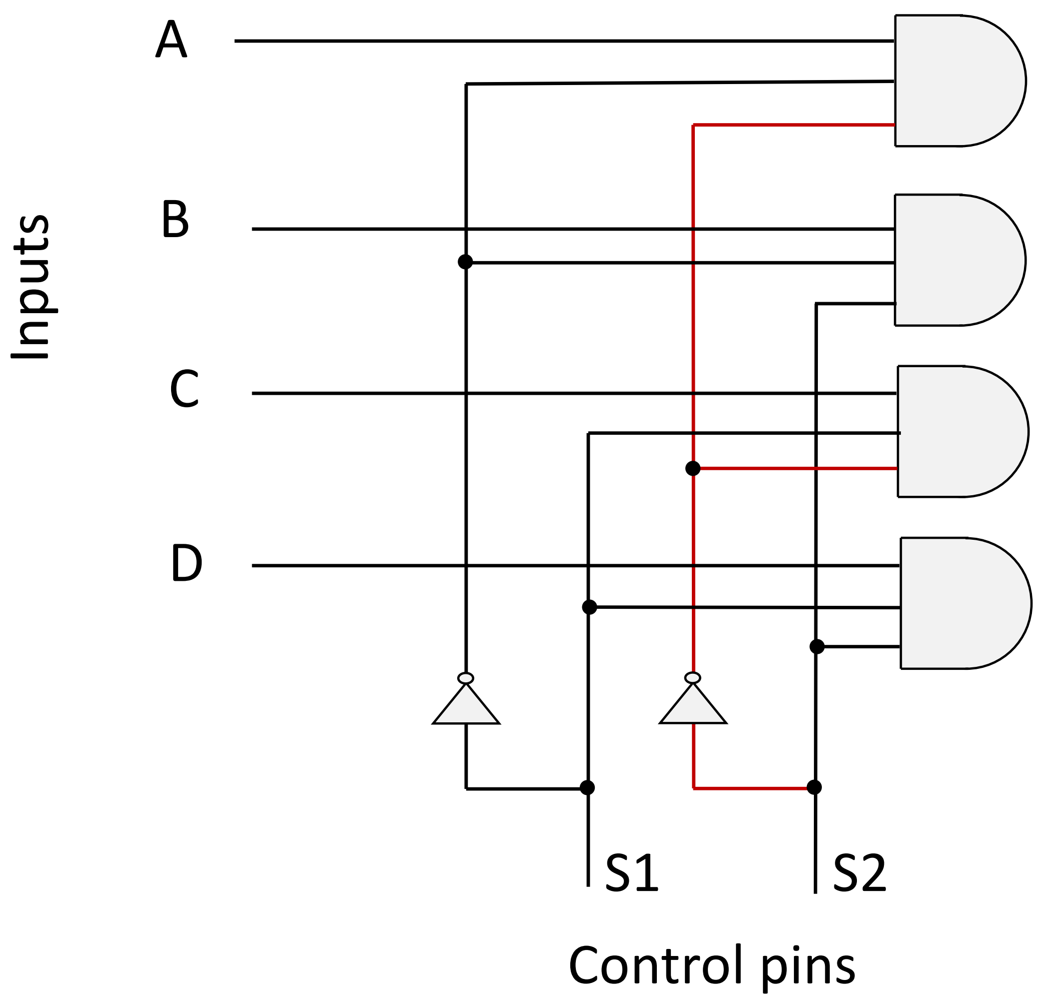
4-to-1 Multiplexer Implementation



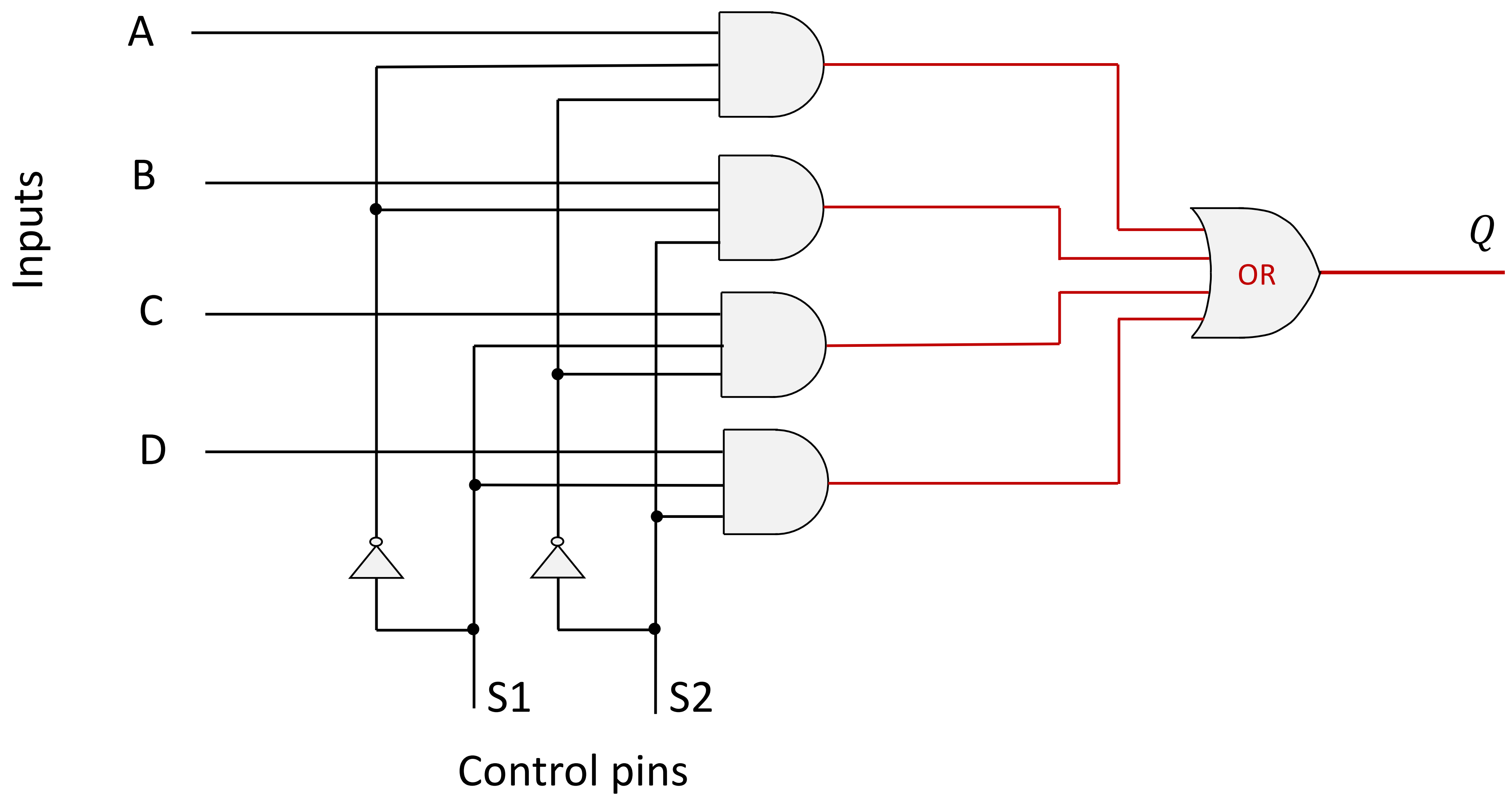
4-to-1 Multiplexer Implementation



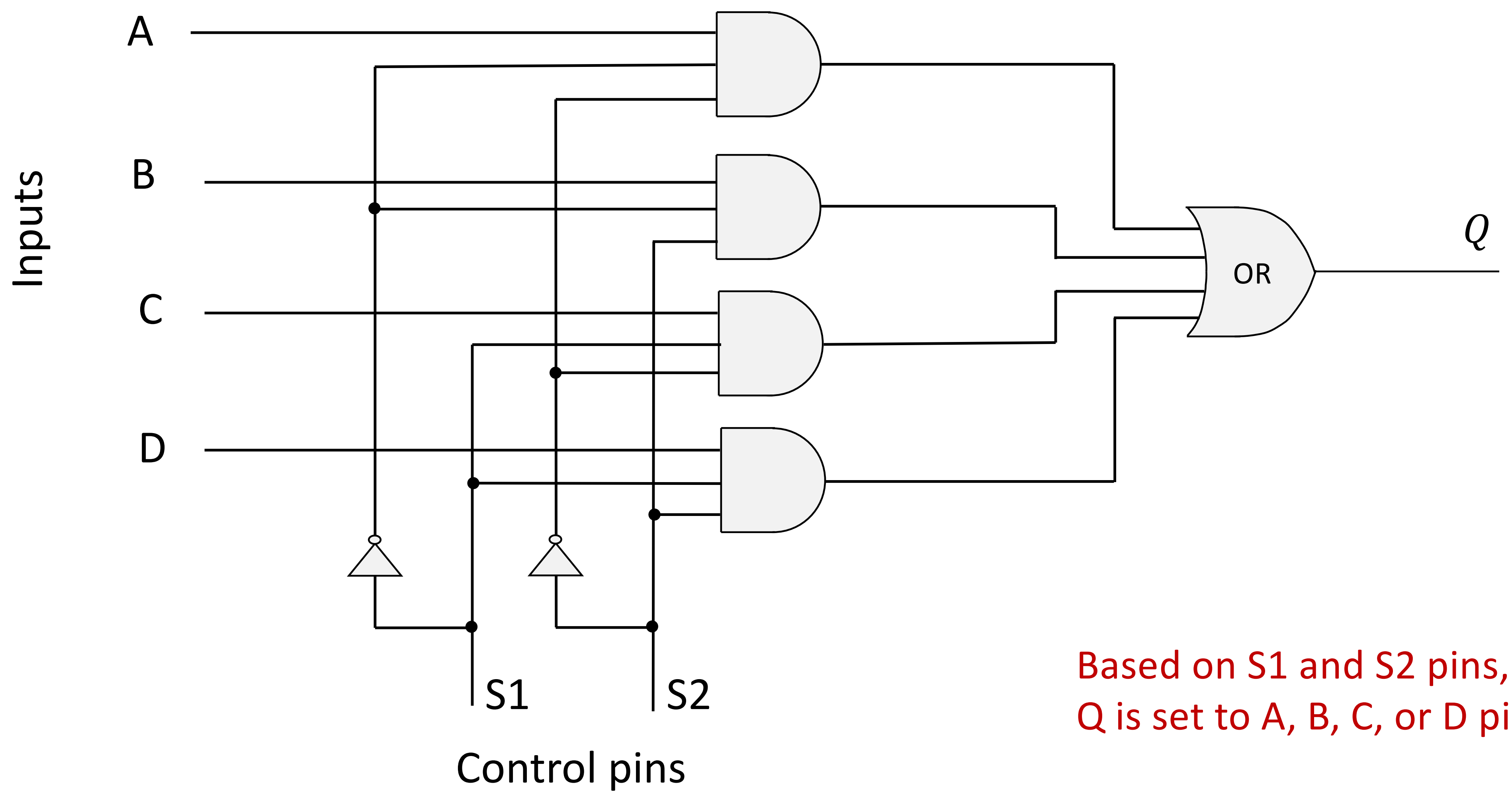
4-to-1 Multiplexer Implementation



4-to-1 Multiplexer Implementation

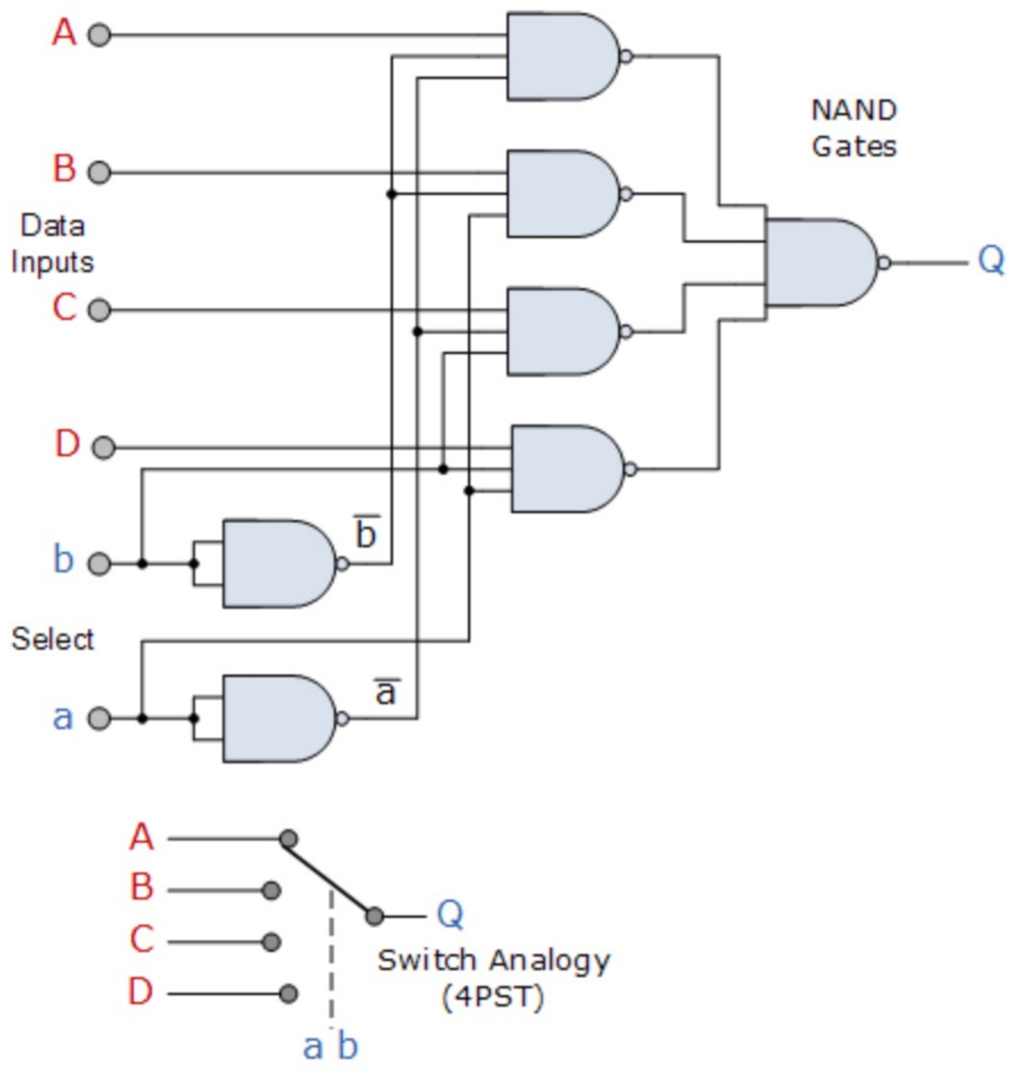


4-to-1 Multiplexer Implementation



Based on S1 and S2 pins,
Q is set to A, B, C, or D pin

An Alternative Implementation: 4-to-1 Multiplexer by Using NAND Gates



A
B
Data Inputs
C
D
b
Select
a
$\overline{b}$
$\overline{a}$
NAND Gates
Q
A
B
C
D
Q
Switch Analogy (4PST)
a b