

Digital Logic

Outline

- **Gates and Boolean Algebra**
- Combinational Logic Circuits
- Sequential Logic Circuits
- Memory Organization

Boolean Algebra

- **Boolean algebra** is a mathematical system for the manipulation of variables that can have one of two values.
 - In formal logic, these values are “true” and “false.”
 - In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”
- A **Boolean operator** performs an operation on Boolean variables.
 - Common Boolean operators include AND, OR, and NOT.
 - A Boolean operator can be completely described using a truth table. A **truth table** lists the output for all possible input combinations.

Boolean Operators

The three basic Boolean operators are as follows:

NOT X	
X	\bar{X}
0	1
1	0

X OR Y		
X	Y	$X+Y$
0	0	0
0	1	1
1	0	1
1	1	1

X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Operators

NOT operator:

- equivalent to `!` in C++
- called the complement
- denoted as either \bar{A} or A'

OR operator:

- equivalent to `||` in C++
- called the Boolean sum
- denoted $A + B$

AND operator:

- equivalent to `&&` in C++
- called the Boolean product
- denoted AB

Boolean Functions

- A **Boolean function** is a function of one or more Boolean variables.
 - Produces an output of one more variables.
- Boolean operations have rules of precedence:
 - NOT (highest priority), AND, OR.
- Boolean functions can be represented:
 - Algebraic expression such as $F(x, y, z) = x\bar{z} + y$
 - Truth table: Explicitly list the output for each input combination
 - How many rows in truth table of function with n variables?
 - 2^n

Truth Table Example $F(x, y, z) = x\bar{z} + y$

Inputs			Intermediates		Output
x	y	z	\bar{z}	$x\bar{z}$	$x\bar{z} + y$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Truth Table Example $F(x, y, z) = x\bar{z} + y$

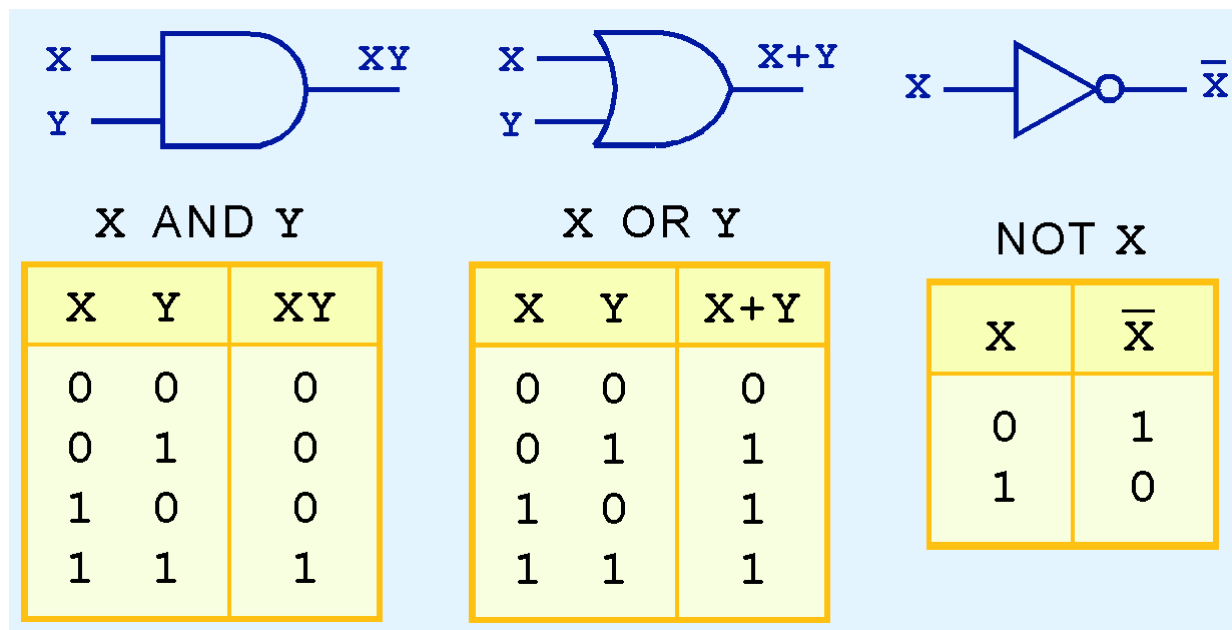
Inputs			Intermediates		Output
x	y	z	\bar{z}	$x\bar{z}$	$x\bar{z} + y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Logic Gates

- Boolean functions are implemented in digital computer circuits called *gates*.
- A gate is an electronic device that produces a result based on one or more input values.
 - In reality, gates consist of one to six (or more) transistors.
 - Digital designers think of them as a single unit (abstraction).

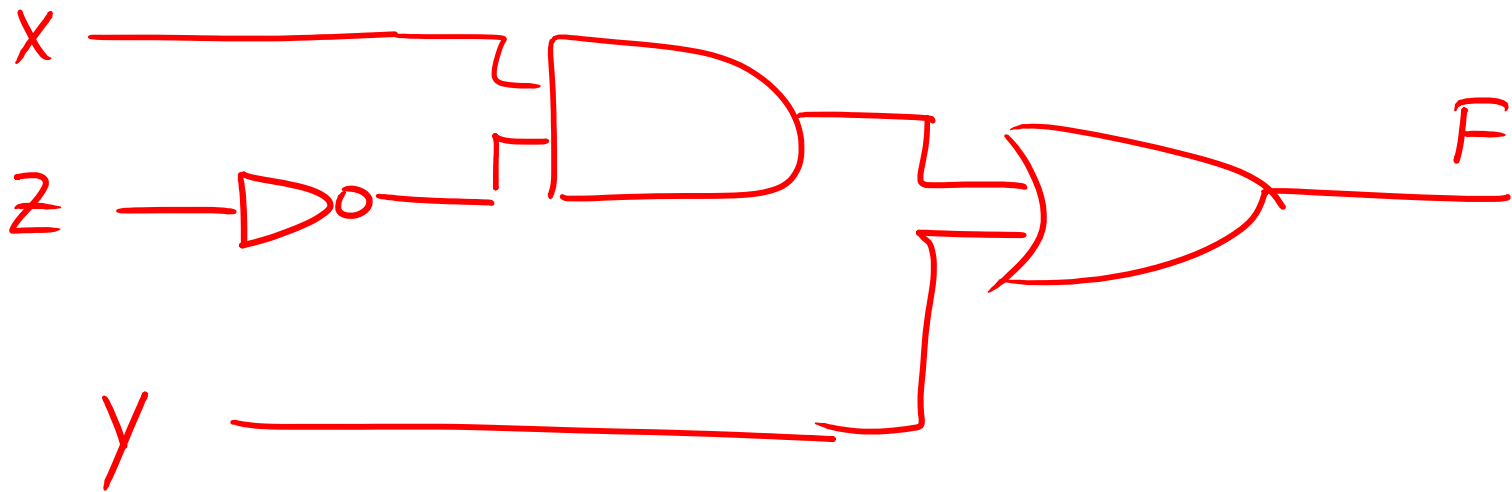
Logic Gates

- These gates directly correspond to their respective Boolean operations:



Logical Circuit Example

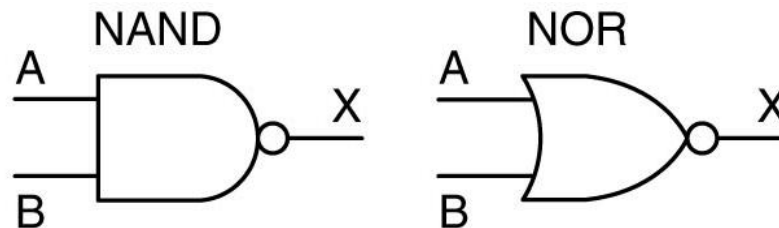
- Draw a circuit representing the function: $F(x, y, z) = x\bar{z} + y$



NAND and NOR Gates

There are also two other important gates:

- Note: The circle in the NAND and NOR gates means complement.
- This shorthand for complement will be used in future circuits.

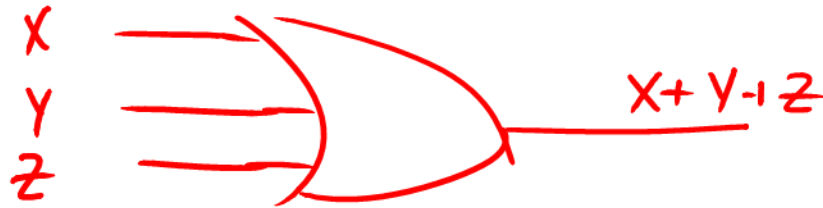


A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

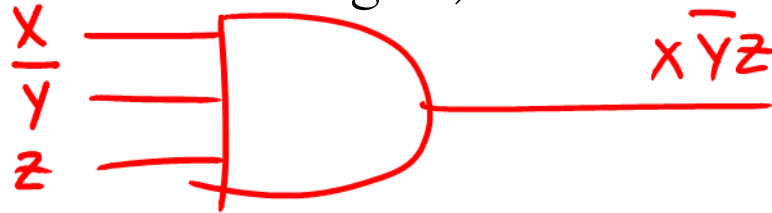
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gate Variations

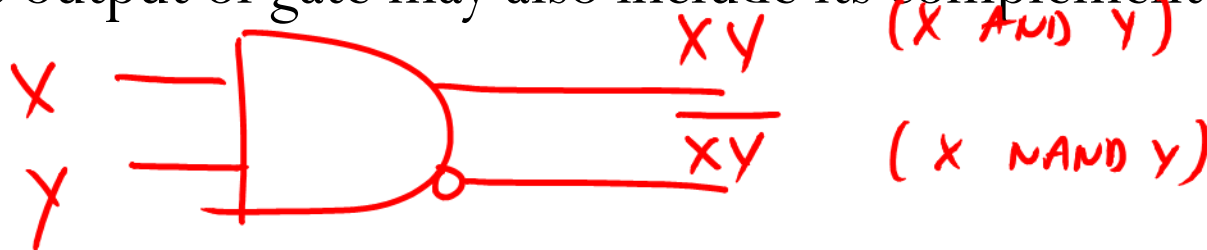
- Gates (except NOT) can have more than two inputs.



- Sometimes the complement is expressed directly as input (without a NOT gate).



- The output of gate may also include its complement.



Designing Logical Circuits

Beyond correctness, what factors does a hardware designer have to account for?

- Minimize power consumption
- Reduce chip area needed
- Increase speed

Simplifying Boolean Functions

- There are many different circuits that can be used to implement the same Boolean function. Which is the best one to use?
- One way of simplifying a complex Boolean function is to use Boolean algebra.
 - Many of the standard algebraic properties apply.
 - Additional properties take advantage of the fact that a variable is either 0 or 1.

Boolean Algebra Properties

Here is table of common Boolean algebra properties:

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	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{\bar{A} + \bar{B}} = \bar{A}\bar{B}$

Minterms and Terms

- A **minterm** is a product (AND) involving all inputs (some may be complemented) to the function.
- For example, $F(A, B, C)$ is a three input function.
 - The following are minterms: $ABC, A\bar{B}C, \bar{A}\bar{B}C$
 - The following are not minterms:
 - AC does not involve B
 - \overline{ABC} this is NAND, not AND
 - $(A + B)C$ OR operator not permitted
- A **term** is identical to a minterm except that it does not require all inputs.
 - For instance, $A\bar{B}$ and A are considered terms but not minterms.

Sum-of-Products Form

- The **sum-of-products form** is a Boolean sum of terms.
- Example: $F(A, B, C) = A\bar{B}C + BC + \bar{A}$
- Any Boolean function can be written in sum-of-products form.
 - Each row in the truth table refers to a different minterm.
 - Simply create a Boolean sum of the minterms that correspond to each row that has an output of 1.

Sum-of-Products Form Example

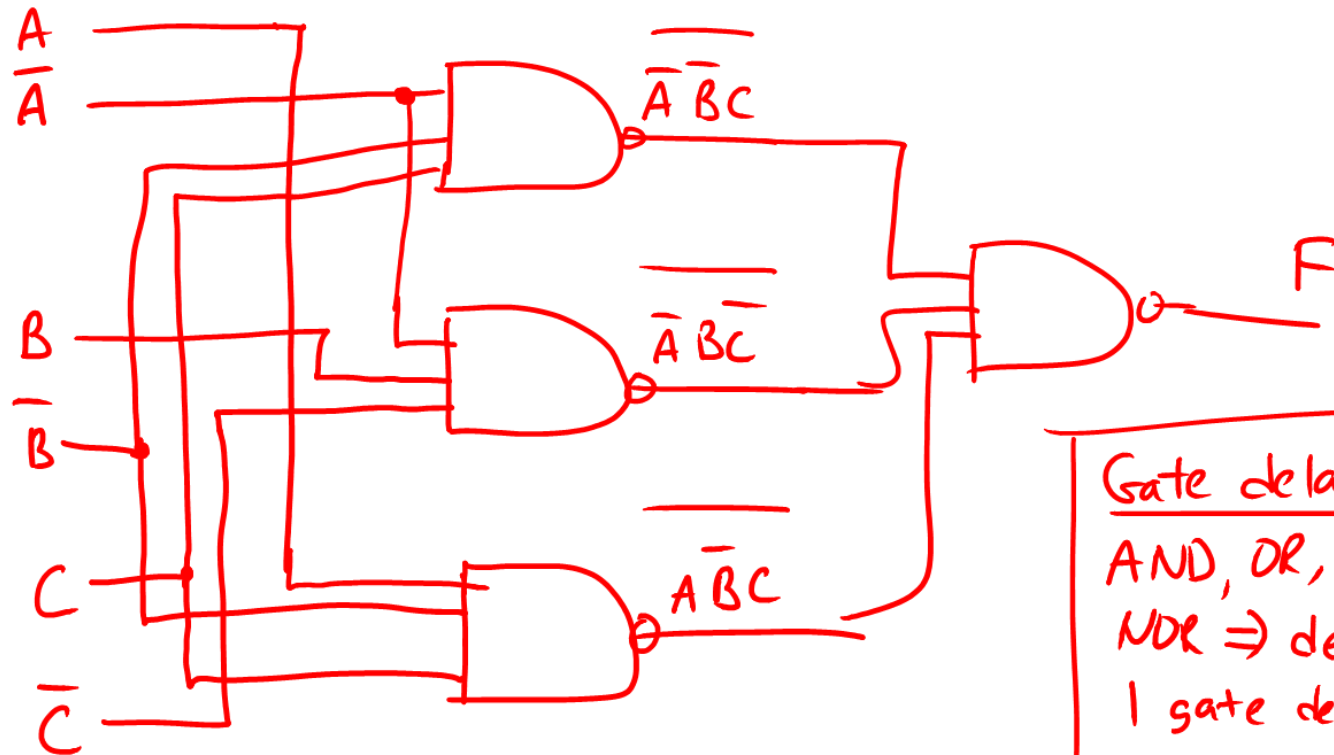
Express the Boolean function represented by the following truth table in sum-of-products form.

- $F = !A!BC + !AB!C + A!BC$

<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Sum-of-Products Form Example

$$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C$$



Gate delays:

AND, OR, NAND,

NOR \Rightarrow delay of
1 gate delay unit

NOT \Rightarrow 0 delay

Sum-of-Products Form

- Advantages from circuit perspective:
 - Fast - maximum gate propagation delay is 2.
 - All gates can be replaced with NAND gates directly.
- Disadvantages from circuit perspective:
 - Might not be optimal in terms of number of gates.
 - Number of inputs on OR gate can be large for functions with several terms.

More on Designing Logical Circuits

- Typically use only NAND gates.
 - It is a universal gate: you can implement any Boolean function with just NAND gates.
 - NAND gates require fewer transistors than AND or OR gates.
- The number of inputs on a gate is capped.
 - Gates with large number of inputs consume too much power and take up too much space.
- Designers use additional algorithms and tools to further minimize Boolean functions.
 - Karnaugh maps and tabulation method

Combinational and Sequential Logic

There are two types of digital circuits:

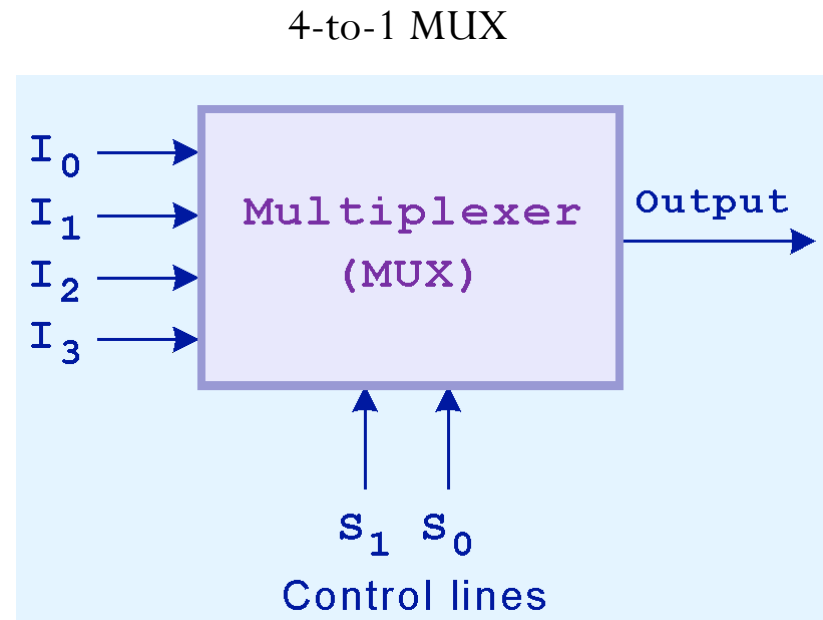
- **Combinational circuits** – the output is exclusively based on the input. There is no memory of prior values.
- **Sequential circuits** incorporate a notion of time and memory. The output depends not only on the input but the prior output.

Outline

- Gates and Boolean Algebra
- **Combinational Logic Circuits**
- Sequential Logic Circuits
- Memory Organization

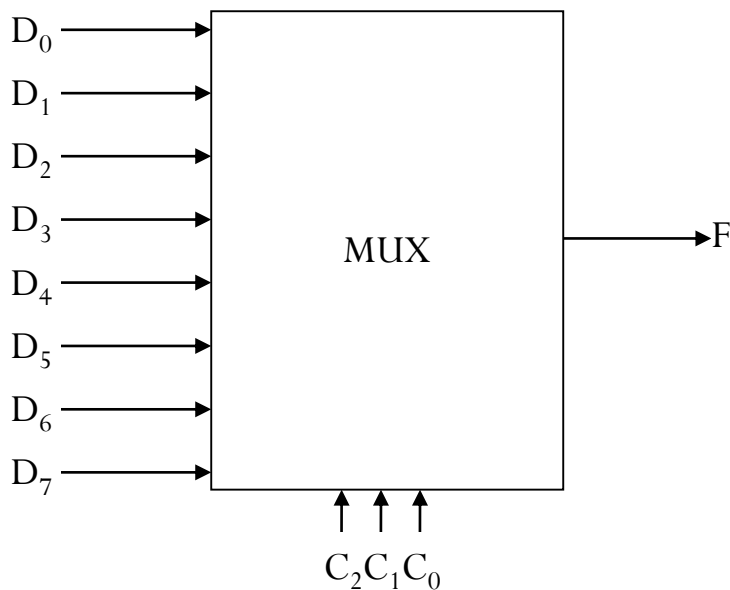
Multiplexers (MUX)

- A **multiplexer (MUX)** selects a single output from several inputs.
- The particular input chosen for output is determined by the value of the multiplexer's control lines.
- To be able to select among n inputs, $\log_2 n$ control lines are needed.
- The term n -to-1 mux is used to indicate there are n data input lines.



Multiplexers

The control inputs are treated as an unsigned binary number and corresponds to which data line to select.



C_2	C_1	C_0	F
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

Multiplexers Applications

- Implementing truth tables
- Parallel-to-serial data conversion
 - Keyboard: keystroke (8 bits) over serial link (USB)
- Inverse of a multiplexer is a **demultiplexer**

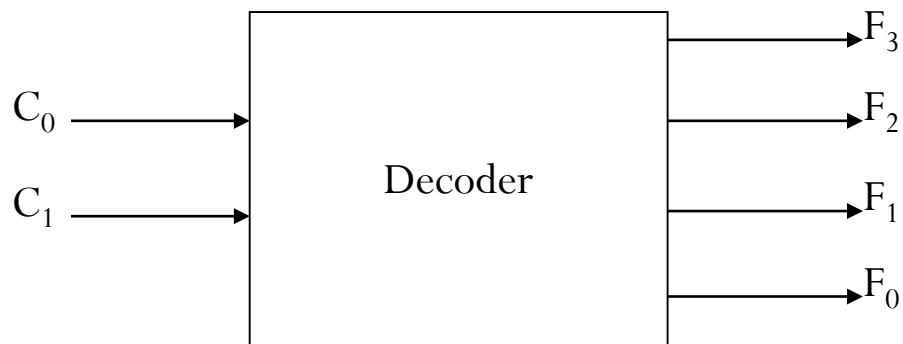
Decoders

- A **decoder** is used to select a line based on a binary number (address).
 - Has n inputs, 2^n outputs.
 - Exactly one output line is on (1), the rest are off (0).
- Think of the input as n bit binary number. The line corresponding to the number is 1.



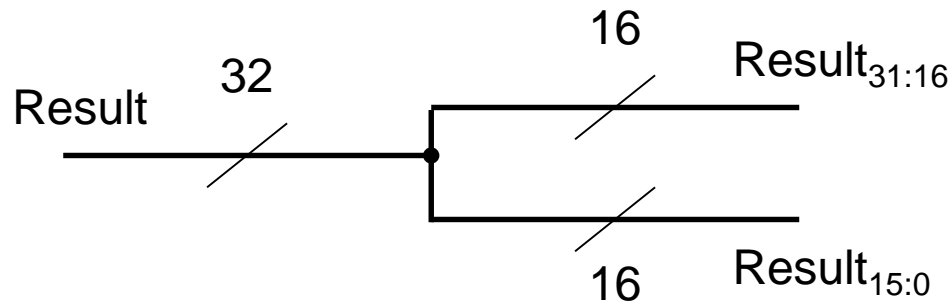
Decoders

C_1	C_0	F_3	F_2	F_1	F_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



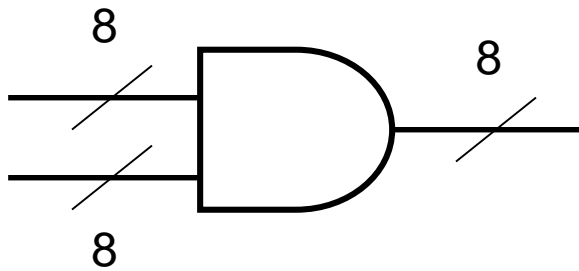
Multiple Bit Wires

- So far all of the components dealt with a small number of bits.
- Circuit diagrams would get large and cluttered if we had to specify each bit independently.
- A single wire in the circuit diagram can represent multiple bits.
 - Number indicates the number of bits.
 - Can also label the wire to give the label meaning. The label can be used to convey the number of bits.
 - Bits are numbered from $n-1$ to 0.

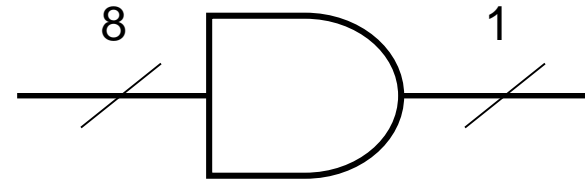


Multiple Bit Components

- Components are also modeled with multiple bit input and outputs:



Eight 2-input AND gates
(bitwise AND)



One 8-input AND gate

Shifters

- A **shifter** takes a series of bits and shift the bits one bit to the left or one bit to the right.
- Let X be input and Y be output. Assume we are shifting one byte.
- For *left shifts*:
 - $Y_n = X_{n-1}$ for $n \neq 0$
 - $Y_0 = 0$
- For *logical right shifts*:
 - $Y_n = X_{n+1}$ for $n \neq 7$
 - $Y_7 = 0$
- For *arithmetic right shifts*:
 - $Y_n = X_{n+1}$ for $n \neq 7$
 - $Y_7 = X_7$

Shifting and Arithmetic

- Shifting a number left one bit is equivalent to multiplying by 2. (overflow is possible)
- Shifting a number right one bit is equivalent to integer division by 2. (remainder is discarded)
- The arithmetic right shift keeps negative numbers negative.
 - The most significant bit (which indicates the sign) is replicated.

Shifter Example

Shift the byte 1011 0100:

a) one bit to the left

0110 1000

b) one bit to the right (logical shift)

0101 1010

c) one bit to the right (arithmetic shift)

1101 1010

Adders and Half Adders

- An **adder** computes the binary sum.
 - Initially look at adding individual bits.
 - Then build a circuit that can add larger numbers.
- **half adder**: adds two bits (A , B) to produce a sum (S) and a carry out bit (C_{out}).

Half Adder Truth Table

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Class Problem: Full Adder

A **full adder** adds three bits: A , B , and carry in (C_{in}). Produces sum (S) and carry out (C_{out}).

Complete the truth table for the full adder and derive the Boolean formulae.

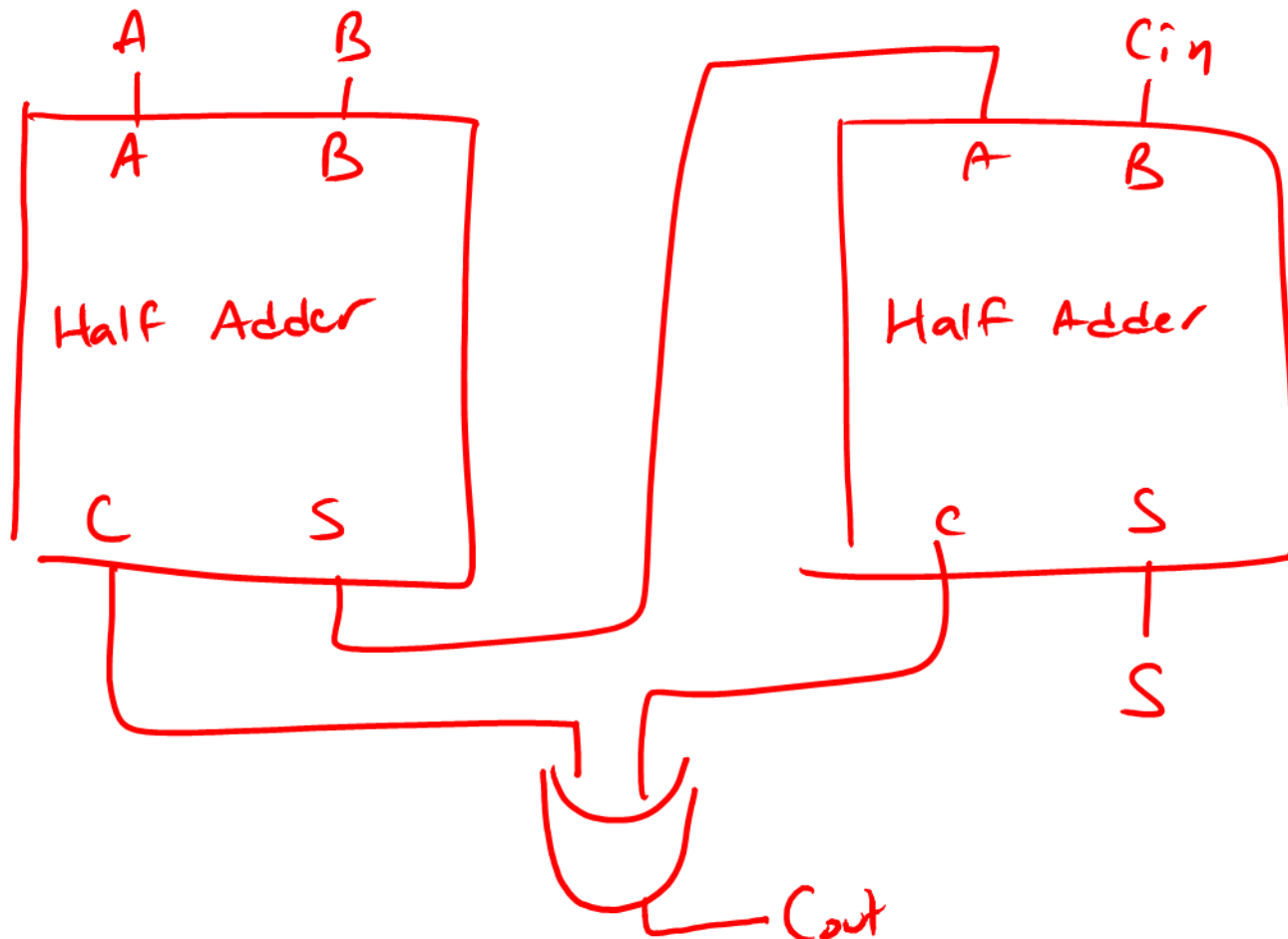
Class Problem Solution

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

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$
$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$

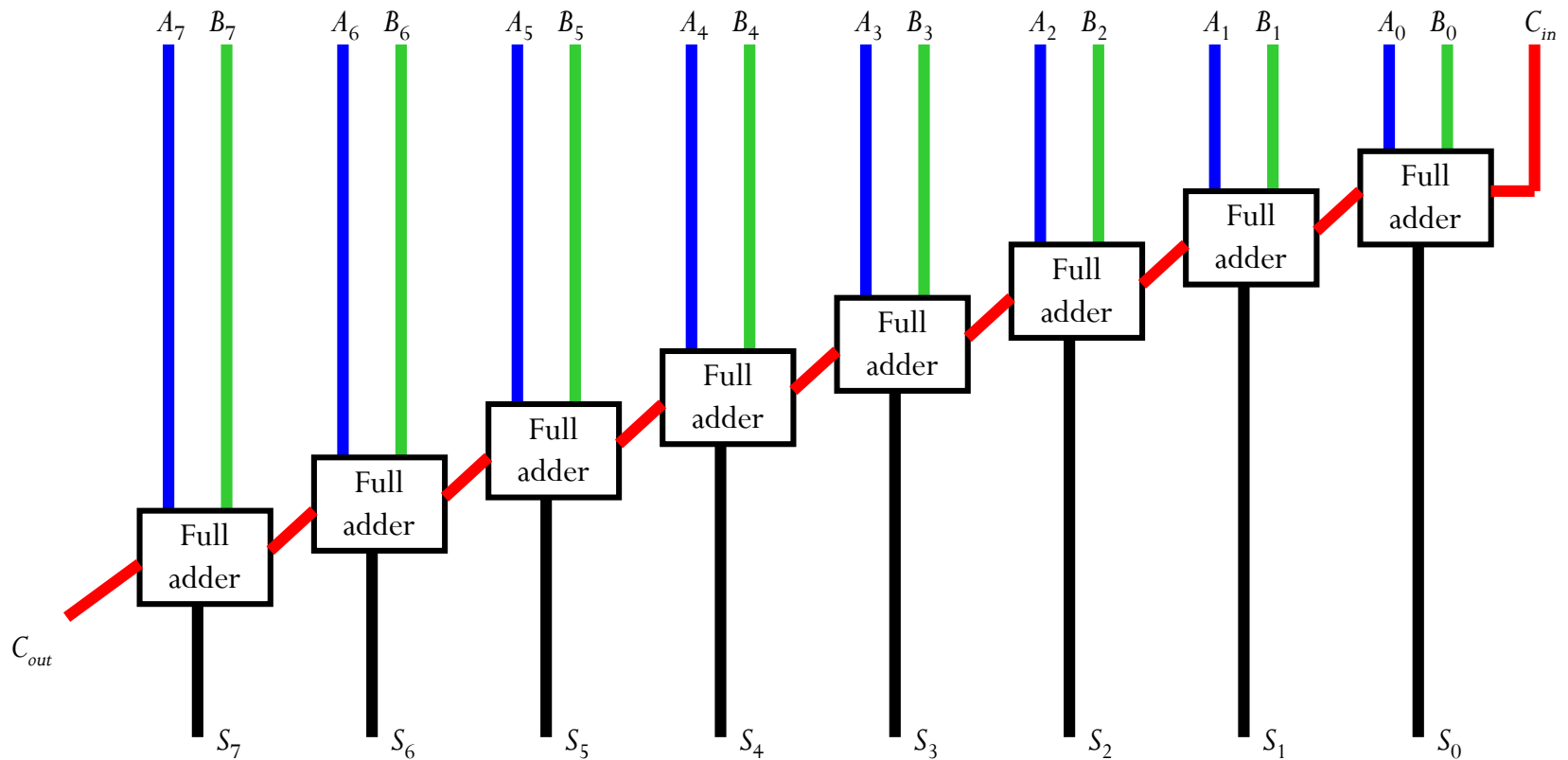
Why a Full Adder?

- A full adder can be constructed using two half adders.



Ripple Carry Adder

- To construct an n -bit adder, we can use n full adders, chain together such that C_{out} of bit i is fed in as C_{in} of bit $i + 1$:
 - How to speed up addition?

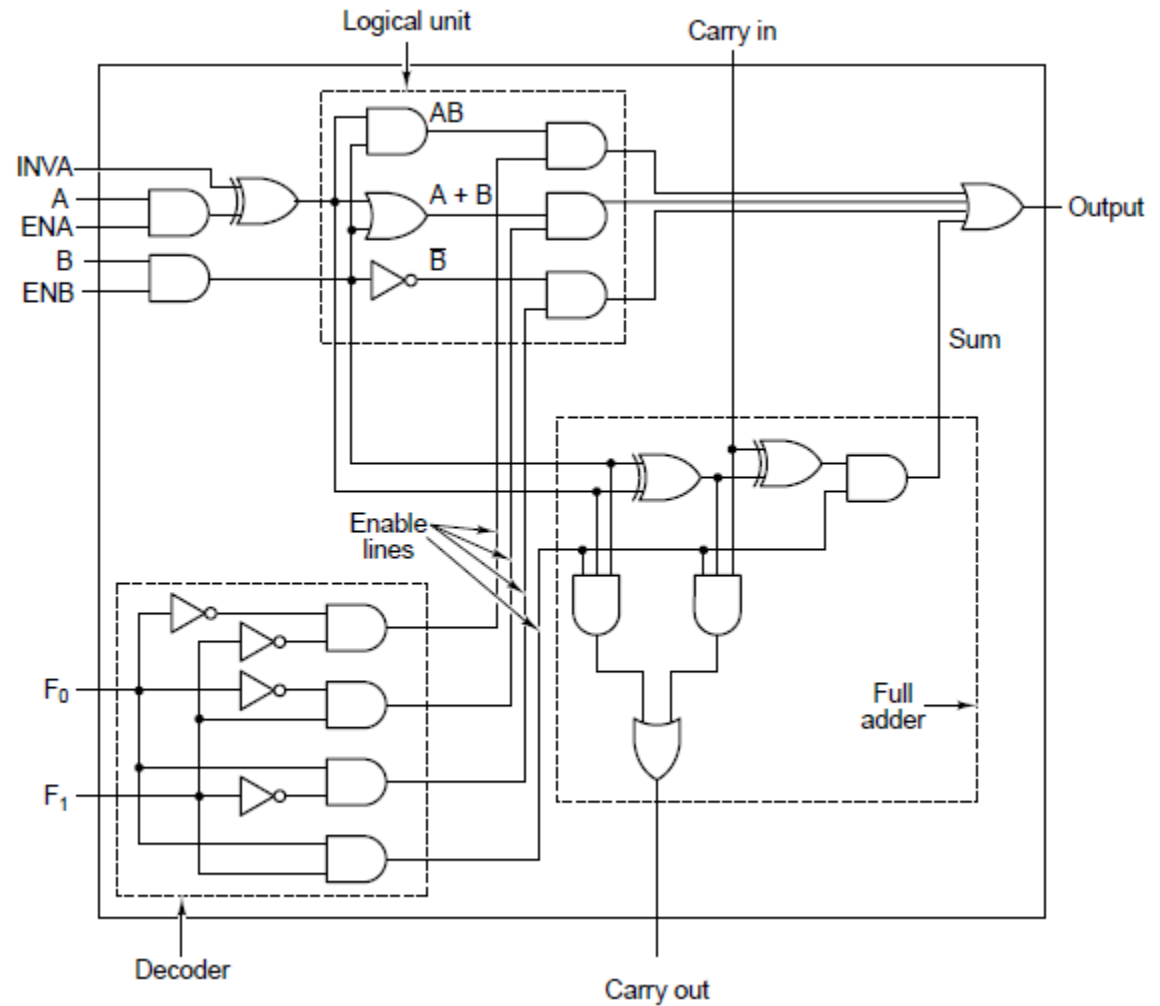


Arithmetic Logic Units (ALU)

- The **Arithmetic Logic Unit (ALU)** is responsible for executing instructions that involve arithmetic and logical operations.
- Inputs:
 - data (two n -bit numbers)
 - control lines (determines the operation to be performed by the ALU)
- Output:
 - result (one n -bit number)
 - status flags (such as overflow)

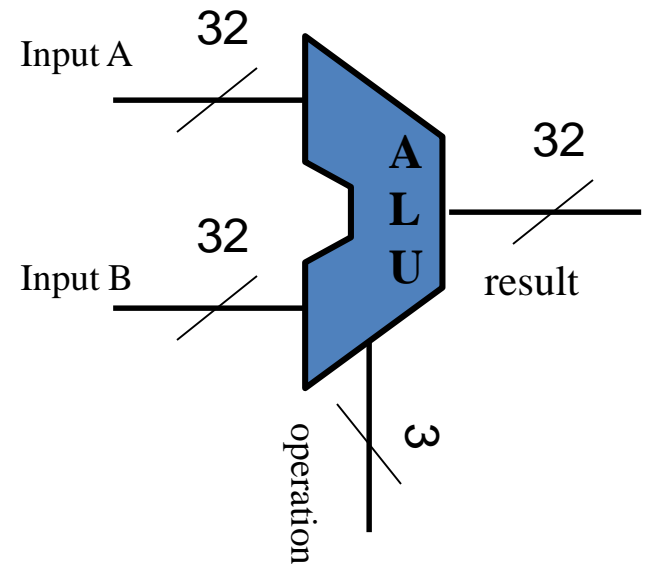
Arithmetic Logic Units

f_0f_1	Operation
00	AB
01	A OR B
10	NOT B
11	A + B



ALU Block Diagram

- If there are n operations, need $\log_2 n$ operation bits.
- An adder uses the same symbol:
 - designated with a '+'
 - only adds - no operation line needed
- A CPU may have one or more adders that are separate from the ALU.

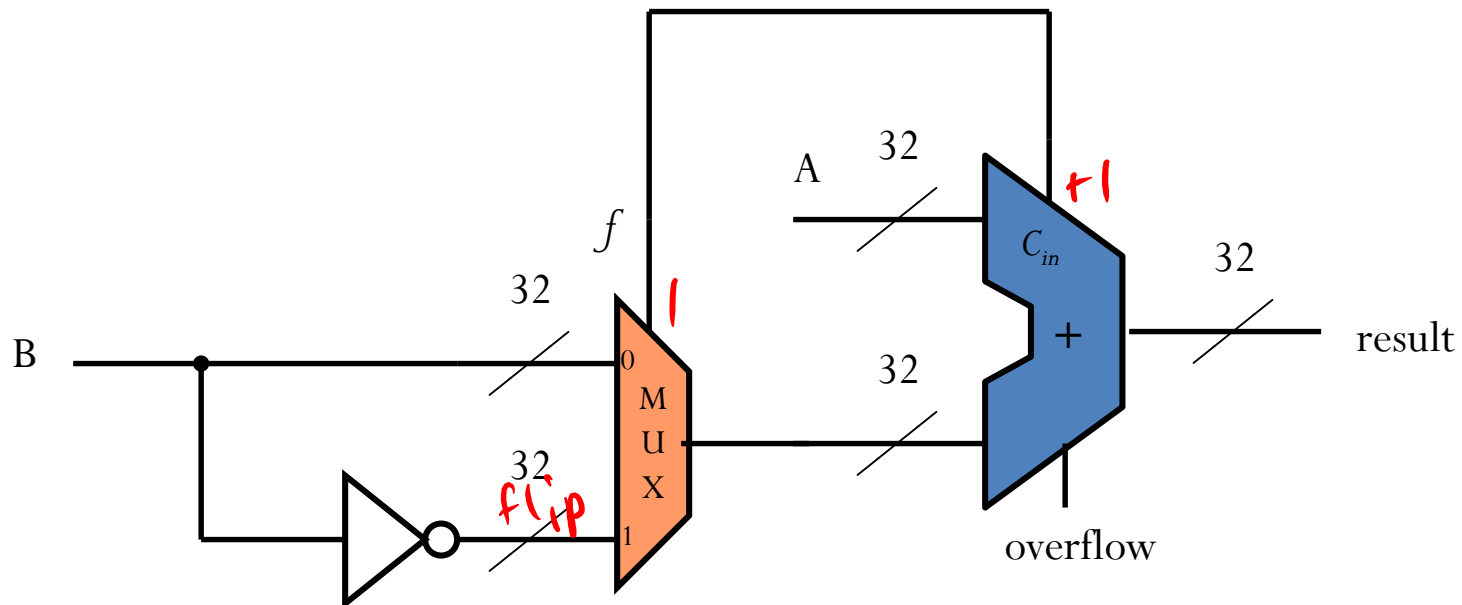


Subtraction using Addition

Consider an ALU that only does addition and subtraction with one control line f :

0 – add ($A + B$)

1 – subtract ($A - B$)



Comparisons

How would you perform the following comparisons? Assume two's complement numbers as always.

- Detecting a number is equal to zero:

If all bits are zero, the number is zero. (NOR.)

- Detecting a number is less than zero:

If the sign bit is 1, number is less than zero.

- Detecting a number is greater than zero:

If the sign bit is zero and the number is not zero, it is greater than zero.

Comparisons

- Determining if A is equal to B:
 - Compute if each bit of A and B match
 - Determine if all the bits match
- Determining if A is greater than ($>$) B:
 - Compute $A - B$ (check for overflow)
 - Determine if the result is greater than zero

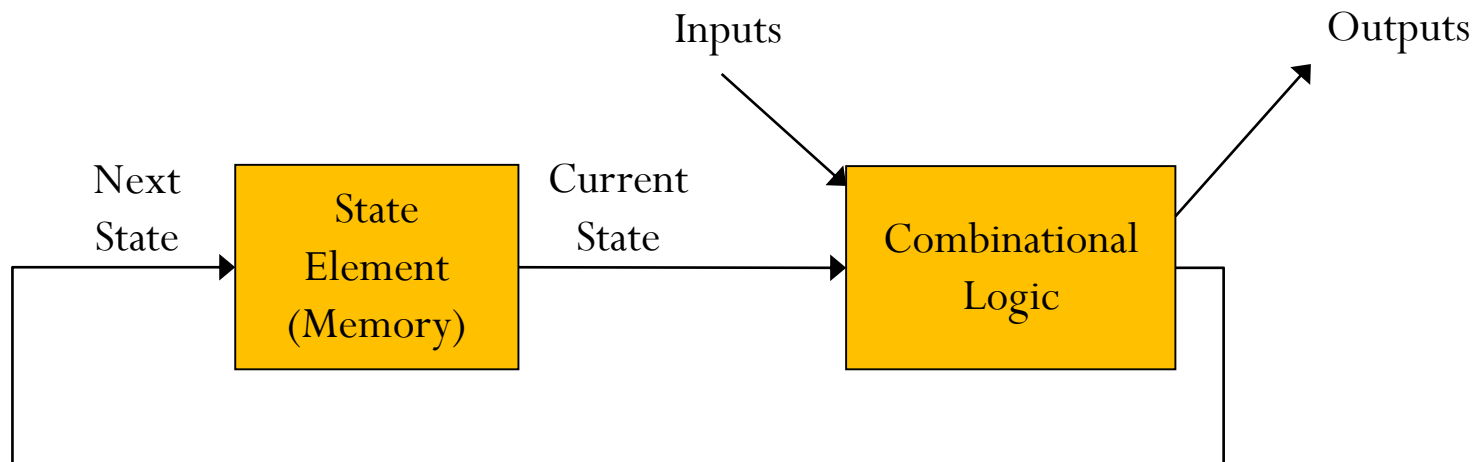
Other comparisons can be derived from these results and their complements.

Outline

- Gates and Boolean Algebra
- Combinational Logic Circuits
- **Sequential Logic Circuits**
- Memory Organization

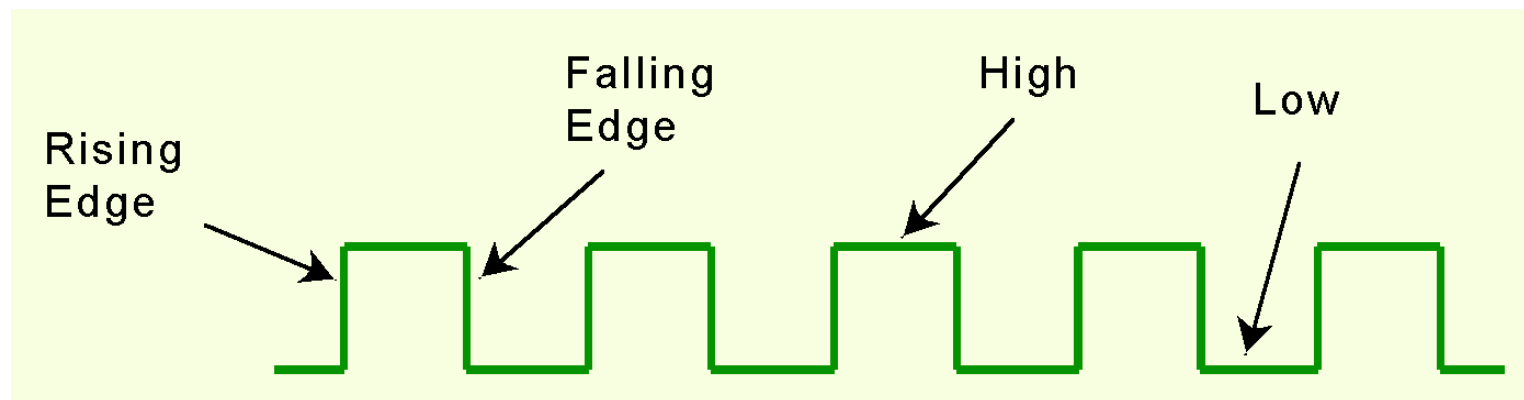
Sequential Logic

- **Sequential logic** circuits allow for a sequence of operations
 - Need a notion of time (clock).
 - Need to remember state in memory (flip-flop).



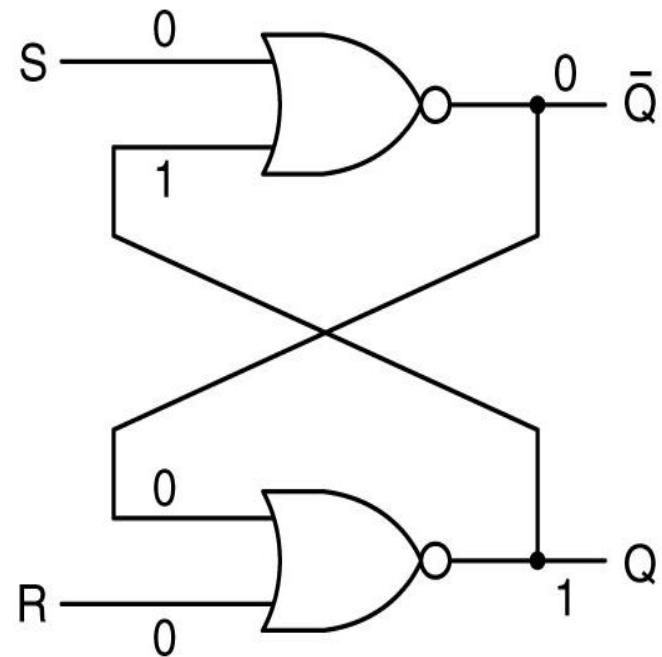
Clocks

- A **clock** is a special circuit that sends electrical pulses at a constant rate.
- State changes occur when the clock ticks.
 - For this class, we'll assume the rising edge.
- The interval between rising edges is the **cycle time**.
- The **frequency** (measured in Hz) is the inverse of the cycle time.



SR Latch

- The **SR (Set/Reset) Latch** is the simplest memory element.
- Changes to the state occur immediately - no clock element has been added yet.
- The output is dependent not only on inputs S and R but the current value of Q.



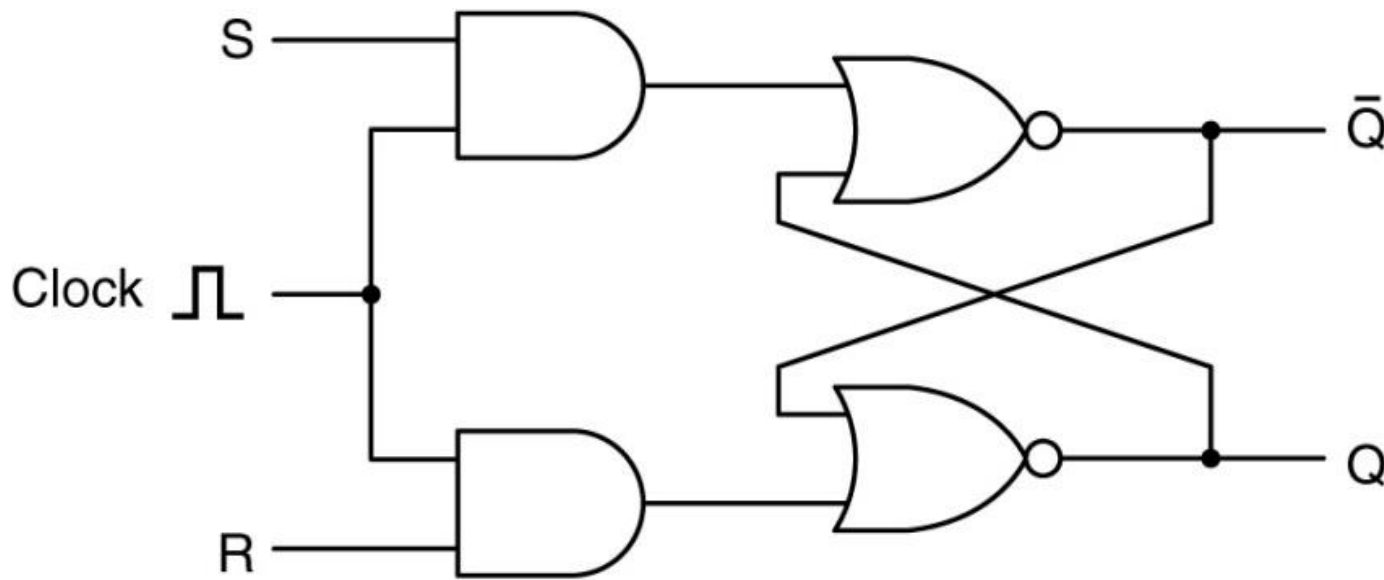
SR Latch Truth Table

S	R	Q_{curr}	Q_{new}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undefined
1	1	1	undefined

S	R	Q_{new}
0	0	Q_{curr} (no change)
0	1	0 (reset)
1	0	1 (set)
1	1	undefined

Clocked SR Latch

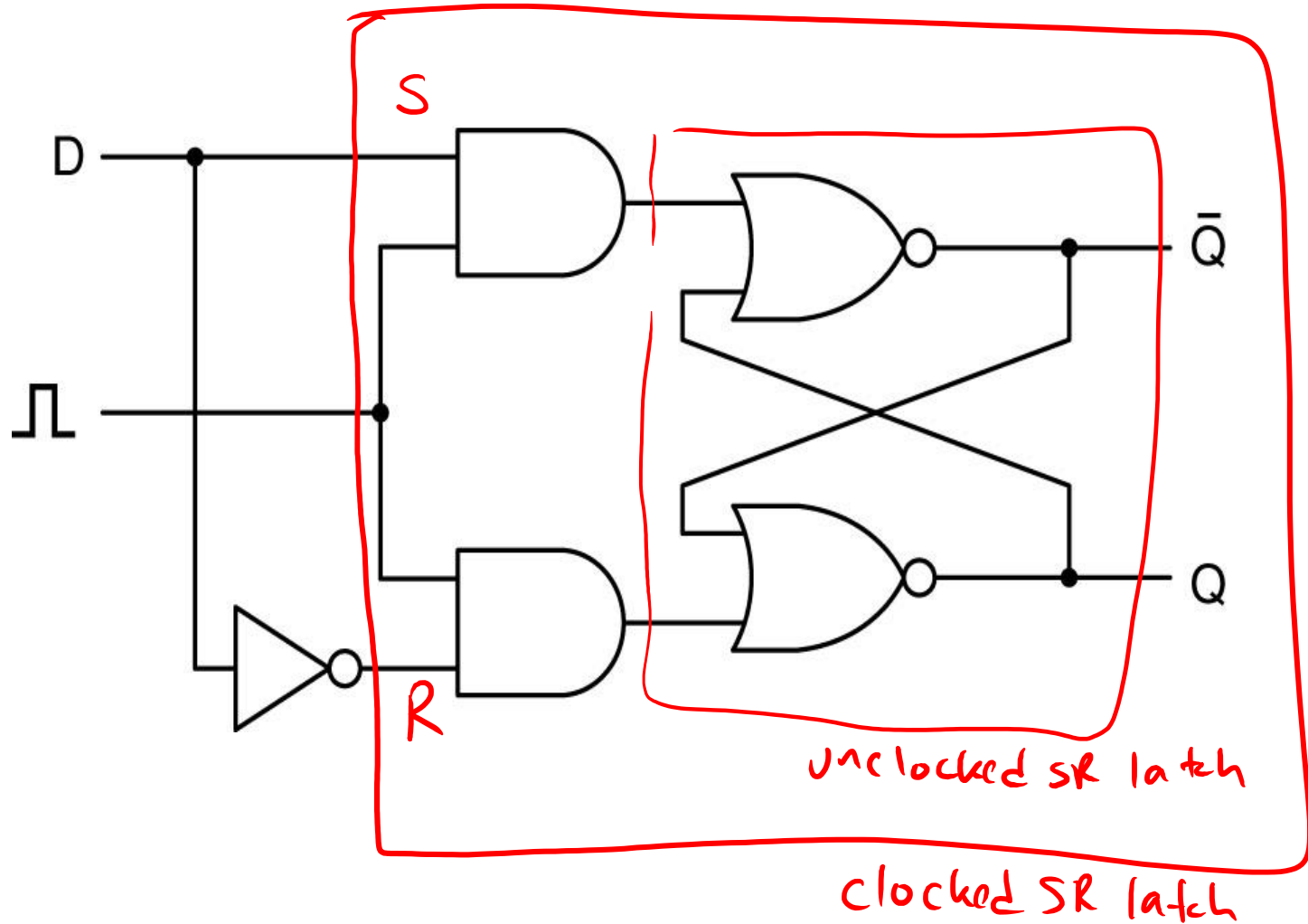
- Adding a clock element as such means the latch will only change state when the clock is high (1). When the clock is low (0), the state is retained.



Clocked D Latch

- A **Clocked D latch** avoids the troublesome combination of when S and R are both 1.
- When the clock is high:
 - The latch is set (to 1) when D is 1.
 - The latch is reset (to 0) when D is 0.
- When the clock is low:
 - The latch state is retained.

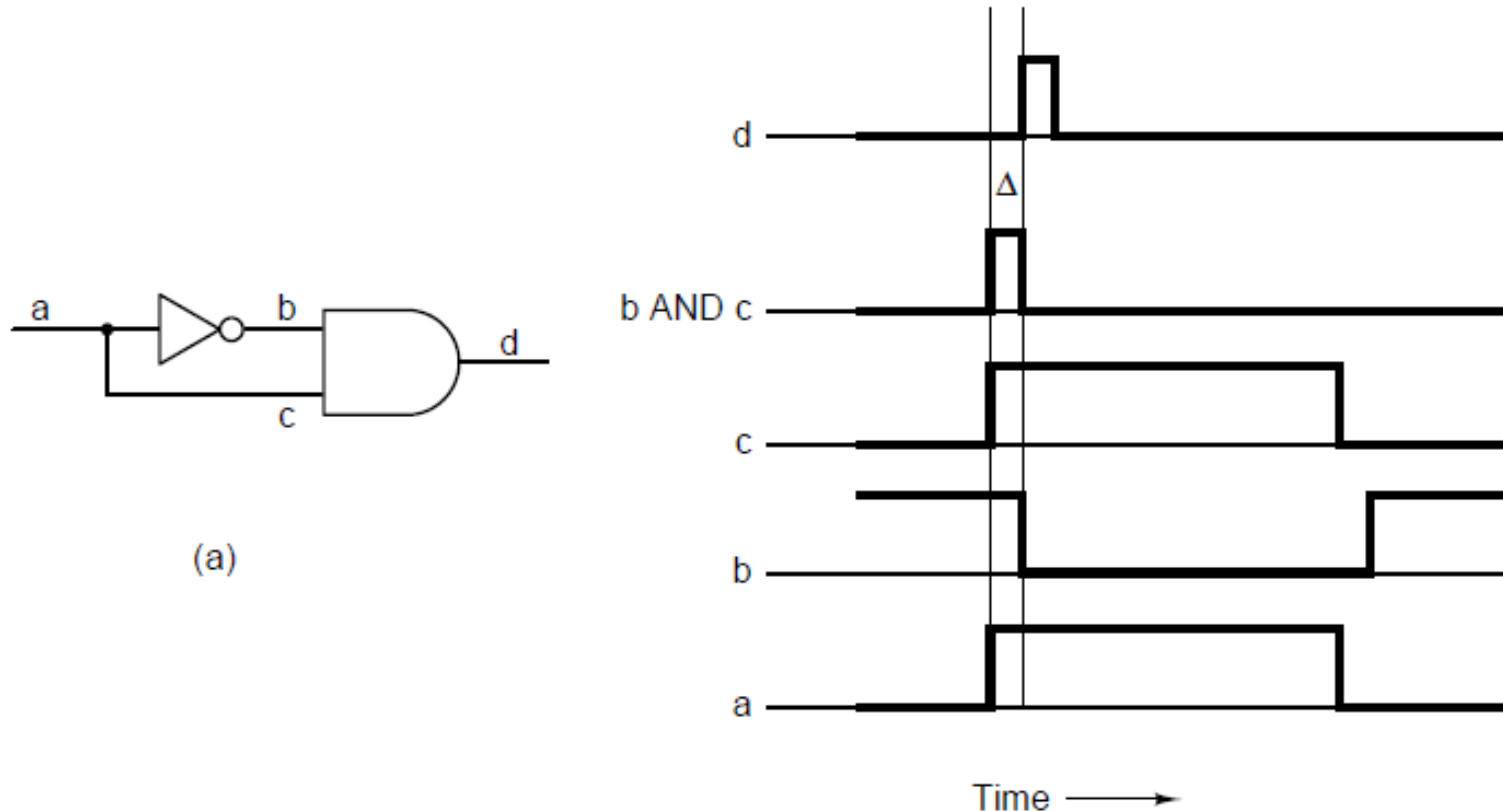
Clocked D Latch



Flip-Flops

- A **flip-flop** is a memory element that is edge triggered.
- The inputs are only scanned on the rising edge of a clock.
- As with a latch, the output is dependent not only on the inputs but the current state of the output.
- Since there is a clock, it can be stated more precisely at: the output at time $t + 1$ (measured in cycles) denoted $Q(t + 1)$ is dependent on the inputs and the output value at time t denoted $Q(t)$.

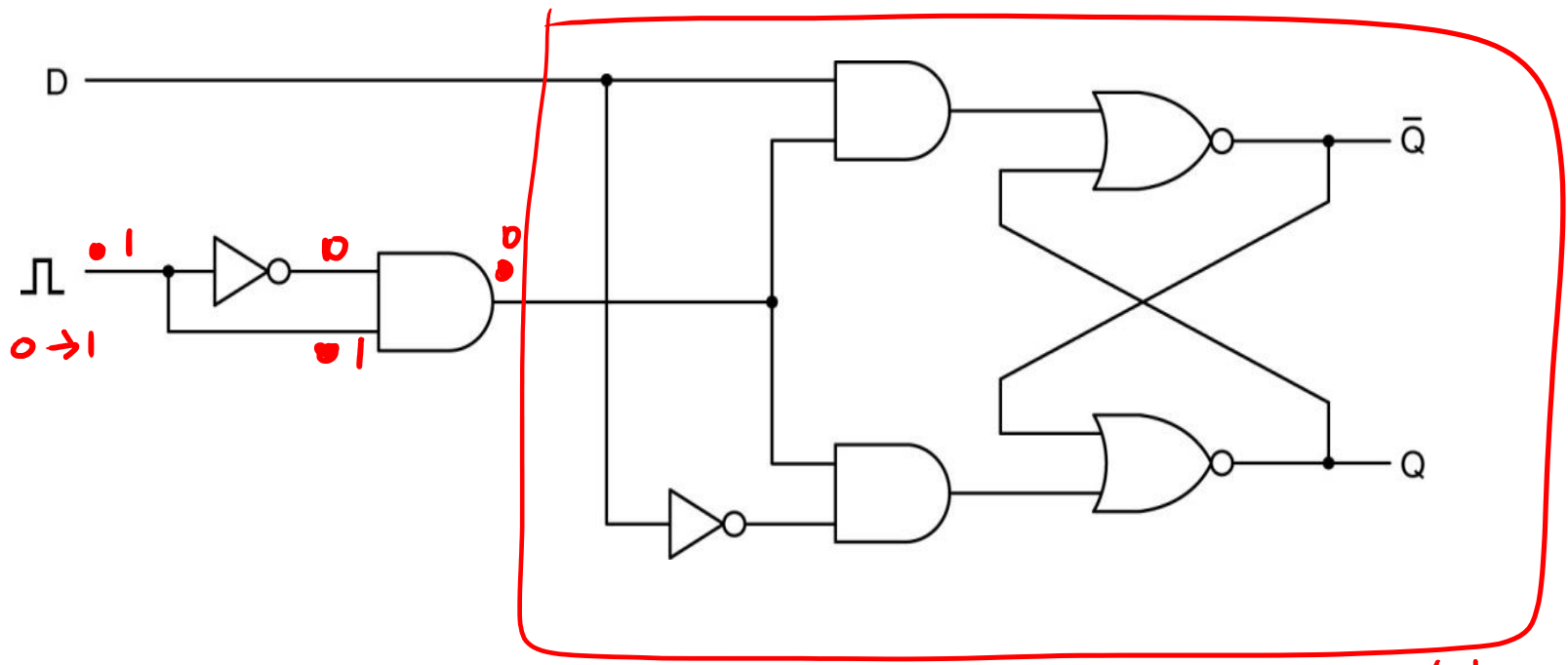
Short-Pulse Generator



(a) A pulse generator. (b) Timing at four points in the circuit.

D Flip-Flop

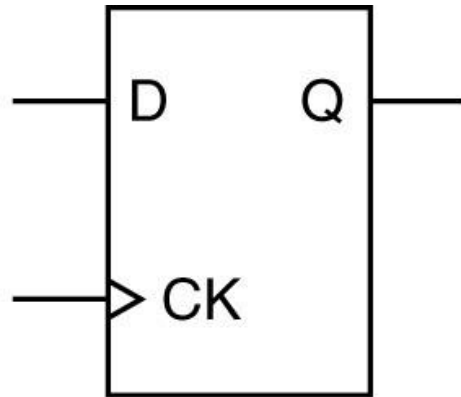
Here is a D flip-flop. The D flip-flop is a true representation of one bit of memory.



clocked D latch

D Flip-Flop Block Diagram

Here is a block diagram that represents a D flip-flop:



Outline

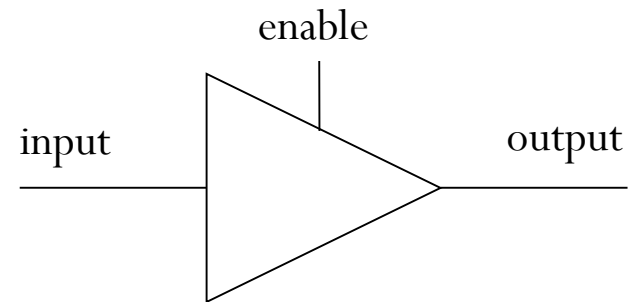
- Gates and Boolean Algebra
- Combinational Logic Circuits
- Sequential Logic Circuits
- **Memory Organization**

Buses

- A **bus** is a set of shared wires that act as a common datapath to connect multiple subsystems within a computer system.
- Only one device may use the bus at the time.
 - Control logic must guarantee this is the case.
 - Often a bottleneck in computer systems.
- Buses are commonly used to transfer data to and from:
 - Memories (both on and off chip)
 - I/O devices

Tri-state Buffers

- A **tri-state buffer** is used to control access to a shared wire or bus.
- Acts like a switch:
 - When enabled – the input is connected directly to output.
 - When disable – disconnects the input (represented with Z).
- Circuit designer must guarantee that at most one tri-state buffer to a shared wire or bus is enabled.



<i>Enable</i>	<i>Input</i>	<i>Output</i>
1	0	0
1	1	1
0	0	Z
0	1	Z

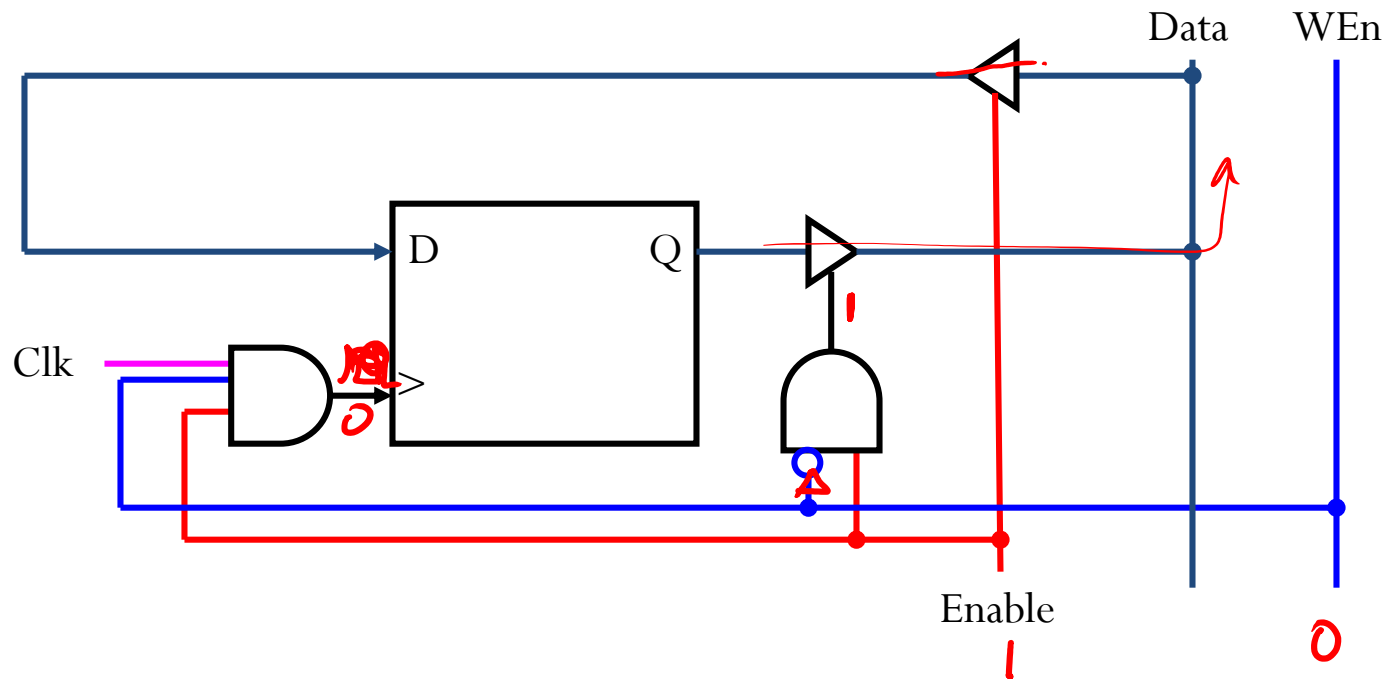
1-bit Memory Cell

A 1-bit memory cell has three lines (plus the clock):

- Enable (input):
 - 0: the memory cell is inactive
 - 1: the memory cell is active
- Write Enable or WEn (input):
 - 0: read data from the memory cell
 - 1: write data to the memory cell
- Data (input/output):
 - If cell is disabled, data is disconnected.
 - If cell is enabled and in read mode, data contains data from memory cell (output).
 - If cell is enabled and in write mode, data contains data to write to memory cell (input).

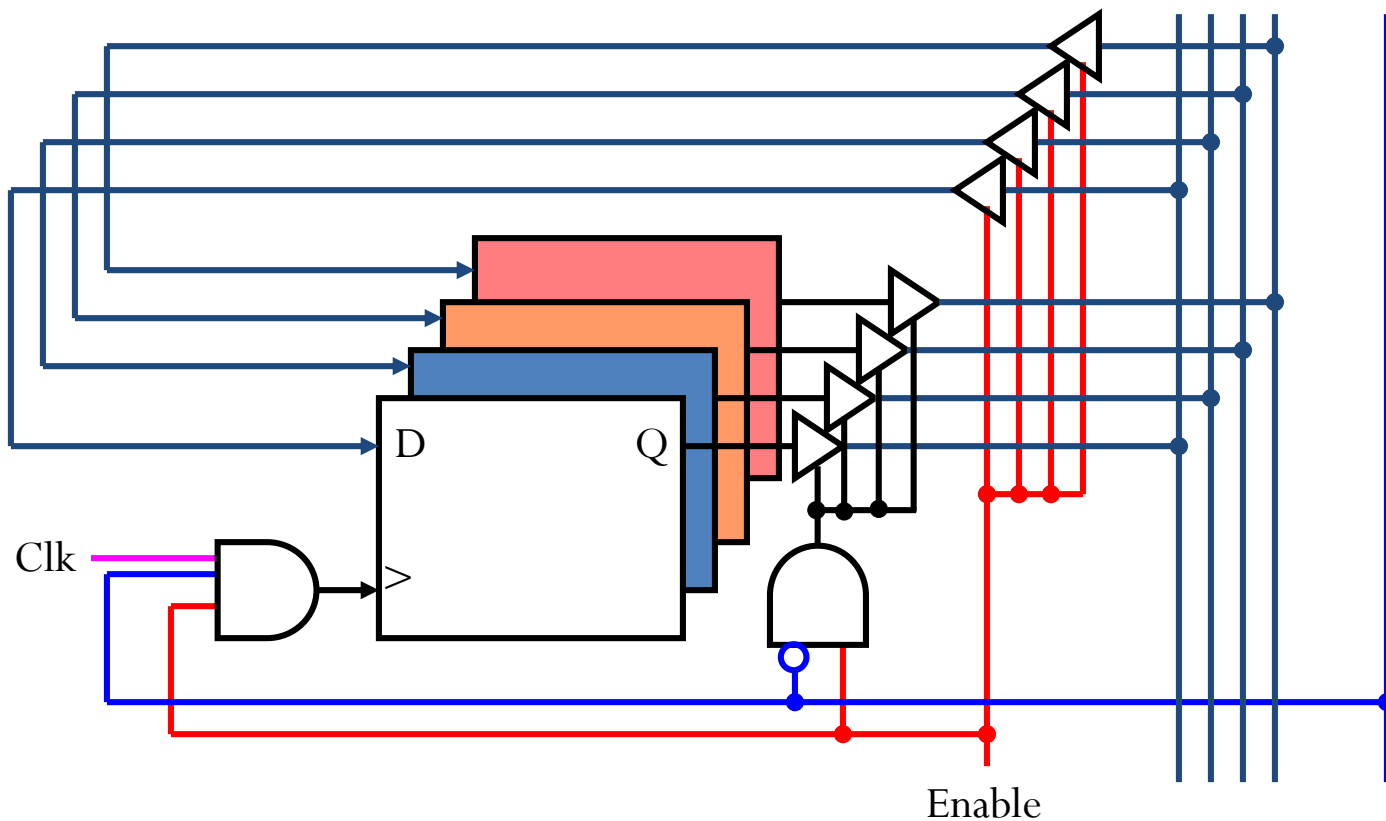
1-bit Memory Cell

Here is a 1-bit memory cell circuit using a D flip-flop:



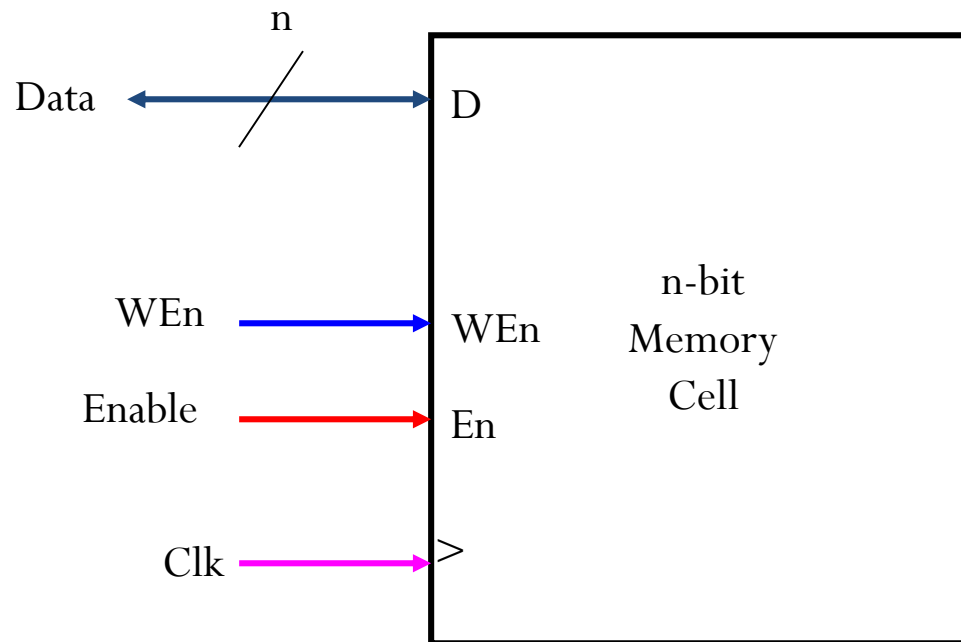
4-bit Memory Cell

Larger memory cells can be constructed with more D flip-flops and a larger data bus. Here is a 4-bit memory cell:



n-bit Memory Cell Block Diagram

Here is a block diagram for an n -bit memory cell:



Memory Addressing

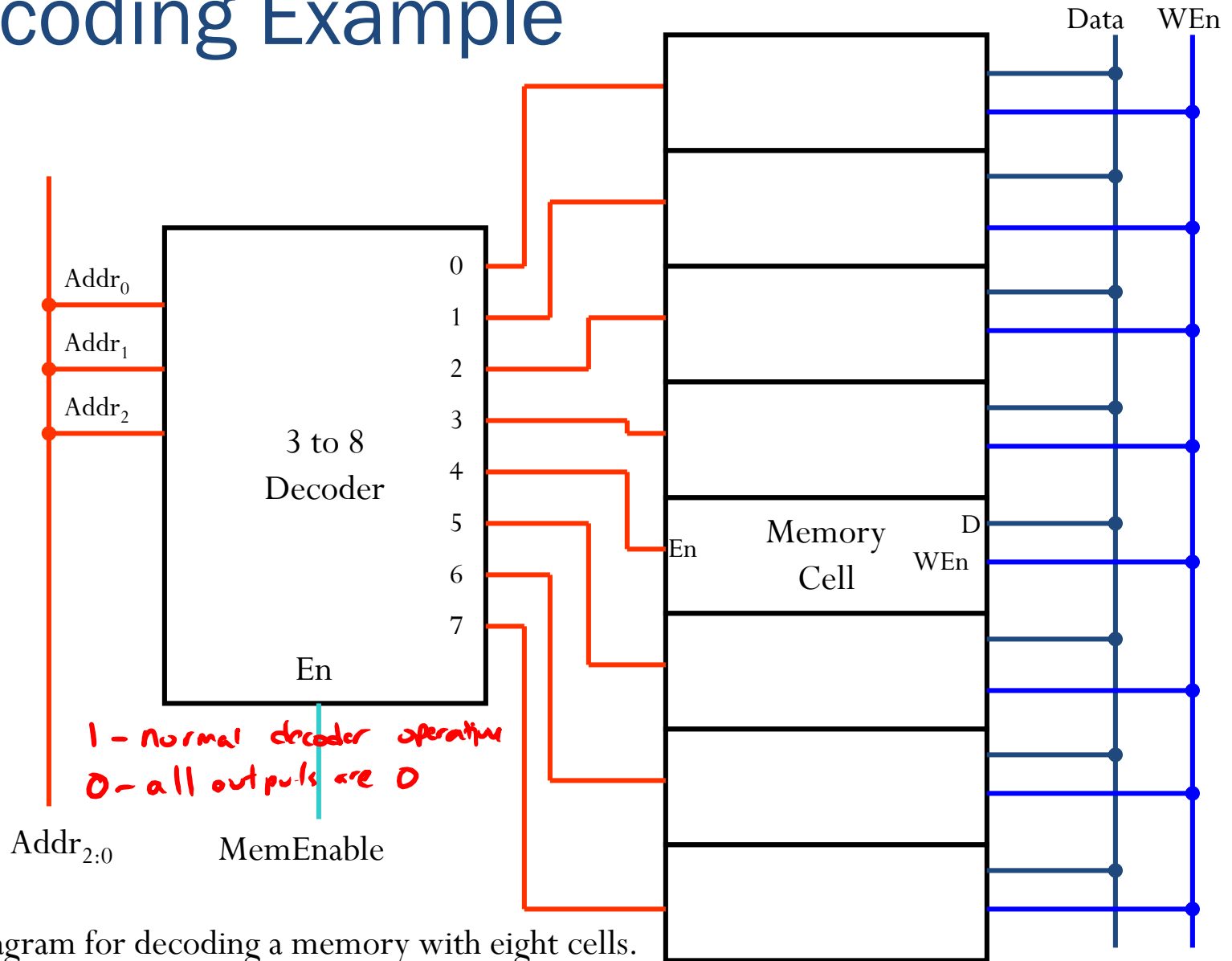
- **Memory** can be thought of as an array of data cells.
- The index into the array is the **address**.
 - Identical to the address stored in a pointer variable (using '&').
- How big are the data cells?
 - *byte-addressable*: each byte has its own address
 - *word-addressable*: each word has its own address
 - A word is often 4 bytes (32 bits) or 8 bytes (64 bits).
- What hardware could be applied for accessing memory locations?

0x0000	
0x0001	
0x0002	
0x0003	
0x0004	
...	
0x7A3E	
0x7A3F	
0x7A40	
0x7A41	
...	
0xFFFFE	
0xFFFF	

Decoding Memory Addresses

- A decoder is used to determine which cell is accessed:
 - Converts an address into the enable lines for the memory cell.
 - Makes sure that only one memory cell is enabled at a time.
- Useful for the decoder to have an enable input that can enable and disable the entire memory.
 - When a memory access occurs, memory enable is set to 1.
 - Decoder behaves normally.
 - When memory is not used, memory enable is set to 0.
 - All outputs of the decoder are 0.
 - No memory cell is enabled.

Decoding Example



Memory Addresses and Sizes

- How many address bits are needed if a memory contains n memory cells?

$$\lceil \log_2 n \rceil$$

$$\log_2 n \text{ (rounded up to next integer)}$$

Powers of Two

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024 = 1 \text{ K (kilo)}$$

$$2^{20} = 1024 \text{ K} = 1 \text{ M (mega)}$$

$$2^{30} = 1024 \text{ M} = 1 \text{ G (giga)}$$

$$2^{40} = 1024 \text{ G} = 1 \text{ T (tera)}$$

$$2^{50} = 1024 \text{ T} = 1 \text{ P (peta)}$$

Class Problem

1. For a byte-addressable memory, how much memory can be accessed using a 36 bit address? Express your answer using metric prefixes.

$$2^{36} \text{ bytes} = 2^6 2^{30} \text{ bytes} = 64 \text{ GB}$$

2. For a byte-addressable memory, how many address bits are needed for 256 MB?

$$\log_2 256 \text{ MB} = \log_2 (2^8 2^{20}) = \log_2 (2^{28}) = 28 \text{ bits}$$

3. Repeat 1 but use a word-addressable memory (word is 64 bits).

$$2^{36} \text{ cells}, \frac{8 \text{ bytes}}{\text{cell}} = 2^{36} \cdot 2^3 \text{ bytes} = 2^{39} \text{ bytes} = 2^9 2^{30} \text{ bytes} \\ = 512 \text{ GB} \qquad 64 \text{ GB} \times 8$$

4. Repeat 2 but use a word-addressable memory (word is 64 bits).

$$\log_2 \left(\frac{256 \text{ MB}}{8 \text{ B}} \right) = \log_2 \left(\frac{2^8 2^{20}}{2^3} \right) = \log_2 2^{25} = 25 \text{ bits}$$

Metric Prefixes

Computer scientists and computer companies abuse metric prefixes.

- $2^{10} = 1,024 = 1 \text{ K}$
- $10^3 = 1,000 = 1 \text{ K}$

What do we use?

- Storage quantities use powers of 2.
- Speed quantities use powers of 10.

Internal CPU Memory

- Inside a CPU, there are two types of memory: registers and cache.
- **Registers** are the fastest form of memory since they are closest to the arithmetic logic unit. The set of registers in a CPU is called the register file.
 - There are very few registers (8, 16, 32) in the CPU.
 - Refer to registers directly using register numbers or names, not by addresses.
- The **cache** inside the CPU stores data from commonly-used addresses.
 - Saves time as it avoids accessing memory outside the chip which is slow.
 - There may be additional caches outside the chip.

Random Access Memory (RAM)

- Main memory is stored in **RAM (Random Access Memory)**.
- *Static RAM*: Implemented using a circuit similar to the D flip-flop circuit.
 - Uses 6 transistors
 - Very fast
- *Dynamic RAM*: Implemented using a transistor and a capacitor.
 - Capacitors must be refreshed periodically
 - Very high density
- RAM is *volatile* — memory cells retain their values as long as the power is on.
 - However, if the power goes off — the values disappear.
 - Registers and caches are also volatile.

Read Only Memory (ROM)

- A **ROM (Read Only Memory)** is a memory where the contents of the memory are hard coded when it is manufactured.
- It is commonly used in "closed" computer systems in appliances, cars, and toys.
- In a traditional computer, the ROM is used to execute code to help boot the computer.
- ROM is *nonvolatile* — its contents remain intact even if the power is turned off.

Programmable ROMs / Flash Memory

The inflexibility of ROMs have given way to "programmable" ROMs or read/write nonvolatile memory:

- PROM (programmable ROM): Can be programmed once.
- EPROM (erasable PROM): Can be field programmed and field erased.
- EEPROM (electrically-erasable PROM): Can be reprogrammed in place without a special device.
- Flash Memory: A form of EEPROM that is block erasable and rewritable.

Solid State Disks

- SSDs use flash storage for random access; no moving parts.
 - Access blocks directly using block number
- Very fast reads
- Writes are slower - need a slow erase cycle (can not overwrite directly)
 - Limit on number of writes per block (over lifetime)
- Do not overwrite; garbage collect later
- Flash reads and writes faster than traditional disks
- Used in high-end I/O applications
 - Also in use for laptops, tablets

Tale of the tape: Flash vs. RAM

Speed:

Flash

RAM

Price:

Flash

RAM

Nonvolatile:

Flash

RAM

Tale of the tape: Flash vs. Hard Disks

Speed:	Flash	Disk
--------	-------	------

Price:	Flash	Disk
--------	-------	------

Capacity:	Flash	Disk
-----------	-------	------

Durability:	Flash	Disk
-------------	-------	------

Tale of the tape: Flash vs. Hard Disks

Speed: **Flash** Disk

Price: Flash **Disk**

Capacity: Flash **Disk**

Durability: **Flash** Disk

- What about longevity?

Thank You!