Memory Elements and Units

COE 233

Digital Logic and Computer Organization

Dr. Muhamed Mudawar

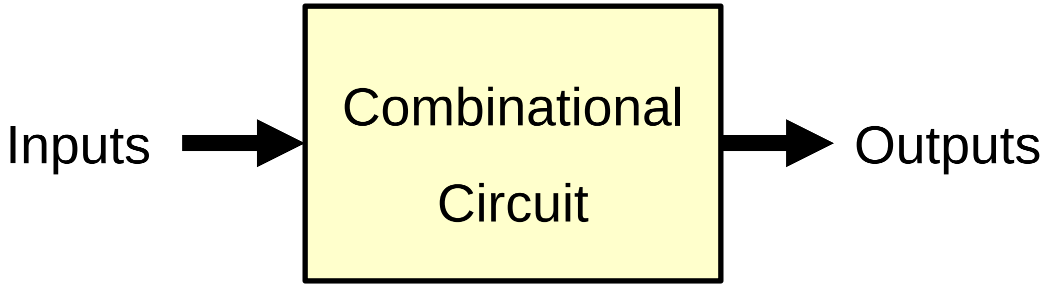
King Fahd University of Petroleum and Minerals

Presentation Outline

- ❖ Introduction to Sequential Circuits
- ❖ Latches
- ❖ Flip-Flops and Registers
- ❖ Memory Units

Combinational versus Sequential

- Two classes of digital circuits
  - Combinational Circuits
  - Sequential Circuits
- Combinational Circuit
  - Outputs = F(Inputs)
  - Function of Inputs only
  - NO internal memory

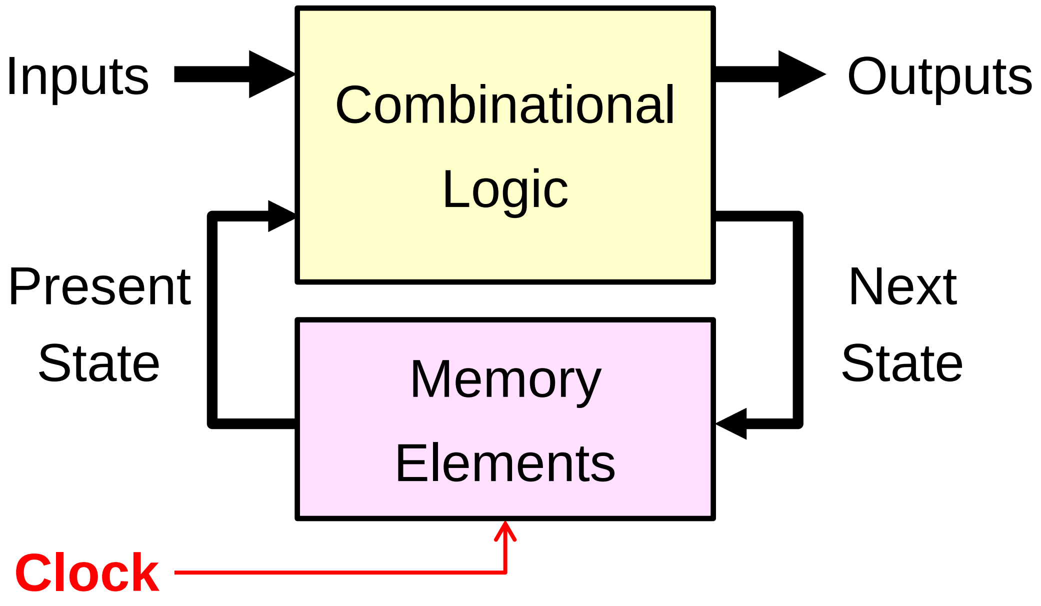


- Sequential Circuit
  - Outputs is a function of Inputs and internal Memory
  - There is an internal memory that stores the state of the circuit
  - Time is very important: memory changes with time

Sequential Circuits

A Sequential circuit consists of:

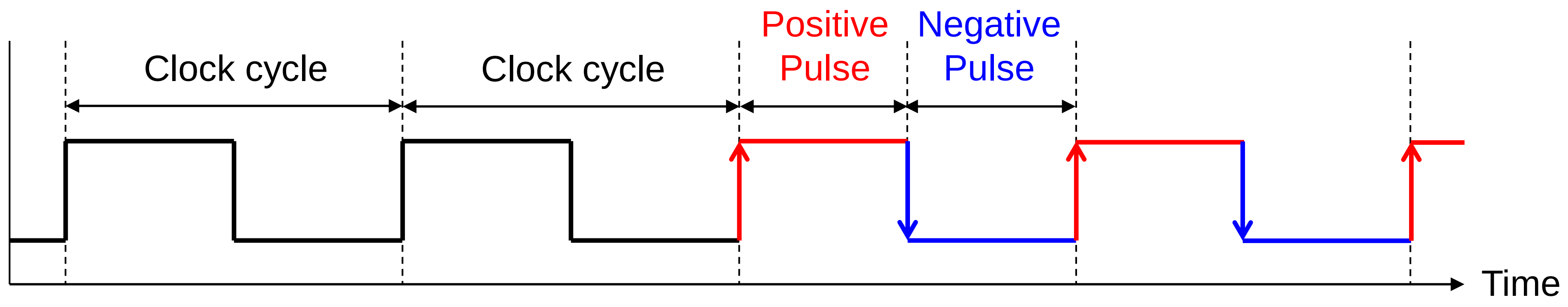
1. Memory elements:
   - ✧ **Latches** or **Flip-Flops**
   - ✧ Store the **Present State**
2. Combinational Logic
   - ✧ Produces the **Outputs**
   - ✧ Produces the **Next State**
   - ✧ **Next state** = memory input (not stored yet)



❖ **Synchronous** sequential circuits use a **clock signal**

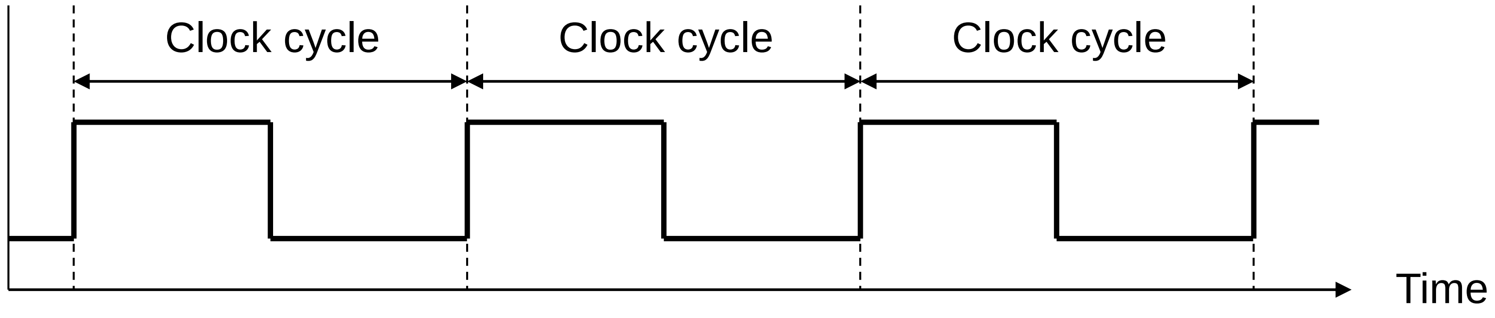
- ✧ The clock signal is an input to the memory elements
- ✧ The clock determines **when** the memory should be updated

The Clock



- Clock is a periodic signal = Train of pulses (1's and 0's)
- The same clock cycle repeats indefinitely over time
- **Positive Pulse**: when the **level** of the clock is **1**
- **Negative Pulse**: when the **level** of the clock is **0**
- **Rising Edge**: when the clock goes **from 0 to 1**
- **Falling Edge**: when the clock goes **from 1 down to 0**

Clock Cycle versus Clock Frequency



- Clock cycle (or period) is a time duration
  - Measured in seconds, milli-, micro-, nano-, or pico-seconds
  - 1 ms = 10^{-3} sec, 1 µs = 10^{-6} sec, 1 ns = 10^{-9} sec, 1 ps = 10^{-12} sec
- Clock frequency = number of cycles per second (Hertz)
  - 1 Hz = 1 cycle/sec, 1 KHz = 10^3 Hz, 1 MHz = 10^6 Hz, 1 GHz = 10^9 Hz
- Clock frequency = 1 / Clock Cycle
  - Example: Given the clock cycle = 0.5 ns = 0.5×10^{-9} sec
  - Then, the clock frequency = $1/(0.5\times10^{-9}) = 2\times10^9$ Hz = 2 GHz

Memory Elements

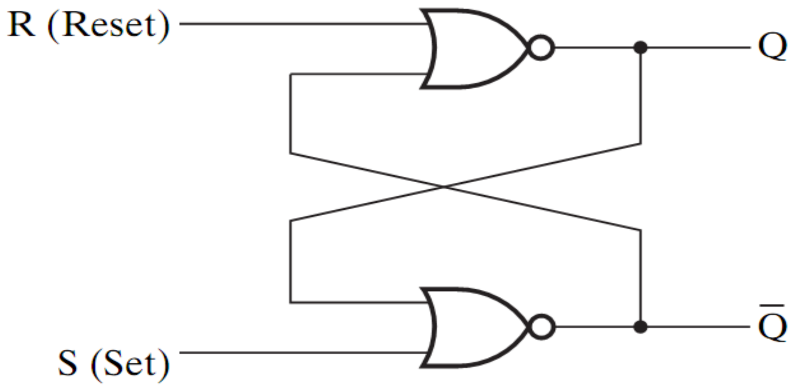
- ❖ Memory can store and maintain binary state (0's or 1's)
  - ✧ Until directed by an input signal to change state
- ❖ Main difference between memory elements
  - ✧ Number of inputs they have
  - ✧ How the inputs affect the binary state
- ❖ Two main types:
  - ✧ Latches are level-sensitive (the level of the clock)
  - ✧ Flip-Flops are edge-sensitive (sensitive to the edge of the clock)
- ❖ Flip-Flips are used in synchronous sequential circuits
- ❖ Flip-Flops are built with latches

Next . . .

- ❖ Introduction to Sequential Circuits
- ❖ Latches
- ❖ Flip-Flops and Registers
- ❖ Memory Units

SR Latch

- A **latch** is a memory element that can store 0 or 1
- An **SR Latch** can be built using two **cross-coupled** NOR gates
- Two inputs: S (Set) and R (Reset)
- Two outputs: Q and $\overline{Q}$



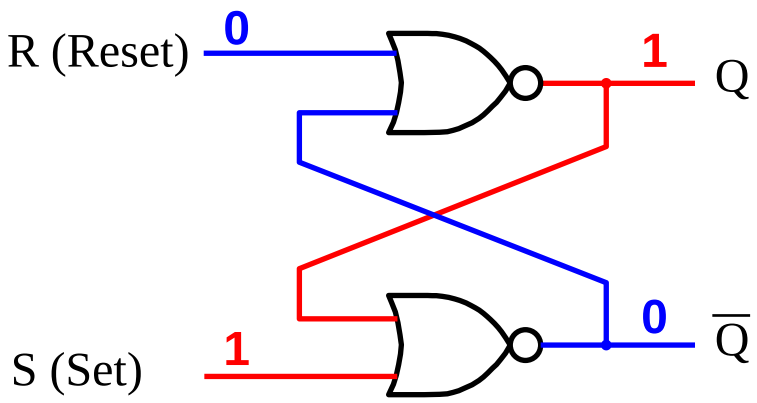
(a) Logic diagram

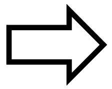
S	R	Q	$\overline{Q}$	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

(b) Function table

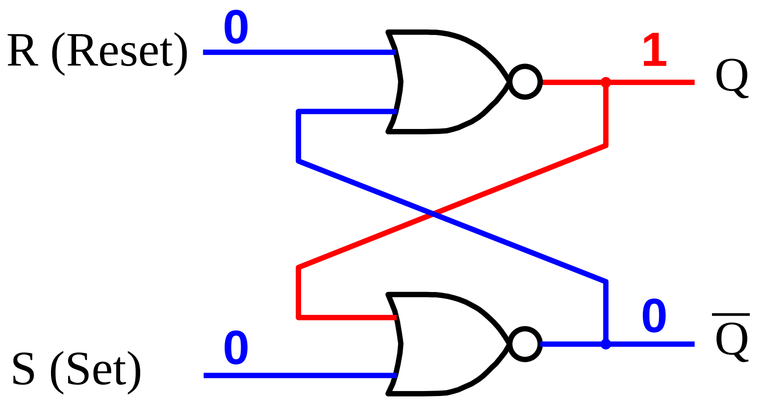
SR Latch Operation

Set Operation

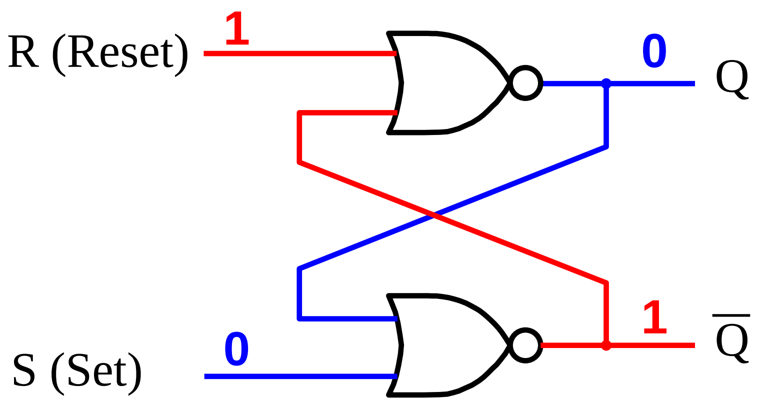


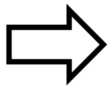


Store Operation

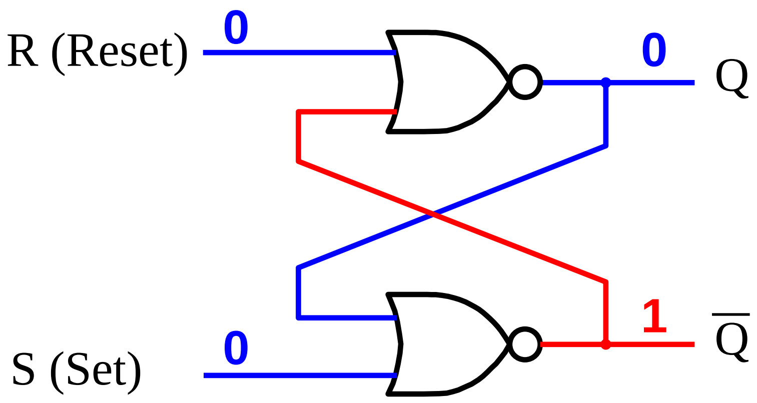


Reset Operation

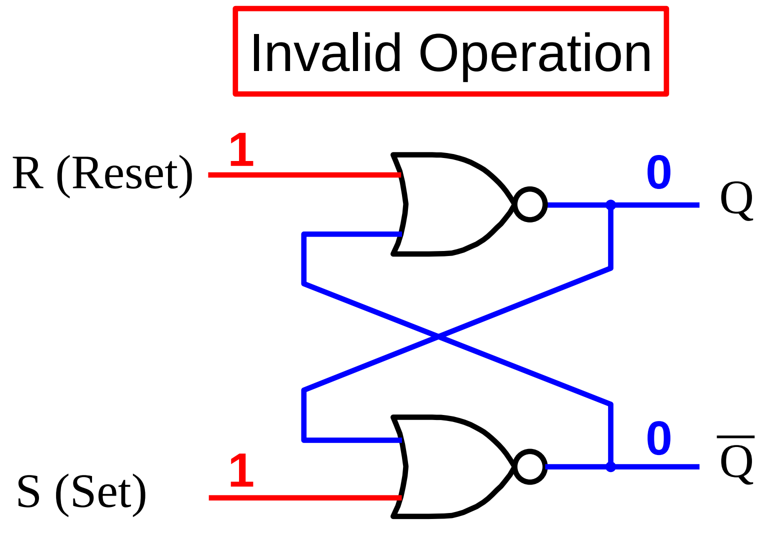


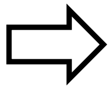


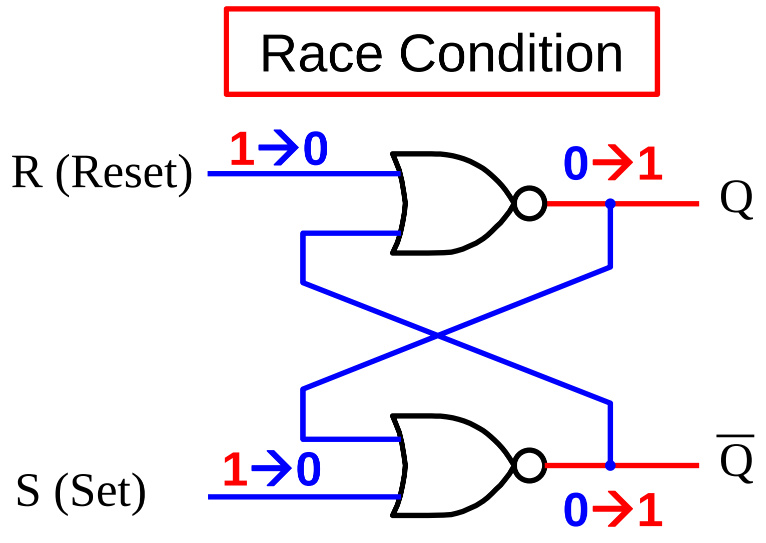
Store Operation



SR Latch Invalid Operation



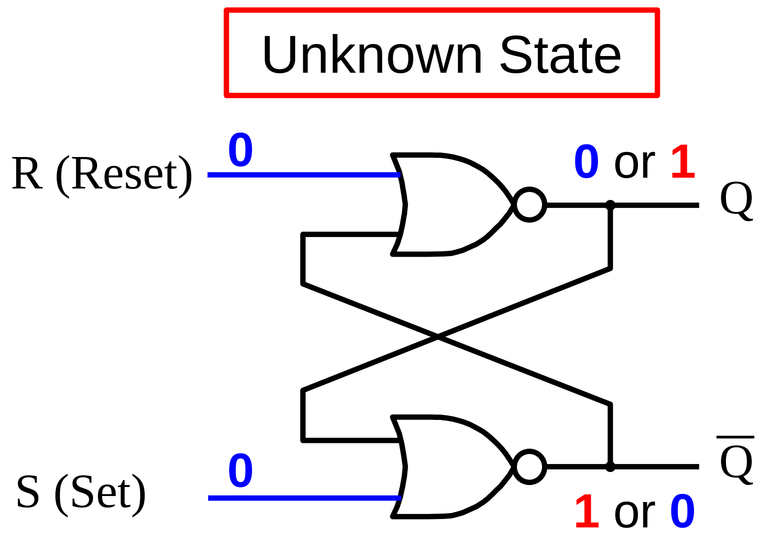




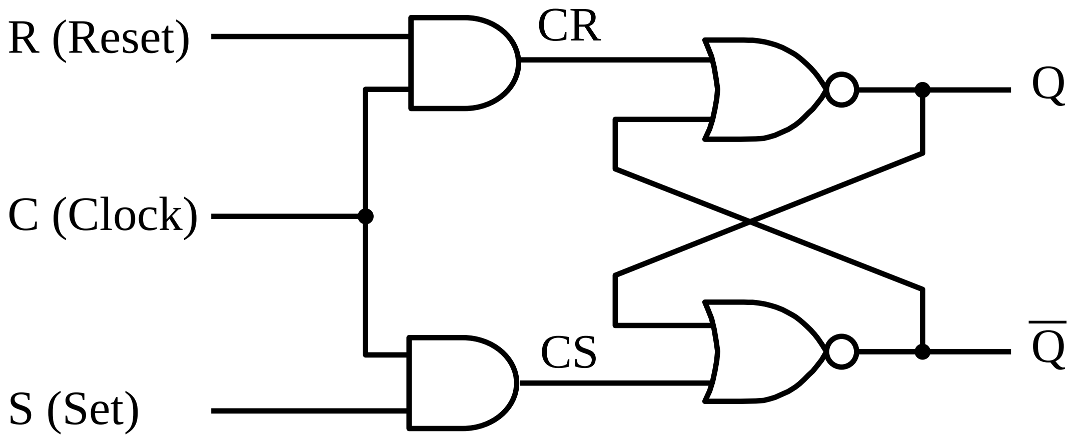
S = R = 1 should never be used

If S and R change from 1 → 0 simultaneously then race condition (oscillation) occurs

Final Q and $\overline{Q}$ are unknown



SR Latch with Clock Input

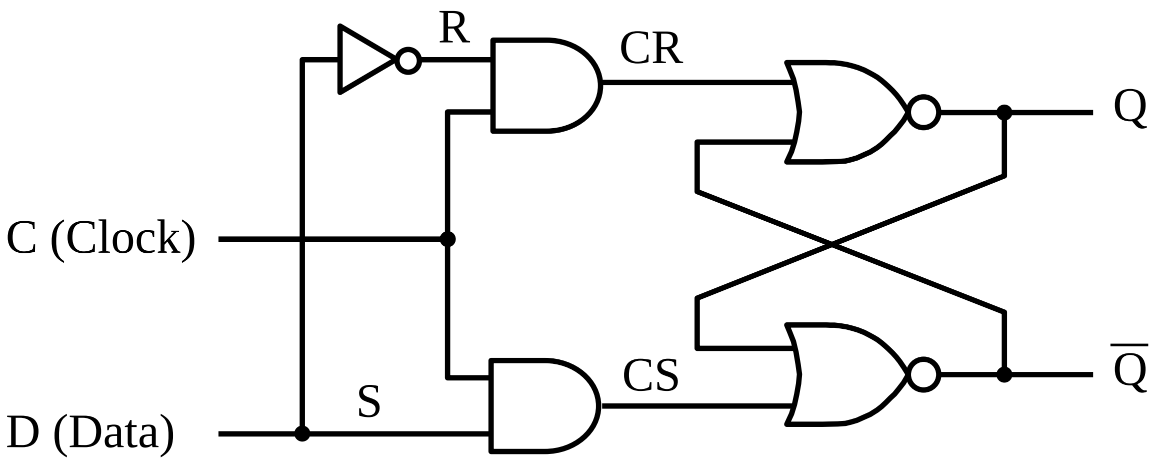


C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

- An additional Clock (enable) input signal **C** is used
- Clock controls **when** the state of the latch can be changed
- When **C=0**, the S and R inputs have no effect on the latch

  The latch will remain in the same state, regardless of S and R
- When **C=1**, then normal SR latch operation

D Latch with Clock Input



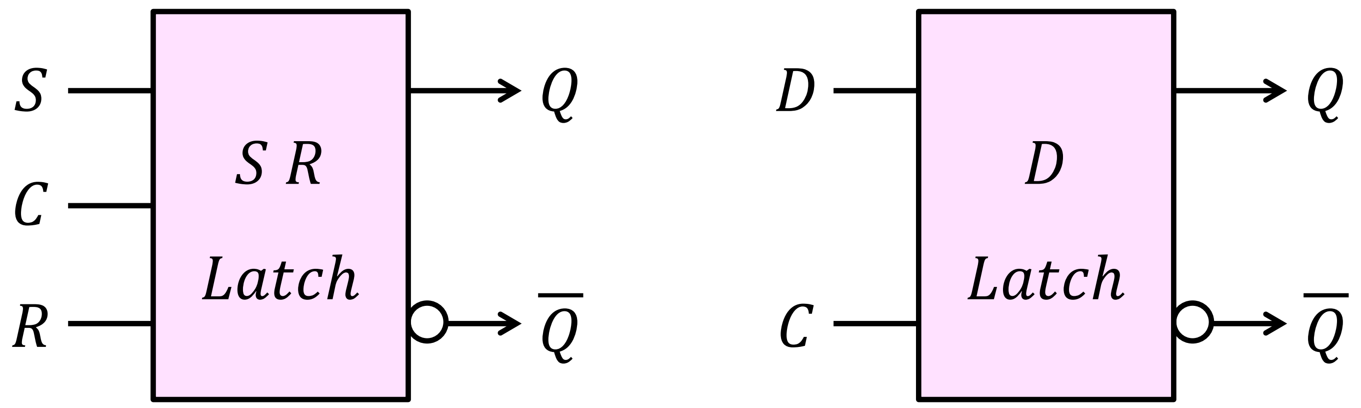
C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

(b) Function table

- ❖ One data input D, $S = D$ and $R = \overline{D}$, No undefined state
- ❖ Clock controls **when** the state of the latch can be changed
- ❖ When **C=0**, the *D* input has no effect on the latch

  The latch will remain in the same state, regardless of *D*
- ❖ When **C=1**, latch is enabled, reset if *D* is 0 and set if *D* is 1

Graphic Symbols for Latches



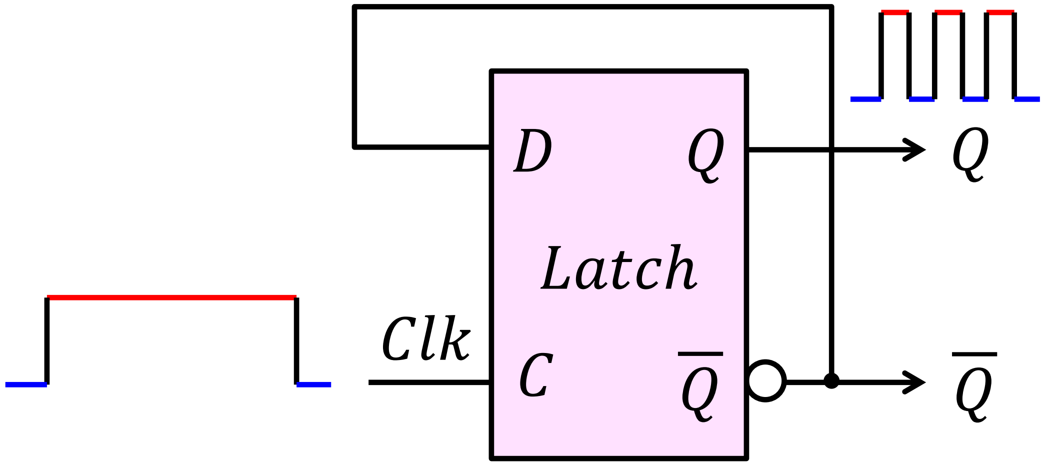
- ❖ A bubble appears at the complemented output $\overline{Q}$
- ❖ Indicates that $\overline{Q}$ is the complement of Q

Problem with Latches

- A latch is **level-sensitive** (sensitive to the level of the clock)
- As long as the clock signal is **high** …

  Any change in the value of input D appears in the output Q
- Output Q keeps changing its value during a clock cycle
- Final value of output Q is uncertain

Due to this uncertainty, latches are NOT used as memory elements in sequential circuits

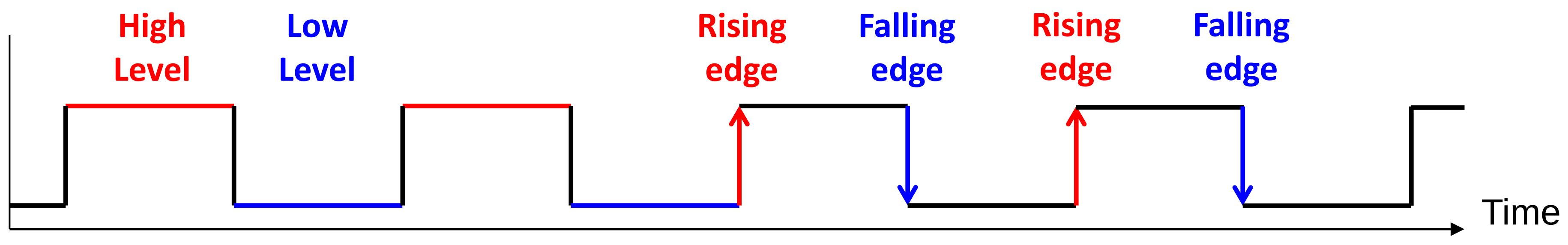


Next . . .

- Introduction to Sequential Circuits
- Latches
- Flip-Flops and Registers
- Memory Units

Flip-Flops

- A **Flip-Flop** is a better memory element for sequential circuits
- Solves the problem of latches in synchronous sequential circuits
- A **latch** is sensitive to the **level** of the clock
- However, a **flip-flop** is sensitive to the **edge** of the clock
- A flip-flop is called an **edge-triggered** memory element
- It changes it output value at the **edge** of the clock

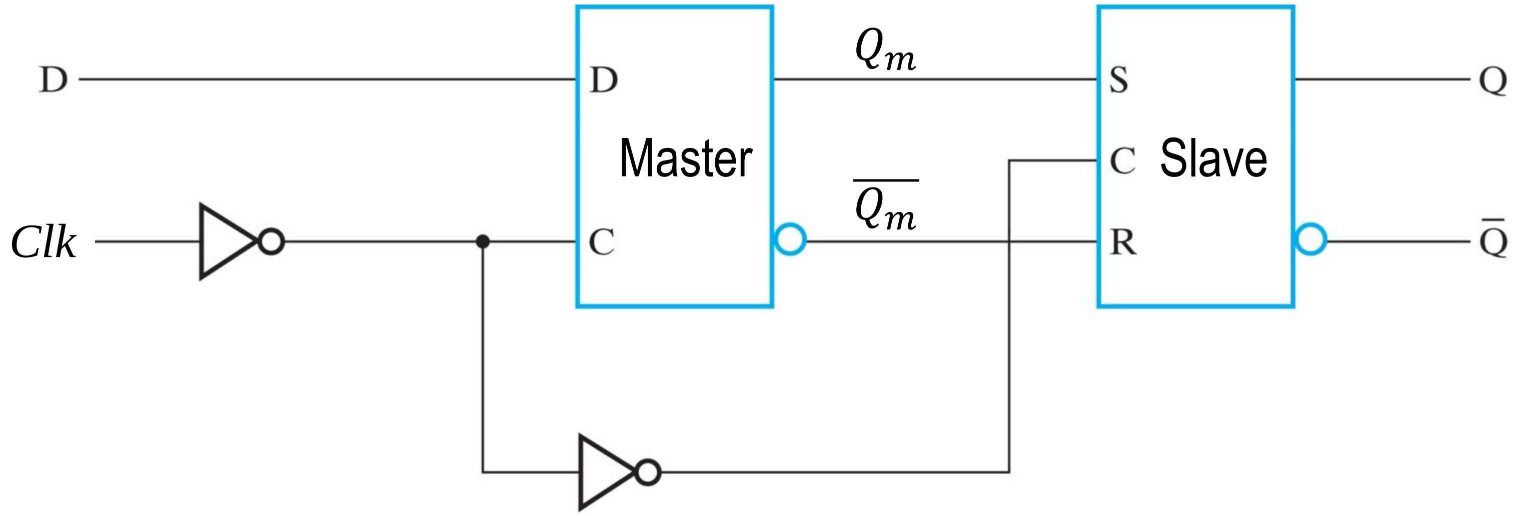


Edge-Triggered D Flip-Flop

- Built using two latches in a **master-slave** configuration
- A master latch (D-type) receives external inputs
- A slave latch (SR-type) receives inputs from the master latch
- Only one latch is enabled at any given time

  When **Clk=0**, the master is enabled and the D input is latched (slave disabled)

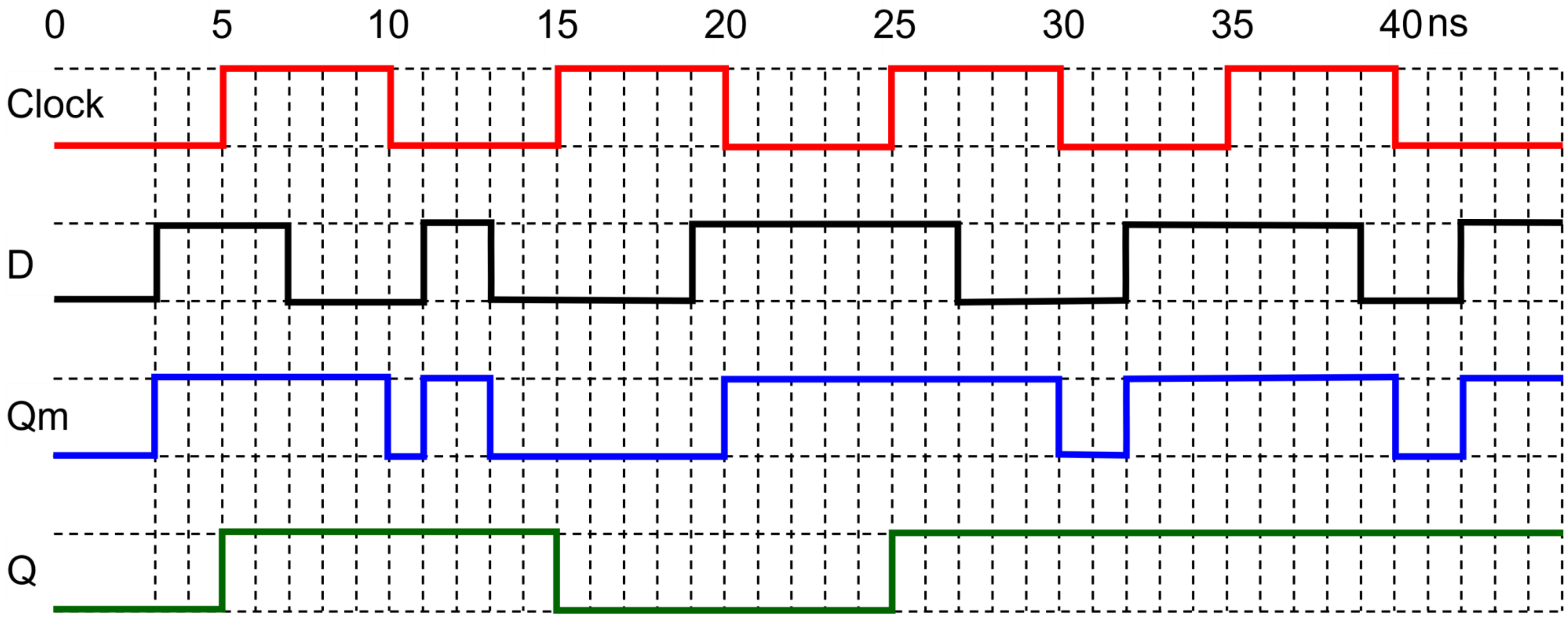
  When **Clk=1**, the slave is enabled to generate the outputs (master is disabled)



Outputs change when *Clk* changes **from 0 to 1**

D Flip-Flop Timing Diagram

- The diagram shows the timing of a positive-edge D Flip-Flop
- The master latch changes its output *Qm* when the clock *C* is 0
- The rising edge of the clock triggers the D Flip-Flop

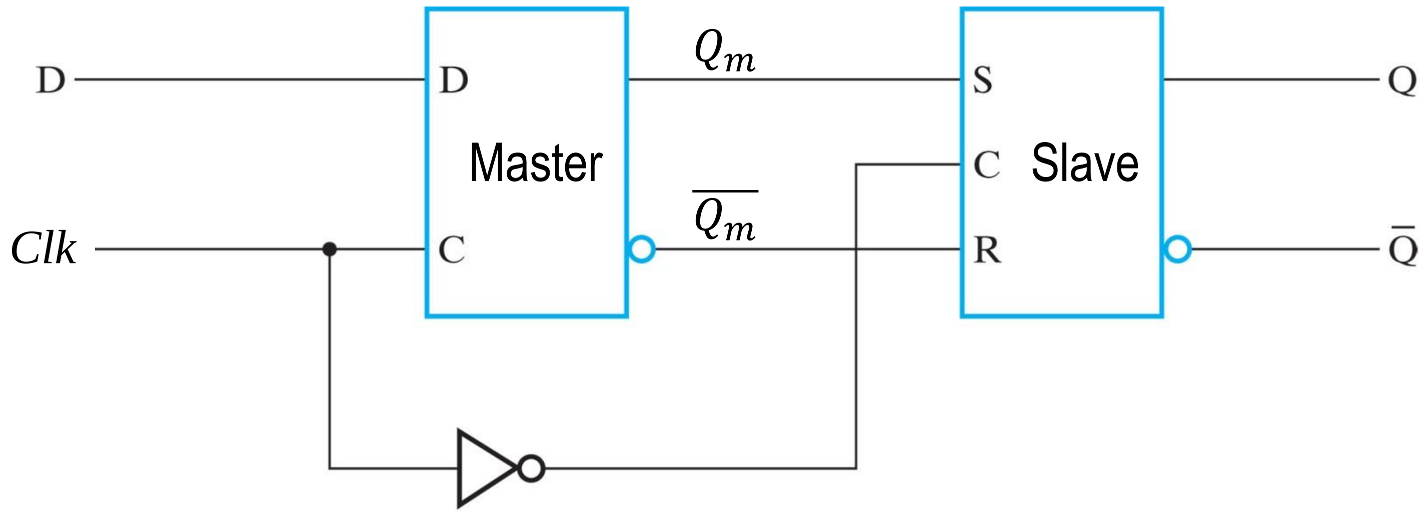


Negative Edge-Triggered D Flip-Flop

- Similar to positive edge-triggered flip-flop
- The first inverter at the Master C input is removed
- Only one latch is enabled at any given time

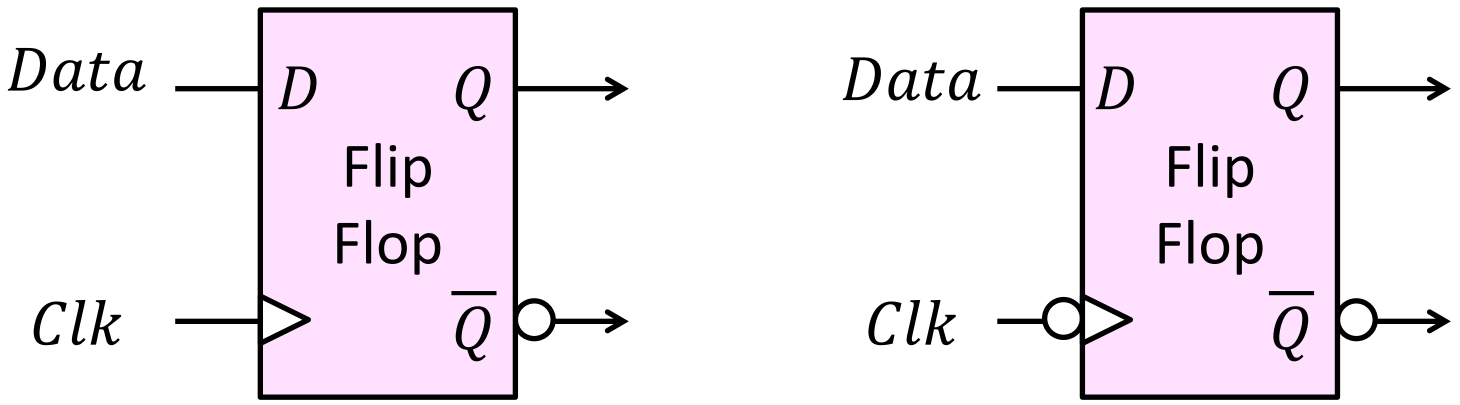
  When **Clk=1**, the master is enabled and the D input is latched (slave disabled)

  When **Clk=0**, the slave is enabled to generate the outputs (master is disabled)



Outputs change when *Clk* changes **from 1 to 0**

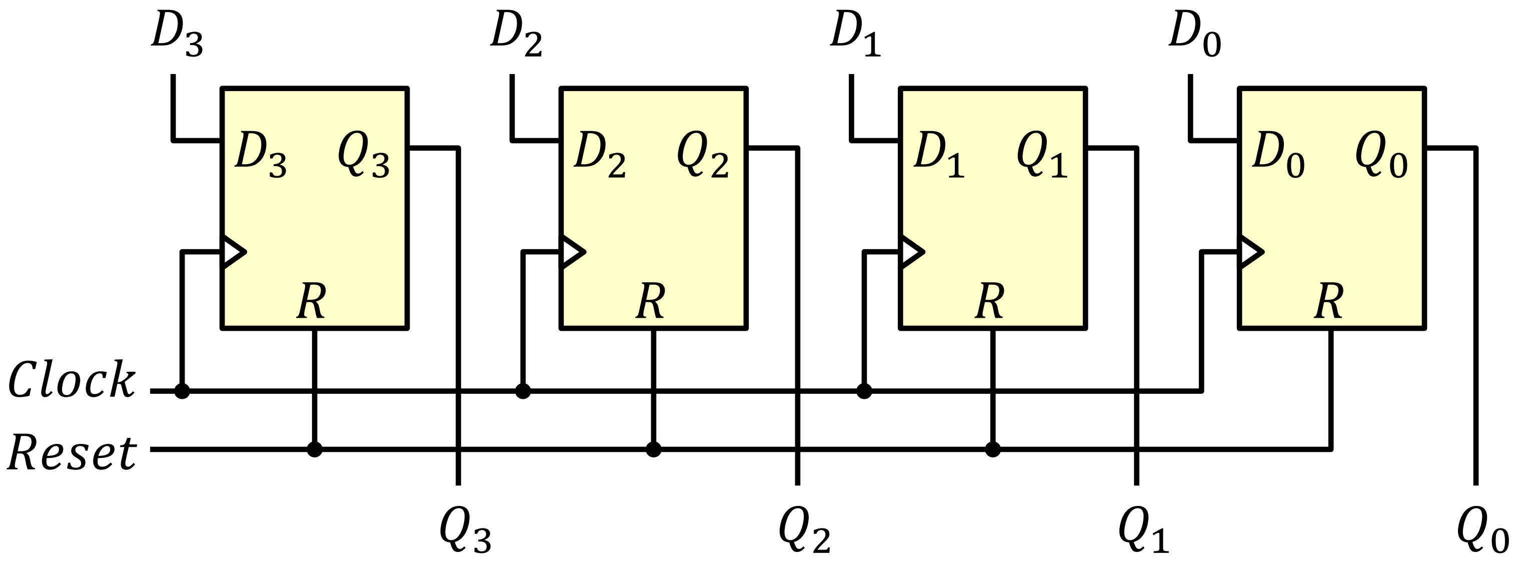
Graphic Symbols for Flip-Flops



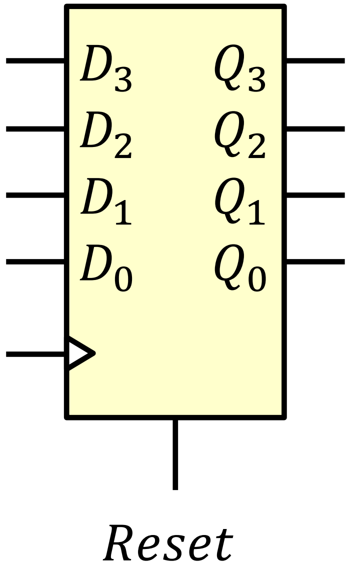
- ❖ A Flip-Flop has a similar symbol to a Latch
- ❖ The difference is the arrowhead at the clock input
- ❖ The arrowhead indicates sensitivity to the edge of the clock
- ❖ A circle at the *Clk* input indicates negative edge-triggered FF

Register

- A register is a circuit capable of storing data
- An n-bit register consists of n Flip-Flops and stores n bits of data
- Common clock: data is loaded in parallel at the same clock edge
- Common reset: All Flip-Flops are reset in parallel



4-bit Register

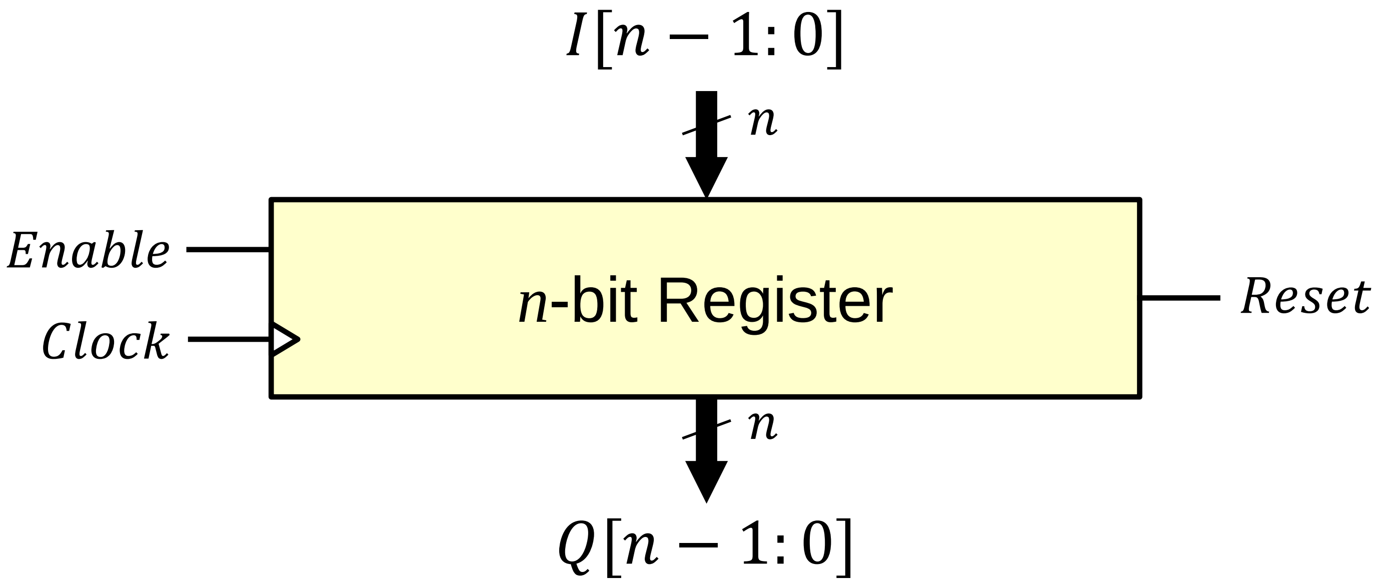


Register Enable

- **Question:** How to control the loading of data into a register?
- **Solution:** Introduce an **Enable** control signal

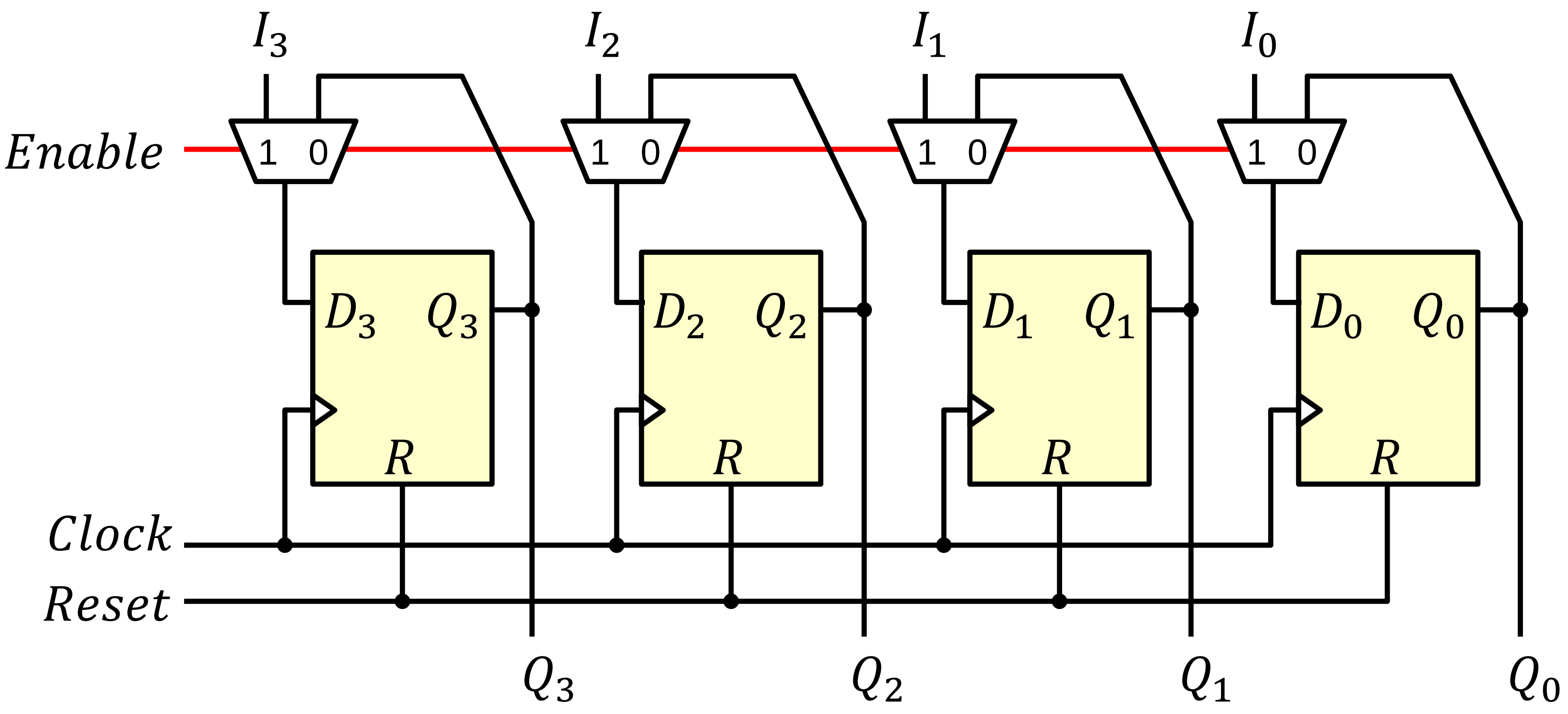
  If the register is enabled, load the data into the register

  Otherwise, do not change the value of the register
- **Question:** How to implement register enable?



Implementing Register Enable

- **Solution:** Add a mux (multiplexer) at the D input of the register
- $D_i = Enable \cdot I_i + \overline{Enable} \cdot Q_i$
- If $Enable$ is **1** then $D_i = I_i$ If $Enable$ is **0** then $D_i = Q_i$

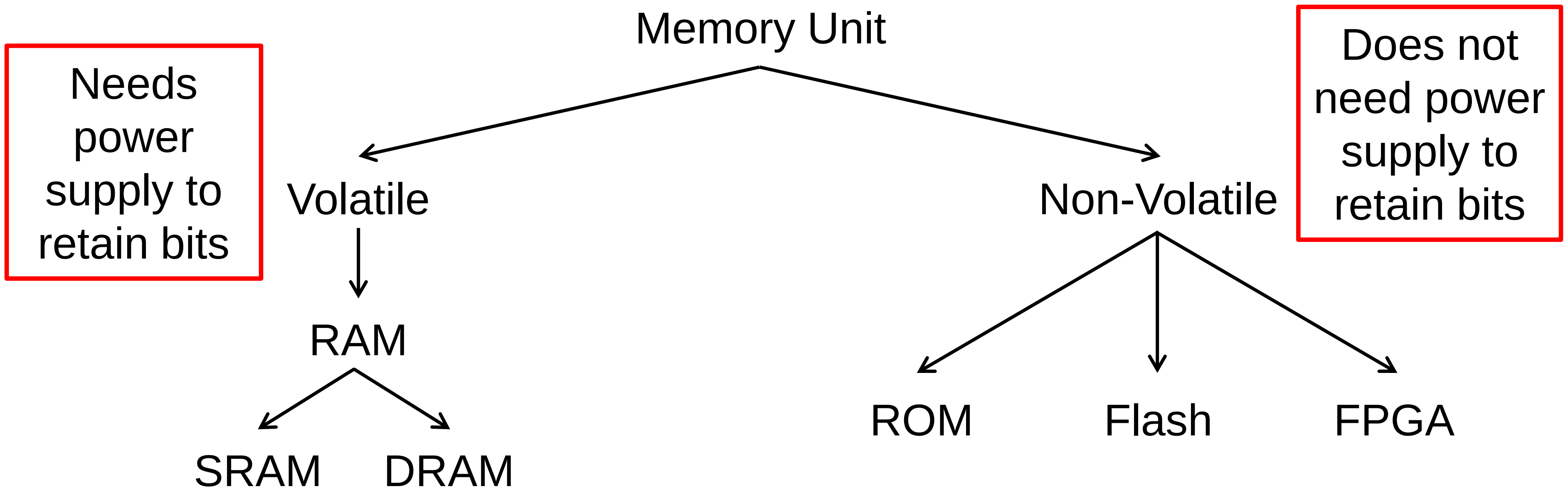


Next . . .

- ❖ Introduction to Sequential Circuits
- ❖ Latches
- ❖ Flip-Flops and Registers
- ❖ Memory Units

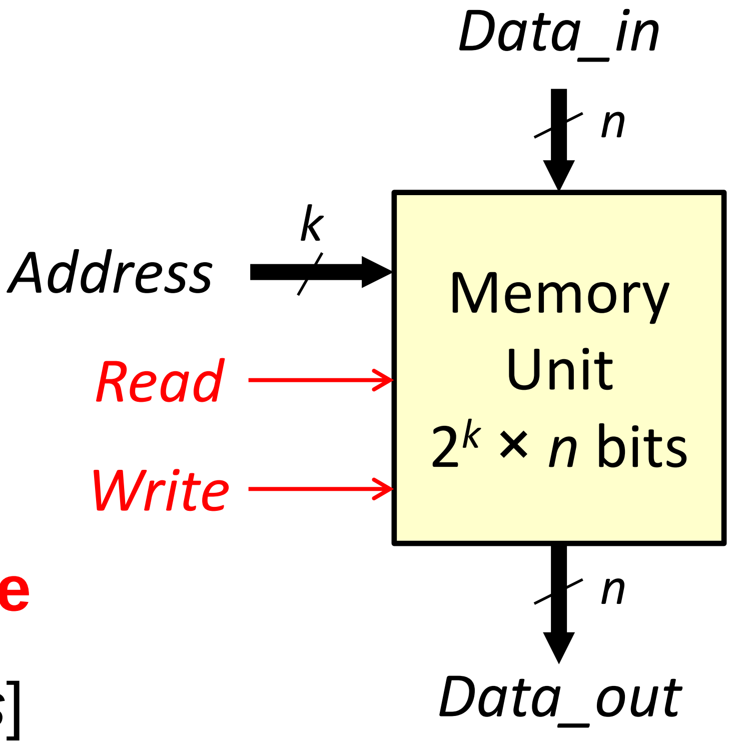
Memory Units

- ❖ A memory unit is a device that can store binary information
- ❖ It is a large group of cells, each capable of storing one bit
- ❖ Two types of memory units: **Volatile** and **Non-Volatile**
- ❖ Random-Access Memory (RAM) is volatile
- ❖ ROM, Flash, and FPGA are non-volatile


Memory Unit
Needs power supply to retain bits
Volatile
RAM
SRAM
DRAM
Non-Volatile
Does not need power supply to retain bits
ROM
Flash
FPGA

Random-Access Memory

- Large array of storage cells, capable of storing many 0's and 1's
- Random Access: bits can be accessed randomly
- Memory is addressable
  - Memory address consists of *k* bits
  - Can address 2^k words in memory
  - Each word consists of *n* bits
- Memory capacity = $2^k \times n$ bits
- Two control functions: **Read** and **Write**
  - **Read:** *Data_out* ← **Memory** [*Address*]
  - **Write:** **Memory** [*Address*] ← *Data_in*



Memory Capacity

- Each memory location is a group of *n* bits, which is read/written

  Byte = 8-bit data

  Half-word = 16-bit data = 2 bytes

  Word = 32-bit data = 4 bytes

- The memory capacity is specified in bytes

  1 KB (Kilo Byte) = 2^{10} = 1024 bytes (more than thousand = 10^3)

  1 MB (Mega Byte) = 2^{20} Bytes (more than million = 10^6)

  1 GB (Giga Byte) = 2^{30} Bytes (more than billion = 10^9)

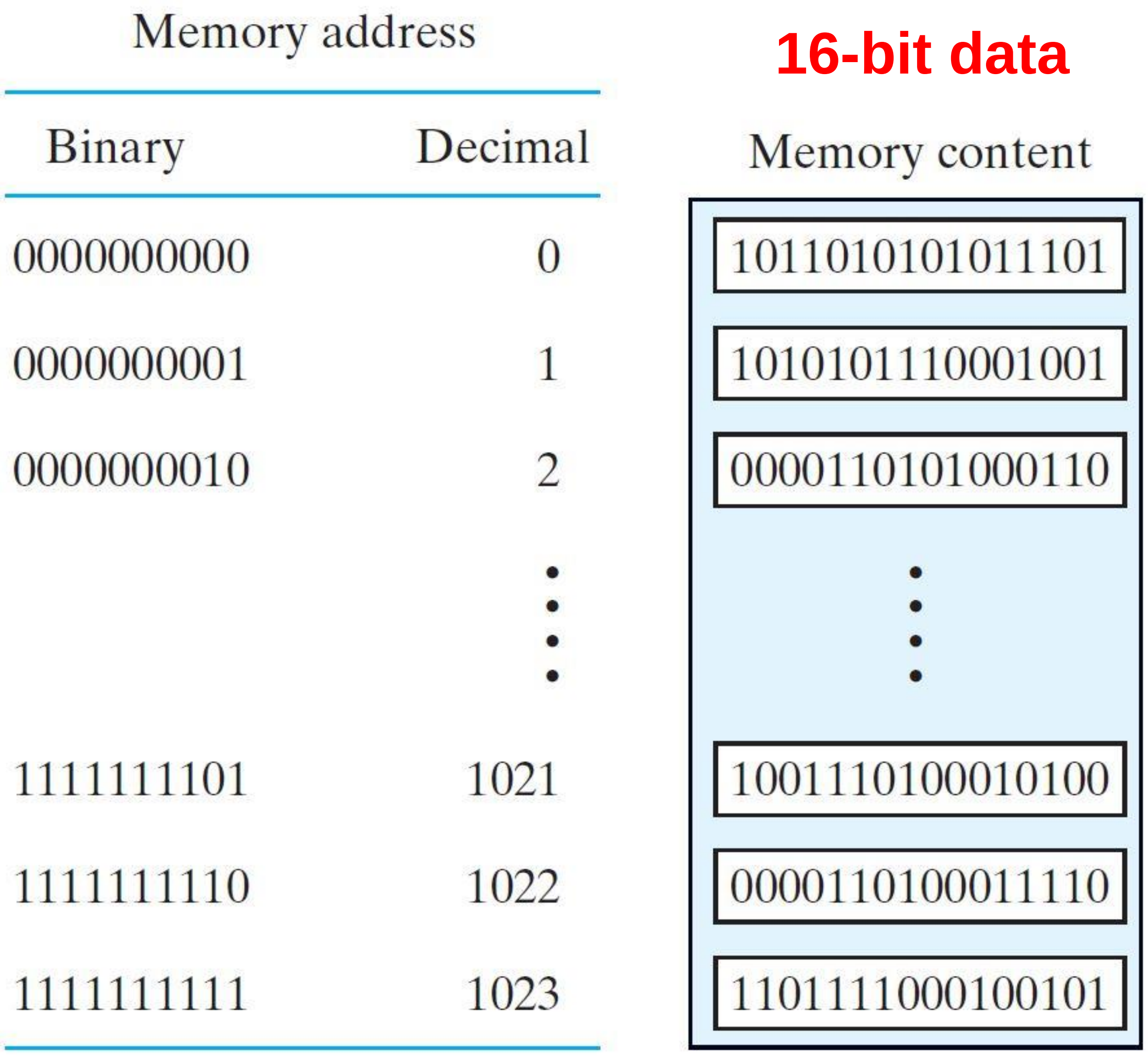
  1 TB (Tera Byte) = 2^{40} Bytes (more than trillion = 10^{12})

- The memory locations can be accessed (addressed) randomly

Memory Address and Content

- Example of a RAM
  - Address = 10 bits
  - 2^{10} addresses
  - From 0 to 1023
  - Data = 16 bits
- Memory capacity =
  $2^{10} \times 16$ bits =
  16 Kbits = 2 KBytes
- Memory can be addressed randomly
- Memory can be read and written

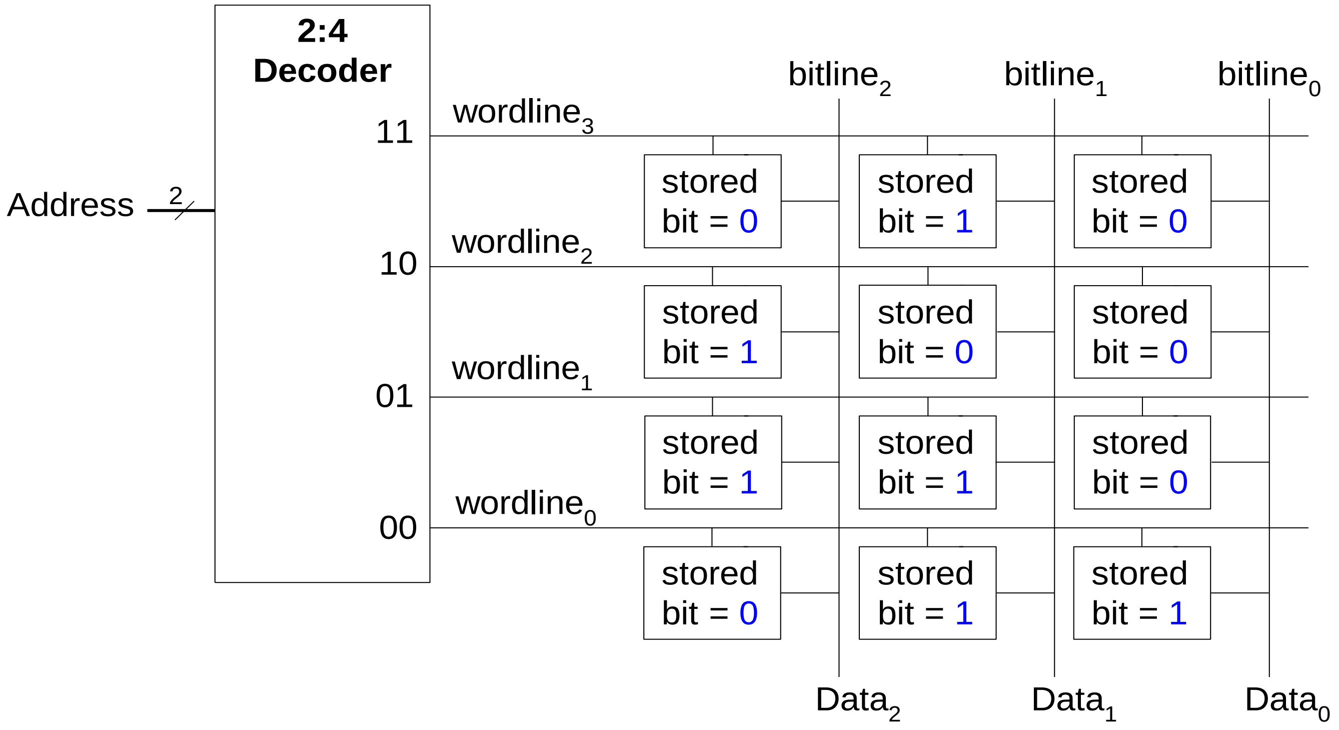
16-bit data



Memory address		Memory content
Binary	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Example of a Small 4×3 Memory Array

- 2-bit Address ➔ 4 wordlines (rows), each storing 3 bits
- The decoder enables one **wordline** (row) at a time
- The data are read on the **bitlines** (columns)
- Each wordline in the memory array can be read and written



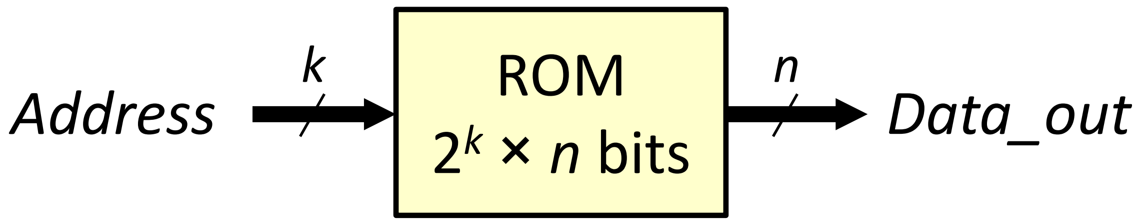
Read-Only Memory (ROM)

❖ Address consists of *k* bits ➔ 2^k memory addresses

❖ At each memory address, there is a word consisting of *n* bits

❖ The *n*-bit word appears at the data output of the ROM

❖ ROM does not have data inputs or a write operation



❖ ROM memory is useful for implementing Boolean Functions

❖ Also useful for storing permanent data

Types of ROMs

- ROM may be programmed in four different ways

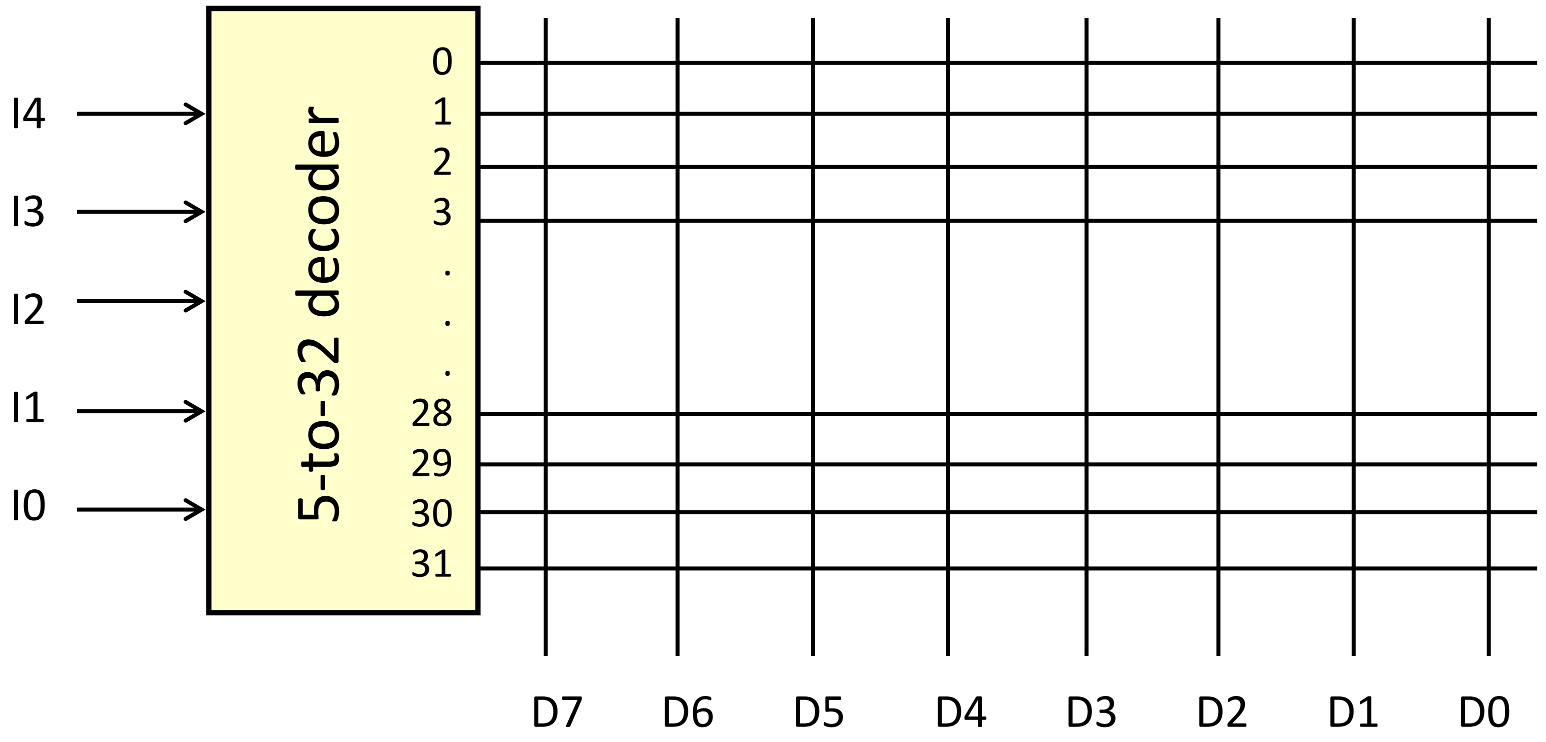
1. **Mask Programming:** done by circuit manufacturer (not by user)
2. **Programmable Read-Only Memory (PROM)**
   - Fuses in the PROM are blown using a high-voltage pulse
   - Can be programmed only once (irreversible)
3. **Electrically Erasable PROM (EEPROM)**

   Can be erased with an electric signal

- **Flash memory** is similar to EEPROMs but has additional circuitry to erase blocks of memory

ROM Internal Structure (32 x 8-bit)

- 5-bit Input ➔ 5-to-32 binary decoder (Only one line is selected)
- Each line = 8 bits ➔ 8-bit Data output



32×8 = 256 connections Bits are stored at the intersection

The 32×8 = 256 intersections are programmable

Implementing a Combinational Circuit

- ❖ Implementing a Combinational Circuit with a ROM is easy
- ❖ Store the truth table of the circuit by programming the ROM

Truth Table with Five Inputs and Eight output functions

I4	I3	I2	I1	I0	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	0	0	0	0	1	1	0
0	0	0	1	0	0	1	0	1	1	0	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
. . .					. . .							
1	1	1	1	1	0	0	1	0	0	0	1	0

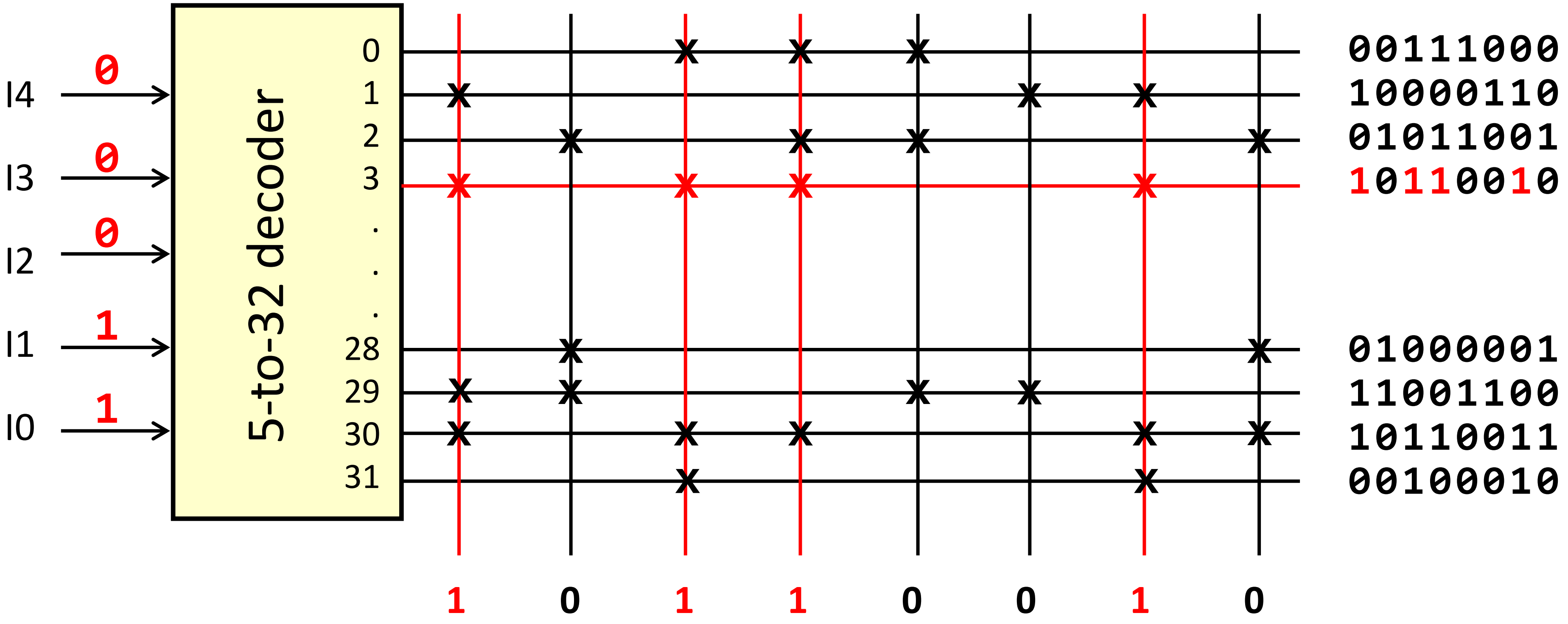
Inputs are used as Address lines to the ROM

Programming a ROM

Every **1** in the truth table ➔ **X** connection is **logic-1**

Every **0** in the truth table ➔ **No X** connection is **logic-0**

Example: At address **`00011`** = (decimal **3**), the word **`10110010`** is stored



Example: Square Function

- Design a square function with a minimum size ROM
- Input **X** = 3-bit number, Output **Y = X²**
- **Solution:** Derive the Truth Table

X_2 X_1 X_0	Square	Y_5 Y_4 Y_3 Y_2 Y_1 Y_0
0 0 0	0	0 0 0 0 0 0
0 0 1	1	0 0 0 0 0 1
0 1 0	4	0 0 0 1 0 0
0 1 1	9	0 0 1 0 0 1
1 0 0	16	0 1 0 0 0 0
1 0 1	25	0 1 1 0 0 1
1 1 0	36	1 0 0 1 0 0
1 1 1	49	1 1 0 0 0 1

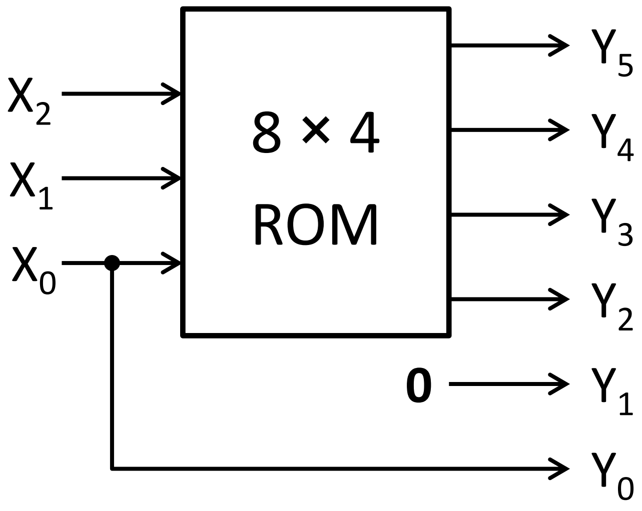
Minimum-Sized ROM Table

- Output Y_0 is identical to input X_0 ➔ No need to store in ROM
- Similarly, Output Y_1 is always **0** ➔ No need to store in ROM
- ROM table ➔ Only need to store Y_5, Y_4, Y_3, and Y_2 in ROM

X_2	X_1	X_0	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

Minimal ROM

Size = $2^3 \times 4$ bits



8 × 4
ROM
X2
X1
X0
Y5
Y4
Y3
Y2
0
Y1
Y0