Contents

Circuits

Now that we know how individual gates work and how they are constructed i.e. how simple Boolean operation can be represented by a logic gate and more complex expressions reduced using Boolean algebra or alternatively, Karnaugh maps, let’s continue with examining how we combine predictable gates to form circuits that perform some function.

More complex Boolean expressions can be represented as a combination of AND, OR and NOT gates, resulting in a logic diagram that describes the entire expression.

At some level, every operation carried out by a computer is an implementation of a Boolean expression.

Categories of Circuits

Recall the two general categories of circuits being:

1. **Combinational Circuits** where the input values explicitly determine the output and
2. **Sequential Circuits** where the output is a function of the input values as well as the existing stage of the circuit
   a. Thus, sequential circuits usually involve the storage of information.

Description of the operations of circuits

The operation of these circuits can be described using three notations:

1. Boolean expressions
2. Logic diagrams and
3. Truth tables.

These notations are different but equally powerful, representative techniques

Classification of Combinational Logic

The combinational logic circuits can be classified into various types based on the ***purpose of usage***, such as:

- arithmetic & logical functions,
- data transmission, and
- code converters.

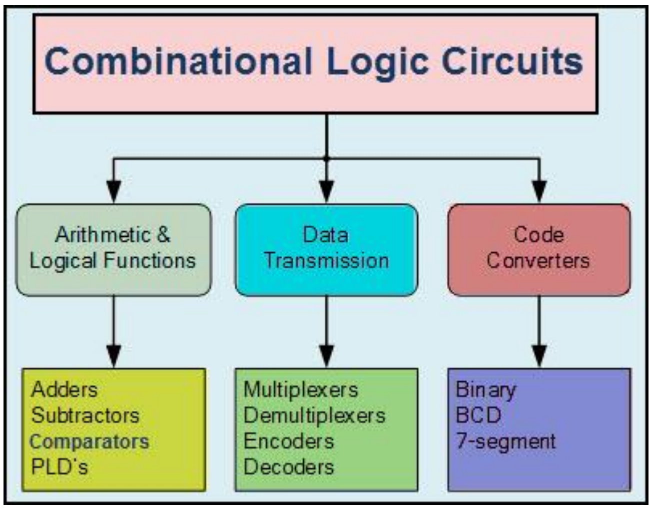
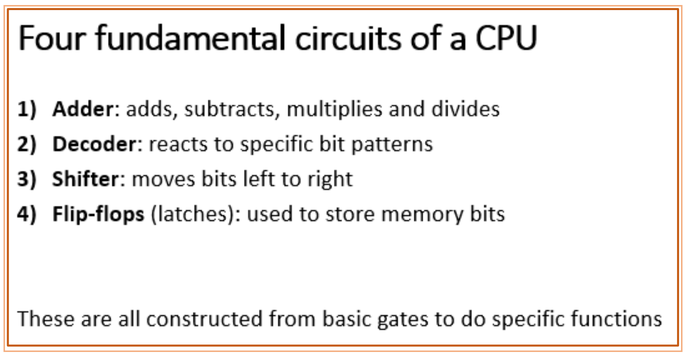


Figure 1: Classifications of Combinational Logic Circuits

As mentioned in an earlier class, there are four fundamental circuits of a CPU, with each constructed from basic gates to do specific functions

1. Adder
2. Decoder
3. Shifter
4. Flip-flops



Four fundamental circuits of a CPU

1) **Adder**: adds, subtracts, multiplies and divides
2) **Decoder**: reacts to specific bit patterns
3) **Shifter**: moves bits left to right
4) **Flip-flops** (latches): used to store memory bits

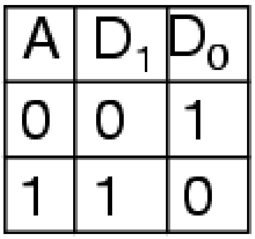
These are all constructed from basic gates to do specific functions

1. Adder

To solve the arithmetic and logical functions we generally use **adders**, subtractors, and comparators which are generally realised by combining various logic gates called combinational logic circuits. This is covered in more depth in the previous notes.

2. Decoder:

- As well as the **adder**, an equally important operation that all computers use frequently is **decoding binary information** from a set of *n* inputs to a maximum of 2^n outputs.
- A ***decoder*** is a circuit that changes a code into a set of signals.
- It uses the inputs and their respective values to select one specific output line
    - i.e. one unique output line is asserted, or set to **1** whereas the other output lines are set to **0**
- The name "Decoder" means to translate or decode coded information from one format into another, so a binary decoder transforms "n" binary input signals into an equivalent code using 2^n outputs.



A	D_1	D_0
0	0	1
1	1	0

A common type of decoder is the line decoder which takes an n-digit binary number and decodes it into 2^n data lines.

The simplest is the 1-to-2 line decoder, as seen here by its truth table

So for example, an inverter (*NOT-gate*) can be classed as a 1-to-2 binary decoder as 1-input and 2-outputs (2^1) is possible:

input **A** produces two outputs **A** and $\overline{\mathbf{A}}$

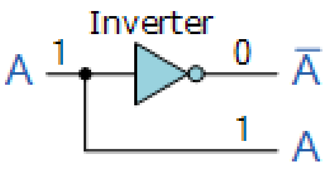
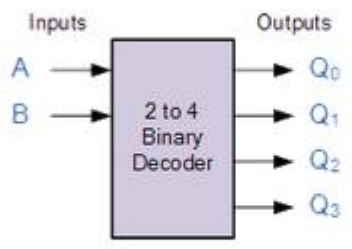
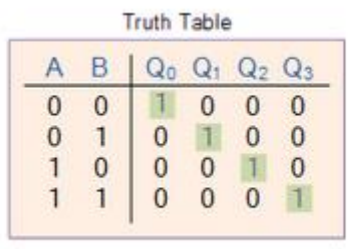


Figure 2 Line Decoder Circuit

- Decoders are normally defined by the number of inputs and the number of outputs e.g. a decoder that has 3 inputs and 8 output s is called a 3-to-8 decoder.
    - **Binary Decoders** are another type of digital logic device that has inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, so a decoder that has a set of two or more bits will be defined as having an *n*-bit code, and therefore it will be possible to represent 2^n possible values.
        - Thus, a decoder generally decodes a binary value into a non-binary one by setting exactly one of its *n* outputs to logic “1”.




Truth Table

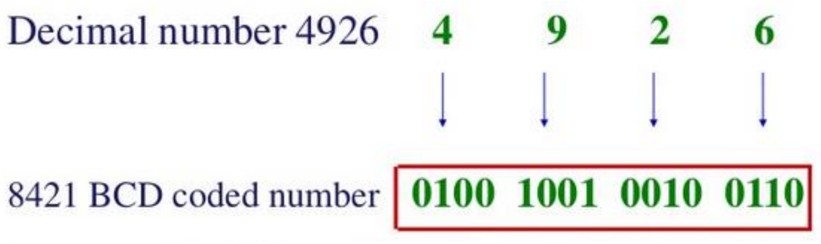
A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

If a binary decoder receives n inputs (usually grouped as a single Binary or Boolean number) it activates one and only one of its 2^n outputs based on that input with all other outputs deactivated.

In other words, a binary decoder looks at its current inputs, determines which binary code or binary number is present at its inputs and selects the appropriate output that corresponds to that binary input.

*Binary-coded decimal (**BCD**) is a class of binary encodings of decimal numbers where each decimal digit is represented by a fixed number of bits, usually four or eight*

A *Binary Decoder* converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to “decode” either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders



Memory Address Decoder

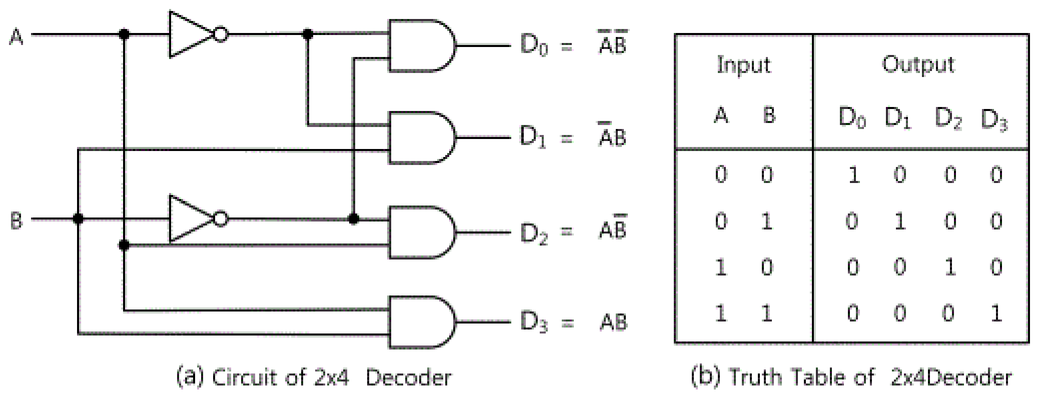
Binary Decoders are most often used in more complex digital systems to access a particular memory location based on an “address” produced by a computing device.

- All modern addresses in a computer are specified as binary numbers.
- When a memory address is referenced (whether for reading or writing), the computer first has to determine the actual address.
  - That is done using a decoder.

In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone.

One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common “**Data Bus**”. In order to prevent the data being “read” from each memory chip at the same time, each memory chip is selected individually one at time and this process is known as **Address Decoding**.

For example:



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

Figure 3: circuit representing a 2-to-4 decoder

Imagine memory consisting of 8 chips, each containing 8K bytes.

Assume that chip 0 contains memory address 0-8191 (≡ $1FFF_{16}$)

chip 1 contains memory address 8192-16383 (≡ 2000_{16}- $3FFF_{16}$) and so on

There is a total of 8K*8 (≡65,536) addresses available.

We will not write down all 64K addresses as binary numbers: however, writing a few addresses in binary form will illustrate why a decoder is necessary.

Given $64=2^6$ and $1K=1^{10}$, then $64K=2^6*2^{10}=2^{16}$, which indicates that we need 16 bits to represent each address.

Recall

- If you have four addresses, addresses 0, 1, 2 and 3, the binary equivalent of theses addresses is 00, 01, 10 and 11, requiring two bits We know $2^2=4$)
- If you have eight addresses. We have to be able to count from 0 to 7 in binary. How many bits does that require? ANS: 3 ($8=2^3$)

All addresses on chip 0 have the format 000xxxxxxxxxxxx.

Because chip 0 contains the addresses 0-8191, the binary representation of these addresses is in the range 0000000000000000 to 0001 1111 1111 1111.

Similarly, all addresses on chip 1 have the format 001xxxxxxxxxxxx and so on for the remaining chips.

The leftmost 3 bits determine on ***which chip*** the address is actually located. We need 16 bits to represent the entire address, but on each chip, we only have 2^{13} addresses.

Therefore, we need only 13 bits to uniquely identify ***an address on a <u>given</u> chip***. The rightmost 13 bits give us this information.

When a computer is given an address, it must first determine ***which*** chip to use: then it must find the ***actual address on that specific chip***

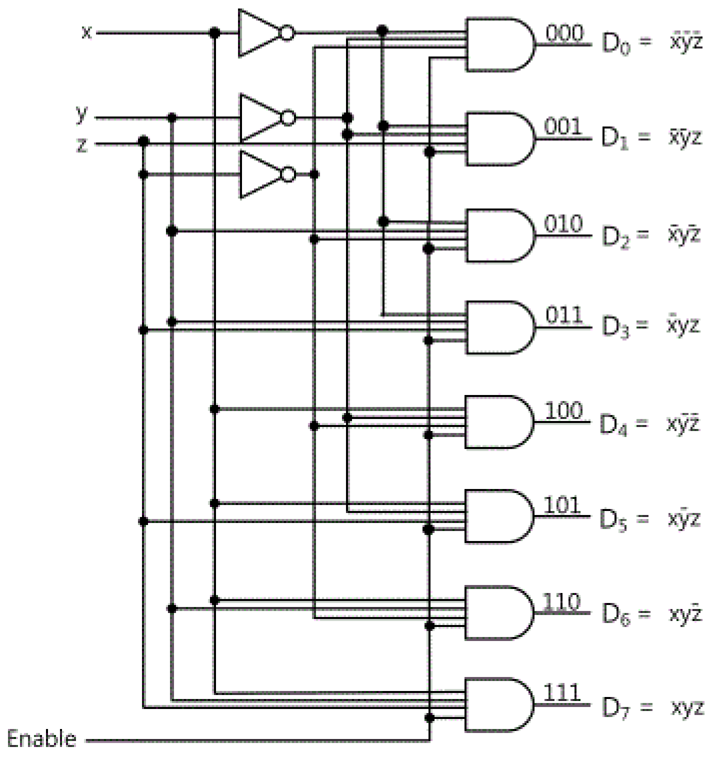
In this example, the computer would use the 3 leftmost bits to pick the chip and then find the actual address on the chip using the remaining 13 bits. These 3 high-order buts are actually used as the inputs to a decoder so that the computer can determine which chip to activate for reading/writing

If the first 3 bits are 000, chip 0 should be activated.

If the first 3 bits are 111, chip 7 should be activated.

If the first 3 bits are 010, chip ___ should be activated? ANS: 2

Turning on a specific wire ***activates a chip***. The output of the decoder is used to activate one, and only one, chip as the addresses are decoded.


x
y
z
Enable
000 $D_0 = \bar{x}\bar{y}\bar{z}$
001 $D_1 = \bar{x}\bar{y}z$
010 $D_2 = \bar{x}y\bar{z}$
011 $D_3 = \bar{x}yz$
100 $D_4 = x\bar{y}\bar{z}$
101 $D_5 = x\bar{y}z$
110 $D_6 = xy\bar{z}$
111 $D_7 = xyz$

Some binary decoders have an additional input pin labelled "***Enable***" that controls the outputs from the device.

This extra input allows the decoders outputs to be turned "ON" or "OFF" as required: if this enable input is **0** all outputs become **0** if this enable input is **1** the decoder operates normally.

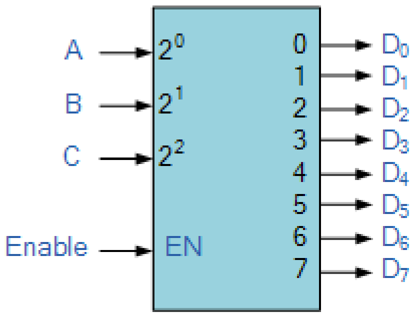
These types of binary decoders are commonly used as “memory address decoders” in microprocessor memory applications.

So, 3 input decoders (have $2^3=8$ outputs) and are composed of AND gates.

One use of them is to change the binary number to octal or decimal number.

E.g. with **Enable**=**1**, **101** is entered to x, y and z inputs, 1 is delivered through the x line, 0 through the $\bar{y}$ line and 1 through the z line to the output **D5** and expressed as a certain signal.

This signal is 101_2 and decoded in octal or decimal number to be expressed as **5**.



We can say that a binary decoder is a **demultiplexer** with an additional data line that is used to enable the decoder

Figure 4 74LS138 Binary Decoder

Another example: One of the most commonly used devices for doing **Digital Decoder** is called the Binary Coded Decimal (BCD) to 7-Segment Display Decoder.

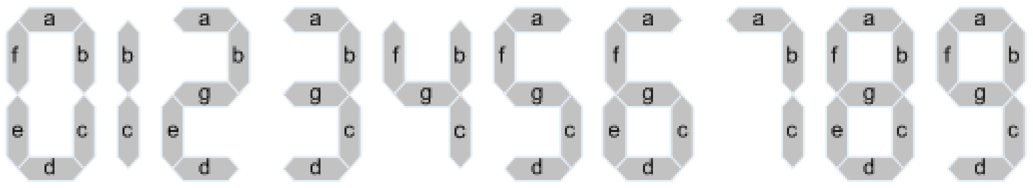


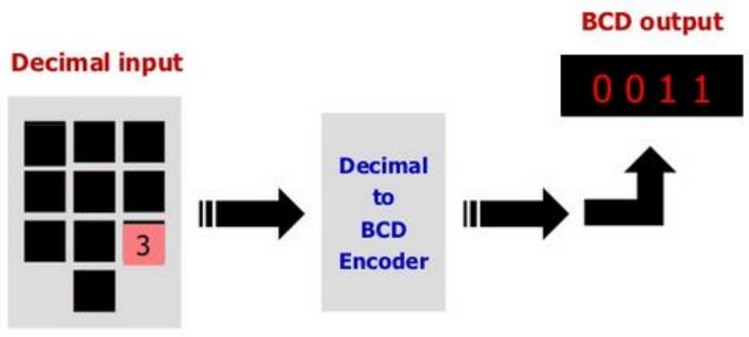
Figure 5 7-segment Display Elements for all Decimal Numbers

A ***demultiplexer***, sometimes abbreviated ***demux***, is a circuit that has one input and more than one output. It is used when a circuit wishes to send a signal to one of many devices. This description sounds similar to the description given for a ***decoder***, but a decoder is used to select among many devices while a demultiplexer is used to send a signal among many devices.

Encoder

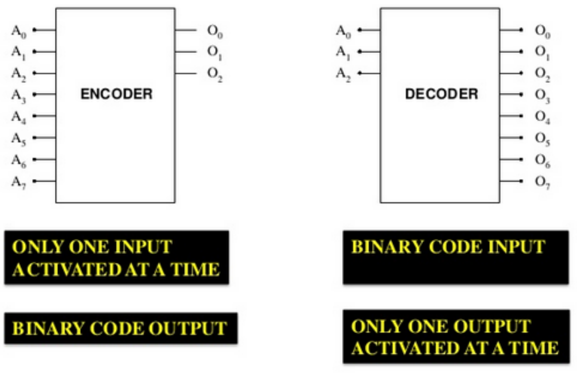
Encoders are combinational logic circuits and they are exactly opposite of decoders. They accept one or more inputs and generate a multibit output code.

The binary output code depends on which of its inputs are activated.



Encoders perform exactly reverse operation than decoder.

An encoder has M input and N output lines. Out of M input lines only one is activated at a time and produces equivalent code on output N lines.


A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7
ENCODER
O_0 O_1 O_2
ONLY ONE INPUT ACTIVATED AT A TIME
BINARY CODE OUTPUT
A_0 A_1 A_2
DECODER
O_0 O_1 O_2 O_3 O_4 O_5 O_6 O_7
BINARY CODE INPUT
ONLY ONE OUTPUT ACTIVATED AT A TIME

If a device output code has fewer bits than the input code has, the device is usually called an encoder.

All of the above are combinational circuits. The problem with these are that they have no memory ability e.g. vending machines – if you put in money, you need it to remember how much you’ve put in! Combinational Circuits will not be able to handle this so now we need a type of circuit that can store data

The second of two general categories of circuits are called **Sequential Circuits** where the output is a function of the input values as well as the existing stage of the circuit

a. Thus, sequential circuits usually involve the storage of information.

Sequential circuits use past inputs to determine present output and therefore, we have event ordering.

Some sequential circuits are asynchronous – they become active the moment any input value changes. Synchronous sequential circuits use clocks to order events and this is the type that we will concentrate on in this set of notes.

A **clock** is a circuit that emits a series of pulses with precise pulse width and a precise interval between consecutive pulses

This is called ***clock cycle time*** and speed is generally measured in MHz or GHz

A clock is used by sequential circuits to decide when to update the state of the circuit

Inputs to the circuit can only affect the storage element at given instances of time.

Most sequential circuits are **edge triggered** (as opposed to being level triggered) which means they are allowed to change their states on either the rising of falling edge of the clock signal.

3. Shifter

A Shifter circuit is also called the Shift Register

It is a ***sequential logic circuit***

It can be used for the storage or the transfer of binary data

This sequential device loads the data present on its inputs and then moves or “shifts” it to its output once every clock cycle, hence the name **Shift Register**.

Shifters are most often found in circuits that work with groups of signals that together represent binary numbers, where they are used to move data bits to new locations on a data bus (i.e., the data bit in position 2 could be moved to position 7 by right-shifting five times), or to perform simple multiplication and division operations.

Shifting a bit one position to the left takes it to the next higher *power of 2*

Shifting an unsigned integer to the left has the same effect as multiplying that integer by 2 (but using significantly less machine cycles to do it!)

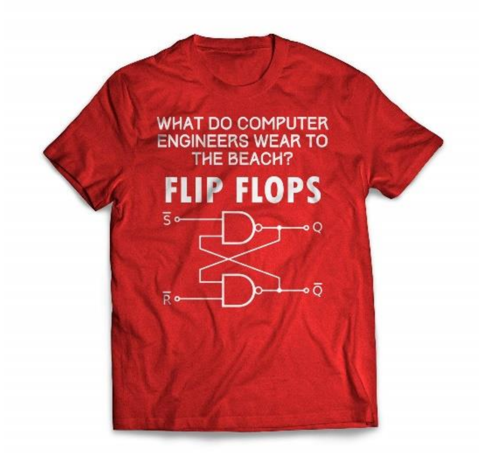
A shifter circuit can multiply a number by 2, 4, or 8 simply by shifting the number right by 1, 2, or 3 bits (and similarly, a shifter can divide a number by 2, 4, or 8 by shifting the number left by 1, 2, or 3 bits).

Shift Registers are used for data storage or for the movement of data and are therefore commonly used inside calculators or computers to store data such as two binary numbers before they are added together

A shifter is a circuit that produces an *N*-bit output based on an *N*-bit data input and an *M*-bit control input, where the *N* output bits are place-shifted copies of the input bits, shifted some number of bits to the left or right as determined by the control inputs.

- Shift Registers are capable of storage and transfer of data
- A shift register is a cascade of flip flops, sharing the same clock
    - they are made up of Flip-Flops which are connected in such a way that the output of one flip-flop serves as the input of the other flip flop, depending in the purpose of the shift register
- One of the most common uses of a shift register is to convert between serial and parallel interfaces.

4. Flip-Flops



Normally, a junction transistor (BJT) is "off" when there is no base current and switches to "on" when the base current flows.

That means it takes an electric current to switch the transistor on or off.

But transistors like this can be hooked up with logic gates so their output connections ***feed*** back into their inputs.

The transistor then stays **on** even when the base current is removed. Each time a new base current flows, the transistor "flips" on or off. It remains in one of those stable states (either on or off) until another current comes along and flips it the other way.

This kind of arrangement is known as a flip-flop and it ***turns a transistor into a simple memory device that stores a zero (when it's off) or a one (when it's on)***.

Flip-flops

- Are the basic technology behind computer memory chips, which is *volatile*.
- Are a sequential logic circuit in which their output is dependent on their input state

Unlike Combinational Logic circuits that change state depending upon the actual signals being applied to their inputs at that time, ***Sequential Logic circuits*** have some form of inherent "Memory" built in.

The output state of a sequential logic circuit is a function of the following three states:

1. the "present input",
2. the "past input" and/or
3. the "past output".

Sequential Logic circuits remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits "Memory".

SR Flip-Flop

The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will "SET" the device (meaning the output = "1"), and is labelled **S** and one which will "RESET" the device (meaning the output = "0"), labelled **R**.

Then the SR description stands for "Set-Reset". The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level "1" or logic "0" depending upon this set/reset condition.

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to its current state or history.

The term "Flip-flop" relates to the actual operation of the device, as it can be "flipped" into one logic Set state or "flopped" back into the opposing logic Reset state.

The NAND Gate SR Flip-Flop

The simplest way to make any basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NAND gates as shown, to form a Set-Reset Bistable also known as an active LOW SR NAND Gate Latch, so that there is feedback from each output to one of the other NAND gate inputs. This device consists of two inputs, one called the *Set*, S and the other called the *Reset*, R with two corresponding outputs: Q and its inverse or complement $\overline{Q}$

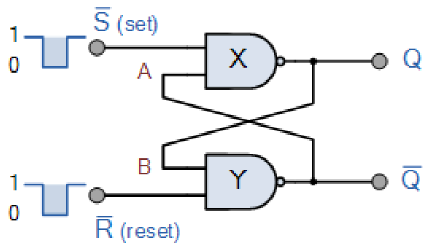
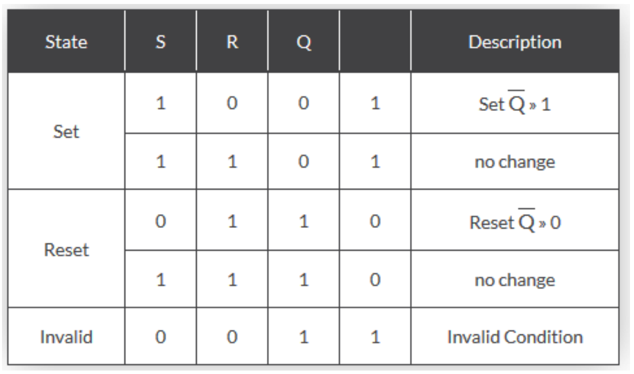


Figure 7 Circuit Diagram for an SR flip-flop



State	S	R	Q		Description
Set	1	0	0	1	Set $\overline{Q}$ » 1
	1	1	0	1	no change
Reset	0	1	1	0	Reset $\overline{Q}$ » 0
	1	1	1	0	no change
Invalid	0	0	1	1	Invalid Condition

Figure 6 Truth Table for the SR function

Consider the circuit shown above.

What is happening at each stage?

Why is there an ***Invalid Condition***?

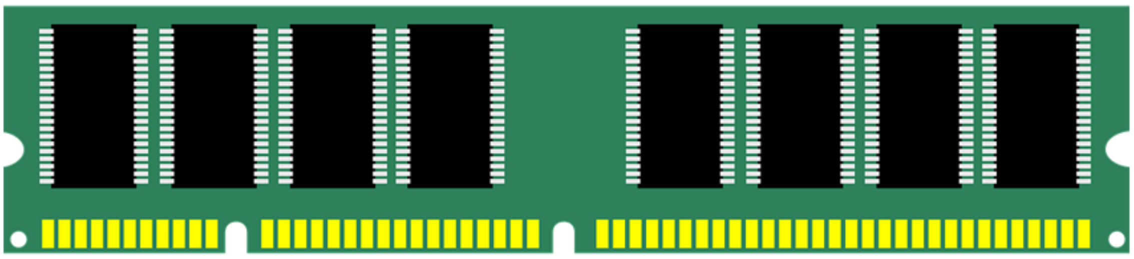
Random Access Memory

Random Access Memory (RAM) is volatile memory that sits next to the CPU. It is used to store instructions and data currently used by the CPU.

RAM consists of billions of **Data Cells**, each data cell being able to store one **bit** of information.

> For instance a 2GB RAM can store c.2,000,000,000 Bytes of information = c.16,000,000,000 bits of information and hence consists of c.16,000,000,000 data cells.

D-Type Flip-Flop Circuits



Random Access Memory (RAM) consists of billions of data cells, each data-cell uses a D-Type flip-flop circuit.

- A D-Type Flip-Flop Circuit is used to store 1 bit of information. It has two input pins (Called D (Data) and E (Enabler) and two output pins (Q and $\overline{Q}$).
- Each data cell consists of a **D-Type Flip-Flop circuit** that is built using four **NAND logic gates** connected as follows:

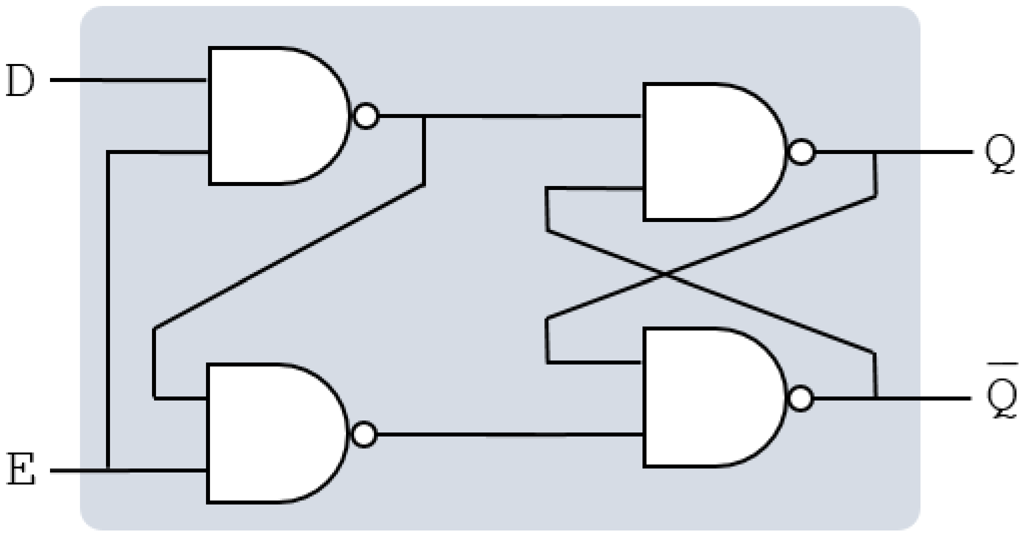
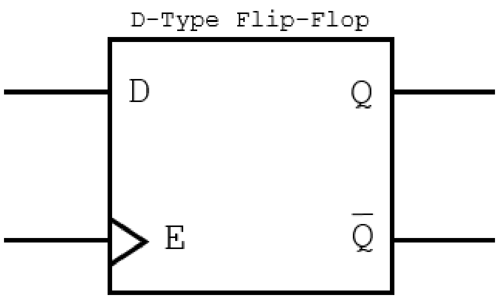


Figure 8 D Type Flip-Flop

We equivalently represent a D-Type Flip-Flop Circuit as follows:



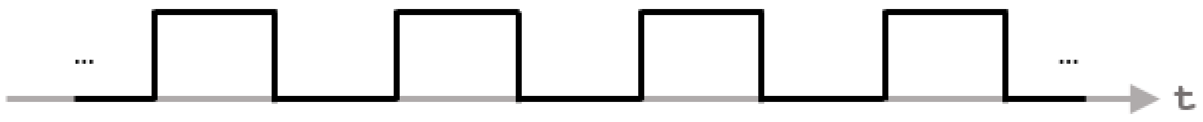
Input		Output	
D	E	Q	$\overline{Q}$
0	1	0	1
1	1	1	0
0	0	Q	$\overline{Q}$
1	0	Q	$\overline{Q}$

Figure 9 Truth Table of D Type Flip Flop

- When the enabler input E is set to 1, the output Q can be set to the Data input D.
- When the enabler input E is set to 0, the output Q cannot be changed. It remains as its previous value.
- In other word it retains its value.
- This is why this circuit is used to create memory cells (e.g in the RAM).

Clock Signal and Delaying Effect

The enabler input E is often connected to another circuit called the **clock** (e.g. CPU clock). The clock signal constantly and regularly alternate between two states: 0 and 1, similar to a heart beat. Inside the CPU the clock signal controls the execution of the fetch-decode-execute cycle.



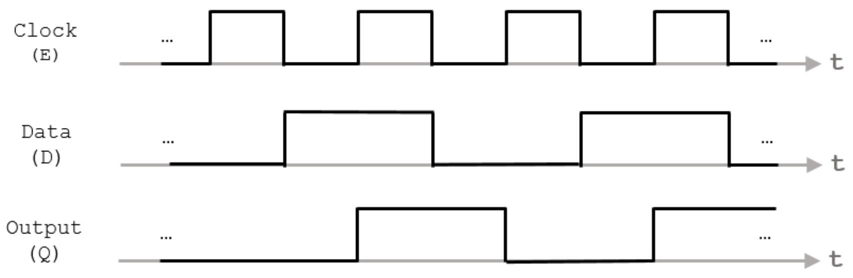
When the clock signal is applied to the Enabler input (in this case also called the clock input), the flip-flop output Q can only change values when triggered by the clock signal. The value of the flip-flop is held or **delayed** until the next clock signal.

This is why we call this circuit a D-Type flip flop where **D** stands for **D**elay.

In other words, the change of input (D) is not applied immediately (to the output Q) but is applied at the next "tick of the clock".

This delaying effect is also called a **latch**.

These are devices which can have their output or outputs set in one of two basic states: 1 or 0 and will remain "latched" (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause a change of state once again.



Circuitry Simulators:

You might find some very useful online Circuitry simulators.

1. http://www.falstad.com/circuit/index.html
2. ..... <<your finds here>>