isaac computer science

Useful circuits

Note: This page is not relevant for GCSE.

In this section, you are going to look at a few circuits that underpin some of the basic operations of a computer system.

A Level

Binary equivalence tester

It's possible to build a circuit to test whether two bit patterns are the same, that is, a circuit that tests binary equivalence. (This isn't a circuit that you need to know for any exam, but it's very simple and therefore a good introduction to useful circuits.)

An equivalence tester should test whether two inputs are the same: if A and B are the same, the circuit should output a 1, and if they are different, it should output a 0. The truth table for an equivalence tester looks like this:

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Compare this to the truth table for an $\veebar$ (XOR) gate, and you will see that is it the opposite.

A	B	$A \veebar B$
0	0	0
0	1	1
1	0	1
1	1	0

So to create an equivalence tester, you simply use an XOR gate followed by a NOT gate.

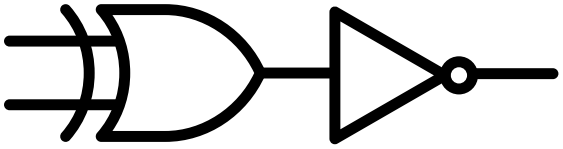


Figure 1: A circuit for testing binary equivalence

A Level

Binary half adder

Processors can only carry out a very limited number of different operations on binary data, and you will start to appreciate this when solving problems using assembly language. And these operations are simply combinations of three basic operations: adding, shifting, and flipping. If you understand how adding works, you will be able to properly grasp what happens in a computer's Arithmetic and Logic Unit (ALU).

A **binary half adder** circuit adds two single-bit binary numbers together. The possible calculations are:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$

The result of the final calculation is interesting. The answer cannot be stored with a single bit. The result of $1 + 1 = 0$ **carry** 1.

Now put the data into a truth table with two inputs and two outputs. The inputs are labelled A and B and these represent the two bits being added. The outputs are labelled C and S. S is the sum (the result of the addition as a single bit), and C is the carry bit.

Here is the truth table:

INPUTS		RESULT	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

You can see that the 'carry' column would be produced by the logic $A \wedge B$.

You can also see that the 'sum' column would be produced by the logic $A \veebar B$.

Using this information, you can create the circuit, which takes two single-bit numbers and adds them together. This is called a **binary half adder**.

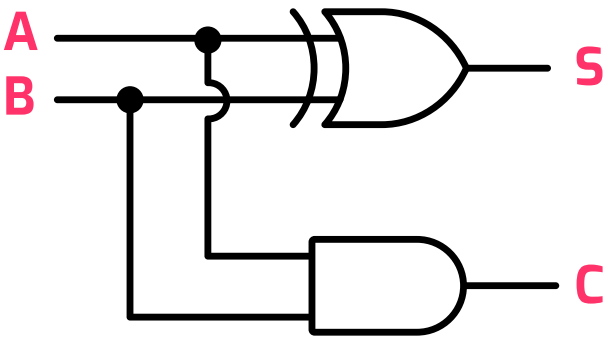


Figure 2: Binary half adder

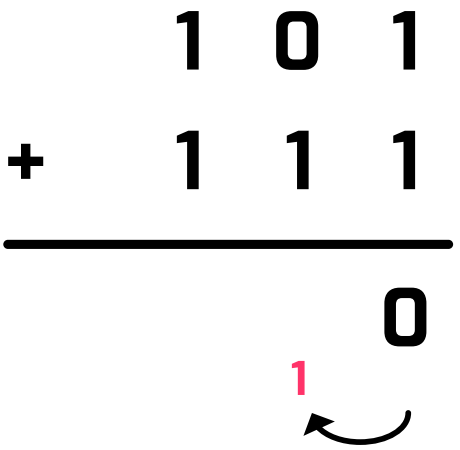
A Level

Binary full adder

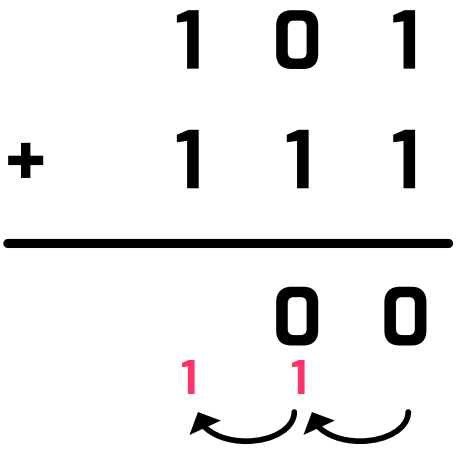
A half adder can only add two single-bit numbers together. To add bigger numbers, a more complicated circuit is needed.

Let's think about adding 101_2 + 111_2.

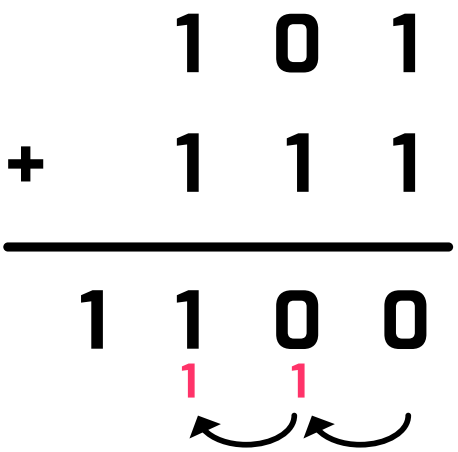
First, $1 + 1$ gives you 10, so put a 0 in the right-most column and carry a 1 to the next column.



To calculate the result of the next column, you need to add in the carried 1. The calculation is $0 + 1 + 1 = 10$, so put a 0 in the working column and carry a 1 to the next column.



You can then add the final pair of bits. Again, you have a carried 1, so the calculation is $1 + 1 + 1 = 11$. You put a 1 in the working column and carry a 1 to the next column. This can be positioned as the answer as there are no more bits to add.



You should see now that to reliably add two single-bit numbers in any position, **three** inputs are needed. These are the values for the two bits and the carry-in from the previous column. The circuit produces two outputs: the sum of the bits and a carry-out (to the next column).

Such a circuit has the following truth table:

A	B	C**in**	C**out**	S **(sum)**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

You can build this circuit from two half adders: use the first half adder to calculate $A + B$, then use the second half adder to add the result of the first half adder and the carry-in from the previous column to get the final sum. If either of the half adders has created a carry-out, set the carry-out to 0; to do this, you **OR** the carry-outs of both half adders to get the final carry-out.

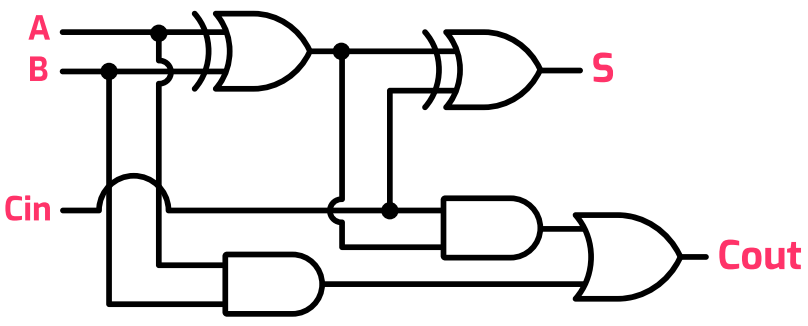


Figure 3: Binary full adder

Binary full adders can be combined to build circuits that will add very long binary numbers.

A Level

Four-bit binary adder

This isn't a circuit that you need to know for any exam, but it shows how full adders can be chained together to add multi-bit numbers.

You can think of a full adder as one functional block with three inputs and two outputs.

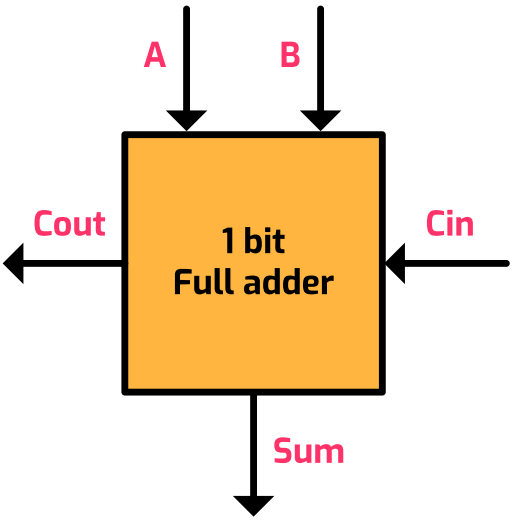


Figure 4: A single-bit binary full adder

You can then chain as many full adders together as you need to add multi-bit numbers. The carry-out from one column feeds into the carry-in for the next column.

The diagram below shows a four-bit binary adder. The four-bit binary number A (broken into bits A0–3) is added to the four-bit binary number B to get the four-bit binary number S and the final carry C4. The value of C4 can be used to control a status register to indicate overflow (a number that is too big to be represented using the current representation).

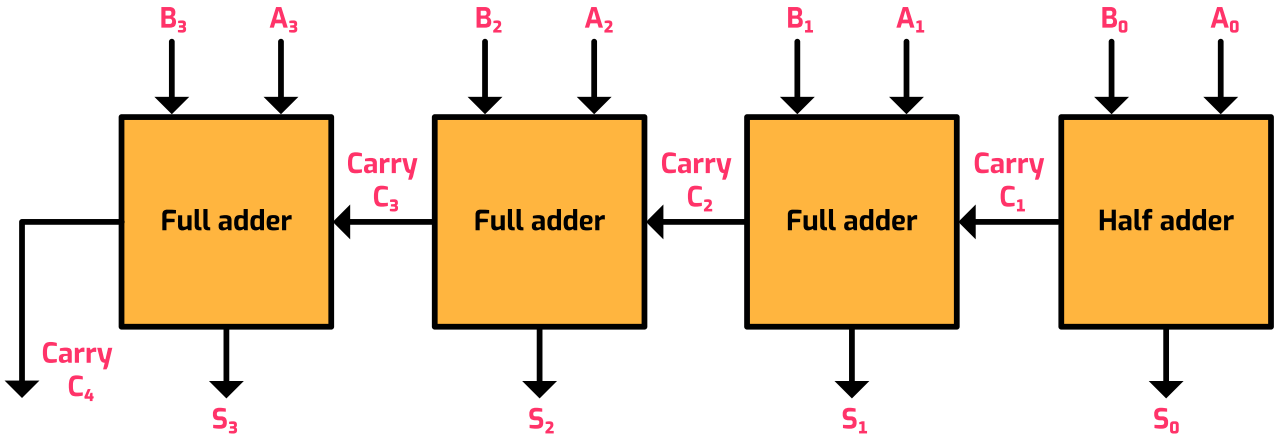


Figure 5: A four-bit binary adder

Notice that the chain (of gates) starts with a half adder because there is no carry-in bit.

A Level

Sequential circuits

So far, all of the circuits you have considered are **combination circuits** where the outputs depend entirely on the inputs. The next set of circuits you will look at are **flip-flops**. These are examples of **sequential circuits** where the ouputs depend not only on the current inputs, but also on the sequence of past inputs. A flip-flop allows a previous output value to be stored, and thus acts as a simple memory device. These memory units are **volatile**, meaning that they do not retain their state when the power is switched off.

Sequential circuits can be asynchronous or synchronous. In an asynchronous circuit, the inputs will be processed as they arrive. In a synchronous circuit, the operations are controlled by a clock. The clock within a computer system is an oscillator that produces a continuous stream of alternating signals, interpreted as 1s and 0s. The elapsed time between the change of signals is a called a clock cycle. The clock signal is carried to every component of the computer system to synchronise operations.

A Level

SR flip-flop

An SR flip-flop is simple one-bit memory device. This asynchronous sequential circuit is built of two cross-coupled **NOR** gates. This means that the output from each gate feeds in as an internal input to the other gate as shown in **Figure 6**. A similar circuit can be built using **NAND** gates.

The circuit has two inputs, labelled S and R. S is used to SET the device and R is used to RESET the device. The circuit also has two outputs — Q and $\neg Q$.

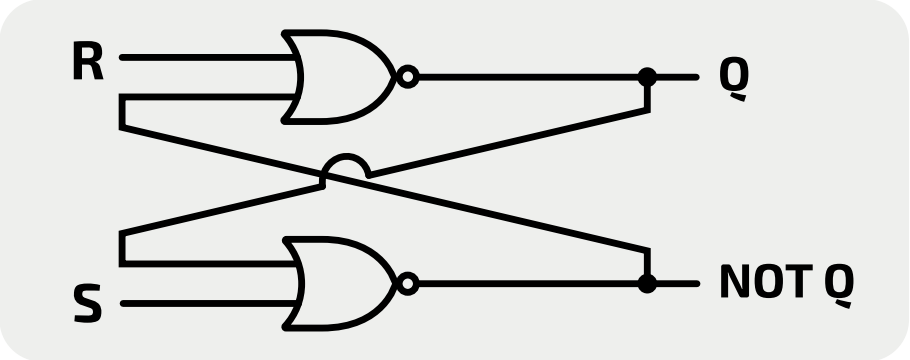


Figure 6: SR flip-flop

In this circuit, when $S = 1$ and $R = 0$, the output Q will be 1. When $S = 0$ and $R = 1$, the output Q will be 0. In this way, S is said to SET the flip-flop and R is said to $RESET$ the flip-flop. After being set or reset, when $S = 0$ and $R = 0$, the previous state of Q will be maintained through the feedback loop, until the circuit is set or reset. In this way, the circuit acts as a single-bit memory device.

Here is the circuit's truth table.

Inputs		Outputs	
S	R	Q	$\neg Q$
0	0	As previous	
0	1	0	1
1	0	1	0
1	1	illegal	

The circuit must not allow both inputs as 1. You will see, if you follow the logic, that this would cause both Q and $\neg Q$ to be 0, which is contradictory.

The SR flip-flop can be unstable if the inputs do not arrive at the same time. This can be improved by the use of a clock signal to synchronise the gate's operation.

A Level

D-type flip-flop

The D-type flip-flop is a synchronous sequential circuit that can be used to store the value of a single binary digit.

The flip-flop has two external inputs: a data signal (D) and a clock signal. It also has an internal loop, which allows the previous output value to be stored.

When the clock signal is low (0), changes at D make no difference to the outputs. When the clock signal is high (1), then the value of input D will appear at output Q. $\neg Q$ is always the inverse of Q.

This is the circuit diagram for the flip-flop:

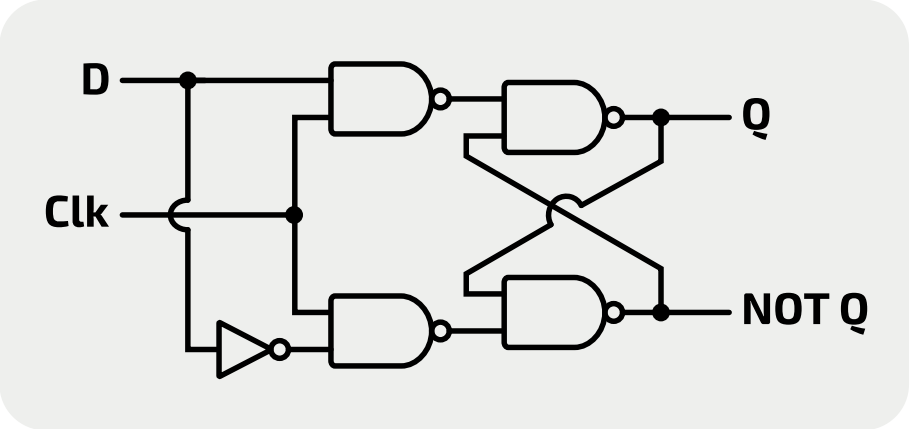


Figure 7: D-type flip-flop

The circuit's truth table is:

Clock	D	Q	$\neg Q$
0	0	Previous value of Q	Previous value of $\neg Q$
0	1	Previous value of Q	Previous value of $\neg Q$
1	0	0	1
1	1	1	0

One of the problems with this flip-flop is that if there are changes in the data during the period when the clock signal is high, the output at Q will change in line with D. This may not be a desirable feature of the circuit. An **edge-triggered** D-type flip flop will deal with this problem.

A Level

Edge-triggered D-type flip-flop

An edge-triggered D-type flip-flop introduces a delay in processing the clock signal so that the data (D) is only processed at the **rising edge** of each signal change from the clock (Clk).

On the rising edge of each clock signal, the state of the data input (D) is accepted, and the output (Q) is updated to reflect this. Once the clock signal input goes **low**, the state of the circuit will not change and it will continue to output (and thus "store"), whatever value was already present on its output.

This process is shown in the diagram in **Figure 8** below. The vertical lines show where the clock signal is at its rising edge and it is only at this point that the data signal is accepted.

- At the outset, the value of Q is low (0)
- At the first rising edge of the clock, the value of D is low (0) so Q does not change
- This is also true at the second rising edge
- At the third rising edge, the value of D is high (1) so the output at Q is updated to reflect this

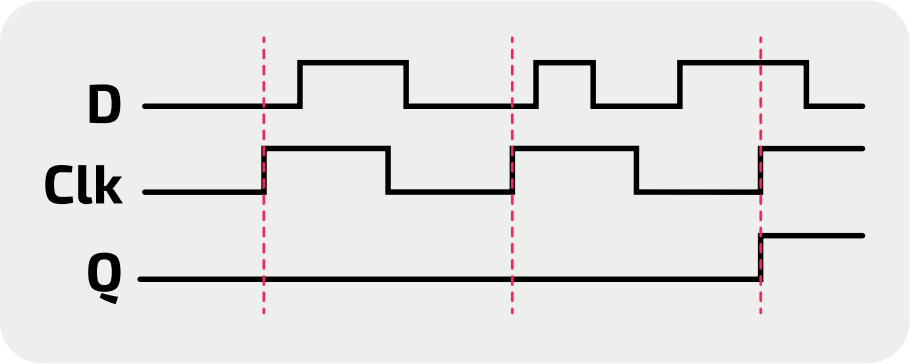


Figure 8: Data is processed on the rising edge of the clock signal

A Level

JK flip-flop

A JK flip-flop is the most widely used flip-flop. It is is more more stable than an SR flip-flop and more flexible than a D-type flip-flop. Because it has two inputs (J and K) in addition to the clock signal (Clk), it has four functions — SET, RESET, HOLD, and TOGGLE.

The circuit is named after its inventor, Jack Kilby.

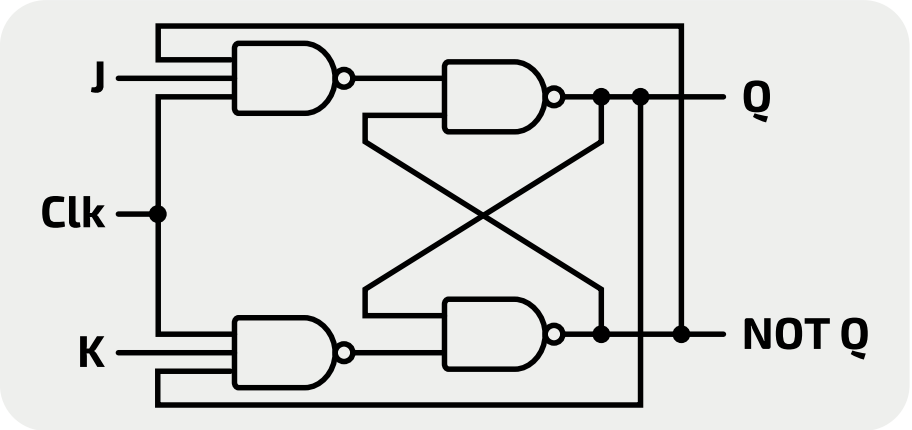


Figure 9: JK flip-flop

When the clock signal is low (0), the outputs do not change.

When the clock signal is high (1) and:

- When J = 0 and K = 0, there is no change to Q
- When J = 0 and K = 1, the flip-flop is RESET and Q = 0
- When J = 1 and K = 0, the flip-flop is SET and Q = 1
- When J = 1 and K = 1, the Q value will toggle

This prevents the illegal states that cannot be allowed. In the same way the D-type flip-flop can be made into an edge-triggered device, practical JK flip-flops are also edge-triggered. This ensures the device only toggles once, when J and K are both equal to 1.

Here is the circuit's truth table:

Inputs			**Outputs**	
Clk	J	K	Q	$\neg Q$
0	0/1	0/1	as previous	
1	0	0	as previous	
1	0	1	0	1
1	1	0	1	0
1	1	1	toggle	

Several JK flip-flops can be used to produce shift registers within a computer system and a simple binary counter can be made by connecting several JK flip-flops.