arrived. The information that someone has turned up for work is stored in the circuit.

That's almost exactly how information is stored in computer memory: all information reduces to the presence or absence of electricity in a circuit. Naturally there's more to it than that, so let's improve the management information system. Suppose we have four separate switch/bulb circuits (the four switches in a row at the door, and the four bulbs in a corresponding row in the office), so that closing the leftmost switch illuminates the leftmost bulb, and so on. Now imagine that every employee is told to close the switches in a unique way, so that when Catherine arrives she closes the first and second switches and opens the third and fourth; Richard closes the fourth switch and opens all the others; Bobby closes the first and third and opens the second and fourth; and so on for all the employees. The lights in the office now show the Manager which of the employees has turned up for work.

Suppose that the OFF position of each switch is labelled 0, and the ON position is labelled 1: therefore Catherine has to set the switches 1100 (first two switches ON, third and fourth OFF), Richard has to make the pattern 0001 (fourth switch ON, the others OFF) and Bobby has to set 1010 (first and third ON, the other two OFF). If the Manager reads each light bulb as 1 if it's ON, and 0 if it's OFF, then both the employees and the Manager will be speaking the same identification language. '0001' means 'Richard' to both people.

How many unique patterns of switches are there? Each switch can be in one of two positions, and there are four switches, so there are $2\times2\times2\times2=16$ different patterns. Let's consider all the possibilities:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

Try as you like, you can't make any more patterns than these, and there are 16 of them.

Notice how quickly we've moved from the concrete picture of light bulbs in a room, to the abstract matter of patterns of 1's and 0's. If we can abstract a little further we can turn these patterns into numbers.

Think about counting and writing down as you count. You can count from nought to nine very easily because each of those numbers has a unique name and a symbol to represent it. But what do you write down after nine? You have a name, ten, for that number, but no separate symbol to represent it. Therefore you must re-use some of the other symbols: 10, 11, 12, and so on until 99, when you run out of possibilities again, so the next number has three columns (100). This seems trivial, but you may remember how difficult it was when you learned it at school: all that squared paper with Hundreds Tens and Units written at the top of each sum? You now know that the number 152 means "1 in the Hundreds, 5 in the Tens, 2 in the Units", or $100+50+2=152$. Counting works like this because we have ten digits (0,1,2,3,...,9) which we arrange to represent all possible numbers.

How does counting work, however, if there are only two digits: 0 and 1? We can count to 1 easily, but how can we represent the next number? We have run out of unique digits, so we must re-use what we have (just as we did when counting with ten digits), and write the next number as 10. Now we know that the next number is called 'two', so in this system 10 represents the number two. The next number as we count is three, and we must write that as 11. Then what? We've run out of two-digit combinations, so the next number, four, must be represented as 100; five must be 101, six is 110, and seven is 111. Here, we are counting in decimal numbers (nought, one, two, etc), but we're writing these down in binary numbers (0,1,10,11,100,101,...).

In the same way as a decimal number such as 152 means: $(1\times100)+(5\times10)+(2\times1)$, the binary number 101 means: $(1\times4)+(0\times2)+(1\times1)$. Instead of having hundreds, tens, and units columns for our numbers, we must use columns marked: fours, twos, and units. In a decimal number the value of a digit is multiplied by ten for every column it moves to the left; in a binary number the value of a digit is multiplied by two for every column it moves to the left.

So that's the binary system: just a different way of representing numbers. If you know Roman numerals you don't find it hard to accept that there are VII dwarfs in *Snow White*; so why not write 111 dwarfs? The actual number of dwarfs is not changed by the way we represent it, but it is a good idea to say the binary number as 'binary one one one', and to write it as '111 b' so that you don't confuse it with a decimal representation.

Now we can return to our original analogy of how the factory workers switch patterns, and decide on a method of making these a little easier to use. The most sensible thing to do is to treat these patterns as four-digit binary numbers. This means that Catherine's signal is 1100 binary, which is 12 decimal. Richard's signal is 0001 binary (1 decimal), and Bobby's signal is 1010 binary (10 decimal). When the Manager looks at a pattern of lights in the office, he or she can read it as a binary number, convert it to its decimal equivalent, and look down the list of employees to see who that number corresponds to. Thus we can say that information is stored in the current of electricity, and the switches make it meaningful.

Our analogy has given a simple picture of how information is represented in a computer: to the computer it's just patterns of voltages (i.e. lights are ON or OFF), but we humans find it easier to consider those patterns as binary numbers. It's all a matter of representation. If you now think of 1010 as the code meaning 'Bobby', then you may start to see how all of this relates to machine code itself. In the next instalment of the Machine Code course, we will look at how binary numbers are used to represent information inside your home computer.