the memory location 1053.

The next stage is that the contents of this address, which is 58, or 00111010 in binary, is placed onto the data bus and 'loaded' into the CPU. Here, because the CPU is expecting an instruction, the byte is interpreted by the control block and causes a very precise sequence of operations to be performed. This particular instruction specifies that the next two bytes in memory will contain 16 bits to be used as a memory location, and that the contents of this location are to be loaded into the CPU's accumulator. As soon as the CPU recognises this instruction, it knows that the next two bytes in memory will specify an address and that the contents of that address will be loaded into the accumulator. It consequently knows that it will not receive another instruction from memory until after these operations have been performed, and that the next instruction will reside in location 1056.

The instruction we are using as an example causes the address bus to be incremented by one, so that the next memory location addressed is 1054. The contents of this location are then put on the data bus and loaded into the CPU. This time, however, they are put into one half of an address register. Having done that, the CPU increments the address bus again so that it now addresses location 1055. The contents of this location go on to the data bus and are similarly loaded into the CPU, this time to be stored in the other half of the address register.

Transferring Numbers

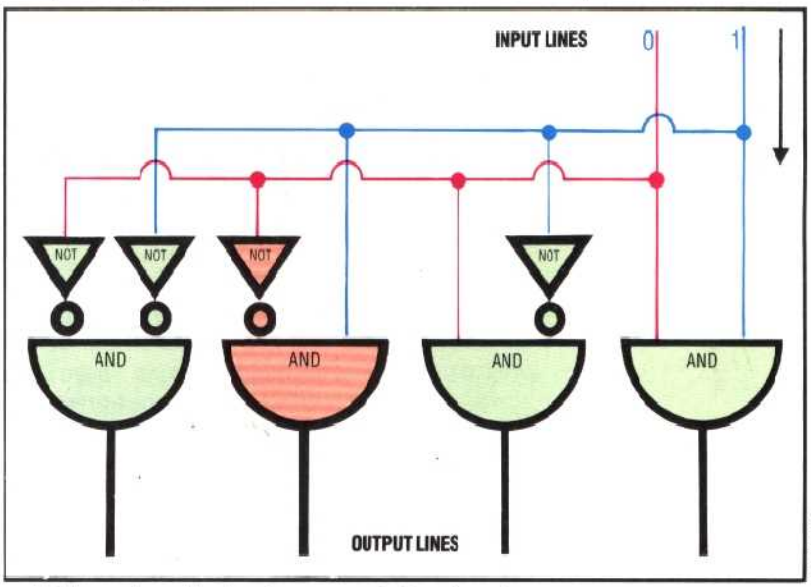
The next stage — and remember that all of these actions happen automatically as a result of the original instruction — is that the numbers in the address register are transferred to the address bus. These numbers, as we can see, are 3071. The memory location now being addressed is therefore 3071. This address (0000101111111111 in binary) is decoded by the address decoder and selects memory cell 3071. The contents of this location, 96, (01100000 in binary) are put onto the data bus and loaded into the CPU. This time, however, the data will be put in the CPU's accumulator. After this the address bus will be set to 1056 and the CPU will expect to find another instruction there.

Now that the CPU has one piece of data in its accumulator, what sort of instruction might be expected to be encountered next? It could be almost anything — CPUs have from a few dozen to a few hundred instructions they recognise, depending on the CPU — but suppose we wanted to invert the data in the accumulator. Inverting means changing each one into a zero and each zero into a one. The instruction to do this would be located at address 1056. On our imaginary CPU, the code for this instruction would be 84. When this number was received by the CPU, the data in the accumulator would be inverted. The number that was in the accumulator was 96 (01100000 in binary). The instruction to invert would cause it to be changed to 10011111 in binary. The instruction to invert a number in the accumulator is a 'one-byte' instruction, so again the CPU would know that the contents of the next memory location, 1057, would again be an instruction rather than data.

This method of addressing a memory location to retrieve a piece of data is only one of several methods available to the programmer. The specific instruction bytes we used in the example (58 to load the accumulator and 84 to invert the contents of the accumulator) are the instructions for our hypothetical CPU, but the same principle applies to all other microprocessor chips. The only difference is that different codes are used for the various instructions and each make of CPU has slightly different 'instruction sets'.

I/O (Input/Output) locations must also have unique addresses, but the principles for addressing them by the CPU are the same. Usually, in eight-bit microprocessors, only eight of the address lines are available for addressing I/O locations, so the maximum number of I/O addresses is 256. This, however, is more than enough for most small computer applications.

Address Decoding
The 16 lines constituting the address bus are capable of uniquely identifying any one of 65,536 separate memory locations. The combination of ones and zeros on the address bus are decoded in address decoders. Part of the decoding is performed by address decoders built up from simple logic gates in chips mounted on the circuit; much of the decoding is performed by equivalent circuits inside the memory chips themselves. The illustration shows how two address lines can be decoded to select one, and only one, of four chips



LIZ DIXON

Address decoding is always needed so that the device selected by the CPU (whether it be a memory location or an I/O location) is made uniquely active while all the other memory or I/O locations remain inactive. This process is called 'enabling'. When there are only a small number of address lines to decode, it is possible to use simple logic gate chips to perform the decoding. The principle of a two-to-four line decoder is shown in the illustration. It is usual to use this type of simple decoding for selecting I/O devices. As the number of address lines increases, however, the complexity of the decoding circuit grows massively. When there are 65,536 separate memory locations that must be individually and uniquely selected, it is usual for most of the address decoding to be performed inside the memory chips.