Decimal	Binary	Hex
0	00000000	0
1	00000001	1
2	00000010	2
3	00000011	3
......	......	......
7	00000111	7
8	00001000	8
9	00001001	9
10	00001010	A
11	00001011	B
12	00001100	C
13	00001101	D
14	00001110	E
15	00001111	F
16	00010000	10
17	00010001	11
......	......	......
24	00011000	18
25	00011001	19
26	00011010	1A
27	00011011	1B
......	......	......
31	00011111	1F
32	00100000	20
33	00100001	21

The range of a single eight-bit byte number, therefore, is eight binary digits, or two hex digits:

```
       0 to 255        in decimal
00000000 to 11111111 in binary
       0 to FF         in hex
```

To convert a hex number into binary, therefore, you simply express each hex digit as a four-bit binary number. If a single-byte number is expressed as a two-digit hex number, then the leftmost hex digit corresponds to the four leftmost binary bits, while the rightmost hex digit corresponds to the four rightmost binary digits. Splitting a byte like this gives us two 'nybbles' (a nybble is half a byte). The leftmost nybble, corresponding to the leftmost hex digit, is called the upper or most significant nybble; and the rightmost nybble is called the lower or least significant nybble. Here is an example:

```
            Upper   Lower
            Nybble  Nybble
               ↓      ↓
206     =   1100   1110   =   C E
 ↑                ↑             ↑
decimal   binary equivalent   hex equivalent
```

It is important to make ourselves as familiar as possible with the hexadecimal number system, for the simple reason that it makes eight-bit byte manipulation much easier than if we were using binary. Convincing yourself of this requires a little practice, not just with number examples, but particularly with memory addresses and the contents of memory bytes. Once this becomes important — and very soon it will — you'll wonder how you ever managed in decimal.

We give programs in this instalment of the Machine Code course, for the BBC Micro, Commodore 64, and the Spectrum, that allow us to look at the contents of specified bytes in memory. These 'Mempeek' programs, as we have called them, ask you first to state the 'Start Address' (i.e. specify the first byte number) and then to give the number of bytes to be looked at. If, for example, you wished to specify byte1953 as your beginning point and request that the contents of the four following bytes be displayed, then the screen will show the decimal number 1953 in the leftmost column, and then list the contents of byte1953, byte1954, byte1955 and byte1956 in the next four columns.

Bear in mind that if the machine shows that byte1956 contains the decimal number 175, what we mean is that in one of the memory chips, an area that the machine calls byte1956 carries a pattern of eight voltage levels. If 0 volts is represented by 0, and 5 volts by 1, then byte1956 carries the voltage pattern 10101111. This we choose to interpret as a binary number, and its decimal equivalent is 175.

It is vital to remember that we use an imprecise kind of shorthand most of the time that we talk about computers; and expanding it into physical description is always salutary, and should help to avoid confusion.

The contents of a byte displayed on the screen are not the 'actual' contents. What we see are character data that have been assigned to the voltage patterns of the bytes. This means that having interpreted the voltage patterns as binary numbers, and having converted the binary to decimal numbers, we are going one step further and converting decimal numbers into characters according to ASCII — the American Standard Code for Information Interchange. This character data is displayed in the last column of the display. This is an internationally recognised code implemented in most computers, which substitutes decimal numbers between 0 and 127 for all the characters on a keyboard (historically, a teleprinter keyboard). In this code the decimal number 65 means the upper-case character 'A', 66 means 'B', 67 means 'C', and so on. Among the non-alphabetic characters, 32 means a space character, 42 means an asterisk, 13 means the Return key, and so on.

The printable ASCII characters start at number 32 and finish at number 127. Codes outside that range are undefined, or not printable, or specific to particular machines. Because of this, when we run the Mempeek programs, the monitor prints a dot to represent any byte containing a number out of range. In the next instalment of the course, we will provide a comprehensive ASCII character set for the values between 0 and 127.

An investigation of the ASCII character set is particularly useful as background to a full understanding of machine code for two important reasons. Firstly, it reinforces the point that how you interpret memory contents is entirely a matter of choice. You can say that a