Byte Order in Memory – Big and Little Endian

Most of the numeric (scalar) values stored in the computer memory consist of more than one byte. On the other hand the memory is organized as ordered collection of bytes, each of which has its own unique address. Let us assume that we have a 32-bit hexadecimal value 44332211 and that it is stored in a 32-bit word in byte-addressable memory at byte location 100. The value consists of four bytes, with the least significant byte containing the value 11 and the most significant byte containing the value 44. There are two ways to store this value in memory:

	Big Endian	**Little Endian**
Memory Address	Value	Value
100	44	11
101	33	22
102	22	33
103	11	44

If the most significant byte is stored in the lowest numerical byte address, this is known as big endian and is equivalent to the left-to-right order of writing in Western culture languages. If the least significant byte is stored in the lowest numerical byte address, this is known as little endian. For a given multibyte scalar value, big endian and little endian are byte-reversed mappings of each other.

The *endianness* problem arises when it is necessary to treat a multiple-byte entity as an atomic or a single data item with a single address, even though it consists of smaller addressable units. Unfortunately, some machines, such as the Intel 80x86, Pentium, VAX, and Alpha, are little-endian machines, whereas others, such as the IBM System 370/390, the Motorola 680xO, Sun SPARC, and most RISC machines, are big endian. This creates problems when a programmer manipulates individual bytes or bits within a multibyte scalar and data are transferred from a machine of one endian type to the other. For example, floating-point numbers occupy 32-bits (or 4 Bytes). If a programmer wants to convert the number to integer, they must be aware of the machine type or the result will wrong.