**1 Memory Operations**

The two basic operations of memory are *Read* and *Write*.

***Read operation***

The Read operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged. To start a Read operation, the processor sends the address of the desired location to the memory and requests that its contents be read. The memory reads the data stored at that address and sends them to the processor.

***Write operation***

The Write operation transfers an item of information from the processor to a specific memory location, overwriting the former contents of that location. To initiate a Write operation, the processor sends the address of the desired location to the memory, together with the data to be written into that location. The memory then uses the address and data to perform the write.

**2 Memory Locations and Addresses**

The memory consists of many millions of storage *cells*, each of which can store a *bit* of information having the value 0 or 1. Because a single bit represents a very small amount of information, bits are seldom handled individually. The usual approach is to deal with them in groups of fixed size. For this purpose the memory is organized so that a group of *n* bits can be stored or retrieved in a single, basic operation. Each group of *n* bits is referred to as a *word* of information, and *n* is called the *word length*. The memory of a computer can be schematically represented as a collection of words, as shown in Figure 1.

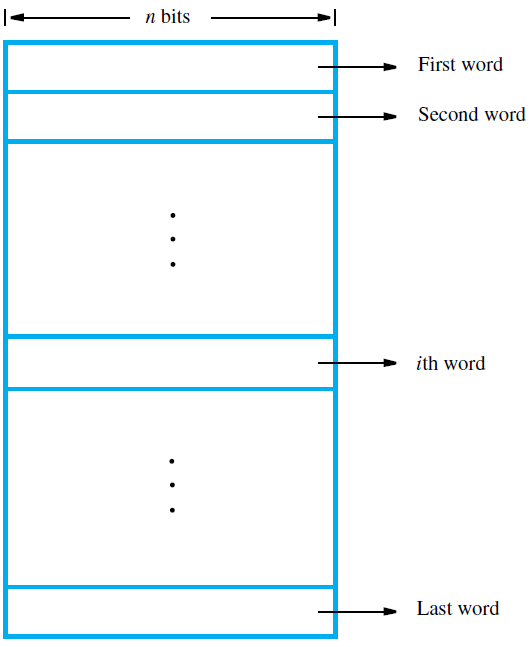


Figure 1: Memory words

Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure 2. A unit of 8 bits is called a *byte*. Machine instructions may require one or more words for their representation. Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or *addresses* for each location.

For example, a 24-bit address generates an address space of 224 (16,777,216) locations. This number is usually written as 16M (16 mega), where 1M is the number 220 (1,048,576). A 32-bit address creates an address space of 232 or 4G (4 giga) locations, where 1G is 230.

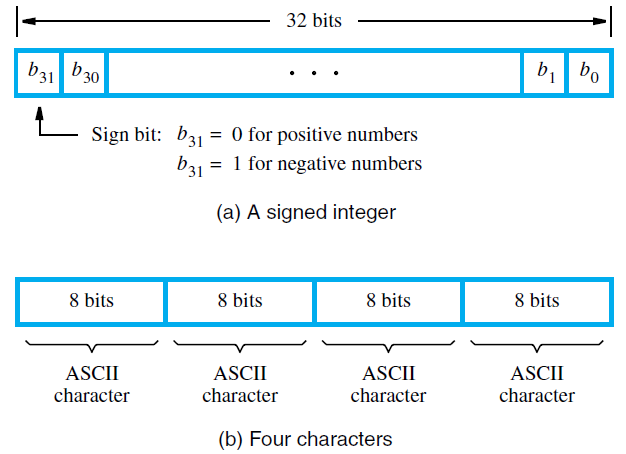


Figure 2: Examples of encoded information in a 32-bit word

**2.1 Byte Addressability**

We have three basic information quantities: bit, byte, and word. A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. This is the assignment used in most modern computers. The term ***byte-addressable memory***is used for this assignment. Byte locations have addresses 0*,* 1*,* 2*, . . . .* Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0*,* 4*,* 8*, . . . ,* with each word consisting of four bytes.

**2.2 Big-Endian and Little-Endian Assignments**

There are two ways that byte addresses can be assigned across words, as shown in Figure 3. The name *big-endian* is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word. The name *little-endian* is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word. The words “more significant” and “less significant” are used in relation to the weights (powers of 2) assigned to bits when the word represents a number. Both little-endian and big-endian assignments are used in commercial machines. In both cases, byte addresses 0*,* 4*,* 8*, . . . ,* are taken as the addresses of successive words in the memory of a computer with a 32-bit word length. These are the addresses used when accessing the memory to store or retrieve a word.

**2.3 Word Alignment**

In the case of a 32-bit word length, natural word boundaries occur at addresses 0*,* 4*,* 8*, . . . ,* as shown in Figure 3. We say that the word locations have *aligned* addresses if they begin at a byte address that is a multiple of the number of bytes in a word. For practical reasons associated with manipulating binary-coded addresses, the number of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0*,* 2*,* 4*, . . . ,* and for a word length of 64 (23 bytes), aligned words begin at byte addresses 0*,* 8*,* 16*, . . . .*

**2.4 Accessing Numbers and Characters**

A number usually occupies one word, and can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte address. For programming convenience it is useful to have different ways of specifying addresses in program instructions.

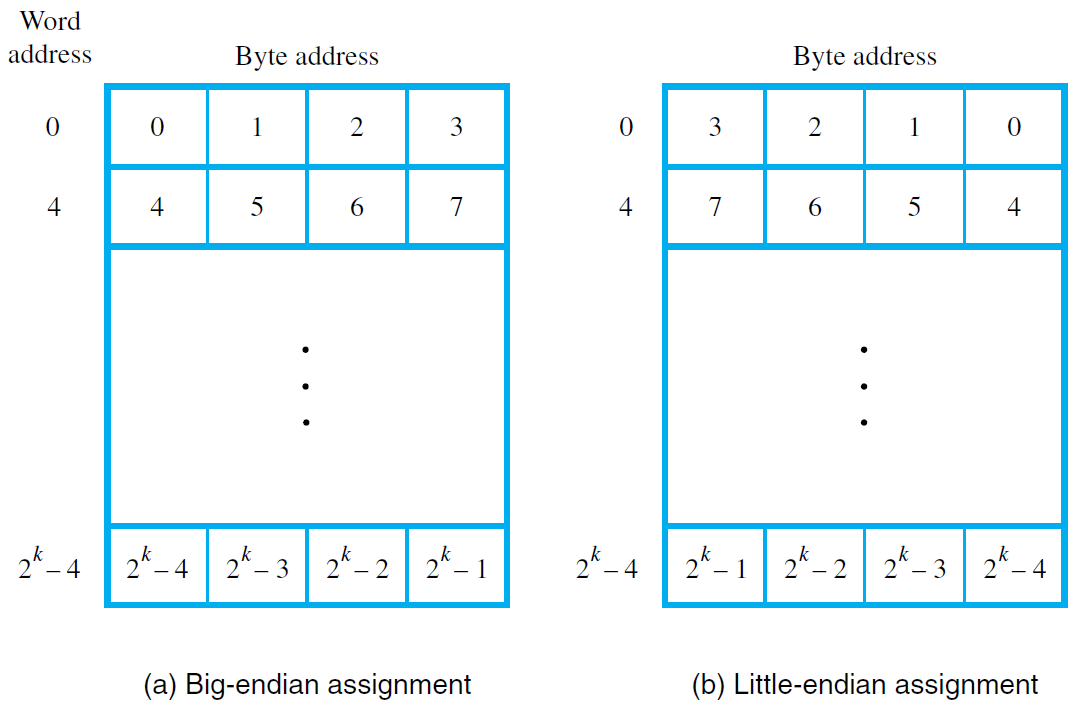


Figure 3: Examples of encoded information in a 32-bit word