3.3 FLOATING-POINT REPRESENTATION

The representation for numbers that we considered previously assumed that the radix point was located in a fixed position, yielding either an integer or a fraction as the interpretation of the internal machine representation. The programmer's responsibility would be to scale numerical operands to fit within a selected word length and then unscale the results to obtain the correct values. Of course, the radix point is not actually stored with the number, but its position must be remembered by the programmer. This method of representation is called *fixed point*.

In practice, the machine value is limited to a finite range which is determined by the number of binary digits used in the representation. For a 32-bit word, the range of signed fixed-point integers is about $+2^{31}$ or $+10^{11}$. Thus, the limited range of fixed-point notation is a drawback for certain applications. Furthermore, arithmetic units operating on fixed-point numbers generally have no capability of rounding results. As discussed in several references in the Further Reading section for this chapter, this limits the usefulness of fixed-point notation in scientific computing.

To overcome many of the limitations of fixed-point notation, a notation that is the counterpart of scientific notation is used for numbers in digital systems. The *floating-point* notation represents a number as a fractional part times a selected base raised to a power. In the machine representation, only the fractional part and the value of the exponent are stored. The decimal equivalent is written as

$$N.n = f \times r^e \tag{3.19}$$

where f is the fraction or *mantissa* and e is a positive or negative integer called the *exponent*. The choice for the base is usually 2, although base 16 is sometimes used.

A number of choices are presented to the designer of a floating-point format. This applies whether the arithmetic operations are carried out by the CPU or its coprocessor directly or by a software package containing routines for floating-point arithmetic. The number of formats is bewildering and only recently has an attempt been made to standardize floating-point arithmetic for computers.

The proposed IEEE standard has been adopted by Motorola for a number of their products. This IEEE standard describes precisely the data formats and other aspects of floating-point arithmetic required to provide consistent operation of a program even when it is executed on different computer systems.

3.3.1 Floating-Point Formats

The typical floating-point format stores the fraction and the exponent together in an m-bit representation. The choice for a fixed-length floating-point format is commonly 32 or 64 bits, referred to as single precision and double precision, respectively. Extended formats with $m \geq 64$ are occasionally used when greater range or precision is required.

Once the length of the floating-point representation is chosen, a number of choices for both the length and the format of the fraction and exponent are possible. Since either or both could be negative as well as positive, a signed fraction and a signed exponent are required. Finally, the interpretation of the bits within the floating-point representation depends on the placement of the fraction and exponent.

Many floating-point formats employ a sign-magnitude representation for the fraction. The most significant bit of the word is reserved for the sign, and this facilitates testing for a positive or negative number. The fraction is generally normalized to yield as many significant digits as possible. Thus, in a base r system, the most significant digit is in the leftmost position in the fraction. For nonzero numbers in the binary system, the leftmost digit will be a 1. As the arithmetic unit or the program shifts the digits in the fraction during arithmetic operations, the exponent is adjusted accordingly. When normalized as a base 2 value, the magnitude of the fraction is

$$0.5 \leq |f| < 1 \tag{3.20}$$

unless the number is zero. The number of digits reserved for the fraction represents a compromise between the precision of the fraction and the range of the exponent. A typical single-precision format (32 bits) might contain an 8-bit exponent and a 23-bit fraction, excluding sign.

An exponent could be represented in two's complement or any other notation that allows signed values. A different alternative, which permits the exponent to be represented internally as a positive number only, is to add an offset value. This value is often called an *excess*. For this format, positive bias or offset is added to all exponents such that the number is read as

$$N.n = fr^{e'-N_b} \tag{3.21}$$

where e' is the actual value of the stored exponent and N_b is the positive offset. For an L-bit exponent in a binary base, the positive number added is usually of the form

$$N_b = 2^{L-1} \tag{3.22}$$

although the IEEE standard format discussed later uses a value of $N_b - 1$.

Example 3.22

One IBM floating-point format has the following representation for a 32-bit word:

(a) sign as the most significant bit (leftmost bit);
(b) next 7 bits as exponent with excess-64 or 40_{16} with radix 16;
(c) next 24 bits as fraction in base 16.

Thus +1.0 is represented as

$$(0.1)_{16} \times 16^1$$

for which the stored exponent becomes 41_{16}. The internal representation is

$$4110\ 0000_{16}.$$

Example 3.23

One PDP-11 (Digital Equipment Corporation) floating-point format uses the following conventions in a 32-bit word:

(a) sign as the most significant bit (leftmost bit);
(b) next 8 bits as exponent in excess-128 notation with radix 2;
(c) next 23 bits as fraction. These 23 bits are derived from a 24-bit fraction always normalized (i.e., the leftmost bit will be a 1 in a nonzero number). The most significant bit of the normalized fraction is not stored since it is always 1.

The representation of +12 is thus

$$0.11_2 \times 2^{132-128}$$

and is represented internally as 1 (HIDDEN)

$$0\ 1000\ 0100\ 1000\ 0000\ 0000\ 0000\ 000_2.$$

Notice that the leading bit of the fraction is a 1 and it is not stored with the floating-point number. This has been termed a *hidden bit* and would be restored by a floating-point hardware processor when the value is processed.

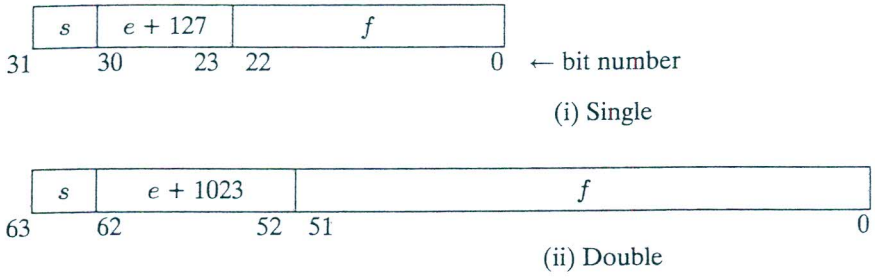
EXERCISES

3.3.1.1. Discuss the various factors that influence the choice of the exponent length, mantissa length, and choice of radix in a floating-point number. Compute the various ranges for a choice of an L-bit exponent in excess notation, a k-bit fraction, and a total length of m bits.

3.3.1.2. Express $1/32$ in binary floating-point representation using an 8-bit excess-128 exponent and a 24-bit fraction with the leading bit implied. The order in the word from left to right is sign, exponent, and then fraction (PDP-11).

3.3.1.3. Convert the numbers indicated to a 32-bit floating-point representation with the characteristics: the leading bit (bit 31) is the sign of the number; the next

Table 3.5 Internal Format by Bit Number



IEEE

9 bits are the exponent in excess-256 notation and following bits are 22 bits of fraction; and a negative number is represented as the integer's two's complement of the positive floating-point number.
(a) +16.0; **(b)** −1.0.

3.3.2 Standard Floating-Point Format

Although the Motorola MC68332 processor does not provide floating-point instructions, many Motorola products support the standard floating-point format proposed by the IEEE. These products include software routines, routines in read-only memory, and a coprocessor chip to support floating-point arithmetic for the M68000 family processors.

The basic format allows a floating-point number to be represented in single or 32-bit format as

$$N.n = (-1)^S 2^{e'-127}(1.f) \tag{3.23}$$

where S is the sign bit, e' is the biased exponent, and f is the fraction stored normalized without the leading 1. Internally, the exponent is 8 bits in length and the stored fraction is 23 bits long. A double-precision format allows a 64-bit representation with a 11-bit exponent and a 52-bit fraction. Table 3.5 shows the format pictorially.

Various features of these floating-point formats are presented in Table 3.5 and Table 3.6. Other formats called extended-precision formats are presented in the references in the Further Reading section of this chapter.

Example 3.24

The numbers +1.0, +3.0, and −1.0 have the following representation in the standard 32-bit format:

(a) Since $1.0 = 1.0 \times 2^0$, the internal value is $3F80\ 0000_{16}$.
(b) Since $3.0 = 1.5 \times 2^1$, the exponent is 128 and the fraction is 1.100_2 without the leading 1. Thus, the internal value is $4040\ 0000_{16}$.
(c) Since $-1 = -1.0 \times 2^0$, the result requires the sign bit to be 1 in the representation $BF80\ 0000_{16}$.

Table 3.6 IEEE Standard Floating-Point Notation

	Single	Double
Length in bits		
Sign	1	1
Exponent	8	11
Fraction	23 + (1)	52 + (1)
Total	$m = 32 + (1)$	$m = 64 + (1)$
Exponent		
Max e'	255	2047
Min e'	0	0
Bias	127	1023

Note:
Fractions are always normalized and the leading 1 (hidden bit) is not stored.

EXERCISES

3.3.2.1. Write the internal machine representation for the following numbers in the standard floating-point single-precision format:
(a) 0.5; **(b)** -0.5; **(c)** 2^{-126}.

3.3.2.2. What is the decimal value of the largest positive number that may be represented in single-precision standard format if the biased exponent is limited to a maximum of 254 when a valid number is being represented? (The value 255 is reserved for special operands.)

3.3.2.3. Express the following numbers in the internal representation using standard double-precision floating-point format:
(a) 7.0; **(b)** -30.