The IEEE 754 standard offers four rounding modes, so that the programmer can use the desired approximation. The rounding modes are the following:

- Round toward plus infinity (always round up);
- Round toward minus infinity (always round down);
- Round toward zero (truncate);
- Round to nearest even.

Two extra bits (guard and round) are always enough for the first three rounding modes. The last rounding mode is provided for the situations when the actual number is exactly halfway between two floating-point representations. To always obtain the correct rounding for the last case, the standard specifies a third bit in addition to guard and round. This is called *sticky* bit (*s*) and it is set whenever there are non-zero bits to the right of the round bit. Once set, the sticky bit remains set. This bit might be set, for example, during addition, when the smaller number is shifted to the right.

Rounding to nearest minimizes the mean error introduced by rounding. If uniform distribution of digits are assumed, rounding to nearest has a mean error of 0.

3.5. Problems

3.5.1. Design combinational circuits for a half subtracter and a full subtracter. For each circuit: (*a*) Derive the truth table; (*b*) Write the Boolean expression; (*c*) Draw the logic diagram.

3.5.2. Design an *n*-bit subtracter whose operation is analogous to that of a ripple carry adder.

3.5.3. Illustrate how to connect *n* half adders to form an *n*-bit combinational incrementer whose function is to add 1 (modulo 2^n) to an *n*-bit number X.

3.5.4. Design a 2's complement adder-subtracter that computes the values $X + Y$, $X – Y$, and $Y – X$. The circuit has an overflow flag and a zero flag. First design a 4-bit adder module using ripple-carry propagation, and then use this module to construct an 8-bit adder-subtracter.

3.5.5. Table 3.13 describes a procedure for the addition of *n*-bit sign-magnitude numbers. Addition and subtraction of sign-magnitude numbers is complicated by the fact that to compute $X + Y$, the magnitudes $|X|$ and $|Y|$ must be compared to determine the operation to perform and the order of operands. Design a circuit to compute the functions $X + Y$, $X – Y$, and $Y – X$ for sign-magnitude numbers. Assume that the word size is 16 bits and that the standard design components are available, including a 16-bit unsigned adder, a 16-bit unsigned subtracter, and a 16-bit magnitude comparator.

Table 3.13. Algorithm for the addition of sign-magnitude numbers.

Condition	Operations
$X > 0, Y > 0$	Add $X = x_{n-1}x_{n-2}...x_0$ and $Y = y_{n-1}y_{n-2}...y_0$ (modulo 2^n) to form the result $Z = z_{n-1}z_{n-2}...z_0$ (n-bit unsigned addition).
$X > 0, Y < 0$	Let $\|X\| = x_{n-2}x_{n-3}...x_0$ and $\|Y\| = y_{n-2}y_{n-3}...y_0$. If $\|X\| < \|Y\|$, subtract X from Y (modulo 2^n). If $\|X\| \geq \|Y\|$, set y_{n-1} to 0 and subtract Y from X (modulo 2^n).
$X < 0, Y > 0$	If $\|Y\| < \|X\|$, subtract Y from X (modulo 2^n). If $\|Y\| \geq \|X\|$, set x_{n-1} to 0 and subtract X from Y (modulo 2^n).
$X < 0, Y < 0$	Add X and Y (modulo 2^n) and set z_{n-1} to 1.

3.5.6. Suppose that the adder-subtracter circuit of Figure 3.32 has been designed for 2's complement numbers. The circuit computes the sum $Z = X + Y$ when the control line $SUB = 0$ and the difference $Z = X - Y$ when $SUB = 1$. An overflow flag OF is to be added to the circuit, but it is not possible to access internal lines. That is, only those data and control lines appearing in the figure can be used to set the overflow flag. Construct a logic circuit for OF.

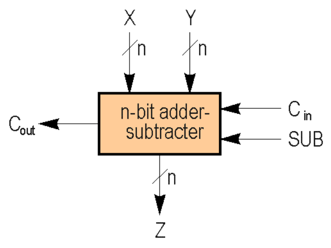

Figure 3.32. An *n*-bit adder-subtracter circuit.

3.5.7. Consider the adder-subtracter circuit of Figure 3.32, assuming now that it has been designed for sign-magnitude numbers. The circuit computes $Z = X + Y$ when $SUB = 0$ and $Z = X - Y$ when $SUB = 1$. Assume that the circuit contains an *n*-bit ripple carry adder and a similar *n*-bit ripple borrow subtracter, and that it is possible to access all the internal lines. Derive a logic equation that defines an overflow flag OF for this circuit.

3.5.8. Derive a general expression for the carry signal C_k in a carry lookahead adder.

3.5.9. Derive the Boolean expressions for the carry signals C_{12} and C_{16} in a carry lookahead adder that adds two 16-bit numbers. Assume that fan-in is limited to 4.

3.5.10. Write the equations for the carry logic of a 64-bit carry lookahead adder using 16-bit adders as building blocks.

3.5.11. Design a 4-bit carry select adder using full adders and multiplexers.

3.5.12. Design a carry save adder for adding four 4-bit numbers. Draw the logic diagram of the adder using full adders. Note that the sum of four *n*-bit numbers can take $n+2$ bits.

3.5.13. Assume that the delay introduced by each full adder is $2T$. Calculate the time of adding four 4-bit numbers using three ripple-carry adders versus the time using the carry save adder from Problem 3.5.12.

3.5.14. Design a serial adder that adds four unsigned binary numbers instead of two handled by a conventional serial adder. The adder has four input lines x, y, z, u and a single output line S. The numbers are presented at the inputs with the least significant bits first.

3.5.15. Design a serial subtracter analogous to the serial adder. The inputs of the subtracter are two unsigned binary numbers x and y; the output is the difference $x - y$.

3.5.16. Design a sequential circuit that multiplies an unsigned binary number N of arbitrary length by 3. N is entered serially via the input line x with its least significant bit first. The result appears serially on the output line z.

3.5.17. Design a BCD adder by using five full adders, two half adders, three two-input NAND gates, and one three-input NAND gate.

3.5.18. A faster version of Booth's multiplication algorithm for 2's complement numbers, known as the *modified Booth algorithm*, examines three adjacent bits $y_{i+1}y_iy_{i-1}$ of the multiplier Y at a time, instead of two bits. Besides the three basic actions performed by the original Booth algorithm, which can be expressed as add 0, X, or $-X$ to the accumulated partial product (A), the modified algorithm performs two more actions: add $+2X$ or $-2X$ to register A. These operations have the effect of increasing the radix from 2 to 4 and allow an n-bit multiplication to be performed in $n/2$ clock cycles instead of n. (*a*) Construct a table that defines the basic actions of the modified Booth algorithm as a function of $y_{i+1}y_iy_{i-1}$. (*b*) Show the steps of the algorithm for two 8-bit 2's complement numbers (including the sign bit). (*c*) Describe the modified Booth algorithm in VHDL.

3.5.19. Draw the detailed logic diagram of a 4-bit multiplier using a Wallace tree (Figure 3.15). Use full adders as basic components.

3.5.20. Draw the block diagram of a multiple-operand adder with multiple levels of carry save adders and one carry lookahead adder, arranged in a Wallace tree. The adder can receive 10 input numbers, where each number has 4 bits. Assuming that each carry save adder has a delay of T, and the carry lookahead adder has a delay of $2T$, estimate the total time required to add 100 4-bit numbers.

3.5.21. Suppose the combinational array multiplier of Figure 3.18 is given the unsigned integer operands X = 1010 and Y = 1001. Determine the output signals generated by every adder cell when the array computes $X \times Y$.

3.5.22. Use the multiplier cell of Figure 3.19 to construct a combinational array multiplier for 5-bit unsigned numbers. Draw the logic diagram of the multiplier and show all the signals applied to every cell.

3.5.23. Design a basic cell for implementing Booth's algorithm by a combinational array. Such a cell must be capable of addition, subtraction, and no operation. Draw the structure of the array multiplier for 4×4-bit numbers. Suppose the array multiplier is given the signed integer operands X = 1010 and Y = 1001. Determine the output signals generated by every cell when the array computes $X \times Y$.

3.5.24. Divider circuits usually include logic to detect a dividend-divisor combination that will cause the quotient to overflow. Suppose that a divider circuit for n-bit unsigned integers has a double-word (2n-bit) dividend and a single-word divisor. (*a*) What condition must be satisfied for quotient overflow to occur? (*b*) How can be modified the divider circuit of Figure 3.23 to introduce an overflow detector using as little extra logic as possible?

3.5.25. Write a nonrestoring division algorithm using the notation of Figure 3.22. Show by an example how the algorithm works.

3.5.26. Write the logic equations for the outputs z and u of cell D in Figure 3.25.

3.5.27. Consider the array divider of Figure 3.26 that is designed for a word size of n = 3, with a double-length (6 bit) dividend. (*a*) Why there are four rows of D cells instead of three? (*b*) Supposing that dividends are restricted to 3 bits instead of 6, which cells can be deleted from the array?

3.5.28. Consider the following 9-bit floating-point format, where the 4-bit exponent is biased by 8.

8	7 4	3 0
S	Exponent	Mantissa

(*a*) Assume that the exponent base is 2, and consider only normalized representations. What are the largest positive value and the smallest negative value representable in this format? (*b*) Repeat part (*a*) when the exponent base is 4.

3.5.29. Add the numbers 6.42×10^1 to 9.51×10^2, assuming that there are only three significant decimal digits, first with guard and round digits, and then without them.

3.5.30. Derive the floating-point division algorithm using a format similar to the multiplication algorithm in Figure 3.31. Then perform the division of numbers 1.11010×10^{10} and 1.10010×10^{-5}, showing the steps of the operation.