Computer Organization and Architecture

Chapter 3: Arithmetic

Engineering Notes — B.Tech / B.E. Computer Science & Engineering

---

The arithmetic unit of a computer is the part of the Central Processing Unit (CPU) that performs all arithmetic operations such as addition, subtraction, multiplication, division and floating-point calculations. Since digital computers work internally with binary numbers, all these operations are implemented using registers, adders, shifters and control logic built from simple digital circuits. This chapter covers the design and working of the addition/subtraction unit, the multiplication of positive and negative numbers (including Booth's algorithm), restoring and non-restoring division, and floating-point addition/subtraction.

3.1 Addition / Subtraction Unit — Block Diagram and Function

A digital computer generally uses a single common hardware circuit to perform both addition and subtraction, instead of building two separate circuits. This is possible because subtraction can always be converted into addition by taking the 2's complement of the number being subtracted. If A and B are two binary numbers, then A − B can be computed as A + (2's complement of B). This trick allows the same parallel adder to be reused for both operations, which saves hardware and simplifies the control logic.

Working Principle

A mode (control) signal, usually called M, decides whether the circuit performs addition or subtraction. Each bit of B is passed through an XOR gate together with the mode signal M before entering the adder. This arrangement is often called an XOR-based complementer:

- When M = 0: B XOR 0 = B, and Cin = 0. The circuit simply adds A and B, giving A + B.
- When M = 1: B XOR 1 = B' (1's complement of B), and Cin = 1 is applied to the adder's carry-in. Adding the 1's complement of B and then adding 1 through the carry-in produces the 2's complement of B, so the adder computes A + (2's complement of B) = A − B.

Block Diagram

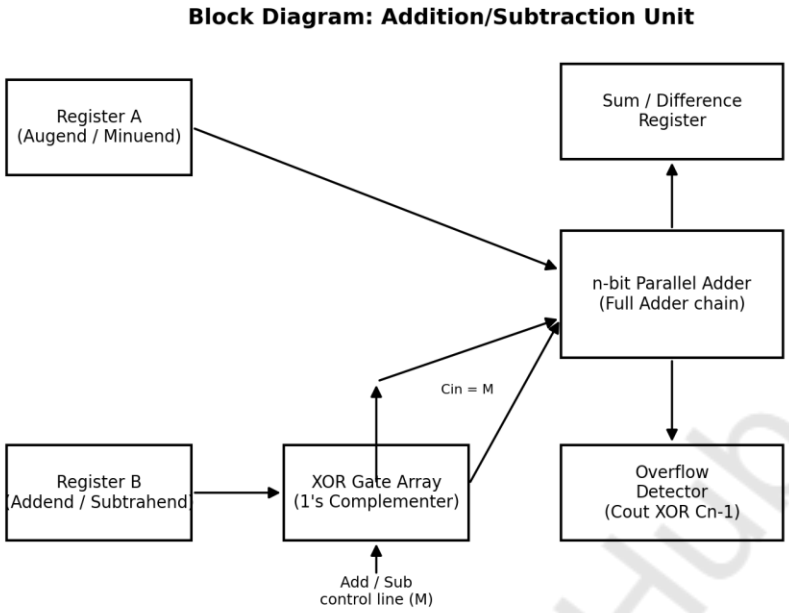

Fig 3.1: Block diagram of a combined Addition/Subtraction unit

Components and Their Function

- Register A and Register B: hold the two operands on which addition or subtraction is to be performed.
- XOR Gate Array: acts as a controlled complementer; it either passes B unchanged or produces its 1's complement, depending on the mode signal M.
- Mode signal (M): a single control line — M = 0 selects addition, M = 1 selects subtraction. M is also fed as the carry-in (Cin) of the adder.
- n-bit Parallel Adder: a chain of full adders that adds register A with the (possibly complemented) value of B, along with the carry-in.
- Sum/Difference Register: stores the final result of the operation — the sum in case of addition, or the difference in case of subtraction.
- Overflow Detector: formed by XOR-ing the carry into the most significant bit (MSB) with the carry out of the MSB; if these two carries differ, an overflow has occurred.

This single reusable circuit — one adder plus one XOR array plus one control line — is the standard way arithmetic/logic units (ALUs) implement addition and subtraction together in real processors.

3.2 Multiplication Circuit Diagram and Multiplication of Positive Numbers

Multiplication of binary numbers is carried out in digital hardware using the shift-and-add method, which mirrors the manual (pencil-and-paper) technique of multiplication but adapted to binary digits. Instead of using n partial-product rows the way we do on paper, the hardware accumulates the partial products one at a time into a single accumulator register, shifting after every step.

Circuit / Hardware Block Diagram

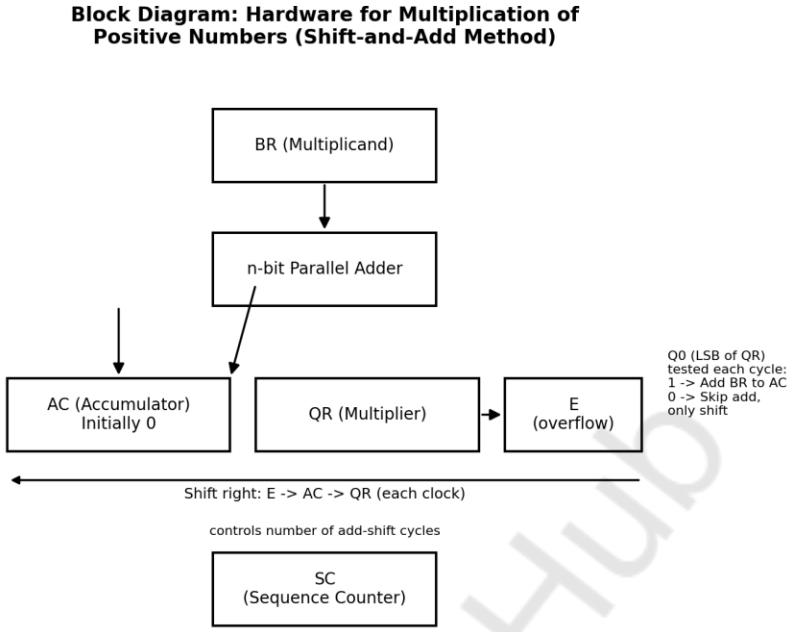

Fig 3.2: Hardware block diagram for binary multiplication (positive numbers)

Registers and Their Role

- BR (Multiplicand Register): holds the multiplicand throughout the process.
- QR (Multiplier Register): holds the multiplier; its least significant bit, Q0, is tested at every cycle.
- AC (Accumulator): initialized to 0; accumulates the partial sums.
- E (overflow/extra flip-flop): stores the carry-out generated when AC and BR are added; it also takes part in the shifting operation.
- SC (Sequence Counter): initialized to n (the number of bits); it is decremented after each cycle, and the process stops when SC = 0.

Algorithm for Multiplying Two Positive (Unsigned) Numbers

1. Set AC = 0, E = 0, and load BR = multiplicand, QR = multiplier; set SC = n.
2. Examine the least significant bit of QR, i.e., Q0.
3. If Q0 = 1, add BR to AC and place the carry-out in E. If Q0 = 0, do nothing (no addition).
4. Shift E, AC and QR together one bit to the right (this passes E into the MSB of AC, and the LSB of AC into the MSB of QR). Set E = 0 for the next cycle.
5. Decrement SC by 1.
6. If SC ≠ 0, repeat from step 2; otherwise stop — the final product (2n bits) is held jointly in AC (most significant half) and QR (least significant half).

Worked Example: Multiply 11 × 13 (Positive Numbers)

Let multiplicand BR = 1011 (11 in decimal) and multiplier QR = 1101 (13 in decimal), with 4-bit registers, so SC starts at 4. AC and E are initialised to 0.

Cycle	Q0	Operation	AC	QR	SC
Initial	1	—	0000	1101	4
1	1	AC = AC + BR, then shift right	0101	1110	3
2	0	No add, only shift	0010	1111	2
3	1	AC = AC + BR, then shift right	0110	1111	1
4	1	AC = AC + BR (carry to E), then shift right	1000	1111	0

Table 3.1: Step-by-step trace of shift-and-add multiplication (11 × 13)

At the end of the fourth cycle, SC = 0 and the process stops. The final product is the combined content of AC and QR: AC:QR = 1000 1111, which is 143 in decimal. This matches the expected result, since 11 × 13 = 143.

3.3 Multiplication of Negative Numbers — Booth's Algorithm

The shift-and-add method described in Section 3.2 works only for unsigned (positive) numbers. When numbers are represented in 2's complement form and can be negative, a more general technique called Booth's Algorithm is used. Booth's algorithm can multiply two signed numbers directly, in 2's complement form, without converting them to positive numbers first, and it also gives a speed advantage when a multiplier has long runs of consecutive 1s or 0s.

Concept Behind Booth's Algorithm

Booth's algorithm examines two adjacent bits of the multiplier at a time: the current bit Qn and the bit to its right, Qn+1 (initially set to 0). Based on the pattern formed by these two bits, the algorithm decides whether to add the multiplicand, subtract the multiplicand, or simply shift, as summarised below:

- Qn Qn+1 = 00: string of 0s continues — no arithmetic operation, only shift.
- Qn Qn+1 = 01: end of a run of 1s — add the multiplicand (BR) to AC, then shift.
- Qn Qn+1 = 10: beginning of a run of 1s — subtract the multiplicand (BR) from AC, then shift.
- Qn Qn+1 = 11: string of 1s continues — no arithmetic operation, only shift.

After every cycle, an arithmetic shift right (ASR) is performed on the combined AC, QR and Qn+1, in which the sign bit of AC is preserved (replicated) while shifting, so that the sign of the running partial result is not disturbed.

Flowchart of Booth's Algorithm

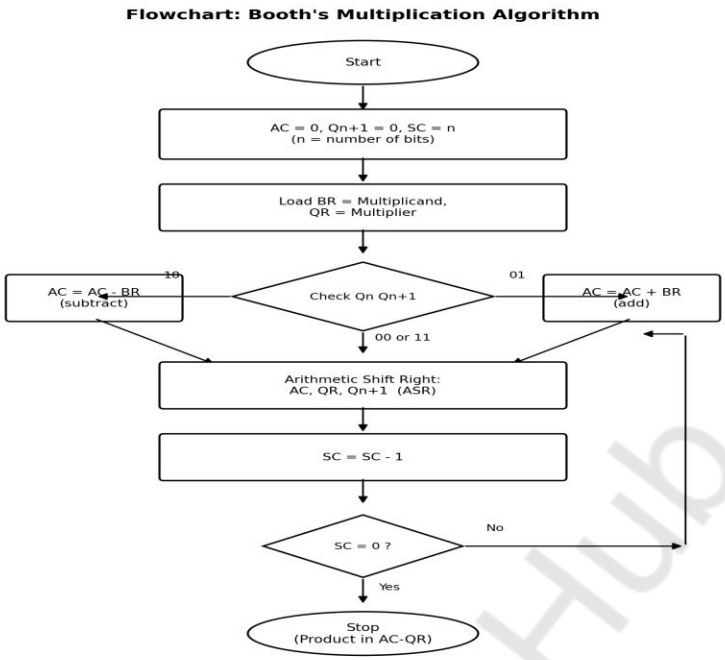

Fig 3.3: Flowchart of Booth's Multiplication Algorithm

Worked Example: Multiply (−7) × (3) Using Booth's Algorithm

Let the multiplicand BR = 1001 (which represents −7 in 4-bit 2's complement) and the multiplier QR = 0011 (which represents 3). Then −BR (2's complement of BR) = 0111. AC = 0000, Qn+1 = 0, and SC = 4.

Cycle	Qn Qn+1	Operation	AC	QR	Qn+1
Initial	1 0	—	0000	0011	0
1	1 0	AC = AC − BR, then ASR	0011	1001	1
2	1 1	No operation, only ASR	0001	1100	1
3	0 1	AC = AC + BR, then ASR	1101	0110	0
4	0 0	No operation, only ASR	1110	1011	0

Table 3.2: Step-by-step trace of Booth's algorithm for (−7) × (3)

After 4 cycles, SC = 0 and the process stops. The final product is AC:QR = 1110 1011, which is the 8-bit 2's complement representation of −21. This is correct, since (−7) × 3 = −21.

3.4 Restoring and Non-Restoring Division

Division in digital hardware is implemented using repeated subtraction combined with shifting, similar to the long-division method used on paper. Two common hardware algorithms for binary division are the Restoring Division algorithm and the Non-Restoring Division algorithm.

3.4.1 Restoring Division

In restoring division, after every trial subtraction of the divisor from the partial remainder, the sign of the result is checked. If the result becomes negative, the divisor is immediately added back (restored) to undo the subtraction, and the corresponding quotient bit is set to 0. If the result stays non-negative, no restoration is needed and the quotient bit is set to 1. This method always leaves register A holding the correct, restored value before the next shift.

Steps of Restoring Division

1. Initialise A = 0 (remainder register), load Q = dividend, M = divisor, and set the repeat counter to n (number of bits).
2. Shift the combined register A, Q one bit to the left.
3. Subtract the divisor: A = A − M.
4. Check the sign of A. If A is negative, restore it by adding back M (A = A + M) and set Q0 = 0. If A is non-negative, keep A as is and set Q0 = 1.
5. Decrement the repeat counter.
6. If the counter is not zero, repeat from step 2; otherwise stop. Q holds the quotient and A holds the remainder.

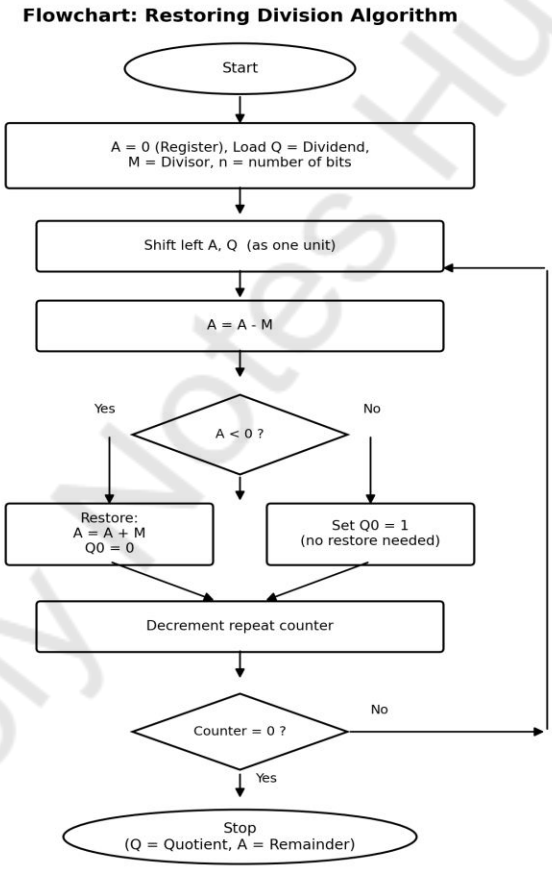

Fig 3.4: Flowchart of the Restoring Division Algorithm

Worked Example: Divide 11 by 3 (Restoring Division)

Let dividend Q = 1011 (11), divisor M = 0011 (3), A = 0000, with 4-bit registers, so the process runs for 4 cycles.

Cycle	A after shift	A = A − M	Decision	Final A	Q
1	0001	1 − 3 = −2	Negative → restore, Q0 = 0	0001	0110
2	0010	2 − 3 = −1	Negative → restore, Q0 = 0	0010	1100
3	0101	5 − 3 = 2	Non-negative → Q0 = 1	0010	1001

Cycle	A after shift	A = A − M	Decision	Final A	Q
4	0101	5 − 3 = 2	Non-negative → Q0 = 1	0010	0011

Table 3.3: Step-by-step trace of restoring division (11 ÷ 3)

At the end, Q = 0011 (quotient = 3) and A = 0010 (remainder = 2), which is correct since 11 ÷ 3 gives quotient 3 and remainder 2.

3.4.2 Non-Restoring Division

Non-restoring division avoids the wasted restoring (add-back) step of the restoring method. Instead of always subtracting and then possibly restoring, it alternates the operation based on the sign of the previous remainder: if the previous remainder was non-negative, it subtracts the divisor in the current step; if the previous remainder was negative, it adds the divisor instead. This way, no extra restoring addition is required inside the loop, and only a single correction step (if needed) is performed at the very end.

Steps of Non-Restoring Division

1. Initialise A = 0, load Q = dividend, M = divisor, and set the repeat counter to n.
2. Shift the combined register A, Q one bit to the left.
3. If the previous value of A was non-negative, compute A = A − M; if the previous value of A was negative, compute A = A + M.
4. Check the sign of the new A. If A ≥ 0, set Q0 = 1; if A < 0, set Q0 = 0.
5. Decrement the repeat counter and repeat from step 2 until the counter reaches 0.
6. After the loop ends, if the final A is negative, perform one last correction: A = A + M, to obtain the true remainder.

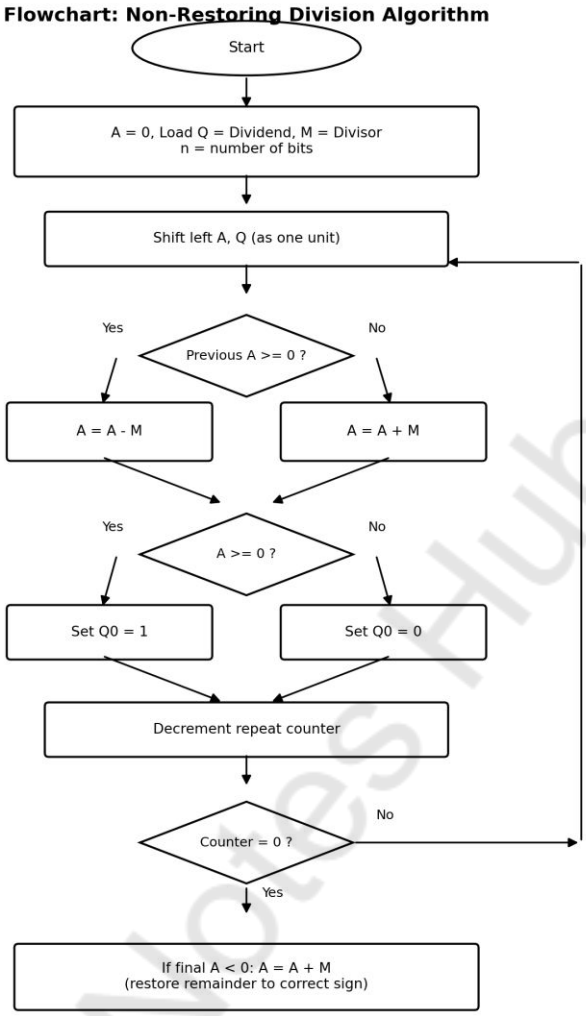

Fig 3.5: Flowchart of the Non-Restoring Division Algorithm

Worked Example: Divide 11 by 3 (Non-Restoring Division)

Cycle	Prev. sign of A	A after shift	Operation	New A	Q0	Q
1	A ≥ 0	0001	A = A − M = 1 − 3	1110 (−2)	0	0110
2	A < 0	1100 (−4)	A = A + M = −4 + 3	1111 (−1)	0	1100
3	A < 0	1111 (−1)	A = A + M = −1 + 3	0010 (2)	1	1001
4	A ≥ 0	0101 (5)	A = A − M = 5 − 3	0010 (2)	1	0011

Table 3.4: Step-by-step trace of non-restoring division (11 ÷ 3)

The final remainder A = 0010 (2) is already non-negative, so no end correction is required. Q = 0011 (quotient = 3), which again matches 11 ÷ 3 = 3 remainder 2.

Restoring vs Non-Restoring Division

Aspect	Restoring Division	Non-Restoring Division
Correction step	Adds back divisor immediately whenever remainder is negative	No immediate correction; alternates add/subtract; at most one correction at the very end

Aspect	Restoring Division	Non-Restoring Division
Number of operations per cycle	Up to two (subtract, then possibly restore)	Exactly one (either add or subtract)
Speed	Slower, due to possible extra restoring addition	Faster, since only one arithmetic operation per cycle
Hardware/control complexity	Simpler control logic	Slightly more complex control logic (must remember previous sign)

3.5 Floating Point Addition / Subtraction — Algorithm and Flowchart

Floating-point numbers are represented using three fields: a sign bit, an exponent, and a mantissa (also called the significand), typically following a standard such as IEEE 754. Because two floating-point numbers may have different exponents, they cannot be added or subtracted directly the way fixed-point (integer) numbers are; their exponents must first be made equal before the mantissas can be combined.

Algorithm for Floating-Point Addition/Subtraction

1. Read the two floating-point operands, A (exponent Ea, mantissa Ma) and B (exponent Eb, mantissa Mb).
2. Compare the exponents Ea and Eb. If they are unequal, align them: shift the mantissa of the number with the smaller exponent to the right by the difference in exponents, and increase that exponent to match the larger one.
3. Once the exponents are equal, add or subtract the mantissas (depending on the operation and the signs of the operands) to obtain the result mantissa Mr, keeping the common exponent E.
4. Check the result mantissa for overflow (i.e., it needs one more bit than the allotted mantissa width). If it has overflowed, shift Mr right by one bit and increment the exponent E by one.
5. Check whether the result mantissa needs normalizing (its most significant bit is 0, i.e., there is a leading zero). If so, shift Mr left and decrement E, repeating until the mantissa is normalized (MSB = 1) or the mantissa becomes zero.
6. If the result mantissa has become zero, force the exponent to zero as well, so that the result is represented as true (clean) zero.
7. Check the exponent for overflow (too large to represent) or underflow (too small/negative to represent) and report an exception if either occurs.
8. If no exponent exception occurred, round the result mantissa to the number of bits allowed by the format, and output the final result as the pair (exponent E, mantissa Mr).

Flowchart

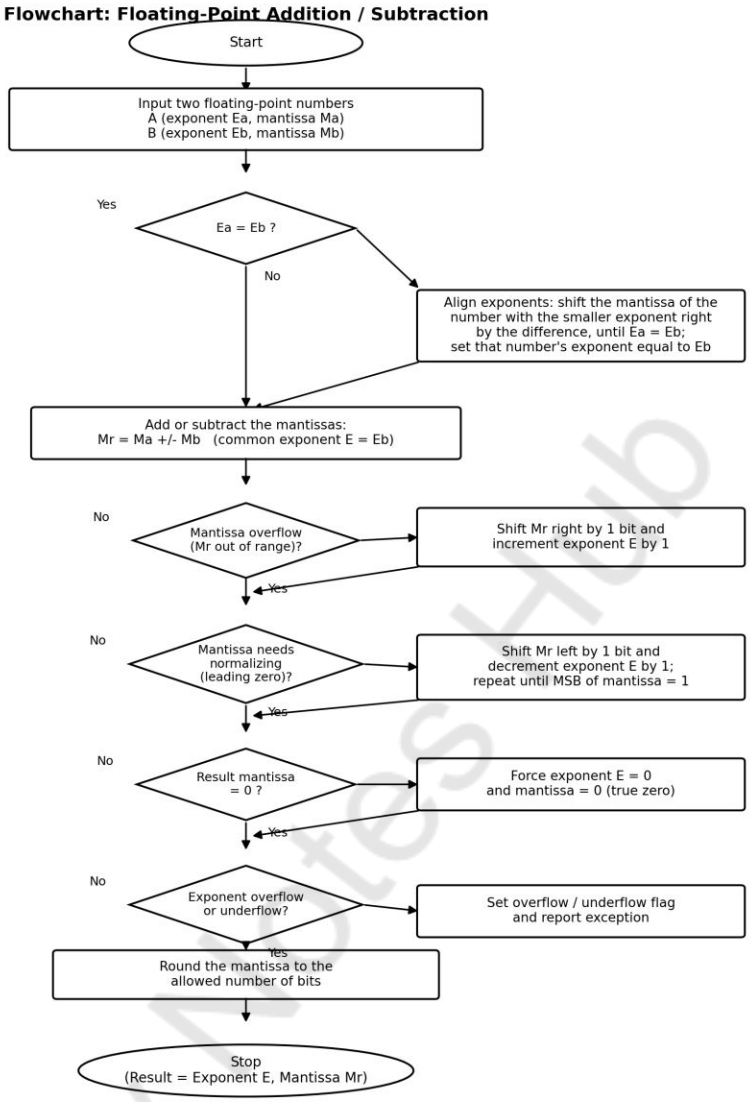

Fig 3.6: Flowchart of Floating-Point Addition / Subtraction

Note: As per the scope of this topic, only the algorithm and flowchart are covered here; a worked numerical example is not included.

Multiple Choice Questions (MCQs)

Answer the following multiple-choice questions. The answer key is provided at the end of this section.

1. Subtraction of B from A in a combined addition/subtraction unit is performed by adding A to:

A) B directly
B) the 1's complement of B
C) the 2's complement of B
D) the 9's complement of B

2. Overflow in a parallel adder is detected by XOR-ing:

A) the MSBs of A and B
B) the carry into the MSB and the carry out of the MSB
C) the LSB of the sum and the carry-in
D) the LSBs of both operands

3. In the hardware multiplier block diagram, the register that holds the multiplicand is:

A) AC
B) QR
C) BR
D) SC

4. In the shift-and-add multiplication algorithm, if Q0 = 0, the action taken is:

A) AC = AC + BR
B) AC = AC − BR
C) Only shift, no addition
D) Halt immediately

5. Booth's algorithm is primarily used for multiplying:

A) only positive numbers
B) only unsigned numbers
C) signed numbers represented in 2's complement form
D) BCD-coded numbers

6. In Booth's algorithm, when the bit pair Qn Qn+1 = 10, the operation performed is:

A) AC = AC + BR
B) AC = AC − BR
C) No operation, only shift
D) AC is reset to 0

7. In Booth's algorithm, the bit Qn+1 represents:

A) the sign bit of AC
B) an extra flip-flop that stores the previous value of Q0
C) the carry flag of the adder
D) the sequence counter

8. When Qn Qn+1 = 00 or 11 in Booth's algorithm, the action taken is:

A) Add the multiplicand
B) Subtract the multiplicand
C) No arithmetic operation, only arithmetic shift right
D) Stop execution

9. In restoring division, if the partial remainder A becomes negative after subtraction, we:

A) set Q0 = 1 and continue without change
B) add the divisor back (restore) and set Q0 = 0
C) halt the process with an error
D) subtract the divisor a second time

10. Non-restoring division improves on restoring division mainly by:

A) skipping the shift step entirely
B) never performing subtraction
C) avoiding the extra restoring addition inside the loop by alternating add/subtract
D) converting all operands to floating point

11. In non-restoring division, if the final remainder is negative, we:

A) leave it unchanged as the final answer
B) add the divisor once more to correct it
C) multiply the remainder by −1
D) subtract the divisor once more

12. In floating-point representation, normalizing the mantissa means ensuring that:

A) it becomes zero
B) it has no fractional part
C) its most significant bit is non-zero (typically 1)
D) it is equal to the exponent

13. Before adding the mantissas of two floating-point numbers, it is necessary to first:

A) normalize the result
B) align the exponents of both numbers
C) round the mantissa
D) complement the mantissa

14. If the mantissa overflows after addition in floating-point arithmetic, the correction applied is:

A) shift mantissa left and decrement the exponent
B) shift mantissa right and increment the exponent
C) set the entire result to zero
D) report an error and halt immediately

15. The Sequence Counter (SC) used in the multiplication/division hardware algorithms is used to:

A) store the partial product
B) count the number of add/subtract-and-shift cycles, equal to the number of bits
C) detect arithmetic overflow
D) hold the sign bit of the result

Answer Key

1 — C 2 — B 3 — C 4 — C 5 — C

6 — B 7 — B 8 — C 9 — B 10 — C

11 — B 12 — C 13 — B 14 — B 15 — B

Broad / Long-Answer Questions

1. Explain the block diagram of a combined addition/subtraction unit and describe how a single adder circuit is used to perform both addition and subtraction.

2. Describe, with a neat block diagram, the hardware algorithm used for multiplying two unsigned (positive) binary numbers.

3. Using the shift-and-add method, multiply two 4-bit positive binary numbers of your choice, showing the accumulator, multiplier register and sequence counter at every step.

4. What is Booth's algorithm? Explain why it is needed for multiplying signed numbers represented in 2's complement form.

5. Draw and explain, step by step, the flowchart of Booth's multiplication algorithm.

6. Illustrate Booth's algorithm with a suitable numerical example involving the multiplication of a positive number by a negative number.

7. Explain the restoring division algorithm in detail, along with its flowchart and a suitable numerical example.

8. Explain the non-restoring division algorithm and compare it with the restoring division algorithm in terms of the number of operations required.

9. Explain the steps involved in floating-point addition/subtraction with the help of a neat flowchart.

10. Compare restoring and non-restoring division algorithms on the basis of speed, hardware/control complexity, and correction steps.