|  |
| --- |
| Experiment No. 6 |
| Implement restoring division algorithm |
| Date of Performance: |
| Date of Correction: |

**Aim:** To implement a program that performs binary division using the non-restoring division algorithm.

**Objective:** To understand and implement the non-restoring division method for binary division used in digital arithmetic units.

**Theory:**

**Introduction:**

In digital systems and computer architecture, division is one of the essential arithmetic operations executed by the **Arithmetic Logic Unit (ALU)**. While restoring division is simple and intuitive, it involves extra steps to restore the original value of the remainder when a subtraction leads to a negative result. To improve performance and reduce hardware complexity, the **non-restoring division algorithm** was developed.

The **non-restoring division algorithm** eliminates the "restore" step used in the restoring division method by keeping track of the sign of the remainder and adjusting the next operation accordingly. This saves hardware time and reduces the number of operations required for division.

**Key Concepts:**

Let’s define:

* **Q (Dividend)**
* **M (Divisor)**
* **A (Accumulator / Remainder)**
* **n (number of bits)**

In non-restoring division:

* If the remainder is **positive**, subtract the divisor.
* If the remainder is **negative**, add the divisor.
* After each operation, shift the partial result left and update the quotient based on the new remainder's sign.

**Steps of Non-Restoring Division Algorithm:**

1. Initialize A = 0 and Q = dividend, M = divisor.
2. Repeat the following for **n** bits:
   * If A is **positive**, A = A − M
   * If A is **negative**, A = A + M
   * Shift A and Q left (logical shift)
   * If A is **positive**, set Q₀ = 1
   * If A is **negative**, set Q₀ = 0
3. After n iterations:
   * If A < 0, add M to A (final correction)
4. Q = quotient, A = remainder

**Comparison with Restoring Division:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Restoring Division** | **Non-Restoring Division** |
| **Handles negative A** | By restoring (A = A + M) | By switching to addition/subtraction |
| **Number of operations** | More due to restore step | Fewer; no explicit restore required |
| **Hardware complexity** | Slightly higher | More efficient for real-time systems |

**Example:**

Let’s divide **Q = 1101 (13)** by **M = 0011 (3)**:

* A starts as 0000
* Loop for 4 bits:
  + Subtract M from A if A ≥ 0
  + Add M to A if A < 0
  + Shift A-Q left
  + Set Q₀ = 1 if A ≥ 0 else 0
* At the end, A = remainder, Q = quotient

**Applications:**

* Implemented in **ALUs** of modern processors.
* Used in **hardware accelerators**, **FPGA arithmetic modules**, and **embedded systems**.
* Forms the basis for **faster division algorithms**, including **SRT** and **Booth's** algorithms.
* Important for developing **custom arithmetic units** in VHDL/Verilog and digital simulation platforms.

**Solution:**

**Conclusion:** We implemented the non-restoring division algorithm and realized how it improves efficiency by avoiding unnecessary restore steps during division.