

**Experiment 4: Booth’s Multiplication Algorithm**

**Objective:**

1. To implement Booth’s Multiplication Algorithm.

**Theory:**

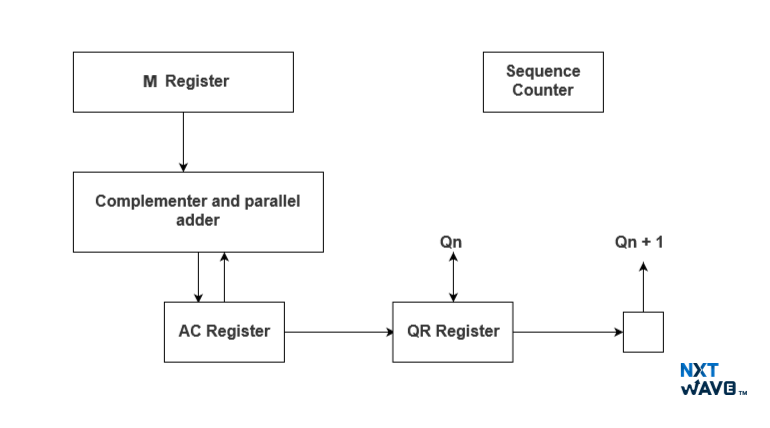
Booth's algorithm is a technique used for multiplying binary numbers. It reduces the number of operations required for multiplication by encoding the multiplier in a specific way. When compared to traditional multiplication algorithms, it treats both positive and negative numbers in a consistent manner, which makes it highly efficient for computer systems where signed numbers are commonly used.

**Hardware Implementation**

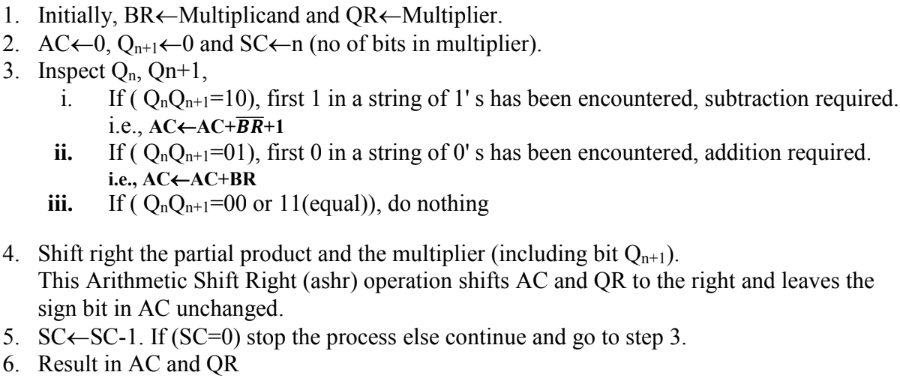
In Booth’s Algorithm, the multiplication process involves:

* Inspecting two consecutive bits of the multiplier at a time.
* Adding, subtracting, or leaving unchanged the current product based on these bits.
* Shifting the product right after each operation gradually forms the final result.

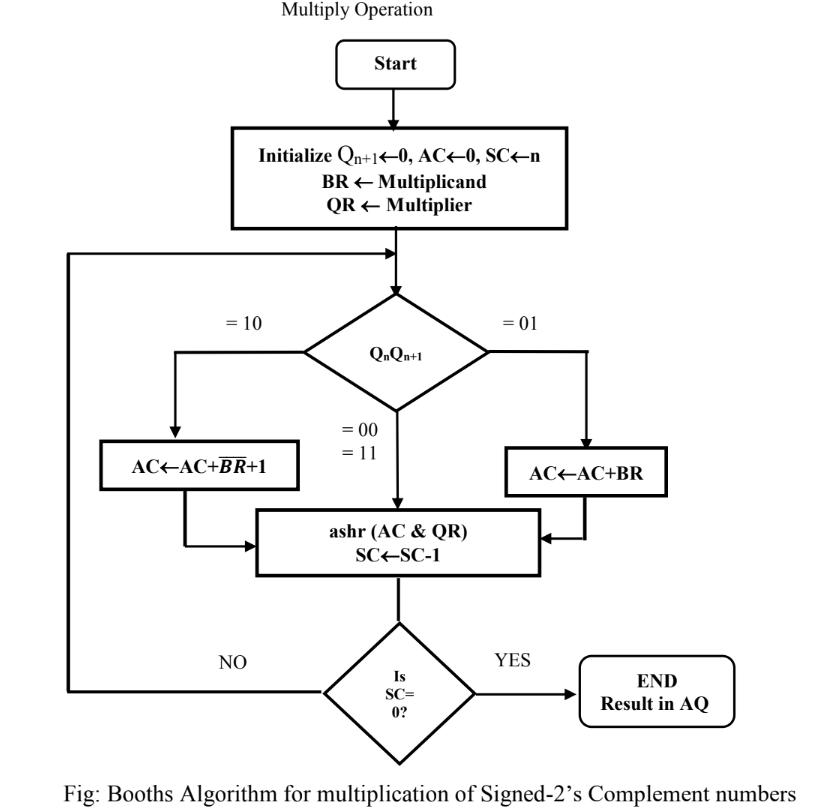
**Implementation of Booth’s Algorithm**



**Algorithm:**



**Flowchart:**



**Source code:**

#include <stdio.h>

void add(int a[], int x[], int q);

void complement(int a[], int n)

{

int i;

int x[8] = {0};

x[0] = 1;

for (i = 0; i < n; i++)

{

a[i] = (a[i] + 1) % 2;

}

add(a, x, n);

}

void add(int ac[], int x[], int q)

{

int i, c = 0;

for (i = 0; i < q; i++)

{

ac[i] = ac[i] + x[i] + c;

if (ac[i] > 1)

{

ac[i] = ac[i] % 2;

c = 1;

}

else

c = 0;

}

}

void ashr(int ac[], int qr[], int \*qn, int q)

{

int temp, i;

temp = ac[0];

\*qn = qr[0];

printf("\t\tashr\t\t");

for (i = 0; i < q - 1; i++)

{

ac[i] = ac[i + 1];

qr[i] = qr[i + 1];

}

qr[q - 1] = temp;

}

void display(int ac[], int qr[], int qrn)

{

int i;

for (i = qrn - 1; i >= 0; i--)

printf("%d", ac[i]);

printf(" ");

for (i = qrn - 1; i >= 0; i--)

printf("%d", qr[i]);

}

int main()

{

int mt[10], br[10], qr[10], sc, ac[10] = {0};

int brn, qrn, i, qn, temp;

printf("\t=== Complied by Jonash Chataut ===\n");

printf("\t===== Booth's Algorithm =====\n");

printf("\n Number of multiplicand bit= ");

scanf("%d", &brn);

printf("\nmultiplicand= ");

for (i = brn - 1; i >= 0; i--)

scanf("%d", &br[i]); // multiplicand

for (i = brn - 1; i >= 0; i--)

mt[i] = br[i];

complement(mt, brn);

printf("\nNo. of multiplier bit= ");

scanf("%d", &qrn);

sc = qrn;

printf("Multiplier= ");

for (i = qrn - 1; i >= 0; i--)

scanf("%d", &qr[i]);

qn = 0;

temp = 0;

printf("qn\tq[n+1]\t\tBR\t\tAC\tQR\t\tsc\n");

printf("\t\t\tinitial\t\t");

display(ac, qr, qrn);

printf("\t\t%d\n", sc);

while (sc != 0)

{

printf("%d\t%d", qr[0], qn);

if ((qn + qr[0]) == 1)

{

if (temp == 0)

{

add(ac, mt, qrn);

printf("\t\tsubtracting BR\t");

for (i = qrn - 1; i >= 0; i--)

printf("%d", ac[i]);

temp = 1;

}

else if (temp == 1)

{

add(ac, br, qrn);

printf("\t\tadding BR\t");

for (i = qrn - 1; i >= 0; i--)

printf("%d", ac[i]);

temp = 0;

}

printf("\n\t");

ashr(ac, qr, &qn, qrn);

}

else if (qn - qr[0] == 0)

ashr(ac, qr, &qn, qrn);

display(ac, qr, qrn);

printf("\t");

sc--;

printf("\t%d\n", sc);

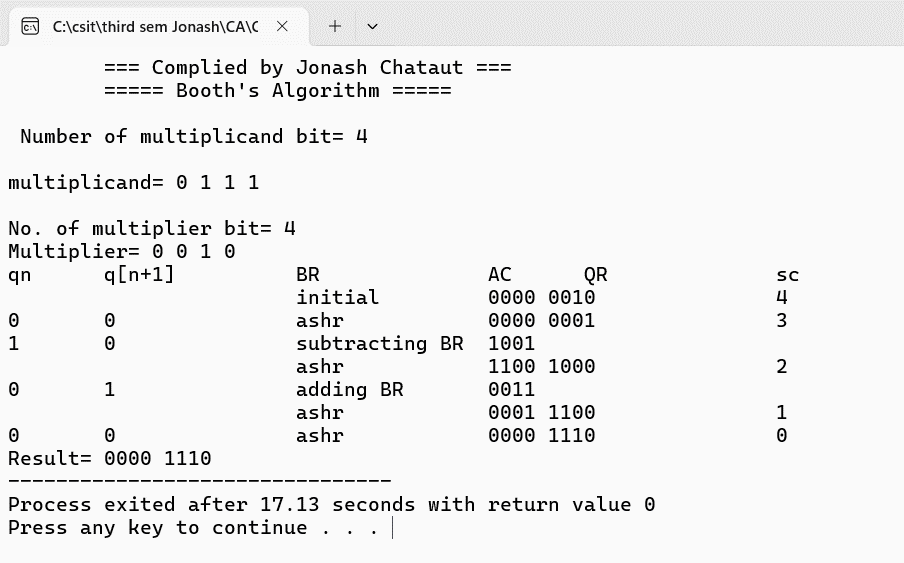
}

printf("Result= ");

display(ac, qr, qrn);

}

**Output:**



**Conclusion:**

Booth’s Algorithm is an efficient way to multiply signed binary numbers using shifts and additions. It handles both positive and negative inputs with ease and mimics how real hardware performs multiplication.

**A blue and white logo

Description automatically generated**

**Experiment 5: Restoring Division Algorithm**

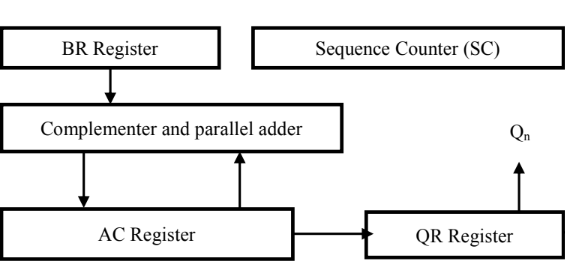
**Objective:**

1. To implement Restoring Division Algorithm.

**Theory:**

The restoring division algorithm is a method used for binary division in computer architecture. It involves repeatedly subtracting the divisor from the dividend and restoring the partial remainder if the result is negative. The algorithm uses a sign bit to track the sign of the partial remainder and adjusts accordingly. The process continues until the division is complete, and the quotient and remainder are obtained.

**Hardware Implementation**



**Algorithm:**

**Step 1:** Initialize A, Q and M registers to zero, dividend and divisor respectively and counter to

n where n is the number of bits in the dividend.

**Step 2:** Shift A, Q left one binary position.

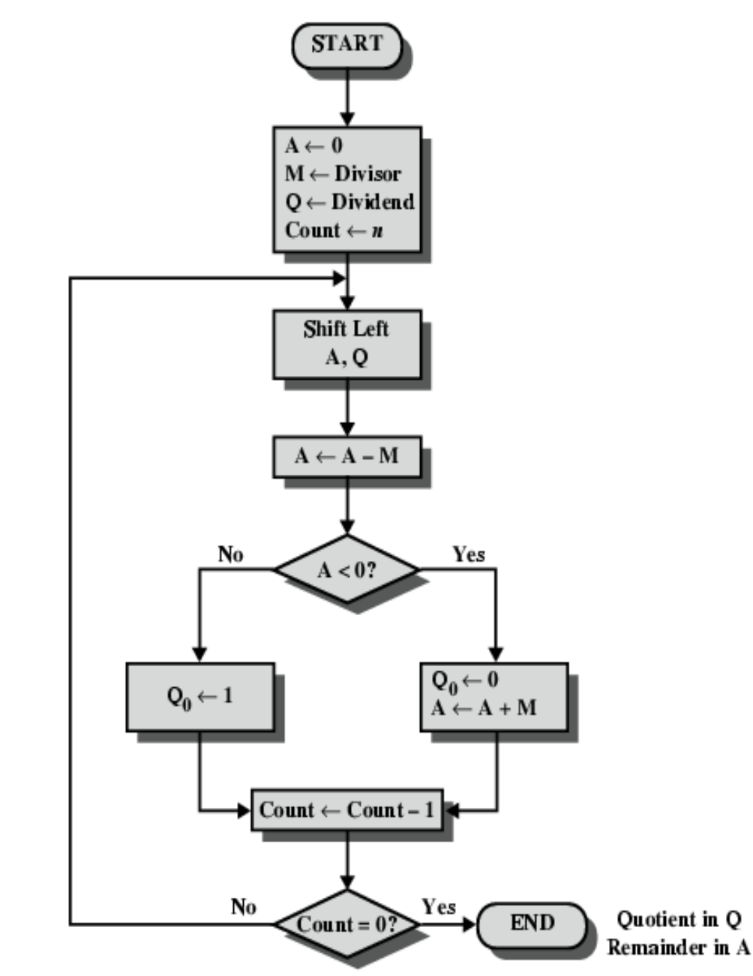
**Step 3:** Subtract M from A placing answer back in A. If sign of A is 1, set Q to zero and add M

back to A (restore A). If sign of A is 0, set Q to 1.

**Step 4:** Decrease counter; if counter > 0, repeat process from step 2 else stop the process. The

final remainder will be in A and quotient will be in Q.

**Flowchart:**

****

**Source code:**

#include <stdio.h>

void print\_binary(int num, int bits)

{

int i;

for (i = bits - 1; i >= 0; i--)

{

printf("%d", (num >> i) & 1);

}

}

void print\_step(int step, const char \*op, int A, int Q, int bits)

{

printf("| %-12s | ", op);

print\_binary(A, bits);

printf(" | ");

print\_binary(Q, bits);

if (op == "Initialize")

printf(" | %2d |\n", step);

else if (op == "Restore A" || op == "Set Q0=1")

printf(" | %2d |\n", step - 1);

else

printf(" | |\n");

}

int main()

{

int dividend, divisor;

int bits = 4;

int A, Q, M;

int sc;

printf("\t=== Complied by Jonash Chataut ===\n");

printf("===== Restoring Division Algorithm (4-bit) =====\n");

printf("Enter Dividend Q (0 - 15): ");

scanf("%d", &dividend);

printf("Enter Divisor M (0 - 15): ");

scanf("%d", &divisor);

A = 0;

Q = dividend;

M = divisor;

sc = bits;

printf("\n+--------------+--------+--------+-----+\n");

printf("| Operation | A | Q | SC |\n");

printf("+--------------+--------+--------+-----+\n");

print\_step(sc, "Initialize", A, Q, bits);

for (sc = bits; sc > 0; sc--)

{

// Shift AQ left

A = (A << 1) | ((Q >> (bits - 1)) & 1);

Q <<= 1;

print\_step(sc, "Shift AQ", A, Q, bits);

// Subtract M from A

A -= M;

print\_step(sc, "A = A - M", A, Q, bits);

if (A < 0)

{

A += M; // Restore

print\_step(sc, "Restore A", A, Q, bits);

}

else

{

Q |= 1; // Set LSB

print\_step(sc, "Set Q0=1", A, Q, bits);

}

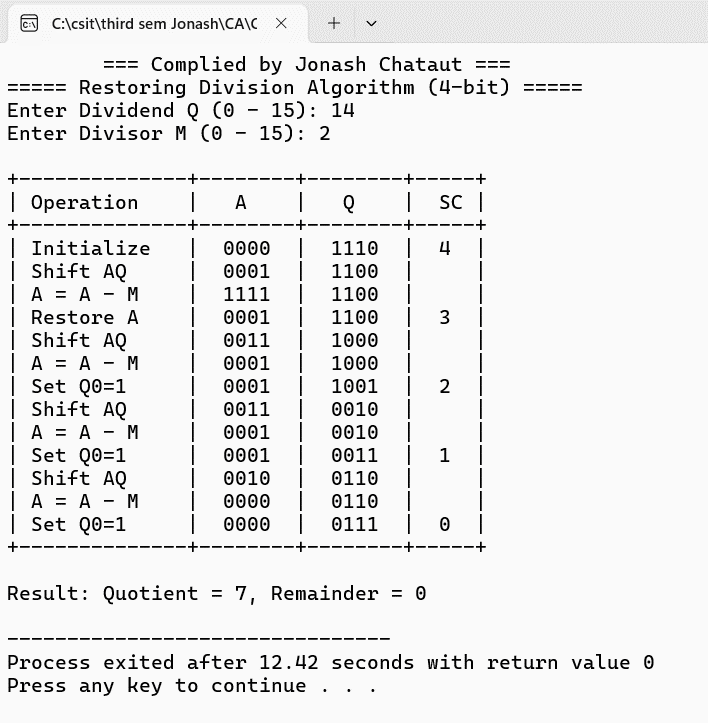
}

printf("+--------------+--------+--------+-----+\n");

printf("\nResult: Quotient = %d, Remainder = %d\n", Q & 0xF, A & 0xF); // Mask to 4 bits

}

**Output:**



**Conclusion:**

The Restoring Division Algorithm provides a systematic method to divide binary numbers using shifting and subtraction. It is simple to implement in C and effectively handles unsigned binary division, giving accurate quotient and remainder.