**BOOTHS MULTIPLICATION USING C**

**A PROJECT REPORT**

***Submitted by***

**ADITHYA KRISHNA[RA2211003011212]**

**GAUTHAM KRISHNA P[RA2211003011222]**

**SHASHANK REDDY[RA2211003011211]**

***for the course 21CSS201T Computer Organization and Architecture***

***Under the Guidance of***

**Dr. G.K. Sandhia**

**Assistant Professor, Department of Computing Technologies**

***In partial satisfaction of the requirements for the degree of***

## **BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE ENGINEERING**

## Logo, company name Description automatically generated

**SCHOOL OF COMPUTING**

# **COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603203**

**NOVEMBER 2023**

Logo, company name

Description automatically generated

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

**BONAFIDE CERTIFICATE**

Certified that the 21CSS201T Computer Organization and Architecture courseproject report titled **“BOOTHS MULTIPLICATION USING C”** is the bonafide work done by **Adithya Krishna [RA22110003011212], Gautham Krishna P[RA2211003011222],**

**, Shashank Reddy[RA2211003011211]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

|  |  |
| --- | --- |
| **Dr. G.K. Sandhia**  Faculty In-Charge,  Assistant Professor,  Department of Computing Technologies,  SRMIST. | **Dr. M. Pushpalatha,**  Professor and Head,  Department of Computing Technologies,  SRMIST. |

**Booth’s Multiplication Using C**

**INDEX**

**OBJECTIVE………………………………………………………………4**

**ABSTRACT……………………………………………………………….5**

**INTRODUCTION………………………………………………………..6**

**HARDWARE & SOFTWARE REQUIREMENTS……………………7**

**CONCEPT/WORKING PRINCIPLE ………………………………….7**

**FLOWCHART……………………………………………………………7**

**PROGRAM………………………………………………………………..8**

**OUTPUT…………………………………………………………………..12**

**CONCLUSION…………………………………………………………...13**

**REFERENCE……………………………………………………………..14**

**OBJECTIVE:**

The primary objectives of Booth's Algorithm are to optimize and streamline the process of binary multiplication. Booth's Algorithm is specifically designed to achieve the following goals:

Efficiency: Booth's Algorithm aims to perform binary multiplication with fewer bit-level operations than traditional methods, resulting in faster execution.

Reduction of Operations: By strategically selecting when to perform additions or subtractions, the algorithm reduces the number of additions and multiplications required to compute the product.

Optimization: The algorithm optimizes the multiplication process by considering the bit patterns in both the multiplier and multiplicand, allowing for more efficient calculation.

Scalability: Booth's Algorithm is not limited to specific data sizes and can be applied to binary multiplication of various lengths, making it adaptable to different applications.

Accuracy: Despite its optimization, Booth's Algorithm maintains the accuracy of the multiplication operation, ensuring that the final result is correct.

Hardware Implementation: Booth's Algorithm is commonly used in hardware design and microprocessor architecture, where efficiency and resource conservation are critical.

**ABSTRACT:**

Booth multiplication is a binary multiplication algorithm used to efficiently calculate the product of two binary numbers. In a C-based implementation of Booth multiplication, the key principles include optimizing the multiplication process by strategically determining when to perform additions and subtractions. Booth multiplication conserves computational resources, making it particularly well-suited for hardware implementations. It balances efficiency with accuracy, resulting in a faster multiplication process while maintaining the correctness of the result. Booth multiplication is widely utilized in computer architecture and digital circuit design, where binary multiplication is a fundamental operation.

**INTRODUCTION:**

The primary goal of this project is to explore and implement Booth's Multiplication Algorithm using the C programming language. This algorithm offers a systematic and efficient way to multiply signed binary numbers, and it can significantly enhance the performance of binary multiplication operations in various computational contexts.

This introduction sets the stage for our exploration of Booth's Algorithm in C. We will delve into the theoretical underpinnings of the algorithm, its advantages over conventional multiplication methods, and the step-by-step process through which it accomplishes multiplication. The implementation in C will be described in detail, including the initialization of variables, the handling of shifting and addition operations, and how overflow conditions are managed.

Moreover, the C program we will develop will offer a user-friendly interface for inputting binary operands and provide clear and accurate output for the resulting product. Robust error handling and thorough documentation will be integral parts of the code, ensuring that it is both efficient and comprehensible.

In essence, this project aims to provide a practical and valuable resource for computer scientists, digital designers, engineers, and students interested in binary multiplication techniques and algorithmic optimization. Furthermore, it aligns with the broader objective of advancing computational efficiency in digital systems and software, making it a critical subject in the field of computer science and digital engineering.

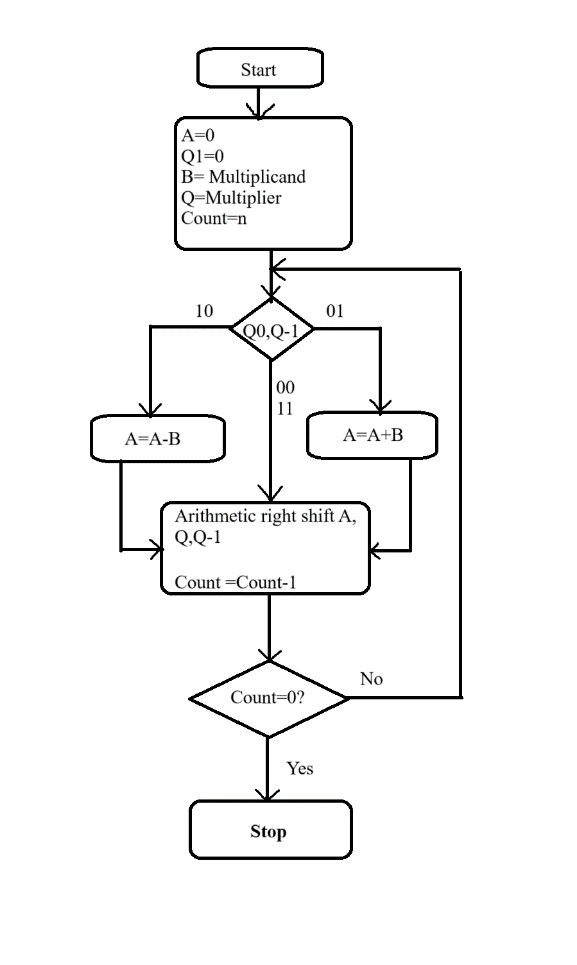
**HARDWARE/SOFTWARE REQUIREMENTS:**

* A PC with any OS.
* C compiler

**CONCEPTS/WORKING PRINCIPLE**

Booth's Multiplication is an efficient binary multiplication algorithm that reduces the number of addition and shifting operations compared to traditional methods. It uses Booth Encoding to identify bit patterns and optimize multiplicative actions. This algorithm is widely used in computer architecture and digital design to improve the efficiency of binary multiplication for large numbers. While it's more complex to implement, it offers advantages in terms of speed and efficiency, especially in hardware design.

**FLOWCHART:**

****

**PROGRAMS:**

#include <stdio.h>

#include <math.h>

int a = 0,b = 0, c = 0, a1 = 0, b1 = 0, com[5] = { 1, 0, 0, 0, 0};

int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};

int acomp[5] = {0}, bcomp[5] = {0}, pro[5] = {0}, res[5] = {0};

void binary(){

a1 = fabs(a);

b1 = fabs(b);

int r, r2, i, temp;

for (i = 0; i < 5; i++){

r = a1 % 2;

a1 = a1 / 2;

r2 = b1 % 2;

b1 = b1 / 2;

anum[i] = r;

anumcp[i] = r;

bnum[i] = r2;

if(r2 == 0){

bcomp[i] = 1;

}

if(r == 0){

acomp[i] =1;

}

}

c = 0;

for ( i = 0; i < 5; i++){

res[i] = com[i]+ bcomp[i] + c;

if(res[i] >= 2){

c = 1;

}

else

c = 0;

res[i] = res[i] % 2;

}

for (i = 4; i >= 0; i--){

bcomp[i] = res[i];

}

if (a < 0){

c = 0;

for (i = 4; i >= 0; i--){

res[i] = 0;

}

for ( i = 0; i < 5; i++){

res[i] = com[i] + acomp[i] + c;

if (res[i] >= 2){

c = 1;

}

else

c = 0;

res[i] = res[i]%2;

}

for (i = 4; i >= 0; i--){

anum[i] = res[i];

anumcp[i] = res[i];

}

}

if(b < 0){

for (i = 0; i < 5; i++){

temp = bnum[i];

bnum[i] = bcomp[i];

bcomp[i] = temp;

}

}

}

void add(int num[]){

int i;

c = 0;

for ( i = 0; i < 5; i++){

res[i] = pro[i] + num[i] + c;

if (res[i] >= 2){

c = 1;

}

else{

c = 0;

}

res[i] = res[i]%2;

}

for (i = 4; i >= 0; i--){

pro[i] = res[i];

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

}

printf(":");

for (i = 4; i >= 0; i--){

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

}

}

void arshift(){

int temp = pro[4], temp2 = pro[0], i;

for (i = 1; i < 5 ; i++){

pro[i-1] = pro[i];

}

pro[4] = temp;

for (i = 1; i < 5 ; i++){

anumcp[i-1] = anumcp[i];

}

anumcp[4] = temp2;

printf("\nAR-SHIFT: ");

for (i = 4; i >= 0; i--){

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

}

printf(":");

for(i = 4; i >= 0; i--){

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

}

}

void main(){

int i, q = 0;

printf("\t\tBOOTH'S MULTIPLICATION ALGORITHM");

printf("\nEnter two numbers to multiply: ");

printf("\nBoth must be less than 16");

//simulating for two numbers each below 16

do{

printf("\nEnter A: ");

scanf("%d",&a);

printf("Enter B: ");

scanf("%d", &b);

}while(a >=16 || b >=16);

printf("\nExpected product = %d", a \* b);

binary();

printf("\n\nBinary Equivalents are: ");

printf("\nA = ");

for (i = 4; i >= 0; i--){

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

}

printf("\nB = ");

for (i = 4; i >= 0; i--){

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

}

printf("\nB'+ 1 = ");

for (i = 4; i >= 0; i--){

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

}

printf("\n\n");

for (i = 0;i < 5; i++){

if (anum[i] == q){

printf("\n-->");

arshift();

q = anum[i];

}

else if(anum[i] == 1 && q == 0){

printf("\n-->");

printf("\nSUB B: ");

add(bcomp);

arshift();

q = anum[i];

}

else{

printf("\n-->");

printf("\nADD B: ");

add(bnum);

arshift();

q = anum[i];

}

}

printf("\nProduct is = ");

for (i = 4; i >= 0; i--){

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

}

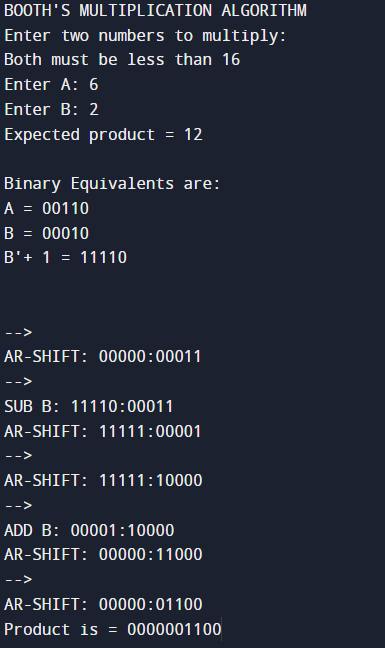
for (i = 4; i >= 0; i--){

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

}

}

**OUTPUT:**



**CONCLUSIONS:**

In conclusion, big font and icon sizes are indispensable components of effective user interface design. They significantly enhance accessibility for a diverse range of users, improve usability, and contribute to a visually pleasing aesthetic. Designers and developers must carefully consider the implementation of these elements to create user interfaces that are both functional and appealing, ultimately leading to a more satisfying user experience. In an era where digital interfaces dominate our interactions, prioritizing the needs of users through thoughtful design choices is paramount.

**REFERENCES:**

Booth’s Algorithm

<https://www.geeksforgeeks.org/computer-organization-booths-algorithm/>

Booth’s Multiplication

<https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm#:~:text=Booth%27s%20multiplication%20algorithm%20is%20a,Birkbeck%20College%20in%20Bloomsbury%2C%20London>.

<http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241_s00/PAPERS/archive/booth51.pdf>

Decimal to Binary Conversion

[**https://www.geeksforgeeks.org/program-decimal-binary-conversion/**](https://www.geeksforgeeks.org/program-decimal-binary-conversion/)

**THANK YOU**