**EXPERIMENT - 2**

**AIM:** To demonstrate the Restoring Division Algorithm for 2 unsigned binary numbers.

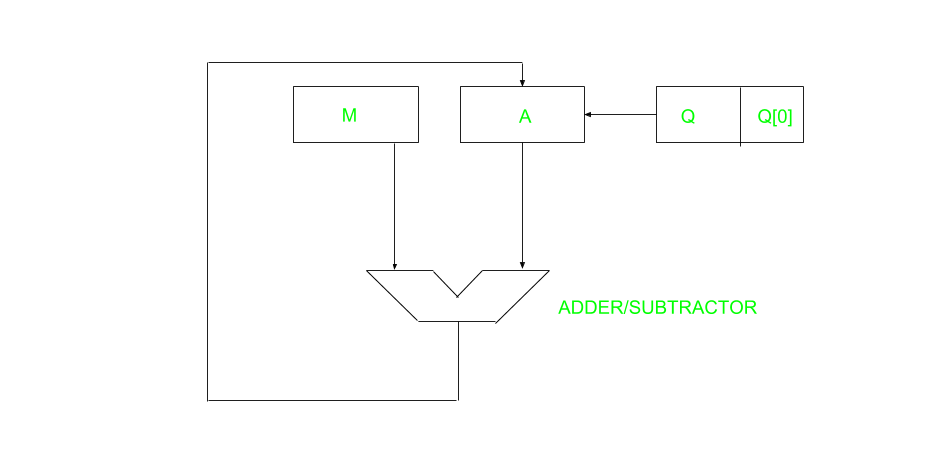
**Submission Sheet**

| **SAP ID** | **Name of Student** | **Date of Experiment** | **Date of Submission** | **Remarks** |
| --- | --- | --- | --- | --- |
| **60004190057** | **Junaid Girkar** | **08-10-2021** | **08-10-2021** |  |

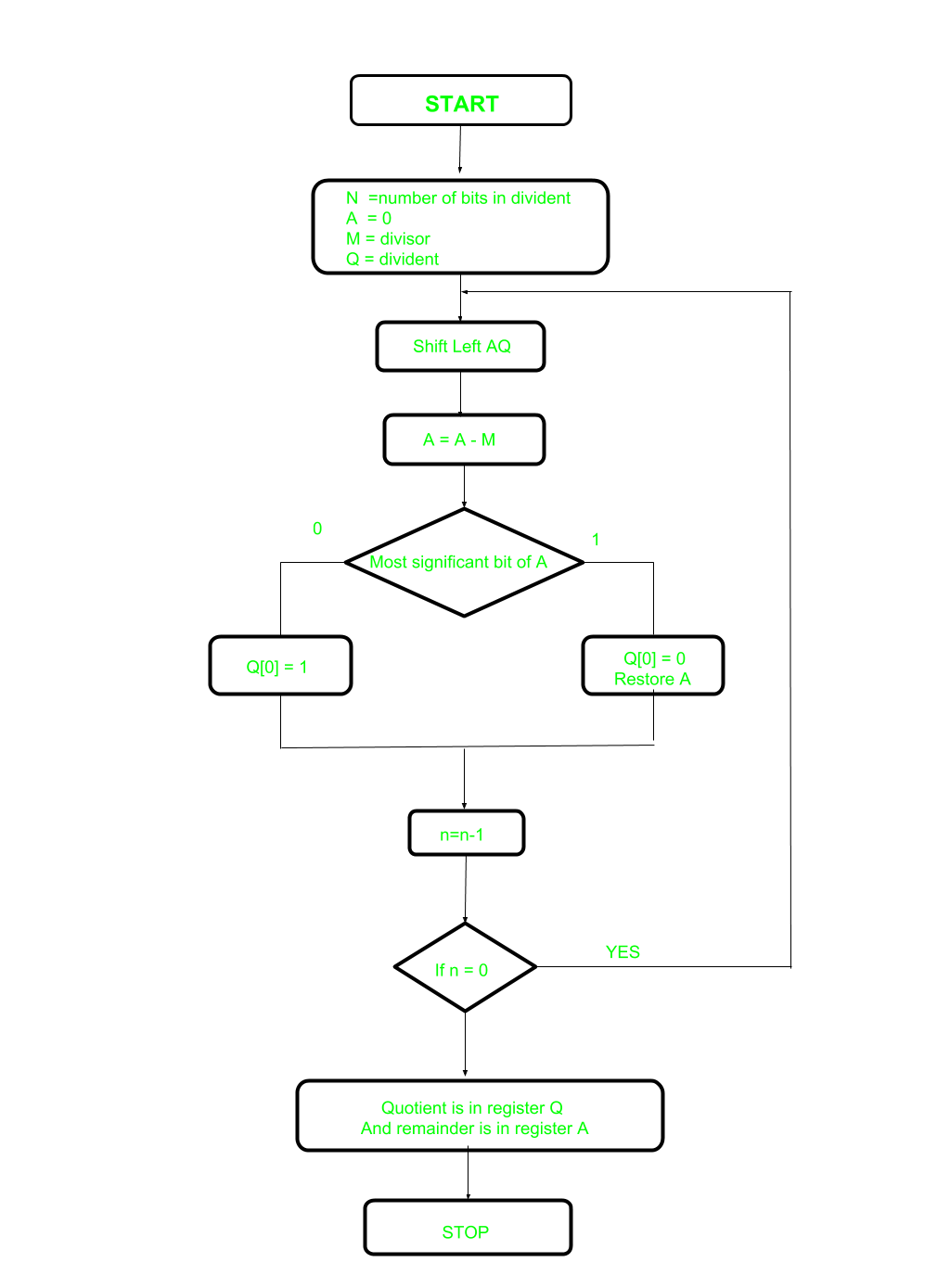
**THEORY:**

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

In this article, will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.



Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.



Let’s pick the step involved:

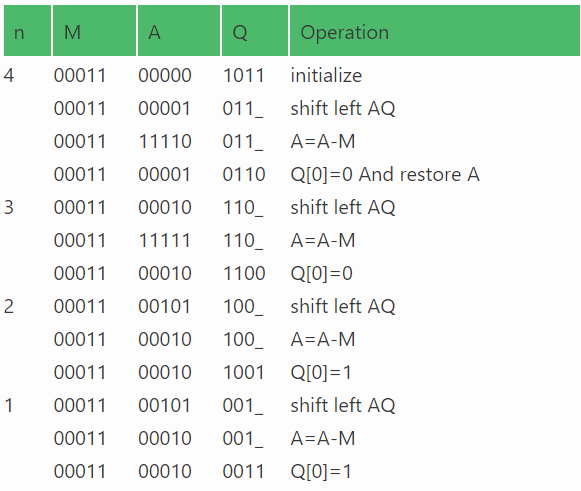
* **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
* **Step-2:** Then the content of register A and Q is shifted right as if they are a single unit
* **Step-3:** Then content of register M is subtracted from A and result is stored in A
* **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
* **Step-5:** The value of counter n is decremented
* **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat fro step 2
* **Step-7:** Finally, the register Q contain the quotient and A contain remainder

**Examples:**

Perform Division Restoring Algorithm

Dividend = 11

Divisor = 3

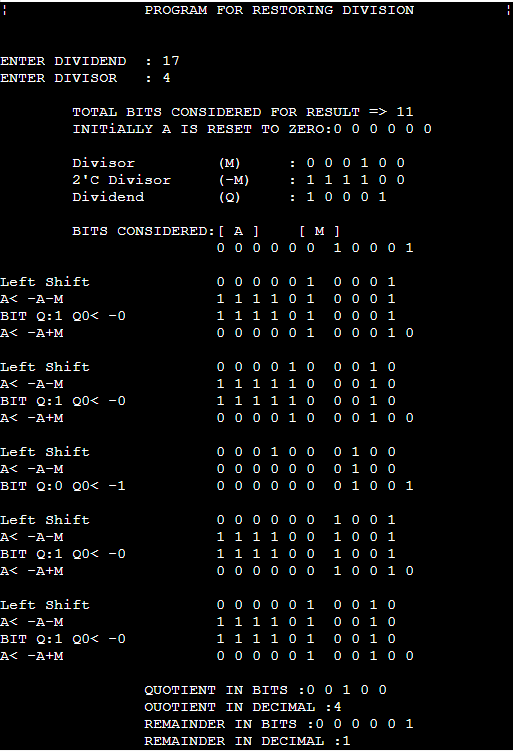


Remember to restore the value of A, most significant bit of A is 1. As that register Q contains the quotient, i.e. 3 and register A contains remainder 2.

CODE:

| #include<stdio.h> #include<conio.h> #include<math.h>  int getsize(int x) { int c; if(x<=1) c = 2; else if(x < 4) c = 2; else if(x< 8) c = 3; else if(x< 16) c = 4; else if(x< 32) c = 5; else if(x< 64) c = 6; else if(x< 128) c = 7; else if(x< 256) c = 8; else if(x< 512) c = 9; return c; }  int max(int x,int y) { if(x< y) return(y); else return(x); }  void main() { int B,Q,Z,M,c,c1,e,f,g,h,i,j,x,y,ch,in,S,G,P; int a[24],b[12],b1[12],q[12],carry=0,count=0,option; long num;  printf("¦\t\tPROGRAM FOR RESTORING DIVISION\t\t¦\n"); printf("\n\nENTER DIVIDEND\t: "); scanf("%d",&Q); y = getsize(Q); printf("ENTER DIVISOR\t: "); scanf("%d",&M); x = getsize(M); Z = max(x,y); printf("\n\tTOTAL BITS CONSIDERED FOR RESULT => %d",2\*Z+1); printf("\n\tINITiALLY A IS RESET TO ZERO:"); for(i=0;i<=Z;i++) printf("%d ",a[i]=0); for(i=Z;i>=0;i--) { b1[i] = b[i] = M%2; M = M/2; b1[i] = 1-b1[i]; } carry = 1; for(i=Z;i>=0;i--) { c1 = b1[i]^carry; carry = b1[i]&&carry; b1[i]=c1; } for(i=2\*Z;i>Z;i--) { a[i] = Q%2; Q = Q/2; } printf("\n\n\tDivisor\t\t(M)\t: "); for(i=0;i<=Z;i++) printf("%d ",b[i]); printf("\n\t2'C Divisor\t(-M)\t: "); for(i=0;i<=Z;i++) printf("%d ",b1[i]); printf("\n\tDividend\t(Q)\t: "); for(i=Z+1;i<=2\*Z;i++) printf("%d ",a[i]); printf("\n\n\tBITS CONSIDERED:[ A ]\t [ M ]"); printf("\n\t\t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i<=2\*Z;i++) printf("%d ",a[i]); count = Z; do{ for(i=0;i<2\*Z;i++) a[i] = a[i+1]; printf("\n\nLeft Shift\t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i< 2\*Z;i++) printf("%d ",a[i]); carry=0; for(i=Z;i>=0;i--) { S=a[i]^(b1[i]^carry); G=a[i]&&b1[i]; P=a[i]^b1[i]; carry=G||(P&&carry); a[i]=S ; } printf("\nA< -A-M \t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i< 2\*Z;i++) printf("%d ",a[i]); ch=a[0]; printf("\nBIT Q:%d",ch); switch (ch) { case 0: a[2\*Z]=1; printf(" Q0< -1\t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i<=2\*Z;i++) printf("%d ",a[i]); break;  case 1: a[2\*Z]=0; printf(" Q0< -0\t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i< 2\*Z;i++) printf("%d ",a[i]); carry=0; for(i=Z;i>=0;i--) { S=a[i]^(b[i]^carry); G=a[i]&&b[i]; P=a[i]^b[i]; carry=G||(P&&carry); a[i]=S ; } printf("\nA< -A+M"); printf("\t\t\t"); for(i=0;i<=Z;i++) printf("%d ",a[i]); printf(" "); for(i=Z+1;i<=2\*Z;i++) printf("%d ",a[i]); break; } count--; }while(count!=0); num=0; printf("\n\n\t\tQUOTIENT IN BITS :"); for(i=Z+1;i<=2\*Z;i++) { printf("%d ",a[i]); num=num+pow(2,2\*Z-i)\*a[i]; } printf("\n\t\tOUOTIENT IN DECIMAL :%ld",num); num=0; printf("\n\t\tREMAINDER IN BITS :"); for(i=0;i<=Z;i++) { printf("%d ",a[i]); num=num+pow(2,Z-i)\*a[i]; } printf("\n\t\tREMAINDER IN DECIMAL :%ld",num); getch();  getch(); } |
| --- |

OUTPUT:



**CONCLUSION**: From this experiment, we learn how to use the restoring division algorithm to divide unsigned bits. We understood that it is a type of slow algorithm along with non-restoring division. We also learn how to write the code for the same algorithm in C language and implement it successfully.