EXPERIMENT - 1

AIM: To demonstrate Booth's Algorithm for multiplication of 2 signed binary numbers.

Submission Sheet

SAP ID	Name of Student	Date of Experiment	Date of Submission	Remarks
60004190057	Junaid Girkar	01-10-2021	01-10-2021	

THEORY:

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation in an efficient way, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values *A* and *S* to a product *P*, then performing a rightward arithmetic shift on *P*. Let **m** and **r** be the multiplicand and multiplier, respectively; and let *x* and *y* represent the number of bits in **m** and **r**.

1. Determine the values of *A* and *S*, and the initial value of *P*. All of these numbers should have a length equal to ($x + y + 1$).
    1. A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining ($y + 1$) bits with zeros.
    2. S: Fill the most significant bits with the value of (−**m**) in two's complement notation. Fill the remaining ($y + 1$) bits with zeros.
    3. P: Fill the most significant *x* bits with zeros. To the right of this, append the value of **r**. Fill the least significant (rightmost) bit with a zero.
2. Determine the two least significant (rightmost) bits of *P*.
    1. If they are 01, find the value of $P + A$. Ignore any overflow.
    2. If they are 10, find the value of $P + S$. Ignore any overflow.
    3. If they are 00, do nothing. Use *P* directly in the next step.

4. If they are 11, do nothing. Use *P* directly in the next step.

3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let *P* now equal this new value.
4. Repeat steps 2 and 3 until they have been done *y* times.
5. Drop the least significant (rightmost) bit from *P*. This is the product of **m** and **r**.

TIME COMPLEXITY: O(n) * (complexity_of_addition + complexity_of_shift)

EXAMPLE: Multiplication of 7 and 3

Q_n	Q_{n+1}	M = (0111) M' + 1 = (1001) & Operation	AC	Q	Q_{n+1}	SC
1	0	Initial	0000	0011	0	4
		Subtract (M' + 1)	1001			
			1001			
		Perform Arithmetic Right Shift operations (ashr)	1100	1001	1	3
1	1	Perform Arithmetic Right Shift operations (ashr)	1110	0100	1	2
0	**1**	Addition (A + M)	0111			
			0101	0100		
		Perform Arithmetic right shift operation	0010	1010	0	1
0	0	Perform Arithmetic right shift operation	**0001**	**0101**	0	0

Final Answer = $(0001\ 0101)_2$
= $(21)_{10}$

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void rightShift();
```

```
int main()
{
   printf("\n");
   printf("BOOTH's Algorithm\n");
   printf("\n");
   printf("Enter two numbers that are to be multiplied : ");//taking two numbers as inputs
   int a,b;
   scanf("%d %d",&a,&b);
   int ap=a,bp=b;
   if(ap<0) // Negetive values check
      ap*=-1;
   if(bp<0) bp*=-1;
   if(bp>ap) //taking greater VALUE as multiplicand
   {
      ap=bp+ap-(bp=ap);
      a=b+a-(b=a);
   }
   int t1=ap,t2=bp;
   int ab[35]={};
   int bb[35]={};
   int i=0;
   while(t1>0)
   {
      ab[i]=t1%2;
      i++;
      t1/=2;
   }
   ab[i]=0;
   int j=0;
   while(t2>0)
   {
      bb[j]=t2%2;
      j++;
      t2/=2;
   }
   while(j<=i) //equating bits to the previous(ab) binary number(ab will either be larger or equal to
bb).
      bb[j++]=0;
   int nb=i+1; //nb is number of bits
   i=0;j=0;
   while(i<nb/2) //converting VALUES to binary
   {
      ab[i]=ab[nb-i-1]+ab[i]-(ab[nb-i-1]=ab[i]);
      i++;
   }
   i=0;
   while(i<nb/2) { bb[i]=bb[nb-i-1]+bb[i]-(bb[nb-i-1]=bb[i]); i++; } int x[35]={0}; int y[35]={0}; i=0; if(a>=0)
```

```
//taking actual binary numbers
  { //x is multiplicand and y is multiplier
    while(i<nb)
    x[i]=ab[i+++1];
  }
  else //2's complimant
  {
    while(i<nb) { if(ab[i]==0) x[i]=1; else x[i]=0; i++; } i=1; x[nb-i]++; while(x[nb-i]==2) { x[nb-i]=0; i++;
x[nb-i]++; } } i=0; if(b>=0)
  {
    while(i<nb)
      y[i]=bb[i+++1];
  }
  else //2's complimant
  {
    while(i<nb) { if(bb[i]==0) y[i]=1; else y[i]=0; i++; } i=1; y[nb-i]++; while(y[nb-i]==2) { y[nb-i]=0; i++;
y[nb-i]++; } } printf("\n"); //output starts here printf("Multiplicand (Q) %d -> ",a);
  i=0;
  printf("Multiplicand (Q) -> ");
  while(i<nb) printf("%d",x[i++]); printf("\nMultiplier (M) %d -> ",b);
  i=0;
  while(i<nb)
    printf("%d",y[i++]);
  printf("\n");
  i=0;
  int ym[35]={0}; //calculating -M
  if(b<0)
  {
    while(i<nb)
    ym[i]=bb[i+++1];
  }
  else
  {
    while(i<nb) { if(bb[i]==0) ym[i]=1; else ym[i]=0; i++; } i=1; ym[nb-i]++; while(ym[nb-i]==2) {
ym[nb-i]=0; i++; ym[nb-i]++; } } printf("we use -(M) i.e. %d -> ",-b);
  i=0;
  while(i<nb)
    printf("%d",ym[i++]);
  printf("\n");
  int q0=0;
  int p[35]={0}; //p here is value that is stored in accumulator. initially set to zero.
  int steps=nb;
  printf("\n");
  printf("n\t");
  i=0;
  while(i<nb)
  {
```

```
    if(i*2==nb || i*2==nb-1)
    printf("A");
    else
    printf(" ");
    i++;
  }
  printf(" ");
  i=0;
  while(i<nb)
  {
    if(i*2==nb || i*2==nb-1)
      printf("Q\t");
    else
      printf(" ");
    i++;
  }
  printf(" Q-1");
  printf("\n");
  j=0;

  while(steps--) //counting down steps.
  {
    printf("%d        ",j++);
    i=0;
    while(i<nb)
      printf("%d",p[i++]);
    printf(" ");
    i=0;
    while(i<nb)
      printf("%d",x[i++]);
    printf(" ");
    printf("%d\n",q0);
    if(x[nb-1]==0 && q0==0) //0-0 condition
    {
      q0=x[nb-1];
      rightShift(p,x,nb);
    }
    else if(x[nb-1]==0 && q0==1) //0-1 condition
    {
      printf("    A + M ");
      i=0;
      while(i<nb)
        printf("%d",y[i++]);
      i=0;
      while(i<nb)
      {
        p[nb-i-1]+=y[nb-i-1];
```

```
        if(p[nb-i-1]==2)
        {
            p[nb-i-1]=0;
            if(nb-i-1!=0)
            p[nb-i-2]++;
        }
        if(p[nb-i-1]==3)
        {
            p[nb-i-1]=1;
            if(nb-i-1!=0)
            p[nb-i-2]++;
        }
        i++;
    }
    printf("\n          ");
    i=0;
    while(i<nb)
        printf("%d",p[i++]);
    printf("\n");
    q0=x[nb-1];
    rightShift(p,x,nb);
}
else if(x[nb-1]==1 && q0==0) //1-0 condition
{
    printf("    A - M ");
    i=0;
    while(i<nb)
        printf("%d",ym[i++]);
    i=0;
    while(i<nb)
    {
        p[nb-i-1]+=ym[nb-i-1];
        if(p[nb-i-1]==2)
        {
            p[nb-i-1]=0;
            if(nb-i-1!=0)
            p[nb-i-2]++;
        }
        if(p[nb-i-1]==3)
        {
            p[nb-i-1]=1;
            if(nb-i-1!=0)
            p[nb-i-2]++;
        }
        i++;
    }
    printf("\n          ");
```

```
            i=0;
            while(i<nb)
               printf("%d",p[i++]);
            printf("\n");
            q0=x[nb-1];
            rightShift(p,x,nb);
         }
         else if(x[nb-1]==1 && q0==1) //1-1 condition
         {
            q0=x[nb-1];
            rightShift(p,x,nb);
         }
      }
      printf("%d        ",j);
      i=0;
      while(i<nb)
         printf("%d",p[i++]);
      printf(" ");
      i=0;
      while(i<nb)
         printf("%d",x[i++]);
      printf(" ");
      printf("%d\n",q0);
      printf("\n");

      printf("Final Product in signed binary number is : ");
      i=0;
      while(i<nb)
         printf("%d",p[i++]);
      i=0;
      printf(" ");
      while(i<nb)
         printf("%d",x[i++]);
      printf("\n\n");
      return 0;
}

void rightShift(int p[],int x[],int nb)
{
      int i=0;
      while(nb-i-1)
      {
         x[nb-i-1]=x[nb-i-2];
         i++;
      }
      x[0]=p[nb-1];
      i=0;
```

```
    while(nb-i-1)
    {
        p[nb-i-1]=p[nb-i-2];
        i++;
    }
}
```

OUTPUT:

```
BOOTH's Algorithm

Enter two numbers that are to be multiplied : 20 35

Multiplicand (Q) -> 0100011
Multiplier (M) 20 -> 0010100
we use -(M) i.e. -20 -> 1101100

n          A       Q        Q-1
0          0000000 0100011 0
    A - M 1101100
          1101100
1          1110110 0010001 1
2          1111011 0001000 1
    A + M 0010100
          0001111
3          0000111 1000100 0
4          0000011 1100010 0
5          0000001 1110001 0
    A - M 1101100
          1101101
6          1110110 1111000 1
    A + M 0010100
          0001010
7          0000101 0111100 0

Final Product in signed binary number is : 0000101 0111100


...Program finished with exit code 0
Press ENTER to exit console.
```