BOOTH'S ALGORITHM FOR MULTIPLICATION

Booth's Multiplication Algorithm is used for multiplying two signed numbers in 2s complement notation.

HOW TO IMPLEMENT?

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**.

Source - https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

ASSUMPTIONS-

- The algorithm multiplies only two numbers at a time and gives the output.
- The numbers entered as input should **only be valid integers in decimal format.**
- The number of bits in the product (output) depends on the number of bits of the multiplicand and the multiplier (input).
- The algorithm is implemented for a 16-bit register.

CONSTRAINT-

- The algorithm is implemented for a 16-bit register which implies that the product has to be in the range of -2^(16-1) to 2^(16-1) -1, i.e., it should lie between -32768 to 32767. If the product is out of this range, then no output will be obtained.

REPRESENTATION-

- The MSB of any binary number in the code is the signed bit, i.e., it represents the sign of the number. A 0 MSB means that the number positive number and a negative number implies that the MSB is 1.
- The remaining bits to the right of the MSB is the binary representation of the number. For example, the representation of +3 is 0 11.
- If the number entered is negative, its binary representation is the number in it's 2s complement form. For example, the representation of -7 is 1 001 where 001 is the 2's complement of 111(7).

INPUT-

The code takes two valid integers in decimal format as input.

OUTPUT-

The output consists of the binary and decimal representation of the product.

EXPLANATION OF CODE-

- <u>FLOW</u>
  - ➔ The multiplicand(m1) and multiplier(m2) are taken as inputs (in decimal format).
  - ➔ The binary values of m1 and m2 are stored in m and r respectively.
  - ➔ x and y stores the length of m and r respectively.
  - ➔ The range of the product is checked. If the product lies outside the range mentioned above, the code terminates.
  - ➔ The initial value of A is determined by filling the most significant bits with the value of m and remaining (y+1) bits with zeroes.
  - ➔ The initial value of S is determined by filling the most significant bits with the value of 2's complement of (-m) and remaining (y+1) bits with zeroes.
  - ➔ The initial value of P is determined by filling the most significant x bits with zeroes, the remaining most significant bits with r and the LSB with 0.
  - ➔ After getting the initial values, a loop is run y times.
  - ➔ The last two (least significant) bits of P are determined and the value of P gets updated according to the operations mentioned in the implementation above.
  - ➔ The product of m and r is given by P after dropping it's LSB.

  <u>CHECKING THE CONSISTENCY OF THE CODE</u>
  - ➔ After printing the binary value, we also check if the answer(P) is correct or not. For this, we find the binary equivalent of P.
  - ➔ If the MSB of P is 0 implies that P is positive, so the magnitude of P should be equal to m*n (done arithmetically). If they match, the answer is correct.
  - ➔ A small check is done in the function to avoid redundancy. In case of a positive number, the leading zeros are removed while converting from binary to decimal.

If the string consists only of zeroes implies that P is 0 and so directly the decimal equivalent 0 is printed.

➔ If the MSB of P is 1 implies that P is negative, so the magnitude of 2’s complement of P should be equal to the magnitude of m*n (done arithmetically). If they match, the answer is correct.

- FUNCTIONS
  - ➔ **get2scomplement-**
    It takes a string as a parameter and returns the 2s complement of the string.
  - ➔ **getbinary-**
    It takes an integer as a parameter and converts the given integer to its binary equivalent in the form of a binary string and appends ‘1’ or ‘0’ in the string as it’s MSB depending on the sign on n. The Binary String is then returned.
  - ➔ **shiftright-**
    This function performs the arithmetic right shift. The MSB of the string given as a parameter is copied along with the string as it is except it’s LSB and is returned.
  - ➔ **binaryaddn-**
    This function takes two strings as parameters, performs binary addition on them and returns the output string.
  - ➔ **binarytodec-**
    This function converts the string given as a parameter and returns it’s decimal equivalent.