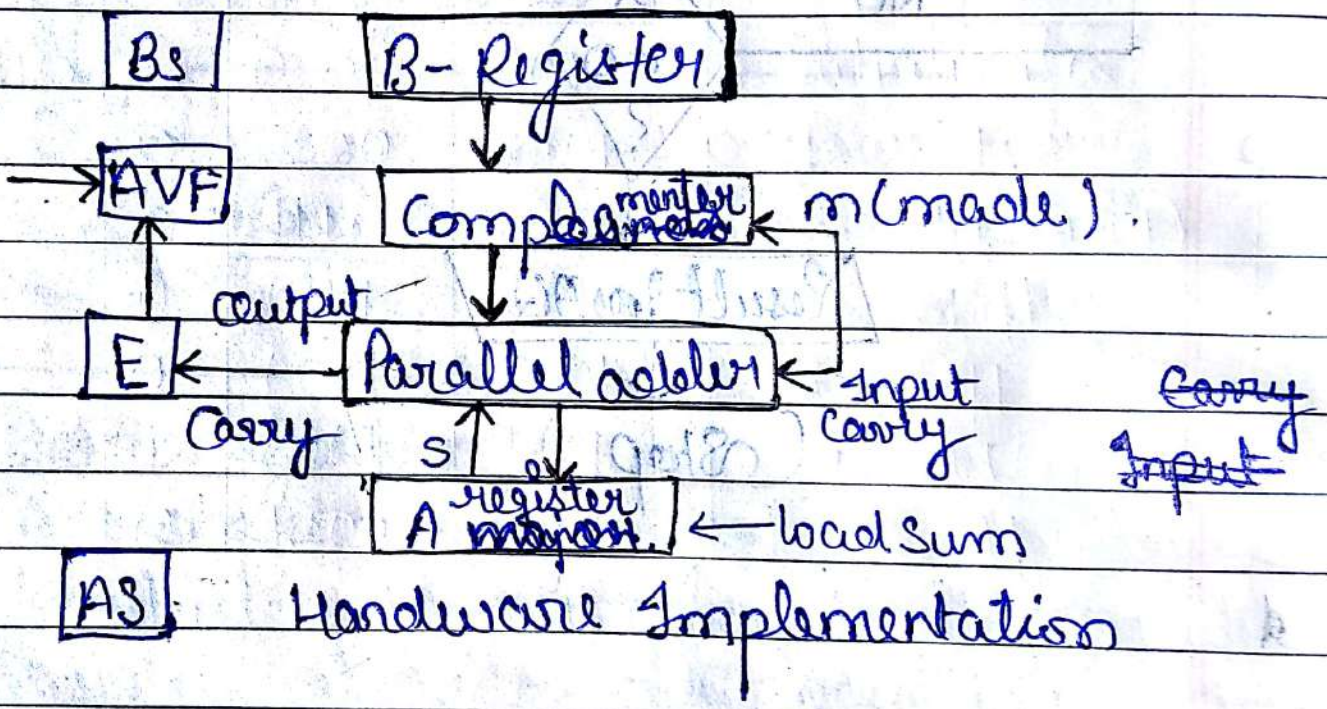
PAGE NO. // //
DATE // //

UNIT-3

★. Arithmetic addition and Subtraction with sign magnitude.-

AVF → Add overflow flipflop.

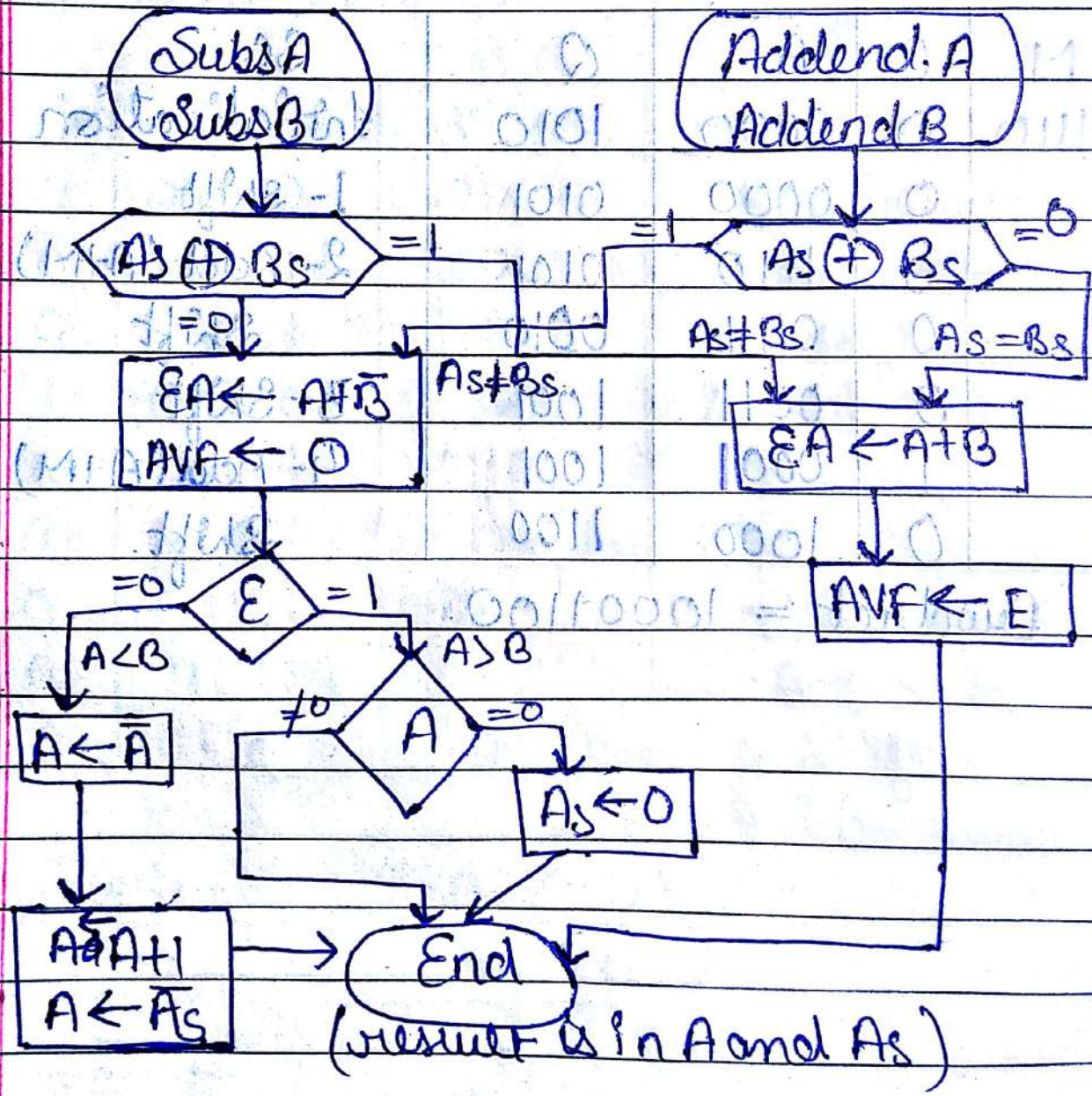


Binary addition and multiplication with sign magnitude example -

operation	Add.	A>B	A<B	A=B
+A + (+B)	+(A+B)			
+A + (-B)		+(A-B)	-(B-A)	+(A-B)
-A + (+B)		-(A-B)	+(B-A)	-(A-B)
-A + (-B)	-(A+B)			
+A - (+B)	+(A+B)	+(A-B)	-(B-A)	+(A-B)
+A - (-B)	+(A+B)			

-A - (+B) -A - (-B)	-(A+B)	-(A-B)	+(B-A)	+(A-B)

★ Flowchart :- (Hardware algorithm).



PAGE NO. // //
DATE // //

Q. Example of Booth multiplication algorithm-

1- 14 × 10.

14 → 1110 — M

10 → 1010 — Q

M	C	A	Q	SC
1110	0	0000	1010	Initialization
	0	0000	0101	1- Shift
	0	1110	0101	2- Add (A+M).
	0	0111	0010	Shift
	0	0011	1001	3- Shift
	1	0001	1001	4- Add (A+M)
	0	1000	1100	Shift.

Product = 10001100

PAGE NO. // //
DATE // //

Multiplication with sign-magnitude (Booth's algorithm)-

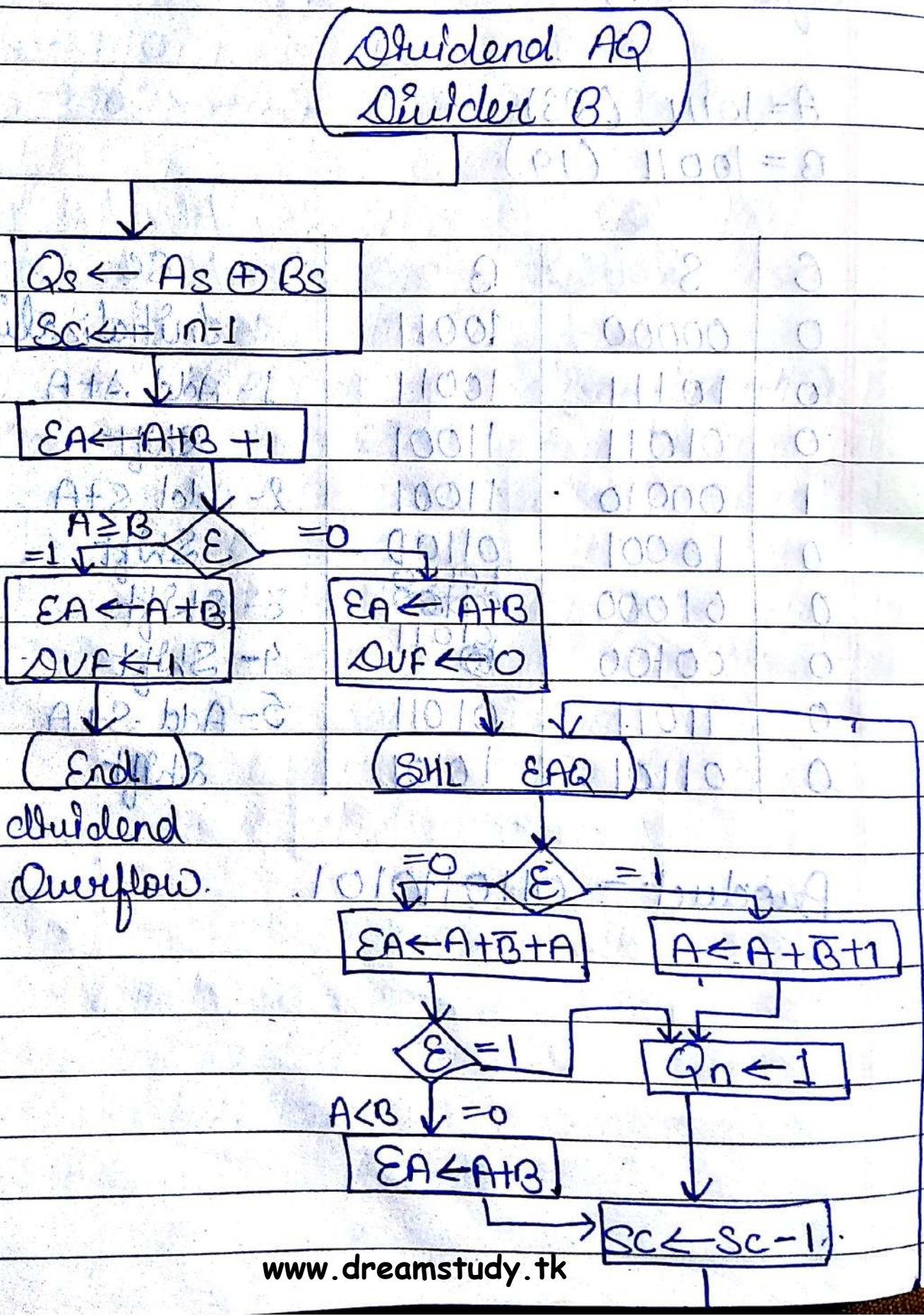
A = 10111 (23)
B = 10011 (19.)

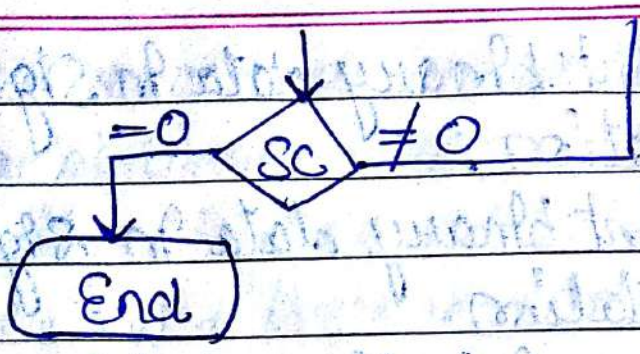
C	S	B	SC
0	00000	10011	Initialization
0	10111	10011	1- Add S+A
0	01011	11001	Shift
1	00010	11001	2- Add S+A
0	10001	01100	Shift
0	01000	00110	3- Shift
0	00100	00011	4- Shift
0	11011	00011	5- Add S+A
0	01101	10101	Shift

Product = 0110110101.

PAGE NO. // //
DATE // //

Binary Division flow-chart :-





★ Introduction:-
Arithmetic Instructions In digital computers manipulate data to produce results necessary for the solutions of computational problems. These Instructions perform arithmetic calculations and are responsible for the bulk of activity Involved In processing data in a computer.

★ What is algorithm?
✓ The Solution to any problem that is stated by a finite number of well-defined procedural steps is called an algorithm.

✓ We develop the various arithmetic algorithms and show the procedure for Implementation them with digital hardware. We consider addition, Subtraction, multi, and division for the following types of data.

PAGE NO. // //
DATE // //

1 ✓ Fixed point binary data in signed-magnitude representation.
2 ✓ Fixed point binary data in signed-2's complement representation.
3 ✓ Floating point binary data.
4 ✓ Binary-coded decimal (BCD) data.

- Addition and Subtraction :-

The addition and subtraction algorithm for data represented in signed magnitude and again data represented in signed-2's complement.

It is important to realize that the adopted representation for negative numbers refers to the representation of numbers in the registers before and after the execution of the arithmetic operations.

- Addition and Subtraction with signed-magnitude data :-

The representation of nos. in signed-magnitude is familiar because it is used in everyday arithmetic calculations.

PAGE NO. // //
DATE // //

The pencil procedure for adding or subtracting two signed binary nos. with papers and pencils is simple and straight-forward. A review of this procedure will be helpful for deriving the hardware algorithm.

- Hardware Implementation:-
To implement the two arithmetic operations with hardware, it is first necessary that the two nos. be stored in registers. Let A and B be two registers that hold the magnitude of the nos, and As and Bs be two flipflops that hold the corresponding signs. The results of the operation may be transferred to a third register. However, a saving achieved if the result is transferred into A and As. Thus A and As together form an accumulator register.
Consider now the hardware implementation of the algorithms above. First, a parallel adder is needed to perform the micro operations A+B.

PAGE NO. // //
DATE // //

Second, comparator circuit is needed to ~~perform the micro~~ establish if A>B, or A<B. third, two parallel subtractor circuits are needed to perform the micro operations A-B and B-A. The sign relationship can be determined from an exclusive-OR gate with As and Bs as input.

★ Flowchart (Hardware algorithm):-

✓ The two signs As and Bs are compared by an exclusive-OR gate. For an add operation, identical signs dictate that the magnitudes be added, for subtract operation different signs dictate that the magnitudes be added. The magnitudes are added with a micro operation EA ← A+B. Where EA is a register that combines E and A.

✓ For A0 indicates that A<B, for this case it is necessary to take the 2's complement of the value in A. this operation can be done with one micro operation A ← $\bar{A}$+1.

- However, we assume that A register as circuits for micro operation: complement and increment, so the 2's compliment is obtain from these two micro operations.
- The value in AVF provides an overflow indication.
- The final value of E is immaterial.

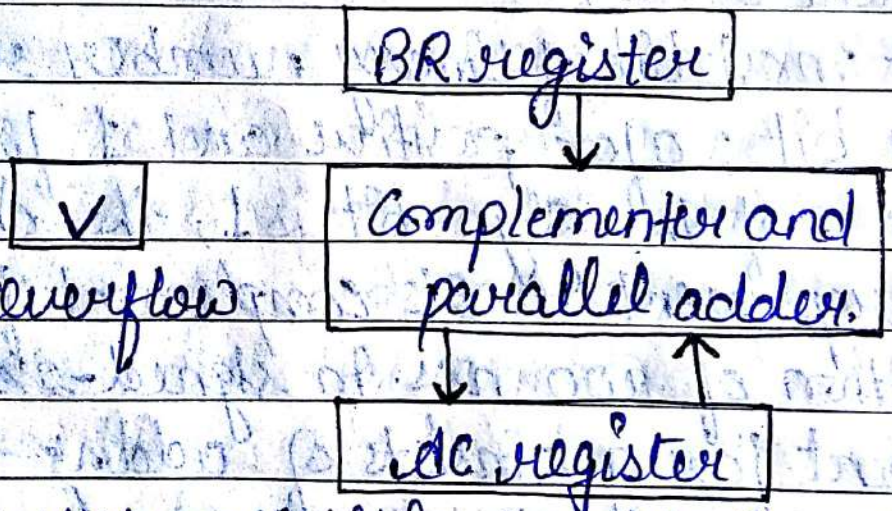
- Addition and Subtraction with Signed 2's Complement data :-

The left most bit of binary number represents the sign bit; 0 for positive and 1 for negative. If the sign bit is 1, the entire no. is represent in 2's complement form. The addition of two nos. in signed-2's complement form consists of adding the nos. with the sign bits treated the same as the other bits of the nos. A carry out of the sign bit position is discarded.

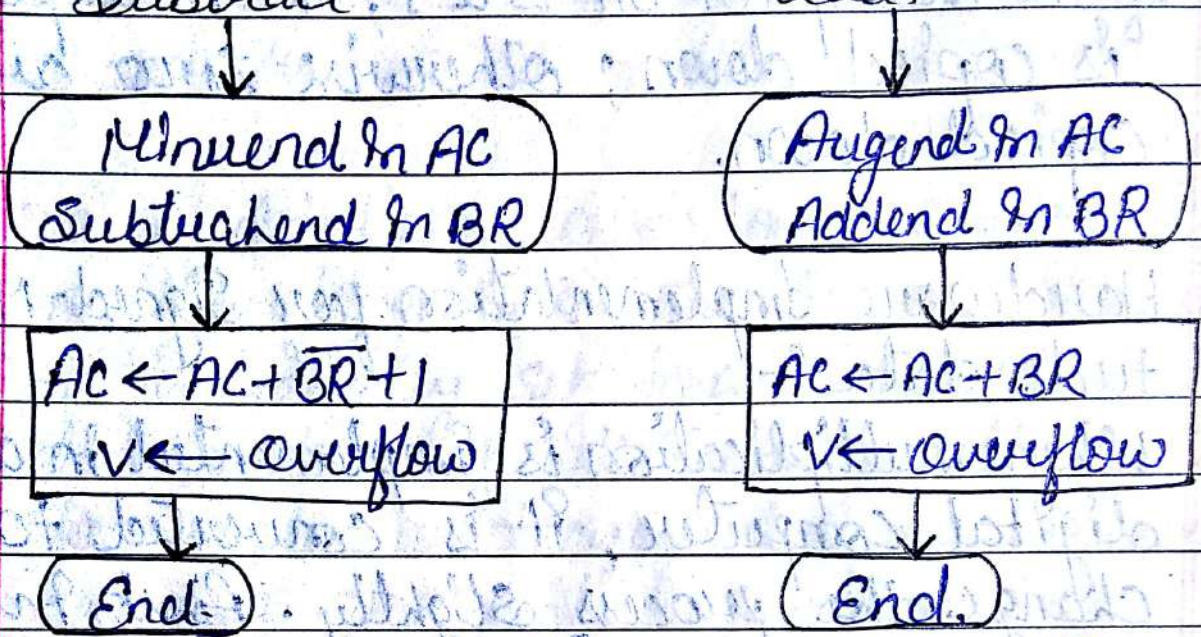
The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.

When two nos. of n digits each are added and the sum occupies n+1 digits, we say that an overflow occured.
When the two carries are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1.

Hardware for Signed-2's Complement addition and Subtraction



→ The left most bit in AC and BR represents the sign bits of the numbers.
→ The over flow flip-flops V is set to 1 if there is an overflow. The output carry in this case is discarded.



Algorithm for adding and subtraction nos. in signed-2's complement representation.

★ Multiplication algorithms:-

Multiplication of two fixed point binary numbers in signed magnitude representation is done with paper and pencil of successive shift and add operation.

$$
\begin{array}{rl}
23 & 10111 \rightarrow \text{Multiplicand} \\
\times\ 19 & \times 10011 \Rightarrow \text{Multiplier} \\
\hline
487 & 10111 \\
 & 10111\times \\
 & 00000\times\times \\
 & 00000\times\times\times \\
 & +\,10111\times\times\times\times \\
\hline
 & 110110101 \quad \text{Product}
\end{array}
$$

If the multiplier bit is a 1, the multiplicand is copied down; otherwise zeros are copied down.

- Hardware Implementation for Signed-Magnitude data :-

→ When multiplication is implemented in a digital computer, it is convenient to change the process slightly. First instead of providing register to store and add simultaneously as many binary nos. as there are bits in the multiplier, as it is convenient to provide an adder for the summation of only two binary nos. and successively accumulate the partial products in a register. Second instead of shifting the multiplicand to the left, the partial product is shifted to the right.

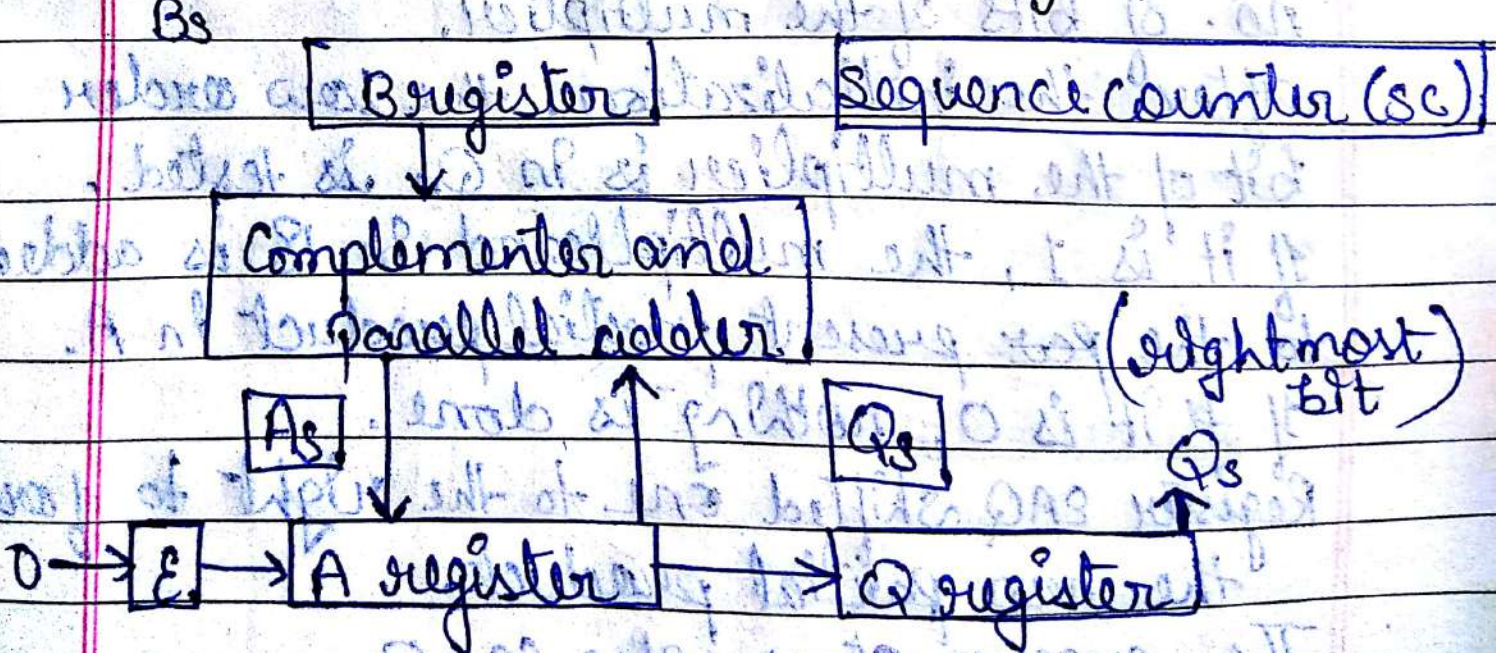
→ The hardware for multiplications consist of the equipment shown in fig. plus two are more registers.

→ These registers are together with registers A and B.

→ The multiplier stored in the Q register and its sign in Q. The Sequence counter SC is initially set to a number equal to the no. of bits in the multiplier. The counter is decremented by 1 after forming each partial product.

→ The Sum of A and B forms a partial product which is transferred to the EA register.

Hardware for multiply operation



PAGE NO. // //
DATE // //

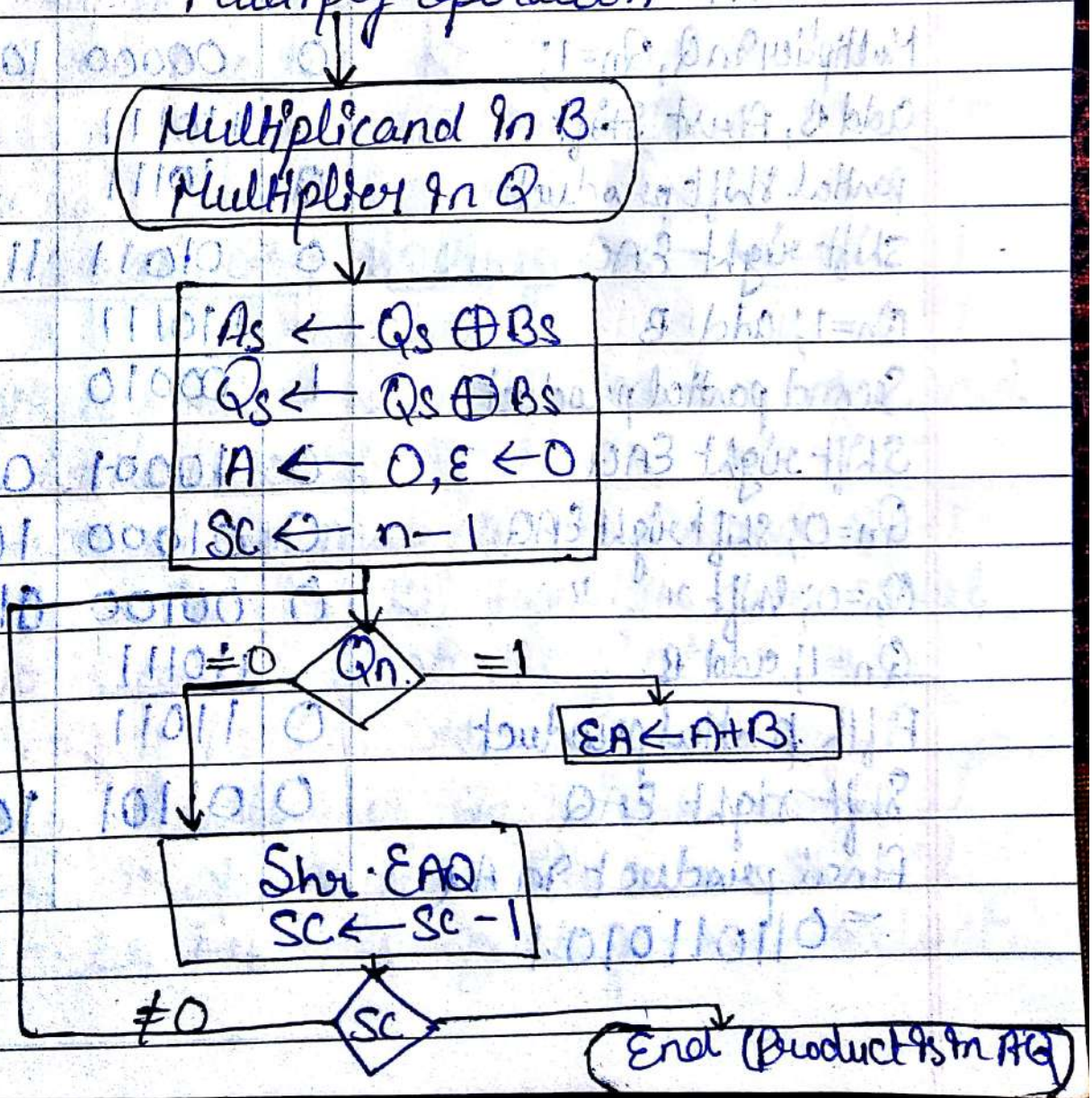
The shift will be denoted by the statement shr EAQ to designate the right shift depicted.
The least significant bit of A is shifted into the most significant position of Q.

- Hardware algorithm :-

Below fig is a flowchart of the hardware multiply algorithm. Initially the multiplicand is ~~in~~ in B and the multiplier in Q there corresponding signs are in Bs and Qs respectively.
Register A and E cleared and the sequence counter SC is set to a no. equal to the no. of bits of the multiplier.
After the initialization, the low order bit of the multiplier is in Qn is tested.
If it is 1, the multiplicand in B is added to the present partial product in A.
If it is 0, nothing is done.
Register EAQ shifted one to the right to form the new partial product.
The process stops when SC = 0.

Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces multiplier. The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

Flowchart for multiply operation.



PAGE NO. // //
DATE // //

- Booth multiplication Algorithm :-

→ Booth algorithm gives a procedure for multiplying binary integers in signed -2's complement representation.

Table → Numerical Example for Binary Multiplier

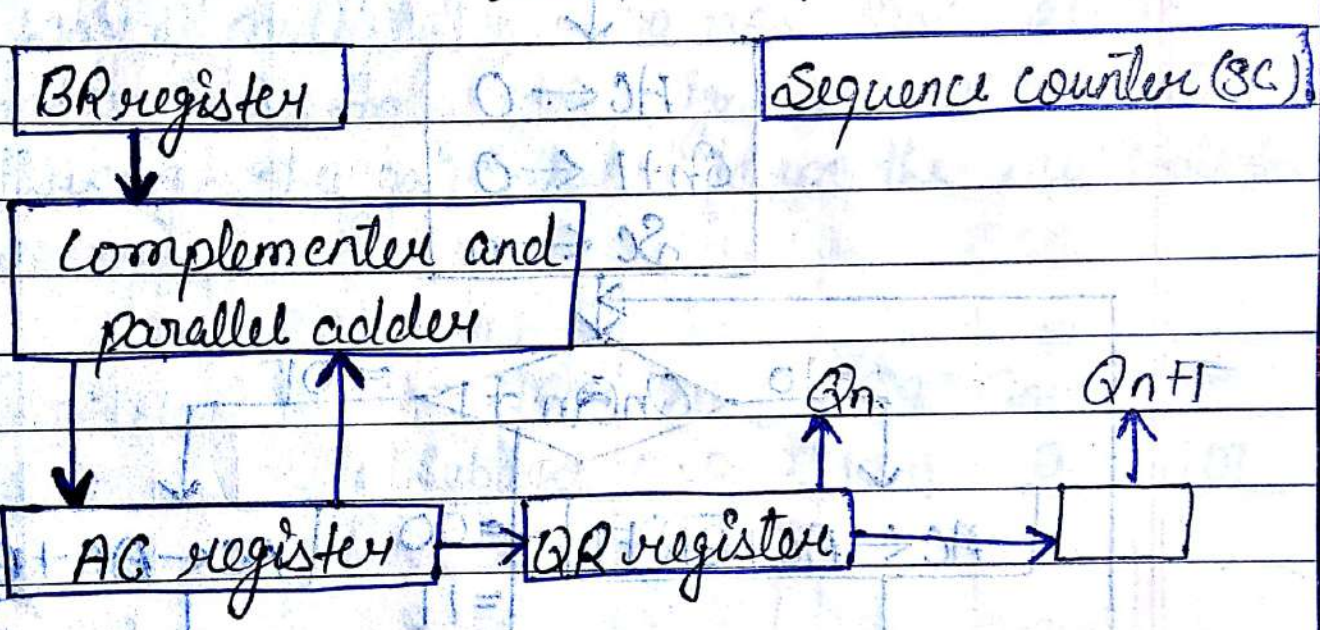
Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q, Qn = 1;	0	00000	10011	101
Add B, first ~~Shift~~		+10111		
partial shift product	0	10111		
Shift right EAQ	0	01011	11001	100
Qn = 1; add B		+10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Qn = 0; shift right EAQ	0	01000	10110	010
Qn = 0; shift " "	0	00100	01011	001
Qn = 1; add B.		+10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 01101101				

PAGE NO. // //
DATE // //

- Hardware for booth algorithm :-

→ The algorithm requires the register configuration as shown in fig.

(Hardware for booth algorithm)

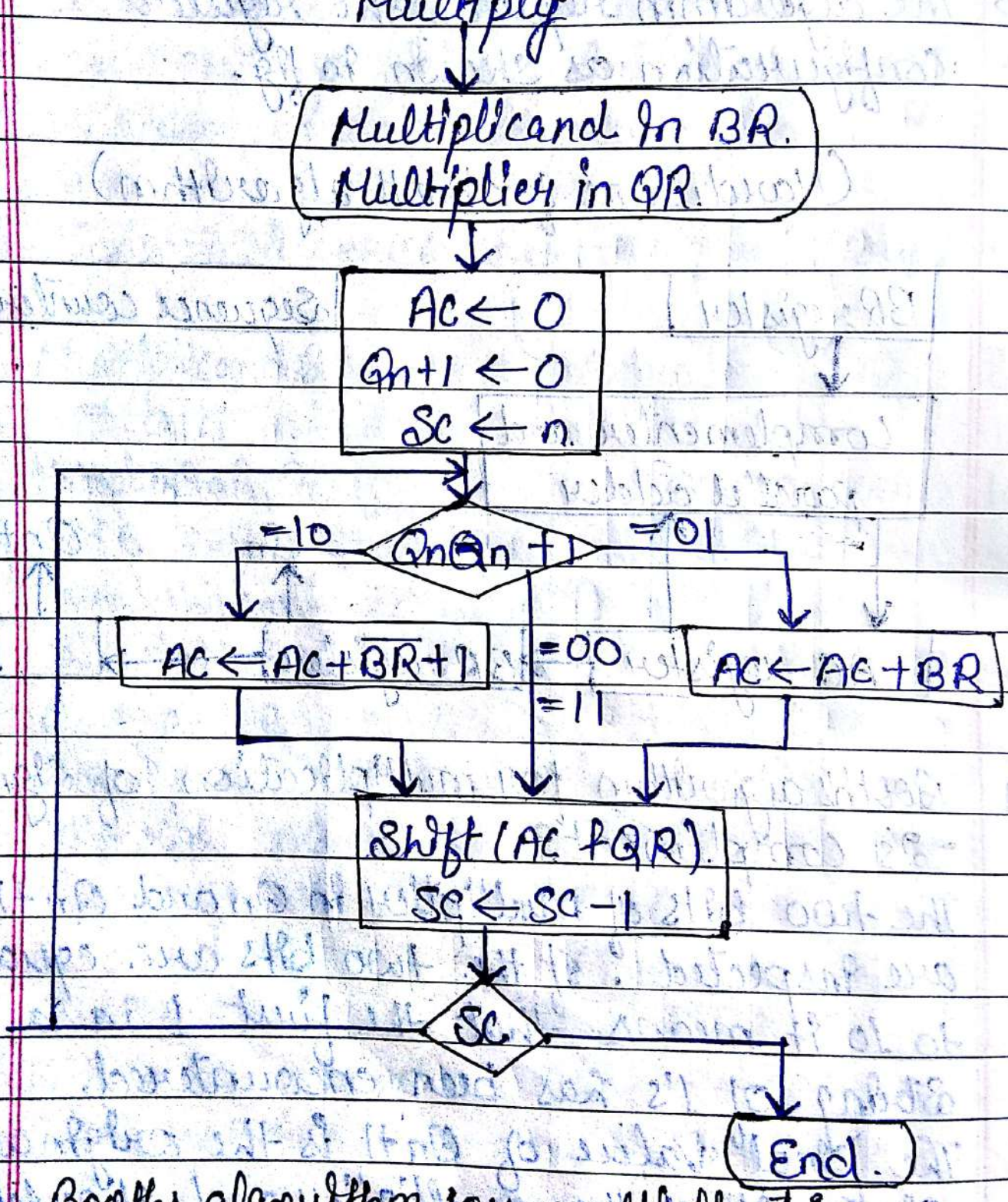


- Booth's algorithm for multiplication of Signed -2's complement :-

The two bits of multiplier in Qn and Qn+1 are inspected. If the two bits are equal to 10 it means that the first 1 in a string of 1's has been encountered. The final value of Qn+1 is the original sign bit of the multiplier and should not be taken as part of the product.

PAGE NO. // //
DATE // //

Multiplication Algorithms.



Booths algorithm for multiplication of Signed - 2's complement nos.

PAGE NO. // //
DATE // //

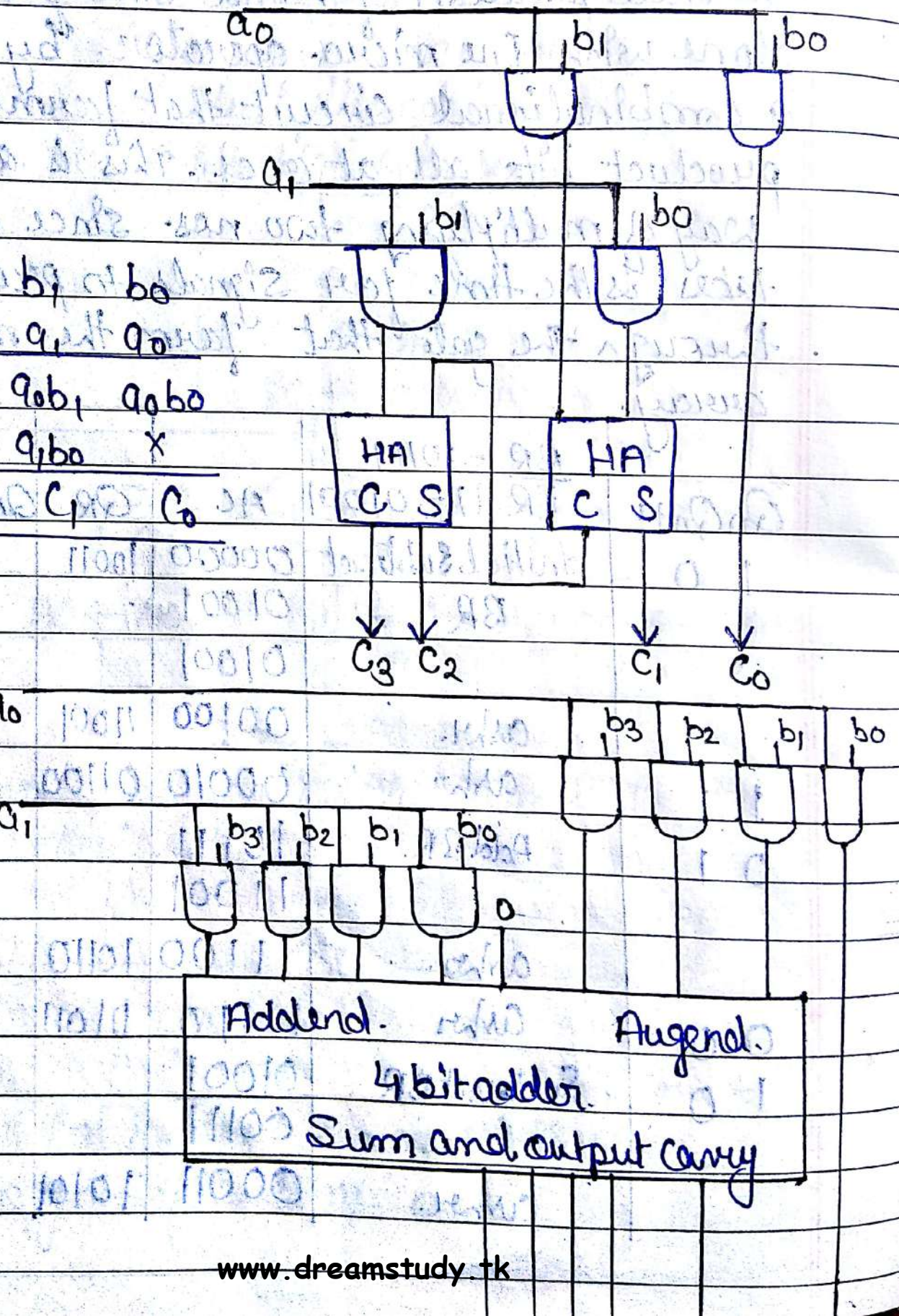
- **Array multiplier :-**

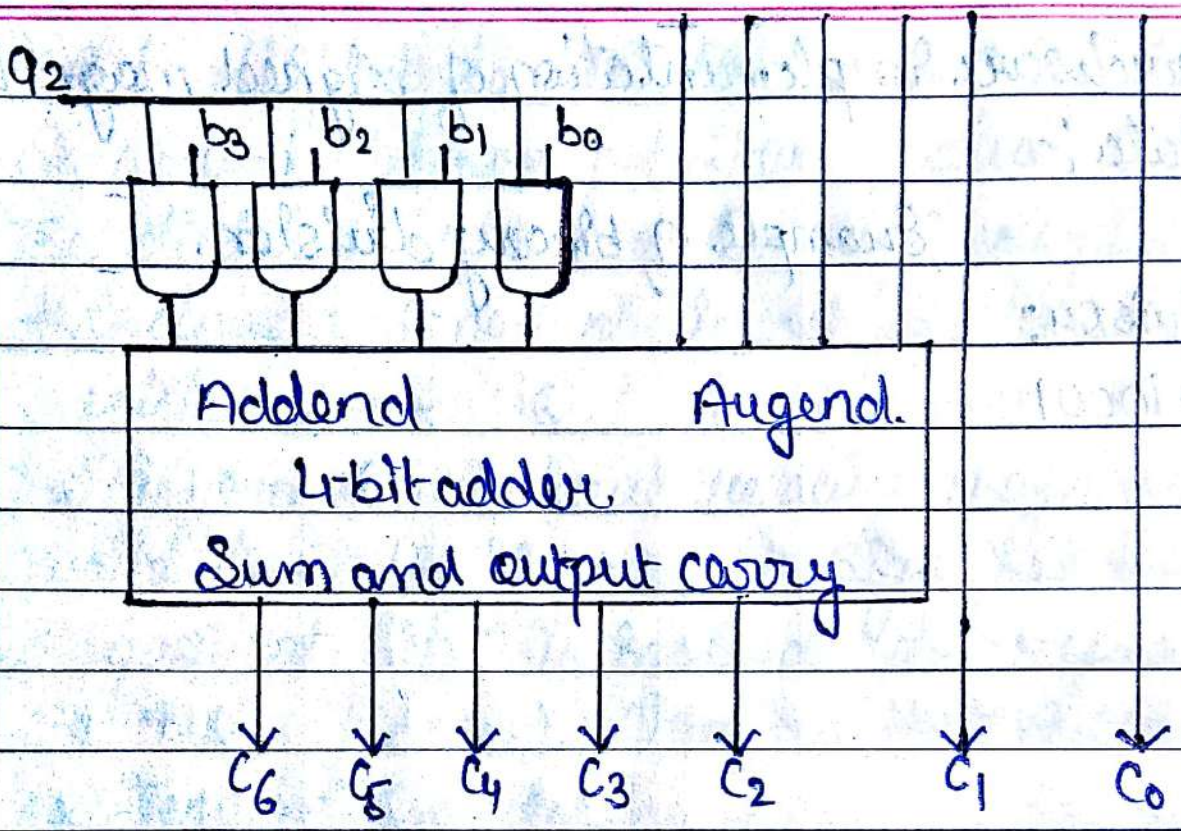
The multiplication of the two binary nos. can be done with one micro-operation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two nos. since all it takes is the time for signals to propagate through the gate that form the multiplication array.

	BR = 10111				
Q_nQ_{n+1}	$\overline{BR}+1 = 01001$	AC	QR	Q_{n+1}	SC.
1 0	Initial Subtract	00000	10011	0	101
↓ ↓	BR	01001			
		01001			
	ashr.	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

PAGE NO. // //
DATE // //

2-bit by 2-bit array multiplier.





★(4bit by 3-bit array multiplier.)

- Division Algorithm :-

→ Division of two fixed-point binary nos. in signed magnitude representation is done with paper and pencil by a process of successive compare, shift, and subtract operations.

A. Example of binary division with digital hardware. Instead of shifting the divisor to the right, the dividend or partial remainder, is shifted to the left, thus leaving the two nos. in the required relative position, subtraction may be achieved by adding A to the 2's compliment of B.

EAQ is shifted to the left with 0 instead of Qn and the previous value of E lost.
The divisor is stored in the B register and the double length dividend is stored in register A and Q.
The information about relative magnitude is available in E. If E=1, it signifies that A≥B. A quotient bit 1 is inserted to repeat the process. If E=0, it signifies that A<B so the quotient in Qn remains a 0.
The sign of the remainder is the same as the sign of the dividend.

Divisor B = 10001, B+1 = 01111

	E	A	Qn	SC
Dividend:	0	01110	00000	5
shl EAQ		11100	00000	
add B+1		+01111		
E=1.	1	01011		
Set Qn=1.	1	01011	00001	4
shl EAQ	0	10110	00010	
Add B+1		+01111		
E=1.	1	00101		
Set Qn=1.	1	00101	00011	3

PAGE NO. " "
DATE " "

Shl EAQ	0	01010	00110	
Add $\bar{B}+1$		+01111		
E=0; leave Q_n=0	0	11001		
Add B		+10001		
Restore remainder	1	01010		2
Shl EAQ		10100	01100	
Add $\bar{B}+1$		+01111		
E=1	1	00011		
Set Q_n=1	1	00011	01101	1
Shl EAQ		00110	11010	
Add $\bar{B}+1$		+01111		
E=0; leave Q_n=0	0	10101	11010	
Add B		+10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Example of binary division with digital hardware.

★ Divide Overflow:-

- This occurs because any dividend will be greater than or equal to zero.

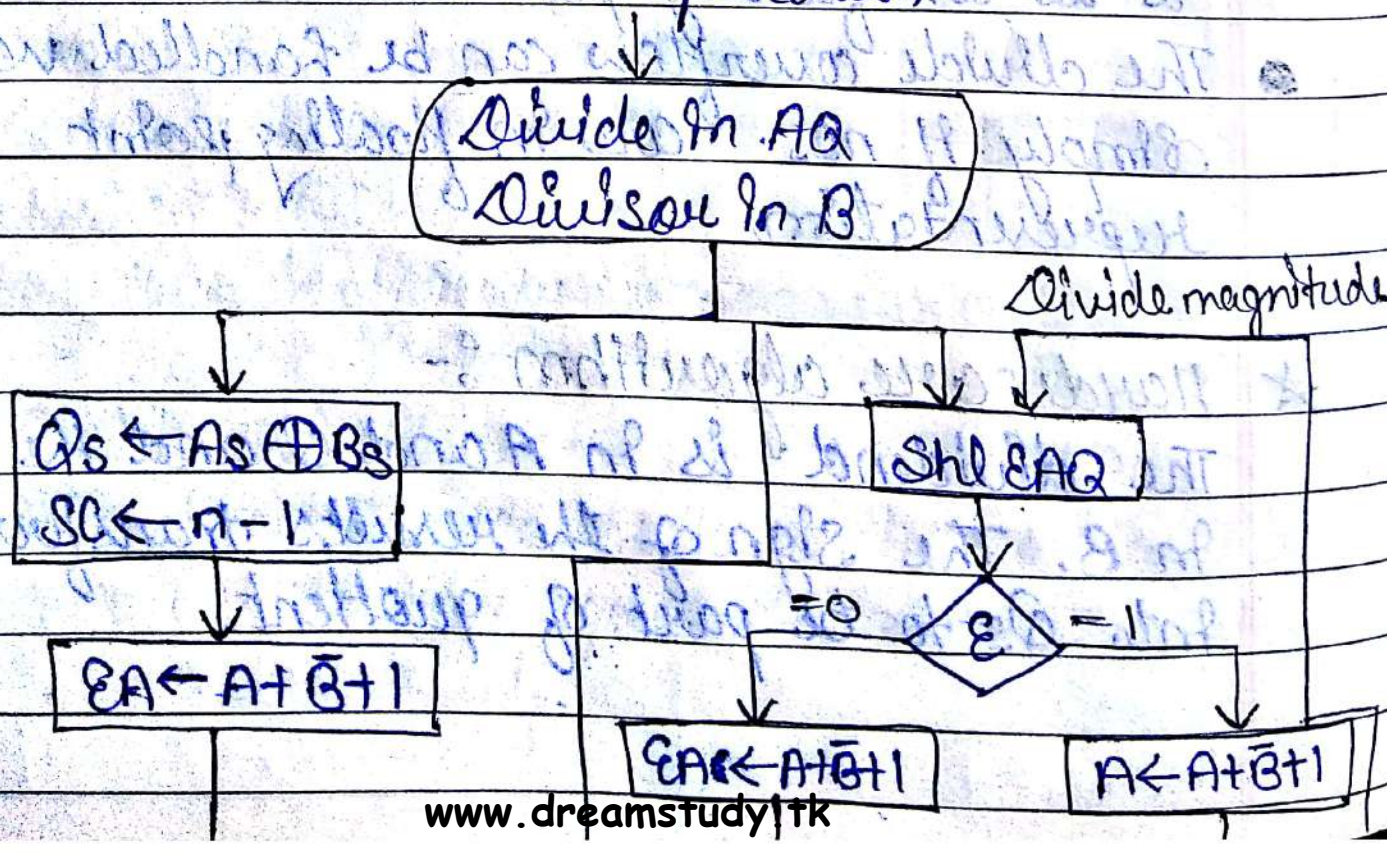
PAGE NO. // //
DATE // //

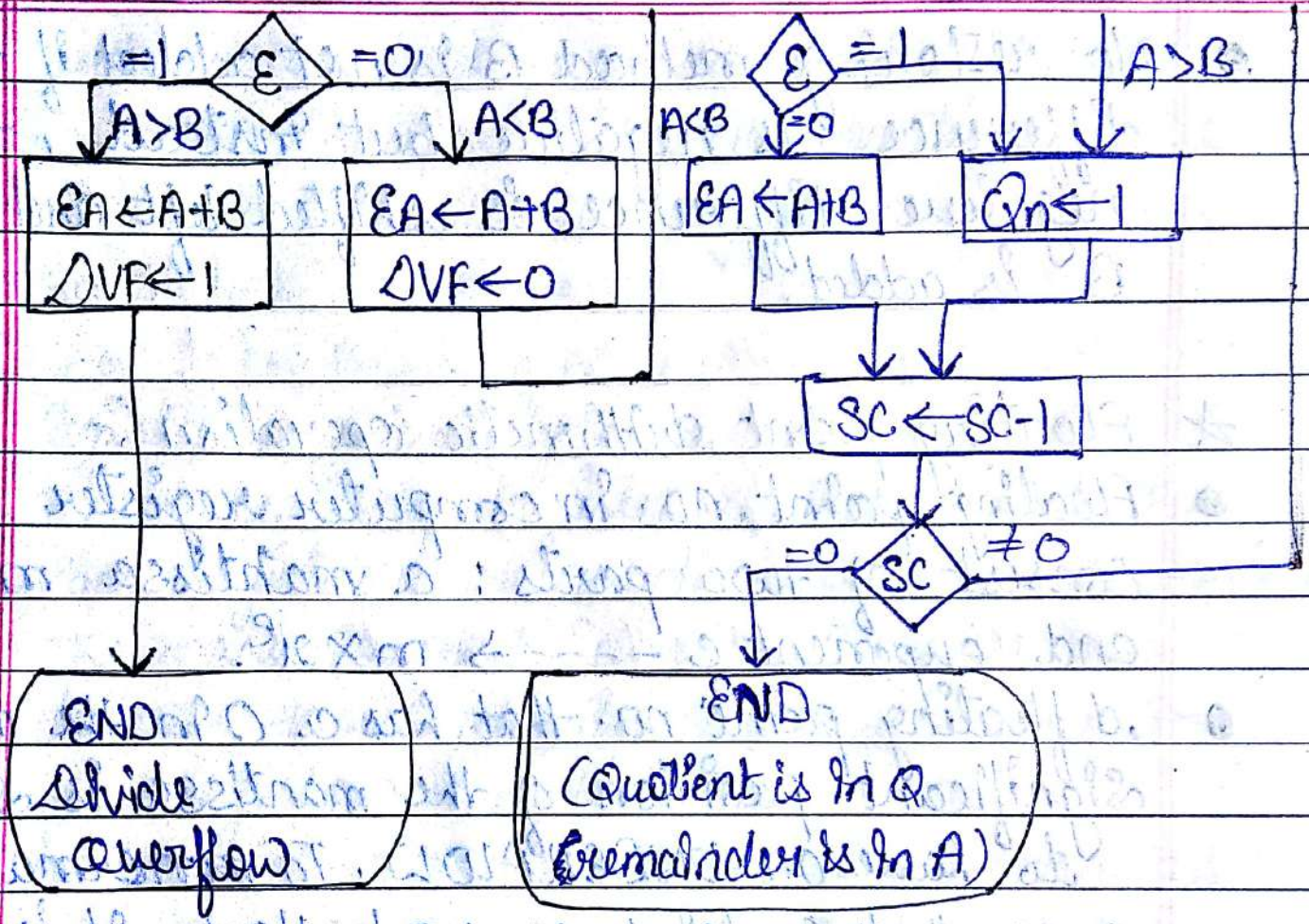
- Over flow condition is usually detected when a special flip-flop is set. which will call it a divide overflow flip-flop and label it DVF.
- The occurience of a divide overflow can be handled in variety of ways.
- In some computers it is the responsibility of the programmers to check if DVF is set after each divide instruction.
- The occurance of a divide overflow stopped the computer and this condition was reffered to as a DIVIDE Stop.
- The best way to avoid a divide overflow is to use floating point data.
- The divide overflow can be handled very simply if nos. are in floating point representation.

★ Hardware algorithm :-
The dividend is in A and Q and the divisor in B. The sign of the results transferred into Qs to be part of quotient.

A divide overflow condition is tested by subtracting divisor in B from half of the bits of the dividend stored in A. If $A \geq B$, the divide overflow flip-flop DVF set and the operation is terminated prematurely. By doing the process as shown in the flowchart the quotient magnitude is formed in register Q and the remainder is found in the register A. The quotient sign is in Qs and the sign of the remainder in As is the same as the original sign of the dividend.

Flowchart for divide operation





★ What is restoring method?

● The hardware method just described is called the RESTORING METHOD. The reason for the name is that the partial remainder is restored by adding the divisor to the negative difference. Two other methods are available for dividing nos., the COMPARISION method and the NONRESTORING method. In the comparision method A and B are compared prior to the subtraction operation.

PAGE NO. // //
DATE // //

- No restoring method B is not added if the difference is negative but instead, the negative difference is shifted left and then B is added.

★ Floating point Arithmetic operation :-

- Floating point no. in computer register consist of two parts : a mantissa m and exponent e ——→ $m \times r^e$.
- A floating point no. that has a 0 in the most significant position of the mantissa is said to have an UNDERFLOW. To normalize a no. that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position.

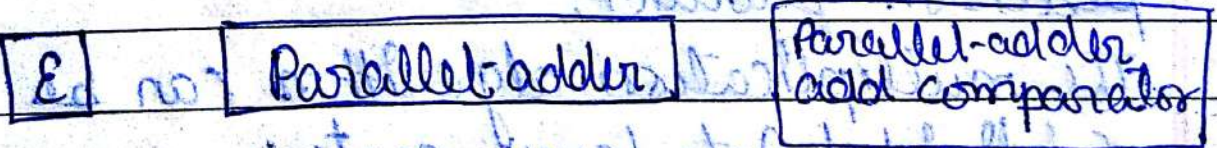
★ Register Configuration :-

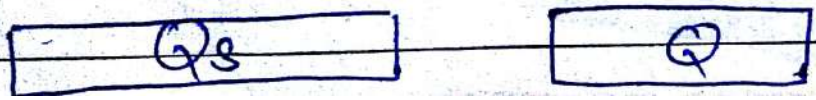
The register configuration for floating point operation is quite similar to the layout for fixed point operation. As a general rule, the same register

and adder used for fixed point arithmetic are used for processing the mantissa, The difference lies in the way the exponents are handled.

★ Register organization :-
There are three registers, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part has same upper case letters, the exponents part uses corresponding lower case letters.
A parallel adder adds the two mantissas and transfers the sum into A and the carry into E. A separate parallel adder is used for the exponents. Since the exponents are biased.

Register for floating point arithmetic operations :-





PAGE NO. // //
DATE // //

★ Addition and Subtraction :-
During addition and subtraction, the two floating point operands are in AC and BR. The sum of difference is formed in the AC. The algorithm can be subdivided into four consecutive parts :-

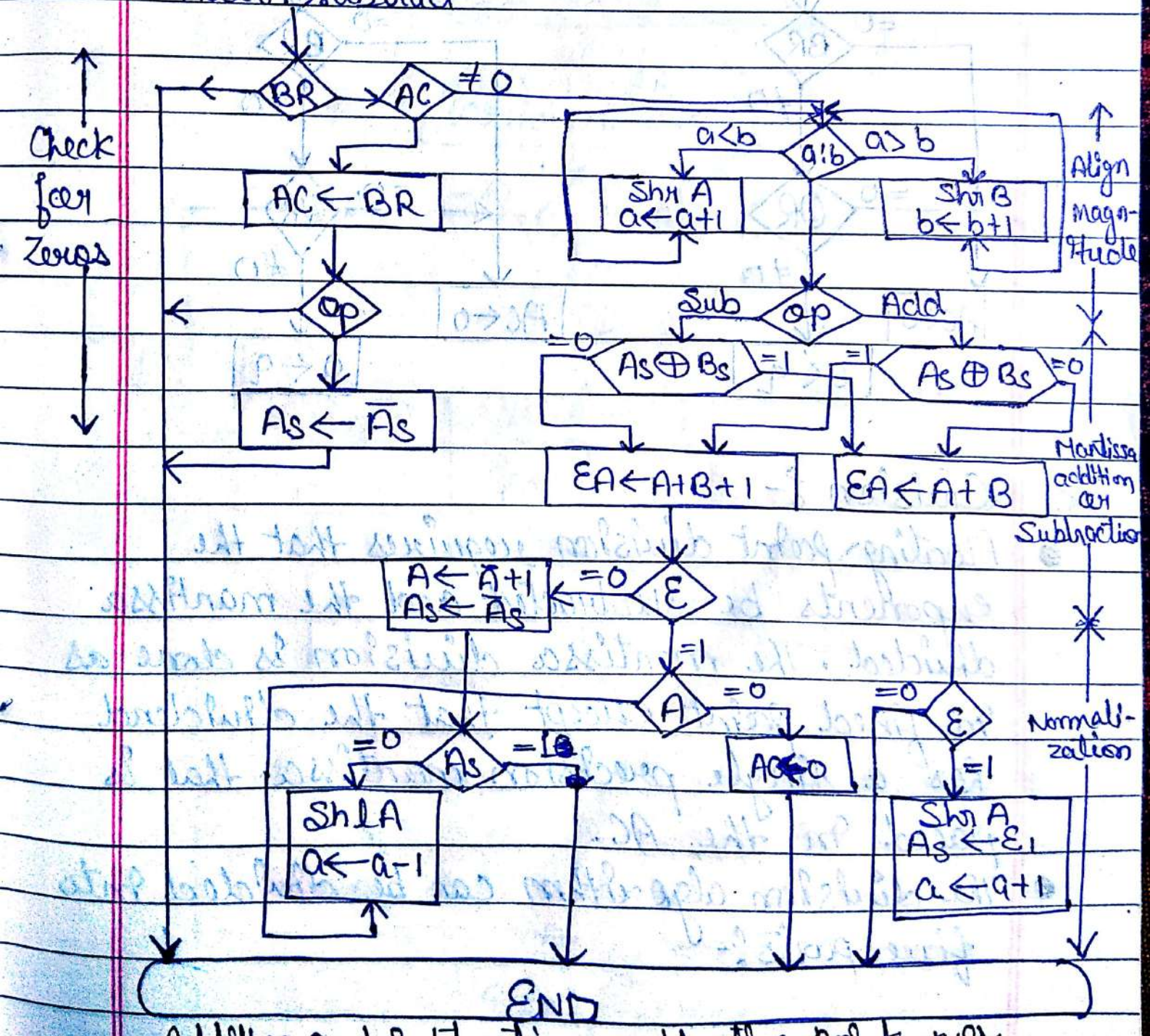
1- Check for zeros.
2- Align the mantissa
3- Add or subtract the mantissa
4- Normalize the result.

Multiplication :-
The multiplication of two floating point nos. requires that we multiply the mantissas and add the exponents. No comparison of exponents or alignment of mantissa is necessary.
The multiplication of the mantissa is performed same as fixed point to provide a double precision product.
The multiplication algorithm can be subdivided into four parts :-

PAGE NO. // //
DATE // //

1- Check for zeros.
2- Add the exponents.
3- Multiply the mantissa.
4- Normalize the product.

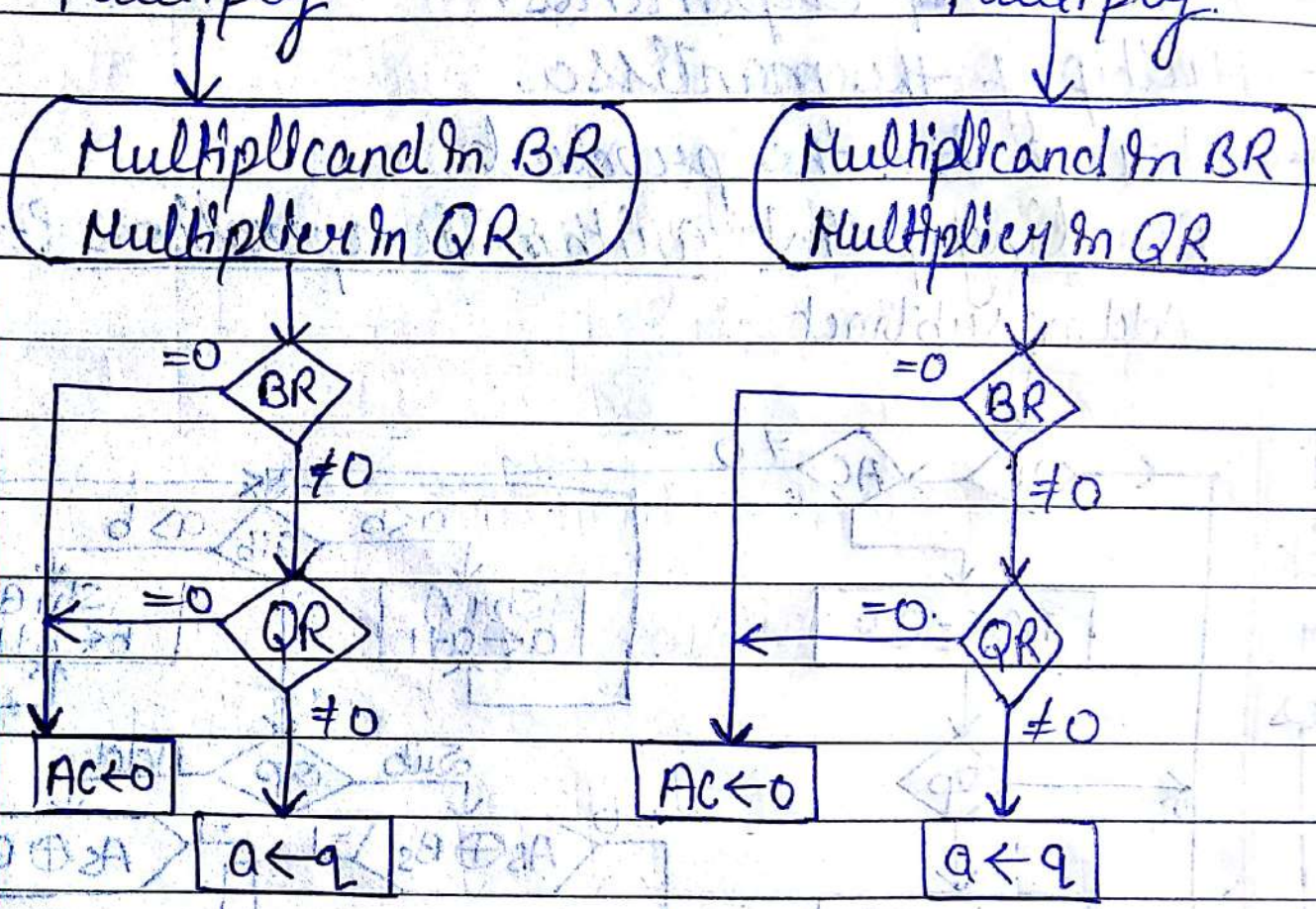
Floating-point arithmetic operations :-



Addition and subtraction of floating point-nos.

PAGE NO. // //
DATE // //

Multiplication of floating point nos.

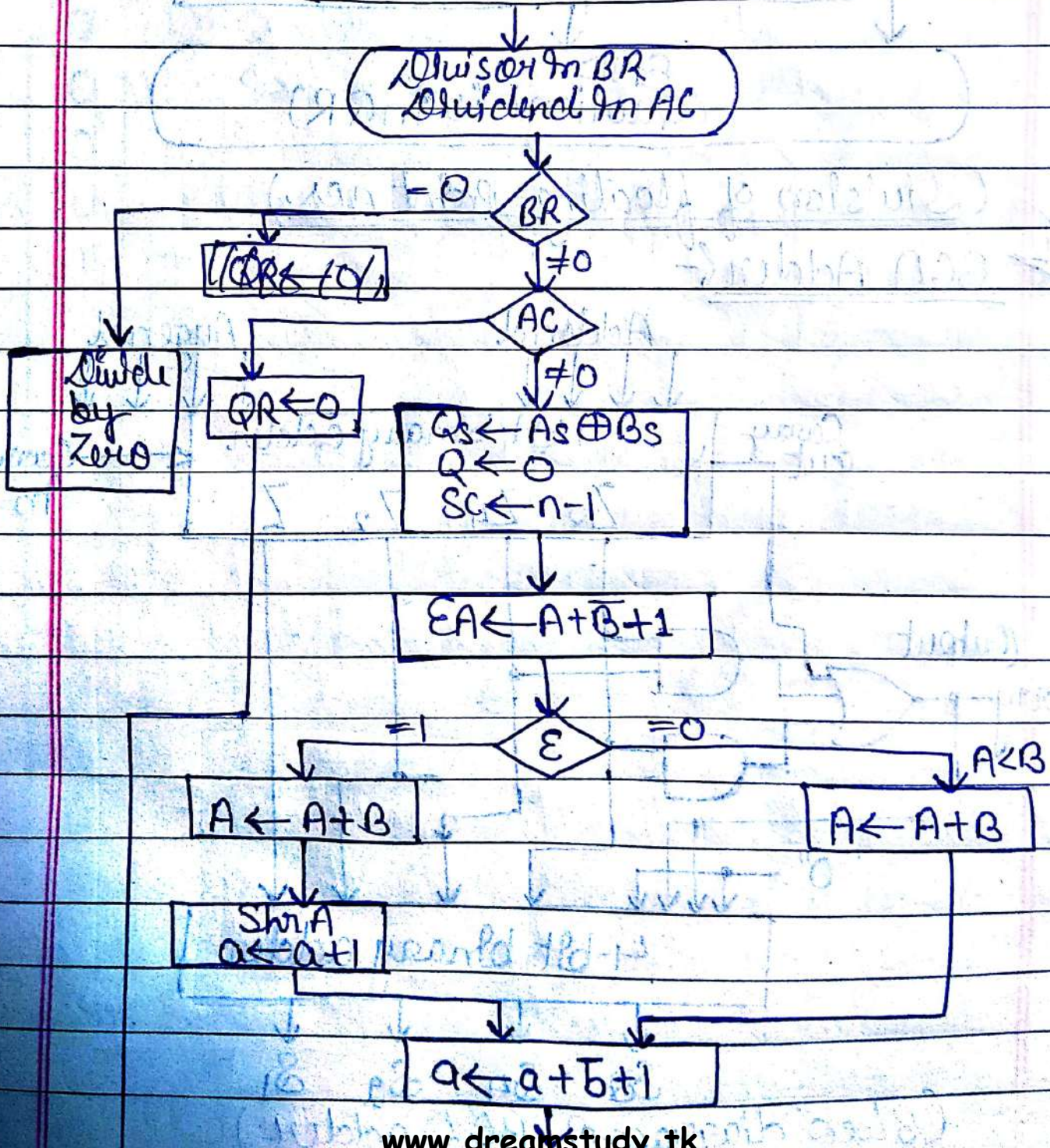


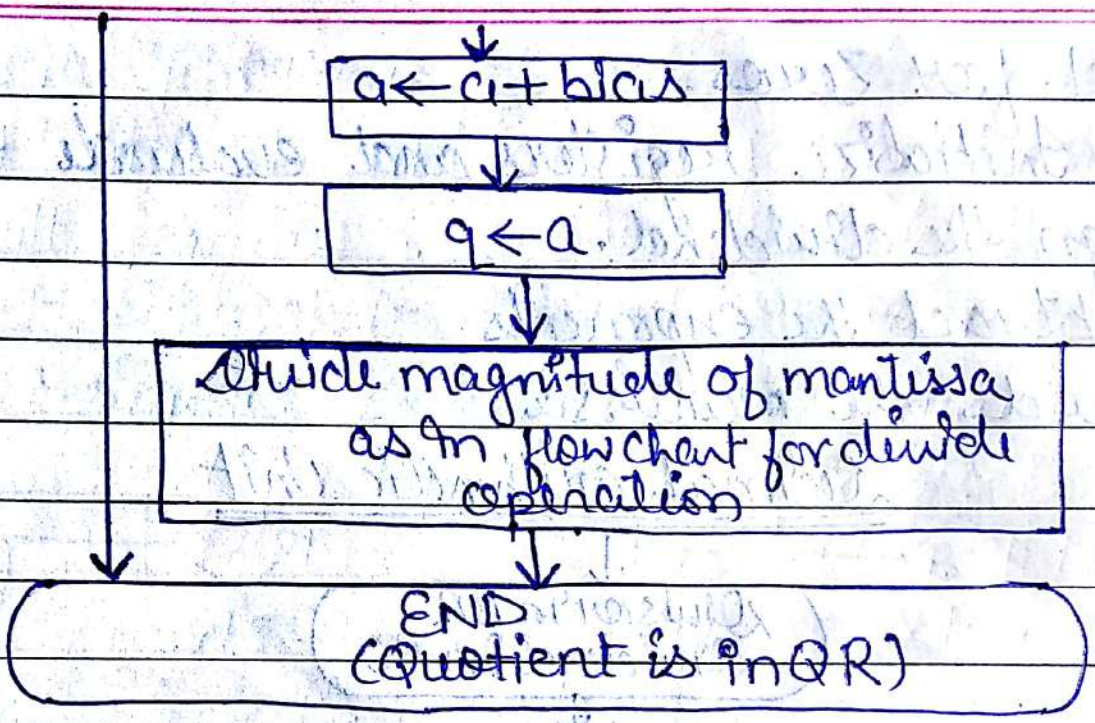
★ Division :-

- Floating point division requires that the exponents be subtracted and the mantissa divided. The mantissa division is done as in fixed point except that the dividend has a single precision mantissa that is placed in the AC.
- The division algorithm can be divided into five parts :-

1- Check for Zeros.
2- Initialize registers and evaluate the sign.
3- Align the dividend.
4- Subtract the exponents.
5- Divide the mantissa.

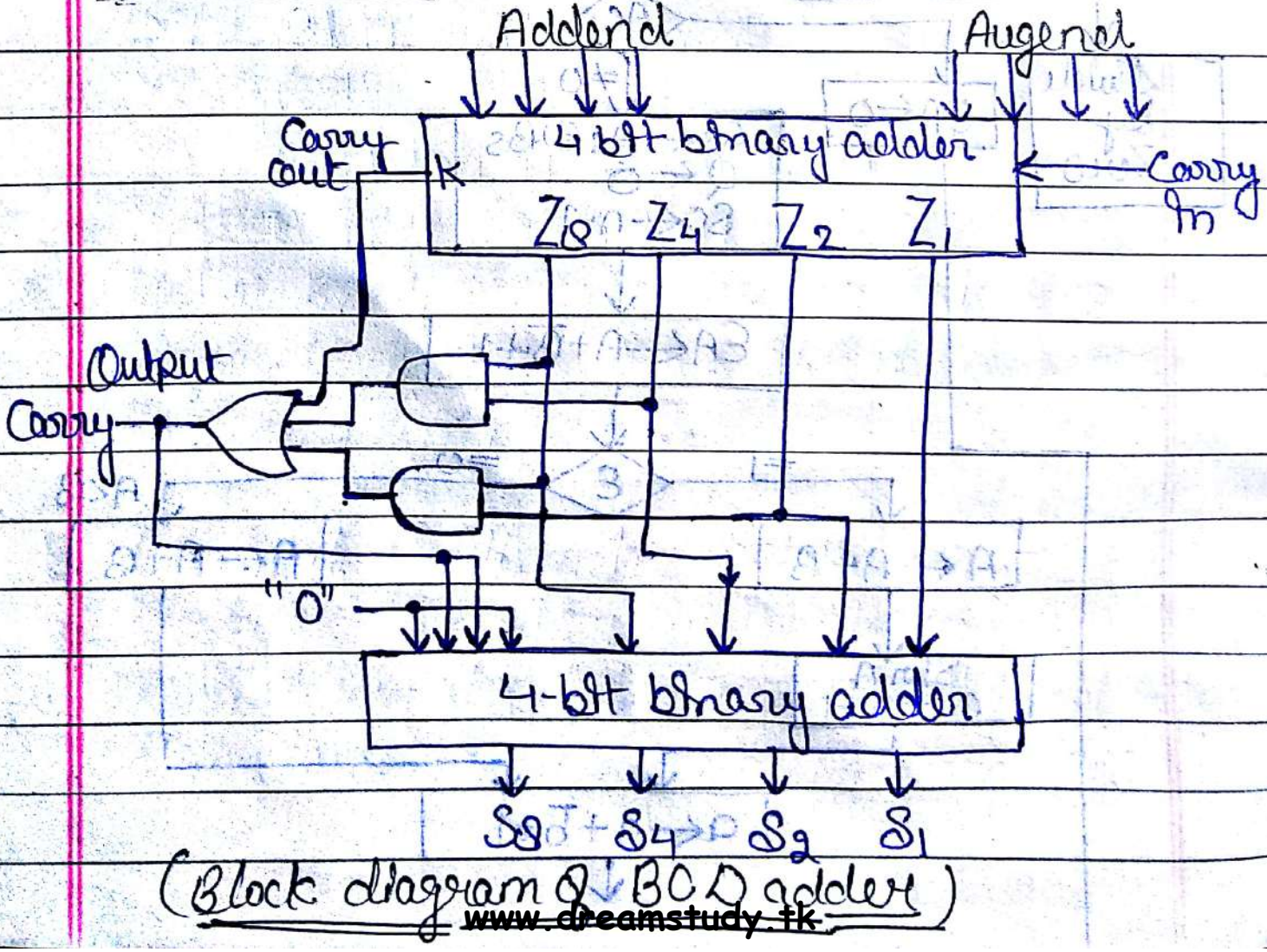
Decimal arithmetic Unit





(Division of floating-point nos.)

★ BCD Adder :-



(Block diagram of BCD adder)

PAGE NO. // //
DATE // //

Table :- Derivation of BCD Adder.

K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal.
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	0	0	0	1	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Binary Sum | BCD Sum

PAGE NO. // //
DATE // //

★ BCD Subtraction :-

1. A straight subtraction of two decimal nos. will require a subtraction circuit that will be somewhat different from a BCD adder.
2. It is more economical to perform the subtraction by taking 9's or 10's complement of the subtrahend and adding it to the minuend. Since the BCD is not a self-complementing code.
3. The Boolean functions for the 9's complement circuit are-

$X1 = B1\,M' + B1'M$

$X2 = B2$

$X4 = B4\,M' + (B4'B2 + B4B2')M$

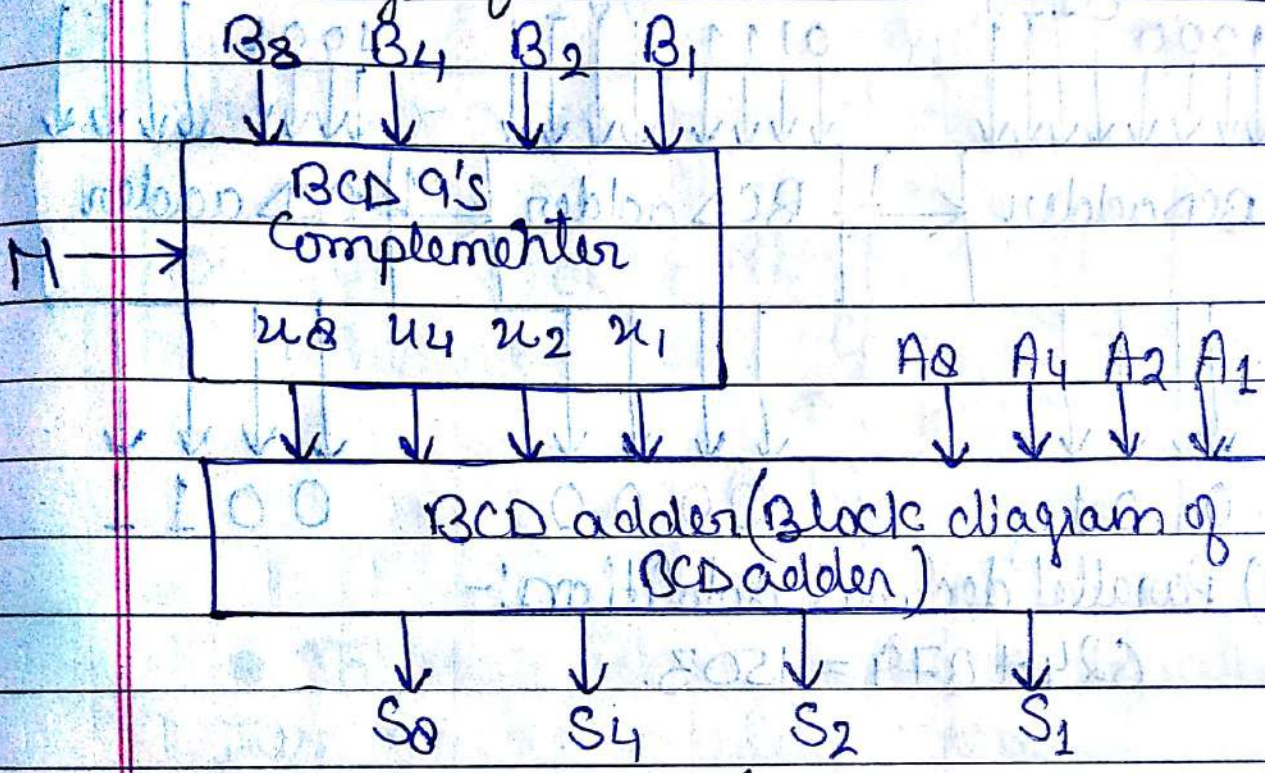
$X8 = B8M' + B8'B4'B2'M$

From these eq^n's we see that X=B when M=0, when M=1, the X output produce the 9's complement of B.

★ Decimal Arithmetic operation :-

- The algorithm for arithmetic operations with decimal data are similar to the algorithms for the corresponding operations with binary data.

One stage of a decimal arithmetic unit.

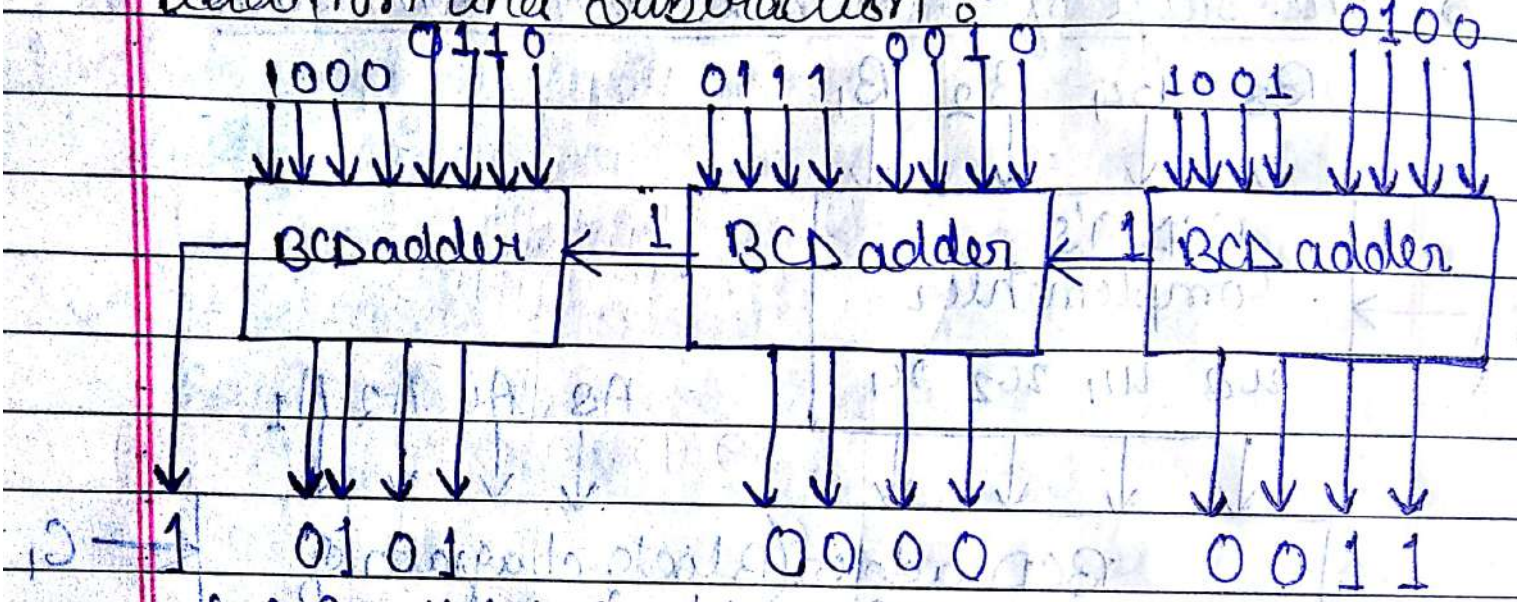


★ Decimal arithmetic micro operation symbols

Table:- Decimal arithmetic Microoperation Symbols

Symbolic designation	Description.
$A \leftarrow A + B$	Add decimal nos. and transfer sum into A
$\bar{B}$	9's complement of B.
$A \leftarrow A + \bar{B} + 1$	Content of A plus 10's complement of B into A.
$Q_1 \leftarrow Q + 1$	Increment BCD no. in Q_1
dshr A	Decimal shift-right register A
dshl A	Decimal shift-Left register A.

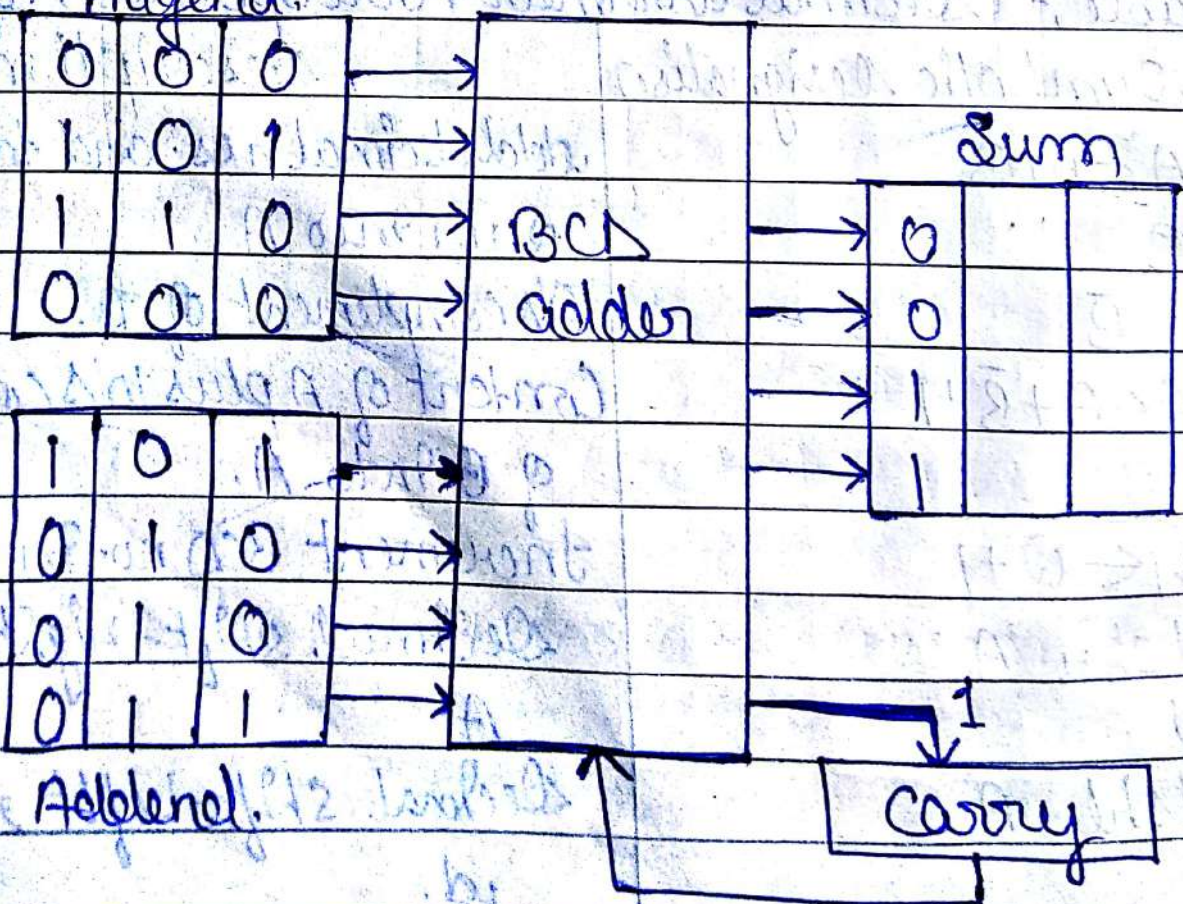
Addition and subtraction :-



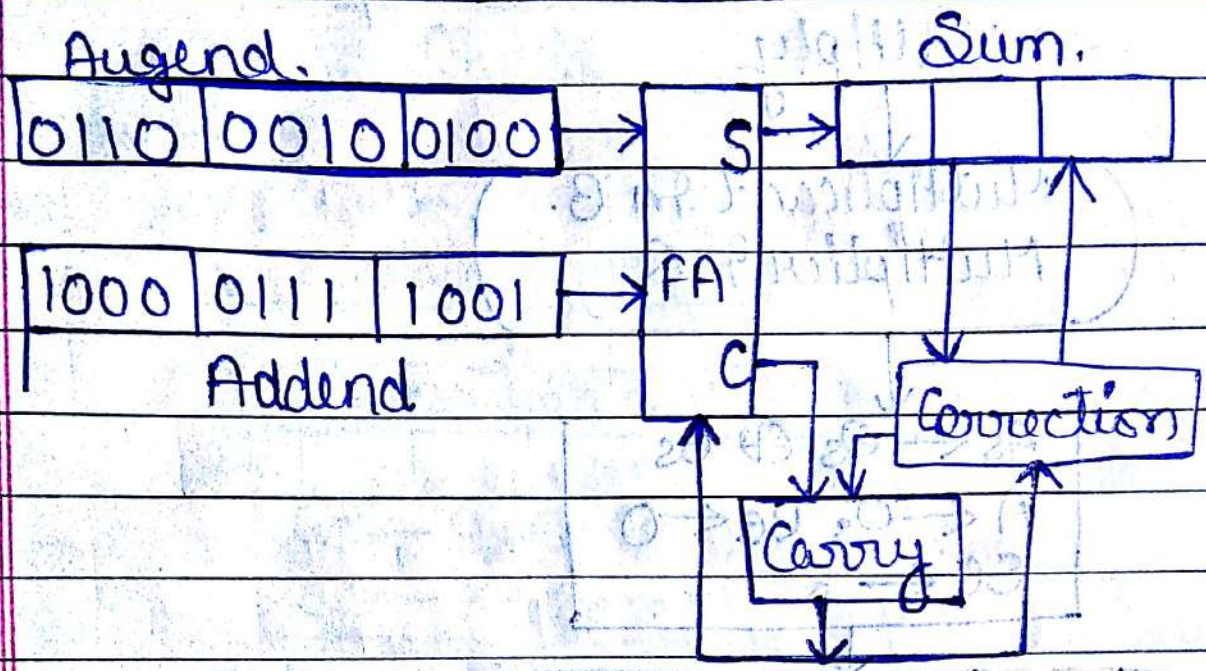
(a) Parallel decimal addition :-

624 + 879 = 1503.

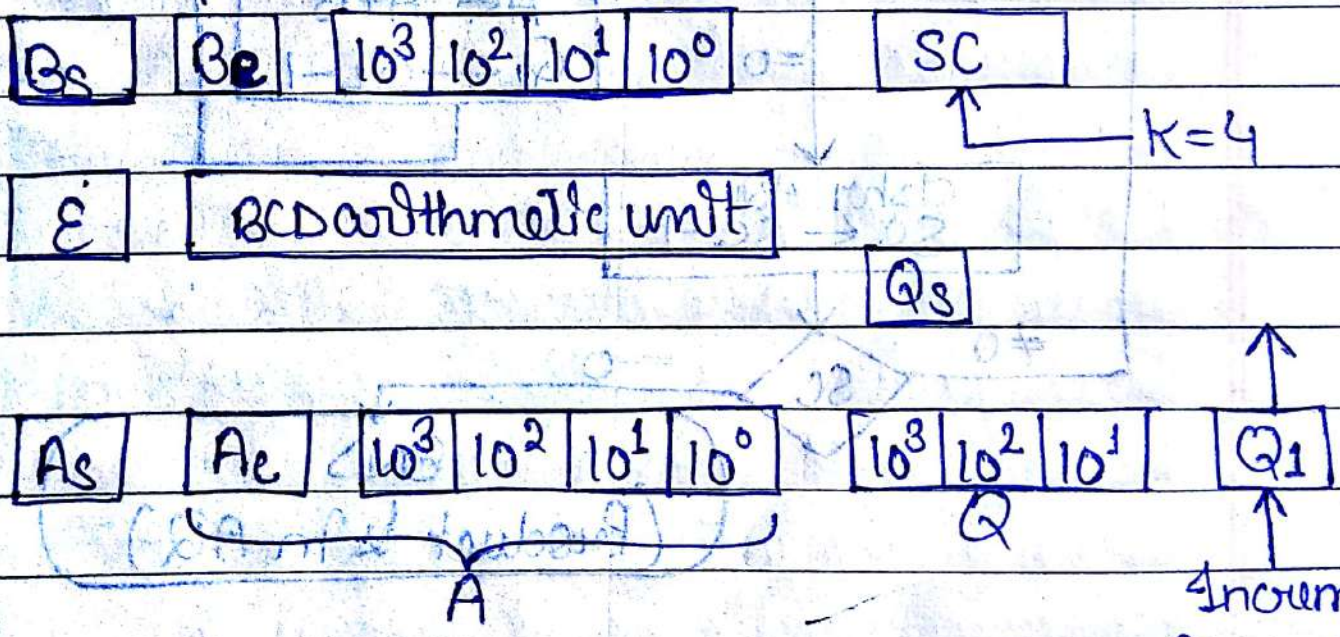
(b) Eight-serial, bit parallel decimal addition



(c) Bll serial decimal addition.



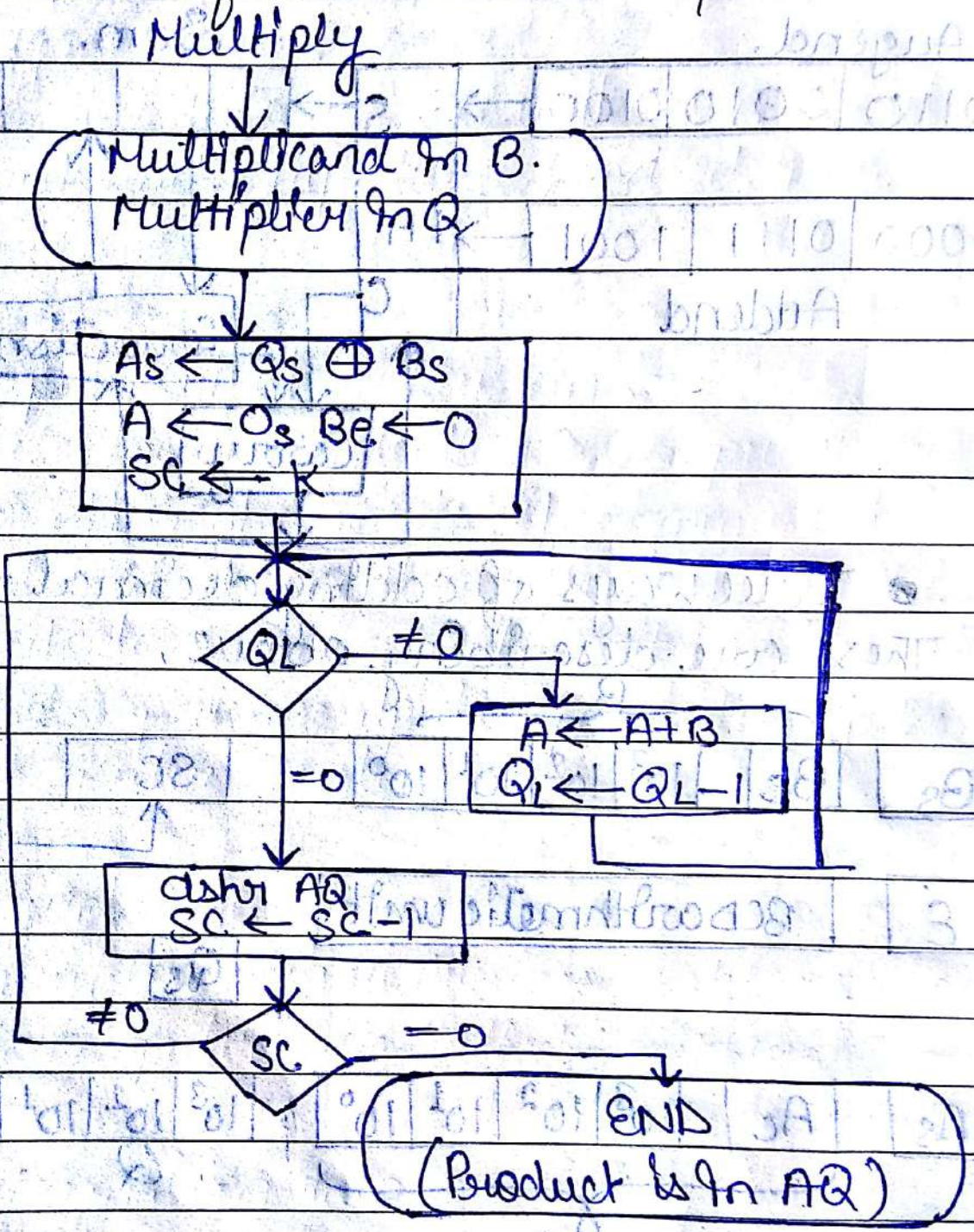
- Three ways of adding decimal nos. There are describing above.



☆ Registers for decimal arithmetic multiplication and division.

PAGE NO. // //
DATE // //

★ Flowchart for decimal multiplication.



★ Flowchart for decimal division :-

PAGE NO. // //
DATE // //

Division

Divide