1d). Why should the sign of the remainder after a division be the same as the sign of the dividend?

Ans: The sign of the remainder after a division is the same as the sign of the dividend to avoid ambiguity and to ensure that the dividend and remainder have the same signs no matter what the signs of the divisor and quotient

2d). What is the purpose of the Decimal Arithmetic Unit?

Ans:

The user of a computer prepares data with decimal numbers and receives results in decimal form. A CPU with an arithmetic logic unit can perform arithmetic microoperations with binary data. To perform arithmetic operations with decimal data, it is necessary to convert the input decimal numbers to binary, to perform all calculations with binary numbers, and to convert the results into decimal. This may be an efficient method in applications requiring a large number of calculations and a relatively smaller amount of input.

A decimal arithmetic unit is a digital function that performs decimal microoperations. It can add or subtract decimal numbers, usually by forming the 9's or 10's complement of the subtrahend. The unit accepts coded decimal numbers and generates results m the same adopted binary code.

A single-stage decimal arithmetic unit consists of nine binary input variables and five binary output variables since a minimum of four bits is required to represent each coded decimal digit. Each stage must have four inputs for the augend digit, four inputs for the addend digit, and an input-carry. The outputs include four terminals for the sum digit and one for the output-carry

2e). State the addition algorithm.

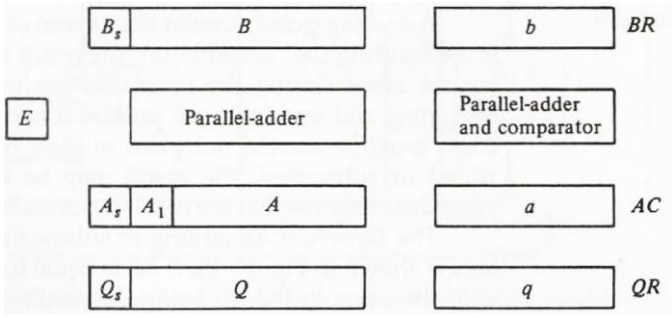
Ans: Available in notes in campx

3b). Explain the register organization for floating point operations.

Ans:

Register Configuration:

- Register organization of floating-point arithmetic consists of 3 registers A, B & Q. Register Q is used in multiplication an division.
- Each of them is divided into two parts.
    - The mantissa part has the same uppercase letter as we used in fixed-point arithmetic.
    - The exponent part used the lowercase letter as shown below



- Assuming that the floating-point number has mantissa in signed magnitude form and a biased exponent.
  - Register A has mantissa whose sign is in A_s, exponent in lowercase letter a.
  - The MSB of A is designated as A_1. A_1 =1 for the number to be normalized.
  - Register B has mantissa whose sign is in B_s, exponent in lowercase letter b.
  - Register Q has mantissa whose sign is in Q_s, exponent in lowercase letter q.
- The parallel adder adds the two numbers and stores the sum in A and the carry-in E
- Separate parallel adders are used to add the exponents.
- Also, the exponents are connected to the comparator.
- The number in mantissa will be taken as a fraction, so the binary point resides to the left of the magnitude part.
- Initially, the numbers in the registers are normalized. After each arithmetic operation, the results are normalized. Thus, the number coming in and going out of registers is always normalized.

3e). Explain BCD Adder.

Ans:

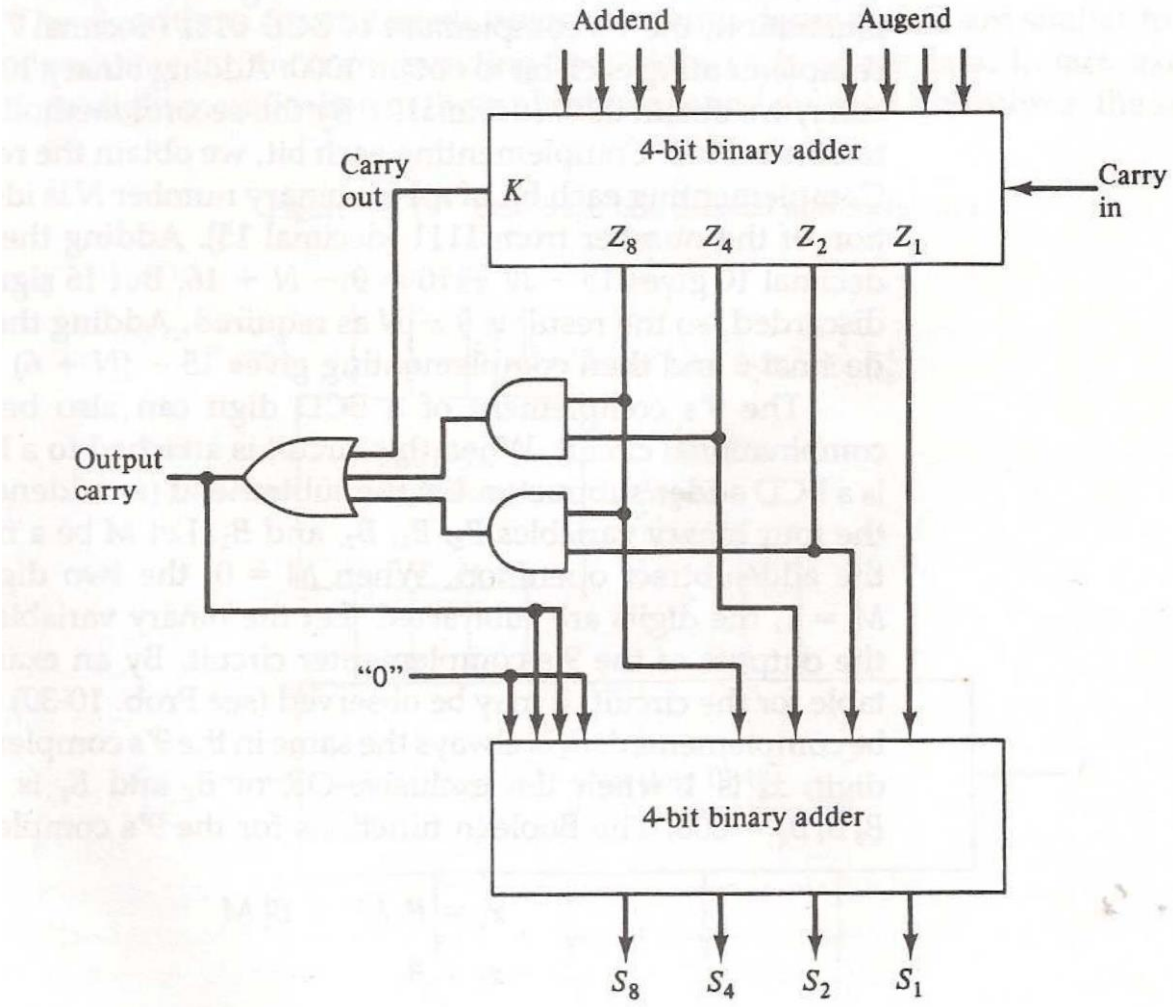
Consider the arithmetic addition of two decimal digits in BCD, together with a possible carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = *19,* the 1 in the sum being an input-carry. Suppose that we apply two BCD digits to a 4-bit binary adder. The adder will form the sum m *binary* and produce a result that may range from o to 19. These binary numbers are listed in Table below and are labeled by symbols *K,* Z8., *Z4, Z2,* and Z1. *K* is the carry and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code. The first column in the table lists the binary sums as they appear in the outputs of a 4-bit *binary* adder. The output sum of two *decimal* numbers must be represented in BCD and should appear in the listed in the second column of the table. The problem is to find a simple rule by which the binary number in the first column can be converted to the correct BCD digit presentation of the number in the second column.

Binary Sum					BCD Sum					
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	1	0	0	0	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

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than IDOl, the corresponding BCD number is identical and therefore no conversion is needed. When the binary sum is greater than 1001, we obtain a nonvalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required. One method of adding decimal numbers in BCD would be to employ one 4-bit binary adder and perform the arithmetic operation one digit at a time. The low-order pair of BCD digits is first added to produce a binary sum. If the result is equal or greater than 1010, it is corrected by adding 0110 to the binary sum. This second operation will automatically produce an output-carry for the next pair of significant digits. The next higher-order pair of digits, together with the input-carry, is then added to produce their binary sum. If this result is equal to or greater than 1010, it is corrected by adding 0110. The procedure is repeated until all decimal digits are added.

The logic circuit that detects the necessary correction can be derived from the table entries. It is obvious that a correction is needed when the binary sum has an output carry K = 1. The other six combinations from 1010 to 1111 that need a correction have a 1 in position 2,. To distinguish them from binary 1000 and 1001 which also have a 1 in position 2" we specify further that either 2 or Z2 must have a 1. The condition for a correction and an output-carry can be expressed by the Boolean function When C =1, it is necessary to add 0110 to the binary sum and provide an output-carry for the next stage. A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD. A BCD adder must include the

correction logic in its internal construction. To add 0110 to the binary sum, we use a second 4-bit binary adder as shown in Fig below The two decimal digits, together with the input-carry, are first added in the top 4-bit binary adder to produce the binary sum. When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to I, binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output-carry generated from the bottom binary adder may be ignored since it supplies information already available in the output-carry terminal.



4b). Explains Booths Multiplication Algorithm with a numerical example

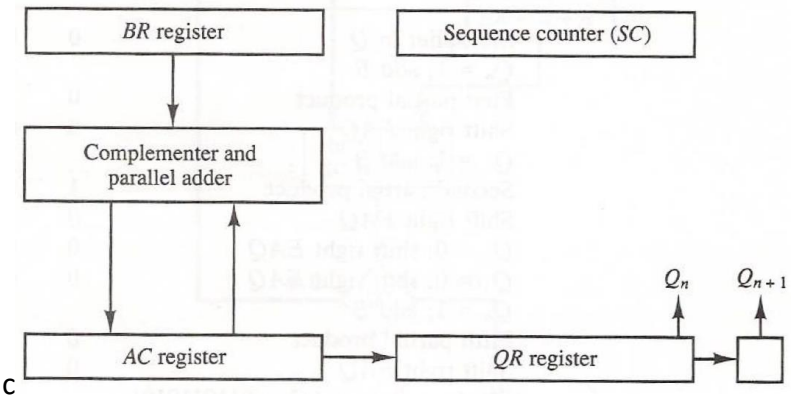
Ans:

Booths Algorithm requires examination of multiplier bits and the shifting of the partial product. But, prior to the shifting the multiplicand me be added or subtracted with the partial product or may be left unchanged based on following conditions.

i. **The multiplicand is subtracted from the partial product upon encountering the first least significant 1 from the string of 1's in multiplier.**
ii. **The multiplicand is added to the partial product upon encountering the first 0(provided there was a previous 1) in a string of 0's in the multiplier.**
iii. **The partial product do not change when the multiplier bit is identical to previous multiplier bit.**

Hardware implementation of Booths Algorithm.

The hardware circuit is as shown below



C

The circuit consist of parallel adder and a complementor to perform the addition and the subtraction operations. The multiplicand is store in the BR register and the multiplier in QR register. AC alone loads the partial product and AC along with QR is used to store the product term. Along with LSB(Q_n) of QR, flip-flop Q_{n+1} is added to check the double bit inspection of the multiplier. The Sequence Counter (SC) is set to the binary value equal to the number of bits in the multiplier.

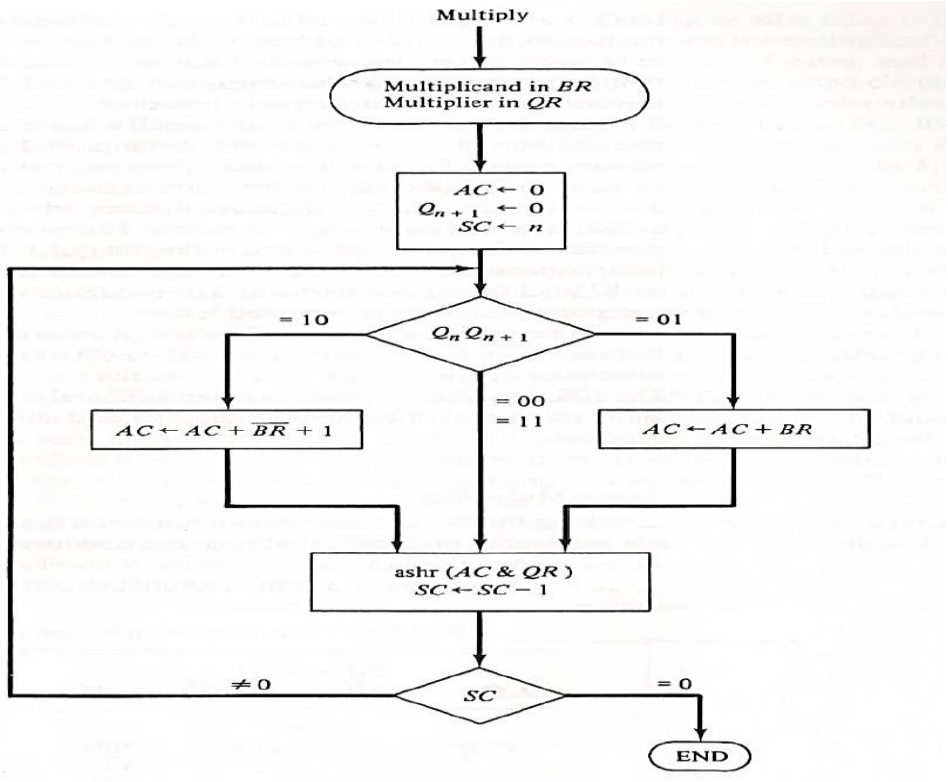
When Q_n Q_{n+1} = 10 implies that the first 1 from the sting of 1's is encountered. This requires a subtraction of multiplicand with the partial product in AC.

When Q_n Q_{n+1} = 01 implies that the first 0 from the sting of 0's is encountered. This requires addition of multiplicand with the partial product in AC.

When Q_n Q_{n+1} = 11 or 00 then the partial product does not change.

Next, we need to arithmetic shift right (asr) the partial product by one bit so that the sign bit in the AC is unchanged.

Upon each shift operation, the SC is decremented by one and the operation is repeated until SC=0. The flowchart for the same Is as shown below.



Example #1: using booths algorithm evaluate (**-9) x (+13).**
Multiplicand = - 9 is loaded into the BR register in 2's complement form and the Multiplier = + 13 is loaded into the QR register in 2's complement form.
- 9 = BR = 10111 and +13 = QR = 01101, Q_n is LSB of QR

Q_n Q_{n+1}	Comment	Operation	AC	QR	Q_{n+1}	SC
		Initial condition	0 0 0 0 0	0 1 1 0 1	0	5
10	First 1 from string of 1's	Sub BR from partial prod	0 1 0 0 1			
		after subtraction	0 1 0 0 1	0 1 1 0 1	0	
		arithmetic shift right ACQR and dec SC	0 0 1 0 0	1 0 1 1 0	1	4
01	First 0 from string of 0's	Add BR to partial product	1 0 1 1 1			
		After addition	1 1 0 1 1	1 0 1 1 0	1	
		arithmetic shift right ACQR and dec SC	1 1 1 0 1	1 1 0 1 1	0	3
10	First 1 from string of 1's	Sub BR from partial prod	0 1 0 0 1			
		after subtraction	0 0 1 1 0	1 1 0 1 1	0	
		arithmetic shift right ACQR and dec SC	0 0 0 1 1	0 1 1 0 1	1	2
11	Second 1 from string	arithmetic shift right ACQR and dec SC	0 0 0 0 1	1 0 1 1 0	1	1
01	First 0 from string of 0's	Add BR to partial product	1 0 1 1 1			
		After addition	1 1 0 0 0	1 0 1 1 0	1	
		arithmetic shift right ACQR and dec SC	1 1 1 0 0	0 1 0 1 1	0	0

When SC=0, the combination of AC and QR stores the product term.

Product term = 1 1 1 0 0 0 1 0 1 1

MSB = 1 specifies the product term is negative; the remaining 9-bits are in 2's complement.
To get the magnitude of the product term, perform the 2's complement of the remaining bits

Magnitude of Product term = 2's (110001011)
= 001110101
= -117

4c). Draw and explain the hardware for signed – magnitude addition and subtraction
Ans:
The steps involved in the hardware implementation of Add/subtract using sign magnitude form are

- ✓ Numbers A and B are stored in Register A and Register B. To save the space, let the result be stored in Register A. Thus, A_s and A forms an Accumulator
- ✓ Let A_s and B_s be the flip-flops that that stores the sign of A and B.
- ✓ To perform A + B, we need a parallel adder (constitutes full adders).

- ✓ To perform A – B, we need a parallel subtractor. However, hardware can be reduced if the subtraction operation is replaced by addition as follows:

$$A - B = A + 2's(B) = A + 1's(B) + 1$$

- ✓ The output carry is transferred to E flip-flop, which checks the relative magnitude of two numbers.
- ✓ AVF (Add-Overflow flip-flop) holds the overflow bit when A and B are added.
- ✓ The block diagram is as shown in fig.1 below.

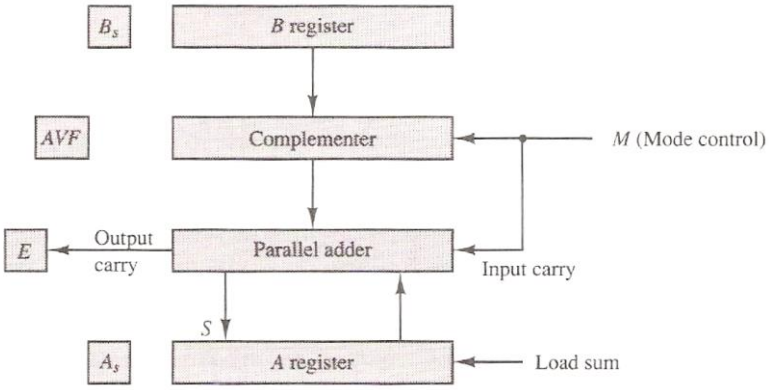


Fig 1: Hardware implementation

- ✓ One of the inputs to the parallel adder is A. The other input is wither B or B' depending on Mode Control M.
- ✓ Mode M to parallel adder to facilitate A +B or A – B
- ✓ The complementor circuit consists of XOR gate and the parallel adder consists of full adders.
- ✓ When M=0, the input is B. Addition is performed i.e., A + B + 0. The result is stored in A_s and A
- ✓ When M=1, the input is B'. Subtraction is performed i.e., A + B'+ 1. The result is stored in A_s and A

Hardware Algorithm

The hardware algorithm is as shown in fig.2. the algorithm is as follows.

- ➢ The two signs A_s and B_s are compared. i.e., A xor B.
- ➢ If output of xor is 0, then the sign bits are identical.
  - o For add operation, identical signs indicates that the magnitudes be added. Sign of result is same as sign of any input.
  - o For subtract operation, identical signs indicates that the magnitudes be subtracted. The sign of the result is
    - For A > B, sign of the result is the same as the sign of A ie., A_s
    - For A < B, sign of result is complement of sign A ie., A_s'
    - For A = B, sign is made +
- ➢ If the output of xor is 1, then the sign bits are different.
  - o For add operation, different signs indicates that the magnitudes be subtracted. The sign of the result is
    - For A > B, sign of result is same as sign of A ie., A_s
    - For A < B, sign of result is complement of Sign A ie., A_s'

        - For A = B, sign is made +
    - For subtraction, different signs indicates that the magnitudes be added. Sign of result is same as sign of A.
- Magnitudes are subtracted by adding A to 2's complement of B for A -B.
- For subtraction operation, there is no overflow. AVF = 0.
- If E=1, then A≥B and the final result is in A.
- If E=0, then A<B and the result is obtained by 2's complement of A. i.e., A ← A' + 1
- If result A=0, then the sign is made + to avoid negative zero.