Lec: 4,
After mids courses

COA Notes 8/12/23-Fr

chapter # 9:
Computer Arithmetic.

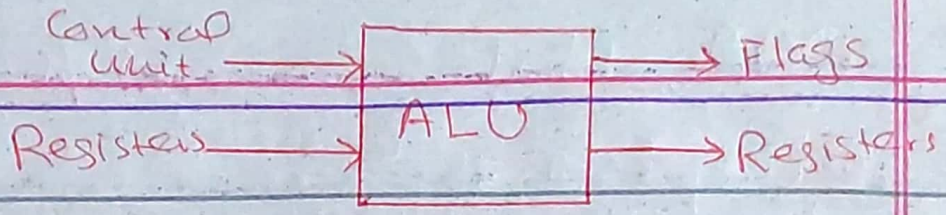
→ The two principal concerns for computer arithmetics are the way in which numbers (integers, floating-point) are represented (binary format) and algorithms used for basic arithmatic operations.

→ Floating-point numbers are:
a number (significand) multiplied by a constant (base) raise to some integer power (exponent).

9.1 The Arithmetic and Logic unit (ALU):

→ All other computer elements units are there mainly to bring data into ALU for processing and take the results back.

→ handle integer and may handle floating-point numbers.



9.2 Integer Representations

→ In binary system, arbitrary numbers can be represented with just the digit zero and one, the minus sign and the period or radix point

$-1101.0101_2 = -13.3125_{10}$

→ In general, n-bit sequences of binary digits $a_{n-1}a_{n-2}\cdots a_1 a_0$ is interpreted as an unsigned integer A, its value is:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

→ In compute we have no minus or period. Everything is represented with 0 and 1.

→ The most significant bit in the word is treated as sign bit. The n^{th} bit is sign while the n-1 bits is magnitude.

→ If bit is zero the number is positive, if bit is 1 number is negative.

+18 = 0001 0010 } sign Magnitude.
-18 = 1001 0010 }

→ for sign magnitude we have some drawbacks:
- +tive & -ive consider both the signs of numbers
- two representation of zero
  +0 = 0000 0000
  -0 = 1000 0000

→ Insted of sign-magnitude representation, most common scheme is twos complement representation.

→ 2's complement also uses the most significant bit as a sign bit.

→ Some Characteristics of 2's Complement
- range = -2^{n-1} through $2^{n-1}-1$
- 1 representation of zero
- Negating is easy: flip all bits and add one to it
  like 0010 → 1101 → 1110

- Expansion of bit length is done by adding the bits like sign bit to left.

→ It is sometime desireable to transfer n-bit number to m-bit number where m > n.

- for sign-magnitude notation, move the sign bit to new leftmost position and add zeros in b/w.
- for 2's-complement notation, move the sign bit to leftmost position and fill in with copies of sign bit.

+18 = 0001001b

+18 = 0001001b

-18 = 1110 1110

-18 = 1111 1111 1110 1110

→ These representations are sometimes refer to as fixed point representation because radix point is fixed and assume to be in right of rightmost digit.

9.3 Integer Arithmetics:

Negation:

→ For sign-magnitude negation is done by inverting the sign bit.

0+0=0	0-0=0
0+1=1	0-1=1 — (1)
1+0=1	1-0=1
1+1=0 — (1)	1-1=0

→ In 2's complement flip all bits and add 1 to it.

+18 = 0001 0010

Complement/Not = 1110 1101

+1 + 1

1110 1110 = -18

→ for shortcut of 2's complement, from right side write the bits exactly the same to first occurring of 1 and then flip all bits from that bit to left most bit.

+18 = 0001 0010

-18 = 1110 1110 → Flip from here

Addition and Subtraction:

→ On any addition, the result may be larger than can be held in the word size being used. This condition is called overflow.

→ If the result is positive of two we get a positive number in 2's complement form, while if negative we get negative number in 2's complement

→ Carry bit beyond the end of

⑥ 18 → Minuend
- 6 → Subtrahend

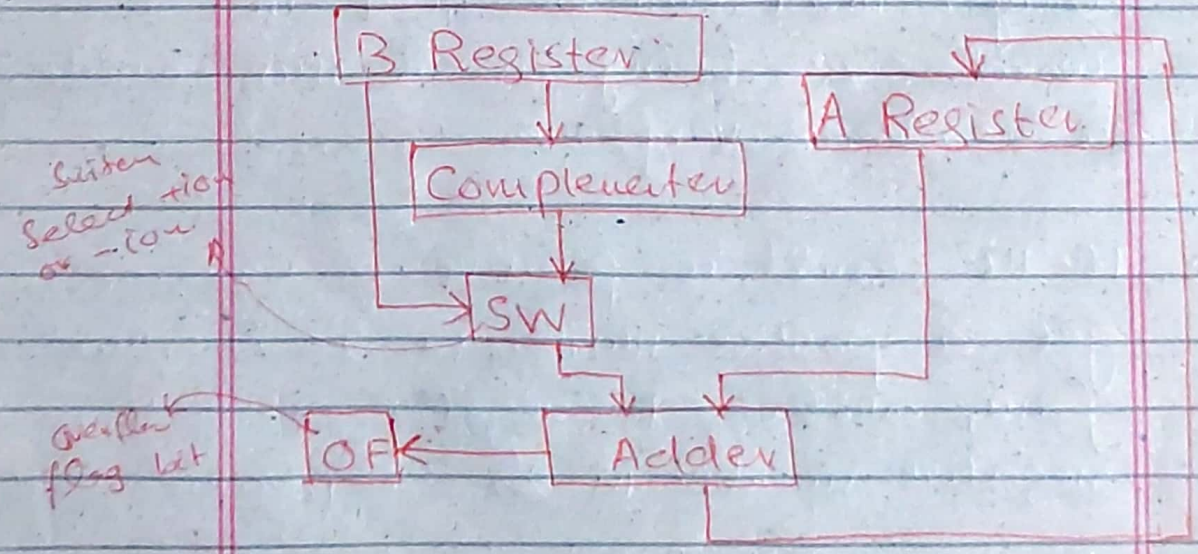
is ignored.

→ To subtract one number (subtrahend) from another (minuend), take twos complement (negation) of the subtrahend and add it to minuend:

* $m - s = m + (-s)$

→ If two numbers are added, and they are both +ve or -ve, then overflow occur if and only if the result has opposite sign.

5 = 0101	-6 = 1010
+ 4 = 0100	-4 = 1100
= 1001 overflow	= 1 0110 overflow



Hardware for +ion and -ion block diagram

0 × 0 = 0
0 × 1 = 0
1 × 0 = 0
1 × 1 = 1

* Multiplication:

→ xion is more complex than tion or -ion. wether performed in software or hardware.

→ Multiplication of unsigned binary integers.

```
      1011    Multiplicand (11)
    x 1101    Multiplier (13
    ------
      1011
     0000     } Partial Products (P.P)
    1011
+   1011
    ------
   10001111   Product (143)
```

- involve generation of partial products one for each bit in multiplier. These are summed to produce final product
- When multiplier bit is 0, the partial product is zero, when it is 1, the p.p is multiplicand.
- Each successive p.p is shifted one position to left (<<) relative to preceding p.p.

(8)

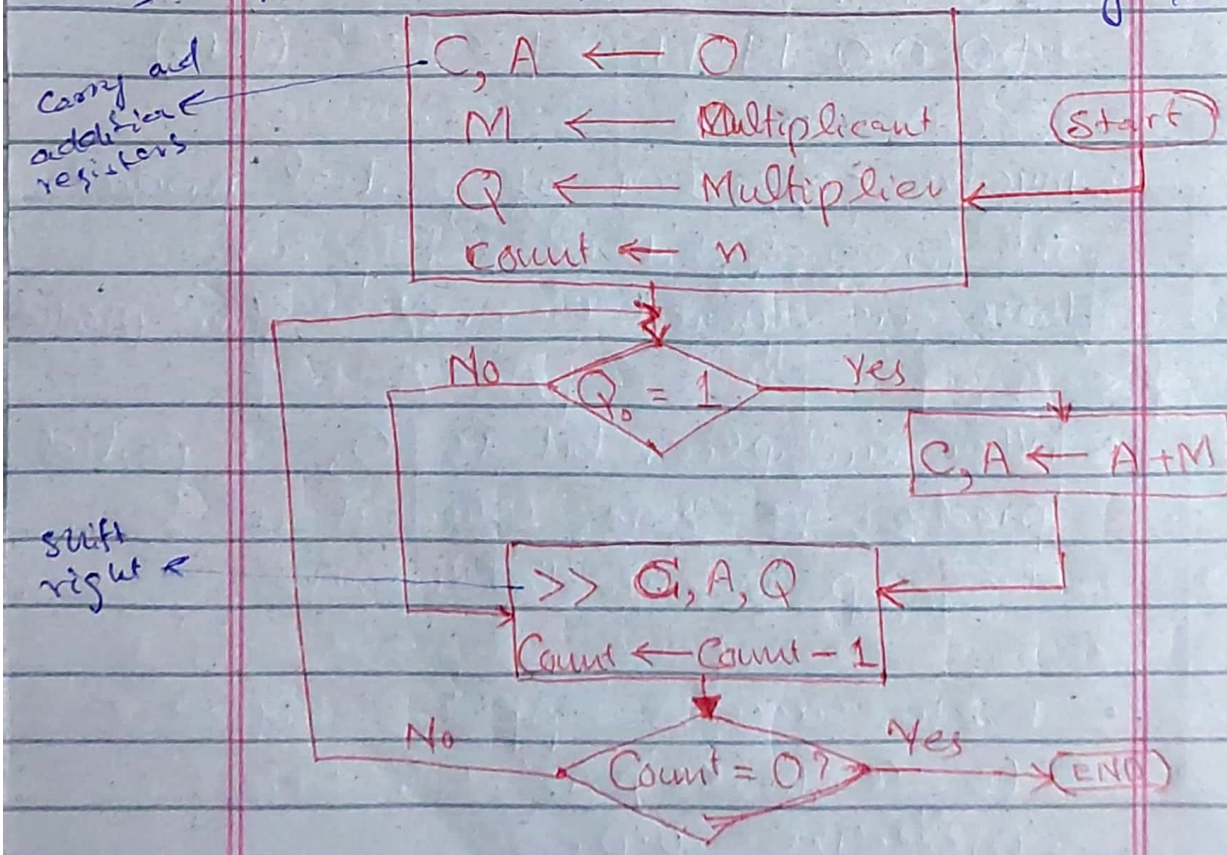
- The xion of two n-bits binary integers results in a product of up to 2n bits in length.

→ For efficient xion;

- we can perform a running addition on p.p instead for waiting till end.
- For each one on multiplier an add and a shift operation is required. but for each zero only a shift is required.

Gives time or generation of pp-

→ Flow chart for unsigned Binary xion.

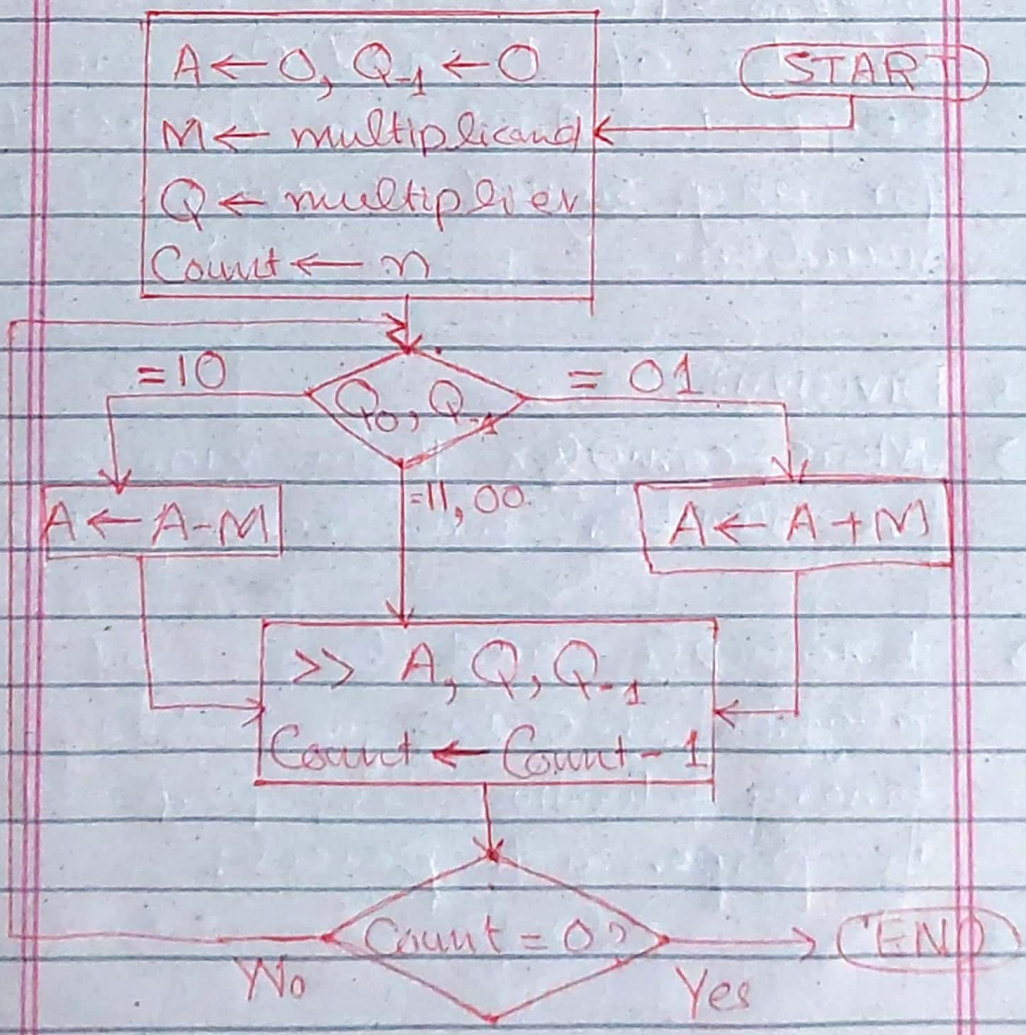


→ Some problem arise with multiplication of negative numbers if either one or two are -ve. (2's C)

→ Solution for xion of -ve numbers are

- Convert the numbers to +ve and xiply as above. If sign were different negate the answer.
- Use Booth's algorithm.

→ Both's algo for 2's complement xion:



(10)

$1 \div 1 = 1$

$1 \div 0 =$ Meaningless / undefine

$0 \div 1 = 0$

$0 \div 0 =$ Meaningless / undefine.

→ Q_0 is least significant bit of register Q.

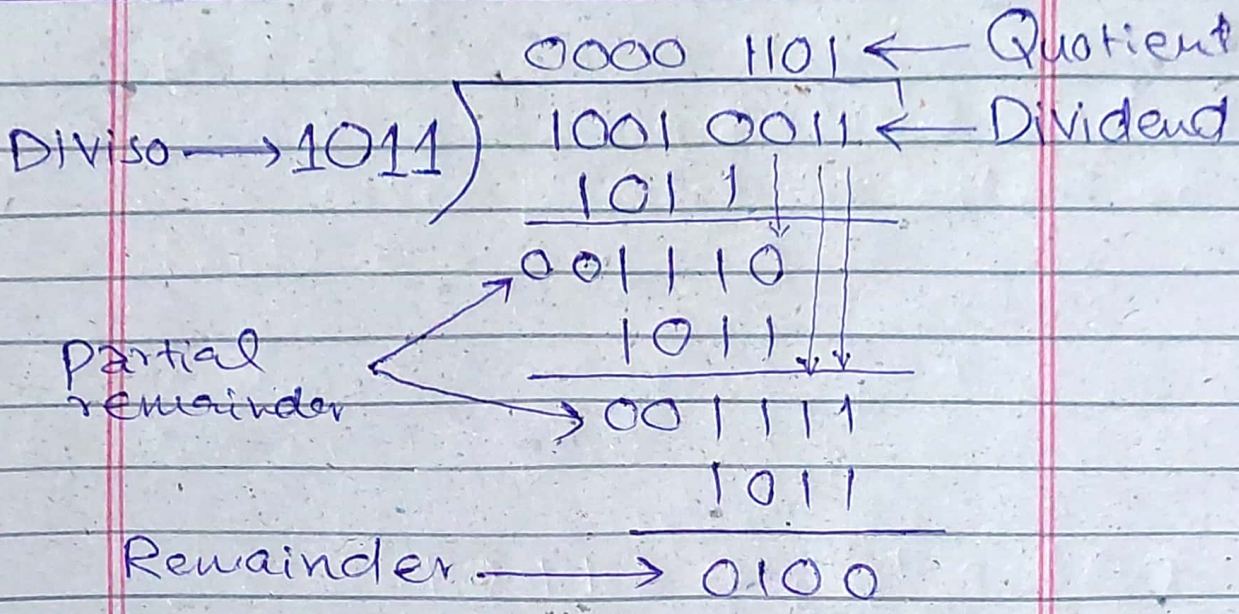
→ The right shift is such that leftmost bit of A, namely A_{n-1}, not only shifted to A_{n-2}, but also remains in A_{n-1}. This preserves sign of A and Q. It is known as arithmetic shift.

→ Booth's algo is also the most efficien algo becaus blocks of 1's and 0's are skipped. And can be used for any signed or unsigned binary numbers.

* Division:

→ More complex than xion and -ve numbers are really bad.

→ Example and basis of algorith is unsigned long binary division as performed in paper and pencil:



→ Flowchart for unsigned binary Division