Number Systems:

Decimal number system - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary number system - 0, 1 (Base (or) radix = 2)

octal number system - 0, 1, 2, 3, 4, 5, 6, 7

Hexa decimal number system - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

2	25	
2	12	1
2	6	0
2	3	0
	1	1

$= 11001$

Complements.

- There are two types of compliments for each base r system: the r's complemend & (r-1)'s Complement

Ex:- for base = 2 (1's comp & 2's comp) — Binary

for base = 10 (10's comp & 9's comp) — Decimal

Subtraction of Unsigned Numbers:-

Unsigned 2's complement Subtraction

Example:-

Find $01010100_2 - 01000011_2$

$$\begin{array}{r} 01010100 \\ -01000011 \end{array} \xrightarrow{2\text{'s comp}} \begin{array}{r} 01010100 \\ +10111101 \\ \hline 111 \end{array} \qquad \begin{array}{r} 10111101 \\ \hline 000100 \end{array}$$

Fixed-point Representation:

- all the +ve integers, and zero can be represented as Unsigned numbers
- To represent negative integers, we need a notation for -ve values
- In ordinary arithmetic, the negative no is indicated by minus sign & positive number by plus sign.
- Because of hardware limitations must represent everything with 1's & 0's, including sign of the no
- Sign bit = 0 for +ve
  = 1 for -ve
- In addition to the sign, a number may have a binary (or decimal) point
- Based on position of Binary point we represent fractions, integers, or integer-fraction nos
- two ways of specifying the position of the Binary point in a register

- The two positions most widely used are
  1) binary point → extreme left of register to make the Stored no a fraction
  2) Binary point → extreme Right → number an integer
  3) Middle → Mixed

Integer Representation:

Sign bit = MSB = 0 ⇒ +ve magnitude & Vise Versa

if MSB = 1 the magnitude may be represented in one of the 3 possible ways.

1) Signed magnitude representation
2) Signed 1's complement representation
3) Signed 2's complement representation

for +ve no there is only 1 way ☺

Signed numbers

Decimal	(-8 to +7) Signed 2's comp	(-7 to +7) Signed 1's Comp	(-7 to +7) Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	— (no seprate 1111 representation	1111	1000
-1	1110 for +0 & -0)	1110	1001
-2	1101	1101	1010
-3	1100	1100	1011
-4	1011	1011	1100
-5	1010	1010	1101
-6	1001	1001	1110
-7	1000	1000	1111
-8		—	—

+ve

upto here all are same

-ve

- The signed-magnitude representation of a -ve no consists of the magnitute & -ve sign
- The -ve no is represented in either 1's or 2's comp of its +ve value
- for ex) consider signed no is stored in an 8-bit registers

↳ 0000 1110 → Binary equivalent
↳ MSB = 0 (+ve 14)

(-14) 1000 1110 (signed magnitude representation)

unsigned - 0 to 2^{n-1}

$-2^{n-1}-1$ to $+2^{n-1}-1$

n=4

-2^3-1 to 7

= -7 to 7

↳ signed (range)

Signed 2's compl

-2^{n-1} to $+2^{n-1}-1$

1's complementing (+14) → 0000 1110

(-14) 11110001 → signed 1's complement representation

(-14) 11110010 → signed 2's complement representation

2's complementing (+14)

Arithmetic addition:

- if Signs are same, we add two magnitudes & give sum to common sign

  −24−36 = −60

  +24+36 = +60

- if Signs are different, we subtract the smaller magnitude from larger. and give larger magnitude no sign to result

  +24−36 = −12

2's complement addition:-

- Signed 2's complement doesn't require comparsion or Subtraction only requires addition and complementation

- procedure
  → Add the two no's including their sign bit
  → discard carry
  → −ve no's must be 2's complimented & carry is ignored

1111 1001
+1
1111 1010

Ex:- 8421

+6 0000 0110
+13 0000 1101
+19 0 0001 0011

MSB=0
positive result

−6 1111 1010 → 2's comp of +6
+13 0000 1101
+7 0 0000 0111

MSB=0
+ve result

11110010

+6 00000110

-13 11110011 → 2's comp of (+13)

-7 (1)11111001

↳ MSB=1 (sign bit)
-ve result
So do 2's compliment

0000 0110
1

0000 0111

So Ans is -0000 0111

-6 1111 1010

-13 1111 0011

-19 (1)1101101

↳ MSB=1 (sign bit)
-ve result
So do 2's Compliment

0001 0010
1

0001 0011
16 8 4 2 1

So Ans is -0001 0011

Arthemetic subtraction:-

- Take 2's compliment of subtrahed & add it to minuend (including sign bit) and discard the carry.

Overflow:-

- when 2 n.o's of (n) digits are added & sum occupies n+1 digits, we say that an overflow occured.
- over flow is a problem in digital computers since the width of registers is finite.
- Hence n+1th bit can't be stored in a register with n bits.
- So may computer detect the occurance of over flow, & corresponding flip-flop is set which can then be checked by user

4+5 = 9

n=4 range -7 to +7

↳ in this range 4,5 will be there but 9 doesn't exits

$-2^{4-1}-1$ to $+2^{4-1}-1$

-7 to 7

-2^7-1

Example:-

n=8 (8bit register)

$-2^{8-1}-1$ to $+2^{8-1}-1$ ⇒ -127 to 127

carries: 0 1

+70 01000110

+80 01010000

+150 1 0010110

MSB=1 (-ve) no but not (carry occupies sign bit)

so

+150 = 0 1001 0110

add extra bit

carries: 1 0

-70 1011 1010

-80 1011 0000

-150 0 1101010

MSB=0 (+ve no) but not

so

-150 = 1 0 1101010

add extra bit

Note:-

the carry out of the sign bit position is taken by the sign bit of the result, the 9-bit answer ~~so~~ be accom-odated within 8 bits. We say that an overflow occured

Decimal fixed-point Representation.

The representation of decimal numbers in registers is a fxn of the binary code used to represent a decimal digit.

in BCD representation (needs 16 flip flops)

each n.o - 4 flip flops

4 digits → 16 ff's

4385 - 0100 0011 1000 0101

→ A 4-bit decimal code requires four flip-flops for each decimal digit

Same 4385 → in Binary needs 13 bits

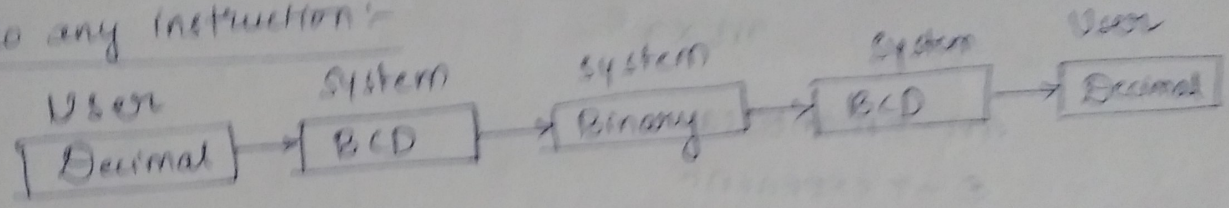
→ Storage will be saved if we used BCD representation compared to decimal

1000100100001 → Binary (4385)

0100 0011 1000 0101 → BCD (4385)

↓ By seeing this we can identify easily compared to above code

To do any instruction:-



→ In BCD

Consider the addition (+375) + (−240) = +135 done in signed-10's complement system

```
  0 3 7 5   (0000 0011 0111 0101) BCD
+ 9 7 6 0   (1001 0111 0110 0000) BCD
-----------------------------------
            (0000 0001 0011 0101) BCD
```

```
  999
- 240
-----
  759
    1
-----
  760 ....
```

```
    1101
    0110
   -----
(1) 0011
```

→ The 9 in the left most digit ((9)760) indicates the number is negative.

Floating-point representation:-

+6132.789

Fraction (or) Mantissa	exponent
+0.6132789	+04

→ Scientific notation $-+0.6132789 \times 10^{+4}$

floating-point → is expressed of the form

$$m \times r^e$$

m → mantissa
e → exponent

- In floating point binary number is represented in a similar manner except that it uses base2 for exponent.

ex:- Binary number +1001.11 is represented with an 8-bit fraction & 6-bit exponent as follows:

Fraction:-

01001110
↳ indicates (+)

Exponent:-

000100

Normalisation:-

Normalisation is the process of moving the binary point so that the first digit after the point is a significant digit

Alphanumeric representation:

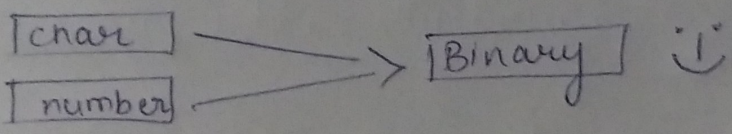
→ Many applications require handling of data, not only numbers, but also letters of the alphabet and certain special characters

→ Alphanumeric set includes 10 decimal digits, 26 alphabets and some special characters like $, +, -, ... etc

- The standard Alphanumeric binary code is the ASCII (American standard code for info interchange) which uses seven bits to code 128 characters

  27

- The codes must be in binary because registers can hold only binary information



Upper case – 65 to 90 → 65+25

lower case – 97 to 122 → 97+25

97
-65
32

ASCII code is used for transmission of binary information

→ Usually each character is represented by 7-bit code and usually 8th bit is inserted for parity

↳ 128 characters

↓

error detecting bit

EBCDIC

- Another alphanumeric (sometimes called alphameric) code used in IBM equipment is the EBCDIC (Extended BCD interchange code)
- It uses 8 bit for each character

Adders :-

- Binary addition is a fundamental operation
- There are a variety of adders, each has certain performance

full adder :-

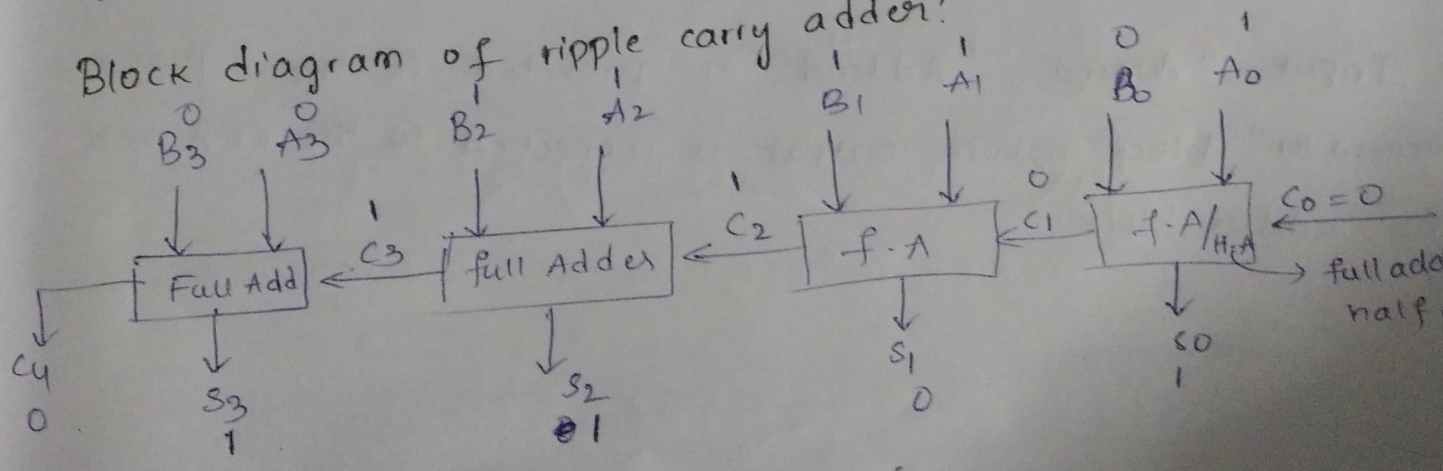
add two bits → half adder
add three bits → full adder
↳ 1 is co and produces sum, carry.

N-bit binary adder / ripple carry adder.

- 4+5 ⇒ not possible using single full (or) half adder $2^{4}=1$
  so ripple carry adder is constructed by cascading full adder block in series.
- The carryout of one stage is fed directly to the carry-i of the next stage
- for an n-bit ripple adder, it requires n full adders / n-1 full adders and 1 half adder

$A_3 A_2 A_1 A_0$
A 0 1 1 1
B 0 1 1 0
$B_3 B_2 B_1 B_0$
1 1 0 1

Block diagram of ripple carry adder:



⇒ 1 1 0 1 ✓

Inputs are ready but fulladder-4 has to wait until
full adder-3 has A, B so lag is occuring.

↓ Disadvantage ①

as no of bits ↑ no of full adders ↑ ⁻I⁻ disad ②

Advantages:-

→ We can add two n-bit numbers easily
(easy to construct)

→ it is Advantageous for less number of bit operations

Distadvantages:-

→ Ripple-carry adder, illustrating the delay of the carry bit.

→ Very slow when one needs to add many bits

- to overcome these disadvantages we use carry look ahead adder.

Carry-look ahead adder:- (CLA)

Contains 3 Blocks
1) P and G generator
2) CarryLook ahead block
3) adder block

- Input "Augend", "Addend" is provided to the "p and G generator" block whose output is connected to the CLA and the adder block.