13/09/2019

Binary addition & subtraction

* One advantage of 1's complement & 2's complement no. system is that subtraction is just an addition. No extra subtractor ckt is required. If you have designed an adder, that is sufficient for subtraction as well.

* Addition we have talked about.

```
B : 0 1 0 0          B : 1 1 0 0
    1 0 0 1 0            0 0 0 1 1
  - 0 1 1 0 0          - 0 0 1 0 0
  ---------            ---------
    0 0 1 1 0          (1) 1 1 1 1 1
```

Subtraction using 1's complement

* Say A-B

(-B) represent by 1's complement.

A + (-B)

= A + B's 1's complement

⇒ If carry is there, add it to result

⇒ The result is +ve

→ If carry is not there, the result is -ve & is in 1's complement form.

eg. 6-2, n=4 (4 bit representation)

6 : 0110

-2 : 1101

10011

→ 1

0100

= +4 (the result is +ve)

2 : 0010

-2 : 1101

eg. 3-5

3 : 0011

-5 : 1010

1101

= -2 in 1's complement form

5 : 0101

(the result is -ve)

try: 5-21 , 3-1 , 13-15

Subtraction using 2's complement

* say, A-B

= A+(-B)

= A+ 2's comp. of B

* ignore carry if present, it's no
* carry not there, result is -ve, and already in 2's complement form

eg.

$$\begin{array}{rl} 6: & 0110 \\ -2: & 1110 \\ \hline & 10100 \end{array}$$

discard ← $= +4$

2 : 0010
2's com of 2 : 1101+1
$= 1110$

+ve no. since carry is coming out.

eg. 3-5

$$\begin{array}{rl} 3: & 0011 \\ -5: & 1011 \\ \hline & 1110 \end{array}$$

5 : 0101
2's com. of -5 : 1011

$= 2$'s comp. of -2

-ve no. since carry is not there

try ! 3-13 , 4-12, 16-14

* Overflow happens when both the no.s are having same sign.

2's complement

$$\begin{array}{rl} +6: & 0110 \\ +5: & 0101 \\ \hline & 1011 \end{array}$$

MSB is 1 : -ve

4 bit 2's complement 11 can not be represented

This is overflow.
out of Range (-7 to +8) is range.

BCD and gray code

* For less complex mathematical operation, we donot need ~~decimal~~ binary decimal-binary conversion. Binary operations are faster.
* decimal to binary & vice versa are complex.

Binary Coded decimal (BCD)

→ In BCD, every decimal digit is represented by 4 bit binary equivalent.

eg. 238_{10} = 0010 0011 1000 in BCD

13_{10} = 0001 0011 in BCD

12:10 = 0001 0010 : 0001 0000

→ Conversion is simple.

→ Six 4-bit combinations are not valid in BCD. those are 1010, 1011, 1100, 1101, 1110, 1111.

BCD	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Addition of BCD no.:

(14/8/2019)

* Add 0110 to the
  → if one of the nibbles have invalid combination
  → if there is a carry from previous nibble

eg. 23 + 46

```
  0010  0011
+ 0100  0110
-----------
  0110  1001     = 69
```

→ valid BCD.
→ no carry.
} → no need to add 0110

⇒

eg. 23+48

```
  0010  0011
  0100  1000
-----------
  0110  1011
      + 0110
-----------
  0111  0001     = 71
```

→ correction step is done at this nibble because the nibble is invalid

→ both nibble invalid means add 0110 to each nibble

eg. 28+39

```
  0010    1000
  0011    1001
-----------------
  0110    0001
        + 0110
-----------------
  0110    0111     = 67
```

try: 23 + 49, 30+49, 59+99

Gray Codes

→ Successive ~~bits~~ nos differ by single bit change.
→ Non weighted binary code.
→ Also called cyclic code.
→ Gray codes are used to reduce dynamic power consumption as it minimizes switching. Also reduce error in conversion

Binary	Gray
000	000
001	001
010	011
011	010
100	110
⋮	⋮

Binary to Gray:

B : 0 1 1

G : 0 1 0

B : $B_2\ B_1\ B_0$

G : $G_2\ G_1\ G_0$

$G_2 = B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

Gray to Binary:

G : $G_2\ G_1\ G_0$

B : $B_2\ B_1\ B_0$

$B_2 = G_2$, $B_1 = G_1 \oplus B_2$, $B_0 = G_0 \oplus B_1$

G: 0 1 0

B: 0 1 1

try: $(10110)_2 = (\ ?\ )_G$

$(0110)_2 = (?)_G$

$(11001)_G = (?)_B$

$(011010)_G = (?)_B$

→ Gray code is called self-reflecting code.

eg. 00 → same thing if you want to represent
01 using 3 bits ↓
10
11

0	00
0	01
0	10
0	11
1	11
1	10
1	01
1	00

mirror image
self reflecting?

similarly try 3 bit representation to 4 bit representation.

ASCII code

→ American standard code for Information exchange.

→ Its a code for representing 128 english character as numbers, with each letter assigned a no. from 0 to 127.

→ Most computers use ASCII code to represent text, which makes it possible to transfer data from one computer to another.

→ Standared ASCII character set uses just 7 bits for each character, while some ch. set uses 8 bits.

→ Also used in telecommunication.

→

A → 065	a → 097
B → 066	b → 098
⋮	⋮
Z → 090	z → 122

0 → 048

1 → 049 & so on.

9 → 057

EBCDIC

(extended binary coded decimal interchange code)

8-bit character encoding used mainly on IBM computers.

8 bit means $(0-(2^8-1))$ possible combinations.

A → 193	a → 129	0 → 240
B → 194	b → 130	1 → 241
⋮	⋮	⋮