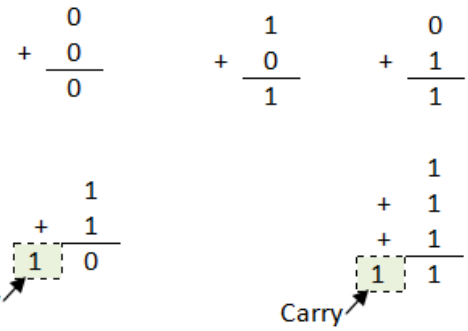
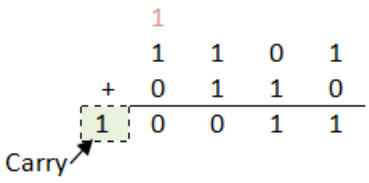
Binary arithmetic:

- **Binary addition**

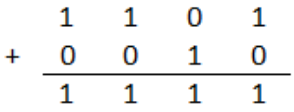


ex:

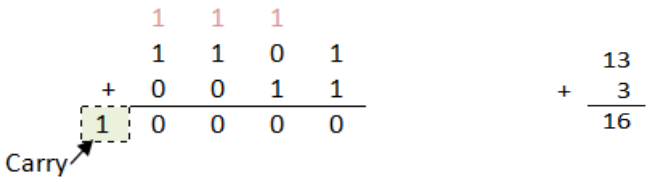
1-



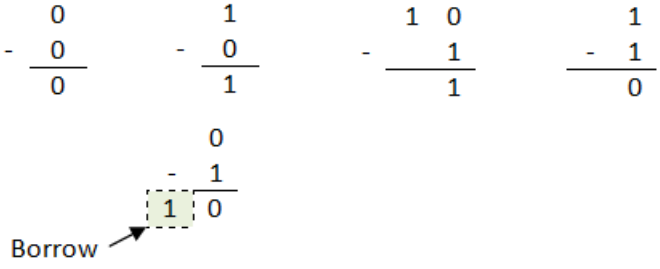
2-



3-

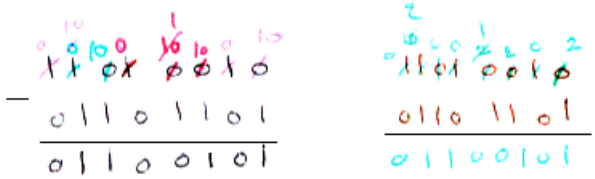


- **Binary subtraction:**



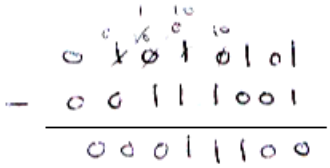
ex: subtract

1-



Note: in binary 10-1=1 not 9

2-



H.W

1- Find a- 1111 +1100

b- 10111 + 01101

2- Find a- 111 - 100

b- 110 – 101

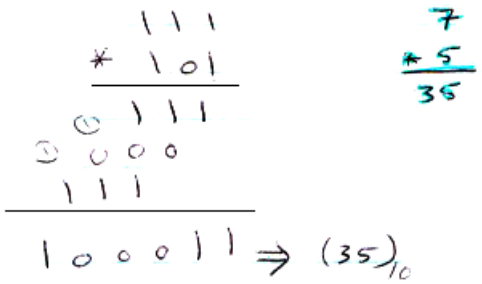
c- 1101 – 0100

d- 1001 – 0111

- **Binary multiplication:**

$$\begin{array}{r} 0 \\ *\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ *\ 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ *\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ *\ 1 \\ \hline 1 \end{array}$$

ex:



H.W:

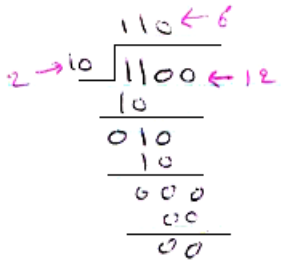
Find a- 1101 * 1010

b- 110 * 111

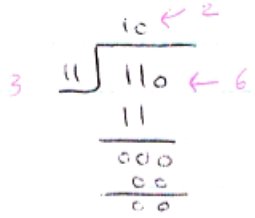
- **Binary** division:

ex:

1-



2-



H.W

Find a- 1100 ÷ 100

b- 1100 ÷ 011

Signed binary numbers:

Digital system , such as the computer , must able to handle both positive and negative numbers.

To representing a binary sign is by adding another bit to the number called sign bit.

The left – most bit in a signed binary number is the sign bit

Sign bit = 0 the number is positive

Sign bit = 1 the number is negative

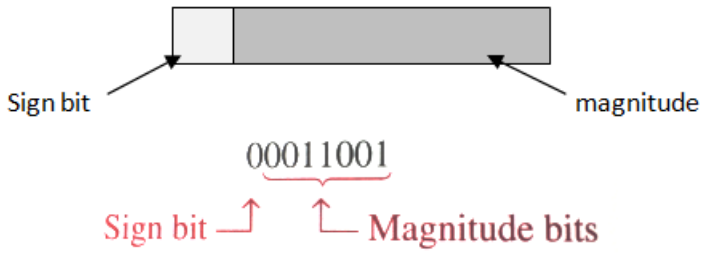
There are three forms to represent the signed binary:

1- Sign – magnitude

2- 1's complement

3- 2's complement

- **Sign – magnitude**



ex: expressed the decimal number + 25 and -25 as an 8 bit signed numbers

+ 25 → 00011001
- 25 → 10011001

This method is simple to implement but it has a disadvantage the sign bit independent of magnitude, the zero number have a sign bit (+0 → 00000000) and (-0 → 10000000) and this makes math hard to do.

- **One's complement:**

Invert all bits ,each 1 become a 0 and each 0 become a 1

ex:

10011001 → 01100110
1's complement

ex:

01001100 → 10110011
1's complement

The MSB is the sign bit, when it is a 0 the number is positive and when it is a 1 the number is negative

Also this method have +0 and -0

+0 → 0000 and -0 → 1111

Make math hard to do

- **Two's complement:**

Is the method to representing positive and negative integer values in binary.

To form the 2's complement is by add 1 to the 1's complement number

ex: 10011001

The 1's complement = 01100110

The 2's complement = 01100110 +1= 01100111

Unsigned number		Signed number	
value	binary	Sign value	binary
0	0000	0	0000
1	0001	+1	0001
2	0010	+2	0010
3	0011	+3	0011
4	0100	+4	0100
5	0101	+5	0101
6	0110	+6	0110
7	0111	+7	0111
8	1000	+8 , -8	1000
9	1001	-7	1001
10	1010	-6	1010
11	1011	-5	1011
12	1100	-4	1100
13	1101	-3	1101
14	1110	-2	1110
15	1111	-1	1111

ex:

4 + (-4) =0

When we add positive number with the same ngative number the result must be zero

So in the signed binary number when we add positive binary number with it's 2's complement and ignored the carry the result it must be zero

ex:

4 → 0100

-4 → 2's (0100) = 1011+1 = 1100

4 + (-4) → 0100 + 1100 = 1 0000

Ignore

ex:

7 → 0111

-7 → 2's (0111) = 1000+1 = 1001

7 + (-7) → 0111+ 1001 = 1 0000

Ignore

ex:

8 → 1000

-8 → 2's (1000) = 0111+1 = 1000

8 + (-8) → 1000+ 1000 = 1 0000

Ignore

H.W:

What is $(-65)_{10}$ in binary?

Note: the two's complement provides an easier way to subtract number using addition

ex:

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ -\ 1\ 1\ 1\ 0 \\ \hline \end{array}$$

$$\begin{array}{r} 10 \\ -\ 14 \\ \hline -4 \end{array}$$

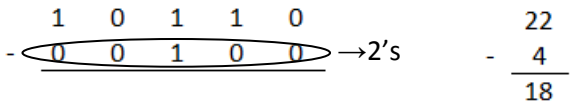
1110 → 1's → 0001 → 2's →0001+1= 0010

$$\begin{array}{r} \ 1 \\ 1\ 0\ 1\ 0 \\ +\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 1\ 0\ 0 \end{array}$$

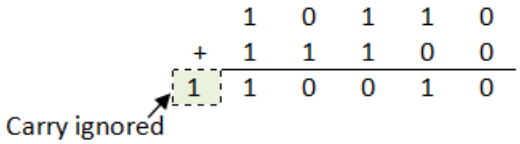
, the 1100 = - (0100) = -4

When there is no carry the answer is in 2's complement

ex:



00100 → 1's → 11011 → 2's →11011+1= 11100



When there is a carry the answer is not in 2's complement

H.W:

Solve the following subtraction using 2's complement

a) 8 -5 b) 89-46 c) 20-60 d) 133-50

binary codes:

1- Weight codes
2- Non-weighted codes
3- Alphanumeric codes
4- Error detection codes

- **Weighted codes:**

BCD 8421 , 6311 , 2421 , 642-3 , 84-2-1

- **Binary – Coded – Decimal (BCD)**

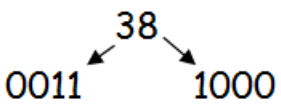
  The 8421 code:

Each decimal digit (0 – 9) is represented by a group of four bit

decimal	8421 BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

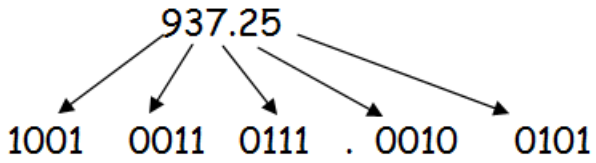
Encode the decimal number in BCD (8421) code

ex: encode $(38)_{10} \rightarrow (\quad)_{BCD}$



= $(00111000)_{BCD}$

ex: encode $(937.25)_{10} \rightarrow (\quad)_{BCD}$

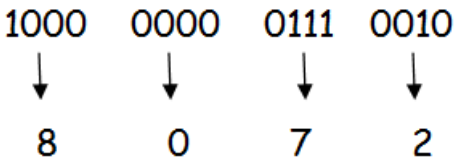


= (100100110111.00100101) $_{BCD}$

Decode BCD (8421) number in decimal form

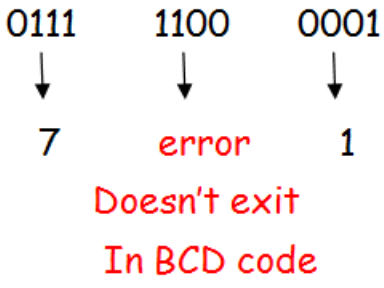
Divide the BCD number into 4-bit and convert each to decimal.

ex: decode $(1000000001110010)_{BCD} \rightarrow (\quad)_{10}$



= $(8072)_{10}$

ex: decode $(011111000001)_{BCD} \rightarrow (\quad)_{10}$



Note: with 4-bits, a sixteen numbers (0000 → 1111) can be represented. but with 8421 BCD code only ten of these are used (0000 → 1001).

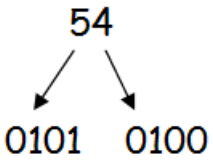
The (1010, 1011 , 1100 , 1101 , 1110 , 1111) are invalid in 8421 BCD code.

ex: convert $(54)_{10}$ to ($)_2$

	54	
2	27	0
2	13	1
2	6	1
2	3	0
2	1	1
2	0	1

= $(54)_{10} \rightarrow (\ 110110\ )_2$

ex: encode $(54)_{10}$ to ($)_{BCD}$



= $(54)_{10} \rightarrow (\ 01010100\ )_{BCD}$

Note: encode the decimal number to BCD code is different than that converting it's to binary.

Other 4-bit BCD codes:

decimal	8421	6311	2421	642-3	84-2-1
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0101	0111
2	0010	0011	0010	0010	0110
3	0011	0100	0011	1001	0101
4	0100	0101	0100	0100	0100
5	0101	0111	1011	1011	1011
6	0110	1000	1100	0110	1010
7	0111	1001	1101	1101	1001
8	1000	1011	1110	1010	1000
9	1001	1100	1111	1111	1111

H.W:

1- encode the decimal number to 8421 BCD code

a- 39 b- 65 c- 40 d- 17 e- 82 f- 99

2-encode the 65137 decimal number to:

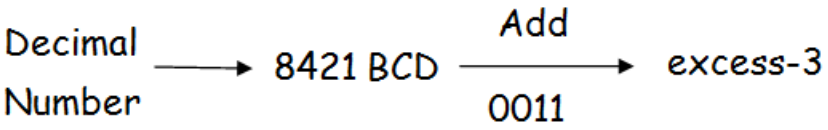
a- 8421 BCD b- 2421 BCD c- 84-2-1 BCD d- 6311 BCD e- 642-3 BCD

3-decode the 8421 BCD number to decimal

a- 10000000 b- 00110110 c- 10010010 d- 01110110 e- 01010101

- **Un-weighted codes**
  - **Excess-3 code**

Also is called Xs-3 code, it is derived from 8421 BCD code adding 0011 to each code.

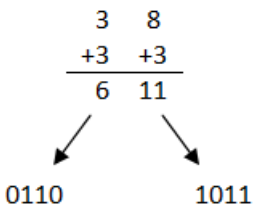


decimal	8421BCD	EXCESS-3 (BCD+0011)
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Encode decimal number to Xs-3

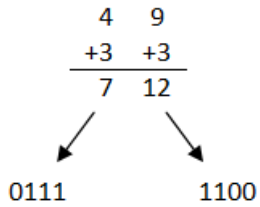
To encode the decimal number in to excess-3 form, add 3 to each decimal digit before converting to binary

ex: encode $(38)_{10} \rightarrow (\quad)_{Xs-3}$



$(38)_{10} = (0110\ 1011)_{Xs-3}$

ex: encode $(49)_{10} \rightarrow (\quad)_{Xs-3}$

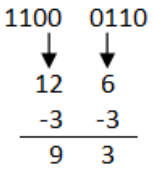


$(49)_{10} = (0111\ 1100)_{Xs-3}$

Note: in the Xs-3 the invalid binary combination are (0000 , 0001 , 0010 , 1101 , 1110 , 1111)

Decode the excess-3 to decimal

ex: decode the $(1100\ 0110)_{Xs-3} \rightarrow (\ )_{10}$



$(1100\ 0110)_{Xs-3} = (93)_{10}$

Note: in excess-6 code each decimal digits N is represented by binary equivalent of N+6

- **Gray code:**

It's an un-weighted code not suited to arithmetic operation but is useful for many applications such as shaft position encoder

decimal	binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Note: only one bit is change in gray code when it's going from any decimal number to the next

ex:

3 → in binary 0011 → in gray 0010

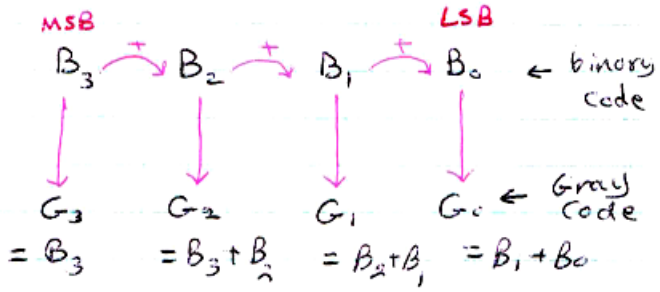
4 → in binary 0100 → in gray 0110

3 bit change one bit change

Encode from binary to gray code:

Rules of convert from binary to gray

1- The first MSB gray digit is the same as the first MSB binary digit
2- Going from left to right, add each adjacent pair of binary bits to get the next gray bit, disregard carries



ex: encode $(1101)_2 \rightarrow (\quad)_{gray}$

$= (1011)_{gray}$

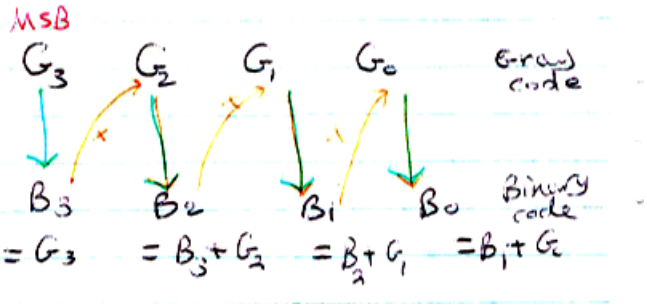
ex: encode $(0010)_2 \rightarrow (\quad)_{gray}$

$= (0011)_{gray}$

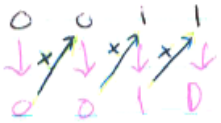
Decode from gray code to binary:

Rules of convert gray to binary

1- Repeat the first digit (MSB)
2- Add diagonally to get the next binary digit, disregard carries



ex: decode $(0011)_{gray} \rightarrow (\quad)_2$



$= (0010)_2$

H.W

1. Convert binary 101101 to gray code

2. Convert gray code 100111 to binary

- Alphanumeric codes:

A computer must be able to recognize codes that represent numbers, letters and special characters, these codes are classified as Alphanumeric codes.

The most common Alphanumeric code known as American Standard Code for Information Interchange ASCII.

ASCII has 128 characters and symbols it is represent by 7-bit binary code.

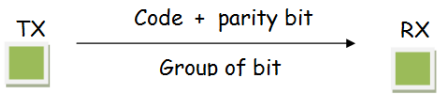
The decimal digit are represent by 8421 code preceded by 011

decimal	ASCII
0	0110000
1	0110001
2	0110010
3	0110011
4	0110100
5	0110101
6	0110110
7	0110111
8	0111000
9	0111001

- **Error detection codes (party)**

Used to detect the error during the data transmission

The parity bit is an extra bit that is attached to a code group that being transferred from one location to another



The parity bit make the make the total number of 1's in a group always even or odd, for example if system used BCD code the group of bit is:

Even parity (all 1's is even)		Odd parity (all 1's is odd)	
BCD	P	BCD	P
0000	0	0000	1
0001	1	0001	0
0010	1	0010	0
0011	0	0011	1
0100	1	0100	0
0101	0	0101	1
0110	0	0110	1
0111	1	0111	0
1000	1	1000	0
1001	0	1001	1

ex: if the Tx and Rx used odd parity checker and the Rx received the code:

10101 → correct

If code is 10001 → error

If code is 10100 → error