FIT1047 PASS WEEK 3

Base conversion
Sign and magnitude, 1's complement, 2's complement
Floating Point Numbers
Error Detection

Base Conversion

Binary (base 2)	Decimal (base 10)	Hexadecimal (base 16)
1 1101	**29**	**1D**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Converting from binary (base 2) to decimal (base 10)

$1100\ 1101_2$	Compute base 10 value for each place of the base 2 number: $\mathbf{(1 \times 2^7)} + \mathbf{(1 \times 2^6)} + (0 \times 2^5) + (0 \times 2^4) + \mathbf{(1 \times 2^3)} + \mathbf{(1 \times 2^2)} + (0 \times 2^1) + \mathbf{(1 \times 2^0)} =$ $(1 \times 2^7) + (1 \times 2^6) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^0) =$ $128 + 64 + 8 + 4 + 1 = \mathbf{205_{10}}$
110011_2	$(1 \times 2^5) + (1 \times 2^4) + (1 \times 2^1) + (1 \times 2^0) =$ $32 + 16 + 2 + 1 = \mathbf{51_{10}}$

Converting from decimal (base 10) to binary (base 2)

169_{10}

2 | 169
2 | 84 1
2 | 42 0
2 | 21 0
2 | 10 1
2 | 5 0
2 | 2 1
2 | 1 0
| 0 1

10101001_2

Division	Result	Remainder	Build base 2 number
169 / 2	84	1	XXXXXXX1
84 / 2	42	0	XXXXXX01
42 / 2	21	0	XXXXX001
21 / 2	10	1	XXXX1001
10 / 2	5	0	XXX01001
5 / 2	2	1	XX101001
2 / 2	1	0	X0101001
1 / 2	0	1	**10101001**

65_{10}

Division	Result	Remainder	Build base 2 number
65 / 2	32	1	XXXXXX1
32 / 2	16	0	XXXXX01
16 / 2	8	0	XXXX001
8 / 2	4	0	XXX0001
4 / 2	2	0	XX00001
2 / 2	1	0	X000001
1 / 2	0	1	**1000001**

Converting from binary (base 2) to hexadecimal (base 16)

$1011\ 1101\ 0011\ 1110_2$

binary	1011	1101	0011	1110
decimal	11	13	3	14
hexadecimal	B	D	3	E

$1011\ 1101\ 0011\ 1110_2$ = $\mathbf{BD3E_{16}}$

<table>
<tr>
<td>$11\ 0010\ 1110_2$</td>
<td>
<table>
<tr><td>binary</td><td>0011</td><td>0010</td><td>1110</td></tr>
<tr><td>decimal</td><td>3</td><td>2</td><td>14</td></tr>
<tr><td>hexadecimal</td><td>3</td><td>2</td><td>E</td></tr>
</table>
$11\ 0010\ 1110_2$ = $\mathbf{32E_{16}}$
</td>
</tr>
</table>

Converting from hexadecimal (base 16) to binary (base 2)

<table>
<tr>
<td>$F23A_{16}$</td>
<td>
<table>
<tr><td>hexadecimal</td><td>F</td><td>2</td><td>3</td><td>A</td></tr>
<tr><td>decimal</td><td>15</td><td>2</td><td>3</td><td>10</td></tr>
<tr><td>binary</td><td>1111</td><td>0010</td><td>0011</td><td>1010</td></tr>
</table>
$F23A_{16}$ = $1111\ 0010\ 0011\ 1010_2$
</td>
</tr>
<tr>
<td>$C4D_{16}$</td>
<td>
<table>
<tr><td>hexadecimal</td><td>C</td><td>4</td><td>D</td></tr>
<tr><td>decimal</td><td>12</td><td>4</td><td>13</td></tr>
<tr><td>binary</td><td>1100</td><td>0100</td><td>1101</td></tr>
</table>
$C4D_{16}$ = $1100\ 0100\ 1101_2$
</td>
</tr>
</table>

Converting from hexadecimal (base 16) to decimal (base 10)

<table>
<tr>
<td>$F23A_{16}$
<table>
<tr><td>op</td><td></td><td></td><td></td></tr>
<tr><td>x16</td><td>F</td><td>15</td><td>240</td></tr>
<tr><td>+2</td><td>2</td><td>2</td><td>242</td></tr>
<tr><td>x16</td><td></td><td></td><td>3872</td></tr>
<tr><td>+3</td><td>3</td><td>3</td><td>3875</td></tr>
<tr><td>x16</td><td></td><td></td><td>62000</td></tr>
<tr><td>+10</td><td>A</td><td>10</td><td>62010_{10}</td></tr>
</table>
</td>
<td>$(A \times 16^0) + (3 \times 16^1) + (2 \times 16^2) + (F \times 16^3) =$
$(10 \times 16^0) + (3 \times 16^1) + (2 \times 16^2) + (15 \times 16^3) =$
$10 + 48 + 512 + 61440 =$
62010_{10}</td>
</tr>
<tr>
<td>$C4D_{16}$</td>
<td>$(D \times 16^0) + (4 \times 16^1) + (C \times 16^2) =$
$(13 \times 16^0) + (4 \times 16^1) + 12 \times 16^2) =$
$13 + 64 + 3072 = \mathbf{3149_{10}}$</td>
</tr>
</table>

Converting from decimal (base 10) to hexadecimal (base 16)

<table>
<tr><td>16_{10}</td><td>

Division	Result	Remainder	Build base 16 number
16 / 16	1	0	0
1 / 16	0	1	1

16_{10} = $\mathbf{10_{16}}$
</td></tr>
<tr><td>1019_{10}</td><td>

Division	Result	Remainder	Build base 16 number
1019 / 16	63	11	B
63 / 16	3	15	F
3 / 16	0	3	3

1019_{10} = $\mathbf{3FB_{16}}$
</td></tr>
</table>

Sign and Magnitude, 1's complement, 2's complement

4-bit binary number	Unsigned	Signed	1's complement	2's complement
0000	0	+0	+0	0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6

1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Using 4-bit to represent numbers, find the value of the binary numbers using unsigned, signed, 1's complement and 2's complement number system.

4-bit binary number	Unsigned	Signed	1's complement	2's complement
0111_2 For **positive** numbers, sign-magnitude = 1's complement = 2's complement.	$2^0 + 2^1 + 2^2$ = $\mathbf{7_{10}}$	Check most significant bit: 0 is positive. + 111_2 = $\mathbf{+ 7_{10}}$	Same as sign-magnitude Hence, $\mathbf{7_{10}}$	Same as sign-magnitude Hence, $\mathbf{7_{10}}$
1011_2 For **negative** numbers, 1's complement & 2's complement will be *different* from sign-magnitude.	$2^0 + 2^1 + 2^3$ = $\mathbf{11_{10}}$	Check most significant bit: 1 is negative. - 011_2 = $\mathbf{- 3_{10}}$	Most significant bit is 1, which is negative. Flip the bits: 1011 -> 0100_2 Calculate as usual: $0100_2 = 4_{10}$ Add back sign, -ve: Hence, $\mathbf{-4_{10}}$	Most significant bit is 1, which is negative. Flip the bits: 1011 -> 0100_2 Add 1 to the bits: 0100 + 1 = 0101_2 Calculate as usual: $0101_2 = 5_{10}$ Add back sign, -ve: Hence, $\mathbf{-5_{10}}$

Using a 4-bit system, represent the decimal number using unsigned, signed, 1's complement and 2's complement number system.

Decimal number	**Unsigned**	**Signed**	**1's complement**	**2's complement**
5 For **positive** numbers, sign-magnitude = 1's complement = 2's complement.	Calculate as usual $5_{10} = 2^0 + 2^2$ = $\mathbf{0101_2}$	Calculate as usual $5_{10} = 2^0 + 2^2$ = $\mathbf{0101_2}$	Same as sign-magnitude: $\mathbf{0101_2}$	Same as sign-magnitude: $\mathbf{0101_2}$
-5 For **negative** numbers, 1's complement & 2's complement will be *different* from sign-magnitude.	Not considered for -ve numbers	Calculate as usual 0101_2 Change the sign bit to 1 (negative) -5_{10} = $\mathbf{1101_2}$	Calculate as usual 0101_2 Flip the bits -5_2 = $\mathbf{1010_2}$	Calculate as usual: 0101_2 Flip the bits: 0101 -> 1010_2 Add 1 to the bits: 1010 + 1 = $\mathbf{1011_2}$

Add the following numbers using 4-bit 2's complement number system

I. 3+4

$3_{10} = 0011_2$
$4_{10} = 0100_2$
$0011 + 0100 = 0111_2 = 7_2$

II. 5-2

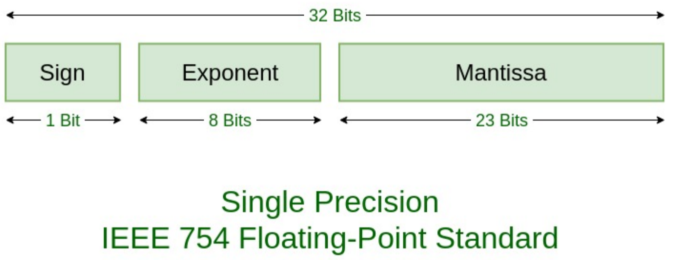
5 - 2 = 5 + (-2)
$5_{10} = 0101_2$
$-2_{10} = -\ 0010_2 \rightarrow 1101_2$ (flipped) $\rightarrow\ 1110_2$ (added 1 bit)
5 + (-2) = 0101 + 1110 = 10011_2

ignore the carry bit and look at rightest 4 bits: $\mathbf{0011_2}$
$0011_2 = 3_{10}$

When would an overflow occur? Give 2 examples.

I. Positive number + positive number gives negative number.
II. Negative number + negative number gives positive number.

Floating-Point Representation



Compute the IEEE standard floating-point representation of 1593_{10}

1. Convert to binary exponent form:

1593_{10} = $110\ 0011\ 1001_2$ = $1.10\ 0011\ 1001 \times 2^{10}$

2. Compute sign-bit: positive = 0 ; negative = 1

Sign bit is **0** because positive exponent.

3. Compute 23-bit mantissa (ignore most significant bit and pad 0 to right)

23 bit mantissa = **1000 1110 0100 0000 0000 000**

4. Compute exponent using excess-k method:

$k = 2^7 - 1 = 127$; exponent = 10

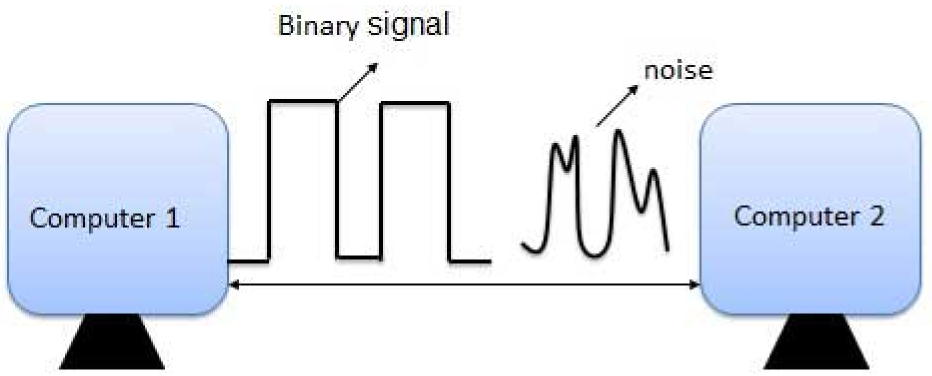
8-bit exponent = exponent + k

In 8-bit excess-127, this is 10 + 127 = 137 = $\mathbf{1000\ 1001_2}$

Therefore, 1593_{10} = $\mathbf{0\ 1000\ 1001\ 1000\ 1110\ 0100\ 0000\ 0000\ 000_2}$

Error Detection

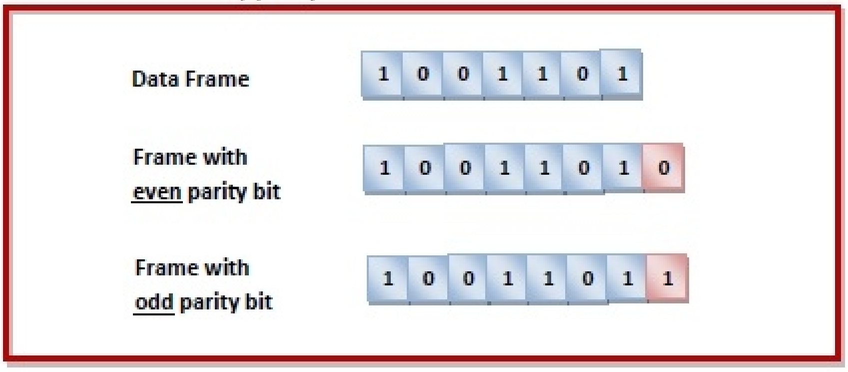
Detect whether there is an error that causes one of the bits to be changed.



- Parity bits
- Checksum
- Cyclic Redundancy Checks (CRC)

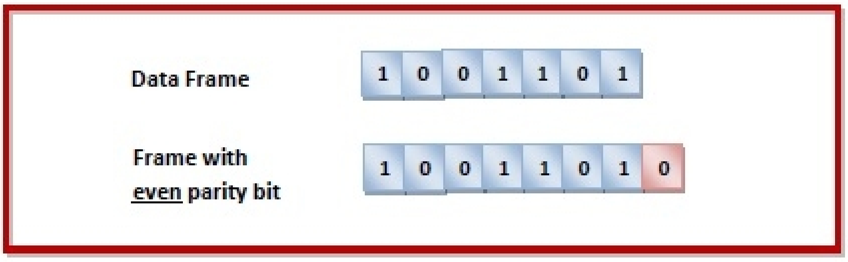
Parity Bits

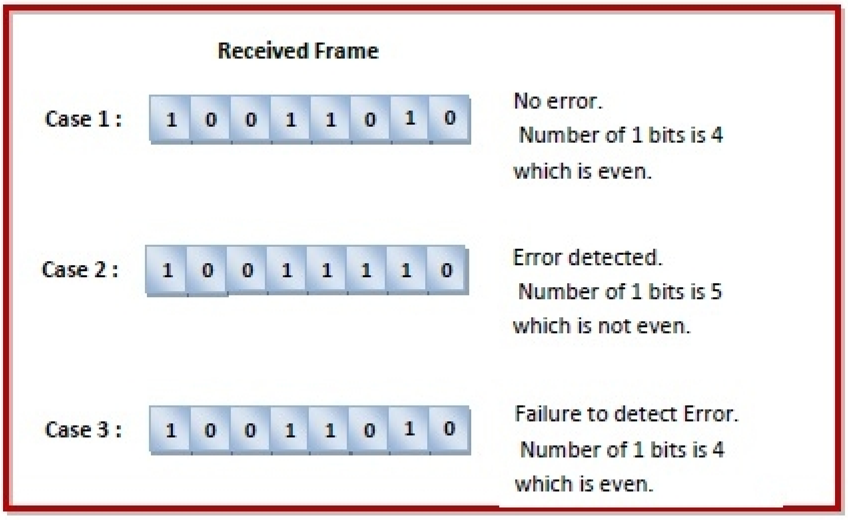
- goal of adding a parity bit → detect 1 single bit change.
- requires an additional bit to be added - determine whether *even* or *odd*



Even Parity	Odd Parity
• If the number of 1s is **even**, parity bit value is **0.** • If the number of 1s is **odd**, parity bit value is **1**.	• If the number of 1s is **odd**, parity bit value is **0**. • If the number of 1s is **even**, parity bit value is **1**.

- Disadvantage: cannot detect **multiple bits error**
- **Example:** sender sends data 1001 101 using **even parity:**





Checksum

- Instead of agreeing on odd or even, one now needs to agree on a particular number.
- Then, the numbers of the message are added up and then divided by the number.
- The remainder is added to the message as checksum. The same process is executed for checking the message. If the remainder mismatches, the message was changed.
- Example for a checksum of message **43 52 43 30 31 30** (agreed divisor = **16**)

sum the numbers	43+52+43+30+31+30=229
divide by agreed divisor	229/16 = 14 R 5 Remainder = **5**
generate message including checksum	**43 52 43 30 31 30 5**
Receiver frame:	
Case 1: single-bit error **43 52 43 29 31 30 5**	• 228/16=14 R **4** (!= 5). • checksum 5 does not match → error is found.
Case 2: multiple-bit error **43 50 43 29 31 30 5**	• 226/16=14 R **2** (!= 5). • checksum 5 does not match → error is found.
Case 3: multiple-bit error that cancels out each other **43 54 43 28 31 30 5**	• 229/16=14 R **5** (==5) • checksum 5 matches → error undetected even though wrong!

- Disadvantage: can in principle detect multiple errors, but only if they don't cancel each other out or are bigger than the divisor agreed on.

Cyclic Redundancy Check (CRC)

- Instead of adding up the numbers, now the numbers are concatenated to build one large number that is then divided by the agreed number.
- Example: 43 52 43 30 31 30 with 16 as the agreed divisor:
    - CRC is calculated by taking the remainder of 435243303130 / 16, which is 10
- It is much more stable than just taking a checksum.

Does CRC code provide security?

CRC does not provide security - attackers can manipulate messages and compute a new CRC code. CRC is about error detection (safety) not malicious attacks (security).