INCHEON NATIONAL UNIVERSITY

IAA6007: Computer Architecture
Ch.3. Data Representation

Wooil Kim
Dept. of Computer Science & Engineering
Incheon National University

2019 Fall

3.1 Data types

- Number system
  - Decimal 724.5 = $7\text{x}10^2 + 2\text{x}10^1 + 3\text{x}10^0 + 5\text{x}10^{-1}$
  - Binary 101101 = $1\text{x}2^5 + 0\text{x}2^4 + 1\text{x}2^3 + 1\text{x}2^2 + 0\text{x}2^1 + 1\text{x}2^0 = 45$
  - Octal, hexadecimal
- Conversion

Integer = 41

$$\begin{array}{r|l} 41 & \\ 20 & 1 \\ 10 & 0 \\ 5 & 0 \\ 2 & 1 \\ 1 & 0 \\ 0 & 1 \end{array}$$

$(41)_{10} = (101001)_2$

Fraction = 0.6875

$$\begin{array}{r} 0.6875 \\ 2 \\ \hline 1.3750 \\ \text{x}\ \ 2 \\ \hline 0.7500 \\ \text{x}\ \ 2 \\ \hline 1.5000 \\ \text{x}\ \ 2 \\ \hline 1.0000 \end{array}$$

$(0.6875)_{10} = (0.1011)_2$

$$(41.6875)_{10} = (101001.1011)_2$$

- Conversion

1	2	7	5	4	3	Octal
1	010	111	101	100	011	Binary

1010	1111	0110	0011	Binary
A	F	6	3	Hexadecimal

3.1 Data types

- Binary coded octal numbers

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

3.1 Data types

- Binary coded hexadecimal numbers

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	Code for one hexadecimal digit
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	
7	0111	7	
8	1000	8	
9	1001	9	
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

3.1 Data types

- Binary coded decimal (BCD) numbers

Decimal number	Binary-coded decimal (BCD) number	
0	0000	
1	0001	
2	0010	
3	0011	Code
4	0100	for one
5	0101	decimal
6	0110	digit
7	0111	
8	1000	
9	1001	
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

3.1 Data types

- Alphanumeric representation
  - ASCII code

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	−	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

3.2 Complements

- Given a binary number N having n-bits
  - 1's complement of N = $(2^n - 1) - N$
    - 1's complement of 1011000 = $(2^7 - 1) - 1011000 = 1111111 - 1011000 = 0100111$
  - 2's complement of N = $2^n - N$
    - 2's complement of 1011000 = $10000000 - 1011000 = 0101000$

3.2 Complements

- Subtraction with complement – 2's complement

$M - N = M + (-N)$

$M + (2^n - N) = 2^n + (M - N)$

If $M \geq N$ end carry 2^n is discarded and the result $M - N$ is left

else the sum is $2^n - (N - M)$ which is the 2's complement of $(N - M)$

  - Ex) X = 1010100 = 84, Y = 1000011 = 67

(a) $X - Y$

$$
\begin{array}{r}
1010100 \\
+\ 0111101 \\
\hline
(1)0010001 = 17
\end{array}
$$

(b) $Y - X$

$$
\begin{array}{r}
1000011 \\
+\ 0101100 \\
\hline
1101111
\end{array}
$$

2's complement of

$0010001 = (17)_{10}$

3.2 Complements

- Examples

$(+2)_{10} + (+3)_{10} = (+5)_{10}$

$$\begin{array}{r} 0010 \\ +\ 0011 \\ \hline 0101 \end{array}$$

$(-6)_{10} + (+3)_{10} = (-3)_{10}$

$$\begin{array}{r} 1010 \\ +\ 0011 \\ \hline 1101 \end{array}$$

$(-3)_{10} + (+5)_{10} = (+2)_{10}$

$$\begin{array}{rr} & 1101 \\ + & 0101 \\ \hline 1 & 0010 \end{array}$$

$(+5)_{10} - (+2)_{10} = (+5)_{10} + (-2)_{10} = (+3)_{10}$

$$\begin{array}{rr} & 0101 \\ + & 1110 \\ \hline 1 & 0011 \end{array} = (+3)_{10}$$

3.2 Complements

- Examples

$(+4)_{10} + (+5)_{10} = (+9)_{10}$

$$\begin{array}{rr} & 0100 \\ + & 0101 \\ \hline & 1001 \end{array} = (-7)_{10}$$

$(-7)_{10} + (-6)_{10} = (-13)_{10}$

$$\begin{array}{rr} & 1001 \\ + & 1010 \\ \hline 1 & 0011 \end{array} = (+3)_{10}$$

$(+7)_{10} - (-5)_{10} = (+7)_{10} + (+5)_{10} = (+12)_{10}$

$$\begin{array}{rr} & 0111 \\ + & 0101 \\ \hline & 1100 \end{array} = (-4)_{10}$$

$(-6)_{10} - (+4)_{10} = (-6)_{10} + (-4)_{10} = (-10)_{10}$

$$\begin{array}{rr} & 1010 \\ + & 1100 \\ \hline 1 & 0110 \end{array} = (+6)_{10}$$

3.3 Fixed point representation

- Signed magnitude representation
  - Leftmost bit is needed as the sign of the number (0 for +, 1 for -)
- Signed 1's complement representation
  - Positive numbers – same as signed magnitude representation
  - Negative numbers – 1's complement
- Signed 2's complement representation
  - Positive numbers – same as signed magnitude representation
  - Negative numbers – 2's complement

3.3 Fixed point representation

- Ex) n = 3

i)	ii)	iii)
$011 = 3$	$011 = 3$	$011 = 3$
$010 = 2$	$010 = 2$	$010 = 2$
$001 = 1$	$001 = 1$	$001 = 1$
$000 = 0$	$000 = 0$	$000 = 0$
$100 = -0$	$110 = -1$	$111 = -1$
$101 = -1$	$101 = -2$	$110 = -2$
$110 = -2$	$100 = -3$	$101 = -3$
$111 = -3$	$111 = -0$	$100 = -4$
$-3 \sim 3$	$-3 \sim 3$	$-4 \sim 3$
$-(2^{n-1}-1) \sim 2^{n-1}-1$	$-(2^{n-1}-1) \sim 2^{n-1}-1$	$-2^{n-1} \sim 2^{n-1}-1$

3.3 Fixed point representation

- Ex) n = 4

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	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	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

3.3 Fixed point representation

- Overflow
  - An overflow may occur if the added two numbers are bth positive or both negative
  - Ex)

	70	01000110
+	80	01010000
	150	10010110

  - Overflow condition
    - The carry out of the sign bit position $\oplus$ the carry into the sign bit position

3.3 Fixed point representation

- Range of representation

	Word size w			
Value	8	16	32	64
$UMax_w$	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	255	65,535	4,294,967,295	18,446,744,073,709,551,615
$TMin_w$	0x80	0x8000	0x80000000	0x8000000000000000
	−128	−32,768	−2,147,483,648	−9,223,372,036,854,775,808
$TMax_w$	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	127	32,767	2,147,483,647	9,223,372,036,854,775,807
−1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	0x00	0x0000	0x00000000	0x0000000000000000

3.4 Floating point representation

- IEEE floating point representation
- Numerical form

$$V = (-1)^s * M * 2^E$$

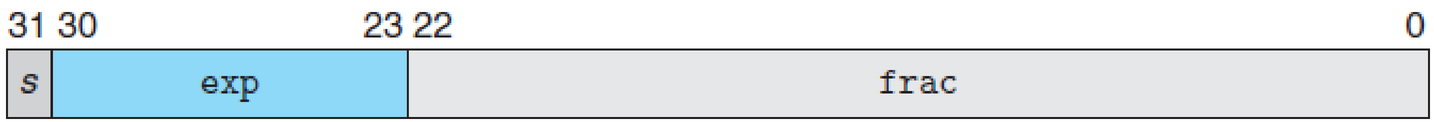
Sign bit Significand Exponent

  - Sign bit s determines whether number is negative of positive
  - Significand M normally a fractional value in range [1.0, 2.0) or [0, 1)
  - Exponent E weight value by power of two
- Encoding
  - MSB is sign bit
  - **`exp`** field encodes E, k bits (note: encode != is)
  - **`frac`** field encodes M, n bits

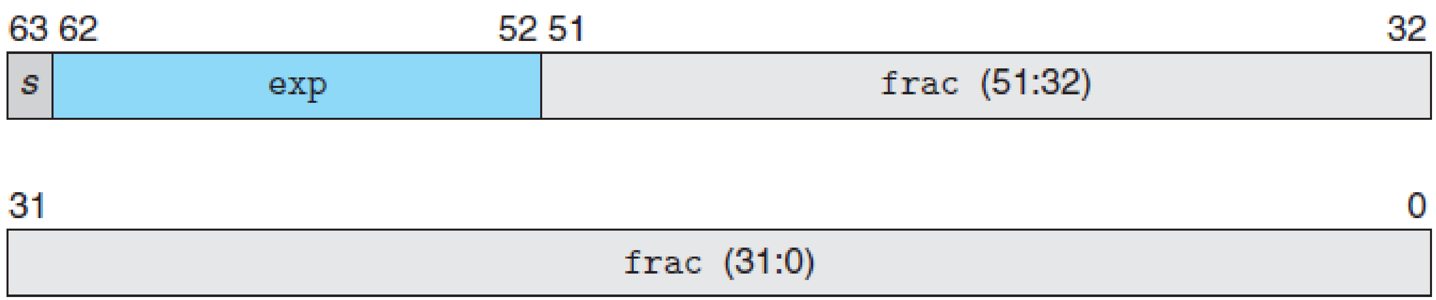
`s`	`exp`	`frac`

3.4 Floating point representation

- Precisions
  - Single precision
    - k = 8 exp bits, n = 23 frac bits (32b total)



  - Double precision
    - k = 11 exp bits, n = 52 frac bits (64b total)



  - Extended precision
    - k = 15 exp bits, n = 63 frac bits
    - Only found in Intel-compatible machines

3.4 Floating point representation

- Categories – three different cases, depending on value exp
  - 1. Normalized, the most common

s	$\neq 0$ & $\neq 255$	f

  - 2. Denormalized

s	0	0	0	0	0	0	0	0	f

  - 3. Special values – infinity and NaN

3a. Infinity

s	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3b. NaN

s	1	1	1	1	1	1	1	1	$\neq 0$

3.4 Floating point representation

- Case 1: normalized number values
- Condition
  - **`exp`** ≠ 000 … 0 and **`exp`** ≠ 111 … 1
- Exponent coded as biased value
  - E = Exp – Bias
    - Exp: unsigned value denoted by **`exp`**
    - Bias: Bias value $2^{k-1} - 1$, k is number of exponent bits
      - Single precision: 127 (Exp: 1, …, 254, E: -126, …, 127)
      - Double precision: 1023 (Exp: 1, …, 2046, E: -1022, …, 1023)
- Significand coded with implied leading 1
  - M = $1.xxx \ldots x_2$ (1+f where f = $0.xxx_2$)
    - xxx … x: bits of **`frac`**
    - Minimum when 000 … 0 (M = 1.0)
    - Maximum when 111 … 1 (M = 2.0 - ϵ)
    - Get extra leading bit for "free"

3.4 Floating point representation

- Case 1: normalized number values
- Value
  - Float F = 15213.0
  - $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$
- Significand
  - $M = 1.1101101101101_2$
  - **`frac`** = 11011011011010000000000
- Exponent
  - E = 13
  - Bias = 127
  - **`exp`** = E + Bias = 140 = 10001100_2

Floating point representation

Hex:	4	6	6	D	B	4	0	0
Binary:	0100	0110	0110	1101	1011	0100	0000	0000
140:	100	0110	0					
15213:			110	1101	1011	01		

3.4 Floating point representation

- Case 1: normalized number values
- Value
  - Float F = 12345.0
  - $12345_{10} = 11000000111001_2 = 1.1000000111001_2 \times 2^{13}$
- Significand
  - $M = 1.1000000111001_2$
  - frac = 10000001110010000000000
    - Drop leading 1, add 10 zeros
- Exponent
  - E = 13
  - Bias = 127
  - $Exp = E + Bias = 140 = 10001100_2$

Floating point representation

Hex:	4	6	4	0	E	4	0	0
Binary:	0100	0110	0100	0000	1110	0100	0000	0000

3.4 Floating point representation

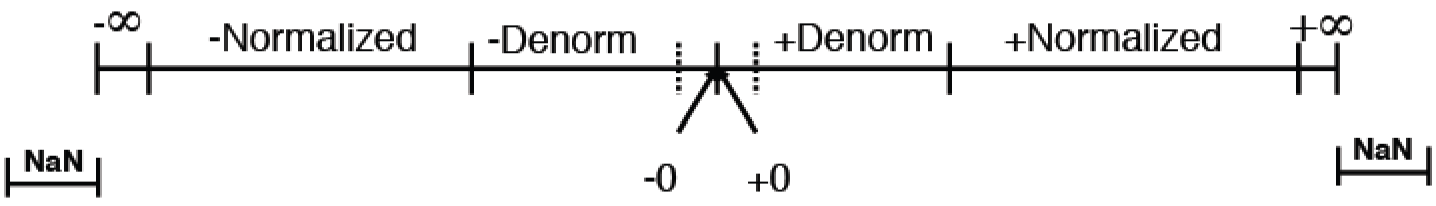
- Case 2: denormalized values
- Condition
  - **`exp`** = 000 … 0
- Value
  - Exponent value E = 1 – Bias
    - Note: not simply E = - Bias
  - Significand value M = $0.xxx \ldots x_2$ (0.f)
    - xxx … x: bits of **`frac`**
- Cases
  - **`exp`** = 000 … 0, **`frac`** = 000 … 0
    - Represent value 0
    - Note that have distinct value +0 and -0
  - **`exp`** = 000 … 0, **`frac`** ≠ 000 … 0
    - Number very close to 0.0

3.4 Floating point representation

- Case 3: special values
- Condition
  - **`exp`** = 111 … 1
- Cases
  - **`exp`** = 111 … 1, **`frac`** = 000 … 0
    - Represent value ∞ (infinity)
    - Operation that overflows
    - Both positive and negative
    - E.g., 1.0/0.0 = -1.0/-0.0 = + ∞, 1.0/-0.0 = - ∞
  - **`exp`** = 111 … 1, **`frac`** ≠ 000 … 0
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - E.g., √-1, (∞ - ∞)

3.4 Floating point representation

- Summary of floating point real number encodings



Description	exp	frac	Single precision Value	Single precision Decimal	Double precision Value	Double precision Decimal
Zero	$00 \cdots 00$	$0 \cdots 00$	0	0.0	0	0.0
Smallest denorm.	$00 \cdots 00$	$0 \cdots 01$	$2^{-23} \times 2^{-126}$	1.4×10^{-45}	$2^{-52} \times 2^{-1022}$	4.9×10^{-324}
Largest denorm.	$00 \cdots 00$	$1 \cdots 11$	$(1-\epsilon) \times 2^{-126}$	1.2×10^{-38}	$(1-\epsilon) \times 2^{-1022}$	2.2×10^{-308}
Smallest norm.	$00 \cdots 01$	$0 \cdots 00$	1×2^{-126}	1.2×10^{-38}	1×2^{-1022}	2.2×10^{-308}
One	$01 \cdots 11$	$0 \cdots 00$	1×2^{0}	1.0	1×2^{0}	1.0
Largest norm.	$11 \cdots 10$	$1 \cdots 11$	$(2-\epsilon) \times 2^{127}$	3.4×10^{38}	$(2-\epsilon) \times 2^{1023}$	1.8×10^{308}

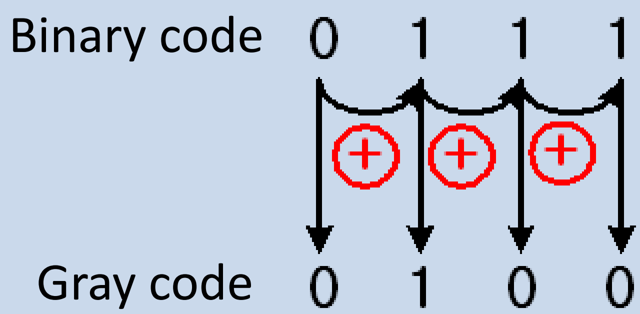
3.5 Other binary codes

- 4-bit Gray code

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

3.5 Other binary codes

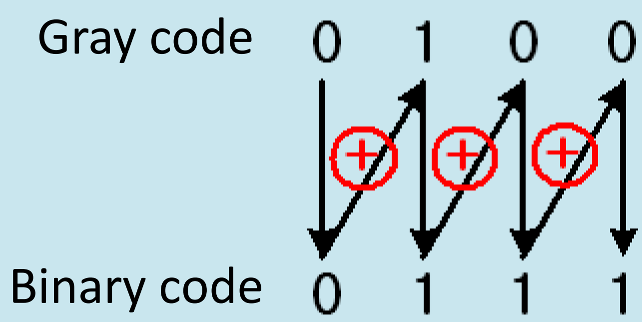
- Binary code -> Gray code



XOR

input		output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

- Gray code -> binary code

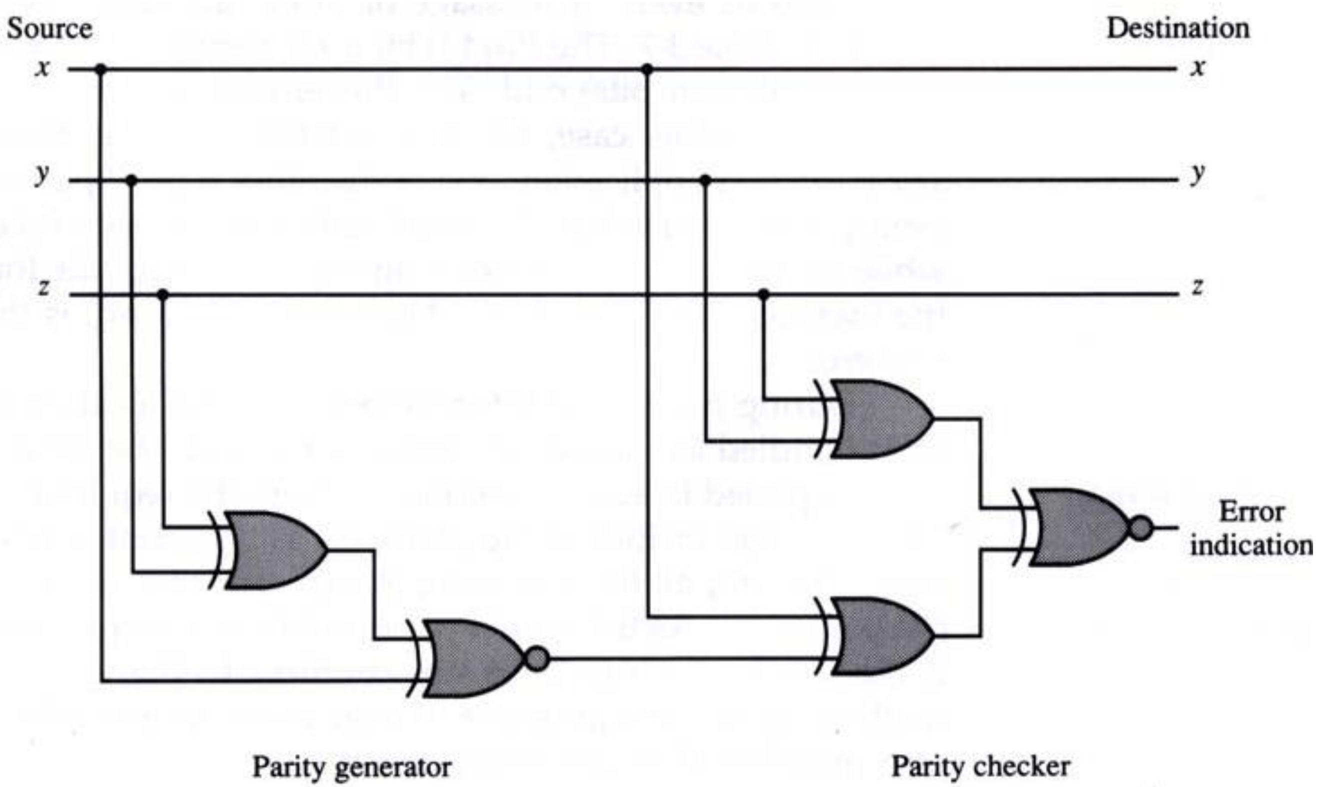


3.5 Other binary codes

Decimal digit	BCD 8421	2421	Excess-3	Excess-3 gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010
Unused bit combi-nations	1010	0101	0000	0000
	1011	0110	0001	0001
	1100	0111	0010	0011
	1101	1000	1101	1000
	1110	1001	1110	1001
	1111	1010	1111	1011

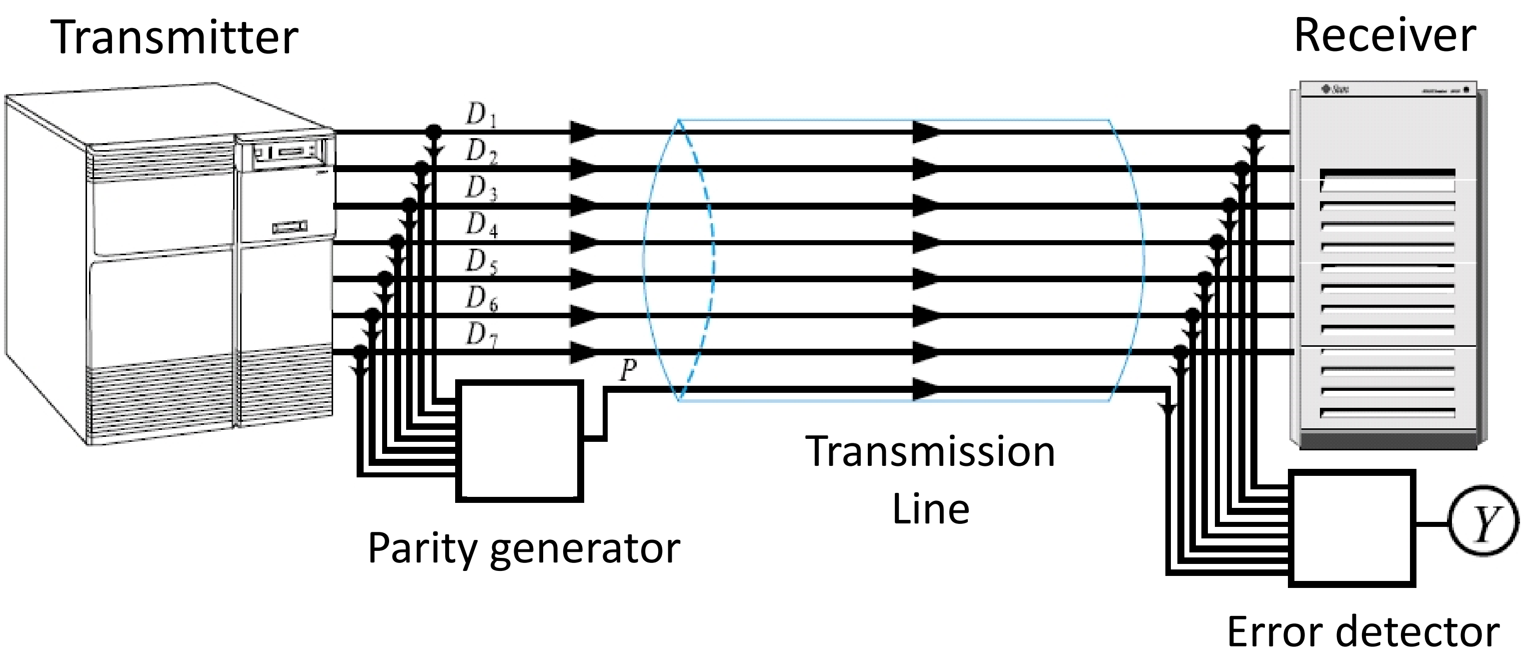
3.6 Error detection code

- Parity bit – an extra included to make the total number of 1's either odd or even
- Error detection with odd parity



3.6 Error detection code

- Error detection using parity bit for data transmission system
  - Transmitter (sender): parity generator
  - Receiver: error detector
  - Y = 0 (no error), Y = 1 (error occurred)



3.6 Error detection code

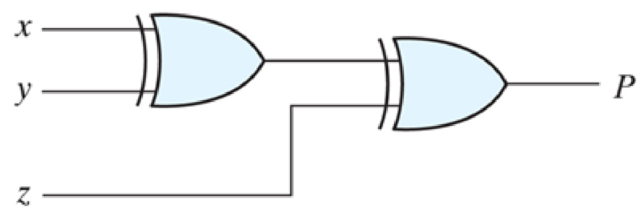
- Parity generator for even parity

$$P = D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7$$

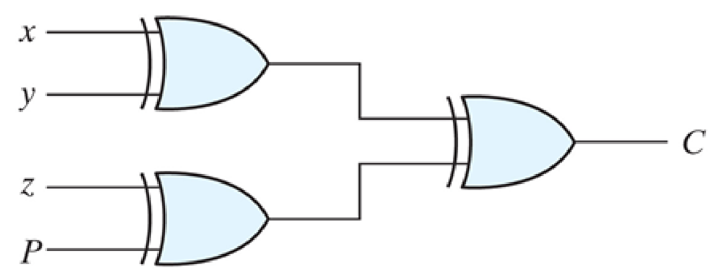
- Error detector for even parity

$$Y = D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus P$$

Four Bits Received				Parity Error Check
x	**y**	**z**	**P**	**C**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



(a) 3-bit even parity generator



(b) 4-bit even parity checker

Problems

- 3-3, 3-4, 3-5, 3-7, 3-9,
- 3-11, 3-12, 3-13, 3-14,
- 3-15, 3-18, 3-22, 3-25, 3-26