Computer System Architecture

Ms. Yuzana Hlaing
M.E(IT), Ph.D. Candidate(thesis)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Periods	Lectures
Week-1	Introduction to Computer System Architecture
Week-2	Data Presentations
Week-3	Register Transfer and Microoperations
Week-4	Basic Computer Organization and Design
Week-5	Programming the Basic Computer
Week-6	Microprogrammed Control
Week-7	Central Processing Unit
Week-8	Pipeline and Vector Processing
Week-9	Computer Arithmetic
Week-10	Input-Output Organization
Week-11	Memory Organization
Week-12	Multiprocessors

Lecture-2

Data Presentations

Lecture-2
Data Presentations

- Data Types
- Complements
- Fixed-point Representation
- Floating-point Representation
- Different Types of Binary Codes
- Error Detection Codes

Data Types

Data Types

- Binary information in digital computers is stored in a memory or processor registers.
- Register contains either data or control information.
- **Control information** is a bit or a group of bits used to specify the sequence of command signals needed for manipulation of the data.
- **Data** are the numbers or other binary-coded information that are operated on to achieve required computational results.
- In digital computers, data types found in the registers may be classified into three categories.
  1. Numbers used in arithmetic computation
  2. Letters of the alphabet used in data processing
  3. Discrete symbols used for specific purposes

Number Systems

- A number system of **base** or **radix**, r is a system that uses distinct symbols for r digits.
- Numbers are represented by a string of digit symbols.
- A **decimal** number system employs the radix 10 system.
- For example, the string of digits 742.5 is interpreted to represent the quantity in decimal form as in follows,

  $7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0 + 5 \times 10^{-1}$
- A **binary** number system uses the radix 2 system.
- For example, the string of digits 101101 is interpreted to represent the quantity to get decimal form as in follows,

  $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
- The equality between decimal (radix 10) and binary (radix 2) number system can be described in the following.

  $101101_2 = 45_{10}$

- A **octal number system** uses the radix 8 system.
- For example, the string of digits 736.4 is interpreted to represent the quantity in decimal form as in follows,

  $7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$
- The equality between decimal (radix 10) and octal (radix 8) number system can be described in the following.

  $736.4_8 = 478.5_{10}$
- A **hexadecimal number system** uses the radix 16 system.
- For example, the string of digits F3 is interpreted to represent the quantity in decimal form as in follows,

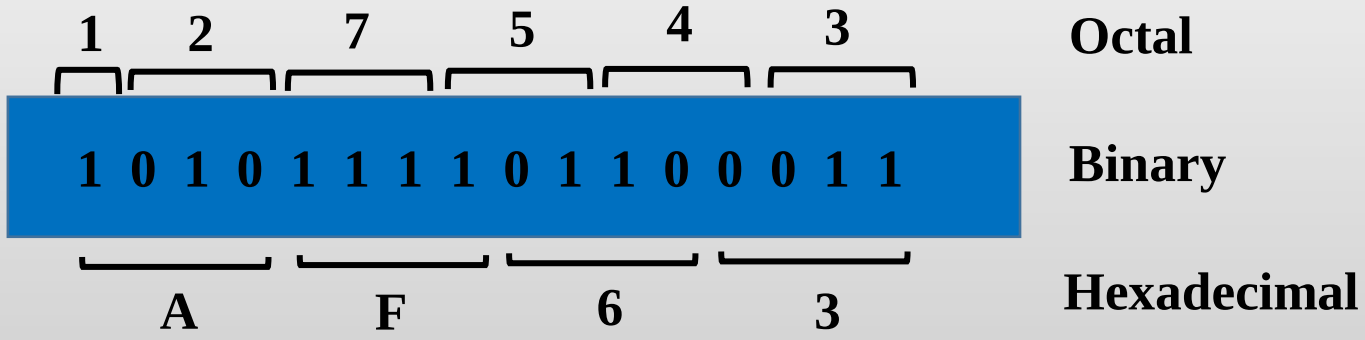
  $F \times 16^1 + 3 \times 16^0$
- The equality between decimal (radix 10) and octal (radix 8) number system can be described in the following.

  $F3_{16} = 243_{10}$

- **Conversion** from decimal to its equivalent representation in the radix **r** system is carried out by separating the numbers into its integer and fraction parts separately.
- The conversion of a decimal integer into a base **r** representation is done by successive division by r and accumulation of the remainder.
- The conversion of a decimal fraction into a base **r** representation is accomplished by successive multiplication by **r** and accumulation of the integer digits so obtained.
- For example, the conversion of decimal **41.6875** into binary is done by separating the number into integer part **41** and fraction part **.6875**.
- The integer part **41** is repeatedly divided by r = 2 until the integer quotient becomes **0**.
- The fraction part **.6875** is converted by multiplying by r= 2 until the fraction part becomes zero or the required accuracy is attained.
- Finally, the successive conversion is done by combining the two parts.

Relationship between Binary, Octal , and Hexadecimal Numbers

- The conversion from and to binary and octal is that each octal digit corresponds to three binary digits, $2^3 = 8$.
- The conversion from binary to octal is accomplished by partitioning the binary number into groups of three bits each.
- The conversion from and to binary and hexadecimal is that each octal digit corresponds to three binary digits, $2^4 = 16$.
- The conversion from binary to hexadecimal is accomplished by partitioning the binary number into groups of four bits each.
- For example, a 16-bit register stores a string of 1's and 0's of the binary value and it can be converted into octal and hexadecimal according to the following figure.



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	

Binary-coded Hexadecimal Numbers

Hexadecimal Number	Binary-coded Hexadecimal	Decimal Equivalent
0	0000	0
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

Code for one hexadecimal digit

Binary-coded Hexadecimal Numbers

Hexadecimal Number	Binary-coded Hexadecimal	Decimal Equivalent	
A	1010	0	↑
B	1011	1	Code
C	1100	2	for one
D	1101	3	hexadecimal
E	1110	4	digit
F	1111	5	↓
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

Decimal Representation

- When decimal numbers are used for internal arithmetic computations, they are converted to a binary code with four bits per digit in order that computer may perform all arithmetic operations in binary and then convert the binary results back to decimal for the users.
- A **binary code** is a group of n-bits that assume up to 2^n distinct combinations of 1's and 0's with each combination representing one element of the set that is being coded.
- A **binary-coded decimal (BCD)** is a binary code that uses four bits per digit to represent each decimal digit 0 to 9.
- The difference between the conversion of decimal numbers into binary and the binary coding of decimal numbers can be clearly seen in the following table.

Difference between binary and BCD Numbers

Decimal Number	Binary-coded Decimal (BCD)	Binary Number
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
10	0001 0000	1000
11	0001 0001	1001
12	0001 0010	1010
13	0001 0011	1011
14	0001 0100	1100
15	0001 0101	1101
16	0001 0110	1111

Alphanumeric Representation

- An **alphanumeric character set** is a set of elements that includes the 10 decimal digits, 26 letters of alphabet and a number special characters such as $, +, !, and so on.
- The standard alphanumeric binary code is **ASCII** (American Standard Code for Information Interchange), which uses seven bits to code 128 characters.
- Each character is represented by a 7-bit code and usually an eighth bit is inserted for parity.
- Ninety-five characters represent **graphic symbols**.
- **Graphic symbols** includes upper and lower case letters, numeral 0 to 9, and special characters.
- Twenty-three characters represent **format effectors**.
- **Format effectors** include functional characters for controlling the layout of printing or display devices.
- The decimal digits in **ASCII** can be converted to **BCD** by removing the three high-order bits.
- By extending **ASCII**, eight bits can be used for 256 characters.

American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	P	101 0000
B	100 0010	Q	101 0001
C	100 0011	R	101 0010
D	100 0100	S	101 0011
E	100 0101	T	101 0100
F	100 0110	U	101 0101
G	100 0111	V	101 0110
H	100 1000	W	101 0111
I	100 1001	X	101 1000
J	100 1010	Y	101 1001
K	100 1011	Z	101 1010
L	100 1100	0	011 0000
M	100 1101	1	011 0001
N	100 1110	2	011 0010
O	100 1111	3	011 0011

Character	Binary code
4	011 0100
5	011 0101
6	011 0110
7	011 0111
8	011 1000
9	011 1001
space	010 0000
.	010 1110
(	010 1000
+	010 1011
$	010 0100
*	010 1010
)	010 1001
-	010 1101
/	010 1111
,	010 1100
=	011 1101

Complements

Complements

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.
- There are two types of complements for each base **r** system, r's complement and (r-1)'s complement.
- For binary system, 2's complement and 1's complement are used.
- For decimal system, 10's complement and 9's complement are used.
- For example in binary system, 1's complement of **N = 101101** is obtained by the complement of each bit like that **010010**.
- 2's complement of **N = 101101** is obtained by adding 1 to the 1's complement of this number like that 010011.
- The complement of the complement is $\mathbf{r^n - (r^n - N)}$ to get the original number back.

Subtraction of Unsigned Numbers

- The subtraction uses the borrow concept that is **1** is borrowed from a higher position when the minuend digit is smaller than the corresponding subtrahend digit.
- The subtraction of two ***n***-digit unsigned numbers **M –N** in base **r** can be done as follows.
  1. Add the minuend M to the r's complement of the subtrahend.
     $$\mathbf{M + (r^n - N) = M - N + r^n}$$
  2. If M>= N, the sum will produce an end carry rn which is discarded.
     M - N
  3. If M < N, the sum does not produce an end carry.
     $$\mathbf{r^n - (N - M)}$$

***Compute the subtraction operation of two unsigned binary numbers; X = 1010100 and Y = 1000011.

Sol:

X	=	1010100
2’s complement of Y	=	+0111101
Sum	=	10010001
Discard end carry 27	=	-10000000
Answer: X - Y	=	0010001

Y	=	1000011
2’s complement of X	=	+0101100
Sum	=	1101111

There is no end carry

Fixed-Point Representation

Fixed-Point Representation

- For a binary number system, the position of binary point is necessary to represent fractions, integers, or mixed integer-fraction numbers.
- There are two ways of specifying the position of binary point in a register: by giving a fixed position or by employing a floating-point representation.
- In fixed-point method, the binary point is always fixed in one position.
- The two positions most widely used are
  1. A binary point in the extreme left of the register to make the stored number a fraction.
  2. A binary point in the extreme right of the register to make the stored number an integer.

Integer Representation

- For signed binary numbers, when an integer is positive number, the sign is represented by **0** and the magnitude by a positive binary number.
- When the integer is negative number, the sign is represented by **1** and the magnitude may be represented in one of these three possible ways.
  1. Signed-magnitude representation
  2. Signed-1's complement representation
  3. Signed-2's complement representation
- The **signed-magnitude representation** of a negative number consists of the magnitude and a negative sign.
- The **signed-1's complement and signed-2's complement representations** of a negative number is represented in either the 1's or 2's complement of its positive value.

- For example, the signed number **14** is stored in the 8-bit register.
- If **14** is the positive number, the sign bit is represented as **0** and the magnitude for **14** is **1110**, so the binary value in the 8-bit register is **00001110**.
- If **14** is the negative number, there are three different ways to represent **-14**:

  In signed-magnitude representation **1 0001110**
  In signed-1's complement representation **1 1110001**
  In signed-2's complement representation **1 1110010**

- For another example, the signed number **14** is stored in the 16-bit register.
- If **14** is the positive number, the sign bit is represented as **0** and the magnitude for **14** is **1110**, so the binary value in the 16-bit register is **0000000000001110**.
- If **14** is the negative number, there are three different ways to represent **-14**:

  In signed-magnitude representation **1 000000000001110**
  In signed-1's complement representation **1 111111111110001**
  In signed-2's complement representation **1 111111111110010**

Arithmetic Addition

- There are two ways for addition of two numbers in signed-magnitude system.
  1. If the signs are the same, add the two magnitudes and give the sum the common sign.
  2. If the signs are different, subtract the smaller magnitude from the larger one and give the result the sign of the larger magnitude.
  3. This is a process that requires the comparison of the signs and the magnitudes and then performing either addition or subtraction.
- For addition of two signed-2's complement numbers, it does not require a comparison or subtraction, only addition and complementation.
  1. Add the two numbers, including the sign bits.
  2. Discard any carry out of the sign (leftmost) bit position.
- For positive numbers, original binary for these numbers and signed-2's complement forms are the same.

Arithmetic Subtraction

- For the subtraction of two signed binary numbers, by giving the negative numbers in 2's complement form, it can be performed as follows.
  1. Take the 2's complement of the subtrahend and add it to the minuend (both including the sign bits).
  2. Discard a carry out of the sign bit position.
- Subtraction operation may be changed to an addition operation if the sign of the subtrahend is changed.

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

- For example, the subtraction of **-6** and **-13** = **(-6) – (-13)** = **+7**
- In binary form, **-6** = **11111010**, and **-13** =**11110011**.
- The subtraction is changed to addition by taking the 2's complement of (**-13**) into binary form to give (**+13**) in binary form.
- **11111010** + **00001101** = **100000111**
- By discarding the end carry, the result is **00000111** = **+7**

Overflow

- When two numbers of **n** digits each are added and the sum occupies **n+1** digits, the **overflow** is occurred.
- An overflow is a problem in digital system due to the finite width of the registers.
- The detection of an overflow after addition of two binary numbers depends on whether the numbers are signed or unsigned.
- For the addition of two unsigned numbers, an overflow can be detected from the end carry out of the most significant position.
- In the case of signed numbers, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
- In this case, the overflow can be detected by taking the logical ex-or operation to the end carry (leftmost bit) and the second leftmost bit.
- An overflow cannot occur after an addition if one number is positive and the other is negative.

Floating-Point Representation

Floating-Point Representation

- The **floating-point representation** of a number has two parts.
- The first part represents a signed, fixed-point number called the **mantissa**.
- The second part designates the position of the decimal or binary point that is called **exponent**.
- The fixed-point mantissa may be a fraction or an integer.
- The value of the exponent indicates the actual position of the decimal point in the fraction.
- Floating-point is interpreted to represent a number in the following form.

$$m \times r^e$$

- A floating-point number is said to be **normalized** if the most significant digit of the mantissa is nonzero.

Different Types of Binary Codes

Different Types of Binary Codes

- In digital computer, the data must be converted to digital form before they can be used.
- Continuous or analog information is converted into digital form by means of a analog-to-digital converter.
- The **reflected binary** or **Gray code** is one of the different types of binary codes that is used for the converted digital data.
- The advantage of gray code over straight binary number is that Gray code changes by only one bit as it sequences from one number to the next.
- Gray code counters are used to provide timing sequences that control the operations in a digital system.

- A **Gray code counter** is a counter whose flip-flops go through a sequence of states specified in the following table.

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

Other Decimal Codes

- Binary codes for decimal digits require a minimum of four bits.
- The most commonly used binary code for decimal digits in arithmetic computation is binary-coded decimal (BCD).
- The disadvantage of using BCD is the difficulty during the computation of 9's complement numbers.
- Two binary codes for decimal digits called **2421** and **excess-3** codes can be applied for 9's complement numbers computation.
- These two codes have a self-complementing property.
- A **self-complementing property** is a property that (9's complement of the decimal number is easily obtained by changing 1's to 0's and 0's to 1's.
- The 2421 is an example of a weighted code.
- In a **weighted code**, the bits are multiplied by the weights indicated and the sum of weighted bits gives the decimal digit.
- The excess-3 code is an unweighted code.

Four Different Binary Codes for Decimal Digits

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

Four Different Binary Codes for Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	Excess-3 gray
	1010	0101	0000	0000
Unused	1011	0110	0001	0001
Bit	1100	0111	0010	0011
combi-	1101	1000	1101	1000
nations	1110	1001	1110	1001
	1111	1010	1111	1011

Error Detection Codes

Error Detection Codes

- A **error detection code** is a binary code that detects digital errors during transmission.
- The detected errors cannot be corrected but their presence is indicated.
- The most common error detection code is the **parity bit**.
- A **parity bit** is an extra bit included with a binary message to make the total number of 1's either odd or even.
- At the sending end, the message is applied to a **parity generator**, where the required parity bit is generated.
- A the receiving end, all the incoming bits are applied to a **parity checker** that checks the proper parity adopted (odd or even).
- The parity method detects the presence of one, three, or any odd number of errors.
- An even number of errors is not detected.

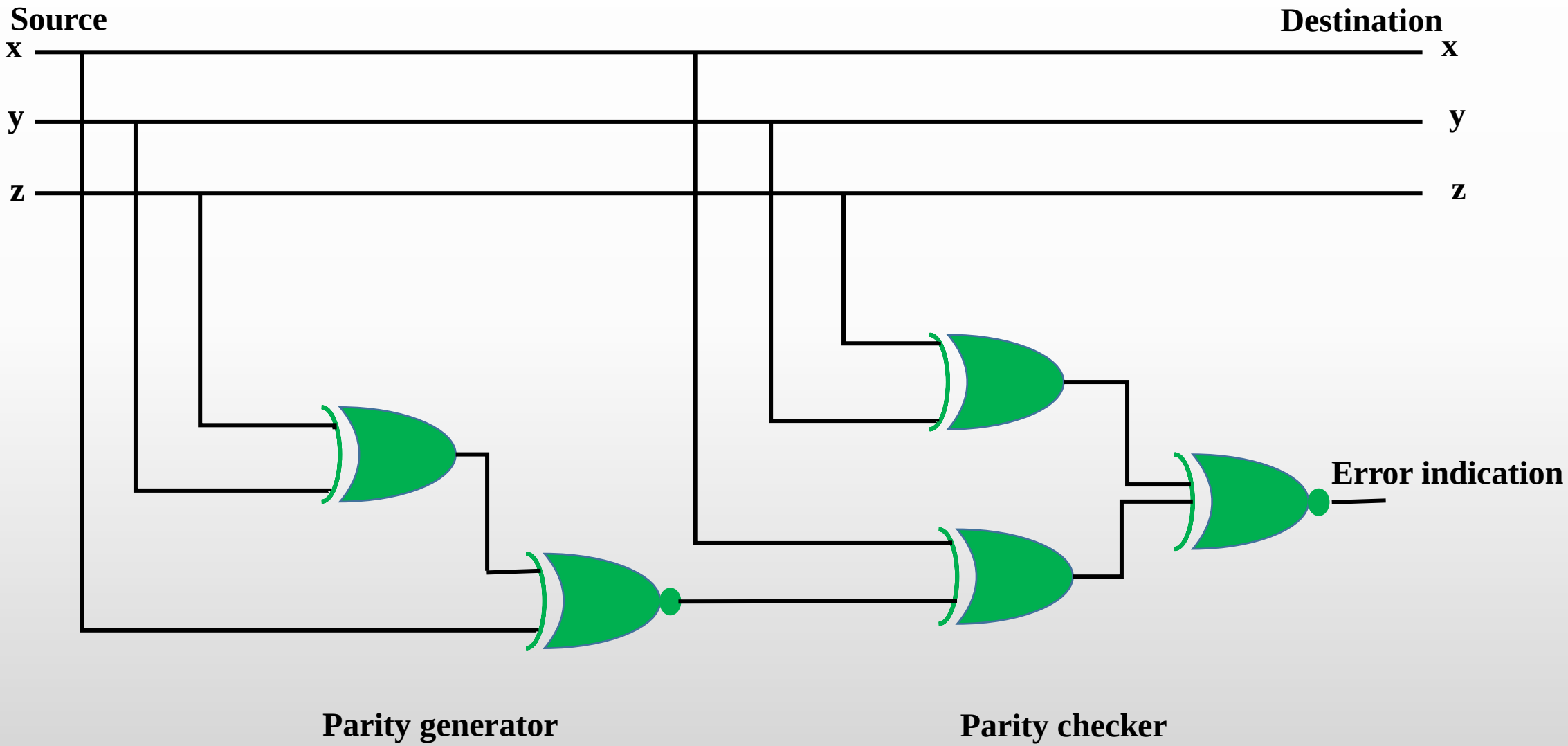
- A message of three bits and two possible parity bits is shown in the following table.

Parity Bit Generation

Message *xyz*	*P* (odd)	*P* (even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

- The parity generator and parity checker networks are logic circuits constructed with exclusive-OR functions.
- The exclusive-OR function of three or more variables is by definition an odd function.
- An odd function is a logic function whose value is binary 1 if and only if an odd number of variables are equal to **1**.
- In figure, the circuit consists of one exclusive-OR and one exclusive-NOR gate.
- The message and the odd-parity bit are transmitted to their destination where they are applied to parity checker.
- The output of the parity checker would be **1** when an error occurs, that is, when the number of 1's in the four inputs is even.
- The exclusive-OR function of the four inputs is an odd function, the output is complemented by using an exclusive-NOR gate.

Error detection with odd parity bit



***Example:** List the ten BCD digits with an even parity and odd parity in the leftmost position (total of five bits per digit).

Sol:

Decimal	Binary with even parity	Binary with odd parity
0	00000	10000
1	10001	00001
2	10010	00010
3	00011	10011
4	10100	00100
5	00101	10101
6	00110	10110
7	10111	00111
8	11000	01000
9	01001	11001

Thank You

Lecture for Next Week

- **Register Transfer and Microoperations**
- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Microoperations
- Logic Microoperations
- Shift Microoperations
- Arithmetic Logic Shift Unit