Lecture #2

-Number Systems, Operations and Codes-

Topics to be discussed

- 2-1 Decimal Numbers
- 2-2 Binary Numbers
- 2-3 Decimal-to-Binary Conversion
- 2-4 Binary Arithmetic
- 2-5 1’s and 2’s Complements of Binary Numbers
- 2-6 Signed Numbers
- 2-7 Arithmetic Operations with Signed Numbers
- 2-8 Hexadecimal Numbers
- 2-9 Octal Numbers
- 2-10 Binary Coded Numbers (BCD)
- 2-11 Digital Codes
- 2-12 Error Detection and Correction Codes

2-1 Decimal Numbers

- These numbers are used every day in our daily lives
- Actually, decimal numbers is a type of weighted system numbers
- Why??
  - The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a *weight*
- The weight → positive powers of ten

$$\ldots 10^2\ 10^1\ 10^0$$

Fractional numbers in decimal numbers

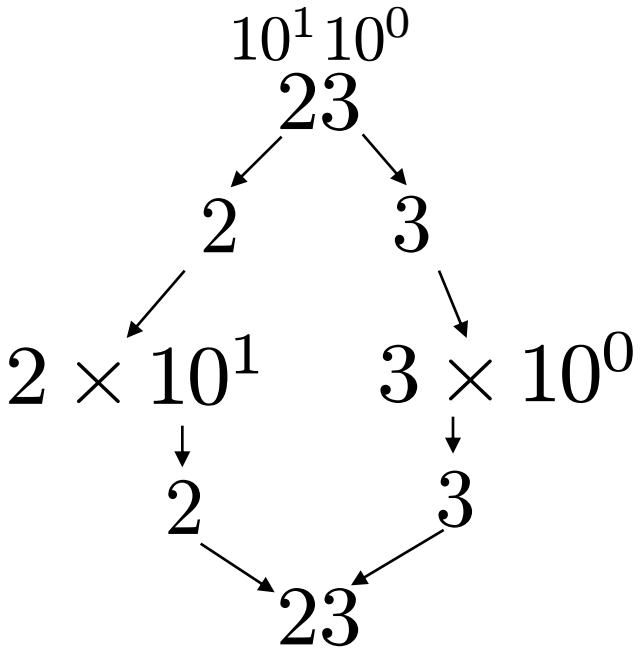
- The weights for fractional numbers in decimal numbers are negative powers of ten
  - They decrease from left to right beginning with 10^{-1}
- They are written as follows

$$10^2\ 10^1\ 10^0 . 10^{-1}\ 10^{-2}\ 10^{-3}\ \ldots$$

Decimal point

What is the meaning of weighted number system??

- As mentioned before, each number will be multiplied by its weight to get the actual number
- Example:
  - Let's consider 23 (in decimal number) or, in words twenty three



2-2 Binary Numbers

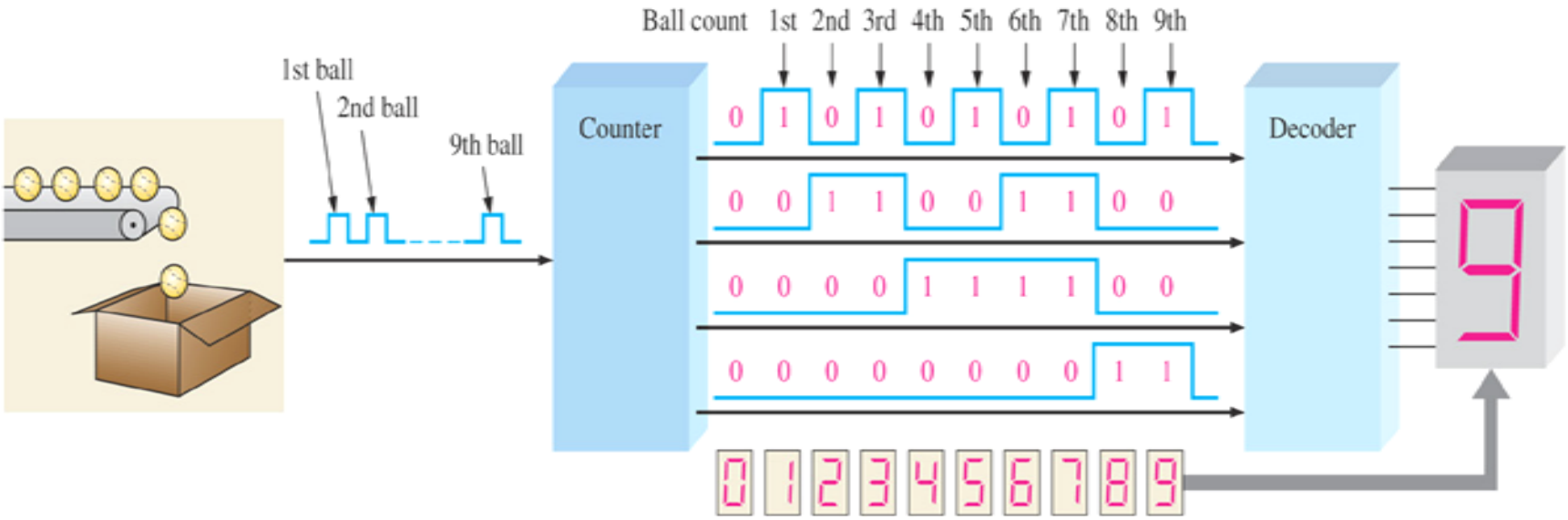
- These types of numbers have only two digits: 1 and 0 (bits)
- Therefore, it is less complicated compared to decimal numbers
- The base is two and written as

$$(0011)_2 = 3_{10}$$

- In general, with n bits we can count up to a number equal to $2^n - 1$

Application using binary numbers

- A counting system that counts tennis ball



The weighting structure of binary numbers

- Similar to decimal numbers, binary numbers are weighted number systems
- How the weights are structured?
  - Based on powers of two, which begins with 0
  - Refer to Table 2-2 for clearer picture
- Let’s look at the two examples in our text book

2-3 Decimal-to-Binary Conversion

- After learning decimal and binary numbers, we must be able to convert between these two
- Decimal-to-binary conversion methods:
  - Sum-of-weight method
    - $9 = 8 + 1 \longrightarrow 9 = 2^3 + 2^1$

This weight must be 2^0

  - Repeated division-by-2 method
    - Check page 54

Converting Decimal Fractions to Binary

- There are two methods:
  - Sum-of-weights
    - $0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$
  - Repeated multiplication by 2
    - Let's look at page 56

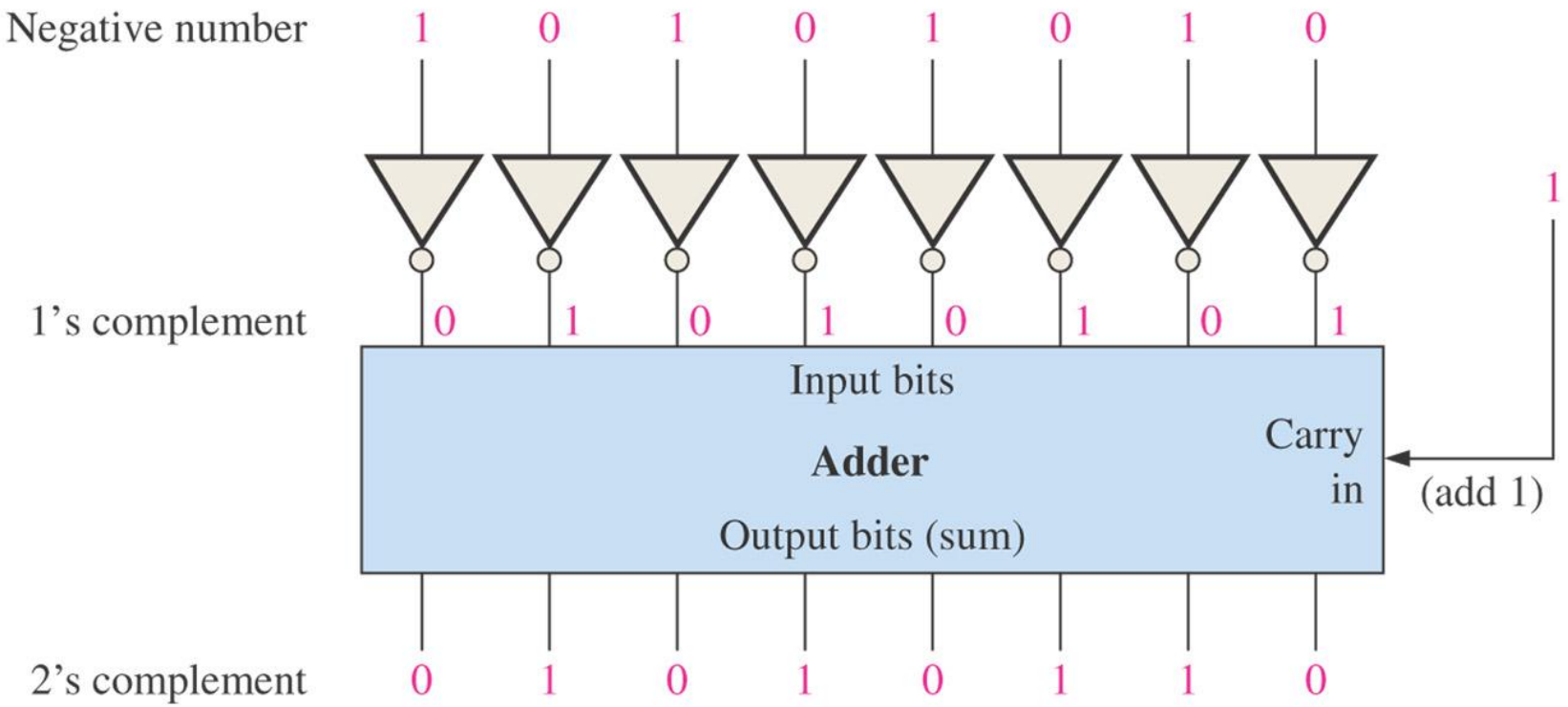
2-4 Binary Arithmetic

- There are 4 arithmetic operations using binary
  - Binary addition
  - Binary subtraction
  - Binary multiplication
  - Binary division
- All arithmetic operations follow the same procedure as decimal numbers have – carry, borrow, partial products

2-5 1's and 2's Complements of Binary Numbers

- These complements of binary numbers are important because *they permit the representation of negative numbers*
- How to find 1's complement?
  - By changing all 1s to 0s and all 0s to 1s
- How to find 2's complement?
  - By changing all 1s to 0s and all 0s to 1s, *then add 1 to the LSB*
  - Alternative method to find 2's complement
    - Start with LSB and write the bits as they are up to and including the first 1
    - Take the 1's complements of the remaining bits

Example of obtaining 2's complement of a negative binary number



Let’s do some exercises!!

- Section 2-5 Review Questions

1. Determine the 1’s complement of each binary number:

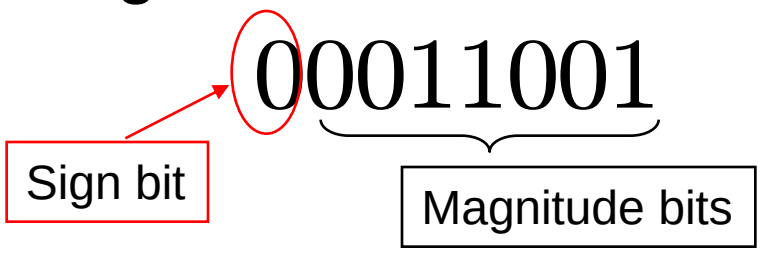
00011010	11110111	10001101
11100101	00001000	01110010

2. Determine the 2’s complement of each binary number:

00010110	11111100	10010001
11101010	00000100	01101111

2-6 Signed Numbers (1)

- The sign bit
  - The left-most bit in a signed binary number is the sign bit, which tells us whether the number is positive or negative
    - 0: positive, 1: negative
- Sign-magnitude form
  - The magnitude bits are the remaining bits after the sign-bit



Signed numbers (2)

- 1’s complement form
  - In the 1’s complement form, a negative number is the 1’s complement of the corresponding positive number
- 2’s complement form
  - In the 2’s complement form, a negative number is the 2’s complement of the corresponding positive number
- The decimal value of signed numbers
  - Sign-magnitude
  - 1’s complement
  - 2’s complement

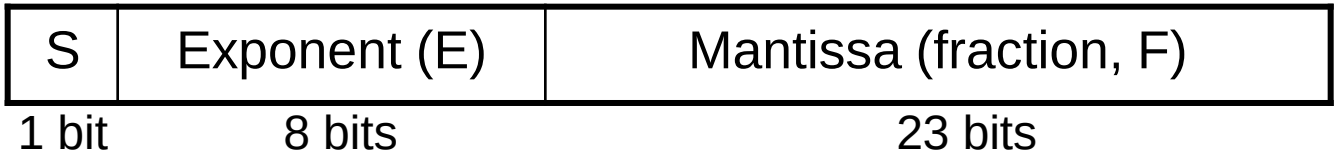
Signed Numbers (3)

- Range of Signed Integer Numbers That Can be Represented
  - 8 bit = 1 Byte
  - With one byte or eight bits, 256 different numbers can be represented. How about 16 bits and 32 bits?? More numbers can be used
  - To know the total of combinations of n bits:
    - Total combinations = 2^n
    - The range of values for n-bit numbers is

$$-(2^{n-1})\text{to}(2^{n-1}-1)$$

Signed Numbers (4)

- Floating-point numbers
  - Consists of two parts;
    - Mantissa – the part of floating-point number that represents the magnitude of the number
    - Exponent – the part of a floating-point number that represents the number of places that the decimal point (or binary) is to be moved



- Formula

$$\text{Number} = (-1)^S(1+F)(2^{E-127})$$

2-7 Arithmetic Operations with Signed Numbers

- It is important to know this for 2's complement form because this type of binary representation is widely used in computers and microprocessor-based systems
- Addition
  - Both positive number
  - Positive number with magnitude larger than negative number
  - Negative number with magnitude larger than positive number
  - Both numbers negative
- We also need to know:
  - Overflow condition – when two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers → resulting *incorrect sign bit*
  - Numbers are added two at a time – for addition, two numbers are added at one time (example 2-19)

Subtraction of signed numbers

- Subtraction is a special case of addition – change the sign of the subtrahend and adds it to the minuend
- What is subtrahend and minuend?
  - Subtrahend – the amount/quantity to be subtracted (book example: 6)
  - Minuend – the amount/quantity to be subtracted from (book example: 9)
- Big hints:
  - *To subtract two signed number, take the 2's complement of the subtrahend and add. Discard any final carry bit*

Multiplication of signed numbers

- Numbers involved in a multiplication are multiplicand, multiplier and product
- Performing multiplication using addition:
  - Direct addition and partial products
- When two binary numbers are multiplied, both numbers must be in true (uncompleted) form
- Look at example 2-22 for more details on this operation

Division of signed numbers

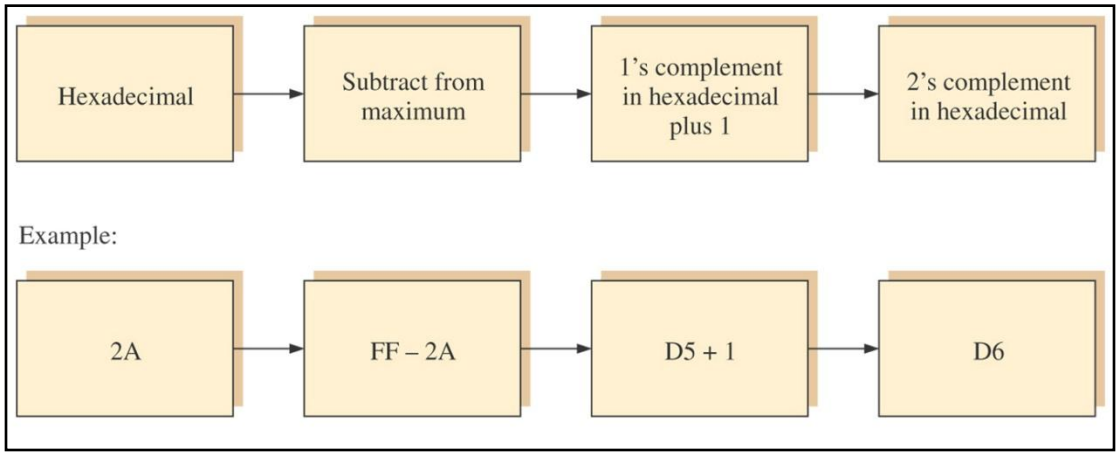
- Numbers involved in a division are dividend, divisor and quotient
- Division can also be performed using an adder – due to division is accomplished using subtraction in computers, and as division is also using adder, division can also be performed with an adder
- The operation stops when the quotient is 0

2-8 Hexadecimal Numbers(1)

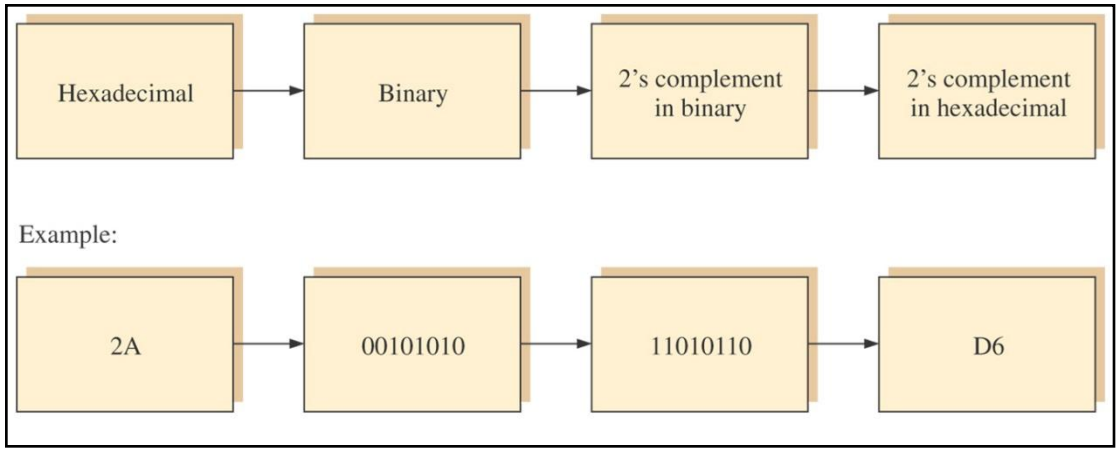
- Hexadecimal numbers are one of the most important numbers in digital systems, as “Assembly” language (machine language) uses this number to program a micro processor system
- It uses sixteen characteristics, but easy to read as each of its 4 bits are used to represent a number between 0-16
- Binary-to-hexadecimal conversion
- Hexadecimal-to-binary conversion
- Hexadecimal-to-decimal conversion
  - Convert to binary and followed by decimal
- Decimal-to-hexadecimal conversion

2-8 Hexadecimal Numbers(2)

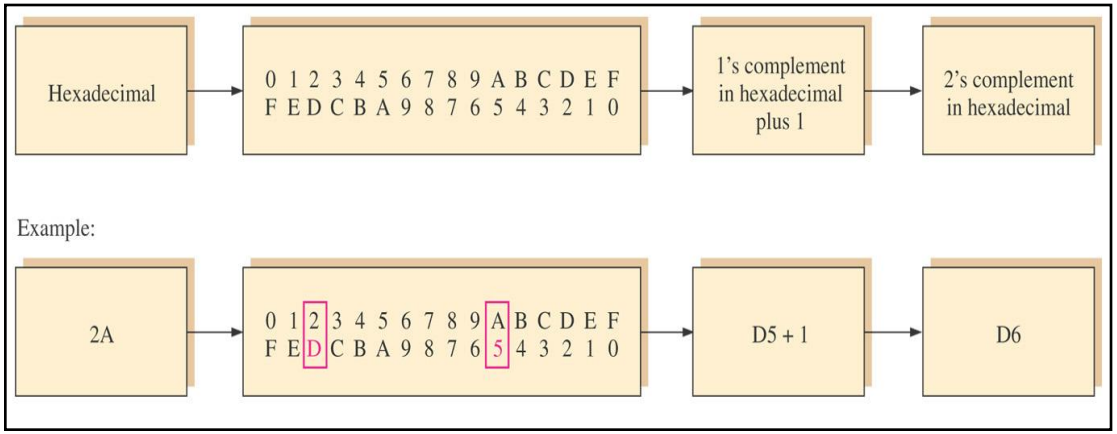
- Hexadecimal addition
  - Use common addition method – less than 15, write the answer in hexadecimal number, and if >15, carry one to the next bit
- Hexadecimal subtraction
  - There are three ways can be used (see next slide)
  - They are all 2’s complement



Method 1



Method 2



Method 3

2-9 Octal Numbers

- This number is composed of eight digits; 0-7
- Octal-to-decimal conversion
  - Similar to others
- Decimal-to-octal conversion
  - Using this following technique
- Binary-to-octal conversion
  - Quite similar with binary-to-hexadecimal conversion
  - If there are not enough bit (for the most left bit), add 1 or 2 zeros as 0s will never affect the binary numbers

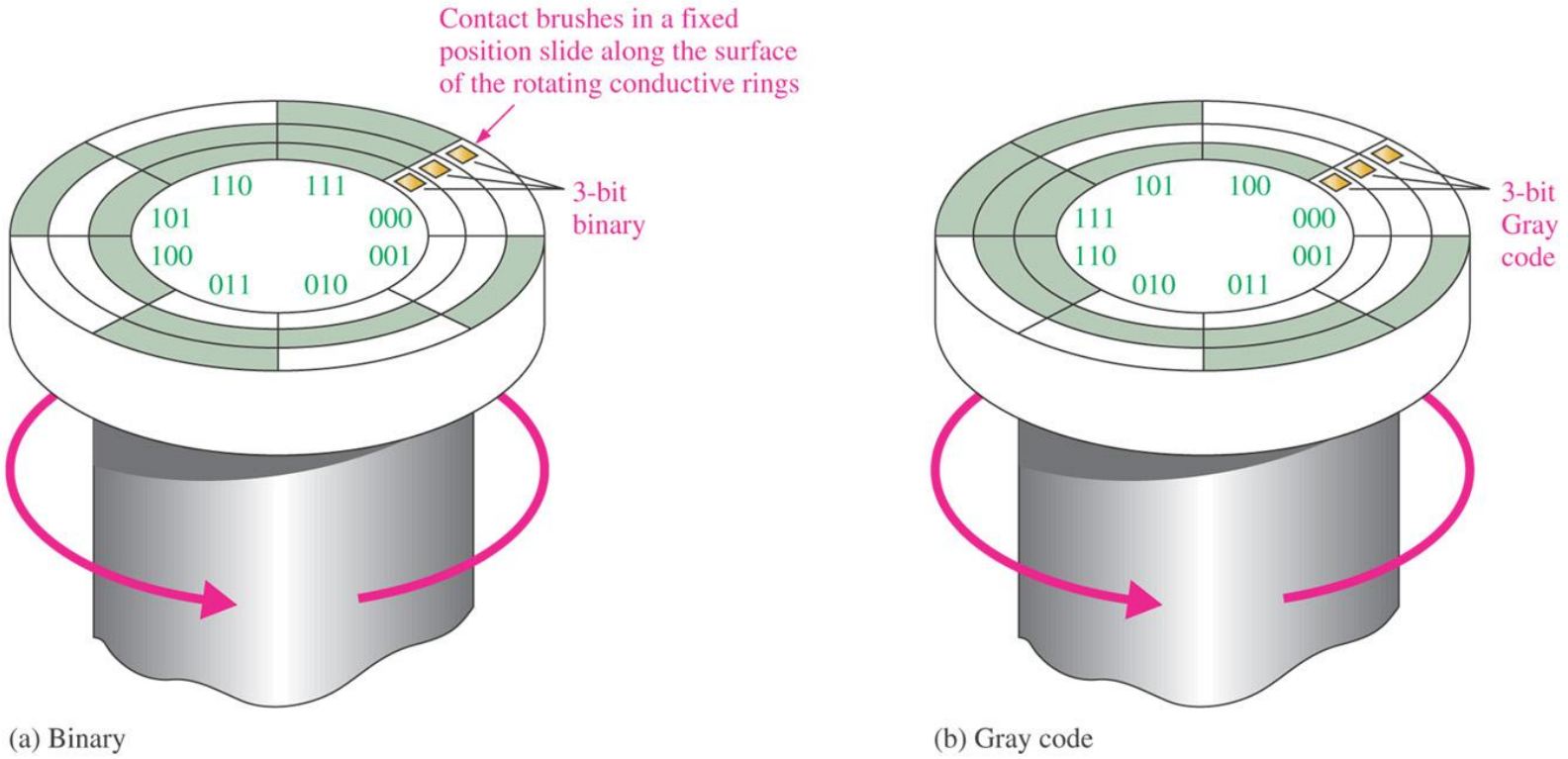
2-10 Binary Coded Decimal (BCD)

- BCD is a way to express each of the decimal digits with a binary code
  - There are only 10 groups in BCD system
- 8421 Code
  - This is a type of BCD code which indicates the binary weights
  - BCD ←→8421 code
- Invalid codes – these are the codes that are not used in BCD (remember that BCD is a 10 groups number AND NOT 16), so A to F are not included
- BCD addition – carefully do this as there are ***valid*** and ***invalid*** answers
  - For invalid answers, just add 6 to them

2-11 Digital Codes

- In digital systems, there are various types of codes and expressed in terms of numbers, characters, alphanumeric, etc.
- Famous codes are Gray code and ASCII
- Gray code:
  - This code is unweighted and is not an arithmetic code
  - Specially used in shaft position encoders
  - Binary-to-Gray code conversion
    - MSB Gray code = MSB Binary code
    - From left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries
  - Gray-to-binary code conversion
    - MSB binary code = MSB Gray code
    - Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries

Gray code application



- Problems arise if binary code is used in this application when 111→000 (counterclockwise) ➔3 bits change
- This problem might be solved if Gray code is used due to only one bit might cause problem

Alphanumeric

- Alphanumeric codes are codes that represent numbers and alphabetic characters (letters)
- Consists of 10 decimal digits and 26 letters of the alphabet
- Bits required is 6 bits → in binary then we need more than $2^5 = 32$ which is $2^6 = 64$
  - The remaining 28 bits are used for other purposes like periods, colons, semicolons, etc.

ASCII

- Stands for “American Standard Code for Information Interchange”
- Has 128 characters and symbols
  - Represented in 7-bit binary code
  - First 32 ASCII characters are used for control purposes, like “null”, “line fee”, etc.
- Extended ASCII characters
  - Used for other than English language (additional of 128 characters)
  - Adopted by IBM to be used in PCs

2-12 Error Detection and Correction Codes

- We can detect a single bit error, or detect and correct a single bit error
- How to do this??
  - By performing parity check
- Parity method for error detection
  - Parity bit is used in many systems as a means for bit error detection
  - Attached at the beginning or end of the code
  - Parity, which is odd or even, is assigned to a group of bits for error detection purpose
    - Note here that parity is either odd or even
    - Odd – total number of 1s is odd, and vice versa to even
  - Detecting an error
    - By using odd or even information. Let's try it

Parity method for error detection

EVEN PARITY		ODD PARITY	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

TABLE 2–10

The BCD code with parity bits.

The Hamming Error Correction Code

- Hamming code provides for single-error correction
  - It provides more information so that correction can also be done besides detection
- Number of parity bits
  - We need to decide how many parity bits to be used to detect and correct an error
  - Check Eq. 2-1

$$2^p \geq d + p + 1$$

- Detecting and correcting an error with the Hamming code
  - Let’s try to look at the examples and related problems for more intuitive explanations

EXAMPLE 2–41

Determine the Hamming code for the BCD number 1001 (data bits), using even parity.

Solution

Step 1: Find the number of parity bits required. Let $p = 3$. Then

$$2^p = 2^3 = 8$$
$$d + p + 1 = 4 + 3 + 1 = 8$$

Three parity bits are sufficient.

$$\text{Total code bits} = 4 + 3 = 7$$

Step 2: Construct a bit position table, as shown in Table 2–12, and enter the data bits. Parity bits are determined in the following steps.

TABLE 2–12

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Data bits			1		0	0	1
Parity bits	0	0		1			

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an even number of 1s (2) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an even number of 1s (2) in this group.

Step 4: These parity bits are entered in Table 2–12, and the resulting combined code is 0011001.

Related Problem Determine the Hamming code for the BCD number 1000 using even parity.

EXAMPLE 2–42

Determine the Hamming code for the data bits 10110 using odd parity.

Solution

Step 1: Determine the number of parity bits required. In this case the number of data bits, d, is five. From the previous example we know that $p = 3$ will not work. Try $p = 4$:

$$2^p = 2^4 = 16$$

$$d + p + 1 = 5 + 4 + 1 = 10$$

Four parity bits are sufficient.

$$\text{Total code bits} = 5 + 4 = 9$$

Step 2: Construct a bit position table, Table 2–13, and enter the data bits. Parity bits are determined in the following steps. Notice that P_4 is in bit position 8.

TABLE 2–13

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Data bits			1		0	1	1		0
Parity bits	1	0		1				1	

Step 3: Determine the parity bits as follows:

Bit P_1 checks bit positions 1, 3, 5, 7, and 9 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_2 checks bit positions 2, 3, 6, and 7 and must be a 0 for there to be an odd number of 1s (3) in this group.

Bit P_3 checks bit positions 4, 5, 6, and 7 and must be a 1 for there to be an odd number of 1s (3) in this group.

Bit P_4 checks bit positions 8 and 9 and must be a 1 for there to be an odd number of 1s (1) in this group.

Step 4: These parity bits are entered in the Table 2–13, and the resulting combined code is 101101110.

Related Problem Determine the Hamming code for 11001 using odd parity.

Detecting and Correcting an Error with the Hamming Code

- Let’s look at the example

EXAMPLE 2–43

Assume that the code word in Example 2–41 (0011001) is transmitted and that 0010001 is received. The receiver does not "know" what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even parity is used.

Solution First, make a bit position table, as indicated in Table 2–14.

TABLE 2–14

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4
BIT POSITION	1	2	3	4	5	6	7
BINARY POSITION NUMBER	001	010	011	100	101	110	111
Received code	0	0	1	0	0	0	1

First parity check:

Bit P_1 checks positions 1, 3, 5, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0 (LSB)

Second parity check:

Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is good. ⟶ 0

Third parity check:

Bit P_3 checks positions 4, 5, 6, and 7.
There is one 1 in this group.
Parity check is bad. ⟶ 1 (MSB)

Result:

The error position code is 100 (binary four). This says that the bit in position 4 is in error. It is a 0 and should be a 1. The corrected code is 0011001, which agrees with the transmitted code.

Related Problem Repeat the process illustrated in the example if the received code is 0111001.

EXAMPLE 2–44

The code 101101010 is received. Correct any errors. There are four parity bits, and odd parity is used.

Solution First, make a bit position table like Table 2–15.

TABLE 2–15

BIT DESIGNATION	P_1	P_2	D_1	P_3	D_2	D_3	D_4	P_4	D_5
BIT POSITION	1	2	3	4	5	6	7	8	9
BINARY POSITION NUMBER	0001	0010	0011	0100	0101	0110	0111	1000	1001
Received code	1	0	1	1	0	1	0	1	0

First parity check:

Bit P_1 checks positions 1, 3, 5, 7, and 9.
There are two 1s in this group.
Parity check is bad. ⟶ 1 (LSB)

Second parity check:

Bit P_2 checks positions 2, 3, 6, and 7.
There are two 1s in this group.
Parity check is bad. ⟶ 1

Third parity check:

Bit P_3 checks positions 4, 5, 6, and 7.
There are two 1s in this group.
Parity check is bad. ⟶ 1

Fourth parity check:

Bit P_4 checks positions 8 and 9.
There is one 1 in this group.
Parity check is good. ⟶ 0 (MSB)

Result:

The error position code is 0111 (binary seven). This says that the bit in position 7 is in error. The corrected code is therefore 101101110.

Related Problem The code 101111001 is received. Correct any error if odd parity is used.

That’s all for this chapter

- Try to solve questions given at the end of each section