Chapter 2: Number Systems and Codes

Lesson 2.3: Codes

Computer Fundamentals

Second Edition

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

On completion of this lesson you will know:

- Basic concepts of data information and codes
- Representation of numeric data using binary numbers,
- About BCD, EBCDIC, ASCII Codes
- About Unicode

- Data are the names given to basic facts such as names and numbers. Unit-price is an examples of data.
- Information is processed data, that is, information is data which have been converted to a more useful form. For example total price = unit price quantity sold. Here total price is information and unit price and quantity sold are data.
- Codes are used to reduce the volume of data. By coding, recording of data can be made sunoke laborious, less prone to error and the data become more manageable and easier to manipulate.

- Data are the names given to basic facts such as names and numbers. Unit-price is an examples of data.
- Information is processed data, that is, information is data which have been converted to a more useful form. For example total price = unit price quantity sold. Here total price is information and unit price and quantity sold are data.
- Codes are used to reduce the volume of data. By coding, recording of data can be made sunoke laborious, less prone to error and the data become more manageable and easier to manipulate.

- Data are the names given to basic facts such as names and numbers. Unit-price is an examples of data.
- Information is processed data, that is, information is data which have been converted to a more useful form. For example total price = unit price quantity sold. Here total price is information and unit price and quantity sold are data.
- Codes are used to reduce the volume of data. By coding, recording of data can be made sunoke laborious, less prone to error and the data become more manageable and easier to manipulate.

- Data are the names given to basic facts such as names and numbers. Unit-price is an examples of data.
- Information is processed data, that is, information is data which have been converted to a more useful form. For example total price = unit price quantity sold. Here total price is information and unit price and quantity sold are data.
- Codes are used to reduce the volume of data. By coding, recording of data can be made sunoke laborious, less prone to error and the data become more manageable and easier to manipulate.

- Numeric data are represented in the computer using binary numbers. So anything which has to be stored in the memory must be converted into a binary form and then the bits can be used.
  - To store any positive integers the location is being filled with bits from right to left. The extra bits are filled up with 0s. To store 423_{10} in a memory location of word length of 16 bits, first

convert 423_{10} into 110100111_2 and fill the extra bits with 0s.

0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1

- Numeric data are represented in the computer using binary numbers. So anything which has to be stored in the memory must be converted into a binary form and then the bits can be used.
  - To store any positive integers the location is being filled with bits from right to left. The extra bits are filled up with 0s. To store 423_{10} in a memory location of word length of 16 bits, first convert 423_{10} into 110100111_2 and fill the extra bits with 0s.

0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1

- Numeric data are represented in the computer using binary numbers. So anything which has to be stored in the memory must be converted into a binary form and then the bits can be used.
  - To store any positive integers the location is being filled with bits from right to left. The extra bits are filled up with 0s. To store 423_{10} in a memory location of word length of 16 bits, first

convert 423_{10} into 110100111_2 and fill the extra bits with 0s.

0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1

- Numeric data are represented in the computer using binary numbers. So anything which has to be stored in the memory must be converted into a binary form and then the bits can be used.
  - To store any positive integers the location is being filled with bits from right to left. The extra bits are filled up with 0s. To store 423_{10} in a memory location of word length of 16 bits, first

convert 423_{10} into 110100111_2 and fill the extra bits with 0s.

0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1

- The left most bit in the word represents the sign bit. If the left most bit is 0, the number stored in the word will be treated as a positive integer. On other hand if the left most bit is 1, then the number in the word will be treated as a negative integer. That bit is called the sign bit.
- For example, storing +50 in a 16-bit word is as follows

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- For example, storing -50 in a 16-bit word is as follows

1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- In the sign magnitue representation, the largest (i.e., positive) integer that can be stored in a 16-bit memory location is $2^{15} - 1$. Similarly the smallest (i.e., negative) integer that can be stored in a 16-bit memory location is -2^{15}.

- The left most bit in the word represents the sign bit. If the left most bit is 0, the number stored in the word will be treated as a positive integer. On other hand if the left most bit is 1, then the number in the word will be treated as a negative integer. That bit is called the sign bit.
- For example, storing +50 in a 16-bit word is as follows

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- For example, storing -50 in a 16-bit word is as follows

1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- In the sign magnitue representation, the largest (i.e., positive) integer that can be stored in a 16-bit memory location is $2^{15} - 1$. Similarly the smallest (i.e., negative) integer that can be stored in a 16-bit memory location is -2^{15}.

- The left most bit in the word represents the sign bit. If the left most bit is 0, the number stored in the word will be treated as a positive integer. On other hand if the left most bit is 1, then the number in the word will be treated as a negative integer. That bit is called the sign bit.
- For example, storing +50 in a 16-bit word is as follows

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- For example, storing -50 in a 16-bit word is as follows

1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- In the sign magnitue representation, the largest (i.e., positive) integer that can be stored in a 16-bit memory location is $2^{15} - 1$. Similarly the smallest (i.e., negative) integer that can be stored in a 16-bit memory location is -2^{15}.

- The left most bit in the word represents the sign bit. If the left most bit is 0, the number stored in the word will be treated as a positive integer. On other hand if the left most bit is 1, then the number in the word will be treated as a negative integer. That bit is called the sign bit.
- For example, storing +50 in a 16-bit word is as follows

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- For example, storing -50 in a 16-bit word is as follows

1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

- In the sign magnitue representation, the largest (i.e., positive) integer that can be stored in a 16-bit memory location is $2^{15} - 1$. Similarly the smallest (i.e., negative) integer that can be stored in a 16-bit memory location is -2^{15}.

- Positive number is identical to that used in the sign magnitude system.
- It consists of a left most sign bit in the leftmost position, followed by the magnitude bits.
- For a positive number, two representations are identical.
- In case of negative numbers, the magnitude bits are represented by their 1's complement which is obtained by inverting each bit including the sign bit.
- For example, the 1's complement of $25_{10} = 11001_2$ is 00110_2. And the 1's complement of -5_{10} is 1010_2.

- Positive number is identical to that used in the sign magnitude system.
- It consists of a left most sign bit in the leftmost position, followed by the magnitude bits.
- For a positive number, two representations are identical.
- In case of negative numbers, the magnitude bits are represented by their 1's complement which is obtained by inverting each bit including the sign bit.
- For example, the 1's complement of $25_{10} = 11001_2$ is 00110_2. And the 1's complement of -5_{10} is 1010_2.

- Positive number is identical to that used in the sign magnitude system.
- It consists of a left most sign bit in the leftmost position, followed by the magnitude bits.
- For a positive number, two representations are identical.
- In case of negative numbers, the magnitude bits are represented by their 1's complement which is obtained by inverting each bit including the sign bit.
- For example, the 1's complement of $25_{10} = 11001_2$ is 00110_2. And the 1's complement of -5_{10} is 1010_2.

- Positive number is identical to that used in the sign magnitude system.
- It consists of a left most sign bit in the leftmost position, followed by the magnitude bits.
- For a positive number, two representations are identical.
- In case of negative numbers, the magnitude bits are represented by their 1's complement which is obtained by inverting each bit including the sign bit.
- For example, the 1's complement of $25_{10} = 11001_2$ is 00110_2. And the 1's complement of -5_{10} is 1010_2.

- Positive number is identical to that used in the sign magnitude system.
- It consists of a left most sign bit in the leftmost position, followed by the magnitude bits.
- For a positive number, two representations are identical.
- In case of negative numbers, the magnitude bits are represented by their 1's complement which is obtained by inverting each bit including the sign bit.
- For example, the 1's complement of $25_{10} = 11001_2$ is 00110_2. And the 1's complement of -5_{10} is 1010_2.

- 2's complement representation of a - ve number is obtained by adding a 1 to the 1's complement of that number.
  Thus the 2's complement of −50 is

1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0

- 2's complement representation of a - ve number is obtained by adding a 1 to the 1's complement of that number.
  Thus the 2's complement of –50 is

1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0

Fixed point representation

There are two most commonly used representations:

- The position of decimal point is fixed which is assumed to be after the first 8 bits in this representation.
- The first 8 bits store the integer portion of the number and the last 8 bits store the fractional part.
- The bits in the integer part are filled from right to left and the remaining bits are padded with 0's. The left most bit is reserved for the sign bit.The representation of -101001011.001001_2 in the 16 bit word is as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Fixed point representation

There are two most commonly used representations:

- The position of decimal point is fixed which is assumed to be after the first 8 bits in this representation.
- The first 8 bits store the integer portion of the number and the last 8 bits store the fractional part.
- The bits in the integer part are filled from right to left and the remaining bits are padded with 0's. The left most bit is reserved for the sign bit.The representation of -101001011.001001_2 in the 16 bit word is as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Fixed point representation

There are two most commonly used representations:

- The position of decimal point is fixed which is assumed to be after the first 8 bits in this representation.
- The first 8 bits store the integer portion of the number and the last 8 bits store the fractional part.
- The bits in the integer part are filled from right to left and the remaining bits are padded with 0's. The left most bit is reserved for the sign bit.The representation of -101001011.001001_2 in the 16 bit word is as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Fixed point representation

There are two most commonly used representations:

- The position of decimal point is fixed which is assumed to be after the first 8 bits in this representation.
- The first 8 bits store the integer portion of the number and the last 8 bits store the fractional part.
- The bits in the integer part are filled from right to left and the remaining bits are padded with 0's. The left most bit is reserved for the sign bit.The representation of -101001011.001001_2 in the 16 bit word is as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Fixed point representation

There are two most commonly used representations:

- The position of decimal point is fixed which is assumed to be after the first 8 bits in this representation.
- The first 8 bits store the integer portion of the number and the last 8 bits store the fractional part.
- The bits in the integer part are filled from right to left and the remaining bits are padded with 0's. The left most bit is reserved for the sign bit.The representation of -101001011.001001_2 in the 16 bit word is as follows :

1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	1

Floating point representation

There are two most commonly used representations:

- It can be used to represent numbers in any positional number system.
- It is written in the format mb^e, where m is the mantissa, which is a signed fixed point number, e is the exponent, which is a signed integer and determines the actual position of the decimal point, b is the base of the number.

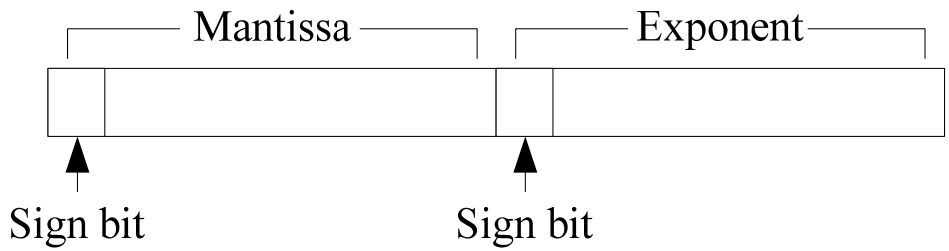

Figure: 2.3.2: Floating point representation

Floating point representation

There are two most commonly used representations:

- It can be used to represent numbers in any positional number system.
- It is written in the format mb^e, where m is the mantissa, which is a signed fixed point number, e is the exponent, which is a signed integer and determines the actual position of the decimal point, b is the base of the number.

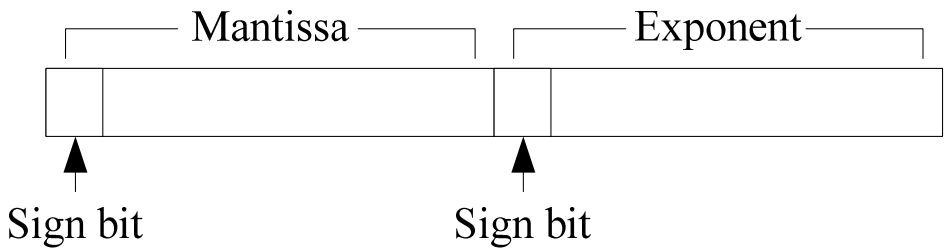

Figure: 2.3.2: Floating point representation

Floating point representation

There are two most commonly used representations:

- It can be used to represent numbers in any positional number system.
- It is written in the format mb^e, where m is the mantissa, which is a signed fixed point number, e is the exponent, which is a signed integer and determines the actual position of the decimal point, b is the base of the number.

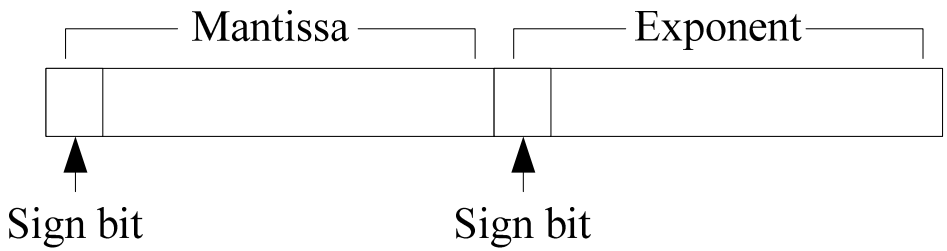

Figure: 2.3.2: Floating point representation

Floating point representation

There are two most commonly used representations:

- It can be used to represent numbers in any positional number system.
- It is written in the format mb^e, where m is the mantissa, which is a signed fixed point number, e is the exponent, which is a signed integer and determines the actual position of the decimal point, b is the base of the number.

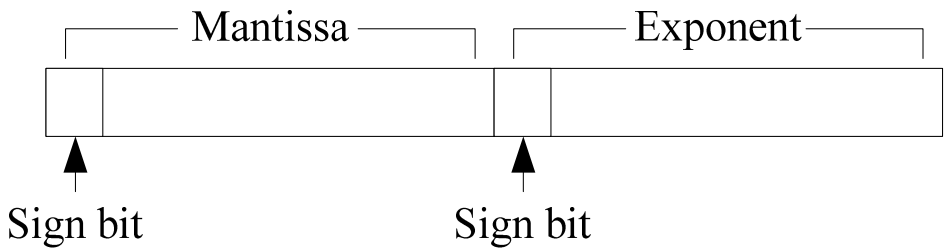

Figure: 2.3.2: Floating point representation

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- In the binary coded decimal (BCD) each decimal digit is represented by 4 bits. When only 4 bits are used a total of 16 configurations are possible.
- The 6 bits are used to represent each character (the four BCD numeric positions and two additional zone positions).
- When only 6 bits are used a total of 64 configurations are possible which means 64 different characters can be represented.
- For example,

Character	Zone	Digit	Octal
A	11	0001	61

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- EBCDIC stands for extended binary coded decimal interchange code.
- In EBCDIC format, 8 bits are used to represent each character.
- In this code, it is possible to represent 256($= 2^8$) different characters. It can be easily divided into two 4 bit groups.
- Each of these 4 bit groups can be represented by 1 hexadecimal digit.
- For example

Character	Zone	Digit	Hexadecimal
A	1100	0001	C1

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows 128($= 2^7$) different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows 256($= 2^8$) different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows $128 (= 2^7)$ different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows $256 (= 2^8)$ different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows 128($= 2^7$) different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows 256($= 2^8$) different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows $128(=2^7)$ different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows $256(=2^8)$ different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows 128($= 2^7$) different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows 256($= 2^8$) different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- American Standard Code for Information Interchange (ASCII) is a very widely used computer code. There are of two types: ASCII-7 and ASCII-8.
- ASCII-7 is a 7 bit code that allows 128($= 2^7$) different characters. The first 3-bits are used as zone bits and the last 4-bits indicate the digit.
- Microcomputers using 8-bits byte use the 7-bits ASCII by leaving the leftmost bit of each byte as a zero. ASCII-8 is 8-bits code that allows 256($= 2^8$) different characters.
- For example

Character	Zone	Digit	Hexadecimal
A	100	0001	41

- Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.
- It provides a unique number for every character as shown in Figure 2.3.3, no matter what the platform, no matter what the program, no matter what the language.

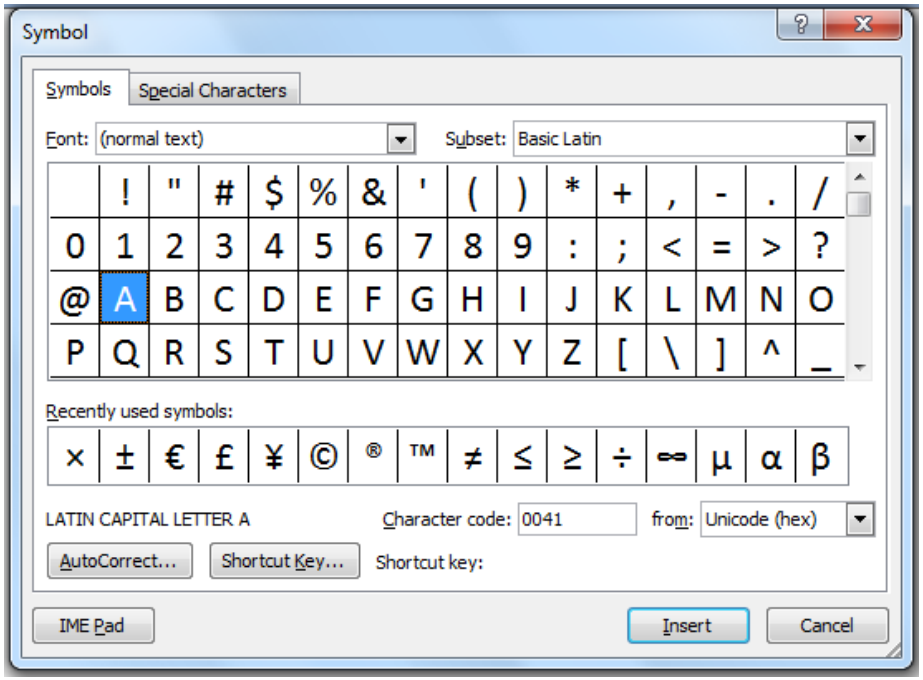

Figure: 2.3.3: Unicode

- Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.
- It provides a unique number for every character as shown in Figure 2.3.3, no matter what the platform, no matter what the program, no matter what the language.

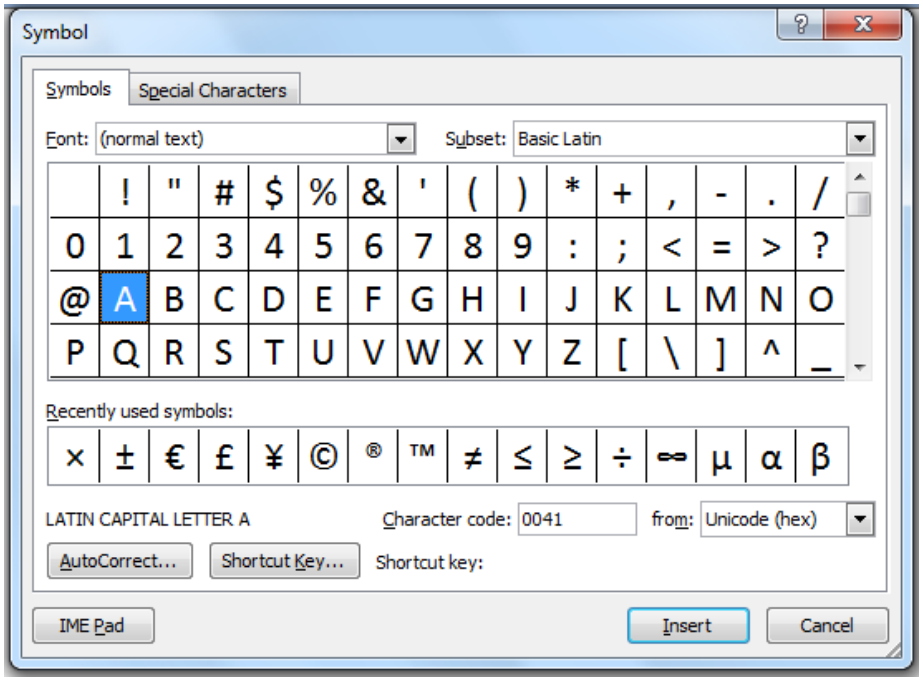

Figure: 2.3.3: Unicode

- Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.
- It provides a unique number for every character as shown in Figure 2.3.3, no matter what the platform, no matter what the program, no matter what the language.

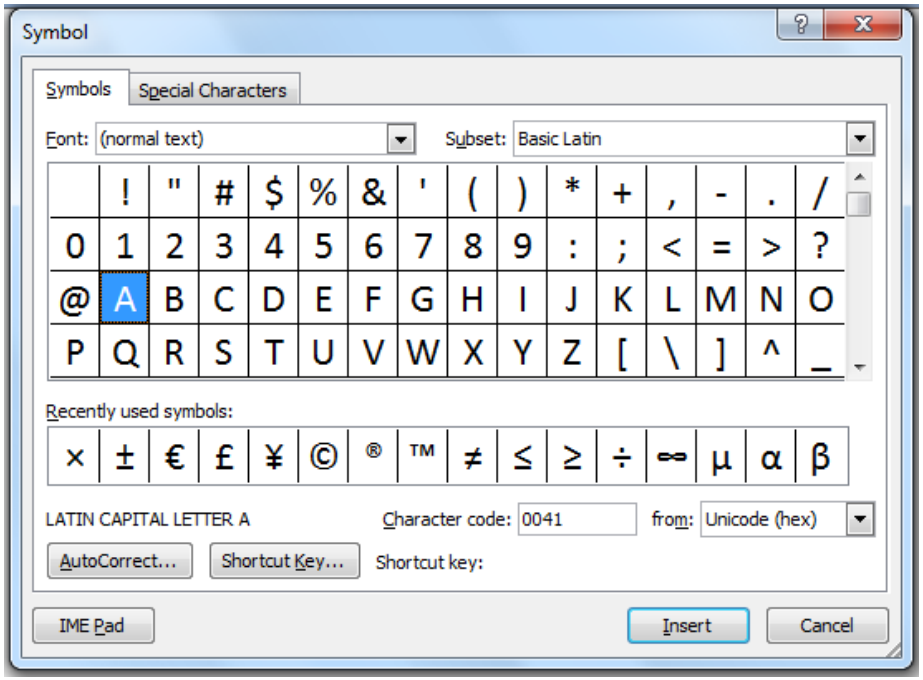

Figure: 2.3.3: Unicode

- Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.
- It provides a unique number for every character as shown in Figure 2.3.3, no matter what the platform, no matter what the program, no matter what the language.

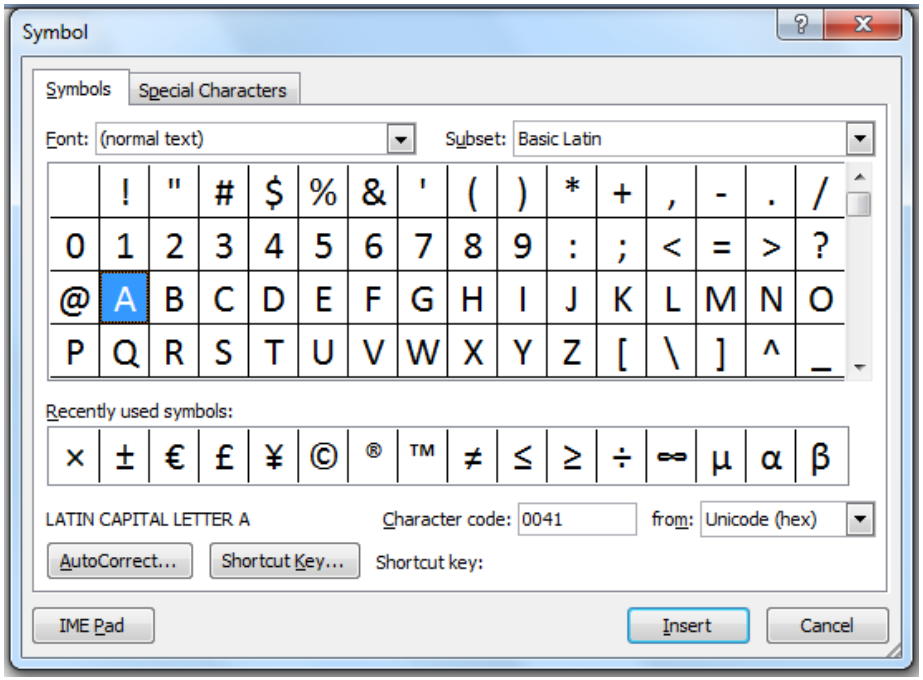

Figure: 2.3.3: Unicode

- The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.
- Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is a bit of overkill for English and Western-European languages, but it is necessary for some other languages, such as Greek, Chinese and Japanese.
- Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.
- The most recent Unicode version as of 2012 is Unicode 6.1.

- The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.
- Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is a bit of overkill for English and Western-European languages, but it is necessary for some other languages, such as Greek, Chinese and Japanese.
- Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.
- The most recent Unicode version as of 2012 is Unicode 6.1.

- The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.
- Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is a bit of overkill for English and Western-European languages, but it is necessary for some other languages, such as Greek, Chinese and Japanese.
- Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.
- The most recent Unicode version as of 2012 is Unicode 6.1.

- The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.
- Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is a bit of overkill for English and Western-European languages, but it is necessary for some other languages, such as Greek, Chinese and Japanese.
- Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.
- The most recent Unicode version as of 2012 is Unicode 6.1.

- The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.
- Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is a bit of overkill for English and Western-European languages, but it is necessary for some other languages, such as Greek, Chinese and Japanese.
- Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.
- The most recent Unicode version as of 2012 is Unicode 6.1.

- The different encodings implement Unicode.
- The most commonly used encodings are UTF-8 (or UCS Transformation Format-8), UTF-16 and UCS-2. UCS-2 uses a 16-bit code unit (two bytes) for each character but cannot encode every character in the current Unicode standard and it is now obsolete.
- UTF-8 uses one byte for any ASCII characters, and up to four bytes for other characters. UTF-16 extends UCS-2, using two 16-bit units to handle each of the additional characters.

- The different encodings implement Unicode.
- The most commonly used encodings are UTF-8 (or UCS Transformation Format-8), UTF-16 and UCS-2. UCS-2 uses a 16-bit code unit (two bytes) for each character but cannot encode every character in the current Unicode standard and it is now obsolete.
- UTF-8 uses one byte for any ASCII characters, and up to four bytes for other characters. UTF-16 extends UCS-2, using two 16-bit units to handle each of the additional characters.

- The different encodings implement Unicode.
- The most commonly used encodings are UTF-8 (or UCS Transformation Format-8), UTF-16 and UCS-2. UCS-2 uses a 16-bit code unit (two bytes) for each character but cannot encode every character in the current Unicode standard and it is now obsolete.
- UTF-8 uses one byte for any ASCII characters, and up to four bytes for other characters. UTF-16 extends UCS-2, using two 16-bit units to handle each of the additional characters.

- The different encodings implement Unicode.
- The most commonly used encodings are UTF-8 (or UCS Transformation Format-8), UTF-16 and UCS-2. UCS-2 uses a 16-bit code unit (two bytes) for each character but cannot encode every character in the current Unicode standard and it is now obsolete.
- UTF-8 uses one byte for any ASCII characters, and up to four bytes for other characters. UTF-16 extends UCS-2, using two 16-bit units to handle each of the additional characters.