Chapter 1
Digital Systems and Information

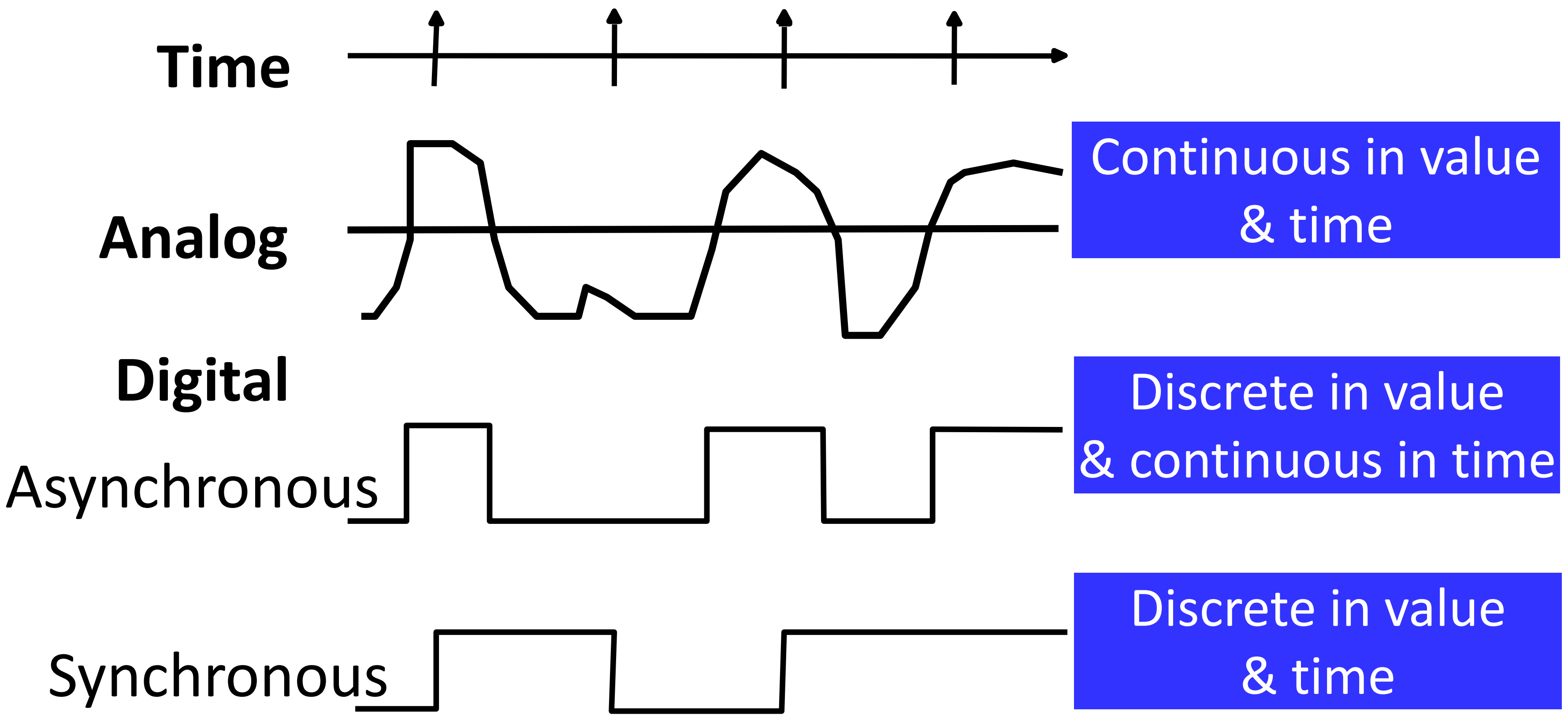
M. Morris Mano, Charles R. Kime. (2015). *Logic and computer design fundamentals* (5th ed.). Pearson.

Contents

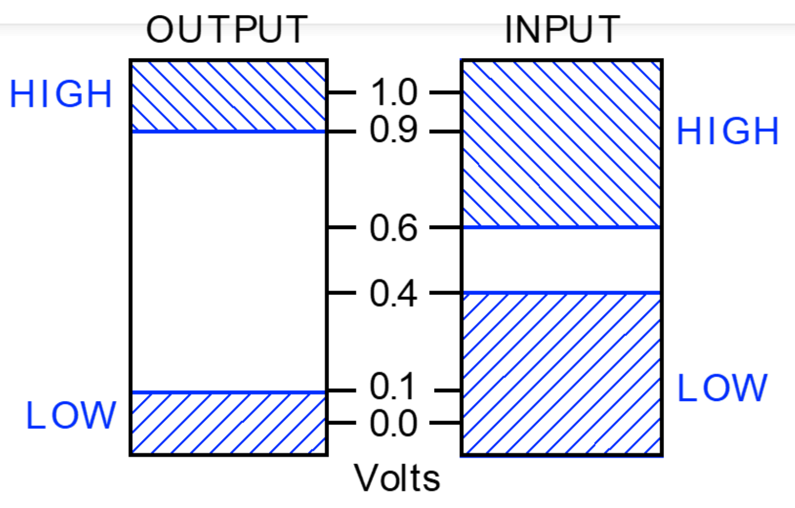
1. Information Representation

- Information represents either physical or man-made phenomena
  - Physical quantities such as weight, pressure, and velocity are mostly continuous (may take any value)
  - Man-made variables can be discrete such as business records, alphabet, integers, and unit of currencies.
- Discrete values can be represented using multiple discrete signals.
  - Two level, or binary signals are the most prevalent values in digital systems.

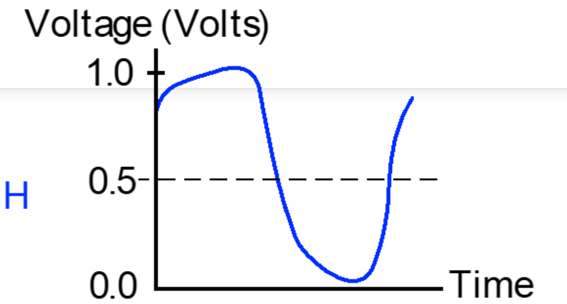
Signal Examples Over Time


Time
Analog
Digital
Asynchronous
Synchronous
Continuous in value & time
Discrete in value & continuous in time
Discrete in value & time

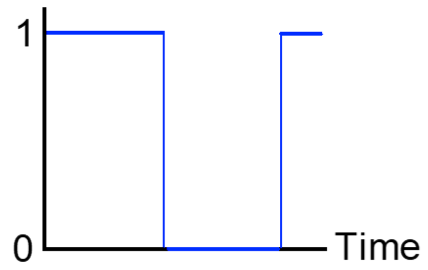
Signal Example



(a) Example voltage ranges



(b) Time-dependent Voltage



(c) Binary model of time-dependent voltage

- The two discrete values are defined by ranges of voltages called LOW and HIGH
- Binary values are represented abstractly by:
  - Digits 0 and 1, words (symbols) False (F) and True (T), words (symbols) Low (L) and High (H), and words On and Off (positive logic)

2. Abstraction Layers in Computer System Design

- Computer systems design is typically performed in a "top down" approach
  - The system is specified at a high level and then the design is decomposed into successively smaller blocks until a block is simple enough that it can be implemented.
  - These blocks are then connected together to make the full system.

2. Abstraction Layers in Computer System Design

Algorithms
Programming Languages
Operating Systems
Instruction Set Architecture
Microarchitecture
Register Transfers
Logic Gates
Transistor Circuits

Typical Layers of Abstraction in Modern Computer Systems

- An important feature of abstraction is that lower layers of abstraction can usually be modified without changing the layers above them.

- Algorithms describe a series of steps that lead to a solution.
- These algorithms are then implemented as a program in a high level programming language such as C++, Python, or Java.
- When the program is running, it shares computing resources with other programs under the control of an operating system.
- Both the operating system and the program are composed of sequences of instructions that are particular to the processor running them; the set of instructions and the registers.
- The processor hardware is a particular implementation of the instruction set architecture, referred to as the microarchitecture; manufacturers very often make several different microarchitectures that execute the same instruction set.
- A microarchitecture can be described as underlying sequences of transfers of data between registers.
- These register transfers can be decomposed into logic operations on sets of bits performed by logic gates.
- Logic gates are electronic circuits implemented with transistors or other physical devices that control the flow of electrons.

3. Number Systems

- Positive radix, positional number system
- A number with *radix r* is represented by a string of digits:

$$\boldsymbol{A_{n-1}A_{n-2}\ldots A_1A_0 . A_{-1}\,A_{-2}\ldots A_{-m+1}\,A_{-m}}$$

  - A_i: Digits, $0 \leq A_i < r$
  - The dot **.** is the *radix point*.
  - A_{n-1} is referred to as the *most significant digit* (MSD) and A_{-m} as the *least significant digit* (LSD) of the number.
- Digits available for any radix *r* system: 0, 1, 2, …, *r*-1
- The string of digits represents the power series:

$$\underbrace{\boxed{A_{n-1}\,r^{n-1}+A_{n-2}\,r^{n-2}+\ldots A_1\,r^1+A_0\,r^0}}_{\text{Integer portion}}+\underbrace{\boxed{A_{-1}\,r^{-1}+A_{-2}r^{-2}+\ldots A_{-m+1}\,r^{-m+1}+A_{-m}\,r^{-m}}}_{\text{Fraction portion}}$$

3. Number Systems

- The following illustrates a base 5 number with *n* = 3 and *m* = 1 and its conversion to decimal:
  - $(312.4)_5 = 3 * 5^2 + 1 * 5^1 + 2 * 5^0 + 4 * 5^{-1}$

    $= 75 + 5 + 2 + 0.8 = (82.8)_{10}$

Binary Numbers

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

- The binary number system is a base 2 system with two digits: 0 and 1. The digits in a binary number are called bits.

$(11010)_2 = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = (26)_{10}$

$(110101.11)_2 = 1*2^5 + 1*2^4+0*2^3+1*2^2+0*2^1+1*2^0+1*2^{-1}+1*2^{-2}$

$= 32 + 16 + 4 + 1 + 0.5 + 0.25 = (53.75)_{10}$

Special Powers of 2

- 2^{10} (1024) is Kilo, denoted "K“
- 2^{20} (1,048,576) is Mega, denoted "M“
- 2^{30} (1,073,741,824) is Giga, denoted "G“
- 2^{40} (1,099,511,627,776) is Tera, denoted “T"

Octal and Hexadecimal Numbers

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

- The octal number system is the base 8 system with digits 0, 1, 2, 3, 4, 5, 6, and 7.
- The hexadecimal number system is a base 16 system with the first 10 digits borrowed from the decimal system and the letters A, B, C, D, E, and F used for the values 10, 11, 12, 13, 14, and 15, respectively.

Octal and Hexadecimal Numbers

- $(127.4)_8 = 1 * 8^2 + 2 * 8^1 + 7 * 8^0 + 4 * 8^{-1} = (87.5)_{10}$
- $(B65F)_{16} = 11 * 16^3 + 6 * 16^2 + 5 * 16^1 + 15 * 16^0 = (46687)_{10}$

Octal and Hexadecimal Numbers

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

- Conversion from **binary** to **octal** is easily accomplished by partitioning the binary number **into groups of three bits each**, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group.
  - $(010\ 110\ 001\ 101\ 011.111\ 100\ 000\ 110)_2 = (26153.7406)_8$

Octal and Hexadecimal Numbers

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1111	F

- Conversion from **binary** to **hexadecimal** is easily accomplished by partitioning the binary number **into groups of four bits each**, starting from the binary point and proceeding to the left and to the right. The corresponding hexadecimal digit is then assigned to each group.

$$(0010\ 1100\ 0110\ 1011.1111\ 0000\ 0110)_2 = (2C6B.F06)_{16}$$

Octal and Hexadecimal Numbers

- Conversion from octal or hexadecimal to binary is done by reversing the procedure just performed. Each octal digit is converted to a 3-bit binary equivalent, and extra 0s are deleted. Similarly, each hexadecimal digit is converted to its 4-bit binary equivalent.
  - $(673.12)_8$ = 110 111 011.001 010 = $(110111011.00101)_2$
  - $(3A6.C)_{16}$ = 0011 1010 0110.1100 = $(1110100110.11)_2$

Nur

Decimal	Binary	Octal	Hexadecimal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Number Ranges

- In digital computers, the range of numbers that can be represented is based on the number of bits available in the hardware structures that store and process information.
- The number of bits in these structures is most frequently a power of two, such as 8, 16, 32, and 64.
  - 8-bit unsigned integers: 0 –> 2^8-1 (0 -> 255)
  - 16-bit unsigned integers: 0 –> 2^{16}-1 (0 -> 65535)
  - 32-bit unsigned integers: 0 –> 2^{32}-1 (0 -> 4,294,967,295)
  - 64-bit unsigned integers: 0 –> 2^{64}-1 (0 -> 18,446,744,073,709,551,615)
- Since the numbers of bits is fixed by the structures, the addition of leading or trailing zeros to represent numbers is necessary, and the range of numbers that can be represented is also fixed.

4. Arithmetic Operations

- Arithmetic operations with numbers in base *r* follow the same rules as for decimal numbers.
  - However, when a base other than the familiar base 10 is used, one must be careful to use only *r* allowable digits and perform all computations with base *r* digits
- Binary addition

Carries	00000	101100
Augend	01100	10110
Addend	+ 10001	+ 10111
Sum	11101	101101

4. Arithmetic Operations

- Binary subtraction: The rules for subtraction are the same as in decimal, except that a borrow into a given column adds 2 to the minuend bit. (A borrow in the decimal system adds 10 to the minuend digit.)

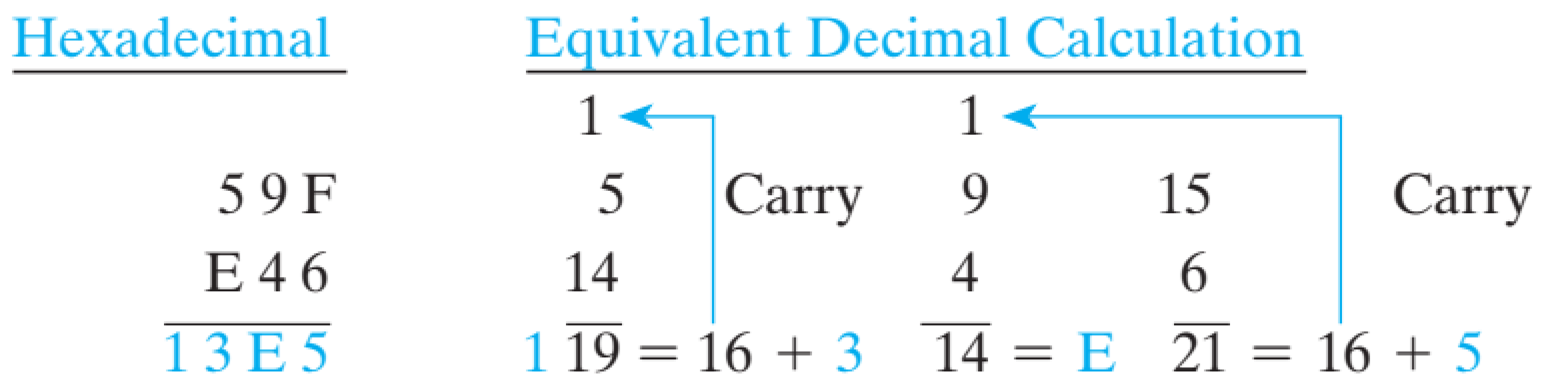
Borrows	00000	00110		00110
Minuend	10110	10110	10011	11110
Subtrahend	- 10010	- 10011	- 11110	- 10011
Difference	00100	00011		-01011

4. Arithmetic Operations

- Binary multiplication
  - The multiplier digits are always 1 or 0. Therefore, the partial products are equal either to the multiplicand or to 0.

```
Multiplicand         1011
Multiplier        x   101
                     1011
                    0000
                   1011
                   ------
Product            110111
```

Hexadecimal Addition



Hexadecimal	Equivalent Decimal Calculation				
	1		1		
5 9 F	5	Carry	9	15	Carry
E 4 6	14		4	6	
1 3 E 5	1 19 = 16 + 3		14 = E	21 = 16 + 5	

Octal Multiplication

- In general, the multiplication of two base *r* numbers can be accomplished by doing all the arithmetic operations in decimal and converting intermediate results one at a time.

Octal	Octal		Decimal		Octal
762	5×2	=	$10 = 8 + 2$	=	12
45	$5 \times 6 + 1$	=	$31 = 24 + 7$	=	37
4672	$5 \times 7 + 3$	=	$38 = 32 + 6$	=	46
3710	4×2	=	$8 = 8 + 0$	=	10
43772	$4 \times 6 + 1$	=	$25 = 24 + 1$	=	31
	$4 \times 7 + 3$	=	$31 = 24 + 7$	=	37

Conversion from Decimal to Other Bases

- If the number includes a radix point, we need to separate the number into an integer part and a fraction part, since the two parts must be converted differently.
    - The conversion of decimal numbers with both integer and fractional parts is done by converting each part separately and then combining the two answers.
- The conversion of a decimal integer to a number in base *r* is done by dividing the number and all successive quotients by *r* and accumulating the remainders.
    - The quotients are divided by r until the result is 0.

Conversion from Decimal to Other Bases

- Conversion of Decimal integers to Octal
  - Convert decimal 153 to octal

$$\begin{array}{rlrcl}
153/8 = 19 + 1/8 & & \text{Remainder} = 1 & \uparrow & \text{Least significant digit} \\
19/8 = 2 \;\; + 3/8 & & = 3 & \vert & \\
2/8 = 0 \;\; + 2/8 & & = 2 & \vert & \text{Most significant digit}
\end{array}$$

$$(153)_{10} = (231)_8$$

Conversion from Decimal to Other Bases

- Conversion of Decimal integers to binary
  - Convert decimal 41 to binary

$41/2 = 20 + 1/2$	Remainder $= 1$	$\uparrow$	Least significant digit
$20/2 = 10$	$= 0$		
$10/2 = 5$	$= 0$		
$5/2 = 2 + 1/2$	$= 1$		
$2/2 = 1$	$= 0$		
$1/2 = 0 + 1/2$	$= 1$		Most significant digit

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

Conversion from Decimal to Other Bases

- The conversion of a decimal fraction to base *r* is accomplished by a method similar to that used for integers, except that **multiplication by *r*** is used instead of division, and integers are accumulated instead of remainders.
    - The process of multiplying fractions by *r* does not necessarily end with zero, so we must decide how many digits of the fraction to use from the conversion.
- Convert decimal 0.6875 to binary:

$$\begin{array}{rcll}
0.6875 \times 2 & = & 1.3750 & \text{Integer} = 1 \quad \text{Most significant digit} \\
0.3750 \times 2 & = & 0.7500 & = 0 \\
0.7500 \times 2 & = & 1.5000 & = 1 \\
0.5000 \times 2 & = & 1.0000 & = 1 \quad \text{Least significant digit}
\end{array}$$

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

Conversion from Decimal to Other Bases

- Conversion of Decimal Fractions to Octal
  - Convert decimal 0.513 to a three-digit octal fraction:

$$\begin{array}{lrl}
0.513 \times 8 = 4.104 & \text{Integer} = 4 & \text{Most significant digit} \\
0.104 \times 8 = 0.832 & = 0 & \big\downarrow \\
0.832 \times 8 = 6.656 & = 6 & \big\downarrow \\
0.565 \times 8 = 5.248 & = 5 & \text{Least significant digit}
\end{array}$$

  - The answer, to three significant figures, is obtained from the integer digits. Note that the last integer digit, 5, is used for rounding in base 8 of the second-to-the-last digit, 6, to obtain

$$(0.513)_{10} = (0.407)_8$$

Conversion from Decimal to Other Bases

- The conversion of decimal numbers with both integer and fractional parts is done by converting each part separately and then combining the two answers.

$$(153.513)_{10} = (231.407)_8$$

5. Decimal Codes

- Information Types
  - Numeric
    - Must represent range of data needed
    - Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted
    - Tight relation to binary numbers
  - Non-numeric
    - Greater flexibility since arithmetic operations not applied.
    - Not tied to binary numbers

5. Decimal Codes

- Given n binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.
- Example: A binary code for the seven colors of the rainbow
  - Code 100 is not used

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

5. Decimal Codes

- Number of bits required for coding
  - Given M elements to be represented by a binary code, the minimum number of bits, n, needed, satisfies the following relationships:

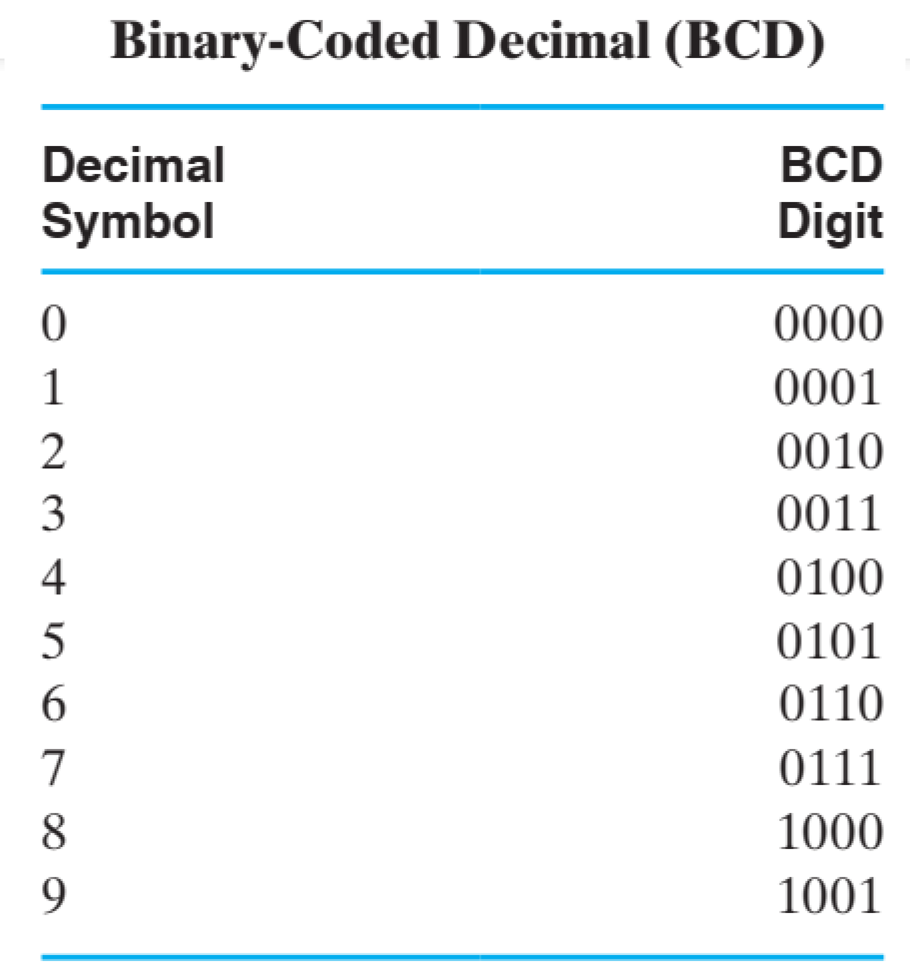
    $2^n \geq M > 2^{(n-1)}$

    $n = \lceil \log_2 M \rceil$ where $\lceil x \rceil$, called the ceiling function, is the integer greater than or equal to x.
  - Example: How many bits are required to represent decimal digits with a binary code? -> 4

5. Decimal Codes

- Binary-Coded Decimal (BCD)
  - Decimal digits stored in binary
  - Four bits/digit
  - A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9, inclusive.

    396 is coded as 0011 1001 0110
  - Easier to interpret, but harder to manipulate.



Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

6. Alphanumeric Codes

- Many applications of digital computers require the handling of data consisting not only of numbers, but also of letters.
- ASCII Character Code
  - American Standard Code for Information Interchange
  - It uses seven bits to code 128 characters.
    - The seven bits of the code are designated by B_1 through B_7, with B_7 being the most significant bit: $B_7B_6B_5B_4B_3B_2B_1$.
      - The letter A, for example, is represented in ASCII as 1000001.
    - 94 Graphic printing characters, 34 Non-printing characters

American Standard Code for Information Interchange (ASCII)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Characters

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Character Code

- ASCII is a 7-bit code, but most computers manipulate an 8-bit quantity as a single unit called a byte. Therefore, ASCII characters most often are stored one per byte, with the most significant bit set to 0.
- The extra bit is sometimes used for specific purposes, depending on the application.
    - For example, some printers recognize an additional 128 8-bit characters, with the most significant bit set to 1. These characters enable the printer to produce additional symbols, such as those from the Greek alphabet or characters with accent marks as used in languages other than English.
- ASCII was developed for the English alphabet but, even extending it to 8-bits, it is unable to support other alphabets and scripts that are commonly used around the world.

Unicode

- Unicode was developed as an industry standard for providing a common representation of symbols and ideographs for the most of the world's languages.
- By providing a standard representation that covers characters from many different languages, Unicode removes the need to convert between different character sets and eliminates the conflicts that arise from using the same numbers for different character sets

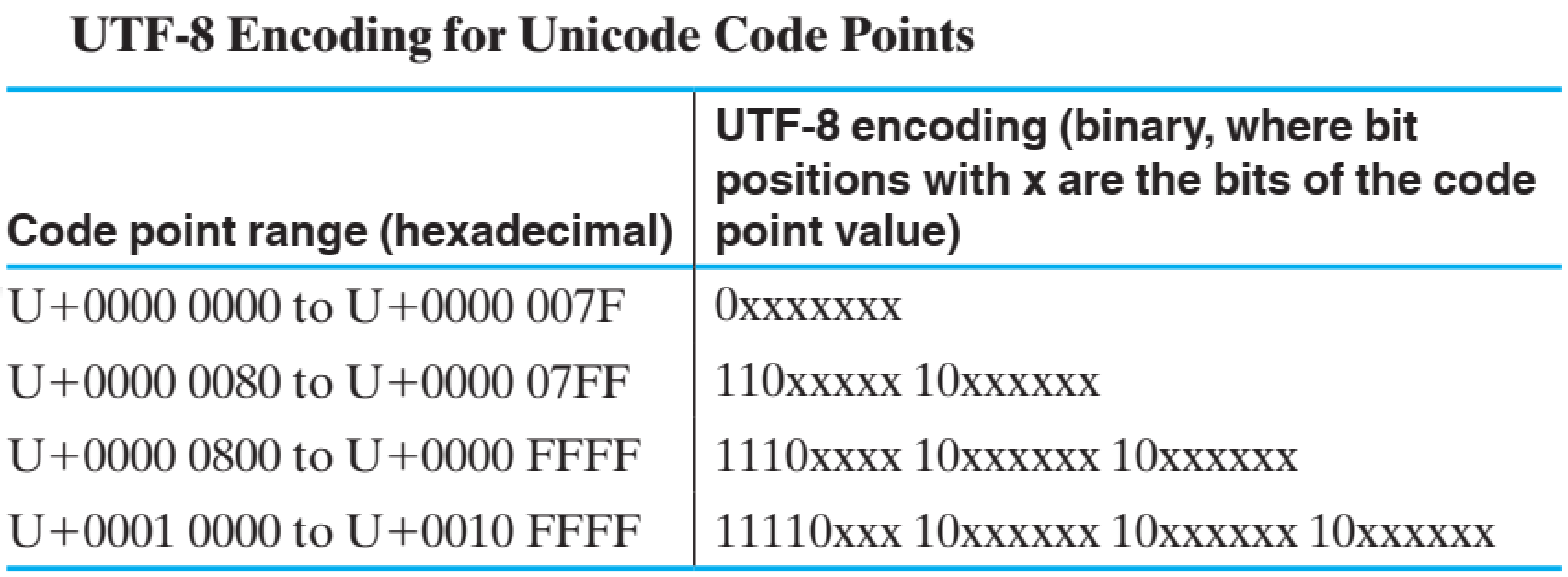
Unicode

- Unicode provides a unique number called a *code point* for each character, as well as a unique name.
  - A common notation for referring to a code point is the characters "U+" followed by the **four to six hexadecimal digits** of the code point.
    - For example, U+0030 is the character "0", named Digit Zero.
  - The first 128 code points of Unicode, from U+0000 to U+007F, correspond to the ASCII characters.
  - Unicode currently supports over a million code points from a hundred scripts worldwide.

Unicode

- There are several standard encodings of the code points that range from 8 to 32 bits (1 to 4 bytes).
  - UTF-8 (UCS Transformation Format, where UCS stands for Universal Character Set) is a variable-length encoding that uses from 1 to 4 bytes for each code point.
  - UTF-16 is a variable-length encoding that uses either 2 or 4 bytes for each code point.
  - UTF-32 is a fixed-length that uses 4 bytes for every code point.

UTF-8

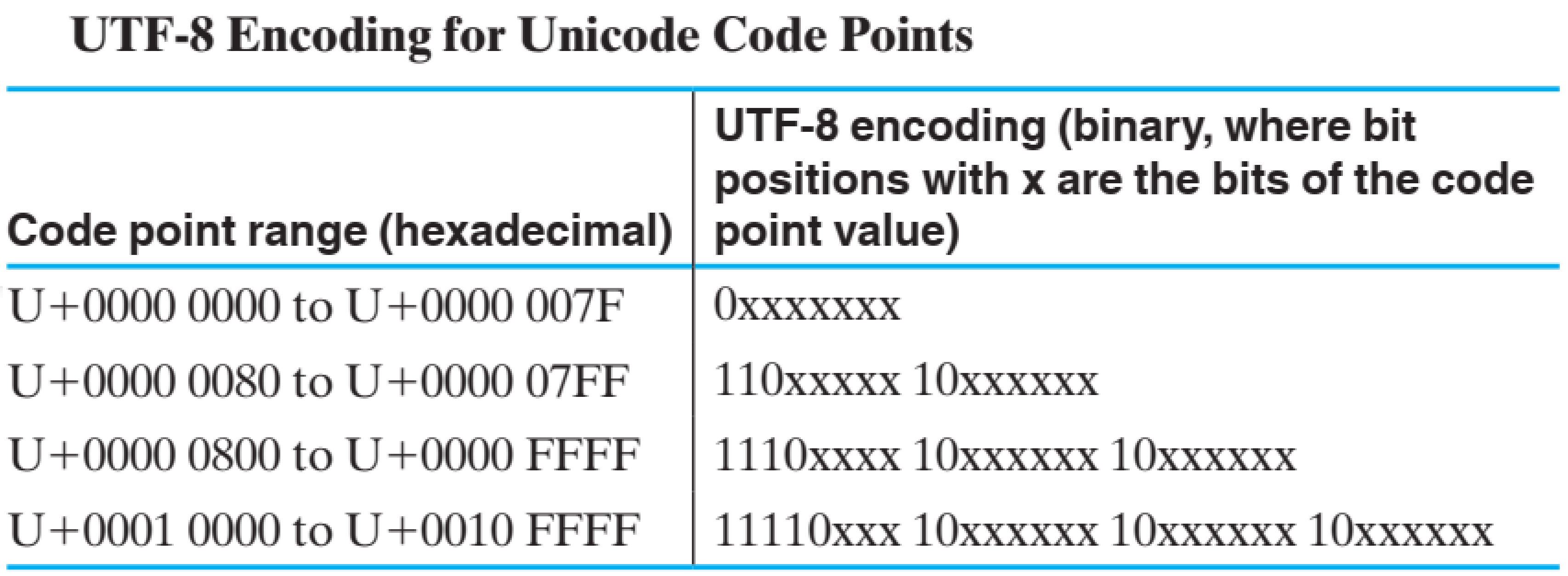


UTF-8 Encoding for Unicode Code Points

Code point range (hexadecimal)	UTF-8 encoding (binary, where bit positions with x are the bits of the code point value)
U+0000 0000 to U+0000 007F	0xxxxxxx
U+0000 0080 to U+0000 07FF	110xxxxx 10xxxxxx
U+0000 0800 to U+0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+0001 0000 to U+0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- The first 128 code points are encoded with a single byte, which provides compatibility between ASCII and UTF-8.
  - Thus a file or character string that contains only ASCII characters will be the same in both ASCII and UTF-8.
- In UTF-8, the number of bytes in a multi-byte sequence is indicated by the number of leading ones in the first byte. Valid encodings must use the least number of bytes necessary for a given code point.

UTF-8



UTF-8 Encoding for Unicode Code Points

Code point range (hexadecimal)	UTF-8 encoding (binary, where bit positions with x are the bits of the code point value)
U+0000 0000 to U+0000 007F	0xxxxxxx
U+0000 0080 to U+0000 07FF	110xxxxx 10xxxxxx
U+0000 0800 to U+0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+0001 0000 to U+0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- The code point U+0054, Latin capital letter T, “T”:
  - in the range of U+0000 0000 to U+0000 007F, so it would be encoded with one byte 0xxxxxxx
  - 54_{16} = $(0101\ 0100)_2$ -> U+0054 would be encoded with a value of $(01010100)_2$.
- The code point U+00B1, plus-minus sign, “±”:
  - is in the range of U+0000 0080 to U+0000 07FFF, so it would be encoded with two bytes 110xxxxx 10xxxxxx
  - $B1_{16}$ = $(1011\ 0001)_2$ -> U+00B1 would be encoded with a value of $(11000010\ 10110001)_2$.

Parity Bit

- To detect errors in data communication and processing, an additional bit is sometimes added to a binary code word to define its parity.
    - A *parity bit* is the extra bit included to make the total number of 1s in the resulting code word either even or odd.
    - Even parity being more common.
    - Parity may be used with binary numbers as well as with codes, and the parity bit may be placed in any fixed position in the code.

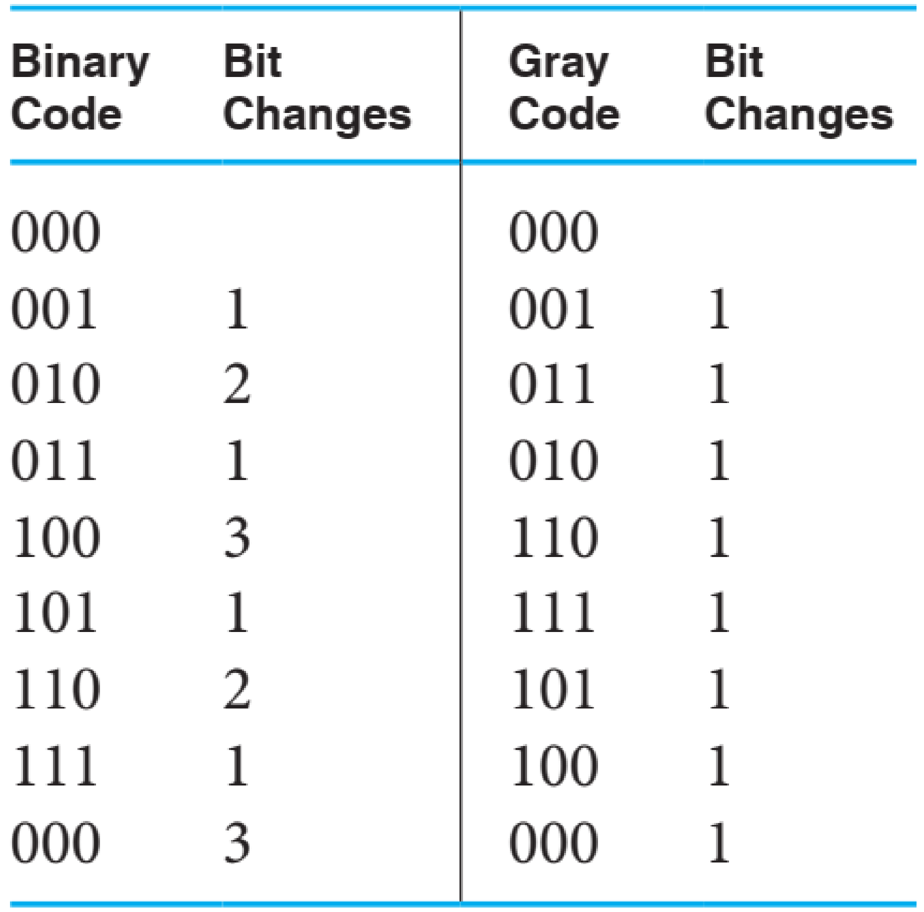
	With Even Parity	**With Odd Parity**
1000001	01000001	11000001
1010100	11010100	01010100

7. Gray Codes

Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

7. Gray Codes

- As we count up or down using binary codes, the number of bits that change from one binary value to the next varies.
- As we count from 000 up to 111 and "roll over" to 000, the number of bits that change between the binary values ranges from 1 to 3.
- For many applications, multiple bit changes as the circuit counts is not a problem. There are applications, however, in which a change of more than one bit when counting up or down can cause serious problems.



Binary Code	Bit Changes	Gray Code	Bit Changes
000		000	
001	1	001	1
010	2	011	1
011	1	010	1
100	3	110	1
101	1	111	1
110	2	101	1
111	1	100	1
000	3	000	1