Chapter 1
Digital Systems and Information

CSCM601150, 2022/2023 - 1

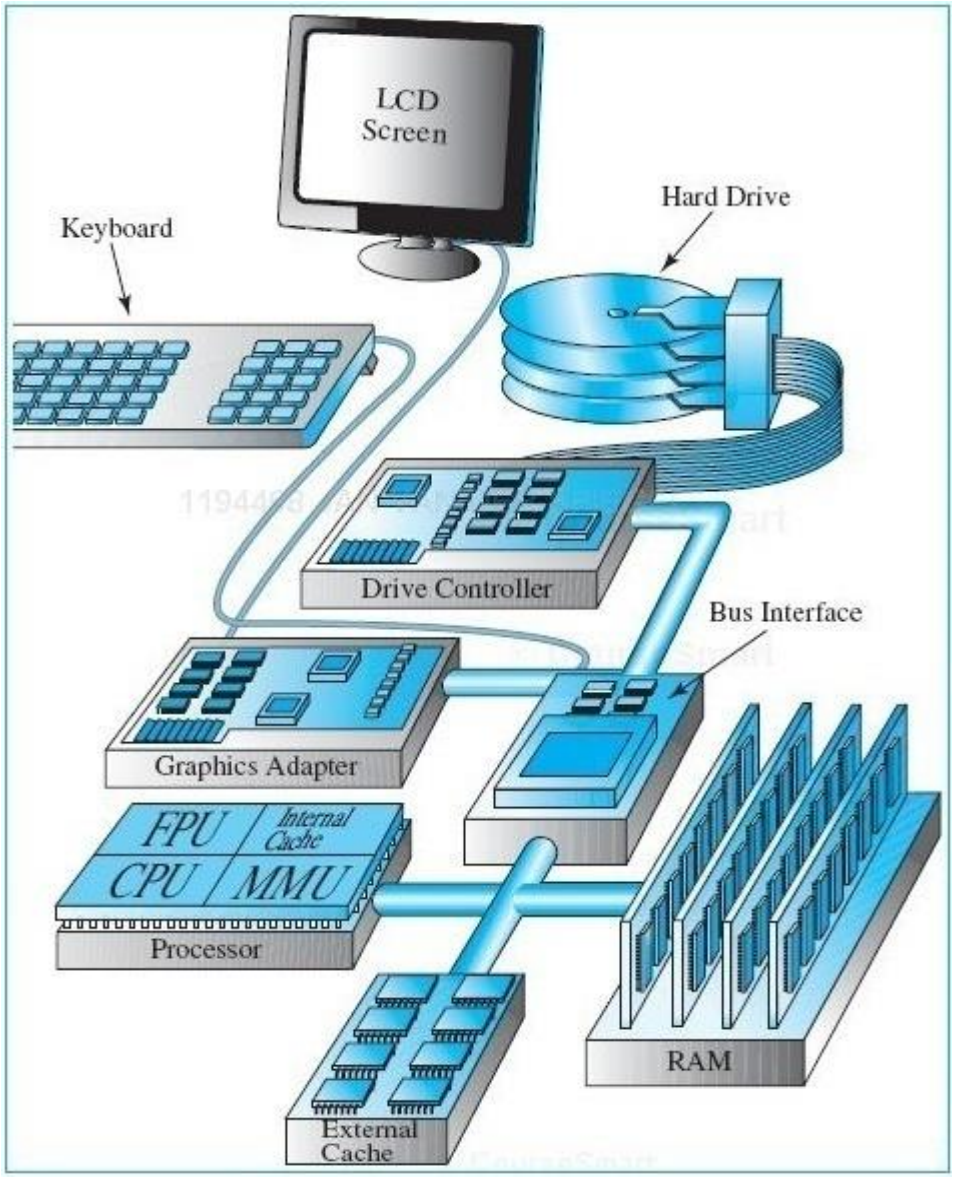
Dosen: Erdefi Rakun dan Tim Dosen PSD

Fasilkom UI

Outline

- Information Representations
- Number Systems
- Arithmetic Operation
- Decimal Codes
- Alphanumeric Codes
- Gray Codes

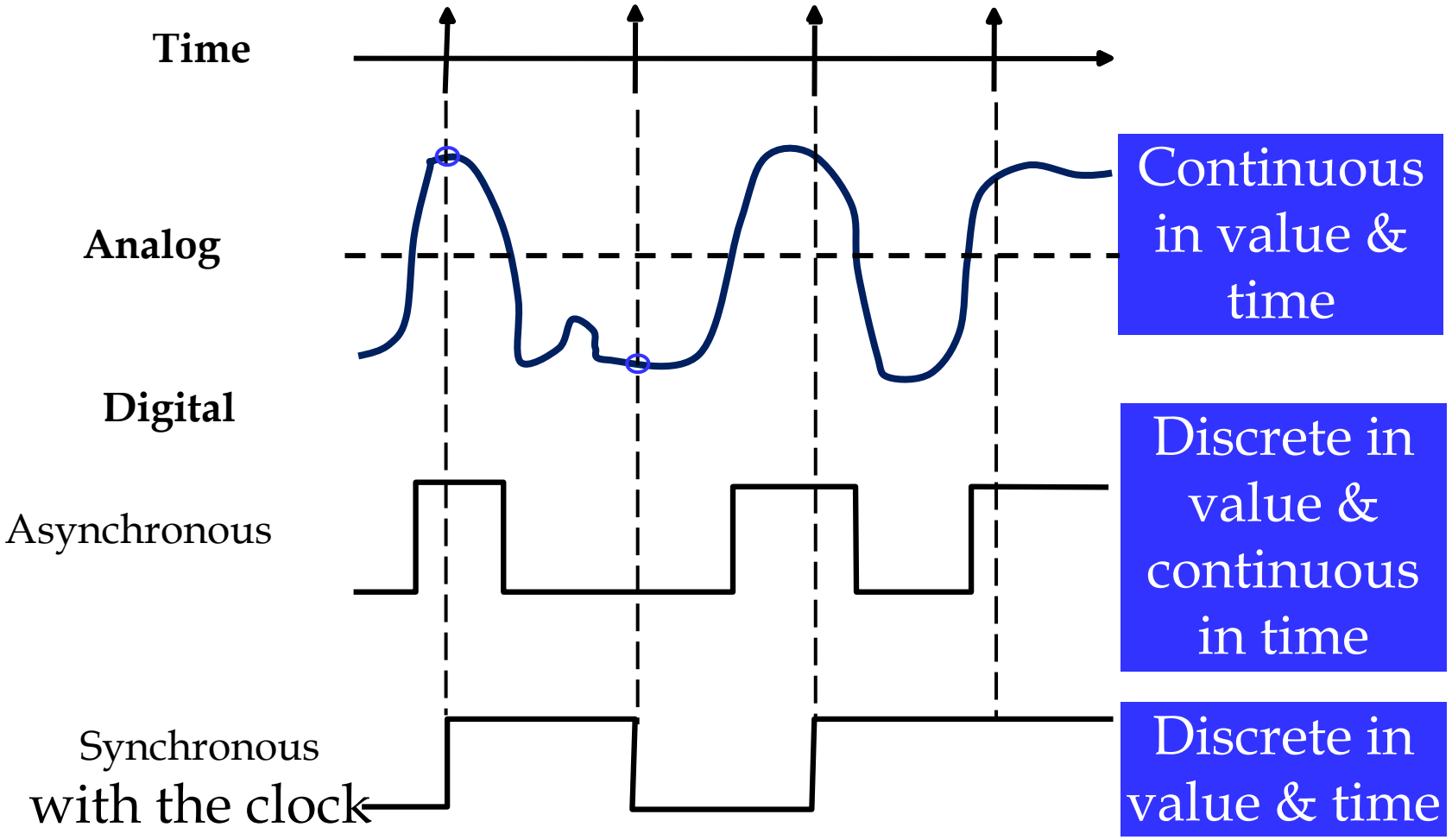
Note: Portion of this material are taken from Aaron Tan's slide and other portions of this material © 2008 by Pearson Education,Inc



1-1 INFORMATION REPRESENTATION in Digital Systems

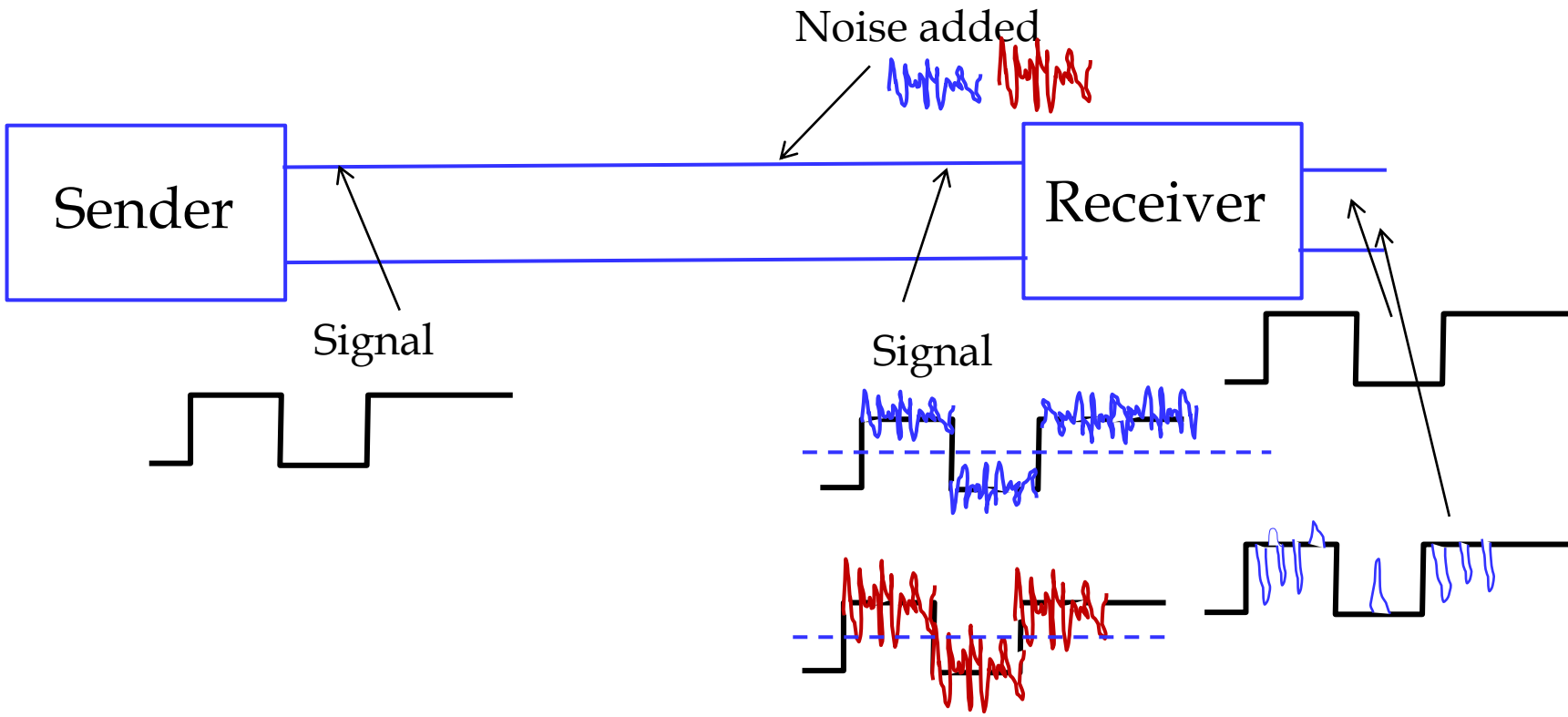
- Information variables represented by physical quantities.
- For digital systems, the variables take on discrete values.
- Two levels or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words Off and On.

Signal Examples Over Time



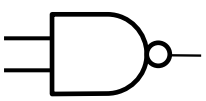
Advantage of Digital Circuit

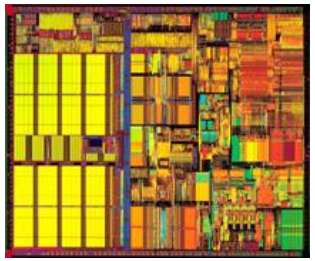
- Immunity to noise
- Illustration:



Binary Values: Other Physical Quantities

- **What are other physical quantities represent 0 and 1?**
  - **Logic Gates, CPU:** Voltage





  - **Disk**

Magnetic Field Direction

  - **CD**

Surface Pits/Light

  - **Dynamic RAM**

Electrical Charge

Information Representation

Bit (*Bi*nary *d*igit)

- 0 and 1
- Represent *false* and *true* in logic
- Represent the *low* and *high* states in electronic devices
- Bits have no inherent meaning, can represent
  - Unsigned and signed integers, Characters, Floating-point numbers, Images, sound, etc.

Other units

- Byte: 8 bits
- Nibble: 4 bits (seldom used)
- Word: Multiples of byte (e.g.: 1 byte, 2 bytes, 4 bytes, 8 bytes, etc.), depending on the architecture of the computer system

1-2 Number Systems

Information Representation:

N bits can represent up to 2^N values.

Examples:

2 bits → represent up to 4 values (00, 01, 10, 11)
3 bits → rep. up to 8 values (000, 001, 010, …, 110, 111)
4 bits → rep. up to 16 values (0000, 0001, 0010, …., 1111)

To represent M values, $\log_2 M$ bits are required.

Examples:

32 values → requires 5 bits
64 values → requires 6 bits
1024 values → requires 10 bits
40 values → how many bits?
100 values → how many bits?

Positional Number Systems

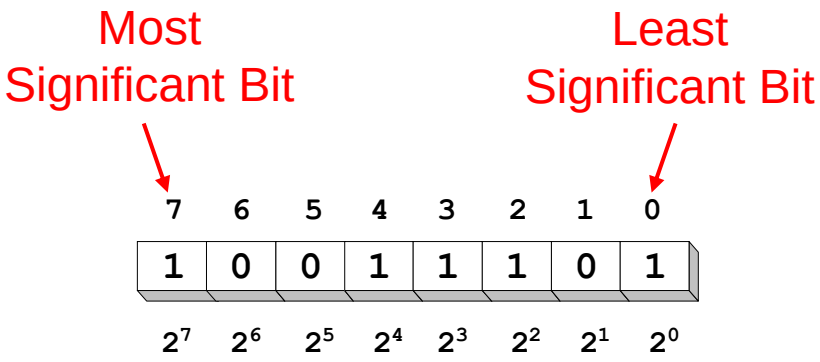
Different Representations of Natural Numbers

1. XXVII Roman numerals (not positional)
2. 27 Radix-10 or decimal number (positional)
3. 11011_2 Radix-2 or binary number (also positional)

Bit Numbering

Least significant bit (LSB) is rightmost (bit 0)

Most significant bit (MSB) is leftmost (bit 7 in an 8-bit number)



Decimal (Base 10) System (1/2)

A weighted-positional number system

- **Base** or **radix** is 10 (the *base* or *radix* of a number system is the total number of symbols/digits allowed in the system)
- Symbols/digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Position is important, as the value of each symbol/digit is dependent on its **type** and its **position** in the number
- Example, the 9 in the two numbers below has different values:
  - $(7594)_{10} = (7 \times 10^3) + (5 \times 10^2) + (9 \times 10^1) + (4 \times 10^0)$
  - $(912)_{10} = (9 \times 10^2) + (1 \times 10^1) + (2 \times 10^0)$
- In general,

$$(a_n a_{n-1} \ldots a_0 . f_1 f_2 \ldots f_m)_{10} = (a_n \times 10^n) + (a_{n-1} \times 10^{n-1}) + \ldots + (a_0 \times 10^0) + (f_1 \times 10^{-1}) + (f_2 \times 10^{-2}) + \ldots + (f_m \times 10^{-m})$$

Decimal (Base 10) System (2/2)

- Weighing factors (or weights) are in powers of 10:

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

- To evaluate the decimal number 593.68, the digit in each position is multiplied by the corresponding weight:

  $5\times10^2 + 9\times10^1 + 3\times10^0 + 6\times10^{-1} + 8\times10^{-2}$

  $= (593.68)_{10}$

Other Number Systems (1/2)

- Binary (base 2)
  - Weights in powers of 2
  - Binary digits (bits): **0, 1**
- Octal (base 8)
  - Weights in powers of 8
  - Octal digits: **0, 1, 2, 3, 4, 5, 6, 7**.
- Hexadecimal (base 16)
  - Weights in powers of 16
  - Hexadecimal digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**.
- Base/radix *R*:
  - Weights in powers of *R*

Other Number Systems (2/2)

In some programming languages/software, special notations are used to represent numbers in certain bases

- In programming language C
  - Prefix 0 for octal. E.g.: 032 represents the octal number $(32)_8$
  - Prefix 0x for hexadecimal. E.g.: 0x32 represents the hexadecimal number $(32)_{16}$
- In PCSpim (a MIPS simulator)
  - Prefix 0x for hexadecimal. E.g.: 0x100 represents the hexadecimal number $(100)_{16}$
- In Verilog, the following values are the same
  - 8'b11110000: an 8-bit binary value 11110000
  - 8'hF0: an 8-bit binary value represented in hexadecimal F0
  - 8'd240: an 8-bit binary value represented in decimal 240

1-4 ARITHMETIC OPERATIONS – Binary Arithmetic

- Single Bit Addition with Carry
- Multiple Bit Addition
- Single Bit Subtraction with Borrow
- Multiple Bit Subtraction
- Multiplication

Single Bit Binary Addition with Carry

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Multiple Bit Binary Addition

- **Extending this to two multiple bit examples:**

Carries	0	0
Augend	01100	10110
Addend	+10001	+10111
Sum		

- *Note: The 0 is the default Carry-In to the least significant bit.*

Single Bit Binary Subtraction with Borrow

- Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):
- Borrow in (Z) of 0:

Z	**0**	**0**	**0**	**0**
X	**0**	**0**	**1**	**1**
- Y	**-0**	**-1**	**-0**	**-1**
BS	**0 0**	**1 1**	**0 1**	**0 0**

- Borrow in (Z) of 1:

Z	**1**	**1**	**1**	**1**
X	**0**	**0**	**1**	**1**
- Y	**-0**	**-1**	**-0**	**-1**
BS	**11**	**1 0**	**0 0**	**1 1**

Multiple Bit Binary Subtraction

- Extending this to two multiple bit examples:

Borrows	0	0
Minuend	10110	10110
Subtrahend	- 10010	- 10011
Difference		

- *Notes:*

  The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.

Binary Multiplication

The binary multiplication table is simple:

0 * 0 = 0 | 1 * 0 = 0 | 0 * 1 = 0 | 1 * 1 = 1

Extending multiplication to multiple digits:

Multiplicand	**1011**
Multiplier	**x 101**
Partial Products	**1011**
	0000 -
	1011 - -
Product	**110111**

QUICK REVIEW QUESTIONS (1)

What is $(10101)_2 \times (011)_2$?

a. $(10000)_2$ b. $(110111)_2$ c. $(111111)_2$ d. $(111011)_2$ e. $(101101)_2$

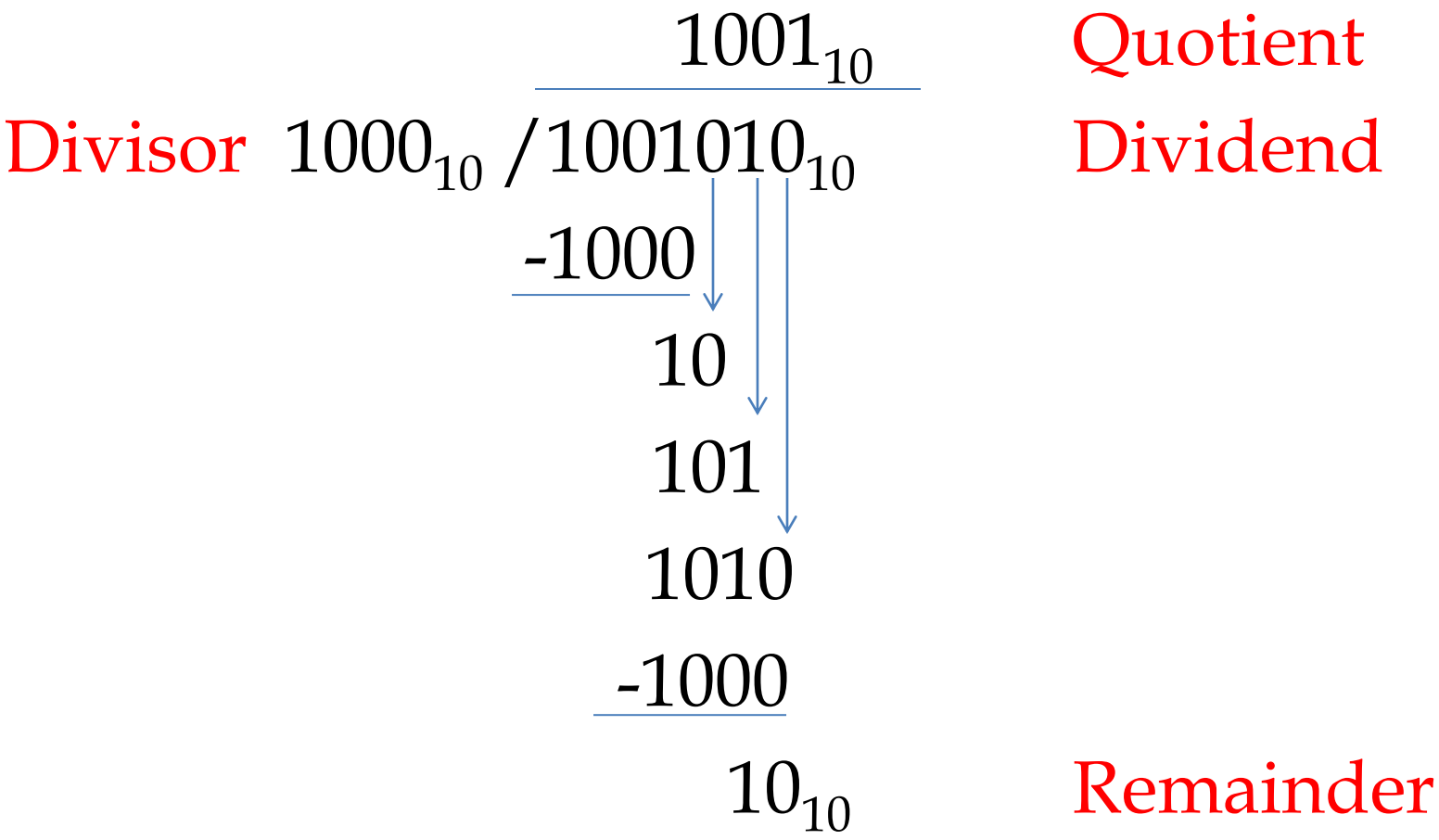
Perform the following operations on binary numbers.

a. $(1011\ 1110)_2 + (1000\ 1101)_2$

b. $(1101\ 0010)_2 - (0110\ 1101)_2$

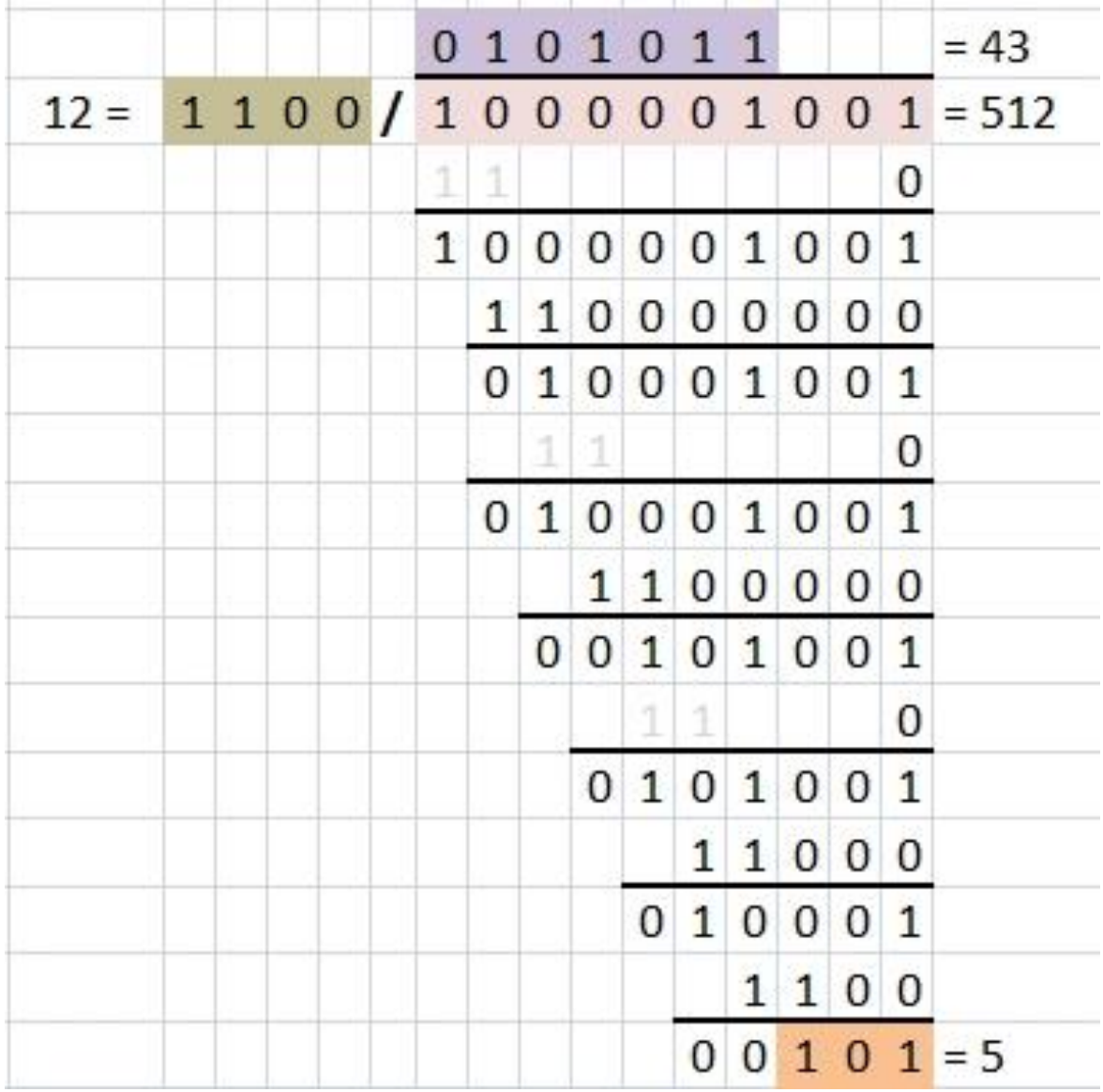
c. $(1110\ 0101)_2 - (0010\ 1110)_2$

Division



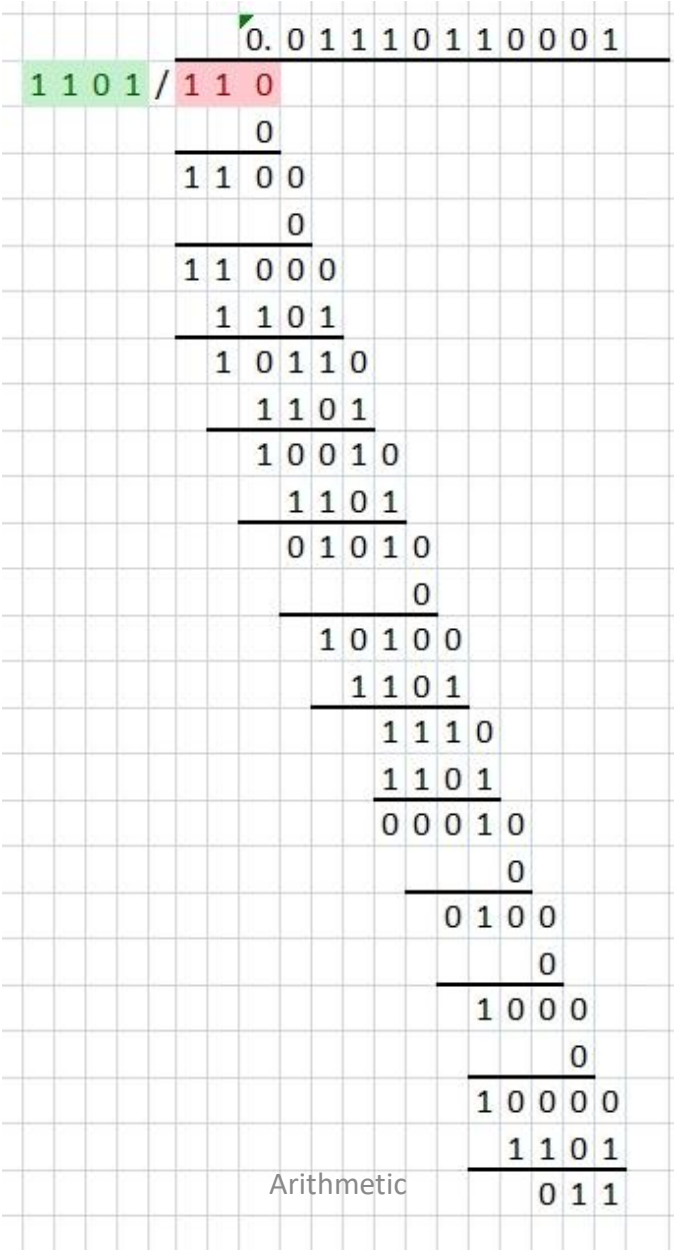
Dividend = Quotient x Divisor + Remainder

Binary Division: 512 / 12



Taken from Daniel J. Sorin Slide

Division Exercise $110_2 : 1101_2$



BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Exponent	Value
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152

Converting Binary to Decimal

- To convert to decimal, use decimal arithmetic to form S (digit × respective power of 2).
- Example:Convert 11010_2 to N_{10}:

Base-*R* To Decimal Conversion

Easy!

$1101.101_2 = 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3}$

$572.6_8 =$

$2A.8_{16} =$

$341.24_5 =$

QUICK REVIEW QUESTIONS (2)

DLD page 37
Questions 2-1 to 2-4.

2-1. Convert the binary number 1011011 to its decimal equivalent.

a. 5 b. 63 c. 91 d. 92 e. 139

2-2. What is the weight of the digit '3' in the base-7 number 12345?

a. 3 b. 7 c. 14 d. 21 e. 49

QUICK REVIEW QUESTIONS (2)

2-3. Which of the following has the largest value?

a. $(110)_{10}$ b. $(10011011)_2$ c. $(1111)_5$ d. $(9A)_{16}$
e. $(222)_8$

2-4. If $(321)_4 = (57)_{10}$, what is the decimal equivalent of $(321000000)_4$?

a. 57×10^4 b. 57×10^6 c. 57×4^4 d. 57×4^6
e. 57^4

Decimal To Binary Conversion

- Method 1
  - Sum-of-Weights Method
- Method 2
  - Repeated Division-by-2 Method (for whole numbers)
  - Repeated Multiplication-by-2 Method (for fractions)

Sum-of-Weights Method

Determine the set of binary weights whose sum is equal to the decimal number

- $(9)_{10} = 8 + 1 = 2^3 + 2^0 = (1001)_2$
- $(18)_{10} = 16 + 2 = 2^4 + 2^1 = (10010)_2$
- $(58)_{10} = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 = (111010)_2$
- $(0.625)_{10} = 0.5 + 0.125 = 2^{-1} + 2^{-3} = (0.101)_2$

Repeated Division-by-2 Method

To convert a whole number to binary, use successive division by 2 until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant bit (LSB)* and the last as the *most significant bit (MSB)*.

$(43)_{10} = (101011)_2$

2	43	
2	21 rem 1	← LSB
2	10 rem 1	
2	5 rem 0	
2	2 rem 1	
2	1 rem 0	
	0 rem 1	← MSB

Repeated Multiplication-by-2 Method

To convert decimal fractions to binary, repeated multiplication by 2 is used, until the fractional product is 0 (or until the desired number of decimal places). The carried digits, or *carries*, produce the answer, with the first carry as the MSB, and the last as the LSB.

$(0.3125)_{10} = (.0101)_2$

	Carry	
$0.3125 \times 2 = 0.625$	0	←MSB
$0.625 \times 2 = 1.25$	1	
$0.25 \times 2 = 0.50$	0	
$0.5 \times 2 = 1.00$	1	←LSB

Conversion Between Decimal and Other Bases

- Base-*R* to decimal: multiply digits with their corresponding weights.
- Decimal to binary (base 2)
  - Whole numbers repeated division-by-2
  - Fractions: repeated multiplication-by-2
- Decimal to base-*R*
  - Whole numbers: repeated division-by-*R*
  - Fractions: repeated multiplication-by-*R*

QUICK REVIEW QUESTIONS (3)

- DLD page 37
  Questions 2-5, 2-7.

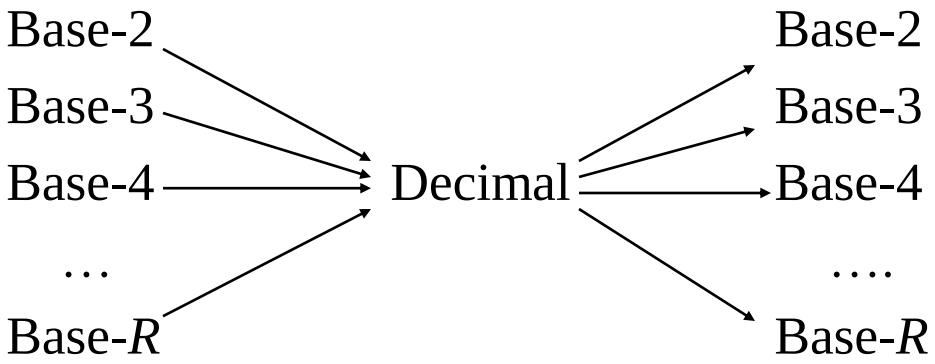
2-5. Convert each of the following decimal numbers to binary (base two) with at most eight digits in the fractional part, rounded to eight places.

a. 2000 b. 0.875 c. 0.07 d. 12.345

2-7. Convert each of the decimal numbers in question 2-5 above to octal (base 8) with at most four digits in the fractional part rounded to four places.

Conversion Between Bases

- In general, conversion between bases can be done via decimal:



- Shortcuts for conversion between bases 2, 4, 8, 16 (see next slide)

Binary To Octal/Hexadecimal Conversion

- Binary → Octal: partition in groups of 3
  $(10\ 111\ 011\ 001\ .\ 101\ 110)_2$ =
- Octal → Binary: reverse
  $(2731.56)_8$ =
- Binary → Hexadecimal: partition in groups of 4
  $(101\ 1101\ 1001\ .\ 1011\ 1000)_2$ =
- Hexadecimal → Binary: reverse
  $(5D9.B8)_{16}$ =

QUICK REVIEW QUESTIONS (4)

- DLD page 37
  Questions 2-9 to 2-10.

2-9. Which of the following octal value is equivalent to the binary number $(110001)_2$?

a. $(15)_8$ b. $(31)_8$ c. $(33)_8$ d. $(49)_8$ e. $(61)_8$

2-10. Convert the binary number $(1001101)_2$ to

a. quaternary b. octal c. decimal d. hexadecimal

Numbers in Different Bases

- **Good idea to memorize!**

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
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

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"

- Note: 8 bits (b) are also called a byte (B)

Exercise: what is $(1111\ 1111)_2$ equal to in decimal?

Also what is the maximum number (in decimal) a two bytes long word can represent?

Hexadecimal Addition

Perform the addition $(59F)_{16} + (E46)_{16}$

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

Perform the multiplication $(762)_8 \times (45)_8$

Octal
762
45
4672
3710
43772

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

1-5 Codes

- Flexibility of representation
  - Within constraints below, can assign any binary combination (called a code word) to any data as long as data is uniquely encoded.
- 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

Non-numeric Binary 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

Number of Bits Required

- 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?

Number of Elements Represented

- Given n digits in radix r, there are r^n distinct elements that can be represented.
- But, you can represent m elements, m < r^n
- Examples:
  - You can represent 4 elements in radix r = 2 with n = 2 digits: (00, 01, 10, 11).
  - You can represent 4 elements in radix r = 2 with n = 4 digits: (0001, 0010, 0100, 1000).
  - This second code is called a "one hot" code.

Decimal Codes

- Humans favor decimal numbers. Binary numbers are natural to computers. Hence, conversion is required.
- If little calculation is required, we can use some **coding schemes** to store decimal numbers, for data transmission purposes.
- Examples: BCD (or 8421), Excess-3, 84-2-1, 2421, etc.
- Each decimal digit is represented as a 4-bit code.
- The number of digits in a code is also called the length of the code.

DECIMAL CODES - Binary Codes for Decimal Digits

There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

Decimal	8,4,2,1	Excess-3	8,4,-2,-1	2,4,2,1	Gray
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 0 0	0 1 1 1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 1 0 1	0 1 1 0	0 0 1 0	0 0 1 1
3	0 0 1 1	0 1 1 0	0 1 0 1	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 0 0	0 1 1 0
5	0 1 0 1	1 0 0 0	1 0 1 1	1 0 1 1	0 1 1 1
6	0 1 1 0	1 0 0 1	1 0 1 0	1 1 0 0	0 1 0 1
7	0 1 1 1	1 0 1 0	1 0 0 1	1 1 0 1	0 1 0 0
8	1 0 0 0	1 0 1 1	1 0 0 0	1 1 1 0	1 1 0 0
9	1 0 0 1	1 1 0 0	1 1 1 1	1 1 1 1	1 1 0 1

Binary Coded Decimal (BCD)

- The BCD code is the 8,4,2,1 code.
- 8, 4, 2, and 1 are weights
- BCD is a *weighted* code
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.
- Example: 1001 (9) = 1000 (8) + 0001 (1)
- How many "invalid" code words are there?
- What are the "invalid" code words?

Excess 3 Code and 8, 4, –2, –1 Code

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

- What interesting property is common to these two codes?

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a BINARY CODE.
- $13_{10} = 1101_2$ (This is conversion)
- $13 \Leftrightarrow 0001\,|\,0011$ (This is coding)

Binary Code Decimal (BCD)

- Some codes are unused, like 1010_{BCD}, 1011_{BCD}, … 1111_{BCD}. These codes are considered as errors.
- Easy to convert, but arithmetic operations are more complicated.
- Suitable for interfaces such as keypad inputs.

Decimal digit	BCD 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD Arithmetic

Given a BCD code, we use binary arithmetic to add the digits:

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**

Note that the result is MORE THAN 9, so must be represented by two digits!

To correct the digit, subtract 10 by adding 6 modulo 16.

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**
	+0110	**so add 6**
carry = 1	**0011**	**leaving 3 + cy**
0001	**0011**	**Final answer (two digits)**

If the digit sum is > 9, add one to the next significant digit

BCD Addition Example

- Add 2905_{BCD} to 1897_{BCD} showing carries and digit corrections.

				0
	0001	**1000**	**1001**	**0111**
+	**0010**	**1001**	**0000**	**0101**
	____	____	____	____

1-6 Alphanumeric Codes (1/3)

- Computers also handle textual data.
- Character set frequently used:

  alphabets: 'A' … 'Z', 'a' … 'z'

  digits: '0' … '9'

  special symbols: '$', '.', '@', '*', etc.

  non-printable: NULL, BELL, CR, etc.
- Examples
  - ASCII (8 bits), Unicode

Alphanumeric Codes (2/3)

- ASCII
  - American Standard Code for Information Interchange
  - 7 bits, plus a parity bit for error detection
  - Odd or even parity

Character	ASCII Code
0	0110000
1	0110001
. . .	. . .
9	0111001
:	0111010
A	1000001
B	1000010
. . .	. . .
Z	1011010
[	1011011
\	1011100

Alphanumeric Codes (3/3)

- ASCII table

A: 100 0001

	MSBs							
LSBs	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC_1	!	1	A	Q	a	q
0010	STX	DC_2	“	2	B	R	b	r
0011	ETX	DC_3	#	3	C	S	c	s
0100	EOT	DC_4	$	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	O	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

ALPHANUMERIC CODES - ASCII Character Codes

- This code is a popular code used to represent information sent as character-based data. It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return)
- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).

ASCII Properties

ASCII has some interesting properties:

- Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}
- Upper case A-Z span 41_{16} to $5A_{16}$
- Lower case a-z span 61_{16} to $7A_{16}$
- Lower to upper case translation (and vice versa) occurs by flipping bit 6
- Delete (DEL) is all bits set, a carryover from when punched paper tape was used to store messages. Punching all holes in a row erased a mistake!

UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words
  - See Reading Supplement – Unicode on the Companion Website http://www.prenhall.com/mano

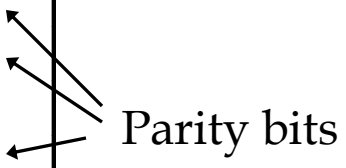
Error Detection (1/4)

- Errors can occur during data transmission. They should be detected, so that re-transmission can be requested.
- With binary numbers, usually single-bit errors occur.
  - Example: 0010 erroneously transmitted as 0011 or 0000 or 0110 or 1010.
- Biquinary code has length 7; it uses 3 additional bits for error-detection.

Error Detection (2/4)

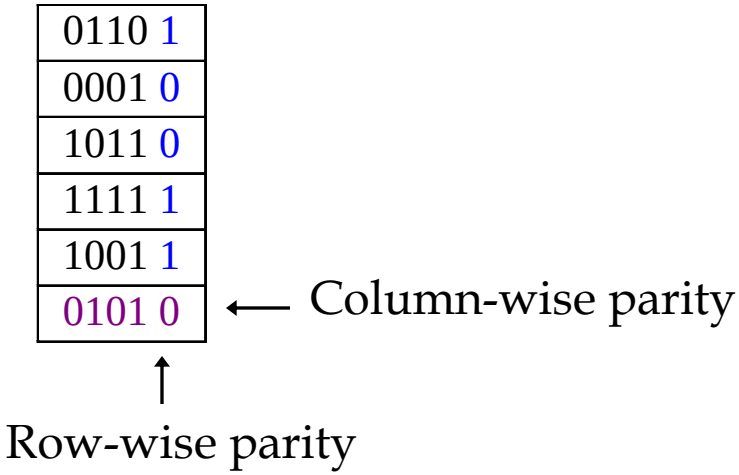
- Parity bit
  - Even parity: additional bit added to make total number of 1's even.
  - Odd parity: additional bit added to make total number of 1's odd.
- Example of odd parity on ASCII values.

Character	ASCII Code
0	0110000 1
1	0110001 0
. . .	. . .
9	0111001 1
:	0111010 1
A	1000001 1
B	1000010 1
. . .	. . .
Z	1011010 1
[	1011011 0
\	1011100 1



Error Detection (3/4)

- Parity bit can detect odd number of errors but not even number of errors.
  - Example: Assume odd parity,
    - 10011 → 10001 (detected)
    - 10011 → 10101 (not detected)
- Parity bits can also be applied to a block of data.



Error Detection (4/4)

- Sometimes, it is not enough to do error detection. We may want to correct the errors.
- Error correction is expensive. In practice, we may use only single-bit error correction.
- Popular technique: Hamming code

Error Correction (1/10)

- Given this 3-bit code C_1

  { 000, 110, 011, 101 }
- With 4 code words, we actually need only 2 bits.
  - We call this *k*, the number of original message bits.
- To add error detection/correction ability, we use more bits than necessary.
  - In this case, the length of each codeword is 3
- We define code efficiency (or rate) by

  k / length of codeword
- Hence, efficiency of C_1 is 2/3.

Error Correction (2/10)

- Given this 3-bit code C_1

  { 000, 110, 011, 101 }
- Can C_1 detect a single bit error?
- Can C_1 correct a single bit error?

Sometimes, we use "1 error" for "single bit error", "2 errors" for "2 bits error", etc.

Error Correction (3/10)

- The distance d between any two code words in a code is the sum of the number of differences between the codewords.
  - Example: $d(000, 110) = 2$; $d(0110,1011) = 3$.
- The Hamming distance δ of a code is the minimum distance between any two code words in the code.
  - Example: The Hamming distance of C_1 is 2.
- A code with Hamming distance of 2 can detect 1 error.

Error Correction (4/10)

- Given this 6-bit code C_2

  { 000000, 111000, 001110, 110011 }
- What is its efficiency?
- What is its Hamming distance?
- Can it correct 1 error?

- Can it correct 2 errors?

Error Correction (5/10)

- Hamming code: a popular error-correction code
- To detect k (or fewer) single-bit errors, the code must have Hamming Distance $D(min) = k + 1$
- To correct k (or fewer) single-bit errors, the code must have Hamming Distance $D(min) = 2k + 1$
- Suppose a code n bits consists of m bit data bits and r parity bits $(n = m + r)$

Taken from Linda Null's book

Error Correction (6/10)

- For single-bit errors:
  - Error can occur in any of n positions. For each valid code word has n illegal code words of distance of 1.
  - $n+1$ bit patterns associated with each legal code word (1 legal + n illegal)
  - $n = m + r$, there are 2^n bit patterns possible

$$(n + 1) \times 2^m \leq 2^n$$

$$(m + r + 1) \times 2^m \leq 2^{m+r}$$

$$(m + r + 1) \leq 2^r$$

- Suppose $m = 8$, $(8 + r + 1) \leq 2^r$, r must be ≥ 4

Taken from Linda Null's book

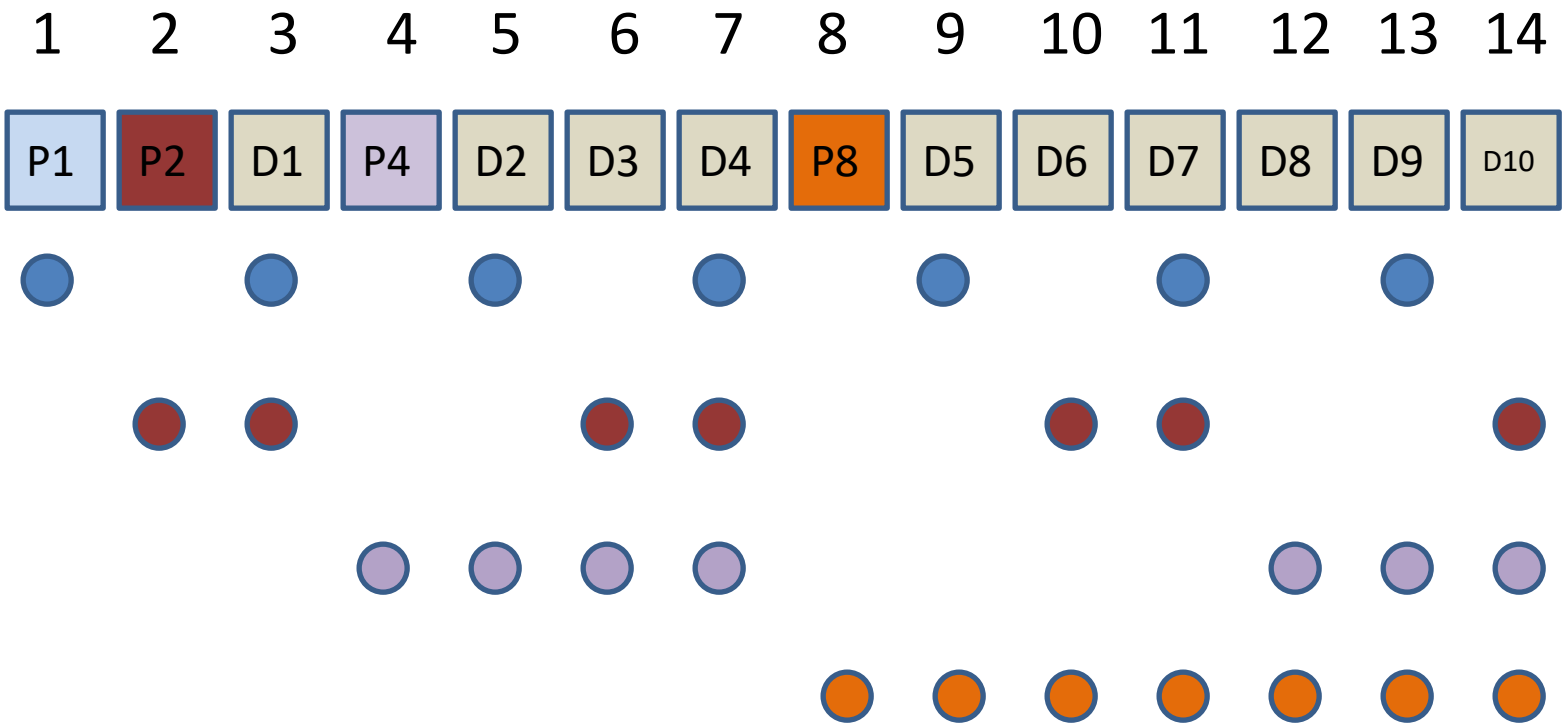
Valid	Invalid
000	100; 010; 001
110	010; 100; 111
011	111; 001; 010
101	001; 111; 100

Error Correction (7/10)

Self-study

- Procedure
  - Parity bits are at positions that are powers of 2 (1, 2, 4, 8, 16, …)
  - All other positions are data bits
  - Each parity bit checks some of the data bits
    - Position 1: Check 1 bit, skip 1 bit (1, 3, 5, 7, 9, 11, …)
    - Position 2: Check 2 bits, skip 2 bits (2, 3, 6, 7, 10, 11, …)
    - Position 4: Check 4 bits, skip 4 bits (4-7, 12-15, 20-23, …)
    - Position 8: Check 8 bits, skip 8 bits (8-15, 24-31, 40-47, …)
  - Set the parity bit accordingly so that total number of 1s in the positions it checks is even.

Error Correction (8/10)



Error Correction (9/10)

Self-study

- Example: Data 10011010
- Insert positions for parity bits:

  _ _ 1 _ 0 0 1 _ 1 0 1 0

  - Position 1: ? _ 1 _ 0 0 1 _ 1 0 1 0 → so ? must be 0
  - Position 2: 0 ? 1 _ 0 0 1 _ 1 0 1 0 → so ? must be 1
  - Position 4: 0 1 1 ? 0 0 1 _ 1 0 1 0 → so ? must be 1
  - Position 8: 0 1 1 1 0 0 1 ? 1 0 1 0 → so ? must be 0

Answer: 0 1 1 1 0 0 1 0 1 0 1 0

Error Correction (10/10)

Self-study

- Suppose 1 error occurred and the received data is:

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

- How to determine which bit is in error?
- Check which parity bits are in error.
  - Answer: parity bits 2 and 8.
- Add the positions of these erroneous parity bits
  - Answer: 2 + 8 = 10. Hence data bit 10 is in error.

Corrected data: 0 1 1 1 0 0 1 0 1 0 1 0

1-7 Gray Code (1/3)

- Unweighted (not an arithmetic code)
- Only a single bit change from one code value to the next.
- Not restricted to decimal digits: n bits → 2^n values.
- Good for error detection.
- Example: 4-bit standard Gray code

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

Gray Code

- Generating a 4-bit standard Gray code sequence.

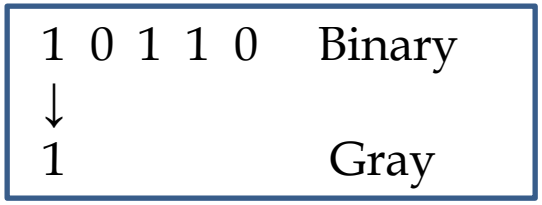
0000	1100
0001	1101
0011	1111
0010	1110
0110	1010
0111	1011
0101	1001
0100	1000

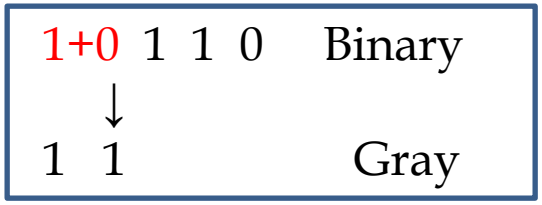
- Questions: How to generate 5-bit standard Gray code sequence? 6-bit standard Gray code sequence?

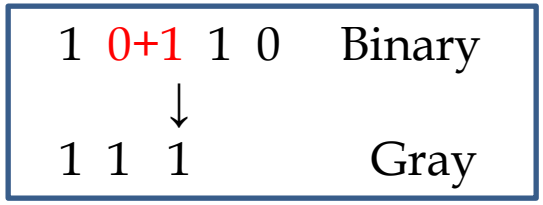
Binary to Gray Conversion

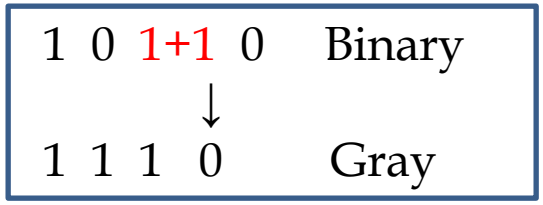
Algorithm:

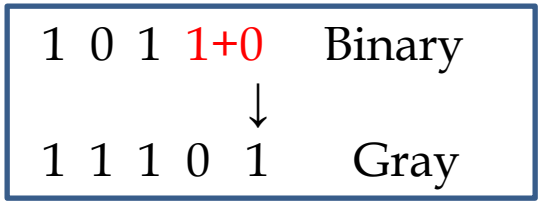
1. Retain the MSB
2. From left to right, add each adjacent pair of binary code bits to get the next Gray code bit, discarding the carry







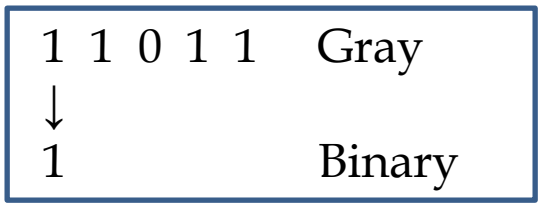


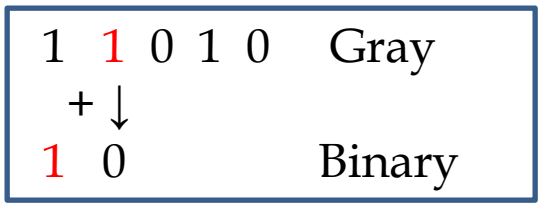


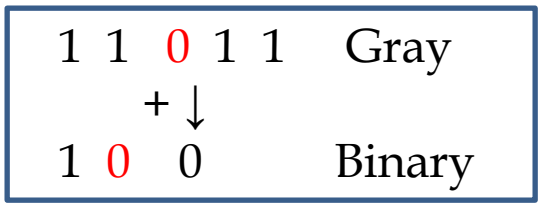
Gray to Binary Conversion

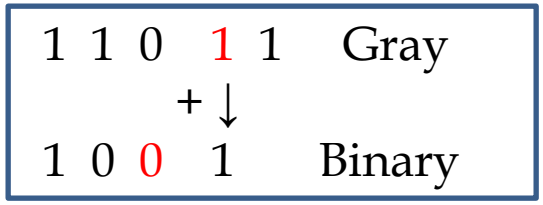
Algorithm:

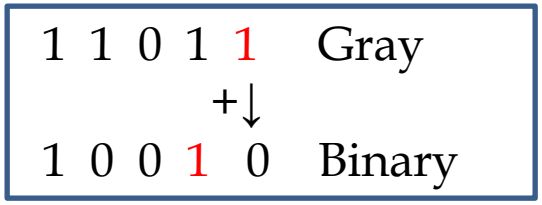
1. Retain the MSB
2. From left to right, add each binary code bit generated to the Gray code bit in the next position, discarding the carry











QUICK REVIEW QUESTIONS

- DLD page 38
  Questions 2-21 to 2-24.

2-21. How to represent $(246)_{10}$ in the following system/code?

a. 10-bit binary

b. b. BCD

c. c. Excess-3

d. d. 2421 code

e. e. 84-2-1 code

QUICK REVIEW QUESTIONS

2-23. Convert **$(101011)_2$** to its corresponding Gray Code value

a. $(101011)_{Gray}$ b. $(010100)_{Gray}$ c. $(110010)_{Gray}$

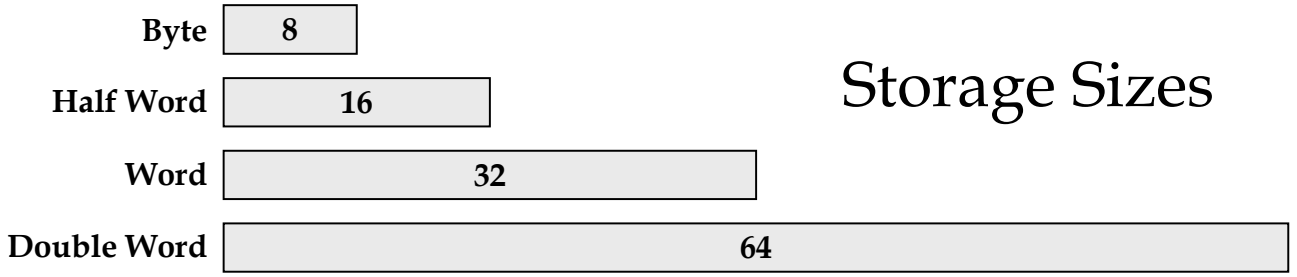
d. $(111110)_{Gray}$ e. $(43)_{Gray}$

2-24. Convert **$(101011)_{Gray}$** to its corresponding binary value.

a. $(101011)_2$ b. $(010100)_2$ c. $(110010)_2$

d. $(111110)_2$ e. $(010101)_2$

Integer Storage Sizes



Storage Sizes

Storage Type	Unsigned Range	Powers of 2
Byte	0 to 255	0 to $(2^8 - 1)$
Half Word	0 to 65,535	0 to $(2^{16} - 1)$
Word	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Double Word	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

What is the largest 20-bit unsigned integer?

Answer: $2^{20} - 1 = 1{,}048{,}575$