BME2322 – Logic Design

The Instructors:

Dr. Görkem SERBES (C317)

gserbes@yildiz.edu.tr

https://avesis.yildiz.edu.tr/gserbes/

Lab Assistants:

Nihat AKKAN

nakkan@yildiz.edu.tr

https://avesis.yildiz.edu.tr/nakkan

LECTURE 1

Assesment

- Midterm : 20%
- Lab and Assignments : 40%
- Final : 40%

Course Outline

1. Digital Computers, Number Systems, Arithmetic Operations, Decimal, Alphanumeric, and Gray Codes
2. Binary Logic, Gates, Boolean Algebra, Standard Forms
3. Circuit Optimization, Two-Level Optimization, Map Manipulation, Multi-Level Circuit Optimization
4. Additional Gates and Circuits, Other Gate Types, Exclusive-OR Operator and Gates, High-Impedance Outputs
5. Implementation Technology and Logic Design, Design Concepts and Automation, The Design Space, Design Procedure, The major design steps
6. Programmable Implementation Technologies: Read-Only Memories, Programmable Logic Arrays, Programmable Array Logic,Technology mapping to programmable logic devices
7. Combinational Functions and Circuits
8. Arithmetic Functions and Circuits
9. Sequential Circuits Storage Elements and Sequential Circuit Analysis
10. Sequential Circuits, Sequential Circuit Design State Diagrams, State Tables
11. Counters, register cells, buses, & serial operations
12. Sequencing and Control, Datapath and Control, Algorithmic State Machines (ASM)
13. Memory Basics

Recommended books

Main course book:

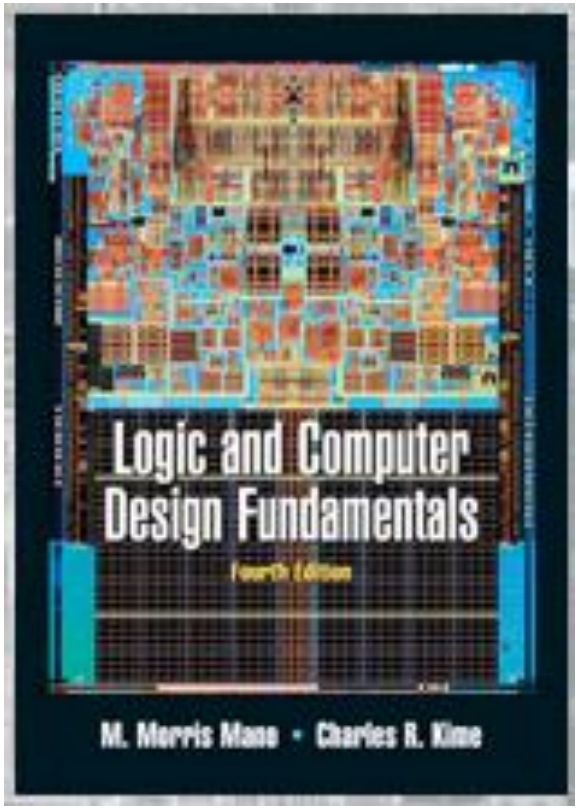
Logic and Computer Design Fundamentals

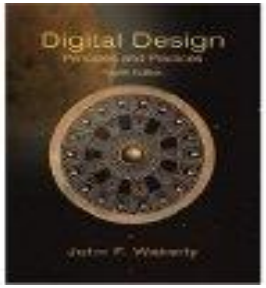
By M. Mano, Charles Kime.

Published by Prentice Hall.

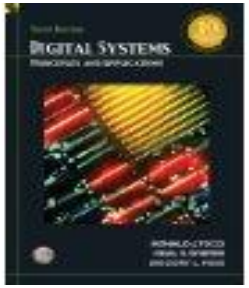
Edition: 4th.

Isbn: 013198926X

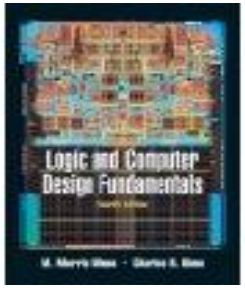




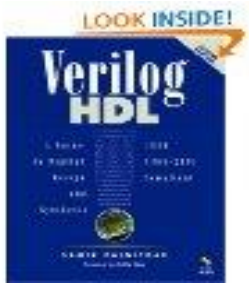
Digital Design: Principles and Practices
by John F. Wakerly



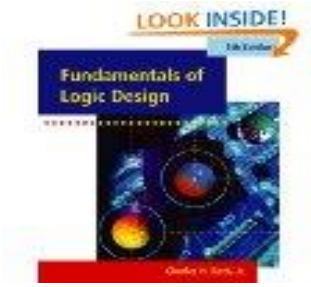
Digital Systems: Principles and Applications
by Ronald Tocci



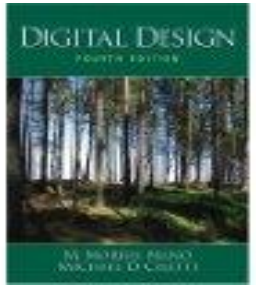
Logic and Computer Design Fundamentals
by M. Morris Mano



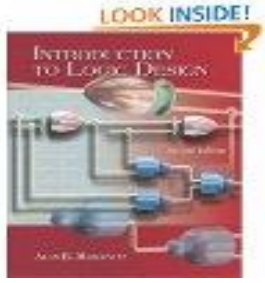
Verilog HDL
by Samir Palnitkar



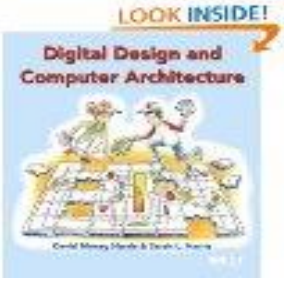
Fundamentals of Logic Design
by Jr., Charles H. Roth



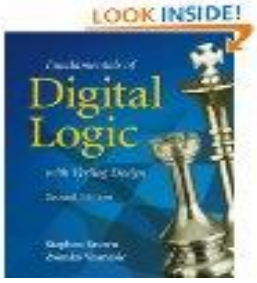
Digital Design
by M. Morris Mano



Introduction to Logic Design
by Alan Marcovitz



Digital Design and Computer Architecture
by David Harris



Fundamentals of Digital Logic with Verilog Design
by Stephen Brown

Rules of the Conduct

- No eating /drinking in class
  - *except water*
- Cell phones must be kept outside of class or switched-off during class
  - *If your cell-phone rings during class or you use it in any way, you will be asked to leave and counted as unexcused absent*.
- No web surfing and/or unrelated use of computers,
  - when computers are used in class or lab.

Rules of the Conduct

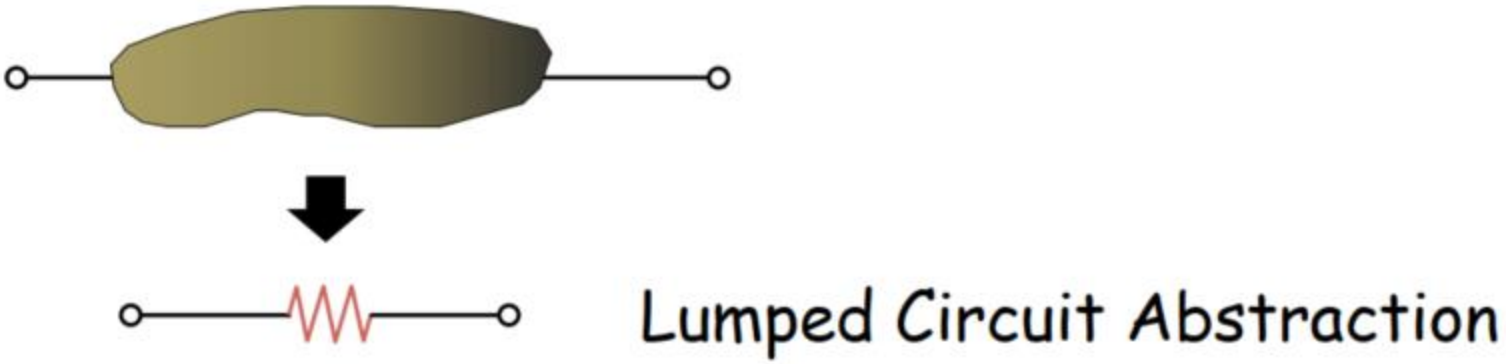
- You are responsible for checking the class web page often for announcements.
- Academic dishonesty and cheating will not be tolerated and will be dealt with according to university rules and regulations
  - *Presenting any work, or a portion thereof, that does not belong to you is considered academic dishonesty*.
- University rules and regulations:
  - http://www.ogi.yildiz.edu.tr/category.php?id=17
  - https://www.yok.gov.tr/content/view/544/230/lang,tr_TR/

Attendance Policy

- The requirement for attendance is **70%**.
  - *Hospital reports are not accepted to fulfill the requirement for attendance.*
  - *The students, who fail to fulfill the attendance requirement, will be excluded from the final exams and the grade of F0 will be given.*

Digital Abstraction

- Discretize matter by observing lumped matter discipline



Lumped Circuit Abstraction

- Analysis tool kit

  KVL/KCL, composition, node, superposition, Thévenin, Norton

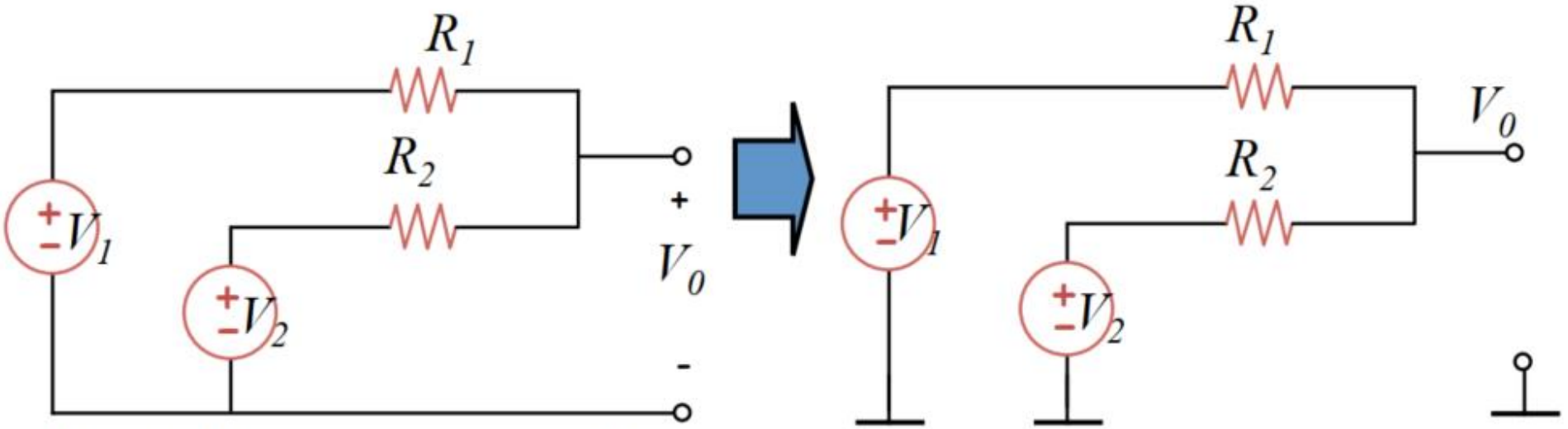
Digital Abstraction (cont.)

In this course we will use Digital Abstraction idea.

But why we need digital signals and systems?

In the past …

Analog signal processing

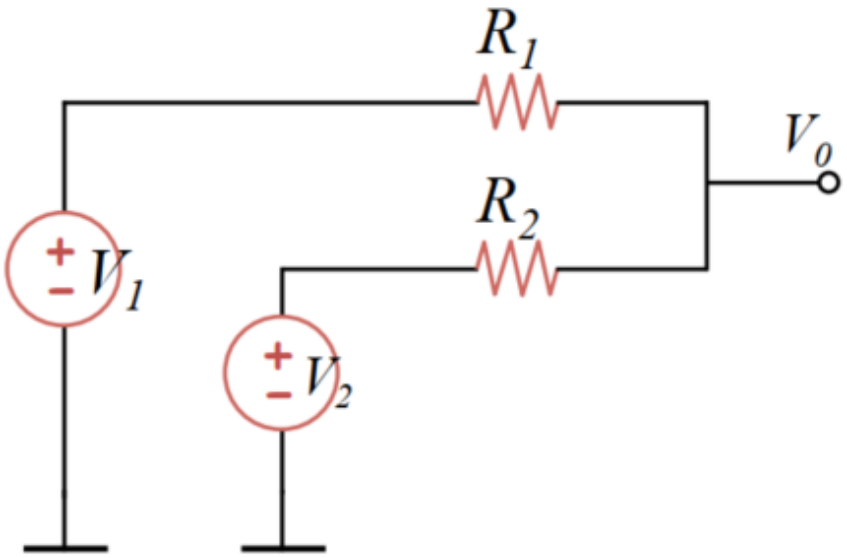


V_1 and V_2 might represent the outputs of two sensors, for e.g.

Shorthand notation (from node method)

Digital Abstraction (cont.)

Analog signal processing



Using Superposition:

$$V_O = \frac{R_2}{R_1 + R_2} V_1 + \frac{R_1}{R_1 + R_2} V_2$$

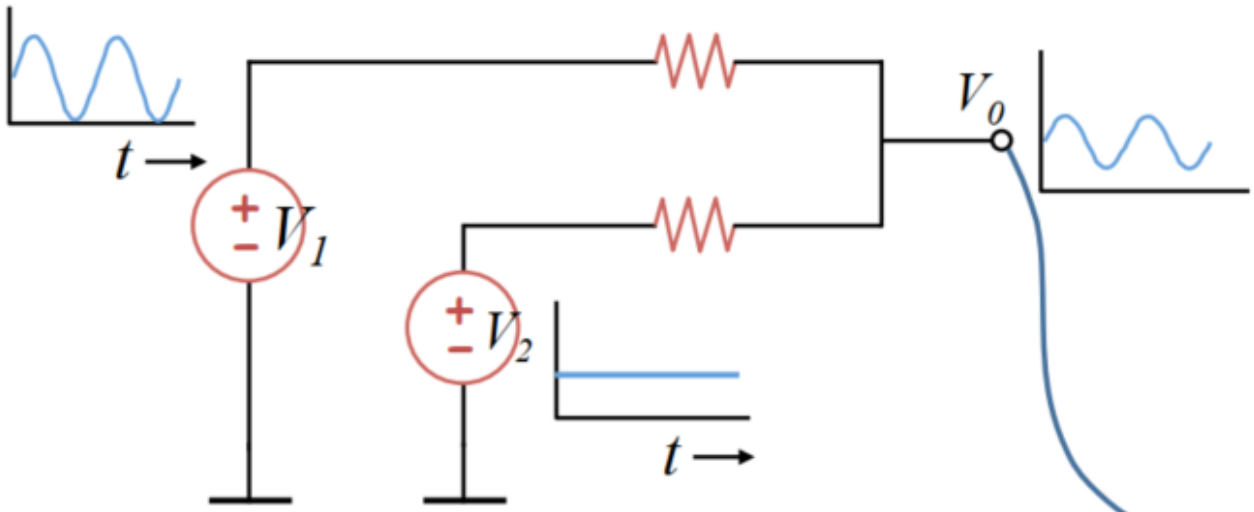
If $R_1 = R_2$

$$V_O = \frac{V_1 + V_2}{2}$$

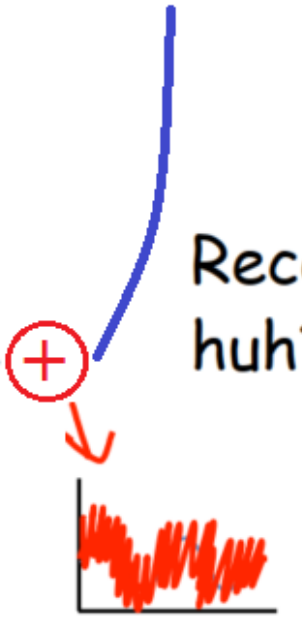
The above is an "adder" circuit.

Digital Abstraction (cont.)

Noise Problem with Analog



Noise added to the signal



Noise hampers our ability to distinguish between small differences in value — e.g. between 3.1V and 3.2V.

Analog Systems lack noise immunity

Digital Abstraction (cont.)

Idea: Value Discretization

Restrict values to be one of two

High	Low
5V	0V
True	False
‘1’	‘0’

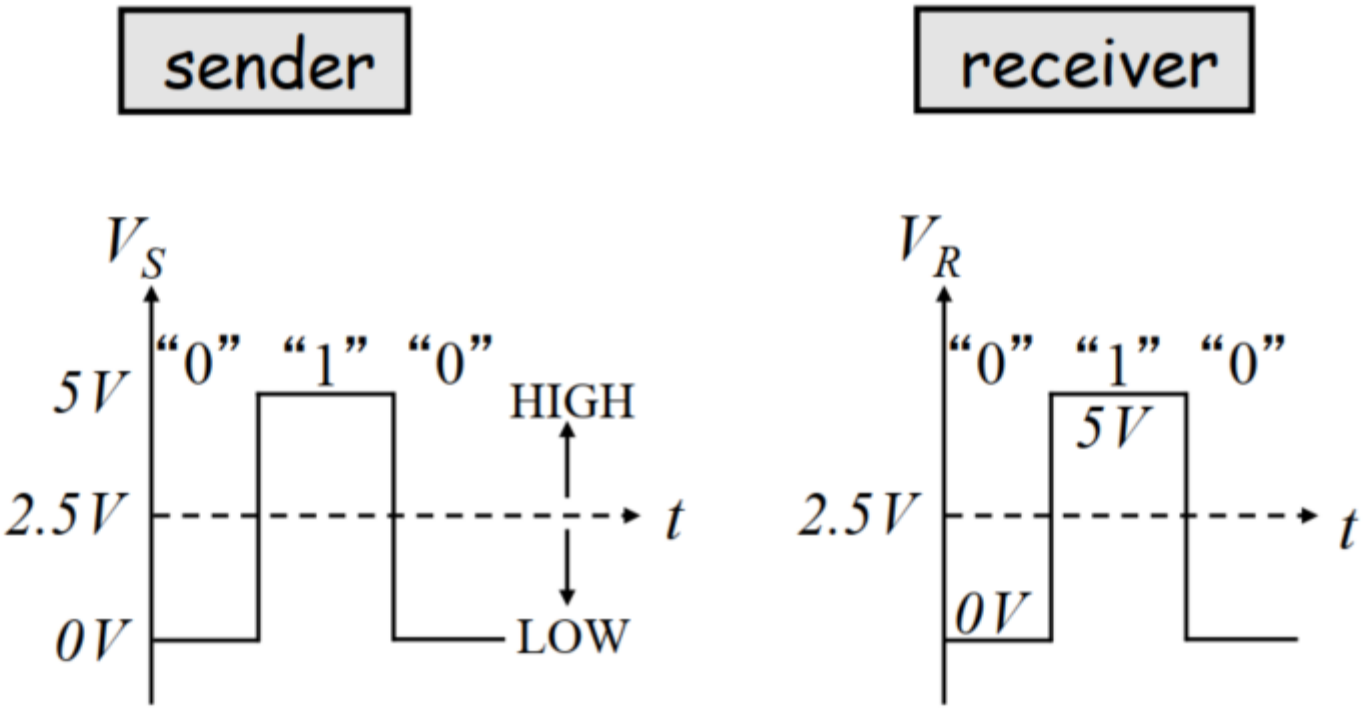
Note: In modern world lower voltage values are used.

...like two digits 0 and 1

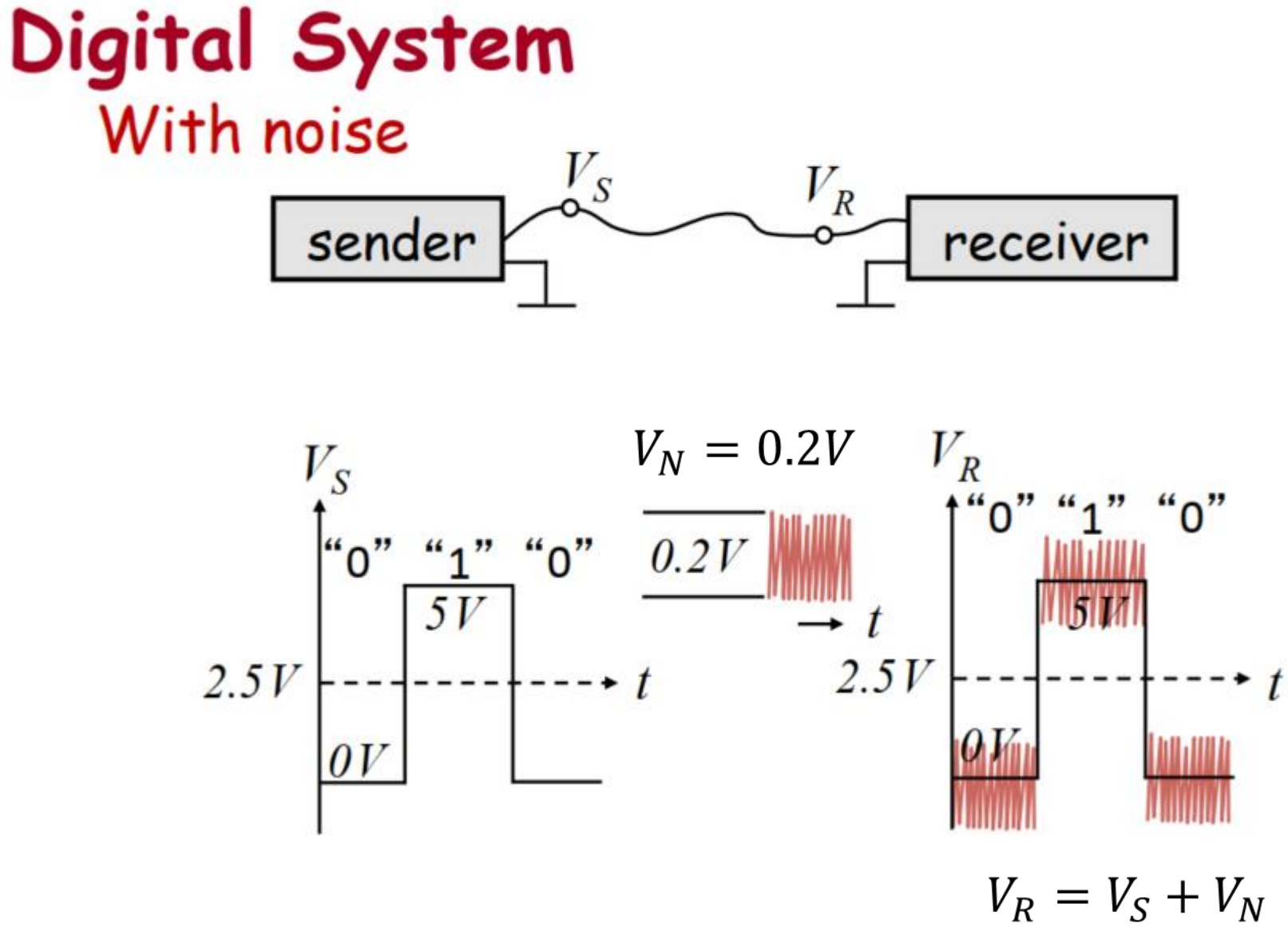
Why is this discretization is useful?

Digital System

Ideal Case

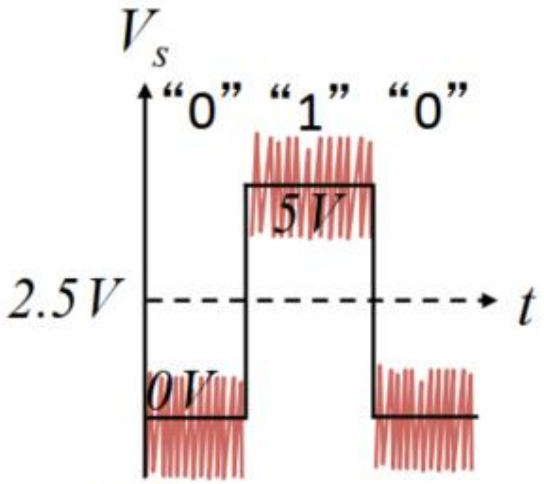


Why is this discretization is useful? (cont.)



Why is this discretization is useful? (cont.)

Digital System

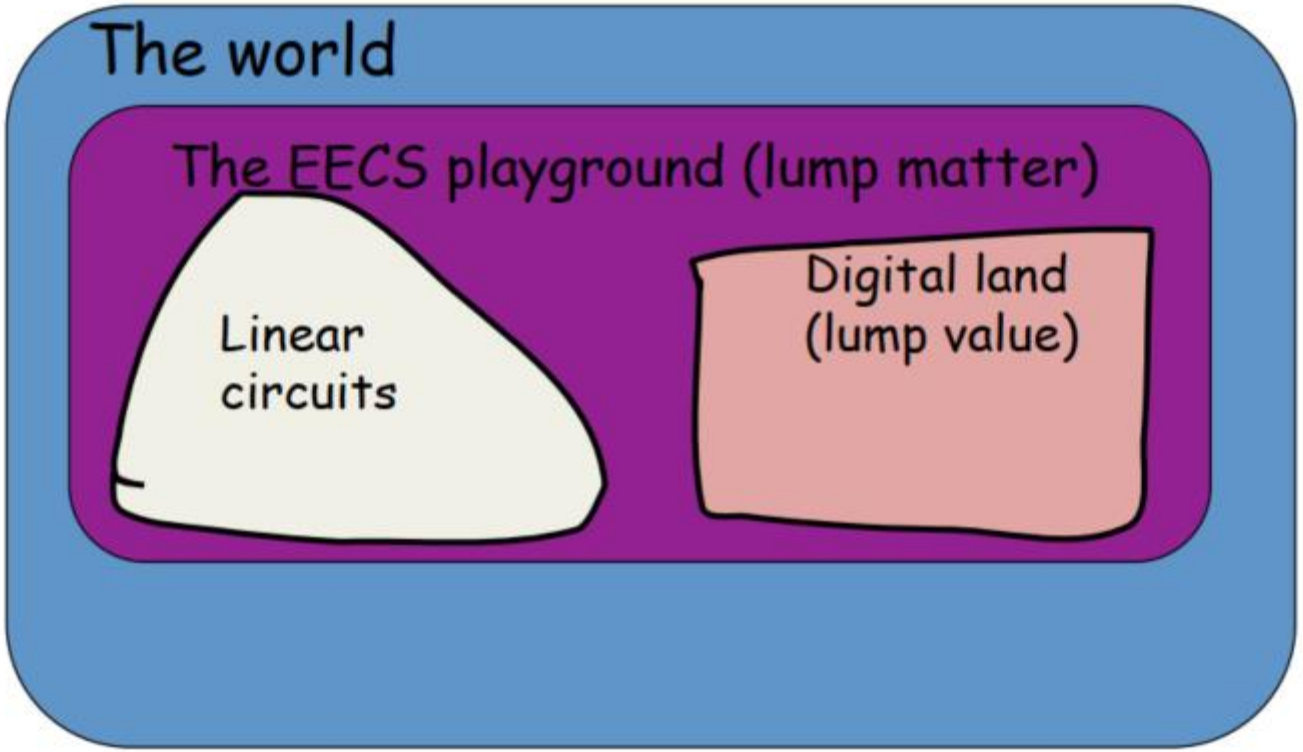


Better noise immunity → Lots of "noise margin"

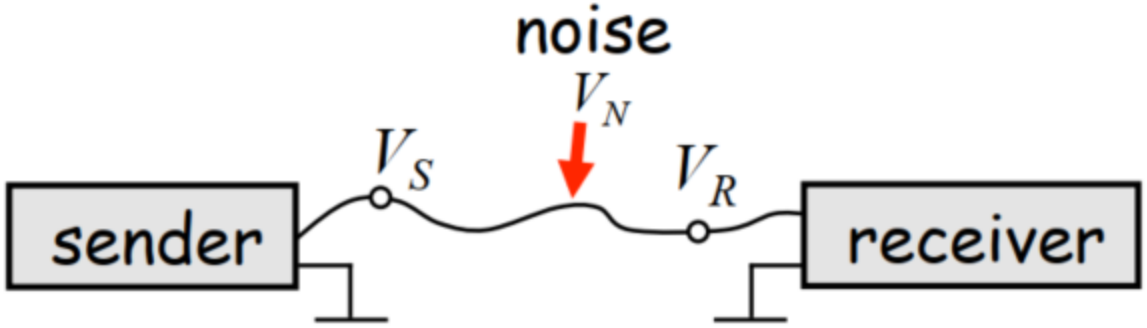
For "1": noise margin $5V \text{ to } 2.5V = 2.5V$

For "0": noise margin $0V \text{ to } 2.5V = 2.5V$

The Big Picture

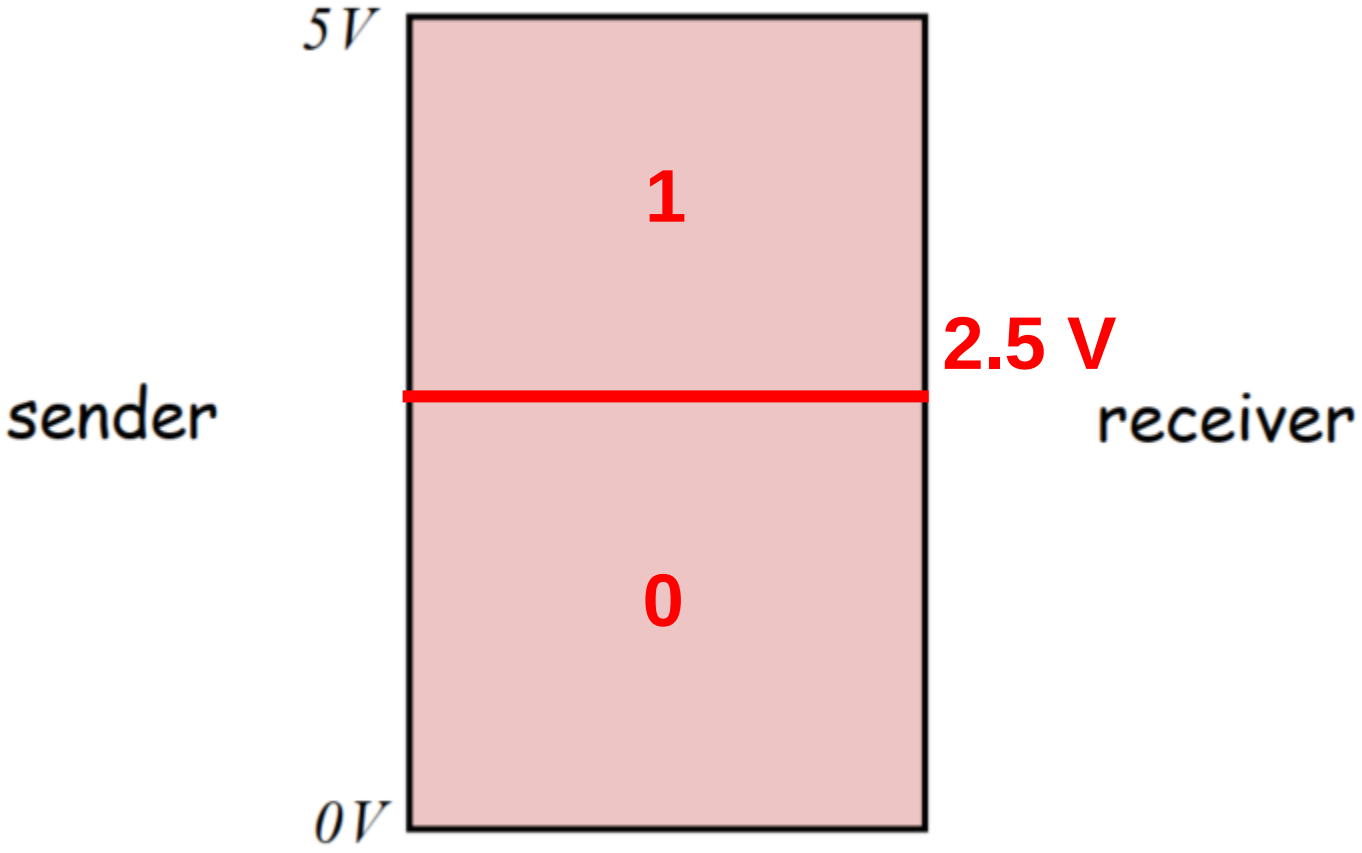


Sender-Receiver Contract



Static Discipline

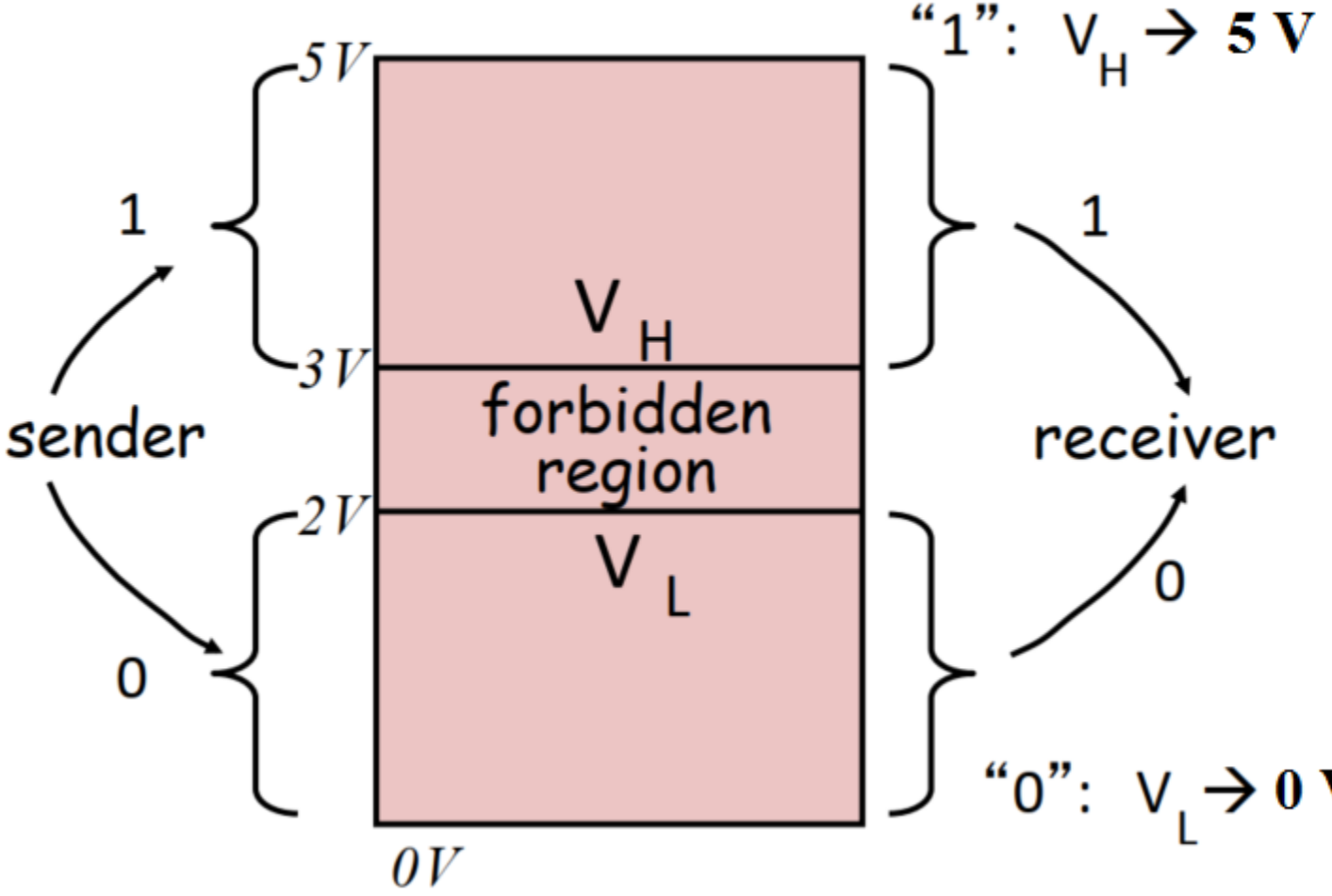
Voltage Thresholds and Logic Values



But, but, but … What about 2.5V?

Static Discipline (Cont.)

"No Man's Land" or Forbidden Region



Where's the noise margin?

What if the sender sent 1: V_H

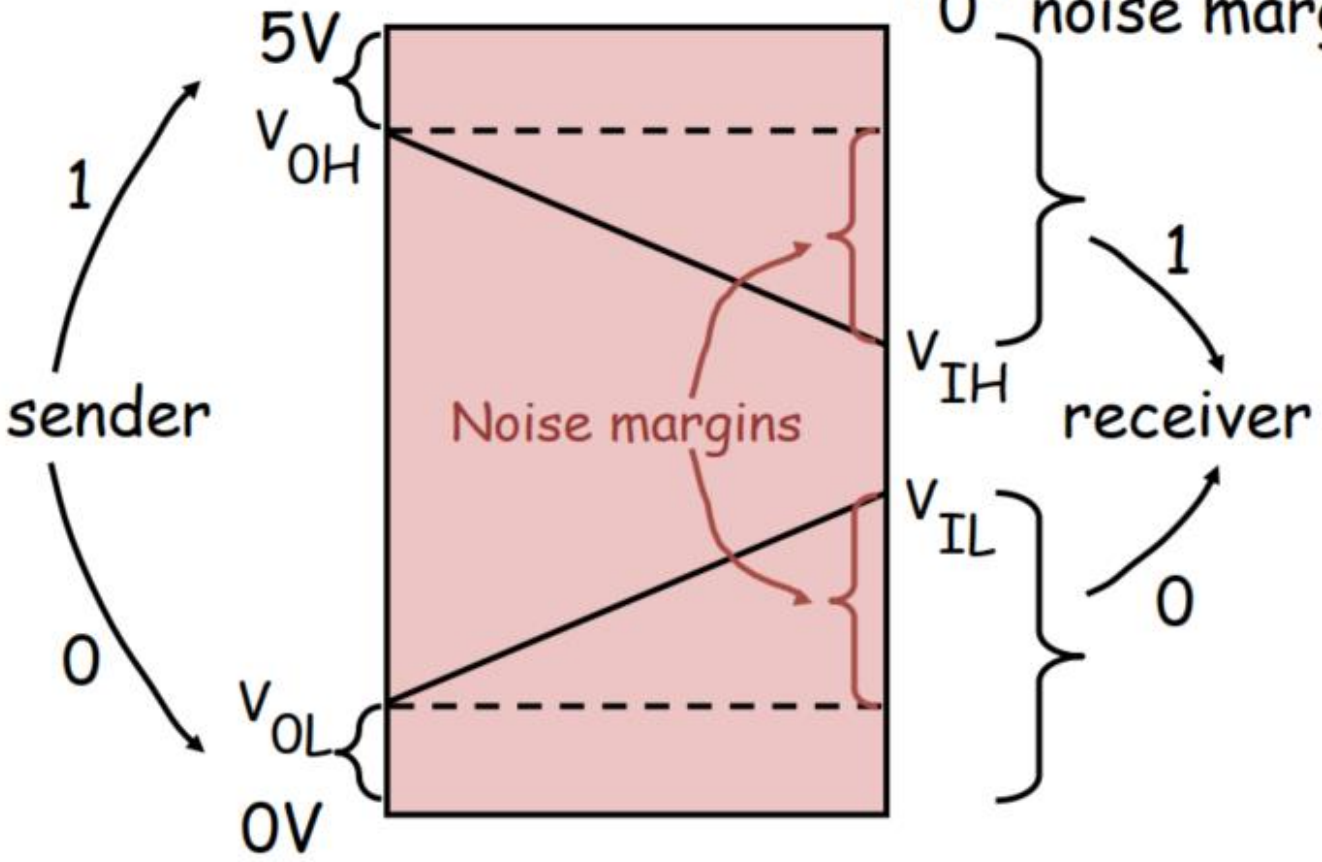
???

Static Discipline (Cont.)

Noise Margins

"1" noise margin: $V_{OH} - V_{IH}$

"0" noise margin: $V_{IL} - V_{OL}$



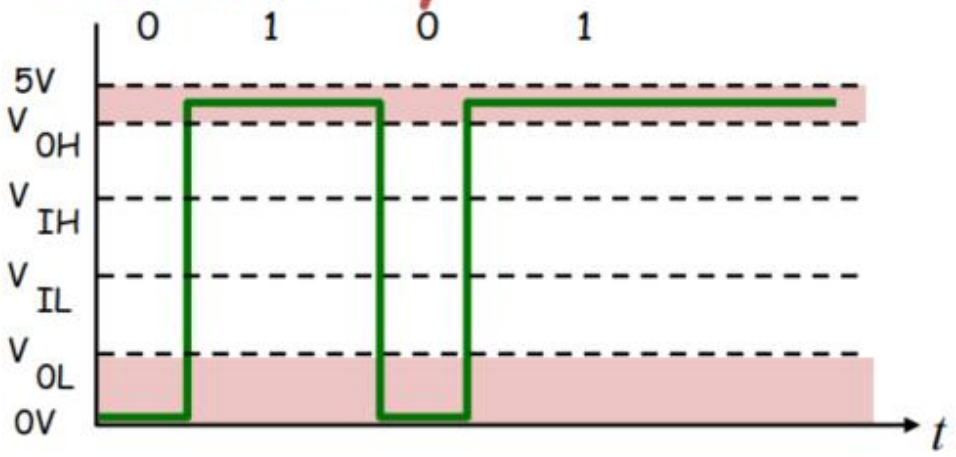
Together, the V_{OH}, V_{IH}, V_{OL}, V_{IL} thresholds define a discipline or standard that digital devices follow so they can talk to each other

Tougher Standards for Sender

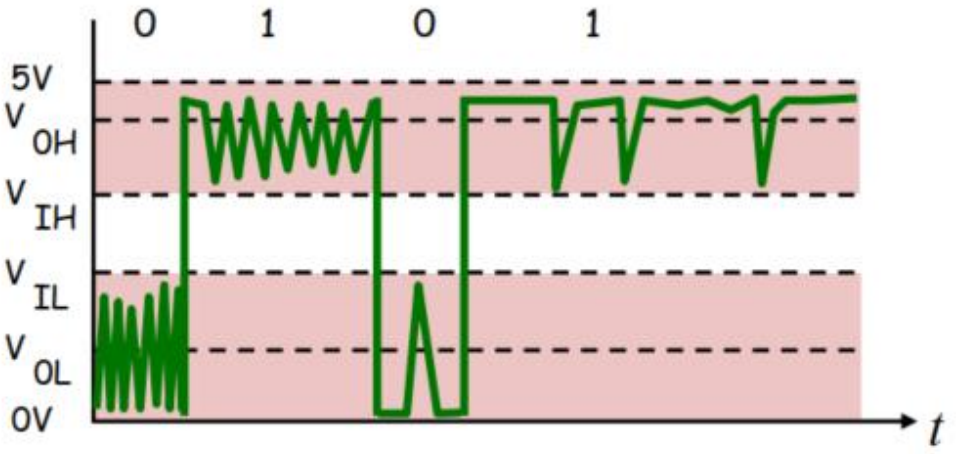
Static Discipline (Cont.)

Noise Immunity

sender



receiver



Digital systems follow static discipline: if inputs to the digital system meet valid input thresholds, then the system guarantees its outputs will meet valid output thresholds.

Digital Information

Processing Digital Signals

Recall, we have only two values —

1, 0 ⇨ Map naturally to logic: T, F

⇨ Can also represent numbers

Information can be stored by using 1 and 0s.
Boolean Logic is used to process this information.

Digital Information

Processing Digital Signals

Boolean Logic

⇨ If **X** is true and **Y** is true
Then **Z** is true, else **Z** is false.

Z = X AND Y

Z = X . Y ⇨ Boolean Equation

X ⇨ F T (0, 1)
Y
Z

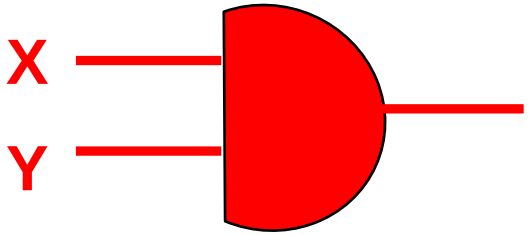
Binary variables

X, Y and Z are digital signals
0 or 1

Digital Information

Processing Digital Signals

$Z = X \cdot Y$ **Boolean Equation**



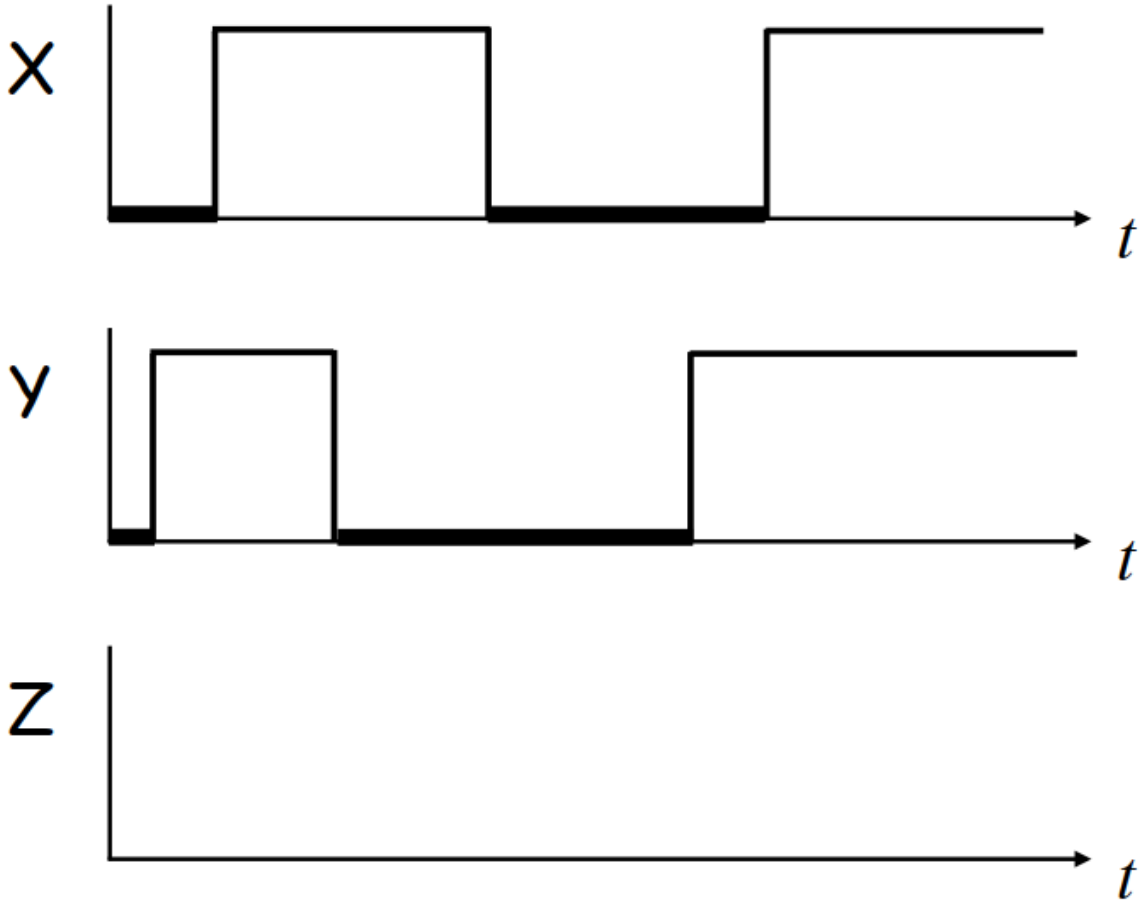
AND Gate

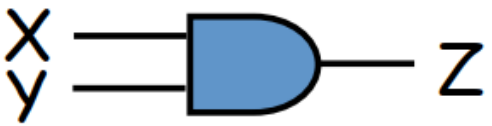
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table
Enumerates input combinations

Digital Information

What is the Output Of This Gate?

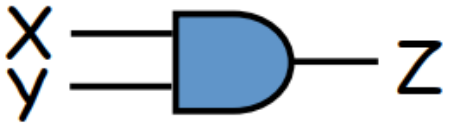




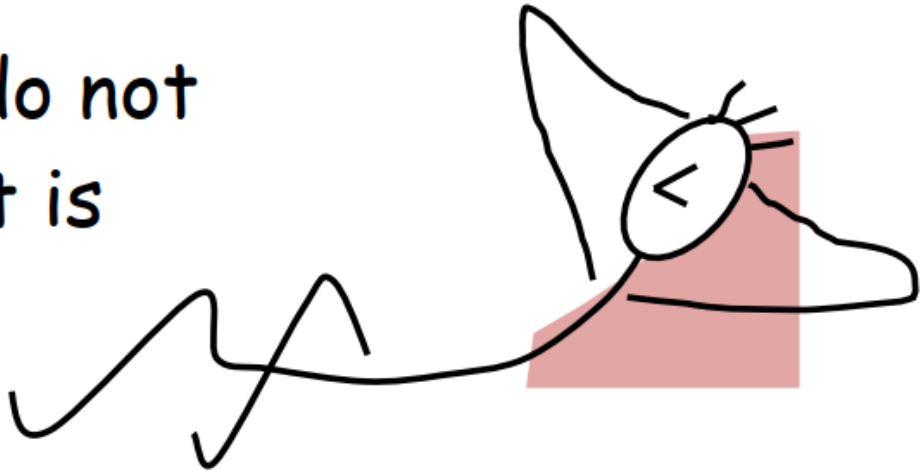
$Z = X \cdot y$

Combinational Gate Abstraction

- Adheres to static discipline
- Outputs are a function of inputs alone.



Digital logic designers do not have to care about what is inside a gate.



What is Information?

Information, *n*. Data communicated or received that resolves uncertainty about a particular fact or circumstance.

Example: you receive some data about a card drawn at random from a 52-card deck. Which of the following data conveys the most information? The least?

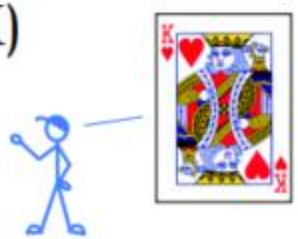
\# of possibilities remaining

13 A. The card is a heart

51 B. The card is not the Ace of spades

12 C. The card is a face card (J, Q, K)

1 D. The card is the "suicide king"



Which of the following data conveys the *most* information about a playing card chosen randomly from a 52-card deck?

Which of the following data conveys the *least* information about a playing card chosen randomly from a 52-card deck?

What is Information? (Cont.)

Quantifying Information
(Claude Shannon, 1948)

Given discrete random variable X

- N possible values: $x_1, x_2, \ldots, x_N$
- Associated probabilities: $p_1, p_2, \ldots, p_N$

Information received when learning that choice was x_i:

$$I(x_i) = \log_2\left(\frac{1}{p_i}\right)$$

Information is measured in bits (binary digits) = number of 0/1's required to encode choice(s)

$1/p_i$ is proportional to the uncertainty of choice x_i.

What is Information? (Cont.)

Information Conveyed by Data

Even when data doesn't resolve all the uncertainty

$$I(\text{data}) = \log_2\left(\frac{1}{p_{\text{data}}}\right) \qquad \text{e.g., } I(\text{heart}) = \log_2\left(\frac{1}{13/52}\right) = 2 \text{ bits}$$

Common case: Suppose you're faced with N equally probable choices, and you receive data that narrows it down to M choices. The probability that data would be sent is M·(1/N) so the amount of information you have received is

$$I(\text{data}) = \log_2\left(\frac{1}{M \cdot (1/N)}\right) = \log_2\left(\frac{N}{M}\right) \text{ bits}$$

What is Information? (Cont.)

Example: Information Content

Examples:

- information in one coin flip:

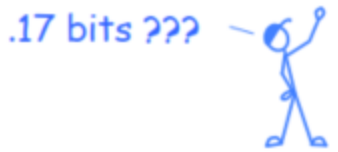
  N= 2 M= 1 Info content= $\log_2(2/1) = 1$ bit

- card drawn from fresh deck is a heart:

  N= 52 M= 13 Info content= $\log_2(52/13) = 2$ bits

- roll of 2 dice:

  N= 36 M= 1 Info content= $\log_2(36/1) = 5.17$



What is Information? (Cont.)

Probability & Information Content

data	p_{data}	$\log_2(1/p_{data})$
a heart	13/52	2 bits
not the Ace of spades	51/52	0.028 bits
a face card (J, Q, K)	12/52	2.115 bits
the "suicide king"	1/52	5.7 bits

Shannon's definition for information content lines up nicely with my intuition: I get more information when the data resolves more uncertainty about the randomly selected card.

Example 1

A) You're given a standard deck of 52 playing cards that you start to turn face up, card by card. So far as you know, they're in completely random order.

- How many new bits of information do you get when the first card is flipped over and you learn exactly which card it is?

  Information (in bits):

- The fifth card?

  Information (in bits):

- The last card?

  Information (in bits):

B) Z is an unknown N-bit binary number (N > 3). You are told that the first three bits of Z are 011. How many bits of information about Z have you been given?

Information (in bits):

Example 1 (Cont.)

A) You're given a standard deck of 52 playing cards that you start to turn face up, card by card. So far as you know, they're in completely random order.

- How many new bits of information do you get when the first card is flipped over and you learn exactly which card it is?

  Information (in bits): 5.7

- The fifth card?

  Information (in bits): 5.585

- The last card?

  Information (in bits): 0

B) Z is an unknown N-bit binary number (N > 3). You are told that the first three bits of Z are 011. How many bits of information about Z have you been given?

Information (in bits): 3

Explanation
Before the first card was flipped over there are 52 choices for what we'll see on the first flip. Turning the first card over narrows the choice down to a single card, so we've received log2(52/1) bits of information.

- The fifth card?

Information (in bits): 5.585 ✔ **Answer:** 5.585

Explanation
After flipping over 4 cards, there are 48 choices for the next card, so flipping over the fifth card gives us log2(48/1) bits of information.

- The last card?

Information (in bits): 0 ✔ **Answer:** 0

Explanation
If all but one card has been flipped over, we know ahead of time what the final card has to be so we don't receive any information from the last flip. Using the formula, there is only 1 "choice" for the card before the card is flipped and we have the same "choice" afterwards, so, we receive log2(1/1) = 0 bits of information.

B) Z is an unknown N-bit binary number (N > 3). You are told that the first three bits of Z are 011. How many bits of information about Z have you been given?

Information (in bits): 3 ✔ **Answer:** 3

Explanation
Since we were told about 3 bits of Z it would make sense intuitively that we've been given 3 bits of information! Turning to the formulas: there are 2^N N-bit binary numbers and 2^{N-3} N-bit binary numbers that begin with 011. So we've been given $log_2\left(2^N/2^{N-3}\right) = log_2\left(2^3\right)$ = 3 bits of information (whew!).

What is Entropy?

Entropy

In information theory, the entropy H(X) is the average amount of information contained in each piece of data received about the value of X:

$$H(X) = E(I(X)) = \sum_{i=1}^{N} p_i \cdot \log_2\left(\frac{1}{p_i}\right)$$

Example: X={A, B, C, D}

$choice_i$	p_i	$log_2(1/p_i)$
"A"	1/3	1.58 bits
"B"	1/2	1 bit
"C"	1/12	3.58 bits
"D"	1/12	3.58 bits

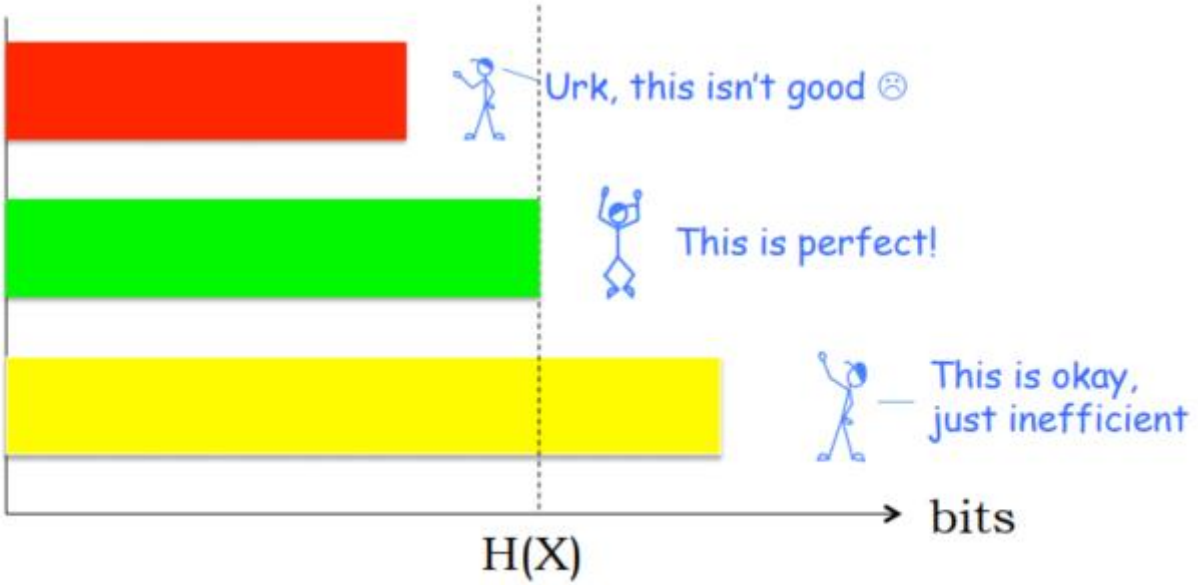
H(X) = (1/3)(1.58) + (1/2)(1) + 2(1/12)(3.58)
= 1.626 bits

What is Entropy? (Cont.)

Meaning of Entropy

Suppose we have a data sequence describing the values of the random variable X.

Average number of bits used to transmit choice



Example 2

Please compute the entropy associated with each the following random variables.

A) The flip of an unfair coin, where $p(\text{heads}) = 0.999$ and $p(\text{tails}) = 0.001$.

Entropy (in bits):

B) The random choice of one of the 16 hex digits, where the probability of choosing any particular digit is $1/16$.

Entropy (in bits):

C) The quiz grade of a randomly-chosen student, where in the 100-student class the grade distribution was 27 A's, 38 B's, 23 C's, 8 D's, and 4 F's.

Entropy (in bits):

Example 2 (Cont.)

Please compute the entropy associated with each the following random variables.

A) The flip of an unfair coin, where $p(\text{heads}) = 0.999$ and $p(\text{tails}) = 0.001$.

Entropy (in bits): 0.0114

B) The random choice of one of the 16 hex digits, where the probability of choosing any particular digit is $1/16$.

Entropy (in bits): 4

C) The quiz grade of a randomly-chosen student, where in the 100-student class the grade distribution was 27 A's, 38 B's, 23 C's, 8 D's, and 4 F's.

Entropy (in bits): 2.0054

A) The flip of an unfair coin, where $p(\text{heads}) = 0.999$ and $p(\text{tails}) = 0.001$.

Entropy (in bits): 0.0114 ✔ **Answer:** .0114

Explanation

Entropy = $p(\text{heads}) \cdot \log_2\left(\frac{1}{p(\text{heads})}\right) + p(\text{tails}) \cdot \log_2\left(\frac{1}{p(\text{tails})}\right)$

= 0.999*log2(1/.999) + 0.001*log2(1/.001) = .0114

B) The random choice of one of the 16 hex digits, where the probability of choosing any particular digit is $1/16$.

Entropy (in bits): 4 ✔ **Answer:** 4

Explanation

Entropy = $\sum_{i=0}^{15} p_i \cdot \log_2\left(\frac{1}{p_i}\right)$

= $\frac{1}{16} * log2\left(\frac{1}{\frac{1}{16}}\right) + \frac{1}{16} * log2\left(\frac{1}{\frac{1}{16}}\right) + \ldots + \frac{1}{16} * log2\left(\frac{1}{\frac{1}{16}}\right)$

= $16 * \frac{1}{16} * log2(16) = 4$

C) The quiz grade of a randomly-chosen student, where in the 100-student class the grade distribution was 27 A's, 38 B's, 23 C'

Entropy (in bits): 2.0054 ✔ **Answer:** 2.005

Explanation

Entropy = 0.27*log2(100/27) + 0.38*log2(100/38) + 0.23*log2(100/23) + 0.08*log2(100/8) + 0.04*log2(100/4)

Encoding

An encoding is an *unambiguous* mapping between bit strings and the set of possible data.

Encoding for each symbol				Encoding for "ABBA"
A	B	C	D	
00	01	10	11	00 01 01 00
01	1	000	001	01 1 1 01
0	1	10	11	0 1 1 0

✗

Encodings as Binary Trees

It's helpful to represent an unambiguos encoding as a binary tree with the symbols to be encoded as the leaves. The labels on the path from the root to the leaf give the encoding for that leaf.

Encoding

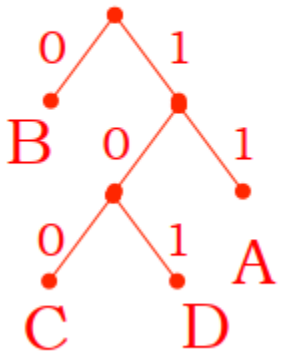
B↔0

A↔11

C↔100

D↔101

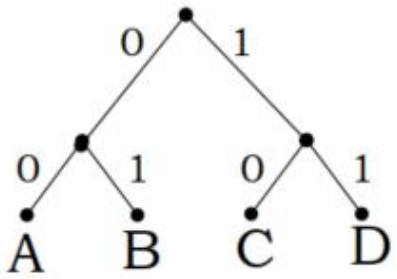
Binary tree



Encoding

Fixed-length Encodings

If all choices are equally likely (or we have no reason to expect otherwise), then a fixed-length code is often used. Such a code will use at least enough bits to represent the information content.



All leaves have the same depth!

Note that the entropy for N equally-probable symbols is

$$\sum_{i=1}^{N}\left(1/N\right)\log_2\left(\frac{1}{1/N}\right) = \log_2(N)$$

Examples:

Fixed-length are often a little inefficient...

- 4-bit binary-coded decimal (BCD) digits $\log_2(10)=3.322$
- 7-bit ASCII for printing characters $\log_2(94)=6.555$

Encoding (Cont.)

Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{N-1} 2^i b_i$$

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$
$= 1024 + 512 + 256 + 128 + 64 + 16$
$= 2000$

Smallest number: 0 Largest number: $2^N - 1$

Encoding (Cont.)

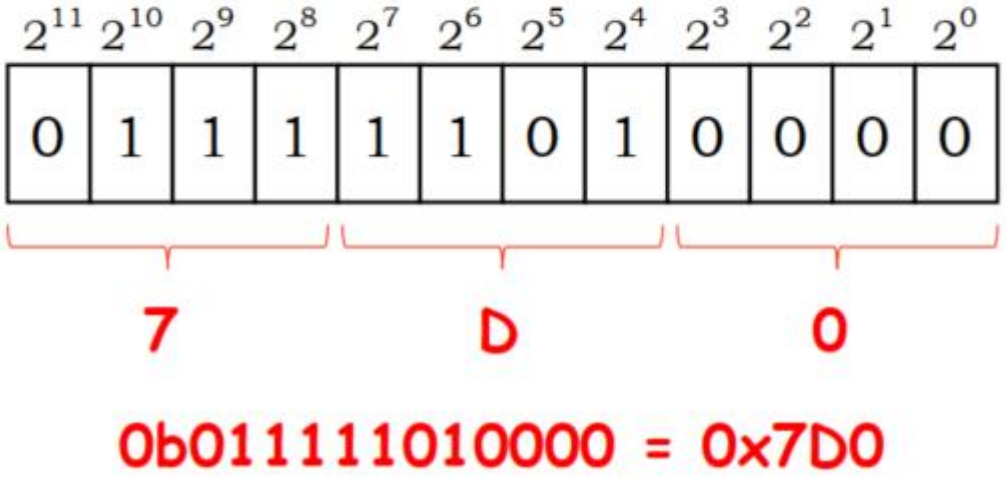
Hexademical Notation

Long strings of binary digits are tedious and error-prone to transcribe, so we usually use a higher-radix notation, choosing the radix so that it's simple to recover the original bits string.

A popular choice is transcribe numbers in base-16, called hexadecimal, where each group of 4 adjacent bits are representated as a single hexadecimal digit.

Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F



Example 3

For the following problems, your answer should be specified as an integer.

How many bits are needed to encode the 10 decimal digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}?

How many bits are needed to encode the 86 ASCII characters?

Example 3 Cont.

How many bits are needed to encode the 10 decimal digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}?

4 ✔ **Answer:** 4

Explanation
The amount of information in 10 decimal digits is $log_2(10)$ = 3.322. So the number of bits required to encode that is the next whole number of bits which is 4 bits.

How many bits are needed to encode the 86 ASCII characters?

7 ✔ **Answer:** 7

Explanation
The amount of information in 86 ASCII characters is $log_2(86)$ = 6.426 is less than 7 bits.

Encoding (Cont.)

Octal Numbering System (Octal Notation)

- Eight symbols: 0, 1, 2, 3, 4, 5, 6, 7
  - Notice that we no longer use 8 or 9
- Base comparison:
  - Base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12…
  - Base 8: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14…
- Example: What is 7061_8 in base 10?
  - $7061_8 = (7 \times 8^3) + (0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 3633_{10}$

Encoding (Cont.)

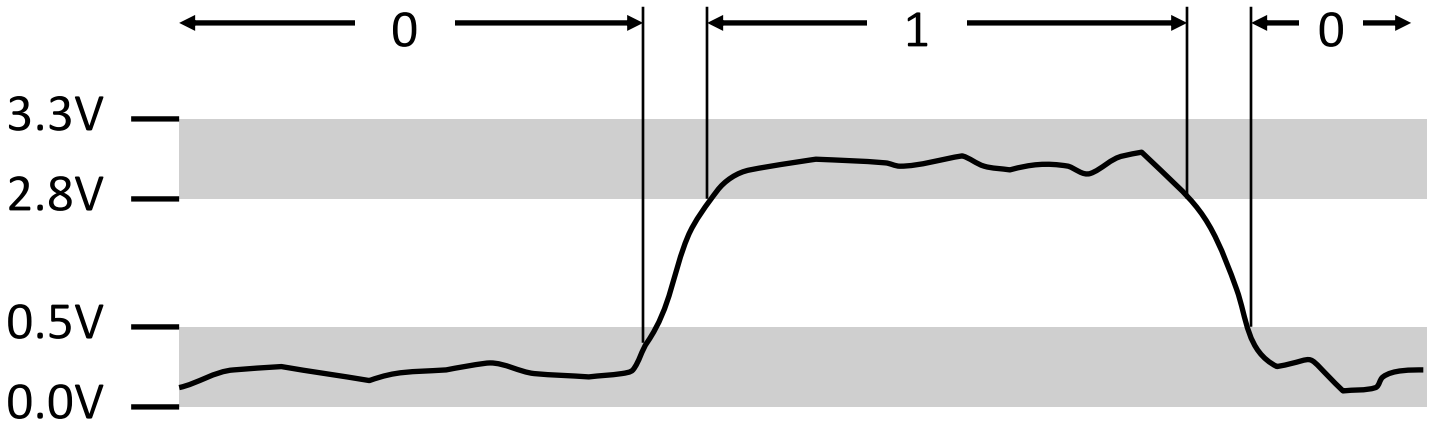
Binary and Hexadecimal

- Binary is base 2
  - Symbols: 0, 1
  - Convention: $2_{10} = 10_2 = 0b10$
- Example: What is 0b110 in base 10?
  - $0b110 = 110_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6_{10}$
- Hexadecimal (hex, for short) is base 16
  - Symbols? 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, …?
  - Convention: $16_{10} = 10_{16} = 0x10$
- Example: What is 0xA5 in base 10?
  - $0xA5 = A5_{16} = (10 \times 16^1) + (5 \times 16^0) = 165_{10}$

Encoding (Cont.)

Aside: Why Base 2?

- Electronic implementation
  - Easy to store with bi-stable elements
  - Reliably transmitted on noisy and inaccurate wires



Encoding (Cont.)

Base Comparison

- Why does all of this matter?
  - *Humans* think about numbers in **base 10**, but *computers* "think" about numbers in **base 2**
  - Binary encoding is what allows computers to do all of the amazing things that they do!

Base 10	Base 2	Base 16
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Encoding (Cont.)

Numerical Encoding

- **AMAZING FACT: You can represent *anything* countable using numbers!**
  - Need to agree on an encoding
  - Kind of like learning a new language

- Examples:
  - Decimal Integers: 0→0b0, 1→0b1, 2→0b10, etc.
  - English Letters: CSE→0x435345, yay→0x796179
  - Emoticons: 😀 0x0, 😥 0x1, 😎 0x2, 😇 0x3, 😈 0x4, 🙋‍♀️ 0x5

Encoding (Cont.)

English Letters: CSE→0x435345, yay→0x796179

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Encoding (Cont.)

Binary Encoding

- With N binary digits, how many "things" can you represent?
  - Need N binary digits to represent n things, where $2^N \geq n$
  - Example: 5 binary digits for alphabet because $2^5 = 32 > 26$

- A binary digit is known as a bit
- A group of 4 bits (1 hex digit) is called a nibble
- A group of 8 bits (2 hex digits) is called a byte (octet)
- A group of 16 bits (4 hex digits) is called a word
  - 1 bit → 2 things, 1 nibble → 16 things, 1 byte → 256 things

https://www.binaryhexconverter.com/hex-to-ascii-text-converter ----→ Hex to Ascii Converter
https://www.rgbtohex.net/hextorgb/ ---→ Hex to RGB converter
https://gregstoll.com/~gregstoll/floattohex/ ---→ Hex to real number converter

So What's It Mean?

- *A sequence of bits can have many meanings!*

- Consider the hex sequence 0x4E6F21
  - Common interpretations include:
    - The decimal number 5140257
    - The characters "No!"
    - The background color of this slide
    - The real number 7.203034×10^{-39}

- It is up to the program/programmer to decide how to interpret the sequence of bits

Encoding (Cont.)

Binary Encoding – Files and Programs

- At the lowest level, all digital data is stored as bits!

- Layers of abstraction keep everything comprehensible
  - Data/files are groups of bits interpreted by program
  - Program is actually groups of bits being interpreted by your CPU

- Computer Memory Demo
  - Can try to open files using a text editor
  - From vim: `%!xxd`

Example 4

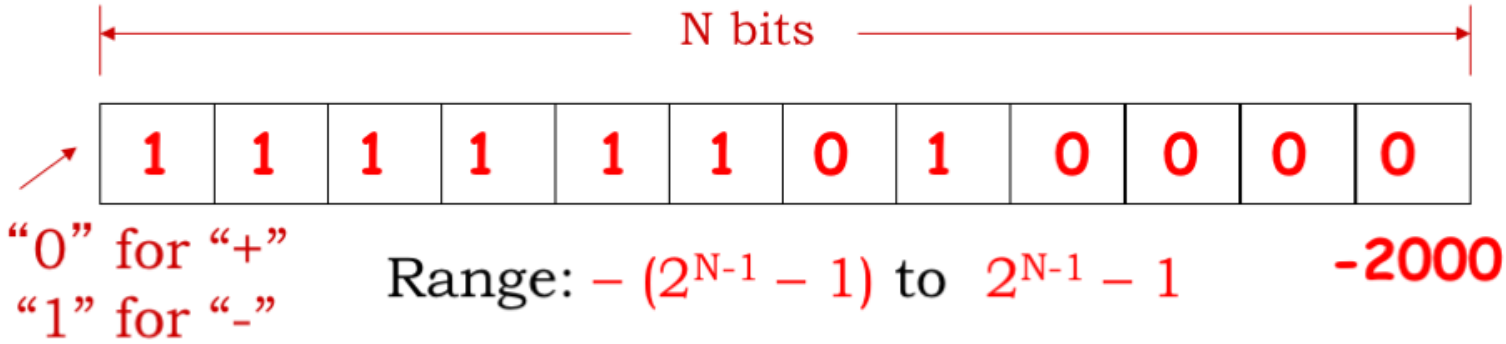
- $(25.625)_{10} = (\quad)_2$
- $(25.625)_{10} = (\quad)_8$
- $(25.625)_{10} = (\quad)_{16}$

Encoding (Cont.)

Encoding Signed Integers

We use a signed magnitude representation for decimal numbers, encoding the sign of the number (using "+" and "-") separately from its magnitude (using decimal digits).

We could adopt that approach for binary representations:

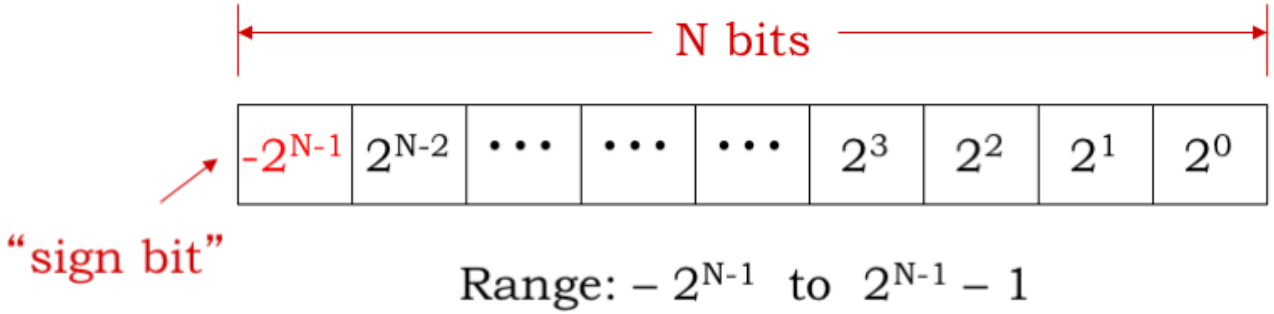


But: two representations for 0 (+0, -0) and we'd need different circuitry for addition and subtraction

Two's Complement Encoding

Two's Complement Encoding

In a two's complement encoding, the high-order bit of the N-bit representation has negative weight:



- Negative numbers have "1" in the high-order bit
- Most negative number: 10…0000 -2^{N-1}
- Most positive number: 01…1111 $+2^{N-1} - 1$
- If all bits are 1: 11…1111 -1
- If all bits are 0: 00…0000 0

Sum and Difference of Binary Numbers

Sum of 2 bits:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = **10** (current step bit: 0,
carry to the next step: 1)

Difference of 2 bits:

0 - 0 = 0

1 - 0 = 1

1 - 1 = 0

0 - 1 = **11** (current step bit: 1
borrow to the next step: 1)

Sum example:

```
      1 1 1   → carry
  1 0 1 0 0 1 0 1 = A
+ 1 0 1 0 1 1 1 = B
-----------------
  1 1 1 1 1 1 0 0
```

Difference example:

```
  1 0 1 0 0 1 0 1 = A
- 1 0 1 0 1 1 1 = B
-----------------
  1 0 1 1 1 1     → borrow
  0 1 0 0 1 1 1 0
```

Example 5

Exercise

```
  10011011 = A
+  1010011 = B
------------
```

```
  10011001 = A
-  1010011 = B
------------
```

Exercise (solution)

```
  0 0 1 0 0 1 1   → carry
  10011011 = A
+  1010011 = B
------------
  11101110
```

```
  10011001 = A
-  1010011 = B
  1     1 1     → borrow
------------
  01000110
```

Two's Complement Encoding (Cont.)

More Two's Complement

- Let's see what happens when we add the N-bit values for -1 and 1, keeping an N-bit answer:

```
  11...1111
+ 00...0001
  0000000
```

Just use ordinary binary addition, even when one or both of the operands are negative. 2's complement is perfect for N-bit arithmetic!

- To compute B-A, we'll just use addition and compute B+(-A). But how do we figure out the representation for -A?

$A+(-A) = 0 = 1 + -1$

$-A = (-1 - A) + 1$

$= \sim A + 1$

$$\begin{array}{r} 1 \\ -A_i \\ \hline \sim A_i \end{array}$$

To negate a two's complement value: bitwise complement and add 1.

Example 6

Convert the following decimal numbers to 6 bit 2's complement representation binary numbers. Provide the binary numbers using the format 0bXXXXXX.

15 = 0b

-15 = 0b

6 = 0b

-6 = 0b

21 = 0b

-21 = 0b

Example 6 (Cont.)

Convert the following decimal numbers to 6 bit 2's complement representation binary numbers. Provide the binary numbers using the format 0bXXXXXX.

15 = 0b 001111 ✔

-15 = 0b 110001 ✔

6 = 0b 000110 ✔

-6 = 0b 111010 ✔

21 = 0b 010101 ✔

-21 = 0b 101011 ✔

Example 6 (Cont.)

Explanation

Binary numbers are represented in the same manner as decimal numbers with the least significant bit representing the 2^0 position, the next bit to the left being the 2^1 position, the next 2^2 and so on. So to represent the positive number 15 which is equal to $8 + 4 + 2 + 1 = 2^3 + 2^2 + 2^1 + 2^0 = 0b001111$. There are 1's in the 0, 1, 2, and 3 positions and 0's in the 4, and 5 positions. Similary, $6 = 4 + 2 = 2^2 + 2^1 = 0b000110$ indicating that the only positions that are non-zero are the 1 and 2 bits. Finally, $21 = 16 + 4 + 1 = 2^4 + 2^2 + 2^0 = 0b010101$ with 1's in the 0, 2, and 4 positions and 0's elsewhere.

In order to convert these numbers to a negative numbers, the way to do that in binary is to first flip all the bits and then add 1.

So -15 = 0b110000 +1 = 0b110001.

-6 = 0b111001 + 1 = 0b111010.

Finally, -21 = 0b101010 + 1 = 0b101011.

Example 7

Convert the following integers to 6-bit 2's complement binary numbers. Binary numbers are prefixed with the string `0b` to indicate that you are specifying a binary number.

- 5 = 0b
- 23 = 0b
- -12 = 0b

Example 7 (Cont)

Convert the following integers to 6-bit 2's complement binary numbers. Binary numbers are prefixed with the string `0b` to indicate that you are specifying a binary number.

- 5 = 0b 000101 ✔
- 23 = 0b 0010111 ✔
- -12 = 0b 110100 ✔

Example 7 (Cont)

Convert the following integers to 6-bit 2's complement binary numbers. Binary numbers are prefixed with the string `0b` to indicate that you are specifying a binary number.

- 5 = 0b _____ **Answer:** 101

  Explanation
  5 = 4 + 1 = $2^2 + 2^0$, so you have 1's in the 0 and 2 positions which correspond to the rightmost bit, and the third bit from the right. The other positions have 0's.

- 23 = 0b _____ **Answer:** 010111

  Explanation
  23 = 16 + 4 + 2 + 1 = $2^4 + 2^2 + 2^1 + 2^0$, so you have 1's in the 0, 1, 2, and 4 positions and 0's elsewhere, where 0 is the rightmost bit.

- -12 = 0b _____ **Answer:** 110100

  Explanation
  12 = 8 + 4 = 0b001100. To get -12, you flip all the bits, and add 1. Flipping all the bits results in 0b110011. Remember that you must use the correct number of bits in your representation which in this case is 6. Now adding 1, results in 0b110100.

Example 8

Octal and hexadecimal representation:

For the following problems, use 24 bit precision when answering the problems.

Convert the following integers to octal (base 8) representation using octal digits 0, 1, 2, 3, 4, 5, 6, and 7. Octal numbers should be prepended with the string `0` to indicate that you are specifying an octal number.

- 21 = O

Convert the following integers to hexadecimal representation. Hexadecimal numbers should be prepended with the string `0x` to indicate that you are specifying a hexadecimal number.

- 73 = 0x
- -7 = 0x

Example 8 (Cont.)

Octal and hexadecimal representation:

For the following problems, use 24 bit precision when answering the problems.

Convert the following integers to octal (base 8) representation using octal digits 0, 1, 2, 3, 4, 5, 6, and 7. Octal numbers should be prepended with the string `0` to indicate that you are specifying an octal number.

- 21 = O 00000025 ✔

Convert the following integers to hexadecimal representation. Hexadecimal numbers should be prepended with the string `0x` to indicate that you are specifying a hexadecimal number.

- 73 = 0x 00049 ✔
- -7 = 0x FFFFF9 ✔

For the following problems, use 24 bit precision when answering the problems.

Convert the following integers to octal (base 8) representation using octal digits 0, 1, 2, 3, 4, 5, 6, and 7. Octal numbers should be prepended with the string `0` to indicate that you are specifying an octal number.

- 21 = O [ ] **Answer:** 25

  Explanation
  21 = 16 + 5 = $2^4 + 2^2 + 2^0$ = 0b000...010101 = 0b 000 000 000 000 000 000 010 101 = O00000025. An octal character represents 3 binary bits so the least significant octal character is 5, and the next octal character is a 2.

Convert the following integers to hexadecimal representation. Hexadecimal numbers should be prepended with the string `0x` to indicate that you are specifying a hexadecimal number.

- 73 = 0x [ ] **Answer:** 49

  Explanation
  73 = 64 + 8 + 1 = $2^6 + 2^3 + 2^0$ = 0b000...01001001 = 0× 0000 0000 0000 0000 0100 1001 = 0×000049. A hex character represents 4 binary bits, so the least significant hex character is 9, and then next one is a 4.

- -7 = 0x [ ] **Answer:** FFFFF9

  Explanation
  7 = 0×000007 = 0b 0000 0000 0000 0000 0000 0111. To get -7, you flip all the bits, and add 1. Flipping all the bits results in 0b 1111 1111 1111 1111 1111 1000. Remember that you must use the correct number of bits in your representation which in this case is 24. Now adding 1, converts the bottom four bits to 1001 in binary, and converting back to hex results in 0xFFFFF9.

Example 9

Perform the following addition problems using 6-bit 2's complement arithmetic. Provide your answer using the format $0bXXXXXX$ if the problem can be solved using 6 bit 2's complement representation. Otherwise provide the answer "overflow".

$0b001101$
$0b001010$
0b

$0b001111$
$0b101110$
0b

$0b011011$
$0b111010$
0b

$0b111010$
$0b110001$
0b

$0b011111$
$0b001100$
0b

Example 9 (Cont.)

Perform the following addition problems using 6-bit 2's complement arithmetic. Provide your answer using the format $0bXXXXXX$ if the problem can be solved using 6 bit 2's complement representation. Otherwise provide the answer "overflow".

$0b001101$
$0b001010$
0b

010111 ✔

$0b001111$
$0b101110$
0b

111101 ✔

$0b011011$
$0b111010$
0b

010101 ✔

$0b111010$
$0b110001$
0b

101011 ✔

$0b011111$
$0b001100$
0b

overflow ✔

$0b001101$
$0b001010$ 0b

010111 ✔ **Answer:** 010111

Explanation
Binary addition follows the same rules as decimal addition where you begin by adding the numbers in the rightmost (least significant) column, keeping track of any carry's into the next column, and repeat this process moving to the left until all column additions have been computed.
For this problem:
Column 0: 1 + 0 =1
Column 1: 0 + 1 = 1
Column 2: 1 + 0 = 1
Column 3:1 + 1 = 0 Carry = 1
Column 4: 0 + 0 + 1 (Carry from column 3) =1
Column 5: 0 + 0 = 0
So sum = 0b010111 = $2^4 + 2^2 + 2^1 + 2^0 = 16 + 4 + 2 + 1 = 23$ as expected since the two binary numbers being added were 13 ($= 2^3 + 2^2 + 2^0 = 8 + 4 + 1$) + 10 ($= 2^3 + 2^1 = 8 + 2$).

$0b001111$
$0b101110$ 0b

111101 ✔ **Answer:** 111101

Explanation
Binary addition follows the same rules as decimal addition where you begin by adding the numbers in the rightmost (least significant) column, keeping track of any carry's into the next column, and repeat this process moving to the left until all column additions have been computed.
For this problem:
Column 0: 1 + 0 = 1
Column 1: 1 + 1 = 0 and Carry = 1
Column 2: 1 + 1 + 1 (Carry from Column 1) = 1 and Carry = 1
Column 3: 1 + 1 + 1 (Carry from Column 2) = 1 and Carry = 1
Column 4: 0 + 0 + 1 (Carry from Column 3) = 1
Column 5: 0 + 1 = 1
So the result is 0b111101.
To double check ourselves, we verify that the two numbers being added are:
0b001111: 8 + 4 + 2 + 1 = 15.
0b101110: Flipping all the bits and adding one which results in 0b010001 +1 = 0b010010 = 16 + 2 = 18.
So the problem we are trying to solve is 15 + (-18) which we expect to result in -3. Looking at our result of 0b111101, flipping all the bits and adding 1 we get 0b000010 + 1 = 0b000011 = 3. So as expected our answer was -3.

0b011011
0b111010 0b

010101 ✔ **Answer:** 010101

Explanation
Binary addition follows the same rules as decimal addition where you begin by adding the numbers in the rightmost (least significant) column, keeping track of any carry's into the next column, and repeat this process moving to the left until all column additions have been computed.
For this problem:
Column 0: 1 + 0 = 1
Column 1: 1 + 1 = 0 and Carry = 1
Column 2: 0 + 0 + 1 (from Carry) = 1
Column 3: 1 + 1 = 0 and Carry = 1
Column 4: 1 + 1 + 1 (from Carry) = 1 and Carry = 1
Column 5: 0 + 1 + 1(from Carry) = 0 and Carry = 1
The last carry is dropped because we are using 6-bit 2's complement, so we do not want to go into the 7th column.
So sum = 0b010101. To double check ourselves, we verify that the two numbers being added are:
0b011011 = 16 + 8 + 2 + 1 = 27.
0b111010: Flipping all the bits gives you 0b000101. Ading one results in 0b000110 = 6. Therefore, the second number is a -6.
27 + -6 = 21.
Looking at our answer:
0b010101 = 16 + 4 + 1 = 21. Everything checks out.

0b111010
0b110001 0b

101011 ✔ **Answer:** 101011

Explanation
Binary addition follows the same rules as decimal addition where you begin by adding the numbers in the rightmost (least significant) column, keeping track of any carry's into the next column, and repeat this process moving to the left until all column additions have been computed.
For this problem:
Column 0: 0 + 1 = 1
Column 1: 1 + 0 = 1
Column 2: 0 + 0 = 0
Column 3: 1 + 0 = 1
Column 4: 1 + 1 = 0 and Carry = 1
Column 5: 1 + 1 + 1 (Carry from column 4) = 1 and Carry = 1
The last carry is dropped because we are using 6-bit 2's complement, so we do not want to go into the 7th column.
So sum = 0b101011. To double check ourselves, we verify that the two numbers being added are:
0b111010: Flipping all the bits gives you 0b000101. Adding one results in 0b000110 = 6. Therefore, the first number is a -6.
0b110001: Flipping all the bits gives you 0b001110. Ading one results in 0b001111 = 15. Therefore, the second number is a -15.
-6 + -15 = -21.
Looking at our answer:
0b101011: Flipping all the bits gives you 0b010100. Adding one results in 0b010101 = 21. Therefore, the sum is -21 as expected.

$$\begin{array}{r} 0b011111 \\ 0b001100 \\ \hline \end{array}$$ 0b

overflow ✔ **Answer:** overflow

Explanation

In this problem, the numbers being added are:
0b011111: 16 + 8 + 4 + 2 + 1 = 31 and
0b001100: 8 + 4 = 12.
If you add these two numbers up you get a number that is larger than what can be represented using 6 bit 2's complement which is $2^5 - 1 = 31$. Therefore, the correct answer is overflow.
Another way to see that overflow occurs is that if you try to add the two binary numbers up, you get the following:
Column 0: 1 + 0 = 1
Column 1: 1 + 0 = 1
Column 2: 1 + 1 = 0 and Carry = 1
Column 3: 1 + 1 + 1 (from Carry) = 1 and Carry = 1
Column 4: 1 + 0 + 1 (from Carry) = 0 and Carry = 1
Column 5: 0 + 0 + 1 (from Carry) = 1
Resulting in 0b101011.
This indicates that you took two positive numbers (their most significant bit was 0) added them up and ended up with a negative numbers (because now the most significant bit is a 1). Since this cannot occur, something else must have gone wrong, and the answer is that overflow occured that that 6 bits is not enough to perform this addition in 2's complement arithmetic.