Error Detection, Error Correction, & Encoding (Fixed vs. Variable Length)

1. Error Detection

Why it matters:
Data can get messed up during transmission or storage (due to noise, hardware faults, etc.). So we need ways to check if something went wrong.

- Parity Bit:
  - Add 1 extra bit to your data to make the number of 1s either even (even parity) or odd (odd parity).
  - Used to detect single-bit errors.

  Example – Even Parity:
  Send: 00000110 + 0 → total 1s = 2 → even
  Receive: 00000100 + 0 → total 1s = 1 → odd → error detected

  Only detects, not fixes the error.

- Hamming Distance:
  - Measures how many bits are different between two binary strings.
  - Bigger distance = better error handling.

  Example:
  10101010 vs 10001010 → 1 bit difference → distance = 1

  Rules:
  - Distance 2 → can detect 1-bit errors
  - Distance 3 → can detect 2-bit, correct 1-bit errors

2. Error Detection and Correction

- Hamming Code (7,4):
  - Takes 4 bits of data and adds 3 parity bits = 7 bits total
  - Uses even parity
  - Parity bits placed in positions 1, 2, and 4 (powers of 2)
  - Each parity bit checks certain bits
  - If a single bit flips, you can locate and correct it

  Used in:
  - ECC RAM
  - Some network and storage systems

  Example:

Data = 1100
→ Add parity bits
→ Receiver rechecks them
→ If some fail → locate bad bit → flip it

3. Encoding: Fixed vs. Variable Length

- Fixed-Length Encoding:
  - All symbols use the same number of bits
  - Simple, fast, and memory-friendly

  Examples:
  - DNA bases (A, C, G, T): 2 bits each
  - ASCII: 8 bits per character

  Pros:
  - Easy decoding
  - Consistent storage

  Cons:
  - Wastes space if symbol frequencies aren't equal

- Variable-Length Encoding:
  - Give shorter codes to frequent symbols, longer to rare ones
  - Saves space when some symbols appear more often

  Issue:
  - Can be ambiguous

  Fix:
  - Use prefix codes (no code is a prefix of another)

  Example:
  a = 0
  b = 10
  c = 110
  d = 111
  → 110100101 = "badc"

4. Huffman's Algorithm (for variable-length encoding)

Purpose:
  - Builds the most efficient prefix code based on symbol frequency

Steps:
1. Count how often each symbol shows up
2. Put them into a min-priority queue
3. Merge two least frequent nodes
4. Repeat until one tree remains
5. Left = 0, Right = 1 -> trace to get codes

Example:

Frequencies:

a: 45, b: 13, c: 12, d: 16, e: 9, f: 5

Resulting Codes:

a = 0

b = 101

c = 100

d = 111

e = 1101

f = 1100

Quick Recap:

Concept:	What it Does:	Key Info:
Parity Bit	Detects single-bit errors	Even/odd check
Hamming Distance	Bit difference measure	Bigger = better error handling
Hamming Code (7,4)	Detects & corrects 1-bit errors	Overlapping parity bits
Fixed-Length Encoding	Same bits for all symbols	Simple but can waste space
Variable-Length Encoding	Shorter codes for common stuff	Must be prefix codes
Huffman Coding	Builds optimal prefix tree	Based on symbol frequency

Graphs, Truth Table, Function(4 inputs), Draw Circuit

Character	Frequency	P(x)
D	5	
E	20	
V	25	
P	~~[illegible]~~	
A	~~[illegible]~~	
R	~~[illegible]~~	
K	~~[illegible]~~	
(space)		

DEV PARKAR

learn.circuitverse.org

{1} = 9 =

12 11 =

1001

Minute notes
numbers

1, 2, 6

2 | 1, 2, 4, 5, 7

0 | 4, 2, 3, 5, 6, 7

A — 12

2

31 — 16

7

~~Magnitude
numbers~~

~~0 | 1 — 5~~

~~0 | 2 — 1, 2, 3, 4~~

~~0 | 0 = 7~~

~~0 | 8 = 7~~

i/p O/P

A A'

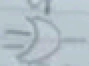
0 → 1

1 → 0

$y = \overline{x}$

AND

OR



NAND

XOR

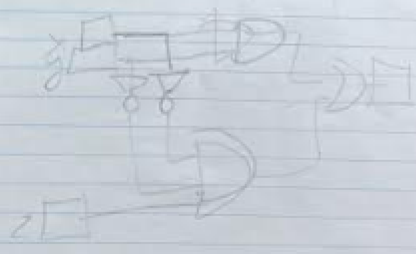
XOR

Decimal Binary:

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

TRUTH TABLE

r	a	b	c	d	e
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	1	0
14	0	1	1	1	1
15	1	0	0	0	0
16	1	0	0	0	0
17	1	0	0	1	0
18	1	0	0	1	1
19	1	0	1	0	0
20	1	0	1	0	1



Computer Architecture
Week 1
Wednesday

Variable Length Encoding:

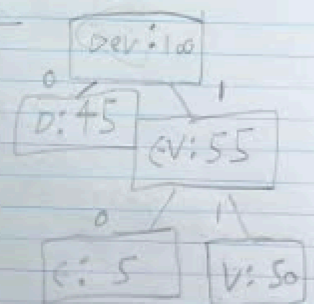
aaabbb	000111
acbd	000111
cabd	000111
acdb	000111
cadb	000111
aaabd	000111
~~acadb~~	000111
acbbb	000111
cabbb	000111

letter	code
a	0
b	1
c	00
d	11

Huffman's Algorithm

Name

character	Frequency
D	45
E	5
V	50



~~Prefix Code~~

~~Letter~~	~~er~~
~~D~~	
~~E~~	
~~V~~	

Letter	Prefix Code:
D	0
E	10
V	11

Seems to be a split system

1 split : restBlock - 1
2 split : restBlock - 2
etc.