Unit 1

Data Representation

1.1 Data Types

1.2 Complements

1.3 Fixed Point Representation

1.4 Floating Point Representation

Sanjeev Thapa, Er. DevOps, SRE, CKA, RHCSA, RHCE, RHCSA-Openstack, MTCNA, MTCTCE, UBSRS, HEv6, Research Evangelist

1.1 DATA TYPES

• Computers store all information in binary (0 and 1).

• Data types define how values are stored, processed, and interpreted.

• **Types of data:**

1. Numeric Data
   - Integer (unsigned, signed magnitude, 1's complement, 2's complement)
   - Floating point (scientific notation using exponent + mantissa)

2. Character Data
   - Stored using ASCII, Unicode, EBCDIC
   - Example: 'A' → 65 → 01000001

3. Alphanumeric Data
   - Combination of characters + digits
   - Example: NEPAL2025

4. Boolean / Logical Data
   - True/False or 1/0

5. Special Data Types
   - Strings, arrays, pointers, multimedia data

• Example:

+13 (8-bit binary) = 00001101

-13 (2's complement) =

Step 1: Invert → 11110010

Step 2: Add 1 → 11110011

1.2 COMPLEMENTS

• Complements simplify binary subtraction and signed number representation.

A. 1's Complement

• Found by inverting all bits.

• Example:

10110001 → 01001110 (1's complement)

B. 2's Complement

• Most commonly used signed representation.

• Steps:

1. Find 1's complement
2. Add 1

• Example:

01101010 →

Invert: 10010101

Add 1: 10010110

C. Subtraction using 2's Complement

• A − B = A + (2's complement of B)

• Example: 18 − 11

18 = 00010010

11 = 00001011

2's complement of 11 = 11110101

Add:

00010010

+ 11110101

00000111 (carry ignored)

• Result = 7

COMPLEMENTS — N-BIT SUBTRACTION RULE

RULE:

• If A and B are both n-bit numbers, then:

A − B = A + (2's complement of B)

• After addition:

Case 1: If the result produces a carry out of the nth bit

→ The carry is discarded

→ Final n-bit result is POSITIVE.

Case 2: If NO carry is produced

→ Result is NEGATIVE

→ The answer is in 2's complement form.

GENERAL STEPS:

1. Represent A in n bits
2. Represent B in n bits
3. Find 2's complement of B
4. Add A + (2's complement of B)
5. Check carry:
   - Carry = 1 → result is positive (discard carry)
   - Carry = 0 → result is negative (take 2's complement)

EXAMPLE 1: (With Carry → Positive Result)

A = 18

B = 11

Use 8 bits.

A = 00010010

B = 00001011

Step 1: 2's complement of B

B = 00001011

1's comp = 11110100

+1 = 11110101

Step 2: Add A + (2’s comp B)

00010010

+ 11110101

---

1 00000111 ← carry out = 1

Step 3: Discard carry → final result:

00000111 = +7

EXAMPLE 2: (No Carry → Negative Result)

A = 9

B = 12

Use 8 bits.

A = 00001001

B = 00001100

Step 1: 2's complement of B

B = 00001100

1's comp = 11110093

+1 = 11110100

Step 2: Add A + (2's comp B)

00001001

+ 11110100

11111101 ← NO carry

Step 3: Negative result → take 2's complement

11111101 → (invert) 00000010

+1 00000011

Final answer = -3

EXAMPLE 3: (Exact n-bit Subtraction Pattern)

A = 25

B = 17

Use 8 bits.

A = 00011001

B = 00010001

Step 1: 2's complement of B

00010001 → invert → 11101110 → +1 → 11101111

Step 2: Add

00011001

+ 11101111

---

1 00001000 ← carry

Step 3: Discard carry → result:

00001000 = +8

EXAMPLE 4: (Negative Result After Subtraction)

A = 5

B = 7

Use 8 bits.

A = 00000101

B = 00000111

Step 1: 2's complement of B

00000111 → invert = 11111000 → +1 = 11111001

Step 2: Add

00000101

+ 11111001

11111110 ← NO carry

Step 3: Negative result → find 2's complement

11111110 → invert → 00000001 → +1 → 00000010

Answer = -2

1.3 FIXED POINT REPRESENTATION

• Binary point is fixed in a predetermined position.

• Used for integers and simple fractional numbers.

A. Integer Fixed-Point

• Unsigned 8-bit range = 0 to 255

• Signed 8-bit (2's complement) range = -128 to +127

B. Fractional Fixed-Point

• Example:

Binary: 1101.101

= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125

= 13.625

C. Q-Notation (Fixed-point DSP format)

• Qm.n → m integer bits, n fractional bits

• Example:

Q4.4 number 1011.1100 = 11.875

D. Overflow Example

• 4-bit signed range: -8 to +7

• Try:

0111 (+7) + 0010 (+2) = 1001 (-7) → Overflow

1.4 FLOATING POINT REPRESENTATION

• Used to store very large or very small real numbers.

• General form:

Value = Sign × Mantissa × 2^(Exponent)

• IEEE-754 Single Precision (32-bit):

- Sign bit: 1 bit

- Exponent: 8 bits (bias = 127)

- Mantissa: 23 bits

Example: Convert 13.25 to IEEE 754

Step 1: Convert to binary:

13 = 1101

0.25 = .01

→ 1101.01

Step 2: Normalize:

1101.01 = 1.10101 × 2^3

Step 3: Fields:

Sign = 0

Exponent = 3 + 127 = 130 → 10000010

Mantissa = 10101000000000000000000

Final IEEE 754:

0 10000010 10101000000000000000000

WHY CRC IS USED?

CRC is used because it is fast, simple, low-cost, and extremely effective at detecting transmission errors, especially burst errors.

1. ERROR DETECTION

- CRC is used to detect errors during data transmission.
- It can identify single-bit errors, double-bit errors, burst errors, and many random errors.

2. HIGH RELIABILITY

- CRC gives very strong mathematical protection.
- It detects more than 99.99% of transmission errors.
- Much more reliable than simple parity or checksum.

3. BURST ERROR DETECTION

- CRC is excellent at detecting burst errors (multiple bits flipped).
- A burst error of length ≤ degree(G) is always detected.

4. SIMPLE HARDWARE IMPLEMENTATION

- CRC can be implemented using simple XOR and shift registers.
- Very fast for real-time transmission (network cards, routers).

5. LOW COST, LOW OVERHEAD

- Only a few extra bits (CRC remainder) are added to the data.
- Very efficient for large frames.

6. USED IN MANY COMMUNICATION SYSTEMS

• CRC is used in Ethernet, Wi-Fi, USB, HDLC, storage devices, optical fiber, satellite links, and many digital protocols.

7. DETECTS ACCIDENTAL ERRORS (NOT RANDOM NOISE)

• CRC detects accidental data changes with very high accuracy.

• Ensures data integrity in noisy communication channels.

CRC EXAMPLE WITH BURST ERROR DETECTION

==========

GIVEN:

• Data bits (message) A: 1011001 (7 bits)

• Generator (divisor) G: 1101 (4 bits)

• Degree of G = 4 − 1 = 3

→ Append 3 zeros to data before division.

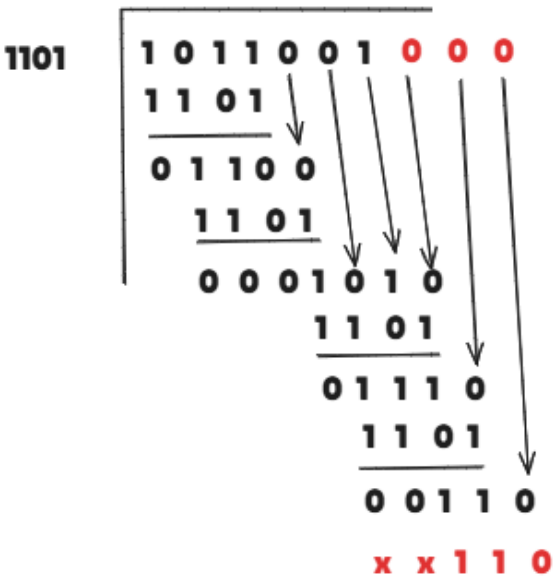
• We will:

(a) Do sender-side CRC computation (long division)

(b) Form transmitted codeword

(c) Introduce a burst error

(d) Check at receiver using CRC



(a) SENDER SIDE: CRC CALCULATION

Step 1: Append (n−1) zeros to data

• Original data: 1011001

• Append 3 zeros: 1011001000 (this is the dividend)

• Generator (G): 1101

We now divide 1011001000 by 1101 using binary XOR division.

LONG DIVISION STEPS (BINARY XOR)

Dividend initial: 1011001000

Divisor (G): 1101

• Step 1:

- Look at the leftmost 4 bits: 1011
- Since leading bit = 1, XOR with 1101

```
  1011
^ 1101
------
  0110
```

- Replace bits 0..3 with 0110
- New dividend after Step 1:

0110001000

• Step 2:

- Move to next bit (position 1)

- Now take bits 1..4: 1100

- XOR with 1101

```
  1100
^ 1101
------
  0001
```

- Replace bits 1..4 with 0001

- New dividend after Step 2:

```
  0000101000
```

• Step 3:

- Next non-zero starting bit is at position 4

- Take bits 4..7: 1010

- XOR with 1101

```
  1010
^ 1101
------
  0111
```

- Replace bits 4..7 with 0111

- New dividend after Step 3:

```
  0000011100
```

• Step 4:

- Next non-zero starting bit is at position 5

- Take bits 5..8: 1110

- XOR with 1101

```
  1110
^ 1101
------
  0011
```

- Replace bits 5..8 with 0011

- New dividend after Step 4:

```
  0000000110
```

Now division ends (we have used all original data bits).

REMAINDER (CRC BITS)

• Final dividend: 0000000110

• Last 3 bits are the remainder (because degree = 3):

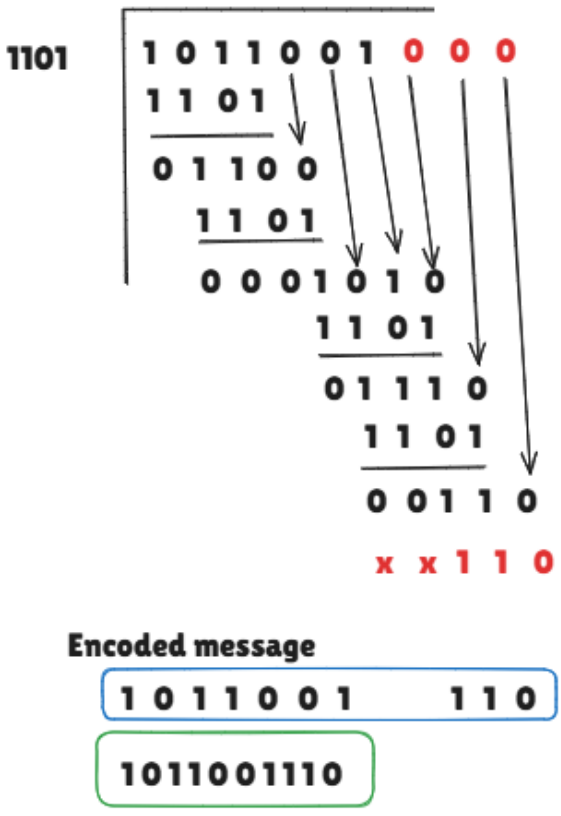
CRC = 110

• Sender forms the codeword:

CODEWORD = [data][CRC]

= 1011001 110

So the bits actually transmitted are:



(b) BURST ERROR IN TRANSMISSION

Suppose during transmission, a burst error flips three consecutive bits.

Original transmitted frame:

1 0 1 1 0 0 1 1 1 0

| | | | | | | | | |

0 1 2 3 4 5 6 7 8 9 (bit positions)

Let a burst error flip bits at positions 4, 5, and 6.

- Original bits at 4,5,6: 0 0 1
- Flipped bits: 1 1 0

So the RECEIVED FRAME becomes:

1011110110 (after burst error)

We now check this at the receiver using the same generator 1101.

(c) RECEIVER SIDE: CRC CHECK

Receiver divides the received frame by the same G = 1101.

Received bits: 1011110110

Append nothing extra here (already includes CRC).

We perform the same XOR division (only summary given):

• After full division of 1011110110 by 1101,

the final remainder is:

Remainder = 101 (NON-ZERO)

(If you do full XOR steps, the final bits end as ...0101,

so the last 3 bits are 101.)

CRC DECISION:

• Rule:

- If remainder = 000 → NO error (accept frame)
- If remainder ≠ 000 → ERROR detected (reject frame)

• Here remainder = 101 ≠ 000

Therefore:

→ The receiver detects that the frame has been corrupted

by an error (burst error is successfully detected).

FINAL SUMMARY

---

1. Data: 1011001
2. Generator: 1101
3. Sender appends 000 and divides:
   - Remainder (CRC) = 110
   - Transmitted codeword = 1011001110

4. Burst error flips bits 4,5,6:

- Received frame = 1011110110

5. Receiver divides by 1101:

- Remainder = 101 ≠ 000 → Error detected.

CRC SUCCESSFULLY DETECTED THE BURST ERROR.