Lecture 03

Data Representation & Combinational Circuits

Number Systems · 2's Complement · IEEE 754 · Boolean Algebra · Logic Gates

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. Convert numbers between binary, decimal, octal, and hexadecimal using positional notation
- 2. Represent signed integers in sign-magnitude, 1's complement, and 2's complement and detect overflow
- 3. Encode and decode IEEE 754 single-precision floating-point numbers including special values
- 4. Apply all Boolean algebra laws including De Morgan's theorems to simplify logic expressions
- 5. Design and analyse a half adder, full adder, MUX, and decoder using Boolean expressions and truth tables

Section 1 · Number Systems and Data Representation

📖 **Stallings Reference**

- Appendix A: Number Systems
- Chapter 9: Computer Arithmetic — Integer representation, IEEE 754 floating-point

All information in a computer — numbers, text, images, machine instructions — is stored as patterns of binary digits (bits). Understanding data representation is essential for debugging low-level code, analysing overflow bugs, and understanding floating-point imprecision.

1.1 Positional Number Systems

In a base-r positional system, a number with n digits d_{n-1} d_{n-2} … d_1 d_0 has the value:

★ **Positional Notation**

- Value = Σ(i=0 to n-1) [d_i × r^i]
- Binary (r=2): digits 0,1; each position doubles in value
- Octal (r=8): digits 0–7; one octal digit = 3 binary bits
- Hexadecimal (r=16): digits 0–9,A–F; one hex digit = 4 binary bits

- Conversion rule: binary ↔ hex is done 4 bits at a time; binary ↔ octal is done 3 bits at a time

Complete conversion table (8-bit values):

Binary	Octal	Decimal	Hex	Notes
00000000	000	0	00	Zero
00000001	001	1	01	One
00001010	012	10	0A	Decimal 10
00001111	017	15	0F	Decimal 15 (max nibble)
01111111	177	127	7F	Max positive signed 8-bit
10000000	200	128	80	Min unsigned / problematic signed
11111111	377	255	FF	Max unsigned 8-bit

1.2 Signed Integer Representation

Scheme	Positive +7 (8-bit)	Negative −7 (8-bit)	Range (n-bit)	Zeros	Arithmetic
Sign-magnitude	00000111	10000111	−(2^{n-1}−1) to +(2^{n-1}−1)	Two: +0 and −0	Requires checking sign bit; two separate adder paths
1's complement	00000111	11111000	−(2^{n-1}−1) to +(2^{n-1}−1)	Two: +0 (00…0) and −0 (11…1)	Negation = invert all bits; end-around carry in adder
2's complement	00000111	11111001	−2^{n-1} to +(2^{n-1}−1)	One: 0 (00…0)	Negation = invert + 1; standard binary adder works directly

★ 2's Complement Conversion Procedure

- Step 1: Take positive representation (e.g. +7 = 00000111)
- Step 2: Invert all bits (NOT): 11111000
- Step 3: Add 1: 11111001 = −7 in 2's complement ✓
- Verify: 11111001 + 00000111 = 100000000; discard the carry-out → 00000000 = 0 ✓
- Key property: Negating −128 in 8-bit overflows back to −128 (no positive +128 exists in 8-bit). This is 2's complement overflow.

⚠ Overflow vs. Carry

- Carry-out (C): the carry bit leaving the MSB position during addition — signals unsigned overflow.
- Overflow flag (V): Positive + Positive = Negative, or Negative + Negative = Positive — signals signed overflow.
- Example: 01111111 + 00000001 = 10000000. Unsigned: correct (128). Signed: overflow (127 + 1 should = 128, but we get −128).

1.3 IEEE 754 Floating-Point

Format	Sign	Biased Exponent	Mantissa (fraction)	Bias	Range
Single (32-bit)	1 bit	8 bits	23 bits	127	±1.18×10^{−38} to ±3.4×10^{38}
Double (64-bit)	1 bit	11 bits	52 bits	1023	±2.2×10^{−308} to ±1.8×10^{308}

★ IEEE 754 Encoding Procedure

- Step 1: Write the number in binary (e.g. 6.5 = 110.1_2)
- Step 2: Normalise to 1.fraction × 2^exponent form: 1.101×2^2
- Step 3: Sign bit = 0 (positive)
- Step 4: Biased exponent = actual exponent + bias = 2 + 127 = 129 = 10000001_2
- Step 5: Mantissa = fractional part of 1.101 = 101 padded to 23 bits = 10100000000000000000000
- Result: 0 10000001 10100000000000000000000 = 0x40D00000

◆ Special IEEE 754 Values

- Zero: exponent = 0, mantissa = 0; sign bit gives ±0
- Denormalised: exponent = 0, mantissa ≠ 0; represents very small numbers near zero
- Infinity: exponent = all 1s, mantissa = 0; represents ±∞ (overflow result)
- NaN (Not a Number): exponent = all 1s, mantissa ≠ 0; represents invalid operations (0/0, √−1)

Section 2 · Boolean Algebra

📖 Stallings Reference

- Appendix B: Digital Logic
- Boolean algebra laws and De Morgan's theorems — foundational for all digital logic design

2.1 Basic Gates and Operations

Gate	Symbol	Expression	Truth Table (A,B → Y)	Key Property
AND	·	Y = A·B	00→0, 01→0, 10→0, 11→1	Output 1 only if ALL inputs 1
OR	+	Y = A+B	00→0, 01→1, 10→1, 11→1	Output 1 if ANY input 1
NOT	'	Y = A'	0→1, 1→0	Inverts; also called complement or inverter

Gate	Symbol	Expression	Truth Table (A,B → Y)	Key Property
NAND	(·)'	Y = (A·B)'	00→1, 01→1, 10→1, 11→0	Universal gate — can build any circuit
NOR	(+)'	Y = (A+B)'	00→1, 01→0, 10→0, 11→0	Universal gate — can build any circuit
XOR	⊕	Y = A⊕B	00→0, 01→1, 10→1, 11→0	Output 1 iff inputs DIFFER
XNOR	⊙	Y = A⊙B	00→1, 01→0, 10→0, 11→1	Output 1 iff inputs are EQUAL

2.2 Boolean Algebra Laws

Law	Expression	Dual Form
Identity	A + 0 = A	A · 1 = A
Null	A + 1 = 1	A · 0 = 0
Idempotent	A + A = A	A · A = A
Complement	A + A' = 1	A · A' = 0
Double negation	(A')' = A	(A')' = A
Commutative	A + B = B + A	A · B = B · A
Associative	(A+B)+C = A+(B+C)	(A·B)·C = A·(B·C)
Distributive	A·(B+C) = A·B + A·C	A+(B·C) = (A+B)·(A+C)
Absorption	A + A·B = A	A · (A+B) = A
De Morgan (1)	(A·B)' = A' + B'	Negate AND → OR with inverted inputs
De Morgan (2)	(A+B)' = A' · B'	Negate OR → AND with inverted inputs

✓ Exam Tip — Using De Morgan's Theorems

- To push a negation through an AND: change AND → OR, negate each literal.
- To push a negation through an OR: change OR → AND, negate each literal.
- Practical use: NAND gate output = (A·B)' = A' + B'. So a NAND gate is equivalent to an OR gate with inverted inputs — useful for technology mapping.
- De Morgan applied twice returns to the original: ((A·B)')' = A·B.

Section 3 · Combinational Logic Circuits

3.1 Half Adder

Adds two 1-bit inputs A and B. Produces a 1-bit Sum and a 1-bit Carry-out.

A	B	Sum (A⊕B)	Cout (A·B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Sum = A XOR B; Carry = A AND B. Implementation: 1 XOR gate + 1 AND gate.

3.2 Full Adder

Adds three bits: A, B, and Carry-in (Cin). Produces Sum and Carry-out. Full adders are chained to build n-bit ripple-carry adders.

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

★ Full Adder Boolean Expressions

- Sum = A ⊕ B ⊕ Cin
- Cout = A·B + A·Cin + B·Cin = A·B + Cin·(A⊕B)
- Can be built from 2 half adders: HA1(A,B) gives S1 and C1; HA2(S1,Cin) gives Sum and C2; Cout = C1 + C2
- Ripple-carry adder: chain N full adders. Carry ripples from LSB to MSB. Delay = N × gate delay.

3.3 Multiplexer (MUX) and Decoder

Component	Function	Boolean Representation
2-to-1 MUX	Selects one of 2 inputs based on 1 select line S	Y = S'·I0 + S·I1 (when S=0: Y=I0; when S=1: Y=I1)
4-to-1 MUX	Selects one of 4 inputs based on 2 select lines S1,S0	Y = S1'·S0'·I0 + S1'·S0·I1 + S1·S0'·I2 + S1·S0·I3
2-to-4 Decoder	Activates exactly one of 4 output lines based on 2-bit input	Y0=A'B', Y1=A'B, Y2=AB', Y3=AB (when Enable=1)
4-to-2 Encoder	Encodes which of 4 active inputs is asserted into a 2-bit code	A=I2+I3; B=I1+I3 (priority encoder for simultaneous inputs)

Section 4 · Practice Exercises

✎ Exercise 3.1 [10 marks] — Number Conversions

- (a) [3] Convert 01101110_2 to: (i) decimal, (ii) hexadecimal, (iii) octal. Show working for each.
- (b) [3] Convert 219 decimal to: (i) 8-bit binary (unsigned), (ii) hexadecimal, (iii) octal.
- (c) [4] Express each of the following in 8-bit two's complement and verify by addition: (i) −75, (ii) −128, (iii) −1. For each, show: positive form → invert → add 1 → verify (sum = 0).

✎ Exercise 3.2 [8 marks] — IEEE 754

- (a) [4] Encode −12.75 in IEEE 754 single-precision. Show all steps: sign, exponent bias calculation, mantissa. Express the final result as 8 hex digits.
- (b) [4] The bit pattern 0 10000100 10010000000000000000000 is a single-precision float. Decode it: find the actual value it represents. Show all steps: sign, unbiased exponent, mantissa, final value.

✎ Exercise 3.3 [12 marks] — Boolean Algebra and Logic Minimisation

- (a) [4] Simplify F = A'BC + ABC + A'B'C + AB'C to its minimal sum-of-products form. Show each step citing the Boolean law used.
- (b) [4] Prove De Morgan's theorem (A+B)' = A'·B' using a truth table for all 4 input combinations. Then show an alternative proof using Boolean algebra (start from (A+B)·A'·B' and show it equals 0).
- (c) [4] A circuit is defined by F = (A NAND B) NAND (C NAND D). Express F in terms of AND/OR/NOT only, applying De Morgan's theorem. Simplify the result.

✎ Exercise 3.4 [10 marks] — Adder Design

- (a) [4] Design a 4-bit ripple-carry adder using four full adder blocks. Show the carry chain connections. Compute 0110_2 + 1011_2 step by step, showing A, B, Cin, Sum, Cout for each bit position.
- (b) [3] The result of adding two 4-bit unsigned numbers is: A=1100, B=0101. (i) What is the Sum output? (ii) Is there a carry-out? (iii) Interpret as signed 2's complement — does overflow occur? Justify.
- (c) [3] A ripple-carry adder for 32-bit operands has a gate delay of 1 ns per full adder stage (2 levels of logic each). What is the worst-case propagation delay? Compare with a carry-lookahead adder that computes all carries simultaneously in 3 gate levels.