Binary Arithmetic

Table of contents

See binary-number-representations[1] for signed-format background (signed magnitude, two's complement) and logic-gates[2]/boolean-algebra[3] for the gate-level building blocks used below.

Quiz: -32 in 8-bit signed magnitude

Signed magnitude splits the word into a sign bit (MSB, 1 = negative) and a magnitude in the remaining bits: 32 = 00100000, so with the sign bit set this is `10100000` (A).

Quiz: -32 in 8-bit two's complement

Two's complement of a positive value: invert all bits, then add 1.

$$00100000 \rightarrow \text{invert} \rightarrow 11011111 \rightarrow +1 \rightarrow 11100000$$

[1] courses/csse2010/binary-number-representations.pdf
[2] courses/csse2010/logic-gates.pdf
[3] courses/csse2010/boolean-algebra.pdf

So -32 in 8-bit two's complement is `11100000` (D) — different from the signed-magnitude answer above, which is the point: the same bit pattern means different things depending on the declared format.

Universal gates recap

NOT/AND/OR can each be built from NAND-only or NOR-only gates (see logic-gates[4] for the equivalent-circuit diagrams) — this was revision from last lecture before moving into arithmetic.

Binary addition

Single-bit addition (no carry-in):

Addend	0	0	1	1
Augend	+0	+1	+0	+1
Sum	0	1	1	0
Carry	0	0	0	1

- The **bit-wise addition procedure is identical** regardless of whether the operands are interpreted as unsigned or two's complement — only the *interpretation* of the result differs.
- **Two's complement**: the carry-out from the MSB is simply discarded to get the correct result.
- **One's complement**: the carry-out from the MSB must be added back into the result (end-around carry) — a practical drawback of one's complement versus two's complement.

Worked example (8-bit):

```
  Decimal   Unsigned      Decimal   2's complement
  10        00001010      10        00001010
+ 243     + 11110011    +(-13)    + 11110011
--------- ------------  -------   -------------
  253       11111101      -3        11111101
```

[4] courses/csse2010/logic-gates.pdf

Overflow

Overflow: the true result doesn't fit in the available bits, so the answer is wrong.

- **Unsigned**: overflow carry-out from the MSB.
- **Two's complement**: overflow carry-in to the MSB carry-out from the MSB, i.e. Overflow $= C_{in} \oplus C_{out}$.
- Equivalently for two's complement: overflow happens when two positives sum to a negative, or two negatives sum to a positive.

Worked example — both cases overflow:

```
  Decimal   Unsigned      Decimal   2's complement
  15        00001111      125       01111101
+ 243     + 11110011    +   4     + 00000100
--------- ------------  -------   -------------
  258       00000010     129        10000001   (wraps to -127, wrong)
```

Quiz: adding two 6-bit two's complement numbers

$110101 + 001111$ in 6 bits:

```
  110101
+ 001111
--------
 1000100   → truncate to 6 bits → 000100
```

Answer: `000100` (A) — the 7th bit is discarded since we're constrained to 6 bits (no overflow here since $C_{in} \oplus C_{out} = 0$ into/out of the MSB).

Half-adder

A device that adds 2 bits with **no carry-in**.

Truth table (inputs A, B; outputs Sum, Carry):

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A	B	Sum	Carry

$$S = A \oplus B \qquad C = AB$$

Full adder

A half-adder alone can't handle a carry-in from a previous stage, so a **full adder** takes three inputs (A, B, C_{in}) and produces Sum and C_{out}:

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + C_{in}(A \oplus B)$$

A full adder is built from **two half-adders plus an OR gate**: the first half-adder combines A and B; its sum output feeds into a second half-adder along with C_{in}; the two half-adders' carry outputs are OR'd together to form C_{out}.

Ripple-carry adder

Cascading full adders — each stage's C_{out} feeding the next stage's C_{in} — builds a multi-bit binary adder (e.g. 4 full adders for 4-bit addition: $A_3B_3 \ldots A_0B_0 \rightarrow S_3 \ldots S_0, C_4$). For plain addition the initial carry-in C_0 is 0. This structure is called a **ripple-carry adder**, since the carry has to propagate ("ripple") through every stage before the final sum is valid.

Reminders

- Week 2 labs: P2 sessions (Mon-Tue) run Lab 2 (Logic Gates); P1 sessions (Thu-Fri) run Lab 3 (Binary Arithmetic).
- Complete the week 1 exercises folder and ask if unclear.
- Watch the summary video on Signed Binary Formats.