Lecture 1 — Course Introduction; Bits, Bytes and Binary

Table of contents

Outline 1
What the course is about 1
Levels of abstraction 2
Computers and binary numbers 3
Bits, bytes, nibbles and words 3
Binary numbers 4
Worked exercise — convert 26_{10} to octal and hexadecimal 4
Which radix notations this course uses 5
Reminders 5

See csse2010[1] for staff, assessment, and course logistics. The technical content of Lecture 1 is below.

Outline

- Course intro, organisation and admin
- Assessment details
- Technical content — binary numbers

What the course is about

Computers, in several different aspects — from a desktop PC down to the microcontrollers embedded in a car, a washing machine or an Arduino board. The semester climbs the stack in three blocks:

[1] courses/csse2010/csse2010.pdf

Weeks	Block
W01–W05	Hardware
W06–W08	Low-level software
W09–W13	High-level software

Levels of abstraction

A computer can be described at six numbered levels, each built on the one below it. Level 0 is where this course starts.

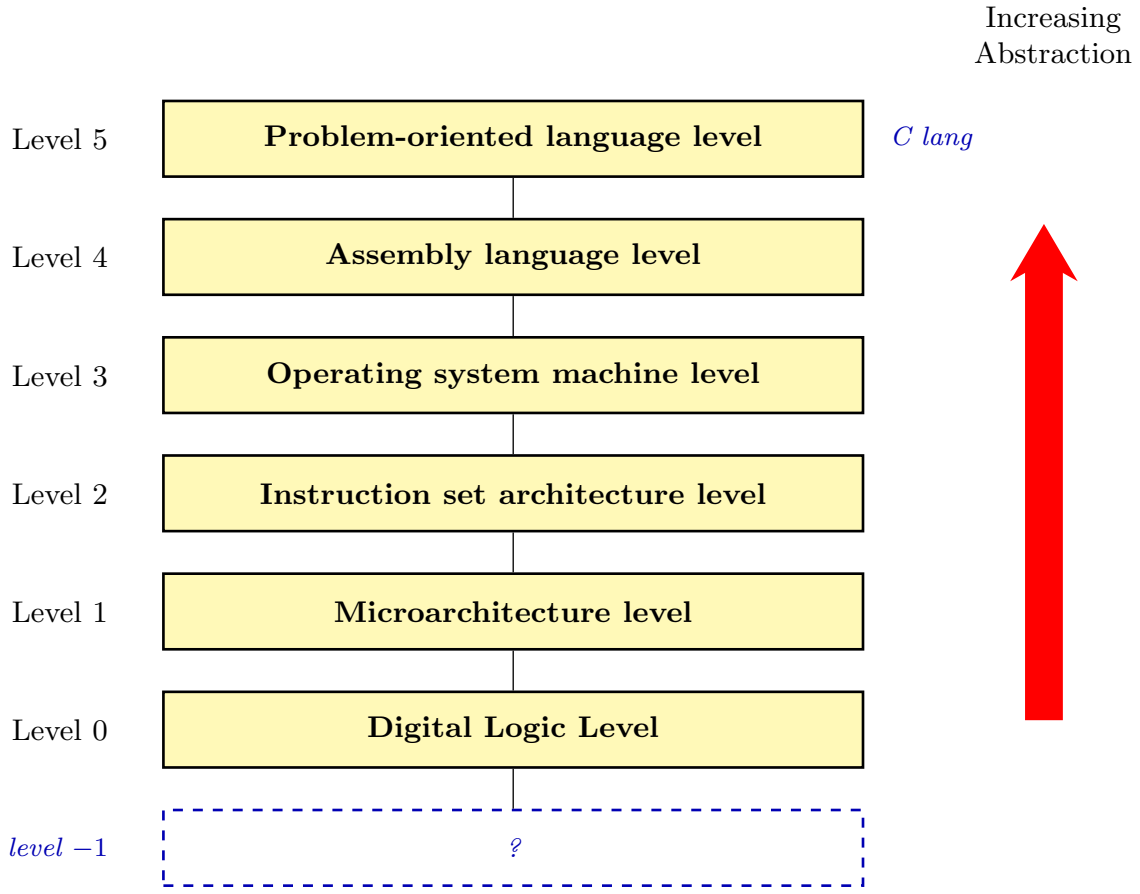

Figure 1: The six levels of abstraction. Each level is implemented in terms of the one below it, so the same machine can be discussed as gates, as an instruction set, or as a C program. In blue: C sits at Level 5, and there is an unnamed *level* −1 underneath the digital logic level, below the bottom of the numbered stack.

The course works up this stack across the semester, starting at the bottom: Lecture 2 is already at the digital logic level. The three course blocks are not tied to particular numbered levels, only ordered along the stack, but the named levels reappear later as topics in their own right (see instruction-set-architecture[2] for Level 2, and assembly-process[3] and compilation-and-linking[4] for how Levels 4 and 5 are turned into machine code).

Computers and binary numbers

A computer — general purpose or embedded — takes **inputs** and **instructions** and produces **outputs**. All three are binary inside the machine.

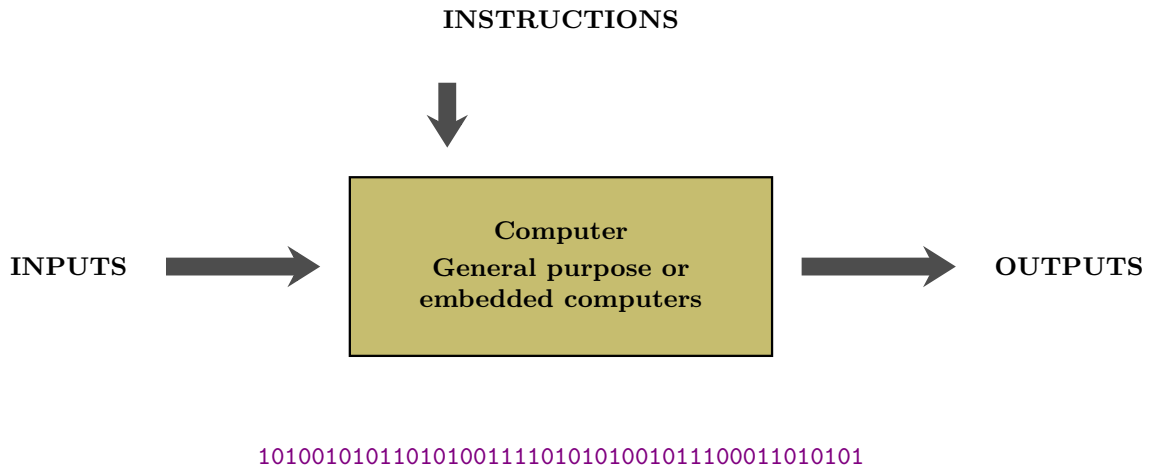

Figure 2: The computer as a black box. Inputs, instructions and outputs are all just bit patterns; nothing inside the machine holds a decimal digit, a letter or a sign. Everything from here on is about what a given bit pattern is agreed to mean.

Bits, bytes, nibbles and words

- **Bit** = binary digit (0 or 1).
- **Nibble** = 4 bits, the unit that makes hexadecimal convenient.
- **Byte** = 8 bits, e.g. `01010111`.

[2] courses/csse2010/instruction-set-architecture.pdf
[3] courses/csse2010/assembly-process.pdf
[4] courses/csse2010/compilation-and-linking.pdf

- **Word** = the natural unit a machine works on, usually a power-of-2 number of bytes: 1, 2, 4 or 8 bytes = 8, 16, 32 or 64 bits. At the two ends of that list, the **AVR** used in this course has an **8-bit** word and **modern CPUs** are 32- or 64-bit.

Larger amounts are counted in kilobytes, megabytes and gigabytes, which mean powers of 2 for memory but usually powers of 10 for disks (memory-technologies[5]).

Binary numbers

Unsigned binary is covered in full (with diagrams and worked examples) in binary-number-representations[6]. In outline:

- each bit position is worth a power of 2, and binary → decimal is just adding the positions where the bit is 1 — e.g. `10010001` = 145;
- **MSB** and **LSB** — the positions worth the most and least; for an n-bit unsigned word the MSB is worth 2^{n-1};
- two methods for decimal → binary (sum of powers of 2; repeated division by 2), both worked on 53;
- the unsigned range: all 0's = 0, all 1's = $2^n - 1$;
- other radices — octal (radix 8, 3 bits per digit) and hexadecimal (radix 16, 4 bits per digit);
- how a radix is indicated in code (`0x`, `$`, leading `0`, `0b`, subscripts).

Signed numbers are Lab 1's material; see week1-lab-number-representations[7].

Worked exercise — convert 26_{10} to octal and hexadecimal

Divide repeatedly by the target radix, exactly as Method 2 does for binary, and read the remainders back.

The same answer falls out of the bit pattern directly: $26_{10} = 00011010_2$, which groups into nibbles as `0001 1010` = `1 A`, and into triples as `00 011 010` = `0 3 2` — see the bit-grouping figure in binary-number-representations[8].

[5] courses/csse2010/memory-technologies.pdf
[6] courses/csse2010/binary-number-representations.pdf
[7] courses/csse2010/week1-lab-number-representations.pdf
[8] courses/csse2010/binary-number-representations.pdf

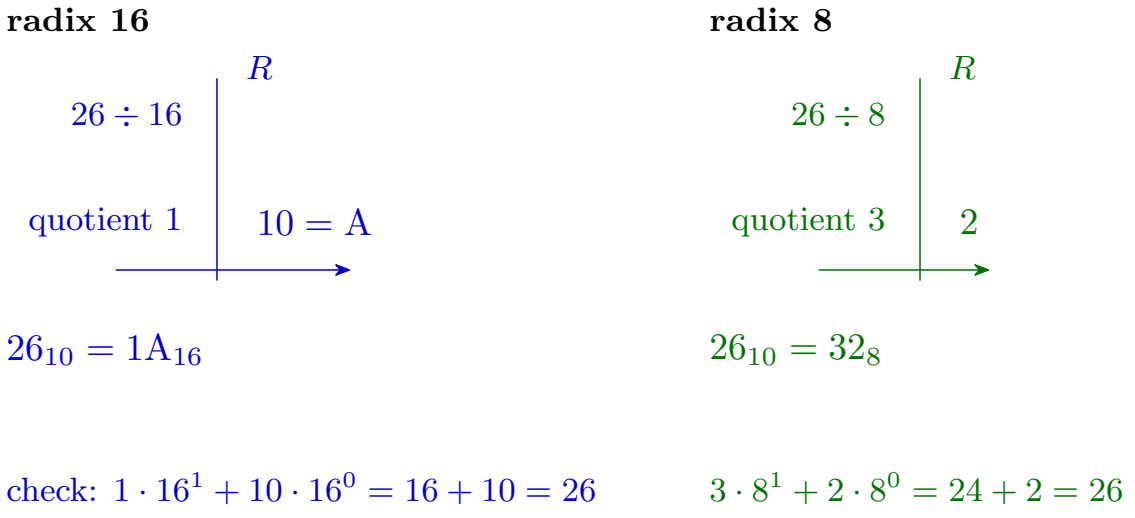

Figure 3: Converting 26_{10} by repeated division. Dividing 26 by 16 leaves quotient 1 and remainder 10, which is the hex digit A, so $26_{10} = 1A_{16}$; dividing by 8 leaves quotient 3 and remainder 2, so $26_{10} = 32_8$. The check underneath expands each result back out as digit $\times$ radix^{position}.

Which radix notations this course uses

Of the ten ways of marking a radix in the full table, five are the ones this course mostly uses:

- a **subscript**, e.g. 101_2 or 101_{16};
- leading `0x` for hex (C, Atmel AVR);
- leading `$` for hex (Atmel AVR assembly);
- leading `0` for octal (C, Atmel AVR) — so `0101` is *not* one hundred and one;
- leading `0b` for binary (Atmel AVR assembly, some C).

The rest (`h`/`q`/`b` suffixes, `@` and `%` prefixes) appear in various assembly languages but not here. Full table in binary-number-representations[9].

Reminders

- Sign up to a pair of lab sessions (P2, P1).
- Labs start from **Thursday** of week 1 (P1 sessions only — no P2 sessions in week 1).
- From week 2, both P2 and P1 sessions run.

[9] courses/csse2010/binary-number-representations.pdf

- Lab 1 covers different binary representation formats for **signed** numbers (two's complement etc.).
- A summary recording on binary numbers will be posted by the end of week 1.
- Lab sign-on issues: email eait.mytimetable@uq.edu.au.

Practice: binary-conversion-drill[10].

[10] courses/csse2010/binary-conversion-drill.pdf