COMS12200 Introduction to Computer Architecture

Dr. Cian O'Donnell

cian.odonnell@bristol.ac.uk

Topic 1: Data, Control & Instructions

Overview of section 2

- All material for exams will be covered in lectures and labs.

- Lecture slides and lab worksheets for each week will be available online by the previous Friday on Blackboard.

- Grading will be done by two oral “viva” exams (15 minute in-person interview), the first during January exam period, the second during May/June exam period.

Overview of section 2

🤔 If something isn't clear to you:

✋ 1. Put your hand up!

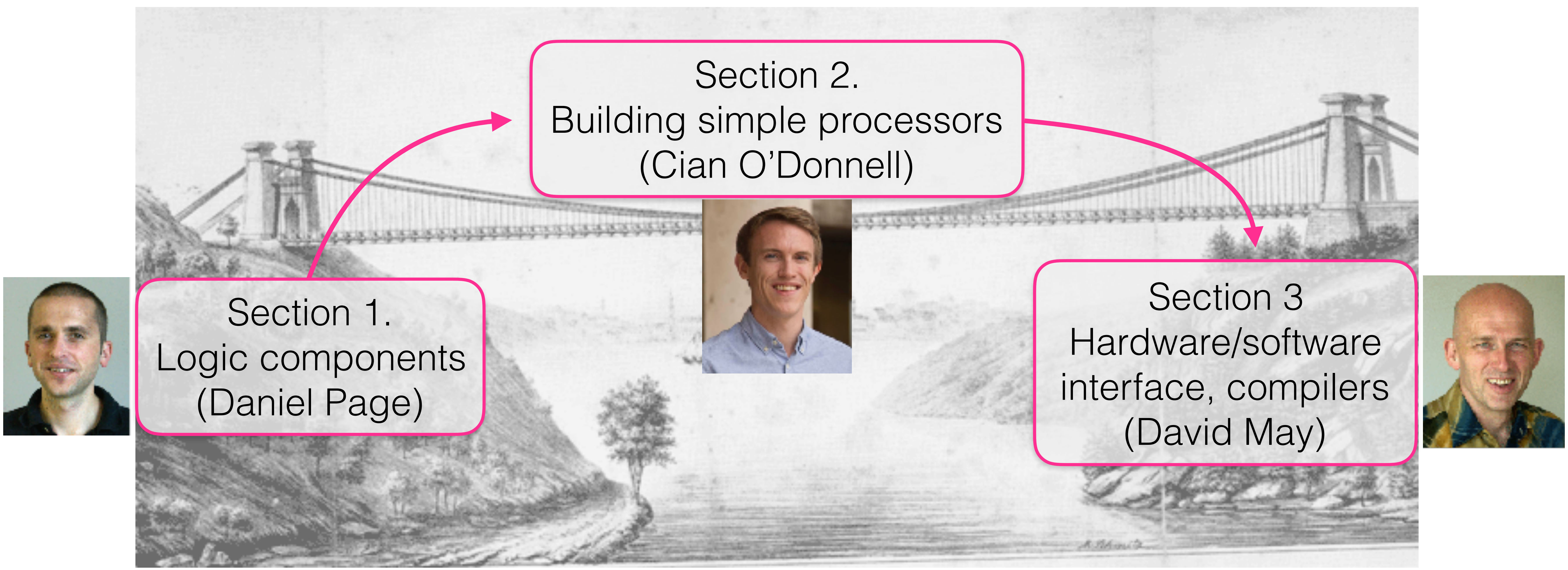
💻 2. Ask a question on the forum.

🤓 3. Ask a question in the labs.

🏃‍♀️🏃 4. Come see me in person in MVB room 3.33.

Overview of section 2

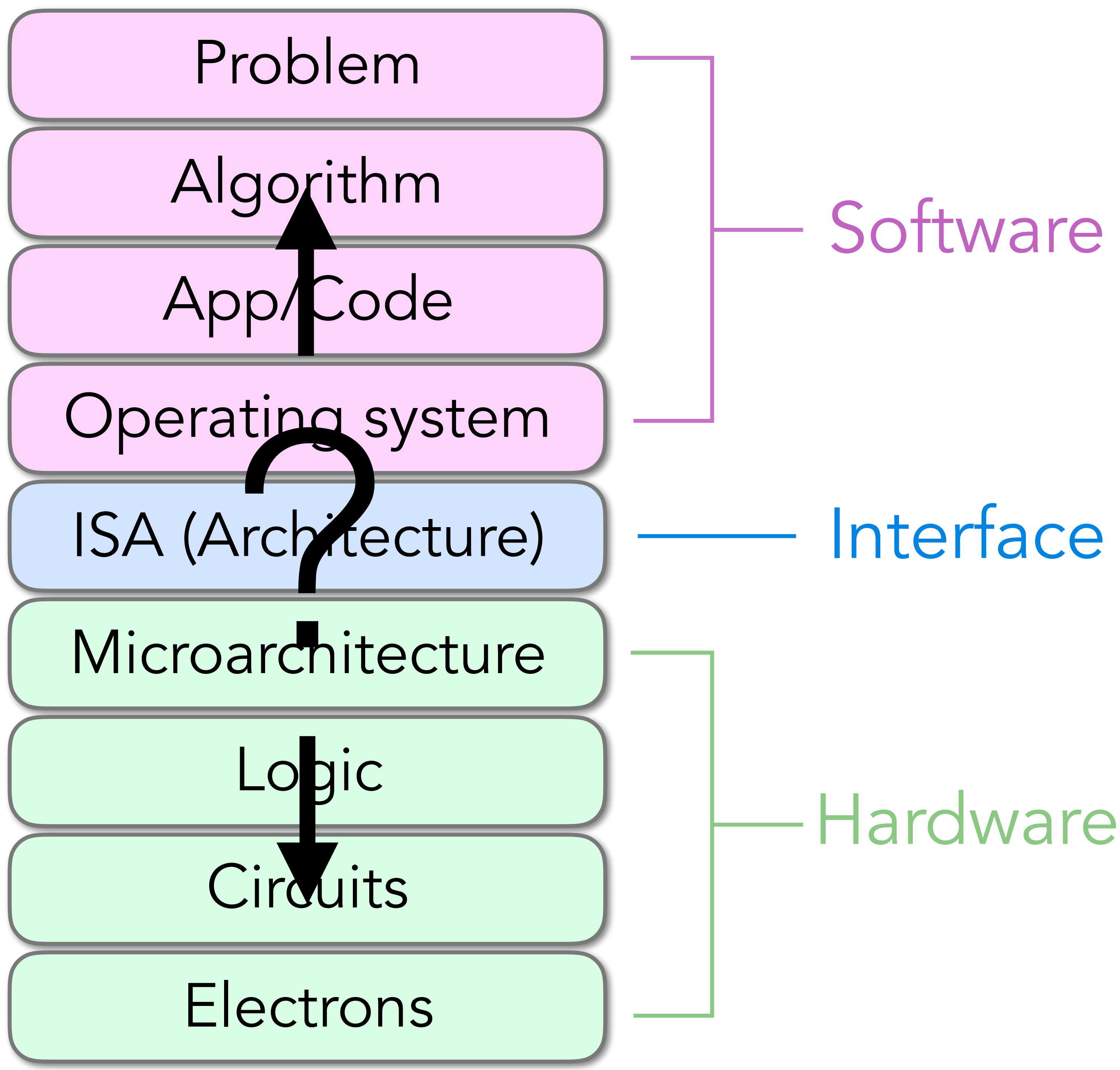
- How to build a simple computer!
- We will black-box the low-level details ~~(physics, transistors, logic gates)~~ and think instead in terms of functioning modules.



What do we use computers to do?

To solve problems

Levels of abstraction



The benefits of abstraction

- A higher level only needs to know how to interface with the next lower level.

So why should we care about hardware?

- Abstraction improves productivity: worker doesn't need to worry about decisions made in underlying levels.

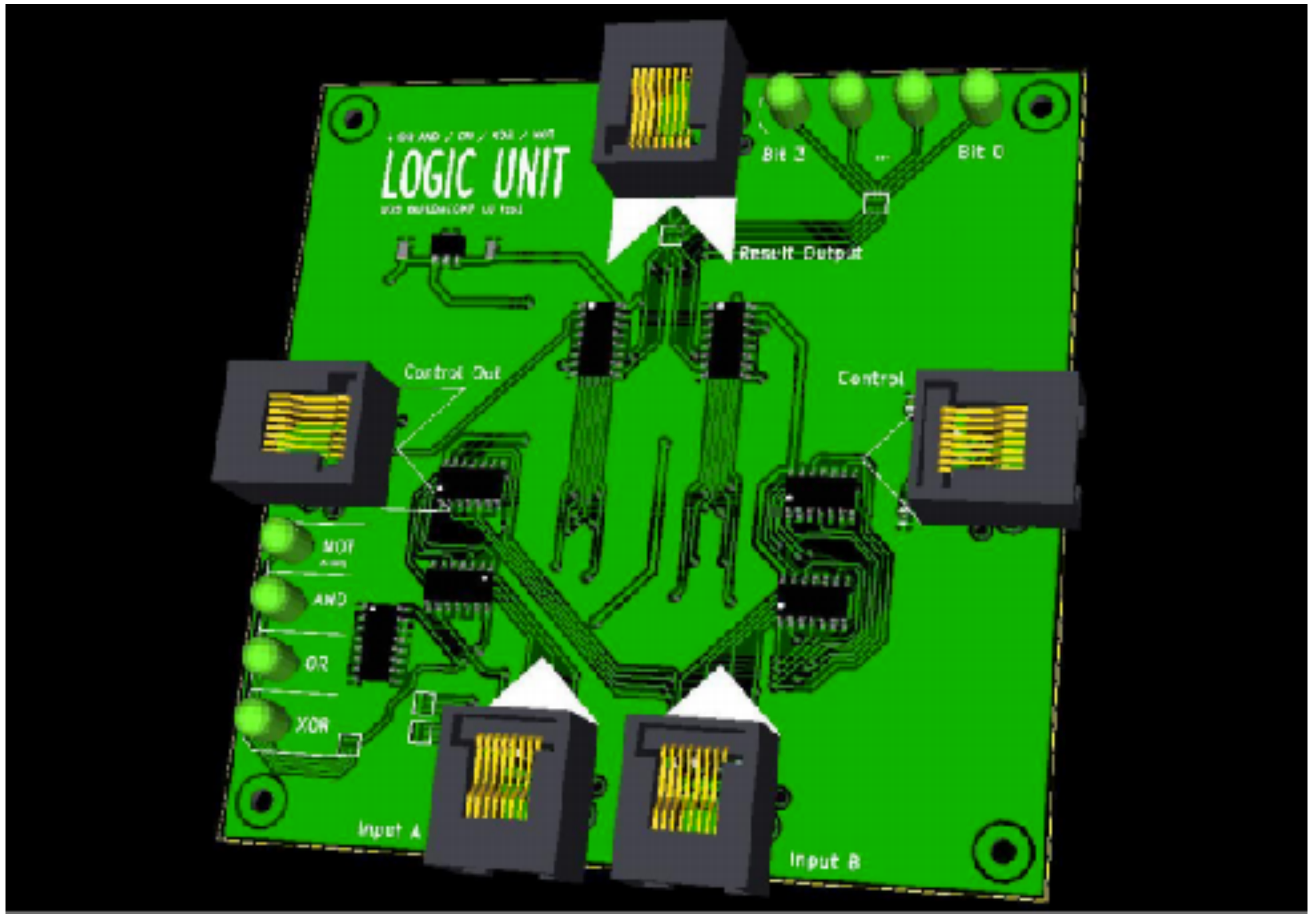
Why understanding the levels matters

- What if
  - the program you wrote is too slow?
  - the program you wrote isn't running correctly?
  - the program you wrote consumes too much energy?
- What if
  - the hardware you designed is too hard to program?
  - the hardware you designed is too slow because it doesn't make the right primitives available to software?
- What if
  - you want to design a more efficient/higher performance system?

Our target

By the end of this part of the course, you will know how to combine blocks of real hardware to make a computer processor, and understand how it works.

We'll be using UoB-developed teaching modules in labs to make this happen.



Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. Machine types
6. State machines and decoding
7. Memory paradigms

Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. Machine types
6. State machines and decoding
7. Memory paradigms

1. Data, control and instructions

Questions we will answer today:

- What the difference between data information and control information?
- What is an instruction and how is it represented by processors?
- What is an op-code?
- How do op-codes get decoded into control signals?

Control vs data

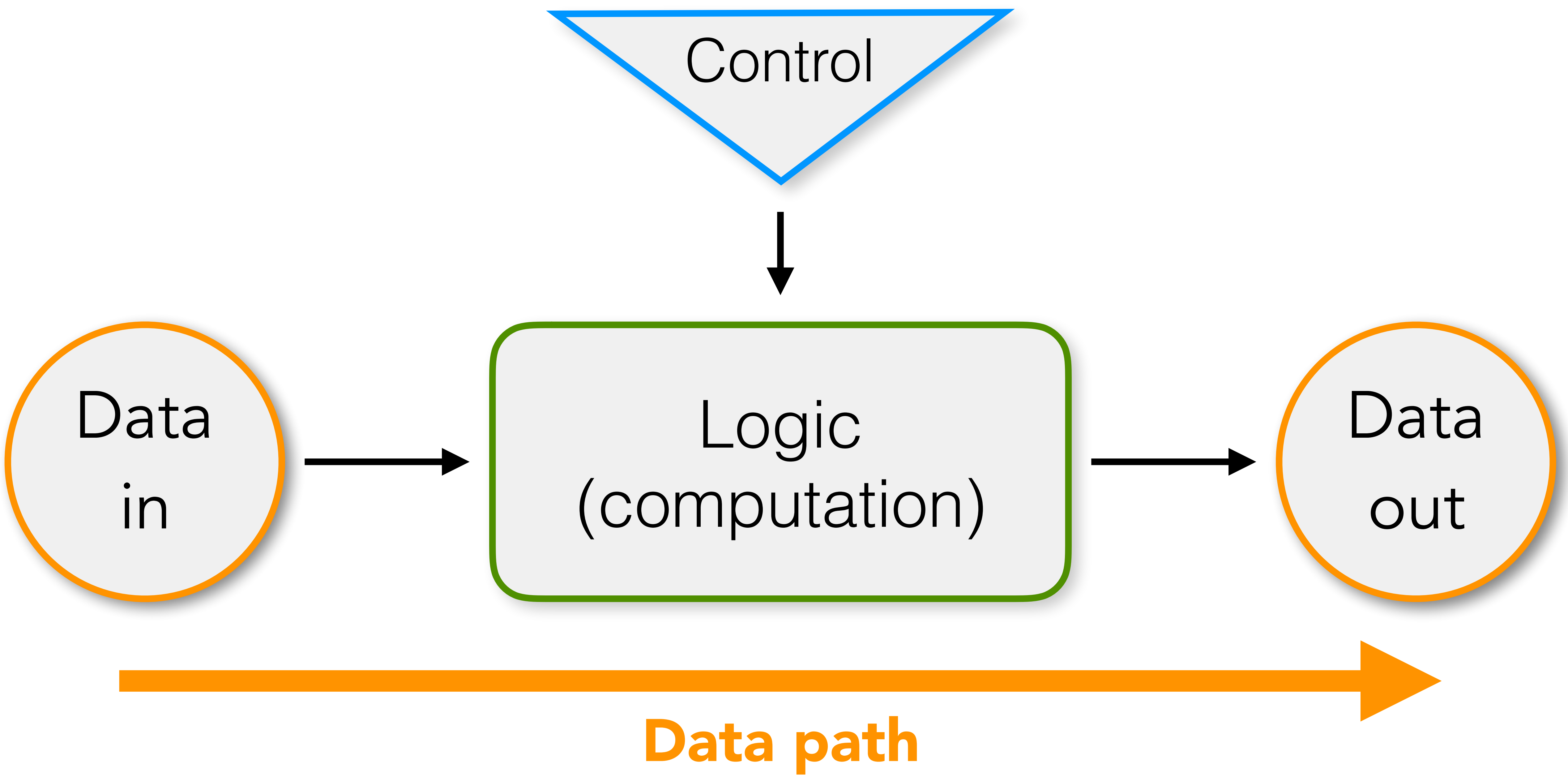
- You can think of a processor's function as being dictated by two separate influences:

Control information
(tells it what to do)

Data information
(operated on to get result)

- These two influences form two paths into the processor logic

Control and data paths



What is DATA?

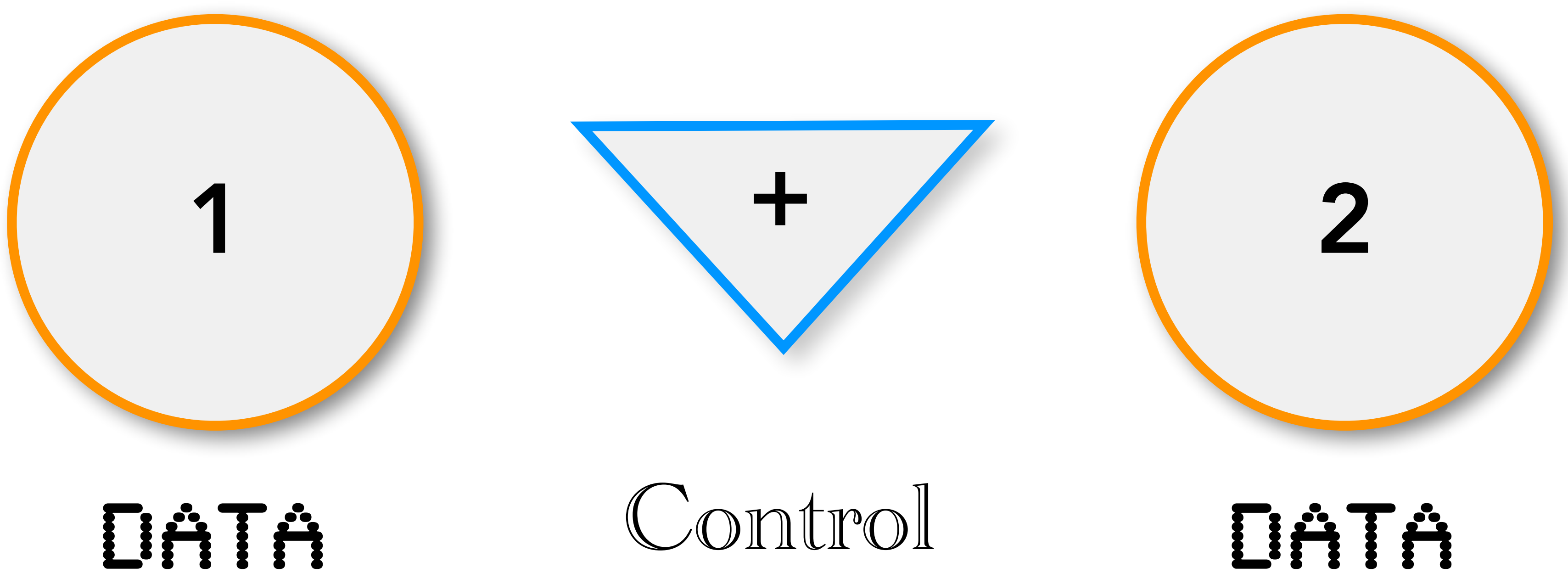
- Data is simply some stored information
- It could be stored in many different ways
- It could be formatted in many different ways
  "`0110`" is data.
  "Hello" is data.
  "☺️" is data.
- Data stores information and forms input to, and outputs from, calculations.

Where does DATA live?

- Data lives in storage elements.
- There are many different storage elements in modern processing systems.
- In this course we'll concentrate on two: memory and registers.
- For this section, both can be treated as **black boxes**.
- Processor instructions always operate on data in registers and sometimes on data in memory too.

Control information

- Control is also information, but its use is different to DATA.
- It specifies what must be done.
- It is applied to a system (but not consumed by it).



Control and instructions

- You’ve seen how to input control information before, in the HP calculator example.
- We’ve encoded choices, such as “`ADD`” with a selection of specific binary control signals being activated and deactivated.

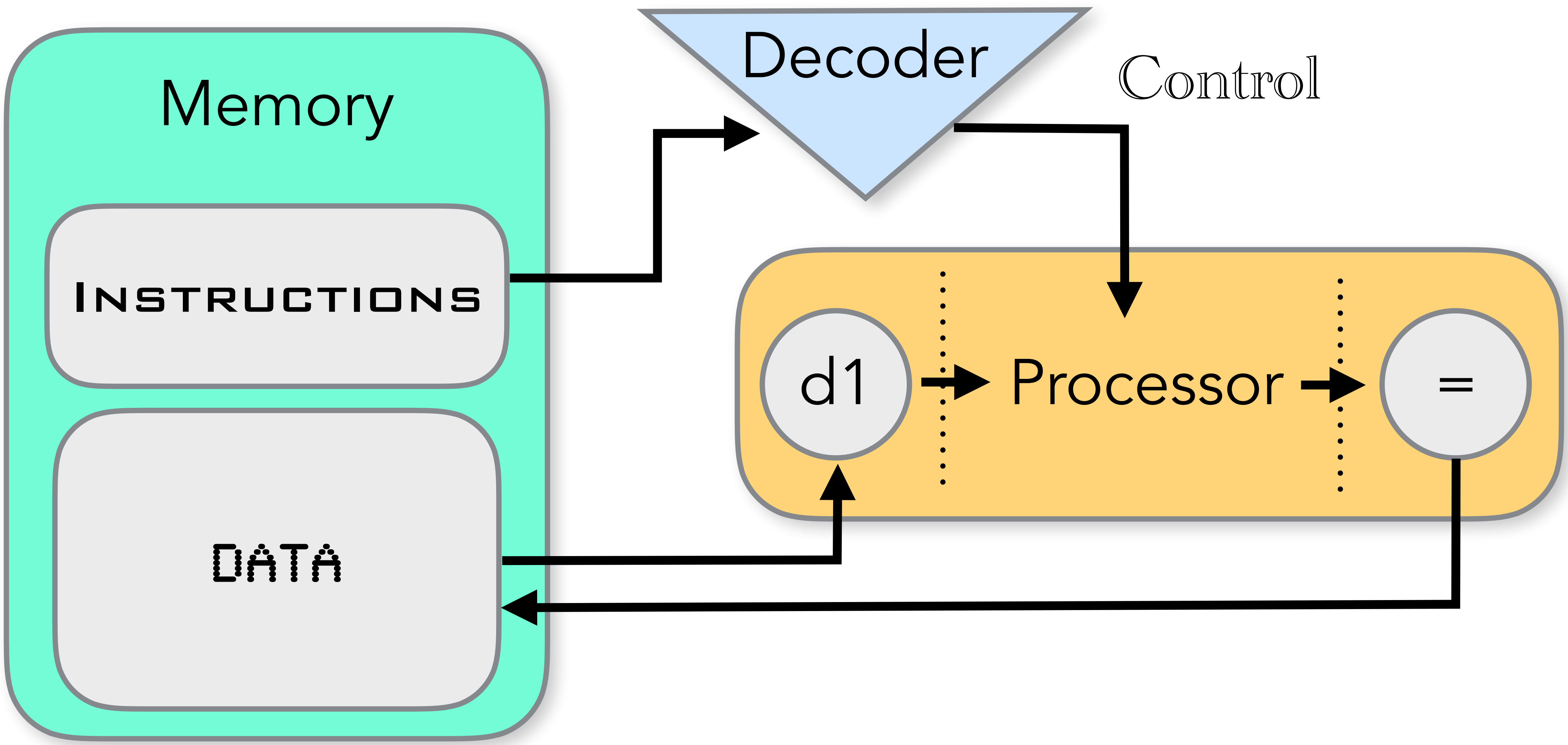
Control and instructions

- In small-scale systems, this is OK, but scaling this up to a modern processor would be a nightmare.
- We need a way of condensing the control information and programming a machine with it.
- The solution is to use INSTRUCTIONS.

Instructions at a high level

- E.g. if an instruction = "A", the processor should do "X", and if the instruction is "B", then the processor should do "Y".
- At the high level, we can treat them as abstract symbols, whose value causes different processor behaviour.
- Instructions need to be *decoded* before they can be used.

How it works



What is an instruction?

- An instruction is also information.
- However, it has a defined purpose: to specify an exact amount of work to be done by a processor.
- This leads it to have a specific form and formatting (more in "ISAs" section of this course).
- Only a sub-set of all possible control values are valid instructions.

Instruction encoding

- Instructions allow us to encode the control information that we need to control a computing system.
- The key is to use a unique code per unique function.
- The specific encoding is called an op-code.

Some op-codes

- Example: here's how several different processor ISAs use different op-codes to encode the same instruction (`ADD` two numbers together).

ISA (Processor instruction set)	Encoding of '`ADD`' (in binary)
ARM	001100
MIPS	100000
Intel x86	00000000
PIC	000111

Decoding

- There are many different possible encodings of instructions for the same meaning.
- Each system has its own tailored (architecture-specific) decode module to figure out the meaning of the op-codes to generate control signals.

Example decoder

There is more than one possible way to build a decoder.

For example, they could be:

1. Combinatorial logic
2. DMUX-based
3. Lookup-based

Instruction	Input op-code	Control signal
`ZERO`	00	0001
`ADD`	01	0010
`SUBTRACT`	10	0100
`MULTIPLY`	11	1000

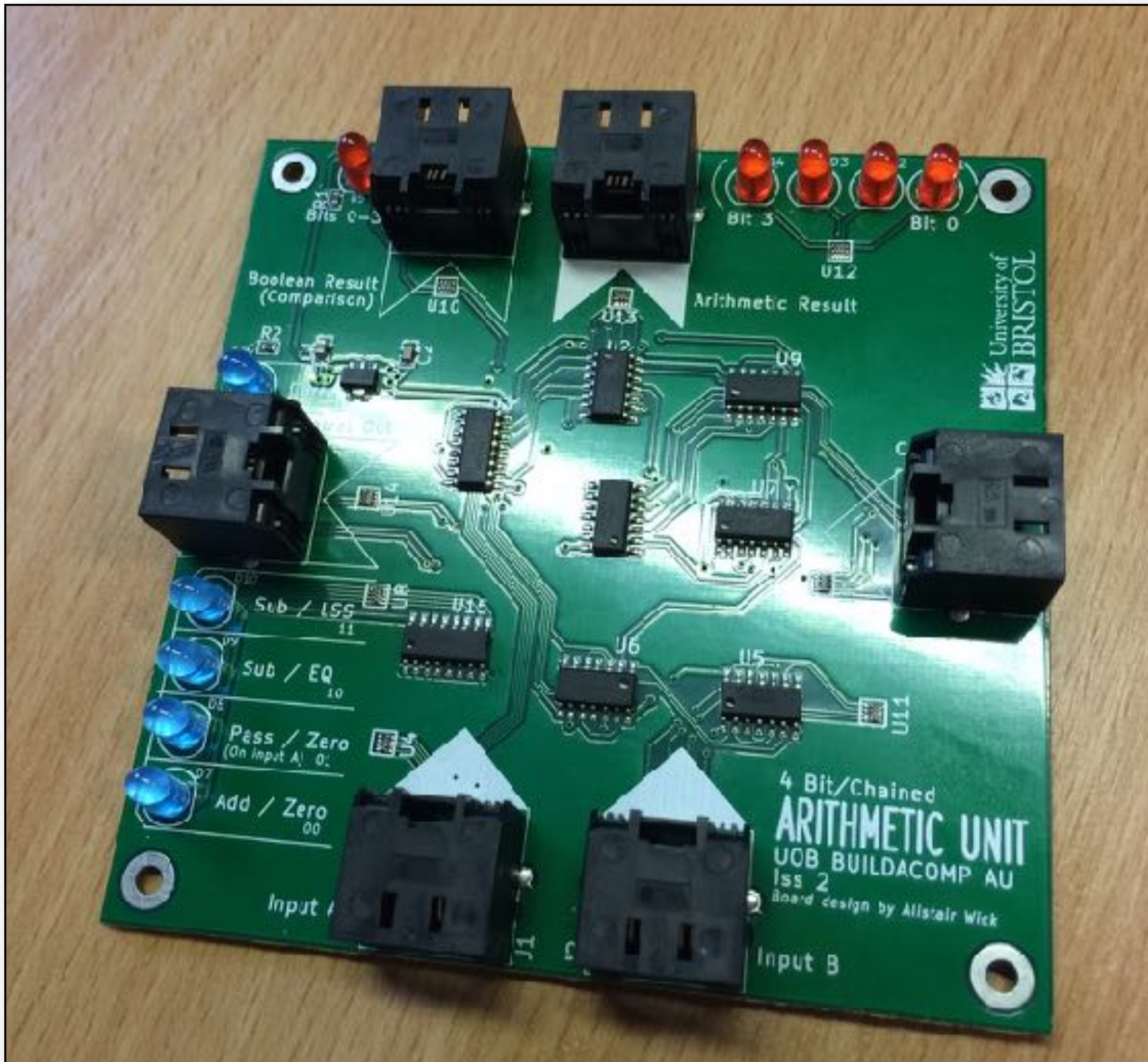
1. Data, control and instructions

Recap

- How data and control information get processed.
- What control information looks like.
- An introduction to instructions.
- How instructions are encoded as op-codes.
- How op-codes can be decoded.

Labs (Friday)

Make a real, physical circuit that performs binary logical operations and stores outputs in a register.



Next lecture

- What is memory?
- The execution cycle.