

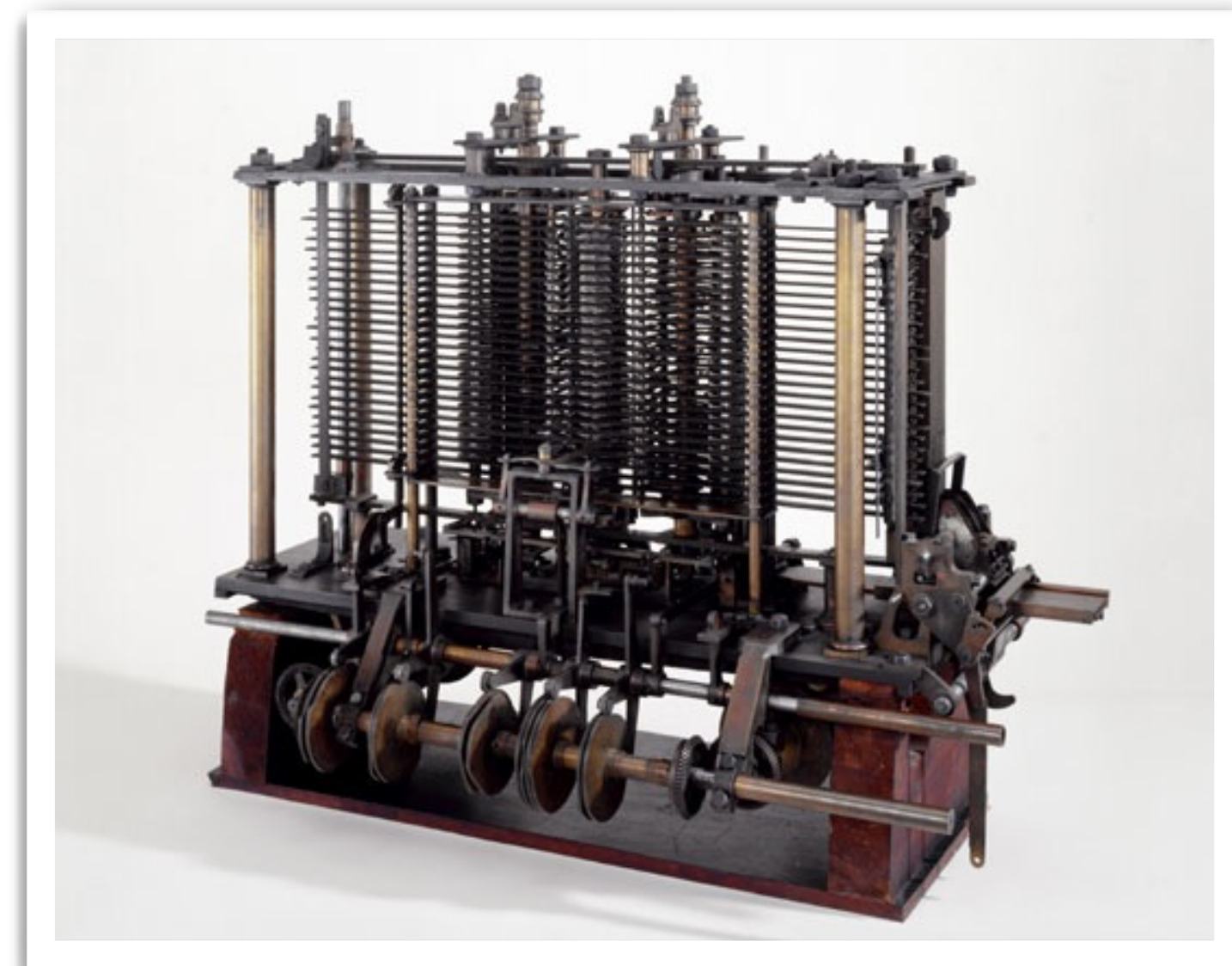


Computer Systems and Professional Practice

Professor Matthew Leeke
School of Computer Science
University of Birmingham

Topic 2 - Data Representation

Work It Out Yet?



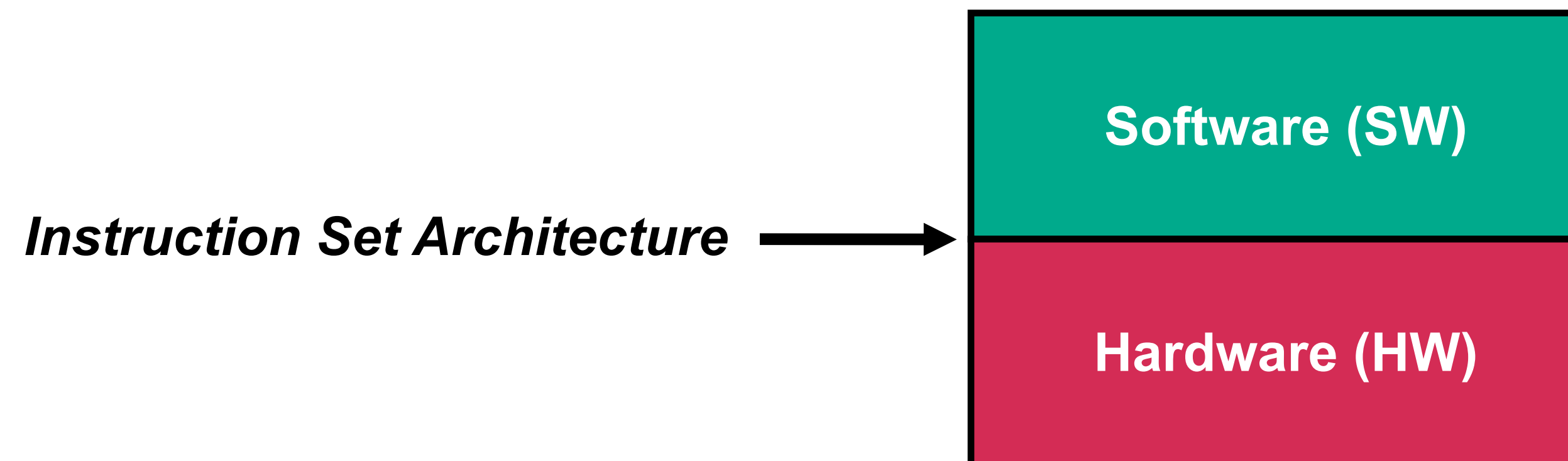
We Should Talk About Computers

How do we define a computer?

How can we build a computer?

Computer architecture is fundamental

Design of layers of abstraction and the intervening interfaces



Session Outline

Three key aspects of data representation in this lecture

Part 01_2 : Representation and Number Systems

Part 10_2 : Addition and Negative Numbers

Part 11_2 : Fractional Numbers

Part 01₂: Representation and Number Systems

Getting Started - Numerical Representations

Tally marks	I, II, III, IIII, IIII, IIII I, IIII II
Roman numerals	I, II, III, IV, V, VI, VII, VIII, IX, X, XI
Decimal integers	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
Fractions	1/2, 1/4, 2/16, 100/1100, 50/250
Scientific	2.998x10 ⁸ m/s (\approx 299792458 m/s)
Decimal fractions	23.5812, 5.07, 4.20, -21.479
Binary	0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011
Octal	0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12 ... 16, 17, 20, 21
Hexadecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12 ... 18, 19, 1A

One Value, Many Representations

In an abstract sense we all know what we mean when we talk about the number "7"

Not everyone would be able to represent the number "7" in binary, octal or hexadecimal

Symbols must be known

Even then there are issues relating to representation

The ideas of value and representation are different

Value is necessarily independent of representation

Roman	Decimal	Binary	Octal	Hex
?	0	0	0	0
I	1	1	1	1
II	2	10	2	2
III	3	11	3	3
IV	4	100	4	4
V	5	101	5	5
VI	6	110	6	6
VII	7	111	7	7
VIII	8	1000	10	8
IX	9	1001	11	9
X	10	1010	12	A
XI	11	1011	13	B
XII	12	1100	14	C
XIII	13	1101	15	D
XIV	14	1110	16	E
XV	15	1111	17	F
XVI	16	10000	20	10
XX	20	10100	24	14

Bases

What are the available symbols?

Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 : Base 10

Octal: 0, 1, 2, 3, 4, 5, 6, 7 : Base 8

Hex: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F : Base 16

Binary: 0, 1 : Base 2

Important to distinguish between representations in use

$$16_{10} = 10000_2 = 20_8 = 10_{16}$$

$$8_{10} = 1000_2 = 10_8 = 8_{16}$$

Positional Representation - Decimal

Regardless of how you were taught the decimal number system this calculation is implicit every time you talk about numbers in decimal

£	3	2	3	7
Position:	Thousands	Hundreds	Tens	Units
=	1000	100	10	1
=	10^3	10^2	10^1	10^0
Value =	$3 \times 10^3 +$	$2 \times 10^2 +$	$3 \times 10^1 +$	7×10^0

$$value = \sum_{i=0}^{N-1} symbol_i \times base^i$$

N: The number of columns
i: The index of position

Positional Representation - Binary

Binary number system operates under the same principles as decimal number systems, just with alternative symbols and position values

1011₂	=	1	0	1	1
Position:		2³	2²	2¹	2⁰
	=	8	4	2	1
Value	=	1 x 8 +	0 x 4 +	1 x 2 +	1 x 1
	=	11₁₀			

$$value = \sum_{i=0}^{N-1} symbol_i \times base^i$$

N: The number of columns
i: The index of position

Counting in Binary

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
•	•	•	

This is how we count in binary – but why is it this particular pattern, and how do we construct it?

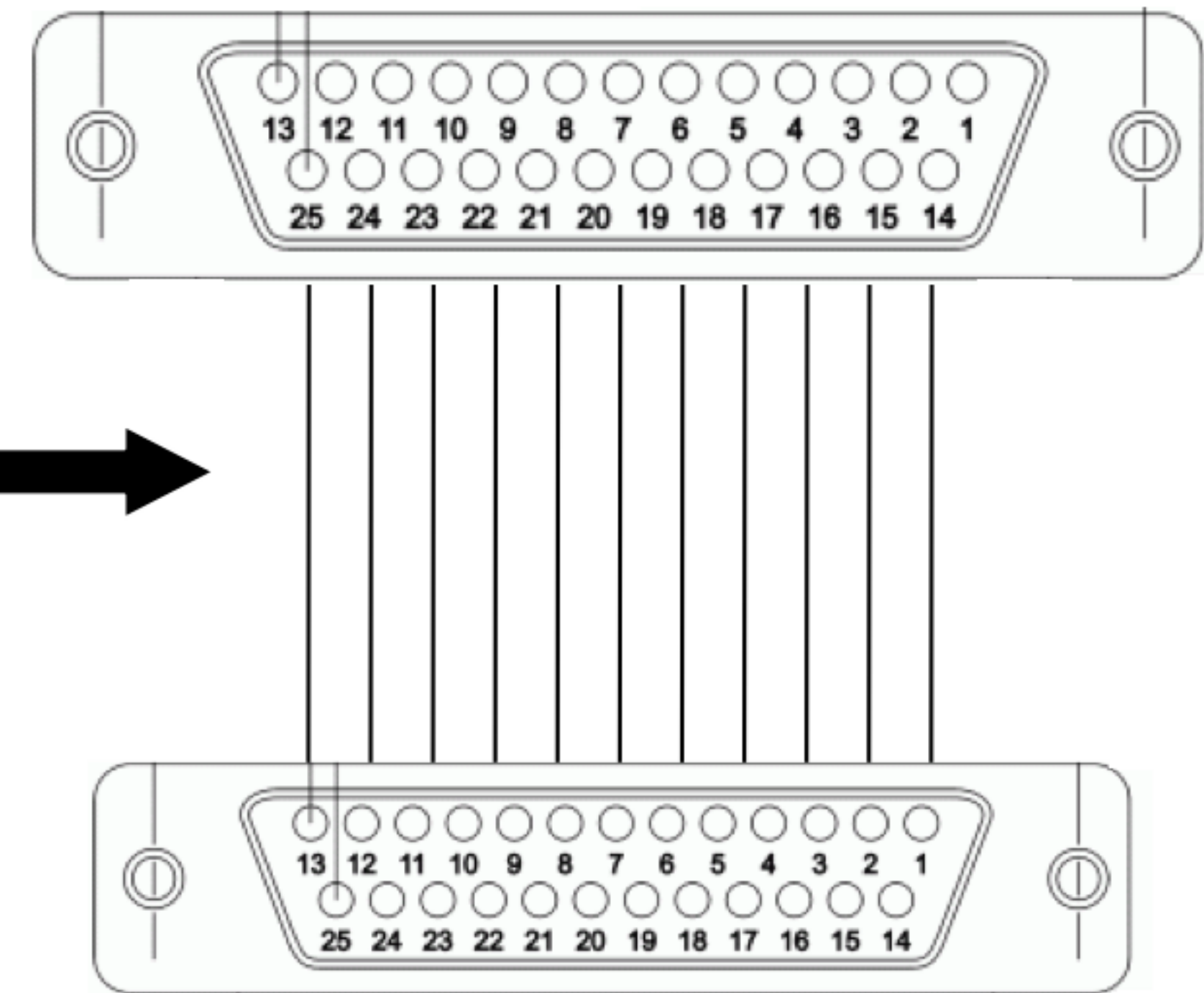
Electrical Circuit Representation

TTL (transistor-transistor logic)

0V – 0.8V represents 0

2.4V – 5V represents 1

Bus 



Parallel bus

Collection of wires communicating a value between sub-circuits

Bits, Bytes, Words and Bus Sizes

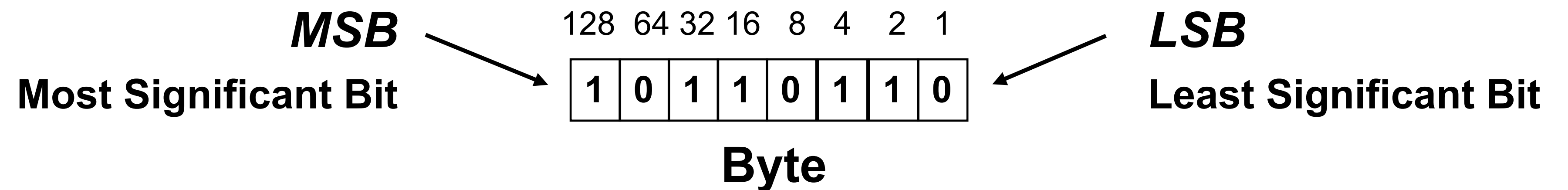
Bit - Binary digit (values 0_2 or 1_2 inclusive)

Byte - 8 bits (values 0_{10} to 255_{10} inclusive)

Word - The number of bits a machine can process simultaneously

4 bits = half a byte = nibble (values 0 - 15_{10})

most significant bit (MSB) and least significant bit (LSB) of a word are the bits with the highest and lowest place values respectively



A Brief History of Processor Representation

Intel 4004 (1971) - The first commercially available microprocessor had a word length of 4-bits

MOS 6502 and Zilog Z80 (~1975) - Popular 8-bit processors in early personal computers, such as the Apple II (6502) and TRS-80 (Z80).

Motorola 68030, PowerPC G4, and Intel Pentium (upto 2004) - Processor at the era of 32-bit architecture, being widely used in personal computers and workstations.

IBM PowerPC G5 and x86-64 (e.g., Intel Core 2 and AMD64, introduced in 2003-2004) - The transition to 64-bit, offering higher performance, larger addressable memory space, and greater efficiency in modern computing.

Modern Processors (2020s onwards) - Processors like the Intel Core i9, AMD Ryzen, and Apple M-series continue to evolve the 64-bit architecture (with multi-core designs, advanced parallel processing, and coprocessors for AI and machine learning) and be informed by design elements of ARM and RISC-V

Sizes of Symbols

One octal symbol can represent 3 bits

$$111_2 = 7_8$$

$$010\ 100\ 011_2 = 243_8$$

One hex symbol can represent 4 bits

$$1111_2 = 15_{10} = F_{16}$$

$$1010\ 0011_2 = A3_{16}$$

One decimal symbol requires ≈ 3.3 bits, so hex and octal are more convenient than decimal when describing values on a bus

Converting from Decimal to Binary

Repeatedly divide the number by the base required, e.g., 2 for binary.

Record the remainder, e.g., for binary it is 0 if the number was even and 1 if it was odd

Quotient	Remainder
----------	-----------

163	-
81	1
40	1
20	0
10	0
5	0
2	1
1	0
0	1



$163_{10} = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1_2$

Converting from Decimal to Octal or Hex

The division method can be used to convert to other bases

Divide by the required base, e.g., 8 for octal and 16 for hexadecimal

It may be easier to convert into binary first and then into the required base

$$\begin{aligned} 23_{10} &= 16 + 4 + 2 + 1 \\ &= 10111_2 \\ &= 27_8 \quad (010 \ 111_2) \\ &= 17_{16} \quad (0001 \ 0111_2) \end{aligned}$$

Pop Quiz - Questions

Covert 113_{10} to binary

Covert $011\ 110\ 111_2$ to decimal

Covert $110\ 101\ 011_2$ to octal

Covert $1100\ 0111\ 0110_2$ to hexadecimal

Pop Quiz - Answers

Covert 113_{10} to binary = **01110001_2**

Covert $011\ 110\ 111_2$ to decimal = **247_{10}**

Covert $110\ 101\ 011_2$ to octal = **653_8**

Covert $1100\ 0111\ 0110_2$ to hexadecimal = **$C76_{16}$**

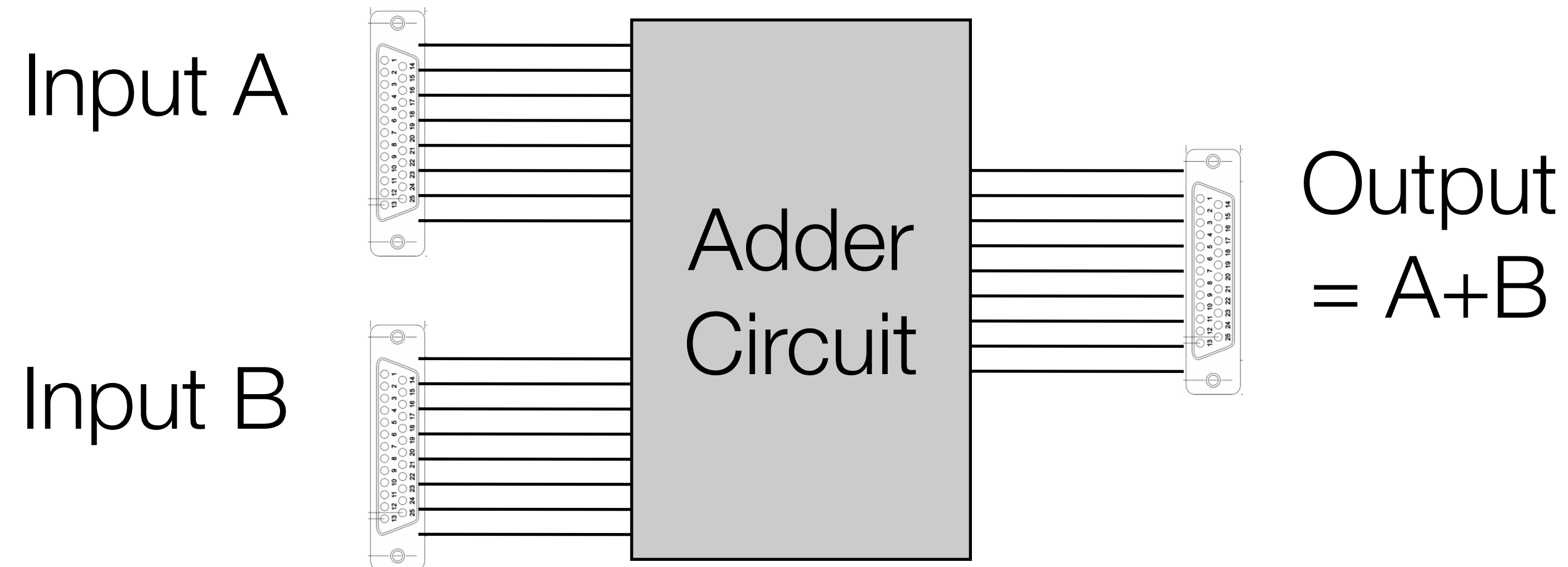
Part 10₂: Addition and Negative Numbers

The Concept of Addition

A simple example - two inputs, add their values and get a value as a result

What is difficult about addition? Don't we do this every day?

We're dealing with values so why does representation make a difference?

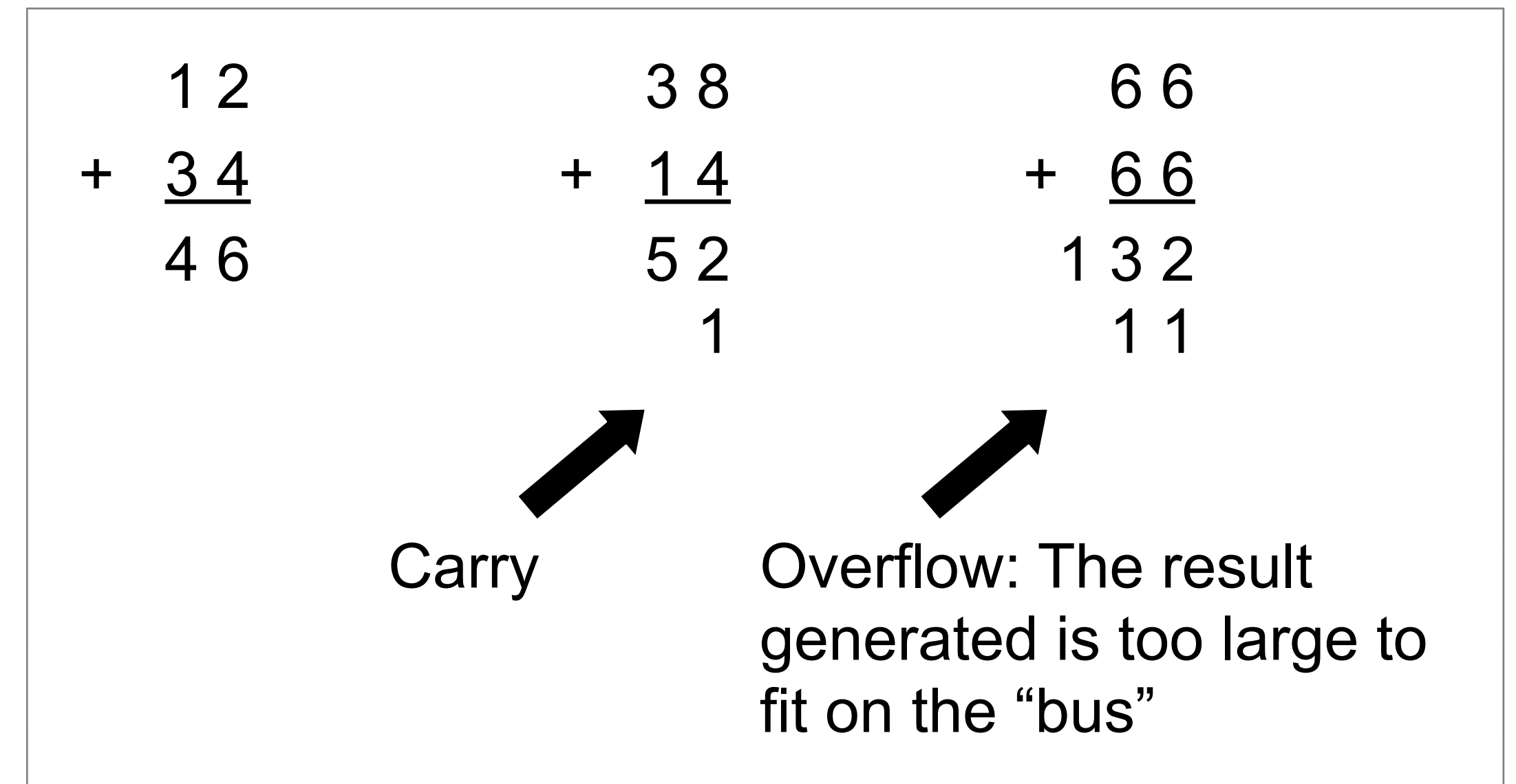


Addition in Decimal

You should be able to see what is going the examples above

It's interesting to consider what might happen when you get an overflow

Naively exceeding the capacity of a bus may mean that an incorrect message could be transmitted or that an additional message must be transmitted to account for the overflow




Addition in Binary

The overflow problem doesn't go away when you switch base

In fact the example of overflow shown above emphasise the need to account for the problem in the context of computer systems

$100 = 4$	$010 = 2$	$011 = 3$	$011 = 3$	$111 = 7$
$+ \underline{010} = 2$	$+ \underline{011} = 3$	$+ \underline{001} = 1$	$+ \underline{011} = 3$	$+ \underline{001} = 1$
$110 = 6$	$101 = 5$	$100 = 4$	$110 = 6$	$1000 = 8$
	1	11	11	111



Overflow: The result generated is too large to fit on the "bus"

We will return to the issue of overflow in later lectures

Subtraction as Addition

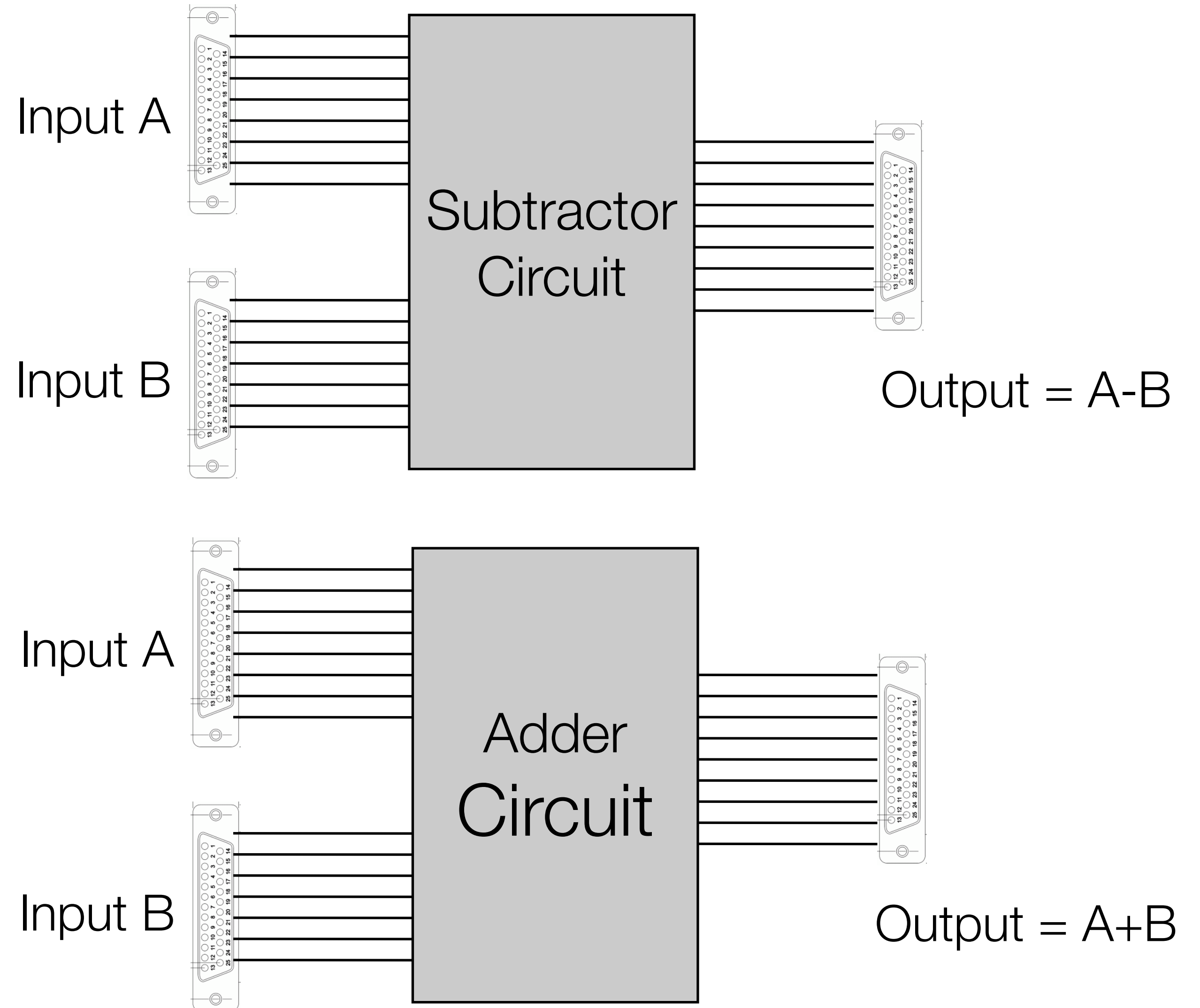
Two viable options for achieving subtraction

Design a dedicated subtractor circuit

Create a representation for negative numbers (A and B can be negative) and extend the adder

You will soon design and construct logic circuits for addition and subtraction

For now it is enough to appreciate that we can use representation to permit subtraction



Signed Magnitude Representation

The MSB is an indicator, i.e., indicates it is negative

The value of the remaining bits are calculated as before

1101_{2SM}:	1	1	0	1
Position:	negative?	2²	2¹	2⁰
=	negative?	4	2	1
Value:	-1 x	(1 x 4 +	0 x 2 +	1 x 1)
=	-5₁₀			

Two's Complement Representation

The most common signed integer representation in modern computing

The MSB has the same value as in the binary positional representation, i.e., 2^{N-1} , but it is negative, i.e., -2^{N-1}

1101_{2TC}:	1	1	0	1
Position:	-2³	2²	2¹	2⁰
=	-8	4	2	1
Value:	(1 x -8 +	1 x 4 +	0 x 2 +	1 x 1)
=	-3₁₀			

Two's Complement Properties

Can represent more negative numbers than positive

The range is asymmetric

$$1000_{2TC} = -8$$

$$0111_{2TC} = +7$$

For N bits, the range is -2^{N-1} to $(2^{N-1}-1)$

Unique zero

Signed magnitude has both +0 and -0

Forming Two's Complement

To form the bit pattern for a negative number in two's complement:

Take the representation of the positive number

Invert the bits, ensuring there are enough bits for the MSB to be the sign

Add 1, ignoring any overflow

What is the two's complement bit pattern for -9?

$$+9_{10} = 1001_2$$

Ensure enough bits: 00001001_2

Invert: 11110110_2

Add 1: 11110111_2

$$-9_{10} = 11110111_{2TC}$$

Addition in Two's Complement

Addition is almost the same as we've seen previously

Just as before but ignore any overflow

$00001001 = +9$	$00001101 = +13$	$00000101 = +5$
$+\underline{11110111} = -9$	$+\underline{11110111} = -9$	$+\underline{11110111} = -9$
$00000000 = 0$	$00000100 = 4$	$11111100 = -4$
11111111	11111111	111

Part 1 1_2 : Fractional Numbers

Fixed Point Representation

Add columns to denote the fractional parts of a value

10.11₂	=	1	0	1	1
Position:		2¹	2⁰	2⁻¹	2⁻²
	=	2	1	0.5	0.25
Value	=	(1 x 2 +	0 x 1 +	1 x 0.5 +	1 x 0.25)
	=	2.75₁₀			

A Problem with Representation?

How could we represent very large and very small numbers using the representations we've covered so far?

The mass of an electron is 0.000000000000000000000000000000000000009g

The mass of the sun is 200g

We would need an incredibly large bus to accurately represent these values in a computer system

Floating Point Representation

Scientific notation allows small and large values to be written succinctly

The speed of light is 3×10^8 m/s \approx 299,792,458 m/s

Floating point uses the same principles as scientific notation

Express numbers as (mantissa $\times 10^{\text{exponent}}$)

The mantissa represents the detail of the value to a certain precision

The exponent represents the magnitude of the value

IEEE Floating Point

Floating point representation trades precision for range

Extending range at the expense of reducing precision

IEEE Standard 754 is widely used and specifies levels of binary precision

Single precision (using 32 bits)

Double precision (using 64 bits)

Quad precision (using 128 bits)



For example, single precision uses 1 bit for the sign (s), 8 bits for the exponent (e) and 23 bits for the mantissa (m)

