CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Number Systems and Their Conversion

—Dr H. Asad

Positional Number Systems

- Numbers can be represented in any **base**
- Humans are used to **base 10**
  - The digits used in base 10 are: 0,1,2,3,4,5,6,7,8 and 9
- We can analyse numbers based on position and powers of 10

Example

$3245 = 3 \times 10^3 + 2 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$

$= 3 \times 1000 + 2 \times 100 + 4 \times 10 + 5 \times 1$

$= 3000 + 200 + 40 + 5$

CITY
UNIVERSITY OF LONDON
EST 1894

Similarly, we can write:

7896

Thousand	Hundred	Ten	Units
7	8	9	6

7896 = 7000 + 800 + 90 + 6

7896 = 7 x 1000 + 8 x 100 + 9 x 10 + 6 x 1

7896 = $7 \times 10^3 + 8 \times 10^2 + 9 \times 10^1 + 6 \times 10^0$

➔ **This can be applied to any base, not just 10.**

CITY
UNIVERSITY OF LONDON
EST 1894

Indicating the base

- We should subscript numbers with their base (e.g., decimal numbers with ten and binary numbers with two)
  - 0011_2 or 0011_{two} or 0011_b (which equals 3_{10} or 3_d)
  - 11_{10} or 11_{ten} or 11_d (which equals 1011_2)
- ten, d and 10 in the above alternative representations indicate base 10

CITY
UNIVERSITY OF LONDON
EST 1894

Positional Numbers and Base (a systematic approach)

In any number base, the value of the i th digit d in a number can be written as:

d x base ^{i}

where i starts at position 0 of the number and increases to the left.

In 1247_{10} what is represented by 2?

- The position *i* of 2 is 2 (as the position starts from right to left and the first position is 0)
- The digit d is 2
- Hence, the value represented by the 2 digit in the second position is
- $2 \text{ X } 10^2 = 200$

CITY UNIVERSITY OF LONDON EST 1894

Binary Numbers

- Base 2 is usually used in computers **(Binary)**
  - This is because of the natural correspondence between high/low signals and 1/0, respectively
  - The digits used in this base are 0 and 1
- A single binary digit is known as a **bit**
  - bit = **bi**nary digi**t**
- Example, binary numbers with two bits:
  - 00 → 0 (decimal)
  - 01 → 1 (decimal)
  - 10 → 2 (decimal)
  - 11 → 3 (decimal)

CITY
UNIVERSITY OF LONDON
EST 1894

Example:
Decimal Number vs Binary Number

Consider the number 13_{10}.

We can write this in the 10-base as:

$13_{10} = 1 \times 10^1 + 3 \times 10^0$ (as $1 \times 10^1 + 3 \times 10^0 = 1 \times 10 + 3 \times 1 = 13$)

And in the 2-base, following the formula ***d x base*** ^{i}, as

1101_2: $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

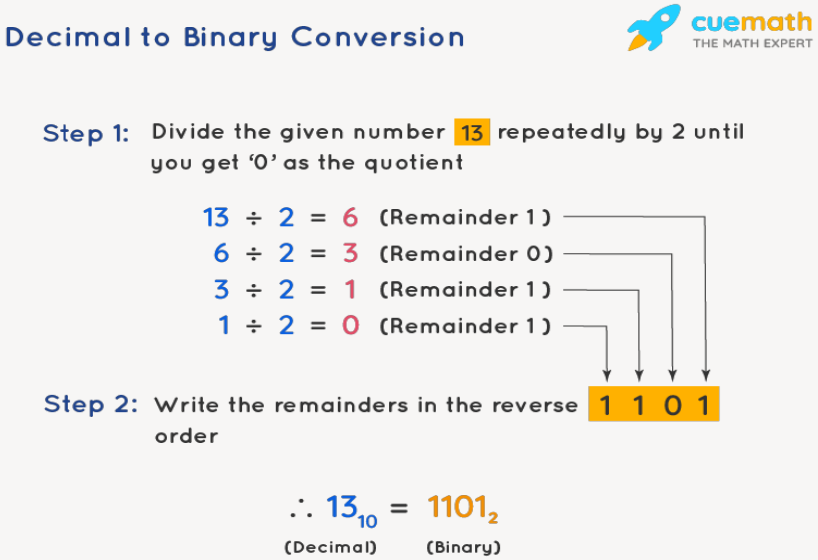
$= 1 \times 8_{10} \quad + 1 \times 4_{10} \quad + 0 \times 2_{10} \quad + 1 \times 1_{10}$

CITY
UNIVERSITY OF LONDON
EST 1894

Decimal Number to Binary Number conversion

- **Step 1:** Divide the given decimal number by 2 and note down the remainder.
- **Step 2:** Divide the obtained quotient by 2, and note the remainder again.
- **Step 3:** Repeat the above step until you get 0 as the quotient.
- **Step 4:** Write the remainders in such a way that the last remainder is written first, followed by the rest in the reverse order.
- **Step 5:** Result number is the binary value of the given decimal number.

Decimal Number to Binary Number conversion



Hexadecimal

- Long Binary numbers are hard to read!
- One Hexadecimal (base 16) digit can represent 4 bits

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

Hexadecimal (cont'd)

- Uses the characters 0123456789ABCDEF to represent the numbers 0 .. 15 corresponding to the 4-bit binary values 0000 .. 1111

$$1357\ 9BDF_{hex}$$
$$0001\ 0011\ 0101\ 0111\ 1001\ 1011\ 1101\ 1111_{two}$$

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

CITY UNIVERSITY OF LONDON EST 1894

Octal

- Octal is a number representation system with **base 8**
- The octal representation uses the characters: 0 1 2 3 4 5 6 7
- Each Octal character corresponds to three bits in binary

What is 27_{ten} in Octal?

Answer:

1. 27 / 8 →24 = 3 x 8 and the remainder is 3
2. 3 / 8 → 0 and the remainder is 3, we stop
3. Write the remainders from bottom to top: 33_8
   - $33_8 = 3 \times 8^1 + 3 \times 8^0$

Exercises

- Convert 27_{10} into binary, octal and hexadecimal

  Hint:
  - Divide by the base of the target representation
  - Compose number as sequence of remainders
- Convert 100111101101 hexadecimal

  Hint:
  - transform subsequences of three bits for octal
  - transform subsequences of four bits for hexadecimal

CITY
UNIVERSITY OF LONDON
EST 1894

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	1B
28	11100	34	1C
29	11101	35	1D
30	11110	36	1E
31	11111	37	1F
32	100000	40	20

100111101101

CITY
UNIVERSITY OF LONDON
EST 1894

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Signed & Unsigned no representation

—Dr H. Asad

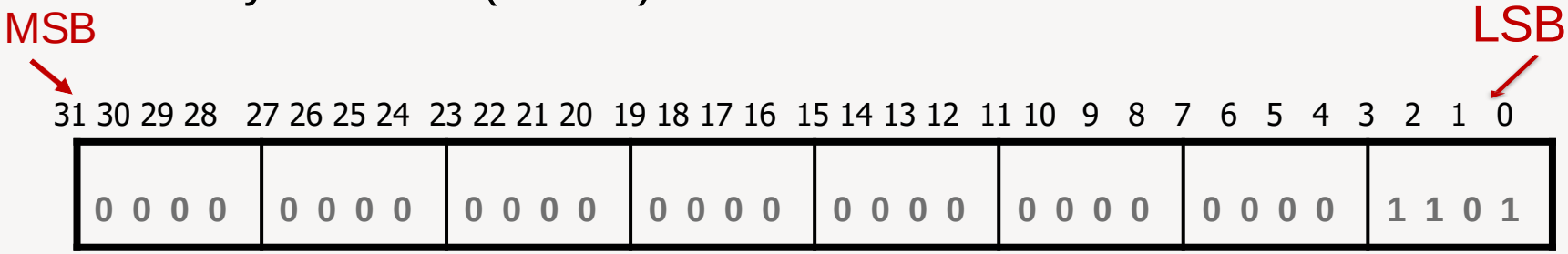
Bits, Bytes and Words

- **Bit**: (binary digit) the most basic unit of information in a digital computer
  - e.g. 1101 4 **bits (also called a nibble)** are used
- **Byte**: a collection of 8 bits
  - e.g. 11010001
  - **Word**: two or more adjacent bytes that are addressed and handled collectively
  - E.g. 11010001 11110000 is a word with two bytes or 16-bits

- The word size represents the data size that is handled more efficiently by a particular architecture
- The number of bits we use to represent a number defines the largest number we can store
- **In contemporary computers, words with 32 bits (i.e., 4 bytes of 8 bits each) or 64 bits (i.e., 8 bytes of 8 bits each) are usual**

Representing Positive Integers

- Binary numbers (32 bits)



- **Least significant bit (LSB)** is the "smallest" bit in a word, i.e., the bit at position 0 (representing the number: LSB x 2^0, which may be the decimal 1 or 0)
- **Most significant bit (MSB)** is the "biggest" bit in a word, i.e. the bit at position 31 (representing the number: LSB x 2^{31}, which may be the decimal$_{13}$ number 2,147,483,648 or ???)

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

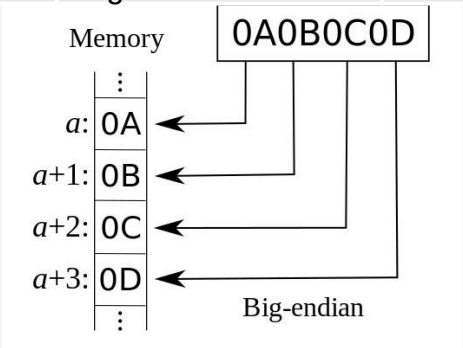
Floating point and Non-Numeric Data Representation

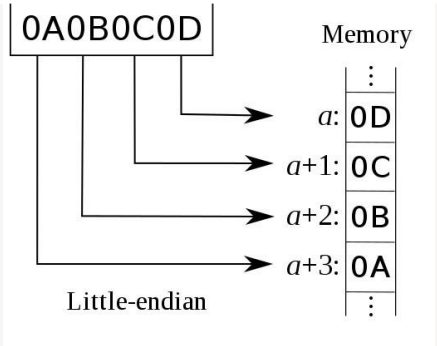
—Dr H. Asad

Endianness

- **To identify the order in which bytes that constitute words are stored in memory**
- **Endianness** defines this (i.e., the order in which bytes are arranged in memory)
- When programming at a low level, **it is important to know what your machine supports!**

Endian	First Byte	Last Byte	Mnemonic	Example computers
Big	Most Significant	Least Significant	Normal way of writing numbers	M68000, IBM/360,motorolla
Little	Least Significant	Most Significant	Arithmetic calculation order	X86, Z80,intel





Some computers are bi-endian (ARM, PowerPC)

Representing Positive Integers: The simple pattern

Question: Assume that we have a hypothetical 3-bit word, how many different patterns of 0s and 1s can we have?

Answer:

We can represent 2^3 different 3-bit patterns, i.e.,

000 001 010 011 100 101 110 111

This represents numbers from 0 to 7 ($7 = 2^3 - 1$)

n-bits can represent 2^n different numbers from 0 to 2^n - 1

Question: If we have a 32-bit word, how many different patterns of 0s and 1s can we have?

Answer: We can represent 2^{32} different 32-bit patterns. This represents numbers from

0 to ($2^{32} - 1$), i.e., 0 to 4,294,967,296 – 1 (or to $4,294,967,295_{10}$)

Representing Positive Integers (cont'd)

32-bit word, how many different patterns of 0s and 1s can we have ?

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = 3_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{two} = 4_{ten}$

… … …

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

CITY UNIVERSITY OF LONDON EST 1894

Representing Positive Integers (the general case)

- A 32 binary number of the form: $n=x_{31}x_{30}x_{29}\ldots x_1x_0$

  can be represented as the bit value times a power of 2

- The general formula:

  $$(x_{31} * 2^{31})+(x_{30} * 2^{30})+(x_{29} * 2^{29})+\ldots+(x_1 * 2^1)+(x_0 * 2^0)$$

  (x_i means the i th bit of n)

- Example:

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = 3_{ten}$

… … …

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$ …

Representing Negative Integers

- So far have involved only unsigned integer numbers
- **They cannot represent negative integer numbers**
- To represent negative and positive integer numbers, we need to have signed integer numbers.

- **Signed binary integer numbers may be expressed by:**
  - **Signed magnitude**
  - **One's complement**
  - **Two's complement**

CITY
UNIVERSITY OF LONDON
EST 1894

Signed-Magnitude Representation

- Assign **one bit to represent the sign**
  (typically the left most digit is used)
- Use the **rest of the bits** to **represent magnitude (value)**

  Example: Assume that 1 represents "-" and 0 represents "+" Then consider 8-bit words:
  - -1 is **1** 000 0001
  - +1 is **0** 000 0001

- What is the range of numbers we can represent with N bits?

$-(2^{(N-1)}-1)$ to $2^{(N-1)}-1$

- This representation has some **disadvantages**:
  - Using a **special sign bit** means that we can represent **fewer numbers**
  - Both +0 (e.g. 0000 0000) and -0 (e.g. 1000 0000) **are valid**, leading to programmer headaches
  - Arithmetic circuits are complicated by the calculation of sign (as we will see later)

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

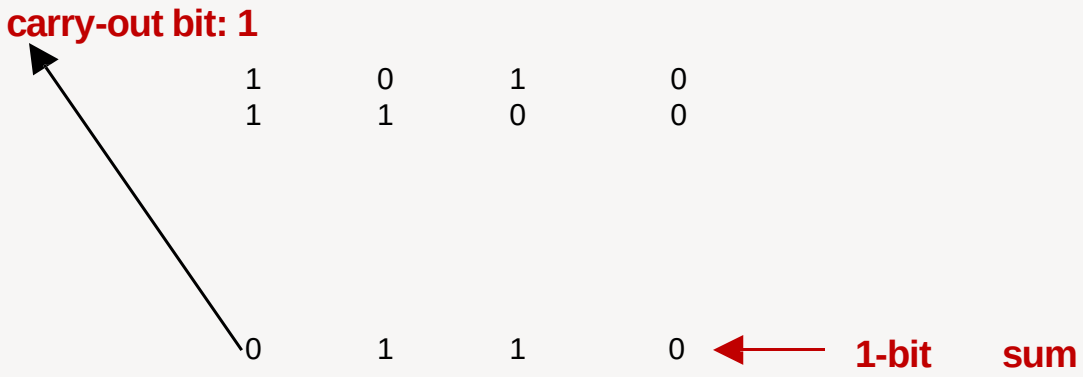
IN1006

Arithmetic Operation Part 1

—Dr H. Asad

1-bit Binary Addition

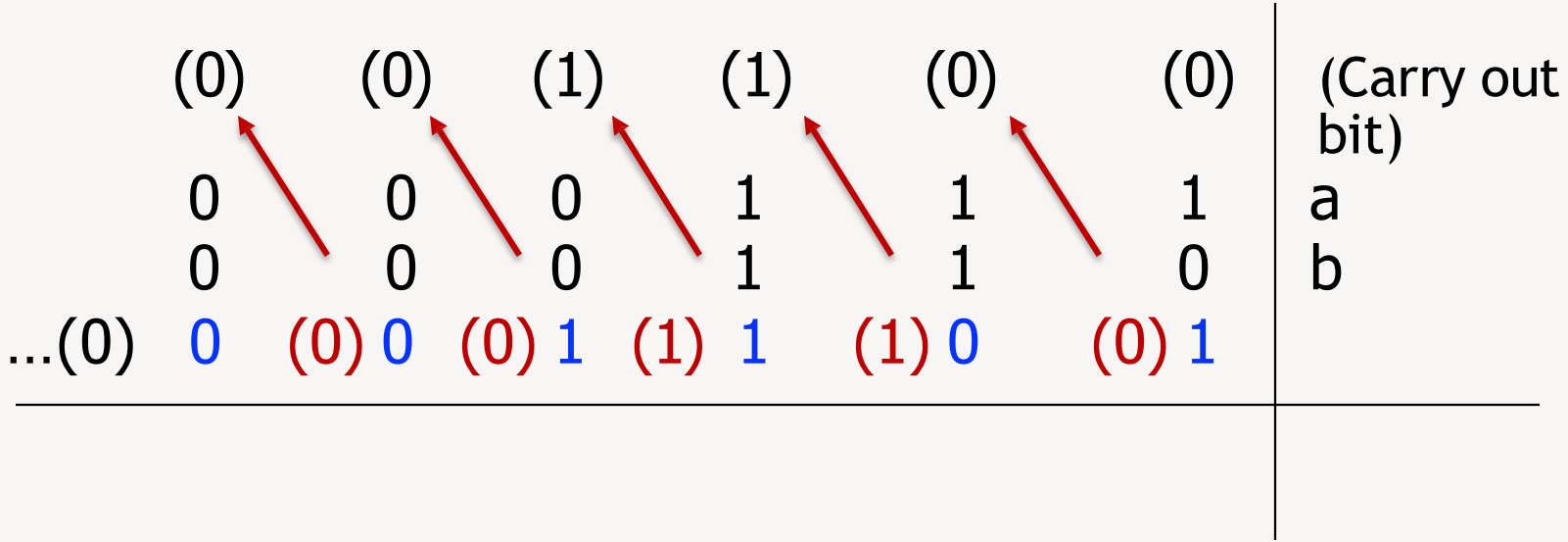
- two 1-bit values gives four cases:



- The simplicity of this system makes it possible for digital circuits to carry out arithmetic operations.

Binary Addition: how the “carry out” bit is propagated?

- 2 bits show all possibilities:



Binary Addition: an Example

- **Example:** using signed magnitude binary arithmetic, find the sum of 75 and 46.

CITY
UNIVERSITY OF LONDON
EST 1894

Binary Addition: an Example

- **Example:** using signed magnitude binary arithmetic, find the sum of 75 and 46.
- **Solution:**
  1) convert 75 and 46 to binary, and arrange as a sum, but separate the (positive) sign bits from the magnitude bits.
  2) Just as in decimal arithmetic, we find the sum starting with the rightmost bit and work left.
  3) In the second bit, we have a “carry” bit, so we note it above the third bit.
  4) The third and fourth bits also give us “carries”.
  5) Once we have worked our way through all eight bits, we are done.

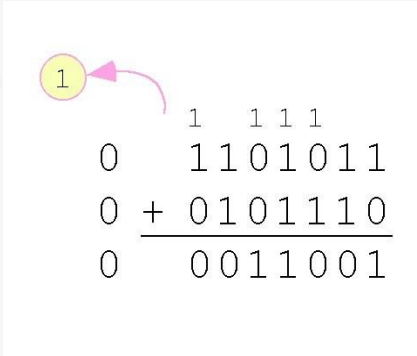
sign bit is separated

```
      1 1 1
0   1 0 0 1 0 1 1
0 + 0 1 0 1 1 1 0
0   1 1 1 1 0 0 1
```

Binary Addition: Example

- **Example:** Using signed magnitude binary arithmetic, find the sum of 107 and 46.

Overflow bit



- We see that the carry from the seventh bit **overflows** and is discarded
- BUT then the addition gives us an **erroneous result: 107 + 46 = 25**.

CITY
UNIVERSITY OF LONDON
EST 1894

Overflow

- It occurs if the result of “add” (or any other arithmetic operation) cannot be represented by the available hardware bits.

- **While we can’t always prevent overflow, we can always detect overflow.**

- **Note:** This is **physical overflow**.

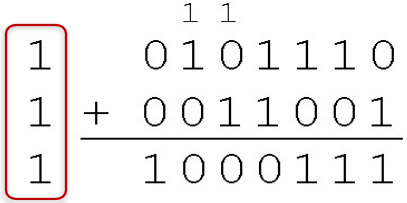
- What happens to the bits that “fall off the left end” is dependant on implementation

CITY
UNIVERSITY OF LONDON
EST 1894

Binary Addition for Negative Numbers

- **Example:** Using **signed magnitude** binary arithmetic, find the sum of - 46 and - 25.
- Because the signs are the same, all we do is add the numbers and supply the negative sign when we are done.

sign bit is separated



Binary subtraction (at the basic bit level)

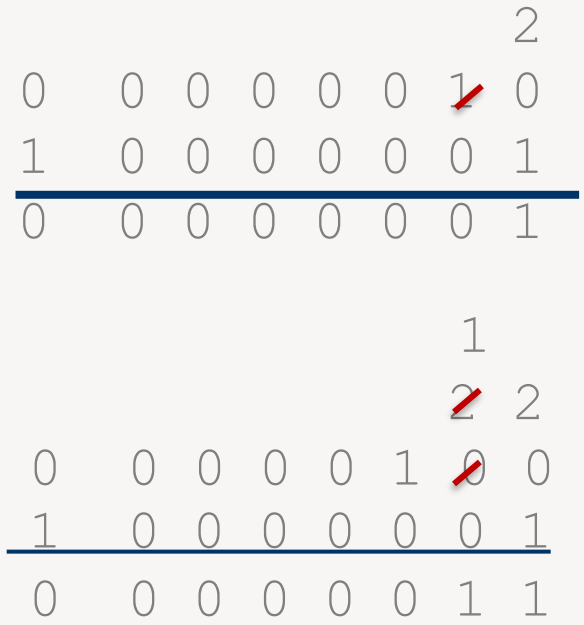
four cases:

	1	0	1	0
–	1	1	0	0
	0	1*	1	0

* But requires borrowing a digit from a higher position (left bit)

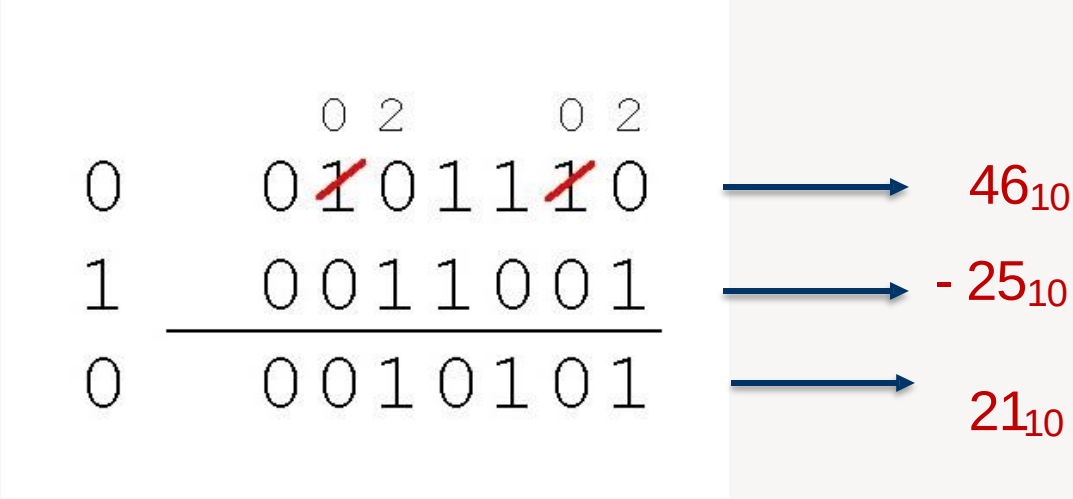
Binary subtraction: Examples with whole numbers

- Determine which operand has the largest magnitude
- The sign of the result gets the sign of the number that is larger
- The magnitude is obtained by subtracting the smaller one from the larger one
  - It might require **borrowing from the next non zero bit (to the left)** or two, three etc. bit to the left (until we find a non zero bit)
  - **When we borrow from a higher bit to the next lower position, we borrow "2"** and reduce the bit that we borrowed from by "1"



Binary subtraction:

- **Example:** Using **signed magnitude** binary arithmetic, find the sum of 46 and – 25.



CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Arithmetic Operation Part 2

—Dr H. Asad

Complement based representation (rationale)

- Signed magnitude representation easy for people to understand, but requires complicated computer hardware.
- Another disadvantage of signed magnitude is that it allows two different representations for zero: positive zero and negative zero

➔ For these reasons (among others) computer systems employ **complement systems** for numeric value representation.

CITY
UNIVERSITY OF LONDON
EST 1894

Two’s complement (what is it?)

- Two’s complement is the radix complement of the binary system:

> the radix complement of a non-zero number N in base r with d digits is
> $r^d - N$ for $N \neq 0$, and 0 for $N = 0$.

Two's complement (what is it?)

- Makes the hardware simple
- MSB is regarded as sign bit (0:+ve, 1:-ve)

Examples:

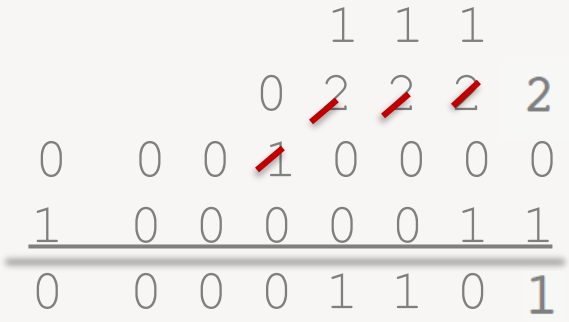
- Radix complement of 2 in base 10 with 3 digits

(or 10's complement of 2 with 3 digits)

is: $10^3 - 2 = 1000 - 2 = 998$

- 2's complement of 4-bit number 0011_2

is: $2^4 - 0011_2 = 10000_2 - 0011_2 = 1101_2$

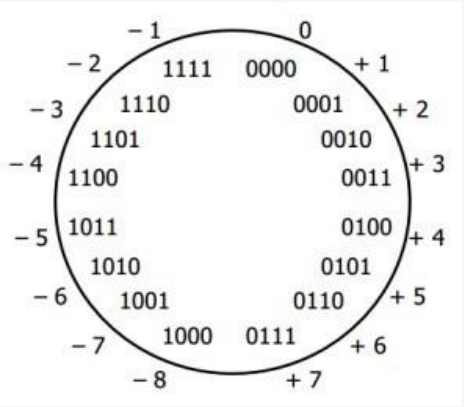


Visualize two’s complement numbers on a 4 bit number

To express a value in two’s complement representation:

1) **If the number is positive**, just convert it to binary and you’re done.

2) **If the number is negative**, **flip bits and then add 1.**



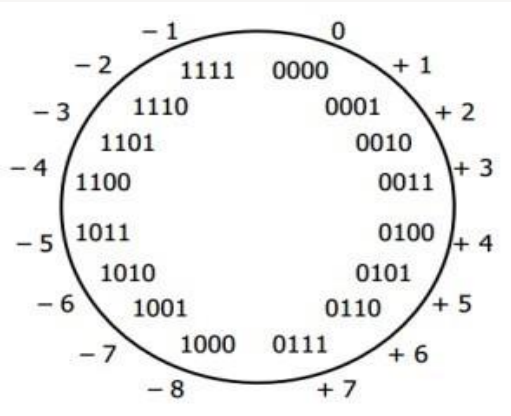
Visualize two's complement numbers on a 4 bit number

Number 4_{10} is 0100_2

Number -4_{10} would be represented using 2's complement as follows

Binary number	0100
Inverse	1011
Add 1	1011 +1
2's complement for – 4	1100

- 4_{10} is 1100_2 on two's complement system



Two's complement (some simple examples)

Decimal value	Binary value	1s complement	2's complement
3	00000011		00000011 No change as it is a positive number
-3	00000011 In 8-bit binary, 3	11111100 (flip bits)	11111101 (add 1) Adding 1 gives us -3 in two's complement

CITY
UNIVERSITY OF LONDON
EST 1894

Two’s complement (some more complex examples)

```
0000 0000 0000 0000 0000 0000 0000 0000 =  0
0000 0000 0000 0000 0000 0000 0000 0001 =  1
0000 0000 0000 0000 0000 0000 0000 0010 =  2

_______________________

0111 1111 1111 1111 1111 1111 1111 1101 =  2,147,248,645
0111 1111 1111 1111 1111 1111 1111 1110 =  2,147,248,646
0111 1111 1111 1111 1111 1111 1111 1111 =  2,147,248,647

1000 0000 0000 0000 0000 0000 0000 0011 = -2,147,248,645

1000 0000 0000 0000 0000 0000 0000 0010 = -2,147,248,646

1000 0000 0000 0000 0000 0000 0000 0001 = -2,147,248,647

_______________________

1111 1111 1111 1111 1111 1111 1111 1110 = -2
1111 1111 1111 1111 1111 1111 1111 1111 = -1
```

Two’s complement shortcut

1. Negation: find the **negative of a Positive binary number**: invert (flip) the bits and add one
2. Transforming a **negative decimal number** into 2’s complement:

   write the binary of the magnitude and invert (flip) the bits and add one
3. Transforming a **negative two’s complement number** into decimal: invert (flip) the bits and add one. Calculate the decimal number from the binary and add (-)

CITY
UNIVERSITY OF LONDON
EST 1894

Two’s complement shortcut (example)

Example: find the negative number of the decimal number 2.

```
0000 0000 0000 0000 0000 0000 0000 0010 = 2
```

Invert the bits and add one

```
  1111  1111  1111  1111  1111  1111  1111  1101
+ 0000  0000  0000  0000  0000  0000  0000  0001

  1111  1111  1111  1111  1111  1111  1111  1110 = -2
```

Two’s complement shortcut (example)

Example: find the decimal number of the two’s complement above (don’t forget the sign -2)

invert the bits and add one BUT going in the other direction

```
  0000  0000  0000  0000  0000  0000  0000  0001
+ 0000  0000  0000  0000  0000  0000  0000  0001

  0000  0000  0000  0000  0000  0000  0000  0010 = 2(adding the sign -> result = -2)
```

Exercise

- Convert the two's complement number into decimal $1111\ 1111\ 1111\ 1111\ 1111\ 1010_2$

1. Flip the bits:

   0000 0000 0000 0000 0000 0101

2. Add 1:

   0000 0000 0000 0000 0000 0001

   0000 0000 0000 0000 0000 0110 $= 6_{10}$

3. Check the **left-most bit of the original number** is 1 and so it's a negative number: **- 6**

Two's complement: some noteworthy points

- the positive half of the numbers, from 0 to $2{,}147{,}483{,}647_{ten}$ (2^{31}-1), are as before
- The pattern $1000\ldots0000_{two}$ represents the most negative number $-2{,}147{,}483{,}648_{ten}$ (-2^{31})
- This is followed by a declining set of negative numbers ($1000\ldots0001_{two}$) down to ($1111\ldots1111_{two}$)
- The number $-2{,}147{,}483{,}648_{ten}$ (-2^{31}) has no corresponding positive number 2^{31}

Binary addition and subtraction

Subtracting 6_{ten} from 7_{ten} is (7-6)=1:

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

Or **via addition using the two's complement** (7 + (-6)):

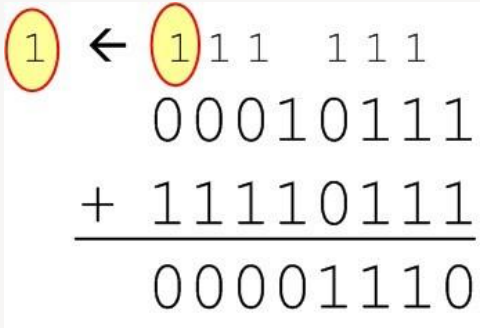
(this requires very simple hardware!)

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\ = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

CITY UNIVERSITY OF LONDON EST 1894

Arithmetic in 2's Complement

- **Example:**
  - Using two's complement binary arithmetic, find the sum of 23 and -9.
  - We see that there is carry into the sign bit and a carry out. The final result is correct: 23 + (-9) = 14.



Overflow in 2’s Complement

- It may occur but it does not introduce erroneous results
- **Rule for detecting signed 2’s complement overflow:**
  - **When the “carry in” and the “carry out” of the sign bit differ, overflow has occurred**
  - **If the carry into the sign bit equals the carry out of the sign bit, no overflow has occurred.**
- Two positive numbers are added and the result is negative. Two negative numbers are added and the result is positive.

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Floating point and Non-Numeric Data Representation

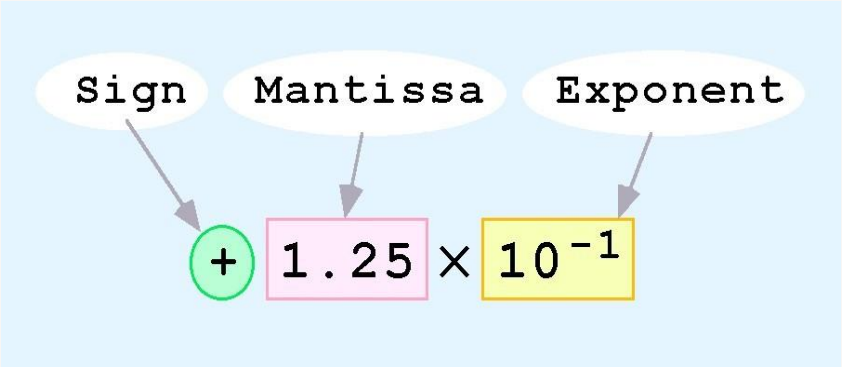
—Dr H. Asad

Floating Point Numbers

- So far we have represented positive and negative numbers using N bits
- How do we represent?
  - Very large numbers: 9,349,398,989,787,762,244,859,087,678
  - Very small numbers: 0.00000000000000000000045691
  - Rational numbers: $2/3$
  - Irrational numbers: $\sqrt{2}$

Recall Scientific Notation

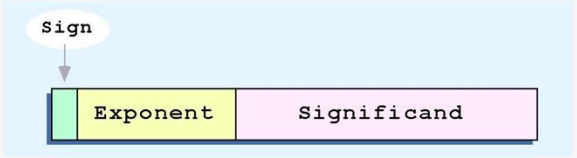
- Computers use a form of scientific notation for **floating-point representation**
- Numbers written in scientific notation have three components:



Single precision FP Representation

- Floating point numbers are of the form:
  - where: S = sign, F = significand field (mantissa), E = exponent

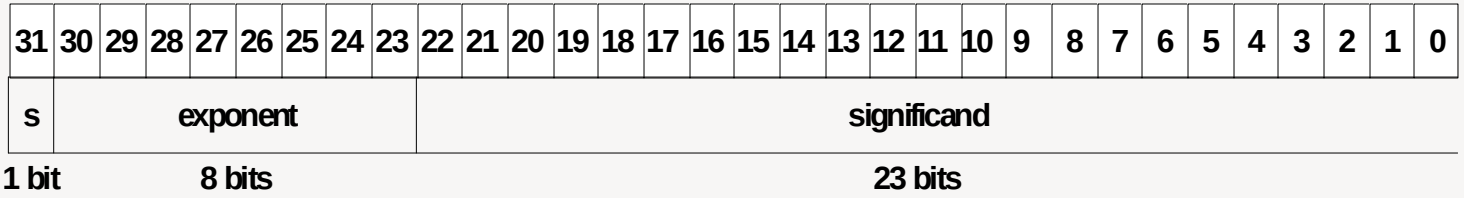
$$(-1)^{S} \times F \times 2^{E}$$



CITY
UNIVERSITY OF LONDON
EST 1894

Single precision FP Representation

- These must be packed into a word of bits, in general:
  - For example:

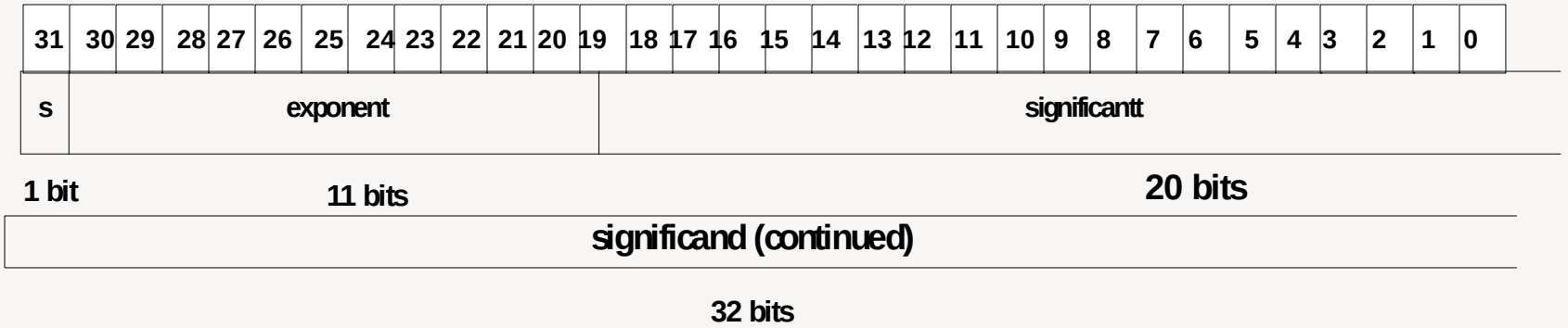


  - The **size of the exponent** field determines the **range of values** that can be represented.
  - The **size of the significand** determines the **precision** of the representation.

CITY
UNIVERSITY OF LONDON
EST 1894

Double precision FP Numbers

- **Bits allocated to the exponent and the significant:**
  - Increasing the bits for the exponent increases the range.
  - Increasing the bits for the significand increases the accuracy.
- **Double precision floating point numbers:**



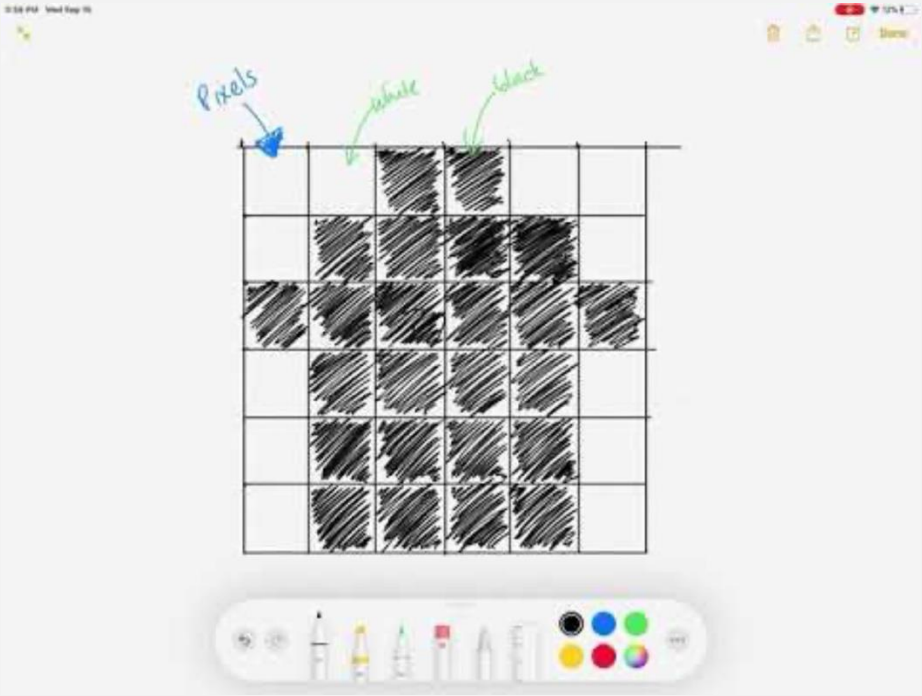
Non numerical data: Characters

- ASCII and UNICODE
- To use ‘1’s and ‘0’s to represent text and characters

CITY
UNIVERSITY OF LONDON
EST 1894

ASCII and UNICODE

Non-numerical data: Bitmaps



Summary

1) Binary
2) Positive and Negative Integers
3) Addition and subtraction
4) Floating Point Numbers
5) Hexadecimal
6) Character Data
7) Bitmaps

School of Science & Technology
City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom

T: +44 (0)20 7040 5060
E: SST-ug@city.ac.uk
www.city.ac.uk/department

Acknowledgement:
Prof George Spanoudakis