

CO1507 Introduction to Networking

Module Introduction and Numbering Systems

Jonathan Edwards 2023

Jonathan Edwards
Module leader

jedwards8@uclan.ac.uk
(+44) 01772 893286
Office CM214

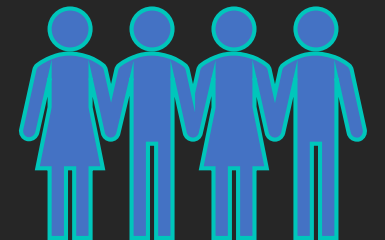
Martin Bateman

mbateman@uclan.ac.uk
Office CM111

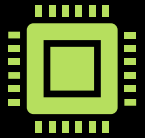
Folayo Aina

faina@uclan.ac.uk
Office CM220

The Teaching Team



Module Aims



To gain a fundamental understanding of data communications and computer networks.



To provide students with the necessary skills to set-up and configure networks.



To equip students with the knowledge to make informed decisions when designing a network.



To foster problem-solving skills through hands-on experience in the lab.

Module Topics Include

Numbering Systems

Networking Devices and Media

Network Addressing

Performance and Error Control

Networking Standards

Local, Wide-area Networks and Routing

Virtualisation

The Linux Operating System

...and plenty more besides

Assessment



1 x **COURSEWORK**
w/b 3rd March 2024

A practical assessment plus MCQs
conducted within your lab class time
Worth **50%** of the module grade



1 x **EXAMINATION**
(90 mins) May 2023

A mixture of MCQs (multi-choice questions)
and longer more detailed written answers
Worth **50%** of the module grade

What is a Computer Network?

‘A collection of computers and other hardware components interconnected by communication channels that allow sharing of resources and information.’

Cisco (n.d.)

Numbering Systems

- It is vitally important for networking that all students can work with numbering systems such as **Binary** and **Hexadecimal**
- You are already familiar with **base 10** otherwise called **decimal** numbers
- You will have been using **base 10** without knowing since you were a small child

Numbering Systems

- **Base 10** uses ten *characters* to represent numbers
- Each of these also represents a number in its own right
- These are **0, 1, 2, 3, 4, 5, 6, 7, 8, and 9**
- There is no *character* for 'ten' so we write '**10**'

Base 10 (Decimal)

- Though you've probably rarely given it a second thought, we can calculate the value of base 10 numbers like so...
- Example number: 4379

1000s (10^3)	100s (10^2)	10s (10^1)	1s (10^0)
4	3	7	9

- $(4 \times 1000) + (3 \times 100) + (7 \times 10) + (9 \times 1)$
- $4000 + 300 + 70 + 9$
- *'Four Thousand Three Hundred and Seventy Nine'*

Binary

- Modern computers use **binary** (**Base 2**) to represent numbers
- Older experimental computers may have used other number bases
- For whichever number base you need a discrete voltage to represent a value
- Older electronics were very variable which made it hard to hold the 'correct' voltage for a value
- **Binary** makes circuitry easier because there are only **2** discrete values e.g. a high and a low voltage
- Writing **binary** numbers only requires two characters **0** and **1**

Binary

- 0 'Zero' this is easy, it's represented by **0**
- 1 'One' this is easy, it's represented by **1**
- 2 'Two' this is less easy, we have run out of characters so we do the same thing as when we got to 'Ten' in decimal
- 2 'Two' is represented by **10**
- 3 'Three' this now becomes **11**
- 4 'Four' we're stuck again, so we must again rollover making **100**

Don't worry all will be explained...

Binary

- Because **binary** only uses **two** discrete characters each position from right to left will be worth **2x** (double) that of the previous position, and not *ten* times as seen in decimal.
- Example number: 1010

8s (2^3)	4s (2^2)	2s (2^1)	1s (2^0)
1	0	1	0

- $(1 \times 8) + (0 \times 2) + (1 \times 2) + (0 \times 1)$
- $8 + 0 + 2 + 0$
- 'Ten'

Binary

- Because **binary** numbers can easily be confused with **decimal** numbers...
- E.g. **1011** might be '*One Thousand and Eleven*' or be '*Eleven*'
- We can make it clear that we are writing binary by beginning the number with '0b'
- So **0b1011** is indisputably binary and equal to the number eleven

Another Way to Convert

- This method doesn't need a 'number line' like on the previous slide
- This method is the "divide by two" method
- In this case we want to convert the number **133** to binary...

	Answer	Remainder
133 ÷ 2	66	1
66 ÷ 2	33	0
33 ÷ 2	16	1
16 ÷ 2	8	0
8 ÷ 2	4	0
4 ÷ 2	2	0
2 ÷ 2	1	0
1 ÷ 2	0	1



Read the remainders
bottom to top

Answer is
10000101

Binary and Hexadecimal

- We often write binary in chunks of **8** digits called a **byte**
- **1 byte** can represent **256** discrete values or the numbers **0 - 255**
- **Binary** is how computers store **ALL** data, however for us humans it's a bit long and unwieldy to use only 1s and 0s
- We can represent **4 times** as much data in the same number of characters using another numbering system called **hexadecimal**

E.g.

○ 1111 = F

○ 11111111 = FF

...but hold on 'F' isn't a number, is it???

Hexadecimal

- Hexadecimal uses **16** characters to write numbers: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F**
- Computers can't easily and reliably work with 16 discrete values, so computers only use **binary**; hexadecimal is for humans
- So don't we stick to decimal ... because there is no easy integer ratio between
- What look like thresholds to us (e.g. 10, 100, 1000) fall in *odd* places

e.g.

- $0b1111 = 15$
- $0b11111111 = 255$

Decimal, Binary, & Hex Comparison

Decimal	Binary	Hexadecimal
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Hexadecimal

- 0 - 9 in **Hexadecimal** is exactly what you would think it is, but then...
- 10 (*Ten*) is **A** (...we **still** only needed a single character)
- 11 (*Eleven*) is **B**
- 12 (*Twelve*) is **C**
- 13 (*Thirteen*) is **D**
- 14 (*Fourteen*) is **E**
- 15 (*Fifteen*) is **F**
- However we cannot write 16 ('*Sixteen*') in a single character...
- 16 becomes... **10**
- This is exactly the same as what happened when we ran out of characters in the other two numbering systems

Hexadecimal

- Because **hexadecimal** uses **16** discrete characters each position from right to left will be worth **16 times** as the previous position, and not ten times as in decimal
- Example number: 4379

4096s (16^3)	256s (16^2)	16s (16^1)	1s (16^0)
4	3	7	9

$$\begin{aligned} &(4 \times 4096) + \\ &(3 \times 256) + \\ &(7 \times 16) + \\ &(9 \times 1) = 17,273 \end{aligned}$$

- *'Seventeen Thousand, Two Hundred and Seventy Three'*

17273

4

A

7

3

9

F

$$4379 =$$

$$(4 \times 4096) +$$

$$(3 \times 256) +$$

$$(7 \times 16) +$$

$$(9 \times 1) = 17,273$$

Hexadecimal

○ Lets look at another example: 3FD7

4096s (16^3)	256s (16^2)	16s (16^1)	1s (16^0)
3	F	D	7

$$\begin{aligned} &(3 \times 4096) + \\ &(15 \times 256) + \\ &(13 \times 16) + \\ &(7 \times 1) = 16,343 \end{aligned}$$

Or 'Sixteen Thousand, Three Hundred and Forty Three'

Hexadecimal

- Just as with binary some numbers could be mistaken for another number system
- E.g. **33** in Hex is actually **51** in decimal '*fifty one*' despite having neither a **5** or a **1** in the number

16s	1s
3	3

$$(3 \times 16) + (3 \times 1) = 48 + 3 = '51'$$

- The only real reason this looks so weird is because your language of numbers is based around decimal
- We can mark **hexadecimal** numbers by preceding the number with '0x' to make sure it isn't mistaken for any other numbers
- e.g. **0x 4379** is **not** *Four Thousand Three Hundred and Seventy Nine*

Hexadecimal

- **Hex** numbers can get pretty big pretty fast
- Conversion to **decimal** is not difficult but may well include numbers beyond your mental arithmetic capability
- However revealing the number in decimal is rarely the point of hex
- Converting **Hex** to **Binary** is more relevant, and is actually very easy

Binary < --- > Hexadecimal

- Simply treat each **1 Hex** characters as **4 bits** regardless of where it falls in the number
e.g. 0x 4379

4	3	7	9
0100	0011	1011	1001

Stick it back together...

0b 0100 0011 1011 1001 = 0 x 4379 = 17,273

Why Important in Networking?

- Standard ipv4 IP addresses are **32 bits** long and are generally expressed as **4** separate '**octets**'
- These octets look like decimal numbers but actually represent 8 bits on binary
- An IP address might be: **192.168.10.1**
- In reality that is: **11000000 10101000 00001010 00000001**
(and the spaces aren't really there)
- It's easy **now** to understand why the largest number in any of the four octets will always be **255**

Why so Important in Networking

- To fully understand IP addressing it is vital to be able to understand the address in binary
- MAC addresses and IPv6 addresses are much longer and commonly expressed in Hex
- You probably don't know what IPv6 or MAC addresses are right now, that's OK we will cover both in due course.

A 3D rendering of a large yellow question mark standing on a field of smaller, dark grey question marks. The yellow question mark is the central focus, standing upright and casting a shadow. It is surrounded by a dense field of smaller, dark grey question marks that recede into the background, creating a sense of depth. The lighting is soft, highlighting the 3D nature of the objects.

Any Questions