Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin



# 3.1 – Binary, Hexadecimal, Bytes & Words

## CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

# Memory

Each memory location can be considered as a collection of electronic "switches"

Each switch can be in one of two states

0 or 1, on or off, true or false, purple or gold, sitting or standing

These ***bits*** (***b***inary dig***its***) are the fundamental unit of data storage in a computer



Accessing each bit individually isn't very useful … we want to store data that can take a wider range of values, e.g. …

the value 214

the letter "b"

# Bytes

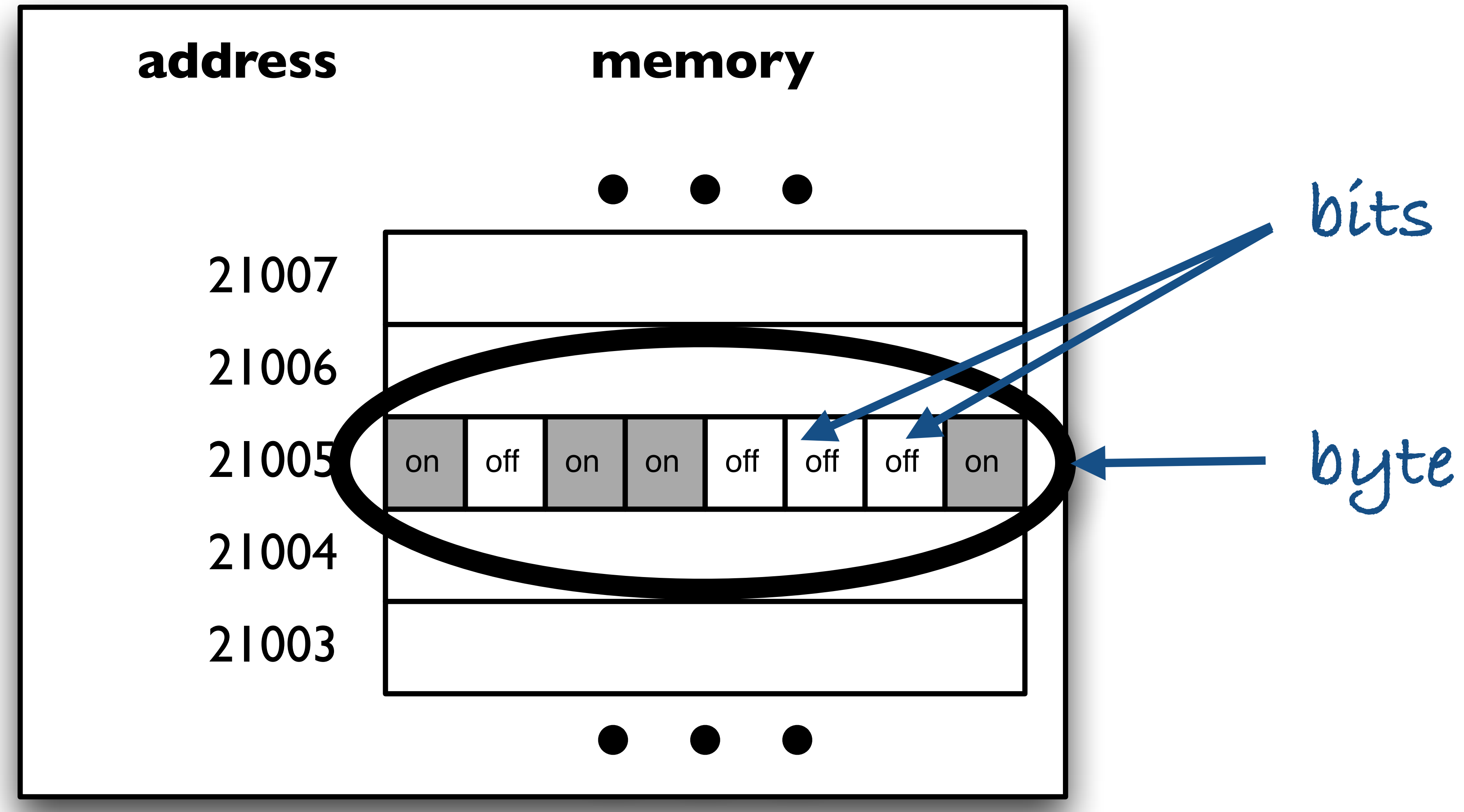


By grouping bits together in a memory location, we can store a wider range of unique values (i.e. more than the 2 values we can store using a bit)

**8 bits = 1 byte**

Bytes are the smallest "addressable" unit of memory storage (or memory location)





# Numeral Systems

We usually use the decimal (base-10) numeral system

Symbols (digits) that can represent ten integer values

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

We represent integer values larger than 9 by using two or more digits

10, 11, 12, …, 112, …, 247



e.g.: 247

$= (2 \times 10^2) + (4 \times 10^1) + (7 \times 10^0)$

2 is the **Most Significant Digit**

7 is the **Least Significant Digit**

Given *n* decimal digits …

how many unique values can we represent?

what range of non-negative integers can we represent with this number of values?

# Binary Numeral System (base 2)



Computer systems store information electronically using bits (binary digits)

Each bit can be in one of two states, which we can take to represent the binary (base-2) digits 0 and 1

**So, the binary number system is a natural number system for computing**



Using a single bit, we can represent integer values 0 and 1

- i.e. two different values

Using two bits, we can represent 00, 01, 10, 11

- i.e. four different values

Given 8 bits …

- how many unique values can we represent?
- what range of non-negative integers can we represent?

| | | | |
|---|---|---|---|
| 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 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |

# Base subscript notation



***There are 10 types of people in the world: those who understand binary and those who don't ...***

The same sequence of symbols can have a different meaning depending on the base being used



Use subscript notation to denote the base being used

$12_{10} = 1100_2$

$1_{10} = 1_2$

Using binary all the time would become quite tedious

The CSU11021 exam is worth $1000110_2$% of the final mark

Convert $00100101_2$ to its decimal …



$$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

… $37_{10}$

Convert $23_{10}$ to its binary equivalent …

| | | | | | remainder |
|---|---|---|---|---|---|
| 23 | / | 2 | = | 11 | 1 |
| 11 | / | 2 | = | 5 | 1 |
| 5 | / | 2 | = | 2 | 1 |
| 2 | / | 2 | = | 1 | 0 |
| 1 | / | 2 | = | 0 | 1 |

read digits



… $10111_2$

# Hexadecimal Numeral System

Base-16 (hexadecimal or “hex”) is a convenient numeral system for computer scientists:

- With binary, we needed 2 symbols (0 and 1)
- With decimal, we needed 10 symbols (0, 1, ..., 9)
- With hexadecimal, we need 16 symbols



Use the same ten symbols as the decimal system for the first ten hexadecimal digits

- “Borrow” the first six symbols from the alphabet for the last six symbols
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Why is hexadecimal useful?

- 16 is a power of 2 ($2^4$), so one “hex” digit corresponds to exactly four binary digits (bits) (and vice versa) making translation and manipulation easy!

# Decimal, Binary and Hexadecimal



| base 10 | base 2 | base 16 |
|---|---|---|
| 0 | 0000 | 0 |
| 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 |



What observation can you make about odd and even numbers in a binary representation?

What observation can you make about values that are a power of 2 (e.g. $2^3$)?

# Alternative base notation



Without a fancy word processor, we won't be able to use the subscript notation to represent different bases

How would we tell a computer whether we mean 1010 or 1010?

Instead we can prefix values with symbols that provide additional information about the base



In **ARM Assembly Language** (which we will be using) we use the following notation:

| | |
|---|---|
| 1000 | No prefix usually means decimal |
| **0x**1000 | Hexadecimal (used often) |
| **&**1000 | Alternative hexadecimal notation |
| **2_**1000 | Binary |
| **n_**1000 | Base n |

# Converting between binary and hex



## Conversion between hex and binary is trivial

One hexadecimal digit represents the same number of unique values as four binary digits

Group the binary digits (bits) into groups of 4 bits **starting from the right, padding with zeros if necessary,** e.g.:




## Hexadecimal is used by convention when referring to memory addresses

e.g. address **0x**00001000, address **0x**0000400A

## What is the binary equivalent of 0x2D?

## What is the hexadecimal equivalent of 111010?

# Bytes, Halfwords and Words



Remember

8 bits = 1 byte

with 8 bits we can represent $2^8 = 256$ unique values

Sometimes useful to group more (than 8) bits together to store an even wider range of unique values

**2 bytes = 16 bits = 1 halfword**

**4 bytes = 32 bits = 1 word**



When we refer to memory locations by address (using the ARM microprocessor), we can only do so in units of **bytes**, **halfwords** or **words**

**the byte at address 0x00005210**

**the halfword at address 0x00005210**

**the word at address 0x00005210**

more on this later!

| address | memory |
|---|---|
| | • • • |
| 0x00005215 | 64 |
| 0x00005214 | 7B |
| 0x00005213 | 5D |
| 0x00005212 | 35 |
| 0x00005211 | 27 |
| 0x00005210 | 89 |
| 0x0000520F | 82 |
| 0x0000520E | 3C |
| 0x0000520D | 8B |
| 0x0000520C | 53 |
| 0x0000520B | A2 |
| 0x0000520A | 9F |
| 0x00005209 | E8 |
| 0x00005208 | 4D |
| 0x00005207 | 0A |
| 0x00005206 | 07 |
| | • • • |

# Larger units



Larger units of information storage

1 **kilobyte** (kB) = $2^{10}$ bytes = 1,024 bytes

1 **megabyte** (MB) = 1,024 KB = $2^{20}$ bytes = 1,048,576 bytes

1 **gigabyte** (GB) = 1,024 MB = $2^{30}$ bytes = …



The following units of groups of bits are also used, usually when expressing **data rates** (e.g. Mbits/s):

1 **kilobit** (kb) = 1,000 bits

1 **megabit** (Mb) = 1,000 kilobits = 1,000,000 bits

IEC prefixes, KiB, MiB, GiB, …

How many bytes are in 1kilobit?

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin



# 3.2 – Representing Text (ASCII)

## CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

# Representing text



So far, we have only considered how computers store (non-negative) integer values using binary digits

What about representing other information, for example text composed of alphanumeric symbols?



**‘T’, ‘h’, ‘e’, ‘ ’, ‘q’, ‘u’, ‘i’, ‘c’, ‘k’, ‘ ’, ‘b’, ‘r’, ‘o’, ‘w’, ‘n’, ‘ ’, ‘f’, ‘o’, ‘x’, …**



We’re still restricted to storing binary digits (bits) in memory

To store alphanumeric symbols or “characters”, we can **assign each character a value**, which can be stored in binary form in memory

**TO:** Bob
**FROM:** Jonathan
**DATE:** 27/09/2016
**SUBJECT:** CSU11021
Hi Bob,
Just checking that you received my email last Thursday …

0110100
0101001
1101101

**TO:** ^f£
**FROM:** *&x7s%cha
**DATE:** he*(!.jjds
**SUBJECT:** sg93jg93
gs98^38998hfhr%
%20g348jg98h9ghw9h9hg49whfh8
w8 7y394hg9))*3093 …

## American Standard Code for Information Interchange

ASCII is a standard used to encode alphanumeric and other characters associated with text

- e.g. representing the word “hello” using ASCII

| | | | | |
|---|---|---|---|---|
| ‘H’ | ‘E’ | ‘L’ | ‘L’ | ‘O’ |
| 0x48 | 0x45 | 0x4C | 0x4C | 0x4F |



Each character is stored in a single byte value (8 bits)

- 1 byte = 8 bits means we can have a possible 256 characters
- In fact, ASCII only uses 7 bits, giving 128 possible characters
- Only 96 of the ASCII characters are **printable**
- Remaining values are **control codes** – examples??

# ASCII



1st digit

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | NUL | DLE | SPACE | 0 | @ | P | ` | p |
| **1** | SOH | DC1 | ! | 1 | A | Q | a | q |
| **2** | STX | DC2 | “ | 2 | B | R | b | r |
| **3** | ETX | DC3 | # | 3 | C | S | c | s |
| **4** | EOT | DC4 | $ | 4 | D | T | d | t |
| **5** | ENQ | NAK | % | 5 | E | U | e | u |
| **6** | ACK | SYN | & | 6 | F | V | f | v |
| **7** | BEL | ETB | | 7 | G | W | g | w |
| **8** | BS | CAN | ( | 8 | H | X | h | x |
| **9** | HT | EM | ) | 9 | I | Y | i | y |
| **A** | LF | SUB | * | : | J | Z | j | z |
| **B** | VT | ESC | + | ; | K | [ | k | { |
| **C** | FF | FS | , | < | L | \ | l | \| |
| **D** | CR | GS | - | = | M | ] | m | } |
| **E** | SO | RS | . | > | N | ^ | n | ~ |
| **F** | SI | US | / | ? | O | _ | o | DEL |

2nd digit

e.g. “E” = 0x45

# ASCII



**The value 0 is not the same as the character ‘0’**

**Similarly, the value 1 is not the same as the character ‘1’**

- 1 is 1 (or 0x01) but ‘1’ is 0x31 (or $00110001_2$)
- 1+1 = 2 but ‘1’+‘1’=?

The ASCII characters ‘0’, ‘1’, ... are used in text to display values in human readable form, **not for arithmetic**



Upper and lower case characters have different codes (‘E’ is 0x45 but ‘e’ is 0x65)

The first printable character is the space symbol ‘ ’ and it has code 0x20 (sometimes written ‘␣’ for clarity)

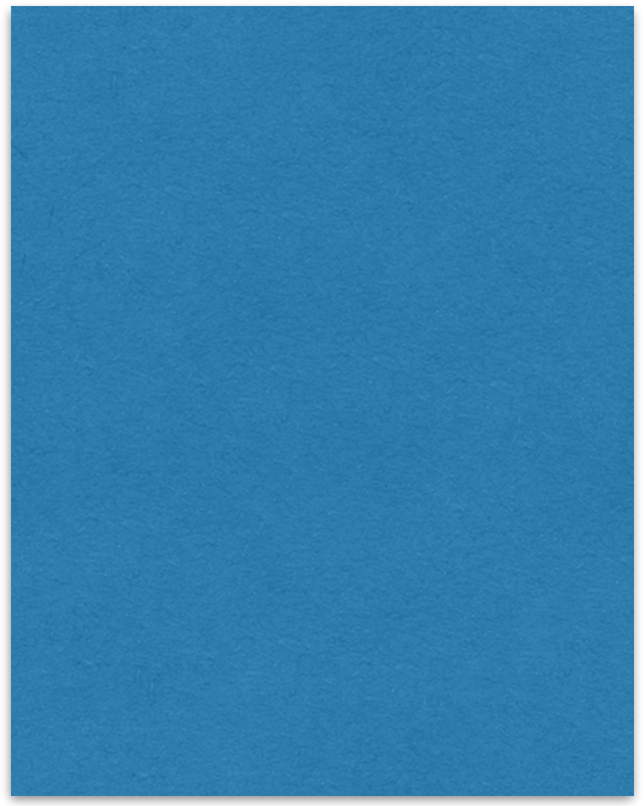
It is almost always more efficient to store a value in its “value” form than its ASCII text form

- the value $10_{10}$ (or $1010_2$) requires 1 byte
- the ASCII characters ‘1’ (0x31) followed by ‘0’ (0x30) require 2 bytes (1 byte each)
- we cannot perform arithmetic directly using the ASCII characters (‘1’ + ‘1’ = 0x31 + 0x31 = 0x62 = ‘b’, i.e. nonsense!!)

# Other Kinds of Information?



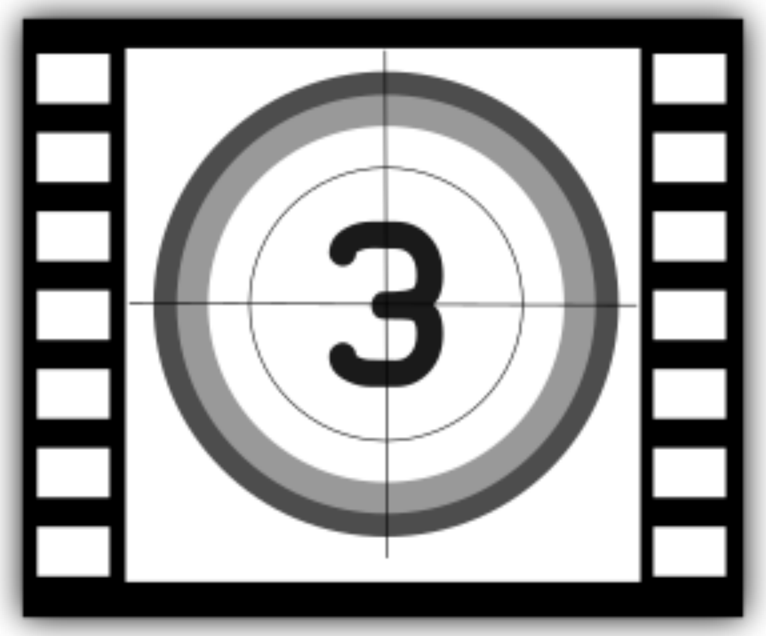
e.g.:



Colours

Images






Videos

Sounds