CS1102
Lecture 2
Binary Number System

CityU

Semester A, 2021-2022

Department of Computer Science

City University of Hong Kong

Think about it

- You may have known or heard people say the followings:
  Computers use binary system to represent information

- But….
  - What is a binary system?
  - How is it different from the number system used by humans?
  - Why and how do computers use binary system?

Number System Used by Humans

- Children often use fingers to help them count
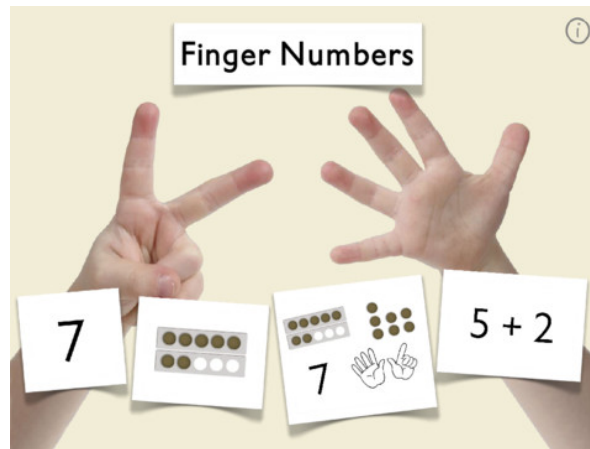
- Humans use these symbols to represent single digits (1-digit number):

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

- After the largest 1-digit number 9, the next number is represented by a 2-digit number, i.e., 10
  - We say that we count in base 10
  - The number system used by humans is called a decimal system

Binary System

- Under a binary system, we only have 2 symbols: 0, 1
- The above symbols are called bits (binary digit)
  - Who invented "bit"? https://en.wikipedia.org/wiki/Bit
- After the largest 1-bit number 1, the next number is represented by a 2-bit number, i.e., 10
- Numbers in a binary system are in base 2
- Sometimes a number in subscript is used to represent the base, e.g.,
  - $10_2 = 2_{10}$ [10 in base 2 is equal to 2 in base 10]

Pronounced as "one zero"

Pronounced as "ten"

Number System Used by Computers

- Computers use binary system to represent information
- Examples:
  - A light switch may be on (1) or off (0)
  - An electronic circuit may be closed (1) or open (0)
  - A specific location on a tape or disk may have a positive charge (1) or negative charge (0)

  This is the reason why a two-state or binary system is used to represent data and instructions in computer

Conversion between Decimal and Binary System

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Given a decimal number, how to convert it to its binary representation (decimal-to-binary conversion)?

Given a binary number, how to convert it to its decimal representation (binary-to-decimal conversion)?

Binary to Decimal Conversion

- Let's first try to examine a multiple digit decimal number
  - e.g., $375_{10} = 3 \times 100 + 7 \times 10 + 5$

    $= 3 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$

    hundreds

    tens

    ones

- This technique can be applied to convert a multiple bit binary number to decimal
  - e.g., $1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
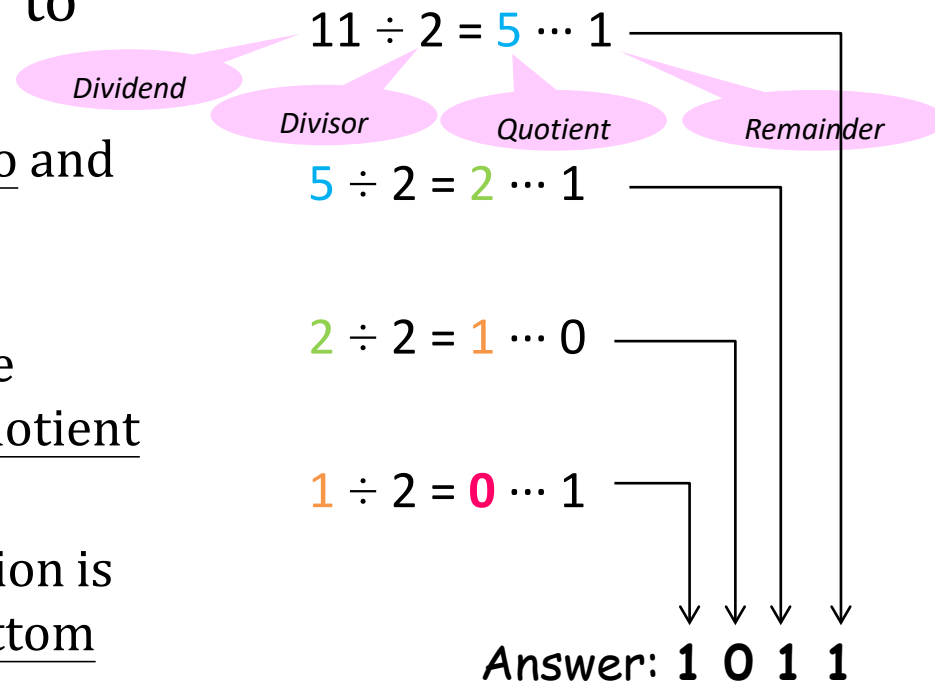
    $= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$

    $= 11_{10}$

Decimal to Binary Conversion

- **Repeated division** can be used to convert a decimal number to binary as follows:
  - Step 1 - <u>divide the value by two</u> and record the remainder
  - Step 2 - <u>continue to divide</u> the quotient by two and record the remainder, <u>until the newest quotient becomes zero</u>
  - Step3 - the binary representation is <u>the remainders listed from bottom to top</u> in the order they were recorded

e.g., what is 11_{10} in binary?

$$11 \div 2 = 5 \cdots 1$$

Dividend *Divisor* *Quotient* *Remainder*

$$5 \div 2 = 2 \cdots 1$$

$$2 \div 2 = 1 \cdots 0$$

$$1 \div 2 = 0 \cdots 1$$

Answer: **1 0 1 1**

Simplified way:

```
2 | 11
2 |  5 ⋯ 1
2 |  2 ⋯ 1
      1 ⋯ 0
```

Note on Conversion between Binary and Decimal Numbers

- The techniques from the previous 2 slides are only applicable for positive integers

- Other techniques are used by computer to represent
  - Negative numbers, e.g., –2 (to be covered later)
  - Fractional numbers, e.g., 3.1415
  - Very large numbers, e.g., 6.022×10^{23}
  - Very small numbers, e.g., 6.626×10^{-34}

Possible Number Range with a Byte

- A byte consists of 8 bits
- The smallest value represented by a byte is: $00000000_2 = 0_{10}$
- The largest value represented by a byte is: $11111111_2 = 255_{10}$
- Thus there can be a total of 256 values (0,1,2,......,254,255) represented by a byte. Alternatively, this range can also be computed as 2^8

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
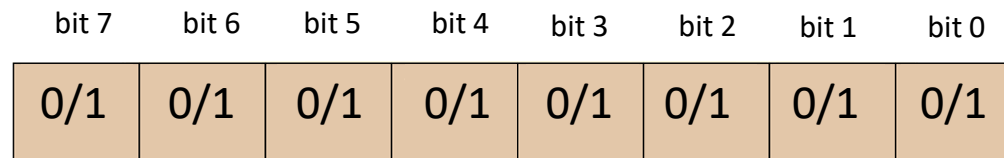
$$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8 = 256$$

Data Size Measurement

- Normally data size is measured by bytes with various order of magnitude:
  - *Kilobytes* (KB): 2^{10} = 1,024 bytes
    - e.g., a simple text file
  - *Megabytes* (MB): 2^{20} = 1,024 KB = 1,048,576 bytes
    - e.g., picture taken by your mobile phone
  - *Gigabytes* (GB): 2^{30} = 1,024 MB = 1,073,741,824 bytes
    - e.g., maximum data usage of your mobile plan
  - *Terabytes* (TB): 2^{40} = 1,024 GB = 1,099,511,627,776 bytes
    - e.g., storage size of your hard drive
  - *Petabytes* (PB): 2^{50} = 1,024 TB = 1,125,899,906,842,624 bytes
    - e.g., scale of total data size of Netflix movies
  - *Exabytes* (EB): 2^{60} = 1,024 PB = 1,152,921,504,606,846,976 bytes
    - e.g., scale of Dropbox storage system

Hexadecimal System

- Under a hexadecimal system, we have the following 16 symbols:

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

- The above symbols are called nibbles

- After the largest 1-nibble number F, the next number is represented by a 2-nibble number, i.e., 10

- Numbers in a hexadecimal system are in base 16

Byte Representing Colors

- Each color shown on screen is a combination of red, green and blue (RGB) with different intensity values

- In a "true color" system, a color is represented by 24 bits, 8 bits for red, 8 bits for green and 8 bits for blue. Each color component, red, green and blue, takes a value ranging from 0 to 255, e.g.,

	Red	Green	Blue	
Purple:	**172**	**73**	**185**	**#AC49B9**
Yellow:	**253**	**249**	**88**	**#FDF958**

Note: in HTML, sometimes text or background color is defined in hexadecimal notation.
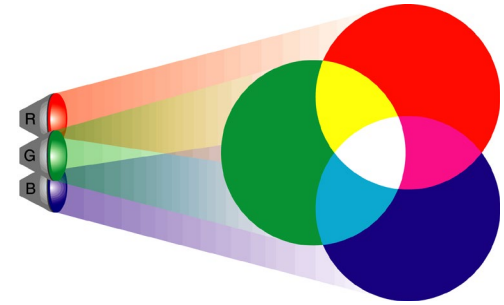
Conversion between Decimal and Hexadecimal System

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10

Given a decimal number, how to convert it to its hexadecimal representation (decimal-to-hexadecimal conversion)?

Given a hexadecimal number, how to convert it to its decimal representation (hexadecimal-to-decimal conversion)?

Hexadecimal to Decimal Conversion
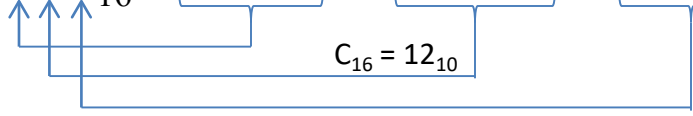
- Remember binary-to-decimal conversion?
  - e.g., $1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

  $$= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$
  $$= 11_{10}$$

- Hexadecimal-to-decimal conversion works in a similar way, while keeping in mind that the input number is in base 16
  - e.g., $3C8_{16} = 3 \times 16^2 + 12 \times 16^1 + 8 \times 16^0$

  $C_{16} = 12_{10}$

  $$= 3 \times 256 + 12 \times 16 + 8 \times 1$$
  $$= 768 + 192 + 8$$
  $$= 968_{10}$$

Decimal to Hexadecimal Conversion

- **Repeated division** can also be used to convert a decimal number to hexadecimal as follows:
  - Step 1 - divide the value by 16 and record the remainder converted to hexadecimal, i.e., 0,1,2,...,9,A,B,C,D,E,F
  - Step 2 - continue to divide the quotient by 16 and record the remainder, until the newest quotient becomes zero
  - Step3 - the hexadecimal representation is the remainders listed from bottom to top in the order they were recorded

e.g., what is 968_{10} in hexadecimal?

$968 \div 16 = 60 \cdots 8$

Dividend   Divisor   Quotient   Remainder

$60 \div 16 = 3 \cdots 12 = C$

$3 \div 16 = 0 \cdots 3$

Answer: **3 C 8**
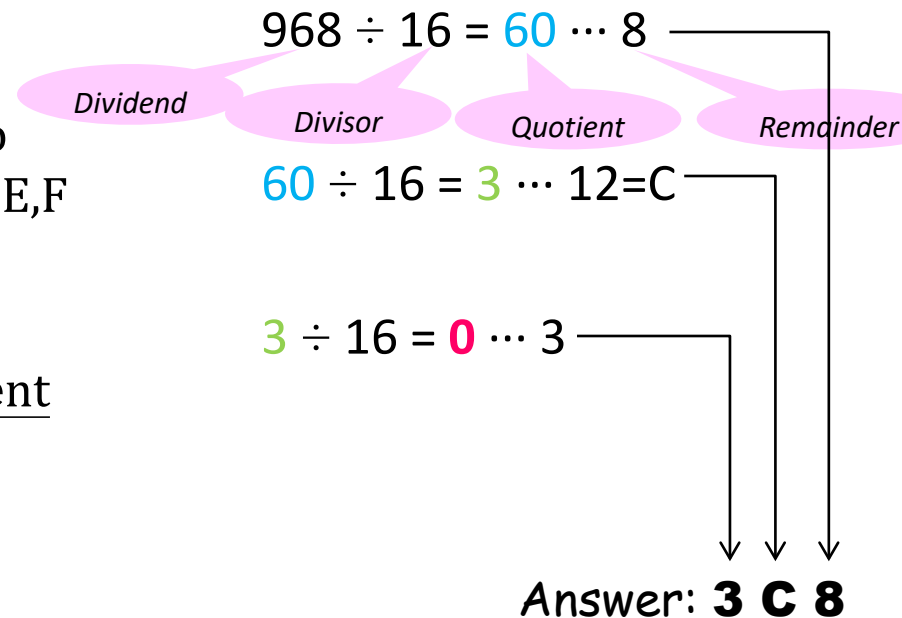
Simplified way:

$$\begin{array}{r|l} 16 & 968 \\ 16 & 60 \cdots 8 \\ & 3 \cdots 12 = C \end{array}$$

Binary ↔ Hexadecimal Conversion

- So far you have learnt about conversions of
  - Decimal ↔ Binary
  - Decimal ↔ Hexadecimal

  How do you convert between binary and hexadecimal numbers?

- What if we carry out the following 2 steps for each case?

binary → hexadecimal

1) binary → decimal
2) decimal → hexadecimal

e.g. 10100101_2 in hexadecimal?

1) $10100101_2 = 2^7+2^5+2^2+1=165_{10}$
2) $165_{10} = 10\times16+5\times1 = A5_{16}$

∴ $10100101_2 = A5_{16}$

hexadecimal → binary

1) hexadecimal → decimal
2) decimal → binary

e.g. 97_{16} in binary?

1) $97_{16} = 9\times16+7\times1=151_{10}$
2) $151_{10} = 128+16+4+2+1= 10010111_2$

∴ $97_{16} = 10010111_2$

Although you can get the answer by this method, this may not be the most efficient way for calculation!

Binary ↔ Hexadecimal Conversion (2)

- Let's look at some examples. Can you identify some patterns?

Hexadecimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Hexadecimal	Binary
34	11 0100
39	11 1001
3C	11 1100
3E	11 1110
74	111 0100
79	111 1001
7C	111 1100
7E	111 1110
C4	1100 0100
C9	1100 1001
CC	1100 1100
CE	1100 1110
E4	1110 0100
E9	1110 1001
EC	1110 1100
EE	1110 1110

hexadecimal → binary:
Each nibble from the hexadecimal number is converted individually to a 4-bit binary number (leading 0s are appended if required, e.g., from 100 to 0100)

binary → hexadecimal:
every 4 bits starting from the right of the binary number are converted individually to a nibble

What is $3A8D_{16}$ in binary?

What is 10011011101011_{2} in hexadecimal?

Binary Arithmetic

- Rules of Binary Addition
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 0 = 1$
  - $1 + 1 = 0$, and carry 1 to the next more significant bit

  - E.g.,

$$
\begin{array}{r}
0\,0\,0\,1\,1\,0\,1\,0 = 26_{10} \\
+\ 0\,0\,0\,0\,1\,1\,0\,0 = 12_{10} \\
\hline
0\,0\,1\,0\,0\,1\,1\,0 = 38_{10}
\end{array}
$$

- Rules of Binary Subtraction
  - $0 - 0 = 0$
  - $0 - 1 = 1$, and borrow 1 from the next more significant bit
  - $1 - 0 = 1$
  - $1 - 1 = 0$

  - E.g.,

$$
\begin{array}{r}
0\,0\,1\,1\,0\,0\,1\,1 = 51_{10} \\
-\ 0\,0\,0\,1\,0\,1\,1\,0 = 22_{10} \\
\hline
0\,0\,0\,1\,1\,1\,0\,1 = 29_{10}
\end{array}
$$

> the concept is the same as how you do carrying/borrowing when adding/subtracting decimal numbers
> - Try to add 00095050 + 00005500 in decimal and see how you do the carrying learnt from your primary school
> - Try to calculate 00550055-00050550 in decimal and see how you do the borrowing learnt from your primary school

Binary Arithmetic

- Rules of Binary Multiplication
  - 0 x 0 = 0
  - 0 x 1 = 0
  - 1 x 0 = 0
  - 1 x 1 = 1

- *i.e. add the shifted multiplicand if the bit is 1*

  - E.g.,

$$
\begin{array}{ll}
0\ 0\ 1\ 0\ 1\ 0\ 0\ 1 & (41_{10}) \\
\times\ \ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0 & (6_{10}) \\
\hline
0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 & \\
0\ 0\ 1\ 0\ 1\ 0\ 0\ 1 & \\
0\ 0\ 1\ 0\ 1\ 0\ 0\ 1 & \\
\hline
0\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0 & (246_{10})
\end{array}
$$

Try to calculate 101001 x 110 as decimal numbers by considering the 1st number multiplying with every digit in the 2nd number, appending the right number of 0s on the right (same as shifting) and then adding them

- Binary Division: repeated process of subtraction

  - E.g.,

$$
\begin{array}{l}
\ \ 1\ 1\ 0\ 1\ 1 \quad (27_{10}) \\
\hline
1\ 0\ 1\ \ )\ \ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \quad (135_{10}) \\
(5_{10}) \quad -\ \ 1\ 0\ 1 \\
\hline
\ 1\ 1\ 0 \\
-\ 1\ 0\ 1 \\
\hline
\ 0\ 1\ 1 \\
-\ \ \ \ \ 0 \\
\hline
\ 1\ 1\ 1 \\
-\ \ 1\ 0\ 1 \\
\hline
\ 1\ 0\ 1 \\
-\ 1\ 0\ 1 \\
\hline
\ 0
\end{array}
$$

Try to calculate 10000111 / 101 in long division form as decimal numbers and remind yourself how you did it in primary school

Word

- A word is the number of bits that can be accessed at one time by the computer central processing unit (CPU). The more bits in a word, the more data a computer can process at one time.
  - E.g., a 64-bit-word computer can access 64 bits (8 bytes) at a time
- Computers represent numbers in binary with a fixed N-bit-word
  - Assume that N=4, then 4 bits are used to represent a number, e.g.,

    $0000_2 = 0_{10}$
    $0001_2 = 1_{10}$
    $0010_2 = 2_{10}$
    ......
    $1111_2 = 15_{10}$
  - Note that leading 0s (underlined in red in the above examples) are added to make up the 4 bits

Representing Signed Integers in Binary

- So far we have learnt about how to deal with unsigned integers (positive integers or zero) in binary

- How do we represent signed integers that include negative integers, e.g., -2, -11, etc. in binary?

  - You may observe that an integer, whether positive or negative, consists of the sign $(+/-)$ and the magnitude (the quantity without the sign), e.g., $+2$ (sign = +, magnitude = 2); -2 (sign = $-$, magnitude = 2)

  - So how about we just use the first bit, i.e., the most significant bit to denote the sign such that we use bit 0 if it is positive or bit 1 if it is negative, and then just represent the magnitude as a positive integer in binary using the remaining bits (same way as before), e.g.,

$$+2_{10} = 0010_2 \qquad\qquad -2_{10} = 1010_2$$

sign: +   sign: $-$
magnitude: $2_{10} = 10_2$ magnitude: $2_{10} = 10_2$

This is known as the sign-magnitude representation

Problems with Sign-Magnitude Representation

1. What is 0 in binary using sign-magnitude representation?
    - Assume an 4-bit word size, both of the following cases correspond to 0:
    $$+0_{10} = 0000_2 \qquad -0_{10}=1000_2$$
    - This is a waste because two different bit patterns are used to denote the same number
    - The numbers that can be represented with 4-bit sign-magnitude representation are -7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7 (15 numbers)
2. How to perform arithmetic?
    E.g., what is the result of 0010_2+1010_2?
    - If you apply the binary arithmetic from previous slide, you would get 1100_2 as the result, which translates to -4_{10} in sign-magnitude representation.
    - However, $0010_2 =+2_{10}$ and $1010_2=-2_{10}$, so their sum should be 0 instead.
    - You can see that the sign-magnitude representation does not have a trivial way of doing arithmetic with negative numbers

Two's Complement

- Computers use another representation known as the Two's Complement to represent signed integers (positive integers, 0, negative integers)

- Two's Complement
  - Positive integers are represented in the same way as sign-magnitude representation
  - 0 is represented by setting all bits to 0
  - Negative integers are represented by bit patterns obtained by the following steps:
    1. Convert the magnitude of the negative number in binary
    2. 'Toggle' each bit , i.e., flip all bits $0 \rightarrow 1$ and $1 \rightarrow 0$
    3. Add 1 to the binary number to get the resulting representation

- Examples with 4-bit word size
  - $+5_{10} = 0101_2$
  - $0_{10} = 0000_2$
  - $-5_{10} = ?$
    1. Magnitude: $+5_{10} = 0101_2$
    2. Toggle: 1010_2
    3. Add 1: $1010_2 + 0001_2 = 1011_2$

  $\therefore -5_{10} = 1011_2$

Two's Complement: Negating Numbers

- In fact, as long as you have the two's complement representation of a number x, you can always apply the last 2 steps (toggle and add 1) to obtain the two's complement representation of its negative counterpart, i.e., $-x$

- Examples with 4-bit word size

$+5_{10} =$ 0101_2 $-5_{10} =$ 1011_2
Toggle: 1010_2 Toggle: 0100_2
Add 1: 1011_2 Add 1: 0101_2
$-5_{10} =$ 1011_2 $+5_{10} =$ 0101_2

Two's Complement: Number Range

Decimal	Two's Complement
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
−1	1111
−2	1110
−3	1101
−4	1100
−5	1011
−6	1010
−7	1001
−8	1000

- The numbers that can be represented with 4-bit two's complement representation are -8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7 (16 numbers)
- You can observe that the most significant (first) bit is still the sign bit such that positive numbers will start with bit 0 and negative numbers start with bit 1
- There is only 1 bit pattern that corresponds to the number 0, with all the bits set to 0

Two's Complement: Arithmetic

- Conventional approach can be used to add binary numbers in two's complement representation
- Examples with 4-bit word size:

$$
\begin{array}{cccc}
 & 0 & 0 & 1 & 0 \\
+ & 0 & 0 & 1 & 1 \\
\hline
 & 0 & 1 & 0 & 1 \\
\end{array}
$$

	0 0 1 0	$(+2_{10})$
+	0 0 1 1	$(+3_{10})$
	0 1 0 1	$(+5_{10})$

	0 0 1 0	$(+2_{10})$
+	1 1 0 0	(-4_{10})
	1 1 1 0	(-2_{10})

	1 1 1 1	(-1_{10})
+	0 0 1 1	$(+3_{10})$
1	0 0 1 0	$(+2_{10})$

	1 1 0 0	(-4_{10})
+	1 1 0 1	(-3_{10})
1	1 0 0 1	(-7_{10})

As the word size is 4-bit, we only keep the last 4 bits as the result and ignore the extra bit

Two's Complement: Subtraction

- Subtraction can be easily performed by using two's complement and addition

- The expression x – y can be calculated as x + (-y), where (-y) is obtained by taking two's complement of y and then added to x

- Examples with 4-bit word size:

```
0101₂-0110₂ = ?                 1100₂-1111₂ = ?
1) Take two's complement of 0110  1) Take two's complement of 1111
   Toggle: 1001                       Toggle: 0000
   Add 1:  1010                       Add 1:  0001
2) Compute 0101₂+1010₂            2) Compute 1100₂+0001₂
     0 1 0 1   (+5₁₀)                   1 1 0 0   (-4₁₀)
   + 1 0 1 0   (-6₁₀)                 + 0 0 0 1   (+1₁₀)
     1 1 1 1   (-1₁₀)                   1 1 0 1   (-3₁₀)
```

Why Two's Complement Works?

- Let's see that 6-6=0:

```
0110₂-0110₂ = ?
1) Take two's complement of 0110
   Toggle: 1001
   Add 1:  1010
2) Compute 0110₂+1010₂
     0 1 1 0   (+6₁₀)
   + 1 0 1 0   (-6₁₀)
   1 0 0 0 0   ( 0₁₀)
```

- So why 6-6=0? It will be obvious by putting the steps together:

```
     0 1 1 0   (+6₁₀)
     1 0 0 1   (Toggle)
   + 0 0 0 1   (Add 1)
   1 0 0 0 0   ( 0₁₀)
```

Two's Complement: Overflow

- With 4-bit word size, the possible numbers are -8,-7,-6,-5, -4,-3,-2,-1,0,1,2,3,4,5,6,7

- Problem will arise if you try to add numbers that sum up outside this range

- Examples:

$$
\begin{array}{cccc}
  & 0 & 1 & 0 & 0 & (+4_{10}) \\
+ & 0 & 1 & 0 & 1 & (+5_{10}) \\
\hline
1 & 0 & 0 & 1 & & (-7_{10})
\end{array}
\qquad
\begin{array}{cccc}
  & 1 & 1 & 0 & 1 & (-3_{10}) \\
+ & 1 & 0 & 1 & 0 & (-6_{10}) \\
\hline
1 & 0 & 1 & 1 & 1 & (+7_{10})
\end{array}
$$

- This problem is known as overflow. You know that there is overflow if you get a negative number after adding 2 positive numbers or if you get a positive number after adding 2 negative numbers

Text Representation

- So far we have learnt about how computers represent numbers, i.e., with a binary system
- How does a computer represent text?
  - The simplest way is to encode each text character with a number
  - For example, if we only need to represent 26 capital letters from A to Z, then we can use a simple encoding scheme A1Z26, i.e., use 1 to represent A, 2 to represent B, ..., 26 to represent Z so 3 19 would mean CS
  - However, A1Z26 can only represent 26 letters and is not enough for computer usage as we need to represent various kinds of characters or symbols
- There are in fact a number of coding schemes and we will introduce a couple of them: ASCII and Unicode

ASCII

- ASCII – American Standard Code for Information Interchange, is the most widely used coding scheme to represent a set of characters (primarily for English), e.g.,

Text: H   e   l   l   o   !

ASCII (Dec): 72 101 108 108 111 33

ASCII (Hex): 48 65 6C 6C 6F 21

Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	HT	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SOH	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C		
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	□

ASCII Example

- If we want to know how computer represents the text `A1?` using ASCII code, then we first look up each of these character from the ASCII code table from previous slide, i.e.

The ASCII code for 'A' is 65_{10} = 41_{16} So the computer is using one byte with binary value 01000001 to represent it

A byte representing the letter A

0	1	0	0	0	0	0	1

The ASCII code for '1' is 49_{10} = 31_{16} So the computer is using one byte with binary value 00110001 to represent it

A byte representing the number 1

0	0	1	1	0	0	0	1

The ASCII code for '?' is 63_{10} = $3F_{16}$ So the computer is using one byte with binary value 00111111 to represent it

A byte representing the symbol ?

0	0	1	1	1	1	1	1

Most significant bit (MSB)

Least significant bit (LSB)

Unicode

- As it is not sufficient to use ASCII to encode a large set of characters and foreign languages, another coding scheme called Unicode has been proposed. From the website http://www.unicode.org:

  The Unicode Standard is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages and technical disciplines of the modern world. In addition, it supports classical and historical texts of many written languages

Lesson Summary

- Computers understand and process information in binary system. Numbers are represented in base-2 where every bit is either 1 or 0.

- A byte has 8 bits and have 256 possible values. It can be used to represent a single character in a computer

- Hexadecimal numbers are in base 16 and in general can represent numbers with fewer nibbles than the number of bits in base 2 or the number of digits in base 10

- Numbers can be converted from one base to another

- Signed integers are represented using two's complement in computers

- Text can be represented in computers by first encoding each character as a number code

Reference

[1] Howstuffworks.com - How Bits and Bytes Work

- http://computer.howstuffworks.com/bytes.htm

[2] Wikipedia - Computer numbering formats

- http://en.wikipedia.org/wiki/Computer_numbering_formats

[3] Virginia Tech - Number Systems

- http://courses.cs.vt.edu/~csonline/NumberSystems/Lessons/index.html

[4] Game – Hex Invaders (Hexadecimal color code)

- http://www.hexinvaders.com

[5] Game – Flippy Bit (Hexadecimal to binary conversion)

- http://flippybitandtheattackofthehexadecimalsfrombase16.com

Reading

- Computing Essentials 2021
  - Chapter 5, p.121-122
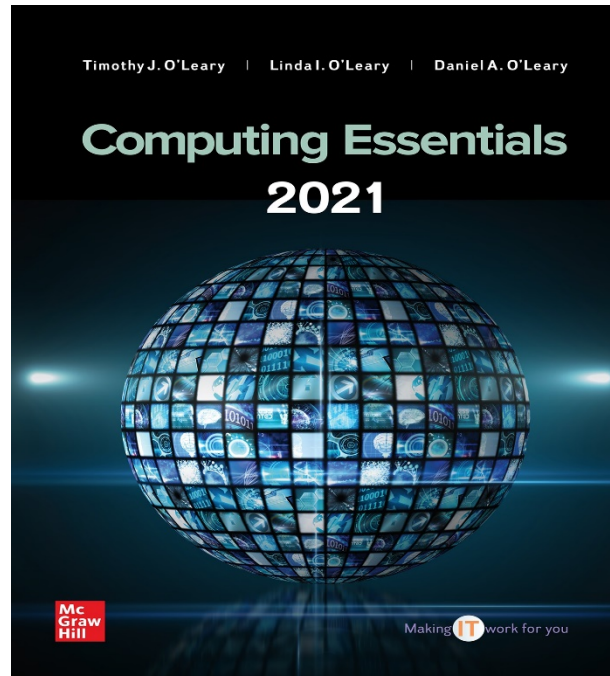
Binary Magic

32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

08	09	10	11
12	13	14	15
24	25	26	27
28	29	30	31
40	41	42	43
44	45	46	47
56	57	58	59
60	61	62	63

04	05	06	07
12	13	14	15
20	21	22	23
28	29	30	31
36	37	38	39
44	45	46	47
52	53	54	55
60	61	62	63

02	03	06	07
10	11	14	15
18	19	22	23
26	27	30	31
34	35	38	39
42	43	46	47
50	51	54	55
58	59	62	63

01	03	05	07
09	11	13	15
17	19	21	23
25	27	29	31
33	35	37	39
41	43	45	47
49	51	53	55
57	59	61	63

I will ask a student to think about a number that appears on these cards. Then I will ask the student to check which cards contain the number he/she is thinking and let me know. Now I can immediately reveal the number that the student is thinking.

Problem 1 (for you to think about)

1024 bottles, 1 poisoned

… …

?

… …

10 prisoners

How can you help the ruler who sent you 10 prisoners to test 1024 bottles of wine and identify which bottle is poisoned within 24 hours? Exactly 1 out of 1024 bottles of wine is poisoned and drinking even one drop of such poisoned wine can cause stomachache sometime within 24 hours (the exact time delay between drinking poisoned wine and having stomachache may differ case by case)
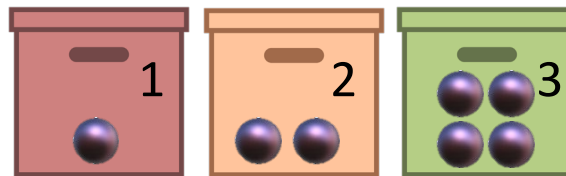
Problem 2 (for you to think about)

- In how many ways can you put 7 identical balls into 3 labelled boxes, where each box may contain 0 to 7 balls, but each ball must be put into one of the 3 boxes?
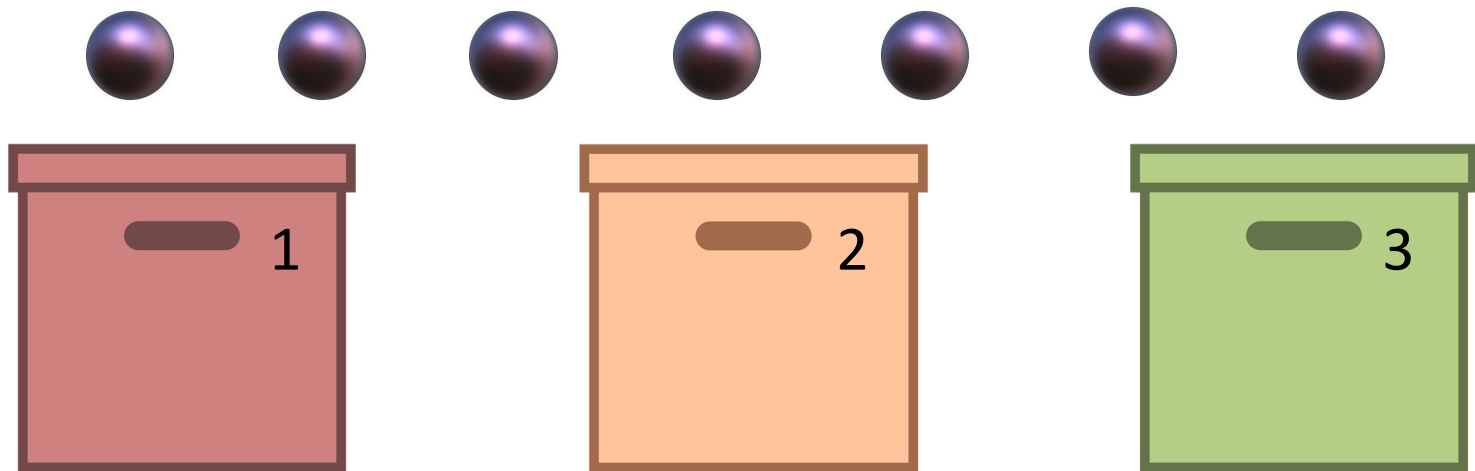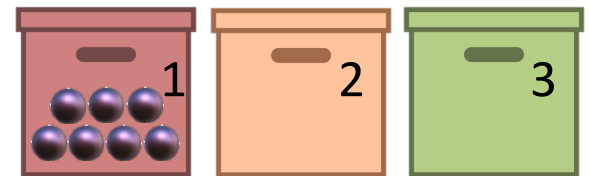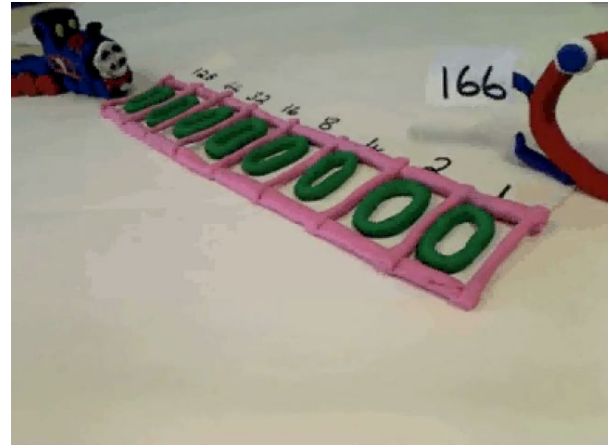
For example:

Way 1   Way 2   Way 3   ......

Video ▶

The Digital Computer (Bits and Bytes, Episode 1)
https://www.youtube.com/watch?v=AdF2uk-EscE
Classic Video from 1983

Fun Binary Conversion
https://www.youtube.com/watch?v=y0BOA1JX8W4

Why U – Decimal, Binary and Hexadecimal
https://www.youtube.com/watch?v=5sS7w-CMHkU

Sign-Magnitude and Two's Complement
https://www.youtube.com/watch?v=Z3mswCN2FJs

Video ▶
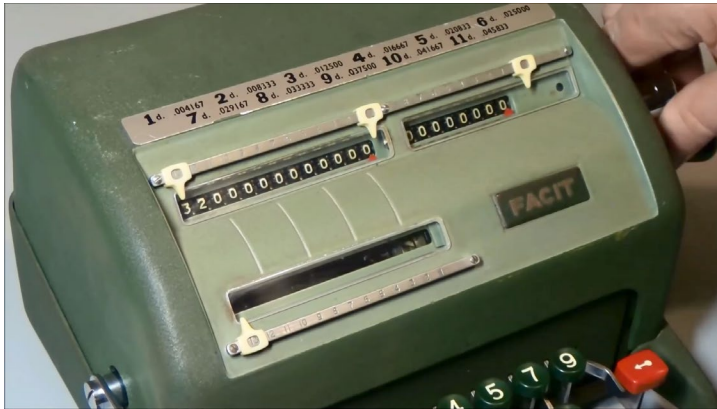
How a Mechanical Computer Handles Division by 0
https://www.youtube.com/watch?v=ByWXR7WzkIE

Difference between 32-bit and 64-bit
https://www.youtube.com/watch?v=B6smnJjNBq8