Lecture 2: Representation

COS10003 Computer Logic and Essentials (Hawthorn)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Semester 1 2021

Today

1. Data
2. Binary
3. Hexadecimal
4. Operations
5. Negative values
6. Floating point
7. Characters

Why computers use binary

The different ways to represent data

Pitfalls and limitations

In the beginning ...

- Computers store and process data
- Data is anything that has interest to the user
- Thoughtfully processed data is called information
- But how do we represent data?

Representations

What is **1 + 1**?

Representations

How about **11** doubled?

Bit and bytes

- Physical devices used to store and process data in a computer are two state devices – on and off
- The smallest unit of data representation is called a bit
- This is short for BInary digiT

Bits and bytes

MSB $\rightarrow$ 0110 0011 0101 1110 $\leftarrow$ LSB

bit : 0 or 1
nibble : 4 bits
byte : 8 bits

How bases work

- Base: the number of different digits/letters that can occur in each position in the number system
- How many digits in base 10/decimal? How many in base 2/binary?
- The relationship between a digit, its position and the base of the number system is expressed as:

$$\text{digit} \times \text{base}^{\text{position \#}}$$

Binary to decimal

What is the value of `0b01110100` in decimal?

0b	0	1	1	1	0	1	0	0
base$^{\text{position \#}}$	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	(128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)

$$= 0 \times 128 + 1 \times 64 + 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 = 116$$

Decimal to binary - Dividing

What is the value of 21 in binary?

$21 \div 2 = 10,$ remainder: 1

$10 \div 2 = 5,$ remainder: 0

$5 \div 2 = 2,$ remainder: 1

$2 \div 2 = 1,$ remainder: 0

0b	0	0	0	1	0	1	0	1
base$^{\text{position \#}}$	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	(128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)

Decimal to binary - Subtracting

What is the value of 21 in binary?

$21 - 32 = ?$ negative

$21 - 16 = 5$

$5 - 8 =$? negative

$5 - 4 = 1$

$1 - 2 =$? negative

$1 - 1 = 0$

0b	0	0	0	1	0	1	0	1
base$^{\text{position \#}}$	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	(128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)

Base 16

- Base 16 or hexadecimal is a shorthand for binary.
- Base 16 has 16 values – but we only have 10 digits??
- Let’s use characters: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Hexadecimal values

Hex	Dec
A	10
B	11
C	12
D	13
E	14
F	15

Hex to decimal conversion

What is the value of `0x35C` in decimal?

0x	3	5	C
base$^{\text{position \#}}$	16^2	16^1	16^0
	(256)	(16)	(1)

$$= 3 \times 256 + 5 \times 16 + 12 \times 1 = 860$$

Hex to decimal conversion

What is 0x2A5 in decimal?

Decimal to hex conversion

- It is trickier to go from decimal to hex.
- Advice is to divide by base and take remainder.
- What is the value of 540 in hexadecimal?

Decimal to hex conversion

$$540 \div 16 = 33, \text{ remainder: } 12$$

$$33 \div 16 = 2, \text{ remainder: } 1$$

0x	2	1	C
base$^{\text{position \#}}$	16^2	16^1	16^0
	(256)	(16)	(1)

Binary Operations

- In binary, this is similar to decimal – line up the digits and add/subtract/multiply them together.

Addition	**Subtraction**	**Multiplication**
$0 + 0 = 0$	$0 - 0 = 0$	$0 \times 0 = 0$
$0 + 1 = 1$	$1 - 0 = 1$	$0 \times 1 = 0$
$1 + 0 = 1$	$1 - 1 = 0$	$1 \times 0 = 0$
$1 + 1 = 0$, 1 carry bit	$0 - 1 = 1$, 1 borrow bit	$1 \times 1 = 1$

Addition in binary

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

Carry bit

Remember 1 + 1 = 10

Sum = 0

Carry = 1

Solution

0b 1110

Do not forget **0b** to indicate your solution is a binary number.

Subtraction

$$
\begin{array}{cccccc}
 & & 1 & 0 & 0 & 1 \\
- & & 0_1 & 1 & 0 & 1 \\
\hline
0b & & 0 & 1 & 0 & 0
\end{array}
$$

Borrow bit

Remember 0 - 1 = 1

With a borrow of 1 from the next digit

Solution

0b 0100

Do not forget **0b** to indicate your solution is a binary number.

Multiplication

$$
\begin{array}{cccccccc}
 & & & & 1 & 0 & 0 & 1 \\
 & & & \times & 0 & 1 & 0 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 1 \\
 & & & 0 & 0 & 0 & 0 & \\
 & & 1 & 0 & 0 & 1 & & \\
 & 0 & 0 & 0 & 0 & & & \\
\hline
0b & 0 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
$$

Solution

0b 0101101

Do not forget **0b** to indicate your solution is a binary number.

Over to you

$$
\begin{array}{ccccc}
 & 1 & 1 & 0 & 0 \\
+ & 1 & 0 & 0 & 1 \\
\hline
\end{array}
\qquad
\begin{array}{ccccc}
 & 1 & 1 & 0 & 1 \\
- & 0 & 0 & 1 & 1 \\
\hline
\end{array}
\qquad
\begin{array}{ccccc}
 & 1 & 0 & 0 & 1 \\
\times & 0 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Lecture 2: Representation

COS10003 Computer Logic and Essentials (Hawthorn)

Semester 1 2021

Up to now

- We've looked at positive values
- Sometimes we need to make use of negative values
  - Sign-Magnitude Representation
  - One's complement
  - Two's complement

Signed vs. unsigned

- Unsigned variables can represent positive values only
- Signed variables can represent positive and negative numbers
- For an 8-bit integer:
  - If unsigned, value can be between 0 and 255
  - If signed, value can be between -128 and 127

Sign-Magnitude Representation

- Convert positive part into binary
- Take the first bit and use it to show sign:
  - 0 = Positive
  - 1 = Negative

Example: what is the value of -21 in `8-bit sign magnitude`?

0b	1	0	0	1	0	1	0	1
base$^{\text{position \#}}$	$Sign$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
		(64)	(32)	(16)	(8)	(4)	(2)	(1)

Problem: Negative 0

One’s complement

- Convert positive number into binary
- Flip all the bits to indicate a negative number

Example: what is the value of -21 in `8-bit` one’s complement?

- 21 in binary: `0b 0001 0101`
- -21 in `8-bit` one’s complement: `0b 1110 1010`

Problem: Negative 0

Two’s complement

- Convert positive number into binary
- Flip all the bits
- Add 1

Example: what is the value of -21 in `8-bit` two’s complement?

- 21 in binary: `0b 0001 0101`
- Flip all the bits (one’s complement): `0b 1110 1010`
- Add 1. -21 in `8-bit` two’s complement: `0b 1110 1011`

Over to you

What is -52 in:

- `8-bit` sign magnitude?
- `8-bit` one's complement?
- `8-bit` two's complement?

Hint: 52 = `0b 0011 0100`

Not all numbers are integers

- Other types of numeric representation are rationals (e.g. $2/3$), or irrationals (e.g. π) which are numbers with a decimal part.
- There are several different ways of representing these, e.g.: fixed-point, or floating-point notation.
- We will focus on the IEEE 754 standard, Single-precision floating-point format.

IEEE 754 standard single-precision

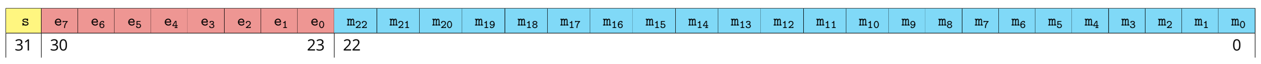
- 32 bits in total to represent a number
- 1 bit for the sign
  - if `s=0` positive
  - if `s=1` negative
- 8 bits for the exponent
  - $Exp = \texttt{0b e}_7\ \texttt{e}_6\ \texttt{e}_5\ \texttt{e}_4\ \texttt{e}_3\ \texttt{e}_2\ \texttt{e}_1\ \texttt{e}_0 - 127$
  - if `0b 0000 0000`, 0 or Denormal number
  - if `0b 1111 1111`, NaN or $\pm\infty$
- 23 bits for the mantissa which determines the value

$$\begin{array}{cccc} \texttt{m}_{22} & \texttt{m}_{21} & \dots & \texttt{m}_0 \\ 2^{(-1)} & 2^{(-2)} & \dots & 2^{(-23)} \end{array}$$

  - Assumed 1 at the MSB

    $M = \texttt{1.m}_{22}\ \texttt{m}_{21}\ \texttt{m}_{20} \dots \texttt{m}_2\ \texttt{m}_1\ \texttt{m}_0$
- Final equation is:

  $Number = (-1)^{\texttt{s}} \times M \times 2^{Exp}$


s
$\texttt{e}_7$ $\texttt{e}_6$ $\texttt{e}_5$ $\texttt{e}_4$ $\texttt{e}_3$ $\texttt{e}_2$ $\texttt{e}_1$ $\texttt{e}_0$
$\texttt{m}_{22}$ $\texttt{m}_{21}$ $\texttt{m}_{20}$ $\texttt{m}_{19}$ $\texttt{m}_{18}$ $\texttt{m}_{17}$ $\texttt{m}_{16}$ $\texttt{m}_{15}$ $\texttt{m}_{14}$ $\texttt{m}_{13}$ $\texttt{m}_{12}$ $\texttt{m}_{11}$ $\texttt{m}_{10}$ $\texttt{m}_9$ $\texttt{m}_8$ $\texttt{m}_7$ $\texttt{m}_6$ $\texttt{m}_5$ $\texttt{m}_4$ $\texttt{m}_3$ $\texttt{m}_2$ $\texttt{m}_1$ $\texttt{m}_0$
31
30
23
22
0

Conversion

Convert to Decimal the following IEEE 754 single-precision number:
`10111110101000000000000000000000`

- `1` `01111101` `01000000000000000000000`
- `Sign = 1`, therefore negative number
- $Exp = \texttt{0b01111101} - 127 = (64 + 32 + 16 + 8 + 4 + 1) - 127 = -2$
- $M = \texttt{1.01000000000000000000000} = 2^0 + 2^{-2} = 1.25$
- $Number = (-1)^{\texttt{s}} \times M \times 2^{Exp} = (-1)^1 \times 1.25 \times 2^{-2} = -0.3125$

Another conversion

Convert to Decimal the following IEEE 754 single-precision number:
`01000011110111101000000000000000`

- `0` `10000111` `10111101000000000000000`
- `Sign = 0`, therefore positive number
- $Exp = \texttt{0b10000111} - 127 = (128 + 4 + 2 + 1) - 127 = 8$
- $M = \texttt{1.10111101000000000000000} =$
  $2^0 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-8} = 1.73828125$
- $Number = (-1)^{\texttt{s}} \times M \times 2^{Exp} = (-1)^0 \times 1.73828125 \times 2^8 = 445$

From number to IEEE 754

Convert 21.3 to IEEE 754 single-precision number.

- Positive number, `Sign = 0`
- Integer 21 = `0b10101`; fraction 0.3?
- $M = \texttt{10101.0100110011001100110}$
- $0.3 \times 2 = \mathbf{0}.6$
- $0.6 \times 2 = \mathbf{1}.2$
- $0.2 \times 2 = \mathbf{0}.4$
- $0.4 \times 2 = \mathbf{0}.8$
- $0.8 \times 2 = \mathbf{1}.6$
- The pattern 0.6, 0.2, 0.4, 0.8 gets repeated which will give `1001`
- We stop at 0 or `24 bits`
- $\texttt{1.01010100110011001100110} \times 2^4$
- $M =$ `01010100110011001100110`
- $Exp = 4 + 127 = 131 =$ `10000011`
- `0` `10000011` `01010100110011001100110`
- Value = 21.299999237060546875

Things to note

- Other formats
  - Single precision (32 bits): $\pm 1.18 \times 10^{-38} to \pm 3.4 \times 10^{38}$ (Approx. 7 decimal digits)
  - Double precision (64 bits): $\pm 2.23 \times 10^{-308} to \pm 1.80 \times 10^{308}$ (Approx. 16 decimal digits)
  - And so on.
- The bias value for the exponent differs for each format
- More bits means more accuracy, however also means more space is needed

Things that are not numbers

- Another data type we work with is characters, and by extension strings.
- Strings are an ordered collection of characters, such as ”computer”.
- Characters include:
  - Letters: a-z, A-Z
  - Digits: 0-9
  - Symbols: !, @, *, /, &, #, $
  - Control characters: `<CR>`, `<BEL>`, `<ESC>`, `<LF>`

ASCII

- De facto worldwide standard for the code numbers used by computers to represent upper and lower case letters, numbers and punctuation.
- There are 128 standard ASCII codes, which can be represented by 7 digit binary numbers (000 0000 to 111 1111).

Example encoding

```
Computer & Logic Essentials 2019.
```

```
43 6f 6d 70 75 74 65 72 20 26 20 4c 6f 67 69 63 20 45 73 73 65 6e 74 69 61 6c 73 20
32 30 31 39 0a
```

Things to note

- Codes for digits are not the same as the numeric value
- Supported by almost every computer
- However most languages need more than 128 characters:
  - Extended ASCII codes represent up to 256 characters
  - Most commonly used extended variant is ISO 8859-1 (Latin 1).

Unicode

- Unicode is designed to overcome the limitation of the number of characters. It assigns unique character codes to characters in a wide range of languages.
- Defines 1,114,112 code points from 0x0 to 0x10FFFF. For example:
  - U+0030: DIGIT ZERO
  - U+2070: SUPERSCRIPT ZERO
  - U+1F602: FACE WITH TEARS OF JOY

See https://en.wikipedia.org/wiki/Plane_(Unicode) for a description of the different planes.

General idea behind encoding

- UTF-8 is commonly used and is a variable-width encoding that keeps compatibility with ASCII
- UTF-16 is also variable width
- UTF-32 is fixed width (32 bits per code point)
- Plus others that are either not commonly used or superseded.

UTF-8: number of bytes

From	To	# bytes	Bits used	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	1	7	0xxxxxxx			
U+0080	U+07FF	2	11	110xxxxx	10xxxxxx		
U+0800	U+FFFF	3	16	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	U+10FFFF	4	21	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Representation of UTF-8: one byte

`U+0030`: DIGIT ZERO

UTF-8 represents everything between U+0000 and U+007F with one byte, using 7 bits prefaced by a 0.

So `U+0030` is `0011 0000` or `30` in hex.

Representation of UTF-8: three bytes

`U+2070`: SUPERSCRIPT ZERO

UTF-8 represents everything between U+0800 and U+FFFF with three bytes, using the format `1110xxxx 10xxxxxx 10xxxxxx`.

So `U+2070` is

`0010 0000 01 11 0000`

to start with, then as UTF-8 becomes

`1110 0010 10 00 0001 10 11 0000`

`= E2 81 B0.`

Representation of UTF-8: two and four bytes

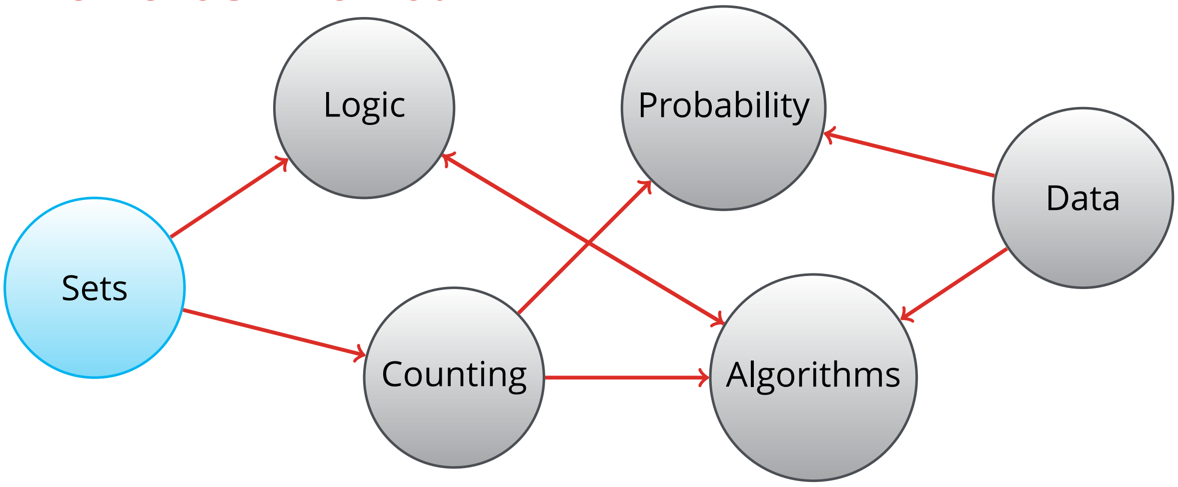
- For code points between U+0080 and U+07FF the format is 110xxxxx 10xxxxxx for two bytes.
- UTF-8 represents everything between U+10000 and U+10FFFF with four bytes, using the format 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx.

This is provided as a table on the formula sheet for the exam.

Reflecting

- Why do computers use binary instead of decimal or other bases?
- How are different types of data represented?
- What are some of the pitfalls and limitations of data representation?

Where to next?



In which we learn how to make collections of objects.

Questions I still have

Topics I need to review