Week 1: Binary numbers

Some quick facts on binary numbers:

- The base for binary numbers is 2
- Radix = 2
- The symbols are: 0,1

Similarly to decimals, the positional value of each binary digit indicates the corresponding power of 2, as follows:

Digit	1	1	0	0	1	1	0	1
Power	128	64	32	16	8	4	2	1
Exponent	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- Decimal is a **base-10** number system
- Binary is a **base-2** number system
  each position represents a power of 2
- Consider binary 1101001:

1 1 0 1 0 0 1

$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$

64 + 32 + 0 + 8 + 0 + 0 + 1
which is 105 (decimal)

- To avoid confusion, we can indicate a number's base as a subscript or some other mechanism
- e.g. 105 dec = 105_{10}
- With 4 bits we can represent:

- 0000 = 0_{10}
- 0001 = 1_{10}
- 0010 = 2_{10}
- 0011 = 3_{10}
- 0100 = 4_{10}
- 0101 = 5_{10}
- 0110 = 6_{10}
- 0111 = 7_{10}
- 1000 = 8_{10}
- 1001 = 9_{10}
- 1010 = 10_{10}
- 1011 = 11_{10}
- 1100 = 12_{10}
- 1101 = 13_{10}
- 1110 = 14_{10}
- 1111 = 15_{10}

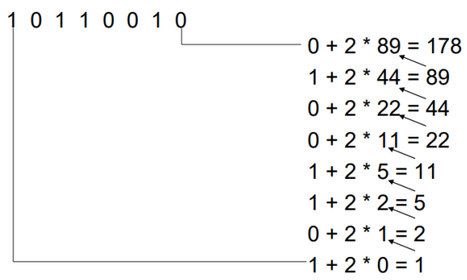
- Each bit position (from right to left) has a weight that is $2^{\text{bit number}}$
  – e.g. position 0 (right-most) has weight 2^0, position 1 has weight 2^1 etc
- With n bits, we can represent values 0 to $2^n - 1$
  – e.g. with 3 bits we can have 0 to $2^3 - 1 = 7$
- The left-most bit is the most-significant bit (MSB)
- The right-most bit is the least-significant bit (LSB)

Successive halving can be used to convert base-10 numbers to base-2 remainder

178/2	0 (LSB i.e. rightmost bit)
89/2	1
44/2	0
22/2	0
11/2	1
5/2	1
2/2	0
1/2	1 (MSB)

Result: $178_{10} = 10110010_2$

Successive doubling can be used to convert base-2 to base-10 numbers



Alternatively, just add the relevant powers of 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	1	0	0	1	0

128 + 32 + 16 + 2 = 178

NB: If you are feeling confused:

1. It may help to do examples using base 10
2. There will be plenty of Tutorial and Practical exercises on these things.

Binary addition:

1 ("carries"- always show)

01010101 +

00010010

01100111

1 1 1 1 ("carries"- always show)

01110101+

00010110

10001011

Carry condition

- Computers can only allocate a finite number of bits to represent a number
- Assume only 8 bits are used for each integer

1 1 1 1 1 1 1 (always show "carries")

1 1 1 1 1 1 1 1 + (255_{10})

0 0 0 0 0 0 0 1

---

1 00000000

this bit is 'lost', this error is known as a 'carry condition'

- The integers we have looked at have been unsigned
- There are other binary coding systems (e.g. two's complement) for representing signed integers
- There are also binary coding systems for representing floating point numbers

- These topics are outside the scope of this course

Some numbers have meaning for different bases, such as 10010 which may be a binary, octal, decimal or hexadecimal number. Hence, it is often necessary to indicate the basis with a little subindex, such as in
10010_2, 74562_8 and 74562_{10}. Sometimes a prefix is used, such as % for binary and $ for hexadecimal.

Octal representation

Expecting human programmers to read/write binary numbers is fraught with danger - humans find binary difficult to work with accurately.

Need another number representation that is easier for humans to use, and easy to convert to/from binary - Octal.

Binary is base 2. Octal is base 8. Available symbols are 0,1,2,3,4,5,6,7.

Consider 2053_8.

$2 * 8^3 + 0 * 8^2 + 5 * 8^1 + 3 * 8^0$

$2 * 512 + 0 * 64 + 5 * 8 + 3 * 1 = 1067_{10}$

- Binary for 1067_{10}

  10000101011

  - break the binary into groups of 3 (from LSB)

    10 000 101 011

  - write binary triplets as $base_{10}$

    2 0 5 3

Hexadecimal representation

As we have seen, binary numbers are very easy for computers to store and manipulate, but for humans they are quite uncomfortable to remember and use. The number 01011110011001111111000000101 is very hard to remember for a human being, while its equivalent in hexadecimal - as we shall see - is AF33D05, which is much easier to recall.

Quick facts:

- The base for hexadecimal numbers is 16.
- Radix = 16 (24)
- The symbols are: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Positional Values	16^4	16^3	16^2	16^1	16^0
Decimal Values	65536	4096	256	16	1

Therefore, the number:

$1D7E_{16}$

$= 1 * 16^3 + D * 16^2 + 7 * 16^1 + E * 16^0$

$= 1 * 4096 + 13 * 256 + 7 * 16 + 14 * 1$

$= 7550_{10}$

It is simple to convert an octal symbol into a binary triplet, and vise-versa. Hexadecimal numbers can be used to represent binary nibbles. With 4 bits, we can represent 16 symbols:

- Decimal only has 10 symbols 0..9
- So, add 6 more symbols A,B,C,D,E,F

With 4 bits we can represent:

- 0000 = 0_{16}
- 0001 = 1_{16}
- 0010 = 2_{16}
- 0011 = 3_{16}
- 0100 = 4_{16}
- 0101 = 5_{16}
- 0110 = 6_{16}
- 0111 = 7_{16}
- 1000 = 8_{16}
- 1001 = 9_{16}
- 1010 = A_{16}
- 1011 = B_{16}
- 1100 = C_{16}
- 1101 = D_{16}
- 1110 = E_{16}
- 1111 = F_{16}

- Consider binary for 1067_{10}

  10000101011

  – break the binary into groups of 4 (from LSB)

  100 0010 1011

  – write binary quartets as $base_{16}$

  4 2 B

  – Binary 10000101011 is hexadecimal 42B

Characters

There are 2 main codes in use for representing characters as binary:

- ASCII (American Standard Code for Information Interchange) used by many older programming languages
- Unicode is a newer series of codes, ranging in size from 8 bit, 16 or 32 bit. The larger number of bits allows a huge number of (non-European) characters to be represented. Currently over 110000 characters have been defined. Used by the Java programming language and .NET

An old code you might hear about was EBCDIC: used in some IBM mainframe systems.

Some web pages use different coding schemes, especially Large Alphabet Languages (mainly CJK), example: **http://www.sina.com.cn/** **(http://www.sina.com.cn/)** .

- Use “view->developer->view source”
- You will find a reference to gb 2312, which this browser understands to be a commonly used coding standard for Chinese (Guo Biao 2312)
- Again it is all binary data, the important thing is the program (browser) knows how to interpret it.

• ASCII is a 7 bit code (often extended to 8 bits)

• Note: 1 is not ‘1’ !!

• Some ASCII characters:

01000001 = ‘A’

01000010 = ‘B’

01000011 = ‘C’

...

01011010 = ‘Z’

...

01100001 = ‘a’

01100010 = ‘b’

00110000 = ‘0’

00110001 = ‘1’

00110010 = ‘2’

00100000 = ‘ ‘

00100001 = ‘!’

00100100 = ‘$’

00000111 = ring bell !!

A question in the practical refers to the 'duosexagesimal' character set. This suggests the use of binary codes as characters.

Such use has a long history - from the 1950's - culminating in the**ASCII** **(http://www.asciitable.com/)** set.

IBM tried to introduce an incompatible set called EBCDIC.

Even then, it was clear that 128 characters was too few for most cultures. The ISO attempted to standardise using character sets such as ISO-1158, and vendors such as Microsoft had their own code pages. They all did this by dumping the use of parity bits, which gave them 128 more characters to use. The lower 128 were standard across all forms.

Quiz

1. What is the largest unsigned base-10 integer that can be stored in 6 bits?
2. Given 01000001 is binary for ASCII ‘A’, what character is represented by 01000100 ?
3. Assuming an 8 bit integer representation, convert 107_{10} to binary. Then convert it to Octal, and Hexadecimal.
4. Convert 11111011110_2 to Octal and decimal.
5. Convert $9A9_{16}$ and then $DEAD_{16}$ to decimal.