Computer Systems Internals – Week 1 - Binary (base 2)

Binary is a system of representing numbers using digits that correspond to powers of 2.

Compare this with the **decimal** numbers we normally use where the digits correspond to powers of 10. Although you wouldn't normally think of it this way, the decimal number 64791 can be expanded as

$6\times10000 + 4\times1000 + 7\times100 + 9\times10 + 1$

or $6\times10\times10\times10\times10 + 4\times10\times10\times10 + 7\times10\times10 + 9\times10 + 1$

or $6\times10^4 + 4\times10^3 + 7\times10^2 + 9\times10^1 + 1\times10^0$

10000, 1000, 100, 10 and 1 are the first 5 "powers of 10".

- We can make any number by adding together powers of 10
- If we limit the number of times we can use each power of 10 to a single digit (0-9) then there is a unique combination of powers that can be added to make each number
  - We use 10000 5 times
  - We use 100 3 times
  - We use 1 8 times
  - The number is therefore 50308
    - we can't use 1000 50 times, therefore there is only one way to make this number
- Because we used powers of 10, and because each digit has ten possible values (0-9), this number is said to be written in "base 10", or "decimal"

Binary is a similar concept. But we construct numbers out of powers of 2, and each binary digit has two possible values (0 and 1). "Binary digit" is usually shortened to "**bit**".

8, 4, 2 and 1 are the first 4 "powers of 2"

- We can make any number by adding together powers of 2
- If we limit the number of times we can use each power of 2 to a single binary digit (0 or 1) then there is a unique combination of powers that can be used to make each number. For example, consider the binary number is therefore 0101
  - We don't use 8
  - We use 4 once
  - We don't use 2
  - We use 1 once
- Because we used powers of 2, and because each digit has 2 possible values (0 and 1), this number is said to be written in "base 2", or "binary"
- If we use decimal arithmetic to add up the powers that we have used

  $0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 1\times4 + 1\times1 = 5$
- **So we can say that binary 0101 and decimal 5 are different representations of the same number**
  - $0101_2 = 5_{10}$
  - if we convert decimal 5 into binary, we get 0101
  - if we convert binary 0101 into decimal, we get 5

Binary may have been invented as a purely abstract concept, with no particular use in mind. But binary storage and adding systems are extremely simple to implement in electronic circuitry, and so virtually all computers use binary for the internal representation of data. **Binary is therefore an important concept to understand.**

I know from experience that some people aren't confident with their ability to convert numbers between decimal and binary, and some people have had limited prior opportunity to find out if they are confident. I hope this handout will address both types of user.

Use of binary is very important for any low-level computing activity, particularly digital communications but also computer hardware, operating systems, graphics and very-low-level programming. It is therefore important that any practising computer professional be comfortable with use of binary. This is not something you will be directly assessed on. It is basic declarative knowledge on which you can build the skills that you **will** be assessed on.

Converting between binary and decimal (in both directions) is a fundamental skill for students of computer science and related subjects, and it is not hard when you take the time to practice. The only mathematical skill it needs is adding small numbers together, and dividing by 2.

Practising this is a good group activity - one person invents a decimal number (say in the range 100-255, but any number can be used) and converts it to binary. Another person sees only the binary and converts it back to decimal. Then you see if the end result matches the original number)

It is vital to know accurately the powers of 2 up to 2^7.

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

These numbers should become **very** familiar to you. Basically, conversion between decimal and binary can be done simply by adding or subtracting these numbers in the right way.

1. Converting from decimal to binary

A simple method of decimal to binary conversion is a just to work out which powers of 2 you add together to get the decimal number you want to convert. Each power of 2 can only be used once.

For example, to convert 153:

- this is clearly more than 128 but less than 256
- 128+32=160, which is too high
- 128+16=144, so another 9 is needed. This leads us to
- 128+16+8+1=153

Then write the powers of 2 in a table (lowest to the right) and put a 1 next to the ones you have used and a 0 next to the ones you have not used.

Powers	128	64	32	16	8	4	2	1
Was it used?	1	0	0	1	1	0	0	1

So the binary for 153 is simply the second row of numbers, written left to right.

10011001

The bit that has the value “1” is known as the “least significant bit” and it put on the right of the number

A different way of converting decimal to binary is by repeated division. Try 135

135÷2=67	remainder	1
67÷2=33	remainder	1
33÷2=16	remainder	1
16÷2=8	remainder	0
8÷2=4	remainder	0
4÷2=2	remainder	0
2÷2=1	remainder	0
1÷2=0	remainder	1

Stop when you get to 0. Then write down the remainders, reading from the bottom upwards

10000111

This is the binary equivalent of 135

2. Converting from binary to decimal

Binary to decimal conversion is simply a matter of adding powers together that correspond to the "1"s in the binary number.

So for the binary number 10101010

Powers	128	64	32	16	8	4	2	1
Was it used?	1	0	1	0	1	0	1	0
Multiply	128	0	32	0	8	0	2	0

128+32+8+2=170, which is the decimal equivalent

And for 01010101

Powers	128	64	32	16	8	4	2	1
Was it used?	0	1	0	1	0	1	0	1
Multiply	0	64	0	16	0	4	0	1

64+16+4+1=85

Testing yourself

There is an application you can use at

http://www.firstyearmatters.info/binary

which will set you questions for decimal→binary and binary→decimal conversion, and tell you if you got the answer right.

- "simple" uses 4-bit numbers (0 to 15), which are OK for initial practice
- eventually you need to be confident with 8-bit numbers (0 to 255)

A footnote

In the real world you will often use calculators to do this kind of thing. However, calculators have the problem that if you press the wrong buttons they give you the wrong answer (known in techie-speak as GIGO – "Garbage In Garbage Out"). Therefore, being able to do the calculations in your head, even only approximately, will occasionally lead you to think "that doesn't seem right, I'd better check the calculation"

You don't need to be a "Countdown" contestant to be able to do these sorts of sums in your heads quickly and accurately. However, there is no substitute for practice. This is why I want you to be able to do these things manually.

More powers of 2

Following on from what I said earlier, it is useful to know accurately the powers of 2 all the way up to 2^{16}

$2^0 = 1$	
$2^1 = 2$	$2^9 = 512$
$2^2 = 4$	$2^{10} = 1024$
$2^3 = 8$	$2^{11} = 2048$
$2^4 = 16$	$2^{12} = 4096$
$2^5 = 32$	$2^{13} = 8192$
$2^6 = 64$	$2^{14} = 16384$
$2^7 = 128$	$2^{15} = 32768$
$2^8 = 256$	$2^{16} = 65536$

These are important, because an n-bit number can hold 2^n different possible combinations of 0s and 1s, from 0 to 2^n-1. **NOTE "bit" means "binary digit"**

So for n=3, there are 2^3 (8) combinations, from 0 to 2^3-1 (7) shown in the following table

Binary	Decimal Equivalent
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Approximating Higher Powers

Higher powers can be approximated easily using some basic maths. For example

$2^{21} = 2^{10} \times 2^{10} \times 2^1$ (since 21=10+10+1)

If we take the **approximation** that 2^{10}=1000 then we get

$2^{21} \approx 1000 \times 1000 \times 2 \approx 2$ million

and similarly

$2^{32} \approx 1000 \times 1000 \times 1000 \times 4 \approx 4$ billion

This works for any power, so

$2^{75} \approx 2^5 \times 1000^7$ which would more traditionally be written 32×10^{21}

$2^{128} \approx 2^8 \times 1000^{12}$ which would more traditionally be written 256×10^{36}

Note these are only approximations, done without any use of calculators. Clearly the higher the power, the less accurate the approximation. However, they are useful for order-of-magnitude calculations (on, for example, the strength of security keys).

Other bases

Computer Scientists make use of two other bases, but these are really only used as more convenient ways of writing down binary numbers (binary numbers are generally hard to remember)

Octal (base 8)

In octal, each digit takes the value 0→7. Note that 0→7 is precisely the numbers that can be represented using three bits (three binary digits) so decimal 0→7 is binary 000→111

Let's use 64791 as an example. Decimal 64791 is equivalent to binary 1111110100010111

If we group the binary into groups of three bits (starting from the least significant bit), we get

1	111	110	100	010	111
1	7	6	4	2	7

Therefore 64791 decimal is equivalent to 176427 octal (it is conventional to write octal numbers with a leading zero, so it would be **0176427**

To check, multiply it out

$$1\times8^5 + 7\times8^4 + 6\times8^3 + 4\times8^2 + 2\times8 + 7 = 64791$$

Hexadecimal (base 16)

In hexademical (usually called "hex"), each digit takes the value 0→15. Note that 0→15 is precisely the numbers that can be represented using four bits (four binary digits) so decimal 0→15 is binary 0000→1111

To use 0→15 conveniently, we use "A" for 10, "B" for 11, "C" for 12, "D" for 13, "E" for 14 and "F" for 15.

As before, let's use decimal 64791 – which is equivalent to binary 1111110100010111

If we group the binary into groups of four bits, we get

1111	1101	0001	0111
F	D	1	7

Therefore 64791 decimal is equivalent to FD17 hexadecimal (it is conventional to write hex numbers with a leading "0x", so it would be **0xFD17**

To check, multiply it out

$$15\times16^3 + 13\times16^2 + 1\times16 + 7 = 64791$$