Data Representation – Binary Numbers

- While in most of applications the base 10 system is used to represent numerical values e.g. :

$$345.501 = 3x10^2 + 4x10^1 + 5x10^0 + 5x10^{-1} + 0x10^{-2} + 1x10^{-3}$$

- We can see that the number is a sum of coefficients multiplied by the base taken to different powers (exponents).

- The binary (base 2) system follows a similar structure.

Integer Conversion Between Decimal and Binary Bases

- Task accomplished by
  - Repeated ***division*** of decimal number by 2 (integer part of decimal number)
  - Repeated ***multiplication*** of decimal number by 2 (fractional part of decimal number)
- Algorithm
  - Divide by target radix (r=2 for decimal to binary conversion)
  - ***Remainders*** become digits in the new representation (0 <= digit < 2)
  - Digits produced in right to left order
  - ***Quotient*** used as next dividend
  - Stop when the quotient becomes zero, but use the corresponding remainder

Convert Decimal to Binary

$$345.865 = 3x10^2 + 4x10^1 + 5x10^0 + 8x10^{-1} + 6x10^{-2} + 5x10^{-3}$$

- First separate the number into two integers: 345 (before decimal place) and 865 after decimal place.
- We will next divide 345 by 2 to obtain 172 with a remainder of 1 (172.5 is 172+1/2). This indicates that the least significant bit is one
- This process is repeated until the integer goes to zero.

Convert Decimal to Binary

- First 345/2 = 172 (remainder 1) – Least Significant Bit (LSB)
- Next 172/2= 86 (remainder 0)
- Then 86/2 = 43 (remainder 0)
- Then 43/2 = 21 (remainder 1)
- Then 21/2 = 10 (remainder 1)
- Then 10/2 = 5 (remainder 0)
- Then 5/2 = 2 (remainder 1)
- Then 2/2 = 1 (remainder 0)
- Then 1/2 = 0 (remainder 1) – Most Significant Bit (MSB)
- End.

This will lead to a binary number {101011001} MSB…...LSB

1+0+0+8+16+0+64+0+256 = 345

Fractional Decimal-Binary Conversion

- Whole and fractional parts of decimal number handled independently
- To convert
  - Whole part: use ***repeated division*** by 2
  - Fractional part: use ***repeated multiplication*** by 2
  - Add both results together at the end of conversion
- Algorithm for converting fractional decimal part to fractional binary
  - Multiply by radix 2
  - Whole part of product becomes digit in the new representation (0 <= digit < 2)
  - Digits produced in left to right order
  - Fractional part of product is used as next multiplicand.
  - Stop when the fractional part becomes zero
    (sometimes it won't)

Convert Decimal to Binary

- In the case of the portion of the number to the right of the decimal place we would perform a multiplication process with the most significant bit coming first.
- First 0.865 x 2 = 1.730 (first digit after decimal is 1)
- Next 0.730 x 2 = 1.460 (second digit after decimal is 1)
- Then 0.460 x 2 = 0.920 (third digit after decimal is 0)
- Then 0.920 x 2 = 1.840 (fourth digit after decimal is 1)

Note that if the term on the right of the decimal place does not easily divide into base 2, the term to the right of the decimal place could require a large number of bits. Typically the result is truncated to a fixed number of decimals.

The binary equivalent of 345.865 = 101011001.1101

Binary Coded Hex Numbers

Decimal	Binary	Hex
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

Decimal to Hex

From a previous example we found that the decimal number 345 was 101011001 in binary notation.

- In order for this to be represented in hex notation the number of bits must be an integer multiple of four. This will require the binary number to be written as:

  0001 0101 1001 (the spaces are for readability).

- This will lead to a hex representation of $159

(this is not to be confused with a decimal number of one hundred and fifty nine. Often the letter "$" is placed at the beginning of a hex number to prevent confusion (e.g. $159).

Integer Number Representation: 3 ways to represent

Representation using 8-bit numbers

- sign-and-magnitude representation
  - MSB represents the sign, other bits represent the magnitude.

  Example:

  +14 = 0000 1110

  -14 = 1000 1110

- In all three systems, leftmost bit is 0 for +ve numbers and 1 for –ve numbers.

Integer Number Representation: 3 ways to represent

Representation using 8-bit numbers

- signed 1's complement representation
  - one's complement of each bit of positive numbers, even the signed bit

Example:

+14 = 0000 1110

-14 = 1111 0001

Note that 0 (zero) has two representations:

+0 = 0000 0000

-0 = 1111 1111

Integer Number Representation: 3 ways to represent

Representation using 8-bit numbers

- signed 2's complement representation
  - two's complement of positive number, including the signed bit, obtained by adding 1 to the 1's complement number

Example:

+14 = 0000 1110

-14 = 1111 0001 + 1 = 1111 0010

Note that 0 (zero) has only one representation

+0 = 0000 0000

-0 = 1111 1111 + 1 = 0000 0000

Arithmetic Addition

- Signed-magnitude:

Example: addition of +25 and -37

- – Compare signs
  - If same, add the two numbers
  - If different
    - – Compare magnitudes
      - » Subtract smaller from larger and give result the sign of the larger magnitude

+25 + -37 = - (37-25) = -12

Note: computer system requires comparator, adder, and subtractor

Arithmetic Addition

- 2's complement numbers: only addition is required
  - Add two numbers including the sign bit
  - Discard any carry
  - Result is in 2's complement form

Example: addition of +25 and -37

  0001 1001 (+25)

\+ 1101 1011 (-37)

  1111 0100 (-12)

Arithmetic Subtraction

- 2's complement numbers: only addition and complementation
  - Take 2's complement of B, add it to A

$$\underline{+}A\ -(+B) = \ \underline{+}A + (-B)$$
$$\underline{+}A\ -(-B) = \ \underline{+}A + (+B)$$

  - Discard any carry, Result is in 2's complement form

Example: $(-6) - (-13) = -6 + 13$

1111 1010 (-6)

+ <u>0000 1101</u> (+13)

1 0000 0111 (+7)

Overflow

- When sum of two *n* digit numbers result in a *n+1* digit number
  - Occurs when both numbers are either +ve or –ve
- Range for a 4-bit number is –8 through +7
- Range for a 8-bit number is –128 through +127

$$(-2^{n-1}) \text{ to } (+2^{n-1} - 1)$$

- Overflow is detected (occurs) when carry into sign bit is not equal to carry out of sign bit

Overflow

Example:

0 100 0110 (+70)	1 011 1010 (-70)
+ 0 101 0000 (+80)	+ 1 011 0000 (-80)
0 1 001 0110 (+150)	1 0 110 1010 (-150)

Overflow is detected (occurs) when carry into sign bit is not equal to carry out of sign bit

- the computer will often use an overflow flag (signal) to indicate this occurrence.

Binary Multiplication

Procedure similar to decimal multiplication

Multiplication in base 10

		2	3	
*		4	6	
	1	3	8	6*23
	9	2	0	40*23
1	0	5	8	Sum

Note that 40*23 = 920 can be represented by shifting 4*23 =92 one position left and inserting a 0 in the vacated position

Example of binary multiplication (positive multiplicand)

Multiplication in binary

	7	6	5	4	3	2	1	0	Column number
A	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	First number = 19
B	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	Second number = 13
	0	0	0	1	0	0	1	1	Number A times B0
	0	0	0	0	0	0	0	0	Number A times B1 shifted left by 1
	0	0	0	1	0	0	1	1	Sum
	0	1	0	0	1	1	0	0	Number A times B2 shifted left by 2
	0	1	0	1	1	1	1	1	Sum
	1	0	0	1	1	0	0	0	Number A times B3 shifted left by 3
	1	1	1	1	0	1	1	1	Sum
	-	-	-	-	-	-	-	-	Process continues. Not shown as all other Bi = 0
	1	**1**	**1**	**1**	**0**	**1**	**1**	**1**	Final result = 247

Binary Multiplication (cont.)

Example of binary multiplication (negative multiplicand)

```
Multiplicand M   (-14)                    1 0 0 1 0
Multiplier Q     (+11)                  x 0 1 0 1 1
                                          ---------
Partial product 0                     1 1 1 0 0 1 0
                                    + 1 1 0 0 1 0
                                    ---------------
Partial product 1                   1 1 0 1 0 1 1
                                  + 0 0 0 0 0 0
                                  ---------------
Partial product 2                 1 1 1 0 1 0 1
                                + 1 1 0 0 1 0
                                ---------------
Partial product 3               1 1 0 1 1 0 0
                              + 0 0 0 0 0 0
                              -----------------
Product P    (-154)             1 1 0 1 1 0 0 1 1 0
```

Binary Division

- Binary division similar to decimal - can be viewed as inverse of multiplication
  - Shifts to *left* replaced by shifts to *right*
    - Shifting by one bit to left corresponds to multiplication by 2, shifting to right is division by 2
  - Additions replaced by subtractions (in 2's complement)
    - Requires comparison of result with 0 to check whether it is not negative
- Unlike multiplication, where after finite number of bit multiplications and additions result is ready, division for some numbers can take *infinite* number of steps, so assumption of termination of process and precision of approximated result is needed

Binary Division – cont.

Division in binary

Divisor = 13	7	6	5	4	3	2	1	0	Column number
				1	**0**	**0**	**1**	**1**	Result of division
1 1 0 1	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	Number to be divided = 247
	1	1	0	1	0	0	0	0	**1** *13*16 = 208
	0	0	1	0	0	1	1	1	Result of subtraction = 39 (non-negative)
	0	0	0	0	0	0	0	0	**0** *13*8 = 0 since 1*13*8 = 104 when subtracted from 39 would give a negative
	0	0	1	0	0	1	1	1	Result of subtraction = 39 (non-negative)
	0	0	0	0	0	0	0	0	**0** *13*4 = 0 since 1*13*4 = 52 when subtracted from 39 would give a negative
	0	0	1	0	0	1	1	1	Result of subtraction = 39 (non-negative)
	0	0	0	1	1	0	1	0	**1** *13*2 =26
	0	0	0	0	1	1	0	1	Result of subtraction = 13 (non-negative)
	0	0	0	0	1	1	0	1	**1** *13*1 =13
	0	0	0	0	0	0	0	0	Result of subtraction = 0 (non-negative)

Floating Point Numbers

The range of values for a 32-bit +ve integer number is

$$2^{32} \approx 4.3x10^9$$

As +ve and –ve integers, range is

$$\approx 0 \; to \; \pm 2.15 \; x \; 10^9$$

As fractions range is ≈ $\pm 4.55 \; x \; 10^{-10}$ to ± 1

These ranges are not sufficient for scientific calculations.

It would be useful to be able to use floating point notation.

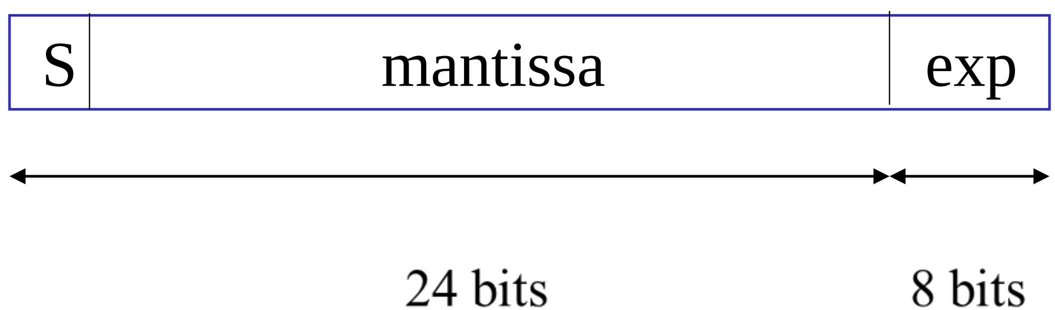
Floating Point Numbers

Consider the number 6132.789

$= +0.6132789\ x\ 10^{+4}$ (decimal point is 4 positions to the right)

$= \pm m\ x\ r^{e}$

where *m* is mantissa, *e* is exponent, and *r* is radix



One bit – the S bit – represents the sign of the number

IEEE Standard

Example:

Unnormalized form: 0.0010110… x 2^9

Normalized form: 1.0110… x 2^6

The 1 before the decimal point does not need to be represented as it is always 1 in normalized form.

S	exp E'	mantissa M

S: 1-bit sign of the number

M: 23-bits mantissa

E': 8-bit signed exponent in excess-127 format

$E' = E + 127$, where E is the actual value of the exponent

IEEE Standard

S	exp E'	mantissa M

Number = $\pm 1.M\ x\ 2^{E'-127}$

Example:

S = 0

M = 00101010000000000000000

E' = 00101000 => E' = E +127 => 40 = E +127 => E = -87

The number is therefore 1.001010 $x\ 2^{-87}$

Note that E' is in the range of $0 \leq E' \leq 255$

0 and 255 has special values, therefore E' is $1 \leq E' \leq 254$,

=> E is in the range of $-126 \leq E \leq 127$

When E' = 0 and M = 0, it represents value exact of 0.

When E' = 255 and M = 0, it represents value of ∞.

When E' = 255 and $M \neq 0$, it is *Not a Number* (NaN), due to the result of performing invalid operation like 0/0 or $\sqrt{-1}$

When E' = 0 and $M \neq 0$, value is $\pm 0.M\ x\ 2^{-126}$. The number is smaller than the smallest normal number -> used for gradual underflow.

Convert Decimal to IEEE format

Decimal number = 2036

Hex equivalent = 07F4

Binary equivalent = 0111 1111 0100 = 01.1111110100 x 2^{10}

$$E' = E + 127 = 10 + 127 = 137 = 1000\ 1001_2$$

Therefore:

S =0, E'= 1000 1001, M = 1111 1101 0000 0000 0000 000

Now try doing reverse, converting Floating point to Decimal:

Number is 1.1111110100 x 2^{10}, since $E' = E + 127$ => E = 10.

= (1 + 1 x 2^{-1}+ 1 x 2^{-2} + 1 x 2^{-3} + 1 x 2^{-4} + 1 x 2^{-5} + 1 x 2^{-6} + 0 x 2^{-7} + 1 x 2^{-8} + 0 x 2^{-9} + 0 x 2^{-10}) x 2^{10}

= 1.98828125 x 2^{10}

= 2036