Fractions (Floating Point Numbers)

- **A part of a whole.**
- **Approximation of the real number line, with limitations imposed by the number of places used.**
- **Consider a decimal number expressed with not more than 5 decimal places.**
  - **How would you represent 0.0000138? Because of the limitation on the representation (only 5 places), we cannot accurately represent this number. Instead, we would use an approximation: 0.00001 This is an example of a *roundoff* error**
  - **If we needed to represent the remaining 0.0000038 of the number, we are out of luck. This is called *underflow error***

Overflow and Underflow

Signed 3-digit fraction + signed 2 digit exponent

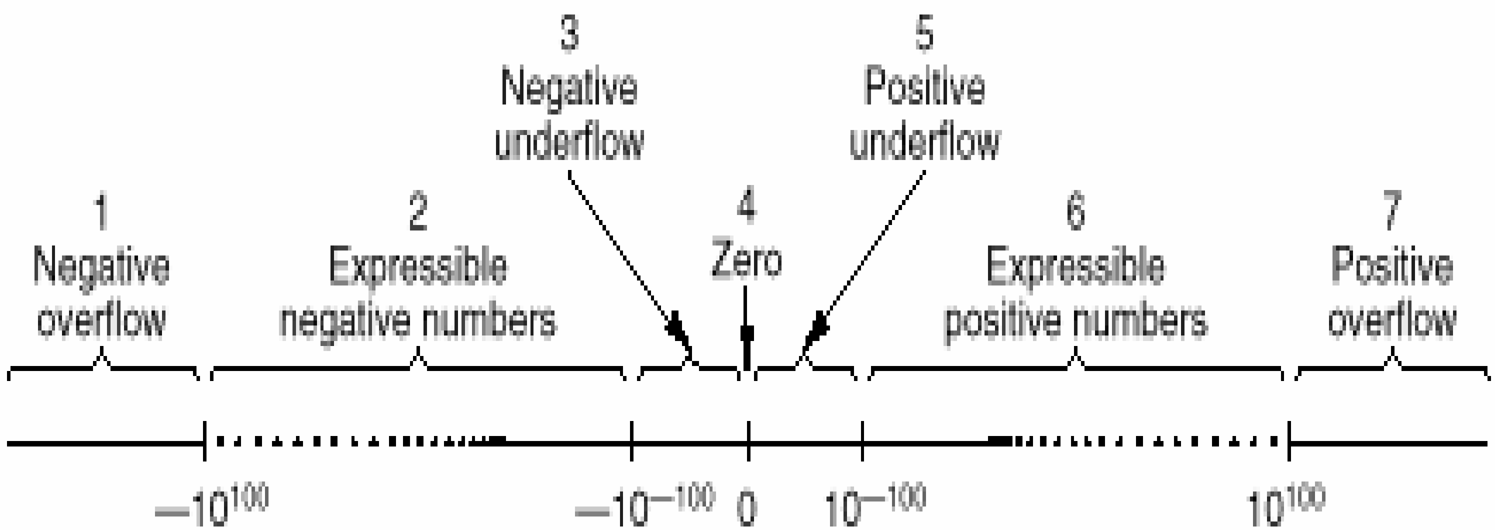


Figure B-1. The real number line can be divided into seven regions.

How to represent fractional values in binary

- We have only two symbols.
- We must represent
  - numbers: 0 and 1
  - sign
  - radix point
- How do we do that?
  - There have been a number of perfectly good solutions. Now that it is common to exchange data between systems, we must have one common method.

Floating point notation

- By example (using 8 bit word size):
  - 2.5 = 10.1 in binary
  - One way to represent the point is to put it in the same place all the time and then not represent it explicitly at all. To do that, we must have a standard representation for a value that puts the point in the same place every time.
  - $10.1 = 1.01 * 2^1$
  - Use 1 bit for sign, 2 bits for exponent, rest for value
  - sign = 0 (positive); exponent = 01; significand = 101
    - point assumed as in 1.01
  - Result= 00110100

Refinement

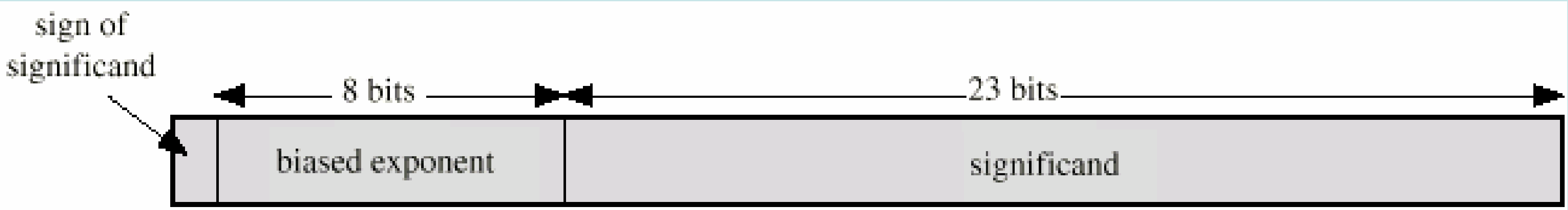
- Use 32 or 64 bits, not 8, to make more reasonable range of values
- Note that the leading 1 (as in $1.01 * 2^{1}$) is always there, so we don't need to waste one of our precious bits on it. Just assume it is always there.
- Use excess something notation for handling negative exponents.
- These ideas came from existing schemes developed for the PDP-11 and CDC 6600 computers.

Floating Point

Sign bit	Biased Exponent	Significand or Mantissa

- +/- .significand x $2^{exponent}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

Floating Point Examples



(a) Format

$$0.11010001 \quad 2^{10100} = 0\ 10010011\ 10100010000000000000000$$

$$-0.11010001 \quad 2^{10100} = 1\ 10010011\ 10100010000000000000000$$

$$0.11010001 \quad 2^{-10100} = 0\ 01101011\ 10100010000000000000000$$

$$-0.11010001 \quad 2^{-10100} = 1\ 01101011\ 10100010000000000000000$$

(b) Examples

Signs for Floating Point

- Mantissa is stored in 2s compliment
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

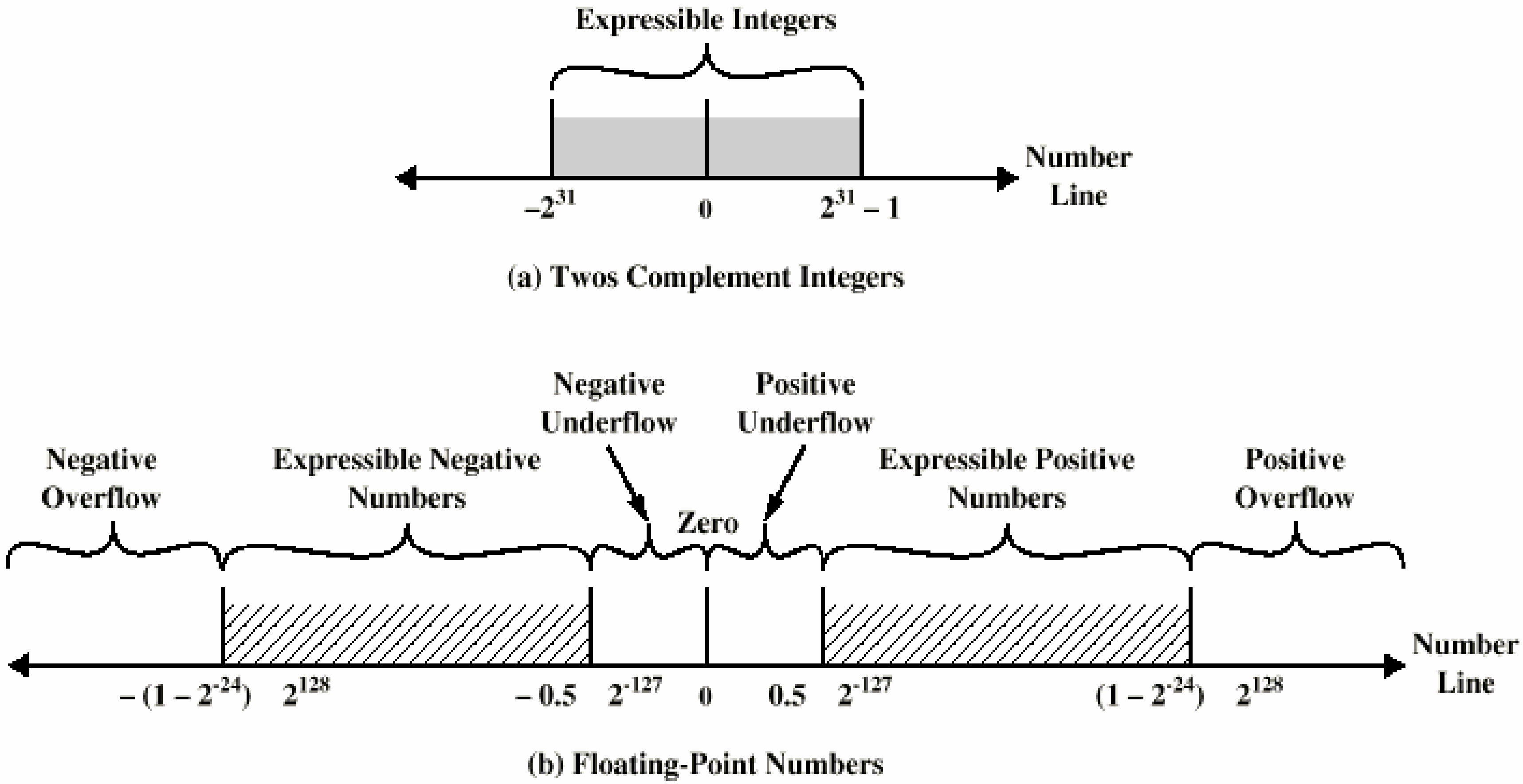
Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. 3.123×10^3)

FP Ranges

- For a 32 bit number
  - 8 bit exponent
  - +/- $2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

Expressible Numbers



IEEE Standard 754

- Provides a standard format for floating point numbers.
- Accepted by most computer manufacturers.
- Defines 3 formats:

  (a) Single precision (32 bits)

  (b) Double precision (64 bits)

  (c) Extended precision (80 bits)

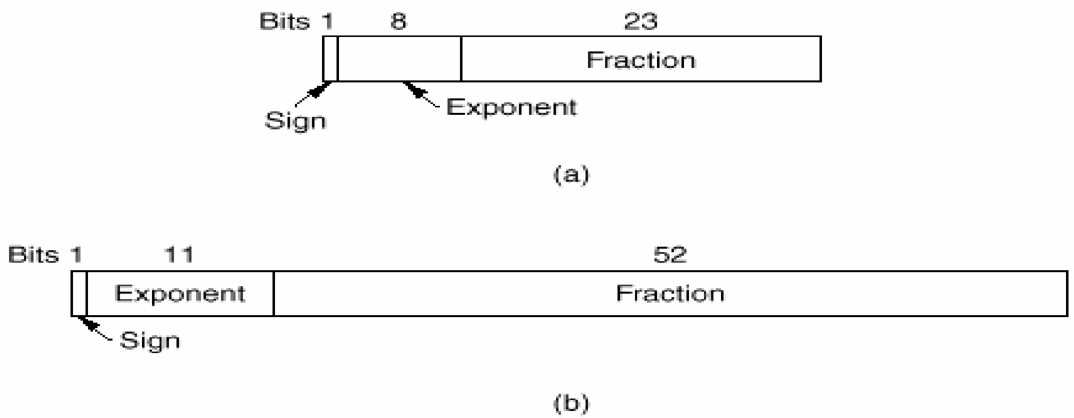


Figure B-4. IEEE floating-point formats. (a) Single precision. (b) Double precision.

Steps to IEEE format

- Convert 35.75, for example
  - **Convert the number to binary**
    - 100011.11
  - **Normalize**
    - 1.0001111×2^5
  - **Fit into the required format**
    - 5 + 127 = 132; hide the leading 1 in the fraction
    - 01000010000011110000000000000000
  - **Use Hexadecimal to make it easier to read**
    - 420F0000

IEEE 754 details

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	−126 to +127	−1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}

What's normalized?

- Normalized means represented in the normal, or standard, notation. Some numbers do not fit into that scheme and have a separate definition.
- Consider the smallest normalized value:
  - 1.000--000 x 2^{-126}
  - How would we represent half of that number?

1.000--000 x 2^{-127} *But we cannot fit 127 into the exponent field*

0.100 --000 x 2^{-126} would do it. *But we are stuck with that implied 1 before the implied point*

So, there are a lot of potentially useful values that don't fit into the scheme. The solution: special rules when the exponent has value 0 (which represents -126).

Denormalization

Denormalization: abandoning the “normal” scheme to exploit possibilities that would otherwise not be available.

Denormalized	±	0	Any nonzero bit pattern

No implied 1 before the implied point
Power of two multiplier is -127

Representing Zero

- How do you represent exactly 0 if there is an implied 1 in the number somewhere?
- A special case of denormalized numbers, when everything is zero, the value of the number is exactly 0.0

Infinity: A special representation consisting of an exponent with all 1s and a fraction of 0.

NaN (Not a Number): One more special case reserved for undefined results.

IEEE numerical types summary

Normalized	±	0 < Exp < Max	Any bit pattern
Denormalized	±	0	Any nonzero bit pattern
Zero	±	0	0
Infinity	±	1 1 1…1	0
Not a number	±	1 1 1…1	Any nonzero bit pattern

Sign bit