FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION department of radio electronics

Number representations

Microprocessor Techniques and Embedded Systems

Tomas Fryza

September 2018

Contents

Contents

Decimal and Hexadecimal (Hex) Systems

- **Decimal** system:
  - Base: 10
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Notation: without any prefix or sufix
- **Hexadecimal** system:
  - Base: 16
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Notation by subscript: $2BA6_{16}$, or prefixes: 0xD2, $25A4
  - Hexadecimal system represents values in more compact form than binary system.
- In the **binary** system there are only two symbols or possible digit values, 0 and 1.
  - Base: 2
  - Digits: 0, 1
  - Notation by subscript: 1100_2, or prefix 0b1010
- In binary system values can be considered as *fixed-point* or *floating-point*:
  - Fixed-point: integers (unsigned, signed), fractional representation.
  - Floating-point (IEEE 754): single, double, extended precision.
- Suppose the integer (8-bit) has the form $\mathbf{d} = (d_7, d_6, \ldots, d_1, d_0)$.
- The left-most bit, d_7, is the *most significant bit* (MSB).
- The right-most bit, d_0, is the *least significant bit* (LSB).

Transforming Numbers Between Systems

Table: Conversion table between dec, bin and hex.

Decimal	**Binary**	**Hexadecimal**	**Decimal**	**Binary**	**Hexadecimal**
0	0000	0	10	1010	a
1	0001	1	11	1011	b
2	0010	2	12	1100	c
3	0011	3	13	1101	d
4	0100	4	14	1110	e
5	0101	5	15	1111	f
6	0110	6	16	10000	10
7	0111	7	17	10001	11
8	1000	8	18	10010	12
9	1001	9	...	...	...

- We would like to transform number with base X into new one, that uses Y base.
- The method of *continuous dividing* and *multiplying* could be used:
  - Dividing numbers on the left side of radix point with Y
  - Multiplying numbers on the right side of radix point by Y

Decimal to Binary Transformation – Integer Numbers

Example

Transform 25 (decimal) into binary number.

Solution

Method of continuous dividing with binary base (i.e. 2).

$$\begin{array}{rcrcrlll}
25 & : & 2 & = & 12, & \textit{remaining } \mathbf{1} & 2^0 & LSB \\
12 & : & 2 & = & 6, & \textit{remaining } \mathbf{0} & 2^1 & \\
6 & : & 2 & = & 3, & \textit{remaining } \mathbf{0} & 2^2 & \\
3 & : & 2 & = & 1, & \textit{remaining } \mathbf{1} & 2^3 & \\
1 & : & 2 & = & 0, & \textit{remaining } \mathbf{1} & 2^4 & MSB
\end{array}$$

Result is $1\,1001_2$.

Decimal to Binary Transformation – Rational Numbers

Example

Transform 0.375 (decimal) into binary number.

Solution

Method of continuous multiplying numbers on the right side of radix point by binary base (i.e. 2).

$$\begin{array}{rcclll} 0.375 & \cdot & 2 & = & \mathbf{0}.75 & 2^{-1} & \textit{first digit after zero} \\ 0.75 & \cdot & 2 & = & \mathbf{1}.5 & 2^{-2} & \\ 0.5 & \cdot & 2 & = & \mathbf{1}.0 & 2^{-3} & \textit{last digit} \end{array}$$

Result is 0.011_2.

Decimal to Binary Transformation – Fast Method

Example

Transform 53 (decimal) into binary number using fast method.

Solution

$$
\begin{array}{rrll}
 & 53 & & \\
- & 32 & \text{i.e. } 2^5 & \\
\hline
 & 21 & & \\
- & 16 & \text{i.e. } 2^4 & \\
\hline
 & 5 & & \\
- & 4 & \text{i.e. } 2^2 & \\
\hline
 & 1 & & \\
- & 1 & \text{i.e. } 2^0 & \\
\hline
 & 0 & \rightarrow & \text{end of procedure}
\end{array}
$$

$2^5 + 2^4 + 2^2 + 2^0 = \mathbf{1} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$ *Result is* $11\,0101_2$.

Decimal to Hexadecimal Transformation

Example

Transform 423.34 (decimal) into hexadecimal number.

Solution

(a) Method of continuous dividing with hexadecimal base (i.e. 16).

$$\begin{array}{rcrcrll}
423 & : & 16 & = & 26, & \text{remaining } \mathbf{7} & 16^0 \quad LSB \\
26 & : & 16 & = & 1, & \text{remaining } \mathbf{10} & 16^1 \\
1 & : & 16 & = & 0, & \text{remaining } \mathbf{1} & 16^2 \quad MSB
\end{array}$$

Result is $1a7_{16}$.

Decimal to Hexadecimal Transformation

Solution

(b) Method of continuous multiplying numbers on the right side of radix point by hexadecimal base (i.e. 16).

$$
\begin{array}{lcccrll}
0.34 & \cdot & 16 & = & \mathbf{5}.44 & 16^{-1} & \text{first digit after zero} \\
0.44 & \cdot & 16 & = & \mathbf{7}.04 & 16^{-2} & \\
0.04 & \cdot & 16 & = & \mathbf{0}.64 & 16^{-3} & \\
0.64 & \cdot & 16 & = & \mathbf{10}.24 & 16^{-4} & \\
0.24 & \cdot & 16 & = & \mathbf{3}.84 & 16^{-5} & \\
\ldots & & & & & &
\end{array}
$$

procedure could continues

Result is approximately $1a7.570a3_{16}$.

Hexadecimal to Binary Transformation – Fast Method

Example

Transform 6f.a (hexadecimal) into binary number.

Solution

$6f.a_{16} = 0110 \quad 1111\ .\ 1010_2$
Result is $110\,1111.101_2$.

Example

Transform 11.0000 1100 1 (binary) into hexadecimal number.

Solution

$11.0000\,1100\,1_2 = 0011\ .\ 0000 \quad 1100 \quad 1000_2 = 3.0c8_{16}$
Result is $3.0c8_{16}$.

Contents

Integer Numbers

- An *unsigned* 32-bit integer has the decimal value

$$v = \sum_{n=0}^{31} d_n 2^n \tag{1}$$

and is in the range $[0, 2^{32} - 1]$.

- *Signed* integers are in the **two's complement** format with the MSB as the sign bit. A signed integer has the decimal value

$$v = -d_{31} 2^{31} + \sum_{n=0}^{30} d_n 2^n \tag{2}$$

which is in the range $[-2^{31}, 2^{31} - 1]$.

- The negative numbers are the two's complement of the positive numbers, and vice versa.

Example

Set decimal ranges for 8-bit unsigned and signed values.

Signed Integer Numbers

- The *n*-bit unsigned number represent a modulo (mod) 2^n system. If 1 is added to the largest number, the operation wraps around to give 0 as the answer. Therefore the finite bit system could be graphically demonstrates as a wheel.
- In signed numbers, the right half of wheel represents the positive numbers and the left half the negative numbers. This representation is the two's complement system.

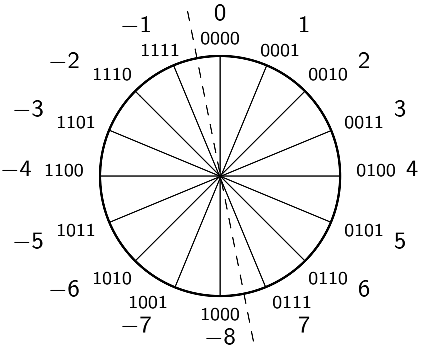

Figure: Number wheel for signed integers (4-bit numbers example).

Two's Complement Format

- Algorithm for two's complement transformation:
  (a) Negation of all bits (i.e. one's complement format).
  (b) Add one.

Example

$-2 + 3 = ?$

Solution

(a) *Use positive value $+2$ and invert all bits: 0010 $\rightarrow$ 1101*

(b) *Add one in binary: 1101 $+$ 0001 $=$ 1110 and get -2*

$$
\begin{array}{lrr}
 & 1110 & -2_{10} \\
+ & 0011 & 3_{10} \\
\hline
 & 1\ 0001 & 1_{10}
\end{array}
$$

Fractional Fixed-Point Representation

- Fractional fixed-point numbers (n-bit) has values $[-1, 1-2^{-(n-1)}]$
- Conversion of binary fraction ($n = 32$) to a decimal fraction:

$$v = -d_0 2^0 + \sum_{n=1}^{32} d_n 2^{-n} \tag{3}$$

- The sign bit has a weight of negative 1 and the weights of the other bits are positive powers of $1/2$

Example

Find out the binary representations for minimal and maximal 8-bit fractional fixed-point values.

Contents

Floating-Point Representation

- IEEE floating-point numbers can be normal numbers, denormalized (or subnormal) numbers, NaNs (not a number), and infinity numbers
- According to IEEE 754 standard, there are three floating-point formats:
  - 32-bit single precision
  - 64-bit double precision
  - 80-bit extended precision
- Every format contains sign bit, biased exponent and mantissa (or fractional part)

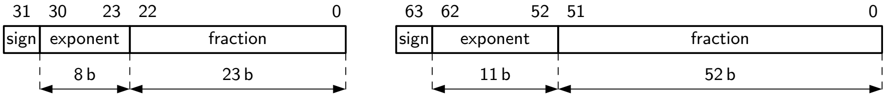

Figure: IEEE single and double precision floating-point formats

Floating-point single-precision format

- Sign bit $= 1 \Leftrightarrow$ negative values:
  - signed values are no more express by the two'complement like fixed-point formats; opposite numbers differ by sign bit only
- Fraction:
  - represents the most important bits
  - binary value is normalized $\Rightarrow$ radix point id shifted after the first "one"
- Exponent:
  - could express both positive and negative offset
  - the bias is always added to the specific offset $2^{nexp-1} - 1$ where $nexp$ is number of bits for single-precision exponent. Therefore the stored exponent value is always positive. Bias for single-precision is 127_{10}

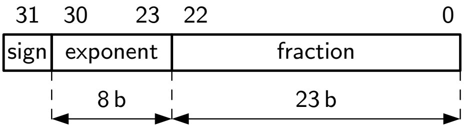

Figure: Structure of floating-point single-precision

- Single-precision range is approx. $\pm 3{,}4 \cdot 10^{38}$

Single Precision Floating-Point Representation

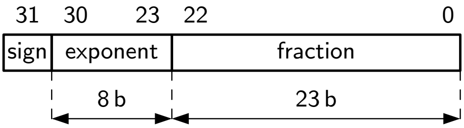

Figure: IEEE single precision floating-point format.

Example

Convert $+14.625$ to single precision floating-point number.

Solution

(1) *14.625*$_{10}$ $\rightarrow$ *1110.101*$_2$,

(2) *Normalized form: 1110.101* $\rightarrow$ *1.***110101**$\cdot 2^3$,

(3) *23-bit fraction:* **1101 01***00 0000 0000 0000 000*

(4) *Exponent:* $3 + bias = 3 + 127 = 130 \rightarrow$ *1000 0010 (*$bias = 2^{nb_{exp}-1} - 1$*, where* nb_{exp} *is a number of bits for exponent part, i.e.* $bias = 2^{8-1} - 1 = 127$*)*

(5) *Positive number* $\rightarrow sign = 0$

Result *0 1000 0010 1101 0100 0000 0000 0000 000*

Floating-point double-precision format

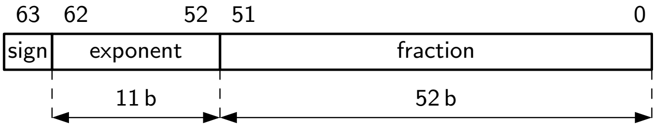

Figure: Structure of floating-point double-precision

- Double-precision format has three parts:
  - sign bit (63)
  - 11-bit exponent (62:52)
  - 52-bit fraction (51:0)
- Real value in decimal system is express by equation $(-1)^{sign} \cdot 2^{exp-bias} \cdot 1, fraction$ where $bias = 2^{nexp-1} - 1$, i.e. for double-precision $bias$=1 023
- Double-precision range is approx. $\pm 1{,}8 \cdot 10^{308}$

Double Precision Floating-Point Representation

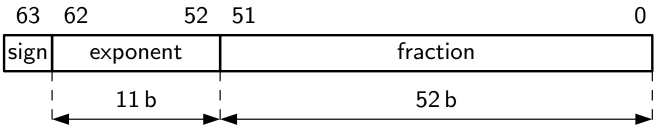

Figure: IEEE double precision floating-point format

Example

According to stand. IEEE 754 convert a floating-point number to the decimal form.

- 1
- 100 0001 1001
- 1101 0110 1111 0011 0100 0101 0100 0000 0000 0000 0000 0000 0000

Solution

(1) *Sign bit* $= 1 \Leftrightarrow$ *negative number*

(2) *Exponent: 100 0001 1001−bias=1 049−1 023=26* $(bias = 2^{nb_{exp}-1} - 1 = 2^{11-1} - 1 = 1023)$

(3) *Normalized form: 1.***1101 0110 1111 0011 0100 0101 0100** $\ldots \cdot 2^{exponent}$

Result *−123 456 789*

Special values of floating-point format

Table: Special values of floating-point format

Value	Sign	Exponent	Fraction
Zero	0	0	0
Zero	1	0	0
Infinity	1 or 0	0b1111_1111	0
NaN (not a number)	1 or 0	0b1111_1111	$\neq 0$

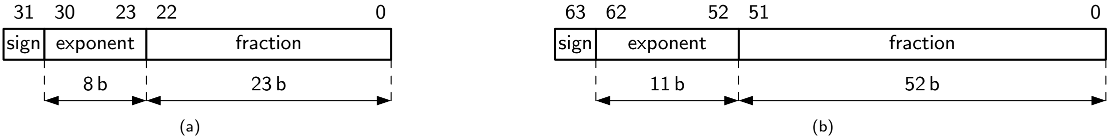

Figure: Floating-point format structure: (a) single-, (b) double-precision