Software Systems-1

Floating Point Arithmetic

Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

1 Introduction

Floating-point representation is an alternative technique based on scientific notation. Scientific notation is a way of expressing numbers that are too large or too small to be conveniently written in decimal form. It may be referred to as scientific form or standard index form. This base ten notation is commonly used by scientists, mathematicians, and engineers, in part because it can simplify certain arithmetic operations. In scientific notation, all numbers are written in the form of

$$m \times 10^n$$

where the exponent n is an integer, and the coefficient m is any real number. The real number m is called the ***significand*** or ***mantissa***. Any given real number can be written in the form of $m \times 10^n$ in many ways:

For example, 350 can be written as 3.5×10^2 or 35×10^1 or 350×10^0.

n normalized scientific notation, the exponent n is chosen so that the absolute value of m remains at least one but less than ten ($1 \le |m| < 10$). Thus 350 is written as 3.5×10^2. Few steps are taken to write a scientific notation as follow:

- ❖ Decide where the decimal must end up so that the one number is to its left.
- ❖ Count how many places you bounce the decimal point.
- ❖ Re-write in the form of $m \times 10^n$.

Ex: Write 0.000811 in scientific notation.

1. Move the decimal to get a number between 1 and 10 (8.11).
2. Set up scientific notation $8.11 \times 10^?$
3. The ***decimal*** needs to move ***Right*** to change 000811 to 8.11, so the exponent will be ***negative***.
4. The decimal needs to move ***4*** places.
5. So 0.000811 written in scientific notation is 8.11×10^{-4}.

2 Floating-point basics

When dealing with very large or very small numbers, a more flexible technique, known as **floating point representation**, can be employed. This allows the way that the bits are allocated to vary so that both very large and very small numbers can be represented.

Ex: Convert $5.5_{(10)}$ to 8-bit representation of

Answer: $5.5_{(10)} = 101.1_{(2)}$

- Converts a to binary scientific notation:

$$\underbrace{101.1_{(2)} = 1.011}_{\text{Mantissa}} \times 2^{\overset{\text{Exponent}}{2}}$$

- Applying technique for mapping that into a set of bits. In this example we'll use **8 bits** to store such a number.

1bit	4bit	3bit
Sign	Exponent+7	Mantissa

We use the first bit to represent the sign (1 for negative, 0 for positive).

- For the exponent bits

Instead of using a separate sign bit for the exponent, the standard uses a biased representation. The value of the bias is 7. Thus an exponent 0 means that –7 is stored in the exponent field. $(2^{n-1}-1)$ = Biased value, n=no of bits. $(2^{4-1}-1) = 7$.

0 ➔ 15 (biased)
-7➔ 8 (unbiased)

$7 + 2$ to $= 9_{(10)} = 1001_{(2)}$

Sign	Exponent+7	Mantissa
0	1001	011

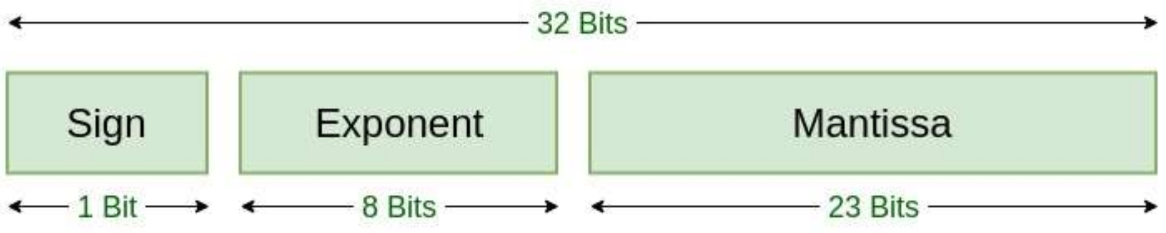
Note that we omit the integer part of the mantissa: Since the mantissa must have exactly one nonzero bit to the left of its decimal point, and the only nonzero bit is 1, we know that the bit to the left of the decimal point must be a 1. There's no point in wasting space in inserting this 1 into our bit pattern, so we include only the bits of the mantissa to the right of the decimal point.

2.1 IEEE 754 Floating binary point

Floating point numbers are an important data type in computation which is used extensively. Yet, many users do not know the standard which is used in almost all computer hardware to store and process these. The Institute of Electrical and Electronic Engineers IEEE in 1985 and augmented in 2008 was evolved to represent floating point numbers and process them. This standard is now used by all computer manufacturers while designing floating point arithmetic units so that programs are portable among computers.

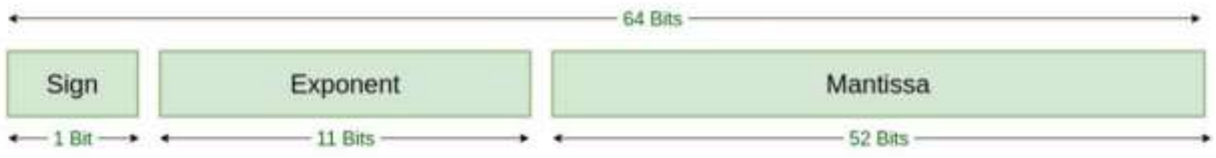
Normalized floating point representation is used for real numbers in computers. In this representation, 32 bits are divided into two parts: a part called the mantissa with its sign and the other called the exponent with its sign.

The number of bits required to represent ***seven decimal digits is approximately 23***. The remaining 8 bits are allocated for the exponent, the exponent will range from ***–127 to +127***.



Single precision floating point representation (32-bit)

1. Sign, for which 1 bit is allocated.
2. Mantissa (called *significand* in the standard) is allocated 23 bits.
3. Exponent is allocated 8 bits. As both positive and negative numbers are required for



Double precision floating point representation (64-bit)

2.2 Decimal to IEEE 754 Floating binary point

Ex1: Convert the value 19.59375 into IEEE 754 standard 32-bit Floating binary point

- **Step1:** Determine the sign bit (0 if positive, 1 if negative).
- **Step2:** Convert to pure binary

	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125
19.59375_{10} =	1	0	0	1	1	1	0	0	1	1

19/2 = 9	Remainder 1 ↑	0.59375 * 2 = 1.1875	1 ↓
9/2 = 4	Remainder 1	0.1875 * 2 = 0.375	0
4/2 = 2	Remainder 0	0.375 * 2 = 0.75	0
2/2 = 1	Remainder 0	0.75 * 2 = 1.5	1
1/2 = 0	Remainder 1	0.5 * 2 = 1	1

- **Step3:** Normalize to determine the mantissa and the ***unbiased exponent*** (place the binary point after leftmost 1)

  1 0 0 1 1. 1 0 0 1 1 = 1.001110011 x 2^4

- **Step4:** Determine the ***biased exponent.***

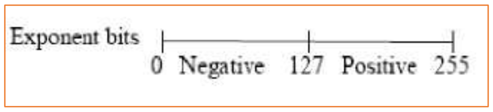
  $4+127 = 131_{10} = 10000011_2$

- **Step5:** Removing the Leading 1 from the mantissa (Remove the leftmost1)

  1. 001 1100 1100 0000 0000 0000 = 001 1100 1100 0000 0000 0000

The value of the bias is $(2^{8-1}-1) = 127$. Thus an exponent 0 means that –127 is stored in the exponent field.

The range 0 to 255 of an 8-bit number is divided into two parts 0 to 127 and 128 to 255. Thus the range of exponents is –127 to +128. Given an exponent string *exp.*, the value of the exponent will be taken as (exp ±127).



S	Exponent								Mantissa																							
0	1	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ex2: Convert the value -123.3 into IEEE 754 standard 32-bit Floating binary point

- **Step1:** Sign bit =1.
- **Step2:** Convert to pure binary

$123.3_{10} = 1111011.01001\,1001\,1001\ldots_2$

123/2 =	61	Remainder 1		0.3 * 2 = 0.6	0
61/2 =	30	Remainder 1		0.6 * 2 = 1.2	1
30/2 =	15	Remainder 0		0.2 * 2 = 0.4	0
15/2 =	7	Remainder 1		0.4 * 2 = 0.8	0
15/2 =	7	Remainder 1		0.8 * 2 = 1.6	1
3/2 =	1	Remainder 1		0.6 * 2 = 1.2	1
1/2 =	0	Remainder 1		0.2 * 2 = 0.4	0

- **Step3:** Normalize to determine the mantissa and the ***unbiased exponent*** (place the binary point after leftmost 1)

1 1 1 1 0 1 1**.** 0 1 0 0 1 1 0 0 1 $\underline{1\ 0\ 0\ 1}$ = 1**.** 1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 x 2^6

- **Step4:** Determine the ***biased exponent.***

$6+127 = 133_{10} = 10000101_2$

- **Step5:** Removing the Leading 1 from the mantissa (Remove the leftmost1)

1**.** 1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 = 1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1

S	Exponent								Mantissa																						
1	1	0	0	0	0	1	0	1	1	1	1	0	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1				

2.3 IEEE 754 Floating binary single precession point to Decimal

Ex3: Convert 01000010011010000000000000000000

- **Step1:** Sign bit = 0 (Positive number).
- Determine the exponent in Decimal: **10000100 = 132$_{10}$**
- Remove the exponent bias, 132-127=5
- Convert the mantissa to binary = 0.5+0.25+0.0625 = 0.8125$_{10}$

0.5	0.25	0.125	0.0625	0.03125	0.015625
•1	1	0	1	0	0

$$= \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = \frac{8+4+1}{16} = \frac{13}{16}$$

- Step5: add 1 to mantissa and include the sign = 1.8125$_{10}$
- Step6: Calculate the final result: 1.8125$_{10}$ x 2^5 = 58

In other words, the final equation that was used in the above example is:

$$(-1)^S \times (1+ \text{Mantissa})_{10} \times 2^{exp-127}$$

Ex4: Convert 11000100100011000000000000000000

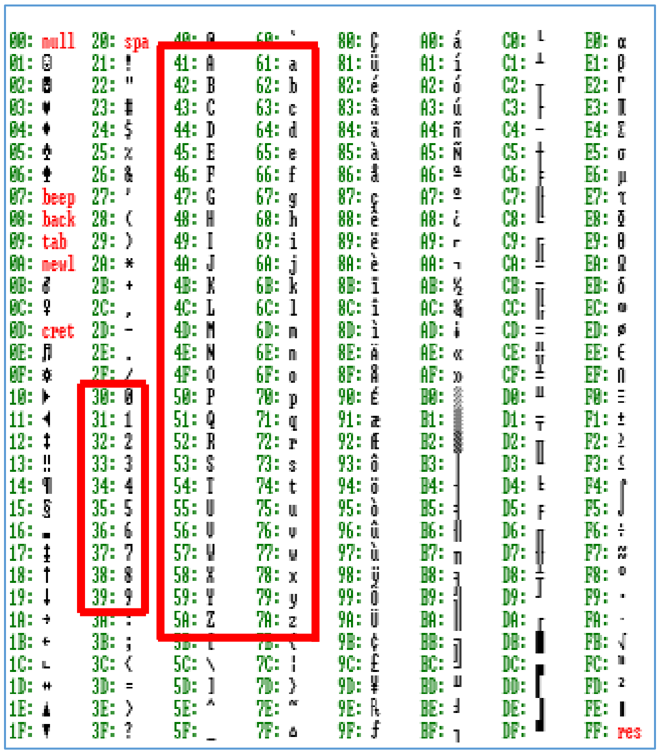
- **Step1:** Sign bit = 1 (Negative number).
- Determine the exponent in Decimal: **10001001 = 137$_{10}$**
- Remove the exponent bias, 137-127= 10
- Convert the mantissa to binary = 0.0625 + 0.03125 =0.09375$_{10}$

0.5	0.25	0.125	0.0625	0.03125	0.015625
•0	0	0	1	1	0

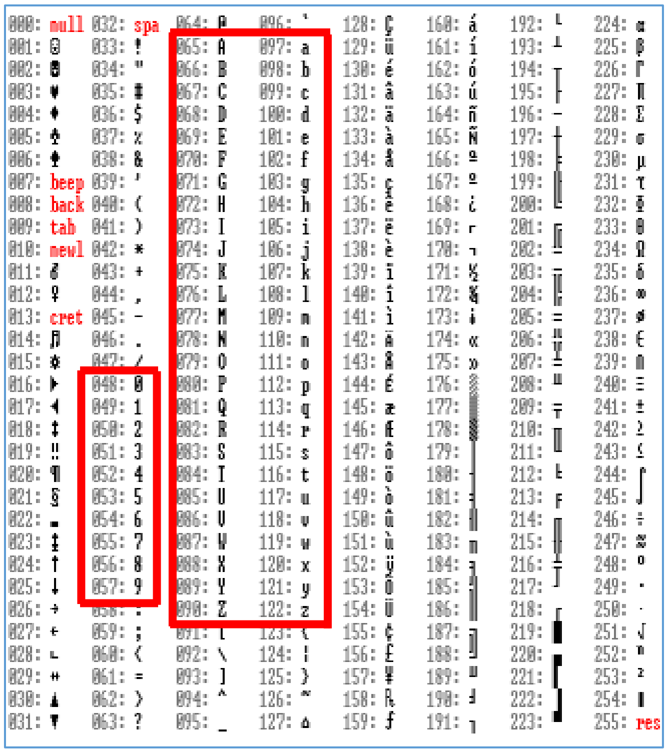
- Step5: add 1 to mantissa and include the sign = -1.09375$_{10}$
- Step6: Calculate the final result: -1.09375$_{10}$ x 2^{10} = -1120

3 CHARACTER REPRESENTATION

The computer convert data to character which is a form that is meaningful to humans. The American Standard Code for Information Interchange (***ASCII***) uses 8-bits to represent 256 characters include Latin alphabet, control characters, special characters, numerical characters. ASCII codes are restricted in their abilities to provide data representation for the non-Latin alphabets used by the majority of the world's population.



00:	null	20:	spa	40:	@	60:	`	80:	Ç	A0:	á	C0:	└	E0:	α
01:	☺	21:	!	41:	A	61:	a	81:	ü	A1:	í	C1:	┴	E1:	ß
02:	☻	22:	"	42:	B	62:	b	82:	é	A2:	ó	C2:	┬	E2:	Γ
03:	♥	23:	#	43:	C	63:	c	83:	â	A3:	ú	C3:	├	E3:	π
04:	♦	24:	$	44:	D	64:	d	84:	ä	A4:	ñ	C4:	─	E4:	Σ
05:	♣	25:	%	45:	E	65:	e	85:	à	A5:	Ñ	C5:	┼	E5:	σ
06:	♠	26:	&	46:	F	66:	f	86:	å	A6:	ª	C6:	╞	E6:	µ
07:	beep	27:	'	47:	G	67:	g	87:	ç	A7:	º	C7:	╟	E7:	τ
08:	back	28:	(	48:	H	68:	h	88:	ê	A8:	¿	C8:	╚	E8:	Φ
09:	tab	29:	)	49:	I	69:	i	89:	ë	A9:	⌐	C9:	╔	E9:	Θ
0A:	newl	2A:	*	4A:	J	6A:	j	8A:	è	AA:	¬	CA:	╩	EA:	Ω
0B:	♂	2B:	+	4B:	K	6B:	k	8B:	ï	AB:	½	CB:	╦	EB:	δ
0C:	♀	2C:	,	4C:	L	6C:	l	8C:	î	AC:	¼	CC:	╠	EC:	∞
0D:	cret	2D:	-	4D:	M	6D:	m	8D:	ì	AD:	¡	CD:	═	ED:	φ
0E:	♫	2E:	.	4E:	N	6E:	n	8E:	Ä	AE:	«	CE:	╬	EE:	ε
0F:	☼	2F:	/	4F:	O	6F:	o	8F:	Å	AF:	»	CF:	╧	EF:	∩
10:	►	30:	0	50:	P	70:	p	90:	É	B0:	░	D0:	╨	F0:	≡
11:	◄	31:	1	51:	Q	71:	q	91:	æ	B1:	▒	D1:	╤	F1:	±
12:	↕	32:	2	52:	R	72:	r	92:	Æ	B2:	▓	D2:	╥	F2:	≥
13:	‼	33:	3	53:	S	73:	s	93:	ô	B3:	│	D3:	╙	F3:	≤
14:	¶	34:	4	54:	T	74:	t	94:	ö	B4:	┤	D4:	╘	F4:	⌠
15:	§	35:	5	55:	U	75:	u	95:	ò	B5:	╡	D5:	╒	F5:	⌡
16:	▬	36:	6	56:	V	76:	v	96:	û	B6:	╢	D6:	╓	F6:	÷
17:	↨	37:	7	57:	W	77:	w	97:	ù	B7:	╖	D7:	╫	F7:	≈
18:	↑	38:	8	58:	X	78:	x	98:	ÿ	B8:	╕	D8:	╪	F8:	°
19:	↓	39:	9	59:	Y	79:	y	99:	Ö	B9:	╣	D9:	┘	F9:	∙
1A:	→	3A:	:	5A:	Z	7A:	z	9A:	Ü	BA:	║	DA:	┌	FA:	·
1B:	←	3B:	;	5B:	[	7B:	{	9B:	¢	BB:	╗	DB:	█	FB:	√
1C:	∟	3C:	<	5C:	\	7C:	|	9C:	£	BC:	╝	DC:	▄	FC:	ⁿ
1D:	↔	3D:	=	5D:	]	7D:	}	9D:	¥	BD:	╜	DD:	▌	FD:	²
1E:	▲	3E:	>	5E:	^	7E:	~	9E:	₧	BE:	╛	DE:	▐	FE:	■
1F:	▼	3F:	?	5F:	_	7F:	⌂	9F:	ƒ	BF:	┐	DF:	▀	FF:	res



000:	null	032:	spa	064:	@	096:	`	128:	Ç	160:	á	192:	└	224:	α
001:	☺	033:	!	065:	A	097:	a	129:	ü	161:	í	193:	┴	225:	ß
002:	☻	034:	"	066:	B	098:	b	130:	é	162:	ó	194:	┬	226:	Γ
003:	♥	035:	#	067:	C	099:	c	131:	â	163:	ú	195:	├	227:	π
004:	♦	036:	$	068:	D	100:	d	132:	ä	164:	ñ	196:	─	228:	Σ
005:	♣	037:	%	069:	E	101:	e	133:	à	165:	Ñ	197:	┼	229:	σ
006:	♠	038:	&	070:	F	102:	f	134:	å	166:	ª	198:	╞	230:	µ
007:	beep	039:	'	071:	G	103:	g	135:	ç	167:	º	199:	╟	231:	τ
008:	back	040:	(	072:	H	104:	h	136:	ê	168:	¿	200:	╚	232:	Φ
009:	tab	041:	)	073:	I	105:	i	137:	ë	169:	⌐	201:	╔	233:	Θ
010:	newl	042:	*	074:	J	106:	j	138:	è	170:	¬	202:	╩	234:	Ω
011:	♂	043:	+	075:	K	107:	k	139:	ï	171:	½	203:	╦	235:	δ
012:	♀	044:	,	076:	L	108:	l	140:	î	172:	¼	204:	╠	236:	∞
013:	cret	045:	-	077:	M	109:	m	141:	ì	173:	¡	205:	═	237:	φ
014:	♫	046:	.	078:	N	110:	n	142:	Ä	174:	«	206:	╬	238:	ε
015:	☼	047:	/	079:	O	111:	o	143:	Å	175:	»	207:	╧	239:	∩
016:	►	048:	0	080:	P	112:	p	144:	É	176:	░	208:	╨	240:	≡
017:	◄	049:	1	081:	Q	113:	q	145:	æ	177:	▒	209:	╤	241:	±
018:	↕	050:	2	082:	R	114:	r	146:	Æ	178:	▓	210:	╥	242:	≥
019:	‼	051:	3	083:	S	115:	s	147:	ô	179:	│	211:	╙	243:	≤
020:	¶	052:	4	084:	T	116:	t	148:	ö	180:	┤	212:	╘	244:	⌠
021:	§	053:	5	085:	U	117:	u	149:	ò	181:	╡	213:	╒	245:	⌡
022:	▬	054:	6	086:	V	118:	v	150:	û	182:	╢	214:	╓	246:	÷
023:	↨	055:	7	087:	W	119:	w	151:	ù	183:	╖	215:	╫	247:	≈
024:	↑	056:	8	088:	X	120:	x	152:	ÿ	184:	╕	216:	╪	248:	°
025:	↓	057:	9	089:	Y	121:	y	153:	Ö	185:	╣	217:	┘	249:	∙
026:	→	058:	:	090:	Z	122:	z	154:	Ü	186:	║	218:	┌	250:	·
027:	←	059:	;	091:	[	123:	{	155:	¢	187:	╗	219:	█	251:	√
028:	∟	060:	<	092:	\	124:	|	156:	£	188:	╝	220:	▄	252:	ⁿ
029:	↔	061:	=	093:	]	125:	}	157:	¥	189:	╜	221:	▌	253:	²
030:	▲	062:	>	094:	^	126:	~	158:	₧	190:	╛	222:	▐	254:	■
031:	▼	063:	?	095:	_	127:	⌂	159:	ƒ	191:	┐	223:	▀	255:	res