

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 2

Representation of information

Computer Structure
Bachelor in Computer Science and Engineering



Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

1. Alphanumeric

1. Characters
2. Strings

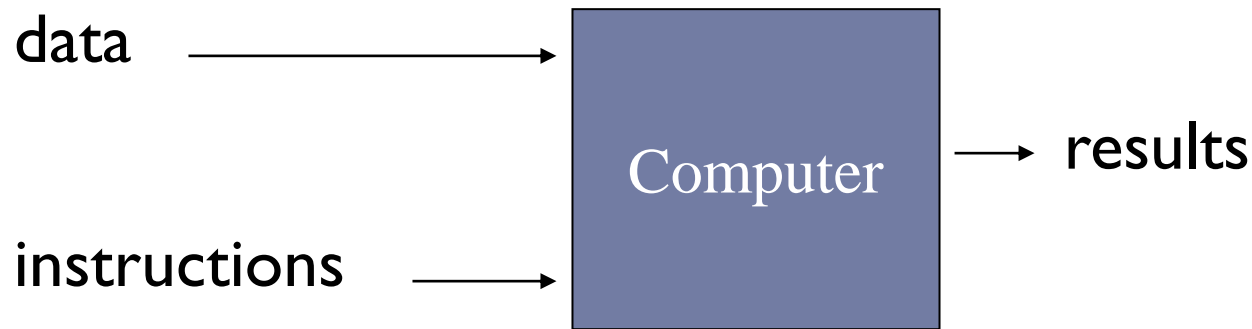
2. Numerical

1. Natural and integer
2. Fixed point
3. Floating point (IEEE 754 standard)

Introduction

Computer

- ▶ A computer is a machine designed to process data.

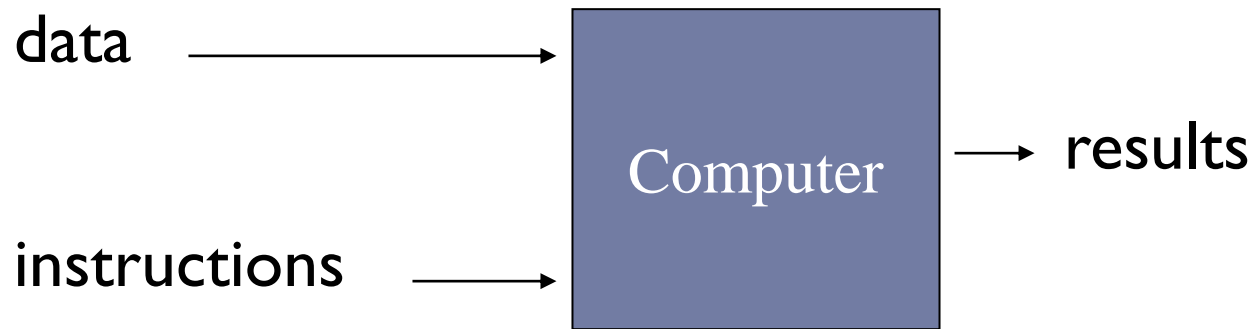


- ▶ Instructions are applied and results are obtained.

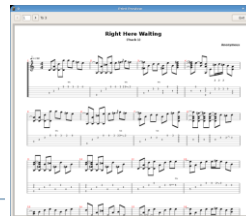
Introduction

Computer

- ▶ A computer is a machine designed to process data.



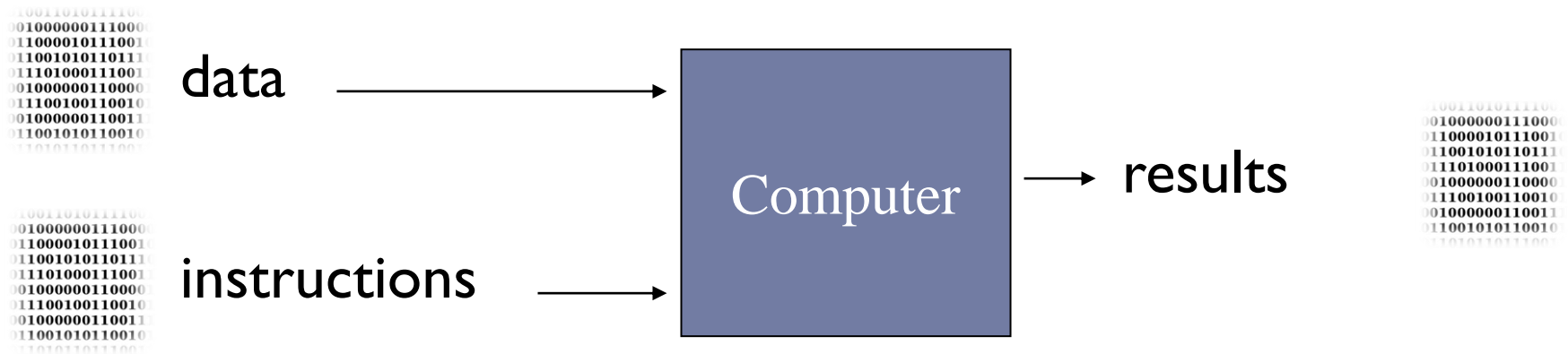
- ▶ Instructions are applied and results are obtained.
- ▶ The data/information can be of **different types**.



Introduction

Computer

- ▶ A computer is a machine designed to process data.

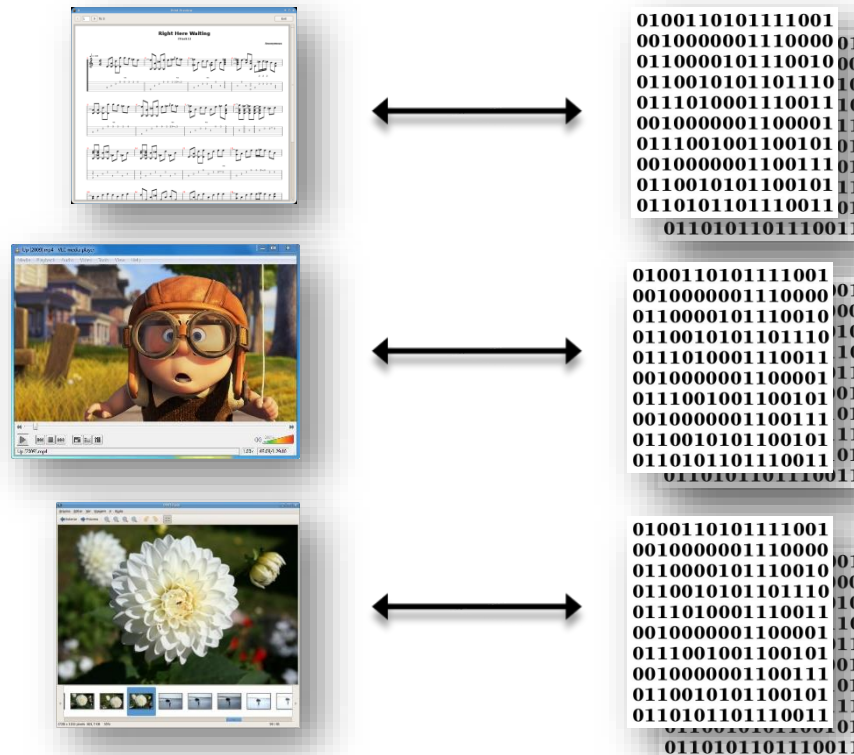


- ▶ Instructions are applied and results are obtained.
- ▶ The data/information can be of **different types**.
- ▶ A computer uses only one representation: **binary**.

Introduction

Information representation

- ▶ The use of a **representation** allows the transformation of different types of information into binary (and vice versa).

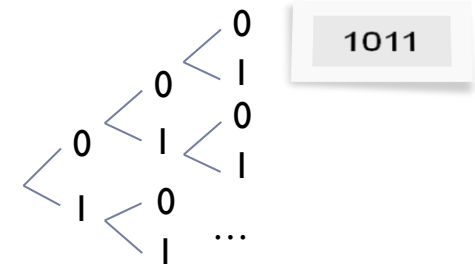


Introduction

Characteristics of the information representation

- ▶ A computer handles a finite set of values

- ▶ Binary type (two states)
- ▶ Finite (bounded representation)
 - ▶ Number of bits of the computer word (32/64) or bit (1), nibble (4), byte (8), half w., double w., ...
 - ▶ With n bits, 2^n different values can be encoded



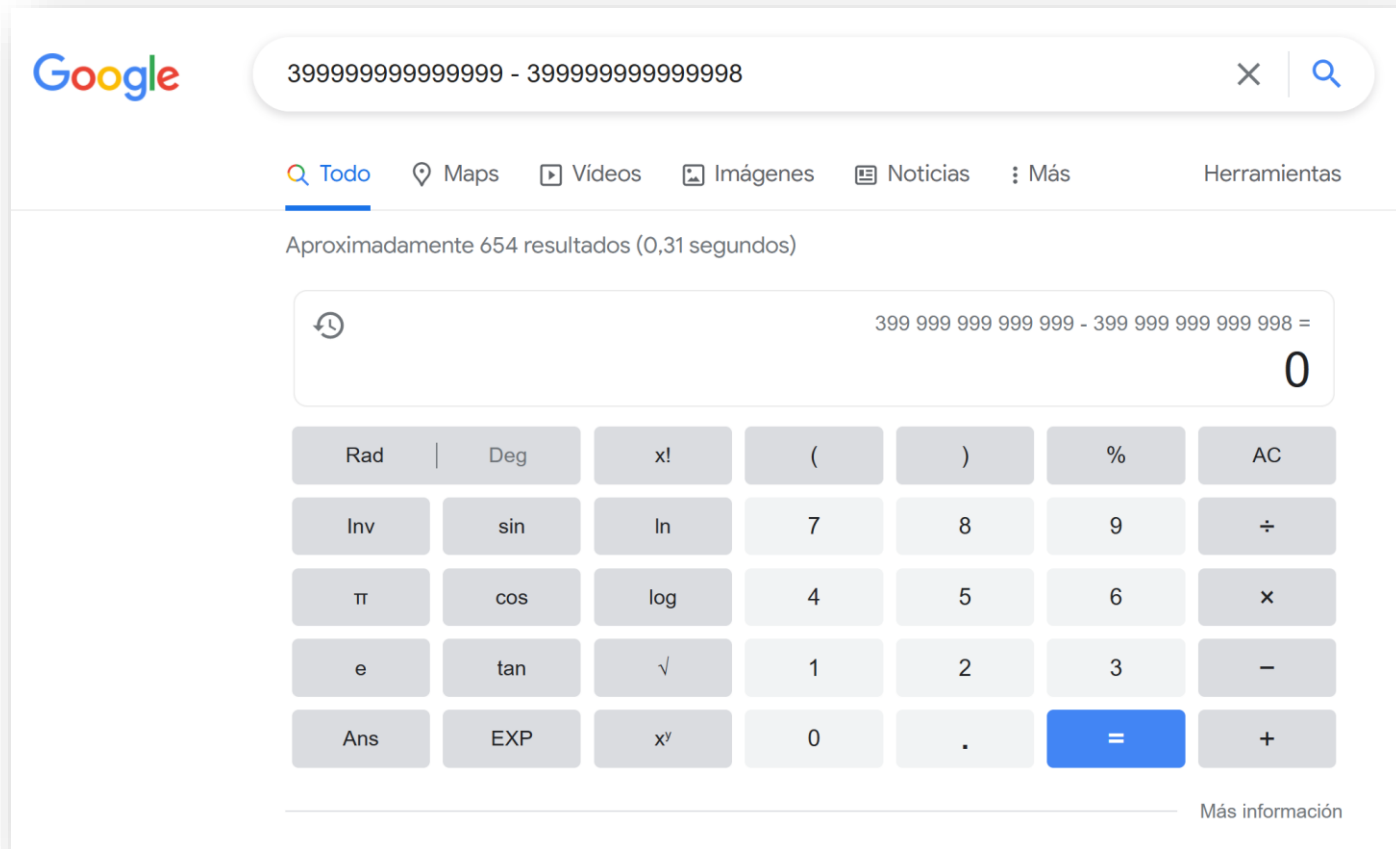
- ▶ There are some types of information that are infinite

- ▶ Impossible to represent all values of natural numbers, real numbers, etc.



- ▶ The chosen representation has limitations.

Example 1: the Google calculator with 15 digits...



<http://www.20minutos.es/noticia/415383/0/google/restar/error/>

Example 2: color depth...

1 bit	2 colors
4 bits	16 colors
8 bits	256 colors



<http://platea.pntic.mec.es/~lgonzale/tic/imagen/conceptos.html>

Example 2: color depth...

1 bit	2 colors
4 bits	16 colors
8 bits	256 colors



<http://platea.pntic.mec.es/~lgonzale/tic/imagen/conceptos.html>

Example 2: color depth...

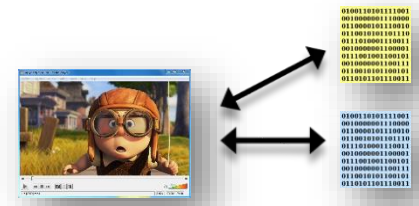
1 bit	2 colors
4 bits	16 colors
8 bits	256 colors



<http://platea.pntic.mec.es/~lgonzale/tic/imagen/conceptos.html>

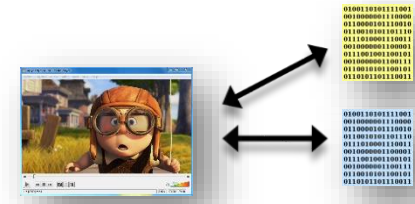
We need...

- To know possible representations:



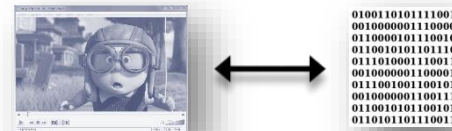
We need...

- ▶ To know **possible representations**:



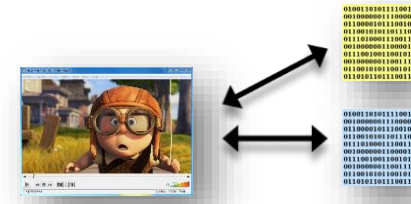
- ▶ To know the **characteristics** of these representations:

- ▶ Limitations



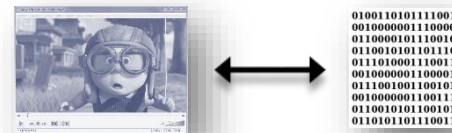
We need...

- ▶ To know **possible representations**:

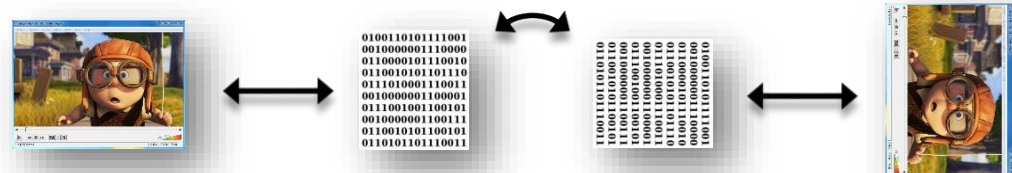


- ▶ To know the **characteristics** of these representations:

- ▶ Limitations



- ▶ To know **how work** with the selected representation:



Contents

1. Introduction

1. Motivation and goals

2. **Positional (numeral) systems**

2. Representations

1. Alphanumeric

1. Characters

2. Strings

2. Numerical

1. Natural and integer

2. Fixed point

3. Floating point (IEEE 754 standard)

Positional representation systems

- ▶ A number is defined by a **ordered list of digits**, each of which is **affected** by a **scaling factor** that **depends** on the **position** it occupies in the list.

- ▶ Given a numbering base b ,
a number X is defined as the list of digits:

$$X = (\dots x_2 x_1 x_0, x_{-1} x_{-2} \dots)_b \quad \text{Con } 0 \leq x_i < b$$

with a list of associated weights:

$$P = (\dots b^2 b^1 b^0 \quad b^{-1} b^{-2} \dots)_b$$

- ▶ Its value is:

$$V(X) = \sum_{i=-\infty}^{+\infty} b^i \cdot x_i = \dots \underbrace{b^2 \cdot x_2}_{\text{bag}} + \underbrace{b^1 \cdot x_1}_{\text{bag}} + \underbrace{b^0 \cdot x_0}_{\text{bag}} + \underbrace{b^{-1} \cdot x_{-1}}_{\text{bag}} + \underbrace{b^{-2} \cdot x_{-2}}_{\text{bag}} \dots$$

Positional representation systems

- ▶ Decimal

$$X = \quad 9 \quad 7 \quad 3 \quad 1 \\ \quad \dots 10^3 10^2 10^1 10^0$$

- ▶ Binary

$$X = \quad 0 \quad 1 \quad 0 \quad 1 \\ \quad \dots 2^3 2^2 2^1 2^0$$

- ▶ Hexadecimal

$$X = \quad 1 \quad F \quad A \quad 8 \\ \quad \dots 16^3 16^2 16^1 16^0$$

Positional representation systems

▶ Decimal

$$X = \begin{array}{cccc} 9 & 7 & 3 & 1 \\ \dots & 10^3 & 10^2 & 10^1 & 10^0 \end{array}$$

▶ Binary

$$X = \begin{array}{cccc} 0 & 1 & 0 & 1 \\ \dots & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

▶ Hexadecimal

$$X = \begin{array}{cccc} 1 & F & A & 8 \\ \dots & 16^3 & 16^2 & 16^1 & 16^0 \end{array}$$


From binary to hexadecimal:

- ▶ Group by 4 bits, right to left
- ▶ Each 4 bits is the value of a hexadecimal digit

E.g.: $\begin{array}{ccccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline & & & & & & & \\ 0x & & A & & & & 5 & \end{array}$

Positional representation systems

- ▶ Decimal

$$X = \begin{array}{cccc} 9 & 7 & 3 & 1 \\ \dots & 10^3 & 10^2 & 10^1 & 10^0 \end{array}$$


¿?

- ▶ Binary

$$X = \begin{array}{cccc} 0 & 1 & 0 & 1 \\ \dots & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

- ▶ Hexadecimal

$$X = \begin{array}{cccc} 1 & F & A & 8 \\ \dots & 16^3 & 16^2 & 16^1 & 16^0 \end{array}$$

Exercise

- To represent 342 in binary:

256	128	64	32	16	8	4	2	1
?	?	?	?	?	?	?	?	?

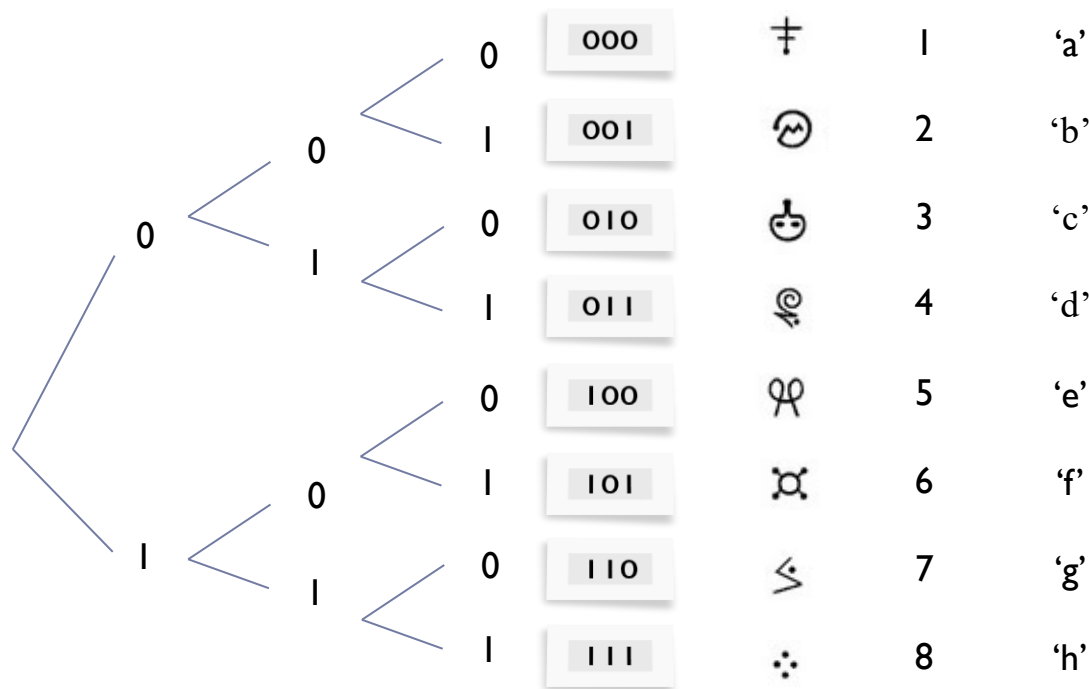
Exercise (solution)

- To represent 342 in binary:

256	128	64	32	16	8	4	2	1
	0		0		0			0
342-256=86	86-64=22	22-16=6	6-4=2	2-2=0				

Positional representation systems


- ▶ With 3 binary digits, up to 8 symbols can be represented:



Positional representation systems

- ▶ How many values can be represented with n bits?
- ▶ How many bits are needed to represent m 'values'?
- ▶ With n bits, if the minimum representable value corresponds to the number 0, what is the maximum representable numerical value?

Positional representation systems

- ▶ How many values can be represented with n bits?
 - ▶ 2^n
 - ▶ E.g.: with 4 bits up to 16 values can be represented
- ▶ How many bits are needed to represent m 'values'?
 - ▶ $\lceil \text{Log}_2(n) \rceil$ ($\text{Log}_2(n)$ round up)
 - ▶ E.g.: 6 bits are required to represent 35 values
- ▶ With n bits, if the minimum representable value corresponds to the number 0, what is the maximum representable numerical value?
 - ▶ $2^n - 1$

Exercise

- ▶ To compute the value of (23 ones):

$$\underbrace{111111111111111111111111}_{23}{}_2$$

Exercise (solution)

- To compute the value of (23 ones):

A horizontal number line with 21 vertical tick marks. The first tick mark on the left is labeled '0'. The second tick mark is labeled '2'. There are no labels for the other tick marks.

$$X = 2^{23} - 1$$

Tip:

$$\begin{array}{r} \text{I I I I I I I I I I I I I I I I I I I}_2 = X \\ + \quad 000000000000000000000000_2 = \text{I} \\ \hline \text{I } 000000000000000000000000_2 = 2^{23} \end{array}$$

$$X = 2^{23} - 1$$

Example: operations

- Add in binary:

$$\begin{array}{rcccccc} & 1 & 1 & 1 & & & \\ & & 1 & 0 & 1 & 0 & 0 \\ + & & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & \end{array}$$

Example: operations

- Add in binary:

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 10100 \\ + 11110 \\ \hline 110010 \end{array}$$

- Subtract in binary:

$$\begin{array}{r} 1 \rightarrow 1 \rightarrow \\ 01100 \\ - 01011 \\ \hline 00001 \end{array}$$

Exercise

2 minutes máx.



You have a 5-liter bottle
and a 3-liter bottle.
How can you get 4 liters
just right?



Exercise (solution)

2 minutes máx.



You have a 5-liter bottle and a 3-liter bottle. How can you get 4 liters just right?



- ▶ Fill the 5-liter jar
- ▶ Empty it into the 3-liter jar
 - ▶ There are 2 left in the 5-liter jar.
- ▶ Throw away what is in the 3-liter jar
- ▶ Transfer the 2 from the 5-liter jar to the 3-liter jar
 - ▶ There are 1 left in the 3-liter jar (-1 to 3).
- ▶ Refill the 5-liter jar
- ▶ Fill the 3-liter jar to the top, what is left in the 5-liter jar is 4 liters

Exercise

2 minutes máx.



- ▶ Using the numbers 112 and -71 in decimal base, perform addition in 10's complement.

Exercise (solution)

2 minutes máx.



- ▶ 10's complement, 112 is: 112
- ▶ 10's complement, -71 is:

$$\begin{array}{r} 1000 \\ -0071 \\ \hline 929 \end{array}$$

- ▶ Adding both:

$$\begin{array}{r} 112 \\ 929 \\ \hline \text{X } 041 \end{array}$$

$$\begin{array}{r} 112 \\ -071 \\ \hline 041 \end{array}$$

Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

1. **Alphanumeric**

1. **Characters**
2. **Strings**

2. Numerical

1. Natural and integer
2. Fixed point
3. Floating point (IEEE 754 standard)

Alphanumeric representation

- ▶ Each character is encoded as one byte.
- ▶ With n bits \Rightarrow up to 2^n characters can be encoded:

# bits	# characters	Includes...	Example
6	64	<ul style="list-style-type: none">• 26 letter: a...z• 10 number: 0...9• punctuation: .,;: ...• specials: + - [...	<i>BCDIC</i>
7	128	<ul style="list-style-type: none">• adds uppercases and control characters	<i>ASCII</i>
8	256	<ul style="list-style-type: none">• adds accented letters, ñ, semigraphic characters	<i>EBCDIC</i> <i>ASCII extended</i>
16	34.168	<ul style="list-style-type: none">• add support for Chinese, Arabic, ...	<i>UNICODE</i>

Example: ASCII table (7 bits)

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	\$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♪	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	▲	DLE	048	0	080	P	112	p
017	▼	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	\$	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↕	ETB	055	7	087	W	119	w
024	↕	CAN	056	8	088	X	120	x
025	↕	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	
029	(cursor left)	GS	061	=	093]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	☐

Copyright 1998, JimPrice.Com Copyright 1992, Loading Edge Computer Products, Inc.

Example: ASCII table (7 bits)

control characters

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	\$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♪	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	▲	DLE	048	0	080	P	112	p
017	▼	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	\$	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↕	ETB	055	7	087	W	119	w
024	↕	CAN	056	8	088	X	120	x
025	↕	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	
029	(cursor left)	GS	061	=	093]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	☐

Copyright 1998, JimPrice.Com Copyright 1992, Loading Edge Computer Products, Inc.

< 32

Example: ASCII table (7 bits)

distance between uppercase and lowercase letters

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	<u>065</u>	<u>A</u>	<u>097</u>	<u>a</u>
002	☹	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	\$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♪	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	▲	DLE	048	0	080	P	112	p
017	▼	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	\$	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↕	ETB	055	7	087	W	119	w
024	↕	CAN	056	8	088	X	120	x
025	↕	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	
029	(cursor left)	GS	061	=	093]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	☐

$$97-65=32$$

Copyright 1998, JimPrice.Com Copyright 1992, Loading Edge Computer Products, Inc.

Example: ASCII table (7 bits)

conversion of a number to a character

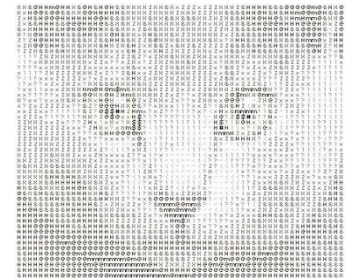
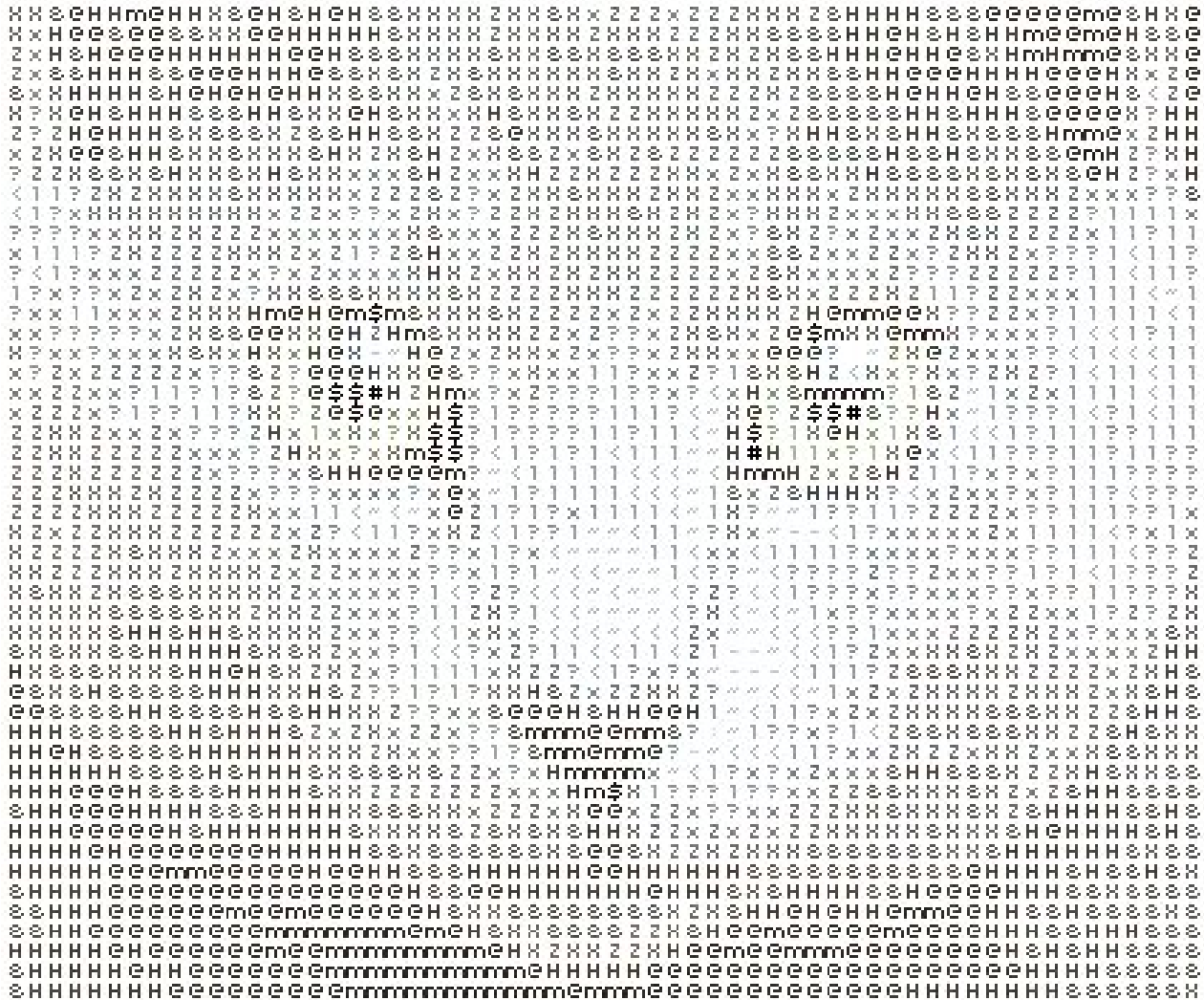
ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☹	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	\$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♪	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	▲	DLE	048	0	080	P	112	p
017	▼	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	\$	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↕	ETB	055	7	087	W	119	w
024	↕	CAN	056	8	088	X	120	x
025	↕	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	}
029	(cursor left)	GS	061	=	093]	125	~
030	(cursor up)	RS	062	>	094	^	126	
031	(cursor down)	US	063	?	095	_	127	☐

Copyright 1998, JimPrice.Com Copyright 1992, Loading Edge Computer Products, Inc.

$$6+48=54$$

Curiosity:

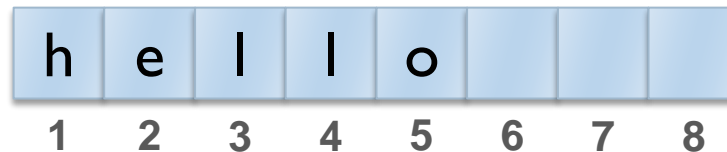
Display “image” with characters



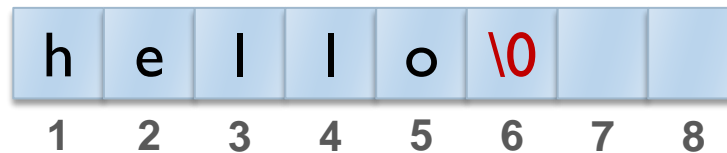
Character strings

1000	00110011
1001	01101100
...	
1008	10100011

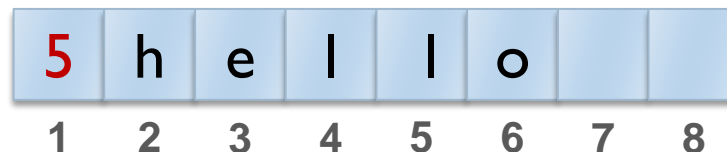
1. Fixed-length string:



2. Variable-length string with delimiter:



3. Variable-length strings with length in header:



Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

1. Alphanumeric

1. Characters
2. Strings

2. **Numerical**

1. **Natural and integer**
2. Fixed point
3. Floating point (IEEE 754 standard)

Numerical representation

- ▶ Classification of real numbers:
 - ▶ Naturals: 0, 1, 2, 3, ...
 - ▶ Integers: ... -3, -2, -1, 0, 1, 2, 3,
 - ▶ Rational: fractions ($5/2 = 2,5$)
 - ▶ Irrational: $2^{1/2}$, π , e, ...
- ▶ Infinite sets but finite representation space:
 - ▶ Impossible to represent all
- ▶ Characteristics of the representation used:
 - ▶ Represented element:
Natural, integer, ...
 - ▶ Representation range:
Interval between minor and major not representable
 - ▶ Resolution of representation:
Difference between a representable number and the following one.
It represents the maximum error committed. It can be cte. or variable.

Most used binary representation systems

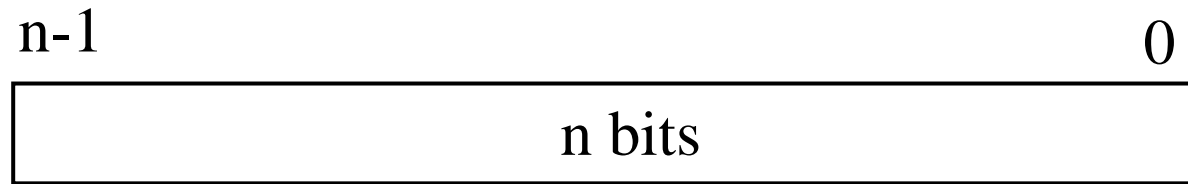
- A. (Pure) binary natural

- B. Sign-Magnitude
- C. One's complement (Ca 1)
- D. Two's complement (Ca 2) integer
- E. Biased $2^{n-1}-1$

- F. Floating point: IEEE 754 standard real

(Pure) binary or unsigned binary [natural numbers]

- Positioning system with base 2 and without fractional part.



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i$$

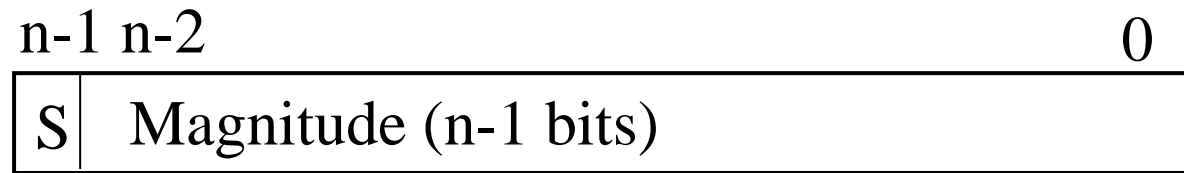
- Representation range: $[0, 2^n - 1]$
- Resolution: 1 unit

Comparative example (3 bits)

Decimal	Pure Binary
+7	111
+6	110
+5	101
+4	100
+3	011
+2	010
+1	001
+0	000
-0	N.A.
-1	N.A.
-2	N.A.
-3	N.A.
-4	N.A.
-5	N.A.
-6	N.A.
-7	N.A.

Signed binary number or Sign-Magnitude [integer numbers]

- One bit (S) is reserved for the sign ($0 \Rightarrow +$; $1 \Rightarrow -$)



$$\begin{array}{l}
 \text{Si } x_{n-1} = 0 \quad v(X) = \sum_{i=0}^{n-2} 2^i \cdot x_i \\
 \text{Si } x_{n-1} = 1 \quad v(X) = - \sum_{i=0}^{n-2} 2^i \cdot x_i
 \end{array}
 \left| \Rightarrow V(X) = (1 - 2 \cdot x_{n-1}) \cdot \sum_{i=0}^{n-2} 2^i \cdot x_i \right.$$

- Representation range: $[-2^{n-1} + 1, 2^{n-1} - 1]$
- Resolution: 1 unit
- Ambiguity of zero + complex hw. for subtraction

Comparative example (3 bits)

Decimal	Pure Binary	Sign-Magnitude
+7	111	N.A.
+6	110	N.A.
+5	101	N.A.
+4	100	N.A.
+3	011	011
+2	010	010
+1	001	001
+0	000	000
-0	N.A.	100
-1	N.A.	101
-2	N.A.	110
-3	N.A.	111
-4	N.A.	N.A.
-5	N.A.	N.A.
-6	N.A.	N.A.
-7	N.A.	N.A.

Example

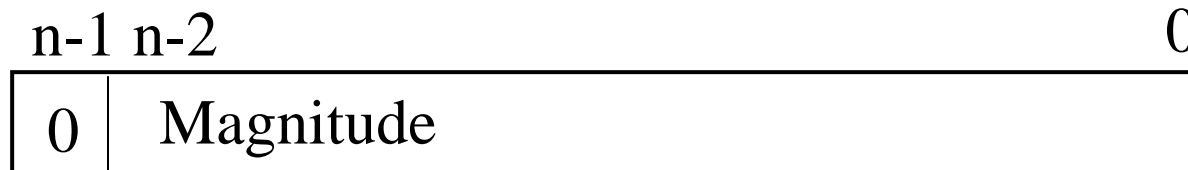
- ▶ Can we represent 745_{10} in sign-magnitude with 10 bits?

Example (solution)

- ▶ Can we represent 745_{10} in sign-magnitude with 10 bits?
- ▶ With 10 bits the range in sign-magnitude is:
 $[-2^9+1, \dots, -0, +0, \dots, 2^9-1] \Rightarrow [-511, 511]$
then, **we cannot represent 745**

One's complement (to the base minus one) [integer] (1 / 3)

- **Positive number:**
is represented in pure binary with $n-1$ bits



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i = \sum_{i=0}^{n-2} 2^i \cdot x_i$$

- Representation range (+): $[0, 2^{n-1} - 1]$
- Resolution: 1 unit

One's complement (to the base minus one) [integer] (2/3)

► **Negative number:**

- Complemented to the base minus one.
- The number $X < 0$ is represented as $2^n - X - 1$ with n bits



$$V(X) = -2^n + \sum_{i=0}^{n-1} 2^i \cdot y_i + 1$$

- Representation range (-): $[-(2^{n-1}-1), -0]$
- Resolution: **1 unit**

One's complement (to the base minus one) [integer] (3/3)

Tip: $C a 1 (X) = X$

$C a 1 (-X) = \text{change the 1's to 0's and the 0's to 1's}$

- ▶ Example: For $n=4 \Rightarrow$ the value $+3_{10} = 0011_2$
- ▶ Example: For $n=4 \Rightarrow$ the value $-3_{10} = 1100_2$
 - ▶ $- \Rightarrow 1$ (sign bit and also part of magnitude)
 - ▶ $C a 1(3) \Rightarrow 2^4 - 0011_2 - 1 = 2^4 - 3 - 1 = 12 \Rightarrow 1100_2$

- Representation range: $[-2^{n-1}+1, 2^{n-1}-1]$
- Resolution: 1 unit
- Zero has a double representation (+0 y -0)
- Symmetrical range

Comparative example (3 bits)

Decimal	Pure Binary	Sign-Magnitude	One's complement
+7	111	N.A.	N.A.
+6	110	N.A.	N.A.
+5	101	N.A.	N.A.
+4	100	N.A.	N.A.
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	N.A.	100	111
-1	N.A.	101	110
-2	N.A.	110	101
-3	N.A.	111	100
-4	N.A.	N.A.	N.A.
-5	N.A.	N.A.	N.A.
-6	N.A.	N.A.	N.A.
-7	N.A.	N.A.	N.A.

Example

With $n = 5$ bits and using one's complement:

- ▶ How is represented $X = 5$?
- ▶ How is represented $X = -5$?
- ▶ What is the value of 00111 in 1's complement?
- ▶ What is the value of 11000 in 1's complement?

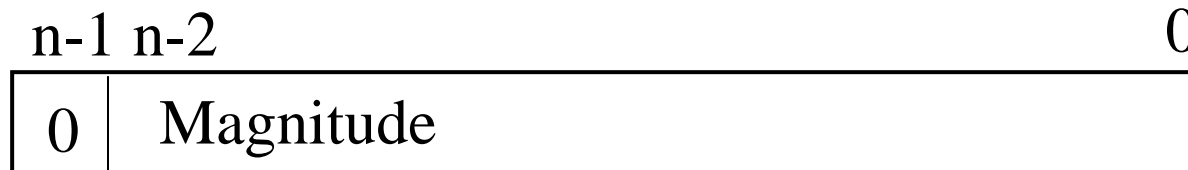
Example (solution)

With $n = 5$ bits and using one's complement:

- ▶ How is represented $X = 5$?
 - ▶ Because is positive then is like (pure) binary
 - ▶ 00101
- ▶ How is represented $X = -5$?
 - ▶ Because is negative, then 5 is complemented to one (00101)
 - ▶ 11010
- ▶ What is the value of 00111 in 1's complement?
 - ▶ Because is positive then its value is 7
- ▶ What is the value of 11000 in 1's complement?
 - ▶ Because is negative, then is complemented and is 00111 (7)
 - ▶ The value is -7

Two's complement (complement to the base) [integer] (1 / 3)

- **Positive number:**
is represented in pure binary with $n-1$ bits



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot X_i = \sum_{i=0}^{n-2} 2^i \cdot X_i$$

- Representation range (+): $[0, 2^{n-1} - 1]$
- Resolution: **1 unit**

Two's complement (complement to the base) [integer] (2/3)

► **Negative number:**

- Complemented to the base.
- The number $X < 0$ is represented as $2^n - X$ with n bits



$$V(X) = -2^n + \sum_{i=0}^{n-1} 2^i \cdot y_i$$

- Representation range (-): $[-2^{n-1}, -1]$
- Resolution: **1 unit**

Two's complement (complement to the base)

[integer] (3/3)

$$\begin{aligned}\text{Tip: } C a 2 (X) &= X \\ C a 2 (-X) &= C a 1 (X) + 1\end{aligned}$$

- ▶ Example: For $n=4 \Rightarrow +3 = 0011_2$
- ▶ Example: For $n=4 \Rightarrow -3 = 1101_2$
 - ▶ $1 \Rightarrow -$ (sign bit and also part of magnitude)
 - ▶ $C a 2 (3) = C a 2 (0011_2) = 2^4 - 3 = 13 \Rightarrow 1101_2$

- Representation range: $[-2^{n-1}, 2^{n-1}-1]$
- Resolution: 1 unit
- 0 has only one representation ($\nexists -0$)
- Asymmetric range

Comparative example (3 bits)

Decimal	Pure Binary	Sign-Magnitude	One's complement	Two's complement
+7	111	N.A.	N.A.	N.A.
+6	110	N.A.	N.A.	N.A.
+5	101	N.A.	N.A.	N.A.
+4	100	N.A.	N.A.	N.A.
+3	011	011	011	011
+2	010	010	010	010
+1	001	001	001	001
+0	000	000	000	000
-0	N.A.	100	111	N.A.
-1	N.A.	101	110	111
-2	N.A.	110	101	110
-3	N.A.	111	100	101
-4	N.A.	N.A.	N.A.	100
-5	N.A.	N.A.	N.A.	N.A.
-6	N.A.	N.A.	N.A.	N.A.
-7	N.A.	N.A.	N.A.	N.A.

Two's complement with 32-bits

$$0000 \dots 0000 \ 0000 \ 0000 \ 0000 \ 0000_{2c} = 0_{(10)}$$

$$0000 \dots 0000 \ 0000 \ 0000 \ 0001_{2c} = 1_{(10)}$$

$$0000 \dots 0000 \ 0000 \ 0000 \ 0010_{2c} = 2_{(10)}$$

...

$$0111 \dots 1111 \ 1111 \ 1111 \ 1101_{2c} = 2,147,483,645_{(10)}$$

$$0111 \dots 1111 \ 1111 \ 1111 \ 1110_{2c} = 2,147,483,646_{(10)}$$

$$0111 \dots 1111 \ 1111 \ 1111 \ 1111_{2c} = 2,147,483,647_{(10)}$$

$$1000 \dots 0000 \ 0000 \ 0000 \ 0000_{2c} = -2,147,483,648_{(10)}$$

$$1000 \dots 0000 \ 0000 \ 0000 \ 0001_{2c} = -2,147,483,647_{(10)}$$

$$1000 \dots 0000 \ 0000 \ 0000 \ 0010_{2c} = -2,147,483,646_{(10)}$$

...

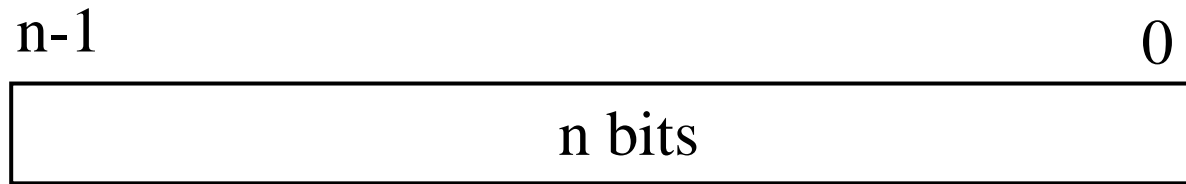
$$1111 \dots 1111 \ 1111 \ 1111 \ 1101_{2c} = -3_{(10)}$$

$$1111 \dots 1111 \ 1111 \ 1111 \ 1110_{2c} = -2_{(10)}$$

$$1111 \dots 1111 \ 1111 \ 1111 \ 1111_{2c} = -1_{(10)}$$

Biased $2^{n-1}-1$ representation [integer]

- ▶ X value with n bits is represented as $X + 2^{n-1} - 1$
- ▶ Bias refers to the value $2^{n-1} - 1$



$$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i - (2^{n-1} - 1)$$

- Representation range: $[-(2^{n-1} - 1), 2^{n-1} - 1]$
- Resolution: 1 unit
- There is no ambiguity with 0

Comparative example (3 bits)

Decimal	Pure Binary	Sign-Magnitude	One's complement	Two's complement	Biased-3
+7	111	N.A.	N.A.	N.A.	N.A.
+6	110	N.A.	N.A.	N.A.	N.A.
+5	101	N.A.	N.A.	N.A.	N.A.
+4	100	N.A.	N.A.	N.A.	111
+3	011	011	011	011	110
+2	010	010	010	010	101
+1	001	001	001	001	100
+0	000	000	000	000	011
-0	N.A.	100	111	N.A.	N.A.
-1	N.A.	101	110	111	010
-2	N.A.	110	101	110	001
-3	N.A.	111	100	101	000
-4	N.A.	N.A.	N.A.	100	N.A.
-5	N.A.	N.A.	N.A.	N.A.	N.A.
-6	N.A.	N.A.	N.A.	N.A.	N.A.
-7	N.A.	N.A.	N.A.	N.A.	N.A.

Representations

summary

Name	Pure binary	Sign-magnitude	Ca1	Ca2	Bias $2^{n-1}-1$
Represent	Natural	Integer	Integer	Integer	Integer
Sign	All bits for magnitude, no sign	MSB is sign ($0 \Rightarrow +$ $1 \Rightarrow -$)	MSB is sign and magnitude ($0 \Rightarrow +$ $1 \Rightarrow -$)	MSB is sign and magnitude ($0 \Rightarrow +$ $1 \Rightarrow -$)	
Range	$[0, 2^n - 1]$	$[-2^{n-1} + 1, 2^{n-1} - 1]$	$[-2^{n-1} + 1, 2^{n-1} - 1]$	$[-2^{n-1}, 2^{n-1} - 1]$	$[-(2^{n-1} - 1), 2^{n-1} - 1]$
Resolution	1 unit	1 unit	1 unit	1 unit	1 unit
Disadvantage	No negative	+0 y -0	+0 y -0	Asymmetric range	Asymmetric range
Advantage		Symmetric range	Symmetric range	(No \exists -0)	(No \exists -0)
Tip		Remove first bit and compute pure binary value	+ : = pure binary - : switch 1 by 0 and 0 by 1	+ : = pure binary - : Ca1 + 1	Subtract bias ($2^{n-1} - 1$)
Value		$V(X) = (1 - 2 \cdot x_{n-1}) \cdot \sum_{i=0}^{n-2} 2^i \cdot x_i$	$\begin{aligned} +: V(X) &= \sum_{i=0}^{n-2} 2^i \cdot x_i \\ -: V(X) &= -2^n + \sum_{i=0}^{n-1} 2^i \cdot x_i + 1 \end{aligned}$	$\begin{aligned} +: V(X) &= \sum_{i=0}^{n-2} 2^i \cdot x_i \\ -: V(X) &= -2^n + \sum_{i=0}^{n-1} 2^i \cdot x_i \end{aligned}$	$V(X) = \sum_{i=0}^{n-1} 2^i \cdot x_i - (2^{n-1} - 1)$

Comparative example (3 bits)

summary

Decimal	Pure Binary	Sign-Magnitude	One's complement	Two's complement	Biased-3
+7	111	N.A.	N.A.	N.A.	N.A.
+6	110	N.A.	N.A.	N.A.	N.A.
+5	101	N.A.	N.A.	N.A.	N.A.
+4	100	N.A.	N.A.	N.A.	111
+3	011	011	011	011	110
+2	010	010	010	010	101
+1	001	001	001	001	100
+0	000	000	000	000	011
-0	N.A.	100	111	N.A.	N.A.
-1	N.A.	101	110	111	010
-2	N.A.	110	101	110	001
-3	N.A.	111	100	101	000
-4	N.A.	N.A.	N.A.	100	N.A.
-5	N.A.	N.A.	N.A.	N.A.	N.A.
-6	N.A.	N.A.	N.A.	N.A.	N.A.
-7	N.A.	N.A.	N.A.	N.A.	N.A.

Example

Indicate the representation of the following numbers, giving a brief justification of your answer:

1. **-32** in one's complement with **6 bits**
2. **-32** in two's complement with **6 bits**
3. **-10** in sign-magnitude with **5 bits**
4. **+14** in two's complement with **5 bits**

Example (solution)

1. With 6 bits **is not representable** in 1C:
 $[-2^{6-1}+1, \dots, -0, +0, \dots, 2^{6-1}-1]$
2. 1C + 1 -> **100000**
3. Sign=1, magnitude=1010 -> **11010**
4. Positive -> 1C=2C=SM -> **01110**

Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

1. Alphanumeric

1. Characters
2. Strings

2. Numerical

1. Natural and integer

1. Arithmetic operations

2. Fixed point
3. Floating point (IEEE 754 standard)

Comparison of arithmetic in B, 1C and 2C

	Binary	One's complement	Two' complement
Add	$\begin{array}{r} 10110 \\ 01100 \\ \hline 100010 \end{array}$	same as binary	same as binary
Subtract	$\begin{array}{r} 10110 \\ 01100 \\ \hline 01010 \end{array}$	add and if there is C_{n-1} then add C_{n-1} to total	add and if there is C_{n-1} then discard it

In hardware, it is easier to operate with complement

Comparison of arithmetic in B, 1C and 2C

why add the carry to the result in 1C

	Bin	ment	
Add	<div><ul style="list-style-type: none">• $-X$ is represented as $2^n - X - 1$• $-Y$ is represented as $2^n - Y - 1$• $-(X + Y)$ is represented as $2^n - (X+Y) - 1$• $-(X + Y)$ the operation gives $2^n + 2^n - (X + Y) - 2$ <div>+ 1</div></div>		
Subtract	<div>10110 01100 ----- 01010</div>	<div>add and if there is C_{n-1} then add C_{n-1} to total</div>	<div>add and if there is C_{n-1} then discard it</div>

Correction of the result by adding the carry...

Comparison of arithmetic in B, 1C and 2C

why discard the carry in 2C

	Bin		ment
Add		<ul style="list-style-type: none"> • $-X$ is represented as $2^n - X$ • $-Y$ is represented as $2^n - Y$ • $-(X + Y)$ is represented as $2^n - (X+Y)$ • $-(X + Y)$ the operation gives $2^n + 2^n - (X + Y)$ 	
Subtract	<pre> 10110 01100 ----- 01010 </pre>	add and if there is C_{n-1} then add C_{n-1} to total	add and if there is C_{n-1} then discard it

Correction of the result by discarding the carry...

Comparison of arithmetic in B, 1C and 2C

	Binary	One's complement	Two' complement
Detect overflow	<p>The result needs 1 bit more</p> <p>There are C_n</p>	<p>Adding ++ is −, Adding − − is +</p> <p>$C_n \neq C_{n-1}$</p>	<p>Adding ++ is −, Adding − − is +</p> <p>$C_n \neq C_{n-1}$</p>
Sign extension	<p>0...0 10110</p>	<p>1...1[↙]10110 0...0[↙]00110</p>	<p>1...1[↙]10110 0...0[↙]00110</p>
...

Example

- ▶ Using 5 bits, compute the following additions in 1's complement:
 - a) $4 + 12$
 - b) $4 - 12$
 - c) $-4 - 12$

Example (solution)

By using 5 bits in 1's complement

a) $4 + 12$

00100

01100

10000 $\Rightarrow -15 \Rightarrow$ negative! \Rightarrow overflow

b) $4 - 12$

00100

10011

10111 $\Rightarrow -8$

c) $-4 - 12$

11011

10011

101110 \Rightarrow 6 bits are needed \Rightarrow overflow

Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

1. Alphanumeric

1. Characters
2. Strings

2. Numerical

1. Natural and integer
2. **Fixed point**
3. Floating point (IEEE 754 standard)

More representation necessities...

► How to represent?

- Very large numbers: $30.556.926.000_{(10)}$
- Very small numbers: $0.0000000000529177_{(10)}$
- Fractional numbers: 1.58567

Reminder

Example of failure...

- ▶ **Ariane 5 explosion (first flight)**
 - ▶ Sent by ESA in June 1996
 - ▶ Cost of development:
10 years and 7 billion dollars
 - ▶ Exploded 40 seconds after launch, at 3700 meters altitude.
 - ▶ Failure due to total loss of altitude information:
 - ▶ The inertial reference system software performed the conversion of a 64-bit floating point real value to a 16-bit integer value.
 - ▶ The number to be stored was greater than 32767 (the largest 16-bit signed integer) and a conversion failure and exception occurred.



Fixed point [rationals]

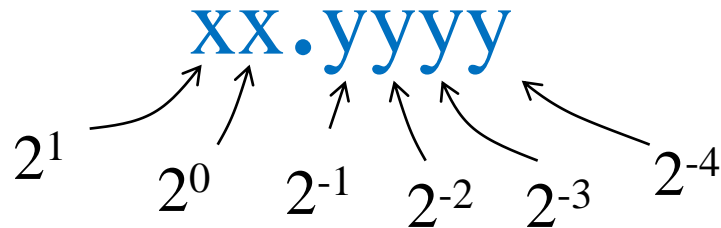
- ▶ The position of the binary point is fixed and the weights associated with the decimal places are used.

- ▶ Example:

$$1001.1010 = 2^4 + 2^0 + 2^{-1} + 2^{-3} = 9.625$$

Fractional values in binary with fixed point

► Example with 6 bits:



- Example:

$$10,1010_{(2)} = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{10}$$

- Using this fixed point, the range is:
 - [0 a 3.9375 (almost 4)]

Fractional powers of 2

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	

Contents

1. Introduction

1. Motivation and goals
2. Positional (numeral) systems

2. Representations

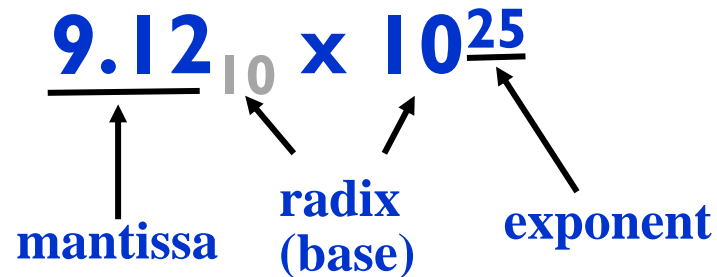
1. Alphanumeric

1. Characters
2. Strings

2. Numerical

1. Natural and integer
2. Fixed point
3. **Floating point (IEEE 754 standard)**

Floating-point numbers



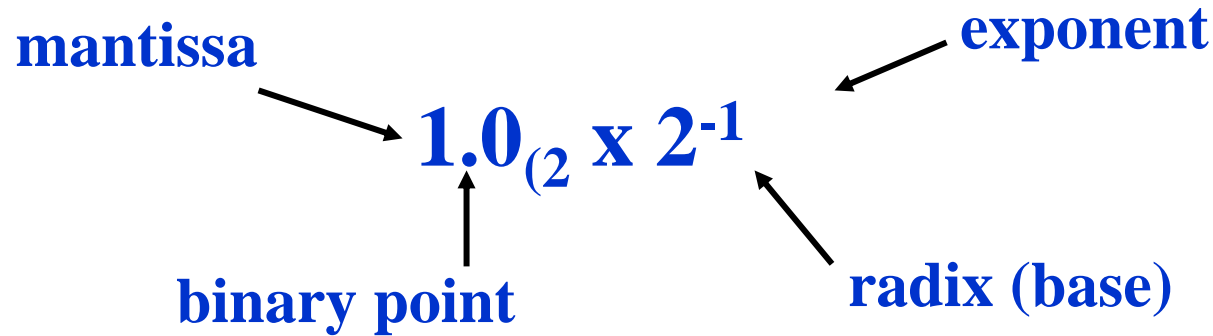
The diagram illustrates the components of the scientific notation 9.12×10^{25} . The mantissa '9.12' is underlined and labeled 'mantissa' with an upward arrow. The radix '10' is labeled 'radix (base)' with an upward arrow. The exponent '25' is underlined and labeled 'exponent' with an upward arrow. The multiplication sign 'x' is positioned between the mantissa and the radix.

$$\underline{9.12} \times 10^{\underline{25}}$$

mantissa radix (base) exponent

- ▶ Each number has a mantissa and an **exponent**
- ▶ Scientific notation (in decimal): normalized form
 - ▶ Only one digit different to 0 on the left of decimal point
- ▶ The number is adapted to the **order of magnitude** of the value to be represented, by translating the *decimal point* by using the exponent

Scientific notation in binary



- ▶ Normalized form:
One 1 (only one digit) in the left of the binary point
 - ▶ Normalized: 1.0001×2^{-9} ,
 - ▶ Not normalized: 0.0011×2^{-8} , 10.0×2^{-10}

IEEE 754 Floating Point Standard

[rationals]



- ▶ Floating point standard used in most computers.
- ▶ **Characteristics** (unless special cases):
 - ▶ Exponent: excess-k with bias $k = 2^{\text{num_bits_in_exponent}} - 1$
 - ▶ Mantissa: sign-magnitude, normalized, with implicit bit
- ▶ Different **formats**:
 - ▶ **Single precision**: 32 bits (sign: 1, exponent: 8, mantissa: 23 and bias: 127)
 - ▶ **Double precision**: 64 bits (sign: 1, exponent: 11, mantissa: 52 and bias: 1023)
 - ▶ **Quad-precision**: 128 bits (sign: 1, exponent: 15, mantissa: 112 and bias: 16383)

Normalization and implicit bit

► Normalization

In order to normalize the mantissa, the exponent is adjusted to have a most significant bit of value 1

► Example: 100100000000000000000000 $\times 2^3$ (already normalized)

► Example: 000100000000010101 $\times 2^3$ (is not)

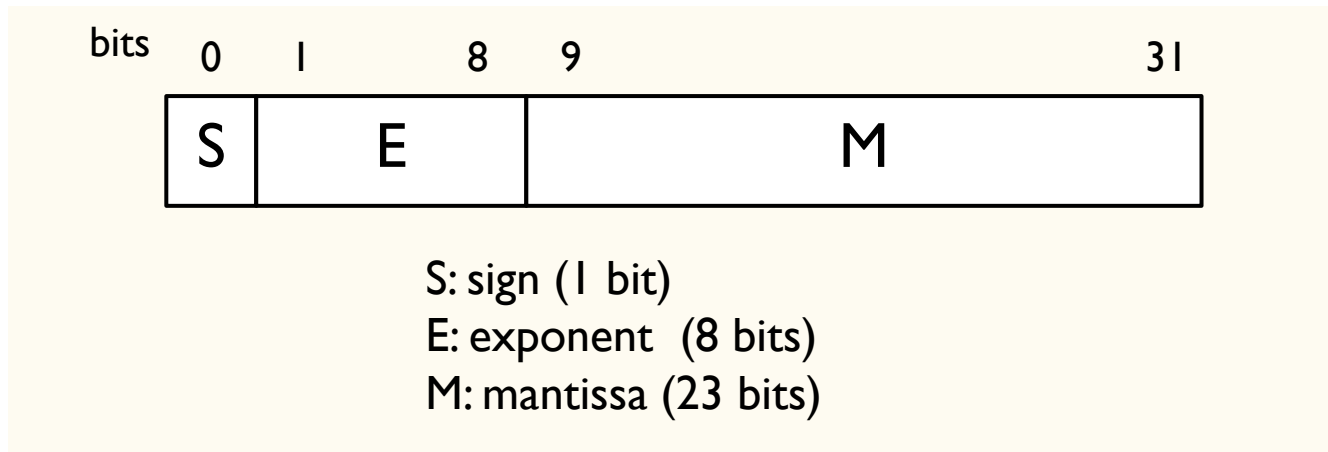
100000000010101000 $\times 2^0$ (now it is)

► Implicit bit

Once normalized, since the most significant bit is 1, it is **not** stored to leave space for one more bit (increases accuracy).

► This makes it possible to represent mantissa with one bit more

IEEE Standard 754 (single precision)



- ▶ The value is computed (unless special cases) as:

$$\mathbf{N = (-1)^S \times 2^{E-127} \times 1.M}$$

where:

$S = 0$ for positive numbers, $S = 1$ for negative numbers

$0 < E < 255$ ($E=0$ y $E=255$ are special cases)

$000000000000000000000000 \leq M \leq 111111111111111111111111$

IEEE Standard 754 (single precision)

[rationals]

► Special cases:

$$(-1)^s \times 0.\text{mantissa} \times 2^{-126}$$

Exponent	Mantissa	Special value
0 (0000 0000)	0	+/- 0 (depends on sign)
0 (0000 0000)	$\neq 0$	Number NOT normalized
255 (1111 1111)	$\neq 0$	NaN (0/0, sqrt(-4),)
255 (1111 1111)	0	+/- infinite (depends on sign)
1-254	Any	Normalized number (no special)

$$(-1)^s \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$$

Examples

S	E	M	N
1	00000000	000000000000000000000000	-0 (Exception 0) E=0 y M=0.
1	01111111	000000000000000000000000	$-2^0 \times 1.0_2 = -1$
0	10000001	111000000000000000000000	$+2^2 \times 1.111_2 = +2^2 \times (2^0+2^{-1}+2^{-2}+2^{-3}) = +7.5$
0	11111111	000000000000000000000000	∞ (Exception ∞) E=255 y M=0
0	11111111	100000000000000000000001	NaN (Not a Number) E=255 y M \neq 0.

Example

- a) Calculate the value in decimal associated to this number
0 10000011 110000000000000000000000
represented in IEEE 754 single precision

Example (solution)

- a) Calculate the value in decimal associated to this number
0 10000011 110000000000000000000000
represented in IEEE 754 single precision

- a) Sign bit: $0 \Rightarrow (-1)^0 = +1$
b) Exponent: $10000011_2 = 131_{10} \Rightarrow E - 127 = 131 - 127 = 4$
c) Mantissa: $110000000000000000000000 \Rightarrow 1 \times 2^{-1} + 1 \times 2^{-2} = 0.75$

The decimal value is $+1 \times 2^4 \times 1.75 = +28$

Exercise

- b) Represent the number -9 using IEEE 754 single precision

Exercise (Solution)

b) Represent the number -9 using IEEE 754 single precision

$$-9_{10} = -1001_2 = -1001_2 \times 2^0 = -1.001_2 \times 2^3 \text{ (normalized mantissa)}$$

a) Sign: negative $\Rightarrow S=1$

b) Exponent: $3+127 \text{ (bias)} = 130 \Rightarrow 10000010$

c) Mantissa: $1.001 \text{ (impl. bit)} \Rightarrow 001000000000000000000000$

-9 is represented by $1 \ 10000010 \ 001000000000000000000000$

IEEE Standard 754 (single precision) [rationals]

- ▶ Range of representable magnitudes (regardless of sign):

- ▶ Smallest normalized:

- $2^{-127} \times 1.000000000000000000000000_2$

- ▶ Largest normalized:

- $2^{254-127} \times 1.111111111111111111111111_2$

- ▶ Smallest not normalized :

- $2^{-126} \times 0.000000000000000000000001_2$

- ▶ Largest not normalized :

- $2^{-126} \times 0.111111111111111111111111_2$

Exponent	Mantissa	Special value
0	$\neq 0$	Not normalized
1-254	any	Normalized

$$(-1)^s * 0.\text{mantisa} * 2^{-126}$$

$$(-1)^s * 1.\text{mantisa} * 2^{\text{exponente}-127}$$

IEEE Standard 754 (single precision)

[rationals]

► Range of representable magnitudes (regardless of sign):

► Smallest normalized:

$$2^{-127} \times 1.000000000000000000000000_2 = 2^{-126}$$

► Largest normalized:

$$2^{254-127} \times 1.111111111111111111111111_2 = 2^{127} \times (2 - 2^{-23}) = 2^{128} \times (1 - 2^{-24})$$

► Smallest not normalized :

$$2^{-126} \times 0.000000000000000000000001_2 = 2^{-149}$$

► Largest not normalized :

$$2^{-126} \times 0.111111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$$

Tip:

$$\begin{array}{rcl} & 1.111111111111111111111111_2 & = X \\ + & 0.000000000000000000000001_2 & = 2^{-23} \\ \hline & 10.000000000000000000000000_2 & = 2 \\ & & X = 2 - 2^{-23} \end{array}$$

IEEE Standard 754 (single precision) [rationals]

► Range of representable magnitudes (regardless of sign):

► Smallest normalized:

$$2^{-127} \times 1.000000000000000000000000_2 = 2^{-126} = 2^{-127} \times 0.5$$

► Largest normalized:

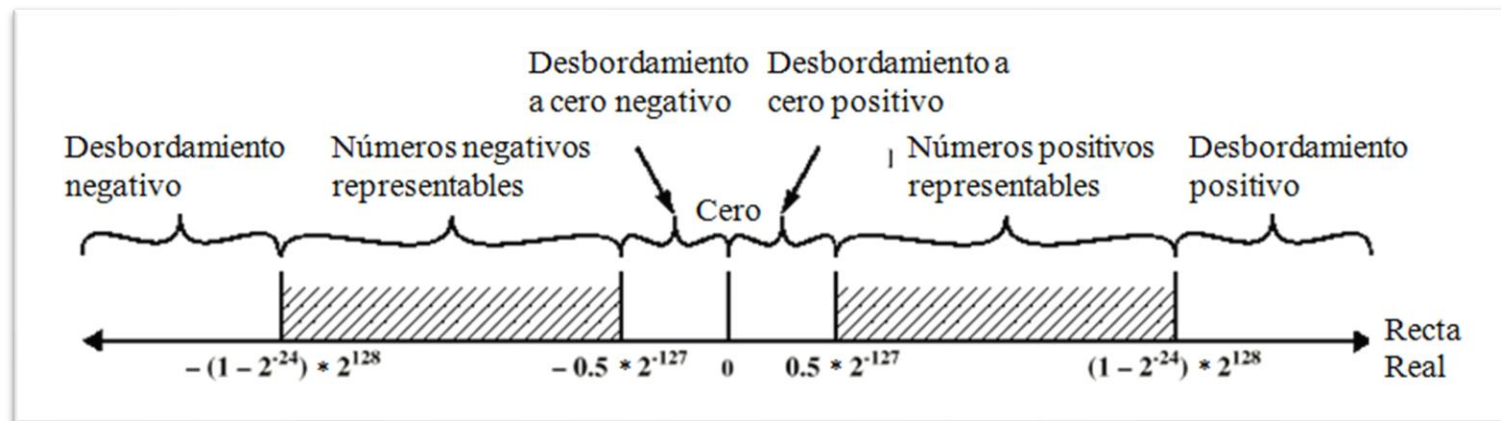
$$2^{254-127} \times 1.111111111111111111111111_2 = 2^{127} \times (2 - 2^{-23}) = 2^{128} \times (1 - 2^{-24})$$

► Smallest not normalized :

$$2^{-126} \times 0.000000000000000000000001_2 = 2^{-149}$$

► Largest not normalized :

$$2^{-126} \times 0.111111111111111111111111_2 = 2^{-126} \times (1 - 2^{-23})$$



Exercise

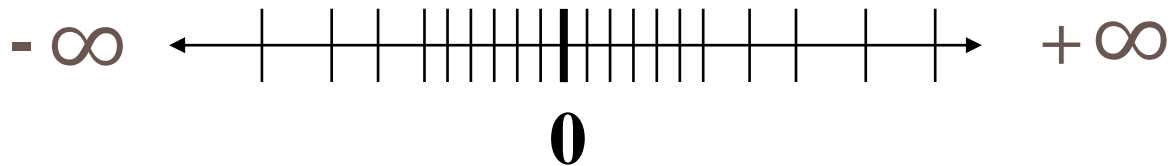
- ▶ How many *floats* (single precision floating point numbers) are between 1 and 2 (not included)?
- ▶ How many *float* (single precision floating point numbers) are between 2 and 3 (not included)?

Exercise (Solution)

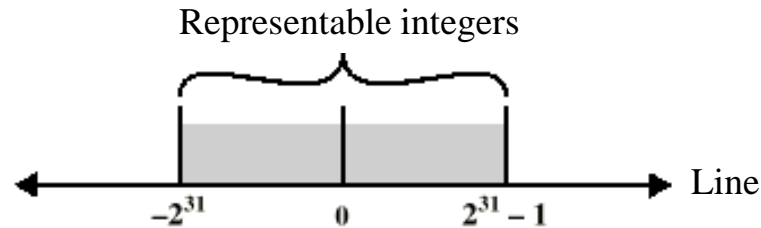
- ▶ How many *floats* (single precision floating point numbers) are between 1 and 2 (not included)?
 - ▶ $1 = 1.000000000000000000000000 \times 2^0$
 - ▶ $2 = 1.000000000000000000000000 \times 2^1$
 - ▶ Between 1 and 2 there are 2^{23} numbers
- ▶ How many *float* (single precision floating point numbers) are between 2 and 3 (not included)?
 - ▶ $2 = 1.000000000000000000000000 \times 2^1$
 - ▶ $3 = 1.100000000000000000000000 \times 2^1$
 - ▶ Between 2 and 3 there are 2^{22} numbers

Discrete representation

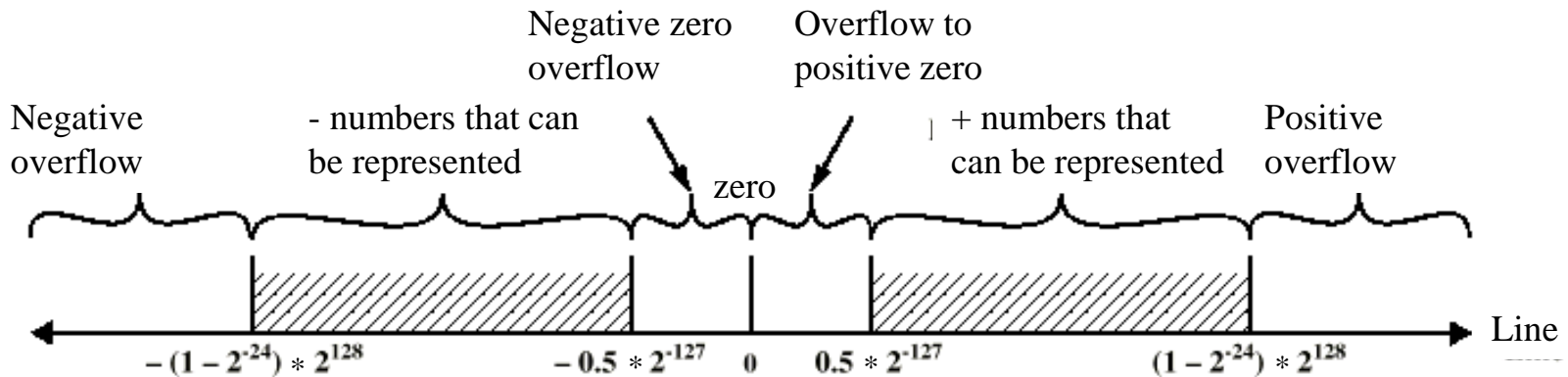
- ▶ Variable resolution:
denser near zero, less towards infinity



Representable numbers



(a) Two's complement integers



(b) Floating point numbers

Example 1

inaccuracy

0.4 →

0	0111101	10011001100110011001101
---	---------	-------------------------



3.9999998 e-1

0.1 →

0	01111011	10011001100110011001100
---	----------	-------------------------



9.9999994 e-2

Example 2

inaccuracy

- ▶ How does C performs a division?

t2.c

```
#include <stdio.h>

int main ( )
{
    float a ;

    a = 3.0/7.0 ;
    if (a == 3.0/7.0)
        printf("Equal\n") ;
    else printf("Not equal\n") ;
    return (0) ;
}
```

Example 2

inaccuracy

- ▶ How does C performs a division?

t2.c

```
#include <stdio.h>

int main ( )
{
    float a ;

    a = 3.0/7.0 ;
    if (a == 3.0/7.0)
        printf("Equal\n") ;
    else printf("Not equal\n") ;
    return (0) ;
}
```

```
$ gcc -o t2 t2.c
$ ./t2
Not equal
```

Example 2

inaccuracy

- ▶ How does C performs a division?

t2.c

```
#include <stdio.h>

int main ( )
{
    float a ;
    a = 3.0/7.0 ;
    if (a == 3.0/7.0)
        printf("Equal\n") ;
    else printf("Not equal\n") ;
    return (0) ;
}
```

float

double

```
$ gcc -o t2 t2.c
$ ./t2
Not equal
```

Example 3

inaccuracy

- ▶ The associative property is not always satisfied
 $a + (b + c) = (a + b) + c$?

t1.c

```
#include <stdio.h>

int main ( )
{
    float x, y, z ;

    x = 10e30;  y = -10e30;  z = 1;
    printf("(x+y)+z = %f\n", (x+y)+z) ;
    printf("x+(y+z) = %f\n", x+(y+z)) ;

    return (0) ;
}
```

Example 3

inaccuracy

- ▶ The associative property is not always satisfied
 $a + (b + c) = (a + b) + c$?

t1.c

```
#include <stdio.h>

int main ( )
{
    float x, y, z ;

    x = 10e30;  y = -10e30;  z = 1;
    printf("(x+y)+z = %f\n", (x+y)+z) ;
    printf("x+(y+z) = %f\n", x+(y+z)) ;

    return (0) ;
}
```

```
$ gcc -o t1 t1.c
```

```
$ ./t1
```

```
(x+y)+z = 1.000000
```

```
x+(y+z) = 0.000000
```


Floating-point is not associative

- ▶ Floating-point is not associative

- ▶ $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, $z = 1.0$

- ▶ $x + (y + z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0)$
 $= -1.5 \times 10^{38} + (1.5 \times 10^{38}) = 0.0$

- ▶ $(x + y) + z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0$
 $= (0.0) + 1.0 = 1.0$

- ▶ Floating point operations are not associative

- ▶ Results are approximated

- ▶ 1.5×10^{38} is so much larger than 1.0

- ▶ $1.5 \times 10^{38} + 1.0$ in floating point representation is still 1.5×10^{38}

Example

int → **float** → **int**

```
if (i == (int) ((float) i)) {  
    printf("true");  
}
```

- ▶ **Not** always prints "true"
- ▶ Most integer values (specially larger ones) don't have an exact floating point representation
- ▶ What about double?

Example

- ▶ The number 133000405 in binary is:
 - ▶ 111111011010110110011010101 (27 bits)
- ▶ 111111011010110110011010101 $\times 2^0$
- ▶ When is normalized:
 - ▶ 1.11111011010110110011010101 $\times 2^{26}$
 - ▶ S = 0 (positive)
 - ▶ e = 26 \rightarrow E = 26 + 127 = 153
 - ▶ M = 11111011010110110011010 (last 3 bits are lost)
- ▶ The normalized number stored is:
 - ▶ 1.11111011010110110011010 $\times 2^{26} =$
 - ▶ 111111011010110110011010 $\times 2^3 = 133000400$

Example

float → **int** → **float**

```
if (f == (float)((int) f)) {  
    printf("true");  
}
```

- ▶ Not always true
- ▶ Numbers with decimals do not have integer representation

Rounding

- ▶ Rounding removes less significant digits from a number to obtain an approximate value.
- ▶ **Types** of rounding:
 - ▶ Round **to + ∞**
 - ▶ Round it “up”: $2.001 \rightarrow 3$, $-2.001 \rightarrow -2$
 - ▶ Round **to - ∞**
 - ▶ Round it “down”: $1.999 \rightarrow 1$, $-1.999 \rightarrow -2$
 - ▶ **Truncate**
 - ▶ Discard last bits: $1.299 \rightarrow 1.2$
 - ▶ Round **to nearest (ties to even)**
 - ▶ $2.4 \rightarrow 2$, $2.6 \rightarrow 3$, $-1.4 \rightarrow -1$
 - ▶ If number falls midway then it is rounded to the nearest value with an even least significant digit ($+23.5 \rightarrow +24 \leftarrow +24.5$; $-23.5 \rightarrow -24 \leftarrow -24.5$)

Rounding

- ▶ Rounding means losing accuracy.
- ▶ Rounding occurs:
 - ▶ When moving to a representation with fewer representables:
 - ▶ E.g.: A value from double to single precision
 - ▶ E.g.: A floating point value to integer
 - ▶ When performing arithmetic operations:
 - ▶ E.g.: After adding two floating-point numbers (using guard bits)

Guard bits

- ▶ **Guard digits** are used to improve accuracy:
 - ▶ FP hardware internally includes additional bits for operations
 - ▶ After operation, guard bits are eliminated: rounding
- ▶ Example: $2.65 \times 10^0 + 2.34 \times 10^2$

	WITHOUT guard bits	WITH guard bits
1.- equalize exponents	0.02×10^2 $+ 2.34 \times 10^2$	$0.02\textcolor{blue}{65} \times 10^2$ $+ 2.34\textcolor{blue}{00} \times 10^2$
2.- add	2.36×10^2	$2.36\textcolor{blue}{65} \times 10^2$
3.- round	$2.3\textcolor{red}{6} \times 10^2$	$2.3\textcolor{red}{7} \times 10^2$

Floating point operations

- ▶ Add

- ▶ Subtract

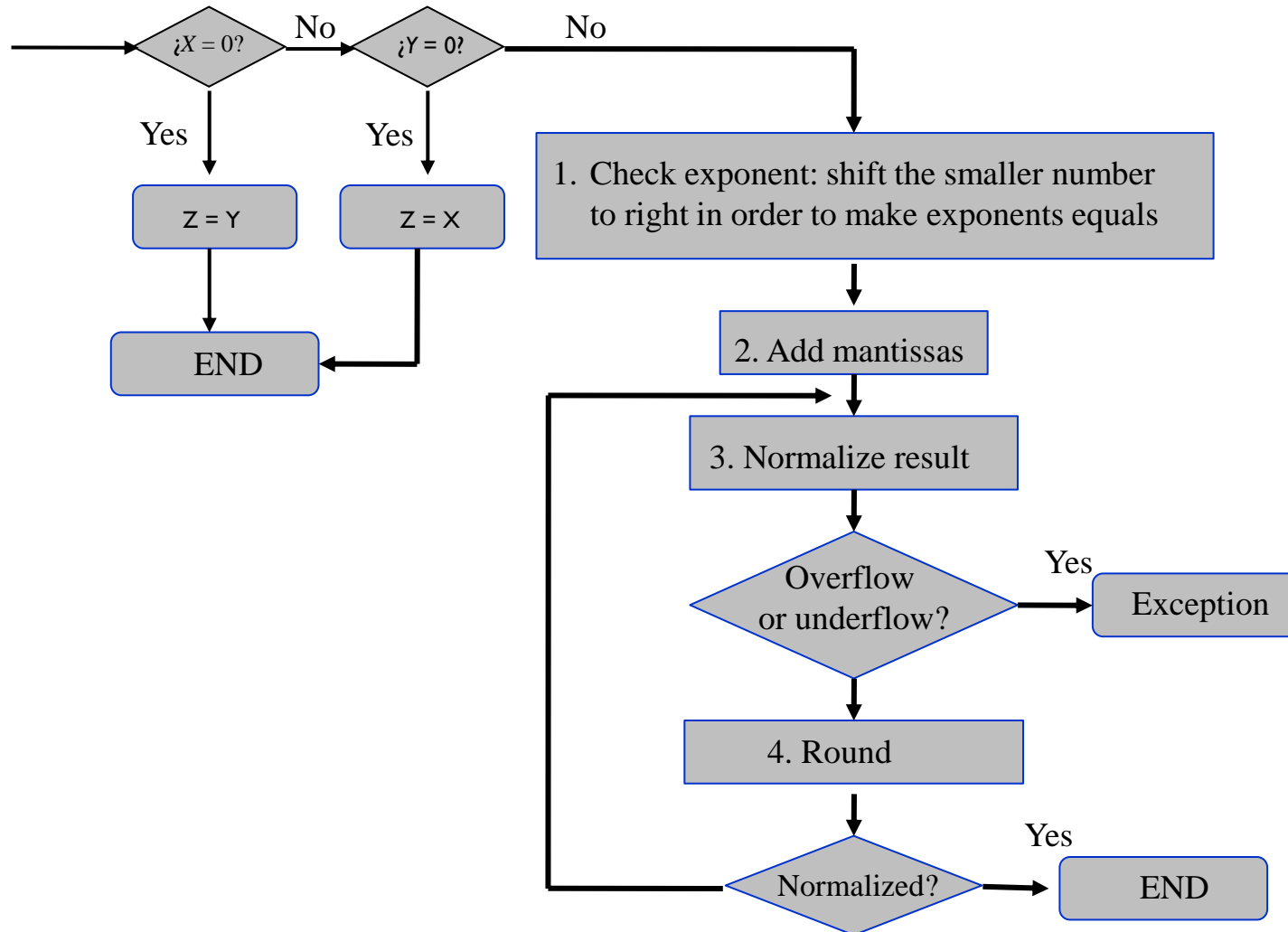
1. Check zero values.
2. Equalize exponents (shift smaller number to the right).
3. Add/subtract mantissa.
4. Normalize the result.

- ▶ Multiply

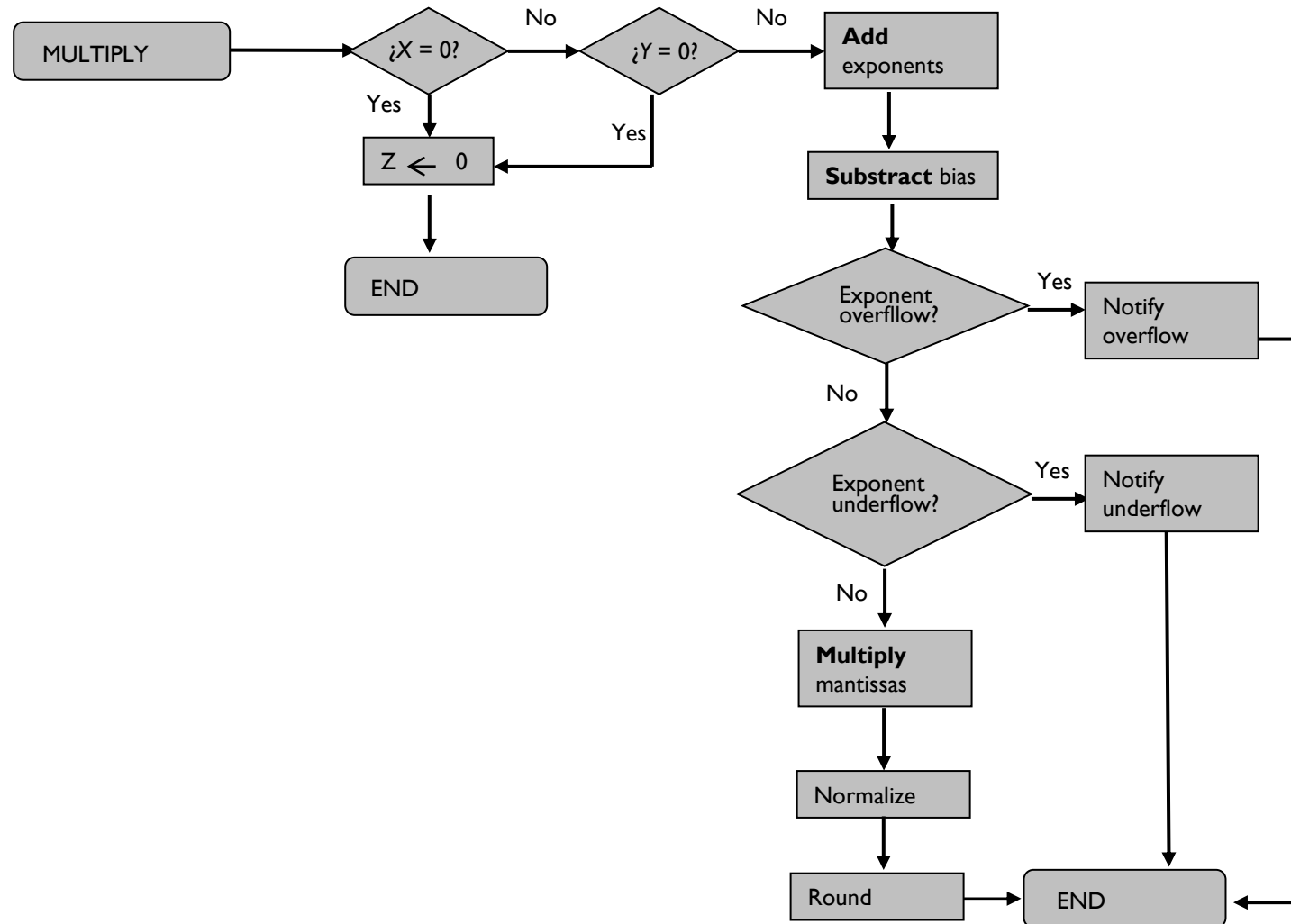
- ▶ Divide

1. Check zero values.
2. Add/subtract exponents.
3. Multiply/divide mantissa (taking into account the sign).
4. Normalize the result.
5. Rounding the result.

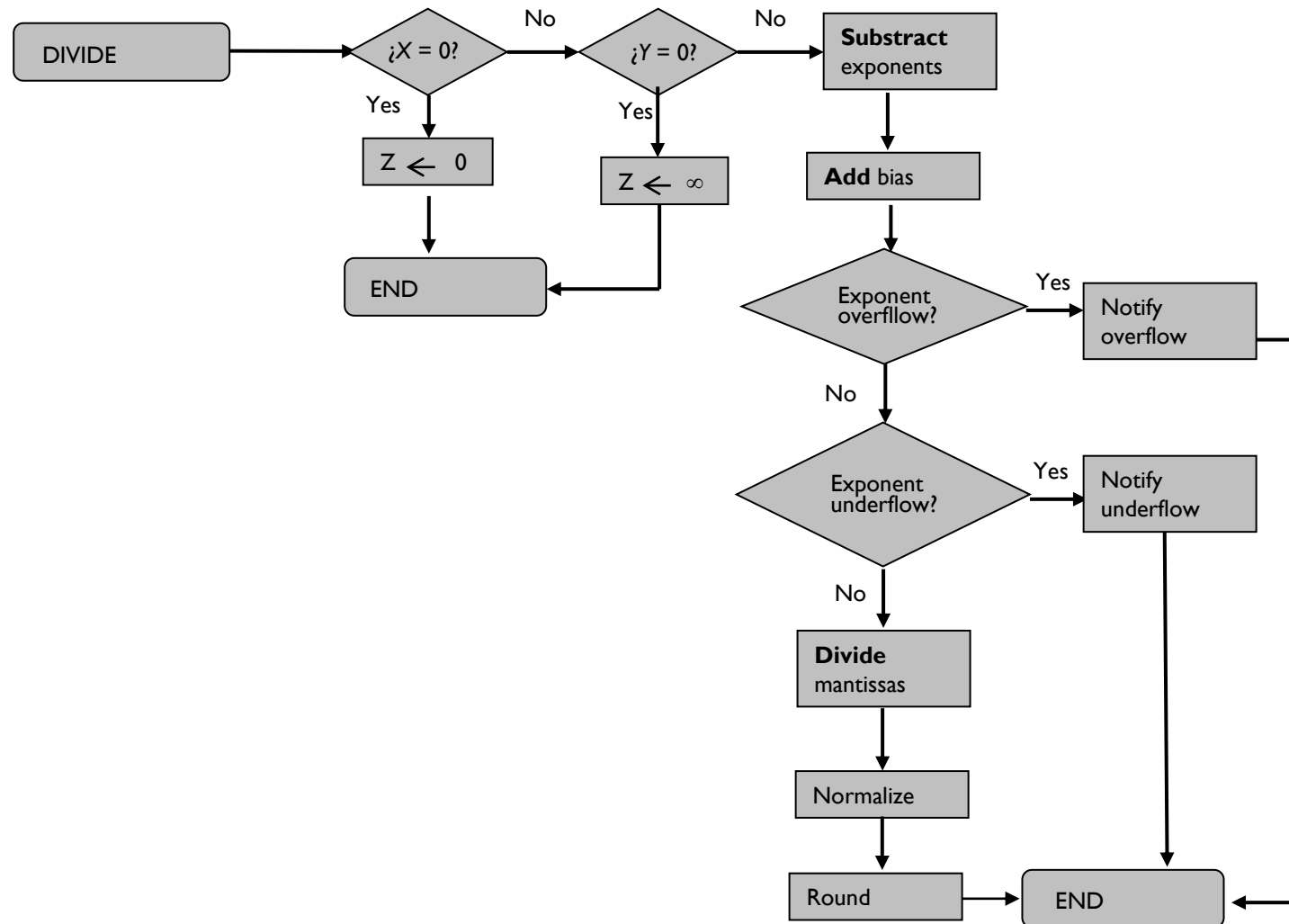
Additions and subtractions: $Z=X+Y$ y $Z=X-Y$



Multiplication: $Z=X*Y$



Division: $Z = X/Y$



Exercise

- ▶ Using the IEEE 754 format, add 7.5 and 1.5 step by step.

Solution

1) $7.5 + 1.5 =$

2) $1.111 * 2^2 + 1.1 * 2^0 =$

3) $1.111 * 2^2 + 0.011 * 2^2 =$

4) $10.010 * 2^2 =$

5) $1.0010 * 2^3$

To binary

Equalize
exponents

Add

Adjust
exponents

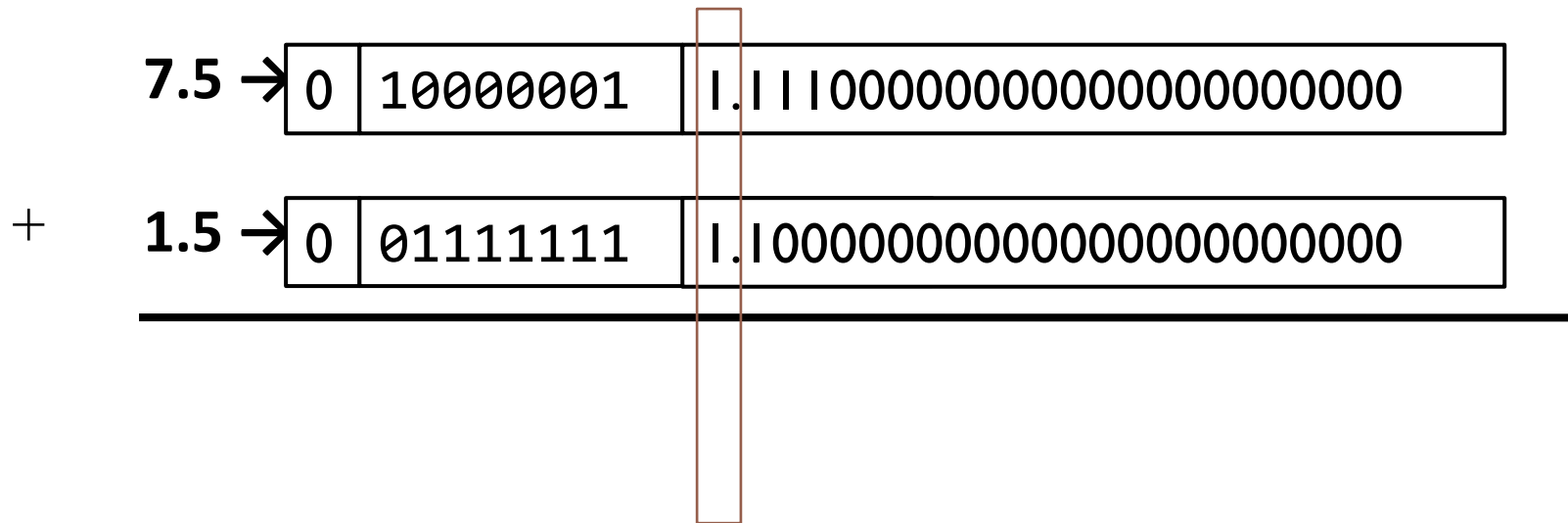
Solution

► Representation of the numbers

$$\begin{array}{r} 7.5 \rightarrow 0 \quad 10000001 \quad 111000000000000000000000 \\ + \quad 1.5 \rightarrow 0 \quad 01111111 \quad 100000000000000000000000 \end{array}$$

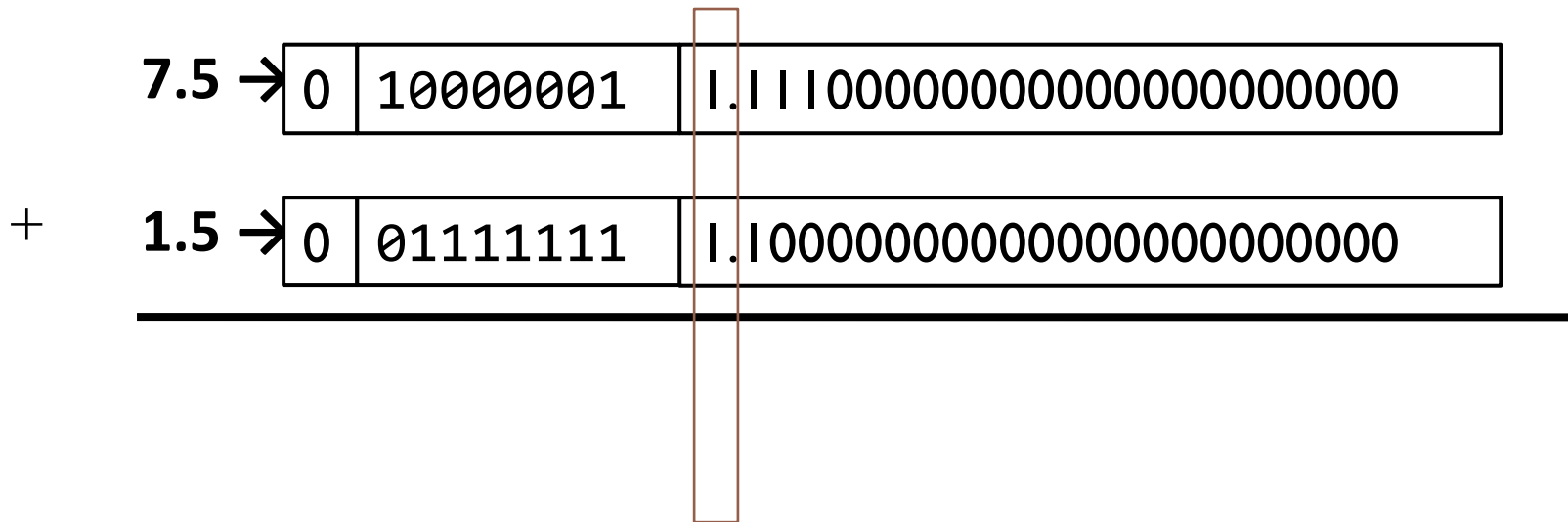
Solution

- Splitting exponents and mantissas, and adding implicit bit



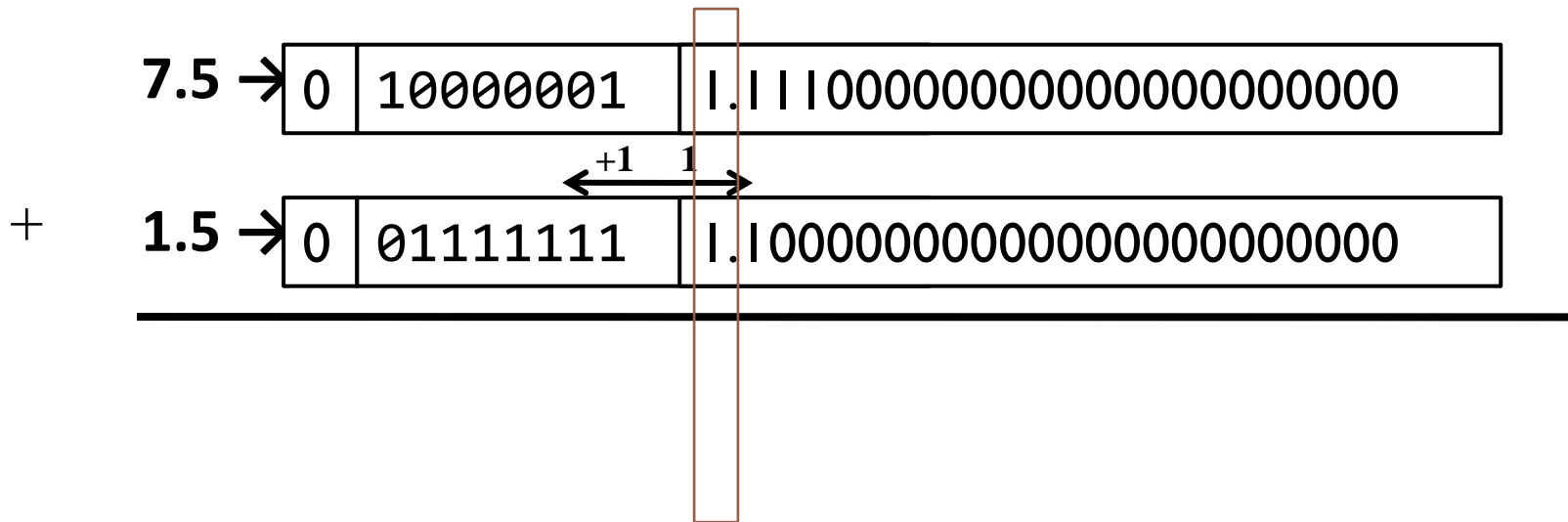
Solution

► Equalize exponents



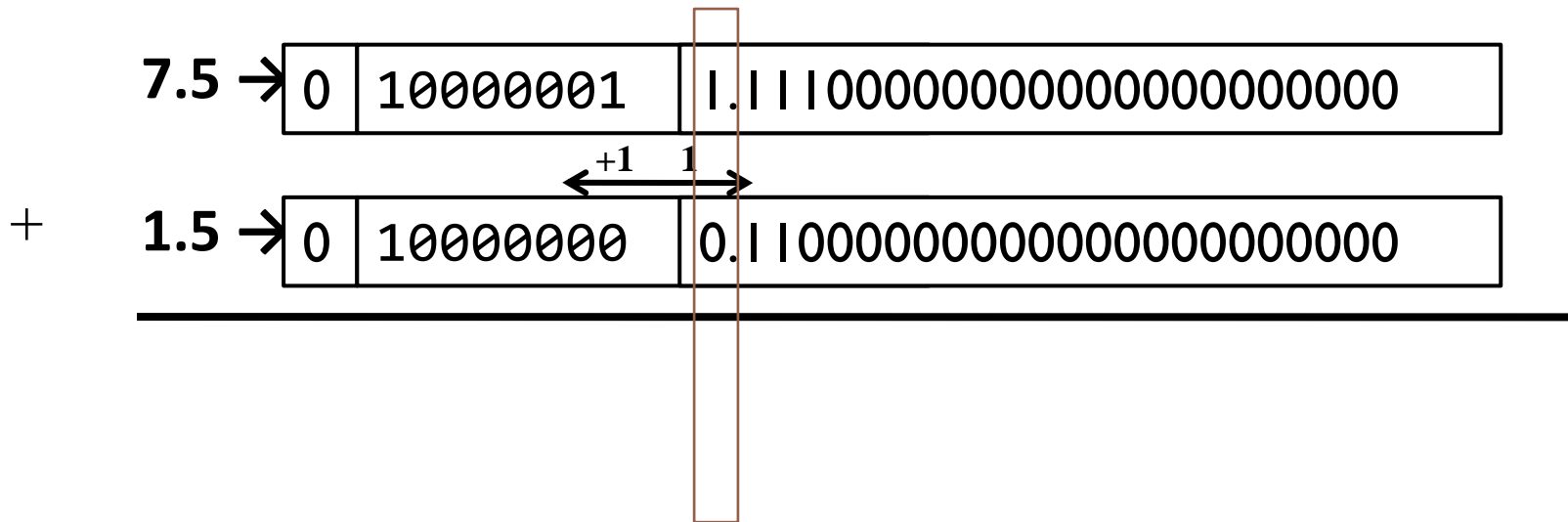
Solution

► Equalize exponents



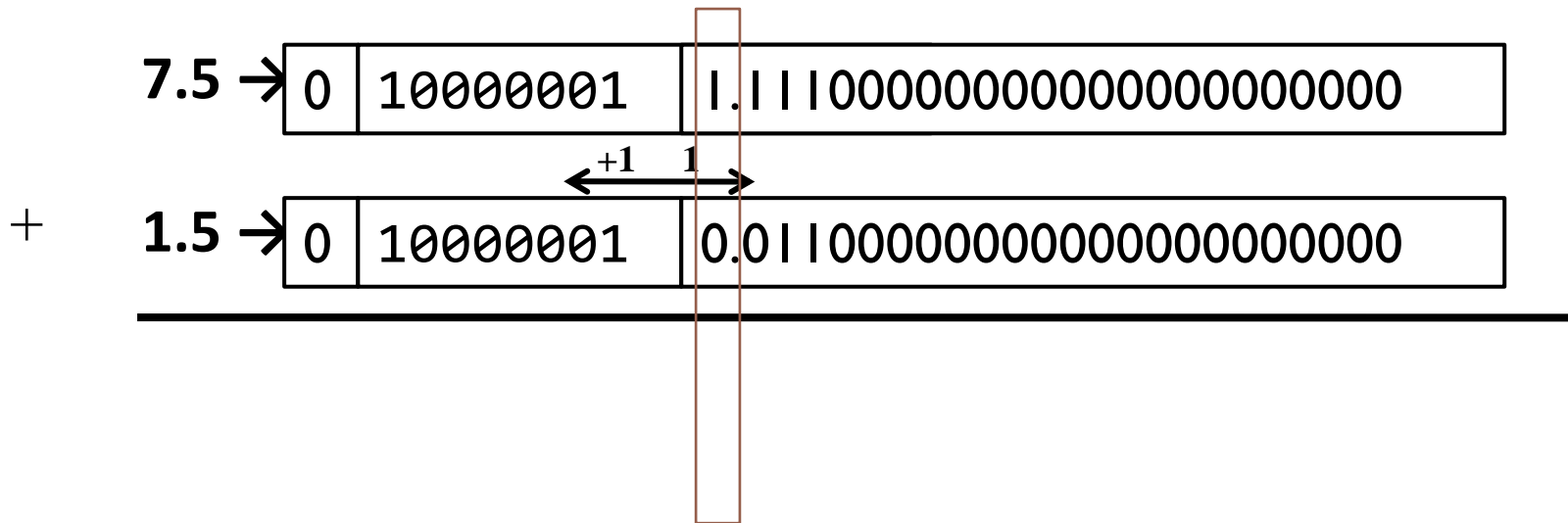
Solution

► Equalize exponents



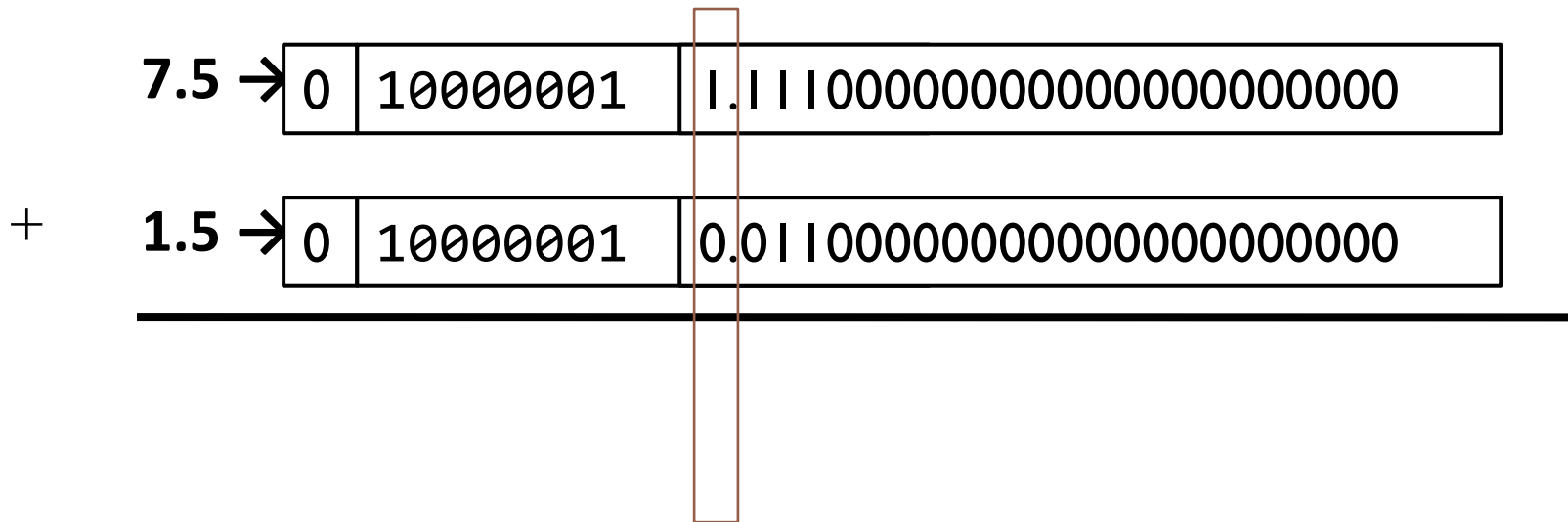
Solution

► Equalize exponents



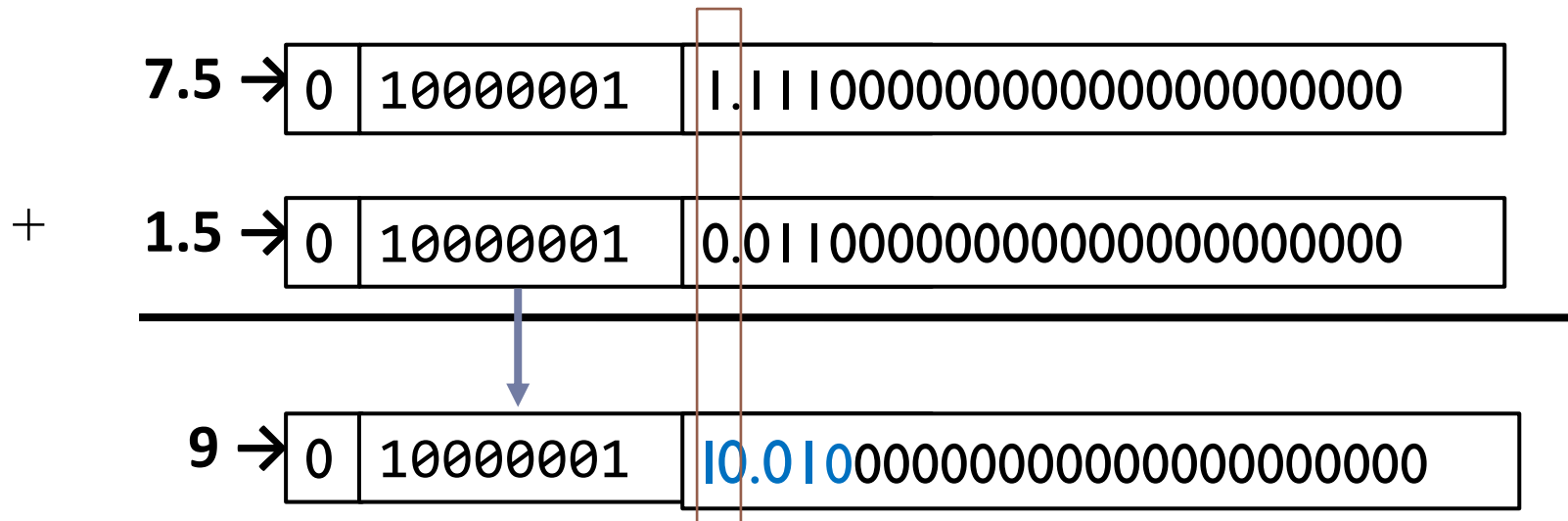
Solution

► Add mantissas



Solution

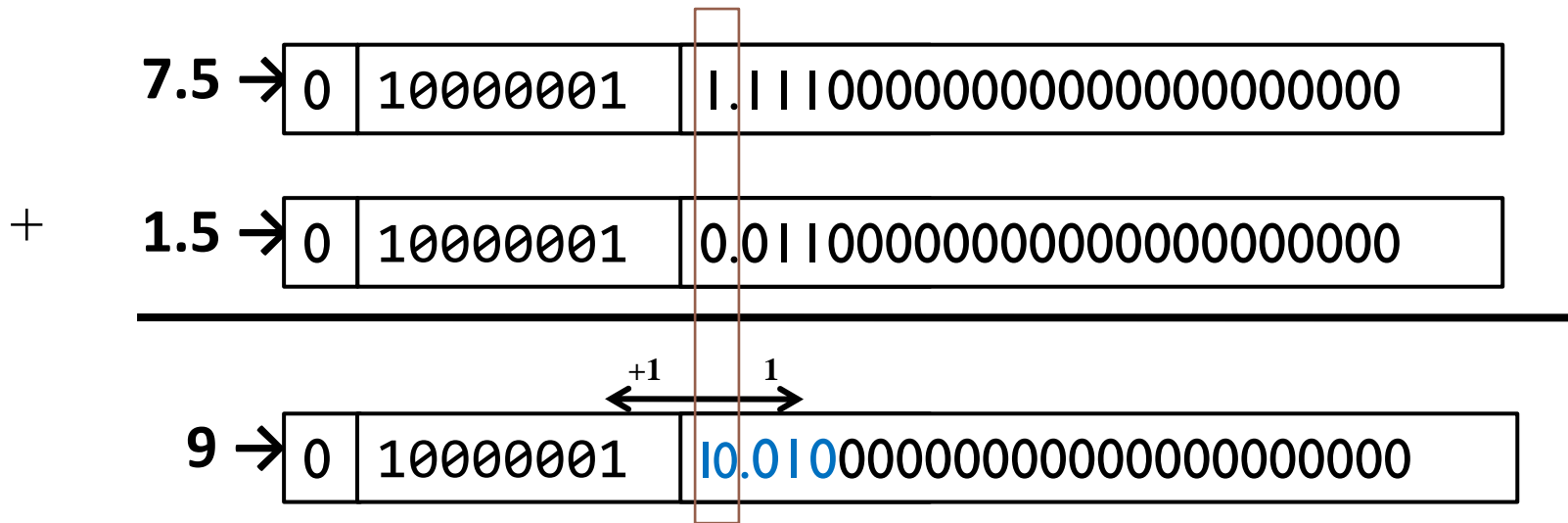
► Normalize result...



There is carry,
non-normalized mantissa

Solution

► **Normalize result...**



There is carry,
non-normalized mantissa

Solution

+

7.5 →	0	10000001	1.11100000000000000000000000000000
1.5 →	0	10000001	0.01100000000000000000000000000000
<hr/>			
9 →	0	10000010	1.00100000000000000000000000000000

Solution

- ▶ Eliminate the implicit bit and store the result

9 → 0 10000010 001000000000000000000000

Exercise

- ▶ Using the IEEE 754 format, multiply 7.5 and 1.5 step by step.

Solution

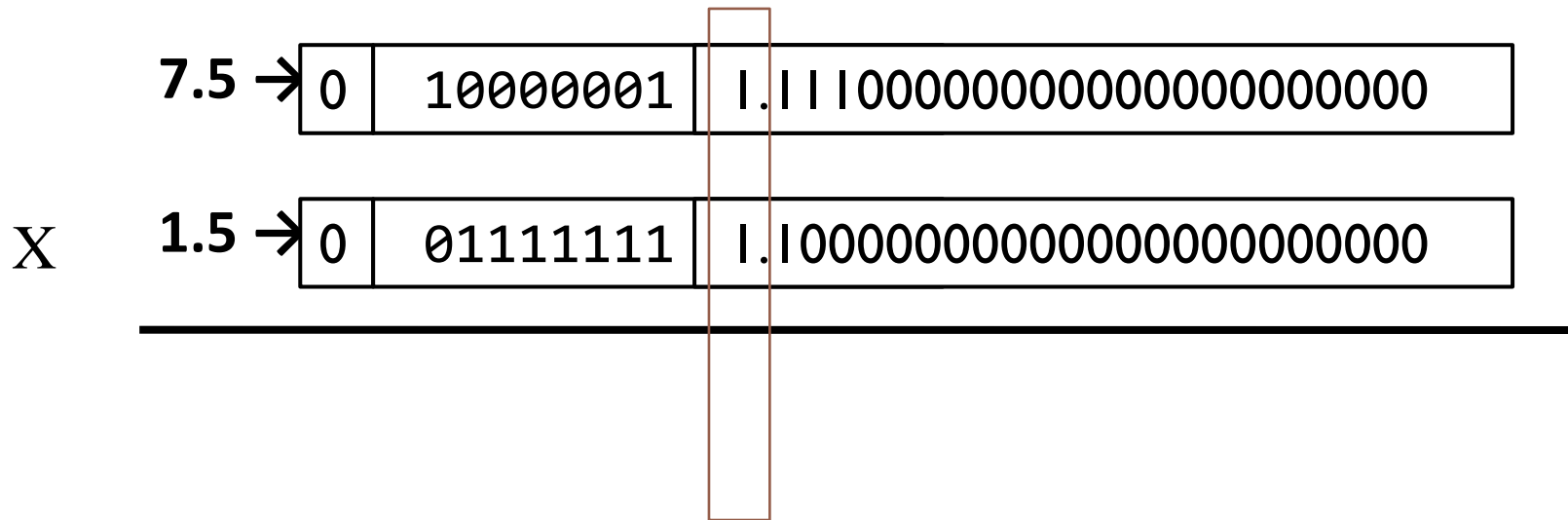
► Representation of the numbers

7.5 → 0 10000001 111000000000000000000000

1.5 → 0 01111111 110000000000000000000000

Solution

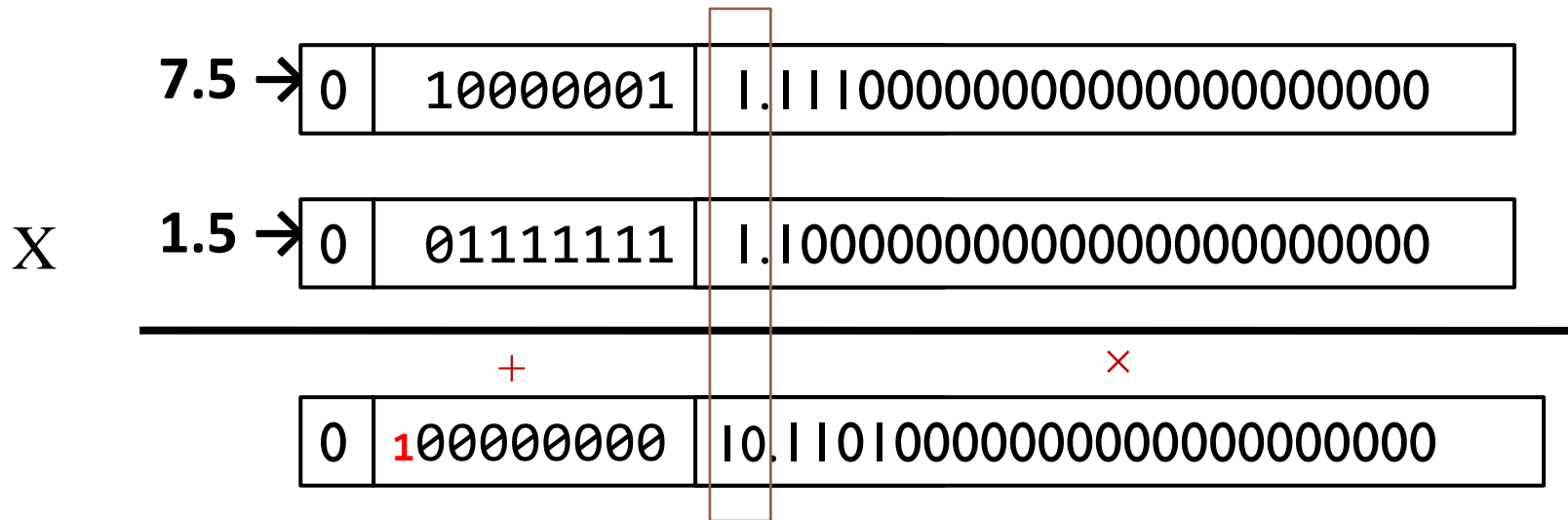
- Splitting exponents and mantissas, and adding implicit bit



The implicit bit is included

Solution

- ▶ Multiply: add exponents and multiply mantissas



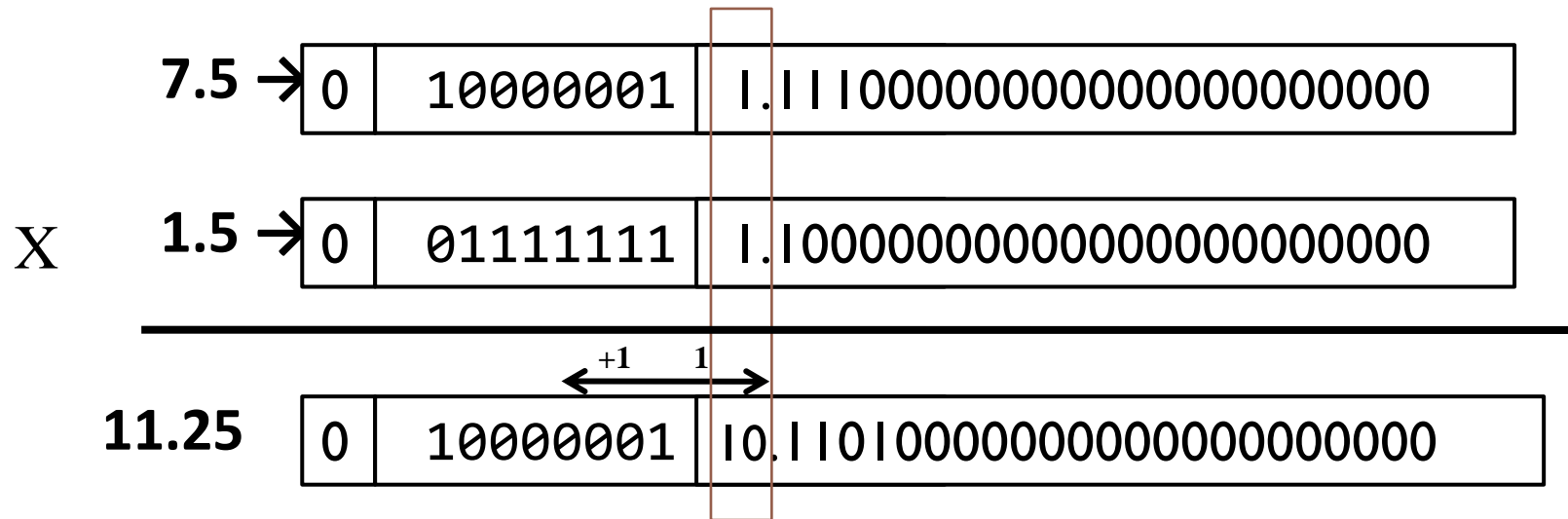
Solution

- ▶ Multiply: remove one bias from exponent (there are two)

$$\begin{array}{r} \text{7.5} \rightarrow \begin{array}{|c|c|c|} \hline 0 & 10000001 & 1.111000000000000000000000 \\ \hline \end{array} \\ \times \quad \text{1.5} \rightarrow \begin{array}{|c|c|c|} \hline 0 & 01111111 & 1.100000000000000000000000 \\ \hline \end{array} \\ \hline \begin{array}{|c|c|c|} \hline 0 & 100000000 & 10.110100000000000000000000 \\ \hline \end{array} \\ - \quad \begin{array}{|c|c|c|} \hline 0 & 01111111 & \\ \hline \end{array} \\ \hline \begin{array}{|c|c|c|} \hline 0 & 10000001 & 10.110100000000000000000000 \\ \hline \end{array} \end{array}$$

Solution

- Multiply: normalize result...



Solution

- ▶ Multiply: normalize result...

	7.5 →	0	10000001	1.111000000000000000000000
X	1.5 →	0	01111111	1.100000000000000000000000
<hr/>				
	11.25	0	10000010	1.011010000000000000000000

Solution

- ▶ Eliminate the implicit bit and store the result

11.25 0 10000010 011010000000000000000000

IEEE 754 Evolution

- ▶ 1985 – IEEE 754
- ▶ 2008 – IEEE 754-2008 (754+854)
- ▶ 2011 – ISO/IEC/IEEE 60559:2011 (754-2008)

Name	Common name	Base	Digits	E min	E max	Notes	Decimal digits	Decimal E max
binary16	Half precision	2	10+1	−14	+15	storage, not basic	3.31	4.51
binary32	Single precision	2	23+1	−126	+127		7.22	38.23
binary64	Double precision	2	52+1	−1022	+1023		15.95	307.95
binary128	Quadruple precision	2	112+1	−16382	+16383		34.02	4931.77
decimal32		10	7	−95	+96	storage, not basic	7	96
decimal64		10	16	−383	+384		16	384
decimal128		10	34	−6143	+6144		34	6144

http://en.wikipedia.org/wiki/IEEE_floating_point

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 2

Representation of information

Computer Structure
Bachelor in Computer Science and Engineering

