COMP2611: Computer Organization

Data representation

Overview

- You will learn the following in this tutorial:
  - data representations
    - Base conversions and two's complement
    - IEEE 754 single/double precision floating point numbers
    - ASCII encoding.

Data representation

Data representation

- base conversions, two's complement
- IEEE 754 floating point format,
- ASCII representation of characters

Examples

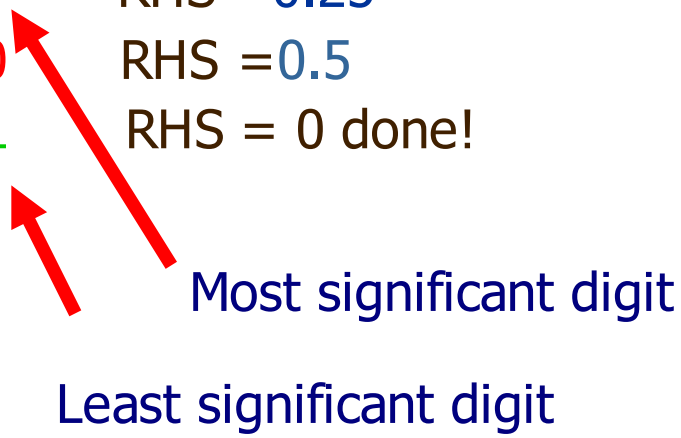
- Example : Convert $.625_{(10)}$ to the binary format:

  0.625 x 2 LHS of decimal pt=1 RHS =0.25

  0.25 x 2 LHS of decimal pt=0 RHS =0.5

  0.5 x 2 LHS of decimal pt=1 RHS = 0 done!

  $0.625_{(10)} => 0.101_{(2)}$



- Example : Convert $101.101_{(2)}$ to the decimal format:

  $(1x2^2) + (0x2^1) + (1x2^0) + (1x2^{-1}) + (0x2^{-2}) + (1x2^{-3}) = 5.625_{(10)}$

  $101.101_{(2)} => 5.625_{(10)}$

Two's complement representation

- Computers use two's complement representation scheme for signed numbers.
- The positive numbers are represented as before.
- The negative numbers are converted as follows:
  - Each bit in the number is inverted to get its one's complement,
  - Add 1 to the one's complement number to get the two's complement.
- Bit 31 is the sign bit. (for +ve numbers bit 31 is 0, for –ve numbers bit 31 is 1)
- Example : -5 ($0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101_{(2)}$)
  - Its 1's complement is

    $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{(2)}$
  - After adding 1 the 2's complement becomes

    $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{(2)}$

Two's complement representation

- To convert a two's complement number back to its decimal form, one needs to do the following:
  - Look at the sign bit, if sign bit is zero, convert the binary number directly to decimal format,
  - If the sign bit is one, invert the number and then add 1 to the inverted number. Convert the result to decimal format and put a –ve sign to the number
- Example : Convert the two's complement number $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{(2)}$ to decimal format
  - Sign bit =1, invert all the bits we have

    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{(2)}$
  - Add 1 to the inverted number we have

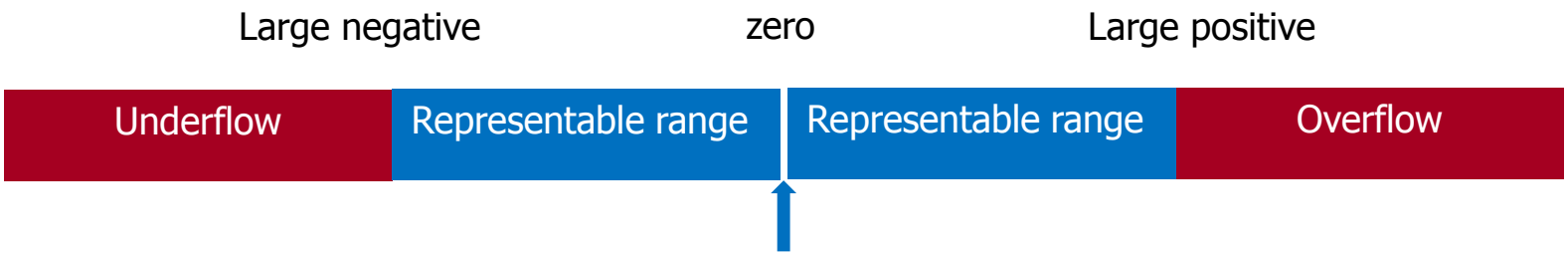
    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101_{(2)}$
  - Convert the result to decimal format and add a –ve sign

    $-5_{(10)}$

Overflow and Underflow in Signed Integer

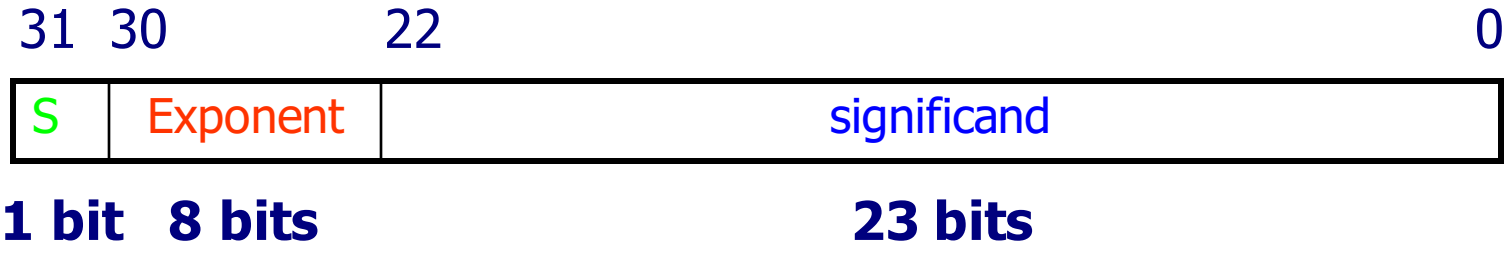
- **Overflow** (signed integer)
  - The value is bigger than the largest integer that can be represented

- **Underflow** (signed integer)
  - The value is smaller than the smallest integer that can be represented



The IEEE 754 single precision floating point format

- The IEEE 754 standard uses 32 bits to represent single precision floating point numbers.

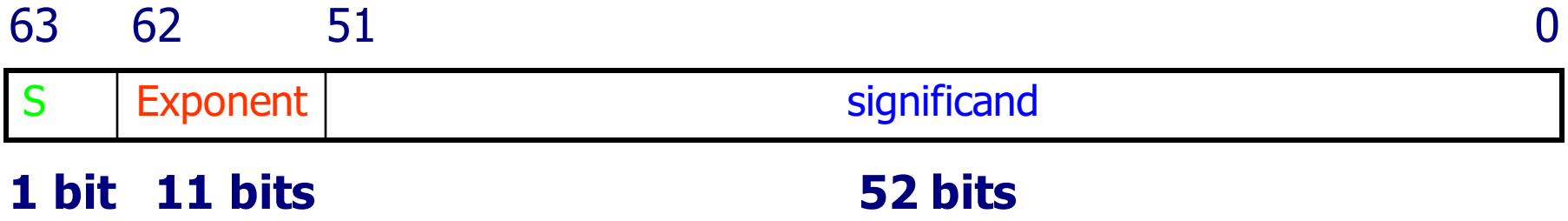


- S : sign bit (0 positive, 1 negative),
- Exponent : 8-bit field, bias = 127,
- Significant : 23-bit field.

Exercise: Convert $-5.625_{(10)}$ to the single precision floating point format:

The IEEE 754 double precision floating point format

- The IEEE 754 standard uses 64 bits to represent double precision floating point numbers.



- S : sign bit (0 positive, 1 negative),
- Exponent : 11-bit field, bias = 1023,
- Significant : 52-bit field.

Exercise: Convert $-5.625_{(10)}$ to the double precision floating point format:

IEEE 754 special cases

- IEEE 754 Single precision format:

Significand \ Exponent	0	1 - 254	255
0	0	$(-1)^S \times (1.F) \times (2)^{E-127}$	$(-1)^S \times (\infty)$
$\neq 0$	$(-1)^S \times (0.F) \times (2)^{-126}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

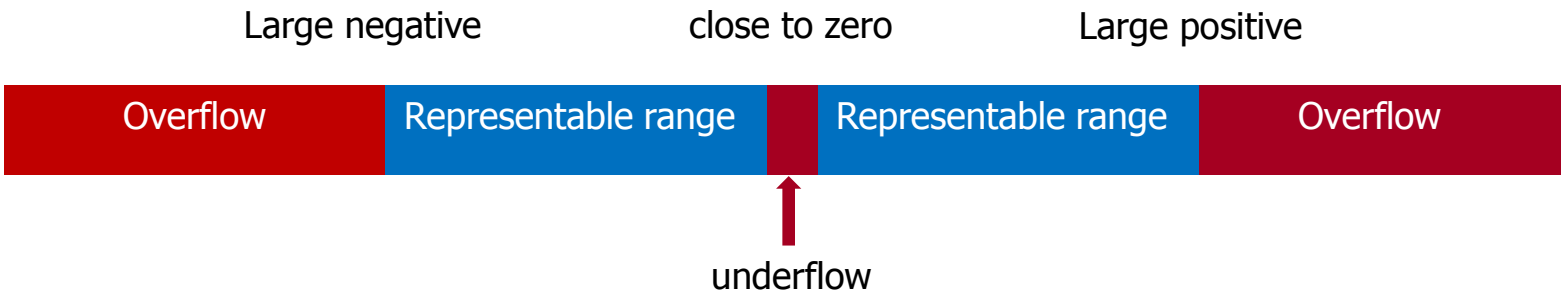
- IEEE 754 Double precision format:

Significand \ Exponent	0	1 - 2046	2047
0	0	$(-1)^S \times (1.F) \times (2)^{E-1023}$	$(-1)^S \times (\infty)$
$\neq 0$	$(-1)^S \times (0.F) \times (2)^{-1022}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

Overflow and Underflow in Floating Point

- **Overflow** (floating-point)
  - A positive exponent becomes too large to fit in the exponent field

- **Underflow** (floating-point)
  - A negative exponent becomes too large to fit in the exponent field



Representing text– the ASCII codes

- The American Standard Code for Information Interchange (ASCII) is a character encoding scheme for encoding text.
- ASCII code uses 8 bits to represent one character.
- The list of the first 128 characters are listed in the table below

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Data representation

Data representation

- base conversions, two's complement
- IEEE 754 floating point format,
- ASCII representation of characters

Examples

Exercises

Question 1: Given the bit pattern 1000 0000 0100 0110 0000 0000 0000 0000

- What is the value if this is a 2's complement representation?
- What if the pattern is an unsigned interger?
- What if it is an IEEE single precision number?
- What if it represents 4 ASCII characters (assume bits 31-24, 23-16, 15-8, 7-0 store the characters, and ASCII value of 128 is the symbol '€').

Exercises

Question 2: Assume the bit pattern 1001 1100 follows the IEEE-like floating point representation format

1 bit 3 bits 4 bits

- What is the bias of the exponent?
- What value is the given pattern representing?
- What is the range of numbers that this IEEE-like floating point representation system can represent?
- What is the granularity of this representation system?

Conclusion

- Today we have reviewed:
  - simple base conversions, the IEEE 754 floating point format, and the ASCII character scheme.