UCCD1133

Introduction to Computer Organisation and Architecture

Chapter 2

Number Systems and Data Representation

Disclaimer

- This slide may contain copyrighted material of which has not been specifically authorized by the copyright owner. The use of copyrighted materials are solely for educational purpose. If you wish to use this copyrighted material for other purposes, you must first obtain permission from the original copyright owner.

Chapter 2-1

Number systems

Outline

- Count in binary, octal and hexadecimal number systems.
- Understand the weighting structure of numbers.
- Perform base conversion of various number systems.
- Carry out arithmetic operations with binary numbers.

Number System

- Digital systems process data in binary format.
  - Need to represent information in binary
  - For manipulation in electronic hardware
- A number system consists of an ordered set of symbols called digits.
  - Total number of digits = base (radix, R) of a number system
  - The range of numbers is 0 to (R - 1).
  - The number can have 2 parts:
    - Integer
    - Fractional
    - Separated by a **radix point** (.)
    - E.g. $(a_{n-1}\ a_{n-2} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-m})_R$

- Commonly used number systems in digital electronics field of study
  - Decimal number system:
    - Digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    - R = 10.
  - Binary number system:
    - Digits {0, 1}
    - R = 2.
  - Hexadecimal number system:
    - Digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
    - R = 16.
  - Octal number system:
    - Digits {0, 1, 2, 3, 4, 5, 6, 7}
    - R = 8.

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10

Counting in Binary

- How to count in binary?

0	First column is full
1	Reset first column and add 1 to second column
10	First two columns are full
11	Reset and add 1 to third column
100	
101	
110	
111	First three columns are full
1000	Reset and add 1 to fourth column
1001	

- With N bits, can count up to (2^N – 1) for a total of 2^N different numbers

Weighting Structure of Numbers

- A positive number, N written in **positional notation**

$$N = (a_{n-1}\ a_{n-2} \ldots a_1\ a_0 \,.\, a_{-1}\ a_{-2} \ldots a_{-m})_R$$
$$= a_{n-1} R^{n-1} + a_{n-2} R^{n-2} + \ldots + a_1 R^1 + a_0 R^0 + a_{-1} R^{-1} + a_{-2} R^{-2} + \ldots + a_{-m} R^{-m}$$

Where

. = radix point. E.g. binary point, hex point, decimal point.

R = radix or base - any positive integer where $R > 1$

n = number of integer digits (left of radix pt)

m = number of fractional digits (right of radix pt)

a_{n-1} = most significant digit (MSD)

a_{-m} = least significant digit (LSD)

- Each digit's position is multiplied by a weighting factor (an integral power of R depending on the position)
  - Example, $N = 251.41_{10}$.
    $= (2 \times 10^2) + (5 \times 10^1) + (1 \times 10^0) + (4 \times 10^{-1}) + (1 \times 10^{-2})$

10^{n-1}	.....	10^2	10^1	10^0	.	10^{-1}	10^{-2}	.....	10^{-n}
		2	5	1	.	4	1		

Weighting Structure of Numbers

- Example, binary weight structure
  $2^{n-1} \ldots 2^2\ 2^1\ 2^0 . 2^{-1}\ 2^{-2} \ldots 2^{-n}$

Positive Powers of 2 (Integer Number)								Negative Powers of 2 (Fractional number)					
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64

- Right-most bit is the **LSB** (Least Significant Bit).

- Left-most bit is the **MSB** (Most Significant Bit).

Base Conversion:
Convert Base N to Decimal (Base 10)

- Example 1

  Convert 1010.101_2 to $?_{10}$.

  Solution 1

  $1010.101_2 = (1 \text{ X } 2^3) + (0 \text{ X } 2^2) + (1 \text{ X } 2^1) + (0 \text{ X } 2^0) + (1 \text{ X } 2^{-1}) + (0 \text{ X } 2^{-2}) + (1 \text{ X } 2^{-3})$

  $= 8_{10} + 0 + 2_{10} + 0 + 0.5_{10} + 0 + 0.125_{10}$

  $= 10.625_{10}$

- Example 2

  Convert 627_8 to $?_{10}$.

  Solution 2

  $627_8 = (6 \text{ X } 8^2) + (2 \text{ X } 8^1) + (7 \text{ X } 8^0)$

  $= 384_{10} + 16_{10} + 7_{10}$

  $= 407_{10}$

Base Conversion:
Convert Integer Decimal (Base 10) to Base N

- Example 1: Method 1

Convert 49_{10} to a binary number.

Solution 1

49 – 32 = 17

17 – 16 = 1

1 – 1 = 0 (conversion completed)

64	32	16	8	4	2	1
	1	1	0	0	0	1

Therefore, $49_{10} = 110001_2$

- Example 2: Method 2 - Divide-by-radix

Convert 49_{10} to a binary number.

Solution 2

49 / 2 = 24 remainder 1 (LSB)

24 / 2 = 12 remainder 0

12 / 2 = 6 remainder 0

6 / 2 = 3 remainder 0

3 / 2 = 1 remainder 1

1 / 2 = 0 remainder 1 (MSB)

Therefore, $49_{10} = 110001_2$

Base Conversion:
Convert Base N to Base M

- Requires an intermediate conversion step
  - First convert base N to base 10
  - Then base 10 to base M.

- Example 1:

  Convert 18_9 to $?_{11}$.

Solution 1

Convert 18_9 to base 10:

$18_9 = (1 \text{ X } 9^1) + (8 \text{ X } 9^0)$

$= 9 + 8$

$= 17_{10}$

Convert from base 10 to base 11 using the divide-by-radix method:

17 / 11 = 1 remainder 6

1 / 11 = 0 remainder 1

Therefore, $18_9 = 16_{11}$

- Example 1:

  Convert 1011011_2 to base 8. 2, 4, 8, 16

  Solution 1

  Since $8 = 2^3$, we can group three binary digits for each octal digit.

  Therefore, $1\_011\_011_2 = 133_8$

  1 3 3

- Example 2

  Convert AF_{16} to base 8.

  Solution 2

  Since 16 is not a power of base 8. But both bases are a power of 2.

  Therefore, we can convert AF_{16} to base 2 and then convert to base 8.

  $16 = 2^4$

  Firstly, each hex digit is replaced by 4 bin digits.

  $AF_{16} = 1010\_1111_2$

  0 10 10 5 1 111 2 7 2

  Then convert the bin number to base 8.

  $10\_101\_111_2 = 257_8$

  Therefore, $AF_{16} = 257_8$.

Binary Arithmetic

- 4 basic types of binary arithmetic:
  - Binary addition
  - Binary subtraction
  - Binary multiplication
  - Binary division

Binary Addition

- Example 1

  Add 11110 and 1100.

  Solution 1

	1	1	1	1	0	Augend
1←	1←	1←	0←	0←	0←	Carries
+	0	1	1	0	0	Addend
	0	1	0	1	0	Sum

- When two n-bit numbers are added, the result is a (n+1)-bit number

- How does the above information relate to circuit?
  - Understand the process of binary addition leads to constructing an adder circuit for each bit
  - Then combine the adders to compute two n-bit numbers
  - Need to capture the info into an intermediate form first e.g. truth table before proceed to adder circuit design.

Inputs			**Outputs**	
Carry-in	**A**	**B**	**Carry-out**	**Sum**
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

Chapter 2-2

Computer codes

Outline

- Understand the concept of computer codes.
- Recognise the commonly used digital codes such as ASCII, Binary Coded Decimal (BCD) and Gray code.
- Express decimal numbers in BCD form.
- Convert between the binary system and the Gray code.

Computer Codes

- A code is the use of a number system for representing information

- Binary system is widely used as codes to represent numbers, text, pictures, etc.
  - Since computers process all info in binary – the most efficient electronic form.

- Several types of well-known codes.
  - Codes to represent numeric.
  - Codes to represent character.
  - Codes for detecting and correcting errors.
  - Codes for serial data transmission and storage.

Binary Coded Decimal (BCD)

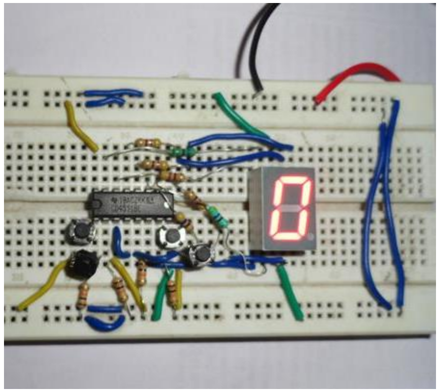
- BCD code is used to represent the decimal digits 0 - 9.
  - Always 4 bits.
  - BCD is a type of *weighted code*
    - Each bit position is **weighted** with 8, 4, 2, 1 from MSB to LSB
    - Hence, also called 8-4-2-1 code

- Note that 1010, 1011, 1100, 1101, 1110 and 1111 are not used.
  - They are invalid in 8421 BCD code.

- Example of applications:
  - Used to encode numbers for output to numerical displays
  - Used in processors that performs decimal arithmetic.

- BCD system is not binary system.
  They are not the same.

Decimal Digit	BCD code (8-4-2-1 Code)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1



< BCD to 7 segment display

Binary Coded Decimal (BCD)

- Example 1:

  Encode the decimal number N = 9750_{10} in BCD.

  Solution 1

  9 -> 1001

  7 -> 0111

  5 -> 0101

  0 -> 0000

  Then the individual codes are concatenated to give

  $9750_{10} = 1001011101010000_{BCD}$

  In binary system, $9750_{10} \Rightarrow 10011000010110_2$

Gray Code

- Gray code is unweighted
  - Cannot be used for arithmetic operations
  - Commonly used in data transfer applications
  - Gray code can be any length

Example, a 4-bit Gray code

Gray code G[3:0]	Binary B[3:0]	Decimal Value
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

- Gray code important feature:
  - Its natural sequence exhibits a single bit change between successive code words.

- Useful for reducing error rate in data transfer applications
  - Lesser bit change while data is being transferring – chances of wrong bit transfer to the other side is lesser.

How to Generate Gray Code

How to ensure only one bit changes between two adjacent numbers?

1	2	3	4	5		
0	0	00	00	000	000	0000
1	1	01	01	001	001	0001
	1	11	11	011	011	0011
	0	10	10	010	010	0010
			10	110	110	0110
			11	111	111	0111
			01	101	101	0101
			00	100	100	0100
					100	1100
					101	1101
					111	1111
					110	1110
					010	1010
					011	1011
					001	1001
					000	1000

<u>From 2 rows to 4 rows</u>

1. Write the first 2 rows
2. Flip 2 rows vertically
3. Set the 2nd bit of the two new rows to 1

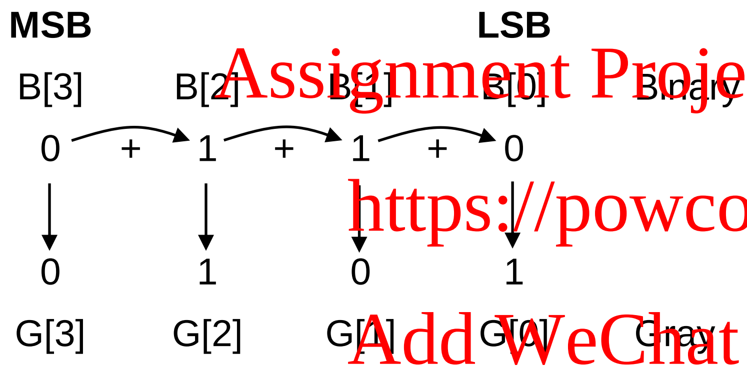
<u>From 4 rows to 8 rows</u>

4. Generate the new 4 rows by flipping the first 4 rows vertically
5. Set the 3rd bit of the new four rows to 1

Repeat the same procedure for the remaining rows

Gray Converters: Binary-to-Gray Converter

- In terms of algorithm:
  - The MSB in the Gray code is the same as the MSB in the binary code.
  - From left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.
  - E.g.



Gray Converters: Gray-to-Binary Converter

- In terms of algorithm:
  - The MSB in the binary code is the same as the MSB in the Gray code.
  - Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.
  - E.g.

MSB			**LSB**	
G[3]	G[2]	G[1]	G[0]	Gray
1	0	1	1	
1	1	0	1	
B[3]	B[2]	B[1]	B[0]	Binary

ASCII (American Standard Code for Information Interchange)

- ASCII is a set of binary codes
- Used to represent the alphanumeric symbols.
- Widely used in data communication.

- An extra bit (parity bit) is attached to an ASCII code during transmission for error detection purpose

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

ASCII (American Standard Code for Information Interchange)

- Example 1:
  ASCII code representation of the word "Digital" is shown

Character	Binary Code	Hexadecimal Code
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

Chapter 2-3

Signed and unsigned numbers

Outline

- Represent signed binary numbers in sign-magnitude and 2’s complement format.

Unsigned versus Signed Number Representation

- So far, we have considered only *unsigned* binary numbers that represent positive range of numbers.

- Sometimes programmers want to deal with both negative and positive range of numbers, which requiring a different binary number system.

- Signed binary numbers can be represented using:
  - Sign-magnitude number system.
  - Two's complement number system.

- You will probably deal with the terms:
  - *zero extension* (for unsigned numbers)
  - *sign extension* (for signed numbers)

Sign-Magnitude Binary Number System

- A number, N is written as:
  - $N = (s\ a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ \ldots\ a_{-m})_{rsm}$
    where s => sign-bit;
    a => magnitude
    - s = 0: N is positive.
    - s = 1: N is negative.

- N-bit number correspond to decimal range: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$.

- Example
  Determine the binary sign-magnitude code for N = -13, expressed as a 5-bit binary number.

  Solution:

  $$\begin{aligned} \mathbf{N} &= \mathbf{-13} \\ &= \mathbf{-1101_2} \\ &= \mathbf{1}\underbrace{\mathbf{1101}}_{}\mathbf{_{2sm}} \end{aligned}$$

  To represent negative sign

  Sign-magnitude code to represent -13

Decimal Number System	Sign-Magnitude Number System (4-bit system)
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111
-8	-

2 possible representations of zero
- can complicate testing for zero - problems for programmers.

Sign-Magnitude Binary Number System

- The implementation of the method is costly in terms of:
  - Circuitry (large arithmetic circuit)
  - Computation time (takes more time to produce the result).

- It is slightly odd to have both +0 and -0 exist.

- Ordinary binary addition does not work. For example, $-5_{10} + 5_{10}$ gives $1101_2 + 0101_2 = 10010_2$, which is nonsense.

- Not popular in practice for representing signed numbers.

2's Complement Number System (2cns)

- 2cns consists of:
  - Positive range
    - $0 \leq N \leq (2^{n-1} - 1)$
    - Same as sign-magnitude number system
  - Negative range
    - $-1 \geq N \geq (-2^{n-1})$
    - using complement of the negative numbers

E.g. an 8-bit digital system can only operate on values between -128 to 127.

- Why use 2cns?
  - Like binary system, natural numbering system for digital circuits.
  - The codes are arranged in such a way that is easy to build hardware to perform signed arithmetic operations (through complement arithmetic method).

Decimal Number System	Sign-Magnitude Number System	2's Complement Number System
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
-0	1000	-
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	-	1000

The MSB is sign bit.
0 => pos numbers.
1 => neg numbers.
For negative numbers, the rest of the bits DO NOT represent magnitude

E.g. does not represent -6

Method to Obtain 2's Complement Representation

- Algorithm to obtain the 2's complement of N:
  1. Find the binary equivalent of N.
  2. Complement each bit (that is, invert 0's to 1's and 1's to 0's).
  3. Add 1.

- Example 1

  Find the 2's complement of -2_{10} as a 4-bit two's complement number.

  Solution
  1. $+2_{10}$ is 0010_2.
  2. Inverting 0010_2 produces 1101_2.
  3. $1101_2 + 1 = 1110_{2c}$.

  So -2_{10} is 1110_{2c}.

- Example 2

  Find the 2's complement of N = -110_2 if a 5-bit number is used.

  Solution

  $$\begin{array}{lrl} N_2 & 0\ 0\ 1\ 1\ 0_2 & \\ \hline & 1\ 1\ 0\ 0\ 1 & \leftarrow \text{Invert (1's complement)} \\ & +\ 1 & \leftarrow \text{Add 1} \\ \hline [N]_2 & 1\ 1\ 0\ 1\ 0_{2c} & \text{(2's complement)} \end{array}$$

2's Complement Representation

- Example 3

  Given an 8-bit digital system, find the decimal number value of N = $1111\ 1010_{2c}$.
  Also find the equivalent sign-magnitude representation.

- Solution

  First, check if the number is negative or positive by looking at the sign bit. If it is positive, simply convert it to decimal. If it is negative, inverts the bits and add a 1.

From the MSB (sign bit), N is a negative number.

$[1111\_1010]_{2c} = 0000\_0101_2 + 1$
$= 0000\_0110_2$

$$
\begin{array}{r l}
1\,1\,1\,1\,1\,0\,1\,0_{2c} & \\
\hline
0\,0\,0\,0\,0\,1\,0\,1 & \leftarrow \textbf{Invert} \\
+\,1 & \leftarrow \textbf{Add 1} \\
\hline
0\,0\,0\,0\,0\,1\,1\,0_2 &
\end{array}
$$

1111_1010_{2c} represents -0000_0110_2 or -6.

The equivalent sign-mag representation of -6 = 1000_0110_{2sm}.

Example

- Example 4

Convert the following numbers to the required number system. Show your steps to obtain the answers.

(i) Convert DA_{16} to binary, then to decimal

(ii) Find the 2's complement for $-0101\ 1101_2$ (answer in 8-bit).

(iii) Find the BCD equivalence for 85_{10}

(iv) Convert $0110\ 1100_{Gray}$ to binary (answer in 8-bit).

(v) Convert $1110\ 0100_2$ to Gray code (answer in 8-bit).

Solution

(i) $DA_{16} = 1101\ 1010_2$

$= (2^7 + 2^6 + 2^4 + 2^3 + 2^1)_{10}$

$= 218_{10}$

(ii) $0101\ 1101_2$

1's complement = 1010 0010

2's complement = 1010 0011

(iii) $85_{10} = 1000\ 0101_{BCD}$

(iv) $0110\ 1100_{Gray} = 0100\ 1000_2$

(v) $1110\ 0100_2 = 1001\ 0110_{Gray}$

Floating point number systems*

- Floating-point number system allows the representation of very large and very small numbers with a *sign, mantissa (M), base (B), and exponent (E).*

$$\pm M \times B^E$$

- Example: $4000 = 4 \times 10^3$

- A single precision floating point value is represented by 32 bits: 1 sign bit, 8 exponent bits , and 23 mantissa bits.

- Example:

  Floating-point representation of decimal number 228.

- Solution:
  - Convert decimal to binary.

$228_{10} = 11100100_2 = 1.11001_2 \times 2^7$

Answer:

1 bit	8 bits	23 bits
0	00000111	111 0010 0000 0000 0000 0000
Sign	**Exponent**	**Mantissa**

In reality, floating point representation follows the following form:

IEEE 754 floating-point standard

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,
Double Precision Bias = 1023.