CSEE 3827: Fundamentals of Computer Systems, Spring 2011

1. Number Representation

Prof. Martha Kim (martha@cs.columbia.edu)
Web: http://www.cs.columbia.edu/~martha/courses/3827/sp11/

Contents (H&H 1.3-1.4, 5.3)

- Digital Information Representation
  - Decimal
  - Hexadecimal
  - BCD
- Terminology:
  - Bit / Byte / Words
  - Highest Order (most significant) Bit, Lowest Order (least significant) bit
- Negative Number Formats:
  - Signed Magnitude
  - 1's Complement
  - 2's Complement
- Fractions via Binary
  - Fixed Point
  - Floating Point

Number systems: Base 10 (Decimal)

- 10 digits = {0,1,2,3,4,5,6,7,8,9}
- example: 4537.8 = $(4537.8)_{10}$

$$\begin{array}{ccccccccc} 4 & & 5 & & 3 & & 7 & . & 8 \\ \times 10^{3} & & \times 10^{2} & & \times 10^{1} & & \times 10^{0} & & \times 10^{-1} \\ \hline 4000 & + & 500 & + & 30 & + & 7 & + & .8 \end{array} = 4537.8$$

Number systems: Base 2 (Binary)

- 2 digits = {0,1}
- example: 1011.1 = $(1011.1)_2$

$$\begin{array}{ccccccccc} 1 & & 0 & & 1 & & 1 & . & 1 \\ \times 2^3 & & \times 2^2 & & \times 2^1 & & \times 2^0 & & \times 2^{-1} \\ \hline 8 & + & 0 & + & 2 & + & 1 & + & .5 \end{array} = (11.5)_{10}$$

Number systems: Base 8 (Octal)

- 8 digits = {0,1,2,3,4,5,6,7}
- example: $(2365.2)_8$

$$
\begin{array}{ccccccccc}
2 & & 3 & & 6 & & 5 & . & 2 \\
\times 8^{3} & & \times 8^{2} & & \times 8^{1} & & \times 8^{0} & & \times 8^{-1} \\
\hline
1024 & + & 192 & + & 48 & + & 5 & + & .25
\end{array}
= (1269.25)_{10}
$$

Number systems: Base 16 (Hexadecimal)

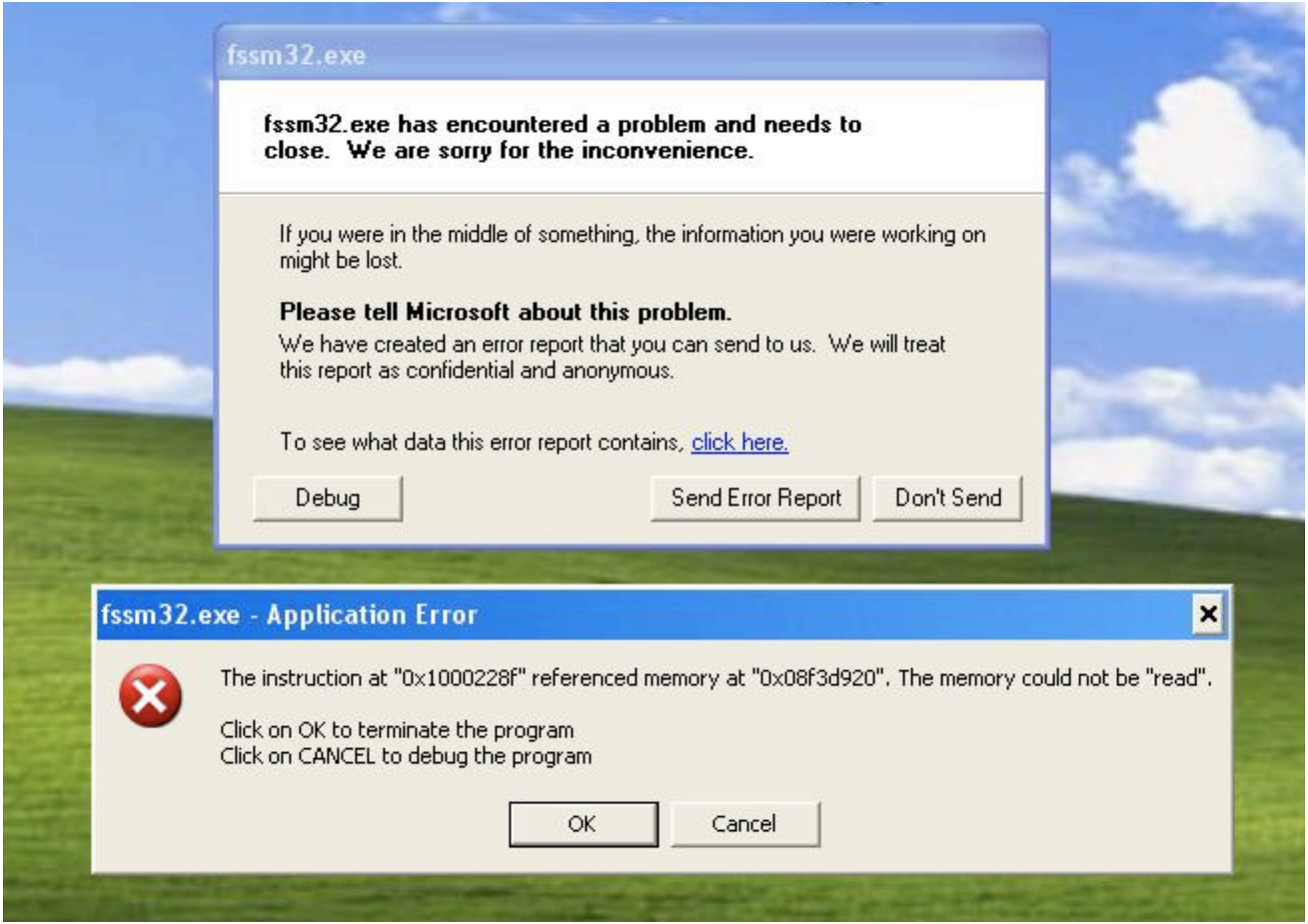
- 16 digits = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
- example: $(26BA)_{16}$ [alternate notation for hex: 0x26BA]

2	6	B	A
$\times 16^3$	$\times 16^2$	$\times 16^1$	$\times 16^0$

$$8192 + 1536 + 176 + 10 = (9914)_{10}$$

Why Important: More concise than binary, but related (a power of 2)

Hexadecimal (or hex) is often used for addressing



Number ranges

- Map infinite numbers onto finite representation for a computer
- How many numbers can I represent with ...

  ... 5 digits in decimal? 10^5 possible values

  ... 8 binary digits? 2^8 possible values

  ... 4 hexadecimal digits? 16^4 possible values

Computer from Digital Perspective

- Information: just sequences of binary (0’s and 1’s)
  - True = 1, False = 0
- Numbers: converted into binary form when “viewed” by computer
  - e.g., 19 = 10011 (16 (1) + 8 (0) + 4 (0) + 2 (1) + 1 (1)) in binary
- Characters: assigned a specific numerical value (ASCII standard)
  - e.g., ‘A’ = 65 = 1000001, ‘a’ = 97 = 1100001
- Text is a sequence of characters:
  - “Hi there” = 72, 105, 32, 116, 104, 101, 114, 101

    = 1001000, 1101001, ...

Terminology: Bit, Byte, Word

- bit = a binary digit e.g., 1 or 0

- byte = 8 bits e.g., 01100100

- word = a group of bits that is **architecture dependent**

  (the number of bits that an architecture can process at once)

  a 16-bit word = 2 bytes e.g., 1001110111000101

  a 32-bit word = 4 bytes e.g., 10011101110001010111011100010 1

OBSERVATION: computers have bounds on how much input they can handle at once ⟶ limits on the sizes of numbers they can deal with

Terminology: MSB, LSB

- Bit at the left is highest order, or most significant bit (MSB)
- Bit at the right is lowest order, or least significant bit (LSB)

MSB LSB

e.g., 1001110111000101

- Common reference notation for k-bit value: $b_{k-1}b_{k-2}b_{k-3}...b_1b_0$

Unsigned numbers *a.k.a. Binary Coded Decimal (BCD)*

Binary numbers represent only non-negative (positive or 0) values

BCD where wordsize=3:

value	BCD
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Addition of binary (unsigned numbers)

Like decimal addition, except:

- 1+1 = 0 with a carry of 1,
- 1+1+1 = 1 with carry of 1
- e.g., wordsize = 5, add 11110 and 10101 (30 + 21)

Overflow

when result cannot fit within the wordsize constraint

$$\begin{array}{rcl} 1\;1\;1 & & \\ 1\;1\;1\;1\;0 & & 30 \\ +\;\;1\;0\;1\;0\;1 & & 21 \\ \hline 1\;0\;0\;1\;1 & & 19 \end{array}$$

e.g., the "correct" answer 110011 requires 6 bits: cannot be represented with only 5 bits in unsigned representation

What about negative numbers?

Given a fixed wordsize how do you represent both positive and negative numbers?

Have certain bit combinations represent negative numbers

- e.g., Signed Magnitude
  - highest order bit (b_{k-1}) indicates sign: 0 = positive, 1 = negative
  - remaining bits indicate magnitude
    - e.g., 0011 = 3
    - e.g., 1011 = -3
    - e.g., 1000 = 0000 = 0
- Positive #'s have same form in both signed magnitude and unsigned
- Easy for humans to interpret, but not easiest form for computers to do addition/subtraction operations

Negative Numbers: 1's Complement Representation

- Non-negative #'s have same representation as unsigned (and signed-mag)
- To negate a #, flip all bits (not just highest-order as in signed-mag)
- e.g., wordsize = 4
  - 0010 = 2
  - 1101 = -2

Suppose wordsize is 8, what is the value of 11101011 when it represents a # in 1's Complement representation?

Let x=11101011; Know X is negative because MSB=1.

Negate X by flipping all bits: -X = 00010100

-X = 20, so X = -20

Note: in 1's complement, there are two ways to represent 0: all 0s and all 1s

Negative Numbers: 2's Complement Representation

- Non-negative #'s have same representation as unsigned (and signed-mag)
- To negate a #, flip all bits and add 1
- e.g., wordsize = 4
  - 0010 = 2, so 1101 + 1 = 1110 = -2
  - 0110 = 6, so 1001 + 1 = 1010 = -6
  - 1010 = -6, so 0101 + 1 = 0110 = 6 (works in both directions)
  - 0000 = 0, so 1111 + 1 = 0000 = 0 (0 is unique in 2's complement)

Note: negation works both ways in all cases except 1 followed by all 0s (e.g., 1000).

for wordsize=k, the value is -2^{k-1} (e.g., k=4, value is -8)

Note: the positive value of 2^{k-1} is not expressible

Number encoding summary

	BCD	Sign&Mag.	1s Comp.	2s Comp.
0 0 0	0	+0	+0	+0
0 0 1	1	+1	+1	+1
0 1 0	2	+2	+2	+2
0 1 1	3	+3	+3	+3
1 0 0	4	0	-3	-4
1 0 1	5	-1	-2	-3
1 1 0	6	-2	-1	-2
1 1 1	7	-3	0	-1
	8 values	*7 values, 2 zeroes*	*7 values, 2 zeroes*	*8 values, 1 zero*

k-bit Words & Ranges of various formats

- Given a k-bit word, what range of numbers can be represented as:
  - unsigned: 0 to $2^k - 1$ (e.g., k=8, 0 to 255)
  - signed mag: $-2^{k-1} + 1$ to $2^{k-1} - 1$ (e.g., k=8, -127 to 127 [2 vals for 0])
  - 1’s complement: same as signed mag (but negative numbers are represented differently)
  - 2’s complement: -2^{k-1} to $2^{k-1} - 1$ (e.g., k=8, -128 to 127 [1 val for 0])

Getting representation

Given an 8-bit wordsize, what is the value of 10001011?

What do you mean, Unsigned, Signed Magnitude, 1's complement or 2's complement?

- Unsigned: 128 + 8 + 2 + 1 = 139
- Signed Mag: -1 * (8 + 2 + 1) = -11
- 1's Complement: the negation of 01110100 = -116
- 2's Complement: 1's complement + 1 = -117

Representation v. Operation

- We have discussed various representations for expressing integers
  - unsigned, signed magnitude, 1's-complement, 2's-complement
- There are also bit-oriented operations that go by the same names
  - 1's-complement: flip all bits
  - 2's-complement: flip all bits and add 1
- Operation can be performed on a number, regardless of representation
  - e.g., let 10111 be a number in signed-magnitude form (value is -7)
  - 2's complement (operation) of 10111 = 01001 (value is 9 in signed-mag form)
- Observe:
  - 2's-complement operation negates a number when in 2's-complement representation
  - 1's-complement operation negates a number when in 1's-complement representation

Automating Subtraction

Why are we interested in 2's-complement when it seems so less intuitive?

Much easier to automate subtraction (i.e., add #'s of opposite sign)

- Just negate subtrahend (bottom # in subtract) and add

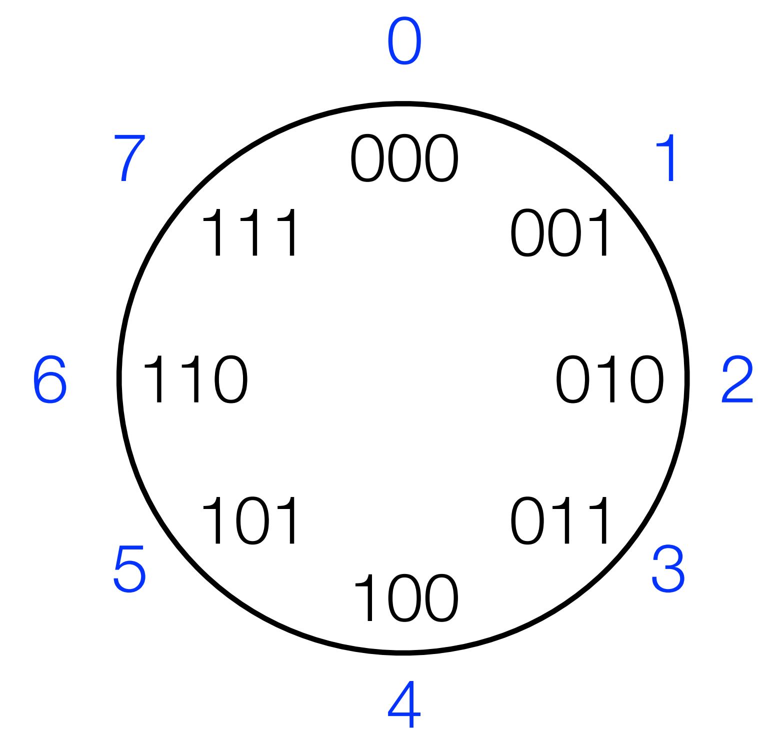
- e.g, wordsize 6, perform 14 - 21 using signed magnitude representation

$$\begin{array}{r} 001110 \\ -\ 010101 \\ \hline \end{array} \xrightarrow{\text{negate subtrahend (2's complement op)}} \begin{array}{r} 001110 \\ +\ 101011 \\ \hline 111001 \end{array}$$

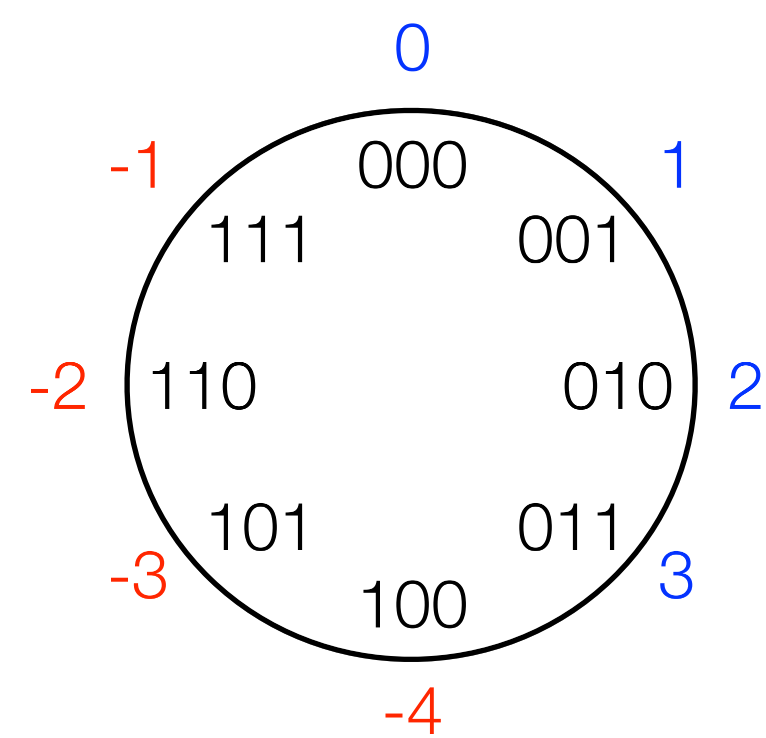
X = 111001, -X = 000111 = 7, X=-7

Why 2's-complement subtraction works (basic idea)

- Think of a pinwheel, here is BCD representation
    - Addition operation of X+Y interprets Y as BCD and shifts Y slots clockwise from X to give the sum



- Now change the representation to 2's complement
    - X+Y still shifts bits (Y in BCD) slots clockwise
    - e.g., 2-3 = 010+101 = 010 shifted 5 slots clockwise = 111 = -1



Another nice feature of 2s complement representation = easy to detect overflow. More on that later. Remainder of the course: unsigned or 2s complement.

What about numbers with fractions?

- Two common notations
    - Fixed-point (the binary point is fixed)
    - Floating-point (the binary point floats to the right of the most significant 1)

Fixed-Point Notation

- Fixed-point representation of 6.75 using 4 integer bits and 4 fraction bits:

01101100

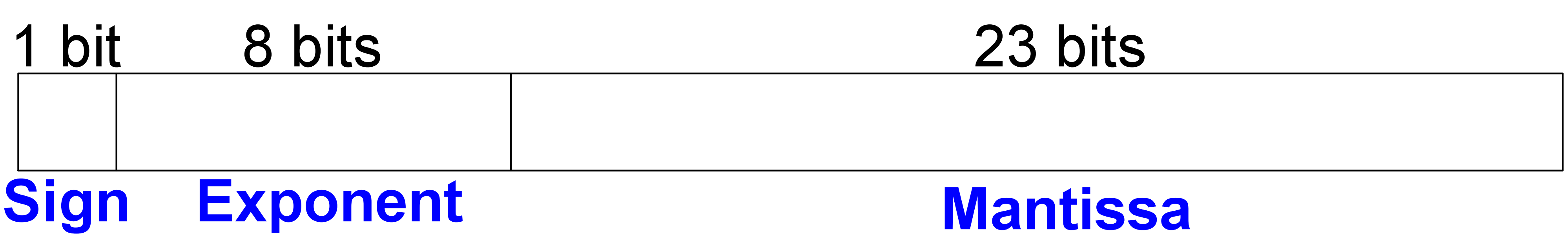
0110.1100

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$$

- The binary point is not a part of the representation but is implied.
- The number of integer and fraction bits must be agreed upon by those generating and those reading the number.

Floating-Point Notation

- The binary point floats to the right of the most significant 1.
- Similar to decimal scientific notation.
- For example, write 273_{10} in scientific notation: $273 = 2.73 \times 10^2$
- In general, a number is written in scientific notation as: $\pm M \times B^E$
  - Where, M = mantissa, B = base, E=exponent



Need a bigger range?

- Change the encoding.
- Floating point (used to represent very large numbers in a compact way)
  - A lot like scientific notation: 5.4×10^{5} exponent mantissa
  - Except that it is binary: $\underline{1001} \times 2^{\underline{1011}}$

What about negative numbers?

- Change the encoding.
  - Sign and magnitude
  - Ones compliment
  - Twos compliment

Sign and magnitude

- Most significant bit is sign
- Rest of bits are magnitude

$$0110 = (6)_{10} \qquad 1110 = (-6)_{10}$$

- Two representations of zero

$$0000 = (0)_{10} \qquad 1000 = (-0)_{10}$$

Ones compliment

- Compliment bits in positive value to create negative value
- Most significant bit still a sign bit

$$0110 = (6)_{10} \qquad 1001 = (-6)_{10}$$

- Two representations of zero

$$0000 = (0)_{10} \qquad 1111 = (-0)_{10}$$

Twos compliment

- Compliment bits in positive value and add 1 to create negative value
- Most significant bit still a sign bit

$$0110 = (6)_{10} \qquad 1001 + 1 = 1010 = (-6)_{10}$$

- One representation of zero

$$0000 = (0)_{10} \qquad 1000 = (-8)_{10} \qquad 1111 = (-1)_{10}$$

- One more negative number than positive

$$\text{MIN: } 1000 = (-8)_{10} \qquad \text{MAX: } 0111 = (7)_{10}$$

How about letters?

- Change the encoding.

☐ **TABLE 1-5**
American Standard Code for Information Interchange (ASCII)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Gray code

Binary numeric encoding where successive numbers differ by only 1 bit

value	BCD	*# bit flips*	Gray	*# bit flips*
0	0 0 0	*3*	0 0 0	*1*
1	0 0 1	*1*	0 0 1	*1*
2	0 1 0	*2*	0 1 1	*1*
3	0 1 1	*1*	0 1 0	*1*
4	1 0 0	*3*	1 1 0	*1*
5	1 0 1	*1*	1 1 1	*1*
6	1 1 0	*2*	1 0 1	*1*
7	1 1 1	*1*	1 0 0	*1*

How about letters?

- Change the encoding.

TABLE 1-5
American Standard Code for Information Interchange (ASCII)

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL