Lecture 3 – Digital Systems and Binary Numbers – Part II

Rami Abielmona
University of Ottawa
January 20, 2015
ITI 1100 B
Digital Systems I

Presentation Outline

- Unsigned and Signed Numbers
- 1’s Complement Numbers
- 2’s Complement Numbers
- Overflow and its Detection
- BCD and ASCII
- Floating-Point Numbers
- Key terms

Why Study Arithmetic ?

- We have a firm grip of arithmetic when it comes to base 10, as the latter is easily visualized on our set of 10 fingers
- But…
  - How are negative numbers represented ?
  - What is the largest number that can be represented in a computer word ?
  - What happens if an operation creates a number bigger than can be represented ?
  - What about fractions and real numbers ?

- the weight of the MSB (Most Significant Bit) is 2^{n-1}
- the range of representable numbers is $[0, 2^n-1]$

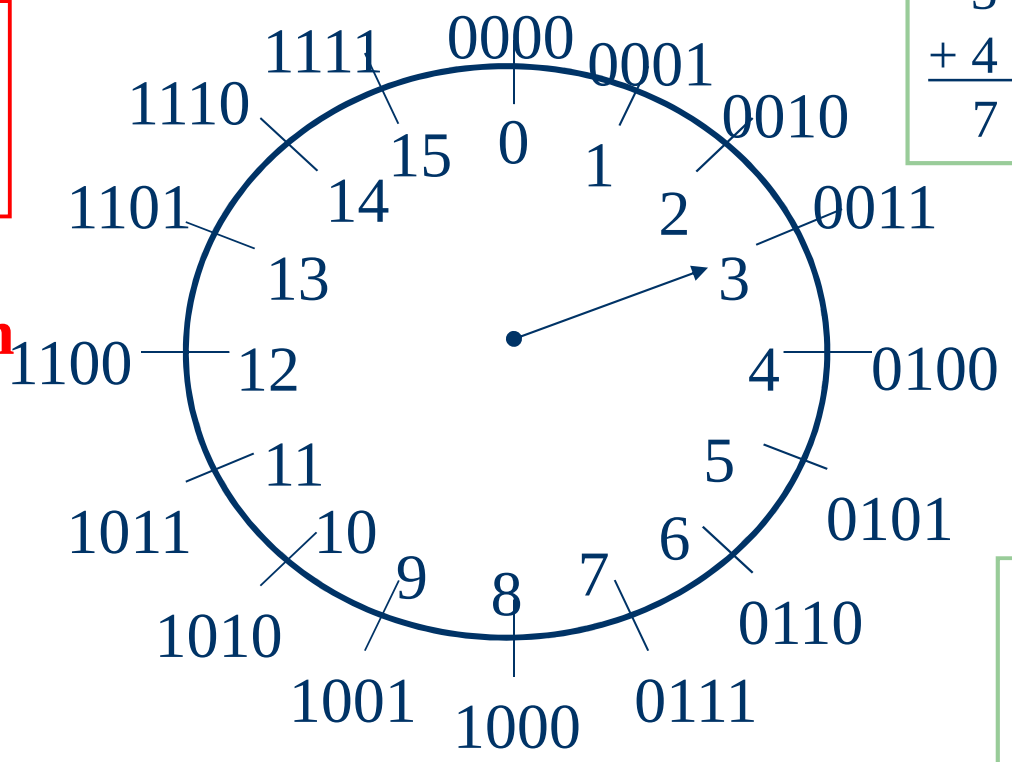
  The result of an operation is out of the domain $[0, 2^n-1]$

Overflow = carry/**borrow** (bit 2^n) is set

Unsigned Numbers

6	0110
- 3	- 0011
3	0011

Subtraction



Addition

3	0011
+ 4	+0100
7	0111

8	1000
- 9	- 1001
-1	**1** 1111

8	1000
+ 9	+1001
17	**1** 0001

Signed and Unsigned Numbers (1)

- Numbers can be represented in any base
  - We prefer base 2 because it corresponds to the on and off signals utilized inside of a computer
- In any number base, the value of the *i*th digit *d* is *d* x Base^{i}
- For example, $(10)_2 = (1 \times 2^1) + (0 \times 2^0) = (2)_{10}$
- Usually, the rightmost bit is the least significant bit (LSB) and the leftmost bit is the most significant bit (MSB)

Signed and Unsigned Numbers (2)

- However, bits are bits!
  - They have no inherent meaning
  - We need a relationship between bits and numbers
- N-bit number can range 0 to $2^n - 1$
- If the n-bit number is represented as $a_{n-1}a_{n-2}\ldots a_1a_0$, then its unsigned integer value is

$$A = \sum_{(i=0)}^{(n-1)} 2^i a^i$$

- Let us now examine different ways of representing positive and negative numbers

Signed Magnitude

- First possible representation is **signed magnitude**

$$A = {}^{(n-2)}\sum_{(i=0)} 2^i a^i \qquad \text{if } a^{n-1} = 0$$

$$A = -\,{}^{(n-2)}\sum_{(i=0)} 2^i a^i \qquad \text{if } a^{n-1} = 1$$

- 000 = +0; 001 = +1; 010 = +2; 011 = +3
- 100 = -0; 101 = -1; 110 = -2; 111 = -3
- MSB is used for sign, and rest is used for magnitude
- Problems arose with this design as well
  - Where to put the sign bit ?
  - Extra step needed to update the sign bit
  - Both a +0 and -0 is represented

Signed Binary numbers

• Recall that digital Systems are made with devices that take on exactly two states : 0 and 1.

•**The only states are “1” and “0”. There is no “-” state!**

→because of hardware limitations computers represent negative numbers by using the leftmost bit for the sign bit.

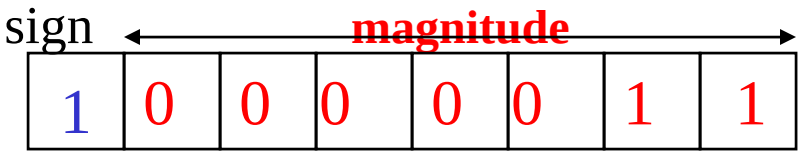
-- “0” indicates a positive number,

-- while a “1” indicates a negative number

Signed Magnitude

•The leftmost bit indicates the sign of the number. The remaining bits give the magnitude of the number

→Using 8 bits to represent binary number the value in the example is:



-3 = 10000011 = 1/ (sign bit) 0000011

→ Sign Magnitude representation is good for having the ability for a human to read and understand what number is represented

Complements in Numbering Systems

- Complements are used in digital systems (computers) for simplifying the Subtraction operation and for logical manipulation
- There are to type of complements for each ***base 'r' system:***

1- Radix complement → r's complement

Ex. ***base 10- → 10's complement***

base 2 → 2's complement

2- Diminished radix complement → (r-1) complement

Ex. ***base 10 → 9's complement***

Base 2 → 1's complement

Radix complement (r's complement)

$$[N]_r = r^n - (N)_r$$

where n is the number of digits in $(N)_r$.

Example

- **2's complement of $(N)_2 = (101001)_2$**

$$[N]_2 = 2^6 - (101001)_2 = (1000000)_2 - (101001)_2 = (010111)_2$$

- **10's complement of $(N)_{10} = (72092)_{10}$**

$$[N]_{10} = (100000)_{10} - (72092)_{10} = (27908)_{10}.$$

1’s Complement

- Next trial is **1’s complement** representation

$$A = -2^{n-1}a^{n-1} + \sum_{(i=0)}^{(n-2)} 2^i a^i \qquad \text{if } a^{n-1} = 0$$

$$A = -2^{n-1}a^{n-1} + 1 + \sum_{(i=0)}^{(n-2)} 2^i a^i \qquad \text{if } a^{n-1} = 1$$

- 000 = +0; 001 = +1; 010 = +2; 011 = +3
- 100 = -3; 101 = -2; 110 = -1; 111 = -0
- Negative number is represented by binary inversion of positive number representation
- Problems arose with this design as well
  - Adders need an extra step to subtract a number
  - Both a +0 and -0 is represented

Diminished radix complement (*r-1*'s complement)

$$[N]_{r-1} = (r^n - 1)r - (N)_r$$

-9's complement of $[546700]_9$

$$= 999999 - 546700 = 453299$$

-1's complement of [1011000]

$$= (10000000 - 1)_2 - (1011000)_2 = (0100111)_2$$

Obtaining 1's complement

- ***1 's complement can be obtained directly from the given number by replacing each of 0s and 1s by 1s and 0s of the number (i.e. complement each bit)***

$$\mathbf{[1011000]} = (10000000 - 1)_2 - (1011000)_2$$

$$= \mathbf{(0100111)_2}$$

1 0 1 1 0 0 0

1 's complement → **0 1 0 0 1 1 1**

Obtaining 2's complement

- **Can be obtained directly from the given number by**
  1-coping each bit of the number starting at the least significant bit and proceeding forward the most significant bit until the first 1 has been copied.

2- After the first 1 has been copied replace each of the remaining 0s and 1s by 1s and 0s respectively

(a) $[1010100]_2 = 2^7 - (1010100)_2 = (1000000)_2 - (1010100)_2 = (\mathbf{0101100})_2$

(b) $[101001]_2 = 2^6 - (101001)_2 = (1000000)_2 - (101001)_2 = (\mathbf{010111})_2$

(a) **1 0 1 0 1 0 0** (b) **1 0 1 0 0 1**

2 's → **0 1 0 1 1 0 0** **0 1 0 1 1 1**

2's Complement (1)

- Final trial was **2's complement** representation

$$A = -2^{n-1}a^{n-1} + \sum_{(i=0)}^{(n-2)} 2^i a^i$$

- 000 = +0; 001 = +1; 010 = +2; 011 = +3
- 100 = -4; 101 = -3; 110 = -2; 111 = -1
- Negative number is represented by binary inversion, then addition of 1, of positive number representation
- Best design because
  - Only one representation for zero (helps in test instructions)
  - No need to update the sign bit
  - Operations easily done on a computer

2's Complement (2)

- Range
  - -2^{n-1} through $2^{n-1} - 1$
- Number of Zero Representations
  - One
- Negation
  - Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer
- Expansion of Bit Length
  - Add additional bit positions to the left and fill in with the value of the original sign bit
- Overflow Rule
  - If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the results has the opposite sign
- Subtraction Rule
  - To subtract B from A, take the twos complement of B and add it to A (e.g. A – B = A + (-B))

2's Complement (3)

- Remember that
  - Positive numbers have an infinite number of 0s on the left
  - Negative numbers have an infinite number of 1s on the left
  - Helps when talking about the sign extension (or expansion of bit length)
- Works because
  - The unsigned sum of an *n*-bit number and its negative is 2^n
  - Hence, the complement of a 2's complement number *x* is $2^n - x$
- The overwhelming choice of ALUs since 1965
- Note that
  - We are discussing *fixed-point representations*, as the *radix point* (binary point) is fixed and assumed to be to the right of the rightmost digit

+3	**0**011
+ (+4)	+**0**100
+7	**0**111

+ 6	110
- (+3)	- 011
+ 6	**0**110
+(-3)	+ **1**101
+ 3	1 **0**011

2's Complement (4)

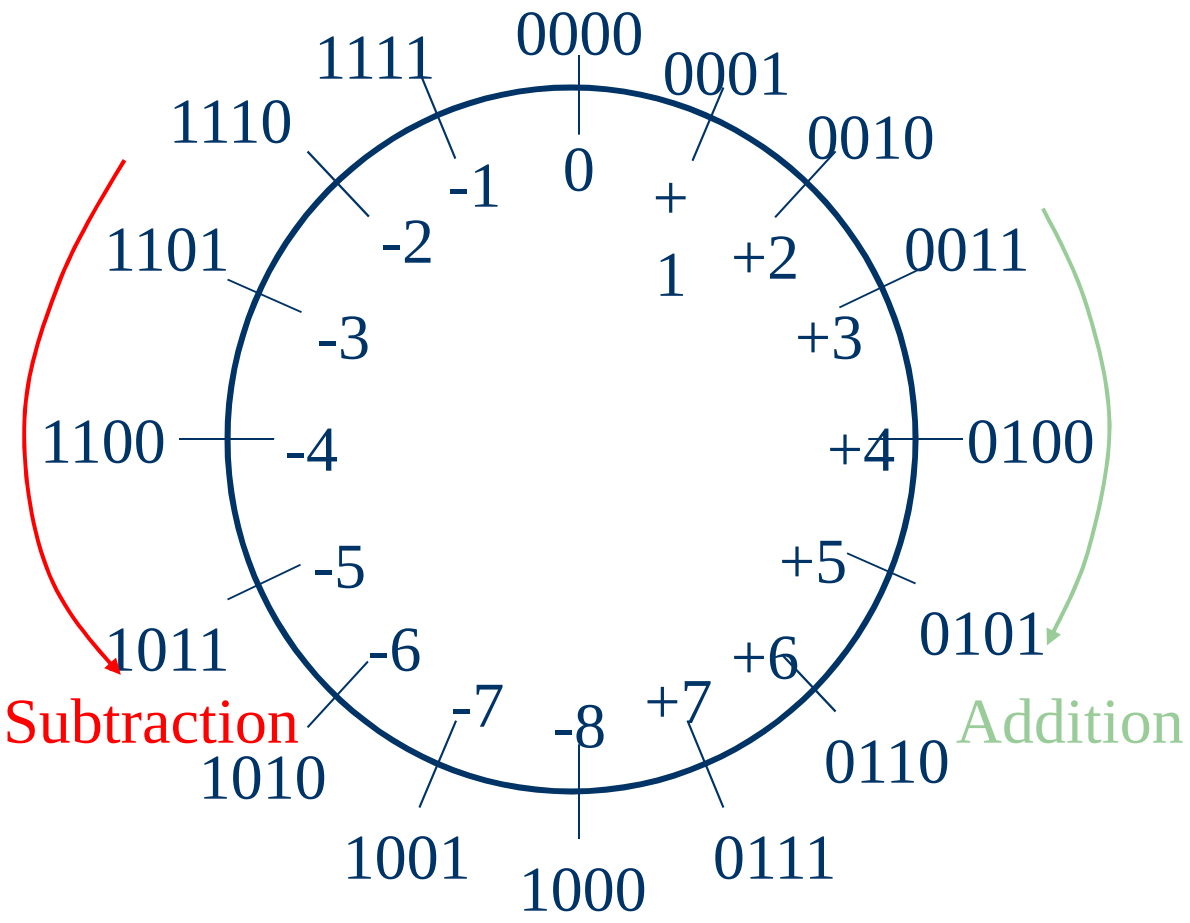
To find
(2's complement of *P*) =
= $2^n - P$ =
= $[(2^n - 1) - P] + 1$ =
= [1's complement of P] +1

$5_{10} = 0101_2$

1010 = *1's complement of 5* +1
$1011 = -5_{10}$

OR

Examine the bits of ***P*** from right to left and copy all bits that are 0 and the first bit that is 1 and then complement the rest of the bits!



32-bit Number Representation

$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = (0)_{10}$
$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001)_2 = +\ (1)_{10}$
$(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2 = +\ (2)_{10}$
...
$(0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_2 = +\ (2{,}147{,}483{,}646)_{10}$
$(0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = +\ (2{,}147{,}483{,}647)_{10}$
$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = -\ (2{,}147{,}483{,}648)_{10}$
$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001)_2 = -\ (2{,}147{,}483{,}647)_{10}$
$(1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2 = -\ (2{,}147{,}483{,}646)_{10}$
...
$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101)_2 = -\ (3)_{10}$
$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_2 = -\ (2)_{10}$
$(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = -\ (1)_{10}$

Subtraction with 2's Complement

- **2's complement are used to convert subtraction to addition, which reduces hardware requirements (only adders are needed).**

$$A - B = A + (-B)$$

(add *2*'s complement of *B* to *A*)

- 2's Complement has the properties of the minus sign

A + (-A) = 0 **A + 2's{A} = 0**

- (- A) = A **2's{ 2's{A}} = A**

A - B = A + (-B) **A - B = A + 2's{B}**

Subtraction with 2's Complement [2]

Examples:

A= 1010100 B= 1000011

- 2's complement

A - B = A + (-B)= A+ [B]

= (1010100) + (0111101) = (0010001)

```
          1010100
+         0111101
-------------------
1 0010001
```

~~1~~

Discard end carry

Subtraction with 1's Complement

Examples: note: same proprieties minus sign as 2's complement.

A= 1010100 B= 1000011

- 1's complement

A - B = A+ [B]= (1010100) + (0111100) = (0010001)

1010100

\+ 0111100

Add end carry 1 0010000

\+ 1

0010001

Subtraction with 10s/9'sComplements

$(72)_{10}$ - $\mathbf{(32)_{10}}$ = $\mathbf{(40)_{10}}$

10's Complement

[32]= 10^2 **– (32)$_{10}$ = (68)$_{10}$**

(72)$_{10}$ + (68)$_{10}$ = ~~**1**~~ **(40)$_{10}$**

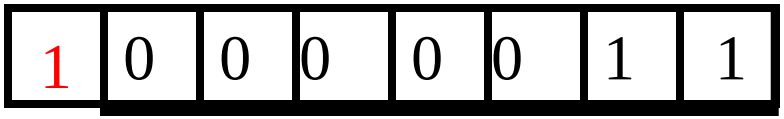
9's Complement

[32]= (10^2 -1) **– (32)$_{10}$ = (67)$_{10}$**

$(72)_{10}$ + **(67)$_{10}$ = (1 + 39)$_{10}$ = (40)$_{10}$**

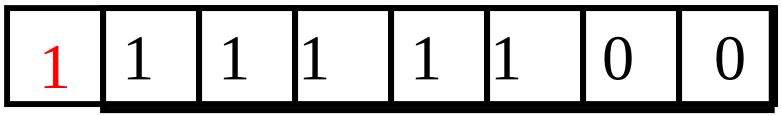
Signed Complement

(a) Signed Magnitude representation



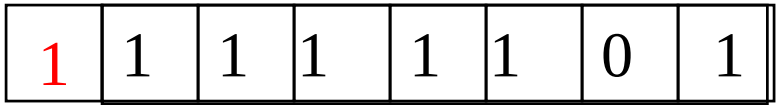
-3 = 10000011 = 1/ (sign bit) 0000011

(b) Signed 1's representation



-3 = 10000011 = 1/ (sign bit) 1111100

(c) Signed 2's representation



-3 = 10000011 = 1/ (sign bit) 1111101

Fixed-Length Registers

- All practical digital devices have fixed-length registers
- This means that numbers in a computer are represented by a fixed number of bits
  - The earliest microprocessors were 4-bit devices
  - Intel 8080 and the 6502 (Apple II) chips were 8-bit
  - Intel 8088 (IBM PC) and Motorola 68000 (Mac) are 16-bit devices
  - Pentium chips and PowerPC chips are 32-bit

Overflow Detection

- Overflow does not occur
  - When adding positive and negative operands
  - When subtracting operands with the same sign (i.e. adding operands of different signs!)

- Overflow does occur
  - When adding two positive numbers and the sum is negative
  - When adding two negative numbers and the sum is positivie
  - When subtracting a negative number from a positive number and the result is negative
  - When subtracting a positive number from a negative number and the result is positive

Range of a number Overflow during addition

- A fixed-length register can only hold a ***Range*** of numbers

→For a 4-bit device, the *range* of positive integers is 0 - 15

→For an 8-bit device the *range* of positive integers is 0 – 255

→ When adding positive integers, ***Overflow*** occurs when the sum falls outside the range of the register

Overflow in Signed Complements

•when numbers are treated as signed complement, a "carry" of 1 from the addition of the most significant bits **DOES NOT** indicate an overflow,

3 00011
+ (-3)+11101
= 00000, with a carry of "1" (2'complement) : We know that addion operation in 2's complement the end-carry is discarded !

•For signed complement, overflow occurs when:

→***The addition of two positive numbers results in a negative number***

OR → ***The addition of two negative numbers results in a positive number***

Overflow Examples

- In a 6-bit register with signed 2's complement

+ 17 = 010001

+ 16 = +010000

=100001

100001 = - (11111) = -$(31)_{10}$ instead of **+ $(33)_{10}$**

- Same with a 7-bit register

+ 17 = 0 010001

+ 16 = +0 010000

= 0100001

0100001 = + 33 No Overflow

Binary codes: *BCD* (1)

- To represent information as strings of alpha-numeric characters.
- ***Binary Coded Decimal (BCD)***
  - Used to represent the decimal digits 0 - 9.
  - 4 bits are used.
  - Each bit position has a weight associated with it (***weighted code***).
  - Weights are: 8, 4, 2, and 1 from MSB to LSB (called 8-4-2-1 code).

Binary codes: *BCD* (2)

– BCD Codes:

0→ 0000	1→ 0001	2→ 0010
3→ 0011	4→0100	5→ 0101
6→ 0110	7→0111	8→ 1000
9→ 1001		

– Used to encode numbers for output to numerical displays

–***Example***: $(9750)_{10} = (1001011101010000)_{BCD}$

BCD – Binary Coded Decimal (1)

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**
$ASCII_{16}$ for Decimal	30	31	32	33	34	35	36	37	38	39

BCD – Binary Coded Decimal (2)

Example

Binary			0111 1111
Decimal			127
BCD			0001 0010 0111
$ASCII_{16}$ for decimal			31 32 37
$ASCII_2$ for decimal	0011 0001	0011 0010	0011 0111
Hex			7F
$ASCII_{16}$ for Hex			37 46
$ASCII_2$ for Hex		0010 0111	0100 0110

Binary codes: *ASCII* [2]

- ***ASCII*** (American Standard Code for Information Interchange) (see table 1.7 of textbook)
  - Most widely used character code.
  - ***Example***: ASCII code representation of the word ***'Digital'***

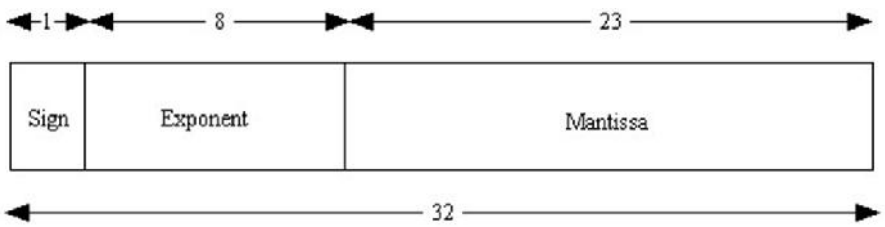
Character	Binary Code	Hexadecimal Code
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

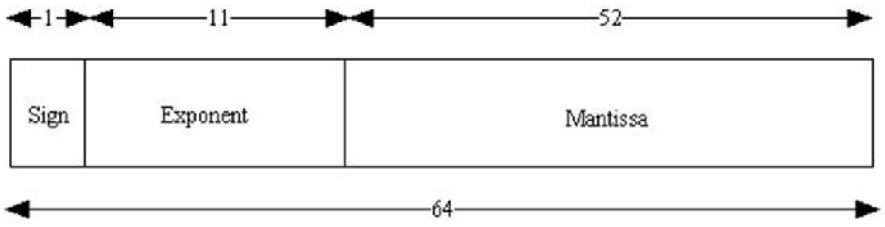
ASCII – American Standard Code for Information Interchange

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**NUL**	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	**TAB**	**LF**	VT	FF	**CR**	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Floating-Point Arithmetic

- Floating-point refers to the variable radix point, as opposed to fixed-point arithmetic
  - IEEE-754 floating-point standard
  - The leading '1' is implicit and not stored
- The general form of a floating-point number is
  $(-1^S) \times (1+M) \times 2^{E-127,}$ where
  S represents the sign bit (1-bit)
  M represents the mantissa (23-bit)
  E represents the exponent (8-bit)
- There are two major precision formats
  - Single-precision is 32-bits
  - Double-precision is 64-bits





IEEE 754 (32-bit) Single Precision

- **Sign bit:**
- - positive, negative
- **Exponent:**
- - using unsigned number to represent signed number
- - biased by adding 127 to the actual value (offset binary)
- **Mantissa:**
- - normalized, if 0<exponent<255, the first bit of mantissa (not shown)
- is 1
- - de-normalized, if exponent = 0, mantissa is not =0,
- - +/- 0, exponent = 0 , mantissa = 0
- - +/- infinity, exponent = 255, mantissa = 0
- - NaN, exponent = 255, mantissa is not 0

Converting to Floating Point

- Express 36.5625_{10} as a 32-bit floating point number?

  Express original value in binary

  $$36.5625_{10} = 100100.1001_2$$

  Normalize

  $$100100.1001_2 = 1.001001001_2 \times 2^5$$

  Put S, E, and M together to form 32-bit binary result

  $$0\ 10000100\ 00100100100000000000000_2$$

  S E = 5+127 M

Converting from Floating Point (1)

- **What decimal value is represented by the following 32-bit floating point number?**

$C17B0000_{16}$

Converting from Floating Point (2)

- Express in binary and find S, E, and M

$C17B0000_{16}$ =

1 10000010 11110110000000000000000$_2$

S E M

1 = negative
0 = positive

Converting from Floating Point (4)

- Express result in decimal

-1111.1011_2

-15

$2^{-1} = 0.5$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

0.6875

Answer: -15.6875

Key Terms and Review Points

- 1’s Complement Representation
- 2’s Complement Representation
- American Standard Code for Information Interchange
- Binary Coded Decimal
- Expansion of Bit Length
- Fixed-Point Representation
- Least Significant Bit
- Mantissa and Exponent
- Most Significant Bit
- Negation
- Overflow Detection
- Radix Complement Representation
- Signed and Unsigned Representation
- Single and Double Precision

- References: Dr. Karmouch & Dr. Groza ITI 1100 Slides