Information representation

CSIM601251

Instructor: Tim Dosen DDAK

Slide By : Erdefi Rakun

Fasilkom UI

Outline

- **Review of information representation**
- Base conversion
- Binary arithmetic operations
- Negative numbers

Information Representation (Review)

Bit (*Bi*nary digi*t*)

0 and 1

Represent *false* and *true* in logic

Represent the *low* and *high* states in electronic devices

Other units

Byte: 8 bits

Nibble: 4 bits (seldom used)

Word: Multiples of byte (e.g.: 1 byte, 2 bytes, 4 bytes, 8 bytes, etc.), depending on the architecture of the computer system

Information Representation (Review)

N bits can represent up to 2^N values.

Examples:

2 bits → represent up to 4 values (00, 01, 10, 11)
3 bits → rep. up to 8 values (000, 001, 010, …, 110, 111)
4 bits → rep. up to 16 values (0000, 0001, 0010, …., 1111)

To represent M values, $\log_2 M$ bits are required.

Examples:

32 values → requires 5 bits
64 values → requires 6 bits
1024 values → requires 10 bits
40 values → how many bits?
100 values → how many bits?

Decimal (Base 10) System Review (1/2)

A weighted-positional number system

- **Base** or **radix** is 10 (the *base* or *radix* of a number system is the total number of symbols/digits allowed in the system)
- Symbols/digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Position is important, as the value of each symbol/digit is dependent on its **type** and its **position** in the number
- Example, the 9 in the two numbers below has different values:
  - $(7594)_{10} = (7 \times 10^3) + (5 \times 10^2) + (9 \times 10^1) + (4 \times 10^0)$
  - $(912)_{10} = (9 \times 10^2) + (1 \times 10^1) + (2 \times 10^0)$
- In general,

$$(a_n a_{n-1} \ldots a_0 . f_1 f_2 \ldots f_m)_{10} = (a_n \times 10^n) + (a_{n-1} \times 10^{n-1}) + \ldots + (a_0 \times 10^0) + (f_1 \times 10^{-1}) + (f_2 \times 10^{-2}) + \ldots + (f_m \times 10^{-m})$$

Decimal (Base 10) System Review (2/2)

- Weighing factors (or weights) are in powers of 10:

  $\ldots\ 10^3\ 10^2\ 10^1\ 10^0\ .\ 10^{-1}\ 10^{-2}\ 10^{-3}\ \ldots$

- To evaluate the decimal number 593.68, the digit in each position is multiplied by the corresponding weight:

  $5\times10^2 + 9\times10^1 + 3\times10^0 + 6\times10^{-1} + 8\times10^{-2}$

  $= (593.68)_{10}$

Other Number Systems (1/2)

- Binary (base 2)
  - Weights in powers of 2
  - Binary digits (bits): **0, 1**
- Octal (base 8)
  - Weights in powers of 8
  - Octal digits: **0, 1, 2, 3, 4, 5, 6, 7**.
- Hexadecimal (base 16)
  - Weights in powers of 16
  - Hexadecimal digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**.
- Base/radix *R*:
  - Weights in powers of *R*

Other Number Systems (2/2)

In some programming languages/software, special notations are used to represent numbers in certain bases

- In programming language C
  - Prefix 0 for octal. E.g.: 032 represents the octal number $(32)_8$
  - Prefix 0x for hexadecimal. E.g.: 0x32 represents the hexadecimal number $(32)_{16}$
- In PCSpim (a MIPS simulator)
  - Prefix 0x for hexadecimal. E.g.: 0x100 represents the hexadecimal number $(100)_{16}$
- In Verilog, the following values are the same
  - 8'b11110000: an 8-bit binary value 11110000
  - 8'hF0: an 8-bit binary value represented in hexadecimal F0
  - 8'd240: an 8-bit binary value represented in decimal 240

Outline

- Review of information representation
- **Base conversion**
- Binary arithmetic operations
- Negative numbers

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Exponent	Value
11	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152

Converting Binary to Decimal

- To convert to decimal, use decimal arithmetic to form S (digit × respective power of 2).
- Example:Convert 11010_2 to N_{10}:

Base-*R* To Decimal Conversion

Easy!

$1101.101_2 = 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3}$

$572.6_8 =$

$2A.8_{16} =$

$341.24_5 =$

Decimal To Binary Conversion

- Method 1
  - Sum-of-Weights Method
- Method 2
  - Repeated Division-by-2 Method (for whole numbers)
  - Repeated Multiplication-by-2 Method (for fractions)

Sum-of-Weights Method

Determine the set of binary weights whose sum is equal to the decimal number

- $(9)_{10} = 8 + 1 = 2^3 + 2^0 = (1001)_2$
- $(18)_{10} = 16 + 2 = 2^4 + 2^1 = (10010)_2$
- $(58)_{10} = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 = (111010)_2$
- $(0.625)_{10} = 0.5 + 0.125 = 2^{-1} + 2^{-3} = (0.101)_2$

Repeated Division-by-2 Method

To convert a whole number to binary, use successive division by 2 until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant bit (LSB)* and the last as the *most significant bit (MSB)*.

$(43)_{10} = (101011)_2$

2	43		
2	21	rem 1	← LSB
2	10	rem 1	
2	5	rem 0	
2	2	rem 1	
2	1	rem 0	
	0	rem 1	← MSB

Repeated Multiplication-by-2 Method

To convert decimal fractions to binary, repeated multiplication by 2 is used, until the fractional product is 0 (or until the desired number of decimal places). The carried digits, or *carries*, produce the answer, with the first carry as the MSB, and the last as the LSB.

$(0.3125)_{10} = (.0101)_2$

	Carry	
$0.3125 \times 2 = 0.625$	0	←MSB
$0.625 \times 2 = 1.25$	1	
$0.25 \times 2 = 0.50$	0	
$0.5 \times 2 = 1.00$	1	←LSB

Conversion Between Decimal and Other Bases

- Base-*R* to decimal: multiply digits with their corresponding weights.
- Decimal to binary (base 2)
  - Whole numbers repeated division-by-2
  - Fractions: repeated multiplication-by-2
- Decimal to base-*R*
  - Whole numbers: repeated division-by-*R*
  - Fractions: repeated multiplication-by-*R*

QUICK REVIEW QUESTIONS (3)

- DLD page 37
  Questions 2-5 to 2-8.

2-5. Convert each of the following decimal numbers to binary (base two) with at most eight digits in the fractional part, rounded to eight places.

a. 2000 b. 0.875 c. 0.07 d. 12.345

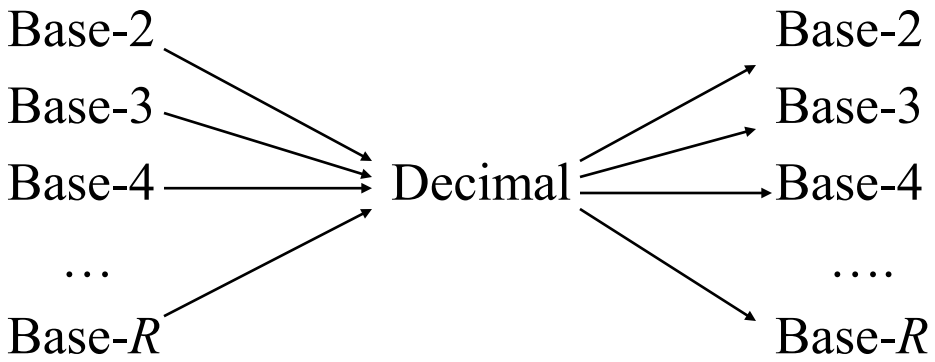
2-6. Convert each of the decimal numbers in question 2-5 above to septimal (base 7) with at most six digits in the fractional part rounded to six places.

2-7. Convert each of the decimal numbers in question 2-5 above to octal (base 8) with at most four digits in the fractional part rounded to four places.

2-8. Convert each of the decimal numbers in question 2-5 above to hexadecimal (base 16) with at most two digits in the fractional part rounded to two places.

Conversion Between Bases

- In general, conversion between bases can be done via decimal:



- Shortcuts for conversion between bases 2, 4, 8, 16 (see next slide)

Binary To Octal/Hexadecimal Conversion

- Binary → Octal: partition in groups of 3

  $(10\ 111\ 011\ 001\ .\ 101\ 110)_2$ =

- Octal → Binary: reverse

  $(2731.56)_8$ =

- Binary → Hexadecimal: partition in groups of 4

  $(101\ 1101\ 1001\ .\ 1011\ 1000)_2$ =

- Hexadecimal → Binary: reverse

  $(5D9.B8)_{16}$ =

QUICK REVIEW QUESTIONS (4)

- DLD page 37
  Questions 2-9 to 2-10.

2-9. Which of the following octal value is equivalent to the binary number $(110001)_2$?

a. $(15)_8$ b. $(31)_8$ c. $(33)_8$ d. $(49)_8$ e. $(61)_8$

2-10. Convert the binary number $(1001101)_2$ to

a. quaternary b. octal c. decimal d. hexadecimal

Numbers in Different Bases

- Good idea to memorize!

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Outline

- Review of information representation
- Base conversion
- **Binary arithmetic operations**
- Negative numbers

ARITHMETIC OPERATIONS – Binary Arithmetic

- Single Bit Addition with Carry
- Multiple Bit Addition
- Single Bit Subtraction with Borrow
- Multiple Bit Subtraction
- Multiplication
- Division

Single Bit Binary Addition with Carry

Given two binary digits (X,Y), a carry in (Z) we get the following sum (S) and carry (C):

Carry in (Z) of 0:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

Carry in (Z) of 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Multiple Bit Binary Addition

- Extending this to two multiple bit examples:

Carries	0	0
Augend	01100	10110
Addend	+ 10001	+ 10111
Sum		

- *Note: The 0 is the default Carry-In to the least significant bit.*

Single Bit Binary Subtraction with Borrow

- Given two binary digits (X,Y), a borrow in (Z) we get the following difference (S) and borrow (B):
- Borrow in (Z) of 0:

Z	**0**	**0**	**0**	**0**
X	**0**	**0**	**1**	**1**
- Y	**-0**	**-1**	**-0**	**-1**
BS	**0 0**	**1 1**	**0 1**	**0 0**

- Borrow in (Z) of 1:

Z	**1**	**1**	**1**	**1**
X	**0**	**0**	**1**	**1**
- Y	**-0**	**-1**	**-0**	**-1**
BS	**11**	**1 0**	**0 0**	**1 1**

Multiple Bit Binary Subtraction

- Extending this to two multiple bit examples:

Borrows	0	0
Minuend	10110	10110
Subtrahend	- 10010	- 10011
Difference		

- *Notes:*

  The 0 is a Borrow-In to the least significant bit. If the Subtrahend > the Minuend, interchange and append a – to the result.

Binary Multiplication

The binary multiplication table is simple:

$0 * 0 = 0 \mid 1 * 0 = 0 \mid 0 * 1 = 0 \mid 1 * 1 = 1$

Extending multiplication to multiple digits:

Multiplicand	1011
Multiplier	× 101
Partial Products	1011
	0000-
	+ 1011--
Product	110111

QUICK REVIEW QUESTIONS (1)

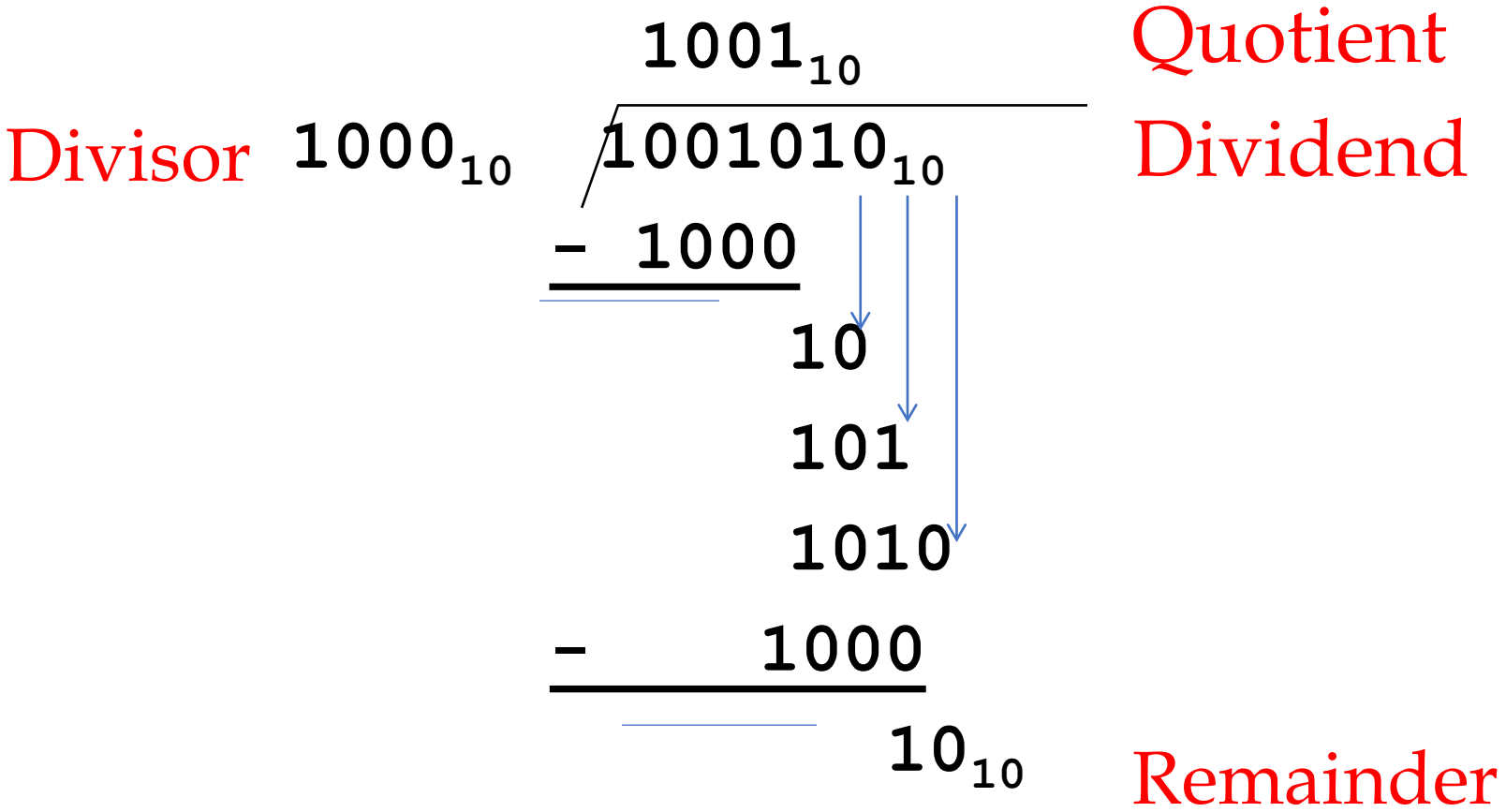
What is $(1011)_2$ x $(101)_2$?

a. $(10000)_2$ b. $(110111)_2$ c. $(111111)_2$ d. $(111011)_2$ e. $(101101)_2$

Perform the following operations on binary numbers.

a. $(10111110)_2 + (10001101)_2$
b. $(11010010)_2 - (01101101)_2$
c. $(11100101)_2 - (00101110)_2$

Division



Dividend = Quotient x Divisor + Remainder

Binary Division

```
                                    101011          = 43
12  /  521  =  01100  /  01000001001
    -  384  =         -   0110000000
       137  =              010001001
    -    0  =         -            0
       137  =              010001001
    -   96  =         -     01100000
        41  =                0101001
    -    0  =         -            0
        41  =                0101001
    -   24  =         -       011000
        17  =                 010001
    -   12  =         -        01100
         5  =                    101
```

Taken from Daniel J. Sorin Slide

Division Exercise

$1110_2 : 0011_2$

Outline

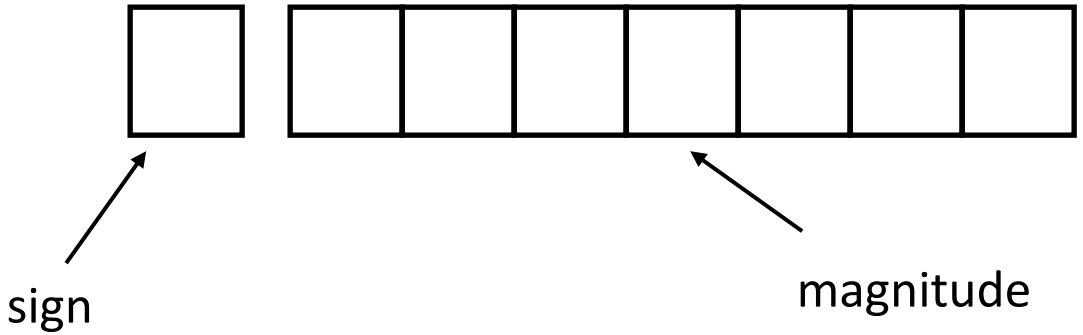
- Review of information representation
- Base conversion
- Binary arithmetic operations
- **Negative numbers**

Negative Number

- Unsigned numbers: only non-negative values.
- Signed numbers: include all values (positive and negative)
- There are 3 common representations for signed binary numbers:
  - Sign-and-Magnitude
  - 1s Complement
  - 2s Complement

Sign and magnitude (1/3)

- The sign is represented by a 'sign bit'
  - 0 for +
  - 1 for -
- E.g.: a 1-bit sign and 7-bit magnitude format.



- $00110100 \rightarrow +110100_2 = +52_{10}$
- $10010011 \rightarrow -10011_2 = -19_{10}$

Sign and magnitude (2/3)

- Largest value: $01111111 = +127_{10}$
- Smallest value: $11111111 = -127_{10}$
- Zeros: $00000000 = +0_{10}$
  $10000000 = -0_{10}$
- Range: -127_{10} to $+127_{10}$
- Question:

  For an *n*-bit sign-and-magnitude representation, what is the range of values that can be represented?

Sign and magnitude (3/3)

- To negate a number, just invert the sign bit.
- Examples:
  - How to negate 00100001_{sm} (decimal 33)?
    Answer: 10100001_{sm} (decimal -33)
  - How to negate 10000101_{sm} (decimal -5)?
    Answer: 00000101_{sm} (decimal +5)

1s complement (1/3)

- Given a number x which can be expressed as an n-bit binary number, its negated value can be obtained in **1s-complement** representation using:

$$-x = 2^n - x - 1$$

- Example: With an 8-bit number 00001100 (or 12_{10}), its negated value expressed in 1s-complement is:

-00001100_2 $= 2^8 - 12 - 1$ (calculation in decimal)
$= 243$
$= 11110011_{1s}$

(This means that -12_{10} is written as 11110011 in 1s-complement representation.)

1s complement (2/3)

- Essential technique to negate a value: invert all the bits.
- Largest value: $01111111 = +127_{10}$
- Smallest value: $10000000 = -127_{10}$
- Zeros: $00000000 = +0_{10}$
  $11111111 = -0_{10}$
- Range: -127_{10} to $+127_{10}$
- The most significant (left-most) bit still represents the sign: 0 for positive; 1 for negative.

1s complement (3/3)

Examples (assuming 8-bit numbers):

$(14)_{10} = (00001110)_2 = (00001110)_{1s}$

$-(14)_{10} = -(00001110)_2 = (11110001)_{1s}$

$-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{1s}$

2s complement (1/3)

- Given a number x which can be expressed as an n-bit binary number, its negated value can be obtained in **2s-complement** representation using:

$$-x = 2^n - x$$

- Example: With an 8-bit number 00001100 (or 12_{10}), its negated value expressed in 1s-complement is:

-00001100_2 $= 2^8 - 12$ (calculation in decimal)
$= 244$
$= 11110100_{2s}$

(This means that -12_{10} is written as 11110100 in 2s-complement representation.)

2s complement (2/3)

- Essential technique to negate a value: invert all the bits, then add 1.
- Largest value: $01111111 = +127_{10}$
- Smallest value: $10000000 = -128_{10}$
- Zero: $00000000 = +0_{10}$
- Range: -128_{10} to $+127_{10}$
- The most significant (left-most) bit still represents the sign: 0 for positive; 1 for negative

2s complement (3/3)

Examples (assuming 8-bit numbers):

$(14)_{10} = (00001110)_2 = (00001110)_{2s}$

$-(14)_{10} = -(00001110)_2 = (11110010)_{2s}$

$-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{2s}$

Comparison

Important!

4-bit system

Positive values

Value	Sign-and-Magnitude	1s Comp.	2s Comp.
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000

Negative values

Value	Sign-and-Magnitude	1s Comp.	2s Comp.
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-	-	1000

Complement on Fraction

- We can extend the idea of complement on fractions.
- Examples:
  - Negate 0101.01 in 1s-complement
    Answer: 1010.10
  - Negate 111000.101 in 1s-complement
    Answer: 000111.010
  - Negate 0101.01 in 2s-complement
    Answer: 1010.11

2s Complement Addition/Subtraction (1/3)

- Algorithm for addition, A + B:
  1. Perform binary addition on the two numbers.
  2. Ignore the carry out of the MSB.
  3. Check for overflow. Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B.
- Algorithm for subtraction, A – B:

  A – B = A + (-B)
  1. Take 2s-complement of B.
  2. Add the 2s-complement of B to A.

Overflow

- Signed numbers are of a fixed range.
- If the result of addition/subtraction goes beyond this range, an **overflow** occurs.
- Overflow can be easily detected:
  - *positive add positive* → negative
  - *negative add negative* → positive
- Example: 4-bit 2s-complement system
  - Range of value: -8_{10} to 7_{10}
  - $0101_{2s} + 0110_{2s} = 1011_{2s}$
    $5_{10} + 6_{10} = -5_{10}$?! (overflow!)
  - $1001_{2s} + 1101_{2s} = \underline{1}0110_{2s}$ (discard end-carry) $= 0110_{2s}$
    $-7_{10} + -3_{10} = 6_{10}$?! (overflow!)

2s Complement Addition/Subtraction (2/3)

- Examples: 4-bit system

```
  +3      0011
+ +4    + 0100
----    -------
  +7      0111
----    -------
```

```
  -2      1110
+ -6    + 1010
----    -------
  -8     11000
----    -------
```

```
  +6      0110
+ -3    + 1101
----    -------
  +3     10011
----    -------
```

```
  +4      0100
+ -7    + 1001
----    -------
  -3      1101
----    -------
```

- Which of the above is/are overflow(s)?

2s Complement Addition/Subtraction (3/3)

- Examples: 4-bit system

```
  -3        1101
+ -6      + 1010
----      -------
  -9       10111
----      -------
```

```
  +5        0101
+ +6      + 0110
----      -------
 +11        1011
----      -------
```

- Which of the above is/are overflow(s)?

1s Complement Addition/Subtraction (1/2)

- Algorithm for addition, A + B:
  1. Perform binary addition on the two numbers.
  2. If there is a carry out of the MSB, add 1 to the result.
  3. Check for overflow. Overflow occurs if result is opposite sign of A and B.
- Algorithm for subtraction, A – B:

  A – B = A + (-B)
  1. Take 1s-complement of B.
  2. Add the 1s-complement of B to A.

1s Complement Addition/Subtraction (2/2)

- Examples: 4-bit system

Any overflow?

```
  +3        0011
+ +4      + 0100
----      -------
  +7        0111
----      -------
```

```
  +5        0101
+ -5      + 1010
----      -------
  -0        1111
----      -------
```

```
  -2        1101
+ -5    +   1010
----      ------
  -7       10111
----     +    1
           ------
            1000
```

```
  -3        1100
+ -7      + 1000
----      -------
 -10       10100
----      +    1
           -------
            0101
```

QUICK REVIEW QUESTIONS (5)

- DLD page 37

2-13. In a 6-bit 2's complement binary number system, what is the decimal value represented by $(100100)_{2s}$?

a. -4 b. 36 c. -36 d. -27 e. -28

2-14. In a 6-bit 1's complement binary number system, what is the decimal value represented by $(010100)_{1s}$?

a. -11 b. 43 c. -43 d. 20 e. -20

2-15. What is the range of values that can be represented in a 5-bit 2's complement binary systems?

a. 0 to 31 b. -8 to 7 c. -8 to 8 d. -15 to 15 e. -16 to 15

QUICK REVIEW QUESTIONS (5)

- DLD page 37

2-16. In a 4-bit 2's complement scheme, what is the result of this operation: $(1011)_{2s}$ + $(1001)_{2s}$?

a. 4 b. 5 c. 20 d. -12 e. overflow

2-17. Assuming a 6-bit 2's complement system, perform the following subtraction operation by converting it into addition operation:

a. $(011010)_{2s}$ - $(010000)_{2s}$

b. $(011010)_{2s}$ - $(001101)_{2s}$

c. $(000011)_{2s}$ - $(010000)_{2s}$

2-18. Assuming a 6-bit 1's complement system, perform the following subtraction operation by converting it into addition operation:

a. $(011111)_{1s}$ - $(010101)_{1s}$

b. $(001010)_{1s}$ - $(101101)_{1s}$

c. $(100000)_{1s}$ - $(010011)_{1s}$

Eight Conditions for Signed-Magnitude Addition/Subtraction

Operation	ADD Magnitudes	SUBTRACT Magnitudes		
		A > B	A < B	A = B
(+A) + (+B)	+ (A + B)			
(+A) + (-B)		+ (A – B)	- (B – A)	+ (A – B)
(-A) + (+B)		- (A – B)	+ (B – A)	+ (A – B)
(-A) + (-B)	- (A + B)			
(+A) - (+B)		+ (A – B)	- (B – A)	+ (A – B)
(+A) - (-B)	+ (A + B)			
(-A) - (+B)	- (A + B)			
(-A) - (-B)		- (A – B)	+ (B – A)	+ (A – B)

Examples

Example of adding two magnitudes when the result is the sign of both operands:

```
   +3   0 011
+  +2   0 010
-------------
   +5   0 101
```

Example of adding two magnitudes when the result is the sign of the larger magnitude:

```
   -3   1 011
+  +2   0 010
-------------
 -( +3     011
  - +2)    010
-------------
  - 1   1 001
```

QUICK REVIEW QUESTIONS (6)

Using 8-bit signed magnitude:

a) $0100\ 1111_2 + 0010\ 0011_2$

b) $0100\ 1111_2 + 0110\ 0011_2$

c) $(99)_{10} - (79)_{10}$

d) $(-19)_{10} + (13)_{10}$

e) Subtract $(-24)_{10}$ from $(-43)_{10}$