

# Number Systems

1

## Number Representation

Binary  
Hexadecimal  
Decimal

2

## Topics to be Discussed

- How are numeric data items actually stored in computer memory?
- How much space (memory locations) is allocated for each type of data?
  - int, float, char, double, etc.
- How are characters and strings stored in memory?
  - Already discussed.

3

## Number System :: The Basics

- We are accustomed to using the so-called *decimal number system*.
  - Ten digits :: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Every digit position has a weight which is a power of 10.
  - **Base** or **radix** is 10.
- Example:

$$234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

$$250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$$

4

## Binary Number System

- Two digits:
  - 0 and 1.
  - Every digit position has a weight which is a power of 2.
  - **Base** or **radix** is 2.

- Example:

$$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$101.01 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

5

## Binary-to-Decimal Conversion

- Each digit position of a binary number has a weight.
  - Some power of 2.

- A binary number:

$$B = b_{n-1} b_{n-2} \dots b_1 b_0 . b_{-1} b_{-2} \dots b_{-m}$$

Corresponding value in decimal:

$$D = \sum_{i=-m}^{n-1} b_i 2^i$$

6

## Examples

$$1. \quad 101011 \rightarrow 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$$

$$= 43$$

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

$$2. \quad .0101 \rightarrow 0x2^{-1} + 1x2^{-2} + 0x2^{-3} + 1x2^{-4}$$

$$= .3125$$

$$(.0101)_2 = (.3125)_{10}$$

$$3. \quad 101.11 \rightarrow 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-2}$$

$$5.75$$

$$(101.11)_2 = (5.75)_{10}$$

7

## Decimal-to-Binary Conversion

- Consider the integer and fractional parts separately.
- For the integer part,
  - Repeatedly divide the given number by 2, go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders *in reverse order*.
- For the fractional part,
  - Repeatedly multiply the given fraction by 2.
    - Accumulate the integer part (0 or 1).
    - If the integer part is 1, chop it off.
  - Arrange the integer parts *in the order* they are obtained.

8

### Example 1 :: 239

2	239	
2	119	--- 1
2	59	--- 1
2	29	--- 1
2	14	--- 1
2	7	--- 0
2	3	--- 1
2	1	--- 1
2	0	--- 1

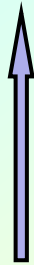


$$(239)_{10} = (11101111)_2$$

9

### Example 2 :: 64

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



$$(64)_{10} = (1000000)_2$$

10

### Example 3 :: .634

$$.634 \times 2 = 1.268$$

$$.268 \times 2 = 0.536$$

$$.536 \times 2 = 1.072$$

$$.072 \times 2 = 0.144$$

$$.144 \times 2 = 0.288$$

:

:



$$(.634)_{10} = (.10100.....)_2$$

11

### Example 4 :: 37.0625

$$(37)_{10} = (100101)_2$$

$$(.0625)_{10} = (.0001)_2$$

$$\therefore (37.0625)_{10} = (100101.0001)_2$$

12

## Hexadecimal Number System

- A compact way of representing binary numbers.
- 16 different symbols (radix = 16).

0 → 0000	8 → 1000
1 → 0001	9 → 1001
2 → 0010	A → 1010
3 → 0011	B → 1011
4 → 0100	C → 1100
5 → 0101	D → 1101
6 → 0110	E → 1110
7 → 0111	F → 1111

13

## Binary-to-Hexadecimal Conversion

- For the integer part,
  - Scan the binary number from *right to left*.
  - Translate each group of four bits into the corresponding hexadecimal digit.
    - Add *leading* zeros if necessary.
- For the fractional part,
  - Scan the binary number from *left to right*.
  - Translate each group of four bits into the corresponding hexadecimal digit.
    - Add *trailing* zeros if necessary.

14

## Examples

1.  $(\underline{1011} \ \underline{0100} \ \underline{0011})_2 = (B43)_{16}$
2.  $(\underline{10} \ \underline{1010} \ \underline{0001})_2 = (2A1)_{16}$
3.  $(\underline{.1000} \ \underline{010})_2 = (.84)_{16}$
4.  $(\underline{101} \ . \ \underline{0101} \ \underline{111})_2 = (5.5E)_{16}$

15

## Hexadecimal-to-Binary Conversion

- Translate every hexadecimal digit into its 4-bit binary equivalent.
  - Discard leading and trailing zeros if desired.
- Examples:
  - $(3A5)_{16} = (0011 \ 1010 \ 0101)_2$
  - $(12.3D)_{16} = (0001 \ 0010 \ . \ 0011 \ 1101)_2$
  - $(1.8)_{16} = (0001 \ . \ 1000)_2$

16



## Number Representation

### Unsigned and Signed numbers

17

## Unsigned Binary Numbers

- An n-bit binary number

$$B = b_{n-1}b_{n-2} \dots b_2b_1b_0$$

( $2^n$  distinct combinations are possible, 0 to  $2^n-1$ )

- For  $n = 3$ , there are 8 distinct combinations.

– 000, 001, 010, 011, 100, 101, 110, 111

- Range of numbers that can be represented

$n=8 \rightarrow 0 \text{ to } 2^8-1 \text{ (255)}$

$n=16 \rightarrow 0 \text{ to } 2^{16}-1 \text{ (65535)}$

$n=32 \rightarrow 0 \text{ to } 2^{32}-1 \text{ (4294967295)}$

18

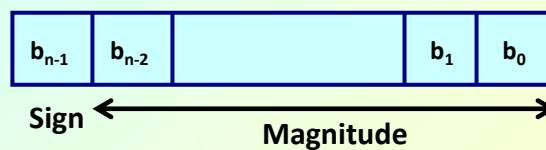
## Signed Integer Representation

- Many of the numerical data items that are used in a program are signed (positive or negative).
  - Question:: How to represent sign?
- Three possible approaches:
  - a) Sign-magnitude representation
  - b) One's complement representation
  - c) Two's complement representation

19

## Sign-magnitude Representation

- For an n-bit number representation
  - The most significant bit (MSB) indicates sign
    - 0 → positive
    - 1 → negative
  - The remaining n-1 bits represent magnitude.



20

**Example :: n=4**

0000 → +0	1000 → -0
0001 → +1	1001 → -1
0010 → +2	1010 → -2
0011 → +3	1011 → -3
0100 → +4	1100 → -4
0101 → +5	1101 → -5
0110 → +6	1110 → -6
0111 → +7	1111 → -7

15 distinct numbers can be represented
--

21

**Contd.**

- Range of numbers that can be represented:

Maximum ::  $+(2^{n-1} - 1)$

Minimum ::  $-(2^{n-1} - 1)$

- A problem:

Two different representations of zero.

+0 → 0 000...0

-0 → 1 000...0

22

## One's Complement Representation

- **Basic idea:**
  - Positive numbers are represented exactly as in sign-magnitude form.
  - Negative numbers are represented in 1's complement form.
- **How to compute the 1's complement of a number?**
  - Complement every bit of the number (1→0 and 0→1).
  - MSB will indicate the sign of the number.
    - 0 → positive
    - 1 → negative

23

### Example :: n=4

0000 → +0	1000 → -7
0001 → +1	1001 → -6
0010 → +2	1010 → -5
0011 → +3	1011 → -4
0100 → +4	1100 → -3
0101 → +5	1101 → -2
0110 → +6	1110 → -1
0111 → +7	1111 → -0

To find the representation of, say, -4, first note that

+4 = 0100

-4 = 1's complement of 0100 = 1011

24

## Contd.

- Range of numbers that can be represented:

Maximum ::  $+(2^{n-1} - 1)$

Minimum ::  $-(2^{n-1} - 1)$

- A problem:

Two different representations of zero.

+0 → 0 000....0

-0 → 1 111....1

- Advantage of 1's complement representation

- Subtraction can be done using addition.
- Leads to substantial saving in circuitry.
- Sign extension is possible to increase number of bits to represent.

-3 = 1100 (in 4 bits) = 1111 1100 (in 8 bits)

+3 = 0011 (in 4 bits) = 0000 0011 (in 8 bits)

25

## Two's Complement Representation

- Basic idea:

- Positive numbers are represented exactly as in sign-magnitude form.
- Negative numbers are represented in 2's complement form.

- How to compute the 2's complement of a number?

- Complement every bit of the number ( $1 \rightarrow 0$  and  $0 \rightarrow 1$ ), and then **add 1** to the resulting number.
- MSB will indicate the sign of the number.

0 → positive

1 → negative

26

**Example :: n=4**

0000 → +0	1000 → -8
0001 → +1	1001 → -7
0010 → +2	1010 → -6
0011 → +3	1011 → -5
0100 → +4	1100 → -4
0101 → +5	1101 → -3
0110 → +6	1110 → -2
0111 → +7	1111 → -1

To find the representation of, say, -4, first note that

$$+4 = 0100$$

$$-4 = 2\text{'s complement of } 0100 = 1011 + 1 = 1100$$

27

**Contd.**

- Range of numbers that can be represented:

$$\text{Maximum} :: +(2^{n-1} - 1)$$

$$\text{Minimum} :: -2^{n-1}$$

- Advantage:

- *Unique representation of zero.*
- Subtraction can be done using addition.
- Leads to substantial saving in circuitry.
- Sign extension is possible to increase number of bits to represent.

$$-3 = 1101 \text{ (in 4 bits)} = 1111 \ 1101 \text{ (in 8 bits)}$$

$$+3 = 0011 \text{ (in 4 bits)} = 0000 \ 0011 \text{ (in 8 bits)}$$

- Almost all computers today use the 2's complement representation for storing negative numbers.

28

## Contd.

- In C (typical values):
  - short int
    - 16 bits →  $+ (2^{15}-1)$  to  $-2^{15}$
  - int
    - 32 bits →  $+ (2^{31}-1)$  to  $-2^{31}$
  - long int
    - 64 bits →  $+ (2^{63}-1)$  to  $-2^{63}$

29

## Binary operations

### Addition

### Subtraction using addition

30

## Binary addition

- Rules for adding two bits:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10, \text{ i.e., } 0 \text{ with carry of } 1$$

- Addition examples for unsigned numbers:

0 1 0 1	0 1 1 1	1 0 0 1
0 0 0 1	0 0 1 1	1 0 1 0
=====	=====	=====
0 1 1 0	1 0 1 0	1 0 0 1 1

Carry

31

## Subtraction Using Addition :: 1's Complement

- How to compute  $A - B$  ?

- Compute the 1's complement of B (say,  $B_1$ ).

- Compute  $R = A + B_1$

- If the carry obtained after addition is '1'

- Add the carry back to R (called *end-around carry*).

- That is,  $R = R + 1$ .

- The result is a positive number.

Else

- The result is negative, and is in 1's complement form.

32



### Example 1 :: 6 – 2

**1's complement of 2 = 1101**

$$\begin{array}{r}
 6 \quad :: \quad 0 \ 1 \ 1 \ 0 \quad A \\
 -2 \quad :: \quad 1 \ 1 \ 0 \ 1 \quad B_1 \\
 \hline
 \text{End-around carry } 1 \quad 0 \ 0 \ 1 \ 1 \quad R \\
 \phantom{1} \quad \quad \quad \quad \quad 1 \\
 \hline
 \phantom{1} \quad \quad \quad 0 \ 1 \ 0 \ 0 \quad \rightarrow +4
 \end{array}$$

**Assume 4-bit representations.**

Since there is a carry, it is added back to the result.

**The result is positive.**

33

### Example 2 :: 3 – 5

**1's complement of 5 = 1010**

$$\begin{array}{rcll} 3 & :: & 0 & 0 & 1 & 1 & A \\ -5 & :: & 1 & 0 & 1 & 0 & B_1 \\ & & \hline & & 1 & 1 & 0 & 1 & R \end{array}$$

↓

-2

**Assume 4-bit representations.**

Since there is no carry, the result is negative.

1101 is the 1's complement of 0010, that is, it represents  $-2$ .

34

## Subtraction Using Addition :: 2's Complement

- How to compute  $A - B$  ?

- Compute the 2's complement of B (say,  $B_2$ ).
- Compute  $R = A + B_2$
- If the carry obtained after addition is '1'

- Ignore the carry.
- The result is a positive number.

Else


- The result is negative, and is in 2's complement form.

35


## Example 1 :: $6 - 2$

2's complement of 2 =  $1101 + 1 = 1110$

6	::	0	1	1	0	A
-2	::	1	1	1	0	$B_2$
		1	0	1	0	0
						R



Ignore carry



+4

Assume 4-bit representations.

Presence of carry indicates that the result is positive.


No need to add the end-around carry like in 1's complement.

36

### Example 2 :: 3 – 5

2's complement of 5 =  $1010 + 1 = 1011$

$$\begin{array}{rcl}
 3 & :: & 0 \ 0 \ 1 \ 1 & A \\
 -5 & :: & 1 \ 0 \ 1 \ 1 & B_2 \\
 \hline
 & & 1 \ 1 \ 1 \ 0 & R
 \end{array}$$

  
 -2

Assume 4-bit representations.

Since there is no carry, the result is negative.

1110 is the 2's complement of 0010, that is, it represents -2.

37

### Floating-point number representation

38

## Floating-point Numbers

- The representations discussed so far applies only to integers.
  - Cannot represent numbers with fractional parts.
- We can assume a decimal point before a 2's complement number.
  - In that case, pure fractions (without integer parts) can be represented.
- We can also assume the decimal point somewhere in between.
  - This lacks flexibility.
  - Very large and very small numbers cannot be represented.

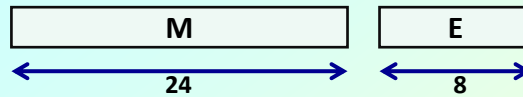
39

## Representation of Floating-Point Numbers

- A floating-point number  $F$  is represented by a doublet  $\langle M, E \rangle$  :
 
$$F = M \times B^E$$
  - $B$  : exponent base (usually 2)
  - $M$  : mantissa
  - $E$  : exponent
  - $M$  is usually represented in 2's complement form, with an implied decimal point before it.
- For example,
  - In decimal,
 
$$0.235 \times 10^6$$
  - In binary,
 
$$0.101011 \times 2^{0110}$$

40

### Example :: 32-bit representation



- M represents a 2's complement fraction  
 $1 > M > -1$
- E represents the exponent (in 2's complement form)  
 $127 > E > -128$

- Points to note:

- The number of *significant digits* depends on the number of bits in M.
  - 6 significant digits for 24-bit mantissa.
- The *range* of the number depends on the number of bits in E.
  - $10^{38}$  to  $10^{-38}$  for 8-bit exponent.

41

### A Warning

- The representation for floating-point numbers as shown is just for illustration.
- The actual representation is a little more complex.
- In C:
  - float :: 32-bit representation
  - double :: 64-bit representation

42

## Representation of Characters and Strings

43

## Representation of Characters

- Many applications have to deal with non-numerical data
  - Characters and strings.
  - There must be a standard mechanism to represent alphanumeric and other characters in memory.
- Three standards in use:
  - Extended Binary Coded Decimal Interchange Code (EBCDIC)
    - Used in older IBM machines.
  - American Standard Code for Information Interchange (ASCII)
    - Most widely used today.
  - UNICODE
    - Used to represent all international characters.
    - Used by Java.

44

## ASCII Code

- Each individual character is numerically encoded into a unique 7-bit binary code.
  - A total of  $2^7$  or 128 different characters.
  - A character is normally encoded in a byte (8 bits), with the MSB not been used.
- The binary encoding of the characters follow a regular ordering.
  - Digits are ordered consecutively in their proper numerical sequence (0 to 9).
  - Letters (uppercase and lowercase) are arranged consecutively in their proper alphabetic order.

45

## Some Common ASCII Codes

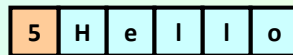
'A'	::	41 (H)	65 (D)
'B'	::	42 (H)	66 (D)
.....			
'Z'	::	5A (H)	90 (D)
'a'	::	61 (H)	97 (D)
'b'	::	62 (H)	98 (D)
.....			
'z'	::	7A (H)	122 (D)

'0'	::	30 (H)	48 (D)
'1'	::	31 (H)	49 (D)
.....			
'9'	::	39 (H)	57 (D)
'('	::	28 (H)	40 (D)
'+'	::	2B (H)	43 (D)
'?'	::	3F (H)	63 (D)
'\n'	::	0A (H)	10 (D)
'\0'	::	00 (H)	00 (D)

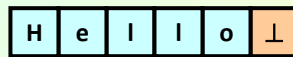
46

## Character Strings

- Two ways of representing a sequence of characters in memory.
  - The first location contains the number of characters in the string, followed by the actual characters.



- The characters follow one another, and is terminated by a special delimiter.



47

## String Representation in C

- In C, the second approach is used.
  - The '\0' character is used as the string delimiter.

- Example:

"Hello" → 

H	e	l	l	o	\0
---	---	---	---	---	----

- A null string "" occupies one byte in memory.
  - Only the '\0' character.

48