

# Number systems

- Refs: O'H&Bryant sect 2.1–2.3, Null&Lobur sect 2.3–2.4, Tanenbaum appendix A, related web links
- decimal
- octal
- hexadecimal
- binary
  - negative numbers, two's complement, overflow, sign extension
- which is the odd one out, and why?
- the Digital Logic Level (of computer organization): gates
- Admin notices:
  - my office hours now up
  - tute week 2: 1 hour, handout Fri 10am week 1
  - assesement dates (Ass1 & MSE)

# Decimal

- base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- $103_{10} = 1 \times 100 + 0 \times 10 + 3 \times 1$   
 $= 1 \times 10^2 + 0 \times 10^1 + 3 \times 10^0$
- $5.702_{10} = 5 \times 10^0 + 7 \times 10^{-1} + 0 \times 10^{-2} + 2 \times 10^{-3}$
- ...  $10^3$   $10^2$   $10^1$   $10^0$  .  $10^{-1}$   $10^{-2}$   $10^{-3}$  ...

# Octal

- base 8: 0, 1, 2, 3, 4, 5, 6, 7
- ...  $8^3$   $8^2$   $8^1$   $8^0$  .  $8^{-1}$   $8^{-2}$   $8^{-3}$  ...

- octal  $\rightarrow$  decimal:

- $103_8 = 1 \times 8^2 + 0 \times 8^1 + 3 \times 8^0$   
 $= 1 \times 64 + 0 \times 8 + 3 \times 1$   
 $= 67_{10}$

- $42_8 = x_{10} ?$

- decimal  $\rightarrow$  octal:

- $132_{10} = ?_8$   
 $132/8 = 16$  remainder 4  
 $16/8 = 2$  remainder 0  
 $2/8 = 0$  remainder 2  
 $\rightarrow 132_{10} = 204_8$

- $42_{10} = x_8 ?$

# Hexadecimal

- base 16: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- ...  $16^3$   $16^2$   $16^1$   $16^0$  .  $16^{-1}$   $16^{-2}$   $16^{-3}$  ...

- hexadecimal  $\rightarrow$  decimal:

- $10E_{16} = 1 \times 16^2 + 0 \times 16^1 + E \times 16^0$   
 $= 1 \times 256 + 0 \times 16 + 14 \times 1$   
 $= 270_{10}$

- $42_{16} = x_{10} ?$

- decimal  $\rightarrow$  hexadecimal:

- $174_{10} = ?_{16}$ :

- $174/16 = 10$  remainder 14 (E)

- $10/16 = 0$  remainder 10 (A)

- $\rightarrow 174_{10} = AE_{16}$

- $42_{10} = x_{16} ?$

- hexadecimal is widely used in computer systems

# Binary

- base 2: 0,1 (true/false) (on/off)
- ...  $2^3$   $2^2$   $2^1$   $2^0$  .  $2^{-1}$   $2^{-2}$   $2^{-3}$  ...  
... 8 4 2 1 . 1/2 1/4 1/8 ...
- binary  $\rightarrow$  decimal:
  - $1001_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$   
 $= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$   
 $= 9_{10}$
  - $1101_2 = x_{10}$  ?
- decimal  $\rightarrow$  binary
  - $19_{10} = ?_2$   
 $19/2 = 9$  remainder 1  
 $9/2 = 4$  remainder 1  
 $4/2 = 2$  remainder 0  
 $2/2 = 1$  remainder 0  
 $1/2 = 0$  remainder 1  
 $\rightarrow 19_{10} = 10011_2$
  - $42_{10} = x_2$  ?

# Binary Integer Addition and Subtraction

- as per decimal

from right to left, propagating 'carries'

- addition:

$$\begin{array}{r} 11010_2 + \\ \underline{01011_2} = \\ 100101_2 \end{array} \qquad \begin{array}{r} 26_{10} + \\ \underline{11_{10}} = \\ 37_{10} \end{array}$$

- subtraction:

$$\begin{array}{r} 10101_2 - \\ \underline{01011_2} = \\ 01010_2 \end{array} \qquad \begin{array}{r} 21_{10} - \\ \underline{11_{10}} = \\ 10_{10} \end{array}$$

## Negative Integers

- positive numbers (one byte):

0000 0000    0  
0000 0001    1  
0000 0010    2  
...            ...  
1111 1111<sub>2</sub>    255<sub>10</sub>

- how can we represent negative numbers? e.g.  $-42_{10}$  ?

$42_{10} = 0001\ 0101_2$ , but the negative?

- reserve one bit for sign (highest bit, most significant bit)

- positive numbers: as above but largest is:

0 111 1111<sub>2</sub>    127<sub>10</sub>

- negative numbers:

1 000 0000     $-0$

1 000 0001     $-1$

...            ...

1 111 1111<sub>2</sub>     $-127_{10}$

- **problems:** two zeros (!), addition is not simple:  $4 + (-1) = -5$

## Two's Complement Representation of Signed Integers

- most common system
  - only one zero; left-most bit indicates sign
  - normal binary addition gives correct result
- rule: to negate a number, flip the bits to the left of the right-most 1
  - equivalently: flip all the bits and add 1 to the result

- examples:

0000 0101	5	1111 1000	-8
1111 1011	-5	0000 1000	8

- addition and subtraction (for 4-bit arithmetic: discard 5th bit, if any):

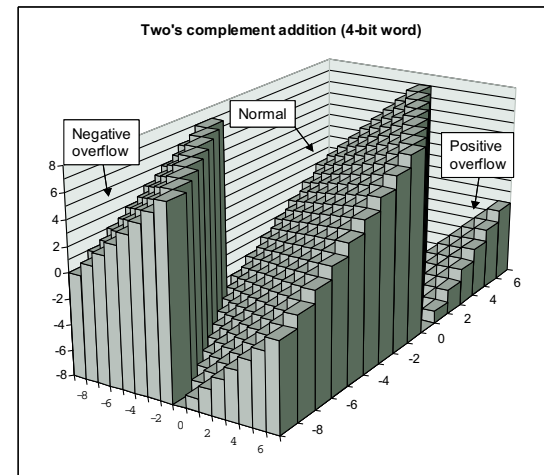
0100	4 +	1101	-3 +
<u>1111</u>	-1 =	<u>1110</u>	-2 =
10011	3	11011	-5

0111	7 +	1010	-6 +
<u>0010</u>	2 =	<u>1010</u>	-6 =
1001	-7 (or 9?)	10100	4 (or -12?)

# Overflow

- overflow occurs when the result cannot be represented in the given number of bits, because the magnitude is too great

O'H&Bryant fig 2.14: 4-bit 2's c. add



- rules:
  - $+x + -y \rightarrow$  no overflow
  - $-x + +y \rightarrow$  no overflow
  - $+x + +y \rightarrow$  overflow if result is  $-z$
  - $-x + -y \rightarrow$  overflow if result is  $+z$

- example:

$$\begin{array}{r}
 0\ 0001 + \quad 1 + \\
 \underline{0\ 1111} = \quad 15 = \\
 1\ 0000 \quad \quad 0
 \end{array}$$

- discussion point: what are the possible consequences? when should it be checked?

## Sign Extension

- example: we want to convert a 10-bit number into a 16-bit number (needed later for PeANUt operands)

- easy for positive case:

0110 6  
0000 0110 6

- negative case?

1010 -6  
0000 1010 10

- solution: fill the extra digits with copies of the sign bit (the leftmost bit)

- positive case:

0111 7  
0000 0111 7

- negative case:

1001 -7  
1111 1001 -7

## The Digital Logic Level: Gates

- Ref: Null&Lobur sect 2.3–2.4, Tanenbaum ch 3
- Null&Lobur fig 1.3: digital logic is lowest level of computer organization
- digital circuit elements (including 1-bit memory cells) are made from gates  
(in turn, from 1 or more transistors or switches, Tanenbaum fig 3.1)
- Null&Lobur figs 3.1-2: NAND and NOR gates manipulate voltage levels (0- low, 1 - high) according to truth tables (Boolean logic)
  - O'H&Bryant p 45: set of colors forming an 8-element boolean algebra
- Null&Lobur figs 3.11-13: from these, higher-level components such as half and full adders are constructed
  - how could these be combined to implement an integer add operation?
- discussion point: why is binary the representation used in computers?