COMP 250
INTRODUCTION TO COMPUTER SCIENCE

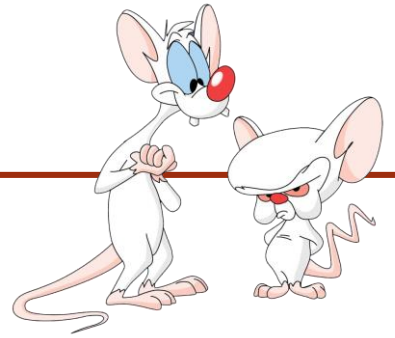
Lecture 3 – Number Representation and Base Conversions

Giulia Alberini, Fall 2018

FROM LAST CLASS

- Logarithms
- Mod Function

WHAT ARE WE GOING TO DO TODAY?



- Number Representation
- Binary numbers
- Base conversion
- Binary Arithmetic

NUMBER REPRESENTATION

BASE 10

When we refer to a decimal (base 10) number, like 5764, we are referring to the value obtained by carrying out the following addition:

$$5000 + 700 + 60 + 4$$

That is, we add together:

- Ones: 4
- Tens: 6
- Hundreds: 7
- Thousands: 5

EXPONENTIAL NOTATION

- Reminder

$10^3 = 10 \cdot 10 \cdot 10 =$ 'ten raised to the power of 3'

$n^0 = 1$ for any number n.

- So, using the exponential notation, we can write

$$5764 = 5 \cdot 10^3 + 7 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0$$

BASE 10

$$5764 = 5 \cdot 10^3 + 7 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0$$

NOTE:

- The digits of the number correspond to the coefficients of the powers of ten
- The position of the digit in the number determines to which power it is associated.

In a similar way, we can write numbers in other base:

- We use the digits that correspond to the coefficients on the corresponding powers (of the given base)

EXAMPLE

- What is the value of the base 5 number $(132)_5$?
- We need to first compute the corresponding powers of 5:

$$5^0 = 1$$
$$5^1 = 5$$
$$5^2 = 25$$

- Now we multiply them by the corresponding digit and sum the results together:

$$1 \cdot 5^2 + 3 \cdot 5^1 + 2 \cdot 5^0 = 25 + 15 + 2 = 42$$

IN GENERAL

Let b be an integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0 = \sum_{i=0}^{k} a_i b^i$$

Where k is a nonnegative integer, $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

The representation of n given above is called *the base b expansion of n*. It is denoted by

$$(a_k a_{k-1} \ldots a_1 a_0)_b$$

DIGITS

- NOTE: In order for every number to have a base b representation, and to assure that no number has more than one such representation, we use only the digits 0 through $(b-1)$ in any base b number. Example:
  - For base 10 numbers we use digits $0-9$.
  - For smaller bases we use a subset of these digits. For instance, for base 8, we used $0-7$.
  - For larger bases we need to have digits for values past 9. In **hexadecimal** (base 16), we use digits $0-9$ and $A-F$, where

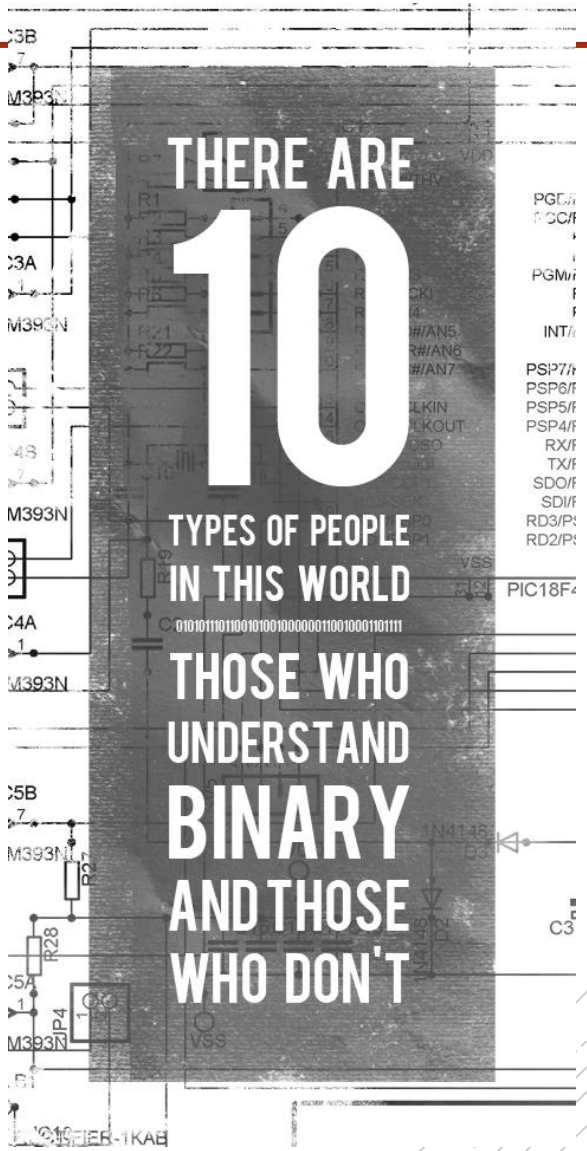
$$A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$$

- What is the value of $(5AF8)_{16}$?
  - $(5AF8)_{16} = 5 * 16^3 + 10 * 16^2 + 15 * 16^1 + 8 * 16^0 = 23{,}288$

NUMBER BASES

Important representations / bases:

- Decimal / base 10
- Binary / base 2
- Hexadecimal / base 16
- Although in everyday life we use decimal notation to express integers, it is often convenient to use bases other than 10. In particular, computers usually use binary notation when carrying out arithmetic, and programmers commonly use hexadecimal notation to refer to locations in memory.



NUMBER BASE CONVERSION

COUNTING IN BINARY

decimal	binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
:	:

COUNTING IN BINARY

decimal	binary
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
:	:

Fixed number of bits (typically 8, 16, 32, 64)

8 bits is called a "byte".

BINARY TO DECIMAL

- By definition, if we use the following binary representation $(a_k a_{k-1} \ldots a_1 a_0)_2$ for a number, then its value is

$$n = a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_1 2^1 + a_0 2^0 = \sum_{i=0}^{k} a_i \cdot 2^i$$

- NOTE: $a_i = 0$ or 1.

  Thus only the terms with the $a_i = 1$ will be left to sum!

BINARY TO DECIMAL - ALGORITHM

Given a binary number, do:

1. Compute the powers of 2 needed (as many as the binary digits in the number).

2. Identify the powers associated to the digits equal to 1.

3. Sum them all together.

i	2^i
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048

BINARY TO DECIMAL – EXAMPLE 1

Convert 11010_2 to a decimal number.

- Computer the powers
- Identify the powers associated to the digits equal to 1.
- Sum them together

Position/ Exponent	4	3	2	1	0
Powers of 2	16	8	4	2	1
Digits	1	1	0	1	0
Output:	$16 + 8 + 2 = \mathbf{26}$				

BINARY TO DECIMAL – EXAMPLE 2

Convert 11010101_2 to a decimal number.

Position/ Exponent	7	6	5	4	3	2	1	0
Powers of 2	128	64	32	16	8	4	2	1
Digits	1	1	0	1	0	1	0	1

Output: $128 + 64 + 16 + 4 + 1 = \mathbf{213}$

ANY BASE TO DECIMAL

- In general, given the base b representation $(a_k a_{k-1} \dots a_1 a_0)_b$ for a number, its value is

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b^1 + a_0 b^0 = \sum_{i=0}^{k} a_i \cdot b^i$$

- NOTE: $a_i \in \{0, 1, \dots, b-1\}$.

DECIMAL TO ANY OTHER BASE

$$(5764)_{10} = (\ ?\ )_b$$

We'll learn today an algorithm for doing so.

FROM THE QUOTIENT REMINDER THEOREM

Recall that, given any integer m, and **a positive** integer b, there exist **unique** integers q and r such that

$$m = q \cdot b + r$$

where $0 \leq r < b$.

That is, for any number m the following property holds

$$m = (\, m/b) * b \quad + \quad m \,\%\, b$$

We'll use this in our decimal-to-any base algorithm.

EXAMPLE

$$m = (m/b) * b \quad + \quad m \,\%\, b$$

Let $m = (5764)_{10}$ and $b = 10$.

- (integer) division by 10 = dropping rightmost digit

  $5764 / 10 = 576$

- Multiplication by 10 = shifting left by one digit

  $576 * 10 = 5760$

- Remainder of integer division by 10 = rightmost digit

  $5764 \,\%\, 10 = 4$

DECIMAL TO DECIMAL

What is 5764_{10} in decimal notation (base 10)? We can use the previous property to extract all the digits as follows:

$$
\begin{aligned}
5764 &= (5764/10) * 10 + 5764\%10 \\
&= 576 * 10 + 4 * 10^0 \\
&= ((576/10) * 10 + 576\%10) * 10 + 4 \\
&= 57 * 10^2 + 6 * 10 + 4 * 10^0 \\
&= ((57/10) * 10 + 57\%10) * 10^2 + 6 * 10 + 4 \\
&= 5 * 10^3 + 7 * 10^2 + 6 * 10 + 4 * 10^0 \\
&= ((5/10) * 10 + 5\%10) * 10^3 + 7 * 10^2 + 6 * 10 + 4 \\
&= 0 * 10^4 + 5 * 10^3 + 7 * 10^2 + 6 * 10 + 4 * 10^0
\end{aligned}
$$

DECIMAL TO DECIMAL

That is, all we need is to compute the quotients and keep track of the remainders:

$$5764/10 = 576\ R\ \mathbf{4}$$

$$576/10 = 57\ R\ \mathbf{6}$$

$$57/10 = 5\ R\ \mathbf{7}$$

$$5/10 = 0\ R\ \mathbf{5}$$

Note that taking the remainders from bottom to top gives us the answer.

DECIMAL TO BINARY

The same property of course works if we choose b to be 2:

$$m = m/2 * 2 + m \,\%\, 2$$

Then, if we think of m in its binary representation, e.g. $m = (10011)_2$,

- (integer) division by 2 = dropping rightmost digit

  $m/2 = (1001)_2$

- Multiplication by 2 = shifting left by one digit

  $(m/2) * 2 = (10010)_2$

- Remainder of integer division by 2 = rightmost digit

  $m \,\%\, 2 = (00001)_2$

DECIMAL TO BINARY

What is 13_{10} in binary? Using the same technique as before (but now with the division by 2) we extract the bits of the binary representation as follows:

$$13/2 = 6\ R\ \mathbf{1}$$

$$6/2 = 3\ R\ \mathbf{0}$$

$$3/2 = 1\ R\ \mathbf{1}$$

$$1/2 = 0\ R\ \mathbf{1}$$

Now, the base 2 representation comes from reading off the remainders from bottom to top!

$$13_{10} = \mathbf{1101_2}$$

DECIMAL TO BINARY – ALGORITHM

ALGORITHM
Constructing Base 2 Expansions

procedure *BinaryExpansion*(m)
$k := 0$
While $m > 0$
    $a_k := m\%2$
    $m := m/2$
    $k := k + 1$
return $(a_{k-1}, \ldots, a_1, a_0)$

This is called the remainder method.

EXAMPLE

To convert 57_{10} in binary:

$$57/2 = 28\ R\ \mathbf{1}$$

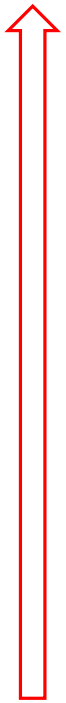
$$28/2 = 14\ R\ \mathbf{0}$$

$$14/2 = 7\ R\ \mathbf{0}$$

$$7/2 = 3\ R\ \mathbf{1}$$

$$3/2 = 1\ R\ \mathbf{1}$$

$$1/2 = 0\ R\ \mathbf{1}$$



Output: 111001

CHECK THE ANSWER!

We think $57_{10} = 111001_2$

$$111001_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$= 32 + 16 + 8 + 1$$

$$= 57$$

BASE CONVERSION – ALGORITHM

The technique just shown works for every base.

In general, given a base b and a decimal number m, repeat the following until the number is 0:

- Divide m by b and prepend the remainder of the division.
- Let the now m be the quotient of the previous division.

ALGORITHM
Constructing Base b Expansions

procedure *BaseExpansion*(m, b)
$k := 0$
While $m > 0$
    $a_k := m \% b$
    $m := m/b$
    $k := k + 1$
return $(a_{k-1}, \ldots, a_1, a_0)$

CONVERSION BETWEEN BINARY AND HEXADECIMAL

Consider the following conversion from binary to hexadecimal:

group 3 group 2 group 1 group 0

$$100100010101111\ (binary) = \mathbf{0100}\ \mathbf{1000}\ \mathbf{1010}\ \mathbf{1111}$$

$$4 \quad 8 \quad 10 \quad 15 \quad \text{(intermediate step through base 10)}$$

$$= 4 \quad 8 \quad A \quad F$$

Hence, $100100010101111_2 = 48AF_{16}$

Reversing the process is just as easy!

Why breaking the number into groups of 4 works?

- Group **0** in the binary representation is a number which is a multiple of 16^0
- Group **1** in the binary representation is a number which is a multiple of 16^1
- Group **2** in the binary representation is a number which is a multiple of 16^2
- Group **3** in the binary representation is a number which is a multiple of 16^3

USE OF HEXADECIMALS

- A common use of hexadecimal is to represent long bit strings.
- Hexadecimal numbers are indicated by adding the 0x prefix.
- Example 1: : 0010 1111 1010 0011

  2 f a 3

  We write 0x2fa3 or 0X2FA3.

- Example 2: 10 1100

  2 c

  We write 0x2c (10 1100), not 0xb0 (1011 00)

BINARY ARITHMETIC

ADDITION IN A GIVEN BASE

You can add in any base as long as you remember that:

- you "carry" when you have a sum that is greater than or equal to your base (instead of greater than or equal to 10)

- what you "carry" is the number of times you can pull out the base from your sum.

ADDITION IN BASE 2

Let's try it in binary.

$$\begin{array}{r} 1110 \\ +\ 1011 \\ \hline ? \end{array} \qquad \begin{array}{r} 18 \\ +\ 15 \\ \hline 33 \end{array}$$

ADDITION IN BASE 2

Let's try it in binary.

carry 1 1 1 0

$$\begin{array}{r} 1110 \\ +\ 1011 \\ \hline 11001 \end{array}$$

Digit 0

$0 + 1 = 1$

$0 + 1 = 1 = \mathbf{0} \cdot 2 + \mathbf{1}$

Digit 1

$1 + 1 + 0 =$

$1 + 1 + 0 = 2 = \mathbf{1} \cdot 2 + \mathbf{0}$

Digit 2

$1 + 0 + 1 =$

$1 + 0 + 1 = 2 = \mathbf{1} \cdot 2 + \mathbf{0}$

Digit 3

$1 + 1 + 1 =$

$1 + 1 + 1 = 3 = \mathbf{1} \cdot 2 + \mathbf{1}$

Digit 4

$0 + 0 + 1 =$

$0 + 0 + 1 = 1 = \mathbf{0} \cdot 2 + \mathbf{1}$

BINARY ADDITION – ALGORITHM

In general, let $a = (a_{n-1} \dots a_0)_2$, and $c = (c_{n-1}, \dots, c_0)_2$ be two numbers in base 2. Note that we can assume that they both have n digits since we can add 0s to the left if necessary. To compute the sum of these two numbers follow the algorithm on the right.

ALGORITHM
Summing numbers base 2

procedure $BinarySum(a, c)$
$carry := 0$
for $i = 0$ **to** $n - 1$ **do**
    $r_k := (a_i + b_i + carry)\%2$
    $carry := (a_i + b_i + carry)/2$
end for
$r_n := carry$
return $(r_n, \dots, r_1, r_0)$

GRADE SCHOOL ARITHMETIC

Recall addition, subtraction, multiplication, division.

There is nothing special about base 10.

These algorithms work for binary (base 2) too.

Indeed they work for any other base too.

- Char and Unicode, review of primitive data type conversions