CSE 1400 Applied Discrete Mathematics
Numeral Systems

Department of Computer Sciences
College of Engineering
Florida Tech
Fall 2011

1 Numeral Systems

There are several numeral systems used to write numbers. For instance, the natural number two thousand and ten can be written as the Roman numeral MMX or as 2010 in decimal Arabic numerals, or as 111 1101 1010 in binary Arabic numerals.

In mathematics and computing, numbers are written in a POSITIONAL system, where the position of a numeral determines its role in the computation of the value. The numeral systems commonly used in computing are decimal, binary, octal, and hexadecimal. For instance, 2010 can written in these four systems as

$$\begin{aligned}2010 &= 1\cdot 2^{10}+1\cdot 2^9+1\cdot 2^8+1\cdot 2^7+1\cdot 2^6+0\cdot 2^5+1\cdot 2^4+1\cdot 2^3+0\cdot 2^2+1\cdot 2^1+0\cdot 2^0\\ &= 3\cdot 8^3+7\cdot 8^2+3\cdot 8^1+2\cdot 8^0\\ &= 2\cdot 10^3+0\cdot 10^2+1\cdot 10^1+0\cdot 10^0\\ &= 7\cdot 16^2+D\cdot 16^1+A\cdot 16^0\end{aligned}$$

In general, a natural number n can be written as

$$n=\alpha_{n-1}\beta^{n-1}+\alpha_{n-2}\beta^{n-2}+\cdots+\alpha_1\beta^1+\alpha_0\beta^0$$

where $\beta = 2, 8, 10, 16$ is one of the common bases, and the coefficients α_k are natural numbers between 0 and $\beta-1$.

Positional representation is a "polynomial form." The number can be represented as a polynomial evaluated at the base β, and the coefficients of the polynomial are integers mod β.

1.1 The Decimal System

The decimal system is the common notation The alphabet of the decimal system is the set of digits

$$\mathbb{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

We'll need to expand the decimal alphabet later, so let's name it now: The alphabet for writing natural numbers is

$$\mathbb{A} = \mathbb{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Natural numbers (unsigned whole values) are written as strings of these decimal numerals or digits. The natural numbers are members of the set

$$\mathbb{N} = \{0, 1, 2, 3, \ldots, n, \ldots\}$$

For instance, $n = 3142768$ is a natural number. For readability it can be convenient to use commas to separate the digits, for instance $n = 3,142,768$. The glyphs used for decimal numerals come from Arabic numerals

0	1	2	3	4	5	6	7	8	9
٠	١	٢	٣	٤	٥	٦	٧	٨	٩

You know how to add digits to compute a sum digit s and a carry digit c, which can be written as an ordered pair (c, s) and interpreted to mean the natural number $10c + s$. The table below shows the sums for adding 0 through 9 to itself. Because addition is commutative the table is symmetric and only the lower triangle needs to be written.

	0	1	2	3	4	5	6	7	8	9
0	(0,0)									
1	(0,1)	(0,2)								
2	(0,2)	(0,3)	(0,4)							
3	(0,3)	(0,4)	(0,5)	(0,6)						
4	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)					
5	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(1,0)				
6	(0,6)	(0,7)	(0,8)	(0,9)	(1,0)	(1,1)	(1,2)			
7	(0,7)	(0,8)	(0,9)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)		
8	(0,8)	(0,9)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	
9	(0,9)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)

To write a integer (a signed whole value) the sign symbols $+$ and $-$ can inserted into the alphabet.

$$\mathbb{Z} = \{0, +1, -1, +2, -2, +3, -3, \ldots\}$$

For instance, $-31,42,768$, 6.0221415 is an integer.

Although it is common to drop the plus sign, a computer stores unsigned integers (natural numbers), say 3, differently from signed integer $+3$.

To write a rational number (a signed fractional value), the fraction symbol / can be included in the alphabet.

$$\mathbb{Q} = \{q : q = a/b,\ a,\ b \in \mathbb{Z},\ b \neq 0\}$$

For instance, $-3,142/768$ is a rational number.

For computability it can be convenient reduce a fraction to lowest terms, so that the numerator and denominator are relatively prime. For instance

$$-\frac{3,142}{768} = -\frac{1,571}{384}$$

A rational number q can also be written in fixed-point notation

$$q = a/b = \pm n.f$$

Here $n \in \mathbb{N}$ is the unsigned whole part of a/b and f is the fractional part of a/b written as a (possibly unbounded) sequence of digits.

A third way to write a rational number q is to use floating-point notation.

The alphabet

$$\mathbb{A} = \mathbb{D} \cup \{+, -, /\} \cup \{,, .\}$$

can be used to represent rational numbers. For instance,

$$-3,142/768 = -4.09114583333\cdots$$

For instance,

$$-409.114583333 = -4.09114583333 \times 10^{-2}$$

$$q = \pm d.f \times 10^e$$

where $d \neq 0$ is a non-zero digit, f is a finite string of digits, and e is a signed integer.

The requirement $d \neq 0$ is a NORMALIZATION, meaning it makes the floating point representation unique.

Notice that the number $\tilde{\pi} = 3.1415$ can be written in many ways:

$$\tilde{\pi} = 3.1415$$
$$\tilde{\pi} = 0.31415 \times 10^1$$
$$\tilde{\pi} = 0.031415 \times 10^2$$
$$\tilde{\pi} = 314.15 \times 10^{-2}$$
$$\tilde{\pi} = 31.415 \times 10^{-1}$$

1.2 *The Unary System*

Although of limited use, the unary system uses an alphabet with just one character, usually written |. The length of unary numbers quickly becomes unwieldy: To write twenty, you'd need to write down 20 strokes Consider writing $1,000,000$ in unary. But in some situations, for example keep track of small counts, using unary makes sense.

Write 𝍸, 𝍸|, 𝍸||, 𝍸||| for 20

One interesting thing about unary is that it does not have a symbol for zero; instead of writing 0, we write nothing at all to represent zero.

Please don't write "nothing at all" when you want to write zero in unary.

Addition in unary is easy to define.

$$|+|=||, \qquad |||+||||=𝍸||$$

1.3 *The Binary System*

The binary system uses an alphabet with two characters.

The characters 0 and 1 are called bits.

$$\mathbb{B} = \{0, 1\}$$

The natural numbers can be written in binary. Below are the first few natural number written in binary and decimal.

Do you see a pattern? Can you translate from one notation to another?

Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010
Decimal	0	1	2	3	4	5	6	7	8	9	10

The table below shows the sum and carry bits when addition is performed on two bits. For instance, the sum of 1 and 1 is 0, with a carry of 1. This can be represented as an ordered pair $(c, s) = (0, 1)$.

For readability, bit strings are often written in groups of four. Each 4-bit string is a hexadecimal numeral. For instance, $(0011\ 1001)_2 = (39)_{16}$.

Binary addition

+	0	1
0	(0, 0)	(0, 1)
1	(0, 1)	(1, 0)

Interestingly, both the sum and carry bits can be computed using Boolean operations on the input bits.

- The sum bit is the XOR of the input bits.
- The carry bit is the AND of the input bits.

1.4 The Hexadecimal System

The hexadecimal system uses an alphabet with sixteen characters.

$$\mathbb{H} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

For readability, 4-bit strings are often converted to their hexadecimal equivalent. For instance, $(0011\ 1001)_2 = (39)_{16}$.

Below are the first few natural number written in hexadecimal and decimal.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A
Decimal	0	1	2	3	4	5	6	7	8	9	10
Hexadecimal	B	C	D	E	F	10	11	12	13	14	15
Decimal	11	12	13	14	15	16	17	18	19	20	21

1.5 The Sexagesimal System

Why are there 60 seconds in a minute? Why are there 60 minutes in an hour? Why are there 360 degrees in an circle? What is the history of base 60?

1.6 Finite Constraints

In practice, finite constraints limit our ability to compute. For instance, the central processing unit (CPU) of a computer contains a few registers, a REGISTER SET, each of which can hold a few bits. Let's suppose we want to store the 8 bits 1011 1000 which could represent for the decimal natural number 184. There are two ways in which the bits may be stored in a register: big- or little-endian.

0	1	2	3	4	5	6	7
1	0	1	1	1	0	0	0

An 8-bit register (*big-endian*).

We will use big-endian storage.

0	1	2	3	4	5	6	7
0	0	0	1	1	1	0	1

An 8-bit register (*little-endian*).

The size of a register is the number of cells it has. The content of a cell is a bit, a 0 or a 1.

Data is stored in data registers. Instructions are stored in instruction registers. Modern computers have many registers of different types.

Common register sizes are 8-bit, 16-bit, 32-bit, and 64-bit. Let's pretend we're working on a computer that has 8-bit registers and that there are $n = 2^k \cdot 8 = 2^{k+3}$ registers in a CPU. Let's also pretend that each bit is ADDRESSABLE, meaning we can read and write individual

In today's practice a machine word may be 64 bits, organized as 8 bytes, and bytes may be addressable.

bits in the register set.

Register Address	Bit Address							
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
⋮	⋮							
$n-1$	$8n-8$	$8n-7$	$8n-6$	$8n-5$	$8n-4$	$8n-3$	$8n-2$	$8n-1$

In theory, the content of a cell can be a higher-order CHARACTER. The content of a register (the string of characters) is called a word. The size of the register is called the word length. The concepts of an empty word can be useful. Call the empty word λ. It's length is 0 and it contains no characters.

Two fundamental questions are:

1. Given that words are n characters long, how many different words can be written? And its inverse question:
2. Given the need to write N different words, what size register should be used?

m^n length n words can be written if there are m characters in the alphabet.

Word length (register size) must be at least $n = \lceil \log_m N \rceil$ bits to write N different words over an alphabet of m characters.

In given instances it is not hard to answer these questions.

Pretend the computer's register size is 8 and any decimal numeral can be held in each of the 8 places. Then any natural number from 0000 0000 to 9999 9999 can be stored in a register. That is, there are $100,000,000 = 10^8$ different words that can be written.

Registers are "padded," meaning 0's fill a register, for instance 77, would be stored as 0000 0077.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	7	7

An 8-digit register (*big-endian*).

If the register size is 8 (a byte) and any hexadecimal numeral can be held in place, then any natural number from 0 to $(FFFF\ FFFF)_{16} = 16^8 - 1 = 4,294,967,295$ can be represented in hardware. That is, there are $4,294,967,296 = 16^8$ different words that can be written. The hexadecimal number $(DEAD\ CAFE)_{16}$ who be stored as

0	1	2	3	4	5	6	7
D	E	A	D	C	A	F	E
7	6	5	4	3	2	1	0

An 8-hexadecimal register (*big-endian*).

In practice, computer registers hold bits: 0 or 1.

On an 8-bit computer any natural numbers from 0 to $2^8 - 1 = 255$ can be stored in a register.

On an 16-bit computer any natural numbers from 0 to $2^{16} - 1 = 65,535$ can be stored in a register.

On an 32-bit computer any natural numbers from 0 to $2^{32} - 1 = 4,294,967,295$ can be stored in a register.

On an 64-bit computer any natural numbers from 0 to $2^{64} - 1 = 18,446,744,073,709,551,615$ can be stored in a register.

The numbers 0, 1, 3, 7, 15, 31, 63, 127, 255, $\ldots 2^n - 1$ are called Mersenne numbers.

Of course, only one number is stored at a given time.

2^{16} is 64 kilobits or 8 kilobytes.

2^{32} is 4 gigabits or 1/2 gigabytes.

2^{64} is 16 exabits or 2 exabytes.

Theorem 1 (String length → count of different words)**.** *There are 2^n different bit strings of length n. More generally, there are m^n different words of length n over an alphabet $\mathbb{A}$ containing m characters,*

Proof. Reason inductively:

- Study initial cases.
  - There is $1 = 2^0$ bit string of length 0: The empty string λ.
  - There are $2 = 2^1$ bit strings of length 1: 0 and 1.
  - There are $4 = 2^2$ bit strings of length 2: 00, 01, 10 and 11.
- Make a prediction: 2^n is the number of different bit strings of length n.
- Establish the conditional implication:

  "If there are 2^n bit strings of length n, then there are 2^{n+1} bit strings of length $n + 1$."

  Pretend there are 2^n bit strings of length n. For each one of them construct new bits strings of length $n + 1$ by appending a 0 once and a 1 second time. This process creates $2^n + 2^n = 2^{n+1}$ different strings of length $n + 1$. There can be no more.

  For instance, the four-bit string 1011 generates two different five-bit strings 01011 and 11011.

□

Theorem 2 (Mersenne Numbers)**.** *The largest natural number that can be written using n bits is the Mersenne numbers number $M_n = 2^n - 1$.*

Proof. Starting with 0 and and ending at $2^n - 1$, 2^n consecutive natural numbers are named using bit strings of length n. Reason inductively:

- Initial cases
  - Using no bits, only the empty word can be written. It is convenient to define the value of the empty word to be 0 so that $2^0 - 1 = 0$ is the largest natural number that can be written using $n = 0$ bits.

  - Using 1 bit, the largest natural number that can be written is $1 = 2^1 - 1$.
  - Using 2 bits, the largest natural number that can be written is $11 = 3 = 2^2 - 1$.
  - Using 3 bits, the largest natural number that can be written is $111 = 7 = 2^3 - 1$.
- Make a prediction: $2^n - 1$ is largest natural number that can be written using n-bits.
- Establish a conditional implication.

  Pretend $2^n - 1$ is the largest natural number that can be written using n bits. The natural number 2^n, written in binary, is 1 followed by n zeros. The sum of 2^n and $2^n - 1$ is the largest natural number that can be written using $n + 1$ bits. And this sum is $2^n + (2^n - 1) = 2^{n+1} - 1$.

□

Now consider the inverse problem: How long do words have to be to name m different things?

Lemma 1. *If real number x is not an integer, then $\lfloor x \rfloor + 1 = \lceil x \rceil$*

Proof. Compute an integer n such that $n - 1 < x < n$. Then $\lfloor x \rfloor = n - 1$ and $\lceil x \rceil = n$ □

Theorem 3 (Count of different words → string length)**.** *To name m different* THINGS *requires words of length*

$$\min \{ \lceil \lg m \rceil , \lfloor \lg m \rfloor + 1 \}$$

Proof. Consider the sequence of powers of 2.

$$\left\langle 2^0, 2^1, 2^2, 2^3, 2^4, \ldots \right\rangle = \langle 1, 2, 4, 8, 16, \ldots \rangle$$

If m is one of these powers of 2, say $m = 2^n$, then, by Theorem 1, there are m things can be named using bit strings of length

$$n = \lg m = \lceil \lg m \rceil < \lfloor \lg m \rfloor + 1$$

Otherwise let m be natural number that is not a power of 2. Compute an exponent $n \geq 1$ such that

$$2^{n-1} < m < 2^n$$

Any value m in this range can be written using n bits.

Take the logarithm base 2 of the terms in the above inequalities to get

$$n - 1 < \lg m < n$$

Then take the ceiling or floor of the logarithm to get

$$n = \lfloor \lg m \rfloor + 1 = \lceil \lg m \rceil = n$$

Reason inductively:

- Study initial cases.
  - The words 0 and 1 name 2 things, and $1 = \lceil \lg 2 \rceil$.
  - The words 00, 01, and 10 name 3 things, and $2 = \lceil \lg 3 \rceil = \lfloor \lg 3 \rfloor + 1$.
  - The words 00, 01, 10, and 11 name 4 things, and $2 = \lceil \lg 4 \rceil$.
  - The words 000, 001, 010, 011, and 100 name 5 things, and $3 = \lceil \lg 5 \rceil = \lfloor \lg 5 \rfloor + 1$.
- Establish a conditional implication.

  Pretend you can name m THINGS using bit strings of length n. If $m < 2^n - 1$, then $m + 1$ THINGS can be named using bit strings of length n and n.

□

2 *Problems on Numeral Systems*

1. The following natural numbers are written using their binary names. What are their hexadecimal names?

   (a) 0000 1001.
   (b) 1111 1111.
   (c) 011 1100 1100.
   (d) 110 1011 1010.

2. Why is it awkward to group the bits by fours from left-to-right when converting a binary string to hexadecimal?
3. Expand the hexadecimal names below to their binary equivalent.

   (a) *FE*.
   (b) *ACED*.
   (c) 577.
   (d) 3141.

4. Explain how to covert from binary-to-octal and octal-to-binary.
5. How many length n decimal strings are there?
6. How many length n octal strings are there?
7. How many length n hexadecimal strings are there?
8. How many length n strings of lower case English letters are there?
9. How many length n strings of lower case English letters and decimal numerals are there?
10. How many length n strings of lower case Greek letters are there?
11. Write the number twenty-five in Roman numerals, in decimal, in binary, in octal and in hexadecimal.
12. How many THINGS can be named using strings of the given length and alphabet?

(a) Length 4 decimal strings.
(b) Length 4 binary strings.
(c) Length 4 octal strings.
(d) Length 4 hexadecimal strings.
(e) Length n octal strings.
(f) Length n decimal strings.
(g) Length n binary strings.
(h) Length n hexadecimal strings.

13. How many characters are needed to name the given number of THINGS ?

(a) 5 THINGS in binary.
(b) 50 THINGS in decimal.
(c) 500 THINGS in hexadecimal.
(d) n THINGS in binary.
(e) n THINGS in decimal.
(f) n THINGS in hexadecimal.

14. Of the 128 ASCII characters, 94 are printable. How many printable strings of ASCII characters are there of the given length?

(a) Length 0 strings.
(b) Length 1 strings.
(c) Length 2 strings.
(d) Length 3 strings.
(e) Length $n - 1$ strings.
(f) Strings of length 0 or 1.
(g) Strings of length 0 or 1 or 2.
(h) Strings of length 0, 1, 2, . . . , $(n - 1)$.

15. Pretend you need to black-box test a program by entering each and every input combination to verify the output is correct for each input. Pretend the set of input combinations is a set of strings of length 1 through 10 drawn from an alphabet of 94 symbols. How many input combinations are there?