COMP 1113

1000011 1001111 1001101 1010000

Applied Mathematics

1000101001

Instructor: Paul Rozman

Office: SW2-229

Phone: 604-451-8718

E-mail: prozman@bcit.ca

Positional Number Systems

- Positional number systems are number systems in which the significance of a digit depends on its position

$$2^2 \; 2^1 \; 2^0$$

Fours Twos Ones

$$(110)_2$$

Binary
(BASE 2)

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

- Each positional number system is determined by a *base* that indicates how many separate symbols are used to represent numbers

Useful Bases in Computer Science

- Only some of the possible bases are of general use in computer science

	REASON FOR USE
Decimal (BASE 10) 0 1 2 3 4 5 6 7 8 9 10 …	Humans use a decimal system
Binary (BASE 2) 0 1 10 …	Representation of data in computers uses two states
Octal (BASE 8) 0 1 2 3 4 5 6 7 10 …	Useful abbreviations of binary: shorter to write, not too many symbols
Hexadecimal (BASE 16) 0 1 2 3 4 5 6 7 8 9 A B C D E F 10 …	

Uses of Hex

Colour On Web Pages

Triplets - Red:Green:Blue (RGB)

255:255:255 decimal

Color	Hexadecimal	Color	Hexadecimal	Color	Hexadecimal	Color	Hexadecimal
aqua	#00FFFF	gr			#000080	silver	#C0C0C0
black	#000000	gr			#808000	teal	#008080
blue	#0000FF	lim			#800080	white	#FFFFFF
fuchsia	#FF00FF	ma			#FF0000	yellow	#FFFF00

R=255
G=255
B=204

Hex: FFFFCC

Cascading Style Sheet (CSS) Code:

background a paragraph p{ color:#FFFFCC}

Uses of Hex

Characters in URL addresses
Space = %20 is hex code for space in ASCII (32)

e.g. http://www.paulssite.fake/my%20Stupid%20frontpage%20Name.html

XML: Characters can be referred to by &#xhhhh

e.g. Greek Lower Case α = &#x03B1

Hex and Octal are useful abbreviations of binary:

shorter to write, not too many symbols.

The above codes would be significantly longer to write in Binary

Bits and Bytes

10110010_2

- In a binary representation, a single binary digit is a *bit*
- Represents the smallest unit of "information" – the answer to a yes-no (true-false) question
- A collection of eight bits is a *byte*
- Depending on the computer architecture, a *word* may be defined as a collection of 2, 4 or 8 bytes (16, 32 or 64 bits)

Larger Collections

- The size of large collections of bits can be given in terms of prefixes similar to metric prefixes. Originally IEC standard

	Abbreviation	Number (of bits or bytes)
kibi bit	Ki b	2^{10} (approx. 10^3 = 1 kilo)
mebi	Mi	2^{20} (approx. 10^6 = 1 Mega)
gibi	Gi	2^{30} (approx. 10^9 = 1 Giga)
tebi	Ti	2^{40} (approx. 10^{12} = 1 Tera)
pebi	Pi	2^{50} (approx. 10^{15} = 1 Peta)
exbi	Ei	2^{60} (approx. 10^{18} = 1 Exa)
zebi	Zi	2^{70} (approx. 10^{21} = 1 Zetta)
yobi	Yi	2^{80} (approx. 10^{24} = 1 Yotta)

kibi =kilo binary

Sizes and Hard drives

Hard Drive Sizes

File sizes

Hard Drive too Small

Symbol (Bytes)	Decimal Meaning	Binary Meaning	Difference in Bytes	% Difference
1 KB	10^3	2^{10}	24	2.40
1 MB	10^6	2^{20}	48576	4.86
1 GB	10^9	2^{30}	73741824	7.37
1 TB	10^{12}	2^{40}	99,511,627,776	9.95
1 PB	10^{15}	2^{50}	125,899,906,842,624	12.59

- https://en.wikipedia.org/wiki/Hard_disk_drive#Units
- https://wiki.ubuntu.com/UnitsPolicy

Converting a Base *b* to Decimal

INTEGERS

- Read the number in each column as a multiplier for the corresponding power of the base

Octal (BASE 8)

$$\begin{array}{ccc} 8^2 & 8^1 & 8^0 \end{array}$$

$$(263)_8 = 2 \times 8^2 + 6 \times 8^1 + 3 \times 8^0$$

$$= 128 + 48 + 3 = (179)_{10}$$

Hexadecimal (BASE 16)

$$\begin{array}{ccc} 16^2 & 16^1 & 16^0 \end{array}$$

$$(2AF)_{16} = 2 \times 16^2 + A \times 16^1 + F \times 16^0$$

$$= 512 + 160 + 15 = (687)_{10}$$

Converting a Base *b* to Decimal
INTEGER AND FRACTIONAL PARTS

- Read the number in each column to the right of the radix point as a multiplier for the corresponding *negative* power of the base

Binary (BASE 2)

$2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$

$(101.011)_2 = 4 + 0 + 1 + 0 + 1/4 + 1/8 = (5.375)_{10}$

radix point

Converting Decimal to a Base b

INTEGERS

- Divide the decimal representation by the base b and record the *remainder*
  - The result of the division should be the number of times the base b goes *evenly* into the decimal representation (integer division)

Convert $(25)_{10}$ to binary.

25 /2 = 12 **rem 1**

- Divide the result by the base b. Record the remainder.

12 /2 = 6 **rem 0**

- Repeat until the result of the division is 0.

Converting Decimal to a Base b

INTEGERS

$25 / 2 = 12$	rem **1**
$12 / 2 = 6$	rem **0**
$6 / 2 = 3$	rem **0**
$3 / 2 = 1$	rem **1**
$1 / 2 = 0$	rem **1**

$$(25)_{10} = (11001)_2$$

Converting Decimal to a Base *b*

FRACTIONAL PARTS

- To convert the fractional part of a number, multiply the decimal representation by the base ***b*** and record the *integer part* of the result

Convert $(0.6875)_{10}$ to binary.

0.6875 x2 = 1.375 1

- Reset the integer part of the result to zero and multiply again by the base ***b***. Record the integer part of the result.

0.375 x2 = 0.75 0

- Repeat until the result of the multiplication is 0.

Converting Decimal to a Base *b*

FRACTIONAL PARTS

$0.6875 \times 2 = 1.375$	1
$0.375 \times 2 = 0.75$	0
$0.75 \times 2 = 1.5$	1
$0.5 \times 2 = 1.0$	1
$0.0 \times 2 = 0.0$	0

$$(0.6875)_{10} = (0.1011)_2$$

Converting Decimal to a Base *b*

FRACTIONAL PARTS

$$(0.6875)_{10} = (0.1011)_2$$

$$\begin{array}{ccccc} & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\ (0. & 1 & 0 & 1 & 1)_2 \\ & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} & \frac{1}{16} \end{array}$$

- When we multiply the decimal representation by the base 2, the integer part of the result is 1 only if the number is greater than or equal to ½
  - So the binary representation must have a 1 in the ½'s column

$$= 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} + 1 \times \frac{1}{16}$$

Converting Decimal to a Base b

FRACTIONAL PARTS

$$(0.6875)_{10} = (0.1011)_2$$

$$\begin{array}{ccccc} & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\ (0. & 1 & 0 & 1 & 1)_2 \\ & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} & \frac{1}{16} \end{array}$$

- Removing the integer part of the result is the same as subtracting ½ from the original
  - Multiplying the fractional part of the result again by 2, the integer part of the result will be 1 only if the number is greater than or equal to 1/4.

$$= 1 \times \frac{1}{2} + 0 \times \frac{1}{4} + 1 \times \frac{1}{8} + 1 \times \frac{1}{16}$$

Converting Decimal to a Base *b*
FRACTIONAL PARTS

Stopping Conditions

- If *fractional part =ZERO* then STOP
- If *fractional part repeats* then STOP
- If *Too Many bits* then STOP

Converting Between Bases When Neither is Decimal

- In general, the easiest way to convert from one base to another is to first convert the number in its original base to decimal and then convert to the destination base.

Convert $(14.3)_5$ to binary.

- Convert to decimal

$$(14.3)_5 = 1 \times 5^1 + 4 \times 5^0 + 3 \times 5^{-1}$$

$$= 5 + 4 + 0.6$$

$$= (9.6)_{10}$$

Converting Between Bases When Neither is Decimal

- Convert to destination base

$$(9.6)_{10}$$

Integer Part		Fractional Part	
$9 / 2 = 4$	rem **1**	$0.6 \times 2 = 1.2$	int **1**
$4 / 2 = 2$	rem **0**	$0.2 \times 2 = 0.4$	int **0**
$2 / 2 = 1$	rem **0**	$0.4 \times 2 = 0.8$	int **0**
$1 / 2 = 0$	rem **1**	$0.8 \times 2 = 1.6$	int **1**
			⋮

$$(14.3)_5 = (9.6)_{10} = (1001.\overline{1001})_2$$

Converting Between Bases
DIFFERENCES IN ACCURACY

- Some numbers that have a finite representation in one base have an infinitely repeating representation in another base

Example: A finite representation in decimal may become infinite in binary.

$$(0.6)_{10} = (0.\overline{1001})_2$$

- Only a finite number of digits are actually stored in a computer's memory, so an infinitely repeating representation must be *truncated* or *rounded off*, and this results in a loss of accuracy

Converting Between Bases

DIFFERENCES IN ACCURACY

Assume that a particular computer **truncates** binary representations after 4 digits. What is the loss of accuracy in storing the number $(0.6)_{10}$

$$(0.6)_{10} = (0.\overline{1001})_2$$

$$(0.5625)_{10} = (0.1001)_2$$

$$(0.0375)_{10}$$

The 4 digit representation **under**estimates the number by $(0.0375)_{10}$

Converting Between Bases

DIFFERENCES IN ACCURACY

Assume that a particular computer **rounds off** binary representations to 4 digits. What is the loss of accuracy in storing the number $(0.6)_{10}$

$$(0.6)_{10} = (0.\overline{1001})_2$$

$$(0.625)_{10} = (0.1010)_2$$

$$-(0.025)_{10}$$

The 4 digit representation **over**estimates the number by $(0.025)_{10}$

Converting Octal to Binary

- When the original base is a *power* of the destination base, we can take advantage of a shortcut that allows us to directly convert without converting to decimal in between
  - It is this shortcut that makes both octal and hexadecimal useful in computer science

Octal (BASE 8) *Binary* (BASE 2)

$$8 = 2^3$$

Each octal digit can be directly converted to a 3-digit binary number

$$(614.7)_8 = (110001100.111)_2$$

Converting Hexadecimal to Binary

- When the original base is a *power* of the destination base, we can take advantage of a shortcut that allows us to directly convert without converting to decimal in between
  - It is this shortcut that makes both octal and hexadecimal useful in computer science

Hexadecimal (BASE 16) *Binary* (BASE 2)

$16 = 2^4$

Each hex digit can be directly converted to a 4-digit binary number

$$(B7.C)_{16} = (10110111.1100)_2$$

Converting Octal to Binary

- When the *destination* base is a power of the *original* base, we can take advantage of the same shortcut in reverse

Binary (BASE 2) *Octal* (BASE 8)

$2^3 = 8$

Each group of 3 binary digits can be directly converted to an octal digit

$$(010001110.100)_2 = (216.4)_8$$

Start the grouping from the radix point and pad the number with leading and/or trailing zeros as necessary to make groups of 3

Converting Hexadecimal to Binary

- When the *destination* base is a power of the *original* base, we can take advantage of the same shortcut in reverse

Binary (BASE 2) *Hexadecimal* (BASE 16)

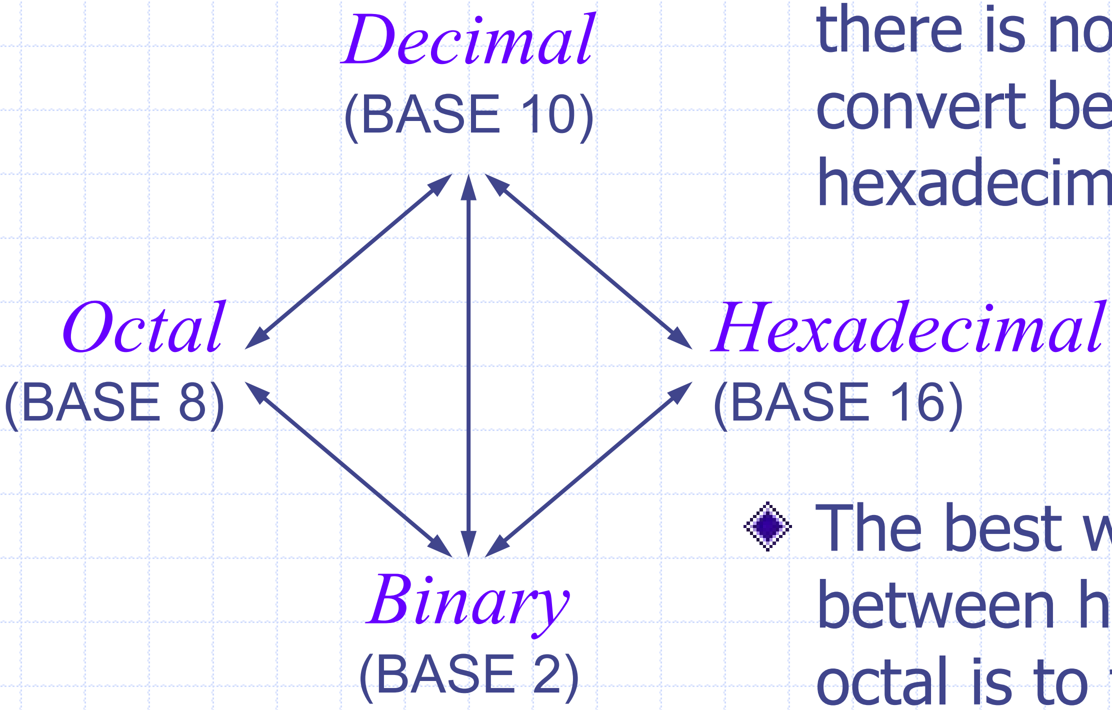
$$2^4 = 16$$

Each group of 4 binary digits can be directly converted to a hex digit

$$(\underbrace{0100}\,\underbrace{1110}.\underbrace{1100})_2 = (4E.C)_{16}$$

Start the grouping from the radix point and pad the number with leading and/or trailing zeros as necessary to make groups of 4

Converting Between Octal and Hexadecimal



- Since 16 is **not** a power of 8, there is no direct shortcut to convert between octal and hexadecimal
- The best way to convert between hexadecimal and octal is to first convert the original base to binary, then to the destination base

Addition of Two Numbers in Different Bases

$9 + E = 23 \Rightarrow 23 - 16 = 7$

$1 + B + A = 22 \Rightarrow 22 - 16 = 6$

- Two numbers can be conveniently added only when they are expressed in the same base
- In each column, if the result of adding the two digits equals or exceeds the base, a carry-out of 1 is added to the next column to the left
- If the carry-out is 1, the digit recorded at the bottom of the column is the result of the addition, *minus the base*
- If the carry-out is 0, the digit recorded at the bottom of the column is just the result of the addition

Carry-Out

$$
\begin{array}{r}
1\ 1 \\
(2B9)_{16} \\
+\ (3AE)_{16} \\
\hline
(667)_{16}
\end{array}
$$

Addition of Two Numbers
OVERFLOW

- It is important to remember that in an actual computer implementation, the result of the addition must be recorded on a register having a fixed number of digits
- If the carry-out from an addition in the most significant column of the register is 1, the result recorded on the register will be in error

- Add the following binary numbers on an 8-bit register

$$
\begin{array}{rr}
 & 1\ 1\ 1\ \ \ \ 1\ 1 \\
 & (10100110)_2 \\
+ & (1100111)_2 \\
\hline
\textit{8-bit register} & \boxed{(00001101)}_2
\end{array}
$$

Overflow Error

Subtraction of Two Numbers

- In each column, if the digit in the *minuend* is equal to or larger than the digit in the *subtrahend*, the result of subtracting one from the other is recorded at the bottom of the column
- If the digit in the minuend is smaller than the digit in the subtrahend, subtract 1 from the next column and add the *base* to the digit in the minuend

$$\begin{array}{rr}
 & \begin{array}{ccc} & 7 & \\ 2 & \cancel{8} & 8 \end{array} \\
\textit{Minuend} & (\cancel{3}\cancel{0}1)_8 \\
\textit{Subtrahend} & -\ (77)_8 \\
\hline
 & (202)_8
\end{array}$$

- If the next column has a 0, then subtract 1 from the first non-zero digit afterwards, and add the *base* to the next column. Then subtract 1 from that column, and add the *base* to the next column,…