1. Introduction to Number Representation Systems

Numbers do exist as what they are - the result of counting; they do not need to be written down to come into existence. Therefore, the often encountered jargon terms: "binary numbers", "octal numbers", "decimal" and "hexadecimal" numbers do not exist as different kinds of numbers. What these jargon terms refer to is how the numbers are recorded/written down. These terms refer to the different number representations in writing: binary number representation, octal number representation, decimal number representation, and hexadecimal number representation.

Over the time, there have been invented many different ways in which numbers can be represented in various physical media. Looking just at the writing for visual reference, all of the following are different representations of the number fifty,

$$\textbf{fifty} = 50 = 110010_2 = 62_8 = 50_{10} = 32_{16} = \textbf{L}$$

or, all of the following are the representations of the number fifty-two,

$$\textbf{fifty-two} = 52 = 110100_2 = 64_8 = 52_{10} = 34_{16} = \textbf{LII},$$

or, for the year numbers in Roman representation

$$\textbf{MCMXCVIII} = 1998_{10}$$

while, neither one of them is either a binary, octal, etc. number.

To bring more order into the picture, we can remind ourselves that our every day's writing of fifty-two in the decimal representation is a shorthand notation for the numerical expression,

$$5\cdot 10^1 + 2\cdot 10^0 = 50 + 2 = 52_{10}\,. \qquad (1\text{-}1)$$

The following binary representation of the number fifty 110010_2 is a shorthand notation of the same type as (1-1)

$$110010_2 = 1\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 32 + 16 + 0 + 0 + 2 + 0 = 50\,. \qquad (1\text{-}2)$$

Our commonly known decimal number representation system belongs to the same class of ***Positional Number Representations***, which is the common form of the following often used number representations:

- decimal,
- binary,
- octal,
- hexadecimal.

POSITIONAL REPRESENTATION OF POSITIVE MIXED NUMBERS

Mixed number: $N = In\{N\} + Fr\{N\}$

Radix r representation of N: $N_r = In\{N_r\} + Fr\{N_r\}$

$$N_r = In\{N_r\} + Fr\{N_r\} = (\overbrace{a_{n-1}a_{n-2}\ldots a_2a_1a_0}^{In\{N_r\}}.\overbrace{a_{-1}a_{-2}\ldots a_{-m}}^{Fr\{N_r\}})_r \quad (1\text{-}3)$$

fractional point

Representation (1-4) is the Expanded positinal representation of the Shorthand form (1-3),

$$N_r = a_{n-1}\cdot r^{n-1} + a_{n-2}\cdot r^{n-2} + \ldots a_2\cdot r^2 + a_1\cdot r^1 + a_0\cdot r^0 + a_{-1}\cdot r^{-1} + a_{-2}\cdot r^{-2} + \ldots + a_{-m}\cdot r^{-m} \quad (1\text{-}4)$$

In the positional representation (1-3),

- N_r is the radix r representation of a mixed positive number N,
- **r** is the radix (***base***) of the representation, that is itself a positive integer number,
- $\mathbf{a}_i$ is the coefficient to the radix power r^i, with
  $a_i \in \{r-1, r-2, \ldots, 2, 1, 0\}$, and
  $\{r-1, r-2, \ldots, 2, 1, 0\}$ = ***set of available digits in radix r representation***,
- n is the number of digits needed for the representation of the integer part of N_r,
- m is the number of digits needed for the representation of the fractional part of N_r,
- when the number of digits available in hardware is greater than the number of digits needed for the integer part of N_r, leading zeros are prepended in hardware to the integer part,
- when the number of digits needed for representation of the given number N, is greater than the number of digits that are provided by the hardware medium being used, the representation of N_r is not possible in that medium.

2. Performing Arithmetic in Binary Representation

Manual addition and multiplication of numbers recorded in binary representation can be conveniently performed following thesame rules for carrying out these two operations in decimal representation.

2.1 Procedure for Manual Subtraction of Positive Integers in Binary Representation

The procedure for manual subtraction is formally the same for positional representation in all radixes. So it is practically similar to the familiar procedure used for decimal representation. The only difference of practical concern is in the step of borrowing, where in the binary representation two units are borrowed from the next higher position digit, instead of the ten units borrowed in decimal representation.

As an example we present here the detailed procedure of calculating the "two's complement" of N= ***seven***, represented in n=4 bits binary representation,

$$\widetilde{N} = 2^n - N = 2^4 - N = 10000_2 - 0111_2 = 1001_2,$$

$$\begin{array}{r} 2^4 2^3 2^2 2^1 2^0 \\ 1\ 0\ 0\ 0\ 0 \\ -\ 0\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1 \end{array}$$

A position by position evaluation of the difference $10000_2 - 0111_2$ proceeds as follows,

- at position 2^0: subtracting one from zero requires borrowing one unit of the 2^1 position of the minuend, which unit has the value of two in the position of 2^0, so one obtains $(2-1)\cdot 2^0=1$ as the LSB of the result;
- at position 2^1: the fact that one unit has already been borrowed from the 2^1 position of the minuend can be accounted for by adding 1 to 2^1 position of the subtrachend, so one needs to subtract 1+1=2 from zero; again, one borrows one unit of the next higher position 2^2 of the minuend, so one calculates $(2-2)\cdot 2^1=0$ as the value to be placed in the 2^1 position of the result;
- at position 2^2: one accounts for the one unit borrowed from 2^2 position of the minuend, by adding 1 to position 2^2 of the subtrachend, so one ought to subtract 1+1=2 from the unaltered value zero at the position 2^2 of the subtrachend; therefore, one borrows now one unit from the next higher position 2^3 of the minuend, and obtains $(2-2)\cdot 2^2=0$ as the value to be placed at the position 2^2 of the result;
- at position 2^3: one accounts for the one unit borrowed from position 2^3 of the minuend, by adding 1 to position 2^3 of the subtrachend, so one ought to subtract 0+1=1 from the unaltered value zero at the position 2^3 of the minuend, one borrows one unit of the next higher position 2^4 of the minuend (which is worth two units at the position 2^3) and one obtains $(2-1)\cdot 2^3=1$ as the value to be placed at the MSB position of the result;
- at position 2^4: one could consider that there is nothing more to be done at this position since the 1 at position 2^4 of the minuend has been already consumed by the borrowing invoved in the calculation at position 2^3; alternatively, one can account for the one unit borrowed from position 2^4 of the minuend, by adding 1 to the implicit zero at the nonexistent position 2^4 of the subtrachend, so one needs to subtract 0+1=1 from the unaltered value 1 at the position 2^4 of the minuend, whereby the result $(1-1)\cdot 2^4=0$ becomes a leading zero, that can be left out of the total result obtained by this hand calculation.

2.2 Procedure for Manual Division of Positive Integers in Binary Representation

The Algorithm for Manual Division of positive integers in binary representation is formally described by the following five steps:

STEP1: align positional values of the divisor underneath the **Active Part of Dividend** (**APD**), which **APD** satisfies the following two conditions:
- **APD** represents a number that is not smaller than the divisor,
- **APD** contains the minimal number of digits;
When such an **APD** does not exist, the quotient is equal to zero, and the remainder is equal to the dividend; otherwise procede to STEP2.

STEP2: dividing **APD** by the divisor necessarily makes the next digit of the quotient equal to 1, so digit 1 is written above the least significant digit (LSD) of the current **APD**;

STEP3: subtract the divisor from the current **APD**;

STEP4: in forming the next **APD** (to which the division by divisor will be next applied) append to the current **APD** the MSD of the yet unprocessed part of the dividend;

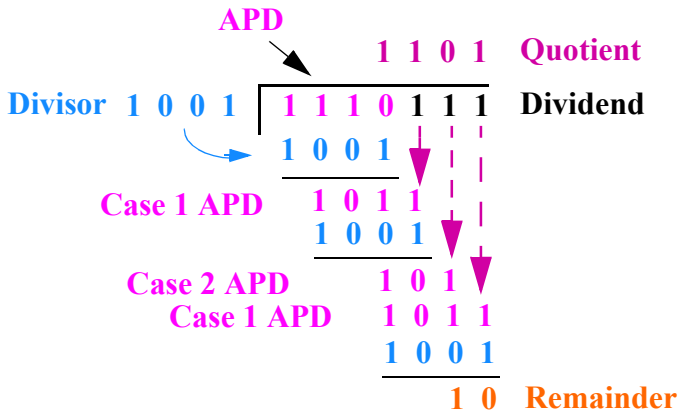
STEP5: in this step, two cases are possible:

Case 1 when **APD** represents a number that is not smaller than divisor, perform STEP2 through STEP5,

Case 2 otherwise,
- append a digit 0 to the current quotient's expression,
- when there are still unprocessed digits of the dividend:
  - apply **STEP4 through STEP5** to the current expression of **APD**;
- otherwise,
  - the **Quotient's representation** has been completed,
  - the last result of STEP3 represents the **Remainder's representation**.

Example 1.6 illustrates the application of the Manual Division Algorithm.

Example 1.6 Division in binary representation: Divide 1110111_2 by 1001_2.



1.3 Converting a Number's Representations Between two Bases/Radixes - p30

An often encountered practical situation is that a number N is available as represented in one of positional representations, but a different positional representation of N is needed by some processing step that is to be performed. This creates a conversion task that can be formally specified as,

Problem: Number Representation Conversion between Two Radixes

Problem Statement:

Given is:

N_{rA}	radix r_A representation of a positive mixed number N,
r_A, r_B	two radixes (bases),
$(r_B)_{rA}$	representation of the radix r_B in radix r_A representation,
$(r_A)_{rB}$	representation of the radix r_A in radix r_B representation.

Determine: N_{rB}, the radix r_B representation of the number N.

p31 1.3.1 Conversion Methods

To improve the clarity and simplicity of this presentation, descriptions of conversion techniques will assume that described conversions are performed:

- from a given radix r_A representation N_{rA},
- to a desired radix r_B representation N_{rB}.

Three preferable basic conversion techniques for positional number representations are listed below, together with their areas of preferable application, where the reason for preferance is that they provide **"single step" conversion methods using decimal arithmetic**:

(a) **expanding positional notation** technique is preferable for:
- conversion from representations in any radix r_A to decimal representation,

(b) **division by $(r_B)_{rA}$** (radix r_B in r_A representation) technique is preferable for:
- conversion of integer numbers from decimal to any radix r_B representation,

(c) **multiplication by $(r_B)_{rA}$** technique is preferable for.
- conversion of fractional numbers from decimal to any radix r_B representation.

From this preference list, one can conclude that when one of the radixes is ten, any representation conversion can be accomplished using only decimal arithmetic, in at most two conversion steps, that is:

- from any radix r_A to decimal representation in a single application of technique (a),
- from decimal to any radix r_B representation:
  - for integer and fractional numbers, in a single application of technique (b) or (c) respectively,
  - for mixed numbers, in one application of each of the techniques (b) and (c) to the integer and fractional part separately,

One more special conversion technique exists for conversions between radixes 2_{10}, 8_{10}, and 16_{10} that avoids using non decimal arithmetic:

(d) **grouping of digits when $r_A = (r_B)^k$** technique is applicable for:
- conversions between representations in radixes 2_{10}, 8_{10}, and 16_{10}:
  - in one step, between representations in radixes 2_{10} and 8_{10}, and 2_{10} and 16_{10},
  - in two steps, between representations in radixes 8_{10} and 16_{10}.

For all other combinations of radixes, two variants are available of the ***general case method***:

(e) **general conversion method** technique is applicable for any combination od radixes:
1. **indirect method**, performs two coversion steps, using in both only decimal arithmetic:
   - first step: from radix r_A to radix 10,
   - second step: from radix 10 to radix r_B,
2. **direct method**, uses non decimal arithmetic in performing a single conversion by one of the basic techniques: (a), (b), or (c).

p31 1.3.1_1 Conversion Method: Series Substitution

Series Substitution conversion method is also known as **Expanding Positional Notation** conversion method.

Positional Representation of Numbers has been introduced by Equation (1.3).

$$N_r = (a_{n-1}a_{n-2}\ldots a_2a_1a_o.a_{-1}a_{-2}\ldots a_{-m})_r = \quad (1\text{-}3)$$

$$= a_{n-1}\cdot r^{n-1}+a_{n-2}\cdot r^{n-2}+\ldots a_2\cdot r^2+a_1\cdot r^1+a_0\cdot r^0+a_{-1}\cdot r^{-1}+a_{-2}\cdot r^{-2}+\ldots a_{-m}\cdot r^{-m} \quad (1\text{-}4)$$

Expanding Positional Notation expands the N_{rA} representation into the sum of products of powers of the radix r_A, multiplied by the corresponding digits of the radix r_A; which is always conveniently doable in radix 10 arithmetic.

This method is practical for hand calculations, but its computer implementation is slow due to involved exponentiations.

p32 1.3.1_2 Conversion Method: Division by Radix $(r_B)_{rA}$

Conversion of integer numbers from decimal to any other radix r_B representation, applies only decimal arithmetic when Division by Radix technique is used.

Division by radix method applies a succession of divisions by the radix $(r_B)_{rA}$, as described by the algorithm DRM:

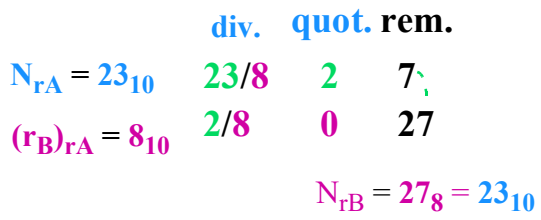
DRM-1: with counter i set to i=0, the first division evaluates the fraction $N_{rA}/(r_B)_{rA}$, after which the quotient Q_0 and remainder R_0 of this division are written on the same line to the right of the fractional expression;

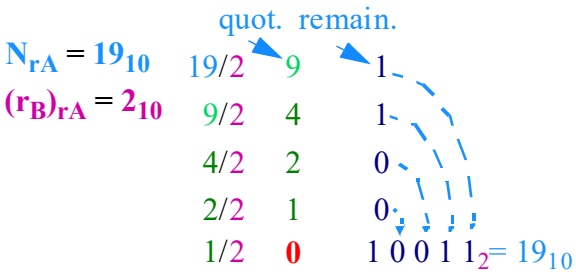
DRM-2: incrementing the counter i by one at each successive division, divisions are applied to the fractions $Q_{i-1}/(r_B)_{rA}$, and the new quotients Q_i and remainders R_i are written to the right of the fractional expressions;

DRM-3: the last division by $(r_B)_{rA}$ will be the one that results in a quotient of value zero; the remainder R_i of this last division then becomes the MSD of N_{rB}, and the other remainders are appended to R_i in the reverse chronological order so that the result of the conversion is,

$$N_{rB} = (R_i, R_{i-1},..., R_1, R_0)_{rB}$$

Two examples of the application of the division by radix method from decimal representations of 23_{10} and 19_{10} to octal and binary representation are shown next.





The "magic" supporting the division by radix $(r_B)_{rA}$ technique is more easily understood when one rewrites the explicit positional number representation of equation (1-2) as,

$$N_{rA} = (((a_{n-1}\cdot r_A + a_{n-2})\cdot r_A + ... + a_2)\cdot r_A + a_1)\cdot r_A + a_o. \qquad (1\text{-}5)$$

The form (1-5), being equivalent to radix r_A expression of the number N, shows why the remainders of the successive divisions by radix $(r_B)_{rA}$ (starting with the division of N itself, and continuing by the divisions of resulting quotients of preceding divisions) constitute the series of digits of the representation N_{rA}.

The initial division by r_A is applied to the right hand side of the equation (1-5), and yields the quotient and remainder as,

$$Q_0 = ((a_{n-1}\cdot r_A + a_{n-2})\cdot r_A + ... + a_2)\cdot r_A + a_1$$

$R_0=a_o$.

The second division by radix r_A is applied to the quotient Q_0, and yields the quotient and the remainder as,

$$Q_1 = ((a_{n-1}\cdot r_A + a_{n-2})\cdot r_A + ... + a_3)\cdot r_A + a_2$$

$$R_1 = a_1.$$

And so on, until the n-th division yields the quotient d_{n-1}, which is not divisible by r_A ($a_{n-1}<r_A$), so that the (n+1)th division yields the quotient 0 (which is the signal for ending the division process) and the remainder a_{n-1}.

Look up the textbook also!

p33 1.3.1_3Conversion Method: Multiplication by Radix $(r_B)_{rA}$

When multiplication by Radix technique is used conversion of fractional numbers from decimal representation to any other radix representation uses only decimal arithmetic.

The information needed for the execution of MRM technique includes:

- radix r_A representation of a fractional number $N_{rA} = (a_{-1}a_{-2} ...a_{-m})_{rA}$
- radix $(r_B)_{rA}$ of the representation N_{rB} which should be created,
- number of digits n_{dB} to be calculated for N_{rB}.

Multiplication by radix technique applies a succession of n_{dB} multiplication steps by the radix $(r_B)_{rA}$, as described by the algorithm MRM:

MRM-1: Execute two initialization settings:
- set counter i to $i = 0$,
- set parameter P_0 to $P_0 = N_{rA}$;

MRM-2: Execute n_{dB} times, for $i = 0$ to $i = n_{dB}-1$:
- increment i to $i = i+1$,
- calculate the product $P_{i-1}\cdot(r_B)_{rA}$, and assign its value to the new parameter P_i,
- set $b_{-i} = In\{P_i\}$,
- set $P_i = Fr\{P_i\}$;

MRM-3: after MRM-2 has been applied n_{dB} times, all n_{dB} digits of the representation N_{rB} have been calculated, and:

$$N_{rB} = (.b_{-1}b_{-2}b_{-3} ... b_{-(ndB-1)}b_{-ndB})_{rB}$$

is the result of conversion.

The following two conversion Examples illustrate the application of multiplication by radix method. The first one converts the fractional number 0.3516_{10} to octal representation, and the second one converts the fractional number 0.73505_{10} to binary representation.

Example 1

Conversion data

$N_{rA} = 0.3516_{10}$

$(r_B)_{rA} = 8_{10}$

$n_{dB} = 6$

Conversion process

$i = 1: \quad 0.3516 \cdot 8 = 2.8128 \quad b_{-1} = 2$

$i = 2: \quad 0.8128 \cdot 8 = 6.5024 \quad b_{-2} = 6$

$i = 3: \quad 0.5024 \cdot 8 = 4.0192 \quad b_{-3} = 4$

$i = 4: \quad 0.0192 \cdot 8 = 0.1536 \quad b_{-4} = 0$

$i = 5: \quad 0.1536 \cdot 8 = 1.2288 \quad b_{-5} = 1$

$i = 6: \quad 0.2288 \cdot 8 = 1.8304 \quad b_{-6} = 1$

Conversion result

$N_{rB} = 0.264011_8 = 0.35156_{10}$

Example 2

Conversion data

$N_{rA} = 0.73505_{10}$

$(r_B)_{rA} = 3_{10}$

$n_{dB} = 8$

Conversion process

$i = 1: \quad 0.73505 \cdot 3 = 2.20515 \quad b_{-1} = 2$

$i = 2: \quad 0.20515 \cdot 3 = 0.61545 \quad b_{-2} = 0$

$i = 3: \quad 0.61545 \cdot 3 = 1.84635 \quad b_{-3} = 1$

$i = 4: \quad 0.84635 \cdot 3 = 2.53905 \quad b_{-4} = 2$

$i = 5: \quad 0.53905 \cdot 3 = 1.61715 \quad b_{-5} = 1$

$i = 6: \quad 0.61715 \cdot 3 = 1.85145 \quad b_{-6} = 1$

$i = 7: \quad 0.85145 \quad = 2.55435 \quad b_{-7} = 2$

$i = 8: \quad 0.55435 \cdot 3 = 1.66305 \quad b_{-8} = 1$

Conversion result

$N_{rB} = 0.20121121_3 = 0.734948_{10}$

It is important to realize that the given two examples also show that the results of conversions are fractional numbers which slightly differ in value from the fractional number representations which were converted.

Understanding how the MRM technique works takes a different kind of consideration than the one used for DRM technique.

p35 1.3.2 General Conversion Algorithms

Algorithm 1.1 Works directly between any two bases, but may require an unfamiliar base arithmetic.

Algorithm 1.2Works in two decimal arithmetic conversions: the first to base 10, and then the second from base 10 .

p36 1.3.3 Conversion between radixes r_A and r_B when $(r_A)^k = r_B$, or $r_A = (r_B)^k$

Bit grouping (more precisely, **digit grouping**) conversion method is applicable to conversion between representations in radixes r_A and r_B, iff $r_A = (r_B)^k$ or $(r_A)^k = r_B$, where k is a positive integer.

Practical cases:

r_B	k	r_A
2	3	8
2	4	16

When $\mathbf{r_A = (r_B)^k}$ and k > 0 is an integer:

1° $\mathbf{r_A > r_B}$,

2° when the digits of r_A and r_B representations are denoted by D_A and D_B, then each D_A is representable by k D_B digits,

3° the following conversions are not possible by this method:
- conversions involving base 10 representation,
- direct conversions between base 8 and base 16 because,

$$\text{generally: } \lg(r_A) = k \cdot \lg(r_B) \qquad k = \frac{\lg(r_A)}{\lg(r_B)} \neq \text{integer}$$

$$\text{specifically: for } r_A=16 \text{ and } r_B=8, \quad \Rightarrow \quad k = \frac{\lg(r_A)}{\lg(r_B)} = \frac{\text{lb}16}{\text{lb}8} = \frac{4}{3} \neq \text{integer}$$

4° indirect conversion between base 8 and base 16 by this method involves two successive conversions:
- the first to a representation in base 2, and
- the second from the representation in base 2.

Bit grouping conversion method is performed separately on the integer and fractional parts.

When $(r_A)^k = r_B$, the digits of radix r_A representation of a number N are separated into groups of k consecutive digits to the left and to the right of the fractional point. This is then followed by expansion of each separate group into one radix r_B digit.

When $r_A = (r_B)^k$, the digits of radix r_A representation of a number N are converted individually into r_B representation, followed by a concatenation of obtained groups of k r_B digits.

Algorithm 1.3 calculates only in base 10 arithmetic, but requires a longer procedure.

p36 **Algorithm 1.3(a)** from base r_B to base r_A for $r_A = (r_B)^k$

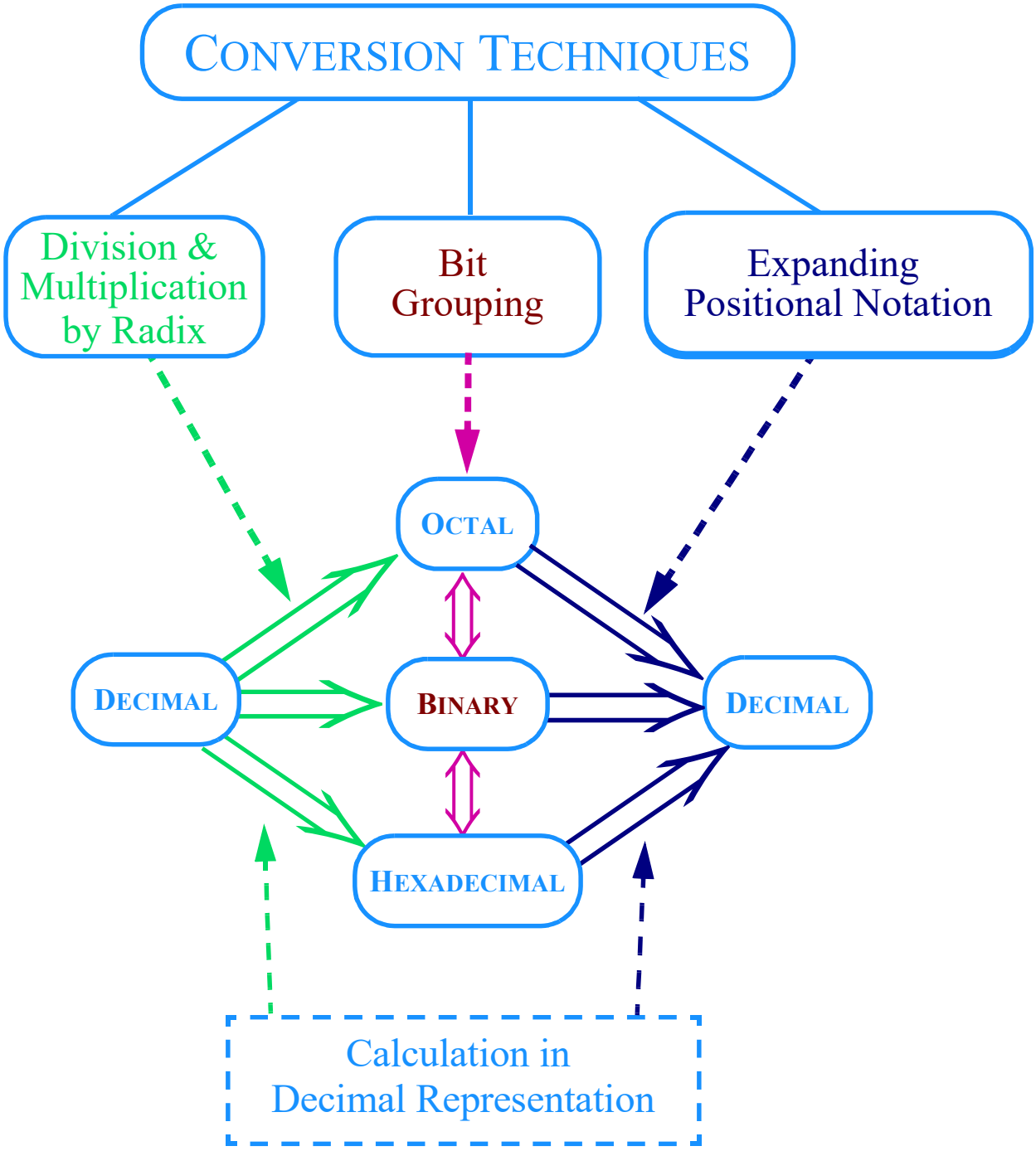
Example 1.24 Binary to Octal Representation Conversion: $(1011011.101011100)_2$

p36 **Algorithm 1.3(b)** from base r_A to base r_B for $r_A = (r_B)^k$

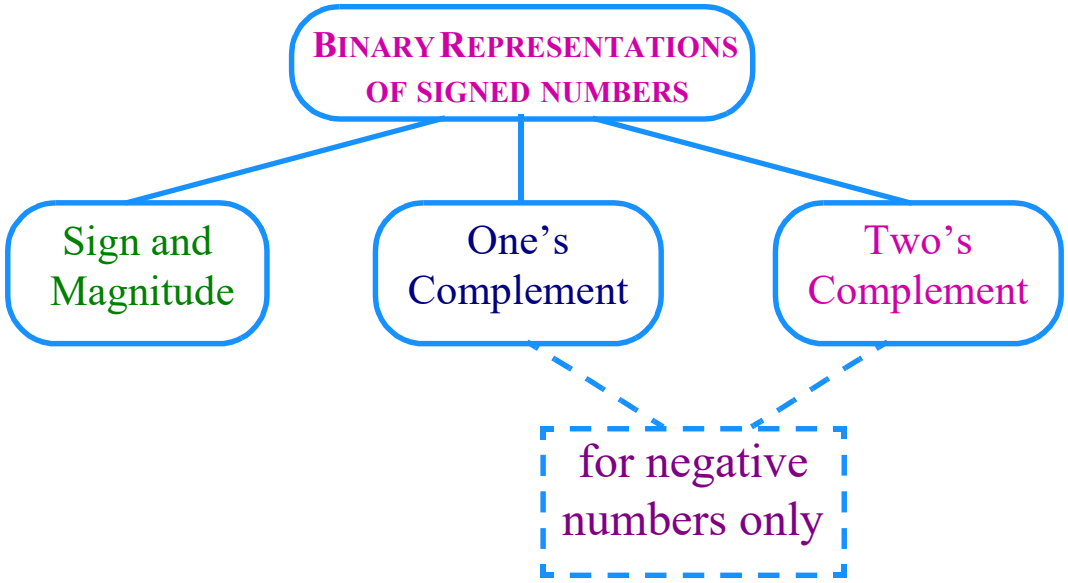
Example 1.25 Hexadecimal to Octal Representation Conversion: $(AF.16C)_{10}$

Conversions are performed separately on the integer part and the fractional parts, in the order: to the left, and to the right from the fractional point.

Conversion Between Practically Significant Positional Number Representations



p37 1.4 Representation of Signed Numbers



p38 1.4.1 Sign and Magnitude Representation

p39 Table 1.6 Signed Number Representation Examples

p38 1.4.2 Complementary Number Systems

p38 1.4.2_1 Radix Complements

Radix complements are used in creating representations of negative numbers.

p42 Algorithm 1.4

p42 Algorithm 1.5

For radix r, the ***diminished radix*** is defined as,

$$\textit{diminished radix of } r = r-1$$

which in particular yields,

radix	r	2	8	10	16
diminished radix	r-1	1	7	9	F

Two types of complements of positional number representations find application in digital computer design: the **radix complement** and the **diminished radix complement**.

Consider a positional representation of an integer N in **n** digits of radix *r*,,

$$N_r = (a_{n-1}a_{n-2}...a_2a_1a_o)_r = a_{n-1}\cdot r^{n-1} + a_{n-2}\cdot r^{n-2} + ... + a_2\cdot r^2 + a_1\cdot r^1 + a_o\cdot r^o \quad (1\text{-}3)$$

The **radix complement** of N_r is defined as,

$$\textit{radix complement of the number } \mathbf{N_r = r^n - N_r = \widetilde{N}} \quad (2\text{-}1)$$

The diminished radix complement of representation (1-3) of the number N_r is defined as,

$$\textit{diminished radix complement of the number } N_r = r^n\text{-}1 - N_r = \mathbf{r^n - N_r - 1 = \overline{N}} \quad (2\text{-}2)$$

It follows from the definitions (2-1) and (2-2) that the radix complement and the diminished radix complement of the number N_r representation are related by equation (2-3),

$$\widetilde{\mathbf{N}} = \overline{\mathbf{N}} + \mathbf{1} \quad (2\text{-}3)$$

One might be tempted to write the relation as $\overline{N} = \widetilde{N} - 1$, but the form (2-3) will be used later on, when we discover that for r=2 there is a simpler way of calculating the radix and diminished radix complements, rather than evaluating the defining expressions (2-1) and (2-2). Which is significant because the binary representation is in practically used in all of today's computers, using a number of bits **n** which is an even integer multiple of eight bits; e.g. $n \in \{16, 32, 64, 128\}$.

The road to this discovery starts by visualizing the composition of the two numbers, $\mathbf{r^n}$ and $\mathbf{r^n\text{-}1}$ (for a given n they are constants) which are used in the definitions (2-1) and (2-2),

	The constant used in the definition of **the *radix complement***	The constant used in the definition of the ***diminished radix complement***
Generally	$r^n = \overbrace{100\ldots000}^{\text{n-zeros}}_r$	$r^n\text{-}1 = \overbrace{(r\text{-}1)(r\text{-}1)\ldots(r\text{-}1)(r\text{-}1)(r\text{-}1)}^{\text{n-digits}}_r$
In particular	$10^n = \overbrace{100\ldots000}^{\text{n-zeros}}_{10}$	$10^n\text{-}1 = \overbrace{99\ldots999}^{\text{n-nines}}_{10}$
	$2^n = \overbrace{100\ldots000}^{\text{n-zeros}}_2$	$2^n\text{-}1 = \overbrace{11\ldots111}^{\text{n-ones}}_2$

The following example will illustrate the point that the values of the radix and diminished radix complements of a number N are functions of both, the number N itself and the number of digits *n* in N's representation for which the complement is calculated. Let us take the number seven, and its radix r=2 representations in n_1=4 and n_2=8 bits; i.e. $N=7=0111_2=00000111_2$, and let us calculate the radix complements of those two representations of the number seven,

$$\widetilde{N}_1 = 2^{n1} - N = 2^4 - N = \overbrace{10000}^{4}_2 - 0111_2 = 1001_2$$

$$\widetilde{N}_2 = 2^{n2} - N = 2^8 - N = \overbrace{100000000}^{8}_2 - 00000111_2 = 11111001_2.$$

$$\overline{N}_1 = 2^{n1} - 1 - N = 2^4 - 1 - N = 10000_2 - 1 - N = 1111 - 0111_2 = 1000_2$$

$$\overline{N}_2 = 2^{n2} - 1 - N = 2^8 - 1 - N = \overbrace{100000000}^{8}_2 - 1 - N = \overbrace{11111111}^{8}_2 - 00000111_2 = 11111000_2$$

These results can be used to make two observations which bear practical consequence on the design of digital hardware,

1. comparing bitwise two binary representations of the numbers N_1 and N_2 with their corresponding diminished radix complements $\overline{N}_1$ and $\overline{N}_2$,

   $\mathbf{N_1 = 0111_2}$ $\quad$ $\mathbf{N_2 = 00000111_2}$

   $\mathbf{\overline{N}_1 = 1000_2}$ $\quad$ $\mathbf{\overline{N}_2 = 11111000_2}$

   shows that one's complement of N is equal to the bitwise complement of N, which then points to an easier way of calculating one's complement than using the diminished radix complement definition (2-2);

2. equation (2-3) $\widetilde{N} = \overline{N} + 1$ can be verified

   $\widetilde{N}_1 = \overline{N}_1 + 1 = 1000_2 + 0001 = 1001_2$

   which shows the way in which the computation of two's complement of binary number representation is done in digital hardware.

Two's complement representation of negative numbers results in the simplest hardware for arithmetic addition and subtraction. One's complement representation may give erroneous results when the -0 representation is involved in addition or subtraction, so its application requires additional hardware to detect the condition and correct the error. Sign and magnitude representation requires even more complex hardware.

In two's complment representation of negative numbers, the process of determining two's complement of a number N is equivalent to the process of negation the number N. As a consequence, 2's complement of a negative number -N is equal to N, because

$$-(-N) = N$$

For example, take N=7 in 8-bit binary representation and determine the 2's complement of it,

$\tilde{N} = \overline{N} + 1$

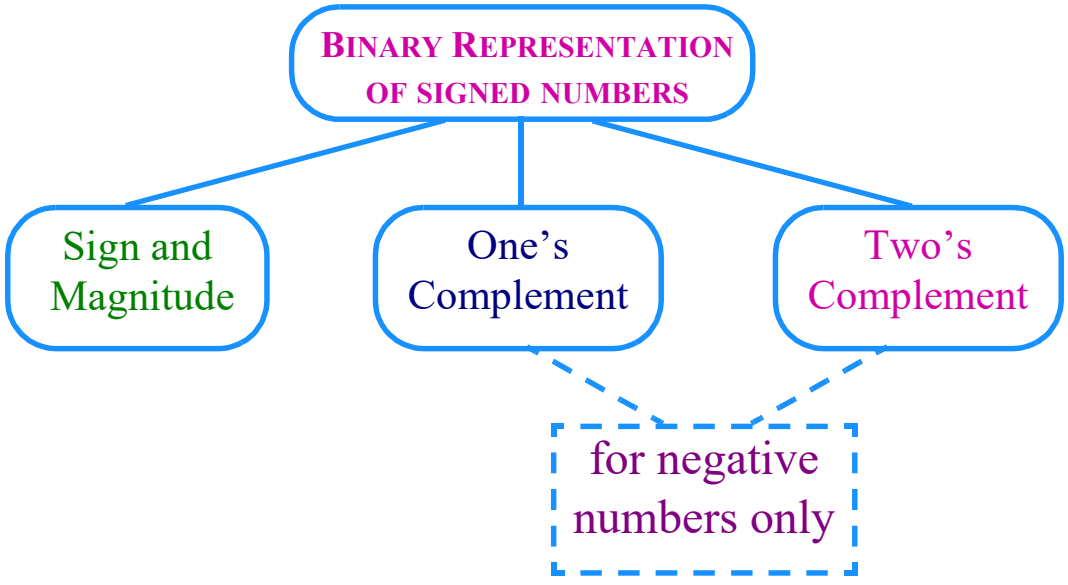
For $N = 7_{10}$,

N :	00000111_2	
bitwise complement of N:	11111000_2	$\overline{N}$
plus one:	$+\ 00000001_2$	1
2's complement of N representation of -N = -7:	11111001_2	$\tilde{N}$
bitwise complement of -N :	00000110_2	
plus one:	$+\ 00000001_2$	1
N :	00000111_2	

The only error which may occur with addition and subtraction operations performed on signed numbers in 2's complement representation of negative numbers is the **overflow condition**; which is not caused by a fault in 2's complement representation, but is caused by the **insufficient** (finite) **size of the hardware** resources.

p38 1.4.2_2 Radix Complement Number Systems

Physical representation of signed numbers in computers and other digital hardware takes place in the same medium which provides only two states of the medium for representation of digits. There are no other means for representing the plus and minus sign but the same two physical states already used for the binary representation of positive numbers. A fiew methods have been devised and used for representation of signed numbers in computing hardware.



Signed numbers representable in n=4 bits

Decimal equivalent	Sign and Magnitude	1's and 2's Complement	Decimal equivalent when the representation is considered to be	
			2's complement	1's complement
+7	0111	0111	+7	+7
+6	0110	0110	+6	+6
+5	0101	0101	+5	+5
+4	0100	0100	+4	+4
+3	0011	0011	+3	+3
+2	0010	0010	+2	+2
+1	0001	0001	+1	+1
(+)0	0000	0000	0	(+)0
(-)0	1000	1111	-1	(-)0
-1	1001	1110	-2	-1
-2	1010	1101	-3	-2
-3	1011	1100	-4	-3
-4	1100	1011	-5	-4
-5	1101	1010	-6	-5
-6	1110	1001	-7	-6
-7	1111	1000	-8	-7

Signed numbers representable in n=5 bits				
Decimal equivalent	Sign and Magnitude	1's and 2's Complement	Decimal equivalent when the representation is considered to be	
			2's complement	1's complement
+15	01111	01111	+15	+15
+14	01110	01110	+14	+14
+13	01101	01101	+13	+13
+12	01100	01100	+12	+12
+11	01011	01011	+11	+11
+10	01010	01010	+10	+10
+9	01001	01001	+9	+9
+8	01000	01000	+8	+8
+7	00111	00111	+7	+7
+6	00110	00110	+6	+6

+5	00101	00101	+5	+5
+4	00100	00100	+4	+4
+3	00011	00011	+3	+3
+2	00010	00010	+2	+2
+1	00001	00001	+1	+1
(+)0	00000	00000	0	(+)0
(-)0	10000	11111	-1	(-)0
-1	10001	11110	-2	-1
-2	10010	11101	-3	-2
-3	10011	11100	-4	-3
-4	10100	11011	-5	-4
-5	10101	11010	-6	-5
-6	10110	11001	-7	-6
-7	10111	11000	-8	-7
-8	11000	10111	-9	-8
-9	11001	10110	-10	-9
-10	11010	10101	-11	-10
-11	11011	10100	-12	-11
-12	11100	10011	-13	-12
-13	11101	10010	-11	-13
-14	11110	10001	-15	-14
-15	11111	10000	-16	-15

5.2 The Range of signed numbers representable in n bits

The number of different zero/one combinations that can be written into n bits is equal to 2^n.

The number of different zero/one combinations in n bits = **2^n**.

n	1	2	3	4	**values of combinations in binary representation**
2^n	2	4	8	16	
combinations	0 1	00 01 10 11	000 001 010 011 100 101 110 111	0000 0001 0010 0011 0100 0101 0110 0111 1000 1 . . . 1111	0 1 2 3 4 5 6 7 8 . 15

The range of non-negative integers which can be represented in n bits is consequently,

$$\mathbf{0 \le N \le 2^n-1}$$

The number of bits provided by the computer hardware for representing numbers is fixed nowdays at $n \in \{16, 32, 64\}$. When one of those bits is used for representing the sign, n-1 bits are left for representing the numbers' magnitudes, which effectively halves the magnitude with respect to the case of nonnegative integers because $2^{n-1}=2^n/2$.

Consequently, the range of signed numbers N representable in three significant representation schemes of negative numbers is,

Representation of negative numbers	The range of signed numbers representable
Sign and Magnitude	$\mathbf{-2^{n-1}+1 \le N \le 2^{n-1}-1}$
1's complement	
2's complement	$\mathbf{-2^{n-1} \le N \le 2^{n-1}-1}$

A full understanding of the meaning of this result can be gained only by considering some examples. Because of its reasonable size, the first good example are the tables of all numbers representable in n=4 bits and in n=5 bits. An interesting practical consideration at this time is the representation using n=32, which is the case in majority of today's personal computers. Disregarding temporarily -1, we find that the largest positive number representable in n=32 bits is by one less than,

$$\mathbf{2^{n-1} = 2^{32-1} = 2^{31} = 2 \cdot 2^{30} = 2 \cdot (2^{10})^3 = 2 \cdot 1024^3 = 2 \cdot 1{,}073{,}741{,}824 = 2{,}147{,}483{,}648.}$$

We can ask ourselves a practical question now: "Is that a big enough number for ordinary people who are not doing scientific research?" An interesting point can be made by comparing it to the accumulated Federal budget deficit. Could a government's office use such a computer to store and manipulate the $ number of the deficit?

p45 1.4.2_3Radix complement arithmetic

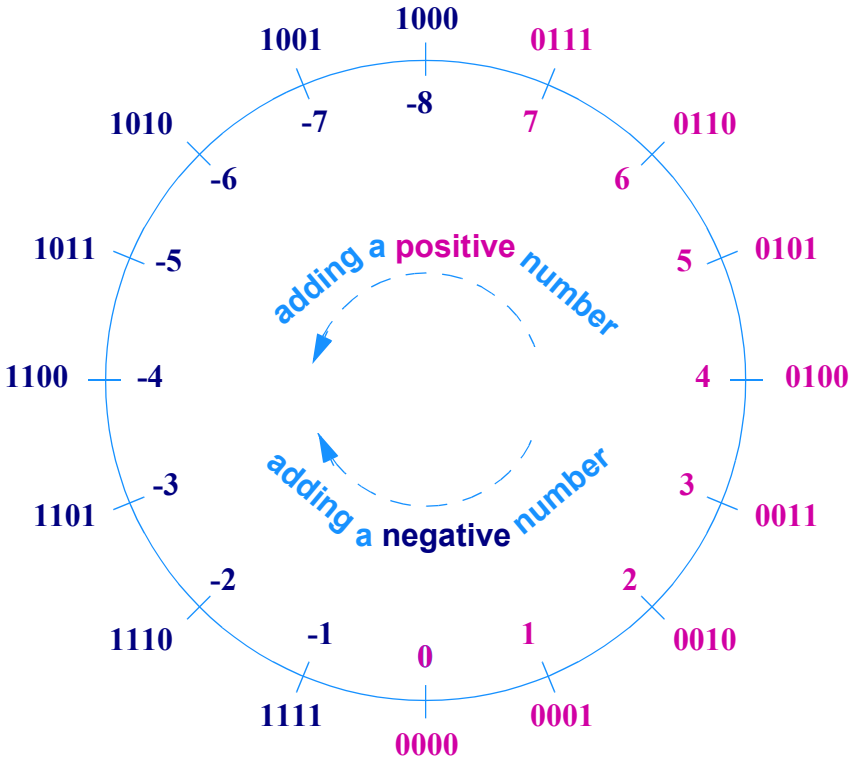
p46 Case 1: A = B + C

p47 Case 2: A = B - C

p49 Case 3: A = -B - C = -(B+C) = $[B+C]_2$

p45The Overflow Problem

The overflow problem may be expected to occur when addition/subtraction operation is applied to signed numbers represented in a fixed number of bits. An easy way to visualize why the overflow happens is by arranging the set of signed integers in a closed circular sequence. For integers representable in n=4 bits, the result with two's complement representation of negative numbers is,



Examples of computations which result into overflow conditions:

$6 + 3 = 0110_2 + 0011_2 = 1001_2 = -7$

$-6 - 3 = -6 + (-3) = 1010_2 + 1101_2 = 10111_2 = 0111_2 = +7$

$-6 - 4 = -6 + (-4) = 1010_2 + 1100_2 = 10110_2 = 0110_2 = +6$

hand hardware

6.Unusual Number Representation Systems

6.1 Double-Based Number Systems (DBNS)

Binary number system is a special case of the Double-Based Number Systems

$$x = \sum_{j,k} d_{jk} \cdot b_1^j \cdot b_2^k \cdot , \; d_{jk} \in \{0,1\}$$

6.2 Residue Number System (RNS)

Given an integer $x \in I$, and a set of integer constants

$$S_M = \{m_1, m_2, \ldots, m_k\}$$

a set of residues (remainders)

$$X_R = \{r_1, r_2, \ldots, r_k\}$$

can be computed, where

$$r_i = x \text{ modulo } m_i, \quad \Rightarrow x = r_i + q_i \cdot m_i \, , \; q_i \in I\text{Þ}, \; 0 \le r_i < m_i$$

The set of residues X_R is unique, i.e. $X_R = X_{RNS}$ when the following two conditions are satisfied,

1. $x \in \{0, 1, 2, \ldots, M\text{-}1\}$, $M = \prod_{i=1}^{k} m_i$,
2. for i,j = 1, 2, …, k, and $i \neq j$, $GCD(m_i, m_j) = 1$, $\Rightarrow$ that elements of S_M are *relatively prime*. Where GCD= *greatest common divisor.*

Which means that under the above two conditions for all numbers $x < M$ a different set $\{r_1, r_2, \ldots, r_k\}$ is obtained.

The advantage of the residue number system is that arithmetic operations addition, subtraction and multiplication can be performed in parallel because no carry is generated.

6.3 The Chinese Remainder Theorem (CRT)

Using the designation,

$$\langle x \rangle_m = x \text{ modulo } m$$

the number x can be recovered from its RNS representation

$$X_{RNS} = \{r_1, r_2, \ldots, r_k\}$$

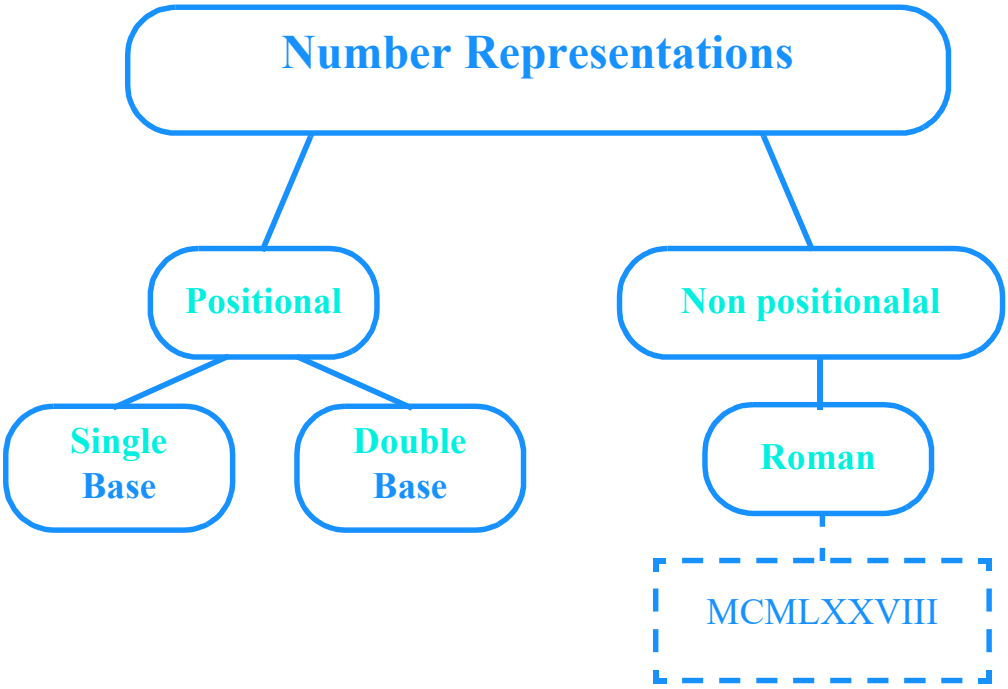
by the application of the CRT

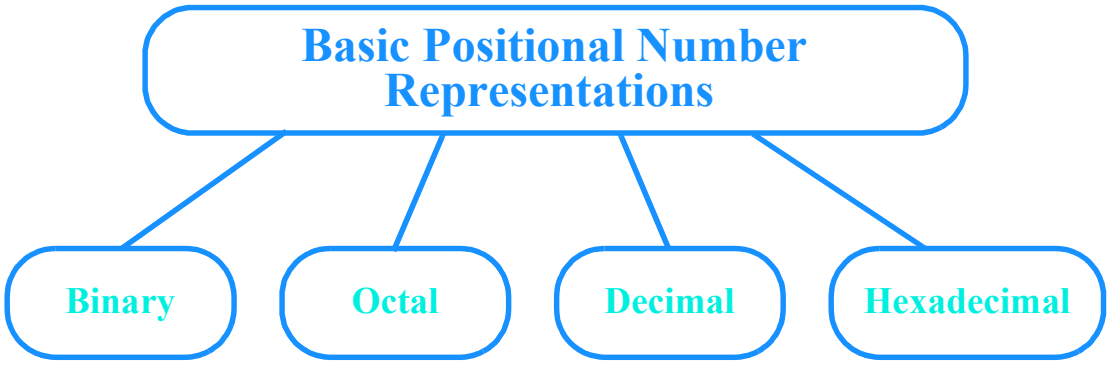
$$x = \left\langle \sum_{i=1}^{k} \langle r_i \cdot N_i \rangle_{m_i} \cdot M_i \right\rangle_M$$

where $M_i = \frac{M}{m_i}$ and $N_i = < M_i^{-1} >_{m_i}$

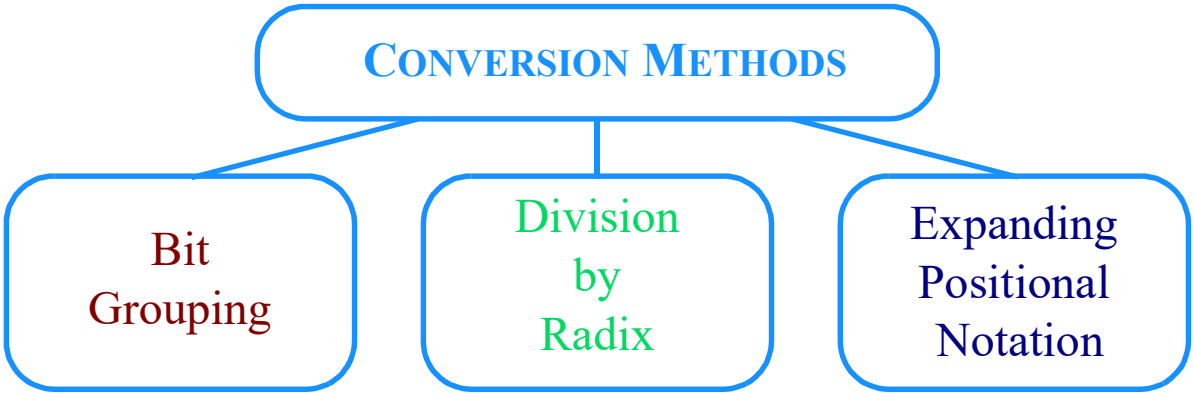
meaning that N_i is the multiplicative inverse of M_i modulo m_i.

NUMBER REPRESENTATIONS





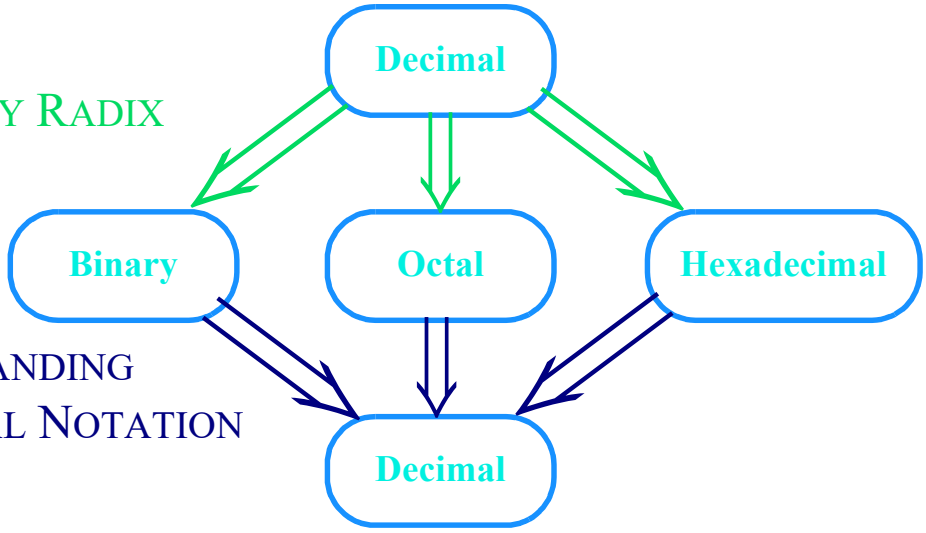
Conversion between Different Number Representations



Bit Grouping: digits ⇔ bit groups ⇔ digits

Division by Radix

Expanding Positional Notation



Conversion Between Practically Significant Positional Number Representations