ISTANBUL UNIVERSITY
Dept. of Computer Engineering

Computer Arithmetic Algorithms

Part 1
Introduction

Prerequisites and textbook

- Prerequisites: courses in
  - Digital Design
  - Computer Organization/Architecture
- Textbook: *Computer Arithmetic Algorithms*, I. Koren, 2nd Edition, A.K. Peters, Natick, MA, 2002
- Textbook web page: http://www.ecs.umass.edu/ece/koren/arith
- Recommended Reading:
  - B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design*, Oxford University Press, 2000
  - M. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufman, 2003

Administrative Details

- **Instructor**:Prof. Ahmet Sertbaş
- **Office**: Tel. 473 70 70 - 17902
- **Email**: asertbas@istanbul.edu.tr

- **Grading**: Undergraduated

• Homework -	20% (yıliçi)
• Project-	30%
• Final	50 %

What is Computer Arithmetic?

Pentium Division Bug (1994-95): Pentium’s radix-4 SRT algorithm occasionally gave incorrect quotient
First noted in 1994 by T. Nicely who computed sums of reciprocals of twin primes:

$$1/5 + 1/7 + 1/11 + 1/13 + \ldots + 1/p + 1/(p + 2) + \ldots$$

Worst-case example of division error in Pentium:

$$c = \frac{4\,195\,835}{3\,145\,727} = \begin{cases} 1.333\,820\,44\ldots & \text{Correct quotient} \\ 1.333\,739\,06\ldots & \text{circa 1994 Pentium double FLP value; accurate to only 14 bits (worse than single!)} \end{cases}$$

Top Ten Intel Slogans for the Pentium

Humor, circa 1995

- 9.999 997 325 It’s a FLAW, dammit, not a bug
- 8.999 916 336 It’s close enough, we say so
- 7.999 941 461 Nearly 300 correct opcodes
- 6.999 983 153 You don’t need to know what’s inside
- 5.999 983 513 Redefining the PC -- and math as well
- 4.999 999 902 We fixed it, really
- 3.999 824 591 Division considered harmful
- 2.999 152 361 Why do you think it’s called “floating” point?
- 1.999 910 351 We’re looking for a few good flaws
- 0.999 999 999 The errata inside

Finite Precision Can Lead to Disaster

Example: Failure of Patriot Missile (1991 Feb. 25)

Source http://www.math.psu.edu/dna/455.f96/disasters.html

American Patriot Missile battery in Dharan, Saudi Arabia, failed to intercept inoming Iraqi Scud missile

The Scud struck an American Army barracks, killing 28

Cause, per GAO/IMTEC-92-26 report: “software problem” (inaccurate calculation of the time since boot)

Problem specifics:

Time in tenths of second as measured by the system’s internal clock was multiplied by 1/10 to get the time in seconds

Internal registers were 24 bits wide

1/10 = 0.0001 1001 1001 1001 1001 100 (chopped to 24 b)

Error ≈ 0.1100 1100 × 2^{-23} ≈ 9.5 × 10^{-8}

Error in 100-hr operation period

≈ 9.5 × 10 –8 × 100 × 60 × 60 × 10 = 0.34 s

Distance traveled by Scud = (0.34 s) × (1676 m/s) ≈ 570 m

Finite Range Can Lead to Disaster

Example: Explosion of Ariane Rocket (1996 June 4)

Source http://www.math.psu.edu/dna/455.f96/disasters.html

Unmanned Ariane 5 rocket of the European Space Agency veered off its flight path, broke up, and exploded only 30 s after lift-off (altitude of 3700 m)

The $500 million rocket (with cargo) was on its first voyage after a decade of development costing $7 billion

Cause: "software error in the inertial reference system"

Problem specifics:

A 64 bit floating point number relating to the horizontal velocity of the rocket was being converted to a 16 bit signed integer

An SRI* software exception arose during conversion because the 64-bit floating point number had a value greater than what could be represented by a 16-bit signed integer (max 32 767)

*SRI = Système de Référence Inertielle or Inertial Reference System

Aspects of, and Topics in, Computer Arithmetic

Hardware (our focus in this book)

Design of efficient digital circuits for primitive and other arithmetic operations such as +, –, ×, ÷, √, log, sin, cos

Issues: Algorithms
Error analysis
Speed/cost trade-offs
Hardware implementation
Testing, verification

Software

Numerical methods for solving systems of linear equations, partial differential equations, etc.

Issues: Algorithms
Error analysis
Computational complexity
Programming
Testing, verification

General-purpose

Flexible data paths
Fast primitive operations like +, –, ×, ÷, √
Benchmarking

Special-purpose

Tailored to applications like:
Digital filtering
Image processing
Radar tracking

Fig. 1.1 The scope of computer arithmetic.

Course Outline

- Number systems and basic arithmetic operations
- Unconventional fixed-point number systems
- Sequential algorithms for multiplication and division
- Floating-point arithmetic
- Algorithms for fast addition
- High-speed multiplication
- Fast division
- Division through multiplication
- Efficient algorithms for evaluation of elementary functions.
- Logarithmic number systems.
- The residue number system; error correction and detection in arithmetic operations, .

The Binary Number System

- In conventional digital computers - integers represented as binary numbers of fixed length n
- An ordered sequence $(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$ of binary digits
- Each digit x_i (bit) is 0 or 1
- The above sequence represents the integer value X

$$X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2 + x_0 = \sum_{i=0}^{n-1} x_i 2^i$$

- Upper case letters represent numerical values or sequences of digits
- Lower case letters, usually indexed, represent individual digits

Radix of a Number System

- The weight of the digit x_i is the i th power of 2
- 2 is the radix of the binary number system
- Binary numbers are radix-2 numbers - allowed digits are 0,1
- Decimal numbers are radix-10 numbers - allowed digits are 0,1,2,…,9
- Radix indicated in subscript as a decimal number
- Example:
  - $(101)_{10}$ - decimal value 101
  - $(101)_2$ - decimal value 5

Range of Representations

- Operands and results are stored in registers of fixed length n - finite number of distinct values that can be represented within an arithmetic unit
- Xmin ; Xmax - smallest and largest representable values
- [Xmin,Xmax] - range of the representable numbers
- A result larger then Xmax or smaller than Xmin - incorrectly represented
- The arithmetic unit should indicate that the generated result is in error -
  an overflow indication

Example - Overflow in Binary System

- Unsigned integers with 5 binary digits (bits)
  - $X_{max} = (31)_{10}$ - represented by $(11111)_2$
  - $X_{min} = (0)_{10}$ - represented by $(00000)_2$
  - Increasing X_{max} by 1 = $(32)_{10} =(100000)_2$
  - 5-bit representation - only the last five digits retained - yielding $(00000)_2 =(0)_{10}$
- In general -
  - A number X not in the range $[X_{min},X_{max}]=[0,31]$ is represented by $X \bmod 32$
  - If $X+Y$ exceeds X_{max} - the result is $S = (X+Y) \bmod 32$
- Example:

$$\begin{array}{rrrl} X & 10001 & 17 & \\ +Y & 10010 & 18 & \\ \hline & 1\ 00011 & 3 & = 35 \bmod 32 \end{array}$$

  - Result has to be stored in a 5-bit register - the most significant bit (with weight $2^5 = 32$) is discarded

Machine Representations of Numbers

- Binary system - one example of a number system that can be used to represent numerical values in an arithmetic unit
- A number system is defined by the set of values that each digit can assume and by an interpretation rule that defines the mapping between the sequences of digits and their numerical values
- Types of number systems -
- conventional (e.g.,binary, decimal)
- unconventional (e.g., signed-digit number system)

Conventional Number Systems

- Properties of conventional number systems:
- Nonredundant -
  - Every number has a unique representation, thus
  - No two sequences have the same numerical value
- Weighted -
  - A sequence of weights $w_{n-1},w_{n-2},...,w_1,w_0$ determines the value of the n-tuple $x_{n-1},x_{n-2},...,x_1,x_0$ by

$$X = \sum_{i=0}^{n-1} x_i w_i .$$

  - w_i - weight assigned to x_i - digit in ith position
- Positional -
  - The weight w_i depends only on the position i of digit x_i
  - $w_i = r^i$

Fixed Radix Systems

- r - the radix of the number system
- Conventional number systems are also called fixed-radix systems
- With no redundancy - $0 \le x_i \le r-1$
- $x_i \ge r$ introduces redundancy into the fixed-radix number system
- If $x_i \ge r$ is allowed -

$$x_i r^i = (x_i - r) r^i + 1 \cdot r^{i+1}$$

- two machine representations for the same value - $(\ldots, x_{i+1}, x_i, \ldots)$ and $(\ldots, x_{i+1}+1, x_i-r, \ldots)$

Representation of Mixed Numbers

- A sequence of n digits in a register - not necessarily representing an integer
- Can represent a mixed number with a fractional part and an integral part
- The n digits are partitioned into two - k in the integral part and m in the fractional part (k+m=n)
- The value of an n-tuple with a radix point between the k most significant digits and the m least significant digits

$$(\underbrace{x_{k-1}x_{k-2}\cdots x_1x_0}_{integral\ part} \; . \; \underbrace{x_{-1}x_{-2}\cdots x_{-m}}_{fractional\ part})_r$$

is

$$X = x_{k-1}r^{k-1} + x_{k-2}r^{k-2} + \cdots + x_1 r + x_0 + x_{-1}r^{-1} + \cdots + x_{-m}r^{-m} = \sum_{i=-m}^{k-1} x_i r^i$$

Fixed Point Representations

- Radix point not stored in register - understood to be in a fixed position between the k most significant digits and the m least significant digits
  - These are called fixed-point representations
- Programmer not restricted to the predetermined position of the radix point
  - Operands can be scaled - same scaling for all operands
- Add and subtract operations are correct -
  - $aX \pm aY = a(X \pm Y)$ (a - scaling factor)
- Corrections required for multiplication and division
  - $aX \bullet aY = a^2 X \bullet Y$; $aX/aY = X/Y$
- Commonly used positions for the radix point -
  - rightmost side of the number (pure integers - $m=0$)
  - leftmost side of the number (pure fractions - $k=0$)

ULP - Unit in Last Position

- Given the length n of the operands, the weight r^{-m} of the least significant digit indicates the position of the radix point
- Unit in the last position (ulp) - the weight of the least significant digit
- $ulp = r^{-m}$
- This notation simplifies the discussion
- No need to distinguish between the different partitions of numbers into fractional and integral parts

Radix Conversions

- Translating a number X represented in one radix number system (source number system) to its representation in another number system (destination)
- Main reason - most arithmetic units operate on binary numbers, while users are accustomed to decimal numbers (requiring fewer digits)
- Given a number X, find its representation in the destination number system with radix r_D
- We distinguish between the conversion of the integral part X_I and the fractional part X_F

Conversion of Integral Part

- Seeking $(x_{k-1}x_{k-2}\cdots x_1x_0)_{r_D}$

$$X_I = \{[\cdots(x_{k-1}r_D + x_{k-2})r_D + \cdots + x_2]\, r_D + x_1\}\, r_D + x_0;$$

- Dividing X_I by r_D
  - * remainder - x_0
  - * quotient - $\{[\cdots(x_{k-1}r_D + x_{k-2})r_D + \cdots + x_2]\, r_D + x_1\}$
- Dividing the quotient by r_D → x_1 is the remainder
- Dividing the quotients repeatedly by r_D until a zero quotient is obtained - the remainders are the required digits

Conversion of Fractional Part

- Seeking $(x_{-1}x_{-2}\cdots x_{-m})_{r_D}$

$$X_F = r_D^{-1}\left\{x_{-1} + r_D^{-1}\left[x_{-2} + r_D^{-1}(x_{-3} + \cdots)\right]\right\}$$

- Multiplying X_F by r_D we obtain a mixed number
  - x_{-1} is the integral part
  - $r_D^{-1}\left[x_{-2} + r_D^{-1}(x_{-3} + \cdots)\right]$ is the fractional part
- Fractional parts multiplied repeatedly by r_D, generated integers are the required digits
- Algorithm not guaranteed to terminate
  - Finite fraction in one number system may correspond to an infinite fraction in another
  - In practice - the process can be terminated after m steps (or a few additional ones for rounding)

Example - Decimal to Binary Conversion

- Converting the decimal mixed number X=46.375 form
- XI=46 - quotients and remainders wh dividing by 2:

Quotient	Remainder
23	$0 = x_0$
11	$1 = x_1$
5	$1 = x_2$
2	$1 = x_3$
1	$0 = x_4$
0	$1 = x_5$

- XF=0.375 - integers and fractions multiplying by 2:

Integer part	Fractional part
$0 = x_{-1}$	.75
$1 = x_{-2}$	.5
$1 = x_{-3}$	.0

- Final result - 46.37510=101110.0112
- If (decimal) fractional part is XF=0.3 - the algorithm never terminates - results in an infinite binary fraction 0.0100110011..2
- All arithmetic operations were performed in source system - decimal
- For binary to decimal conversion
  - either perform the algorithm in the source binary system
  - or, more conveniently, use X = $\sum_{i=-m}^{k-1} x_i r^i$
- perform the conversion in the destination decimal system using equation (\ref{eq:3}).

Representation of Negative Numbers

- Fixed-point numbers in a radix r system
- Two ways of representing negative numbers:
- Sign and magnitude representation (or signed-magnitude representation)
- Complement representation with two alternatives
  - Radix complement (two's complement in the binary system)
  - Diminished-radix complement (one's complement in the binary system)

Signed-Magnitude Representation

- Sign and magnitude are represented separately
- First digit is the sign digit, remaining $n-1$ digits represent the magnitude
- Binary case - sign bit is 0 for positive, 1 for negative numbers
- Non-binary case - 0 and $r-1$ indicate positive and negative numbers
- Only $2r^{n-1}$ out of the r^{n} possible sequences are utilized

Range of Representable Numbers

- n-1 digits representing magnitude - partitioned into k-1 and m digits in integral and fractional parts
- Largest representable value is $X_{max} = (r^{k-1} - ulp)$ where $ulp = r^{-m}$ with representation 0 (r-1) … (r-1)
- Range of positive numbers is $[0, r^{k-1} - ulp]$
- Range of negative numbers is $[-(r^{k-1} - ulp), -0]$ represented by (r-1) (r-1) … (r-1) to (r-1) 0 … 0
- Two representations for zero - positive and negative
- Inconvenient when implementing an arithmetic unit - when testing for zero, the two different representations must be checked

Range of Binary System

- All 2^n sequences are utilized
- 2^{n-1} sequences from $00 \cdots 0$ to $01 \cdots 1$ represent positive numbers
- Remaining 2^{n-1} sequences from $10 \cdots 0$ to $11 \cdots 1$ represent negative numbers
- If $k=n$ ($m=0$ and $ulp=2^0=1$)
  - range of positive numbers is $[0,2^{n-1} - 1]$
  - range of negative numbers is $[-(2^{n-1} - 1), -0]$

Disadvantage of the Signed-Magnitude Representation

- Operation may depend on the signs of the operands
- Example - adding a positive number X and a negative number -Y :   X+(-Y)
- If Y>X, final result is -(Y-X)
- Calculation -
  - switch order of operands
  - perform subtraction rather than addition
  - attach the minus sign
- A sequence of decisions must be made, costing excess control logic and execution time
- This is avoided in the complement representation methods

Complement Representations of Negative Numbers

- Two alternatives -
  - * Radix complement (called two's complement in the binary system)
  - * Diminished-radix complement (called one's complement in the binary system)
- In both complement methods - positive numbers represented as in the signed-magnitude method
- A negative number -Y is represented by R-Y where R is a constant
- This representation satisfies -(-Y)=Y since R-(R-Y)=Y

Advantage of Complement Representation

- No decisions made before executing addition or subtraction
- Example: X-Y=X+(-Y)
- -Y is represented by R-Y
- Addition is performed by X+(R-Y) = R-(Y-X)
- If Y>X, -(Y-X) is already represented as R-(Y-X)
- No need to interchange the order of the two operands

Requirements for Selecting R

- If $X>Y$ - the result is $X+(R-Y)=R+(X-Y)$ instead of $X-Y$ - additional R must be discarded
- R selected to simplify or eliminate this correction
- Another requirement - calculation of the complement $R-Y$ should be simple and done at high speed
- **Definitions:**
- Complement of a single digit x_i

  $\bar{x}_i = (r-1) - x_i$
- Complement of an n-tuple X

  $\bar{X} = (\bar{x}_{k-1},\bar{x}_{k-2},...,\bar{x}_{-m})$ obtained by complementing every digit in the sequence corresponding to X

Selecting R in Radix-Complement Rep.

$$\begin{array}{rcccccl}
 & X & x_{k-1} & x_{k-2} & \cdots & x_{-m} & \\
 & +\overline{X} & \bar{x}_{k-1} & \bar{x}_{k-2} & \cdots & \bar{x}_{-m} & \\
\hline
 & & (r-1) & (r-1) & \cdots & (r-1) & \\
+ulp & & & & & 1 & \\
\hline
 & 1 & 0 & 0 & \cdots & 0 & = r^k
\end{array}$$

- $X+\bar{X}+ulp = r^k$
- Result stored into a register of length n(=k+m)
- Most significant digit discarded - final result is zero
- In general, storing the result of any arithmetic operation into a fixed-length register is equivalent to taking the remainder after dividing by r^k
- $r^k - X = \bar{X} + ulp$
- Selecting $R = r^k$: $R - X = r^k - X = \bar{X} + ulp$
- Calculation of R-X - simple and independent of k
- This is radix-complement representation
- $R=r^k$ discarded when calculating R+(X-Y) - no correction needed when X+(R-Y) is positive (X>Y)

Example - Two's Complement

- $r=2$, $k=n=4$, $m=0$, $ulp=2^0=1$
- Radix complement (called two's complement in the binary case) of a number $X = 2^4 - X$
- It can instead be calculated by $\bar{X}+1$
- 0000 to 0111 represent positive numbers 0_{10} to 7_{10}
  - The two's complement of 0111 is 1000+1=1001 - it represents the value $(-7)_{10}$
  - The two's complement of 0000 is 1111+1=10000=0 mod 2^4 - single representation of zero
- Each positive number has a corresponding negative number that starts with a 1
- 1000 representing $(-8)_{10}$ has no corresponding positive number
- Range of representable numbers is $-8 \leq X \leq 7$

The Two's Complement Representation

Sequence	Two's complement	One's complement	Signed-magnitude
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	−1	−0	−7
1110	−2	−1	−6
1101	−3	−2	−5
1100	−4	−3	−4
1011	−5	−4	−3
1010	−6	−5	−2
1001	−7	−6	−1
1000	−8	−7	−0

Example - Addition in Two's complement

- Calculating X+(-Y) with Y>X - 3+(-5)

```
        0011     3
   +    1011    -5
   ___________________
        1110    -2
```

- Correct result represented in the two's complement method - no need for preliminary decisions or post corrections

- Calculating X+(-Y) with X>Y - 5+(-3)

```
        0101     5
   +    1101    -3
   ___________________
     1  0010     2
```

- Only the last four least significant digits are retained, yielding 0010

A 2nd Alternative for R : Diminished-Radix Complement Representation

- Selecting R as $R=r^k - ulp$
- This is the diminished-radix complement
- $R - X = (r^k - ulp) - X = \bar{X}$
- Derivation of the complement is simpler than the radix complement
- All the digit-complements $\bar{x}_i$ can be calculated in parallel - fast computation of $\bar{X}$
- A correction step is needed when $R+(X-Y)$ is obtained and $X-Y$ is positive

Example - One's Complement in Binary System

- $r=2$, $k=n=4$, $m=0$, $ulp=2^0=1$
- Diminished-radix complement (called one's complement in the binary case) of a number X = $(2^4 - 1) - X = \bar{X}$
- As before, the sequences 0000 to 0111 represent the positive numbers 0_{10} to 7_{10}
- The one's complement of 0111 is 1000, representing $(-7)_{10}$
- The one's complement of zero is 1111 - two representations of zero
- Range of representable numbers is $-7 \le X \le 7$

Comparing the Three Representations in a Binary System

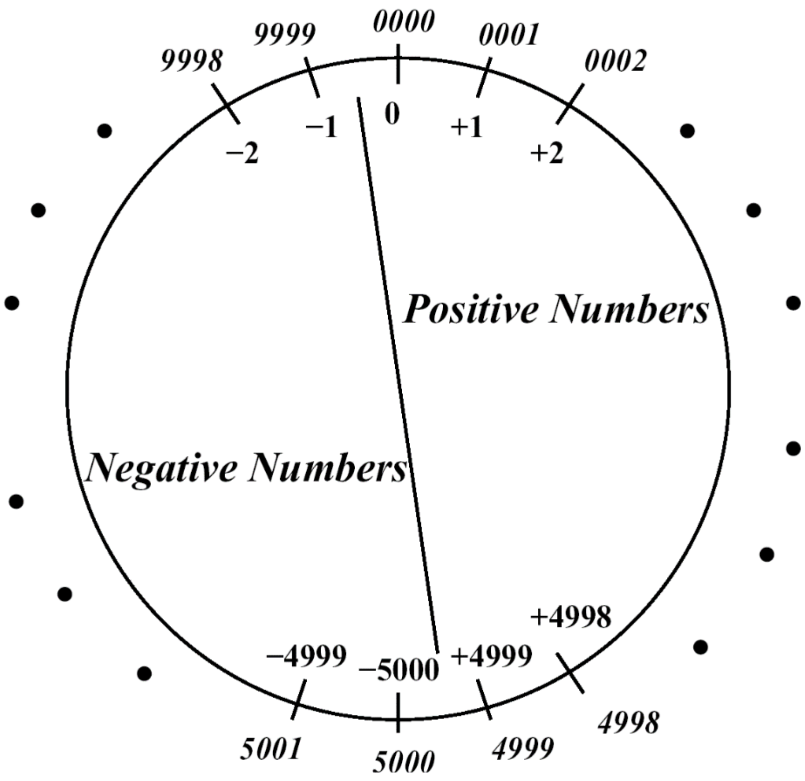
Sequence	Two's complement	One's complement	Signed-magnitude
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	-1	-0	-7
1110	-2	-1	-6
1101	-3	-2	-5
1100	-4	-3	-4
1011	-5	-4	-3
1010	-6	-5	-2
1001	-7	-6	-1
1000	-8	-7	-0

Range of Representable Numbers in Complement Methods

- **Binary system** - most significant digit is 0 or 1 - a “true" sign digit in all three methods
- **Non-binary system** - restricting the most significant digit to 0 and r-1 reduces the number of utilized sequences to $2r^{n-1}$ out of r^n
- ***Alternative*** - let the most significant digit assume all values and partition r^n equally between positive and negative values
- To have unambiguous representations, the regions for positive and negative numbers should not overlap - $|X| \leq R/2$
- If $X=R/2+1$ is included in the region of representable numbers, then the negative number $-X$ is represented by $R-X=R/2-1$ - already representing a positive number

Example: Radix-Complement Decimal System

- Leading digit 0,1,2,3,4 - positive
- Leading digit 5,6,7,8,9 - negative
- Example - n=4
- 0000 to 4999 - positive
- 5000 to 9999 - negative - (-5000) to -1
- Range - $-5000 \leq X \leq 4999$
- Y=1234
- Representation of -Y=-1234 - radix complement R-Y with $R=10^4$
- $R-Y = \bar{Y} + ulp$
- Digit complement = 9 - digit ; ulp=1
- $\bar{Y}$=8765 ; $\bar{Y}$+1 =8766 - representation of -Y
- $Y+(-Y)=1234+8766=10^4 = 0 \mod 10^4$



The Two's Complement Representation

From now on, the system is: r=2, k=n, ulp=1

- Range of numbers in two's complement method: $-2^{n-1} \leq X \leq 2^{n-1} - ulp$ $(ulp=2^{0}=1)$
- Slightly asymmetric - one more negative number
- -2^{n-1} (represented by 10⋯0) does not have a positive equivalent
- A complement operation for this number will result in an overflow indication
- On the other hand, there is a unique representation for 0

Numerical Value of a Two's Complement Representation

- Numerical value X of representation $(x_{n-1}, x_{n-2}, \ldots, x_0)$ in two's complement -
- If $x_{n-1}=0$ - $X = \sum_{i=0}^{n-1} x_i 2^i,$
- If $x_{n-1}=1$ - negative number - absolute value obtained by complementing the sequence (i.e., complementing each bit and adding 1) and adding a minus sign
- Example - Given the 4-tuple 1010 - negative - complementing - 0101+1=0110 - value is 6 - original sequence is -6

Different Calculation of Numerical Value

$$\boxed{X = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i.}$$

- **Example** - **1010** - **X=-8+2=-6**
- **Proof** - **If $x_{n-1}=0$ - same equation**
- **If $x_{n-1}=1$ -**

$$
\begin{aligned}
-[\overline{X}+ulp] &= -\left[\sum_{i=0}^{n-2} \bar{x}_i 2^i + 1\right] = -\left[\sum_{i=0}^{n-2} (1-x_i) 2^i + 1\right] \\
&= -\left[\sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} x_i 2^i + 1\right] = -\left[\left(2^{n-1}-1\right) - \sum_{i=0}^{n-2} x_i 2^i + 1\right] \\
&= -2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i
\end{aligned}
$$

which is the previous equation for $x_{n-1}=1$

The One's Complement Representation

- Derivation of one's complement - simpler than two's complement
- Calculating $\bar{x}_i=1-x_i$ for each digit - Boolean complement, can be done in parallel for all digits
- Symmetric range of representable numbers
  $-(2^{n-1}\ -ulp) \le\ X \le\ 2^{n-1}\ -\ ulp\quad (ulp=2^{0}=1)$
- As a result - two representations of zero
  - positive zero: 000...0; negative zero: 111...1
- Calculating the numerical value of a sequence

$$X \;=\; -x_{n-1}(2^{n-1}-ulp)+\sum_{i=0}^{n-2} x_i 2^i$$

- Example - 4-tuple 1001 - X=-7+1=-6

Addition and Subtraction

- In signed-magnitude representation -
  - Only magnitude bits participate in adding/subtracting - sign bits are treated separately
  - Carry-out (or borrow-out) indicates overflow
- Example -

```
   0     1001    +9
   0   + 0111    +7
   ----------------------------
   0   1 0000     0= 16 mod 16
```

- Final result positive (sum of two positive numbers) but wrong
- In both complement representations -
  - All digits, including the sign digit, participate in the add or subtract operation
  - A carry-out is not necessarily an indication of an overflow

Addition/Subtraction in Complement Methods

- Example - (two's complement)

$$\begin{array}{rr} 01001 & 9 \\ 11001 & -7 \\ \hline 1\;\;00010 & 2 \end{array}$$

  - * Carry-out discarded - does not indicate overflow

- In general, if X and Y have opposite signs - no overflow can occur regardless of whether there is a carry-out or not

- Examples - (two's complement)

$$\begin{array}{cccccccl} & 0 & 0 & 1 & 0 & 1 & 5 & \\ + & 1 & 0 & 1 & 1 & 0 & -10 & \\ \hline & 1 & 1 & 0 & 1 & 1 & -5 & \text{No carry-out} \end{array}$$

$$\begin{array}{cccccccl} & 0 & 1 & 0 & 1 & 0 & 10 & \\ + & 1 & 1 & 0 & 1 & 1 & -5 & \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 5 & \text{Carry-out} \end{array}$$

Addition/Subtraction - Complement - Cont.

- If X and Y have the same sign and result has different sign - overflow occurs
- Examples - (two's complement)

```
        10111   -9
        10111   -9
   ------------------------------
   1    01110    14 = -18 mod 32
```

  - Carry-out and overflow

```
        01001    9
        00111    7
   ------------------------------
   0    10000   -16 = 16 mod 32
```

  - No carry-out but overflow

Addition/Subtraction - One's Complement

- Carry-out - indicates the need for a correction step
- Example - adding positive X and negative $-Y$

  $X+(2^n - ulp)-Y = (2^n - ulp)+(X-Y)$
- If $X>Y$ - correct result is $X-Y$
- 2^n represents the carry-out bit - discarded in a register of length n
- Result is $X-Y-ulp$ - corrected by adding ulp
- Example -

		01001	9
	+	11000	-7
	1	00001	
Correction		1	ulp
		00010	2

- The generated carry-out is called end-around carry - it is an indication that a 1 should be added to the least significant position

Addition/Subtraction - One's Complement -Cont.

- If $X<Y$ - the result $X-Y=-(Y-X)$ is negative
- Should be represented by $(2^n-ulp) - (Y-X)$
- There is no carry-out - no correction is needed
- Example -

$$\begin{array}{rr} 10110 & -9 \\ 00111 & 7 \\ \hline 11101 & -2 \end{array}$$

- No end-around carry correction is necessary in two's complement addition

Subtraction

- In both complement systems - subtract operation, $X-Y$, is performed by adding the complement of Y to X
- In the one's complement system -

  $X-Y=X+\bar{Y}$
- In the two's complement system -

  $X-Y=X+(\bar{Y}+ulp)$
- This still requires only a single adder operation, since ulp is added through the forced carry input to the binary adder

Arithmetic Shift Operations

- Another way of distinguishing among the three representations of negative numbers - the infinite extensions to the right and left of a given number
- Signed-magnitude method - the magnitude $x_{n-2},...,x_0$ can be viewed as the infinite sequence $...,0,0,\{x_{n-2},...,x_0\},0,0,...$
- Arithmetic operation resulting in a nonzero prefix - an overflow
- Radix-complement scheme - the infinite extension is $...,x_{n-1},x_{n-1},\{x_{n-1},...,x_0\},0,0,...$ (x_{n-1} - the sign digit)
- Diminished-radix complement scheme - the sequence is $...,x_{n-1},x_{n-1},\{x_{n-1},...,x_0\},x_{n-1},x_{n-1},...$

Arithmetic Shift Operations - Examples

- 1010., 11010.0, 111010.00 - all represent -6 in two's complement
- 1001., 11001.1, 111001.11 - all represent -6 in one's complement
- Useful when adding operands with different numbers of bits - shorter extended to longer
- Rules for arithmetic shift operations: left and right shift are equivalent to multiply and divide by 2, respectively

Two's complement

Sh.L{00110=6}=01100=12
Sh.R{00110=6}=00011=3
Sh.L{11010=-6}=10100=-12
Sh.R{11010=-6}=11101=-3

One's complement

Sh.L{11001=-6}=10011=-12
Sh.R{11001=-6}=11100=-3