Number Systems

010.133
Digital Computer Concept and Practice
Spring 2013

Lecture 02

Number Systems

- A number system is a system of representing numbers
  - A set of basic symbols (called digits or numerals)
  - The ways in which the digits can be combined to represent the numbers
- Represent integers, fractions, or mixed numbers
  - A mixed number = integer part + fraction part
    - The integer part tells you the whole
    - The fraction part is less than one whole
    - A radix point (.) separates the integer part and the fraction part
  - E.g., 3.14159265

Positional Number Systems

- A number is represented by a string of digits
  - The value of each digit in the string is determined by the position it occupies
- A number is represented by a string of digits and each digit position has an associated weight
- Decimal number system
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - The decimal number of the form $d_{p-1}d_{p-2}\cdots d_1d_0.d_{-1}d_{-2}\cdots d_{-n}$ has the value,

$$\sum_{i=-n}^{p-1} d_i \cdot 10^i$$

  - Base (or radix) = 10
  - Radix point = decimal point (.)
  - The leftmost digit (d_{p-1}) - most significant digit (MSD)
  - The rightmost digit (d_{-n}) - least significant digit (LSD)
- For example, 754.82 = 7×100 + 5×10 + 4×1 + 8×0.1 + 2×0.01

Base-r Number System

- When we replace the base 10 with some other whole number r, then we have base-r number system
  - r distinct digits
  - The weight in position i is ri
  - The base-r number of the form $x_{p-1}x_{p-2}\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-n}$ has the value,

$$\sum_{i=-n}^{p-1} x_i \cdot r^i$$

- When r ≤ 10, the first r decimal digits serve as the digits
- When r > 10, the first r – 10 uppercase letters of the alphabet in addition to 10 decimal digits

Commonly Used Number Systems

Name	Base	Digits
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- 101
  - Base-2: 101_2
  - Base-8: 101_8
  - Base-10: 101_{10}
  - Base-16: 101_{16}

Scientific Notation

- A scheme of representing decimal numbers that are very small or vary large
- A number is represented in the form:

$$x \times 10^y$$

  - x: coefficient, significand, or mantissa
    - Real number
  - y: exponent
    - Integer
- For example,
  - $-320000000 = -32.0 \times 10^7$

Normalized Scientific Notation

- There are many ways to represent a number in scientific notation
- Adopt the convention of making the significand, x, always in the range:
  - $1 \leq |x| < 10$
  - We can not express zero

Significant Digits

- Those digits whose removal changes the numerical value
- A number's precision or accuracy
  - The number of significant digits it contains
- Leading zeroes
  - Any consecutive zeroes that appear in the leftmost positions of the number's representation
- Trailing zeroes
  - Any consecutive zeroes in the number's representation after which no other digits follow
- Trailing zeroes that appears to the right of a radix point and leading zeroes are insignificant
  - For example, the removal of leading and trailing zeroes in 0003.140010 does not affect the value

Fixed-point and Floating-point Numbers

- Fixed-point numbers
  - Have an implicit radix point at some fixed position
  - For example,
    - Integer: the radix point is immediately to the right of its LSD
    - Fraction: the radix point is immediately to the left of its MSD
- Floating-point numbers
  - Have a radix point that can be placed anywhere relative to their significant digits
    - The position is indicated separately and encoded in representation
  - Scientific notation is closely related to the floating-point numbers

Binary Number System

- Directly related to the Boolean logic used in the computer
- The computer internally represents numeric data in a binary form
- Binary digits (bits): 0 or 1
- The general form of a binary number is $b_{p-1}b_{p-2}\cdots b_1b_0.b_{-1}b_{-2}\cdots b_{-n}$
  - Its value is:

$$\sum_{i=-n}^{p-1} b_i \cdot 2^i$$

- Base = 2
- Radix point = binary point (.)
- b_{p-1} : MSB, b_{-n} : LSB
- For example,
  - $10011 = 1\cdot2^4+1\cdot2^1+1\cdot2^0 = 19$
  - $101.001 = 1\cdot2^2+1\cdot2^0+1\cdot2^{-3} = 5.125$

Octal Number System

- Is useful for representing multi-bit binary numbers
  - Three bits in a binary number can be uniquely represented with a single octal digit
- For example,
  - 100011001110 = 100 011 001 110 = 4316_8
  - 10.1011001011 = 010 . 101 100 101 100 = 2.5454_8

Hexadecimal Number System

- Is also useful for representing multi-bit binary numbers
  - Each group of four bits in a binary number can be uniquely represented by a single hexadecimal digit
- For example,
  - $100011001110 = 1000\ 1100\ 1110 = 8CE_{16}$
  - $10.1011001011 = 0010\ .\ 1011\ 0010\ 1100 = 2.B2C_{16}$

4-bit Binary, Octal, Hexadecimal, and Decimal

Binary	Octal	Hexadecimal	Decimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

Unsigned and Signed Binary Numbers

- N bits can represent 2^n different binary numbers
  - Fixed precision
- Unsigned numbers
  - To represent only positive values with n-bit binary numbers
- Signed numbers
  - To encode positive numbers and negative numbers in binary
  - Several ways
    - Setting the MSB to 1 and using the remaining bits to represent the value
      - The MSB is referred to as the sign bit
- To keep the computer hardware implementation as simple as possible, almost all today's computers internally use two's complement representation

Decimal to Binary Conversion for Integers

Let N be the decimal number and $b_{n-1}b_{n-2} \cdots b_0$ be the binary number after the conversion.

1. Set i to 0.
2. Divide N by 2 and obtain a quotient and a remainder.
3. Set b_i to the remainder and N to the quotient.
4. If N is not zero, set i to $i+1$ and go to step 2.

Converting 108 to binary

i	N	b_i
0	$108/2 = 54$	0 (LSB)
1	$54/2 = 27$	0
2	$27/2 = 13$	1
3	$13/2 = 6$	1
4	$6/2 = 3$	0
5	$3/2 = 1$	1
6	$1/2 = 0$	1 (MSB)

Decimal to Binary Conversion for Fractions

Let N be the decimal fraction and $b_{-1}b_{-2} \cdots b_{-n}$ be the binary number after the conversion.

1. Set i to 1.
2. Multiply N by 2 and set N to the result.
3. If N is less than 1, set b_{-i} to 0.
4. Otherwise, set b_{-i} to 1 and N to N-1.
5. If i is less than n, set i to $i+1$ and go to step 2.

Converting 0.735 to binary

i	N	b_{-i}
1	$0.735 \times 2 = 1.47$	1
2	$0.47 \times 2 = 0.94$	0
3	$0.94 \times 2 = 1.88$	1
4	$0.88 \times 2 = 1.76$	1

Truncation!

Word

- A group of bits that are handled as a unit by the computer
- Word size
  - The number of bits in a word
    - An important characteristic of a computer architecture
  - Determines the range of possible numbers that can be represented
    - For example, 8 bits can represent $256 = 2^8$ distinct numeric values
  - The precision of expressing a numerical quantity

Dyadic Fraction

- A rational number whose denominator is a power of two
  - For example, 1/2 or 3/16 is a dyadic fraction, but not 1/3
  - Dyadic decimal fractions convert to finite binary fractions
  - Represented in full precision if the word size is greater than or equal to the length of the binary representation
- Non-dyadic decimal fractions convert to infinite binary fractions
  - A repeating sequence of the same bit pattern
- Truncation (also known as chopping)
  - The process of discarding any unwanted digits of a number
- Rounding
  - A number is replaced with another number that is as close to the original number as possible

Decimal to Binary Conversion for Mixed Numbers

- Convert the integer part and the fraction part to binary form separately
- Then, combine the results
- For example, a mixed number 108.731_{10} is converted to 1101100.1011_2

Decimal to Octal Conversion

- Similar to the decimal to binary conversion

i	N	b_i	
0	$108/8 = 13$	4	(LSB)
1	$13/8 = 1$	5	
2	$1/8 = 0$	1	(MSB)

i	N	b_{-i}
1	$0.735 \times 8 = 5.88$	5
2	$0.88 \times 8 = 7.04$	7
3	$0.04 \times 8 = 0.32$	0
4	$0.32 \times 8 = 2.56$	2

Converting 108.731 to an octal number

Decimal to Hexadecimal Conversion

- Similar to the decimal to binary conversion

i	N	b_i	
0	$108/16 = 6$	$12(C)$	(LSB)
1	$6/16 = 0$	6	(MSB)

i	N	b_{-i}
1	$0.735 \times 16 = 11.76$	$11(B)$
2	$0.76 \times 16 = 12.16$	$12(C)$
3	$0.16 \times 16 = 2.56$	2
4	$0.56 \times 16 = 8.96$	8

Converting 108.731 to a hexadecimal number

Another Way

- Converting a given decimal number to a binary number
- Then, convert the result to an equivalent octal or hexadecimal number
- However, the precision matters

$$108.731_{10} = 001\ 101\ 100\ .\ 101\ 111\ 000\ 010_2$$
$$= 154.5702_8$$

$$108.731_{10} = 0110\ 1100\ .\ 1011\ 1100\ 0010\ 1000_2$$
$$= 6C.BC28_{16}$$

Unsigned Binary Addition

- Similar to decimal addition

$$
\begin{array}{rccccc}
\text{carry} & 0 & 0 & 0 & 1 & \\
 & & 1 & 0 & 0 & 1 \\
+ & & 0 & 1 & 0 & 1 \\
\hline
 & 0 & 1 & 1 & 1 & 0
\end{array}
$$

Overflow

- An overflow occurs when a computation produces a value that falls outside the range of values that can be represented
  - The carry bit from the MSB indicates an overflow occured

$$
\begin{array}{rccccc}
\text{carry} & 1 & 1 & 1 & 1 & \\
 & & 1 & 0 & 1 & 1 \\
+ & & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 0
\end{array}
$$

Unsigned Binary Subtraction

- Similar to decimal subtraction

$$
\begin{array}{rccccc}
\text{borrow} & 0 & 1 & 0 & 0 & \\
 & & 1 & 0 & 0 & 1 \\
- & & 0 & 1 & 0 & 1 \\
\hline
 & 0 & 0 & 1 & 0 & 0
\end{array}
$$

Unsigned Binary Multiplication

- Similar to decimal multiplication

$$
\begin{array}{rrrrrrrr}
 & & & & 1 & 0 & 0 & 1 \\
\times & & & & 0 & 1 & 0 & 1 \\
\hline
 & & & & 1 & 0 & 0 & 1 \\
 & & & 0 & 0 & 0 & 0 & \\
 & & 1 & 0 & 0 & 1 & & \\
 & 0 & 0 & 0 & 0 & & & \\
\hline
 & 0 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
$$

Unsigned Binary Division

- Similar to decimal Division

$$
\begin{array}{rcccccccl}
 & & & & & 1 & 1 & 1 & \text{quotient} \\ \cline{3-8}
100 &) & 1 & 1 & 1 & 0 & 1 & & \\
 & & 1 & 0 & 0 & & & & \\ \cline{3-8}
 & & 0 & 1 & 1 & 0 & 1 & & \\
 & & & 1 & 0 & 0 & & & \\ \cline{3-8}
 & & & 0 & 1 & 0 & 1 & & \\
 & & & & 1 & 0 & 0 & & \\ \cline{3-8}
 & & & & 0 & 0 & 1 & & \text{remainder}
\end{array}
$$

One's Complement Representation

- One of the methods to represent a negative number
  - Not popular
- The bitwise inversion (flipping 0's for 1's and vice-versa) of a negative number's positive counterpart
  - The one's complement representation of -6 (- 0110) is 1001
- With n bits,
  - The maximum: $2^{n-1} - 1$
  - The minimum: $-2^{n-1} - 1$

Positive		Negative	
Ones' complement	Decimal	Ones' complement	Decimal
0000	0	1111	-0
0001	1	1110	-1
0010	2	1101	-2
0011	3	1100	-3
0100	4	1011	-4
0101	5	1010	-5
0110	6	1001	-6
0111	7	1000	-7

One's Complement Representation (contd.)

- Two zeroes
  - For example, 0000 and 1111 are the two 4-bit ones' complement representations of zero
- Addition is similar to the unsigned binary addition
  - If there is a carry from the MSB (called an end-around carry), then add the carry back into the resulting sum

$$\begin{array}{lccccccl}
\text{carry} & 1 & 1 & 0 & 0 & & & \\
 & & 0 & 1 & 0 & 0 & & (4) \\
+ & & 1 & 1 & 1 & 0 & & (-1) \\
\hline
 & & 0 & 0 & 1 & 0 & & \\
 & & & & & 1 & & (\text{add the end-around carry}) \\
\hline
 & & 0 & 0 & 1 & 1 & & (3)
\end{array}$$

Two's Complement Representation

- The most common method of representing signed integers internally in computers
- Simplifies the complexity of the ALU
  - The same circuit that is used to implement arithmetic operations for unsigned numbers
  - The major difference is the interpretation of the result
- With n bits,
  - The maximum: $2^{n-1} - 1$
  - The minimum: -2^{n-1}

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

Converting to the Two's Complement

- Convert the number to its ones' complement
- Then one is added to the result to produce its two's complement
- Another way of the conversion
  - Flipping all bits but the first least significant 1 and all the trailing 0s
- For example, the two's complement representation of –6 (–0110) is 1010

Two's Complement Addition and Subtraction

- Two's complement addition is exactly the same as that of unsigned binary numbers
- Subtraction can be handled by converting it to addition:
  - $x - y = x + (-y)$
- Overflow occurs if n+1 bits are required to contain the result from an n-bit addition or subtraction
  - If the sign bits were the same for both numbers and the sign of the result is different to them, an overflow has occurred

1. Perform n-bit unsigned binary addition.
2. If the carry into the MSB is not equal to the carry out of the MSB, an overflow has occurred.

$$
\begin{array}{rccccc l}
\text{carry} & 0 & 1 & 1 & 1 & & \\
 & & 0 & 1 & 1 & 1 & (7) \\
+ & & 0 & 1 & 1 & 1 & (7) \\
\hline
 & 0 & 1 & 1 & 1 & 0 &
\end{array}
$$

Shift Operations

- Every bit in the operand is moved a given number of positions to a specified direction
  - Shift left: <<
  - Shift right: >>
- Logical shift
  - Moves bits to the left or right
  - The bits that fall off at the end of the word are discarded
  - The vacant positions in the opposite end are filled with 0s
- Arithmetic shift
  - The left shift is the same as the logical left shift
  - The right shift is different
    - The leftmost bits are filled with the sign bit of the original number

	$x >> 3$	$x << 3$
Logical shift	00010011	11101000
Arithmetic shif	11110011	11101000

The difference between 8-bit logical shift and arithmetic shift for x = 10011101

Multiplication or Division by Powers of Two

- An n-bit logical left shift operation on unsigned integers is equivalent to multiplication by 2^n
  - $00001011_2\,(13_{10}) << 2_{10} = 13 \times 2^2 = 00101100_2\,(52_{10})$
- An n-bit logical right shift on unsigned integers is equivalent to division by 2n
  - $00101101_2\,(53_{10}) >> 2_{10} = 53 \div 2^2 = 00001011_2\,(13_{10})$
  - We obtain the quotient of the division

Multiplication or Division by Powers of Two (contd.)

- Using arithmetic shifts, we can efficiently perform multiplication or division of signed integers by powers of two

- The floor of an integer x (denoted by $\lfloor x \rfloor$)
  - The greatest integer less than or equal to x
  - For example, $\lfloor 11/2 \rfloor = 5$ and $\lfloor -3/2 \rfloor = -2$

Multiplication or Division by Powers of Two (contd.)

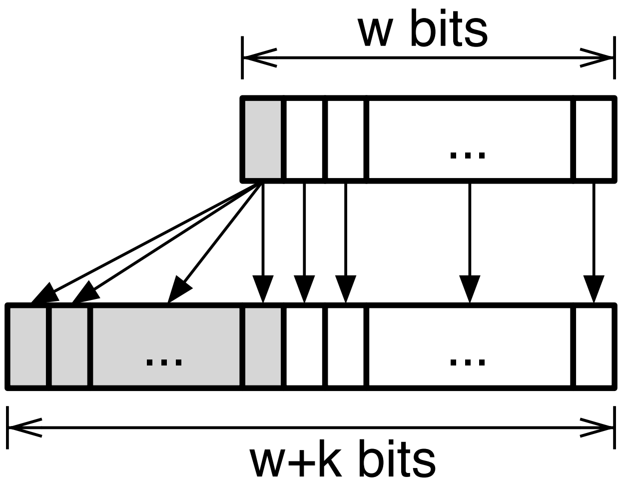
- An arithmetic shift operation on an integer in two's complement representation takes the floor of the result of multiplication or division
  - 1101_2 (−3) << 1 = 1010_2 (-6)
  - 1101_2 (−3) >> 1 = 1110_2 (-2)
  - 1101_2 (−3) >> 2 = 1111_2 (-1)
  - 1100_2 (−4) << 2 = 0000_2 (0) Overflow
  - 0100_2 (4) << 1 = 1000_2 (-8) Overflow

Note on Arithmetic Right Shifts

- In two's complement representation, an arithmetic right shift is not equivalent to division by a power of two
- An arithmetic right shift on a 4-bit -1 (1111_2),
  - You still get -1 as a result
- For negative values, the ANSI/ISO C99 standard does not specify the definition of the C language's right shift operator
  - The behavior of the right shift operator is dependent on the C compiler

Sign Extension

- When converting an w-bit integer in two's complement representation into the one with w+k bits and the same value
  - Perform sign extension



Decimal	4-bit	8-bit
6	0110	00000110
-6	1010	11111010