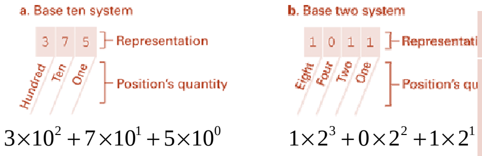
Binary numbers

Binary number system

- Computer (electronic) systems prefer binary numbers
- Binary number: represent a number in base-2



Bit pattern	Hexadecimal representation
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

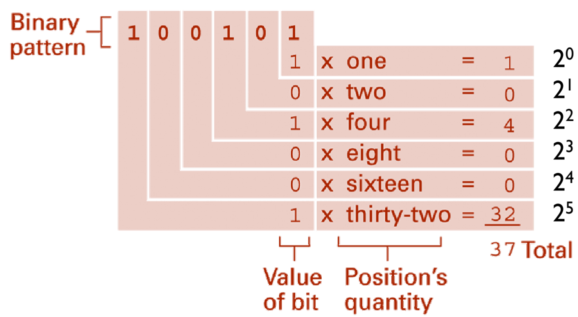
- Some terminology
  - **Bit:** a binary digit (0 or 1)
  - **Hexadecimal notation** (十六進位)
    - Represents each 4 bits by a single symbol
    - Example: A3 denotes 1010 0011

Outline

- Integer: decimal-binary conversion
- Integer addition
- Negative integer (2's complement representation)
- Real numbers (floating point representation)

Binary to decimal

- What is the decimal number of 100101_b?



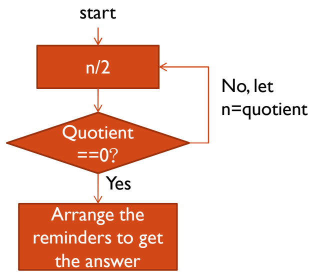
Decimal to binary

- What is the binary number of 13_d?
  - Step 1: Divide the value by 2 and record the remainder
  - Step 2: If quotient is not zero, use the quotient as the new value and repeat step 1
  - Step 3: The binary representation is the recorded remainders listed from right to left

2)13 → 6	Quotient = 6 Remainder = 1
2)6 → 3	Quotient = 3 Remainder = 0
2)3 → 1	Quotient = 1 Remainder = 1
2)1 → 0	Quotient = 0 Remainder = 1

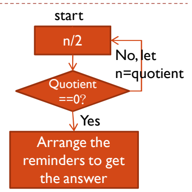
Algorithm

- A finite sequence of instructions to describe a systematical method to solve a problem
- We can represent the 'decimal-to-binary' algorithm by a flow chart
  - It has starting point [step 1]
  - Step 2 is a condition statement
  - Step1+step 2 is a loop statement
  - The problem size is shrunk after each loop
  - The loop can be terminated [when quotient ==0]



- Homework: write an algorithm to convert binary to decimal

十進位轉二進位演算法 Three algorithms

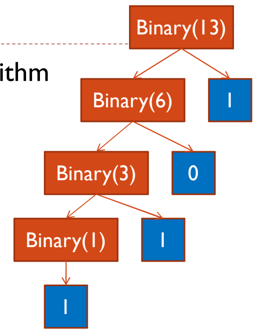


- Problem: converting n_d to m_b.
- Algorithm 1: as mentioned in the last class
- Algorithm 2:
  - $m_b = 0$
  - For i = 1 to n_d
    - $m_b = m_b + 1$

What is the cost of $m_b = m_b+1$?

- Algorithm 3:
  - $m_b = 0$
  - While n_d is not 0
    - Find $2^k \le n_d < 2^{k+1}$ → What is the cost of finding k?
    - $m_b = m_b + 2^k$ → The cost of m_b+2^k is just putting 1 at the kth position of m_b
    - $n_d = n_d - 2^k$ → What is the cost of $n_d = n_d - 2^k$?

Recursive algorithm



- The first algorithm is a recursive algorithm
  - To answer what the binary of 13_d is, you need to answer what the binary of 6_d is.
  - To answer what the binary of 6_d is, you need to answer what the binary of 3_d is.
  - To answer what the binary of 3_d is, you need to answer what the binary of 1_d is.
- Recursive algorithm
  - Turn a big problem into one or several smaller subproblems
  - Each subproblem is identical to the original one except size
    - Thus, they can be solved by the same method.
  - Need a termination condition.

Integer addition

- One bit addition

```
  0     1     0     1
+ 0   + 0   + 1   + 1
---   ---   ---   ---
  0     1     1    10
```

- What is 5_d+9_d using binary number representation?

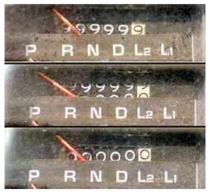
```
  0 1 0 1   ← 5
+ 1 0 0 1   ← 9
---------
  1 1 1 0   → 14
```

Another example

```
   1 1 1   1
   00111010
+  00011011
-----------
   01010101
```

Binary number in computer systems

- Mathematically, a binary integer can have arbitrary number of bits.
- In computer systems, all data has a limited number of bits.
  - For example, there are different sized (data type) integers
    - char: 8 bits: [0, 255]
    - unsigned short: 16 bits: [0, 65535]
    - unsigned int: 32 bits: [0, 4294967295]
- **Overflow**: when adding two integers, the result exceeds the numerical range of the data type
  - Ex: (char) 123+(char) 234 = (char) 101



How about negative integers?

- We use sign to distinguish positive and negative numbers.
- In computer system, we can use a bit to present the sign.
  - In a 4 bit integer, 0001 for 1 and 1001 for -1. (sign bit)
- Good for notation, but difficult for calculation.
  - One bit subtraction

```
  0     1     1     0
- 0   - 0   - 1   - 1
---   ---   ---   ---
  0     1     0   -1 1 = -1*2+1*1=-1
```

- Example: 4-1

```
   -1  1
  0 1 0 0
- 0 0 0 1
---------
  0 0 1 1
```

Negative integer

- Can we design a negative number representation such that 4-1=4+(-1) can be done easily (as easy as addition)?
- Hint: all number representation in computers has a finite number of bits.
- If we use 4 bits to represent an integer
  - Zero is 0000, and one is 0001. What is -1?
  - Find b_3, b_2, b_1, b_0 such that

$$
\begin{array}{r}
b_3\ b_2\ b_1\ b_0 \\
+\ 0\ 0\ 0\ 1 \\
\hline
\boxed{1}\,0\ 0\ 0\ 0
\end{array}
$$

This 1 will be "truncated" since it is a 4 bits integer

- Thus, we can use 1111_b to represent -1_d.

Two's complement

- This number representation is called the "two's complement".
- Algorithm to find the 2's complement of an integer
  - Step 1: invert each bit, 0 to 1 and 1 to 0
  - Step 2: Add 1.

$6_d = 0110_b \rightarrow 1001_b$

$$
\begin{array}{r}
1001_b \\
+\ 0001_b \\
\hline
1010_b = -6_d
\end{array}
\qquad
\begin{array}{rr}
0110_b & (6) \\
+\ 1010_b & (-6) \\
\hline
1\ 0000_b & (0)
\end{array}
$$

truncated

$-6_d = 1010_b \rightarrow 0101_b$

$$
\begin{array}{r}
0101_b \\
+\ 0001_b \\
\hline
0110_b = 6_d
\end{array}
$$

- Textbook uses a different algorithm, which is used in circuit design

Data type

- 1010_b can be 10_d or -6_d. How to tell?
- Given a bit pattern, one need to specify its 'data type'
  - 1010_b is 10_d for unsigned 4 bit int and -6_d for signed 4 bit int
- In C, there are data types for signed and unsigned integer

Data type	Number of bits	Numerical range
char	8	[0,255]
unsigned short	16	[0,65525]
short	16	[−32768 , +32767]
unsigned int (long)	32	[0, 4,294,967,295]
int, long	32	[−2,147,483,648 , +2,147,483,647]
unsigned long long	64	[0, $2^{64}-1$]
long long	64	[-2^{63}, $2^{63}-1$]

2's complement to decimal

- Give a 2's complement representation, how to know it is a positive number or a negative number?
- Observe:
  - The left most bit of all positive numbers is 0, and of all negative numbers is 1
  - The left most bit of singed data type is called the 'sign bit'
- Converting 2's complement to decimal
  - Step1: check the sign bit to tell the sign
  - Step 2: If it is a negative number, convert it to its 2's complement
  - Step 3: Convert the number to decimal and add the sign

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

- Step 2 and step 3 are like subroutines, which invoke other algorithms mentioned before.

2’s complement addition

- Examples

Problem in base ten		Problem in two's complement		Answer in base ten
3 + 2	→	0011 + 0010 = 0101	→	5
−3 + −2	→	1101 + 1110 = 1011	→	−5
7 + −5	→	0111 + 1011 = 0010	→	2

Another type of overflow

- What is 5+4 in signed 4 bit representation?

$5_d+4_d=0101_b + 0100_b = 1001_b$

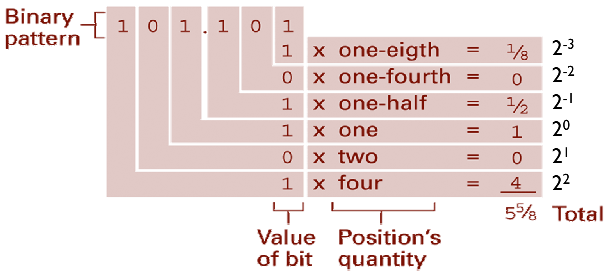
- This is another type of overflow
  - Adding two positive numbers results a negative number; or adding two negative numbers results a positive number.
  - A 4 bits 2’s complement system can only represent 7~ −8

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Fraction

- The binary number of fractions.
  - 5.625



Fraction point

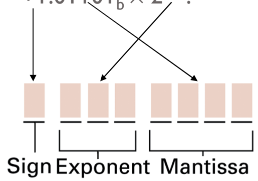
- To represent a wide range of numbers, we allow the decimal point to “float”.

$40.1_d = 4.01_d \times 10^1 = 401_d \times 10^{-1} = 0.401_d \times 10^2$

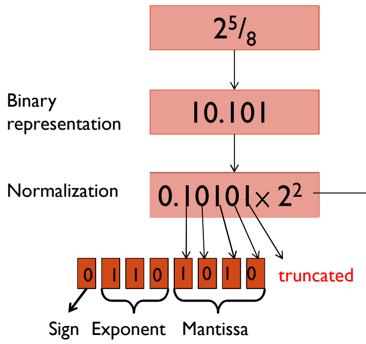
  - It is just like the scientific notation of numbers.

$101.101_b = +1.01101_b \times 2^{2_d} = +1.01101_b \times 2^{10_b}$.

- This is called the floating point representation of fractions.



Coding the value of $2^5/_8$



- Exponent uses excess notation

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Signed number representations

- Comparison of 4 bit signed integer representation by sign-bit notation, 2's complement, and excess notation

	Sign-bit notation	2's complement	Excess notation
8			1111
7	0111	0111	1110
6	0110	0110	1101
5	0101	0101	1100
4	0100	0100	1011
3	0011	0011	1010
2	0010	0010	1001
1	0001	0001	1000
0	0000, 1000	0000	0111
-1	1001	1111	0110
-2	1010	1110	0101
-3	1011	1101	0100
-4	1100	1100	0011
-5	1101	1011	0010
-6	1110	1010	0001
-7	1111	1001	0000
-8		1000	

Floating-point numbers

- In C (and most programming languages), there are two data types for real numbers

Data type	Size	Structure	Range	Precision
float	32 bits	Sign: 1 bit Exponent: 8 bits Mantissa: 23 bits	$\pm\sim 10^{-44.85}$ to $\sim 10^{38.53}$	$\sim 10^{8}$
double	64 bits	Sign: 1 bit Exponent: 11 bits Mantissa: 52 bits	$\pm\sim 10^{-323.3}$ to $\sim 10^{308.3}$	$\sim 10^{16}$

Truncation error

- Mantissa field is not large enough
  - $2^5/_8 = 2.625 \Rightarrow$ 2.5 + round off error (0.125)
- Nonterminating representation
  - $0.1 = {}^1/_{16} + {}^1/_{32} + {}^1/_{256} + {}^1/_{512} + \ldots$
  - Change the unit of measure
- Order of computation:
  - $2.5 + 0.125 + 0.125 \Rightarrow$ 2.5+0+0 =2.5
  - $2.5 + (0.125+0.125) \Rightarrow$ 2.5+0.25=2.75

More in 數值分析