CM1101 COMPUTATIONAL THINKING

COMPUTER ARCHITECTURE 2

Dr Jing Wu

Room WX/2.06B

wuj11@cardiff.ac.uk

Outline

- Radix Number Systems
  - Decimal (base 10) Number System
  - Binary (base 2) Number System
  - Octal (base 8), Hexadecimal (base 16)
  - Conversions
- Negative Numbers
  - Sign and Magnitude
  - Two's Complement
- Fixed-Point Number System
  - Range, Precision

Decimal (base 10) number system

- The radix or base of a positional number system defines the digits that can be used.
- Our "usual" system of numbers is called the decimal number system. It is based on the digits 0, 1, …, 9, and is known as base 10.

Decimal (base 10) representation of number "147"				
100's		10's		1's
1×100	+	4×10	+	7×1
1×10^2		4×10^1		7×10^0
1		4		7

Note how the exponents increase as we move left from the right-most digit.

Binary (base 2) number system

- The binary number system (also known as base 2) uses two digits: 0 or 1.
- The 0's and 1's are known as BInary digiTs or bits.
- The bases most used in computers are base 2 (binary), base 8 (octal), and base 16 (hexadecimal).

Binary (base 2) number "10010011_2" = 147_{10}							
128's	64's	32's	16's	8's	4's	2's	1's
1×128	0×64	0×32	1×16	0×8	0×4	1×2	1×1
1×2^7	0×2^6	0×2^5	1×2^4	0×2^3	0×2^2	1×2^1	1×2^0
1	0	0	1	0	0	1	1

Note how the powers of two increase as we move left from the right-most digit.

Converting from binary to decimal

$1001001.101_2 = 73.625_{10}$?

6	5	4	3	2	1	0	-1	-2	-3
2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
64	32	16	8	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$
1x64	+0x32	+ 0x16	+ 1x8	+ 0x4	+ 0x2	+ 1x1	+ 1x $\frac{1}{2}$	+ 0x $\frac{1}{4}$	+ 1x $\frac{1}{8}$

Converting from decimal to binary

Example: convert 83.375_{10} to binary.

Convert integer part first using the remainder method:

$83 \div 2 = 41$	remainder	1
$41 \div 2 = 20$	remainder	1
$20 \div 2 = 10$	remainder	0
$10 \div 2 = 5$	remainder	0
$5 \div 2 = 2$	remainder	1
$2 \div 2 = 1$	remainder	0
$1 \div 2 = 0$	remainder	1

Read backwards

Double check the answer

64's	32's	16's	8's	4's	2's	1's
1×64	0×32	1×16	0×8	0×4	1×2	1×1
1×2^6	0×2^5	1×2^4	0×2^3	0×2^2	1×2^1	1×2^0
1	0	1	0	0	1	1

Converting from decimal to binary

Example: convert 83.375_{10} to binary.

Then convert fractional part using the multiplication method:

$0.375 \times 2 = 0.75$ print 0

$0.750 \times 2 = 1.50$ print 1

$0.500 \times 2 = 1.00$ print 1

Read forwards

Once the fractional part is 0, the process is complete.

Double check the answer

$\frac{1}{2}$'s	$\frac{1}{4}$'s	$\frac{1}{8}$'s
$0\times\frac{1}{2}$	$1\times\frac{1}{4}$	$1\times\frac{1}{8}$
0×2^{-1}	1×2^{-2}	1×2^{-3}
0	1	1

Exercise

- Convert 25.125_{10} to binary
- Answer

$$25.125_{10} = 11001.001_2$$

Example:

Consider converting 0.91_{10} to binary:

$$0.91 \times 2 = 1.82$$
$$0.82 \times 2 = 1.64$$
$$0.64 \times 2 = 1.28$$
$$0.28 \times 2 = 0.56$$
$$0.56 \times 2 = 1.12$$
$$0.12 \times 2 = 0.24$$
$$0.24 \times 2 = 0.48$$
$$0.48 \times 2 = 0.96$$
$$0.96 \times 2 = 1.92$$
$$\ldots$$

The process of repeated multiplication is going on a bit!

Exact representation

- Exact conversions between decimal and binary are not always possible.
- And not necessary, if the conversion implies an accuracy not present in the original decimal data.

$$0.91_{10} \rightarrow 0.111010001\ldots_2$$

$$10^{-2} = \frac{1}{100} \qquad 2^{-9} = \frac{1}{512}$$

- Usually we stop when we have obtained a comparable degree of accuracy with the original number.
- Having decided to stop, you will need to round your answer.

Rounding binary numbers

Consider our example of 0.91_{10}=0.111010001…

- Rounding to three places of accuracy will yield 0.111_2.
- When you are rounding to N places you look at the (N+1)th place:
  - If it contains a 0, you do nothing (round down).
  - If it contains a 1, you add 1 to the Nth position (round up).
- Rounding to four places of accuracy will yield 0.1111_2.
- In this case there is a 1 in the fifth place, so we need to add 1 to the fourth.
- Rounding to two places of accuracy…

Binary addition

- Just like with decimal addition, we add numbers digit by digit, starting from the right.
- In binary system, 1+1=10.

  ↑
  carry

- Example:

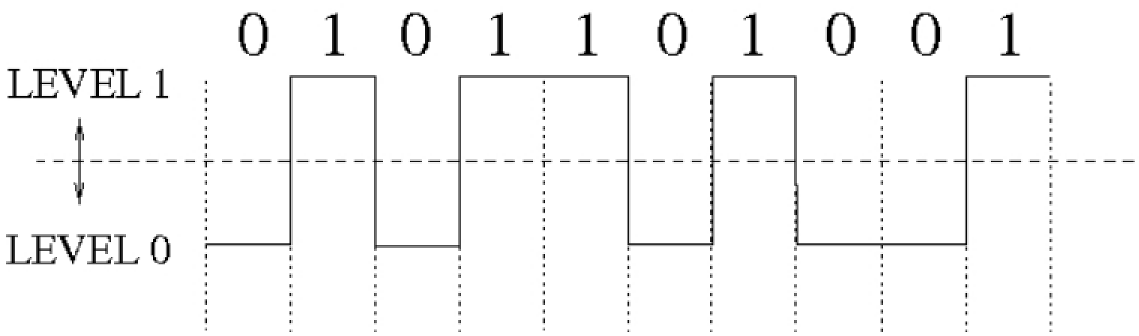
$$\begin{array}{rl} & 0\,.\,1\,1 \\ + & 0\,.\,0\,1 \\ & {\color{red}1}\;\;{\color{red}1} \\ \hline & 1\,.\,0\,0 \end{array} \qquad \begin{array}{rl} & 0\,.\,1\,1 \\ + & 0\,.\,1\,1 \\ & {\color{red}1}\;\;{\color{red}1} \\ \hline & 1\,.\,1\,0 \end{array}$$

Rounding binary numbers: Example

How would you round 0.01111_2 to four places?

- First look at the fifth position, which contains a 1.
- So you add 1 to the fourth position. Essentially, computing the sum $0.0111_2+0.0001_2$.
- The final result is 0.1000_2.
- Note: it is important to write down the trailing zeros as they indicate the value is accurate to the nearest $1/2^4$.
- Simply writing 0.1_2 would imply an accuracy only to the nearest ½.

Why do computers speak 0's and 1's?



- In electronic computers, values of 1's and 0's are represented by voltage levels.
- It is easier to make hardware components (using *e.g.* transistors) which can distinguish between and operate on two values than multiple values.
- Using two well separated signals, *i.e.* high and low voltages, there is less chance of error in the interpretation of the voltage.

Other frequently used bases

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

Converting from decimal to octal

- The digits used in octal (base 8) system are 0,1, …, 7.
- Use remainder method to convert decimal number to octal

$83_{10} = 123_{8}$?

$83 \div 8 = 10$ remain 3

$10 \div 8 = 1$ remain 2

$1 \div 8 = 0$ remain 1

Converting from decimal to octal

- The digits used in octal (base 8) system are 0,1, …, 7.

0_8	1_8	2_8	3_8	4_8	5_8	6_8	7_8
000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2

To convert from decimal to octal:

- Using remainder method directly, or
- First convert from decimal to binary.
- Then group the bits into threes, starting from right hand side and pad to the left with 0's where necessary.

Converting from decimal to octal

Example

83_{10} in binary is: 1010011_2

Group into threes starting from the right hand side:

1   0 1 0   0 1 1

Pad to the left with 0's:

0 0 1   0 1 0   0 1 1

Translate each group of three bits to an octal digit:

1   2   3

The final answer is: $83_{10} = 1010011_2 = 123_8$

Converting from decimal to hexadecimal

- The digits used in hexadecimal (base 16) system are 0, 1, …, 9, A, B, C, D, E, F.

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

To convert from decimal to hexadecimal

- Using remainder method directly, or
- First convert from decimal to binary.
- Then group the bits into fours, starting from right hand side and pad to the left with 0’s where necessary.

Converting from decimal to hexadecimal

Example

83_{10} in binary is: 1010011_2

Group into fours starting from the right hand side:

1 0 1   0 0 1 1

Pad to the left with 0's:

0 1 0 1   0 0 1 1

Translate each group of four bits to a hex digit:

5   3

The final answer is: $83_{10} = 1010011_2 = 53_{16}$

Negative numbers

- How do we cope with negative numbers in computer?
- To keep things small and simple, let us assume that we have 8 bits to represent an integer.
- If we are only interested in non-negative numbers, we can represent the integers from 0 to 255.

$$00000000_2 \longrightarrow 0_{10}$$
$$00000001_2 \longrightarrow 1_{10}$$
$$00000010_2 \longrightarrow 2_{10}$$
$$\ldots\ldots \qquad \ldots\ldots$$
$$11111111_2 \longrightarrow 255_{10}$$

Sign and magnitude

- If, we need to consider negative numbers, it would make sense to divide the patterns evenly between the positive and negative numbers *i.e.* 128 patterns for positive numbers and 128 patterns for negative numbers.
- 7 bits can give 2^7 =128 patterns.
- We can use one bit (usually leftmost) to represent the sign: “0” for positive numbers and “1” for negative numbers. This bit is called sign bit.
- Use the rest to represent the magnitude.

Example, 8-bit:

00001101 → +13

10001101 → -13

Sign and magnitude

The range of numbers in sign and magnitude for an *n* bit word is:

$$-(2^{n-1} - 1) \quad \text{to} \quad +(2^{n-1} - 1)$$

For example, for an 8 bit word:

01111111 → +127

11111111 → -127

Sign and magnitude

Unfortunately, there is a problem with this idea, as we need to represent zero, which means we cannot evenly distribute the remaining 255 patterns. We could choose to have two patterns for zero *i.e.* -0 and +0 as follows:

00000000 → +0

10000000 → -0

Two’s complement

- An alternative scheme to sign and magnitude for representing negative numbers is two’s complement.
- For an 8 bit word, the positive integers from 0 to 127 are represented in the same way as in sign and magnitude.

00000000	→	0
00000001	→	1
00000010	→	2
…		
01111111	→	127

Two's complement

- The negative numbers:

10000000	→	-128
10000001	→	-127
10000010	→	-126
…		
11111110	→	-2
11111111	→	-1

Increase toward zero

- Note that in the same way as for sign and magnitude, the left most digit is the sign bit: it is 0 for positive numbers and 1 for negative numbers. This makes it easy to determine whether a number is positive or negative.

Two's complement

- A simple way to interpret two's complement is to view the place value at the leftmost bit as a power of two with negative magnitude, and at the other bits are positive powers of two.
- For an 8-bit representation:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1

$$00100011_2=(1\times 32)+(1\times 2)+(1\times 1)=35_{10}$$

$$10100011_2=(1\times -128)+(1\times 32)+(1\times 2)+(1\times 1)$$
$$= -128_{10}+35_{10}= -93_{10}$$

Two's complement

- An important advantage of two's complement representation is that it ensures that the addition of a number with its negative under binary addition yields zero.

  00000010 → 2

  11111110 → -2

  $1\boxed{00000000}$

- Where the leftmost 1 is carried out or discarded because we only have 8 bits, leaving us with the two's complement representation of zero as expected.

Two's complement

- Convert decimal numbers to two's complement
- Have to consider how many bits we are given, and fill it up.
- If the number is positive:

1) Ordinary binary conversion

2) Filling up the left bits with zeros

- To store the integer +14 in 8 bit register

14 → 1110 → 00001110

Two's complement

- If the number is negative:

1) Carry out a standard binary conversion of the magnitude;
2) Fill up the spaces with zeros;
3) Invert all the bits (0 becomes 1, and 1 becomes 0).
4) Finally, add 1.

- To store the integer -12 in 8 bit register

12 → 1100

1100 → 00001100

00001100 → 11110011

11110011 + 1 = 11110100

Double check: $(1\times -2^7)+(1\times 2^6)+(1\times 2^5)+(1\times 2^4)+(1\times 2^2)$
$= -128+64+32+16+4 = -12$

Complementary addition

- Using an 8-bit register to store

5_{10}		00000101
-9_{10}	+	11110111
$5_{10} - 9_{10}$		11111100

- Double check

$$(1\times-2^7)+(1\times2^6)+(1\times2^5)+(1\times2^4)+(1\times2^3))+(1\times2^2)$$
$$= -128+64+32+16+8+4 = -128+124 = -4$$

Fixed-point numbers

- A simple and easy way to express fractional numbers, using a fixed number of digits, with a fixed position of the point.
- Examples:
  - Decimal system: 0.1_{10}, 4.6_{10}, 8.9_{10}, …
  - Binary system: 0.11_2, 1.10_2, 0.01_2, …
- Integers are also fixed-point numbers.
  - The position of the decimal point: 8., 74., 163.
  - There are also implied 0’s to the left: 008., 074., 163.

Range and precision

- Fixed-point representation is characterised by:
  - Range: from the minimum number possible to the maximum number possible.
  - Precision: difference between two adjacent numbers.
- Decimal Example
  - Consider again the numbers: 0.1, 4.6, 8.9, …
  - Range is from 0.0 to 9.9, denoted: [0.0, 9.9]
  - Precision is 0.1

Trade-off between range and precision

Using our decimal system example:

- With the decimal point at the far right, the range is [00, 99] and the precision is 1
- With the decimal point at the far left, the range is [.00, .99] and the precision is .01
- Either way, there are only 10^2 different decimal numbers from 00 to 99, or from .00 to .99
- Thus it is only possible to represent 100 different items, however we apportion range and precision.