

CS35L

Week 6 Lecture 2

Rishab Ketan Doshi

Overview

- Number systems
 - Binary
 - Octal
 - Decimal
 - Hexadecimal
- Representing -ve numbers on the computer
 - Signed bit
 - 1s complement
 - 2s complement
- Endianness
- Pthread example code
- Embarrassingly parallel examples
- Homework explanation + hints

Number Systems

- We are all familiar with decimal numbers.

Ex: $735 = 700 + 30 + 5 = 7 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$

In decimal system, the base is 10.

Base is the number(raised to the power) you multiply with for every position in the number.

Base is the number of digits in that number system.

Binary Number System

Binary - Base is 2. Only two digits allowed 0 or 1.

Ex: 101 is a valid binary (base 2) number

Converting binary to decimal:

$$101 \text{ (base 2)} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5 \text{ (base 10)}$$

Converting Decimal to Binary

Successive Division by 2

$2 \overline{) 29}$	
$2 \overline{) 14}$	1
$2 \overline{) 7}$	0
$2 \overline{) 3}$	1
$2 \overline{) 1}$	1
0	1

Remainders

LSB

MSB

Read the remainders
from the bottom up

29 decimal = 11101 binary

Octal Number System

Octal - Base is 8. Only eight digits allowed 0 – 7 (inclusive).

Ex: 731 is a valid octal (base 8) number

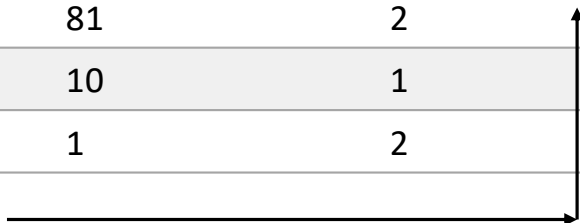
Converting octal to decimal:

$$731 \text{ (base 8)} = 7 \times 8^2 + 3 \times 8^1 + 1 \times 8^0 = 473 \text{ (base 10)}$$

Converting decimal to octal?

- Same remainder technique.

	Number	Remainder
8	650	
8	81	2
8	10	1
8	1	2



650(base 10) = 1212(base 8)

Hexadecimal Number System

Octal - Base is 16. Sixteen digits allowed

- 0 – 9
- A – F (A – 10, B – 11, C-12, D-13, E-14, F-15)

Ex: A3E is a valid hexadecimal (base 16) number

Converting hexadecimal to decimal:

$$A3E \text{ (base 16)} = 10 \times 16^2 + 3 \times 16^1 + 14 \times 16^0 = 2622 \text{ (base 10)}$$

Conversion across base systems

- How to convert Base 4 to Base 3
 - Convert original number to decimal
 - Use remainder technique to convert decimal number in prev step.
- Ex: Convert 1023(base 4) to (base 3)
 - $1023(\text{base } 4) = 75(\text{base } 10)$
 - Using remainder with repeated division technique $75(\text{base } 10)$ is $2210(\text{base } 3)$
 - Summarizing: $1023(\text{base } 4) \rightarrow 75(\text{base } 10) \rightarrow 2210(\text{base } 3)$

Computer Memory and Data Representation

- n bits can represent 2^n numbers.
 - Example. $n=3$ can have the one of the 8 values:
 - 000, 001, 010, 011, 100, 101, 110, or 111
 - You could use them to represent numbers 0 to 7, numbers 8881 to 8888, characters 'A' to 'H', or up to 8 kinds of fruits like apple, orange, banana; or up to 8 kinds of animals like lion, tiger, etc
- Important to note that a computer memory location merely *stores a binary pattern*.
- It is entirely up to you, as the programmer, to decide on how these patterns are to be *interpreted*.
 - For example, the 8-bit binary pattern "0100 0001B" can be interpreted as an unsigned integer 65, or an ASCII character 'A', or some secret information known only to you. In other words, you have to first decide how to represent a piece of data in a binary pattern before the binary patterns make sense.
- The interpretation of binary pattern is called *data representation* or *encoding*

Interesting Story: Rosette Stone and the Decipherment of Egyptian Hieroglyphs

Egyptian hieroglyphs (next-to-left) were used by the ancient Egyptians since 4000BC.

Unfortunately, since 500AD, no one could longer read the ancient Egyptian hieroglyphs, until the re-discovery of the Rosette Stone in 1799 by Napoleon's troop (during Napoleon's Egyptian invasion) near the town of Rashid (Rosetta) in the Nile Delta.

The Rosetta Stone (left) is inscribed with a decree in 196BC on behalf of King Ptolemy V. The decree appears in *three* scripts: the upper text is *Ancient Egyptian hieroglyphs*, the middle portion Demotic script, and the lowest *Ancient Greek*. Because it presents essentially the same text in all three scripts, and Ancient Greek could still be understood, it provided the key to the decipherment of the Egyptian hieroglyphs.

The moral of the story is unless you know the encoding scheme, there is no way that you can decode the data.



Integer Representation

- Computers use *a fixed number of bits* to represent an integer. The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit or 64-bit. Besides bit-lengths, there are two representation schemes for integers:
- *Unsigned Integers*: can represent zero and positive integers.
- *Signed Integers*: can represent zero, positive and negative integers. Three representation schemes had been proposed for signed integers:
 - Sign-Magnitude representation
 - 1's Complement representation
 - 2's Complement representation

Unsigned Representation

- Unsigned integers can represent zero and positive integers, but not negative integers.
- **Example 1:** Suppose that $n=8$ and the binary pattern is 0100 0001B, the value of this unsigned integer is $1 \times 2^0 + 1 \times 2^6 = 65D$.
- **Example 2:** Suppose that $n=16$ and the binary pattern is 0000 0000 0000 0000B, the value of this unsigned integer is 0.
- An n -bit pattern can represent 2^n distinct integers. An n -bit unsigned integer can represent integers from 0 to $(2^n)-1$, as tabulated:

n	Minimum	Maximum
8	0	$(2^8)-1$ (=255)
16	0	$(2^{16})-1$ (=65,535)
32	0	$(2^{32})-1$ (=4,294,967,295) (9+ digits)
64	0	$(2^{64})-1$ (=18,446,744,073,709,551,615) (19+ digits)

Sign-Magnitude representation

- In sign-magnitude representation:
 - The most-significant bit (msb) is the *sign bit*, with value of 0 representing positive integer and 1 representing negative integer.
 - The remaining $n-1$ bits represents the magnitude (absolute value) of the integer. The absolute value of the integer is interpreted as "the magnitude of the $(n-1)$ -bit binary pattern".

Suppose that $n=8$ and the binary representation is 1 000 0001B.

Sign bit is 1 \Rightarrow negative

Absolute value is 000 0001B = 1D

Hence, the integer is -1D

Problems

N = 4,
all
possible
values

Sign Bit

1	1	1	1	-7
1	1	1	0	-6
1	1	0	1	-5
1	1	0	0	-4
1	0	1	1	-3
1	0	1	0	-2
1	0	0	1	-1
1	0	0	0	-0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

2 representations for same 0.

+0 and -0

Math doesn't work out directly

Ex: $5 + (-5)$

$$\begin{array}{r}
 \overset{1}{0} \overset{1}{1} 0 1 \\
 + 1 1 0 1 \\
 \hline
 1 0 0 1 0
 \end{array}$$

Carry in green
Discarded bit in red(n=4)

0010 is not equal to 1000(-0) or
0000(+0)

1s complement representation

In 1's complement representation:

- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining $n-1$ bits represents the magnitude of the integer, as follows:
 - for positive integers, the absolute value of the integer is equal to "the magnitude of the $(n-1)$ -bit binary pattern".
 - for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement (inverse)* of the $(n-1)$ -bit binary pattern" (hence called 1's complement).

Examples:

Suppose that $n=8$ and the binary representation 1 000 0001B.

Sign bit is 1 \Rightarrow negative

Absolute value is the complement of 000 0001B, i.e., 111 1110B = 126D

Hence, the integer is -126D

Suppose that $n=8$ and the binary representation 0 100 0001B.

Sign bit is 0 \Rightarrow positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Problems

Ones	Complement
1 0 0 0	-7
1 0 0 1	-6
1 0 1 0	-5
1 0 1 1	-4
1 1 0 0	-3
1 1 0 1	-2
1 1 1 0	-1
1 1 1 1	-0
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7

2 representations for 0 (+0 and -0)

Math still doesn't work out directly. Specifically, all answers are off by one

Handwritten binary addition examples showing errors in ones complement arithmetic:

- $$\begin{array}{r} 0101 \\ + 1010 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} 5 \\ + (-5) \\ \hline -0 \end{array}$$
- $$\begin{array}{r} 0011 \\ + 1100 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} 3 \\ + (-3) \\ \hline -0 \end{array}$$
- $$\begin{array}{r} 0101 \\ 1100 \\ \hline 10001 \end{array}$$

$$\begin{array}{r} 5 \\ + (-3) \\ \hline 1 \end{array}$$
- $$\begin{array}{r} 0110 \\ 1101 \\ \hline 10011 \end{array}$$

$$\begin{array}{r} 6 \\ + (-2) \\ \hline 3 \end{array}$$

2s complement

- In 2's complement representation:
- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining $n-1$ bits represents the magnitude of the integer, as follows:
 - for positive integers, the absolute value of the integer is equal to "the magnitude of the $(n-1)$ -bit binary pattern".
 - for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the $(n-1)$ -bit binary pattern *plus one*" (hence called 2's complement).
- **Example 1:** Suppose that $n=8$ and the binary representation 0 100 0001B.
Sign bit is 0 \Rightarrow positive
Absolute value is 100 0001B = 65D
Hence, the integer is +65D
- **Example 2:** Suppose that $n=8$ and the binary representation 1 000 0001B.
Sign bit is 1 \Rightarrow negative
Absolute value is the complement of 000 0001B plus 1, i.e., 111 1110B + 1B = 127D
Hence, the integer is -127D

Any problems?

Twos Complement	
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Single representation for 0
Math works out

Handwritten binary addition examples:

$$\begin{array}{r} 111 \\ 0101 \\ + 1011 \\ \hline 10000 \end{array}$$
$$\begin{array}{r} 11 \\ 0110 \\ + 1110 \\ \hline 10100 \end{array}$$

Annotations for the first example: A '5' is written above the result, and $+(-5)$ is written to the right.

Annotations for the second example: A '6' is written above the result, and $+(-2)$ is written to the right.

Sign bit can be thought of as place value of -8 .

Ex:

$$1001 = 1*(-8) + 0*(4) + 0*(2) + 1*(1) = -7$$

Endianness

- Modern computers store one byte of data in each memory address or location, i.e., byte addressable memory. An 32-bit integer is, therefore, stored in 4 memory addresses.
- The term "Endian" refers to the *order* of storing bytes in computer memory. In "Big Endian" scheme, the most significant byte is stored first in the lowest memory address (or big in first), while "Little Endian" stores the least significant bytes in the lowest memory address.
- For example, the 32-bit integer 12345678H (305419896_{10}) is stored as 12H 34H 56H 78H in big endian; and 78H 56H 34H 12H in little endian. An 16-bit integer 00H 01H is interpreted as 0001H in big endian, and 0100H as little endian.

Pthread examples

- Run examples from <https://github.com/rishabkdoshi/CS35L/blob/master/week6/learningByDoing/pthreads.md>

Ray Tracing

- An advanced computer graphics technique for rendering 3D images
- Mimics the propagation of light through objects
- Simulates the effects of a single light ray as it's reflected or absorbed by objects in the images

Computational Resources

- Ray Tracing produces a very high degree of visual realism at a high cost (yields high quality rendering)
- The algorithm is *computationally intensive*
- Good candidate for multithreading (embarrassingly parallel)
 - Threads need not synchronize with each other, because each thread works on a different pixel

Homework Steps

- You have a single threaded program, you need to implement a multi-threaded version
- Understand what makefile is doing
- Read code, starting from main function in main.c
- Don't change any functions in any files
 - This should help you narrow down places to look for changes
- This task of ray tracing is embarrassingly parallel
 - Identify which part of code has to be parallelized
 - Identify which variables need to be made global or passed to your thread function