Bits, Bytes, Bit Operations and image processing

15-123

Systems Skills in C and Unix
Representing Information

- Smallest Data unit is the "bit"
  \[ \begin{array}{c}
  0 \\
  1 
  \end{array} \]

- Smallest addressable unit is the "byte"
  \[ \begin{array}{c}
  0010 \ 1001 
  \end{array} \]

- Each computer has a "word" size
  - characteristic of computer architecture
  - Indicate the nominal size of integers and pointers
  - Most common size is 32-bits
  - How many addressable units are there then?
  \[ 2^{32} \]
Data Sizes

• Here are the typical 32-bit allocation for data types (in bytes)
  – char (1), short int (2), int (4), long int (4)
    • In compaq alpha long int is 8
  – char* (4), float (4), double (8)

• The exact size of data allocation depends on both compiler and machine
Data value ranges

• `<limits.h>` library defines the range of values any data type can assume.
• Applications that are designed to be portable must use symbolic constants.
• Some examples
  • `INT_MIN`  
    – Minimum value for a variable of type `int`.  
    – $\text{INT_MIN} = 2^{31} - 1$  
  • `INT_MAX`  
    – Maximum value for a variable of type `int`.  
    – $\text{INT_MAX} = 2^{31} - 1$  
  • `UINT_MAX`  
    – Maximum value for a variable of type `unsigned int`.  
    – $\text{UINT_MAX} = 2^{32} - 1$  
  • `LONG_MIN`  
    – Minimum value for a variable of type `long`.  
    – $\text{LONG_MIN} = 2^{31} - 1$  
  • `LONG_MAX`  
    – Maximum value for a variable of type `long`.  
    – $\text{LONG_MAX} = 2^{32} - 1$
Storage Classes

- **auto**
  - Typical variables defined inside functions

- **static**
  - Variables that retain values between function calls

- **extern**
  - Declared within a function, but specifications given elsewhere

- **register**
# Representation formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Base</th>
<th>Formula</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>2</td>
<td>$\sum_{i=0}^{n-1} a_i 2^i$</td>
<td>$a_i = 0, 1$</td>
</tr>
<tr>
<td>Octal</td>
<td>8</td>
<td>$\sum_{i=0}^{n} a_i 8^i$</td>
<td>$a_i = 0, 1, \ldots, 7$</td>
</tr>
<tr>
<td>Decimal</td>
<td>10</td>
<td>$\sum_{i=0}^{n} a_i 10^i$</td>
<td>$a_i = 0, 1, \ldots, 9$</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>16</td>
<td>$\sum_{i=0}^{n} a_i 16^i$</td>
<td>$a_i = 0, 1, \ldots, 9, A, B, C, D, E, F$</td>
</tr>
</tbody>
</table>
Addressing and byte ordering

• Little Endian
  – Least significant byte first (DEC, Intel)

• Big Endian
  – Most significant byte first (IBM, Motorola, SUN)
  – I have been using big endian??

• Application programmers may not care about this ordering
When byte ordering becomes an issue

- Communication of binary data over a network between different machines
- Code written for networking applications must then do their own conversions between machines
Integer Representations

• Typical 32-bit machine uses
  – 32-bit representation for int and unsigned
    • Range: $\left[ -2^{31}, 2^{31}-1 \right] \cup \left[ 0, 2^{32}-1 \right]$

  – Compaq alpha uses 64 bits for long int
    • Range:
Closer look at signed and unsigned integers

- Consider a n-bit integer representation of an unsigned integer

\[ n = a_0a_1\ldots a_{n-1} \]

\[ \text{num} = \sum_{i=0}^{n-1} a_i \cdot 2^i \]

- Consider a n-bit integer representation of a signed integer

\[ \text{num} = -2^{n-1} \cdot a_{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \]
Representing negative numbers using 2’s complement

- One’s complement
  \[ \sim x \]
  \[
  x = 0001 1010 \\
  \sim x = 1110 0101
  \]

- Two’s complement
  \[ 1 + \sim x \]
  \[
  -x = 1110 0110
  \]

\[ +1 \]
Signed and unsigned numbers

• By default all constant values are signed
  – int x = 20, y =0x45

• Can create an unsigned constant using
  – unsigned x = 0x123u (or U)
Adding numbers

- Adding unsigned ints
  - unsigned int $x = 0x0000FA02$
  - unsigned int $y = 0x000102FA$
  \[ \begin{array}{c}
    0001 \quad \text{FCFC}
  \end{array} \]

- Adding signed ints
  - int $x = 75$, $y = -34$
  - Binary representations
    - $75 = 01001011$
    - $34 = 00100010$
    - $34 = 1101110$
    - $1 + 1 = 11011110$
    - $34 + 1 = 11011110$
    - $75 + (-34) = 11000001$
    - $75 + (-34) = 41$
Bit Operations in C

- Bitwise AND (\&)
  - 0x75 \& 0x96
Bitwise OR ( | )

0x75 | 0x96

0111 0101 | 1001 0110

= 1111 0111
Bitwise negation ($\sim$)

\[ \sim\text{0x75} \]

\[ \begin{array}{c}
0111 \\
\sim = 1000
\end{array} \]
XOR ( ^ )

0x75 ^ 0x96

\[
\begin{align*}
A_i + B_i \\
\overline{A_i + B_i}
\end{align*}
\]

\[X \ ^\  y = 0 \Rightarrow x = y\]
Logic for adding bit by bit

- $S_i = (A_i \wedge B_i) \wedge C_{in}$
- $C_{out} = (A_i \wedge B_i) \lor ((A_i \wedge B_i) \wedge C_{in})$

<table>
<thead>
<tr>
<th>$S_i$</th>
<th>$C_{out}$</th>
<th>$A_i$</th>
<th>$B_i$</th>
<th>$C_{in}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Bit adder

• Exercise: Given two unsigned char’s write a bit-by-bit adder using formulas for sum and carry. How would you recognize an overflow situation?

• unsigned char (unsigned char A, unsigned char B){
Logical Operations in C are different

- Logical AND (&&)
  - 0x75 && 0x96

- Logical OR (||)

- Logical Not (!)
Shifting in C

• Left Shift ( \(<<\) )
  – Right most bits are filled by 0’s

• Right shift ( \(>>\) )
  – Left most bits are filled by most significant bit
Counting number of 1’s

• Let $C(n)$ be the number of 1’s in $n \geq 0$
• What is the relation between $C(n)$ and $C(n/2)$?
  – When $n$ is even
  – When $n$ is odd

\[
C(n) = C(n/2)
\]

\[
C(n) = 1 + C(n/2)
\]
Exercises

• Add two unsigned ints using only bit operations (shifts and bitwise adding)

• Given an IP address: 64.65.221.109, extract the 2\textsuperscript{nd} byte (ie 65)

• Print any given number in octal using bit operations.
getbit function

#define MASK(j) (1<<j)

int getBit(char w, unsigned j){
    return (( w & MASK(j)) == 0) ? 0 : 1;
}

• What is an alternative way to write this?

   \((w \gg j) \& 1\)
printBinary

• Complete the function printBinary

void printBinary(unsigned char w){
    // Some code here
}

}
```c
#define MASK(j) (1<<j)

int setBit(char w, unsigned j, short value){
    if (value == 0) return (w & ~MASK(j));
    else if (value == 1) return w | MASK(j);
    else return w;
}
```
Masking

- Masking is a technique to extract bits from a value
  - Determine if the number is even or odd
    \[ n \& \text{mask} \rightarrow 1 \text{ odd} \quad 0 \text{ even} \]
  - Determine if the number is divisible by 8
    \[ n \& 7 = 0 \quad 11110000 \]
  - IP addresses – 192.168.0.20
Masking

- Compiler symbol tables
  - Maintain information in most compact ways
  - #define KEYWORD 01
  - #define EXTERNAL 02
  - #define STATIC 04
  - Suppose flags is a byte that maintain this information
    - What does `flags |= EXTERNAL | STATIC` mean?
    - What does `flags &= ~(EXTERNAL | STATIC)` mean?
Image manipulations
Bitmap format

- Developed by Microsoft
- Each pixel is represented by RGB
  - 3 bytes per pixel
- Each byte value vary from 0-255
  - 0- darker, 255-lighter
- Each bmp file has a header
  - 54 bytes
typedef struct {
    unsigned short int type;  /* BMP type identifier */
    unsigned int size;       /* size of the file in bytes*/
    unsigned short int reserved1, reserved2;
    unsigned int offset;     /* starting address of the byte */
} HEADER;
Binary Files

- Any file is an array of bytes
- Files can be ASCII or binary
- Binary files are read byte by byte

NAME
fread, fwrite  - binary stream input/output

SYNOPSIS
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

DESCRIPTION
The function fread reads nmemb elements of data, each size bytes long, from the stream pointed to by stream, storing them at the location given by ptr.

The function fwrite writes nmemb elements of data, each size bytes long, to the stream pointed to by stream, obtaining them from the location given by ptr.
Exercises

• Read a BMP image and find its file size

```c
FILE* fp = fopen("x.bmp", "r");
char buf[54];
fwrite(buf, 54, 1, fp);
int size = *((int*) (buf+2));
```
The next 40 bytes are reserved for a structure as follows.

typedef struct {
    unsigned int size; /* Header size in bytes */
    int width, height; /* Width and height in pixels */
    unsigned short int planes; /* Number of color planes */
    unsigned short int bits; /* Bits per pixel */
    unsigned int compression; /* Compression type */
    unsigned int imagesize; /* Image size in bytes */
    int xresolution, yresolution; /* Pixels per meter */
    unsigned int ncolors; /* Number of colors */
    unsigned int importantcolors; /* Important colors */
} INFOHEADER;
Exercises

• Read a BMP image and find its length and width
How bmp images can be manipulated?
Application

Dealing with Byte alignments

361x315 image
File size = 342228
3*m*n+54 is 341199
the difference is 1083
Exercises

• Remove red color altogether from an image

• Make a color RGB image BW
  – hard
Bit packing

struct {
    unsigned leading : 3;
    unsigned flag1 : 1;
    unsigned flag2 : 1;
    trailing : 11;
} flags;

• fields within the struct are not variables
  – cannot be used with the address operator (&)

• printf("The leading field is %d \n", flags.leading);
Unix File Permissions

- what does > chmod 117 file means?
Code Examples