Bitwise Operations

J.W. Choi

2024

Number System
(number conversion)

Number Systems

- Decimal (base 10)
  - 0,1,2,3,4,5,6,7,8,9
- Binary (base 2)
  - 0,1
- Octal (base 8)
  - 0,1,2,3,4,5,6,7
- Hexadecimal (base 16)
  - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Number Representations

- Octal requires 3 bits
  - 00, 01, 02, 03, 04, 05, 06, 07
  - (Binary) 000, 001, 010, 011, 100, 101, 110, 111
  - 463_8 -> 100 110 011
- Hexadecimal requires 4 bits
  - 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8,
    0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF
  - (binary) 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
  - $3FC_{16}$ -> 0011 1111 1011

Example

- Decimal
  - 5346 = $(5 * 10^3) + (3 * 10^2) + (4 * 10^1) + (6 * 10^0)$
- Binary
  - 100101 = $(1 * 2^5) + (0 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 32 + 4 + 1 = 37$
- Octal
  - 0463 = $(4 * 8^2) + (6 * 8^1) + (3 * 8^0) = 256 + 48 + 3 = 307$
- Hexadecimal
  - 0x3FC = $(3 * 16^2) + (F * 16^1) + (C * 16^0) = (3 * 16^2) + (15 * 16^1) + (12 * 16^0) = 768 + 240 + 12 = 1020$

Notations in C

- Octal
  - 0 prefix (0463)
- Hexadecimal
  - 0x prefix (0x3FC)

- Printing (printf)
  - octal (%o)
  - hexadecimal (%0x)

Storing Numbers

- In binary form, regardless of the base used

- int num = 5346;
  - /* decimal 5346 */
  - 0001 0101 0010 0010
- int num = 0x3FC;
  - /* hexadecimal 0x3FC = decimal 1020 */
  - 0000 0100 0000 0100

Converting a Decimal Number to a Binary Number: Example

156_{10}

512 256 128 64 32 16 8 4 2 1

Decimal to Binary

156_{10}

512 256 128 64 32 16 8 4 2 1

1

156 − 128 = 28

Decimal to Binary

156_{10}

512	256	128	64	32	16	8	4	2	1
		1	0	0	1				

28 - 16 = 12

Decimal to Binary

156_{10}

512	256	128	64	32	16	8	4	2	1
		1	0	0	1	1			

$12 - 8 = 4$

Decimal to Binary

156_{10}

512	256	128	64	32	16	8	4	2	1
		1	0	0	1	1	1		

$4 - 4 = 0$

Decimal to Binary

156_{10}

512	256	128	64	32	16	8	4	2	1
		1	0	0	1	1	1	0	0

$156_{10} = 10011100$

Printing Numbers

```
char str[20] = "hello, world";
char ch='c';
int dec=15, hex=0x12AF3, oct=0172;
float db=3.14;

printf("%s ₩n", str);
printf("%c ₩n", ch);
printf("%d %x %o₩n", dec, dec, dec);
printf("%#X ₩n", hex);
printf("%#o ₩n", oct);
printf("%f %g ₩n", db, db);
```

Bitwise Operations

Bitwise Operation

- Data in C is stored as a sequence of bits.
  - char (1 byte)
  - int (4 bytes)
  - float (8 bytes)
- Operator applies to every bit of the bytes separately
- Uses
  - saves memory space by encoding separate values (char or int) into a single value
  - Applications that require bit-level data processing (e.g. graphics, games)

Examples

- 0 00110000
- 1 00110001
- ...
- 9 00111001
- ...
- a 01100001
- b 01100010
- c 01100011
- ..
- z 01111010
- A 01000001
- B 01000010
- ...
- Z 01011010

Bitwise Operations

&	bit by bit AND	(&& for Boolean AND)
|	bit by bit OR	(|| for Boolean OR)
^	bit by bit XOR	
~	bit by bit one's complement (switches 0 and 1)	
<< n	left shift n bits (vacated bit positions are filled with 0)	
>> n	right shift n bits (vacated bit positions are filled with sign bit)	

0 AND 0 = 0	
0 AND 1 = 0	x AND 0 = 0
1 AND 0 = 0	x AND 1 = x
1 AND 1 = 1	

0 OR 0 = 0	
0 OR 1 = 1	x OR 0 = x
1 OR 0 = 1	x OR 1 = 1
1 OR 1 = 1	

0 XOR 0 = 0	
0 XOR 1 = 1	x XOR 0 = x
1 XOR 0 = 1	x XOR 1 = ~x
1 XOR 1 = 0	

Examples (1/2)

- AND
- ```
    0110 1011 1000 0101
  & 0001 1111 1011 1001
  ---------------------------
    0000 1011 1000 0001
  ```
- OR
- ```
    0110 1011 1000 0101
  | 0001 1111 1011 1001
  ---------------------------
    0111 1111 1011 1101
  ```
- XOR
- ```
    0110 1011 1000 0101
  ^ 0001 1111 1011 1001
  ---------------------------
    0111 0100 0011 1100
  ```
  ```

Examples (2/2)

- Bitwise Shift
  - x = 0110 1111 1001 0001
  - x = x >> 4;
  - x = 0000 0110 1111 1001

Exercises

```
  10110011
& 11010101
```

```
  10110011
| 11010101
```

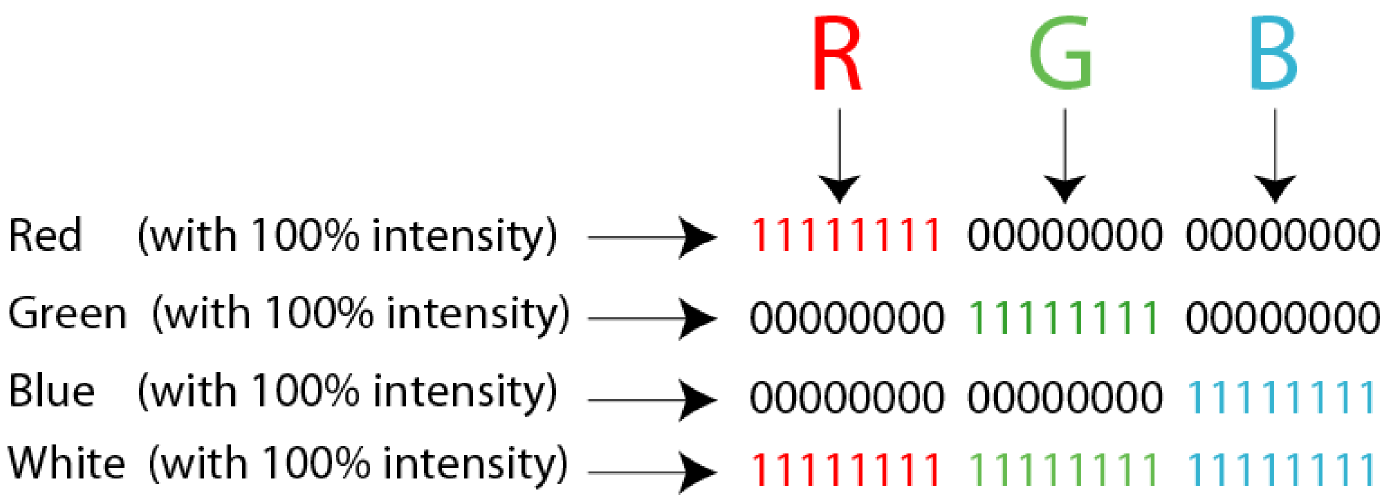
```
  10110011
^ 11010101
```

```
~ 10110011
```

```
10110011 << 4
```

Representation of Color Pixels

- Each colored pixel is decomposed into red, green, blue.
- Intensity of each color is measured and a bit pattern (usually 8-bit) is assigned to it.



Example

- 32-bit color in graphics
  - alpha, red, green, blue (each takes 8 bits)
  - AAAA AAAA RRRR RRRR GGGG GGGG BBBB BBBB

- Problem: Extract the value for red

Solution (1/2)

- Define a mask (0x00FF0000)
  - 0000 0000 1111 1111 0000 0000 0000 0000
- ** how to define a mask
  - set 1 where the bit value of the target word is to be kept.
  - set 0 where the bit value of the target word is to be removed.

Example (2/2)

- Apply the mask to the color value (AND)
  -   AAAA AAAA RRRR RRRR GGGG GGGG BBBB BBBB
    & 0000 0000 1111 1111 0000 0000 0000 0000
    ----
    result: 0000 0000 RRRR RRRR 0000 0000 0000 0000
- Do bitwise right shift of result by 16 positions (>> 16)
  - result: 0000 0000 RRRR RRRR 0000 0000 0000 0000
  - shift >> 16
  - new result:
  - 0000 0000 0000 0000 0000 0000 RRRR RRRR
  - == RRRR RRRR

Solution In C

- #define mask 0x00FF0000
- (color_word & mask) >> 16
- (color_word & 0x00FF0000) >> 16

Exercise

- From a 16-bit color word, extract the value for red.
- Color word: RRRR RGGG GGGB BBBB
  Red mask: 1111 1000 0000 0000 == 0xF800
  Green mask: 0000 0111 1110 0000 == 0x07E0
  Blue mask: 0000 0000 0001 1111 == 0x001F

Example

- 32-bit color in graphics
  - alpha, red, green, blue (each takes 8 bits)
  - AAAA AAAA RRRR RRRR GGGG GGGG BBBB BBBB

- Problem: Change the value for green

Solution

- Clear the value for green
  - AAAA AAAA RRRR RRRR 0000 0000 BBBB BBBB
- Define a mask (with a new value for green)
  - 0000 0000 0000 0000 GGGG GGGG 0000 0000
- Apply OR
- color word: AAAA AAAA RRRR RRRR 0000 0000 BBBB BBBB
  mask: | 0000 0000 0000 0000 GGGG GGGG 0000 0000
  ----
  result: AAAA AAAA RRRR RRRR GGGG GGGG BBBB BBBB

Bitmasks

```
unsigned int flags; // contains a set of flags

#define DIRTY 0x01
#define OPEN 0x02
#define VERBOSE 0x04
#define RED 0x08
#define SEASICK 0x10

// Testing, setting and clearing a flag
if (flags & DIRTY)      /* code for dirty case */
if (!(flags & OPEN))  /* code for closed case */
if (flags & DIRTY) means ``if the DIRTY bit is on''.
flags = flags | DIRTY;   /* set DIRTY bit */
flags = flags & ~DIRTY;   /* clear DIRTY bit */
```

Lab

- Programming Exercises #1
  - Write a C program that reads a character, and displays each bit of the character.
  - Note: A character has 8 bits.
  - *Hint: Use a mask 0x80, and left shift the mask (or right shift the character read) 1 bit at a time.*

Problem Statement

- Write a C program that displays the first 8 bits of each character value input into a variable named *ch*.
- (Hint: Assuming each character is stored using 8 bits, start by using the hexadecimal mask 80, which corresponds to the binary number 10000000.
- If the result of the masking operation is a 0, display a 0; else display a 1.
- Then shift the mask one place to the right to examine the next bit, and so on until all bits in the variable *ch* have been processed.)
- * Test your program against 4 characters (2 capital and 2 lower case)
- * Check your result against the “alphabet to binary” tables in the next 2 pages.

Alphabet (Capital) in Binary

Alphabet in Binary (CAPITAL letters)

A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010

Alphabet (Lower Case) in Binary

Alphabet in Binary (lowercase letters)	
a	01100001
b	01100010
c	01100011
d	01100100
e	01100101
f	01100110
g	01100111
h	01101000
i	01101001
j	01101010
k	01101011
l	01101100
m	01101101
n	01101110
o	01101111
p	01110000
q	01110001
r	01110010
s	01110011
t	01110100
u	01110101
v	01110110
w	01110111
x	01111000
y	01111001
z	01111010

Lab

- Programming Exercises #2
  - Write a C program to convert decimal number from 1 to 1000 to binary string and hexadecimal string
  - Output:

DEC 1:	BIN 1	HEX 1
DEC 2:	BIN 10	HEX 2
.		
.		
DEC 254:	BIN 11111110	HEX FE
.		
.		
DEC 1000:	BIN 1111101000	HEX 3E8

End of Lecture