1

Lab

Data Lab: Tính toán Bits

Data Lab: Manipulating Bits

Thực hành Lập trình Hệ thống

Lưu hành nội bộ

A. TỔNG QUAN

A.1 Mục tiêu

- Tìm hiểu ứng dụng của các phép toán trên các bit nhị phân (**Bitwise** Operation):
  - **NOT** (~),
  - **AND** (&),
  - **OR** (|),
  - **XOR** (^),
  - **Left Shift** (<<), **Right Shift** (>>)
- Nắm vững các kiến thức về lập trình tương tác bit nhị phân trên C
- Thực hành giải các bài toán về bit nhị phân

A.2 Môi trường

- IDE hỗ trợ ngôn ngữ **C/C++.** Ví dụ: Microsoft Visual Studio

A.3 Liên quan

- Sinh viên cần vận dụng kiến thức trong Chương 2 (Lý thuyết).
- Các kiến thức này đã được giới thiệu trong nội dung lý thuyết đã học do đó sẽ không được trình bày lại trong nội dung thực hành này.
- Tham khảo tài liệu (Mục E). Tham khảo thêm sách **Computer System: A Programmer's Perspective (CSAPP)**

B. THỰC HÀNH

B.1 Yêu cầu chung

- **Các bài thực hành dưới đây yêu cầu viết chương trình C/C++**
  Khuyến nghị: Tất cả các hàm đặt chung trong 1 Project duy nhất.
- Trong quá trình giải các câu đố, sinh viên **chỉ được sử dụng** các toán tử gồm:
  - Các toán tử trên bit gồm: **~, &, ^, |, <<, >>**
  - Một số toán tử khác được phép: **!, + và = (gán)**
  - **Không được** dùng toán tử khác với các toán tử trên như **&&, ||, -, ==, !=, *,/,% ….**
  - **Không được** dùng các phép lập trình cấu trúc (control structures) như **loop,** if/else và if inline (toán tử **?** và **:**)**, switch,…**
- Các yêu cầu được chia làm 2 phần chính với các yêu cầu khác nhau ở kết quả đầu ra.
- Điểm tối đa của bài lab: **20**.
- Mỗi câu đố trong các mục 1 và 2 bao gồm 4 nội dung chính:
  - **Tên hàm**: Tên của hàm và kiểu giá trị trả về chứa nội dung lời giải.
  - **Yêu cầu thực thi**: Yêu cầu cần đạt được khi giải câu đố, trong đó bao gồm giải thích ý nghĩa của hàm và các phép thử mẫu dùng cho việc đánh giá điểm số (với PT là viết tắt của *Phép Thử*).
  - **Điểm đánh giá**: Mức điểm đánh giá cho mỗi câu đố, mức điểm cao hơn cho biết câu đố đó khó hơn.
  - **Max Ops**: Số lượng toán tử ***tối đa*** được phép sử dụng trong quá trình lập trình giải câu đố. Nếu có kết quả đúng nhưng số toán tử sử dụng vượt quá số lượng này, điểm đánh giá sẽ bị chia đôi.

B.2 Bài tập

B.2.1 Tính toán bit đơn giản

Các hàm này có đầu vào là các số nguyên 32 bit có dấu và đầu ra cũng là các số nguyên **32 bit** được tính toán bằng các toán tử trên bit.

- Số câu đố: **5**
- Điểm tối đa: **10**

Viết các hàm với các yêu cầu như sau:

STT	Tên hàm	Yêu cầu thực thi	Điểm	Max Ops
1.1	int **bitOr**(x,y)	Thực hiện | bit 2 số nguyên: **x | y** mà **không dùng** toán tử **|**, chỉ dùng toán tử **& và ~** • Input: 2 số nguyên **x, y** tùy ý • PT: **bitOr**(3, -9) == (3 | -9)	1	8
1.2	int **negative**(x)	Tính giá trị của **–x** không dùng dấu – • Input: Số nguyên x • PT: **negative**(0)==0 && **negative**(9)==-9 && **negative**(-5)==5	1	8

1.3	int **flipByte**(x,n)	Lật 1 Byte thứ n của số nguyên **x** *(các byte được đánh thứ tự từ 0 đến 3 từ phải sang trái)*, từ bit 0 thành 1 và ngược lại.
		• Input:
		  • Số **x** nguyên
		  • Số **n** ($0 \leq n \leq 3$)
		• PT: **flipByte** (10,0)==245 && **flipByte**(0,1)== 65280 && **flipByte** (0x5501,1)==0xaa01
			2	10

1.4	int **mod2n**(x,n)	Tính kết quả phép chia lấy dư x % 2^n
		• Input:
		  • Số **x** nguyên dương
		  • Số **n** ($0 \leq n \leq 31$)
		• PT: **mod2n**(2, 1) == 0 && **mod2n**(30, 2) == 2 && **mod2n**(63, 6) == 63
			2	20

1.5	unsigned int **divpw2**(x,n)	Tính kết quả x/2^n
		• Input:
		  • Số **x** nguyên dương
		  • Số **n *âm*** ($-31 \leq n \leq -1$)
		• PT: **divpw2**(10, -1) == 20 && **divpw2**(15, -2) == 60 && **divpw2**(2, -4) == 32
		Nâng cao:
		Tính kết quả x/2^n với mọi giá trị n ($-31 \leq n \leq 31$)
			3/**4**	8/**25**

B.2.2 Các phép kiểm tra dựa trên tính toán bit

Các hàm này có đầu vào là các số nguyên 32 bit có dấu và đầu ra là các giá trị đại diện cho **True/False**, tức là **0 hoặc 1** của phép đánh giá nào đó.

- Số câu đố: **4**
- Điểm tối đa: **10**

Viết các hàm với các yêu cầu như sau:

STT	Tên hàm	Yêu cầu thực thi	Điểm	Max Ops
2.1	int **isSameSign**(x, y)	Kiểm tra x và y có cùng dấu (cùng âm hoặc cùng không âm) không. Trả về 1 nếu x, y cùng dấu, 0 nếu ngược lại. • Input: Số nguyên x, y • PT: **isSameSign**(4,10)==1 && **isSameSign**(-5,2)==0 && **isSameSign**(-5,-9)==1	2	8
2.2	int **is16x**(x)	Kiểm tra 1 số nguyên x có chia hết cho 16 hay không? Trả về 1 nếu chia hết ngược lại trả về 0. • Input: Số nguyên x không âm • PT: **is16x**(16)==1 && **is16x**(23)==0 && **is16x**(0)==1	2	10
2.3	int **isPositive**(x)	Trả về 1 nếu x dương (x > 0) • Input: Số nguyên x • PT: **isPositive**(10)==1 && **isPositive**(-5)==0 && **isPositive**(0)==0	3	15

2.4	int **isLess2n**(x,n)	Trả về 1 nếu $x < 2^n$
		• Input:
		  • Số **x** nguyên dương
		  • Số **n** ($0 \leq n \leq 30$)
		• PT: **isLess2n**(12,4) ==1 && **isLess2n**(8,3)==0 && **isLess2n**(15,2)==0
			3	20

C. THAM KHẢO

C.1 Bảng chân trị

Dưới đây là bảng chân trị tham khảo cho 2 giá trị X và Y với các phép toán Bitwise

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Trong bài lab này, chỉ có toán tử **!** trả về dưới dạng 2 giá trị true (1) hoặc false (0), có thể sử dụng trong các câu đố có trả về 1 (True) hoặc 0 (False).

C.2 Hàm hỗ trợ: PrintBits & PrintBitsOfByte

Dưới đây là các hàm trung gian có thể giúp chúng ta quan sát rõ hơn giá trị của 1 số biểu diễn dưới dạng nhị phân:

- ***Hàm PrintBits:*** *biểu diễn 1 số dưới dạng nhị phân dưới dạng đầy đủ 32 bit*

```
void PrintBits(unsigned int x) {
	int i;
	for (i = 8 * sizeof(x)-1; i >= 0; i--) {
		(x & (1 << i)) ? putchar('1') : putchar('0');
	}
	printf("\n");
}
```

Ví dụ: **PrintBits**(16) == 0000 0000 0000 0000 0000 0000 0001 0000

- ***Hàm PrintBitsOfByte:*** *biểu diễn 1 số dưới dạng nhị phân có 8 ký tự (chỉ lấy 8 bit thấp)*

```
void PrintBitsOfByte(unsigned int x) {
	int i;
	for (i = 7; i >= 0; i--) {
		(x & (1 << i)) ? putchar('1') : putchar('0');
	}
	printf("\n");
}
```

Ví dụ: **PrintBitsOfByte**(16) == 0001 0000

C.3 Một số phép tính hữu ích

*Lưu ý: các phép tính dưới đây áp dụng cho số biểu diễn bằng **32 bit**.*

	Phương pháp	Phép tính	Ví dụ
Xét dấu của một số x	Xét giá trị bit có trọng số cao nhất	***x >> 31*** hoặc ***(x >> 31) & 1***	- (x >> 31) = 0 hoặc (x >> 31) & 1 = 0 thì x ≥ 0 - (x >> 31) = 0xFFFFFFFF hoặc (x >> 31) & 1 = 1 thì x < 0
Giữ lại một số bit nhất định, còn lại bỏ qua của số x	- Các bit cần lấy được **&** với bit 1 để giữ nguyên giá trị của nó - Các bit bị bỏ sẽ được **&** với bit 0	- Tạo một "mask" có bit 1 và 0 tương ứng với vị trí bit cần lấy và bỏ. - Sử dụng toán tử **&** số x và "mask"	Lấy 4 bit thấp nhất của x. - Mask: 0000 … 0000 0000 1111 = 0x0000000F - x **&** mask = kết quả

C.4 Gợi ý Hàm (1.1): integer bitOr(x,y)

- **Yêu cầu**: Thực thi **x | y** chỉ dùng & và **~**. Maximum Operators là 8 (tối đa 8 toán tử).
- **Gợi ý giải:**

Sử dụng định luật De Morgan về mối quan hệ giữa các phép and, or và not:

$$\overline{A.B} = \overline{A} + \overline{B}$$
$$\overline{A+B} = \overline{A}.\overline{B}$$

Với các ký hiệu . tương ứng với &, + tương ứng với | và $\bar{a}$ tương ứng với ~a, ***vậy biểu thức a | b có thể tính bằng cách nào?***

Ta có thể sử dụng các hàm PrintBits() chỉ để quan sát các giá trị, các toán tử trong các hàm này không tính vào số toán tử sử dụng.

```
int bitOr(int x, int y)
{
      int result;
      // Ap dung cong thuc De Morgan
      // ...
      PrintBits(x);
      PrintBits(y);
      PrintBits(~x);
      PrintBits(~y);
      result = ?? ;
      return result;
}
```

C.5 Gợi ý Hàm (2.3): integer isPositive(x)

- **Yêu cầu**: Trả về True nếu x > 0. Maximum Operators là 8 (tối đa sử dụng 8 toán tử).
- **Giải:** với x đầu vào có thể có 3 trường hợp tương ứng với các đầu ra tương ứng:

x	Output
<0	0
= 0	0
>0	1

Ta thấy cần phân biệt các trường hợp sau:

- **x âm hay dương**: dựa vào bit có trọng số cao nhất (mục **C.3**) xét (x >> 31) hoặc (x >> 31) & 1. Ở đây để xét (x >> 31) có giá trị hay không, có thể dùng ***!(x>>31)***.
- **x có bằng 0 hay không**: dùng ***!x***.

Với 2 toán hạng cần xét là **!(x>>31)** và **!x**, sinh viên thử tìm tiếp toán hạng cần sử dụng giữa chúng để ra output mong muốn?

x	!(x>>31)	!x	Output
< 0	0	0	0
= 0	1	1	0
> 0	1	0	1

Lưu ý: vì các hàm ở phần 2 yêu cầu chỉ đưa ra giá trị 0 hoặc 1 đại diện cho False hoặc True, kết quả cuối cùng có thể dùng phép **&** với **0x1** để chỉ lấy 1 bit thấp nhất.

D. YÊU CẦU & ĐÁNH GIÁ

Sinh viên thực hiện bài thực hành theo **nhóm tối đa 3 SV**, có thể nộp bài theo 2 hình thức:

- Nộp trực tiếp trên lớp: báo cáo và demo kết quả với GVTH.
- Nộp file code tại website môn học theo thời gian quy định.

Lưu ý: Cần chú thích cách giải của mỗi câu đố trong các hàm.

Chỉ nộp 1 file .cpp chứa tất cả các hàm giải các câu đố với tên theo quy tắc:

LabX-NhomY-MSSV1-MSSV2-MSSV3.cpp

Ví dụ: *Lab1-Nhom2-22520901-22520111-22520800.cpp*

E. THAM KHẢO

[1] Basics of Bit Manipulation [Online] Available at: https://www.hackerearth.com/fr/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

[2] Randal E. Bryant, David R. O'Hallaron (2011). *Computer System: A Programmer's Perspective (CSAPP)*

HẾT