

# LẬP TRÌNH HỆ THỐNG

---

ThS. Đỗ Thị Thu Hiền  
(hiendtt@uit.edu.vn)



nc.uit.edu.vn

**TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN - ĐHQG-HCM**  
**KHOA MẠNG MÁY TÍNH & TRUYỀN THÔNG**  
FACULTY OF COMPUTER NETWORK AND COMMUNICATIONS

Tầng 8 - Tòa nhà E, trường ĐH Công nghệ Thông tin, ĐHQG-HCM  
Điện thoại: (08)3 725 1993 (122)

# Bit, Bytes và Integers



# Nội dung

---

- **Biểu diễn thông tin dưới dạng bit**
- **Tính toán bit**
- **Integers – Số nguyên**
  - Biểu diễn: không dấu (unsigned) và có dấu (signed)
  - Cộng, nhân, dịch bit
- **Biểu diễn trong bộ nhớ, con trỏ, chuỗi**



Câu hỏi có điểm cộng

# Trong máy tính: Mọi thứ đều dưới dạng bit

---

- **Mỗi bit bằng 0 hoặc 1**
- **Sử dụng các chuỗi bit, máy tính có thể:**
  - Biểu diễn các lệnh (instructions) → xác định cần làm gì
  - Biểu diễn và tính toán các số, tập, chuỗi, v.v... → xác định cần dùng dữ liệu gì

# Ví dụ, biểu diễn số trong hệ nhị phân

## ■ Biểu diễn số dưới dạng nhị phân

- Biểu diễn  $15213_{10}$  dưới dạng nhị phân?

$$15213_{10} = 11101101101101_2$$

- $1.20_{10} = 1.0011001100110011[0011]..._2$
- $1.5213 \times 10^4 = 1.1101101101101_2 \times 2^{13}$

# Ví dụ, Chương trình Hello.c trong C

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

# Các hệ biểu diễn số?

## Biểu diễn $15213_{10}$ ở các hệ biểu diễn số khác nhau?

### ■ Hệ thập phân – Decimal (Base 10)

$15213_{10}$

### ■ Hệ nhị phân – Binary (Base 2)

- Chỉ dùng 1 và 0 trong biểu diễn số
- Từ hệ 10: Chia số 15213 cho 2, lưu lại số dư của mỗi lần chia và viết theo thứ tự ngược lại.

$$15213_{10} = 11101101101101_2$$

### ■ Hệ thập lục phân – Hexadecimal (Base 16)

- Sử dụng các ký tự từ '0' – '9' và 'A' – 'F'
- Từ hệ 10: Chia số 15213 cho 16, lưu lại số dư của mỗi lần chia và viết theo thứ tự ngược lại. 10 = A, 11 = B, 12 = C, 13 = D, 14 = E, 15 = F.
- Từ hệ 2: Gom từ phải sang trái từng nhóm 4 bit và chuyển sang giá trị tương ứng ở hệ 16.

$$15213_{10} = 11\ 1011\ 0110\ 1101_2 = 3B6D_{16}$$

# Bytes

## ■ Byte = 8 bits

- Biểu diễn giá trị từ  $00000000_2$  đến  $11111111_2$
- Trong hệ 10 (decimal): giá trị từ  $0_{10}$  đến  $255_{10}$
- Trong hệ 16 (hexadecimal):  $00_{16}$  đến  $FF_{16}$ 
  - Viết số hệ 16  $FA1D37B1_{16}$  trong C như sau:
    - `0xFA1D37b1`
    - `0xfa1d37b1`

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



# Biểu diễn các kiểu dữ liệu

Đơn vị: **bytes**

Kiểu dữ liệu	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	–	–	10/16
pointer	4	8	8

# Nội dung

---

- Biểu diễn thông tin dưới dạng bit
- **Các phép tính toán bit**
- Integers – Số nguyên
  - Biểu diễn: không dấu (unsigned) và có dấu (signed)
  - Cộng, nhân, dịch bit
- Biểu diễn trong bộ nhớ, con trỏ, chuỗi

# Phép toán trên bit (Bit-wise operations)

- Thực hiện trên các bit nhị phân 0 hoặc 1
- Áp dụng các phép toán Boolean trên từng bit:

## And (&)

- $A \& B = 1$  khi cả  $A=1$  và  $B=1$

&	0	1
0	0	0
1	0	1

## Not (~)

- $\sim A = 1$  khi  $A=0$

~	
0	1
1	0

## Or (|)

- $A | B = 1$  khi hoặc  $A=1$  hoặc  $B=1$

	0	1
0	0	1
1	1	1

## Exclusive-Or (Xor) (^)

- $A \wedge B = 1$  khi  $A$  và  $B$  khác nhau, và ngược lại

^	0	1
0	0	1
1	1	0

# Phép toán trên bit với chuỗi nhiều bit?

- Các phép toán trên bit có thể thực hiện trên **chuỗi các bit**
  - Thực hiện trên từng cặp 1-bit tương ứng

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01010101
<u>          </u>	<u>          </u>	<u>          </u>	<u>          </u>
01000001	01111101	00111100	10101010

# Phép toán trên bit trong C

## ■ Các phép toán &, |, ~, ^ đều hỗ trợ trong C

- Có thể dùng với bất kỳ kiểu dữ liệu nào: long, int, short, char,...
- Khi đó, xem mỗi số hạng là chuỗi nhiều bit
- Phép toán được áp dụng trên từng bit

## ■ Ví dụ:

- $\sim 0x41 \ \& \ 0xBE$ 
  - $\sim 01000001_2 \ \& \ 10111110_2$
- $\sim 0x00 \ | \ 0xFF$ 
  - $\sim 00000000_2 \ | \ 11111111_2$
- $0x69 \ \& \ 0x55 \ ^ \ 0x41$ 
  - $01101001_2 \ \& \ 01010101_2 \ ^ \ 01000001_2$
- $0x69 \ | \ 0x55 \ \& \ 0x7D$ 
  - $01101001_2 \ | \ 01010101_2 \ \& \ 01111101_2$

# Các phép toán dịch bit (shift)

## ■ Dịch trái: $x \ll n$

- Dịch chuỗi bit biểu diễn  $x$  sang trái  $n$  lần
  - Các bit bên trái bị bỏ đi dần
  - Điền vào bên phải các bit 0

Argument $x$	01100010
$\ll 3$	00010000
Log. $\gg 2$	00011000
Arith. $\gg 2$	00011000

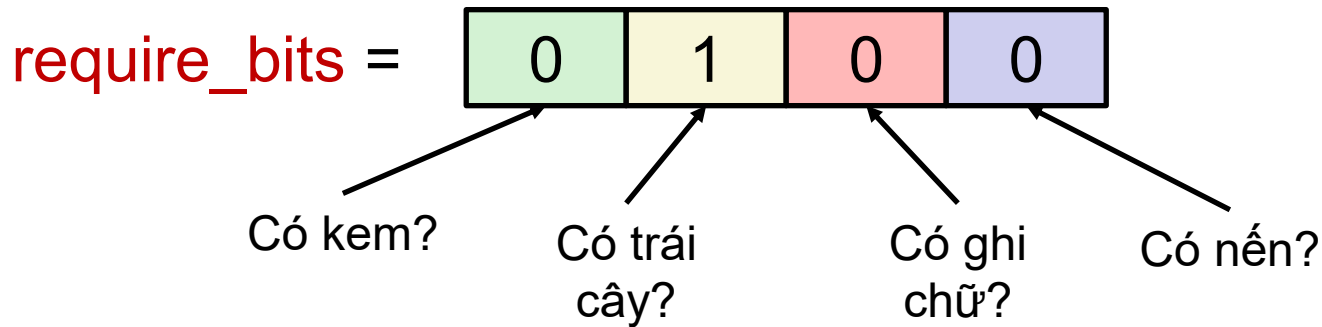
## ■ Dịch phải: $x \gg n$

- Dịch chuỗi bit biểu diễn  $x$  sang phải  $n$  lần
  - Các bit bên phải bị bỏ đi dần
- Dịch phải luận lý
  - Không quan tâm đến dấu của số  $x$
  - Điền vào bên trái các bit 0
- Dịch phải toán học
  - Quan tâm đến dấu của số  $x$
  - Điền vào bên trái bit dấu

Argument $x$	10100010
$\ll 3$	00010000
Log. $\gg 2$	00101000
Arith. $\gg 2$	11101000

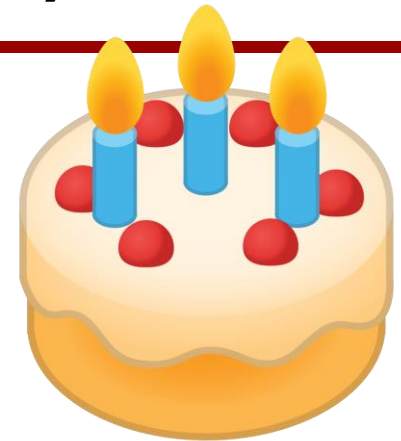
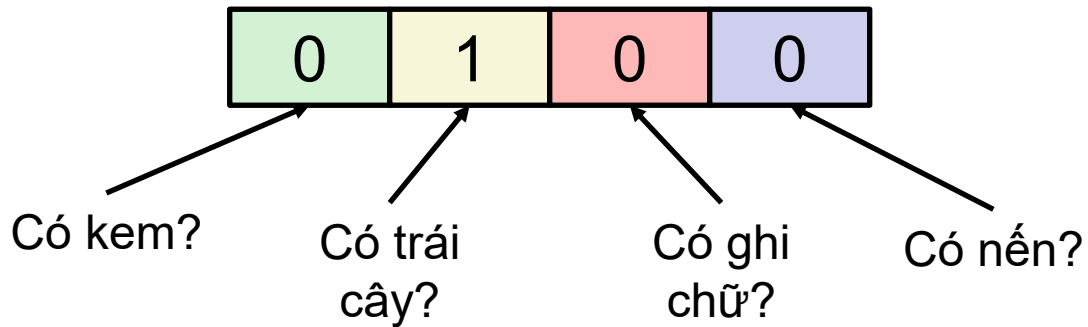
# Phép toán trên bit: Ứng dụng (1)

- **Case:** Dùng bộ **4 bit** đại diện cho các yêu cầu về đặc điểm của **1 cái bánh kem** được đặt trước.



- Ví dụ:
  - 1011: Bánh **có kem**, không trái cây, **có ghi chữ** và **có nến**
  - 0000: Bánh kem không 😊

# Phép toán trên bit: Ứng dụng (2)



## ■ **Case 1:** 1 bánh kem (0011) muốn **có trái cây**

- Giữ nguyên những yêu cầu còn lại → Chỉ cần **gán bit thứ 2 là 1**
- Giải pháp??

**^ (xor)**

0	1	0	0
---	---	---	---

**| (or)**

0	1	0	0
---	---	---	---

**| (or)**

0	0	1	1
---	---	---	---

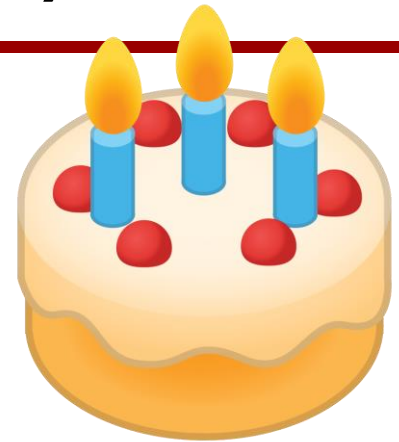
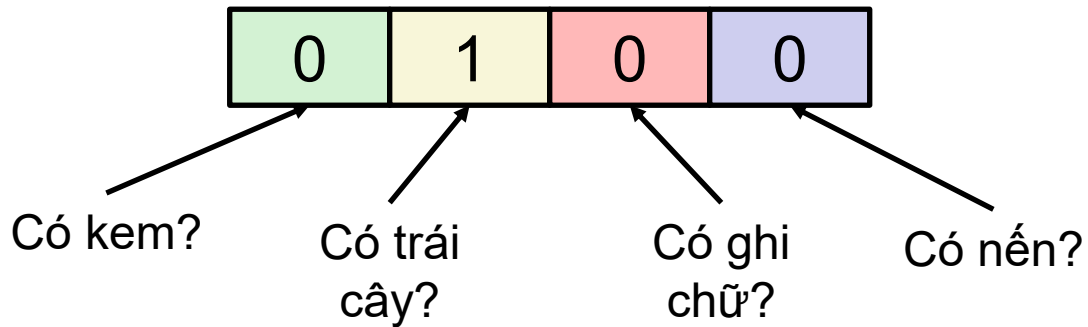
--	--	--	--

0	1	1	1
---	---	---	---

**(mask)**



# Phép toán trên bit: Ứng dụng (3)



## ■ Case 2: Đổi yêu cầu thành **không ghi chữ?** ★

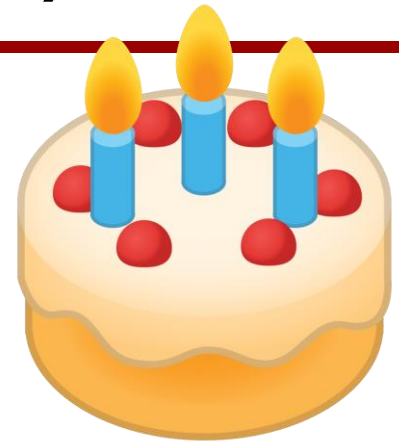
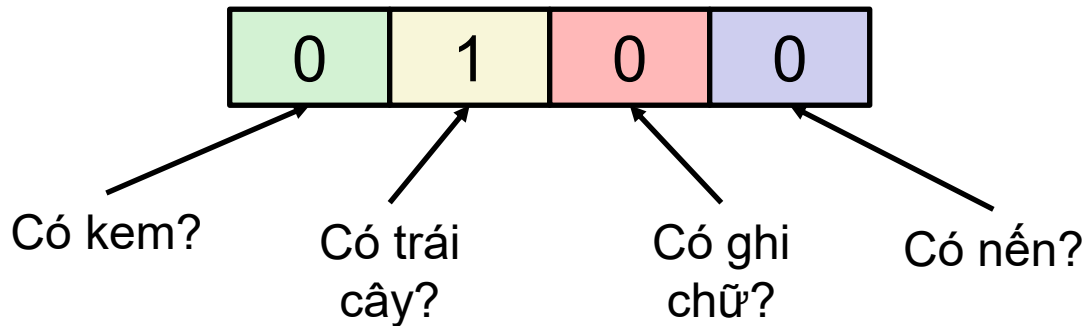
- Giữ nguyên những yêu cầu còn lại → Chỉ cần **gán bit thứ 3 là 0**
- Giải pháp??

& (and)

0	1	1	1
0	1	0	1

(mask)

# Phép toán trên bit: Ứng dụng (4)



## ■ Case 3: Chỉ lấy yêu cầu về **có nến** hay không của đơn hàng?

- Cần **lấy bit thứ 4** → giữ nguyên, các bit còn lại không lấy → đưa về 0
- Giải pháp??

& (and)

0	0	1	1
0	0	0	1
0	0	0	1

(mask)

## ■ Case 4: Kiểm tra đơn đặt hàng **có yêu cầu có ghi chữ** không??

# Phép toán trên bit: Ứng dụng (5)

---

- **Các phép dịch bit (shift):** Các phép nhân và chia với lũy thừa của 2 ( $2^n$ )

# Lưu ý: **dễ nhầm lẫn** với Phép toán logic trong C

## ■ Khác biệt của các phép toán Logic

- &&, ||, !
  - Vẫn áp dụng các phép boolean
  - **Xem 0 là False**
  - **Các giá trị khác 0 là True**
  - **Chỉ trả về 0 hoặc 1**
  - Điều kiện kết thúc sớm của if

Phép toán	Phép toán trên bit	Phép toán logic
AND	&	&&
OR		
NOT	~	!
XOR	^	

## ■ Ví dụ:

- !0x41 & 0x00
- !0x00 | 0x01
- 0x69 && 0x55 | 0x01
- p && \*p (tránh truy xuất con trỏ có giá trị null)

# Phép toán trên bit vs Phép toán logic trong C

## ■ Ví dụ so sánh

x	y	Phép toán trên bit	Phép toán logic
0x41	0x10	0x41 <b>&amp;</b> 0x10 = 0100 0001 & 0001 0000 = 0000 0000 = <b>0x0</b>	0x41 <b>&amp;&amp;</b> 0x10 = 0x1 && 0x1 = <b>0x1</b>
0x41	0x10	0x41 <b> </b> 0x10 = 0100 0001   0001 0000 = 0101 0001 = <b>0x51</b>	0x41 <b>  </b> 0x10 = 0x1    0x1 = <b>0x1</b>
0x41		<b>~0x41</b> = ~0100 0001 = 1011 1110 = 0xBE	<b>!0x41</b> = !0x1 = 0x0

# Nội dung

---

- Biểu diễn thông tin dưới dạng bit
- Tính toán bit
- **Integer – Số nguyên**
  - **Biểu diễn: không dấu (unsigned) và có dấu (signed)**
  - Cộng, nhân, dịch bit
- Biểu diễn trong bộ nhớ, con trỏ, chuỗi

# Biểu diễn số nguyên (integer)

- **Quy ước:** trong hệ biểu diễn  $w$ -bit, các bit được đánh thứ tự từ 0 đến  $w-1$  từ phải sang trái.

- **Số không dấu (unsigned)**

- Tất cả các bit đều biểu diễn giá trị
- Tính giá trị: 
$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

- **Số có dấu (signed)**

- Bit trọng số cao nhất ( $w-1$ ) biểu diễn dấu
  - 0: không âm
  - 1: âm

- Tính giá trị: 
$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$



**Bit dấu**

# Biểu diễn số nguyên – Giới hạn biểu diễn?

- **Quy ước:** trong hệ biểu diễn  $w$ -bit, các bit được đánh thứ tự từ 0 đến  $w-1$  từ phải sang trái.

- **Số không dấu (unsigned)**

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

- Giá trị lớn nhất? **Tất cả các bit là 1 =  $2^w - 1$**
- Giá trị nhỏ nhất? **Tất cả các bit là 0 = 0**

- **Số có dấu (signed)**

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

- Bit trọng số cao nhất ( $w-1$ ) biểu diễn dấu
- Giá trị lớn nhất? **Bit dấu là 0, tất cả các bit còn lại là 1 =  $2^{w-1} - 1$**
- Giá trị nhỏ nhất? **Bit dấu là 1, tất cả các bit còn lại là 0 =  $-2^{w-1}$**



# Biểu diễn số nguyên (integer): Ví dụ

- Trong hệ biểu diễn **8-bit có dấu**, đây là những **số nguyên** nào?

- **0**000 0100 = 4

- **0**001 0111 = 23

- **0**110 0001 = 97

- **1**000 1000 = -120

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

# Biểu diễn số đối (negation): Ví dụ (1)

## ■ Biểu diễn các số (hệ biểu diễn 16-bit):

- $x = 15213 = 0011\ 1011\ 0110\ 1101$
- $y = -15213 = \text{Biểu diễn bù 2 của } 15213$

**B1:** Thực hiện phép  $\sim$  trên biểu diễn nhị phân của 15213

$$\sim x = \sim 0011\ 1011\ 0110\ 1101 = 1100\ 0100\ 1001\ 0010$$

**B2:** Cộng thêm 1 vào bit thấp nhất bên phải

$$\begin{aligned}\sim x + 1 &= 1100\ 0100\ 1001\ 0010 + 1 \\ &= 1100\ 0100\ 1001\ 0011\end{aligned}$$

**1100 0100 1001 0011 chính là biểu diễn của -15213**

Với số nguyên  $x$ :  **$-x = \sim x + 1$**

# Biểu diễn số đối (negation): Ví dụ (2)

```
x = 15213:  
00111011 01101101  
y = -15213:  
11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum	15213		-15213	

# Biểu diễn số không và có dấu

X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

## ■ Tương đương

- Các số không âm có biểu diễn giống nhau trong cả trường hợp có và không có dấu

## ■ Duy nhất

- Mỗi chuỗi bit biểu diễn một giá trị số duy nhất
- Mỗi giá trị biểu diễn được có duy nhất một chuỗi biểu diễn

# Ánh xạ giữa số không và có dấu (1)



## ■ Nguyên tắc:

- Trường hợp **chuỗi biểu diễn có bit trọng số cao nhất là 0**, giá trị khi biểu diễn không và có dấu là **như nhau**.
- Ngược lại, bit trọng số cao nhất là 1:
  - Giữ nguyên chuỗi bit biểu diễn
  - Thay đổi giá trị của số theo bit cao nhất

## ■ Trong hệ sử dụng $n$ bit để biểu diễn số, với mỗi chuỗi biểu diễn có **bit trọng số cao nhất = 1**:

- Giá trị không dấu (unsigned) = giá trị có dấu (signed) +  $2^n$
- Giá trị có dấu (signed) = giá trị không dấu (unsigned) -  $2^n$

# Ánh xạ giữa số không và có dấu (2)

Bits	Signed		Unsigned
0000	0		0
0001	1		1
0010	2		2
0011	3		3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8		8
1001	-7		9
1010	-6		10
1011	-5		11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15

# Thêm: số không và có dấu trong C

- Mặc định trong C, các số nguyên là số nguyên **có dấu (signed)**
- **Số nguyên không dấu (unsigned):** thêm hậu tố **U** phía sau:

`0U, 4294967259U`

- Ép kiểu giữa unsigned và signed trong C tương tự như phép ánh xạ giá trị.
- Lưu ý: trong biểu thức chứa cả số có dấu và không dấu, các số có dấu sẽ được chuyển sang không dấu
  - `<, >, ==, <=, >=`

# Nội dung

---

- Biểu diễn thông tin dưới dạng bit
- Tính toán bit
- **Integers – Số nguyên**
  - Biểu diễn: không dấu (unsigned) và có dấu (signed)
  - **Cộng, nhân, dịch bit**
- Biểu diễn trong bộ nhớ, con trỏ, chuỗi



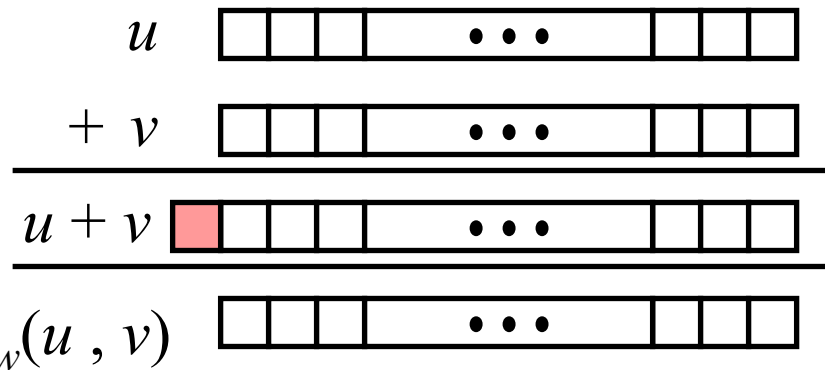
# Phép cộng

## ■ Cộng

Operands:  $w$  bits

True Sum:  $w+1$  bits

Discard Carry:  $w$  bits



- Tổng thực tế có thể yêu cầu  $w+1$  bit, tuy nhiên hệ biểu diễn  $w$  bit bỏ bit cao nhất (MSB).

→ Tràn số (overflow)

# Tràn số trong phép cộng: Ví dụ

- Giả sử dùng **4 bit** để biểu diễn số.
  - Không dấu (unsigned): biểu diễn từ 0 đến 15
  - Có dấu (signed): biểu diễn từ -8 đến +7.
- **Cộng số không dấu (unsigned):**
  - $8 + 8 = 1000 + 1000 = \textcolor{red}{1} 0000 = \textcolor{red}{0}$
  - $9 + 10 = 1001 + 1010 = \textcolor{red}{1} 0011 = \textcolor{red}{3}$
- **Cộng số có dấu (signed):**
  - $7 + 7 = 0111 + 0111 = 1110 = \textcolor{red}{-2}$   
*→ sum > giá trị dương lớn nhất sẽ thành âm*
  - $-5 + -5 = 1011 + 1011 = \textcolor{red}{1} 0110 = \textcolor{red}{6}$   
*→ sum < giá trị âm nhỏ nhất sẽ thành dương*

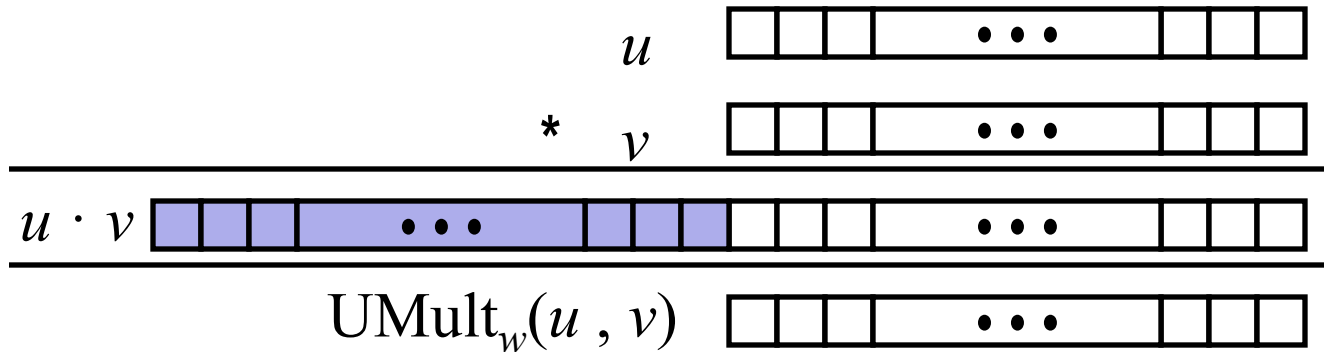
# Phép nhân

## ■ Nhân

Operands:  $w$  bits

True Product:  $2*w$  bits

Discard  $w$  bits:  $w$  bits



- Tích thực tế có thể yêu cầu  $2*w$  bit, tuy nhiên hệ biểu diễn  $w$  bit bỏ các bit cao hơn  $w$ .
- Tràn số (overflow)
- Phép nhân có thể khác nhau trong một vài trường hợp của số có dấu và không dấu
  - Các bit thấp vẫn giống nhau

# Phép nhân với $2^n$ bằng shift trái (1)

- $u \ll k$  tương đương với  $u * 2^k$
- Áp dụng được cho cả số nguyên có dấu (signed) và không dấu (unsigned)
- Với  $u$  được biểu diễn bằng  $w$  bit, kết quả có thể cần  $w + k$  bit để biểu diễn  $\rightarrow$  tràn số
- Ví dụ:
  - $u \ll 3 \quad \quad \quad == \quad \quad u * 8$
  - $(u \ll 5) - (u \ll 3) \quad == \quad u * 24$

# Phép nhân với $2^n$ bằng shift trái (2)

- Hầu hết các máy tính thực hiện **shift** và **cộng** nhanh hơn phép **nhân**
  - Compiler tự động tạo ra mã **shift/cộng** khi nhân **hằng số**

## C Function

```
long mul12(long x)
{
    return x*12;
}
```

## Compiled Arithmetic Operations

```
leaq (%rax,%rax,2), %rax
salq $2, %rax
```

## Explanation

```
t = x+x*2;
return t << 2;
```

# Phép chia không dấu cho $2^n$ bằng shift phải (1)

- $u \gg k$  tương đương với  $u / 2^k$ 
  - Giá trị **nguyên** của phép chia
  - Sử dụng shift luận lý (logic shift)
    - Không quan tâm đến dấu
    - Điền bit 0 dần vào các bit trọng số cao bên trái

	Division	Computed	Hex	Binary
<b>x</b>	<b>15213</b>	<b>15213</b>	3B 6D	00111011 01101101
<b>x &gt;&gt; 1</b>	<b>7606.5</b>	<b>7606</b>	1D B6	00011101 10110110
<b>x &gt;&gt; 4</b>	<b>950.8125</b>	<b>950</b>	03 B6	00000011 10110110
<b>x &gt;&gt; 8</b>	<b>59.4257813</b>	<b>59</b>	00 3B	00000000 00111011

# Phép chia không dấu cho $2^n$ bằng shift phải (2)

## C Function

```
unsigned long udiv8
(unsigned long x)
{
    return x/8;
}
```

## Mã assembly đã biên dịch

```
shrq $3, %rax
```

## Giải thích

```
# Logical shift
return x >> 3;
```

- Sử dụng shift luận lý với số unsigned
- Trong Java
  - Logical shift ký hiệu là >>>

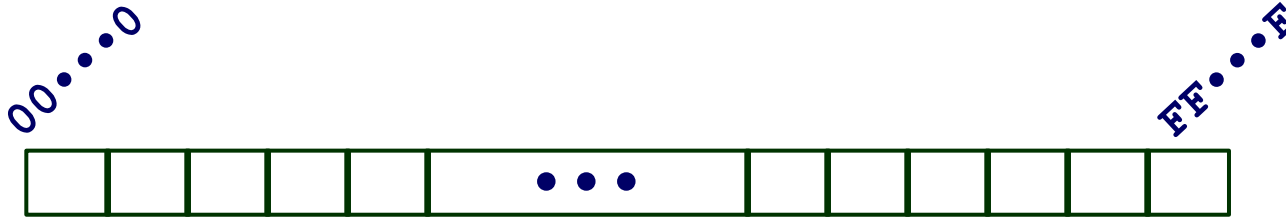
# Nội dung

---

- Biểu diễn thông tin dưới dạng bit
- Tính toán bit
- Integers – Số nguyên
  - Biểu diễn: không dấu (unsigned) và có dấu (signed)
  - Cộng, nhân, dịch bit
- **Biểu diễn trong bộ nhớ, con trỏ, chuỗi**



# Tổ chức bộ nhớ theo byte



- **Chương trình tham chiếu đến dữ liệu bằng địa chỉ**
  - Về mặt lý thuyết, có thể xem bộ nhớ “như” một mảng byte rất lớn
  - Một địa chỉ như một index trong mảng đó
    - Biến pointer chứa một địa chỉ
- **Lưu ý: hệ thống cung cấp các không gian địa chỉ riêng cho mỗi “tiến trình”**
  - 1 tiến trình = 1 chương trình được thực thi

# Word trong máy tính

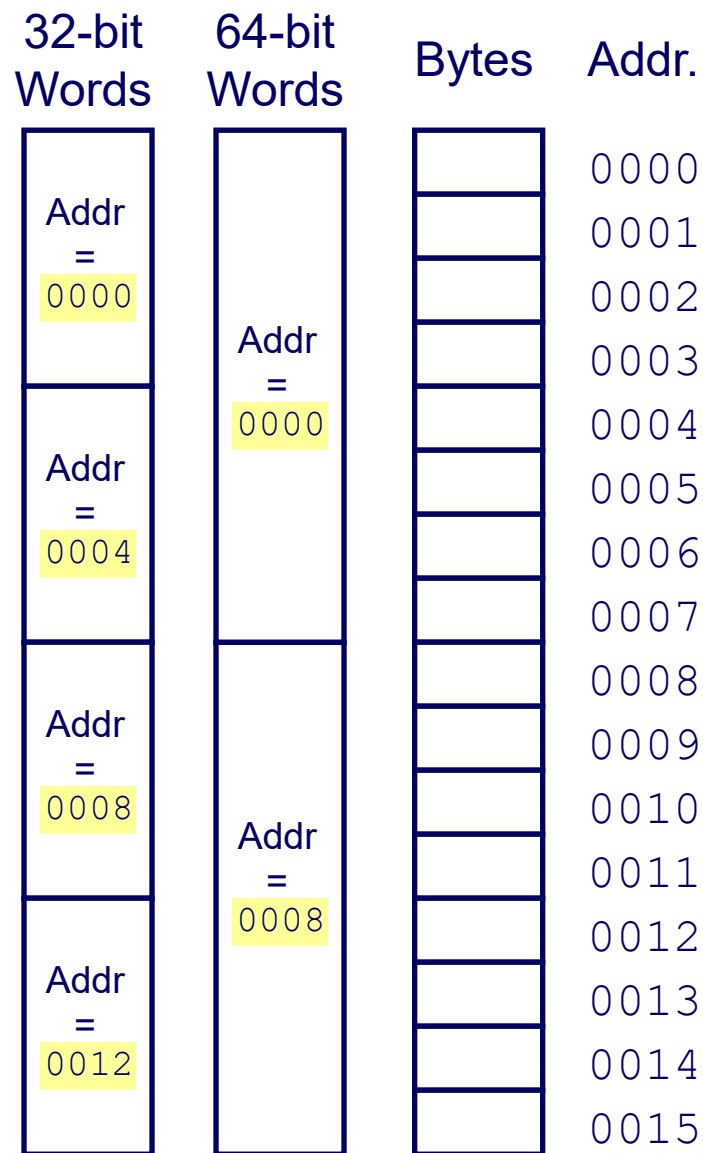
---

- **Một máy tính có 1 “word size”**
  - Kích thước của dữ liệu kiểu integer
    - và của 1 địa chỉ
  - Đến nay, hầu hết các máy tính có word size 32 bits (4 bytes)
  - Ngày càng nhiều các máy có word size 64 bit (8 bytes)

# Tổ chức bộ nhớ theo word

## ■ Địa chỉ xác định vị trí của byte

- Địa chỉ của byte đầu tiên trong word
- Địa chỉ của các word tiếp theo cách nhau 4 (32 bit) hoặc 8 (64 bit)



# Nhắc lại về kích thước kiểu dữ liệu

Đơn vị: bytes

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	–	–	10/16
pointer	4	8	8

Kích thước phụ thuộc vào kích thước của 1 địa chỉ

# Thứ tự byte

---

- Bộ nhớ như một mảng lưu các byte liên tục

→ Vậy với **một word gồm nhiều byte**, các byte sẽ được lưu trữ theo thứ tự nào trong bộ nhớ?

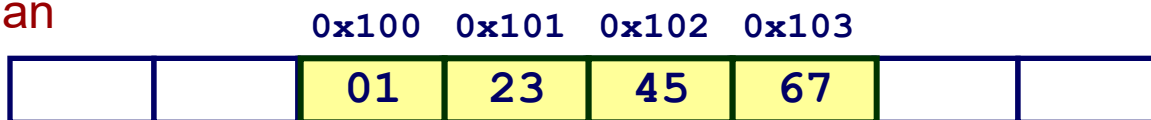
- **2 dạng:**

- **Big Endian:** byte có trọng số thấp nhất nằm ở địa chỉ cao nhất
  - Sun, PPC Mac, Internet
- **Little Endian:** byte có trọng số thấp nhất nằm ở địa chỉ thấp nhất
  - x86, bộ xử lý ARM chạy Android, iOS và Windows

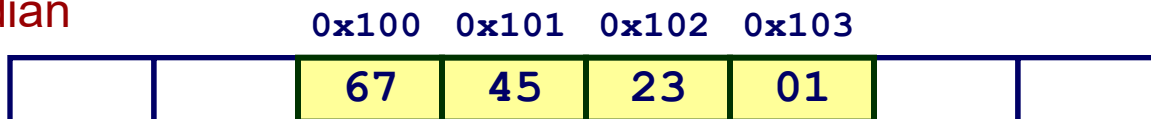
# Thứ tự byte: Ví dụ

- Cho biến **x** có giá trị **0x01234567**
- Địa chỉ để lưu **x** là **0x100**
- Byte thấp nhất **0x67** sẽ lưu ở đâu?

Big Endian



Little Endian



# Ví dụ: Biểu diễn và lưu trữ số nguyên

■ Cho

**int A = 15213 = 0x00003B6D;**

**int B = -15213 = 0xFFFFC493;**

Lưu trữ A, B như thế nào trong các hệ thống:

- IA32, x86-64 (Little Endian)
- Sun (Big Endian)?

`int A = 15213;`

IA32, x86-64	Sun
6D	00
3B	00
00	3B
00	6D

Decimal: 15213

Binary: 0... 0011 1011 0110 1101

Hex: 00... 3 B 6 D

Decimal: -15213

Binary: 1... 1100 0100 1001 0011

Hex: FF... C 4 9 3

`int B = -15213;`

IA32, x86-64	Sun
93	FF
C4	FF
FF	C4
FF	93

# Ví dụ: Code hiển thị byte của 1 dữ liệu (1)

- Code in biểu diễn dưới dạng các byte với đúng thứ tự trong bộ nhớ của dữ liệu
  - Tham số **start** là vị trí lưu của dữ liệu
  - Vì sao phải dùng kiểu **unsigned char\***?
    - Giả sử kiểu dữ liệu là **int**, **start** sẽ là **int\***, **start[i]** sẽ cách nhau mỗi 4 bytes
    - Với ép kiểu pointer sang **unsigned char\***, **start[i]** sẽ cách nhau 1 byte → truy xuất được từng byte của dữ liệu với **i**

Trong hàm **printf**:

**%p**: Print pointer

**%x**: Print Hexadecimal

```
typedef unsigned char *pointer;

void show_bytes(pointer start, size_t len){
    size_t i;
    for (i = 0; i < len; i++)
        printf("%p\t0x%.2x\n", start+i, start[i]);
    printf("\n");
}
```

```
int a = 15213;
printf("int a = 15213;\n");
show_bytes((pointer) &a, sizeof(int));
```



# Ví dụ: Code hiển thị byte của 1 dữ liệu (2)

```
int a = 15213;  
printf("int a = 15213;\n");  
show_bytes((pointer) &a, sizeof(int));
```

## Result (Linux x86-64):

```
int a = 15213;  
0x7fffb7f71dbc    6d  
0x7fffb7f71dbd    3b  
0x7fffb7f71dbe    00  
0x7fffb7f71dbf    00
```

# Biểu diễn con trỏ (pointer)

```
int B = -15213;  
int *P = &B;
```

Sun

EF
FF
FB
2C

IA32

AC
28
F5
FF

x86-64

3C
1B
FE
82
FD
7F
00
00

P = 0xEFFFB2C

P = 0xFFF528AC

P = 0x00007FFD82FE1B3C

- Các compilers và máy tính khác nhau sẽ gán những vị trí khác nhau cho các object.
- Thậm chí khác nhau trong mỗi lần chạy chương trình.

# Biểu diễn chuỗi (strings)

## ■ String trong C

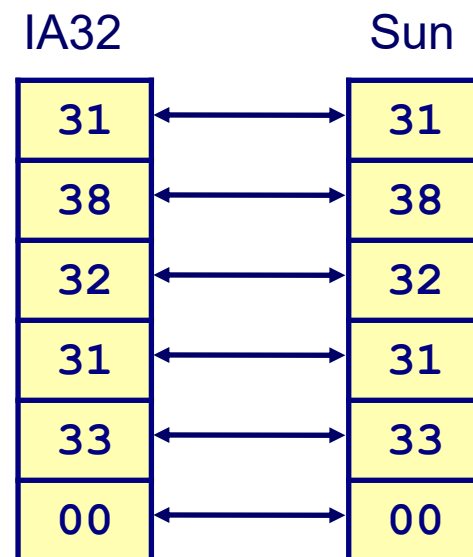
- Là một mảng các ký tự
- Mỗi ký tự ở dạng mã ASCII
  - Chuẩn 7-bit
  - Ký tự '0' tương ứng mã 0x30
    - Số  $i$  tương ứng với mã  $0x30 + i$
- String cần được kết thúc bằng null
  - Ký tự cuối cùng là giá trị 0 ( $\neq$  ký tự '0')

```
char S[6] = "18213";
```

```
0x31 0x38 0x32 0x31 0x33
```

## ■ Lưu ý

- Thứ tự byte của hệ thống không ảnh hưởng đến cách lưu chuỗi
  - Ký tự đầu tiên **luôn luôn** lưu ở địa chỉ thấp nhất



# Nội dung thêm

---

- Phép chia có dấu cho  $2^n$  bằng shift phải
- Đọc các giá trị gồm nhiều bytes trong assembly

# Nội dung tự tìm hiểu: Floating point

- Cách biểu diễn số thực: Floating point
- Phép cộng với số thực

- **(Optional) Bài tập thu hoạch**

- Nộp trên Courses
- 5 bài nhanh nhất



Giải thích vì sao có sự khác biệt bên dưới?

$$(1e20 + -1e20) + 3.14 = \mathbf{3.14}$$

$$1e20 + (-1e20 + 3.14) = \mathbf{0}$$

# Nội dung

## ■ Các chủ đề chính:

- 1) Biểu diễn các kiểu dữ liệu và các phép tính toán bit
- 2) Ngôn ngữ assembly
- 3) Điều khiển luồng trong C với assembly
- 4) Biểu diễn mảng, cấu trúc dữ liệu trong C
- 5) Các thủ tục (procedure) trong C ở mức assembly
- 6) Phân cấp bộ nhớ, cache
- 7) Linking trong biên dịch file thực thi

## ■ Lab liên quan

- Lab 1: Nội dung 1
- Lab 2: Nội dung 1, 2, 3
- Lab 3: Nội dung 1, 2, 3, 4
- Lab 4: Nội dung 1, 2, 3, 4, 5
- Lab 5: Nội dung 1, 2, 3, 4, 5
- Lab 6: Nội dung 1, 2, 3, 4, 5

# Môi trường - Công cụ hỗ trợ

## ■ Hệ điều hành Linux

- Máy ảo/thật
- Hệ thống 32/64 bit
- (Khuyến khích) Tương tác qua giao diện command



**Linux**

## ■ GCC - Trình biên dịch C trên Linux

## ■ Các IDE lập trình

## ■ Phần mềm dịch ngược:

- IDA Pro (GUI)
- GDB (command line)

**IDA**



# Đánh giá

**30%** quá trình/giữa kỳ + **20%** thực hành + **50%** cuối kỳ

- ❑ **Quá trình/giữa kỳ:**

- Bài tập assignment trên lớp
- Kiểm tra giữa kỳ

- ❑ **Thực hành:**

- 6 labs
- Vắng từ 3 buổi thực hành trở lên → trừ **tối thiểu 1/3** số điểm

- ❑ **Cuối kỳ:**

- Trắc nghiệm + Tự luận
- Có thể cho phép sử dụng **01 tờ A4** viết tay



# Yêu cầu

---

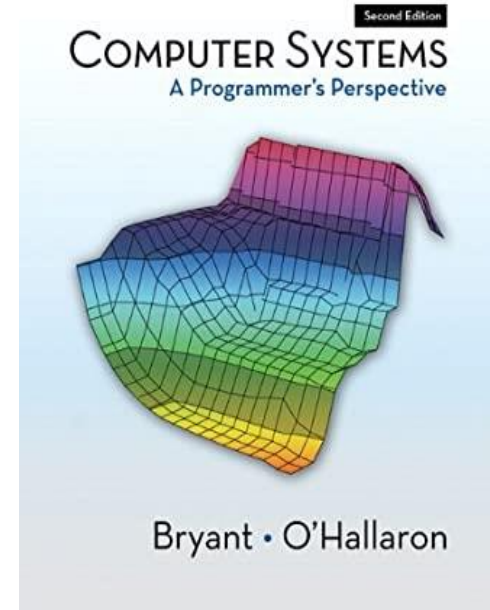
- Đến lớp đúng giờ
- Tìm hiểu trước bài giảng
- Thực hiện đủ Bài tập trên lớp
- Khi làm nhóm:
  - Không ghi nhóm → sao chép
- Sao chép bài → **0**

# Giáo trình

## ■ Giáo trình chính

### ***Computer Systems: A Programmer's Perspective***

- Second Edition (CS:APP2e), Pearson, 2010
- Randal E. Bryant, David R. O'Hallaron
- <http://csapp.cs.cmu.edu>



## ■ Tài liệu khác

- *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Brian Kernighan and Dennis Ritchie
- *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, 1st Edition, 2008
  - Chris Eagle
- *Reversing: Secrets of Reverse Engineering*, 1st Edition, 2011
  - Eldad Eilam



**KEEP  
CALM  
AND  
ENJOY YOUR  
SEMESTER :)**