

LẬP TRÌNH HỆ THỐNG

ThS. Đỗ Thị Thu Hiền
(hiendtt@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)

Machine-level programming

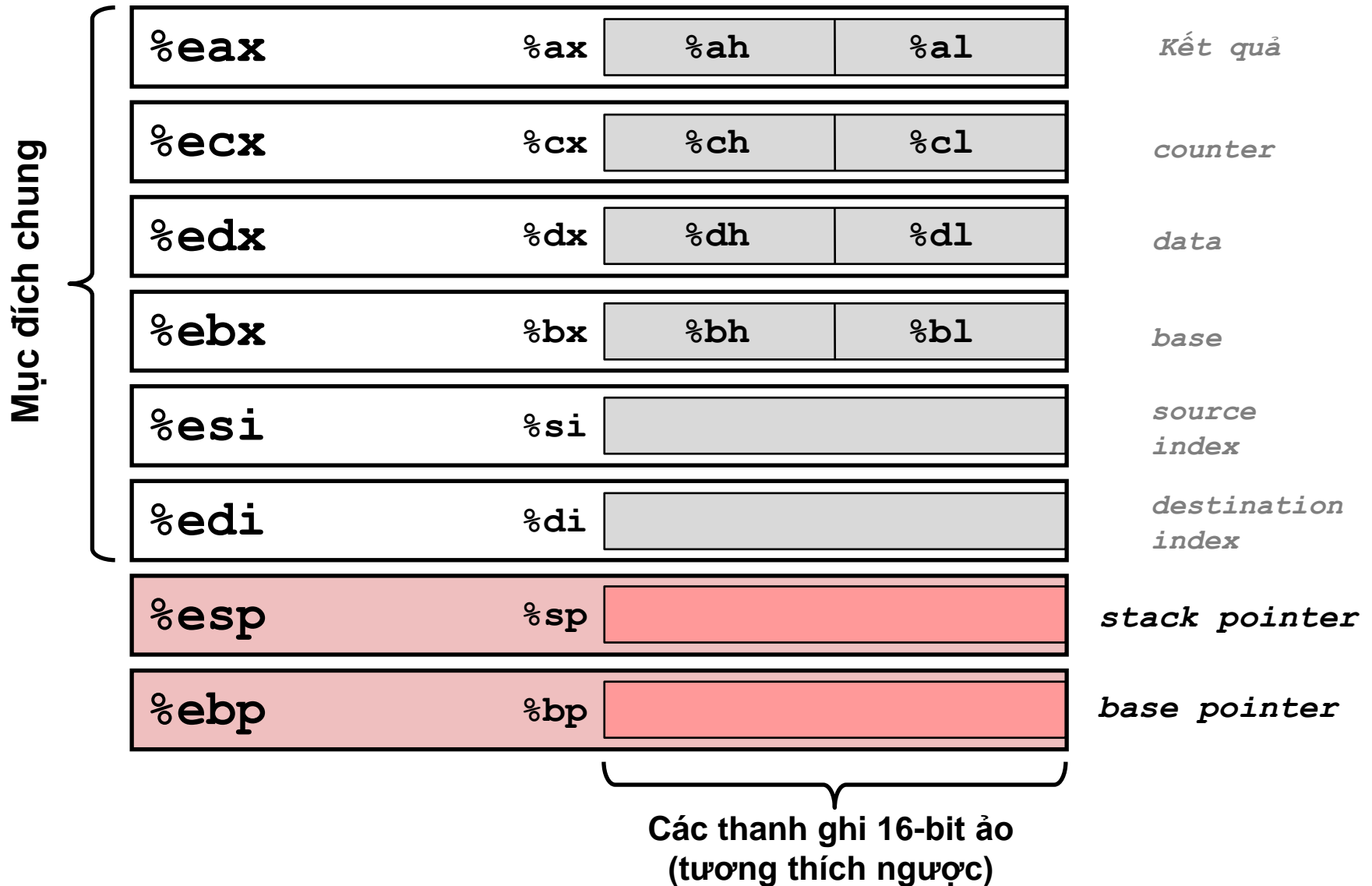
Bài tập



Nội dung

- **Review: Cơ bản về assembly**
 - Registers, move
 - Các phép tính toán học và logic
 - Điều khiển luồng: if-else
- Bài tập 1, 2, ... n
- Assignment 3 & 4 😊

Các thanh ghi IA32 – 8 thanh ghi 32 bit



Các thanh ghi x86-64 – 16 thanh ghi

%rax	%eax
%rbx	%ebx
%rcx	%ecx
%rdx	%edx
%rsi	%esi
%rdi	%edi
%rsp	%esp
%rbp	%ebp

%r8	%r8d
%r9	%r9d
%r10	%r10d
%r11	%r11d
%r12	%r12d
%r13	%r13d
%r14	%r14d
%r15	%r15d

- Mở rộng các thanh ghi 32-bit đã có thành 64-bit, thêm 8 thanh ghi mới.
- **%ebp/%rbp** thành thanh ghi có mục đích chung.
- Có thể tham chiếu đến các 4 bytes thấp (cũng như các 1 & 2 bytes thấp)

Chuyển dữ liệu - Moving Data (IA32)

■ Chuyển dữ liệu

`movl Source, Dest`

■ Các kiểu toán hạng

- **Immediate – Hằng số:** Các hằng số nguyên
 - Ví dụ: `$0x400`, `$-533`
 - Giống hằng số trong C, nhưng có tiền tố ``$'`
 - Mã hoá với 1, 2, hoặc 4 bytes
- **Register – Thanh ghi:** Các thanh ghi được hỗ trợ
 - Ví dụ: `%eax`, `%esi`
 - Nhưng `%esp` và `%ebp` được dành riêng với mục đích đặc biệt
 - Một số khác có tác dụng đặc biệt với một số instruction
- **Memory – Bộ nhớ:** 4 bytes liên tục của bộ nhớ tại địa chỉ nhất định, có thể địa chỉ đó được lưu trong thanh ghi
 - Ví dụ: `(0x100)`, `(%eax)`
 - Có nhiều “address mode” khác

`%eax`

`%ecx`

`%edx`

`%ebx`

`%esi`

`%edi`

`%esp`

`%ebp`

Lưu ý: Suffix cho lệnh mov trong AT&T

■ Quyết định số byte dữ liệu sẽ được “move”

- `movb` 1 byte
- `movw` 2 bytes
- `movl` 4 bytes
- `movq` 8 bytes (dùng với các thanh ghi x86_64)
- `mov` Số bytes tùy ý (phù hợp với tất cả số byte ở trên)

■ Lưu ý: **Các thanh ghi** dùng trong lệnh mov cần đảm bảo phù hợp với **suffix**

- Số byte dữ liệu sẽ được move

*? Có bao nhiêu lệnh mov **hợp lệ** trong các lệnh bên?*

```
movl %eax, %ebx
```

```
movb $123, %bl
```

```
movl %eax, %bl
```

```
movb $3, (%ecx)
```

```
mov (%eax), %bl
```

Các tổ hợp toán hạng cho movl

	Source	Dest	Src, Dest	C Analog
movl	Imm	Reg	movl \$0x4, %eax	temp = 0x4;
		Mem	movl \$-147, (%eax)	*p = -147;
	Reg	Reg	movl %eax, %edx	
		Mem	movl %eax, (%edx)	*p = temp;
	Mem	Reg	movl (%eax), %edx	temp = *p;

Không thể thực hiện chuyển dữ liệu bộ nhớ - bộ nhớ với duy nhất 1 instruction!

Các chế độ đánh địa chỉ bộ nhớ đầy đủ

■ Dạng tổng quát nhất

$$D(Rb, Ri, S) \quad \text{Mem}[\text{Reg}[Rb] + S * \text{Reg}[Ri] + D]$$

- D: Hằng số “dịch chuyển” 1, 2, hoặc 4 bytes
- Rb: Base register: Bất kỳ thanh ghi nào được hỗ trợ
- Ri: Index register: Bất kỳ thanh ghi nào, ngoại trừ %rsp hoặc %esp
- S: Scale: 1, 2, 4, hoặc 8 (*vì sao là những số này?*)

■ Các trường hợp đặc biệt

$$(Rb, Ri) \quad \text{Mem}[\text{Reg}[Rb] + \text{Reg}[Ri]]$$

$$D(Rb, Ri) \quad \text{Mem}[\text{Reg}[Rb] + \text{Reg}[Ri] + D]$$

$$(Rb, Ri, S) \quad \text{Mem}[\text{Reg}[Rb] + S * \text{Reg}[Ri]]$$

Instruction tính toán địa chỉ: `leal`

■ `leal Src, Dst`

- `Src` là biểu thức tính toán địa chỉ
- Gán `Dst` thành địa chỉ được tính toán bằng biểu thức trên

■ Tác dụng

- Tính toán địa chỉ ô nhớ mà ***không tham chiếu đến ô nhớ***
 - Ví dụ, trường hợp `p = &x[i];`
- Tính toán biểu thức toán học có dạng $x + k*i + d$
 - $i = 1, 2, 4, \text{ hoặc } 8$

■ Ví dụ

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

Chuyển sang assembly bằng compiler:

```
leal (%eax,%eax,2), %eax # t <- x+x*2
sall $2, %eax           # return t<<2
```

Một số phép tính toán học (1)

■ Các Instructions với 2 toán hạng:

Định dạng

Phép tính

`addl` *Src, Dest* $\text{Dest} = \text{Dest} + \text{Src}$

`subl` *Src, Dest* $\text{Dest} = \text{Dest} - \text{Src}$

`imull` *Src, Dest* $\text{Dest} = \text{Dest} * \text{Src}$

`sall` *Src, Dest* $\text{Dest} = \text{Dest} \ll \text{Src}$

`sarl` *Src, Dest* $\text{Dest} = \text{Dest} \gg \text{Src}$

`shrl` *Src, Dest* $\text{Dest} = \text{Dest} \gg \text{Src}$

`xorl` *Src, Dest* $\text{Dest} = \text{Dest} \wedge \text{Src}$

`andl` *Src, Dest* $\text{Dest} = \text{Dest} \& \text{Src}$

`orl` *Src, Dest* $\text{Dest} = \text{Dest} | \text{Src}$

Cũng được gọi là `shll`

Arithmetic (shift phải toán học)

Logical (shift phải luận lý)

■ Cần thận với thứ tự của các toán hạng!

■ Không có khác biệt giữa signed và unsigned int

Một số phép tính toán học (2)

■ Các Instructions với 1 toán hạng

`incl` *Dest* $Dest = Dest + 1$

`decl` *Dest* $Dest = Dest - 1$

`negl` *Dest* $Dest = -Dest$

`notl` *Dest* $Dest = \sim Dest$

■ Tham khảo thêm các instruction trong giáo trình

Các câu lệnh jump kết hợp với so sánh

- Các lệnh jump thường kết hợp với các lệnh so sánh/test
 - Kết quả của lệnh so sánh/test quyết định có thực hiện jump hay không.

```
cmpl src2, src1
```

```
jX label
```

jX	Điều kiện nhảy
je	src1 == src2
jne	src1 != src2
jg	src1 > src2
jge	src1 ≥ src2
jl	src1 < src2
jle	src1 ≤ src2

Chuyển mã rẽ nhánh có điều kiện

Phương pháp chung

C code

```
if (test-expr)
    then-statement;
else
    else-statement;
```

Dạng Goto (thực hiện tính toán và luồng tương tự mã assembly)

```
t = test-expr;
if (!t)
    goto False;
then-statement;
goto Done;

False:
    else-statement;

Done:
```

Assembly code

```
...
<instructions to check !t>
jX False
<instructions of then-statement>
jmp Done
False:
    <instruction of else-statement>
Done:
...
```

Nội dung

- Review: Cơ bản về assembly
 - Registers, move
 - Các phép tính toán học và logic
 - Điều khiển luồng: if-else
- Bài tập 1, 2, ... n
- Assignment 3 & 4 😊

Bài tập 1

- Cho trước những giá trị như hình bên được lưu trữ trong bộ nhớ và các thanh ghi

Thanh ghi	Giá trị	Memory	Addr
%eax	0x100	0x11	0x10C
%ecx	0x1	0x13	0x108
%edx	0x3	0xAB	0x104
		0xFF	0x100

Giả sử ta có lệnh `movl X, %ebx`. Với những toán hạng **X** dưới đây, ta sẽ lấy được những giá trị gì để đưa vào `%ebx`?

STT	Toán hạng X	Giá trị lấy được
1	%eax	Giá trị đang lưu trong thanh ghi eax: 0x100
2	\$0x108	Hằng số 0x108
3	0x104	Giá trị trong ô nhớ có địa chỉ 0x104 : 0xAB
4	(%eax)	Giá trị trong ô nhớ có địa chỉ lưu trong %eax, tức 0x100: 0xFF
5	9(%eax, %edx)	Giá trị trong ô nhớ có địa chỉ $\%eax + \%edx + 9 = 0x10C$: 0x11
6	0xFC(,%ecx,4)	Giá trị trong ô nhớ có địa chỉ $4 * \%ecx + 0xFC = 0x100$: 0xFF
7	(%eax, %edx, 4)	Giá trị trong ô nhớ có địa chỉ $\%eax + \%edx * 4 = 0x10C$: 0x11

Bài tập 2

- Cho đoạn mã assembly bên dưới, biết `%eax` lưu giá trị tính toán cuối cùng

x at %ebp+8, y at %ebp+12, z at %ebp+16

```
1.  movl    12(%ebp), %eax
2.  xorl    8(%ebp), %eax
3.  sall    $5, %eax
4.  notl    %eax
5.  subl    16(%ebp), %eax
```

`%eax = y`

`%eax = %eax ^ x = y ^ x (t1)`

`%eax = %eax << 5 = t1 << 5 (t2)`

`%eax = ~%eax = ~t2 (t3)`

`%eax = %eax - z = t3 - z`

- Điền vào những phần còn trống trong hàm C tương ứng dưới đây:

```
1. int arith(int x, int y, int z)
2. {
3.     int t1 = y ^ x ;
4.     int t2 = t1 << 5;
5.     int t3 = ~t2    ;
6.     int t4 = t3 - z ;
7.     return t4;
8. }
```

Bài tập 3

- Cho đoạn mã assembly bên dưới:

```
//x tại địa chỉ %ebp+8, y tại địa chỉ %ebp+12, z tại địa chỉ %ebp+16
1.  movl    16(%ebp), %edx          // %edx = z
2.  movl    12(%ebp), %eax          // %eax = y
3.  addl    8(%ebp), %eax           // %eax = %eax + x = x + y ←
4.  leal    (%edx,%edx,4), %edx      // %edx = %edx * 5 = 5z
5.  addl    %edx, %edx              // %edx = %edx + %edx = 10z ←
6.  xorl    %edx, %eax              // %eax = %eax ^ %edx = ??
7.  ret                               // Trả về
```

- Điền vào những phần còn trống trong mã C tương ứng dưới đây:

```
1. int fun2(int x, int y, int z)
2. {
3.     int t1 = x + y;
4.     int t2 = 10 * z;
5.     return t1 ^ t2;
6. }
```

Bài tập 4 – If/else

Assembly code

```
x at ebp+8, y at ebp+12, sum at eax
1.      movl 8(%ebp),%ecx    //x
2.      movl 12(%ebp),%ebx   //y
3.      cmpl $0,%ecx
4.      jle .L2
5.      leal (%ecx,%ebx),%eax
6.      jmp .L3
7. .L2:
8.      movl %ebx,%eax
9.      subl %ecx,%eax
10. .L3:
```

```
if (x > 0)
    sum = x + y;
else
    sum = y - x;
```

Dự đoán Code C của if/else?

- Điều kiện **true** của if:
x > 0

- Đoạn code tương ứng với true?
Dòng code 5:
sum = x + y;

- Đoạn code tương ứng với false?
Dòng code 8-9:
sum = y;
sum = sum - x;
hay
sum = y - x;

Bài tập 5 (tự làm)

- Alice mới học code assembly cơ bản và mong muốn chuyển đoạn mã C dưới đây thành một đoạn mã assembly:

```
1. int func5(char* str)
2. {
3.     int a = str[0] - '0';
4.     int b = str[1] - '0';
5.     return a + b;
6. }
```

- **str** là một số có 2 chữ số ở dạng chuỗi, ví dụ '12'
- Hàm **func5** tính tổng của các chữ số trong **str**
- Tham số đầu vào (**ở vị trí ebp + 8**) là **địa chỉ lưu chuỗi str** trong bộ nhớ
- Ký tự '0' có mã ASCII là **48 (0x30)**

- Đoạn code assembly được viết bên dưới có chỗ chưa đúng, hãy chỉ ra và đề xuất cách sửa?

```
1.  movl    8(%ebp), %eax    //địa chỉ của str
2.  movl    (%eax), %al      // str[0]
3.  subl    $0x48, %eax      // str[0] - '0'
4.  mov     1(%eax), %bh      // str[1]
5.  subl    $'0', %ebx        // str[1] - '0'
6.  addl    %ebx, %eax
```

Nội dung

- Review: Cơ bản về assembly
 - (Registers, move)
 - Các phép tính toán học và logic
 - Điều khiển luồng: if-else
- Bài tập 1, 2, ... n
- **Assignment 3 & 4 😊**

Assignment 3 – Machine programming Basic

Hãy điền vào bảng giá trị của các thanh ghi, địa chỉ ô nhớ có giá trị bị thay đổi, và giá trị thay đổi đó **sau khi thực thi** từng câu lệnh trên?

ebp

0x100

eax

0x2

```

1 movl    $2, -16(%ebp)
2 movl    $3, -12(%ebp)
3 movl    $0x1, -4(%ebp)
4 movl    -4(%ebp), %eax
5 subl    $1, %eax
6 movl    -16(%ebp,%eax,4), %eax
7 sall    $3, %eax
8 movl    %eax, -8(%ebp)
    
```

Câu lệnh	Thanh ghi thay đổi giá trị	Giá trị thanh ghi	Địa chỉ ô nhớ thay đổi giá trị	Giá trị ô nhớ	Giải thích
1	-	-	0xF0	2	Gán giá trị hằng số 2 vào ô nhớ có địa chỉ bằng $(\%ebp - 16) = 0xF0$
2					
3					
4	%eax	?	-	-	Gán giá trị đang lưu trong ô nhớ có địa chỉ $(\%ebp - 4) = 0xFC$ vào thanh ghi %eax
5					
6					
7					
8					

Assignment 4 – Assembly → C

//a at %ebp+8, b at %ebp+12, c at %ebp+16

```
1.      movl    $0, -4(%ebp)    # result
2.      cmpl    $1, 12(%ebp)
3.      jne     .L2
4.      movl    16(%ebp), %eax
5.      addl    %eax, 8(%ebp)
6.      jmp     .L3
7. .L2:
8.      movl    8(%ebp), %eax
9.      subl    %eax, 16(%ebp)
10. .L3:
11.     movl    8(%ebp), %eax
12.     cmpl    16(%ebp), %eax
13.     jle     .L4
14.     movl    16(%ebp), %eax
15.     addl    %eax, %eax
16.     andl    8(%ebp), %eax
17.     movl    %eax, -4(%ebp)
18.     jmp     .L5
19. .L4:
20.     movl    8(%ebp), %eax
21.     orl     16(%ebp), %eax
22.     movl    %eax, -4(%ebp)
23. .L5:
24.     movl    -4(%ebp), %eax    # result
```

A. Chuyển đoạn **assembly** thành mã **C** tương ứng?

```
int example(int a, int b, int c)
{
    ...
    return result;
}
```

Gợi ý:

- Trong code có các khối lệnh **if/else**
- Ô nhớ tại **%ebp - 4** lưu biến **result**

B. Tìm giá trị **result** với các giá trị tham số **a, b, c**

Lấy 3 số cuối của MSSV lần lượt là các giá trị **a, b, c**.

VD: 20520**123** → **a = 1, b = 2, c = 3**

Bài tập bonus



Giả sử ta có đoạn mã assembly như bên dưới

x at %ebp+8, n at %ebp+12

```
1.  movl    12(%ebp), %ecx    // n
2.  movl    8(%ebp), %edx     // x
3.  xorl    %eax, %eax
4.  addl    $1, %eax
5.  sall    %ecx, %eax
6.  andl    %edx, %eax
7.  sarl    %ecx, %eax
8.  andl    $1, %eax
```

Trả lời các câu hỏi sau:

1. Instruction thứ 3 (lệnh **xor**) có tác dụng gì?
2. Instruction thứ 5 & 7 thực hiện các phép dịch bit (**sall** và **sarl**) với số bit cần dịch lưu trong thanh ghi **%ecx**, tuy nhiên đang bị lỗi. Lý giải nguyên nhân bị lỗi và sửa lại cho đúng?
3. Viết hàm C tương ứng với mã assembly trên: `int func3(int x, int n)`
Thử dự đoán chức năng của đoạn mã này?

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) Các thủ tục/hàm (procedure) trong C ở mức assembly
- 5) Biểu diễn mảng, cấu trúc dữ liệu trong C
- 6) Một số topic ATTT: reverse engineering, bufferoverflow
- 7) Phân cấp bộ nhớ, cache
- 8) Linking trong biên dịch file thực thi

■ Lab liên quan

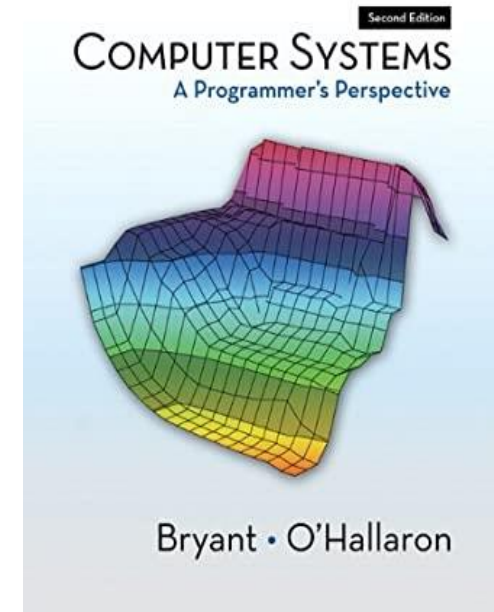
- | | |
|---|---|
| ▪ Lab 1: Nội dung <u>1</u> | ▪ Lab 4: Nội dung 1, <u>2</u> , 3, <u>4</u> , 5, <u>6</u> |
| ▪ Lab 2: Nội dung 1, <u>2</u> , <u>3</u> | ▪ Lab 5: Nội dung 1, <u>2</u> , 3, <u>4</u> , 5, <u>6</u> |
| ▪ Lab 3: Nội dung 1, <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , <u>6</u> | ▪ Lab 6: Nội dung <u>1</u> , <u>2</u> , <u>3</u> , <u>4</u> , <u>5</u> , <u>6</u> |

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 :)**