LẬP TRÌNH HỆ THỐNG

ThS. Đỗ Thị Hương Lan
(landth@uit.edu.vn)

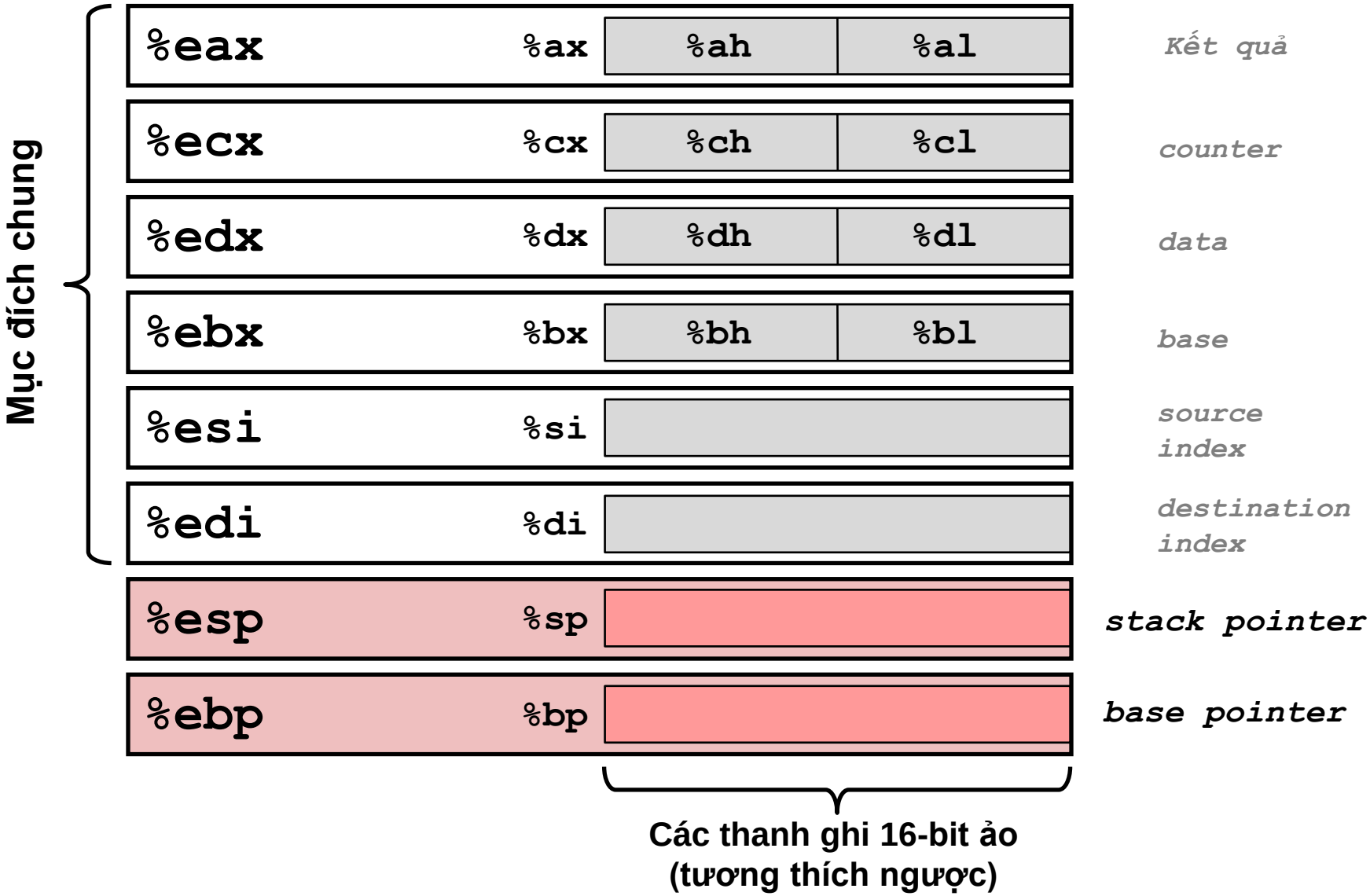
TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN - ĐHQG-HCM
KHOA MẠNG MÁY TÍNH & TRUYỀN THÔNG
FACULTY OF COMPUTER NETWORK AND COMMUNICATIONS
Tầng 8 - Tòa nhà E, trường ĐH Công nghệ Thông tin, ĐHQG-HCM
Điện thoại: (08)3 725 1993 (122)

Machine-level programming
Bài tập

Nội dung

- **Review: Cơ bản về assembly**
  - **Registers, move**
  - **Các phép tính toán học và logic**
- **Bài tập 1, 2, … n**
- **Assignment 2 (& bonus) ☺**

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`, `(0x100)`, `(%eax)`
    - Có nhiều "address mode" khác

Lưu ý: **Suffix cho lệnh `mov` trong AT&T**

- **Quyết định số byte dữ liệu sẽ được "move"**
  - mov**b** 1 byte
  - mov**w** 2 bytes
  - mov**l** 4 bytes
  - mov**q** 8 bytes (dùng với các thanh ghi x86_64)
  - mov Số bytes tuỳ 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) **Mem[Reg[Rb]+S*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) **Mem[Reg[Rb]+Reg[Ri]]**

  D(Rb,Ri) **Mem[Reg[Rb]+Reg[Ri]+D]**

  (Rb,Ri,S) **Mem[Reg[Rb]+S*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, 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*	Dest = Dest + Src	
`subl`	*Src,Dest*	Dest = Dest – Src	
`imull`	*Src,Dest*	Dest = Dest * Src	
`sall`	*Src,Dest*	Dest = Dest << Src	*Cũng được gọi là shll*
`sarl`	*Src,Dest*	Dest = Dest >> Src	*Arithmetic (shift phải toán học)*
`shrl`	*Src,Dest*	Dest = Dest >> Src	*Logical (shift phải luận lý)*
`xorl`	*Src,Dest*	Dest = Dest ^ Src	
`andl`	*Src,Dest*	Dest = Dest & Src	
`orl`	*Src,Dest*	Dest = Dest | Src	

- **Cẩn 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 = ~Dest*

- **Tham khảo thêm các instruction trong giáo trình**

0000 1001
0000 1010

Nội dung

- Review: Cơ bản về assembly
  - Registers, move
  - Các phép tính toán học và logic
- Giải bài tập trắc nghiệm
- **Bài tập 1, 2, … n**
- Assignment 2 (& bonus) ☺

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
%eax	0x100
%ecx	0x1
%edx	0x3

Memory	Addr
0x11	0x10C
0x15	0x108
0xAB	0x104
0xF9	0x100

Những câu lệnh sau ảnh hưởng đến giá trị của thanh ghi/ô nhớ như thế nào?

Câu lệnh	Thanh ghi/ô nhớ bị thay đổi	Giá trị thay đổi như thế nào?
addl %ecx, (%eax)	ô nhớ	(%eax) = (%eax) + %ecx = 0xF9 + 0X1 = 0xF
imull $2, (%eax, %edx, 4)	Ô NHớ	(eax + 4*%edx) = (0x100+ 4*0x3) = (5*%edx) *$2
subl %ecx, %eax	thanh ghi	%eax = %eax - %ecx = 0xFF
movl (%eax, %ecx, 8), %eax	thanh ghi	%eax = [%eax + %ecx*8] = (0x100 + 8*0x1) = (0x108) = 0x15
leal (%eax, %ecx, 8), %edx	thanh gh	%edx = %eax + %ecx*8 = 0x1(

Bài tập 2

Lệnh nào có thể gán %ebx = 2?

A. `movl %eax, %ebx`

B. `movl 2, %ebx`

C. `addl %eax, %ebx`

D. `imull %eax, %ebx`

E. `movb $2, %bl`

F. `movl (%edx,%eax,2), %ebx`

G. `movl 1(%eax), %ebx`

H. `addl (%ecx), %ebx`

leal (%eax), %ebx

sall $1, %ebx
movl (%ebx, (

mov 4(%eax), %bx
sall $1, %ebx
imul/movl 0x104, %ebx

Registers

%eax	0x2
%ebx	0x1
%ecx	0x100
%edx	0x104

Memory

	Address
0x1	0x108
0x2	0x104
0x1	0x100

100000000
100000100

Bài tập 3

Các lệnh không hợp lệ?

```
(1) movl %eax, %bl
(2) movb $3, (%ebx)
(3) movw %eax, $0x100
(4) mov (0x102), (%ecx)
(5) mov 8(%ebx), %edx
(6) movl %ecx, 0x303
```

Bài tập 4

- 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.    incl   %eax
5.    subl   16(%ebp), %eax
```

eax = y
eax = y ^ x = 010 ^101 = 111

eax = (y^x)<<5 = 7*2^5 = 224 (11100

eax= ((y^x)<<5)++ = 225

eax = eax - (ebp +16)
= (((y^x)<<5)++)-z = 225 -3 = 22

- Giả sử x = 2, y = 5, z = 3. Hỏi kết quả cuối cùng là bao nhiêu?

Bài tập 5: Viết code assembly

- Cho **x** lưu ở ô nhớ **8(%ebp)**, **y** lưu ở ô nhớ **12(%ebp)**.

Viết đoạn chương trình tính toán biểu thức (lấy phần nguyên), kết quả cuối lưu vào thanh ghi **%eax**?

$$\frac{(x+y)^2}{2}$$

```
movl 8(ebp), ea
addl 12(ebp),ea
imull eax,eax
shrl $1,eax
```

Bài tập 6: Viết lệnh assembly

Viết một số lệnh hiện thực các ý tưởng sau

Lưu ý: Các lệnh thực thi riêng biệt không liên quan đến nhau. Hệ thống đang xét 32bit

	Tác vụ	Lệnh assembly
0	Tăng giá trị thanh ghi %eax lên 1 đơn vị	incl %eax addl $1, %eax leal 1(%eax), %eax
1	Nhân giá trị đang lưu trong %ecx với 4	imull $4, %ecx, sall $2,%ecx
2	Tính toán địa chỉ ở ô nhớ nằm trên địa chỉ đang lưu trong %eax 12 byte. Kết quả lưu trong %ebx	leal 12(%eax), %ebx # %ebx = %eax + 12
3	Trừ giá trị đang lưu trong ô nhớ địa chỉ 0x104 cho %edx, lưu kết quả vào chính ô nhớ đó.	subl %edx, (0x104)
4	Giữ nguyên 4 bit thấp nhất của giá trị thanh ghi %ecx, các bit còn lại gán về 0.	
5	Chỉ lấy 2 byte từ ô nhớ nằm dưới địa chỉ đang lưu trong %eax 4 byte, kết quả lưu trong thanh ghi 2 byte của %ecx.	

Bài tập 7: Hoàn thiện đoạn lệnh

Cho trước **giá trị biến a** đang lưu trong ô nhớ có **địa chỉ 0x102**, **%ebx = 0x100**. Giả sử cần tính toán biểu thức **10a + 4**. Hoàn thiện các lệnh assembly bên dưới để hiện thực ý tưởng như đã ghi chú?

1. **movl** (0x102), %eax *# Chuyển a từ ô nhớ 0x102 sang %eax*
2. **leal** (eax,eax4), %eax *# %eax = 5*%eax = 5*a*
3. **addl** $2,eax, %eax *# %eax = %eax + 2 = 5*a + 2*
4. **imull** $2, %eax *# %eax = %eax*2 = 10*a + 4*

Bài tập 8: Viết đoạn lệnh assembly

Cho **%eax = a, %ebx = b.**
Hãy dùng lệnh assembly để tính toán **6a + b + 4** lưu trong %eax trong phạm vi **2 lệnh**?

```
leal (%eax,%eax,2), %eax      # eax = 3a
leal 4(%ebx,%eax,2), %eax     # eax = b + (3a)*2 + 4 = 6a + b + 4
```

Bài tập 9

- **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
```

đổi giá trị ô nhớ-> eax -> đổi thanh ghi ->

lấy bit cao -> sai -> đổi thành bit thấp -

Nội dung

- Review: Cơ bản về assembly
  - (Registers, move)
  - Các phép tính toán học và logic
- Giải bài tập trắc nghiệm
- Bài tập 1, 2, … n
- **Assignment 2 (& bonus) ☺**

Assignment 2 – 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 0xFC **eax** 0x1

```
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  -12(%ebp, %eax, 4), %eax
7. sall  $2, %eax
8. movl  %eax, -8(%ebp)
```

Câu lệnh	Vị trí thay đổi (Ô nhớ/thanh ghi?)	Tên thanh ghi/địa chỉ ô nhớ	Giá trị mới?	Giải thích
1	Ô nhớ	0xEC	2	Ô nhớ có địa chỉ **(%ebp – 16) = 0xFC – 16 = 0xEC** được gán giá trị hằng số 2
2	Ô nhớ .	0XF0	3	Ô nhớ có địa chỉ (%ebp-12) = (0XFC-12) = (0XF0) được gán giá trị hằng số 3 .
3	Ô nhớ .	0XF8	0X1	Ô nhớ có địa chỉ (%ebp-4) = (0XFC-4) = (0XF8) được giá trị hằng số 0x1
4	Thanh ghi	%eax	?? 0x1	Gía trị lưu tại ô nhớ (%ebp-4) = (0XFC-4) = (0XF8) là lưu vào thanh ghi %eax
5	Thanh ghi	%eax	0x0	eax = eax - 1 = 0x1 -1 =0
6	Thanh ghi	%eax	0x3	giá trị lưu tại ô nhớ (%ebp+%eax*4-12) = (0XFC+0-12) (0XF0) là 0x3 được gán vào eax
7	Thanh ghi	%eax	0xC	dịch trái eax=0x3 2 bit -> 0xc lưu lại vào ea
8	Ô nhớ .	0xF4	0xc	Ô nhớ có địa chỉ (%ebp-8) = (0XFC-8) = (0XF4) được gán giá trị là 0xc

Nội dung

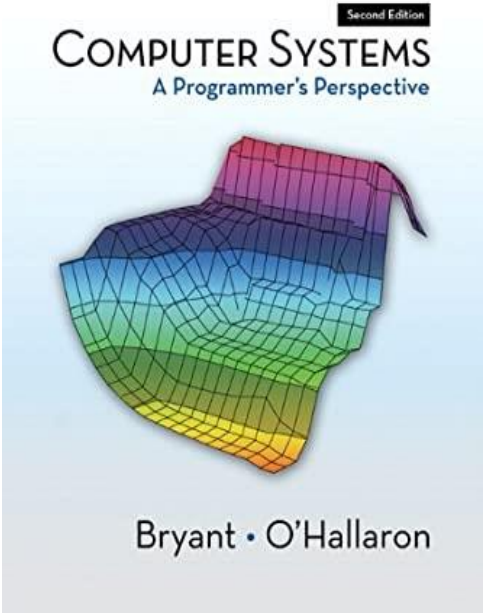
- **Các chủ đề chính:**
  1) Biểu diễn các kiểu dữ liệu và các phép tính toán bit
  2) Ngôn ngữ assembly cơ bản
  3) Điều khiển luồng trong C với assembly
  4) Các thủ tục/hàm (procedure) trong C ở mức assembly
  5) Biểu diễn mảng, cấu trúc dữ liệu trong C
  6) Một số topic ATTT: reverse engineering, bufferoverflow
  7) Phân cấp bộ nhớ, cache
  8) Linking trong biên dịch file thực thi
- **Lab liên quan**
  - Lab 1: Nội dung **1**
  - Lab 2: Nội dung 1, **2**, **3**
  - Lab 3: Nội dung 1, **2, 3, 4, 5, 6**
  - Lab 4: Nội dung 1, **2,** 3**,** **4**, 5, **6**
  - Lab 5: Nội dung 1, **2,** 3**,** **4**, 5, **6**
  - Lab 6: Nội dung **1, 2, 3**, **4, 5, 6**

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 :)