LẬP TRÌNH HỆ THỐNG

TS. Phạm Văn Hậu - ThS. Đỗ Thị Thu Hiền
haupv@uit.edu.vn - 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)

Giới thiệu nội dung môn học

Khảo sát: Lỗi trong C

Tràn số?

Segmentation fault?

Khảo sát

Reverse engineering?

Buffer overflow?

Chèn mã độc vào 1 chương trình thực thi?

Khảo sát

CTF?

Câu lạc bộ Wanna.W1n?

Lập trình?

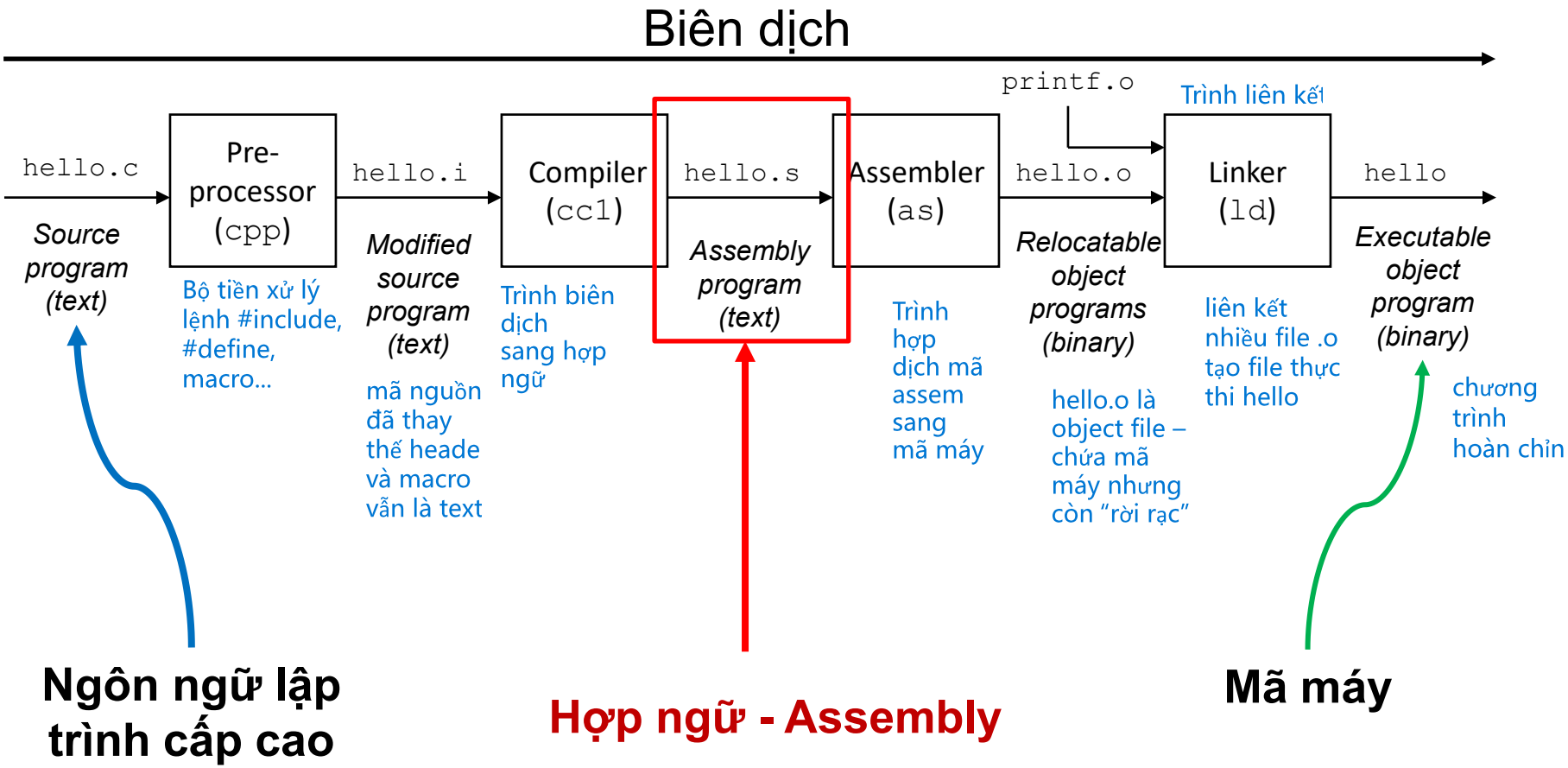
code/intro/hello.c

```
1   #include <stdio.h>
2
3   int main()
4   {
5       printf("hello, world\n");
6   }
```

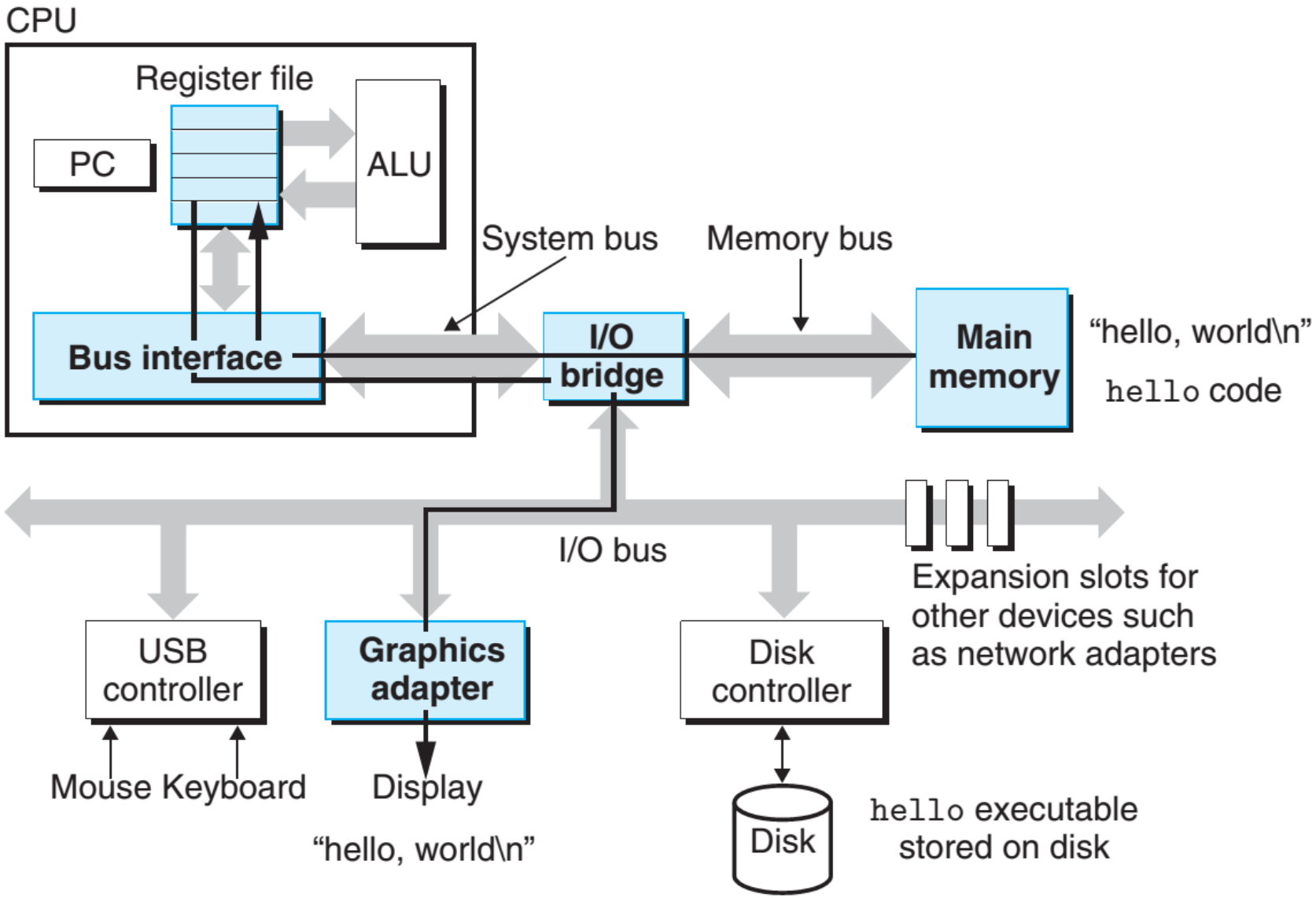
code/intro/hello.c

Figure 1.1 **The `hello` program.**

Hiểu các hoạt động ở mức máy tính?



Ví dụ: Xuất “Hello, World” ra màn hình



Thông tin môn học

- **Môn học:** Lập trình hệ thống
- **30 tiết** lý thuyết (**15** buổi – **2 tiết**/buổi)
  - Tiết 4 – 5 thứ 3 hàng tuần

- **Giảng viên lý thuyết:**
  - TS **Phạm Văn Hậu**
  - ThS **Đỗ Thị Thu Hiền**
  - Email: **haupv@uit.edu.vn - hiendtt@uit.edu.vn**

- **Kênh trao đổi thông tin**:
  - **Email** (*Nhớ kèm theo **mã lớp** trên **Tiêu đề mail!***)
  - Courses

Mục tiêu

Cung cấp các kiến thức gồm:

- Khái niệm cơ bản về lập trình hệ thống ở dạng **hợp ngữ - asssembly**, cách chuyển đổi ngôn ngữ cấp cao sang mã hợp ngữ và ngược lại.
- Những khái niệm về bộ nhớ, stack, pointer, cache và kiến trúc máy tính.
- Kiến thức và kỹ năng tối ưu hóa chương trình (nâng cao)

***Nhằm*:**

- Xây dựng được chương trình an toàn hơn, hiệu quả hơn và có tầm nhìn hệ thống hơn.
- Phục vụ cho các **kỹ thuật dịch ngược**, **debug** và **kiểm lỗi** phần mềm.

Nội dung

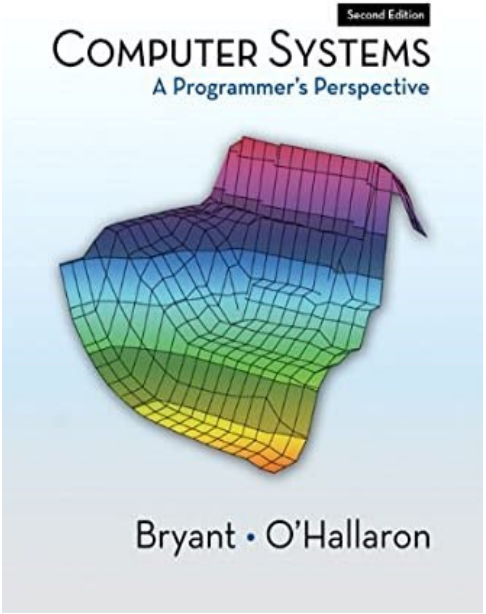
- **Các chủ đề chính:**
  1) Biểu diễn các kiểu dữ liệu cơ bản và các phép tính toán bit
  2) Ngôn ngữ assembly
  3) Điều khiển luồng trong C với assembly
  4) Các thủ tục/hàm (procedure) trong C ở mức assembly
  5) Biểu diễn mảng, cấu trúc dữ liệu trong C
  6) Một số topic ATTT: reverse engineering,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



- **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

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
- **GCC** - Trình biên dịch C trên Linux
- Các **IDE lập trình**
- **Phần mềm dịch ngược**:
  - **IDA Pro (GUI)**
  - GDB (command line)

Linux

Đánh giá

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

- **Quá trình/giữa kỳ:**
  - Bài tập assignment trên lớp/về nhà + Bài tập CTF
    - Bài tập bắt buộc/tự tìm hiểu
  - Thi giữa kỳ (*tự tổ chức*)
- **Thực hành:**
  - **6** labs
  - Có tính điểm chuyên cần :)
- **Cuối kỳ:**
  - Trắc nghiệm + Tự luận
  - Có thể cho phép sử dụng **01 tờ A4** viết tay

Yêu cầu

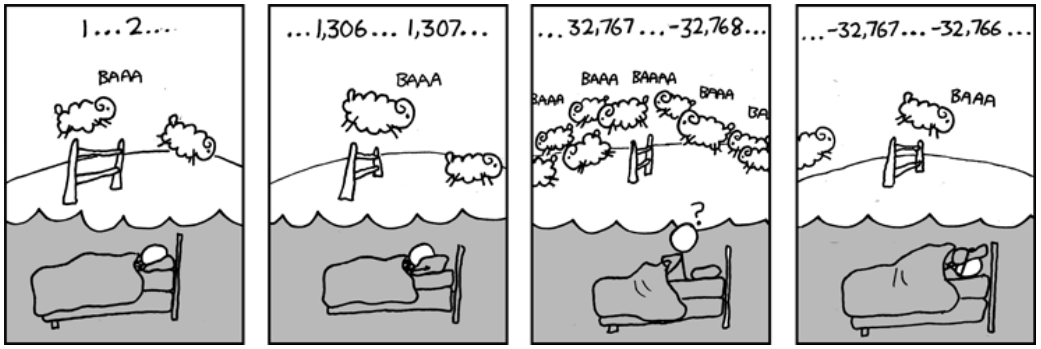
- Đến lớp đúng giờ
- Tìm hiểu trước bài giảng
- Thực hiện đủ Bài tập trên lớp + về nhà
- Khi làm nhóm:
  - Không ghi nhóm → sao chép
- Sao chép bài → **0**

Đánh giá… thêm :)

- Trả lời các câu hỏi
- Điểm tích luỹ các bài tập assignment tự tìm hiểu

Vấn đề #1:
Kiểu Int hay Float có thực sự là số nguyên, số thực?

- **Ví dụ 1: Có chắc $x^2 \geq 0$?**
  - Float: Đúng!



  - Int:
    - 40.000 * 40.000 = 1.600.000.000
    - 50.000 * 50.000 = **??**

- **Ví dụ 2: Có chắc (x + y) + z = x + (y + z)?**
  - Kiểu int có dấu và không dấu: Đúng!
  - Float:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> **??**

(-1e20 + 3.14) gần như vẫn là -1e20 (3.14 quá nh so với 1e20, bị làm tròn).

1e20 + (-1e20) → ~0

Kết quả cuối cùng → 0.0 ✗ (mất 3.14)

Tính toán số học trong máy tính?

- Các phép tính toán số học có những tính chất quan trọng
- Không thể giả định tất cả tính chất toán học "thông thường"
  - Do đặc điểm biểu diễn giá trị trong máy tính
  - Các phép tính số nguyên thoả mãn các tính chất:
    - Giao hoán, kết hợp, phân phối
  - Các phép tính số float thoả mãn các tính chất:
    - Tính đơn điệu, các dấu

→ Cần phải hiểu kiểu nào được áp dụng trong ngữ cảnh nào

→ Vấn đề quan trọng đối với lập trình compiler và lập trình các ứng dụng quan trọng

Vấn đề #2:
Cần phải biết Assembly – Hợp ngữ

- **Hiểu assembly = hiểu quá trình thực thi ở mức máy tính**
  - **Hành vi của các chương trình có bug**
    - Vấn đề đang xảy ra ở ngôn ngữ lập trình cấp cao
  - **Tăng hiệu suất thực thi của chương trình**
    - Hiểu được các bước tối ưu hoá mà các compiler thực hiện
    - Hiểu được nguyên nhân làm hiệu suất chương trình thấp
  - **Triển khai các phần mềm hệ thống**
  - **Tạo/chống các malware**
    - Assembly x86 là lựa chọn hay dùng!

Vấn đề #3:
Ảnh hưởng của bộ nhớ: Vấn đề khi truy cập bộ nhớ?

- **Bộ nhớ (memory) có giới hạn**
  - Cần được cấp phát và quản lý hợp lý
  - Nhiều ứng dụng bị chi phối bởi bộ nhớ
- **Các bug/lỗi khi truy xuất bộ nhớ rất nguy hiểm**
  - Ảnh hưởng lớn đến cả thời gian và không gian thực thi của ứng dụng
- **Hiệu suất của bộ nhớ**
  - Cache và bộ nhớ ảo có thể tác động lớn đến hiệu suất chương trình
  - Chương trình thích nghi được với đặc điểm của hệ thống bộ nhớ có thể cải thiện đáng kể tốc độ

Ví dụ: Bug khi tham chiếu bộ nhớ – Tại sao?

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

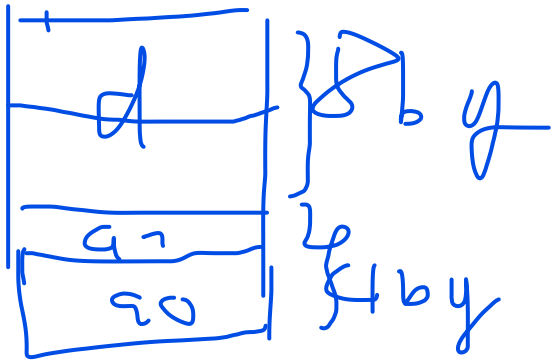
```
fun(0)   =      3.14
fun(1)   =      3.14
fun(2)   =      3.1399998664856
fun(3)   =      2.0000061035156
fun(4)   =      3.14
fun(6)   =      Segmentation fault
```

ghi đè 4 byte đầu của d-> thay đổi ít

ghi đè 4 byte sau của d-> thành số khac

ghi đè 4 byte sau sô d-> không ảnh hư

truy cập vùng không thuộc



- Kết quả phụ thuộc vào từng hệ thống

Các lỗi tham chiếu bộ nhớ

- **C và C++ không hỗ trợ bảo vệ bộ nhớ (memory protection)**
  - Out of bounds khi tham chiếu array (mảng)
  - Giá trị pointer không hợp lệ
  - Lạm dụng các hàm malloc/free
- **Có thể dẫn đến các lỗi**
  - Có thể dẫn đến bug hay không phụ thuộc vào hệ thống và compiler
  - Tác động
    - Thay đổi các object không liên quan đến object đang được truy xuất
    - Bug có thể chỉ được thấy sau một thời gian dài đã tồn tại
- **Cách khắc phục?**
  - Lập trình bằng Java, Ruby, Python, ML, …
  - Hiểu những tương tác nào có thể xảy ra
  - Dùng hoặc phát triển các công cụ phát hiện lỗi tham chiếu (vd. Valgrind)

Vấn đề #4: **Có nhiều thứ ảnh hưởng đến hiệu suất của chương trình hơn là độ phức tạp**

- **Số lượng phép tính toán có thể vẫn chưa dự đoán được hiệu suất**
  - Cần tối ưu hoá ở nhiều mức: giải thuật, biểu diễn dữ liệu, procedure, các vòng lặp…
- **Phải hiểu được ở mức độ hệ thống để tối ưu hiệu suất**
  - Hiểu cách chương trình được biên dịch và thực thi
  - Hiểu cách tính toán hiệu suất và xác định được thành phần bottleneck
  - Hiểu cách cải thiện hiệu suất mà không ảnh hưởng đến các tính mô-đun và tổng quát của code

Ví dụ: Hiệu suất của bộ nhớ

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

CPU tải một cache line (ví dụ 64 byte), chứa được nhiều phần tử liên tiếp của hàng

4.3ms **81.8ms**

2.0 GHz Intel Core i7 Haswell

Mỗi lần truy xuất phải nạp cache line mới

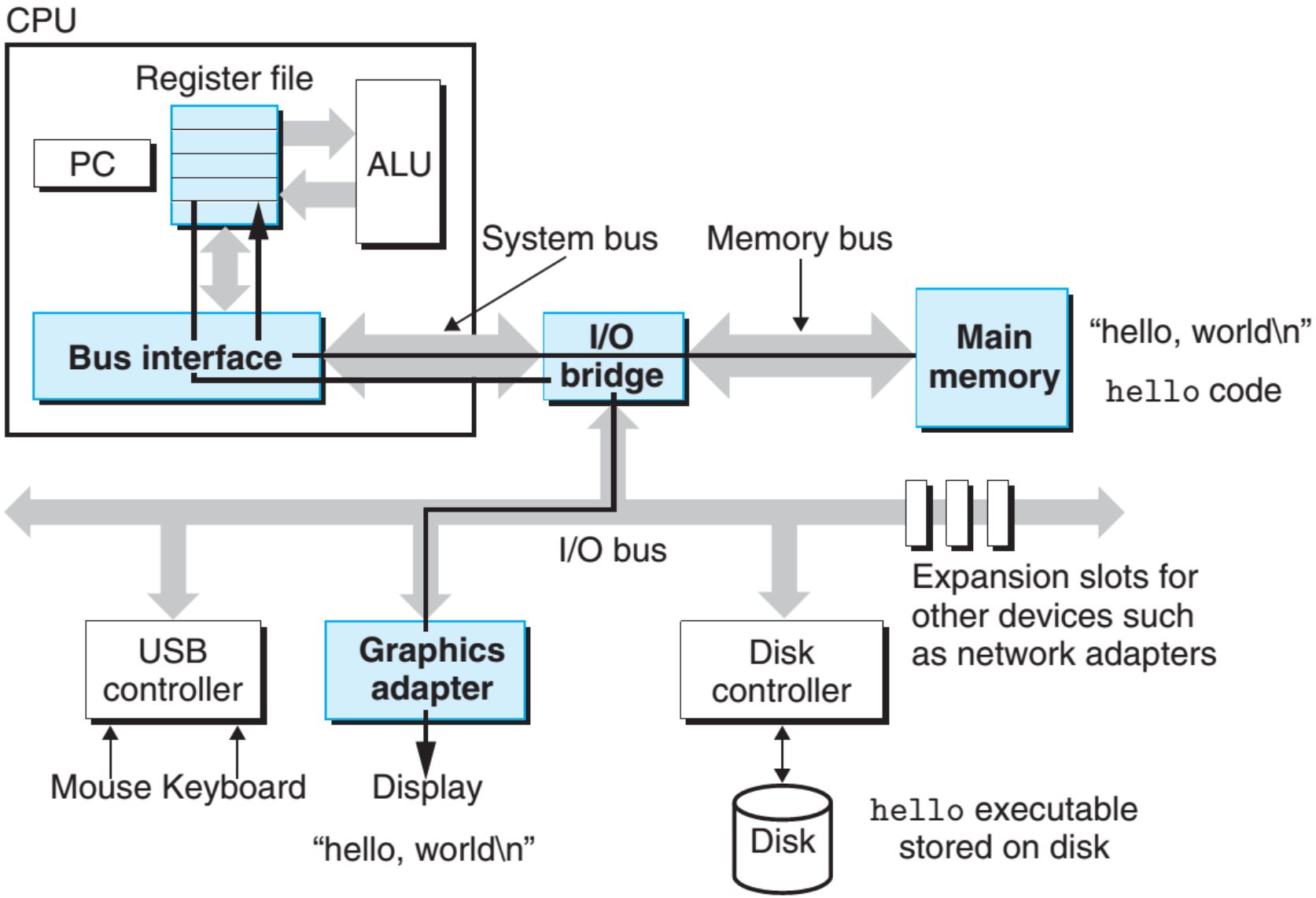
- **Hiệu suất phụ thuộc vào cách truy xuất**
  - Bao gồm cách truy xuất các phần tử trong mảng đa chiều

    phần tử có chỉ số hàng giống nhau nằm gần nhau trong bộ n
    src[0][0], src[0][1], src[0][2], ... src[0][2047],
    src[1][0], src[1][1], ...

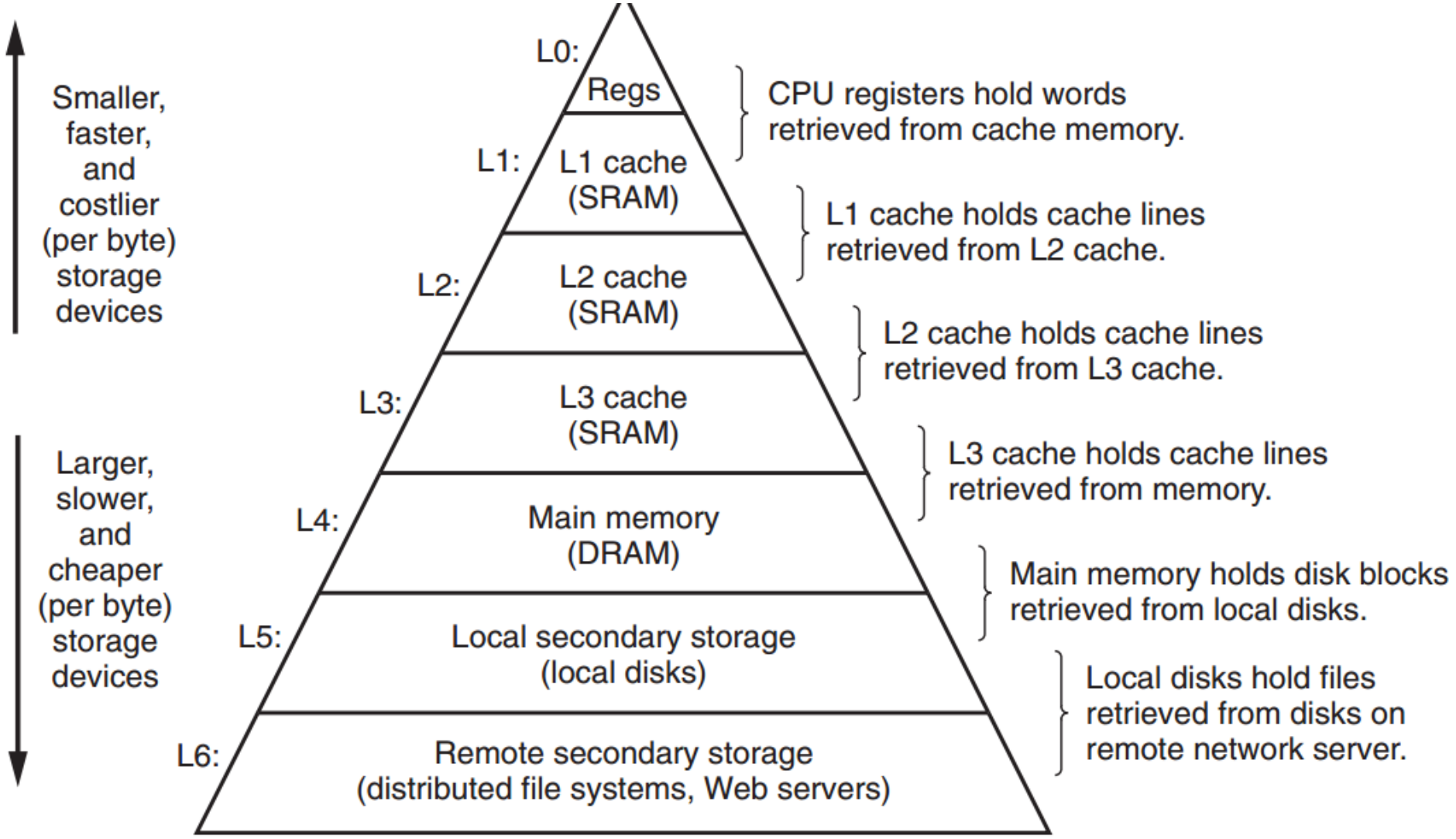
Vấn đề #5: **Máy tính làm nhiều hơn việc chỉ thực thi các chương trình**

- **Máy tính cần đọc và ghi dữ liệu**
  - Vấn đề I/O ảnh hưởng đến độ tin cậy và hiệu suất chương trình
- **Máy tính kết nối với các máy tính khác qua mạng**
  - Nhiều vấn đề cấp hệ thống phát sinh khi có mạng
- **Hệ thống lưu trữ có nhiều phân cấp**
  - Kích thước, tốc độ truy xuất, giá thành khác nhau

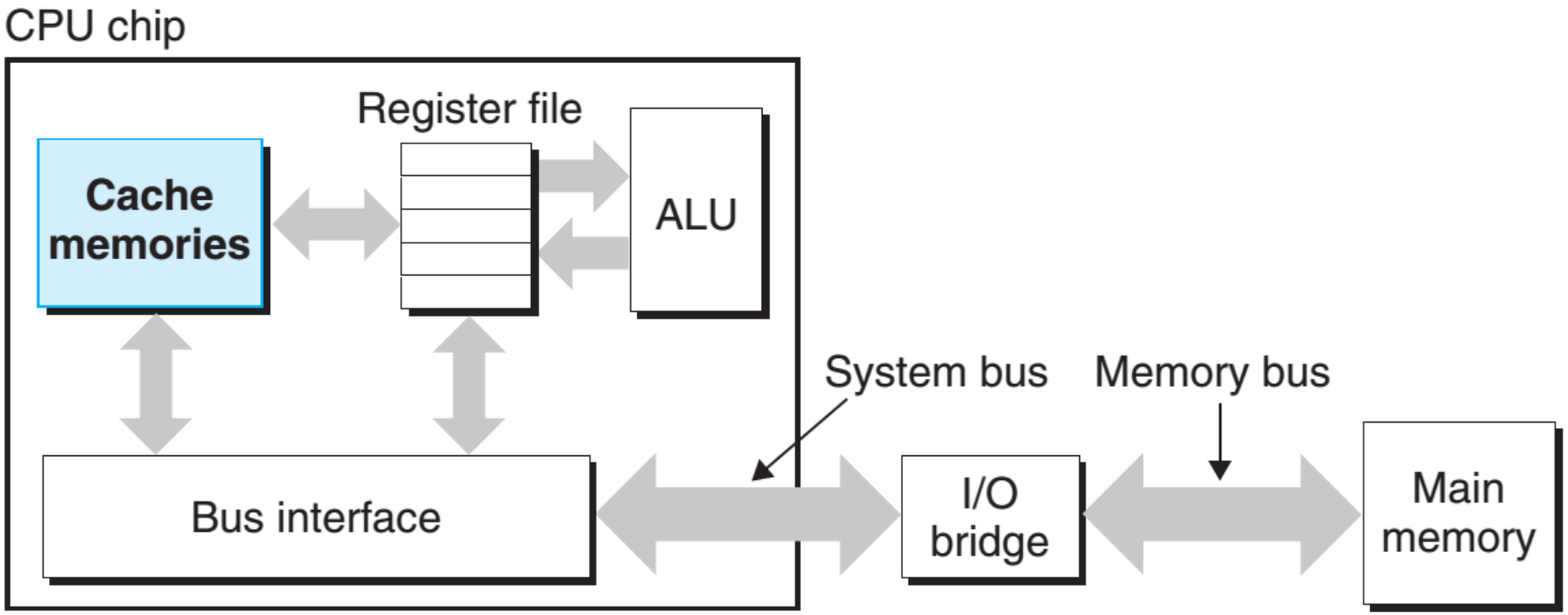
Ví dụ: Xuất “Hello, World” ra màn hình



Kiến trúc phân cấp bộ nhớ



Bộ nhớ Cache


CPU chip
Register file
Cache memories
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory


KEEP
CALM
AND
ENJOY YOUR
SEMESTER :)