Course Overview

Arquitectura de Computadores
Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Luís Nogueira (lmn@isep.ipp.pt)

Overview

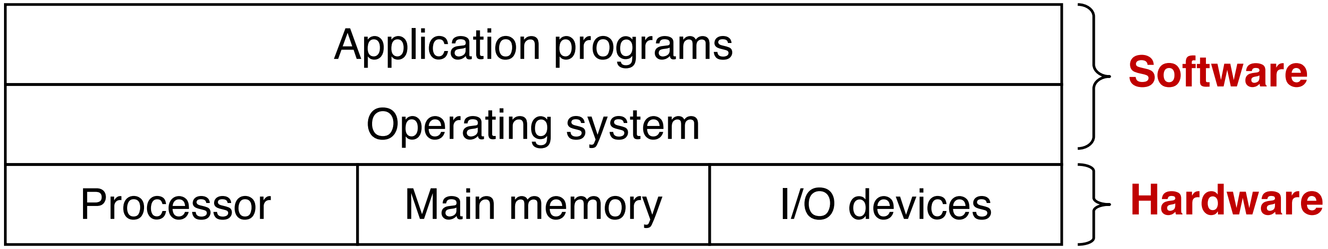
- Course theme
- How it all works
- Main topics
- Logistics

Course Theme

COMPUTER SYSTEMS FROM A PROGRAMMER'S PERSPECTIVE

Course theme

- A *computer system* consists of hardware and system software that work together to run application programs



- How these components work and interact affects the correctness and performance of application programs

Course theme

- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
  - Computational complexity
  - …

- **Abstraction is good but don’t forget reality**
  - These abstractions have limits, especially in the presence of bugs
  - Need to understand details of underlying implementations

Course perspective

- **Our course is programmer-centric**
  - Purpose is to show how by knowing more about the underlying system, one can be more effective as a programmer
- **Enable you to…**
  - Write programs that are more reliable and efficient
  - Incorporate features that require hooks into OS
    - E.g., concurrency, signal handlers
- **Not just a course for dedicated hackers**
  - We bring out the hidden hacker in everyone
- **Cover material in this course that you won't see elsewhere**

Expected outcomes

- **Become a more effective programmer**
  - Able to write better programs by learning what is going on “under the hood” of a computer system
  - Able to find and eliminate bugs efficiently
  - Able to understand and tune for program performance

- **Prepare for more advanced topics in CS & ECE**
  - Compilers, Operating Systems, Concurrent Programming, Networks, Computer Architecture, Embedded Systems

The big picture

HOW IT ALL WORKS

How it all works

- **We begin our study by tracing the lifetime of the *hello* program**
  - Although it is a very simple program, every major part of the system must work in concert for it to run to completion

- **In a sense, the goal of this course is to help you understand what happens and why, when you run *hello* on your system**

```
#include <stdio.h>

int main(){
   printf("hello, world\n");
}
```

The source file

- **Our *hello* program begins its life as a *source file***
  - Created with an editor and saved in a text file called “*hello.c*”

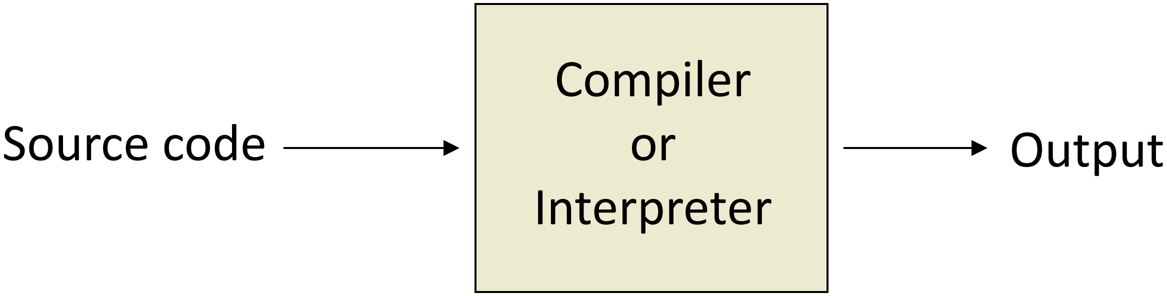
- **Text characters are encoded using the ASCII standard**
  - Represents each character with a unique byte-sized integer value

```
#include <stdio.h>

int main(){
   printf("hello, world\n”);
}
```

```
35 105 110 99 108 117 100 101 32 60 115 116 100 105 111 46 104 62
10 10 105 110 116 32 109 97 105 110 40 41 10 123 10 32 32 32 32
112 114 105 110 116 102 40 34 104 101 108 108 111 44 32 119 111
114 108 100 92 110 34 41 59 10 125
```

The source file

- **How can we run *hello.c* on our system?**
- **We depend on tools such as compilation and interpretation in order to get our written code into a form that the computer can execute**
  - Both compilation and interpretation offer benefits and pitfalls
  - Tradeoff is efficiency vs flexibility

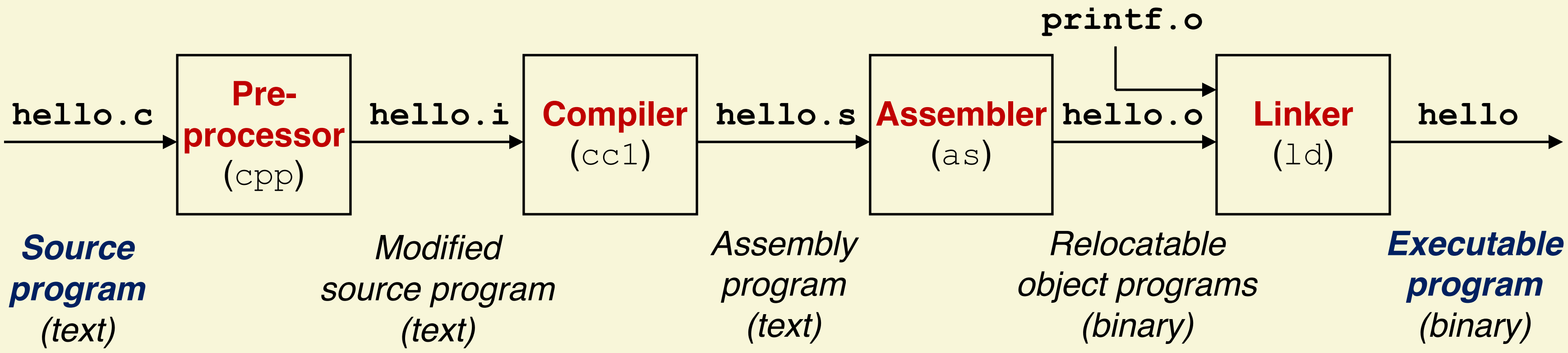


The compilation system

- The individual C statements are translated into a sequence of **low-level machine-language instructions**

- This is actually a multi-step process

```
unix_shell> gcc hello.c -o hello
```



Executing the program

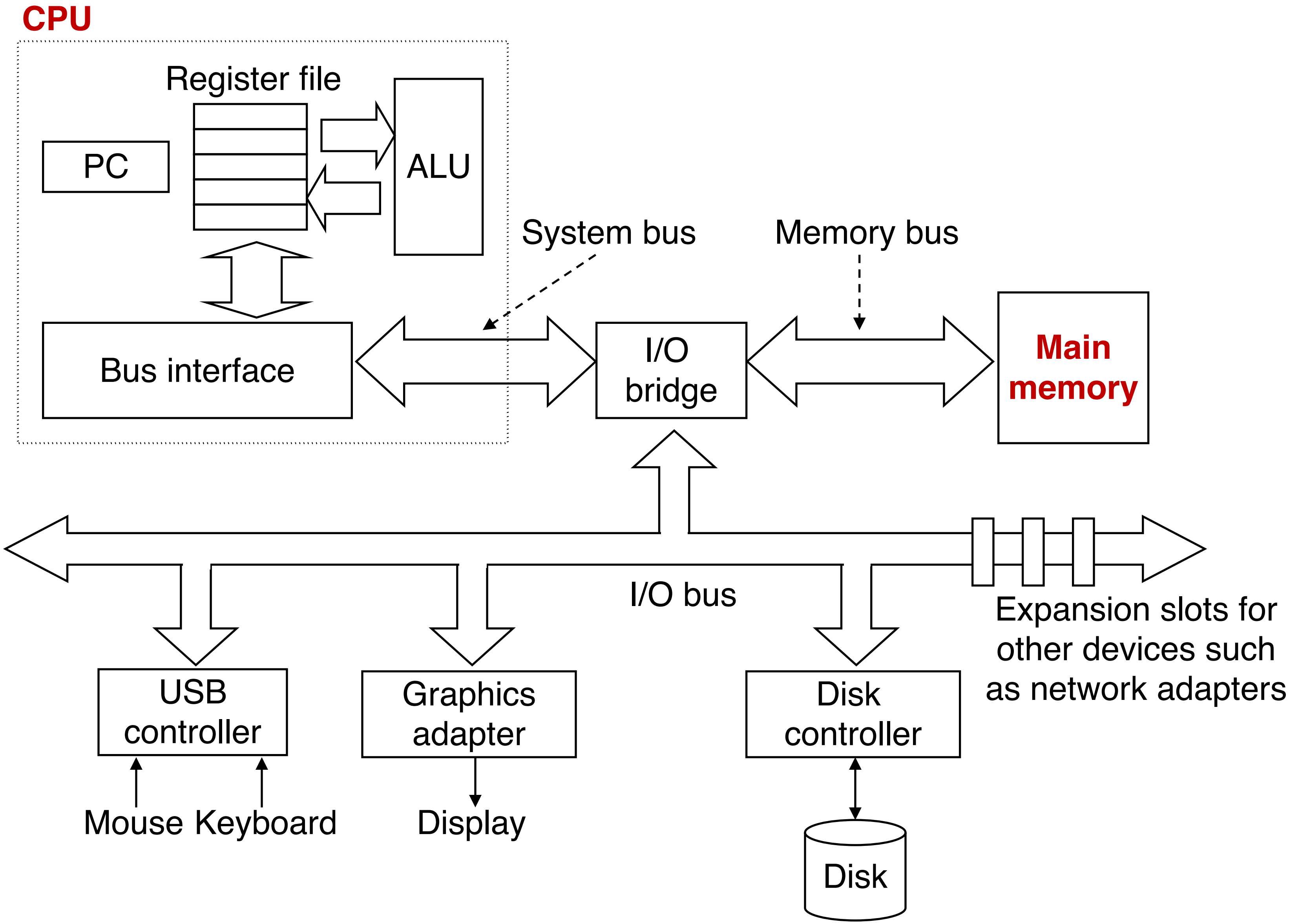
- At this point, our *hello.c* source program has been translated into an ***executable object file*** called *hello* that is stored on disk

- To run the executable file on a Unix system, we type its name to an application program known as a *shell*

```
unix_shell> ./hello

hello, world

unix_shell>
```

To understand what happens, we need to understand the organization of a typical system

Hardware organization of a system



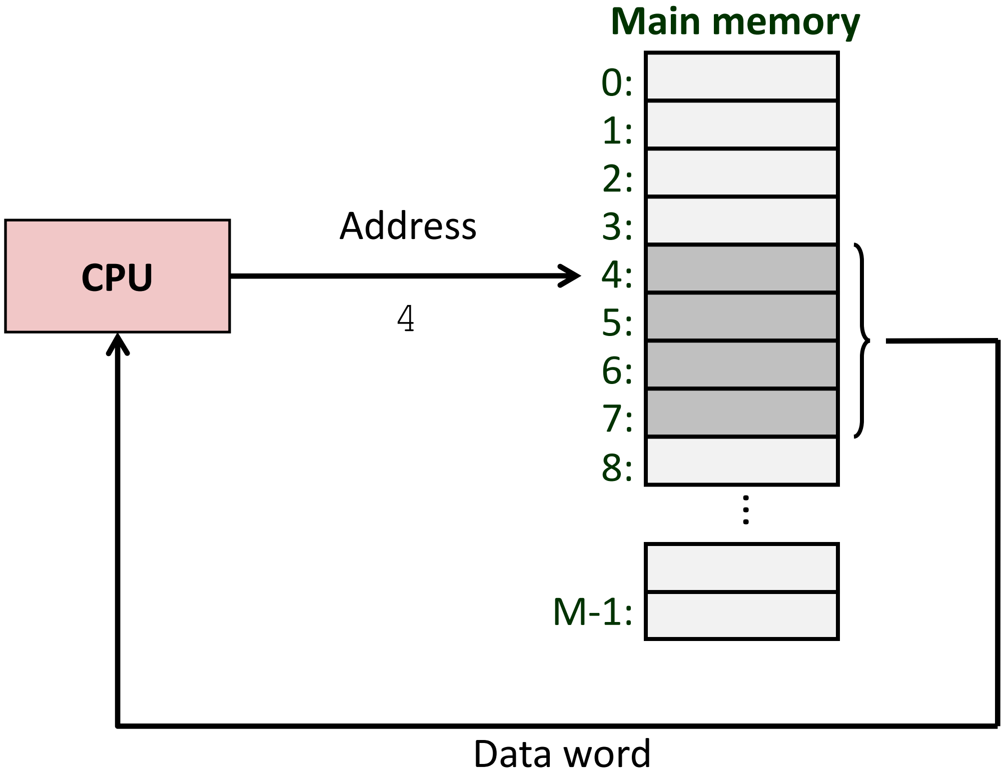
The Central Processing Unit

- **The engine that interprets (or *executes*) instructions stored in main memory**
  - Reads the instruction from memory pointed at by the PC register
  - Interprets the bits in the instruction
  - Performs the operation dictated by the instruction
  - Updates the PC to point to the next instruction

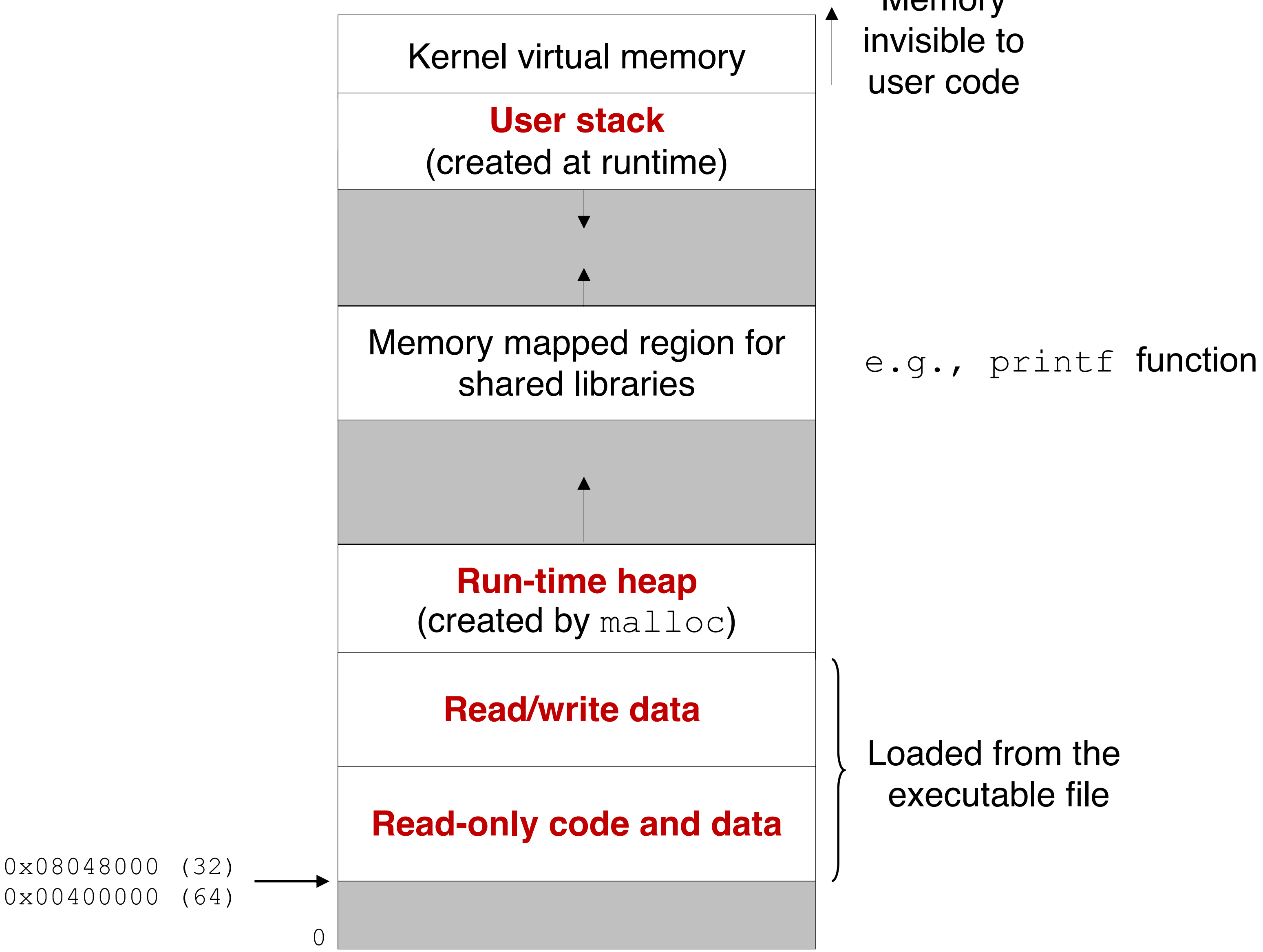
- **The *instruction-set architecture* (ISA)**
  - The set of instructions supported by a particular processor and their byte-level encodings
  - Different "families" of processors, such as Intel IA32, IBM/Freescale PowerPC, and the ARM processor family have different ISAs
  - **A program compiled for one type of machine will not run on another**

Main memory

- Memory is **logically organized as a linear array of bytes**, each with its own unique address, starting at zero



Address space



Course overview

MAIN TOPICS

Abstractions of programming languages

- Cannot assume all usual mathematical properties due to finiteness of representations

- Several surprises to beginning programmers when casting, signed/unsigned, expanding/truncating

```
unsigned int x = 0xDEADBEEF;
short y = 0xFFFF;
int z = -1;

if (x > z)
   printf("Hello");

if (x > y)
   printf("World");
```

You've got to know Assembly

- **Understand the underlying computer system in order to understand its impact on your application programs**

C code

```
int sum(int x, int y)
{
  int t = x+y;

  return t;
}
```

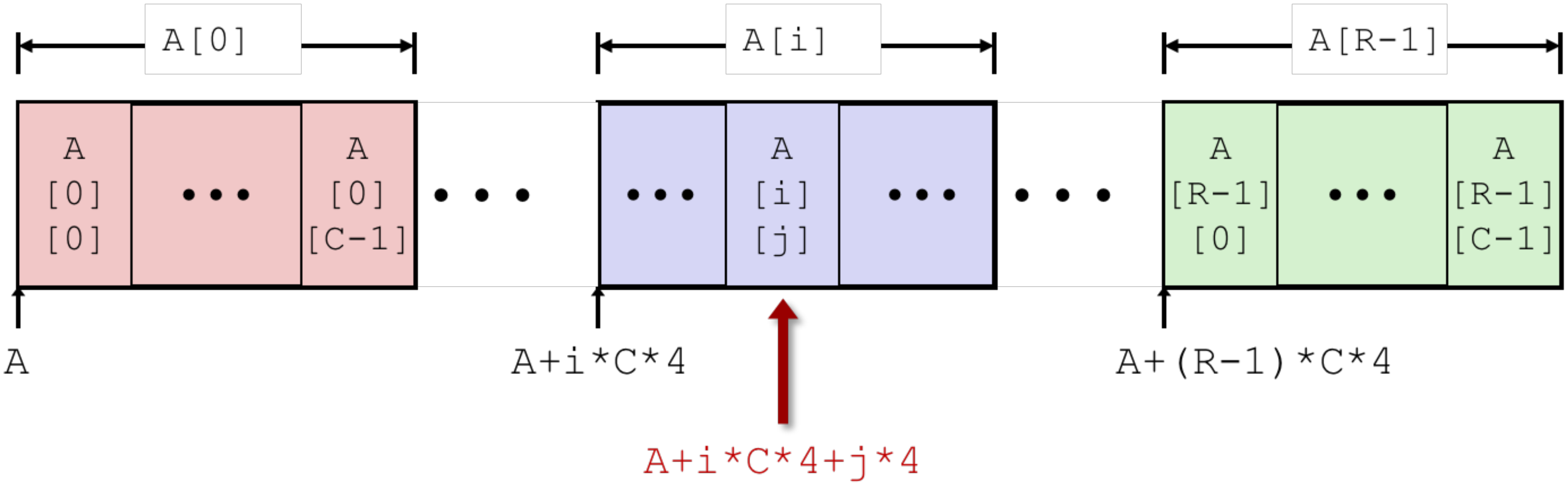
IA32 Assembly

```
sum:
   pushl %ebp                     Set
   movl %esp,%ebp                 Up


   movl 12(%ebp),%eax
   addl 8(%ebp),%eax              Body


   movl %ebp,%esp
   popl %ebp                      Finish
   ret
```

Memory allocation and access

- **Static vs dynamic memory allocation and its implication on data access**
  - Both in C and Assembly

```
int A[R][C];
```



Tuning program performance

- Understand optimizations done/not done by the compiler

- Understand sources of program inefficiency

```
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];
}
```

4.3ms

81.8ms

2.67 GHz Intel Core i7 Haswell

Course perspective

LOGISTICS

Course components

- **Lectures**
  - Higher and lower level concepts of computer systems from a programmer's perspective
- **TP classes**
  - Applied concepts, important tools and skills for labs, clarification of lectures
- **Lab classes**
  - Explore the interactions between C programs and their machine-language counterpart running on Linux systems on top of IA32 architectures
  - Provide in-depth understanding of particular aspects of systems
  - Programming and measurement

Teaching staff

- **Lectures**
  - Luís Nogueira (lmn@isep.ipp.pt)
- **TP classes**
  - Luís Nogueira (lmn@isep.ipp.pt), Raquel Faria (arf@isep.ipp.pt)
- **Lab classes**
  - Luís Nogueira (lmn@isep.ipp.pt), Raquel Faria (arf@isep.ipp.pt)
  - Carlos Gonçalves (cag@isep.ipp.pt), André Andrade (lao@isep.ipp.pt)
  - Paulo Ferreira (pdf@isep.ipp.pt), Marta Fernandes (ald@isep.ipp.pt)
  - André Pedro, Ênio Filho

Textbooks

- **“Computer Systems: A Programmer’s Perspective, 2nd Edition”**, Randal E. Bryant and David R. O’Hallaron, Prentice Hall, 2010

- **“Computer Organization and Design: The Hardware/Software Interface, 5th Edition”**, David A. Patterson and John L. Hennessy, Morgan Kaufmann, 2013

- **“The C Programming Language, Second Edition”**, Brian Kernighan and Dennis Ritchie, Prentice Hall, 1988

- **“Professional Assembly Language”**, Richard Blum, Wrox, 2005

Getting Help

- **Course Web Page: http://moodle.isep.ipp.pt**
  - Copies of lectures, assignments, exams, solutions
  - Message board

- **Staff mailing list**
  - Send email to individual instructors to schedule appointments or getting help

Lab rationale

- **Each module (two weeks) has a well-defined goal**
  - Doing the module's exercises should result in new skills and concepts

- **Work groups of two students**
  - Solutions of lab exercise must be submitted to the group's Bitbucket repository

- **Group computer exercise in the 6th and 10th weeks**
  - Evaluates acquired skills in previous modules
  - Both students must be present and work cooperatively in a solution

- **Individual computer exercise in the 12th week**
  - Sets a reasonable threshold for exam's access
  - All topics can be evaluated in this exercise

Grading

- **Lab classes (60%): minimal grade of 7.5**
  - Weighted according to:
    - Group computer exercises: 40%
    - Individual computer exercise: 20%

- **Final exam (40%): minimal grade of 7.5**
  - Can only attend if Lab's grade is >= 7.5
  - You can only repeat the final exam, keeping all other components their relative weights

- **Supplemental exam**
  - Retake the final exam **and** an additional closed-book exam to replace lab's grade

Welcome
and Enjoy!