

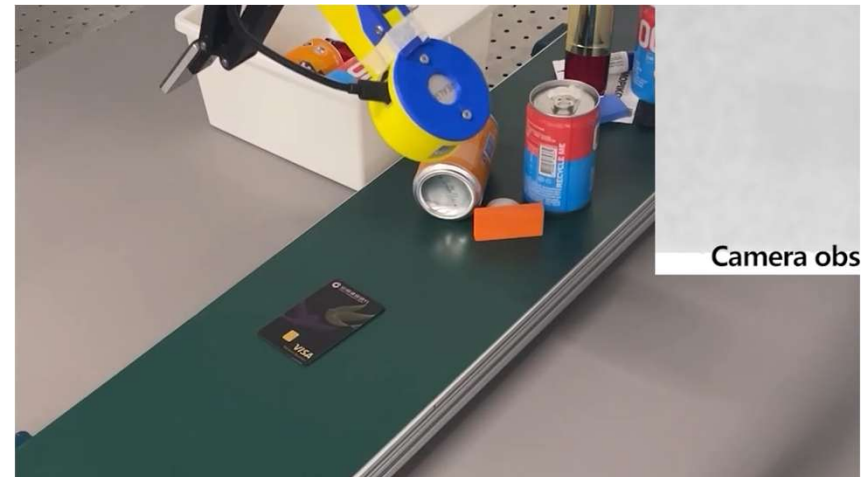
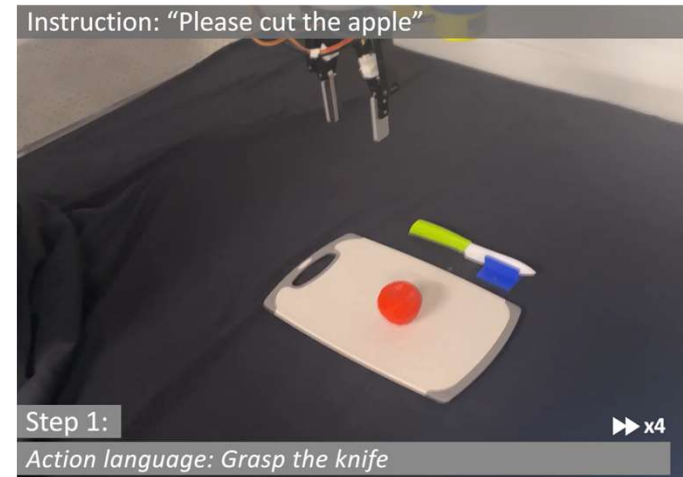
计算机组成与系统结构

赵超

吉林大学人工智能学院

具身智能与机器人实验室

Embodied Intelligence and Robotics Lab (EIRL)



具身智能与机器人实验室

Embodied Intelligence and Robotics Lab (EIRL)

■ 研究方向

- 灵巧手操纵(dexterous hands)、触觉多模态(tactile multimodality)和人形机器人的长期自主(long-term autonomy in humanoid robots)

■ 招生信息

- 每年招收一名博士生和两到三名硕士生，欢迎对科研有浓厚兴趣、能够投入时间的本科生加入团队。
- 动手能力强，曾参与 Robotmaster 优先；能进行机器人系统组装、焊接、3D建模、3D打印，对机械臂熟练操作者更佳；
- 可能30% - 50%时间与硬件打交道，“没有实物实验，就没有真正的机器人学和具身智能”，对实验有吃苦耐劳、严谨认真的态度，若不喜欢与真实机器人打交道，请谨慎考虑。

一些有用的信息

■ 课程网站

- eirl-jlu.github.io/JLU-COA/
 - slides、通知以及作业等

■ Office hour

- Mon 10 - 11 am
- 正新楼603

■ 我的邮箱

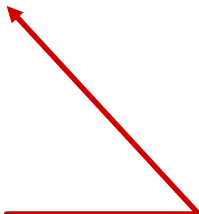
- chaozhao@jlu.edu.cn
- 1:1 Appointments

教材

- 计算机组成与设计：硬件/软件接口（第5版）
 - 作者：David A.Patterson

参考书目：

- 计算机组成与体系结构（第4版）
 - 作者：Linda Null、Julia Lobur



今年课程大纲修订后预计会迁移到这个版本

评价标准

■ 平时20%+作业20%+期末60%

- 4次作业，各5%，晚于Date Due提交会失去成绩
- 平时作业与期末，王湘浩实验班与其他班会有一定区别

■ 五个宽限日

- 作业或者签到
- 用完后，我们不会接受任何延长时限的请求
- 例如：Date Due 5月13日，如使用两个宽限日，5月15日23:59前提交都可以

关于这门课

■ 先修课程

- 一种高级过程编程语言的经验，大学数学

■ 课程目标

- 理解计算机的层次化结构与数据表示
- 掌握指令系统与处理器工作原理理解
- 存储层次与内存管理机制
- 深入理解程序与操作系统的交互。

关于这门课

■ 先修课程

- 一种高级过程编程语言的经验，大学数学

■ 课程目标

- 理解计算机的层次化结构与数据表示
- 掌握指令系统与处理器工作原理理解
- 存储层次与内存管理机制
- 深入理解程序与操作系统的交互。

■ 简而言之：

- 初步涉猎如何设计一台高效的计算机
- 初步涉猎如何让程序在计算机上高效运行

课程内容安排

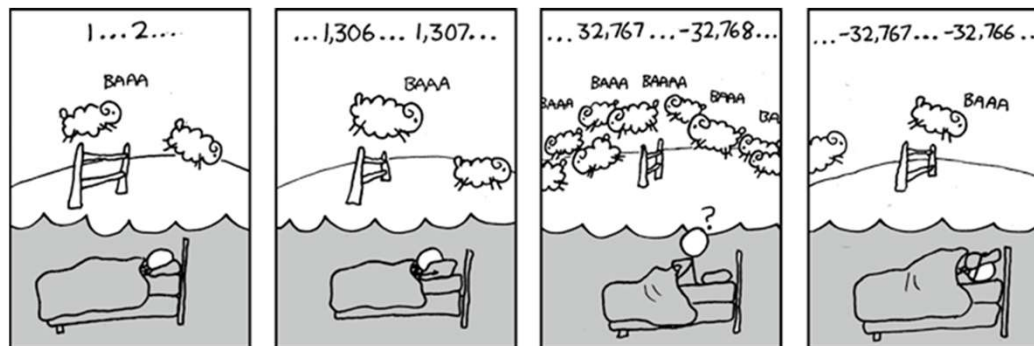
- **Part1:** 系统中的数据表示和处理
- **Part2:** 指令集与处理器体系结构
- **Part3:** 存储系统与内存管理
- **Part4:** 输入输出系统
- **Part5:** 并行与多处理器，网络通信等

核心1:

■ 例子 1: $x^2 \geq 0$ 吗?

■ 整数:

- $40000 * 40000 = 1600000000$
- $50000 * 50000 = ??$
- $300 * 400 * 500 * 600 = ??$

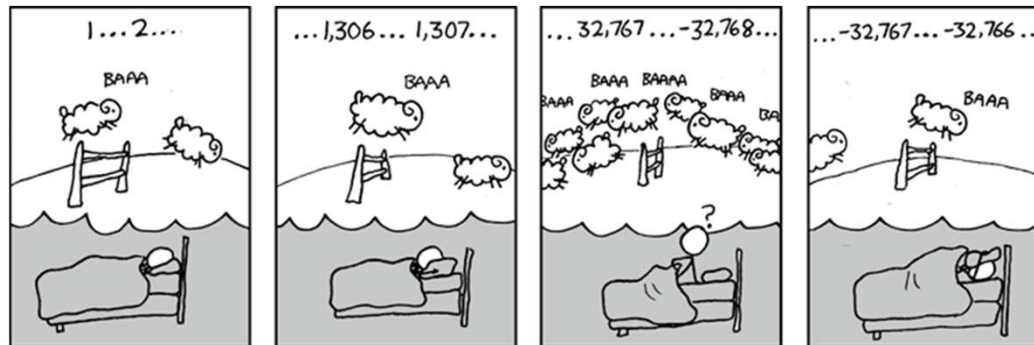


核心1:

■ 例子 1: $x^2 \geq 0$ 吗?

■ 整数:

- $40000 * 40000 = 1600000000$
- $50000 * 50000 = ??$



■ 例子 2: 加法满足结合律吗?

- $(x + y) + z = x + (y + z)?$
- 浮点数:
 - $(1e20 + -1e20) + 3.14 = 3.14$
 - $1e20 + (-1e20 + 3.14) = ??$

计算机算术运算

■ 不会生成随机值

- 算术运算有严格的数学性质，不是随机的结果。

■ 但不能假设它满足所有常见的数学性质

- 原因在于计算机表示的有限性。
- 整数运算：满足“环 (ring)”的代数性质
 - 交换律、结合律、分配律仍然成立
- 浮点运算：只满足“有序性 (ordering)”性质
 - 单调性、符号规则成立，但结合律/分配律可能不成立

■ 启发

- 我们需要理解：在什么情况下这些抽象规律成立，在什么情况下会失效。
- 这是编译器开发者和高级程序员必须认真对待的问题。

核心2：懂一点汇编（assembly）

- 也许，你这一生都不会直接用汇编写程序
 - 现代编译器的优化能力非常强，比你更聪明、更高效、也更有耐心。
- 但是理解汇编是掌握“机器级执行模型”的关键
 - 帮助理解程序在出现 bug 时的真实行为
 - 高级语言的抽象在复杂 bug 面前往往会“失效”，这时只有理解底层汇编，才能看懂计算机实际在做什么。
 - 优化程序性能
 - 了解哪些优化是编译器自动完成的，哪些是编译器不会做的。
 - 理解性能瓶颈的根源。
 - 应对安全与恶意软件
 - x86 汇编都是安全领域的首选语言。

核心3：内存很重要

- 随机访问存储器（RAM）并不是一个真正“无限、均匀”的概念
 - 内存不是无限的
 - 必须经过分配和管理
 - 很多应用程序都是受限于内存的
 - 内存性能并不均匀
 - 缓存和虚拟内存的特性会显著影响程序性能
 - 如果根据内存体系的特点优化程序，性能可能会获得极大提升
 - 内存引用错误尤其棘手
 - 错误的影响跨越时间和空间，可能很久之后才显现

内存引用错误示例

```
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; /* 可能越界访问 */  
    return s.d;  
}
```

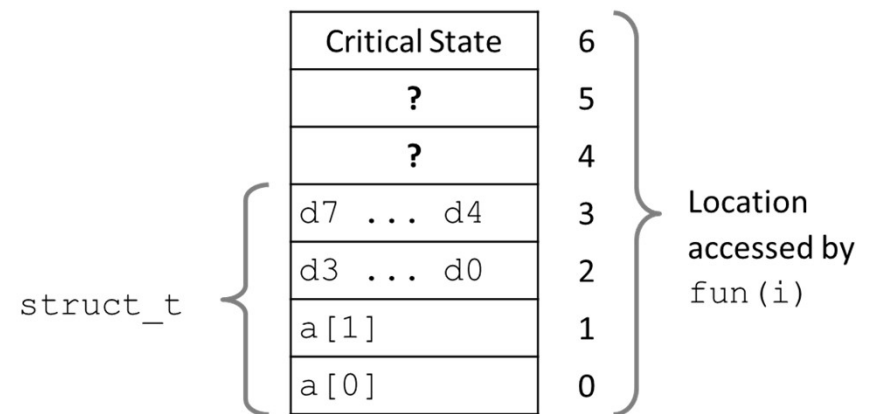
■ 运行结果示例:

```
fun(0) -> 3.14  
fun(1) -> 3.14  
fun(2) -> 3.139998664856  
fun(3) -> 2.00000061035156  
fun(4) -> 3.14  
fun(6) -> 程序崩溃 (Segmentation fault)
```

内存引用错误示例

```
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; /* 可能越界访问 */
    return s.d;
}
```



运行结果示例:

```
fun(0) -> 3.14
fun(1) -> 3.14
fun(2) -> 3.139998664856
fun(3) -> 2.00000061035156
fun(4) -> 3.14
fun(6) -> 程序崩溃 (Segmentation fault)
```

- 结果是系统相关的，不同平台表现可能不同。

内存引用错误总结

■ C / C++ 不提供任何内存保护

- 可能出现数组越界访问
- 无效指针引用

■ 这些问题会导致严重的 Bug

- Bug 的影响常常取决于系统和编译器
- 错误可能出现在“很远的地方”，让人难以定位
- 数据损坏可能在很久之后才表现出来

■ 如何应对？

- 使用带内存保护的語言，例如 Python 等
- 理解底层数据结构的内存布局，预防潜在错误
- 使用工具，如 Valgrind，自动检测内存引用错误

核心4：性能不仅仅取决于算法复杂度

■ 常数因子同样重要！

- 在算法分析中，我们通常关注时间复杂度，但在实际编程中，常数因子也会极大影响性能。

■ 即使精确的操作计数，也不能预测程序性能

- 同样的操作数量，代码写法不同，性能可能相差 10 倍。
- 必须在多个层面上优化：算法、数据表示、函数过程、循环结构等。

■ 必须理解系统，才能真正优化性能

- 了解程序是如何被编译和执行的。
- 学会如何衡量程序性能并识别性能瓶颈。
- 在不破坏代码模块化和通用性的前提下，提升性能。

内存系统性能示例

```
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];
}
```

4.3ms

```
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];
}
```

81.8ms

2.0 GHz Intel Core i7

- 现代计算机采用分层式内存组织。
- 程序性能高度依赖访问模式：
 - 如果访问是连续的，缓存命中率高，性能大幅提升。
 - 如果访问是跳跃的，缓存失效频繁，性能显著下降。
- 多维数组的遍历顺序会直接影响缓存效率。

核心5：计算机的世界远不止于运行程序

■ 计算机需要与外界交换数据

- 数据输入与输出（I/O）是任何计算机系统的核心功能。
- I/O 系统的设计和性能直接影响程序的可靠性和运行速度。
- 如果 I/O 设计不当，即使计算速度再快，也可能被数据传输瓶颈拖慢。

■ 并行与多处理器的挑战

- 现代计算机通常包含多个处理器核心，支持并行计算以提高性能。但并行并不意味着简单地“越多越快”

■ 网络通信

- 在今天的世界里，计算机几乎从不孤立运行，而是通过网络协作完成更复杂的任务。

课程内容安排

- **Part1:** 系统中的数据表示和处理
- **Part2:** 指令集与处理器体系结构
- **Part3:** 存储系统与内存管理
- **Part4:** 输入输出系统
- **Part5:** 并行与多处理器，网络通信等