STAT243 Lecture 1.2 Computer Architecture

Logic

目的：为后续 Shell、编程、性能优化提供硬件与系统层面的上下文。聚焦计算机的关键部件、程序如何运行、文件系统与操作系统抽象。

---

1 Components of Computer Architecture（部件一览）

Logic

以数据在各层之间流动为主线：CPU ↔ Cache ↔ RAM ↔ Storage，经由 Bus 与 Ports 连接外设。

1.1 CPU（Central Processing Unit）

- 职责：执行 **arithmetic / logic / control / I/O** 指令，是计算机的“控制中心”。
- 典型组成：
  - **Control Unit**：负责取指、译码与控制执行流程。
  - **ALU（Arithmetic Logic Unit）**：算术、逻辑、位运算。
  - **Registers**：极少量、超高速的专用/通用寄存器，用于暂存数据与地址。
  - **Cache**：小而快的就近存储，降低访存延迟（见下节）。
  - **MMU（Memory Management Unit）**：配合 OS 管理虚拟内存（有的架构可选/集成）。
  - **Firmware**：上电自检、初始化等固件程序。
- 集成形态：现代 CPU 常以单芯片集成于 **motherboard** 上，与内存与总线协同工作。

1.2 Cache（高速缓存）

- 特性：容量小（KB–MB 级），速度比 RAM 快 10–100 倍；通常紧贴 CPU。
- 作用：自动保留“最近/常用”数据副本；对 Cache 的修改由硬件协议回写到主存。
- 直觉：若数据访问**有局部性**（temporal/spatial locality），Cache 命中率高→性能好。

1.3 Bus（总线）

- 定义：连接 CPU、内存、外设的通信电路；**带宽/延迟**直接影响整机性能。

1.4 Main Memory（RAM）

- 特性：随机访问、容量中等、相对快速；**易失性**（断电即失）。
- 备注：用户常说的“16 GB 内存”即指 RAM。

1.5 External Storage（外部存储）

- 类型：HDD/SSD/U 盘等；容量大、但相对 RAM **非常慢**。
- 物理约束：旋转磁盘与甚至 SSD 都对访问顺序较敏感，顺序 I/O 往往优于随机 I/O。

1.6 Ports（外设端口）

- 含义：与外围设备交互的可寻址接口；具体控制由 **device driver**（设备驱动）实现。

Remark

性能三分法：**compute-bound / memory-bound / I/O-bound**。在数据密集工作负载中，Cache 命中与内存/总线带宽往往比“CPU 峰值算力”更关键。

---

2 How Programs Run（程序如何运行）

Logic

程序本质是机器码指令序列，CPU 通过寄存器与内存协作逐条执行。语言与编译/解释器是在此之上的抽象。

2.1 Machine code 与指令格式

- 指令基本形态：`opcode [operands]`，其中 **opcode** 是硬连线操作编号，**operands** 可能是寄存器名、内存地址或立即数。
- 执行流程：CPU 获得程序入口地址 → 取指 → 译码 → 执行 → 更新状态（寄存器/内存/标志位）。

Example

以 Intel x86 的 Euclid GCD（最大公约数）为例：机器码难读，汇编（assembly）只是在机器码上添加最低限度的助记符与语法。

Asm

```
pushl %ebp
movl  %esp, %ebp
; ... 省略若干指令（比较、条件跳转、减法循环）...
leave
ret
```

近“金属”的编程有助理解寄存器、堆栈、跳转等底层机制，但日常工作更依赖高级语言。

2.2 控制流关键寄存器

- **PC / IP（Program Counter / Instruction Pointer）**：指向下一条待执行指令；遇到分支/跳转/调用/返回会被相应修改。
- **SP（Stack Pointer）**：指向栈区顶端；函数调用时用于暂存参数、返回地址、临时变量等。

2.3 Programs that run other programs（编译/解释）

- **Compiler（编译器）**：如 C/C++ 编译为本机机器码 → 可执行文件。
- **VM（Virtual Machine）**：如 **JVM**，将字节码视为“理想化机器”的机器码，由运行时程序解释/执行。
- **Interpreter（解释器）/Scripting**：如 `python`、`Rscript`、`ruby` 等进程直接读取并执行更高层语言的指令。

Remark

实际系统中常见**混合执行**：JIT、解释与原生扩展（如 C 扩展、Cython、Rcpp）并存，以兼顾开发效率与性能。

---

3 Files and Directories（文件与目录）

- **File**：持久化的数据对象，具备权限与元数据。

- **Directory（Folder）**：包含文件/子目录的特殊文件；文件系统呈树状层级。
- **Path**：唯一地址，例如 `/Users/spock/logic-puzzles/Kolinahr` 表示 `root` → `Users` → `spock` → `logic-puzzles` → 文件 `Kolinahr`。
- GUI（图形界面）以“图标/文件夹”隐喻呈现；CLI/程序中更常使用路径与系统调用直接操作。

Example

`find /Users/spock -name "Kolinahr"` 可在命令行按路径树搜索文件；在 GUI 中则逐层点击目录导航。

---

4 Operating System（操作系统的角色）

Logic

OS 作为“资源管理者 + 抽象提供者”，把复杂硬件细节隐藏在统一的编程接口之下。

- 资源抽象：
  - **Virtual Memory**：给每个进程提供近似“线性、独占”的地址空间；按需将页面在 RAM 与磁盘间换入换出（paging）。
  - **File Abstraction**：逻辑上连续，物理上可能分散，OS 负责映射与调度读写。
  - **Process / Thread**：调度执行、隔离与通信（IPC）。
- 能力暴露：通过 **system calls**（系统调用）请求资源或与外设交互（如打开文件、网络通信、进程控制）。