第 2 讲：操作系统与系统结构和程序设计语言

第二节：从 OS 角度看 RISC-V CPU

向勇、陈渝

清华大学计算机系

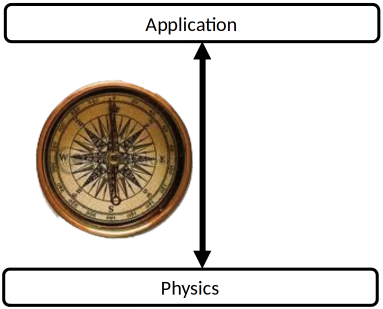
xyong,yuchen@tsinghua.edu.cn

2020 年 2 月 16 日

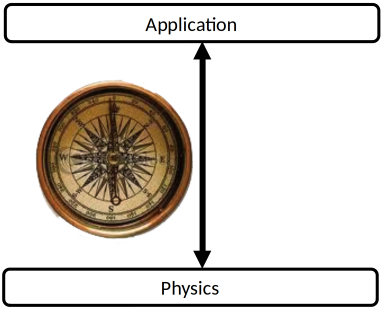
提纲

1. 第二节：从 OS 角度看 RISC-V CPU
   - Why RISC-V
   - RISC-V 特权架构

Why RISC-V： 计算机系统 [CS−152Berkeley]

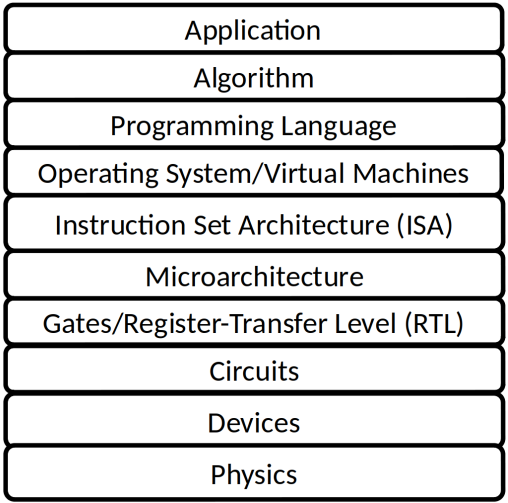


Why RISC-V： 计算机系统 [CS−152Berkeley]

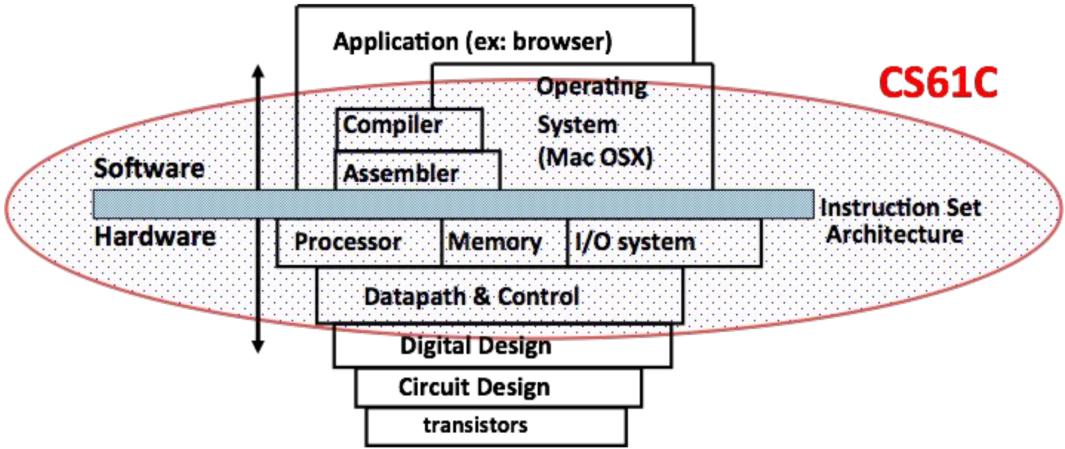


广义的定义, 计算机系统（computer architecture）是一种抽象层次的设计，用于实现可有效使用现有制造技术的信息处理应用。[CS−152Berkeley]

Why RISC-V：计算机系统的抽象层次 [CS−152Berkeley]



Why RISC-V： 软硬件接口 [CS−152Berkeley]

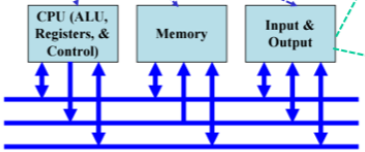


Why RISC-V：OS 与体系结构的关系

Kernel

- **Process**
- **Memory**
- **Device**
- **File System**
- **…**

- Memory
- ALU
- Control Unit
- I/O System



Why RISC-V： 主流 CPU 指令集比较 [*Waterman*2017]

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

ARM architectures

Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985; 31 years ago
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

RISC-V

Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010
Version	2.2
Design	RISC
Type	Load-store
Encoding	Variable
Branching	Compare-and-branch
Endianness	Little

Why RISC-V： 主流 CPU 指令集比较 [*Waterman*2017]

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

ARM architectures

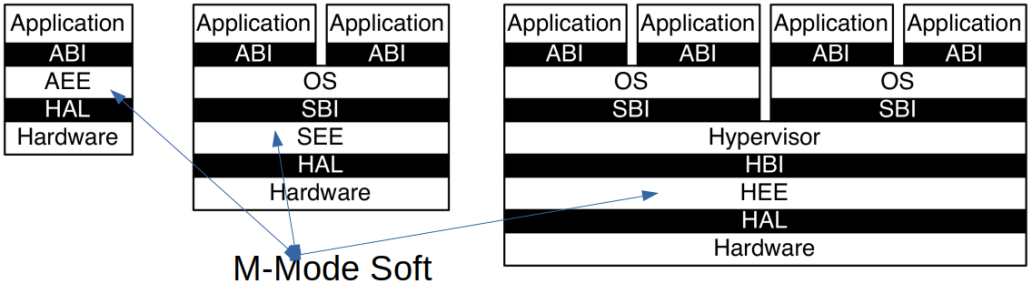
Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985; 31 years ago
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

RISC-V

Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010
Version	2.2
Design	RISC
Type	Load-store
Encoding	Variable
Branching	Compare-and-branch
Endianness	Little

ISA	Pages	Words	Hours to read	Weeks to read
RISC-V	236	76,702	6	0.2
ARM-32	2736	895,032	79	1.9
x86-32	2198	2,186,259	182	4.5

RISC-V 特权架构：隔离



- 不同软件层有清晰的硬件隔离
- AEE：Application Execution Environment
- ABI：Application Bianry Interface
- **MODE – U**：User | **S**: Supervisor | **H**: Hypervisor | **M**: Machine

RISC-V 特权模式架构：模式组合

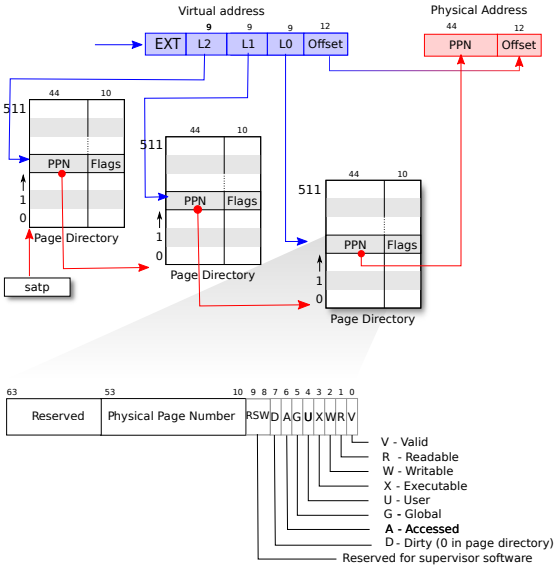
0	00	User/Application	U
1	01	Supervisor	S
2	10	Hypervisor	H
3	11	Machine	M

- M, S, U for systems running Unix-like general operating systems

RISC-V 特权模式架构：控制状态寄存器

- 设置 CSR(控制状态寄存器) 实现隔离
  - 防止应用程序访问设备和敏感的 CPU 寄存器
  - 例如地址空间配置寄存器
- 强制隔离以避免对整个系统的可用性/可靠性/安全影响
  - 运行的程序通常是隔离的单元
  - 防止恶意程序、病毒、木马破坏或监视应用程序或干扰操作系统
    - 读/写内存: mstatus/satp CSR
    - 使用 100%的 CPU：mstatus/stvec CSR
    - mstatus/satp/stvec CSR 页表异常处理

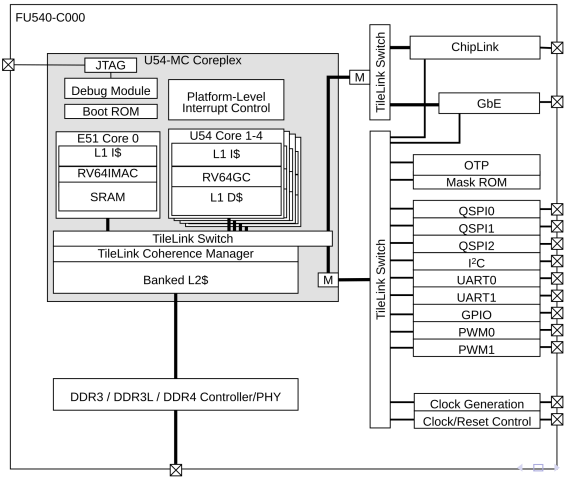
RISC-V 虚拟内存



- 有虚拟内存的好处/坏处
  - 灵活的不连续内存分配
  - 多个运行程序的地址空间相互隔离/共享
  - 进一步隔离应用程序与 OS 内核
  - **多了访问页表的开销**

RISC-V 中断机制

- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
- Timer 可以稳定定时地产生中断
  - 防止应用程序死占着 CPU 不放, 让 OS Kernel 能得到执行权...



RISC-V 中断机制

- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
- Timer 可以稳定定时地产生中断
  - 防止应用程序死占着 CPU 不放, 让 OS Kernel 能得到执行权...
  - 由高特权模式下的软件获得 CPU 控制权
  - 也可由高特权模式下的软件授权低特权模式软件处理中断