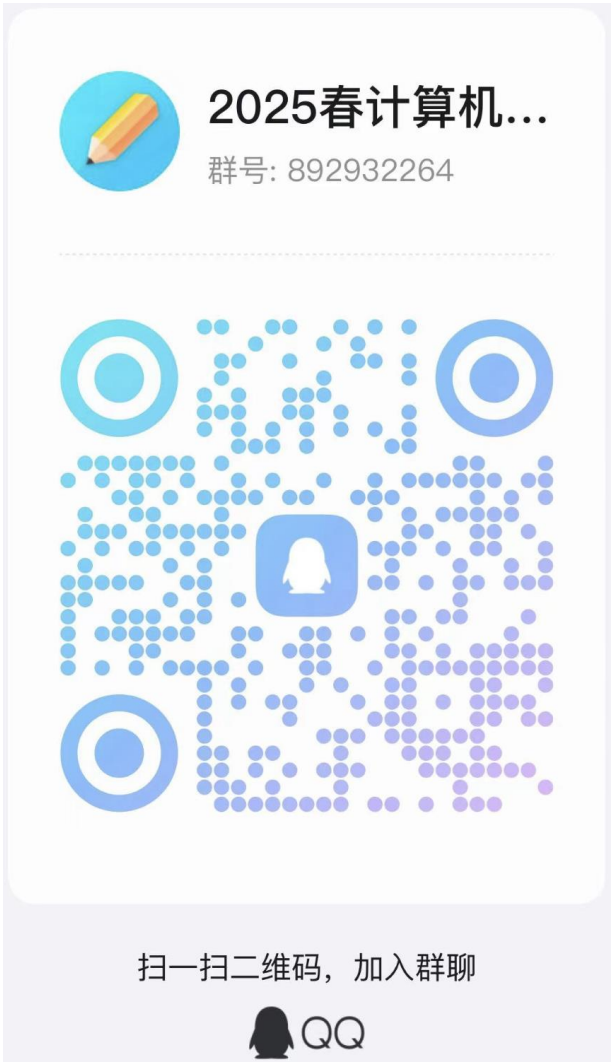
计算机组成原理



花忠云
https://huazhongyun.github.io/
http://faculty.hitsz.edu.cn/huazhongyun

计算机科学与技术学院

第三章 RISC-V汇编及其指令系统

- **RISC-V概述**
- RISC-V汇编语言
- RISC-V指令表示
- 案例分析

第三章 RISC-V汇编及其指令系统

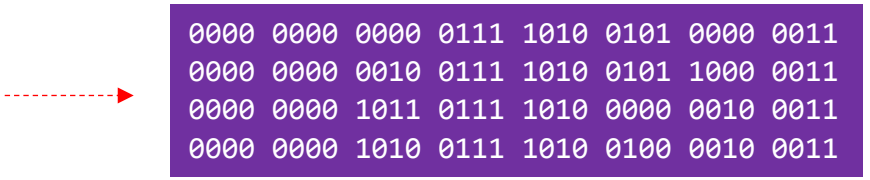
- **RISC-V概述**
  - **指令系统的基本概念**
  - 主流指令集及发展方向
  - RISC-V指令集

指令系统基本概念

- **机器指令（指令）**
  - 计算机能直接识别、执行的某种操作命令，它是一串二进制代码
  - 例如： add x1, x2, x3

    sub x1, x2, x3

    addi x3, x4, -10



- **指令系统（指令集，IS: Instruction Set）**
  - 一台计算机中所有机器指令的集合

指令系统基本概念

- **指令集系统架构（ISA: Instruction Set Architecture）**
  - **简称“架构”，也可称为：处理器架构、指令集体系结构**
  - 包含了程序员正确编写二进制机器语言程序所需的全部信息。
  - 例如：如何使用硬件、指令格式，操作种类、操作数所能存放的寄存器组和结构，包括每个寄存器名称、编号、长度和用途等。

- **系列机**
  - 基本指令系统相同，基本系统结构相同的计算机
  - 解决软件兼容的问题。**给定一个ISA，可以有不同的实现方式**；例如AMD/Intel CPU 都是X86-64指令集。ARM ISA 也有不同的实现方式
  - IBM 360 是**第一个**将ISA与其实现分离的系列机

指令集架构

功能

数据类型

存储模型

软件可见的处理器状态

- 通用寄存器、PC
- 处理器状态

指令集

- 指令类型与编码
- 寻址模式
- 数据结构

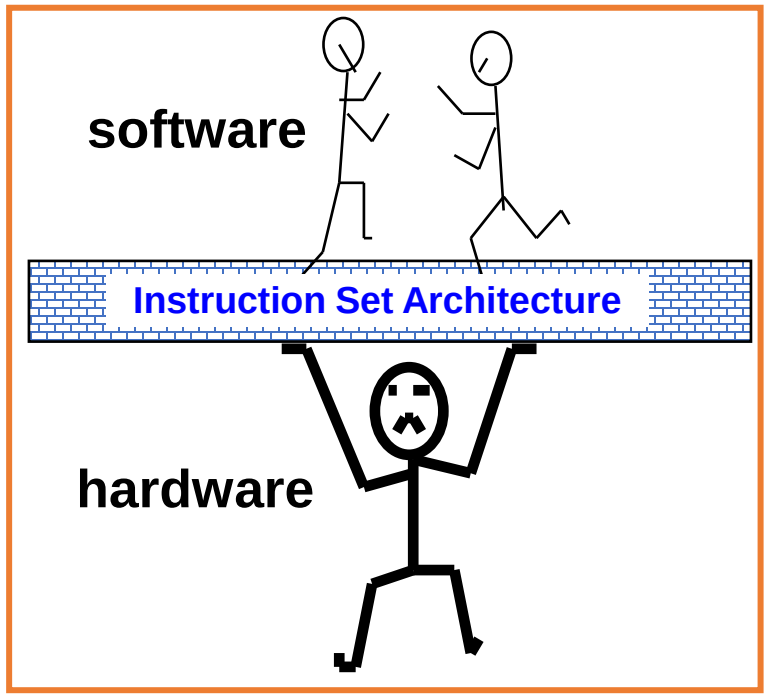
系统模型

- 状态、特权级别
- 中断和异常

外部接口

- 输入/输出接口
- 管理

ISA——抽象层，软件子系统与硬件子系统的桥梁和接口



特性

成本和资源占用低

简洁性：指令规整简洁；

架构和具体实现分离

- 可持续多代，向后兼容

可扩展空间

- 用户可根据应用领域灵活扩展(桌面、服务器、 嵌入式应用)

易于编程/编译/链接

- 为高层软件的设计与开发提供方便的功能

性能好

- 方便低层硬件子系统高效实现

指令集架构(ISA)位宽

- **ISA位宽**：指通用寄存器的宽度，决定了寻址范围的大小、数据运算能力的强弱
- ISA位宽和指令编码长度不一定相等：即便在64位架构中，也大量存在16位编码的指令，且基本上很少出现过64位长的指令编码。

不考虑实际成本和实现技术的前提下

ISA位宽

越大越好

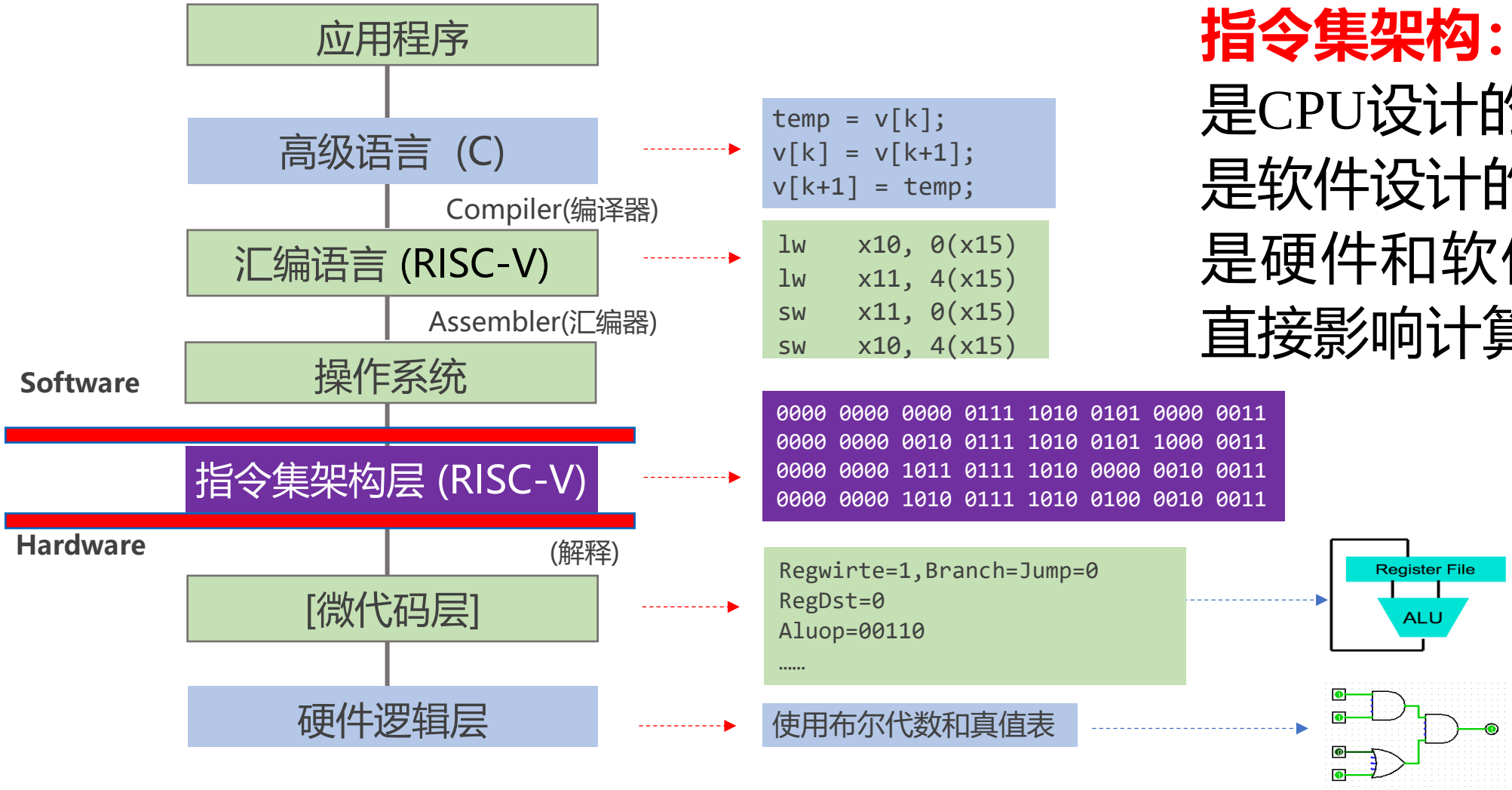
寻址范围更大
运算能力更强

指令编码长度

越短越好

节省代码存储空间

指令集架构层次



指令集架构：

是CPU设计的依据、

是软件设计的基础、

是硬件和软件间的分界面、

直接影响计算机系统性能。

指令系统的评价

- **指令系统的评价**
  - 方便硬件设计，方便编译器实现，性能更优，成本功耗更低
- **硬件设计四原则**
  - 简单性来自规则性（Simplicity favors regularity）
    - 指令越规整，设计越简单
  - 越小越快 （Smaller is faster）
  - 加快经常性事件（Make the common case fast）
  - 好的设计需要适度的折衷 (Good design demands good compromises)

指令系统的评价——续

- **指令系统的评价**
  - 方便硬件设计，方便编译器实现，性能更优，成本功耗更低
- **性能要求**
  - 完备性：指令丰富，功能齐全，使用方便
  - 高效性：程序占空间小，执行速度快
  - 规整性：RISC-V指令长度是32位和16位的压缩指令

    （关于规整性，X86中还包括对称性、匀齐性、一致性的定义）
  - 兼容性：系列机软件向上兼容

有关ISA的若干问题

- 存储器寻址
- 操作数的类型与大小
- 所支持的操作
- 控制转移类指令
- 指令格式

存储器寻址

- 1980年以来几乎所有机器的存储器都是按字节编址
- 一个存储器地址可以访问：
  - 1个字节、2个字节、4个字节、更多字节.....
- 不同体系结构对字的定义是不同的
  - 16位字（Intel X86）、32位字（MIPS、RISC-V）
- 如何读多个字节，例如：32位字
  - 每次一个字节，四次完成；每次一个字，一次完成
- 如何将字节地址映射到字地址 (**尾端问题**)
- 一个字是否可以存放在任何字节边界上(**对齐问题**)

尾端问题 （小端little endian **vs** 大端big endian）

在一个（双）字内部的字节顺序问题

高　　低

例如：设地址addr存储的字为 0x89ABCDEF，addr，addr+1，addr+2，addr+3四个字节存放的分别是什么数据？

地址

	addr+3	addr+2	addr+1	addr
大端(字地址为高字节地址)	**EF**	**CD**	**AB**	**89**
小端(字地址为低字节地址)	**89**	**AB**	**CD**	**EF**

大端(MIPS, IBM360...)

小端(Intel 80x86, RISC V...)

填空题 4分

设置

假设机器字长是32位，数据按大端模式存储，设地址addr存储的字为0x11223344， 则addr，addr+1，addr+2，addr+3四个字节存放的分别是 [填空1] ， [填空2] ， [填空3] ， [填空4] 。

正常使用填空题需3.0以上版本雨课堂

作答

对齐问题

- 假设对s个字节长的对象访问地址为A，如果A mod s =0称为边界对齐。
- 边界对齐的原因是存储器本身读写的要求，存储器本身读写通常就是边界对齐的，对于不是边界对齐的对象的访问可能要导致存储器的两次访问，然后再拼接出所需要的数。（或发生异常）
- RISC-V和x86没有对齐要求，但是MIPS有对齐要求
  - 即要求字的起始地址必须是4的倍数，而双字的起始地址必须是8的倍数。对齐限制会使数据传输更快

对齐问题

Address mod 8	0	1	2	3	4	5	6	7
Byte	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 Bytes	Aligned		Aligned		Aligned		Aligned	
2 Bytes		Misaligned		Misaligned		Misaligned		Misalign
4 Bytes	Aligned				Aligned			
4 Bytes		Misaligned				Misaligned		
4 Bytes			Misaligned				Misaligned	
4 Bytes				Misaligned				Misalign
8 Bytes	Aligned							
8 Bytes		Misaligned						

寻址方式

- **寻址方式：**通过指令中的操作数（不同方式）计算出地址
- **有效地址：**由寻址方式说明的某一存储单元的实际存储器地址。有效地址vs. 物理地址

寄存器寻址

立即数寻址

寄存器间接寻址

带偏移量的间接寻址

绝对寻址

相对基址变址寻址

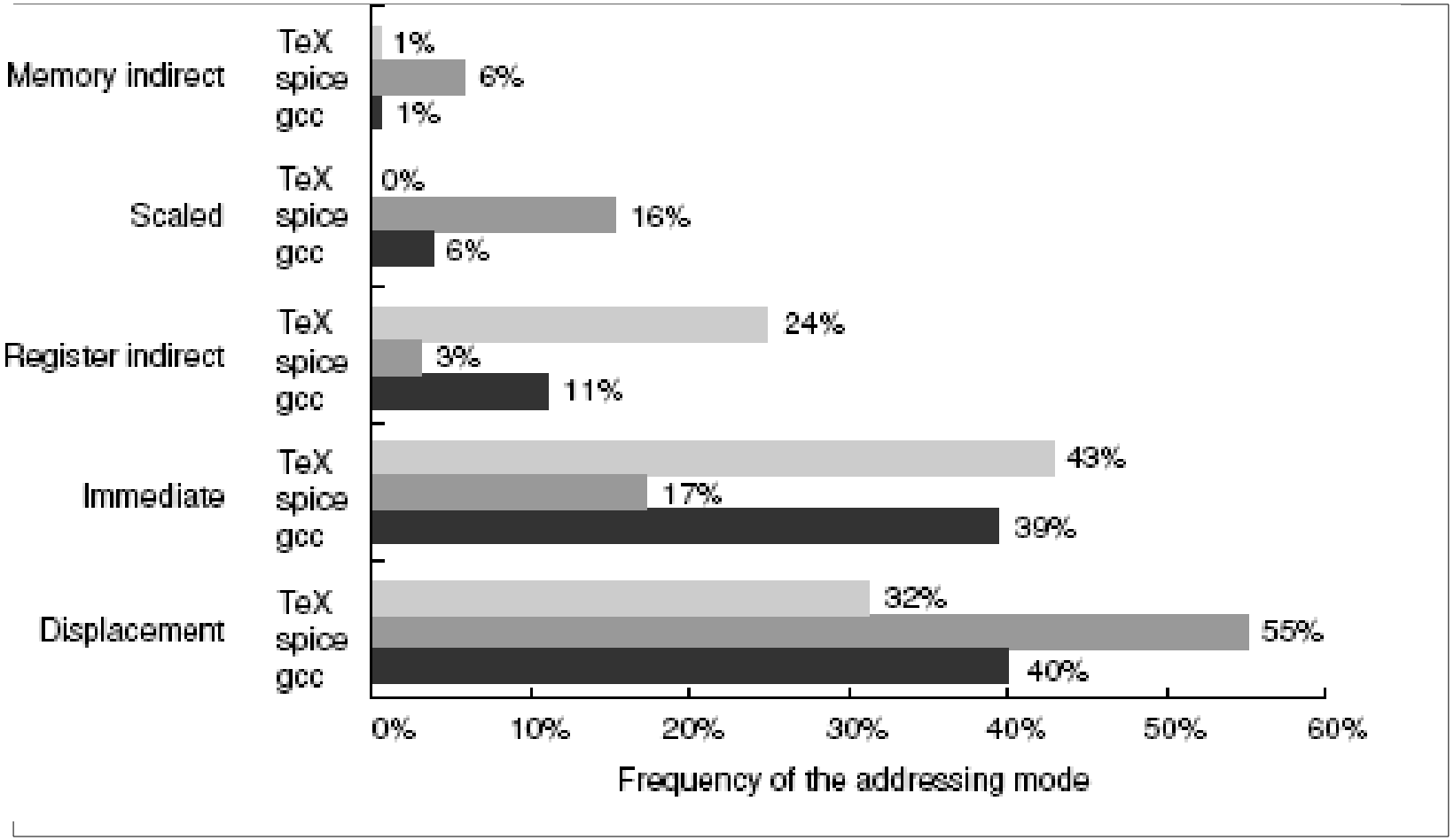
比例变址寻址

后增寄存器间接寻址

前增寄存器间接寻址

Mode	Example	Meaning	When used
Register	Add R1, R2	R1 ← R1 + R2	Values in registers
Immediate	Add R1, 100	R1 ← R1 + 100	For constants
Register Indirect	Add R1, (R2)	R1 ← R1 + Mem(R2)	R2 contains address
Displacement	Add R1, (R2+16)	R1 ← R1 + Mem(R2+16)	Address local variables
Absolute	Add R1, (1000)	R1 ← R1 + Mem(1000)	Address static data
Indexed	Add R1, (R2+R3)	R1 ← R1 + Mem(R2+R3)	R2=base, R3=index
Scaled Index	Add R1, (R2+s*R3)	R1 ← R1 + Mem(R2 + s*R3)	s = scale factor = 2, 4, or 8
Post-increment	Add R1, (R2)+	R1 ← R1 + Mem(R2) R2 ← R2 + s	Stepping through array s = element size
Pre-decrement	Add R1, -(R2)	R2 ← R2 – s R1 ← R1 + Mem(R2)	Stepping through array s = element size

各种寻址方式的使用频率



三个SPEC89程序(详见黑书1.9)在VAX结构上的测试结果：

立即寻址，偏移寻址（**带偏移量的间接寻址**）使用较多

操作数的类型、表示

- **操作数类型**：面向应用、软件系统所处理的各种数据类型
  - 整型、浮点型、字符、字符串、向量类型等
  - 类型**由操作码确定**或**数据附加硬件解释的标记（现已不用）**
- 操作数在机器中的**表示**：硬件结构能够识别，指令系统可以直接使用的表示格式
  - 整型：原码、反码、补码、移码
  - 浮点：IEEE 754标准
  - 十进制：BCD码（二进制十进制表示）

常用操作数类型

- ASCII character = 1 byte (64位寄存器能存8个ASCII字符）
- Unicode character or Short integer = 2 bytes = 16 bits （half word)
- 32位=4字节（字）
  - Integer (很多RISC处理器上字的大小)、Single-precision float(单精度浮点)、**long int(Windows)**
- 64位=8字节（双字）
  - Long long int、Double-precision float(双精度浮点)、pointer(指针)、**long int（linux)**
- Extended-precision float = 10 bytes = 80 bits (Intel architecture)
- Quad-precision float(四精度浮点) = 16 bytes = 128 bits

第三章 RISC-V汇编及其指令系统

- **RISC-V概述**
  - 指令系统的基本概念
  - 主流指令集及发展方向
  - RISC-V指令集

指令集架构：CISC & RISC

CISC

- 复杂指令系统计算机
  (Complex Instruction Set Computer)
- **指令数目多**：含有处理器常用和不常用的特殊指令

RISC

- 精简指令系统计算机
  (Reduced Instruction Set Computer)
- **指令数目少**：仅含有处理器常用指令；对于不常用的指令，通过执行多条常用指令的方式实现

指令集体系结构 (ISA)

- 不同类型的CPU执行不同指令集，ISA是设计CPU的依据

1970 DEC PDP-11 1992 ALPHA(64位)

1978 **x86**，2001 IA64

1980 PowerPC

1981 **MIPS**

1985 SPARC

1991 **ARM**

2010 **RISC-V**

国内主流指令集

- MIPS：龙芯等
  - Microprocessor without Interlocked Piped Stages
  - 无内部互锁流水级的微处理器
- X86：兆芯（VIA），海光（AMD）等
- ARM：飞腾，海思，展讯，松果等
  - ARM处理器是英国Acorn有限公司设计的低功耗成本的第一款RISC微处理器。全称为**A**dvanced **R**ISC **M**achine。
- 自主指令:申威 (Alpha指令集扩展)、龙芯LoongArch等
- RISC-V：阿里-平头哥，华米科技等

典型应用场景

服务器

- Intel公司X86架构的高性能CPU占垄断地位
- ARM服务器已经进入该领域（华为鲲鹏处理器）

桌面个人计算机

- Intel或AMD公司X86架构的高性能CPU占垄断地位
- ARM服务器已经进入该领域

嵌入式移动设备

ARM Cortex-A架构占垄断地位

嵌入式实时设备

深嵌入式

ARM 架构占最大份额，其他RISC架构嵌入式CPU也有应用

RISC的定义和特点

- **RISC是一种计算机体系结构的设计思想，不是一种产品。**
  - 直到现在，RISC没有一个确切的定义
- **是近代计算机体系结构发展史中的一个里程碑**
- **早期对RISC特点的描述**
  - 大多数指令在单周期内完成、采用Load/Store结构、硬布线控制逻辑
  - 减少指令和寻址方式的种类、固定的指令格式
  - 注重代码的优化
- **从目前的发展看，RISC体系结构还应具有如下特点：**
  - 面向寄存器结构
  - 十分重视流水线的执行效率－尽量减少断流
  - 重视优化编译技术
- **减少指令平均执行周期数是RISC思想的精华**

精简指令系统(RISC)

- 指令条数少，保留使用频率最高的简单指令，指令定长
  - 便于硬件实现，用软件实现复杂指令功能。
- Load/Store架构：只有存/取数指令才能访问存储器，其余指令的操作都在寄存器之间进行，便于硬件实现。
- 指令长度固定，指令格式简单、寻址方式简单，便于硬件实现。
- CPU设置大量寄存器（32~192），便于编译器实现
- RISC CPU采用硬布线控制（而CISC采用微程序控制）
- 一个时钟周期完成一条机器指令（单周期模型）。

OpenRISC

- 是一个开放源码处理器设计项目
  - 适用于教学、科研和工业界的实现（Hennessy和Patterson）。
- 主要问题
  - 主要是开源处理器设计项目，而不是开源的ISA 规格说明，**ISA和实现是紧密耦合的**
  - 固定的32位编码与16位立即数阻碍了压缩ISA扩展
  - 硬件不支持IEEE 754-2008标准
  - 用于分支和条件转移的条件码使高性能实现复杂化
  - ISA对位置无关的寻址方式支持较弱
  - OpenRISC不利于虚拟化。从异常返回的指令L.RFE，定义为在用户模式下功能，而不是捕获
  - 值得一提的是：2010年这两个问题都得到了解决:延迟插槽已经成为可选的，64位版本已经定义(但是，据我们所知，从未实现过)。
  - 最终，我们（UCB）认为最好从头开始，而不是相应地修改OpenRISC。

指令集架构：CISC & RISC

- ISA的功能设计
  - 任务：确定硬件支持哪些操作
  - 方法：统计的方法
- CISC（Complex Instruction Set Computer）
  - 目标：**强化指令功能**，减少运行的指令条数，**提高系统性能**
  - 方法：面向目标程序的优化，面向高级语言和编译器的优化
- RISC（Reduced Instruction Set Computer）
  - 目标：通过**简化指令系统**，用**高效**的方法实现**最常用的指令**
  - 方法：充分发挥流水线的效率，降低（优化）CPI

问题

RISC的指令系统精简了，CISC中的一条指令实现的功能可能由多条RISC指令才能完成，那么为什么RISC执行程序的速度比CISC还要快？（执行时间=CPI*IC*T）

	IC	CPI	T
CISC	**1**	**2~15**	**33ns~5ns**
RISC	**1.3~1.4**	**1.1~1.4**	**10ns~2ns**

IC：指令条数，实际统计：RISC的IC只比CISC 长30％~40%
CPI: CISC中一般在为4~6，RISC中一般为1 , Load/Store 为2
T: RISC采用硬布线逻辑，指令要完成的功能比较简单

RISC为什么会减少CPI

- 硬件方面：
  - 硬布线控制逻辑
  - 减少指令和寻址方式的种类
  - 使用固定格式
  - 采用Load/Store
  - 指令执行过程中设置多级流水线。
- 软件方面：十分强调优化编译的作用

指令系统发展方向（CISC-RISC）

- CISC—复杂指令集计算机(Complex Instruction Set Computer)
  - 指令数量多，指令功能复杂，几百条指令。
  - 每条指令都有对应的电路设计，CPU电路设计复杂，功耗较大。
  - 对应编译器的设计简单（各种操作都有对应的指令）。
  - Intel x86
- RISC---精简指令集计算机(Reduced Instruction Set Computer)
  - 指令数量少，指令功能单一，通常只有几十条指令。
  - CPU设计相对简单，功耗较小。
  - 编译器的设计比较复杂（许多操作需要一些指令的灵活组合）
  - 1982年后的指令系统基本都是RISC
  - ARM、MIPS、RISC-V
- CISC、 RISC互相融合

第三章 RISC-V汇编及其指令系统

- **RISC-V概述**
  - 指令系统的基本概念
  - 主流指令集及发展方向
  - **RISC-V指令集**

RISC-V指令集历史

- 加州大学伯克利分校Krste Asanovic教授、Andrew Waterman和Yunsup Lee等开发人员于2010年发明。
  - 其中"RISC"表示精简指令集，而其中"V"表示伯克利分校从RISC I开始设计的第五代指令集。
- 基于BSD协议许可的免费开放的指令集架构
- 适合多层次计算机系统
  - 从 微控制器 到 超级计算机
  - 支持大量定制与加速功能
  - 32bit, 64bit, 128bit
- 规范由RISC-V非营利性基金会维护
  - RISC-V基金会负责维护RSIC-V指令集标准手册与架构文档

RISC-V ISA设计理念

- 通用的ISA
  - 能适应从最袖珍的嵌入式控制器，到最快的高性能计算机等各种规模的处理器。
  - 能兼容各种流行的软件栈和编程语言。
  - 适应所有实现技术，包括现场可编程门阵列（FPGA）、专用集成电路（ASIC）、全定制芯片，甚至未来的技术。
  - 对所有微体系结构实现方式都有效。例如：
    - 微编码或硬连线控制；顺序或乱序执行流水线；单发射或超标量等等。
  - 支持广泛的定制化，成为定制加速器的基础。随着摩尔定律的消退，加速器的重要性日益提高。
  - 基础的指令集架构是稳定的。不能像以前的专有指令集架构一样被弃用，例如AMD Am29000、Digital Alpha、Digital VAX、Hewlett Packard PA-RISC、Intel i860、Intel i960、Motorola 88000、以及Zilog Z8000。
  - 完全开源

RISC-V架构的特点

- 指令集架构简单
  - 指令集238页，特权级编程手册135页，其中RV32I只有16页
  - 作为对比，Intel的处理器手册有5000多页
  - 新的体系结构设计吸取了经验和最新的研究成果
  - 指令数量少，基本的RISC-V指令数目仅有40多条，加上其他的模块化扩展指令总共几十条指令。
- 模块化的指令集设计
  - 不同的部分还能以模块化的方式组织在一起
  - ARM的架构分为A、R和M三个系列，分别针对于Application（应用操作系统）、Real-Time（实时）和Embedded（嵌入式）三个领域，彼此之间并不兼容
  - RISC-V嵌入式场景，用户可以选择RV32IC组合的指令集，仅使用Machine Mode（机器模式）；而高性能操作系统场景则可以选择譬如RV32IMFDC的指令集，使用Machine Mode（机器模式）与User Mode（用户模式）两种模式，两种使用方式的共同部分相互兼容

RISC-V的模块化设计

- RISC-V指令集使用模块化的方式进行组织，每个模块使用一个英文字母来表示
- RISC-V最基本也是唯一强制要求实现的指令集部分是由I字母表示的基本整数指令子集，使用该整数指令子集，便能够实现完整的软件编译器
- 其他的指令子集部分均为可选的模块，具有代表性的模块包括M/A/F/D/C
- RISC-V预留了大量的指令编码空间用于用户的自定义扩展，还定义了四条Custom指令可供用户直接使用，每条Custom指令都有几个比特位的子编码空间预留，用户可以直接使用四条Custom指令扩展出几十条自定义的指令。

模块化的RISC-V 指令子集

基本指令集	指令数	描述
RV32I	**47**	32位地址空间与整数指令，支持32个通用整数寄存器
RV32E	**47**	RV32I的子集，仅支持16个通用整数寄存器
RV64I	**51**	64位地址空间与整数指令及一部分32位的整数指令
RV128I	**71**	128位地址空间与整数指令及一部分64位和32位的指令

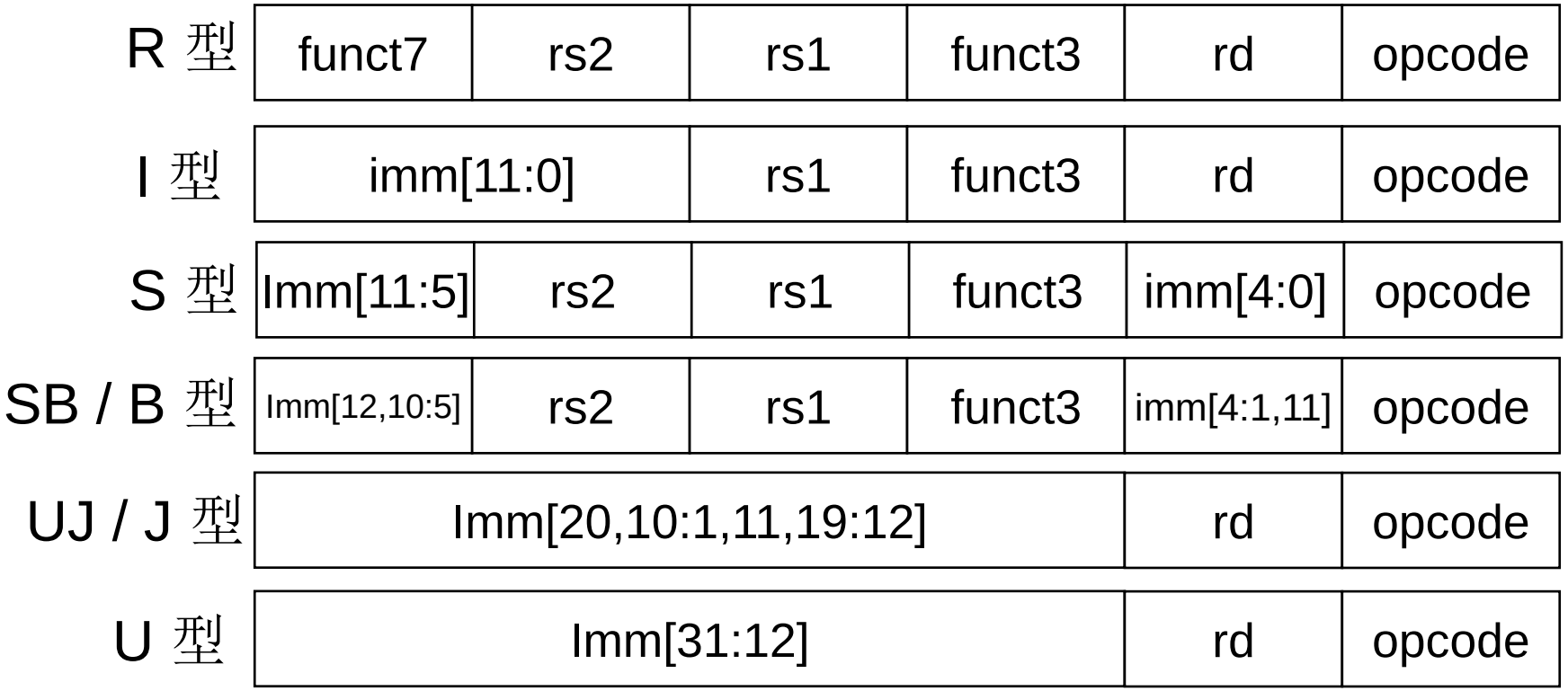
RV64扩展指令集	指令数	描述
I	**51**	基本体系结构
M	**13**	整数乘法与除法指令
A	**22**	原子操作(存储原子操作和load-reserved/store-conditional指令)
F	**30**	单精度（32bit）浮点指令
D	**32**	双精度（64bit）浮点指令
C	**36**	压缩指令

可配置的寄存器组

- 通用寄存器（GPR：General Purpose Registers）
  - **32位**架构(RV**32**I)：32个**32位**的通用寄存器；
  - **64位**架构(RV**64**I)：32个**64位**的通用寄存器
  - 嵌入式架构RV32E有16个32位的通用寄存器
  - 支持单精度浮点数（F），或者双精度浮点数（D），另外增加一组独立的通用浮点寄存器组，f0~f31
- 控制状态寄存器（CSR：Control and Status Registers）
  - 用于配置或记录一些运行的状态（异常和中断处理中常用）
  - 处理器核内部的寄存器，使用专有的12位地址码空间

规整的指令编码

- 所有通用寄存器在指令码的位置是一样的，方便译码；
- 所有的指令都是32位字长，有 6 种指令格式：寄存器型，立即数型，存储型，分支指令、跳转指令和大立即数



RISC-V的数据传输指令

- 专用内存到寄存器之间传输数据的指令，其它指令都只能操作寄存器
  - 简化硬件设计
  - 支持字节（8位），半字（16位），字（32位），双字（64位，64位架构）的数据传输
  - 推荐但不强制地址对齐
  - 小端格式

RISC-V的特权模式

- RISC-V架构定义了三种工作模式，又称特权模式（Privileged Mode）：
  - Machine Mode：机器模式，简称M Mode
  - Supervisor Mode：监督模式，简称S Mode
  - User Mode：用户模式，简称U Mode
- RISC-V架构定义M Mode为必选模式，另外两种为可选模式。通过不同的模式组合可以实现不同的系统

RISC-V

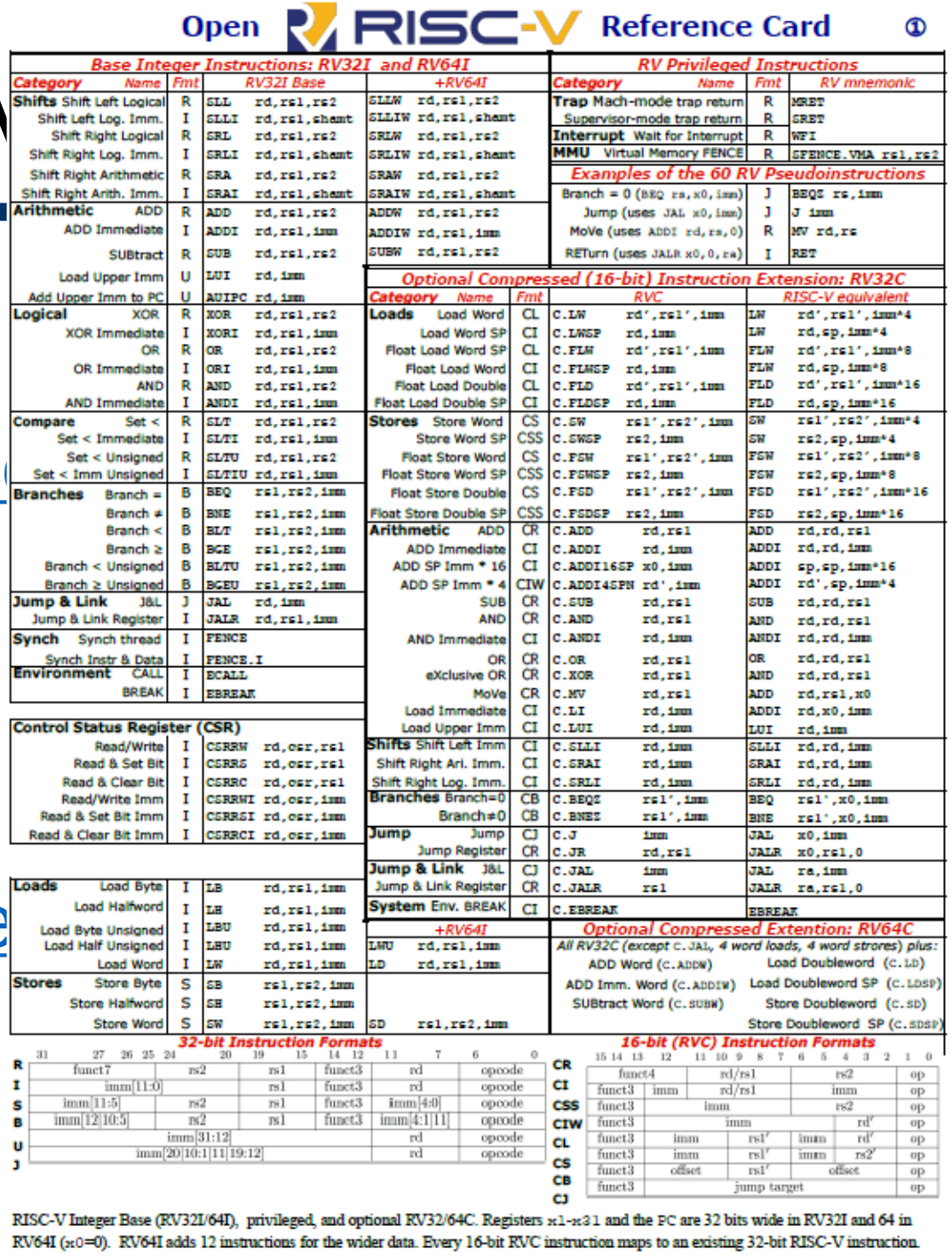
- RISC-V官方指令集手册

https://riscv.org/specificati

- 中文简化版

http://riscvbook.com/chine

v2p1.pdf



RISC V指令集系统架构小结

- **完全开放的 ISA**
- **精简**
  - 包含一个最小的ISA固定核心（可支撑OS，方便教学）
  - 适合硬件实现，而不仅仅是适用于模拟或者二进制翻译
- **后发优势**
  - 模块化的可扩展指令集
  - 简化硬件实现，提升性能
    - 更规整的指令编码、更简洁的运算指令、更简洁的访存模式：Load/Store架构
    - 高效分支跳转指令（减少指令数目）、简洁的子程序调用
    - 无条件码执行、无分支延迟槽

小结

- **RISC-V概述**
  - 指令系统的基本概念
  - 主流指令集及发展方向
  - RISC-V指令集