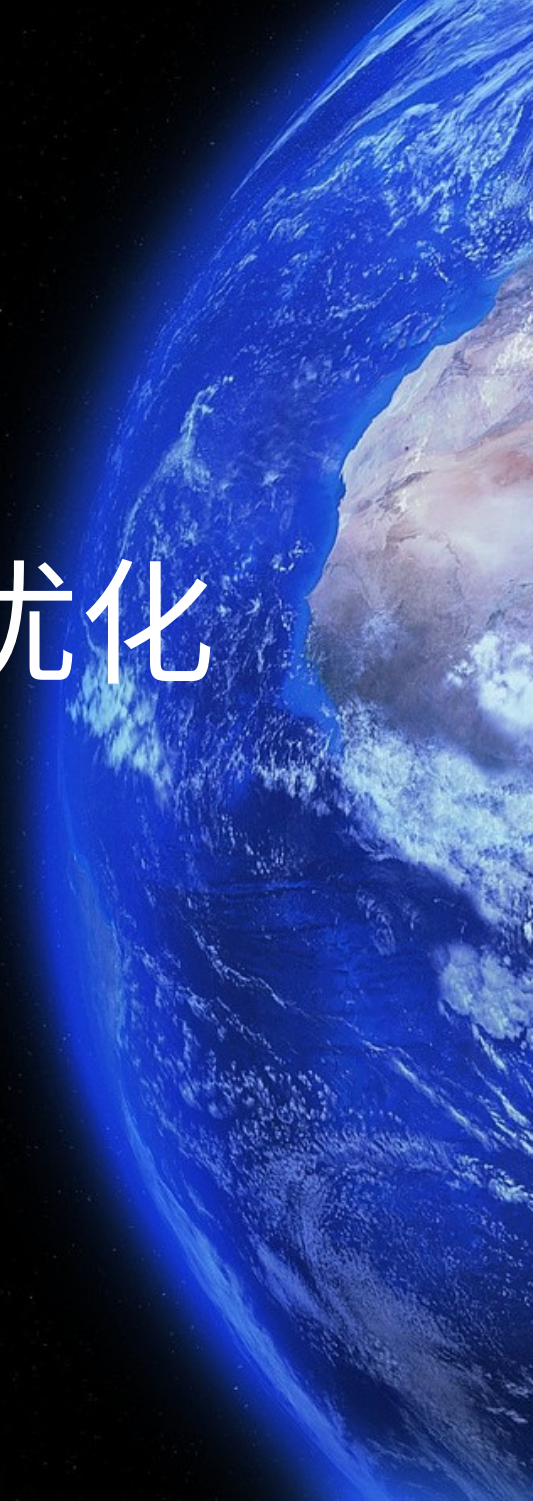
龙芯架构的设计与优化

中科院计算所微处理器中心

2021/08/04



1 龙芯指令集

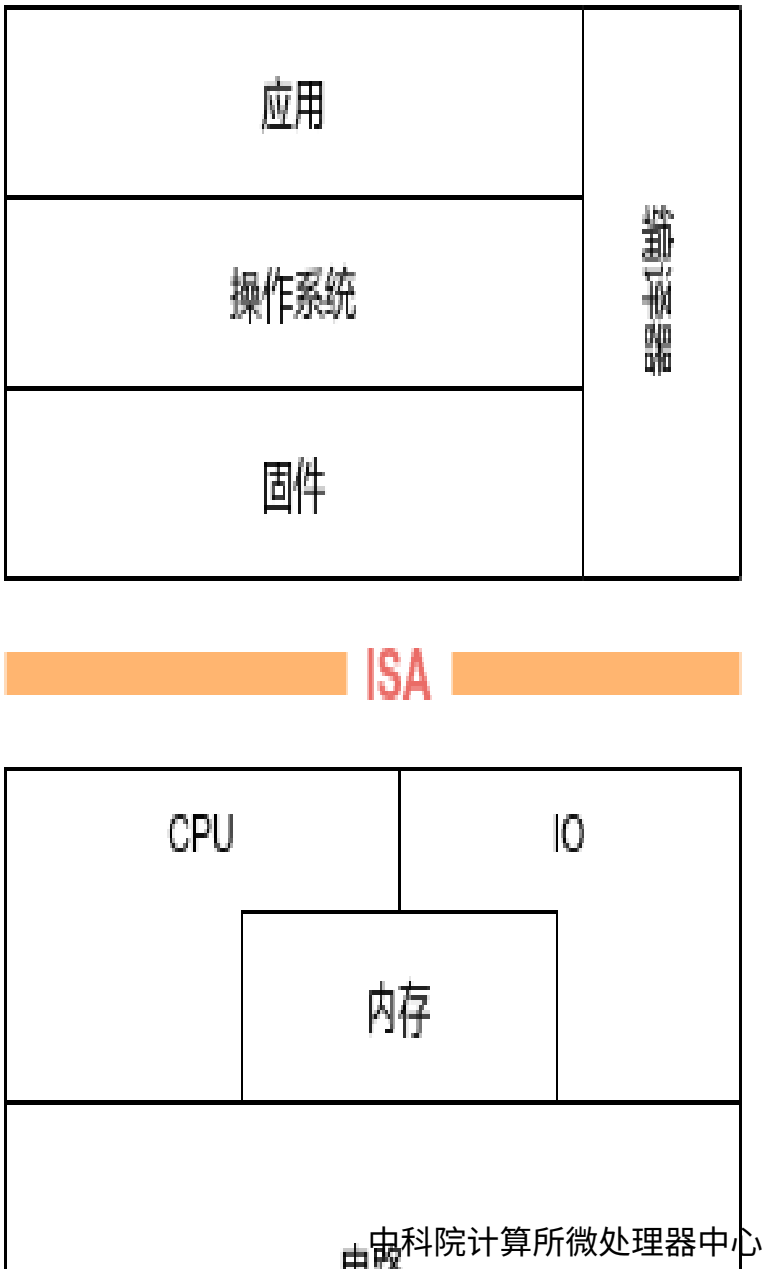
2 基于自主指令系统的生态建设

3 龙芯架构的安全设计

龙芯指令集

什么是指令集?

- ISA
  - Instruction Set Architecture
- 软硬件之间的 '合同'
  - 硬件所支持的操作 (指令)、运行模式和存储界面等的功能定义
  - 上述硬件资源的访问方式
- 承载软件生态的根基



主流指令集

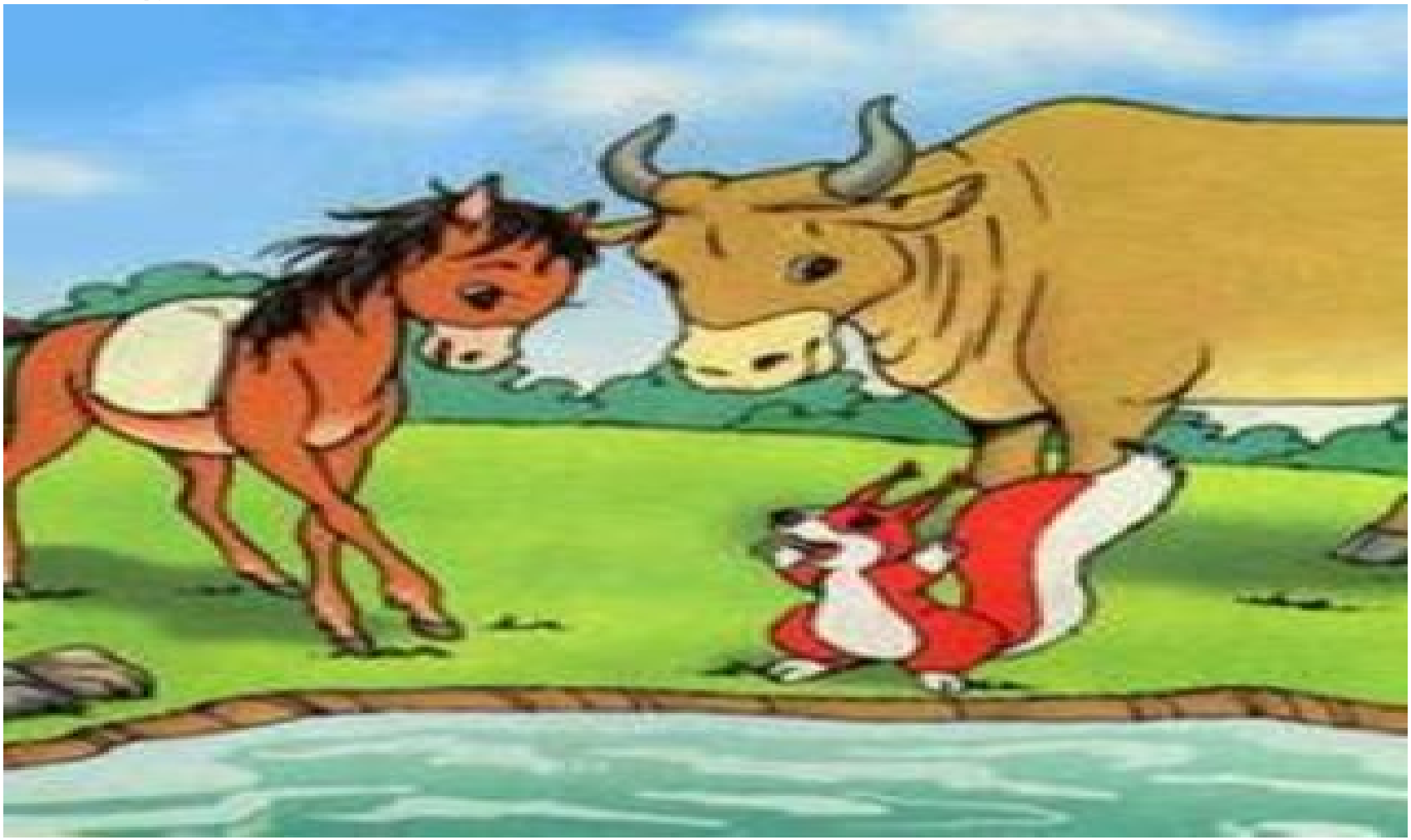
- X86
  - 桌面和服务器的主流指令系统
  - Intel 主导
  - 不授权
  - Amd 的授权来源于历史原因
- ARM
  - 嵌入式和移动设备的主流指令系统
  - Arm 主导
  - 可公开授权
    - 以 IP 核授权为主
    - 体系结构授权极少
    - 不允许自行扩展

为什么要设计自主指令集?

- 我国不可能基于国外指令系统建设自主信息产业生态
  - 中国人可以用英文写文章，但不可能用英文发展民族文化
  - 做跟班可以，想超过不行（华为超过思科、龙芯超过 MIPS）
  - 克服奴才心态，自主 CPU 之间不能靠比谁的美国“主子”发展得好来论英雄
- 自主与兼容指令系统的长期（>15 年）争论
  - 兼容的好处：软件生态，直接利用 X86 和 ARM 的现成生态
  - 兼容的弊端：受制于人，X86 不授权，ARM 严格授权，阻碍自主基础软件的发展
- 能不能做到既自主又兼容?
  - 通过二进制翻译技术兼容过渡
  - 苹果 M1 处理器的案例

设计指令集到底有多难?

- 实践出真知



龙芯指令集 (LoongArch) 的设计考虑

- 先进性
  - 积极吸收现代研究成果，摒弃一些过时的技术包袱
- 扩展性
  - 模块化
  - 预留足够的指令槽用于后续扩展
- 兼容性
  - 融合 X86/MIPS/Arm 等主流指令系统的主要特点，高效支持二进制翻译

LoongArch 概况

- 充分考虑兼容需求的自主指令系统
- LA64 约 2000 条指令
  - 基础指令 337 条
  - 虚拟机扩展 10 条
  - 二进制翻译扩展 170+
  - 128 位向量扩展 700+
  - 256 位向量扩展 700+
- 支持大幅简化的 LoongArch Primary
  - 将开源部分处理器核和设计平台 chiplab

Figure: LoongArch 指令集

LoongArch32 整数指令一览

指令类型	指令
算术运算	ADD.W, SUB.W, ADDI.W, ALSL, WLU12I.W SLT, SLTU, SLTI, SLTUI PCADDI, PCADDU12I, PCALAU12I AND, OR, NOR, XOR, ANDN, ORN, ANDI, ORI, XORI MUL.W, MULH.W, MULH.WU, DIV.W, MOD.W, DIV.WU, MOD.WU
移位运算	SLL.W, SRL.W, SRA.W, ROTR.W, SLLI.W, SRLI.W, SRAI.W, ROTRI.W
位操作	EXT.W.B, EXT.W.H, CLO.W, CLZ.W, CTO.W, CTZ.W, BYTEPICK.W REVB.2H, BITREV.4B, BITREV.W, BSTRINS.W, BSTRPICK.W MASKEQZ, MASKNEZ
转移指令	BEQ, BNE, BLT, BGE, BLTU, BGEU, BEQZ, BNEZ, B, BL, JIRL
访存指令	LD.B, LD.H, LD.W, LD.BU, LD.HU, ST.B, ST.H, ST.W, PRELD
原子访存	LL.W, SC.W
栅障指令	DBAR, IBAR
其他杂项	SYSCALL, BREAK, RDTIMEL.W, RDTIMEH.W, CPUCFG

LoongArch 特点

- 用户态指令
  - 保持典型 RISC 风格: 定长、32 通用定/浮点、load/store 结构
- 系统态
  - 吸取实践经验，完全重新设计：规整、可扩展
- 面向桌面和服务器设计的 64 位指令集
  - 效率优先

LoongArch 和 MIPS 性能比较

- MIPS 比 LA 用时多 12%
  - 相同的微结构
  - SPEC CPU 2000
    - test input
    - gcc -O2
  - FPGA 20M 平台
- 相当于一代处理器 IPC 提升
  - 2010-2015 年平均年提升仅 3%

Benchmark	MIPS	LA	MIPS/LA
164.gzip	2063	1695	122%
175.vpr1	575	466	123%
175.vpr2	606	503	120%
176.gcc	156	162	97%
181.mcf	1474	1554	95%
186.crafty	766	761	101%
197.parser	394	354	111%
252.eon1	38	32	119%
252.eon2	54	46	117%
252.eon3	207	179	115%
253.perlbmk1	1014	923	110%
253.perlbmk2	1032	634	163%
253.perlbmk3	613	618	97%
255.vortex	416	437	95%
256.bzip2	1292	1413	91%
300.twolf	691	549	126%
average	687	625	112%

动态指令数对比

	X86_64	MIPS	LoongArch	LA/X86	LA/MIPS
动态指令数	98,974,167,562	116,349,250,572	96,022,813,068	97%	83%

Coremark 1.0, 300000 iterations

程序执行时间 = 执行的动态指令数 * 平均每条指令的执行周期数 * 每周期的时间

效率提升的原因之一：丢弃不必要的历史包袱

- 案例：取消延迟槽
  - 延迟槽技术规定跳转指令后面一条指令一定会执行
  - 有效加速单发射静态流水线
    - 消除跳转带来的流水线空泡
  - 不适合现代多发射动态流水实现
    - 指令级并行技术更好地解决相关
    - 编译器无法填满延迟槽
    - 成为实现负担

效率提升的原因之二：吸收优秀的研究成果

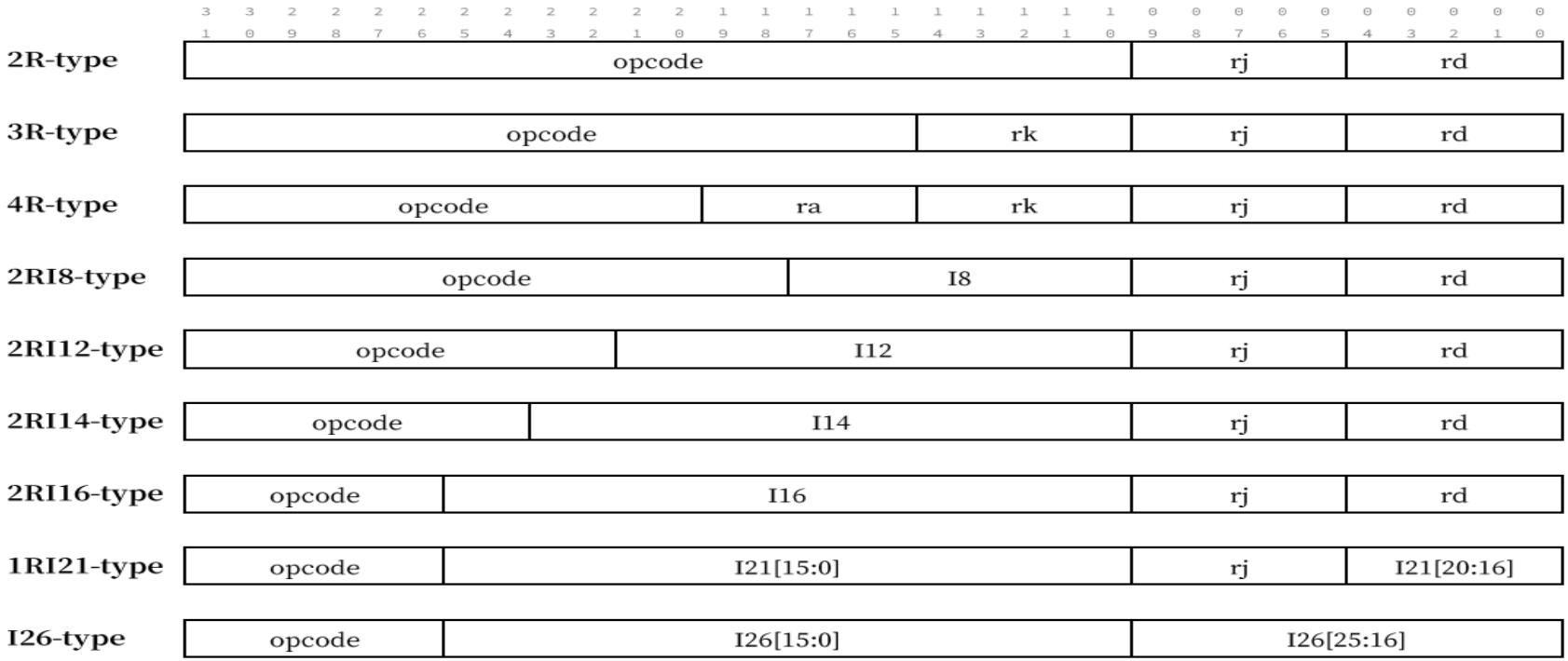
- 案例 1: 对立即数域的合理设计
- 案例 2：ABI 优化

LoongArch 和 MIPS 格式

MIPS 格式

R-type	OP(6)	RS1(5)	RS2(5)	RD(5)	SA(5)	OPX(6)
I-type	OP(6)	RS(5)	RD(5)	Immediate		
J-type	OP(6)	target				

LA 指令格式: 精打细算，预留足够的扩展空间



立即数域域的合理设计

- 该长的长：支持长跳转
  - MIPS 16 位偏移 vs LoongArch 21 位
  - 现代程序越来越大，短偏移可能导致效率下降
    - 为了防止超出跳转范围可能要生成更低效的代码
- 该短的短：
  - MIPS 采用统一 16 位立即数导致浪费指令编码空间
  - 多个现代指令系统采用 12 位 load/store 偏移

关于 ABI

- Application Binary Interface, 应用程序二进制接口
  - 寄存器使用约定
  - 函数调用约定
  - 系统调用和库函数
  - 可执行文件格式
  - 等等
- 不同的 ABI 可能带来不同效率

LoongArch 和 MIPS ABI(更多可用寄存器)

- MIPS: 32 个定点寄存器有 22 个编译器可用
- LoongArch: 32 个定点寄存器有 26 个编译器可用

LoongArch 和 MIPS ABI(更多可用寄存器)

- MIPS N64

R0:	永远为0
R1:	at临时寄存器
R2–R3:	v0/v1返回值
R4–R11：	参数a0–a7
R12–R15:	t0–t3临时寄存器
R16–R23:	s0–s7 callee save
R24–R25:	t8/t9
R26–R27:	k0/k1,内核用
R28:	gp,一般指向GOT表
R29:	sp,栈指针
R30:	s8/fp
R31:	ra,返回地址

- LA64 ABI

R0:	永远为0
R1:	ra返回地址
R2:	tp,线程指针
R3:	sp,栈指针
R4–R11：	参数a0–a7,a0/a1返回
R12–R20:	t0–t8临时寄存器
R21:	reserve
R22:	fp
R23–R31:	s0–s8 callee save
–	
–	
–	

LoongArch 和 MIPS ABI(重定位)

- 现代程序大都是可重定位
  - 代码可以被装载到不同位置运行
  - 函数引用全局变量需要访问全局偏移表 (GOT)
    - 函数头用当前 PC 算 GOT 的位置
- MIPS
  - 约定用 t9 寄存器调用函数
  - 基于 t9 用三条指令计算 GOT 位置
- LoongArch
  - 提供基于 PC 的运算指令
  - 一条指令即可从当前 PC 计算 GOT 位置

LoongArch 和 MIPS ABI(重定位)

```
//编译选项: gcc -O2 -fPIC -c test.c
int a; int test(void) { return a; }
```

- MIPS 汇编

```
 0:    lui  v1,0x0
 4:    daddu    v1,v1,t9
 8:    daddiu   v1,v1,0
 c:    ld   v0,0(v1)
10:    jr   ra
14:    lw   v0,0(v0)
```

- LA 汇编

```
0:    pcaddu12i    t0,0


4:    ld.d     t0,t0,0
8:    ldptr.w  a0,t0,0
c:    jirl     zero,ra,0
```

对二进制翻译的支持

- 指令融合技术
  - X86/Arm eflags
  - 浮点栈访问模式
  - RISC-V 同步指令
  - **并非简单的并集!**
- 高效二进制翻译过程
  - 地址翻译加速
  - 便签寄存器、专用插桩转移指令等加速翻译中边角情况处理
  - 间接跳转加速（3A6000 实现）等等
- 目标：“十九八七”
  - MIPS/Linux 应用, Android/Arm 应用, X86/Linux 应用, X86/windows 系统
    - 翻译性能为原生的 100%、90%、80% 和 70% 以上

二进制翻译加速案例

0		SUB	ECX	EDX		
1		JE	X86_target			

- EFLAGS 翻译加速
  - X86/Arm 运算指令同时产生多个标志位
  - RISC 需要 40 多条指令完全模拟
  - 定义一条指令产生标志
    - RISC 风格

0.00		SUBU	Result	Recx	Redx	
0.01		SRL	Rsf	Result	31	/*SF=Result[31]*/
0.02		BEQ	Result	R0	L1	
0.03		ADD	Rzf	R0	R0	/*ZF=0*/
0.04		B	L2			
0.05		NOP				
0.06	L1:	ADDI	Rzf	R0	1	/*ZF=1*/
0.07	L2:	SRL	Rtmp1	Result	31	
0.08		SRL	Rtmp2	Redx	31	
0.09		SRL	Rtmp3	Recx	31	
0.10		BEQ	Rtmp1	Rtmp3	L3	
0.11		NOP				
0.12		BEQ	Rtmp2	Rtmp3	L3	
0.13		NOP				
0.14		ADDI	Rof	R0	1	/*OF=1*/
0.15		B	L4			
0.16		NOP				
0.17	L3:	ADD	Rof	R0	R0	/*OF=0*/
0.18	L4:	SRL	Rhigh2	Recx	16	
0.19		SRL	Rhigh1	Redx	16	
0.20		SUBU	Rtmp	Rhigh2	Rhigh1	
0.21		BEQ	Rtmp	R0	L5	
0.22		NOP				
0.23		BLTZ	Rtmp	L7		
0.24		NOP				
0.25		B	L6			
0.26		NOP				
0.27	L5:	SLL	Rlow2	Recx	16	
0.28		SRL	Rlow2	Rlow2	16	
0.29		SLL	Rlow1	Redx	16	
0.30		SRL	Rlow1	Rlow1	16	
0.31		SUBU	Rtmp	Rlow2	Rlow1	
0.32		BLTZ	Rtmp	L7		
0.33		NOP				
0.34	L6:	ADD	Rcf	R0	R0	/*CF=0*/
0.35		B	L8			
0.36		NOP				
0.37	L7:	ADDI	Rcf	R0	1	/*CF=1*/
0.38	L8	ADD	Recx	Result	R0	
1.00		BNE	Rzf	R0	MIPS_target	
1.01		NOP				

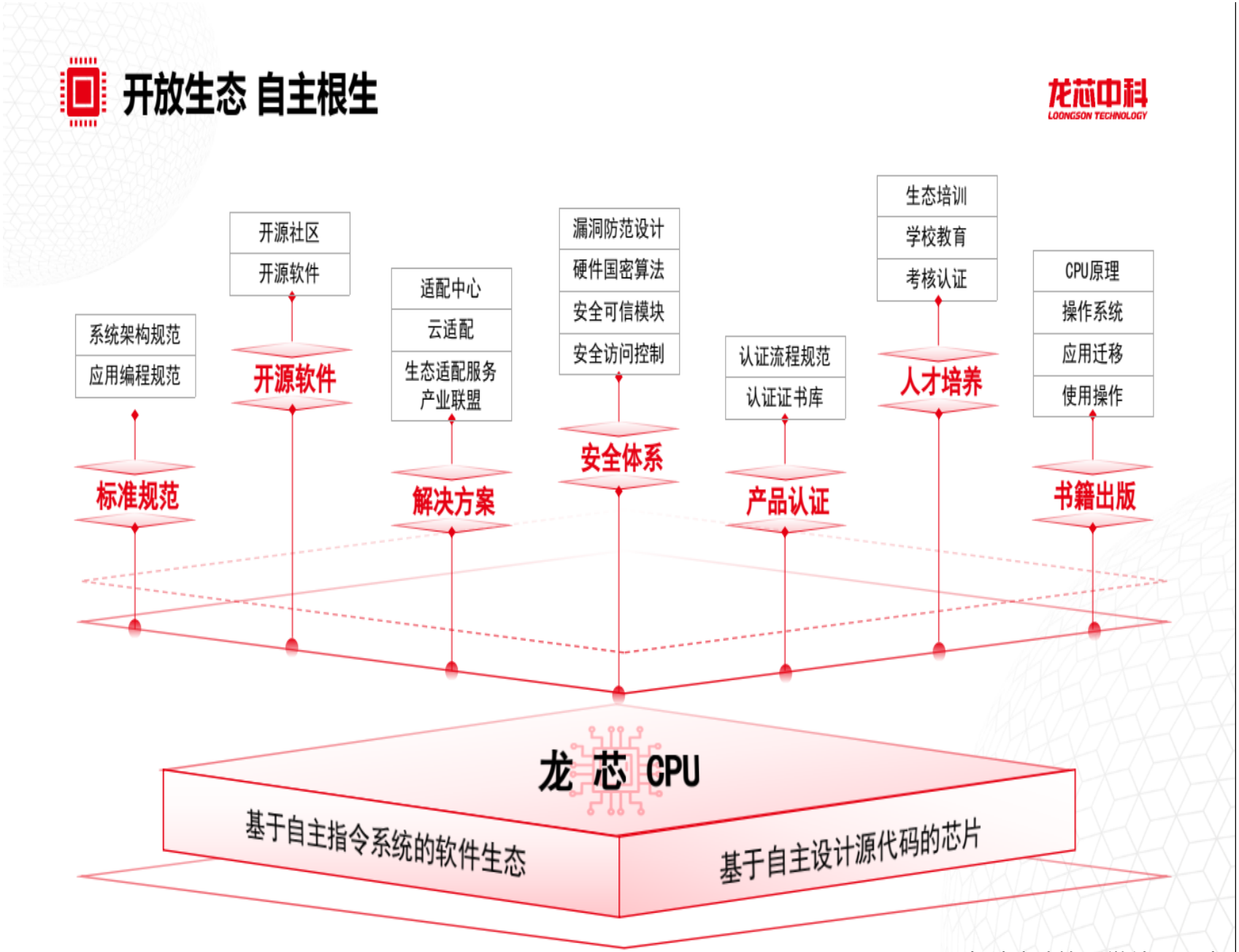
0.0		SUB	Result	Recx	Redx	/*Generating Sub result*/
0.1		X86SUB	Reflag	Recx	Redx	/*Generating EFLAGS*/
1.0		X86JE	Reflag	MIPS_target		/*Branch on EFLAGS*/

基于自主指令系统的生态建设

持续 20 年的生态能力建设

- 自主掌握几乎所有大型基础软件
  - 固件：PMON/uboot/UEFI Bios
  - 内核：Linux/vxWorks
  - 编译器: GCC/LLVM/Java/.Net/GoLang
  - 浏览器: Firefox/Chrome
  - 云：kvm/docker/openstack/alphapine...
  - 图形和多媒体：ffmpeg/libavx/pixman/mesa...
  - 发行版：Loongnix
- 逐步主导了众多 MIPS 平台软件的开源社区
- 积累了软硬件磨合的大量经验

龙芯生态建设


开放生态 自主根生
龙芯中科
LOONGSON TECHNOLOGY
系统架构规范
应用编程规范
标准规范
开源社区
开源软件
开源软件
适配中心
云适配
生态适配服务
产业联盟
解决方案
漏洞防范设计
硬件国密算法
安全可信模块
安全访问控制
安全体系
认证流程规范
认证证书库
产品认证
生态培训
学校教育
考核认证
人才培养
CPU原理
操作系统
应用迁移
使用操作
书籍出版
龙芯CPU
基于自主指令系统的软件生态
基于自主设计源代码的芯片

统一 API 和统一系统架构

统一API与统一系统架构

龙芯中科 LOONGSON TECHNOLOGY

应用开发者的接口（API）是建生态的必争

- API是操作系统的“指令系统”，生态的关键是开发者，API决定开发者习惯，具有极强的技术粘性。
- 对重要API进行优化：GCC、浏览器、JS、图形库、GIS显控，Java等。

统一系统架构接口是组织产业链的关键

- 学习Intel统一系统架构。系统架构的范围包括主板、整机、BIOS和操作系统。所有基于龙芯CPU的电脑整机都能够安装相同的操作系统。

龙芯已支持Linux全部主流开发环境，维护生态稳定发展

基础软件类型	名 称	典型开发工具
编程语言	Java	OpenJDK
	C/C++	GCC
	PHP	Apache
	Python	Python虚拟机
	Ruby	Ruby虚拟机
	Node.js	Node.js虚拟机，Electron
	Go	golang
	C#	.Net Core
函数库	本地图形界面库	Qt
	Java图形界面库	AWT/Swing，JavaFX
	3D图形库	OpenGL
	视频解码库	ffmpeg、openh264、libvpx
平台引擎	Web中间件	Tomcat
	J2EE引擎	GlassFish
	数据库	Mysql，Mongodb
	3D中间件	osgEarth
	云平台	Docker，KVM，K8s，Openstack
	大数据	Hadoop，Spark，Storm
浏览器	HTML/CSS/JavaScript	Firefox，Chromium
	JavaScript框架	jQuery，AngularJS
	CSS框架	Bootstrap
	Flash ActionScript	Flash插件
	HTML5	Firefox，Chromium
	WebGL	Firefox，Chromium
	WebRTC	Firefox，Chromium
性能分析工具	Profiling Tool	Oprofile，Perf
集成开发环境	Java/C/C++ IDE	Eclipse
	Qt IDE	Qt Creator

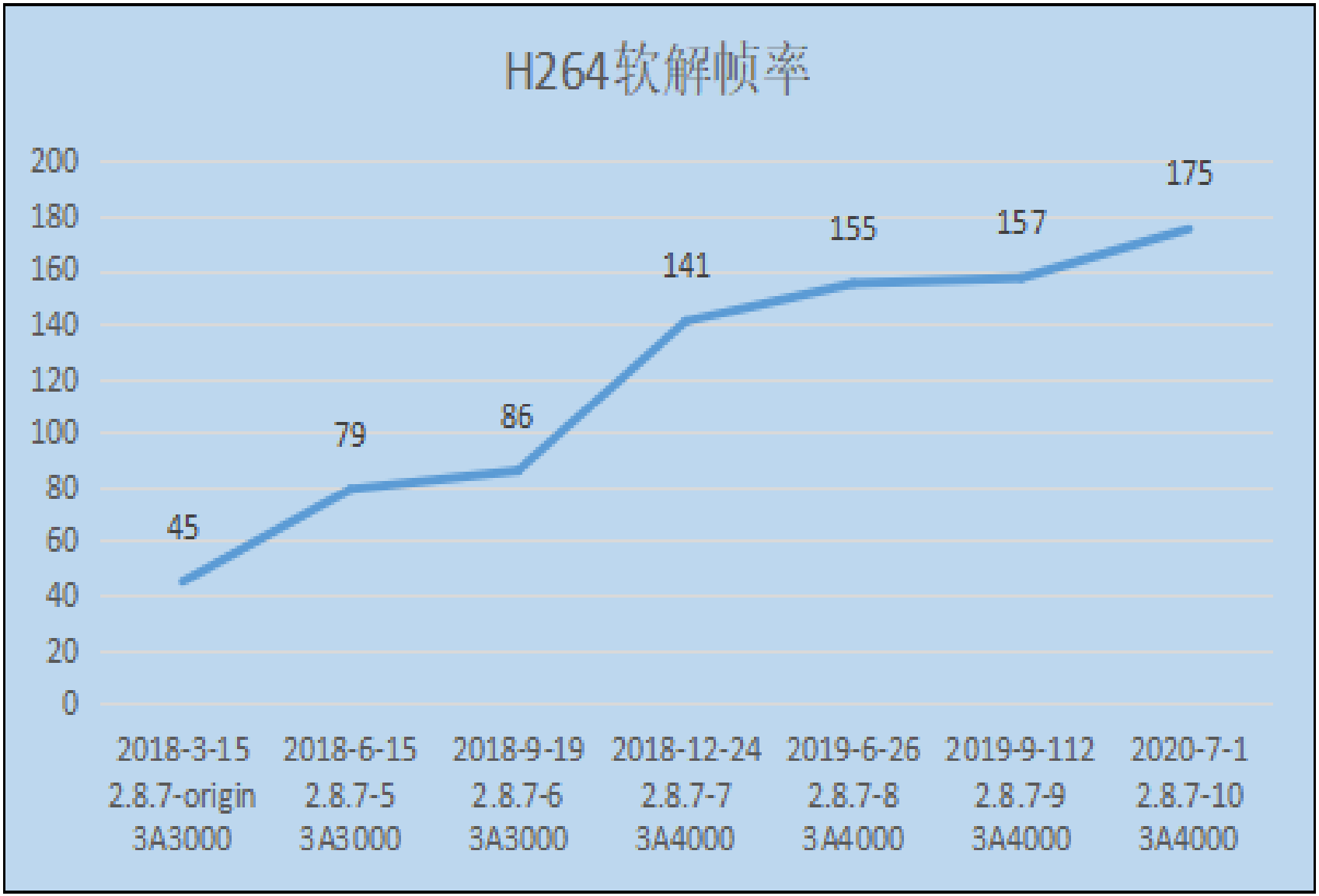
龙芯生态建设案例：Java

– 两个 OpenJDK 社区 maintainer

	龙芯OpenJDK	Oracle Java SE	Debian OpenJDK	AdoptOpen JDK	Azul Zulu
基于OpenJDK	√	√	√	√	√
100%开源，随意使用和重新发布	√	×	√	√	√
通过TCK测试，符合Java规范	√	√	×	×	√
是否免费	√	×	√	√	部分
及时获取最新安全更新和修复	√	√	OS版本维护周期内	√	√
支持多平台（x86/aarch64…，Windows/Linux…）	×	√	√	√	√
支持MIPS64	√	×	√	×	×
同Oracle Java SE性能相当	√	√	部分	√	√
支持多种安装方式（tar/deb/rpm/docker/JDK/JREs）	部分	部分	×	部分	√
OpenJFX（JavaFX）	√	√	√	×	√
Java Web Start和Applets	√	√	√	×	×
专业技术支持	√	√	×	×	√

龙芯生态建设案例：多媒体

- 贡献数万行开源代码



龙芯生态建设案例：图书

开放知识库：通过书籍出版让用户和开发者更容易地获取学习资源。

龙芯书籍种类：覆盖从CPU设计原理，系统应用开发标准，龙芯电脑使用等全链条系列教程

书名	出版时间	类别
龙芯自主可信计算及应用	2018年	理论
龙芯应用开发标准教程	2018年	应用开发
龙芯电脑使用解析（中标OS）	2019年	使用操作
龙芯电脑使用解析（统信UOS）	2020年	使用操作
用芯探核	2020年	基础软件技术
“龙芯派”开发实战	2020年	嵌入式
龙芯操作系统制作	2021年	操作系统
龙芯服务器管理和运维实战	2021年	使用操作
龙芯WPS办公应用	2021年	使用操作
龙芯CPU通识课	2021年	CPU科普

迅速建立 LoongArch 的基础生态

- 龙芯社区操作系统移植过程
  - 大约 6 个月形成 GCC 工具链、定义 ABI，构建最小系统
  - +6 个月完成常用的 3000+ 软件包移植
    - 包括 LLVM/Chrome/Javascript/openJDK 等基础软件
  - +6 个月完成 2 万 + 软件包移植
    - debian 10 中合计 29000 余包，约 3000 个软件包有结构相关
    - 40 万行架构相关代码 + 40 万行安全相关补丁
  - +6 个月测试开始收敛，基本达到 MIPS 同等成熟度
    - 展开大量测试和优化
- 两款商业 OS 和众多商业软件完成移植
  - 约半年时间

软件和指令集持续迭代优化

- OpenJDK 优化
  - 解释器 (235 条字节码)：
    - 优化前：51 个 LA 翻译用了比 MIPS 多的指令, 95 个相等，89 个更少，总体略有优势
    - 优化后：没有一个比 MIPS 多用指令, 196 条用更少，总体提升近 20%
    - 方法：优化立即数生成代码、采用 LA 的基址变址等指令
  - 翻译器
    - 充分利用 LA 更大的跳转范围和相对 PC 跳转等优势
    - 定位缺乏条件传送指令导致个别基准程序性能大幅低于 MIPS 问题

龙芯架构的安全设计

从处理器角度看安全

- 非国产一定不安全，国产不一定安全
  - 需要有严密的安全机制来保障安全
- 满足多个不同领域不同等级的安全需求
  - 为安全软件提供高效的支持
  - 避免侧信道攻击等硬件漏洞
  - 提供可信体系
  - 信任＋监管

龙芯 3A5000 处理器的安全机制

- 安全软件效率
  - 加解密加速指令
- 漏洞免疫
  - meltdown
  - spectre
- 安全支持机制
  - 读禁止和执行禁止权限
  - 返回地址检查栈
  - IO 防护
  - 可信体系
  - 安全执行域

返回地址检查栈

- 硬件维护函数返回地址的栈结构
  - 其内容和压栈、弹栈操作软件不可见
- 匹配检查
  - 函数调用时自动压栈，返回是自动弹栈并与实际跳转地址比较，不匹配则触发例外
- 可扩展设计
  - 硬件缓冲放不下时触发例外，软件可以利用安全存储区域存放更多内容
- 特殊情况处理
  - 提供 safepop 指令处理特殊的不成对汇编情况

IO 防护

- 支持指定若干地址窗口 IO 访问控制
  - 例如，禁止修改 CPU 配置、烧写 BIOS rom 以及硬盘 MBA 等区域
  - 配合安全执行域机制，可以做到即使内核沦陷也有基本安全保障

安全执行域

- 安全配置空间
  - 只能通过专门安全指令访问
  - 与内存、IO 空间完全隔离
- 安全数据区域
  - 只能通过专门指令读写
- 安全指令区域
  - 安全指令只能在给定区域执行
- 运行过程动态监管
  - 可定义各类检查

相关研究工作

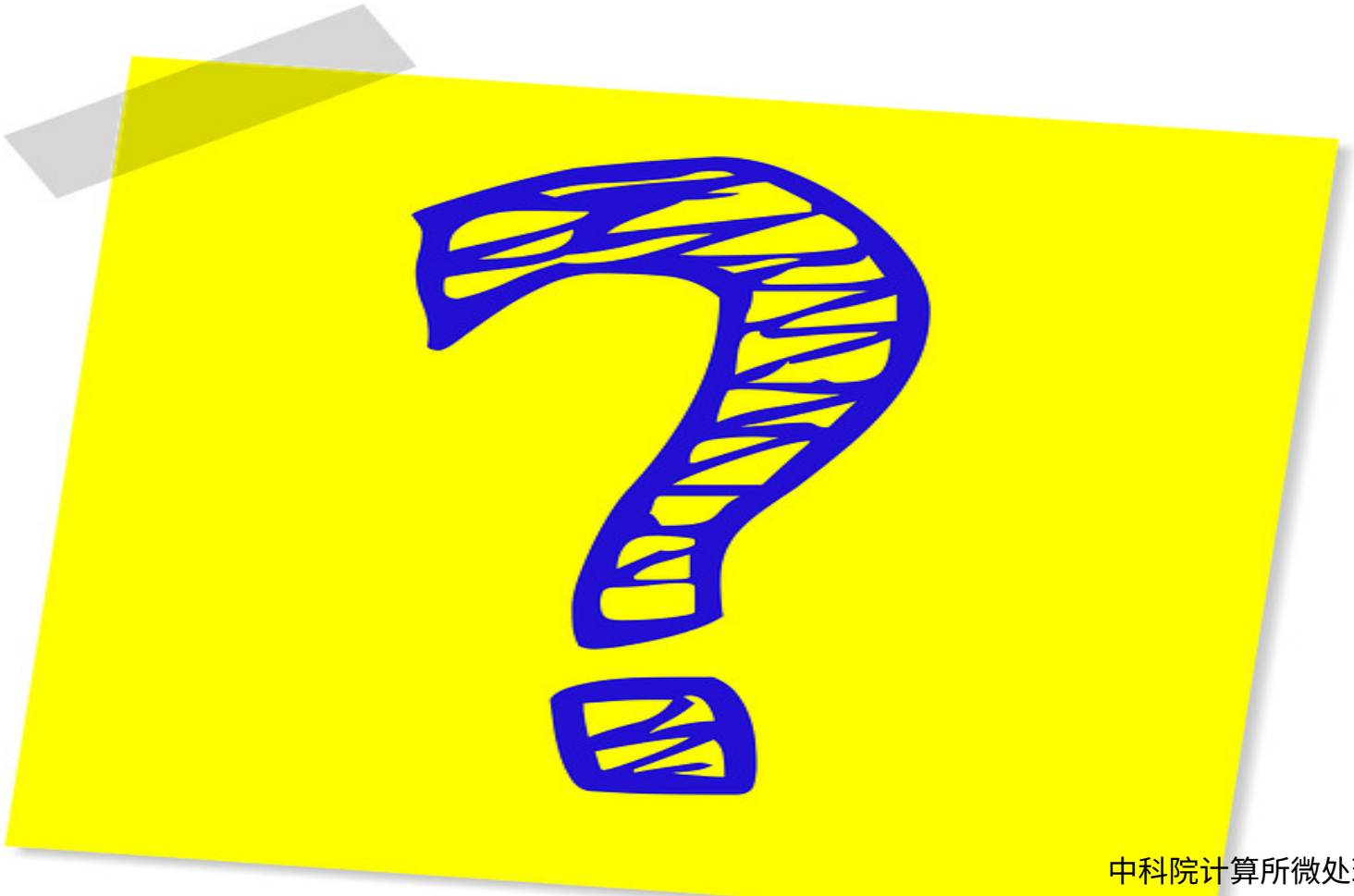
- 指令集安全扩展
  - 读写指针的特殊指令等
- 硬件木马检测和防范技术

小结

- LoongArch 指令集简介
  - 通过自主指令集达成更大的自由
  - 能够实现一定的先进性
- LoongArch 的软件生态工作
  - 设计自主指令集的真正困难所在
  - 长期积累的基础使之成为可能
  - 还需持续迭代优化
- LoongArch 的安全设计
  - 自主架构带来的巨大创新空间
  - 面向实际需求设计方案

Q&A

- name: 张福新
- mobile: 13915631204
- email: fxzhang@ict.ac.cn



参考资料

- 计算机体系结构量化分析，第 6 版
  - 附录 A/H/K
- CIS 501 Computer Architecture slides on ISA
  - https://www.cis.upenn.edu/~milom/cis501-Fall08/lectures/03_isa.pdf
- RISC-V 作者之一的博士论文
  - Design of the RISC-V Instruction Set Architecture, by Andrew Shell Waterman
- 龙芯社区网站
  - www.loongnix.org