课外阅读材料 1 号

0x00：MIPS 的起源

MIPS（Microprocessor without Interlocked Pipeline Stages，无互锁流水线微处理器）是一种精简指令集（RISC）架构，由美国 MIPS 计算机系统公司（由斯坦福大学团队于 1984 年创立，现为美普思科技）开发，以其高效、简洁和强大的特性，在嵌入式系统、工作站和超级计算机等领域得到了广泛应用。

MIPS 架构有多个版本，包括 MIPS I、II、III、IV，以及 MIPS V，这五个版本又分别分为 MIPS32/64 Release（即其 32 位/64 位实现）。截至 2017 年 4 月的最新版本是 MIPS32/64 Release 6；2021 年 3 月，美普思科技宣布停止开发 MIPS 架构，并加入 RISC-V 基金会，未来的处理器设计将基于 RISC-V 架构。

0x01：MIPS 的设计思想

MIPS 体系结构属于典型的 RISC 架构，其设计流程围绕“简单即快速”这一核心理念。其中，将三类（R-Type、I-Type、J-Type）机器指令分为若干个固定长度的操作码和操作数字段，使得指令译码、指令执行和寻址速度同时得到提升。这种规整性的设计不仅降低了指令的复杂性，还提高了处理器的执行效率。用一句话概括 MIPS 的设计思想：通过减少指令的种类和复杂性来提升性能。

相比于 CISC（复杂指令集计算机）体系结构，MIPS 指令集更为精简，每个指令都只执行单一的操作，如加、减、乘、除、移位等，这种设计使得处理器在解析并执行指令时更加高效。此外，MIPS 体系结构还采用了加载/存储式的存储器访问机制，即只有专门的加载（Load）和存储（Store）指令能访问主存中的数据，其他指令只能对寄存器中的数据进行操作。从指令集总体考虑，这种设计最大化了非访存指令的数量，简化了指令的复杂性，提高了处理器的性能。

MIPS 体系结构的另一个显著特点，是其高效的并行性。这里所提到的并行性，可以分为指令级并行性（ILP）和数据级并行性（DLP）两类。通过将计算机指令的执行过程简化为若干个用时相当的基本操作，MIPS 体系结构能够实现高效的指令流水线和运算流水线，提升指令执行效率。同时，在分支预测等先进控制技术支持下，MIPS 的指令执行效率得到了进一步优化。结合高效的编译器和汇编器，为各种嵌入式系统、工作站和超级计算机提供了强大的计算支持。

综上所述，MIPS 体系结构以其独特的设计理念和卓越的性能表现，在计算机体系结构领域占据了一席之地。它的起源可以追溯到初代 RISC 处理器，其设计思想主要围绕“简单即快速”这一核心理念展开。随着技术的不断进步和应用需求的不断变化，MIPS 及其衍生的体系结构也将继续发展和完善，为计算机系统设计带来更多的创新和突破。

0x02：掰扯一下 MIPS 和龙芯

龙芯系列处理器是由中国科学院中科技术有限公司设计研制的，具有自主知识产权的芯片产品。该系列处理器从 2001 年开始研发，至今已经推出了包括龙芯 1 号、龙芯 2 号和龙芯 3 号等多个系列产品，以及配套的龙芯桥片系列。从计算机专业视角来看，龙芯和 MIPS 之间既存在联系，又具有不小的差异。

首先来看二者的联系，MIPS 是一种经典的芯片设计架构，以其高效、稳定的特点而著称。在过去的芯片设计中，龙芯中科曾经采用 MIPS 架构作为基础，通过扩充指令集的方式进行创新。MIPS 架构的简洁性、高性能和可扩展性为龙

芯中科的芯片提供了有力的支持。基于 MIPS 架构，龙芯中科能够提高其芯片的性能并降低功耗，从而增强其芯片的市场竞争力。

再来看二者的区别，随着龙芯中科在芯片设计领域的不断发展和创新，他们开始寻求摆脱 MIPS 授权的方式。基于二十年的 CPU 研制和生态建设积累，龙芯中科于 2021 年推出了采用全新龙架构（LoongArch™）的龙芯 3A5000 处理器。这一新架构是龙芯中科完全自主设计的，从架构的顶层规划，到各部分的功能定义，再到细节上每条指令的名称、编码格式、操作定义均为自主设计，无需国外授权。龙架构的推出，标志着龙芯中科在自主研发 CPU 方面取得了重大突破，并且开启了从技术升级迈向全面生态建设的新阶段。

龙架构在设计上充分考虑了兼容性的需求，通过高效的硬件二进制转译技术对现有 X86、ARM 等主流架构的应用提供运行支持。这使得龙架构在保持自主性的同时，也能够与现有的软件生态进行良好的兼容。此外，龙架构还吸纳了近年来指令系统设计领域诸多先进的技术发展成果，从而使其具有先进性和高效性。龙芯处理器的主要系列和应用领域如下表所示：

龙芯系列处理器进化历程

应用领域	架构	系列	型号
通用处理器	MIPS64 兼容	龙芯 3 号	龙芯 3A/B1000-4000
	LoongArch	龙芯 3 号	龙芯 3A5000/6000
超算处理器	MIPS64 兼容	龙芯 2/3 号	龙芯 2F、龙芯 3A/B1000
应用处理器	MIPS64 兼容	龙芯 2 号	龙芯 2H/K1000、龙芯 2K2000
微控制器	MIPS32 兼容	龙芯 1 号	龙芯 1A/B/C/D/G/H
航天专用	MIPS64 兼容	龙芯 1 号	龙芯 1E0300、龙芯 1E1000
	MIPS32 兼容	龙芯 1 号	龙芯 1E/F04、龙芯 1J

从设计理念上看，MIPS 架构注重简洁性和高效性，通过减少指令种类和复杂性来提高处理器性能。而龙芯架构则在保持高效性的同时，更加注重自主性和兼容性。它摒弃了传统指令系统中不适应当前软硬件设计技术发展趋势的陈旧内容，并融合了 X86、ARM 等架构的主要特点，以支持更广泛的软件生态。

从指令格式上看，MIPS 架构的指令集相对固定，虽然稳定、高效，但扩展能力有限。而龙架构在设计时就充分考虑了扩展性，指令字中给未来扩展留有余地，这使得龙架构更能适应今后的持续演进。

从应用领域上看，MIPS 架构已广泛应用于嵌入式系统、工作站和超算。而龙架构则更注重国产化桌面计算机和网络服务器等领域。随着信息技术的快速发展，龙架构在国产计算机生态建设中必将扮演越来越重要的角色。

从发展前景上看，MIPS 架构在来自其他架构的竞争压力下，美普思科技已于 2021 年 3 月宣布停止技术开发，其生态活跃度可能随之降低，未来的市场地位能否维持仍不明朗。而龙芯架构作为新兴的自主指令集架构，在国内政策和市场的支持下，有着巨大的发展空间和潜力。随着技术的不断进步和应用场景的不断拓展，龙芯架构有望在未来实现更广泛的应用和更大的突破。

0x03：值得一提的龙芯黑科技

龙架构基于 LoongArch 指令集，是一种精简指令集计算机（RISC）。其指令长度固定且编码格式规整，绝大多数指令有两个源操作数和一个目的操作数。此外，龙架构分为 32 位和 64 位两个版本，分别称为 LA32 和 LA64，其中 LA64 架构能够向下兼容 LA32 架构。

在兼容性方面，龙架构的“黑科技”在于能够通过二进制转译功能，从硬件层面上支持多种体系结构的软件，这项先进技术得益于龙芯团队在二进制转译方面十余年的技术积累和创新。龙芯中科董事长、总经理胡伟武表示，龙芯将“通过指令系统的创新，消灭指令系统的差异”。

具体来说，龙架构的二进制转译系统包括 LATM（LAT from MIPS）、LATA（LAT from ARM）和 LATX（LAT from X86），分别用于支持 MIPS、ARM 和 x86 平台的应用在龙芯平台上的安装和运行。这意味着，原本基于 MIPS、ARM 或 x86 体系结构的软件，可以通过龙架构的二进制转译功能，在龙芯平台上实现良好的运行效果。

龙架构在功能上针对 MIPS、X86、ARM、RISC-V 的特征，绝大多数指令可以做到 1 对 1 或 1 对 2 翻译；还包括对 x86 的 EFLAGS 支持、RISC-V 的原子同步指令支持；以及，ABI 方面支持 x86/MIPS 系统调用兼容，支持 MIPS 汇编码直接翻译成 LoongArch 二进制。

在主流体系结构兼容性方面，龙架构还通过开源的 wine 中间件技术，在类 UNIX 系统上模拟 Windows 的 API。再加上二进制转译技术的加持，使得龙芯平台上可以流畅运行各种常用的 Windows 桌面应用，如微信、Photoshop 等，甚至还可以运行安卓 APP。当然，进行二进制转译后性能会有一些（40%~60%）牺牲，但获得了较强的生态支持。考虑到日常使用中 CPU 性能是过剩的，转译后并不会有明显的卡顿现象。

通过二进制转译技术，龙架构为上层应用软件提供了目标指令集的良好虚拟运行环境。龙芯平台还对打印机、扫描仪等办公外设提供了“零适配、全兼容、免网络、高安全”的解决方案。原生 Windows 驱动程序可直接在龙芯平台上运行，不需要外设厂商重新适配；市场上现有的全部打印机型号，即使没有 Linux 版本驱动，也都可以在龙芯电脑上正常使用，具有高度的兼容性；驱动程序在类似沙盒（Sandbox）的封闭隔离环境中运行，且只有对特定外设的访问能力，而无法访问龙芯电脑本机上的文件和网络资源，可有效保障数据安全。

面向未来，龙芯中科提出了关于 LAT 二进制转译效率的“十九八七”设计指标：即力争做到 MIPS 应用的静态/动态翻译效率 100%（没有任何性能损失），ARM 应用的静态/动态翻译效率不低于 90%，x86 Linux 应用的动态翻译效率不低于 80%；x86 Windows 的动态翻译效率不低于 70%。

总的来说，龙架构的二进制转译功能使得龙芯平台能够兼容并运行多种体系结构的软件，并为用户提供更的安全性和更好的使用体验。

0x04：书归正传，经典的 MIPS

下面给出 MIPS 处理器中的通用寄存器和核心指令列表，供同学们课外阅读并自行研究。

另附超好玩网站一枚，可直接在浏览器里运行 MIPS 指令，并查看结果：https://rivoire.cs.sonoma.edu/cs351/wemips/

MIPS 处理器中的寄存器

寄存器编号	别名	功能
$0	$zero	常量寄存器，其值永远为 0（看起来像浪费资源，其实很有用） Constant 0
$1	$at	汇编暂存寄存器（保留给汇编器使用，被调用者无需保存内容） Assembly Temporary
$2,$3	$v0,$v1	用于存储子程序的返回值（子程序内使用时无需保存内容） Function Result
$4-$7	$a0-$a3	子程序调用的前 4 个参数（子程序内使用时无需保存内容） Argument 1 to 4
$8-$15	$t0-$t7	临时变量寄存器（子程序内使用时无需保存内容） Unsaved Temporary
$16-$23	$s0-$s7	变量寄存器（子程序内如要使用，必须保存内容并在返回前复原） Saved Temporary
$24-$25	$t8,$t9	临时变量寄存器（子程序内使用时无需保存内容） Unsaved Temporary
$26,$27	$k0,$k1	保留给中断和自陷程序使用 Reserved for EXCEPTION
$28	$gp	全局指针 Pointer to Global Data
$29	$sp	堆栈指针 Stack Pointer
$30	$fp	帧指针 Frame Pointer
$31	$ra	函数返回地址 Return Address

硬件上，这些寄存器并没有区别(除了 0 号以外)，区分的目的是为了不同的编译器产生的代码可以正常的互相调用。

对于 Linux 操作系统，位置无关代码的用户空间调用约定中还要求：当调用函数时，$t9 寄存器必须包含该函数的地址。这个约定源于 MIPS 的 System V ABI 补充规定。

MIPS 指令集 ➔ 核心指令汇总（共 31 条）

助记符	指令格式						示例	示例含义	操作及其解释
Bit #	31..26	25..21	20..16	15..11	10..6	5..0			
R-type	op	rs	rt	rd	shamt	func			
add	000000	rs	rt	rd	00000	100000	add $1,$2,$3	$1=$2+$3	rd ← rs + rt；其中 rs＝$2，rt=$3，rd=$1
addu	000000	rs	rt	rd	00000	100001	addu $1,$2,$3	$1=$2+$3	rd ← rs + rt；其中 rs＝$2，rt=$3，rd=$1（无符号数）
sub	000000	rs	rt	rd	00000	100010	sub $1,$2,$3	$1=$2-$3	rd ← rs - rt；其中 rs＝$2，rt=$3，rd=$1
subu	000000	rs	rt	rd	00000	100011	subu $1,$2,$3	$1=$2-$3	rd ← rs - rt；其中 rs＝$2，rt=$3，rd=$1（无符号数）
and	000000	rs	rt	rd	00000	100100	and $1,$2,$3	$1=$2&$3	rd ← rs & rt；其中 rs＝$2，rt=$3，rd=$1
or	000000	rs	rt	rd	00000	100101	or $1,$2,$3	$1=$2|$3	rd ← rs | rt；其中 rs＝$2，rt=$3，rd=$1
xor	000000	rs	rt	rd	00000	100110	xor $1,$2,$3	$1=$2^$3	rd ← rs xor rt；其中 rs＝$2，rt=$3，rd=$1（异或）
nor	000000	rs	rt	rd	00000	100111	nor $1,$2,$3	$1=~($2|$3)	rd ← ~(rs | rt)；其中 rs＝$2，rt=$3，rd=$1（或非）
slt	000000	rs	rt	rd	00000	101010	slt $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0；其中 rs＝$2，rt=$3，rd=$1
sltu	000000	rs	rt	rd	00000	101011	sltu $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0；其中 rs＝$2，rt=$3，rd=$1（无符号数）

sll	000000	00000	rt	rd	shamt	000000	sll $1,$2,10	$1=($2<<10)	rd ← (rt << shamt)；shamt 存放移位的位数（立即数）， 其中 rt=$2，rd=$1
srl	000000	00000	rt	rd	shamt	000010	srl $1,$2,10	$1=($2>>10)	rd ← (rt >> shamt)；逻辑移位，其中 rt=$2，rd=$1
sra	000000	00000	rt	rd	shamt	000011	sra $1,$2,10	$1=($2>>10)	rd ← (rt >> shamt)；算数移位，注意符号位保留， 其中 rt=$2, rd=$1
sllv	000000	rs	rt	rd	00000	000100	sllv $1,$2,$3	$1=($2<<$3)	rd ← (rt << rs)；其中 rs＝$3，rt=$2，rd=$1
srlv	000000	rs	rt	rd	00000	000110	srlv $1,$2,$3	$1=($2>>$3)	rd ← (rt >> rs)；逻辑移位，其中 rs＝$3，rt=$2，rd=$1
srav	000000	rs	rt	rd	00000	000111	srav $1,$2,$3	$1=($2>>$3)	rd ← (rt >> rs)；算数移位，注意符号位保留， 其中 rs＝$3，rt=$2，rd=$1
jr	000000	rs	00000	00000	00000	001000	jr $31	goto $31	PC ← rs
Bit #	31..26	25..21	20..16	15..0					
I-type	op	rs	rt	immediate					
addi	001000	rs	rt	immediate			addi$1,$2,100	$1=($2+100)	rt ← (rs + (sign-extend) immediate)；其中 rt=$1，rs=$2
addiu	001001	rs	rt	immediate			addiu$1,$2,100	$1=($2+100)	rt ← (rs + (zero-extend) immediate)；其中 rt=$1，rs=$2
andi	001100	rs	rt	immediate			andi$1,$2,10	$1=($2 & 10)	rt ← (rs & (zero-extend) immediate)；其中 rt=$1，rs=$2
ori	001101	rs	rt	immediate			andi$1,$2,10	$1=($2 | 10)	rt ← (rs | (zero-extend) immediate)；其中 rt=$1，rs=$2
xori	001110	rs	rt	immediate			andi$1,$2,10	$1=($2 ^ 10)	rt ← (rs xor (zero-extend) immediate)；其中 rt=$1，rs=$2

lui	001111	00000	rt	immediate	lui$1,100	$1=100*65536	rt ← (immediate*65536)； 将 16 位立即数放到目标寄存器高 16 位，低 16 位填 0
lw	100011	rs	rt	immediate	lw$1,10($2)	$1=memory[$2+10]	rt ← memory[rs + (sign-extend) immediate]；rt=$1，rs=$2
sw	101011	rs	rt	immediate	sw$1,10($2)	memory[$2+10]=$1	memory[rs + (sign-extend) immediate] ← rt；rt=$1，rs=$2
beq	000100	rs	rt	immediate	beq$1,$2,10	if($1==$2) goto (PC+4+40)	if (rs == rt) PC ← (PC+4 + ((sign-extend) immediate<<2))
bne	000101	rs	rt	immediate	bne$1,$2,10	if($1!=$2) goto (PC+4+40)	if (rs != rt) PC ← (PC+4 + ((sign-extend) immediate<<2))
slti	001010	rs	rt	immediate	slti $1,$2,10	if($2<10) $1=1 else $1=0	if (rs < (sign-extend) immediate) rt=1 else rt=0； 其中 rs＝$2，rt=$1
sltiu	001011	rs	rt	immediate	sltiu $1,$2,10	if($2<10) $1=1 else $1=0	if (rs < (zero-extend) immediate) rt=1 else rt=0； 其中 rs＝$2，rt=$1
Bit #	31..26	25..0					
J-type	op	address					
j	000010	address			j 10000	goto 10000	PC ← (PC+4)[31..28],address,0,0；address=10000/4
jal	000011	address			jal 10000	$31=(PC+4) goto 10000	$31 ← (PC+4)；PC ← {(PC+4)[31..28],address,00_b} 注意：指令字中的 address=(10000/4)

MIPS 指令集的不同版本包含不同数量的指令，上述核心指令仅为 MIPS 基本指令集的一个子集。在 MIPS 指令集版本更替的过程中，可能增加新的指令或去除老旧的指令，感兴趣的同学请自行查阅 MIPS 官方提供的文档。