龙芯中科
LOONGSON TECHNOLOGY

LoongArch精简版指令集介绍

汪文祥

龙芯中科
LOONGSON TECHNOLOGY

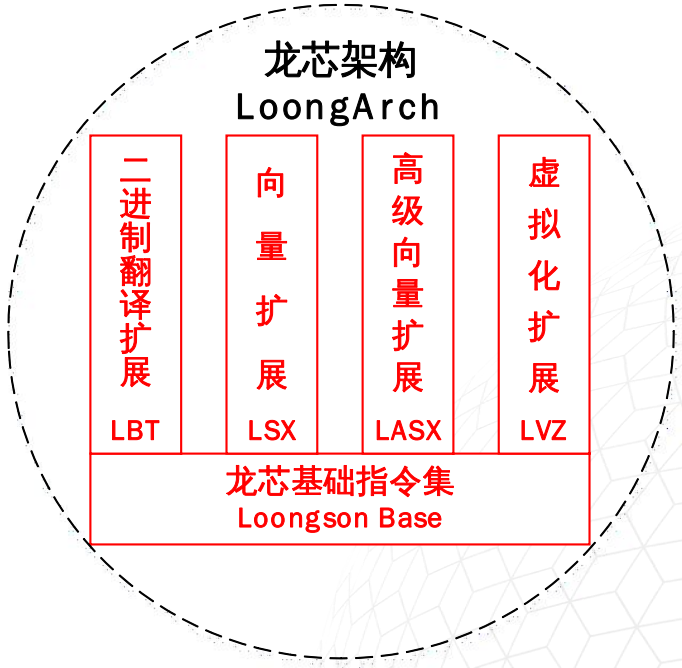
目录

龙芯中科
LOONGSON TECHNOLOGY

目录

龙芯自主指令系统架构 LoongArch

- “充分考虑兼容需求的自主指令系统”
- 遵循RISC指令设计风格
  - 32位定长指令编码风格
  - load/store架构
  - 32个通用寄存器
  - 32个浮点/向量寄存器
- 分为32位和64位两个版本，简称LA32和LA64架构
- 采用基础部分加扩展部分的整体架构
  - 基础指令集（~340条）
  - 二进制翻译扩展、向量扩展、虚拟化扩展



面向高校教学科研的LoongArch精简版

龙芯中科 LOONGSON TECHNOLOGY

- 基于LoongArch 32位版本的精简子集——LoongArch32 Reduced
- 用户态：保留典型应用中最常用的指令
  - 整数指令~50条，访存指令仅“基址+偏移”寻址方式且要求地址对齐
  - 浮点数指令可以不实现，也可以只实现单精度部分
  - 原子同步指令仅LL/SC，软件维护指令与数据Cache的数据一致性
- 核心态：支持主流类Unix操作系统
  - 仅包含PLV0和PLV3两个特权等级
  - 支持例外与中断，但入口为同一个
  - 支持TLB MMU，软件负责TLB重填
- 软件生态：维护一个独立的小系统
  - QEMU、GCC、……
  - PMON、Linux kernel、busybox

龙芯中科
LOONGSON TECHNOLOGY

目录

指令集用户态部分介绍提纲

- 数据类型
- 寄存器
- 指令编码格式
- 用户态指令功能速览
- C语言的机器表示

数据类型

龙芯中科 LOONGSON TECHNOLOGY

- 比特（bit，简记b）
- 字节（Byte，简记B，8bit）
- 半字（Halfword，简记H，16bit）
- 字　（Word，简记W，32bit）

	LA	MIPS I	RV32I
bit	bit	bit	bit
8bit	Byte, B	Byte, B	Byte, B
16bit	Halfword, H	Halfword, H	Halfword, H
32bit	Word, W	Word, W	Word, W

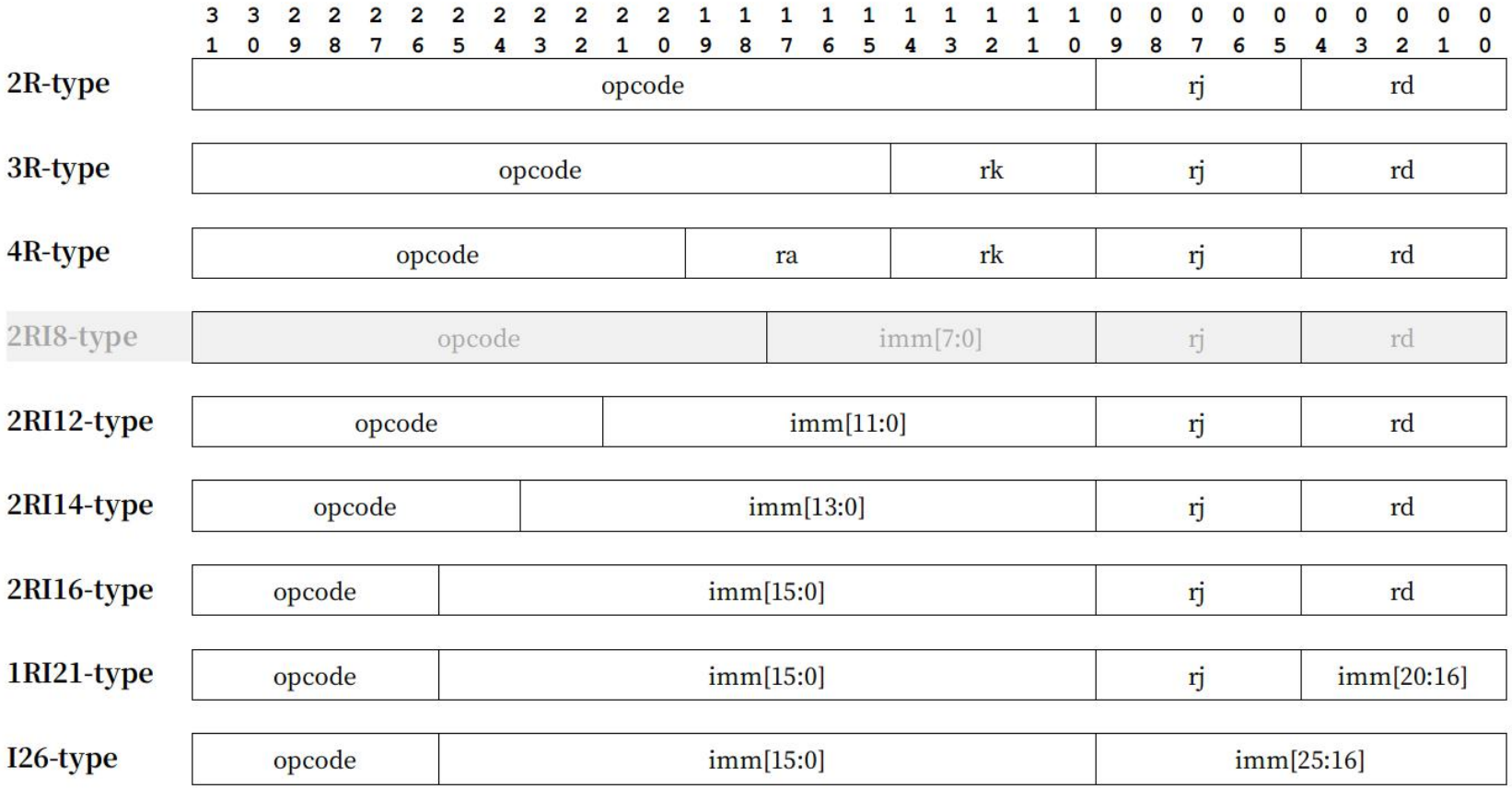
寄存器

- 通用寄存器（GR）
  - 32个32位宽寄存器，0号寄存器恒为0
- 程序计数器（PC）
  - 32位宽
  - 独立于通用寄存器，仅被转移指令修改
- 浮点指令操作浮点寄存器，其独立于通用寄存器

	LA	MIPS I	RV32I
通用寄存器	32个32位（0号恒为0）	32个32位（0号恒为0）	32个32位（0号恒为0）
程序计数器	独立于通用寄存器，仅被转移指令修改	独立于通用寄存器，仅被转移指令修改	独立于通用寄存器，仅被转移指令修改
浮点寄存器	独立于通用寄存器	独立于通用寄存器	独立于通用寄存器

LA精简版指令格式

- 指令都是32位长
- 指令均需4字节边界对齐
- 8种指令格式
  - 单一操作码域(opcode)
  - 寄存器操作数域(rd, rj, rk, ra)
    - rd—通常作为目的寄存器
    - rj—通常作为源寄存器1
    - rk—通常作为源寄存器2
    - ra—通常作为源寄存器3
  - 立即数域(imm)
    - (1+)5种不同的长度
    - 访存与整型计算：12位
    - CSR寻址：14位
    - 转移指令：16位、21位、26位
  - 所有指令至多一个目的寄存器
  - “两源一目的”的3R-type是最常见的指令格式；4R-type仅出现在浮点运算中。



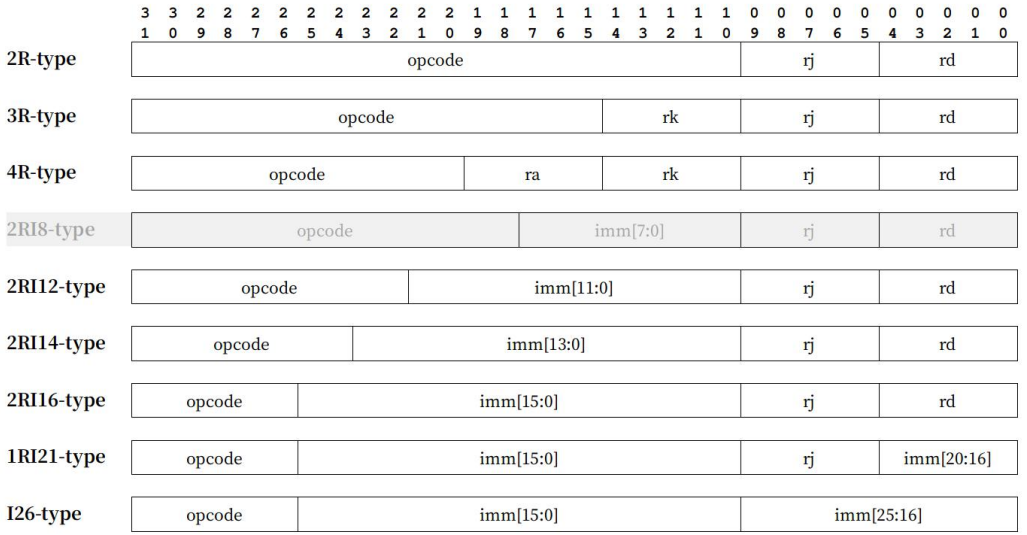
LA精简版指令格式举例

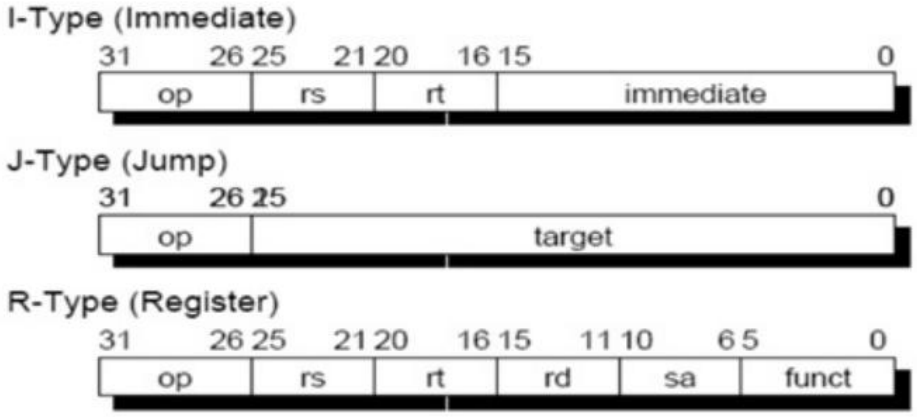
		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
ADD.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 rk rj rd	3R-type
FADD.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 fk fj fd	3R-type
FABS.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 fj fd	2R-type
SLTI	rd, rj, si12	0 0 0 0 0 0 1 0 0 0 si12 rj rd	2RI12-type
CSRXCHG	rd, rj, csr	0 0 0 0 0 1 0 0 csr rj!=0,1 rd	2RI14-type
FMADD.S	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 0 0 1 fa fk fj fd	4R-type
LU12I.W	rd, si20	0 0 0 1 0 1 0 si20 rd	1RI20-type
LL.W	rd, rj, si14	0 0 1 0 0 0 0 0 si14 rj rd	2RI14-type
BCEQZ	cj, offs	0 1 0 0 1 0 offs[15:0] 0 0 cj offs[20:16]	1RI21-type
B	offs	0 1 0 1 0 0 offs[15:0] offs[25:16]	I26-type
BEQ	rj, rd, offs	0 1 0 1 1 0 offs[15:0] rj rd	2RI16-type

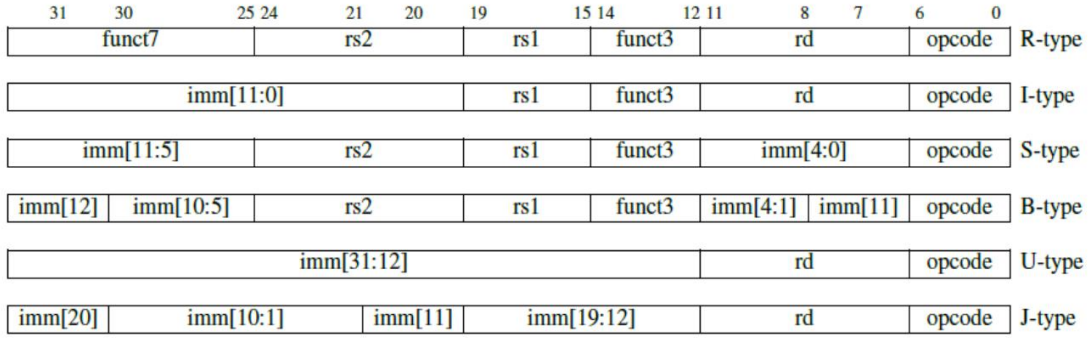
三种RISC指令格式比较

龙芯中科 LOONGSON TECHNOLOGY

- 三种RISC指令格式具有共性：
  - 定长[1]、对齐、指令格式种类少
  - 操作码域、寄存器域和立即数域
  - “两源一目的”三寄存器操作数格式最常用
  - 要读写的寄存器号出现的位置（基本）固定不变
    - RV最彻底，MIPS和LA有少量例外







注：
1. 严格来说，RISC-V的指令长度是16位的倍数，是一种具有RISC编码风格的变长指令编码。不过，教学中常用的RV32G所包含的指令都是32位定长的。

整数加减运算

龙芯中科 LOONGSON TECHNOLOGY

- **add.w (add word)** add.w rd, rj, rk GR[rd] = GR[rj] + GR[rk]
- **sub.w (subtract word)** sub.w rd, rj, rk GR[rd] = GR[rj] - GR[rk]
- **addi.w (add immediate word)** addi.w rd, rj, si12 GR[rd] = GR[rj] + sext32(si12)

LA	MIPS I	RV32I
add.w rd, rj, rk	addu rd, rs, rt	add rd, rs1, rs2
sub.w rd, rj, rk	subu rd, rs, rt	sub rd, rs1, rs2
addi.w rd, rj, si12	addiu rs, rs, si16	addi rd, rs1, si12

注：
- sext32(A)表示将数值A符号扩展到32位宽。例如sext32(0x01)=0x00000001，sext32(0x80)=0xffffff80。

整数比较运算

龙芯中科 LOONGSON TECHNOLOGY

- **slt (set less than)** slt rd, rj, rk GR[rd] = GR[rj] $<_{signed}$ GR[rk]
- **sltu (set less than unsigned)** sltu rd, rj, rk GR[rd] = GR[rj] $<_{unsigned}$ GR[rk]
- **slti (set less than immediate)** slt rd, rj, si12 GR[rd] = GR[rj] $<_{signed}$ sext32(si12)
- **sltui (set less than unsigned immediate)** slt rd, rj, si12 GR[rd] = GR[rj] $<_{unsigned}$ sext32(si12)

LA	MIPS I	RV32I
`slt rd, rj, rk`	`slt rd, rs, rt`	`slt rd, rs1, rs2`
`sltu rd, rj, rk`	`sltu rd, rs, rt`	`sltu rd, rs1, rs2`
`slti rd, rj, si12`	`slti rd, rs, si16`	`slti rd, rs1, si12`
`sltui rd, rj, si12`	`sltiu rd, rs, si16`	`sltiu rd, rs1, si12`

注：

- $<_{signed}$ 将两个操作数视作有符号数进行小于比较；$<_{unsigned}$ 将两个操作数视作无符号数进行小于比较。

逻辑位运算

龙芯中科 LOONGSON TECHNOLOGY

• **and (and)**	and rd, rj, rk	GR[rd] = GR[rj] & GR[rk]
• **or (or)**	or rd, rj, rk	GR[rd] = GR[rj] | GR[rk]
• **nor (not or)**	nor rd, rj, rk	GR[rd] = ~(GR[rj] | GR[rk])
• **xor (exclusive or)**	xor rd, rj, rk	GR[rd] = GR[rj] ^ GR[rk]
• **andi (and immediate)**	andi rd, rj, ui12	GR[rd] = GR[rj] & zext32(ui12)
• **ori (or immediate)**	ori rd, rj, ui12	GR[rd] = GR[rj] | zext32(ui12)
• **xori (exclusive or immediate)**	xori rd, rj, ui12	GR[rd] = GR[rj] ^ zext32(ui12)

LA	MIPS I	RV32I
and rd, rj, rk	and rd, rs, rt	and rd, rs1, rs2
or rd, rj, rk	or rd, rs, rt	or rd, rs1, rs2
nor rd, rj, rk	nor rd, rs, rt	N.A.
xor rd, rj, rk	xor rd, rs, rt	xor rd, rs1, rs2
andi rd, rj, ui12	andi rd, rs, ui16	andi rd, rs, si12
ori rd, rj, ui12	ori rd, rs, ui16	ordi rd, rs, si12
xori rd, rj, ui12	xori rd, rs, ui16	xori rd, rs, si12

注：
- zext32(A)将数值A零扩展到32位宽。例如zext32(0x01)=0x00000001，zext32(0x80)=0x00000080。

移位运算

龙芯中科 LOONGSON TECHNOLOGY

• **sll.w (shift left logic word)**	sll.w rd, rj, rk	GR[rd] = GR[rj] << GR[rk][4:0]
• **srl.w (shift right logic word)**	srl.w rd, rj, rk	GR[rd] = GR[rj] >> $_{logic}$ GR[rk][4:0]
• **sra.w (shift right arithmetic word)**	sra.w rd, rj, rk	GR[rd] = GR[rj] >>$_{arith}$ GR[rk][4:0]
• **slli.w (shift left logic immediate word)**	slli.w rd, rj, ui5	GR[rd] = GR[rj] << ui5
• **srli.w (shift right logic immediate word)**	srli.w rd, rj, ui5	GR[rd] = GR[rj] >>$_{logic}$ ui5
• **srai.w (shift right arithmetic immediate word)**	srai.w rd, rj, ui5	GR[rd] = GR[rj] >>$_{arith}$ ui5

LA	MIPS I	RV32I
sll.w rd, rj, rk	sllv rd, rt, rs	sll rd, rs1, rs2
srl.w rd, rj, rk	srlv rd, rt, rs	srl rd, rs1, rs2
sra.w rd, rj, rk	srav rd, rt, rs	sra rd, rs1, rs2
slli.w rd, rj, ui5	sll rd, rt, sa	slli rd, rs1, shamt
srli.w rd, rj, ui5	srl rd, rt, sa	srli rd, rs1, shamt
srai.w rd, rj, ui5	sra rd, rt, sa	srai rd, rs1, shamt

注：

- >>$_{logic}$ 表示逻辑右移，>>$_{arith}$ 表示算术右移。

立即数装载

- **lu12i (load upper from bit12 immediate)** lu12i rd, si20 GR[rd] = {si20, 12'b0}

LA	MIPS I	RV32I
`lu12i rd, si20`	`lui rd, si16`	`lui rd, si20`

- 与ori指令配合装载一个值落在[-2^{11}, $2^{12}-1$]之外的32位立即数IMM
  - lu12i $t0, IMM[31:12]
  - ori $t0, $t0, IMM[11:0]

- MIPS采用高、低两个16位的拆分风格，因其普通立即数运算指令的立即数是16位宽
- RV32I采用高20位、低12位的拆分风格，因其普通立即数运算指令的立即数是12位宽
  - RV32I不能只用“lui+ori”的指令对，因其ori指令的立即数采用的是符号扩展

PC相对计算

龙芯中科 LOONGSON TECHNOLOGY

- **pcaddu12i (pc add upper from bit12 immediate)** pcaddu12i rd, si20 GR[rd] = PC + {si20, 12'b0}

LA	MIPS I	RV32I
`pcaddu12i rd, si20`	`N.A.`	`auipc rd, si20`

- 与jirl、load/store指令配合完成PC相对32位偏移地址的跳转、访存
  - pcaddu12i $t0, IMMH #假设期望的PC相对32位偏移值为OFFSET，且为4的倍数
  - jirl $zero, $t0, IMML #OFFSET = {IMMH[19:0],12'b0} + sext32({IMML,2'b0}, 32}

  - pcaddu12i $t0, IMMH #假设期望的PC相对32位偏移值为OFFSET
  - ld.b $t1, $t0, IMML #OFFSET = {IMMH[19:0],12'b0} + sext32(IMML, 32)

乘除运算

龙芯中科 LOONGSON TECHNOLOGY

- **mul.w (multiply word signed)** mul.w rd, rj, rk GR[rd] = (GR[rj] $*_{signed}$ GR[rk])[31:0]
- **mulh.w (multiply high word signed)** mulh.w rd, rj, rk GR[rd] = (GR[rj] $*_{signed}$ GR[rk])[63:32]
- **mulh.wu (multiply high word unsigned)** mulh.wu rd, rj, rk GR[rd] = (GR[rj] $*_{unsigned}$ GR[rk])[63:32]
- **div.w (divide word signed)** div.w rd, rj, rk GR[rd] = GR[rj] $/_{signed}$ GR[rk]
- **mod.w (modulo word signed)** mod.w rd, rj, rk GR[rd] = GR[rj] $\%_{signed}$ GR[rk]
- **div.wu (divide word unsigned)** div.wu rd, rj, rk GR[rd] = GR[rj] $/_{unsigned}$ GR[rk]
- **mod.wu (modulo word unsigned)** mod.wu rd, rj, rk GR[rd] = GR[rj] $\%_{unsigned}$ GR[rk]

注：
- 取64位乘积低32位
- 取64位乘积高32位

LA	MIPS I	RV32M
mul.w rd, rj, rk	mult rs, rt	mul rd, rs1, rs2
mulh.w rd, rj, rk	mult rs, rt	mulh rd, rs1, rs2
mulh.wu rd, rj, rk	multu rs, rt	mulhu rd, rs1, rs2
div.w rd, rj, rk	div rs, rt	div rd, rs1, rs2
mod.w rd, rj, rk	div rs, rt	rem rd, rs1, rs2
div.wu rd, rj, rk	divu rs, rt	divu rd, rs1, rs2
mod.wu rd, rj, rk	divu rs, rt	remu rd, rs1, rs2

注：
- $*_{signed}$ 将两个操作数视作有符号数进行乘法；$*_{unsigned}$ 将两个操作数视作无符号数进行乘法。 $/_{signed}$ 将两个操作数视作有符号数进行除法取商，$\%_{signed}$ 将两个操作数视作有符号数进行除法取余；$/_{unsigned}$ 将两个操作数视作无符号数进行除法取商，$\%_{unsigned}$ 将两个操作数视作无符号数进行除法取余。

条件分支

龙芯中科 LOONGSON TECHNOLOGY

• **beq (branch on equal)**	beq rj, rd, off16	if (GR[rj]==GR[rd]) PC = TakenTgt
• **bne (branch on not equal)**	bne rj, rd, offs16	if (GR[rj]!=GR[rd]) PC = TakenTgt
• **blt (branch on less than signed)**	blt rj, rd, offs16	if (GR[rj]<$_{signed}$GR[rd]) PC = TakenTgt
• **bge (branch on greater than or equal signed)**	bge rj, rd, offs16	if (GR[rj]>=$_{signed}$GR[rd]) PC = TakenTgt
• **bltu (branch on less than unsigned)**	bltu rj, rd, offs16	if (GR[rj]<$_{unsigned}$GR[rd]) PC = TakenTgt
• **bgeu (branch on greater than or equal unsigned)**	bgeu rj, rd, offs16	if (GR[rj]>=$_{unsigned}$GR[rd]) PC = TakenTgt

TakenTgt = PC + sext32({off16, 2'b0})

LA	MIPS I	RV32I
beq rj, rd, offs16	beq rs, rt, off16	beq rs1, rs2, offs12
bne rj, rd, offs16	bne rs, rt, off16	bne rs1, rs2, offs12
blt rj, rd, offs16	N.A.	blt rs1, rs2, offs12
bge rj, rd, offs16	N.A.	bge rs1, rs2, offs12
bltu rj, rd, offs16	N.A.	bltu rs1, rs2, offs12
bgeu rj, rd, offs16	N.A.	bgeu rs1, rs2, offs12

没有延迟槽　　有延迟槽　　没有延迟槽

无条件相对PC跳转

- **b (branch)** b offs26 PC = BrTarget
- **bl (branch and link)** bl offs26 GR[1] = PC+4; PC = BrTarget

BrTarget = PC + sext32({off26, 2'b0})

LA	MIPS I	RV32I
b offs26	j target	jal zero, offs20
bl offs26	jal target	jal rd, offs20
没有延迟槽	有延迟槽	没有延迟槽

无条件间接跳转

- **jirl (jump indirect register link)** jirl rd, rj, offs16 GR[rd] = PC+4;
  PC = GR[rj]+sext32({offs16, 2'b0})

LA	MIPS I	RV32I
`jirl rd, rj, offs16`	`jr rs` `jalr rd, rs`	`jalr rd, rs1, offs12`
没有延迟槽	有延迟槽	没有延迟槽

几种常用的间接跳转指令是jirl指令的特殊形式

1）非link的间接跳转：jr rj，其实是 jirl r0, rj, 0；其中最常见的函数返回 jr ra，是jirl r0, r1, 0

2）常用间接函数调用（用jr ra指令返回的）：jirl rj，其实是 jirl r1, rj, 0。

普通访存

龙芯中科 LOONGSON TECHNOLOGY

- **ld.b (load byte signed)** ld.b rd, rj, si12 GR[rd] = sext32(MEM[GR[rj]+sext32(si12)][7:0])
- **ld.bu (load byte unsigned)** ld.bu rd, rj, si12 GR[rd] = zext32(MEM[GR[rj]+sext32(si12)][7:0])
- **ld.h (load halfword signed)** ld.h rd, rj, si12 GR[rd] = sext32(MEM[GR[rj]+sext32(si12)][15:0])
- **ld.hu (load halfword unsigned)** ld.hu rd, rj, si12 GR[rd] = zext32(MEM[GR[rj]+sext32(si12)][15:0])
- **ld.w (load word signed)** ld.w rd, rj, si12 GR[rd] = MEM[GR[rj]+sext32(si12)][31:0]
- **st.b (store byte)** st.b rd, rj, si12 MEM[GR[rj]+sext32(si12)][7:0] = GR[rd][7:0]
- **st.h (store halfword)** st.h rd, rj, si12 MEM[GR[rj]+sext32(si12)][15:0] = GR[rd][15:0]
- **st.w (store word)** st.w rd, rj, si12 MEM[GR[rj]+sext32(si12)][31:0] = GR[rd][31:0]

LA	MIPS I	RV32I
ld.b rd, rj, si12	lb rt, offs16(rs)	lb rd, offs12(rs1)
ld.bu rd, rj, si12	lbu rt, offs16(rs)	lbu rd, offs12(rs1)
ld.h rd, rj, si12	lh rt, offs16(rs)	lh rd, offs12(rs1)
ld.hu rd, rj, si12	lhu rt, offs16(rs)	lhu rd, offs12(rs1)
ld.w rd, rj, si12	lw rt, offs16(rs)	lw rd, offs12(rs1)
st.b rd, rj, si12	sb st, offs16(rs)	sb rs2, offs12(rs1)
st.h rd, rj, si12	sh st, offs16(rs)	sh rs2, offs12(rs1)
st.w rd, rj, si12	sw st, offs16(rs)	sw rs2, offs12(rs1)

原子访存

龙芯中科 LOONGSON TECHNOLOGY

- **ll.w (load linked word)** ll.w rd, rj, si14 GR[rd] = MEM[GR[rj]+sext32({si14,2'b0})][31:0]; LLbit=1
- **sc.w (store conditional word)** sc.w rd, rj, si14 if atomic_update_is_ok
  MEM[GR[rj]+sext32({si14,2'b0})][31:0] = GR[rd];
  GR[rd] = 1
  else
  GR[rd] = 0

LA	MIPS I	RV32A
ll.w rd, rj, si14	ll rt, offs16(rs)	lr.w rd, (rs1)
sc.w rd, rj, si14	sc rt, offs16(rs)	sc.w rd, rs2, (rs1)

```
//compare_and_swap  MEM[a0],  GR[a1] -- expected value, GR[a2] -- new value
1:      ori     t0, a2, 0           //move t0, a2
        ll.w    t1, a0, 0
        bne     t1, a1, 2f
        sc.w    t0, a0, 0
        beq     t0, r0, 1b
2:
```

预取指令

龙芯中科 LOONGSON TECHNOLOGY

- **preld (preload)** preld hint, rj, si12 Prefetch(hint, MEM[GR[rj]+sext(si12)])

LA	MIPS I	RV32I
`preld hint, rj, si12`	N.A.	N.A.

- 没有实现Cache时，preld的执行应视同nop；即使实现Cache，将preld实现为nop也不会导致正常程序功能出错。
- preld**不应**触发任何与地址错误相关的异常。

栅障指令

- **dbar(data barrier)** dbar hint Set a data barrier between load/store operations
- **ibar(instruction barrier)** ibar hint Set a memory barrier between previous load/store operations and following instruction fetching

LA	MIPS32	RV32I
dbar hint	sync	fence
ibar hint	N.A.	fence.i

- dbar的hint=0是必须实现的，其余hint可以按照hint=0实现。实现时，dbar前的load/store指令都彻底执行完成（执行效果全局可见）后，dbar后的load/store指令才能开始执行。
- ibar指令主要应用在有自修改代码的场景中。为降低硬件实现复杂度，LA32R相比于LA64要求软件负责自修改代码场景中数据Cache和指令Cache间的一致性维护，这就意味着ibar的实现能简化为：等到ibar前的访存类指令（含cacop指令）都执行完成退出流水线后，ibar后的指令才能开始取指。

系统调用及陷入

- **syscall** syscall code Cause a SYS exception
- **break** break code Cause a BRK exception

LA	MIPS I	RV32I
`syscall code`	`syscall`	`ecall`
`break code`	`break`	`ebreak`

时间读取

- **rdcntvl.w** rdcntvl.w rd read the 64-bit stable counter, wirte cnt[31:0] to GR[rd]
- **rdcntvh.w** rdcntvh.w rd read the 64-bit stable counter, wirte cnt[63:32] to GR[rd]
- **rdcntid** rdcntid rj read id of this stable counter

LA	MIPS I	RV32I
rdcntvl.w rd	N.A.	rdcyle rd
rdcntvh.w rd	N.A.	rdcyleh rd
rdcntid rj	N.A.	N.A.

- 龙芯架构32位精简版定义了一个恒定频率计时器，其主体是一个64位的计数器，称为Stable Counter。Stable Counter在复位后置为0，随后每个计数时钟周期自增1，当计数至全1时自动绕回至0继续自增。同时每个计时器都有一个软件可配置的全局唯一编号，称为Counter ID。
- 当处理器没有时钟变频设计时，恒定频率计时器可以直接使用处理器核的时钟。

C语言的机器表示

- C语言的控制流语句
  - 选择语句：if~else， switch~case
  - 循环语句：for，while，do~while
  - 辅助控制语句：break，continue，goto，return
- C语言的控制流语句在LoongArch汇编中映射为各种分支指令（B类）或其组合
  - 选择语句和循环语句映射为条件分支
  - 辅助控制语句映射为无条件分支或跳转（return语句）

LoongArch ABI

- 应用程序二进制接口（Application Binary Interface，简称ABI）
  - 数据表示和对齐
  - 寄存器使用
  - 函数调用
  - 栈布局
  - 目标文件和可执行文件格式
  - ……
- LoongArch ABI分为3种
  - LP32：　32位处理器上执行的32位程序
  - LPX32：　64位处理器上执行的64位程序，但指针和long整型的宽度为32位
  - LP64：　在64位处理器上执行的64位程序

LoongArch ABI寄存器使用约定

- LoongArch ABI有以下约定
  - a0-a7: 函数参数
  - v0/v1: 函数返回值，a0/a1别名
  - t*: 临时变量，子函数可改
  - s*: 子函数不改的变量
  - ra: 返回地址
  - tp: 线程指针
  - sp: 栈顶指针
  - fp: 栈帧指针

寄存器号	助记符	Saver
0	zero	-
1	ra	caller
2	tp	-
3	sp	callee
4-11	a0-a7 v0/v1=a0/a1	caller
12-20	t0-t8	caller
21	reserved	-
22	fp	callee
23-31	s0-s8	callee

流程控制语句 – 示例1

- 当条件表达式t0等于0（BEQZ）时，跳转到标号.L1
- 标号.L2为公用的程序出口

```
// C if ~ else
if (cond_exp)
  <then_statement>
else
  <else_statement>
```

```
# ASM if~else
    move        $t0, cond_exp
    beqz        $t0, .L1
    <then_statement>
    b   .L2
.L1:
    <else_statement>
.L2:
```

流程控制语句 – 示例2

```
// C for
int test_for(int a) {
  int sum = 0;
  int i = 0;
  for (i = 0; i < a; i++) {
    sum += i;
  }
  return sum;
}
```

```
test_for:
        or      $t0,$r0,$r0
        or      $t1,$r0,$r0
.L2:    blt     $t0,$a0,.L3
        or      $a0,$t1,$r0
        jr      $ra
.L3:    add.w   $t1,$t1,$t0
        addi.w  $t0,$t0,1
        b       .L2
```

```
// C while
int test_while(int a) {
  int sum = 0;
  int i = 0;
  while (i < a) {
    sum += i;
    i++;
  }
  return sum;
}
```

```
test_while:
        or      $t0,$r0,$r0
        or      $t1,$r0,$r0
.L2:    blt     $t0,$a0,.L3
        or      $a0,$t1,$r0
        jr      $ra
.L3:    add.w   $t1,$t1,$t0
        addi.w  $t0,$t0,1
        b       .L2
```

```
// C do-while
int test_dowhile(int a) {
  int sum = 0;
  int i = 0;
  do {
    sum += i;
    i++;
  } while (i < a);
  return sum;
}
```

```
test_dowhile:
        or      $t0,$r0,$r0
        or      $t3,$r0,$r0
.L2:    add.w   $t1,$t3,$t0
        addi.w  $t2,$t0,1
        or      $t3,$t1,$r0
        or      $t0,$t2,$r0
        blt     $t2,$a0,.L2
        or      $a0,$t1,$r0
        jr      $ra
```

流程控制语句 – 示例3

- switch-case采用跳转表实现

```
int st(int a, int b, int c)
{
  switch (a) {
    case 15:
        c = b & 0xf;
    case 10:
        return c + 50;
    case 12:
    case 17:
        return b + 50;
    case 14:
        return b;
    default:
        return a;
  }
}
```

```
    .text
st:
    addi.w      $t0,$a0, -10      //a-10
    sltiu       $t1,$t0, 8
    beqz        $t1, default      //(a-10)>=8
    la          $t2, jr_table
    slli.w      $t1, $t1, 2       //(a-10)*4
    add.w       $t1, $t1, $t2     //(a-10)*4+jr_table
    ld.w        $t0, $t1, 0
    jr          $t0
default:
    or          $a1,$a0,$r0
case_14:
    or          $a0,$a1,$r0
    jr          $ra         //return b for case_14,
                            //return a for default
case_15:
    andi        $a2,$a1,0xf //b & 0xf
case_10:
    addi.w      $a1,$a2,50  //c+50
    b           case_14
case_12_17:
    addi.w      $a1,$a1,50  //b+50
    b           case_14
```

```
# jump table
.section .rodata
.align   3
jr_table:
.word    case_10
.word    default
.word    case_12_17
.word    default
.word    case_14
.word    case_15
.word    default
.word    case_12_17
```

流程控制语句 – 示例4

- switch-case采用一串比较实现

```
int st(int a, int b, int c)
{
  switch (a) {
    case 15:
        c = b & 0xf;
    case 10:
        return c + 50;
    case 12:
    case 17:
        return b + 50;
    case 14:
        return b;
    default:
        return a;
  }
}
```

```
st:
        addi.w   $t0,$r0,14
        beq      $a0,$t0,.L7       //(a==14)?
        blt      $t0,$a0,.L3       //(a>14)?
        addi.w   $t0,$r0,10
        beq      $a0,$t0,.L4       //(a==10)?
        addi.w   $t0,$r0,12
        beq      $a0,$t0,.L5       //(a==12)?
        jr       $ra      //return a
.L3:    addi.w   $t0,$r0,15
        beq      $a0,$t0,.L6       //(a==15)?
        addi.w   $t0,$r0,17
        beq      $a0,$t0,.L5       //(a==17)?
        jr       $ra      //return a
.L6:    andi     $a2,$a1,0xf       //b & 0xf
.L4:    addi.w   $a0,$a2,50        //c + 50
        jr       $ra
.L5:    addi.w   $a0,$a1,50        //b + 50
        jr       $ra
.L7:    or       $a0,$a1,$r0       //return b
        jr       $ra
```

过程调用 – 示例

龙芯中科 LOONGSON TECHNOLOGY

- 过程调用流程
  - 调用者（Caller）将实参放入寄存器或栈中
  - 使用调用指令调用被调用者（Callee）
  - Callee在栈中分配自己所需的局部变量空间
  - 执行callee过程
  - Callee释放局部变量空间（将栈指针还原）
  - Callee使用JR返回调用者

```
int add(int a,int b)
{
    return a+b;
}
int ref()
{
    int t1 = 12;
    int t2 = 34;
    return add(t1,t2);
}
```

```
add:
      add.w   $a0, $a0, $a1  //a+b
      jr      $ra            //return
ref:
      addi.w $sp, $sp, -16  //stack allocate
      addi.w $a1, $r0, 34   //t2=34
      addi.w $a0, $r0, 12   //t1=12
      st.w   $ra, $sp, 4    //save $ra
      bl     add            //call add()
      ld.w   $ra, $sp, 4    //restore $ra
      addi.w $sp, $sp, 16   //stack release
      jr     $ra            //return
```

龙芯中科
LOONGSON TECHNOLOGY

目录

指令集特权态部分介绍提纲

- 特权等级
- 异常与中断
- 存储管理
- 特权指令与控制状态寄存器（Control Status Register, CSR）

特权等级

- 两个特权等级
  - 对应用户态的：PLV3（当CSR.CRMD.PLV[1]=3时）
  - 对应核心态的：PLV0（当CSR.CRMD.PLV=0时）
- 不同特权等级下的操作权限
  - PLV3：仅可执行用户态指令、不可访问CSR、仅可访问用户态(虚)地址空间、不可操作TLB
  - PLV0：可执行所有指令、可访问CSR、可访问整个地址空间、可操作TLB

LA	MIPS I	RV32I
PLV3 (CSR.CRMD.PLV=3)	User Mode (CP0.Status.ERL=0 && CP0.Status.EXL=0 && CP0.Status.KSU=2)	U-mode[2]
PLV0 (CSR.CRMD.PLV=0)	Kernel Mode (CP0.Status.ERL=1 || CP0.Status.EXL=1 || CP0.Status.KSU=0)	S-mode
		M-mode

注：
1. CSR是Control Status Register(控制状态寄存器)的简写。CSR.CRMD.PLV指代号为CRMD的CSR寄存器中PLV域。
2. RISC-V中CPU当前处于M、S还是U Mode虽然硬件是可以推导出来的，但是并没有CSR的相关域直接表征。

异常与中断

龙芯中科
LOONGSON TECHNOLOGY

- 异常的处理过程
- 异常具体定义
- 中断的相关定义
- 异常处理相关CSR

异常处理过程

龙芯中科 LOONGSON TECHNOLOGY

异常触发

① 异常处理准备
- 记录触发异常指令PC至CSR.ERA
- 硬件保存部分现场（PPLV←PLV, PIE←IE）
- 提升特权等级至最高（PLV←0）
- 记录其它异常相关信息（e.g. CSR.BADV）

② 确定异常来源，进入异常处理入口。
- 记录异常类型（CSR.ESTAT.Ecode、EsubCode）
- TLB重填异常入口（CSR.TLBRENTRY），其余异常入口（CSR.EENTRY）

I. 保存执行状态
II. 处理异常
III. 恢复执行状态
IV. 执行ertn指令返回
- 恢复硬件保存的那部分现场（PLV←PPLV, IE←PIE）
- 跳转到CSR.ERA所存的返回地址处

程序继续执行

异常具体定义（一）

Ecode	EsubCode	异常代号	异常类型	判定条件
0x0		INT	中断	接收到外部硬件中断、核间中断、内部软中断、定时器中断
0x1		PIL	load操作页无效	load指令访问的页表项无效
0x2		PIS	store操作页无效	store指令访问的页表项无效
0x3		PIF	取指操作页无效	取指操作访问的页表项无效
0x4		PME	页修改	store指令访问一个可写位和脏位不全为1的有效页表项
0x7		PPI	页特权等级不合规	访问的有效页表项的PLV等级权限高于CPU当前的PLV等级
0x8	0x0	ADEF	取指地址错	取指PC不对齐； 映射地址模式下，CPU当前处于PLV3，PC第31位为1且不落在任何有效的直接映射窗口中
0x8	0x1	ADEM	访存指令地址错	映射地址模式下，CPU当前处于PLV3，访存指令虚地址的第31位为1且不落在任何有效的直接映射窗口中
0x9		ALE	地址非对齐	非字节访存指令的地址不是自然对齐的

异常具体定义（二）

Ecode	EsubCode	异常代号	异常类型	判定及处理
0xB		SYS	系统调用	执行syscall指令
0xC		BRK	断点	执行break指令
0xD		INE	指令不存在	当前指令是一条未定义（/未实现）指令
0xE		IPE	指令特权等级错	CPU当前处于PLV3，执行特权指令
0xF		FPD	浮点指令未使能	CPU实现了浮点指令前提下，当CSR.EUEN.FPE=0时执行浮点指令
0x12		FPE	基础浮点运算异常	浮点运算过程中满足IEEE754规范中触发浮点运算异常的情况
0x3F		TLBR	TLB重填	映射地址模式下，访存地址不落在任何有效的直接映射窗口中，且在TLB中找不到对应的TLB表项

中断相关定义

代号	名称	状态位	局部使能	全局使能	来源	中断入口地址
SWI0	软件中断0	CSR.ESTAT.IS[0]	CSR.ECFG.LIE[0]	CSR.CRMD.IE	软件设置	CSR.EENTRY
SWI1	软件中断1	CSR.ESTAT.IS[1]	CSR.ECFG.LIE[1]			
HWI0	硬件中断0	CSR.ESTAT.IS[2]	CSR.ECFG.LIE[2]		外部中断控制器或设备	
HWI1	硬件中断1	CSR.ESTAT.IS[3]	CSR.ECFG.LIE[3]			
HWI2	硬件中断2	CSR.ESTAT.IS[4]	CSR.ECFG.LIE[4]			
HWI3	硬件中断3	CSR.ESTAT.IS[5]	CSR.ECFG.LIE[5]			
HWI4	硬件中断4	CSR.ESTAT.IS[6]	CSR.ECFG.LIE[6]			
HWI5	硬件中断5	CSR.ESTAT.IS[7]	CSR.ECFG.LIE[7]			
HWI6	硬件中断6	CSR.ESTAT.IS[8]	CSR.ECFG.LIE[8]			
HWI7	硬件中断7	CSR.ESTAT.IS[9]	CSR.ECFG.LIE[9]			
TI	定时器中断	CSR.ESTAT.IS[11]	CSR.ECFG.LIE[11]		核内定时器	
IPI	核间中断	CSR.ESTAT.IS[12]	CSR.ECFG.LIE[12]		其它核	

定时器中断相关定义

龙芯中科 LOONGSON TECHNOLOGY

- 中断来源：处理器核内一个倒计时计数器（timer_cnt），不超过32位宽。
  - 该计数器与rdcnt指令读取的计时器采用同一个时钟
- 中断标记：timer_cnt值为0时标记中断，CSR.ESTAT.IS[11]←1
- 中断清除：对CSR.TICLR.CLR位写1的动作
- timer_cnt配置：
  - 有自己的计数使能位（CSR.TCFG.En）
  - 倒计时的初始值软件可配置（CSR.TCFG.InitVal）
  - 倒计至0后的处理方式有两种（通过CSRT.TCFG.Periodic位配置）
    - 非周期性：停止工作
    - 周期性：自动复位到CSR.TCFG.InitVal配置的初始值后继续工作

注：
1. 停止工作可以简单理解为停止计数，但是这并不意味着timer_cnt一定要停在0值上，而且即使停在0值上也不会标记新的定时器中断。

异常相关CSR

助记符	编号	说明
CRMD	0x0	处理器当前运行模式及地址翻译模式、全局中断使能等配置信息
PRMD	0x1	触发当前普通异常的现成的运行模式、全局中断使能等配置信息
EUEN	0x2	扩展部件的使能控制
ECFG	0x4	局部中断使能配置信息
ESTAT	0x5	记录异常和中断发生原因
ERA	0x6	普通异常处理返回地址
BADV	0x7	记录触发地址相关异常的访存虚地址
EENTRY	0xC	配置普通异常处理入口地址
SAVE0~3	0x30~0x33	保存临时数据
TID	0x40	恒定频率定时器相关控制状态寄存器
TCFG	0x41	
TVAL	0x42	
TICLR	0x44	

存储管理

龙芯中科
LOONGSON TECHNOLOGY

- 虚实地址翻译模式
- 直接映射地址翻译模式
- 页表映射地址翻译模式

虚实地址翻译模式

- 直接地址翻译模式（CSR.CRMD.DA=1且CSR.CRMD.PG=0）
- 映射地址翻译模式（CSR.CRMD.DA=0且CSR.CRMD.PG=1）
  - 直接映射地址翻译模式（翻译规则来自于直接映射配置窗口CSR.DMW0~1）
  - 页表映射地址翻译模式（翻译规则来自于页表）

- 直接地址翻译模式下，物理地址直接等于虚地址

直接映射地址翻译模式

```
for (i=0; i<2; i++) {
  if (CSR.DMW[i].PLV[curr_plv] && vaddr[31:29]==CSR.DMW[i].VSEG) {
    dmw_hit = TRUE;
    paddr = {CSR.DMW[i].PSEG, vaddr[28:0]};
    mat = CSR.DMW[i].MAT;
  }
}
```

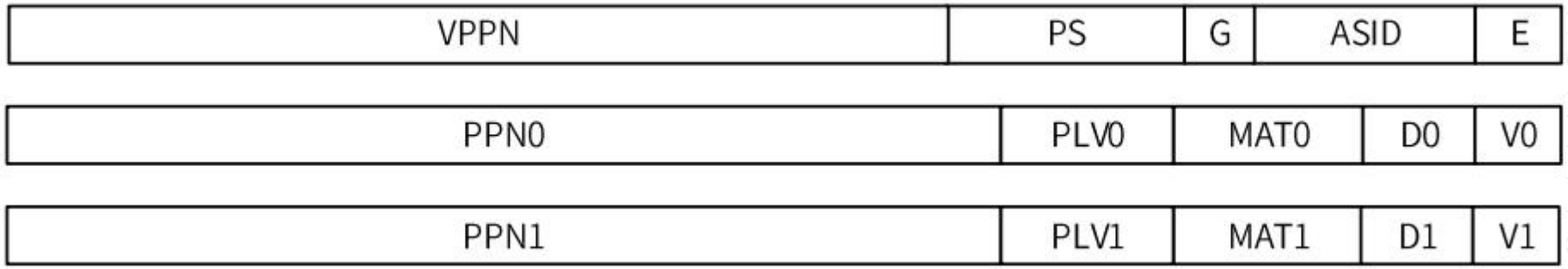
位	名字	读写	描述
0	PLV0	RW	为 1 表示在特权等级 PLV0 下可以使用该窗口的配置进行直接映射地址翻译。
2:1	0	R0	保留域。读返回 0，且软件不允许改变其值。
3	PLV3	RW	为 1 表示在特权等级 PLV3 下可以使用该窗口的配置进行直接映射地址翻译。
5:4	MAT	RW	虚地址落在该映射窗口下访存操作的存储访问类型。
24:6	0	R0	保留域。读返回 0，且软件不允许改变其值。
27:25	PSEG	RW	直接映射窗口的物理地址的[31:29]位。
28	0	R0	保留域。读返回 0，且软件不允许改变其值。
31:29	VSEG	RW	直接映射窗口的虚地址的[31:29]位。

页表映射地址翻译模式

- TLB结构
- TLB查找与虚实地址翻译
- TLB维护相关CSR
- TLB维护相关指令

TLB结构

- TLB采用全相联查找表的组织形式
- 每一个TLB表项包含比较部分和物理转换部分，后者存有一对奇偶相邻页表的物理转换信息
  - 存在位(E)，1bit
  - 地址空间标识(ASID)，10bit
  - 全局标志位(G)，1bit
  - 页大小(PS)，6bit
  - 虚双页号(VPPN)，19bit
  - 偶页有效位(V0)，1bit
  - 偶页脏位(D0)，1bit
  - 偶页存储访问类型(MAT0)，2bit
  - 偶页特权等级（PLV0)，2bit
  - 偶页物理页号(PPN0)，(PALEN-12)bit
  - 奇页有效位(V1)，1bit
  - 奇页脏位(D1)，1bit
  - 奇页存储访问类型(MAT1)，2bit
  - 奇页特权等级（PLV1)，2bit
  - 奇页物理页号(PPN1)，(PALEN-12)bit



TLB查找与虚实地址翻译过程（一）

```
tlb_found = 0
for i in range(TLB_ENTRIES) :
        if (TLB[i].E==1) and
         ((TLB[i].G==1) or (TLB[i].ASID==CSR.ASID.ASID)) and
         (TLB[i].VPPN[VALEN-1: TLB[i].PS+1]==va[VALEN-1: TLB[i].PS+1]) :
                if (tlb_found==0) :
                        tlb_found = 1
                        found_ps = TLB[i].PS
                        if (va[found_ps]==0) :
                                found_v = TLB[i].V0
                                found_d = TLB[i].D0
                                found_mat = TLB[i].MAT0
                                found_plv = TLB[i].PLV0
                                found_ppn = TLB[i].PPN0
                        else :
                                found_v = TLB[i].V1
                                found_d = TLB[i].D1
                                found_mat = TLB[i].MAT1
                                found_plv = TLB[i].PLV1
                                found_ppn = TLB[i].PPN1
                else:
                        #出现多项命中，处理器运行结果不确定
```

TLB查找与虚实地址翻译过程（二）

龙芯中科 LOONGSON TECHNOLOGY

```
if (tlb_found==0) :
        SignalException(TLBR)                   #报TLB重填例外

if (found_v==0) :
        case mem_type :
                FETCH : SignalException(PIF)                    #报取指操作页无效例外
                LOAD  : SignalException(PIL)                    #报load操作页无效例外
                STORE : SignalException(PIS)                    #报store操作页无效例外
elif (plv > found_plv) :
        SignalException(PPE)                    #报页特权等级不合规例外
elif (mem_type==STORE) and (found_d==0)) :          #禁止写允许检查功能未开启
        SignalException(PME)                    #报页修改例外
else :
        pa = {found_ppn[PALEN-1:found_ps], va[found_ps-1:0]}
        mat = found_mat
```

TLB维护相关CSR

- TLB访问交互接口相关
  - TLBEHI ：TLB指令操作时与TLB表项高位部分虚页号相关的信息
  - TLBELO0、TLBELO1 ：TLB指令操作时TLB表项低位部分0、1两个物理页号等相关的信息
  - TLBIDX ：TLB指令操作TLB时相关的索引值、页大小等信息
  - ASID ：访存操作和TLB指令所用的地址空间标识符（ASID）信息
- TLB地址翻译各例外相关
  - TLBRENTRY ：用于配置TLB重填例外的入口地址
  - BADV ：引发TLB地址翻译相关异常的虚地址
- 软件页表遍历相关
  - PGDL ：用于配置低半地址空间的全局目录的基址
  - PGDH ：用于配置高半地址空间的全局目录的基址
  - PGD ：当前CSR.BADV中出错虚地址所对应的全局目录基址信息

TLB维护相关指令（一）

龙芯中科 LOONGSON TECHNOLOGY

- **tlbsrch**
  - 使用CSR.ASID和CSR.TLBEHI的信息去查询TLB
  - 命中：CSR.TLBIDX.Index←命中项索引值，CSR.TLBIDX.NUL←0；不命中：CSR.TLBIDX.NUL←1
- **tlbrd**
  - 用CSR.TLBIDX.Index域的值作为索引值去读取TLB中的指定项
  - 读出TLB项E=1：CSR.TLBIDX.NUL←0，CSR.TLBEHI、CSR.TLBELO0、1和CSR.TLBIDX.PS写入读出值
  - 读出TLB项E=0：CSR.TLBIDX.NUL←1，CSR.TLBEHI、CSR.TLBELO0、1和CSR.TLBIDX.PS不变或置0
- **tlbwr**
  - 将TLB访问交互接口相关CSR中所存放的页表项信息写入到TLB中**索引位置为CSR.TLBIDX.Index**的那一项
  - CSR.TLBIDX.NUL=0：填入一个有效TLB项，值来自CSR.TLBEHI、CSR.TLBELO0、1和CSR.TLBIDX.PS
  - CSR.TLBIDX.NUL=1：填入一个无效TLB项
- **tlbfill**
  - 将TLB访问交互接口相关CSR中所存放的页表项信息写入到TLB中**索引位置为随机值**的一项中
  - 其余操作同TLBRD指令

TLB维护相关指令（二）

- invtlb　op, rj, rk
  - 用于无效TLB中的内容，以维持TLB与内存之间页表数据的一致性

op	操作
0x0	清除所有页表项
0x1	清除所有页表项。此时操作效果与op=0完全一致
0x2	清除所有G=1的页表项
0x3	清除所有G=0的页表项
0x4	清除所有G=0，且ASID等于GR[rj][9:0]的页表项
0x5	清除G=0，ASID等于GR[rj][9:0]且VA等于GR[rk][31:0]的页表项
0x6	清除所有G=1或者ASID等于GR[rj][9:0]且VA等于GR[rk][31:0]的页表项

LoongArch32位精简版指令简明列表

ADD.W	rd, rj, rk
ADDI.W	rd, rj, si12
SUB.W	rd, rj, rk
LU12I.W	rd, si20
PCADDU12I	rd, si20
SLT	rd, rj, rk
SLTU	rd, rj, rk
SLTI	rd, rj, si12
SLTUI	rd, rj, si12

SLL.W	rd, rj, rk
SRL.W	rd, rj, rk
SRA.W	rd, rj, rk
SLLI.W	rd, rj, ui5
SRLI.W	rd, rj, ui5
SRAI.W	rd, rj, ui5

MUL.W	rd, rj, rk
MULH.W	rd, rj, rk
MULH.WU	rd, rj, rk
DIV.W	rd, rj, rk
MOD.W	rd, rj, rk
DIV.WU	rd, rj, rk
MOD.WU	rd, rj, rk

NOR	rd, rj, rk
AND	rd, rj, rk
OR	rd, rj, rk
XOR	rd, rj, rk
ORN	rd, rj, rk
ANDN	rd, rj, rk
ANDI	rd, rj, ui12
ORI	rd, rj, ui12
XORI	rd, rj, ui12

LD.B	rd, rj, si12
LD.H	rd, rj, si12
LD.W	rd, rj, si12
ST.B	rd, rj, si12
ST.H	rd, rj, si12
ST.W	rd, rj, si12
LD.BU	rd, rj, si12
LD.HU	rd, rj, si12
LL.W	rd, rj, si14
SC.W	rd, rj, si14
PRELD	hint, rj, si12
DBAR	hint
IBAR	hint

JIRL	rd, rj, offs
B	offs
BL	offs
BEQ	rj, rd, offs
BNE	rj, rd, offs
BLT	rj, rd, offs
BGE	rj, rd, offs
BLTU	rj, rd, offs
BGEU	rj, rd, offs

RDCNTVL.W	rd
RDCNTVH.W	rd
RDCNTID	rj
BREAK	code
SYSCALL	code

CSRRD	rd, csr
CSRWR	rd, csr
CSRXCHG	rd, rj, csr
CACOP	code, rj, si12
TLBSRCH	
TLBRD	
TLBWR	
TLBFILL	
INVTLB	op, rj, rk
ERTN	
IDLE	hint

龙芯架构32位精简版开发资源一览

龙芯中科 LOONGSON TECHNOLOGY

- **文档及学习资料**
  - 《龙芯架构32位精简版参考手册》：https://loongson.cn/FileShow
  - 《计算机体系结构基础》(第3版)：https://foxsen.github.io/archbase/
- **开发工具与基础软件**
  - GCC交叉编译工具链：https://gitee.com/loongson-edu/la32r-toolchains
  - QEMU：https://gitee.com/loongson-edu/la32r-QEMU
  - NEMU：https://gitee.com/wwt_panache/la32r-nemu
  - PMON：https://gitee.com/loongson-edu/la32r-pmon
  - Linux kernel：https://gitee.com/loongson-edu/la32r-Linux
- **开发平台**
  - 开源SoC开发平台chiplab：https://gitee.com/loongson-edu/chiplab
    - 功能测试用例(func)及验证环境、随机指令序列仿真验证环境、Linux kernel启动仿真验证环境
    - 参考SoC设计代码及配套FPGA工程文件

龙芯中科
LOONGSON TECHNOLOGY

为人民做芯芯