龙芯架构 ELF psABI 规范

龙芯中科技术股份有限公司

Version 1.00

目录

寄存器使用约定

Table 1. 通用寄存器使用约定

名称	别名	用途	在调用中是否保留
`$r0`	`$zero`	常数 `0`	（常数）
`$r1`	`$ra`	返回地址	否
`$r2`	`$tp`	线程指针	（不可分配）
`$r3`	`$sp`	栈指针	是
`$r4` - `$r5`	`$a0` - `$a1`	传参寄存器、返回值寄存器	否
`$r6` - `$r11`	`$a2` - `$a7`	传参寄存器	否
`$r12` - `$r20`	`$t0` - `$t8`	临时寄存器	否
`$r21`		保留	（不可分配）
`$r22`	`$fp` / `$s9`	栈帧指针 / 静态寄存器	是
`$r23` - `$r31`	`$s0` - `$s8`	静态寄存器	是

Table 2. 浮点寄存器使用约定

名称	别名	用途	在调用中是否保留
`$f0` - `$f1`	`$fa0` - `$fa1`	传参寄存器、返回值寄存器	否
`$f2` - `$f7`	`$fa2` - `$fa7`	传参寄存器	否
`$f8` - `$f23`	`$ft0` - `$ft15`	临时寄存器	否
`$f24` - `$f31`	`$fs0` - `$fs7`	静态寄存器	是

临时寄存器也被称为调用者保存寄存器。 静态寄存器也被称为被调用者保存寄存器。

返回值寄存器的别名写法

在一些早期的 LoongArch 汇编代码中，您可能会见到形如 `$v0` `$v1` `$fv0` `$fv1` 的寄存器写法：这些名字分别等价于 `$a0` `$a1` `$fa0` `$fa1` 。 这些别名最初是仿照 MIPS 的分立传参、返回值寄存器写法而设计的。 由于 LoongArch 实际并没有专门的返回值寄存器，这种写法反而会造成误解， 因而不建议使用。

由于各下游项目的实现细节差异， 给一个寄存器赋予多个 ABI 名字并不一定是简单的事情。 新写作的处理 LoongArch 汇编语言的程序不应当实现该套别名。 可移植的 LoongArch 汇编代码不应当使用该套别名。

NOTE

对于龙芯公司提供的工具链组件，迁移流程为：

设本规范生效时相应组件的当前版本为 N，

1. 在版本 N 及其稳定分支（补丁版本）保留支持，
2. 在版本 N+1 对该用法进行警告，
3. 在版本 N+2 删除该用法的支持。

对于这些组件相应的上游项目，已进入上游的那部分如果存在对该用法的支持，则按上述流程进行，“版本 N”理解为第一次加入 LoongArch 支持的那个正式发布版本。 对于暂未进入上

> 游，且不与预期必须使用该用法的其他组件交互的组件，上游版本将自始不支持该用法。

C 语言数据类型规格

Table 3. LP64 数据模型 （对应基础 *ABI* 类型：`lp64d` `lp64f` `lp64s`）

标量类型	大小（字节）	对齐（字节）
`bool` / `_Bool`	1	1
`unsigned char` / `char`	1	1
`unsigned short` / `short`	2	2
`unsigned int` / `int`	4	4
`unsigned long` / `long`	8	8
`unsigned long long` / `long long`	8	8
指针类型	8	8
`float`	4	4
`double`	8	8
`long double`	16	16

Table 4. ILP32 数据模型 （对应基础 *ABI* 类型：`ilp32d` `ilp32f` `ilp32s`）

标量类型	大小（字节）	对齐（字节）
`bool` / `_Bool`	1	1
`unsigned char` / `char`	1	1
`unsigned short` / `short`	2	2
`unsigned int` / `int`	4	4
`unsigned long` / `long`	4	4
`unsigned long long` / `long long`	8	8
指针类型	4	4
`float`	4	4
`double`	8	8
`long double`	16	16

对于任何基础 ABI 类型，`char` 默认是有符号类型。

ELF 目标文件

本节内容中关于 ELF 目标文件的通用格式定义 均参考 最新版本的 SysV gABI。

`EI_CLASS`: ELF 文件格式

EI_CLASS	枚举值	含义
`ELFCLASS32`	`1`	32 位 ELF 格式
`ELFCLASS64`	`2`	64 位 ELF 格式

`e_machine`: 体系结构 ID

`LoongArch (258)`

`e_flags`: ABI 类型和版本标记

`[31:8]` 位	`[7:6]` 位	`[5:3]` 位	`[2:0]` 位
（保留）	ABI 版本	ABI 扩展特性	基础 ABI 类型

`e_flags[7:0]` 完整标记了 ELF 目标文件使用的 ABI 类型。

Table 5. 基础 *ABI* 类型标记

基础 **ABI** 名称	枚举值 (`e_flags[2:0]`)	含义
	`0x0`	保留值
`lp64s`	`0x1`	使用 64 位通用寄存器和栈传参， 数据模型为 LP64（`long` 和指针类型宽度为64位，`int` 为32位）
`lp64f`	`0x2`	使用 64 位通用寄存器，32位浮点寄存器和栈传参，数据模型为 LP64（`long` 和指针类型宽度为64位，`int` 为32位）
`lp64d`	`0x3`	使用 64 位通用寄存器，64位浮点寄存器和栈传参，数据模型为 LP64（`long` 和指针类型宽度为64位，`int` 为32位）
	`0x4`	保留值
`ilp32s`	`0x5`	使用 32 位通用寄存器和栈传参， 数据模型为 ILP32（`int`，`long` 和指针类型宽度为32位）
`ilp32f`	`0x6`	使用 32 位通用寄存器，32位浮点寄存器和栈传参，数据模型为 ILP32（`int`，`long` 和指针类型宽度为32位）
`ilp32d`	`0x7`	使用 32 位通用寄存器，64位浮点寄存器和栈传参，数据模型为 ILP32（`int`，`long` 和指针类型宽度为32位）

Table 6. ABI 扩展特性标记

ABI 扩展特性名称	枚举值 (`e_flags[5:3]`)	含义
`base`	`0x0`	默认，无扩展特性
	`0x1` - `0x7`	保留值

`e_flags[7:6]` 标记了 ELF 目标文件使用的 ABI 版本。

Table 7. ABI 版本标记

ABI 版本	枚举值	描述
`v0`	`0x0`	支持具有栈操作语义的重定位类型
`v1`	`0x1`	按需保留
	`0x2` `0x3`	保留值

重定位类型

Table 8. ELF 重定位类型

枚举值	名称	描述	语义
0	R_LARCH_NONE		
1	R_LARCH_32	动态符号地址解析	*(int32_t *) PC = RtAddr + A
2	R_LARCH_64	动态符号地址解析	*(int64_t *) PC = RtAddr + A
3	R_LARCH_RELATIVE	模块动态加载地址修正	*(void **) PC = B + A
4	R_LARCH_COPY	可执行映像数据动态填充	memcpy (PC, RtAddr, sizeof (sym))
5	R_LARCH_JUMP_SLOT	PLT 跳转支持	由具体实现定义
6	R_LARCH_TLS_DTPMOD32	TLS-GD 动态重定位支持	*(int32_t *) PC = ID of module defining sym
7	R_LARCH_TLS_DTPMOD64	TLS-GD 动态重定位支持	*(int64_t *) PC = ID of module defining sym
8	R_LARCH_TLS_DTPREL32	TLS-GD 动态重定位支持	*(int32_t *) PC = DTV-relative offset for sym
9	R_LARCH_TLS_DTPREL64	TLS-GD 动态重定位支持	*(int64_t *) PC = DTV-relative offset for sym
10	R_LARCH_TLS_TPREL32	TLS-IE 动态重定位支持	*(int32_t *) PC = T
11	R_LARCH_TLS_TPREL64	TLS-IE 动态重定位支持	*(int64_t *) PC = T
12	R_LARCH_IRELATIVE	本地间接跳转解析	*(void **) PC = (((void *)(*)()) (B + A)) ()
… 动态链接器保留项			
20	R_LARCH_MARK_LA	标记 la.abs 宏指令	静态填充符号绝对地址
21	R_LARCH_MARK_PCREL	标记外部标签跳转	静态填充符号地址偏移量
22	R_LARCH_SOP_PUSH_PCREL	将符号相对地址压栈	push (S - PC + A)
23	R_LARCH_SOP_PUSH_ABSOLUTE	将常数或绝对地址压栈	push (S + A)
24	R_LARCH_SOP_PUSH_DUP	复制栈顶元素	opr1 = pop (), push (opr1), push (opr1)

枚举值	名称	描述	语义
25	R_LARCH_SOP_PUSH_GPREL	将符号的 GOT 表项偏移量压栈	push (G)
26	R_LARCH_SOP_PUSH_TLS_TPREL	将 TLS-LE 偏移量压栈	push (T)
27	R_LARCH_SOP_PUSH_TLS_GOT	将 TLS-IE 偏移量压栈	push (IE)
28	R_LARCH_SOP_PUSH_TLS_GD	将 TLS-GD 偏移量压栈	push (GD)
29	R_LARCH_SOP_PUSH_PLT_PCREL	将符号 PLT stub 的地址偏移量压栈	push (PLT - PC)
30	R_LARCH_SOP_ASSERT	断言栈顶元素为真	assert (pop ())
31	R_LARCH_SOP_NOT	栈顶运算	push (!pop ())
32	R_LARCH_SOP_SUB	栈顶运算	opr2 = pop (), opr1 = pop (), push (opr1 - opr2)
33	R_LARCH_SOP_SL	栈顶运算	opr2 = pop (), opr1 = pop (), push (opr1 << opr2)
34	R_LARCH_SOP_SR	栈顶运算	opr2 = pop (), opr1 = pop (), push (opr1 >> opr2)
35	R_LARCH_SOP_ADD	栈顶运算	opr2 = pop (), opr1 = pop (), push (opr1 + opr2)
36	R_LARCH_SOP_AND	栈顶运算	opr2 = pop (), opr1 = pop (), push (opr1 & opr2)
37	R_LARCH_SOP_IF_ELSE	栈顶运算	opr3 = pop (), opr2 = pop (), opr1 = pop (), push (opr1 ? opr2 : opr3)
38	R_LARCH_SOP_POP_32_S_10_5	指令立即数重定位	opr1 = pop (), (*(uint32_t *) PC) [14 ... 10] = opr1 [4 ... 0] 带 5 位有符号数溢出检测功能
39	R_LARCH_SOP_POP_32_U_10_12	指令立即数重定位	opr1 = pop (), (*(uint32_t *) PC) [21 ... 10] = opr1 [11 ... 0] 带 12 位无符号数溢出检测功能

枚举值	名称	描述	语义
40	R_LARCH_SOP_POP_32_S_10_12	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [21 ... 10] = opr1 [11 ... 0]` 带 12 位有符号数溢出检测功能
41	R_LARCH_SOP_POP_32_S_10_16	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [15 ... 0]` 带 16 位有符号数溢出检测功能
42	R_LARCH_SOP_POP_32_S_10_16_S2	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]` 带 18 位有符号数溢出和4字节对齐检测功能
43	R_LARCH_SOP_POP_32_S_5_20	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [24 ... 5] = opr1 [19 ... 0]` 带 20 位有符号数溢出检测功能

44	R_LARCH_SOP_POP_32_S_0_5_10_16_S2	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [4 ... 0] = opr1 [22 ... 18],`
			
			`(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]`
			
			带 23 位有符号数溢出和4字节对齐检测功能

45	R_LARCH_SOP_POP_32_S_0_10_10_16_S2	指令立即数重定位	`opr1 = pop (), (*(uint32_t *) PC) [9 ... 0] = opr1 [27 ... 18],`
			
			`(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]`
			
			带 28 位有符号数溢出和4字节对齐检测功能

46	R_LARCH_SOP_POP_32_U	指令修正	`(*(uint32_t *) PC) = pop ()` 带 32 位无符号数溢出检测功能
47	R_LARCH_ADD8	8 位原地加法	`*(int8_t *) PC += S + A`
48	R_LARCH_ADD16	16 位原地加法	`*(int16_t *) PC += S + A`

枚举值	名称	描述	语义
49	R_LARCH_ADD24	24 位原地加法	*(int24_t *) PC += S + A
50	R_LARCH_ADD32	32 位原地加法	*(int32_t *) PC += S + A
51	R_LARCH_ADD64	64 位原地加法	*(int64_t *) PC += S + A
52	R_LARCH_SUB8	8 位原地减法	*(int8_t *) PC -= S + A
53	R_LARCH_SUB16	16 位原地减法	*(int16_t *) PC -= S + A
54	R_LARCH_SUB24	24 位原地减法	*(int24_t *) PC -= S + A
55	R_LARCH_SUB32	32 位原地减法	*(int32_t *) PC -= S + A
56	R_LARCH_SUB64	64 位原地减法	*(int64_t *) PC -= S + A
57	R_LARCH_GNU_VTINHERIT	GNU C++ vtable 支持	
58	R_LARCH_GNU_VTENTRY	GNU C++ vtable 支持	

动态链接器路径

Table 9. 标准动态链接器路径列表：

基础 **ABI** 类型	**ABI** 扩展特性	操作系统 / **C** 库	**Glibc** 动态链接器路径
`lp64d`	`base`	Linux, Glibc	`/lib64/ld-linux-loongarch-lp64d.so.1`
`lp64f`	`base`	Linux, Glibc	`/lib64/ld-linux-loongarch-lp64f.so.1`
`lp64s`	`base`	Linux, Glibc	`/lib64/ld-linux-loongarch-lp64s.so.1`
`ilp32d`	`base`	Linux, Glibc	`/lib32/ld-linux-loongarch-ilp32d.so.1`
`ilp32f`	`base`	Linux, Glibc	`/lib32/ld-linux-loongarch-ilp32f.so.1`
`ilp32s`	`base`	Linux, Glibc	`/lib32/ld-linux-loongarch-ilp32s.so.1`