

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 1
 - 返回值寄存器的别名写法 1
- C 语言数据类型规格 3
- ELF 目标文件 4
 - EI_CLASS**: ELF 文件格式 4
 - e_machine**: 体系结构 ID 4
 - e_flags**: ABI 类型和版本标记 4
- 重定位类型 6
- 动态链接器路径 10

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
<code>\$r0</code>	<code>\$zero</code>	常数 0	(常数)
<code>\$r1</code>	<code>\$ra</code>	返回地址	否
<code>\$r2</code>	<code>\$tp</code>	线程指针	(不可分配)
<code>\$r3</code>	<code>\$sp</code>	栈指针	是
<code>\$r4 - \$r5</code>	<code>\$a0 - \$a1</code>	传参寄存器、返回值寄存器	否
<code>\$r6 - \$r11</code>	<code>\$a2 - \$a7</code>	传参寄存器	否
<code>\$r12 - \$r20</code>	<code>\$t0 - \$t8</code>	临时寄存器	否
<code>\$r21</code>		保留	(不可分配)
<code>\$r22</code>	<code>\$fp / \$s9</code>	栈帧指针 / 静态寄存器	是
<code>\$r23 - \$r31</code>	<code>\$s0 - \$s8</code>	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
<code>\$f0 - \$f1</code>	<code>\$fa0 - \$fa1</code>	传参寄存器、返回值寄存器	否
<code>\$f2 - \$f7</code>	<code>\$fa2 - \$fa7</code>	传参寄存器	否
<code>\$f8 - \$f23</code>	<code>\$ft0 - \$ft15</code>	临时寄存器	否
<code>\$f24 - \$f31</code>	<code>\$fs0 - \$fs7</code>	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。

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，

NOTE

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 (<code>e_flags[5:3]</code>)	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	<code>*(int32_t *) PC = RtAddr + A</code>
2	R_LARCH_64	动态符号地址解析	<code>*(int64_t *) PC = RtAddr + A</code>
3	R_LARCH_RELATIVE	模块动态加载地址修正	<code>*(void **) PC = B + A</code>
4	R_LARCH_COPY	可执行映像数据动态填充	<code>memcpy (PC, RtAddr, sizeof (sym))</code>
5	R_LARCH_JMP_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	<code>*(int32_t *) PC = ID of module defining sym</code>
7	R_LARCH_TLS_DTPMOD64	TLS-GD 动态重定位支持	<code>*(int64_t *) PC = ID of module defining sym</code>
8	R_LARCH_TLS_DTPREL32	TLS-GD 动态重定位支持	<code>*(int32_t *) PC = DTV-relative offset for sym</code>
9	R_LARCH_TLS_DTPREL64	TLS-GD 动态重定位支持	<code>*(int64_t *) PC = DTV-relative offset for sym</code>
10	R_LARCH_TLS_TPREL32	TLS-IE 动态重定位支持	<code>*(int32_t *) PC = T</code>
11	R_LARCH_TLS_TPREL64	TLS-IE 动态重定位支持	<code>*(int64_t *) PC = T</code>
12	R_LARCH_IRELATIVE	本地间接跳转解析	<code>*(void **) PC = (((void *) (*)()) (B + A)) ()</code>
... 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	<code>push (S - PC + A)</code>
23	R_LARCH_SOP_PUSH_ABSOLUTE	将常数或绝对地址压栈	<code>push (S + A)</code>
24	R_LARCH_SOP_PUSH_DUP	复制栈顶元素	<code>opr1 = pop (), push (opr1), push (opr1)</code>

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_1_0_12	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [21 ... 10] = opr1 [11 ... 0]</pre> <p>带 12 位有符号数溢出检测功能</p>
41	R_LARCH_SOP_POP_32_S_1_0_16	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [15 ... 0]</pre> <p>带 16 位有符号数溢出检测功能</p>
42	R_LARCH_SOP_POP_32_S_1_0_16_S2	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]</pre> <p>带 18 位有符号数溢出和4字节对齐检测功能</p>
43	R_LARCH_SOP_POP_32_S_5_20	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [24 ... 5] = opr1 [19 ... 0]</pre> <p>带 20 位有符号数溢出检测功能</p>
44	R_LARCH_SOP_POP_32_S_0_5_10_16_S2	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [4 ... 0] = opr1 [22 ... 18],</pre> <pre>(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]</pre> <p>带 23 位有符号数溢出和4字节对齐检测功能</p>
45	R_LARCH_SOP_POP_32_S_0_10_10_16_S2	指令立即数重定位	<pre>opr1 = pop (), (*(uint32_t *) PC) [9 ... 0] = opr1 [27 ... 18],</pre> <pre>(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]</pre> <p>带 28 位有符号数溢出和4字节对齐检测功能</p>
46	R_LARCH_SOP_POP_32_U	指令修正	<pre>(*(uint32_t *) PC) = pop ()</pre> <p>带 32 位无符号数溢出检测功能</p>
47	R_LARCH_ADD_8	8 位原地加法	<pre>*(int8_t *) PC += S + A</pre>
48	R_LARCH_ADD_16	16 位原地加法	<pre>*(int16_t *) PC += S + A</pre>

枚举值	名称	描述	语义
49	R_LARCH_ADD 24	24 位原地加法	<code>*(int24_t *) PC += S + A</code>
50	R_LARCH_ADD 32	32 位原地加法	<code>*(int32_t *) PC += S + A</code>
51	R_LARCH_ADD 64	64 位原地加法	<code>*(int64_t *) PC += S + A</code>
52	R_LARCH_SUB 8	8 位原地减法	<code>*(int8_t *) PC -= S + A</code>
53	R_LARCH_SUB 16	16 位原地减法	<code>*(int16_t *) PC -= S + A</code>
54	R_LARCH_SUB 24	24 位原地减法	<code>*(int24_t *) PC -= S + A</code>
55	R_LARCH_SUB 32	32 位原地减法	<code>*(int32_t *) PC -= S + A</code>
56	R_LARCH_SUB 64	64 位原地减法	<code>*(int64_t *) PC -= S + A</code>
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