LoongArch ELF ABI specification

Loongson Technology Corporation Limited

Version 1.00

Table of Contents

Register Convention

Table 1. Integer Register Convention

Name	Alias	Meaning	Preserved across calls
`$r0`	`$zero`	Constant zero	(Constant)
`$r1`	`$ra`	Return address	No
`$r2`	`$tp`	Thread pointer	(Non-allocatable)
`$r3`	`$sp`	Stack pointer	Yes
`$r4` - `$r11`	`$a0` - `$a7`	Argument registers	No
`$r4` - `$r5`	`$v0` - `$v1`	Return value	No
`$r12` - `$r20`	`$t0` - `$t8`	Temporary registers	No
`$r21`		Reserved	(Non-allocatable)
`$r22`	`$fp` / `$s9`	Frame pointer / Static register	Yes
`$r23` - `$r31`	`$s0` - `$s8`	Static registers	Yes

Table 2. Floating-point Register Convention

Name	Alias	Meaning	Preserved across calls
`$f0` - `$f7`	`$fa0` - `$fa7`	Argument registers	No
`$f0` - `$f1`	`$fv0` - `$fv1`	Return value	No
`$f8` - `$f23`	`$ft0` - `$ft15`	Temporary registers	No
`$f24` - `$f31`	`$fs0` - `$fs7`	Static registers	Yes

Temporary registers are also known as caller-saved registers. Static registers are also known as callee-saved registers.

Type Size and Alignment

Table 3. LP64 Data Model (base ABI types: `lp64d` `lp64f` `lp64s`)

Scalar type	Size (Bytes)	Alignment (Bytes)
`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
pointer types	8	8

Table 4. ILP32 Data Model (base ABI types: `ilp32d` `ilp32f` `ilp32s`)

Scalar type	Size (Bytes)	Alignment (Bytes)
`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
pointer types	4	4

For all base ABI types of LoongArch, the `char` datatype is signed by default.

ELF Object Files

All common ELF definitions referenced in this section can be found in the latest SysV gABI specification.

`EI_CLASS`: File class

EI_CLASS	Value	Description
`ELFCLASS32`	`1`	ELF32 object file
`ELFCLASS64`	`2`	ELF64 object file

`e_machine`: Identifies the machine

`LoongArch (258)`

`e_flags`: Identifies ABI type and version

Bit 31 - 8	Bit 7 - 6	Bit 5 - 3	Bit 2 - 0
(reserved)	ABI version	ABI extension	Base ABI

The ABI type of an ELF object is uniquely identified by `e_flags[7:0]` in its header.

Table 5. Base ABI Types

Name	Value of `e_flags[2:0]`	Description
	`0x0`	(reserved)
`lp64s`	`0x1`	Uses 64-bit GPRs and the stack for parameter passing. Data model is LP64, where `long` and pointers are 64-bit while `int` is 32-bit.
`lp64f`	`0x2`	Uses 64-bit GPRs, 32-bit FPRs and the stack for parameter passing. Data model is LP64, where `long` and pointers are 64-bit while `int` is 32-bit.
`lp64d`	`0x3`	Uses 64-bit GPRs, 64-bit FPRs and the stack for parameter passing. Data model is LP64, where `long` and pointers are 64-bit while `int` is 32-bit.
	`0x4`	(reserved)
`ilp32s`	`0x5`	Uses 32-bit GPRs and the stack for parameter passing. Data model is ILP32, where `int`, `long` and pointers are 32-bit.
`ilp32f`	`0x6`	Uses 32-bit GPRs, 32-bit FPRs and the stack for parameter passing. Data model is ILP32, where `int`, `long` and pointers are 32-bit.
`ilp32d`	`0x7`	Uses 32-bit GPRs, 64-bit FPRs and the stack for parameter passing. Data model is ILP32, where `int`, `long` and pointers are 32-bit.

Table 6. ABI Extension types

Name	Value of `e_flags[5:3]`	Description
`base`	`0x0`	No extra ABI features.
	`0x1` - `0x7`	(reserved)

`e_flags[7:6]` marks the ABI version of an ELF object.

Table 7. ABI Version

ABI version	Value	Description
`v0`	`0x0`	Stack operands base relocation type.
`v1`	`0x1`	Another relocation type IF needed.
	`0x2` `0x3`	Reserved.

Relocations

Table 8. ELF Relocation types

Enum	ELF reloc type	Usage	Detail
0	R_LARCH_NONE		
1	R_LARCH_32	Runtime address resolving	`*(int32_t *) PC = RtAddr + A`
2	R_LARCH_64	Runtime address resolving	`*(int64_t *) PC = RtAddr + A`
3	R_LARCH_RELATIVE	Runtime fixup for load-address	`*(void **) PC = B + A`
4	R_LARCH_COPY	Runtime memory copy in executable	`memcpy (PC, RtAddr, sizeof (sym))`
5	R_LARCH_JUMP_SLOT	Runtime PLT supporting	*implementation-defined*
6	R_LARCH_TLS_DTPMOD32	Runtime relocation for TLS-GD	`*(int32_t *) PC = ID of module defining sym`
7	R_LARCH_TLS_DTPMOD64	Runtime relocation for TLS-GD	`*(int64_t *) PC = ID of module defining sym`
8	R_LARCH_TLS_DTPREL32	Runtime relocation for TLS-GD	`*(int32_t *) PC = DTV-relative offset for sym`
9	R_LARCH_TLS_DTPREL64	Runtime relocation for TLS-GD	`*(int64_t *) PC = DTV-relative offset for sym`
10	R_LARCH_TLS_TPREL32	Runtime relocation for TLE-IE	`*(int32_t *) PC = T`
11	R_LARCH_TLS_TPREL64	Runtime relocation for TLE-IE	`*(int64_t *) PC = T`
12	R_LARCH_IRELATIVE	Runtime local indirect function resolving	`*(void **) PC = (((void *)(*)()) (B + A)) ()`
… Reserved for dynamic linker.			
20	R_LARCH_MARK_LA	Mark la.abs	Load absolute address for static link.
21	R_LARCH_MARK_PCREL	Mark external label branch	Access PC relative address for static link.
22	R_LARCH_SOP_PUSH_PCREL	Push PC-relative offset	`push (S - PC + A)`
23	R_LARCH_SOP_PUSH_ABSOLUTE	Push constant or absolute address	`push (S + A)`
24	R_LARCH_SOP_PUSH_DUP	Duplicate stack top	`opr1 = pop (), push (opr1), push (opr1)`

Enum	ELF reloc type	Usage	Detail
25	R_LARCH_SOP_PUSH_GPREL	Push for access GOT entry	`push (G)`
26	R_LARCH_SOP_PUSH_TLS_TPREL	Push for TLS-LE	`push (T)`
27	R_LARCH_SOP_PUSH_TLS_GOT	Push for TLS-IE	`push (IE)`
28	R_LARCH_SOP_PUSH_TLS_GD	Push for TLS-GD	`push (GD)`
29	R_LARCH_SOP_PUSH_PLT_PCREL	Push for external function calling	`push (PLT - PC)`
30	R_LARCH_SOP_ASSERT	Assert stack top	`assert (pop ())`
31	R_LARCH_SOP_NOT	Stack top operation	`push (!pop ())`
32	R_LARCH_SOP_SUB	Stack top operation	`opr2 = pop (), opr1 = pop (), push (opr1 - opr2)`
33	R_LARCH_SOP_SL	Stack top operation	`opr2 = pop (), opr1 = pop (), push (opr1 << opr2)`
34	R_LARCH_SOP_SR	Stack top operation	`opr2 = pop (), opr1 = pop (), push (opr1 >> opr2)`
35	R_LARCH_SOP_ADD	Stack top operation	`opr2 = pop (), opr1 = pop (), push (opr1 + opr2)`
36	R_LARCH_SOP_AND	Stack top operation	`opr2 = pop (), opr1 = pop (), push (opr1 & opr2)`
37	R_LARCH_SOP_IF_ELSE	Stack top operation	`opr3 = pop (), opr2 = pop (), opr1 = pop (), push (opr1 ? opr2 : opr3)`
38	R_LARCH_SOP_POP_32_S_10_5	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [14 ... 10] = opr1 [4 ... 0]` with check 5-bit signed overflow
39	R_LARCH_SOP_POP_32_U_10_12	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [21 ... 10] = opr1 [11 ... 0]` with check 12-bit unsigned overflow

Enum	ELF reloc type	Usage	Detail
40	R_LARCH_SOP_POP_32_S_10_12	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [21 ... 10] = opr1 [11 ... 0]` with check 12-bit signed overflow
41	R_LARCH_SOP_POP_32_S_10_16	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [15 ... 0]` with check 16-bit signed overflow
42	R_LARCH_SOP_POP_32_S_10_16_S2	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]` with check 18-bit signed overflow and 4-bit aligned
43	R_LARCH_SOP_POP_32_S_5_20	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [24 ... 5] = opr1 [19 ... 0]` with check 20-bit signed overflow

44	R_LARCH_SOP_POP_32_S_0_5_10_16_S2	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [4 ... 0] = opr1 [22 ... 18],`
			
			`(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]`
			
			with check 23-bit signed overflow and 4-bit aligned

45	R_LARCH_SOP_POP_32_S_0_10_10_16_S2	Instruction imm-field relocation	`opr1 = pop (), (*(uint32_t *) PC) [9 ... 0] = opr1 [27 ... 18],`
			
			`(*(uint32_t *) PC) [25 ... 10] = opr1 [17 ... 2]`
			
			with check 28-bit signed overflow and 4-bit aligned

46	R_LARCH_SOP_POP_32_U	Instruction fixup	`(*(uint32_t *) PC) = pop ()` with check 32-bit unsigned overflow
47	R_LARCH_ADD8	8-bit in-place addition	`*(int8_t *) PC += S + A`
48	R_LARCH_ADD16	16-bit in-place addition	`*(int16_t *) PC += S + A`

Enum	ELF reloc type	Usage	Detail
49	R_LARCH_ADD24	24-bit in-place addition	`*(int24_t *) PC += S + A`
50	R_LARCH_ADD32	32-bit in-place addition	`*(int32_t *) PC += S + A`
51	R_LARCH_ADD64	64-bit in-place addition	`*(int64_t *) PC += S + A`
52	R_LARCH_SUB8	8-bit in-place subtraction	`*(int8_t *) PC -= S + A`
53	R_LARCH_SUB16	16-bit in-place subtraction	`*(int16_t *) PC -= S + A`
54	R_LARCH_SUB24	24-bit in-place subtraction	`*(int24_t *) PC -= S + A`
55	R_LARCH_SUB32	32-bit in-place subtraction	`*(int32_t *) PC -= S + A`
56	R_LARCH_SUB64	64-bit in-place subtraction	`*(int64_t *) PC -= S + A`
57	R_LARCH_GNU_VTINHERIT	GNU C++ vtable hierarchy	
58	R_LARCH_GNU_VTENTRY	GNU C++ vtable member usage	

Program Interpreter Path

Table 9. Standard Program Interpreter Paths

Base ABI type	ABI extension type	Operating system / C library	Program interpreter path
`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`