Chapter 18

Calling Convention

This chapter describes the C compiler standards for RV32 and RV64 programs and two calling conventions: the convention for the base ISA plus standard general extensions (RV32G/RV64G), and the soft-float convention for implementations lacking floating-point units (e.g., RV32I/RV64I).

Implementations with ISA extensions might require extended calling conventions.

18.1 C Datatypes and Alignment

Table 18.1 summarizes the datatypes natively supported by RISC-V C programs. In both RV32 and RV64 C compilers, the C type `int` is 32 bits wide. `long`s and pointers, on the other hand, are both as wide as a integer register, so in RV32, both are 32 bits wide, while in RV64, both are 64 bits wide. Equivalently, RV32 employs an ILP32 integer model, while RV64 is LP64. In both RV32 and RV64, the C type `long long` is a 64-bit integer, `float` is a 32-bit IEEE 754-2008 floating-point number, `double` is a 64-bit IEEE 754-2008 floating-point number, and `long double` is a 128-bit IEEE floating-point number.

The C types `char` and `unsigned char` are 8-bit unsigned integers and are zero-extended when stored in a RISC-V integer register. `unsigned short` is a 16-bit unsigned integer and is zero-extended when stored in a RISC-V integer register. `signed char` is an 8-bit signed integer and is sign-extended when stored in a RISC-V integer register, i.e. bits (XLEN-1)..7 are all equal. `short` is a 16-bit signed integer and is sign-extended when stored in a register.

In RV64, 32-bit types, such as `int`, are stored in integer registers as proper sign extensions of their 32-bit values; that is, bits 63..31 are all equal. This restriction holds even for unsigned 32-bit types.

The RV32 and RV64 C compiler and compliant software keep all of the above datatypes naturally aligned when stored in memory.

第 18 章

调用约定

本章描述了 RV32 和 RV64 程序的 C 编译器标准，以及两种调用约定：适用于基础 ISA 加标准通用扩展（RV32G/RV64G）的约定，以及针对缺少浮点单元实现（如 RV32I/RV64I）的软浮点约定。

带有 ISA 扩展的实现可能需要扩展的调用约定。

18.1 C 数据类型与对齐

表 18.1 总结了 RISC-V C 程序原生支持的数据类型。在 RV32 和 RV64 的 C 编译器中，C 类型 int 均为 32 位宽。而 long 和指针的宽度与整数寄存器相同，因此在 RV32 中两者均为 32 位宽，在 RV64 中则为 64 位宽。等效地说，RV32 采用 ILP32 整数模型，RV64 采用 LP64 模型。在 RV32 和 RV64 中，C 类型 long long 均为 64 位整数，float 为 32 位 IEEE 754-2008 浮点数，double 为 64 位 IEEE 754-2008 浮点数，long double 为 128 位 IEEE 浮点数。

C 类型 char 和 unsigned char 为 8 位无符号整数，存入 RISC-V 整数寄存器时进行零扩展。unsigned short 为 16 位无符号整数，存入寄存器时同样进行零扩展。signed char 为 8 位有符号整数，存入寄存器时进行符号扩展（即 XLEN-1 至 7 的所有高位均与符号位相同）。short 为 16 位有符号整数，存入寄存器时进行符号扩展。

在 RV64 架构中，32 位类型（如 int）作为其 32 位值的正确符号扩展存储于整数寄存器中；即第 63 至 31 位全部相等。这一限制甚至适用于无符号 32 位类型。

RV32 和 RV64 的 C 编译器及合规软件在内存中存储时会保持上述所有数据类型自然对齐。

C type	Description	Bytes in RV32	Bytes in RV64
`char`	Character value/byte	1	1
`short`	Short integer	2	2
`int`	Integer	4	4
`long`	Long integer	4	8
`long long`	Long long integer	8	8
`void*`	Pointer	4	8
`float`	Single-precision float	4	4
`double`	Double-precision float	8	8
`long double`	Extended-precision float	16	16

Table 18.1: C compiler datatypes for base RISC-V ISA.

18.2 RVG Calling Convention

The RISC-V calling convention passes arguments in registers when possible. Up to eight integer registers, `a0`–`a7`, and up to eight floating-point registers, `fa0`–`fa7`, are used for this purpose.

If the arguments to a function are conceptualized as fields of a C `struct`, each with pointer alignment, the argument registers are a shadow of the first eight pointer-words of that `struct`. If argument $i < 8$ is a floating-point type, it is passed in floating-point register `fa`i; otherwise, it is passed in integer register `a`i. However, floating-point arguments that are part of `union`s or array fields of structures are passed in integer registers. Additionally, floating-point arguments to variadic functions (except those that are explicitly named in the parameter list) are passed in integer registers.

Arguments smaller than a pointer-word are passed in the least-significant bits of argument registers. Correspondingly, sub-pointer-word arguments passed on the stack appear in the lower addresses of a pointer-word, since RISC-V has a little-endian memory system.

When primitive arguments twice the size of a pointer-word are passed on the stack, they are naturally aligned. When they are passed in the integer registers, they reside in an aligned even-odd register pair, with the even register holding the least-significant bits. In RV32, for example, the function `void foo(int, long long)` is passed its first argument in `a0` and its second in `a2` and `a3`. Nothing is passed in `a1`.

Arguments more than twice the size of a pointer-word are passed by reference.

The portion of the conceptual `struct` that is not passed in argument registers is passed on the stack. The stack pointer `sp` points to the first argument not passed in a register.

Values are returned from functions in integer registers `a0` and `a1` and floating-point registers `fa0` and `fa1`. Floating-point values are returned in floating-point registers only if they are primitives or members of a `struct` consisting of only one or two floating-point values. Other return values that fit into two pointer-words are returned in `a0` and `a1`. Larger return values are passed entirely in memory; the caller allocates this memory region and passes a pointer to it as an implicit first parameter to the callee.

C 语言类型 描述		RV32 中的字节	RV64 中的字节
char	字符值/字节	1	1
短整型	短整型	2	2
int	整型	4	4
long	长整型	4	8
长长整型	长长整型	8	8
空指针类型	指针	4	8
浮点数	单精度浮点数	4	4
双精度	双精度浮点数	8	8
长双精度	扩展精度浮点数	16	16

表 18.1：基础 RISC-V ISA 的 C 编译器数据类型。

18.2 RVG 调用约定

RISC-V 调用约定尽可能通过寄存器传递参数，最多使用八个整数寄存器 a0–a7 和八个浮点寄存器 fa0–fa7 来实现这一目的。

若将函数的参数概念化为 C 语言结构体的字段，每个字段具有指针对齐，则参数寄存器相当于该结构体前八个指针字大小的字段的影子。若第 i（i < 8）个参数为浮点类型，则通过浮点寄存器 fai 传递；否则通过整数寄存器 ai 传递。但属于联合体或结构体数组字段的浮点参数需通过整数寄存器传递。此外，变参函数中的浮点参数（显式列在形参列表中的除外）也通过整数寄存器传递。

小于指针字大小的参数通过参数寄存器的最低有效位传递。相应地，在栈上传递的子指针字大小参数会出现在指针字的低地址处，因为 RISC-V 采用小端内存系统。

当原始参数大小是指针字长的两倍并通过栈传递时，它们会自然对齐。当它们通过整数寄存器传递时，这些参数会存放在一个对齐的偶奇寄存器对中，其中偶寄存器存放最低有效位。例如在 RV32 中，函数 void foo(int, long long)的第一个参数通过 a0 传递，第二个参数通过 a2 和 a3 传递，而 a1 不用于传递任何内容。

大小超过指针字长两倍的参数通过引用传递。

概念性结构中未通过参数寄存器传递的部分会通过栈传递。栈指针 sp 指向第一个未通过寄存器传递的参数。

函数返回值通过整数寄存器 a0 和 a1 以及浮点寄存器 fa0 和 fa1 传递。只有当浮点值是原始类型或仅包含一个或两个浮点值的结构体成员时，才会通过浮点寄存器返回。其他能容纳在两个指针字长内的返回值通过 a0 和 a1 返回。更大的返回值则完全通过内存传递；调用方会分配该内存区域，并将指向它的指针作为隐式第一个参数传递给被调用方。

In the standard RISC-V calling convention, the stack grows downward and the stack pointer is always kept 16-byte aligned.

In addition to the argument and return value registers, seven integer registers `t0`–`t6` and twelve floating-point registers `ft0`–`ft11` are temporary registers that are volatile across calls and must be saved by the caller if later used. Twelve integer registers `s0`–`s11` and twelve floating-point registers `fs0`–`fs11` are preserved across calls and must be saved by the callee if used. Table 18.2 indicates the role of each integer and floating-point register in the calling convention.

Register	ABI Name	Description	Saver
`x0`	`zero`	Hard-wired zero	—
`x1`	`ra`	Return address	Caller
`x2`	`sp`	Stack pointer	Callee
`x3`	`gp`	Global pointer	—
`x4`	`tp`	Thread pointer	—
`x5–7`	`t0–2`	Temporaries	Caller
`x8`	`s0/fp`	Saved register/frame pointer	Callee
`x9`	`s1`	Saved register	Callee
`x10–11`	`a0–1`	Function arguments/return values	Caller
`x12–17`	`a2–7`	Function arguments	Caller
`x18–27`	`s2–11`	Saved registers	Callee
`x28–31`	`t3–6`	Temporaries	Caller
`f0–7`	`ft0–7`	FP temporaries	Caller
`f8–9`	`fs0–1`	FP saved registers	Callee
`f10–11`	`fa0–1`	FP arguments/return values	Caller
`f12–17`	`fa2–7`	FP arguments	Caller
`f18–27`	`fs2–11`	FP saved registers	Callee
`f28–31`	`ft8–11`	FP temporaries	Caller

Table 18.2: RISC-V calling convention register usage.

18.3 Soft-Float Calling Convention

The soft-float calling convention is used on RV32 and RV64 implementations that lack floating-point hardware. It avoids all use of instructions in the F, D, and Q standard extensions, and hence the `f` registers.

Integral arguments are passed and returned in the same manner as the RVG convention, and the stack discipline is the same. Floating-point arguments are passed and returned in integer registers, using the rules for integer arguments of the same size. In RV32, for example, the function `double foo(int, double, long double)` is passed its first argument in `a0`, its second argument in `a2` and `a3`, and its third argument by reference via `a4`; its result is returned in `a0` and `a1`. In RV64, the arguments are passed in `a0`, `a1`, and the `a2-a3` pair, and the result is returned in `a0`.

The dynamic rounding mode and accrued exception flags are accessed through the routines provided

在标准的 RISC-V 调用约定中，栈向下增长，且栈指针始终保持 16 字节对齐。
除了参数和返回值寄存器外，七个整数寄存器 t0–t6 和十二个浮点寄存器 ft0–ft11 是临时寄存器，它们在调用过程中易失，若后续需使用则必须由调用者保存。十二个整数寄存器 s0–s11 和十二个浮点寄存器 fs0–fs11 在调用间保留，若被使用则必须由被调用者保存。表 18.2 展示了调用约定中每个整数和浮点寄存器的角色。

寄存器 ABI 名称 描述		保存者
x0 zero 硬件固定零值		—
x1 ra 返回地址		调用者
x2 sp 栈指针		被调用者
x3 gp 全局指针		—
x4 tp 线程指针		—
x5–7 t0–2 临时寄存器		调用者
x8 s0/fp 保存寄存器/帧指针	被调用者	
x9 s1 保存寄存器		被调用者
x10–11 a0–1 函数参数/返回值	调用者	
x12–17 a2–7 函数参数		调用者
x18–27 s2–11 保存寄存器		被调用者
x28–31 t3–6 临时寄存器		调用者
f0–7 ft0–7 浮点临时寄存器		调用者
f8–9 fs0–1 浮点保存寄存器		被调用者
f10–11 fa0–1 浮点参数/返回值	调用者	
f12–17 fa2–7 浮点参数		调用者
f18–27 fs2–11 浮点保存寄存器		被调用者
f28–31 寄存器 ft8–11 寄存器	浮点临时寄存器	调用者

表 18.2：RISC-V 调用约定寄存器使用情况

18.3 软浮点调用约定

软浮点调用约定适用于缺乏浮点硬件支持的 RV32 和 RV64 实现。它避免使用 F、D 和 Q 标准扩展中的所有指令，因此也不使用 f 寄存器。
整型参数的传递与返回方式遵循 RVG 调用约定，且栈规则相同。浮点参数通过整型寄存器传递和返回，采用与相同大小整型参数相同的规则。例如在 RV32 中，函数`double foo(int, double, long double)`的第一个参数通过 a0 传递，第二个参数通过 a2 和 a3 传递，第三个参数通过 a4 以引用方式传递；其结果通过 a0 和 a1 返回。在 RV64 中，参数分别通过 a0、a1 以及 a2-a3 寄存器对传递，结果通过 a0 返回。
动态舍入模式和累积异常标志通过提供的例程进行访问

by the C99 header `fenv.h`.

由 C99 头文件 fenv.h 定义。