Chapter 18

ABI（Application Binary Interface，应用二进制接口）规范

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).

RVG 通用RISC-V

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.

区别主要在于 long 和 pointer 的占用空间不一样

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.

把参数的参数看作结构体的成员变量,并按照指针的大小对齐

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 unions 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. 变参函数的浮点参数也被写入到整数寄存器中

Union 联合体，其所有成员公用同一段内存

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`. long long 8 Byte, pointer 4 Byte

Arguments more than twice the size of a pointer-word are passed by reference. 超过两倍指针字长的使用引用传递

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. 寄存器无法存放的参数使用栈传递

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. caller 分配一块内存，并将指向这块内存的指针隐式地传递给 callee

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

通常堆从低向高拓展，栈从高向低拓展,两者相互逼近可以提高内存的利用率;
栈的底部可以设置一个保护页,在栈溢出的时候触发一个内存保护异常

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.

临时寄存器：由调用方保存，如果需要在函数调用后继续使用。
保留寄存器：由被调用方保存，如果在函数内部使用了这些寄存器。

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.

软浮点调用约定,在没有浮点计算单元的平台上避免使用浮点指令,使用软件模拟浮点运算

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

动态舍入模式指舍入模式可以在程序运行时动态改变，而不是在编译时固定;
累积异常标志用于记录程序执行期间出现的浮点异常

by the C99 header `fenv.h`.