6 The Base Procedure Call Standard

The base standard defines a machine-level calling standard for the A64 instruction set. It assumes the availability of the vector registers for passing floating-point and SIMD arguments. Application code is expected to conform to one of three data models defined in this standard; ILP32, LP64 or LLP64.

6.1 Machine Registers

The Arm 64-bit architecture defines two mandatory register banks: a general-purpose register bank which can be used for scalar integer processing and pointer arithmetic; and a SIMD and Floating-Point register bank. In addition, the architecture defines an optional set of scalable vector registers that overlap the SIMD and Floating-Point register bank, accompanied by a set of scalable predicate registers.

6.1.1 General-purpose Registers

There are thirty-one, 64-bit, general-purpose (integer) registers visible to the A64 instruction set; these are labeled r0-r30. In a 64-bit context these registers are normally referred to using the names x0-x30; in a 32-bit context the registers are specified by using w0-w30. Additionally, a stack-pointer register, SP, can be used with a restricted number of instructions. Register names may appear in assembly language in either upper case or lower case. In this specification upper case is used when the register has a fixed role in this procedure call standard. Table 2, General-purpose registers and AAPCS64 usage summarizes the uses of the general-purpose registers in this standard. In addition to the general-purpose registers there is one status register (NZCV) that may be set and read by conforming code.

Table 2, General-purpose registers and AAPCS64 usage

Register	Special	Role in the procedure call standard
SP		The Stack Pointer.
r30	LR	The Link Register.
r29	FP	The Frame Pointer
r19…r28		Callee-saved registers
r18		The Platform Register, if needed; otherwise a temporary register. See notes.
r17	IP1	The second intra-procedure-call temporary register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
r16	IP0	The first intra-procedure-call scratch register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
r9…r15		Temporary registers
r8		Indirect result location register
r0…r7		Parameter/result registers

The first eight registers, r0-r7, are used to pass argument values into a subroutine and to return result values from a function. They may also be used to hold intermediate values within a routine (but, in general, only between subroutine calls).

Registers r16 (IP0) and r17 (IP1) may be used by a linker as a scratch register between a routine and any subroutine it calls (for details, see Use of IP0 and IP1 by the linker). They can also be used within a routine to hold intermediate values between subroutine calls.

The role of register r18 is platform specific. If a platform ABI has need of a dedicated general-purpose register to carry inter-procedural state (for example, the thread context) then it should use this register for that purpose. If the platform ABI has no such requirements, then it should use r18 as an additional temporary register. The platform ABI specification must document the usage for this register.

> ***Note***
>
> Software developers creating platform-independent code are advised to avoid using r18 if at all possible. Most compilers provide a mechanism to prevent specific registers from being used for general allocation; portable hand-coded assembler should avoid it entirely. It should not be assumed that treating the register as callee-saved will be sufficient to satisfy the requirements of the platform. Virtualization code must, of course, treat the register as they would any other resource provided to the virtual machine.

A subroutine invocation must preserve the contents of the registers r19-r29 and SP. All 64 bits of each value stored in r19-r29 must be preserved, even when using the ILP32 data model **(Beta)**.

In all variants of the procedure call standard, registers r16, r17, r29 and r30 have special roles. In these roles they are labeled IP0, IP1, FP and LR when being used for holding addresses (that is, the special name implies accessing the register as a 64-bit entity).

> ***Note***
>
> The special register names (IP0, IP1, FP and LR) should be used only in the context in which they are special. It is recommended that disassemblers always use the architectural names for the registers.

The NZCV register is a global condition flag register with the following properties:

- The N, Z, C and V flags are undefined on entry to and return from a public interface.

6.1.2 SIMD and Floating-Point registers

The Arm 64-bit architecture also has a further thirty-two registers, v0-v31, which can be used by SIMD and Floating-Point operations. The precise name of the register will change indicating the size of the access.

> ***Note***
>
> Unlike in AArch32, in AArch64 the 128-bit and 64-bit views of a SIMD and Floating-Point register do not overlap multiple registers in a narrower view, so q1, d1 and s1 all refer to the same entry in the register bank.

The first eight registers, v0-v7, are used to pass argument values into a subroutine and to return result values from a function. They may also be used to hold intermediate values within a routine (but, in general, only between subroutine calls).

Registers v8-v15 must be preserved by a callee across subroutine calls; the remaining registers (v0-v7, v16-v31) do not need to be preserved (or should be preserved by the caller). Additionally, only the bottom 64 bits of each value stored in v8-v15 need to be preserved [8]; it is the responsibility of the caller to preserve larger values.

The FPSR is a status register that holds the cumulative exception bits of the floating-point unit. It contains the fields IDC, IXC, UFC, OFC, DZC, IOC and QC. These fields are not preserved across a public interface and may have any value on entry to a subroutine.

The FPCR is used to control the behavior of the floating-point unit. It is a global register with the following properties.

- The exception-control bits (8-12), rounding mode bits (22-23), flush-to-zero bits (24), and the AH and FIZ bits (0-1) may be modified by calls to specific support functions that affect the global state of the application.
- The NEP bit (bit 2) must be zero on entry to and return from a public interface.