

**NAME**

**arch** - Architecture-specific details

**DESCRIPTION**

Differences between CPU architectures and platforms supported by FreeBSD.

**Introduction**

This document is a quick reference of key ABI details of FreeBSD architecture ports. For full details consult the processor-specific ABI supplement documentation.

If not explicitly mentioned, sizes are in bytes. The architecture details in this document apply to FreeBSD 12.0 and later, unless otherwise noted.

FreeBSD uses a flat address space. Variables of types *unsigned long*, *uintptr\_t*, and *size\_t* and pointers all have the same representation.

In order to maximize compatibility with future pointer integrity mechanisms, manipulations of pointers as integers should be performed via *uintptr\_t* or *intptr\_t* and no other types. In particular, *long* and *ptrdiff\_t* should be avoided.

On some architectures, e.g., powerpc and AIM variants of powerpc64, the kernel uses a separate address space. On other architectures, kernel and a user mode process share a single address space. The kernel is located at the highest addresses.

On each architecture, the main user mode thread's stack starts near the highest user address and grows down.

FreeBSD architecture support varies by release. This table shows currently supported CPU architectures along with the first FreeBSD release to support each architecture.

<b>Architecture</b>	<b>Initial Release</b>
aarch64	11.0
amd64	5.1
armv6	10.0
armv7	12.0
i386	1.0
powerpc	6.0
powerpcspe	12.0
powerpc64	9.0
powerpc64le	13.0

riscv64 12.0

Discontinued architectures are shown in the following table.

Architecture	Initial Release	Final Release
alpha	3.2	6.4
arm	6.0	12.x
armeb	8.0	11.4
ia64	5.0	10.4
mips	8.0	13.x
mipsel	9.0	13.x
mipselhf	12.0	13.x
mipshf	12.0	13.x
mipsn32	9.0	13.x
mips64	9.0	13.x
mips64el	9.0	13.x
mips64elhf	12.0	13.x
mips64hf	12.0	13.x
pc98	2.2	11.4
riscv64sf	12.0	13.x
sparc64	5.0	12.x

### Type sizes

All FreeBSD architectures use some variant of the ELF (see elf(5)) **Application Binary Interface** (ABI) for the machine processor. All supported ABIs can be divided into two groups:

ILP32     *int*, *long*, *void \** types machine representations all have 4-byte size.

LP64     *int* type machine representation uses 4 bytes, while *long* and *void \** are 8 bytes.

Some machines support more than one FreeBSD ABI. Typically these are 64-bit machines, where the "native" LP64 execution environment is accompanied by the "legacy" ILP32 environment, which was the historical 32-bit predecessor for 64-bit evolution. Examples are:

LP64	ILP32 counterpart
amd64	i386
powerpc64	powerpc
aarch64	armv6/armv7

aarch64 will support execution of armv6 or armv7 binaries if the CPU implements AArch32 execution

state, however older armv4 and armv5 binaries aren't supported.

On all supported architectures:

<b>Type</b>	<b>Size</b>
short	2
int	4
long	sizeof(void*)
long long	8
float	4
double	8

Integers are represented in two's complement. Alignment of integer and pointer types is natural, that is, the address of the variable must be congruent to zero modulo the type size. Most ILP32 ABIs, except arm, require only 4-byte alignment for 64-bit integers.

Machine-dependent type sizes:

<b>Architecture</b>	<b>void *</b>	<b>long double</b>	<b>time_t</b>
aarch64	8	16	8
amd64	8	16	8
armv6	4	8	8
armv7	4	8	8
i386	4	12	4
powerpc	4	8	8
powerpcspe	4	8	8
powerpc64	8	8	8
powerpc64le	8	8	8
riscv64	8	16	8

**time\_t** is 8 bytes on all supported architectures except i386.

### Endianness and Char Signedness

<b>Architecture</b>	<b>Endianness</b>	<b>char Signedness</b>
aarch64	little	unsigned
amd64	little	signed
armv6	little	unsigned
armv7	little	unsigned
i386	little	signed
powerpc	big	unsigned

powerpcspe	big	unsigned
powerpc64	big	unsigned
powerpc64le	little	unsigned
riscv64	little	signed

### Page Size

Architecture	Page Sizes
aarch64	4K, 2M, 1G
amd64	4K, 2M, 1G
armv6	4K, 1M
armv7	4K, 1M
i386	4K, 2M (PAE), 4M
powerpc	4K
powerpcspe	4K
powerpc64	4K
powerpc64le	4K
riscv64	4K, 2M, 1G

### Floating Point

Architecture	float, double	long double
aarch64	hard	soft, quad precision
amd64	hard	hard, 80 bit
armv6	hard	hard, double precision
armv7	hard	hard, double precision
i386	hard	hard, 80 bit
powerpc	hard	hard, double precision
powerpcspe	hard	hard, double precision
powerpc64	hard	hard, double precision
powerpc64le	hard	hard, double precision
riscv64	hard	hard, quad precision

### Default Tool Chain

FreeBSD uses clang(1) as the default compiler on all supported CPU architectures, LLVM's ld.lld(1) as the default linker, and ELF Tool Chain binary utilities such as objcopy(1) and readelf(1).

### MACHINE\_ARCH vs MACHINE\_CPUARCH vs MACHINE

MACHINE\_CPUARCH should be preferred in Makefiles when the generic architecture is being tested. MACHINE\_ARCH should be preferred when there is something specific to a particular type of architecture where there is a choice of many, or could be a choice of many. Use MACHINE when referring to the kernel, interfaces dependent on a specific type of kernel or similar things like boot

sequences.

MACHINE	MACHINE_CPUARCH	MACHINE_ARCH
arm64	aarch64	aarch64
amd64	amd64	amd64
arm	arm	armv6, armv7
i386	i386	i386
powerpc	powerpc	powerpc, powerpcspe, powerpc64, powerpc64le
riscv	riscv	riscv64

### Predefined Macros

The compiler provides a number of predefined macros. Some of these provide architecture-specific details and are explained below. Other macros, including those required by the language standard, are not included here.

The full set of predefined macros can be obtained with this command:

```
cc -x c -dM -E /dev/null
```

Common type size and endianness macros:

Macro	Meaning
<code>__LP64__</code>	64-bit (8-byte) long and pointer, 32-bit (4-byte) int
<code>__ILP32__</code>	32-bit (4-byte) int, long and pointer
<code>BYTE_ORDER</code>	Either <code>BIG_ENDIAN</code> or <code>LITTLE_ENDIAN</code> . <code>PDP11_ENDIAN</code> is not used on FreeBSD.

Architecture-specific macros:

Architecture	Predefined macros
aarch64	<code>__aarch64__</code>
amd64	<code>__amd64__</code> , <code>__x86_64__</code>
armv6	<code>__arm__</code> , <code>__ARM_ARCH &gt;= 6</code>
armv7	<code>__arm__</code> , <code>__ARM_ARCH &gt;= 7</code>
i386	<code>__i386__</code>
powerpc	<code>__powerpc__</code>
powerpcspe	<code>__powerpc__</code> , <code>__SPE__</code>
powerpc64	<code>__powerpc__</code> , <code>__powerpc64__</code>
powerpc64le	<code>__powerpc__</code> , <code>__powerpc64__</code>
riscv64	<code>__riscv</code> , <code>__riscv_xlen == 64</code>

Compilers may define additional variants of architecture-specific macros. The macros above are preferred for use in FreeBSD.

### Important make(1) variables

Most of the externally settable variables are defined in the build(7) man page. These variables are not otherwise documented and are used extensively in the build system.

**MACHINE** Represents the hardware platform. This is the same as the native platform's `uname(1) -m` output. It defines both the userland / kernel interface, as well as the bootloader / kernel interface. It should only be used in these contexts. Each CPU architecture may have multiple hardware platforms it supports where **MACHINE** differs among them. It is used to collect together all the files from `config(8)` to build the kernel. It is often the same as **MACHINE\_ARCH** just as one CPU architecture can be implemented by many different hardware platforms, one hardware platform may support multiple CPU architecture family members, though with different binaries. For example, **MACHINE** of i386 supported the IBM-AT hardware platform while the **MACHINE** of pc98 supported the Japanese company NEC's PC-9801 and PC-9821 hardware platforms. Both of these hardware platforms supported only the **MACHINE\_ARCH** of i386 where they shared a common ABI, except for certain kernel / userland interfaces relating to underlying hardware platform differences in bus architecture, device enumeration and boot interface. Generally, **MACHINE** should only be used in `src/sys` and `src/stand` or in system imagers or installers.

**MACHINE\_ARCH** Represents the CPU processor architecture. This is the same as the native platforms `uname(1) -p` output. It defines the CPU instruction family supported. It may also encode a variation in the byte ordering of multi-byte integers (endian). It may also encode a variation in the size of the integer or pointer. It may also encode a ISA revision. It may also encode hard versus soft floating point ABI and usage. It may also encode a variant ABI when the other factors do not uniquely define the ABI. It, along with **MACHINE**, defines the ABI used by the system. Generally, the plain CPU name specifies the most common (or at least first) variant of the CPU. This is why `powerpc` and `powerpc64` imply 'big endian' while 'armv6' and 'armv7' imply little endian. If we ever were to support the so-called x32 ABI (using 32-bit pointers on the amd64 architecture), it would most likely be encoded as `amd64-x32`. It is unfortunate that `amd64` specifies the 64-bit evolution of the x86 platform (it matches the 'first rule') as everybody else uses `x86_64`. There is no standard name for the processor: each OS selects its own

conventions.

**MACHINE\_CPUARCH** Represents the source location for a given **MACHINE\_ARCH**. It is generally the common prefix for all the **MACHINE\_ARCH** that share the same implementation, though 'riscv' breaks this rule. While amd64 and i386 are closely related, **MACHINE\_CPUARCH** is not x86 for them. The FreeBSD source base supports amd64 and i386 with two distinct source bases living in subdirectories named amd64 and i386 (though behind the scenes there's some sharing that fits into this framework).

**CPUTYPE** Sets the flavor of **MACHINE\_ARCH** to build. It is used to optimize the build for a specific CPU / core that the binaries run on. Generally, this does not change the ABI, though it can be a fine line between optimization for specific cases.

**TARGET** Used to set **MACHINE** in the top level Makefile for cross building. Unused outside of that scope. It is not passed down to the rest of the build. Makefiles outside of the top level should not use it at all (though some have their own private copy for hysterical reasons).

**TARGET\_ARCH** Used to set **MACHINE\_ARCH** by the top level Makefile for cross building. Like **TARGET**, it is unused outside of that scope.

## SEE ALSO

src.conf(5), build(7)

## HISTORY

An **arch** manual page appeared in FreeBSD 11.1.