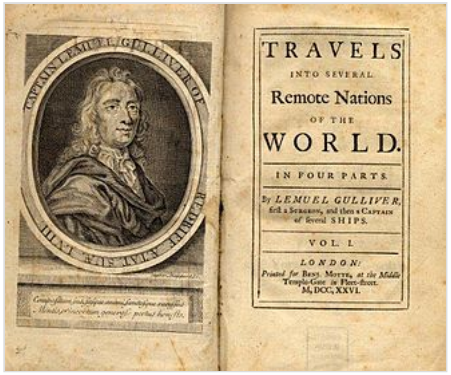
Endianness

In computing, **endianness** is the order in which bytes within a word of digital data are transmitted over a data communication medium or addressed (by rising addresses) in computer memory, counting only byte significance compared to earliness. Endianness is primarily expressed as **big-endian** (BE) or **little-endian** (LE), terms introduced by Danny Cohen into computer science for data ordering in an Internet Experiment Note published in 1980.[1] The adjective *endian* has its origin in the writings of 18th century Anglo-Irish writer Jonathan Swift. In the 1726 novel *Gulliver's Travels*, he portrays the conflict between sects of Lilliputians divided into those breaking the shell of a boiled egg from the big end or from the little end.[2][3] By analogy, a CPU may read a digital word big end first, or little end first.



Gulliver's Travels by Jonathan Swift, the novel from which the term was coined

Computers store information in various-sized groups of binary bits. Each group is assigned a number, called its *address*, that the computer uses to access that data. On most modern computers, the smallest data group with an address is eight bits long and is called a byte. Larger groups comprise two or more bytes, for example, a 32-bit word contains four bytes. There are two possible ways a computer could number the individual bytes in a larger group, starting at either end. Both types of endianness are in widespread use in digital electronic engineering. The initial choice of endianness of a new design is often arbitrary, but later technology revisions and updates perpetuate the existing endianness to maintain backward compatibility.

A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest. A little-endian system, in contrast, stores the least-significant byte at the smallest address.[4][5][6] Of the two, big-endian is thus closer to the way the digits of numbers are written left-to-right in English, comparing digits to bytes. *Bi-endianness* is a feature supported by numerous computer architectures that feature switchable endianness in data fetches and stores or for instruction fetches. Other orderings are generically called *middle-endian* or *mixed-endian*.[7][8][9][10]

Big-endianness is the dominant ordering in networking protocols, such as in the Internet protocol suite, where it is referred to as *network order*, transmitting the most significant byte first. Conversely, little-endianness is the dominant ordering for processor architectures (x86, most ARM implementations, base RISC-V implementations) and their associated memory. File formats can use either ordering; some formats use a mixture of both or contain an indicator of which ordering is used throughout the file.[11]

Characteristics

Computer memory consists of a sequence of storage cells (smallest addressable units); in machines that support byte addressing, those units are called *bytes*. Each byte is identified and accessed in hardware and software by its memory address. If the total number of bytes in memory is n, then addresses are enumerated from 0 to $n - 1$.

Computer programs often use data structures or fields that may consist of more data than can be stored in one byte. In the context of this article where its type cannot be arbitrarily complicated, a "field" consists of a consecutive sequence of bytes and represents a "simple data value" which – at least potentially – can be manipulated by *one* single hardware instruction. On most systems, the address of a multi-byte simple data value is the address of its first byte (the byte with the lowest address). There are exceptions to this rule – for example, the Add instruction of the IBM 1401 addresses variable-length fields at their low-order (highest-

addressed) position with their lengths being defined by a word mark set at their high-order (lowest-addressed) position. When an operation such as addition is performed, the processor begins at the low-order positions at the high addresses of the two fields and works its way down to the high-order.

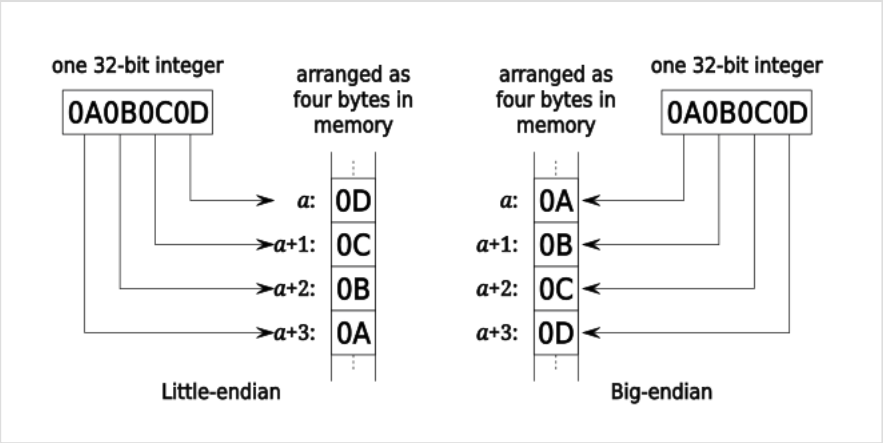


Diagram demonstrating big- versus little-endianness

Another important attribute of a byte being part of a "field" is its "significance". These attributes of the parts of a field play an important role in the sequence the bytes are accessed by the computer hardware, more precisely: by the low-level algorithms contributing to the results of a computer instruction.

Numbers

Positional number systems (mostly base 2, or less often base 10) are the predominant way of representing and particularly of manipulating integer data by computers. In pure form this is valid for moderate sized non-negative integers, e.g. of C data type `unsigned`. In such a number system, the *value* of a digit which it contributes to the whole number is determined not only by its value as a single digit, but also by the position it holds in the complete number, called its significance. These positions can be mapped to memory mainly in two ways:[12]

- Decreasing numeric significance with increasing memory addresses (or increasing time), known as *big-endian* and
- Increasing numeric significance with increasing memory addresses (or increasing time), known as *little-endian*.

In these expressions, the term "end" is meant as the extremity where the *big* resp. *little* significance is written *first*, namely where the field *starts*.

The integer data that are directly supported by the computer hardware have a fixed width of a low power of 2, e.g. 8 bits ≙ 1 byte, 16 bits ≙ 2 bytes, 32 bits ≙ 4 bytes, 64 bits ≙ 8 bytes, 128 bits ≙ 16 bytes. The low-level access sequence to the bytes of such a field depends on the operation to be performed. The least-significant byte is accessed first for addition, subtraction and multiplication. The most-significant byte is accessed first for division and comparison. See § Calculation order.

Text

When character (text) strings are to be compared with one another, e.g. in order to support some mechanism like sorting, this is very frequently done lexicographically where a single positional element (character) also has a positional value. Lexicographical comparison means almost everywhere: first character ranks highest – as in the telephone book. Almost all machines which can do this using a single instruction are big-endian or at least mixed-endian.

Integer numbers written as text are always represented most significant digit first in memory, which is similar to big-endian, independently of text direction.

Byte addressing

When memory bytes are printed sequentially from left to right (e.g. in a hex dump), little-endian representation of integers has the significance increasing from left to right. In other words, it appears backwards when visualized, which can be counter-intuitive.

This behavior arises, for example, in FourCC or similar techniques that involve packing characters into an integer, so that it becomes a sequence of specific characters in memory. For example, take the string "JOHN", stored in hexadecimal ASCII. On big-endian machines, the value appears left-to-right, coinciding with the correct string order for reading the result ("J O H N"). But on a little-endian machine, one would see "N H O J". Middle-endian machines complicate this even further; for example, on the PDP-11, the 32-bit value is stored as two 16-bit words "JO" "HN" in big-endian, with the characters in the 16-bit words being stored in little-endian, resulting in "O J N H".

Byte swapping

Byte-swapping consists of rearranging bytes to change endianness. Many compilers provide built-ins that are likely to be compiled into native processor instructions (`bswap`/`movbe`), such as `__builtin_bswap32`. Software interfaces for swapping include:

- Standard network endianness functions (from/to BE, up to 32-bit).[13] Windows has a 64-bit extension in `winsock2.h`.
- BSD and Glibc `endian.h` functions (from/to BE and LE, up to 64-bit).[14]
- macOS `OSByteOrder.h` macros (from/to BE and LE, up to 64-bit).
- The `std::byteswap` function in C++23.[15]

Some CPU instruction sets provide native support for endian byte swapping, such as `bswap`[16] (x86 — 486 and later, i960 — i960Jx and later[17]), and `rev`[18] (ARMv6 and later).

Some compilers have built-in facilities for byte swapping. For example, the Intel Fortran compiler supports the non-standard `CONVERT` specifier when opening a file, e.g.: `OPEN(unit, CONVERT='BIG_ENDIAN',...)`. Other compilers have options for generating code that globally enables the conversion for all file IO operations. This permits the reuse of code on a system with the opposite endianness without code modification.

Considerations

Simplified access to part of a field

On most systems, the address of a multi-byte value is the address of its first byte (the byte with the lowest address); little-endian systems of that type have the property that, for sufficiently low data values, the same value can be read from memory at different lengths without using different addresses (even when alignment restrictions are imposed). For example, a 32-bit memory location with content `4A 00 00 00` can be read at the same address as either 8-bit (value = 4A), 16-bit (004A), 24-bit (00004A), or 32-bit (0000004A), all of which retain the same numeric value. Although this little-endian property is rarely used directly by high-level programmers, it is occasionally employed by code optimizers as well as by assembly language programmers. While not allowed by C++, such type punning code is allowed as "implementation-defined" by the C11 standard[19] and commonly used[20] in code interacting with hardware.[21]

Calculation order

Some operations in positional number systems have a natural or preferred order in which the elementary steps are to be executed. This order may affect their performance on small-scale byte-addressable processors and microcontrollers. However, high-performance processors usually fetch multi-byte operands from

memory in the same amount of time they would have fetched a single byte, so the complexity of the hardware is not affected by the byte ordering.

Addition, subtraction, and multiplication start at the least significant digit position and propagate the carry to the subsequent more significant position. On most systems, the address of a multi-byte value is the address of its first byte (the byte with the lowest address). The implementation of these operations is marginally simpler using little-endian machines where this first byte contains the least significant digit.

Comparison and division start at the most significant digit and propagate a possible carry to the subsequent less significant digits. For fixed-length numerical values (typically of length 1,2,4,8,16), the implementation of these operations is marginally simpler on big-endian machines.

Some big-endian processors (e.g. the IBM System/360 and its successors) contain hardware instructions for lexicographically comparing varying length character strings.

The normal data transport by an assignment statement is in principle independent of the endianness of the processor.

Hardware

Many historical and extant processors use a big-endian memory representation, either exclusively or as a design option. The IBM System/360 uses big-endian byte order, as do its successors System/370, ESA/390, and z/Architecture. The PDP-10 uses big-endian addressing for byte-oriented instructions. The IBM Series/1 minicomputer uses big-endian byte order. The Motorola 6800 / 6801, the 6809 and the 68000 series of processors use the big-endian format. Solely big-endian architectures include the IBM z/Architecture and OpenRISC.

The Datapoint 2200 used simple bit-serial logic with little-endian to facilitate carry propagation. When Intel developed the 8008 microprocessor for Datapoint, they used little-endian for compatibility. However, as Intel was unable to deliver the 8008 in time, Datapoint used a medium-scale integration equivalent, but the little-endianness was retained in most Intel designs, including the MCS-48 and the 8086 and its x86 successors.[22][23] The DEC Alpha, Atmel AVR, VAX, the MOS Technology 6502 family (including Western Design Center 65802 and 65C816), the Zilog Z80 (including Z180 and eZ80), the Altera Nios II, and many other processors and processor families are also little-endian.

The Intel 8051, unlike other Intel processors, expects 16-bit addresses for LJMP and LCALL in big-endian format; however, xCALL instructions store the return address onto the stack in little-endian format.[24]

The IA-32 and x86-64 instruction set architectures use the little-endian format. Other instruction set architectures that follow this convention, allowing only little-endian mode, include Nios II, Andes Technology NDS32, and Qualcomm Hexagon.

Some instruction set architectures are "bi-endian" and allow running software of either endianness; these include Power ISA, SPARC, ARM AArch64, C-Sky, and RISC-V. IBM AIX and IBM i run in big-endian mode on bi-endian Power ISA; Linux originally ran in big-endian mode, but by 2019, IBM had transitioned to little-endian mode for Linux to ease the porting of Linux software from x86 to Power.[25][26] SPARC has no relevant little-endian deployment, as both Oracle Solaris and Linux run in big-endian mode on bi-endian SPARC systems, and can be considered big-endian in practice. ARM, C-Sky, and RISC-V have no relevant big-endian deployments, and can be considered little-endian in practice.

Bi-endianness

Some architectures (including ARM versions 3 and above, PowerPC, Alpha, SPARC V9, MIPS, Intel i860, PA-RISC, SuperH SH-4 and IA-64) feature a setting which allows for switchable endianness in data fetches and stores, instruction fetches, or both. This feature can improve performance or simplify the logic of networking devices and software. The word *bi-endian*, when said of hardware, denotes the capability of the machine to compute or pass data in either endian format.

Many of these architectures can be switched via software to default to a specific endian format (usually done when the computer starts up); however, on some systems, the default endianness is selected by hardware on the motherboard and cannot be changed via software (e.g. the Alpha, which runs only in big-endian mode on the Cray T3E).

The term *bi-endian* refers primarily to how a processor treats data accesses. Instruction accesses (fetches of instruction words) on a given processor may still assume a fixed endianness, even if data accesses are fully bi-endian, though this is not always the case, such as on Intel's IA-64-based Itanium CPU, which allows both.

Some nominally bi-endian CPUs require motherboard help to fully switch endianness. For instance, the 32-bit desktop-oriented PowerPC processors in little-endian mode act as little-endian from the point of view of the executing programs, but they require the motherboard to perform a 64-bit swap across all 8 byte lanes to ensure that the little-endian view of things will apply to I/O devices. In the absence of this unusual motherboard hardware, device driver software must write to different addresses to undo the incomplete transformation and also must perform a normal byte swap.

Some CPUs, such as many PowerPC processors intended for embedded use and almost all SPARC processors, allow per-page choice of endianness.

SPARC processors since the late 1990s (SPARC v9 compliant processors) allow data endianness to be chosen with each individual instruction that loads from or stores to memory.

The ARM architecture supports two big-endian modes, called *BE-8* and *BE-32*.[27] CPUs up to ARMv5 only support BE-32 or word-invariant mode. Here any naturally aligned 32-bit access works like in little-endian mode, but access to a byte or 16-bit word is redirected to the corresponding address and unaligned access is not allowed. ARMv6 introduces BE-8 or byte-invariant mode, where access to a single byte works as in little-endian mode, but accessing a 16-bit, 32-bit or (starting with ARMv8) 64-bit word results in a byte swap of the data. This simplifies unaligned memory access as well as memory-mapped access to registers other than 32-bit.

Many processors have instructions to convert a word in a register to the opposite endianness, that is, they swap the order of the bytes in a 16-, 32- or 64-bit word.

Recent Intel x86 and x86-64 architecture CPUs have a MOVBE instruction (Intel Core since generation 4, after Atom),[28] which fetches a big-endian format word from memory or writes a word into memory in big-endian format. These processors are otherwise thoroughly little-endian.

There are also devices which use different formats in different places. For instance, the BQ27421 Texas Instruments battery gauge uses the little-endian format for its registers and the big-endian format for its random-access memory.

SPARC historically used big-endian until version 9, which is bi-endian. Similarly early IBM POWER processors were big-endian, but the PowerPC and Power ISA descendants are now bi-endian. The ARM architecture was little-endian before version 3 when it became bi-endian.

Floating point

Although many processors use little-endian storage for all types of data (integer, floating point), there are a number of hardware architectures where floating-point numbers are represented in big-endian form while integers are represented in little-endian form.[29] There are ARM processors that have mixed-endian floating-point representation for double-precision numbers: each of the two 32-bit words is stored as little-endian, but the most significant word is stored first. VAX floating point stores little-endian 16-bit words in big-endian order. Because there have been many floating-point formats with no network standard representation for them, the XDR standard uses big-endian IEEE 754 as its representation. It may therefore appear strange that the widespread IEEE 754 floating-point standard does not specify endianness.[30] Theoretically, this means that even standard IEEE floating-point data written by one machine might not be readable by another. However, on modern standard computers (i.e., implementing IEEE 754), one may safely assume that the endianness is the same for floating-point numbers as for integers, making the conversion straightforward regardless of data type. Small embedded systems using special floating-point formats may be another matter, however.

Variable-length data

Most instructions considered so far contain the size (lengths) of their operands within the operation code. Frequently available operand lengths are 1, 2, 4, 8, or 16 bytes. But there are also architectures where the length of an operand may be held in a separate field of the instruction or with the operand itself, e.g. by means of a word mark. Such an approach allows operand lengths up to 256 bytes or larger. The data types of such operands are character strings or BCD. Machines able to manipulate such data with one instruction (e.g. compare, add) include the IBM 1401, 1410, 1620, System/360, System/370, ESA/390, and z/Architecture, all of them of type big-endian.

Middle-endian

Numerous other orderings, generically called *middle-endian* or *mixed-endian*, are possible.

The PDP-11 is in principle a 16-bit little-endian system. The instructions to convert between floating-point and integer values in the optional floating-point processor of the PDP-11/45, PDP-11/70, and in some later processors, stored 32-bit "double precision integer long" values with the 16-bit halves swapped from the expected little-endian order. The UNIX C compiler used the same format for 32-bit long integers. This ordering is known as *PDP-endian*.[31]

UNIX was one of the first systems to allow the same code to be compiled for platforms with different internal representations. One of the first programs converted was supposed to print out `Unix`, but on the Series/1 it printed `nUxi` instead.[32]

A way to interpret this endianness is that it stores a 32-bit integer as two little-endian 16-bit words, with a big-endian word ordering:

Storage of a 32-bit integer, `0x0A0B0C0D`, on a PDP-11

byte offset	8-bit value	16-bit little-endian value
0	$0B_h$	$0A0B_h$
1	$0A_h$	
2	$0D_h$	$0C0D_h$
3	$0C_h$	

Segment descriptors of IA-32 and compatible processors keep a 32-bit base address of the segment stored in little-endian order, but in four nonconsecutive bytes, at relative positions 2, 3, 4 and 7 of the descriptor start.[33]

Software

Logic design

Hardware description languages (HDLs) used to express digital logic often support arbitrary endianness, with arbitrary granularity. For example, in SystemVerilog, a word can be defined as little-endian or big-endian.

Files and filesystems

The recognition of endianness is important when reading a file or filesystem created on a computer with different endianness.

Fortran sequential unformatted files created with one endianness usually cannot be read on a system using the other endianness because Fortran usually implements a record (defined as the data written by a single Fortran statement) as data preceded and succeeded by count fields, which are integers equal to the number of bytes in the data. An attempt to read such a file using Fortran on a system of the other endianness results in a run-time error, because the count fields are incorrect.

Unicode text can optionally start with a byte order mark (BOM) to signal the endianness of the file or stream. Its code point is U+FEFF. In UTF-32 for example, a big-endian file should start with `00 00 FE FF`; a little-endian should start with `FF FE 00 00`.

Application binary data formats, such as MATLAB *.mat* files, or the *.bil* data format, used in topography, are usually endianness-independent. This is achieved by storing the data always in one fixed endianness or carrying with the data a switch to indicate the endianness. An example of the former is the binary XLS file format that is portable between Windows and Mac systems and always little-endian, requiring the Mac application to swap the bytes on load and save when running on a big-endian Motorola 68K or PowerPC processor.[34]

TIFF image files are an example of the second strategy, whose header instructs the application about the endianness of their internal binary integers. If a file starts with the signature `MM` it means that integers are represented as big-endian, while `II` means little-endian. Those signatures need a single 16-bit word each, and they are palindromes, so they are endianness independent. `I` stands for Intel and `M` stands for Motorola. Intel CPUs are little-endian, while Motorola 680x0 CPUs are big-endian. This explicit signature allows a TIFF reader program to swap bytes if necessary when a given file was generated by a TIFF writer program running on a computer with a different endianness.

As a consequence of its original implementation on the Intel 8080 platform, the operating system-independent File Allocation Table (FAT) file system is defined with little-endian byte ordering, even on platforms using another endianness natively, necessitating byte-swap operations for maintaining the FAT on these platforms.

ZFS, which combines a filesystem and a logical volume manager, is known to provide adaptive endianness and to work with both big-endian and little-endian systems.[35]

Networking

Many IETF RFCs use the term *network order*, meaning the order of transmission for bytes *over the wire* in network protocols. Among others, the historic RFC 1700 (https://datatracker.ietf.org/doc/html/rfc1700) defines the network order for protocols in the Internet protocol suite to be big-endian.[36]

However, not all protocols use big-endian byte order as the network order. The Server Message Block (SMB) protocol uses little-endian byte order. In CANopen, multi-byte parameters are always sent least significant byte first (little-endian). The same is true for Ethernet Powerlink.[37]

The Berkeley sockets API defines a set of functions to convert 16- and 32-bit integers to and from network byte order: the `htons` (host-to-network-short) and `htonl` (host-to-network-long) functions convert 16- and 32-bit values respectively from machine (*host*) to network order; the `ntohs` and `ntohl` functions convert from network to host order.[38][39] These functions may be a no-op on a big-endian system.

While the high-level network protocols usually consider the byte (mostly meant as *octet*) as their atomic unit, the lowest layers of a network stack may deal with ordering of bits within a byte.

See also

- Bit order – Convention to identify bit positions

References

1. Cohen, Danny (1980-04-01). *On Holy Wars and a Plea for Peace* (https://www.rfc-editor.org/ien/ien137.txt). IETF. IEN 137. Also published at Cohen, Danny (October 1981). "On Holy Wars and a Plea for Peace" (https://ieeexplore.ieee.org/document/1667115). *IEEE Computer*. **14** (10): 48–54. doi:10.1109/C-M.1981.220208 (https://doi.org/10.1109%2FC-M.1981.220208).
2. Swift, Jonathan (1726). "A Voyage to Lilliput, Chapter IV" (https://en.wikisource.org/wiki/The_Works_of_the_Rev._Jonathan_Swift/Volume_6/A_Voyage_to_Lilliput/Chapter_4). *Gulliver's Travels*. Archived (https://web.archive.org/web/20220920173204/https://en.wikisource.org/wiki/The_Works_of_the_Rev._Jonathan_Swift/Volume_6/A_Voyage_to_Lilliput/Chapter_4) from the original on 2022-09-20. Retrieved 2022-09-20.
3. Bryant, Randal E.; David, O'Hallaron (2016), *Computer Systems: A Programmer's Perspective* (3 ed.), Pearson Education, p. 79, ISBN 978-1-488-67207-1
4. "Understanding big and little endian byte order" (https://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/). Archived (https://web.archive.org/web/20190524124000/https://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/) from the original on 2019-05-24. Retrieved 2019-05-20.
5. "Byte Ordering PPC" (https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFMemoryMgmt/Concepts/ByteOrdering.html#//apple_ref/doc/uid/20001150-CJBEJBHH). Archived (https://web.archive.org/web/20190509013855/https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFMemoryMgmt/Concepts/ByteOrdering.html#//apple_ref/doc/uid/20001150-CJBEJBHH) from the original on 2019-05-09. Retrieved 2019-05-20.
6. "Writing endian-independent code in C" (https://developer.ibm.com/articles/au-endianc/). Archived (https://web.archive.org/web/20190610085733/https://developer.ibm.com/articles/au-endianc/) from the original on 2019-06-10. Retrieved 2019-05-20.
7. "Internet Hall of Fame Pioneer" (http://internethalloffame.org/inductees/danny-cohen). *Internet Hall of Fame*. The Internet Society. Archived (https://web.archive.org/web/20210721114730/https://www.internethalloffame.org/inductees/danny-cohen) from the original on 2021-07-21. Retrieved 2015-10-07.
8. Cary, David. "Endian FAQ" (https://web.archive.org/web/20171109112101/http://david.carybros.com/html/endian_faq.html). Archived from the original (http://david.carybros.com/html/endian_faq.html) on 2017-11-09. Retrieved 2010-10-11.
9. James, David V. (June 1990). "Multiplexed buses: the endian wars continue". *IEEE Micro*. **10** (3): 9–21. doi:10.1109/40.56322 (https://doi.org/10.1109%2F40.56322). ISSN 0272-1732 (https://www.worldcat.org/issn/0272-1732). S2CID 24291134 (https://api.semanticscholar.org/CorpusID:24291134).

10. Blanc, Bertrand; Maaraoui, Bob (December 2005). "Endianness or Where is Byte 0?" (http://3bc.bertrand-blanc.com/endianness05.pdf) (PDF). Archived (https://web.archive.org/web/20071203190220/http://3bc.bertrand-blanc.com/endianness05.pdf) (PDF) from the original on 2007-12-03. Retrieved 2008-12-21.
11. *A File Format for the Exchange of Images in the Internet* (https://datatracker.ietf.org/doc/html/rfc1314#page-7). April 1992. p. 7. doi:10.17487/RFC1314 (https://doi.org/10.17487%2FRFC1314). RFC 1314 (https://datatracker.ietf.org/doc/html/rfc1314). Retrieved 2021-08-16.
12. Tanenbaum, Andrew S.; Austin, Todd M. (4 August 2012). *Structured Computer Organization* (https://books.google.com/books?id=m0HHygAACAAJ). Prentice Hall PTR. ISBN 978-0-13-291652-3. Retrieved 18 May 2013.
13. `byteorder(3) (https://manned.org/byteorder.3)` – Linux Programmer's Manual – Library Functions
14. `endian(3) (https://manned.org/endian.3)` – Linux Programmer's Manual – Library Functions
15. "std::byteswap" (https://en.cppreference.com/w/cpp/numeric/byteswap). *en.cppreference.com*. Archived (https://web.archive.org/web/20231120095109/https://en.cppreference.com/w/cpp/numeric/byteswap) from the original on 20 November 2023. Retrieved 3 October 2023.
16. "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 (2A, 2B & 2C): Instruction Set Reference, A-Z" (http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf) (PDF). Intel. September 2016. p. 3–112. Archived (https://ghostarchive.org/archive/20221009/http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf) (PDF) from the original on 2022-10-09. Retrieved 2017-02-05.
17. "i960® VH Processor Developer's Manual" (https://datasheets.chipdb.org/Intel/80960/manuals/27317301.PDF) (PDF). Intel. October 1998. Archived (https://web.archive.org/web/20240402165236/https://datasheets.chipdb.org/Intel/80960/manuals/27317301.PDF) (PDF) from the original on 2024-04-02. Retrieved 2024-04-02.
18. "ARMv8-A Reference Manual" (http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487a.k_10775/index.html). ARM Holdings. Archived (https://web.archive.org/web/20190119214452/http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487a.k_10775/index.html) from the original on 2019-01-19. Retrieved 2017-02-05.
19. "C11 standard" (https://www.iso.org/standard/57853.html). ISO. Section 6.5.2.3 "Structure and Union members", §3 and footnote 95. Archived (https://web.archive.org/web/20200328063511/https://www.iso.org/standard/57853.html) from the original on 28 March 2020. Retrieved 15 August 2018.
20. "3.10 Options That Control Optimization: -fstrict-aliasing" (https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Type-punning). *GNU Compiler Collection (GCC)*. Free Software Foundation. Archived (https://web.archive.org/web/20230701053330/https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Type-punning) from the original on 1 July 2023. Retrieved 15 August 2018.
21. Torvalds, Linus (5 Jun 2018). "[GIT PULL] Device properties framework update for v4.18-rc1" (https://lkml.org/lkml/2018/6/5/769). *Linux Kernel* (Mailing list). Archived (https://web.archive.org/web/20180815090956/https://lkml.org/lkml/2018/6/5/769) from the original on 15 August 2018. Retrieved 15 August 2018.
22. House, David; Faggin, Federico; Feeney, Hal; Gelbach, Ed; Hoff, Ted; Mazor, Stan; Smith, Hank (2006-09-21). "Oral History Panel on the Development and Promotion of the Intel 8008 Microprocessor" (http://archive.computerhistory.org/resources/text/Oral_History/Intel_8008/Intel_8008_1.oral_history.2006.102657982.pdf#page=5) (PDF). Computer History Museum. p. b5. Archived (https://web.archive.org/web/20140629084907/http://archive.computerhistory.org/resources/text/Oral_History/Intel_8008/Intel_8008_1.oral_history.2006.102657982.pdf#page=5) (PDF) from the original on 2014-06-29. Retrieved 23 April 2014.
23. Lunde, Ken (13 January 2009). *CJKV Information Processing* (https://books.google.com/books?id=SA92uQqTB-AC&pg=PA29). O'Reilly Media, Inc. p. 29. ISBN 978-0-596-51447-1. Retrieved 21 May 2013.
24. "Cx51 User's Guide: E. Byte Ordering" (http://www.keil.com/support/man/docs/c51/c51_xe.htm). *keil.com*. Archived (https://web.archive.org/web/20150402094251/http://www.keil.com/support/man/docs/c51/c51_xe.htm) from the original on 2015-04-02. Retrieved 2015-03-28.
25. Jeff Scheel (2016-06-16). "Little endian and Linux on IBM Power Systems" (https://developer.ibm.com/articles/l-power-little-endian-faq-trs/). *IBM*. Archived (https://web.archive.org/web/20220327025540/https://developer.ibm.com/articles/l-power-little-endian-faq-trs/) from the original on 2022-03-27. Retrieved 2022-03-27.

26. Timothy Prickett Morgan (10 June 2019). "The Transition To RHEL 8 Begins On Power Systems" (https://www.itjungle.com/2019/06/10/the-transition-to-rhel-8-begins-on-power-systems/). *ITJungle*. Archived (https://web.archive.org/web/20220124063316/https://www.itjungle.com/2019/06/10/the-transition-to-rhel-8-begins-on-power-systems/) from the original on 24 January 2022. Retrieved 26 March 2022.
27. "Differences between BE-32 and BE-8 buses" (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0290g/ch06s05s01.html). Archived (https://web.archive.org/web/20190212070549/http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0290g/ch06s05s01.html) from the original on 2019-02-12. Retrieved 2019-02-10.
28. "How to detect New Instruction support in the 4th generation Intel® Core™ processor family" (https://software.intel.com/sites/default/files/article/405250/how-to-detect-new-instruction-support-in-the-4th-generation-intel-core-processor-family.pdf) (PDF). Archived (https://web.archive.org/web/20160320222513/https://software.intel.com/sites/default/files/article/405250/how-to-detect-new-instruction-support-in-the-4th-generation-intel-core-processor-family.pdf) (PDF) from the original on 20 March 2016. Retrieved 2 May 2017.
29. Savard, John J. G. (2018) [2005], "Floating-Point Formats" (http://www.quadibloc.com/comp/cp0201.htm), *quadibloc*, archived (https://web.archive.org/web/20180703001709/http://www.quadibloc.com/comp/cp0201.htm) from the original on 2018-07-03, retrieved 2018-07-16
30. "pack – convert a list into a binary representation" (http://www.perl.com/doc/manual/html/pod/perlfunc/pack.html). Archived (https://web.archive.org/web/20090218010333/http://perl.com/doc/manual/html/pod/perlfunc/pack.html) from the original on 2009-02-18. Retrieved 2009-02-04.
31. *PDP-11/45 Processor Handbook* (http://bitsavers.org/pdf/dec/pdp11/handbooks/PDP1145_Handbook_1973.pdf) (PDF). Digital Equipment Corporation. 1973. p. 165. Archived (https://ghostarchive.org/archive/20221009/http://bitsavers.org/pdf/dec/pdp11/handbooks/PDP1145_Handbook_1973.pdf) (PDF) from the original on 2022-10-09.
32. Jalics, Paul J.; Heines, Thomas S. (1 December 1983). "Transporting a portable operating system: UNIX to an IBM minicomputer" (https://doi.org/10.1145%2F358476.358504). *Communications of the ACM*. **26** (12): 1066–1072. doi:10.1145/358476.358504 (https://doi.org/10.1145%2F358476.358504). S2CID 15558835 (https://api.semanticscholar.org/CorpusID:15558835).
33. *AMD64 Architecture Programmer's Manual Volume 2: System Programming* (https://web.archive.org/web/20180218024045/http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/24593_APM_v21.pdf) (PDF) (Technical report). 2013. p. 80. Archived from the original (http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/24593_APM_v21.pdf) (PDF) on 2018-02-18.
34. "Microsoft Office Excel 97 - 2007 Binary File Format Specification (*.xls 97-2007 format)" (http://download.microsoft.com/download/0/B/E/0BE8BDD7-E5E8-422A-ABFD-4342ED7AD886/Excel97-2007BinaryFileFormat(xls)Specification.xps). Microsoft Corporation. 2007. Archived (https://web.archive.org/web/20081222093136/http://download.microsoft.com/download/0/B/E/0BE8BDD7-E5E8-422A-ABFD-4342ED7AD886/Excel97-2007BinaryFileFormat(xls)Specification.xps) from the original on 2008-12-22. Retrieved 2014-08-18.
35. Matt Ahrens (2016). *FreeBSD Kernel Internals: An Intensive Code Walkthrough* (http://open-zfs.org/wiki/Documentation/Read_Write_Lecture). OpenZFS Documentation/Read Write Lecture. Archived (https://web.archive.org/web/20160414051047/http://open-zfs.org/wiki/Documentation/Read_Write_Lecture) from the original on 2016-04-14. Retrieved 2016-03-30.
36. Reynolds, J.; Postel, J. (October 1994). "Data Notations" (https://datatracker.ietf.org/doc/html/rfc1700#page-3). *Assigned Numbers* (https://datatracker.ietf.org/doc/html/rfc1700). IETF. p. 3. doi:10.17487/RFC1700 (https://doi.org/10.17487%2FRFC1700). STD 2. RFC 1700 (https://datatracker.ietf.org/doc/html/rfc1700). Retrieved 2012-03-02.
37. Ethernet POWERLINK Standardisation Group (2012), *EPSG Working Draft Proposal 301: Ethernet POWERLINK Communication Profile Specification Version 1.1.4*, chapter 6.1.1.
38. IEEE and The Open Group (2018). "3. System Interfaces". *The Open Group Base Specifications Issue 7* (https://pubs.opengroup.org/onlinepubs/9699919799/functions/htonl.html). Vol. 2. p. 1120. Archived (https://web.archive.org/web/20210418041546/https://pubs.opengroup.org/onlinepubs/9699919799/functions/htonl.html) from the original on 2021-04-18. Retrieved 2021-04-09.
39. "htonl(3) - Linux man page" (https://linux.die.net/man/3/htonl). *linux.die.net*. Archived (https://web.archive.org/web/20210418054331/https://linux.die.net/man/3/htonl) from the original on 2021-04-18. Retrieved 2021-04-09.

External links

- The dictionary definition of *endianness* at Wiktionary
