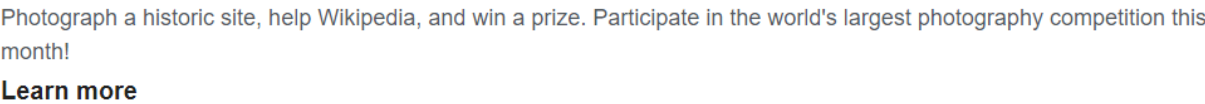


Document title:	Memory address - Wikipedia
Capture URL:	https://en.wikipedia.org/wiki/Memory_address
Page loaded at (UTC):	Sat, 21 Oct 2023 16:55:58 GMT
Capture timestamp (UTC):	Sat, 21 Oct 2023 16:56:27 GMT
Capture tool:	10.33.1
Collection server IP:	44.219.7.193
Browser engine:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.215 Safari/537.36
Operating system:	Windows_NT (Node 16.17.1)
PDF length:	4
Capture ID:	cimnK8Au2XUebTcXDnAjBD
User:	hh-bbrownlee



文 35 languages ▾

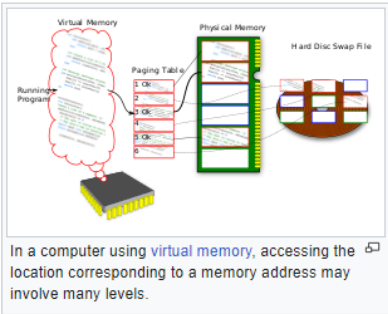
Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help [improve this article](#) by adding citations to reliable sources. Unsourced material may be challenged and removed.
 Find sources: "Memory address" – news · newspapers · books · scholar · JSTOR (February 2018) ([Learn how and when to remove this template message](#))

In **computing**, a **memory address** is a reference to a specific **memory** location used at various levels by **software** and **hardware**.^[1] Memory addresses are fixed-length sequences of **digits** conventionally displayed and manipulated as **unsigned integers**. Such numerical semantic bases itself upon features of CPU (such as the **instruction pointer** and incremental **address registers**), as well upon use of the memory like an **array** endorsed by various **programming languages**.

Types [[edit](#)]Physical addresses [\[edit \]](#)

A digital computer's main memory consists of many **memory locations**. Each memory location has a **physical address** which is a code. The CPU (or other device) can use the code to access the corresponding memory location. Generally only **system software**, i.e. the **BIOS**, operating systems, and some specialized **utility programs** (e.g., **memory testers**), address physical memory using machine code **operands** or **processor registers**, instructing the CPU to direct a hardware device, called the **memory controller**, to use the **memory bus** or **system bus**, or separate **control**, **address** and **data busses**, to execute the program's commands. The memory controllers' **bus** consists of a number of **parallel** lines, each represented by a **binary digit** (bit). The width of the bus, and thus the number of addressable storage units, and the number of bits in each unit, varies among computers.

Logical addresses [\[edit \]](#)

A [computer program](#) uses memory addresses to execute [machine code](#), and to store and retrieve [data](#). In early computers logical and physical addresses corresponded, but since the introduction of [virtual memory](#) most [application programs](#) do not have a knowledge of physical addresses. Rather, they address [logical addresses](#), or [virtual addresses](#), using the computer's [memory management unit](#) and [operating system](#) memory mapping; see [below](#).

Unit of address resolution [\[edit \]](#)

See also: *Word (computer architecture)* and *Binary prefix § Main memory*

Most modern computers are *byte-addressable*. Each address identifies a single *byte* (eight bits) of storage. Data larger than a single byte may be stored in a sequence of consecutive addresses. There exist *word-addressable* computers, where the minimal addressable storage unit is exactly the processor's *word*. For example, the *Data General Nova minicomputer*, and the *Texas Instruments TMS9900* and *National Semiconductor IMP-16 microcomputers* used 16 bit *words*, and there were many *36-bit mainframe computers* (e.g., *PDP-10*) which used *18-bit word addressing*, not *byte addressing*, giving an address space of 2^{18} 36-bit words, approximately 1 megabyte of storage. The efficiency of addressing of memory depends on the bit size of the bus used for addresses – the more bits used, the more addresses are available to the computer. For example, an 8-bit-byte-addressable machine with a 20-bit *address bus* (e.g. *Intel 8086*) can address 2^{20} (1,048,576) memory locations, or one *MiB* of memory, while a 32-bit bus (e.g. *Intel 80386*) addresses 2^{32} (4,294,967,296) locations, or a 4 *GiB* address space. In contrast, a 36-bit word-addressable machine with an 18-bit address bus addresses only 2^{18} (262,144) 36-bit locations (9,437,184 bits), equivalent to 1,179,648 8-bit bytes, or 1152 *KiB*, or 1.125 *MiB* — slightly more than the 8086.

Some older computers (*decimal computers*), were *decimal digit-addressable*. For example, each address in the *IBM 1620's magnetic-core memory* identified a single six bit *binary-coded decimal* digit, consisting of a *parity bit*, *flag bit* and four numerical bits. The 1620 used 5-digit decimal addresses, so in theory the highest possible address was 99,999. In practice, the CPU supported 20,000 memory locations, and up to two optional external memory units could be added, each supporting 20,000 addresses, for a total of 60,000 (00000–59999).

Word size versus address size [\[edit \]](#)

Word size is a characteristic of [computer architecture](#) denoting the number of bits that a CPU can process at one time. Modern processors, including [embedded systems](#), usually have a word size of 8, 16, 24, 32 or 64 bits; most current general-purpose computers use 32 or 64 bits. Many different sizes have been used historically, including 8, 9, 10, 12, 18, 24, 36, 39, 40, 48 and 60 bits.

Contents [hide]

(Top)

✓ Types

Physical addresses

Logical addresses

✓ Unit of address resolution

Word size versus address size

✓ Contents of each memory location

Address space in application programming

Addressing schemes

✓ Memory models

Memory models in x86 architecture

See also

References

memory identified a single six bit **binary-coded decimal** digit, consisting of a **parity bit**, **flag bit** and four numerical bits. The 1620 used 5-digit decimal addresses, so in theory the highest possible address was 99,999. In practice, the CPU supported 20,000 memory locations, and up to two optional external memory units could be added, each supporting 20,000 addresses, for a total of 60,000 (00000–59999).

Word size versus address size [edit]

Word size is a characteristic of **computer architecture** denoting the number of bits that a CPU can process at one time. Modern processors, including **embedded systems**, usually have a word size of 8, 16, 24, 32 or 64 bits; most current general-purpose computers use 32 or 64 bits. Many different sizes have been used historically, including 8, 9, 10, 12, 18, 24, 36, 39, 40, 48 and 60 bits.

Very often, when referring to the *word size* of a modern computer, one is also describing the size of address space on that computer. For instance, a computer said to be "**32-bit**" also usually allows 32-bit memory addresses; a byte-addressable 32-bit computer can address 2³² = 4,294,967,296 bytes of memory, or 4 gibibytes (GiB). This allows one memory address to be efficiently stored in one word.

However, this does not always hold true. Computers can have memory addresses larger or smaller than their word size. For instance, many **8-bit** processors, such as the **MOS Technology 6502**, **supported 16-bit addresses**— if not, they would have been limited to a mere 256 **bytes** of memory addressing. The 16-bit **Intel 8088** and **Intel 8086** supported 20-bit addressing via **segmentation**, allowing them to access 1 MiB rather than 64 KiB of memory. All Intel **Pentium** processors since the **Pentium Pro** include **Physical Address Extensions** (PAE) which support mapping 36-bit physical addresses to 32-bit virtual addresses. Many early processors held **2 addresses per word** , such as **36-bit** processors.

In theory, modern byte-addressable **64-bit** computers can address 2⁶⁴ bytes (16 **exbibytes**), but in practice the amount of memory is limited by the CPU, the **memory controller**, or the **printed circuit board** design (e.g., number of physical memory connectors or amount of soldered-on memory).

Contents of each memory location [edit]

*See also: **binary data***

Each memory location in a **stored-program computer** holds a **binary number** or **decimal number** *of some sort*. Its interpretation, as data of some **data type** or as an instruction, and use are determined by the **instructions** which retrieve and manipulate it.

Some early programmers combined instructions and data in words as a way to save memory, when it was expensive: The **Manchester Mark 1** had space in its 40-bit words to store little bits of data – its processor ignored a small section in the middle of a word – and that was often exploited as extra data storage.^[*citation needed*] **Self-replicating** programs such as **viruses** treat themselves sometimes as data and sometimes as instructions. **Self-modifying code** is generally **deprecated** nowadays, as it makes testing and maintenance disproportionately difficult to the saving of a few bytes, and can also give incorrect results because of the compiler or processor's assumptions about the machine's **state**, but is still sometimes used deliberately, with great care.

Address space in application programming [edit]

In modern **multitasking** environment, an **application process** usually has in its address space (or spaces) chunks of memory of following types:

- **Machine code**, including:
 - program's own code (historically known as ***code segment*** or ***text segment***);
 - **shared libraries**.
- **Data**, including:
 - initialized data (**data segment**);
 - uninitialized (but allocated) variables;
 - run-time stack;
 - **heap**;
 - **shared memory** and **memory mapped files**.

Some parts of address space may be not mapped at all.

Some systems have a "split" **memory architecture** where machine code, constants, and data are in different locations, and may have different address sizes. For example, **PIC18** microcontrollers have a 21-bit program counter to address machine code and constants in Flash memory, and 12-bit address registers to address data in SRAM.

Addressing schemes [edit]

*Main article: **Addressing mode***

A computer program can access an address given explicitly – in low-level programming this is usually called an **absolute address**, or sometimes a specific address, and is known as **pointer** data type in higher-level languages. But a program can also use **relative address** which specifies a location in relation to somewhere else (the ***base address***). There are many more indirect **addressing modes**.

Mapping logical addresses to physical and virtual memory also adds several levels of indirection; see below.

Memory models [edit]

Many programmers prefer to address memory such that there is no distinction between code space and data space (see **above**), as well as from physical and virtual memory (see **above**) — in other words, numerically identical pointers refer to exactly the same byte of RAM.

However, many early computers did not support such a ***flat memory model*** — in particular, **Harvard architecture** machines force program storage to be completely separate from data storage. Many modern **DSPs** (such as the **Motorola 56000**) have three separate storage areas — program storage, coefficient storage, and data storage. Some commonly used instructions fetch from all three areas simultaneously — fewer storage areas (even if there were the same total bytes of storage) would make those instructions run slower.

Contents [hide]

(Top)

Types

- Physical addresses
- Logical addresses

Unit of address resolution

- Word size versus address size

Contents of each memory location

- Address space in application programming
- Addressing schemes

Memory models

- Memory models in x86 architecture

See also

References

specifies a location in relation to somewhere else (the *base address*). There are many more indirect **addressing modes**.

Mapping logical addresses to physical and virtual memory also adds several levels of indirection; see below.

Memory models [edit]

Many programmers prefer to address memory such that there is no distinction between code space and data space (see *above*), as well as from physical and virtual memory (see *above*) — in other words, numerically identical pointers refer to exactly the same byte of RAM.

However, many early computers did not support such a *flat memory model* — in particular, **Harvard architecture** machines force program storage to be completely separate from data storage. Many modern **DSPs** (such as the **Motorola 56000**) have three separate storage areas — program storage, coefficient storage, and data storage. Some commonly used instructions fetch from all three areas simultaneously — fewer storage areas (even if there were the same total bytes of storage) would make those instructions run slower.

Memory models in x86 architecture [edit]

Main article: x86 memory models

Early x86 processors use the **segmented memory model** addresses based on a combination of two numbers: a **memory segment**, and an **offset** within that segment.

Some segments are implicitly treated as *code segments*, dedicated for *instructions*, *stack segments*, or normal *data segments*. Although the usages are different, the segments do not have different **memory protections** reflecting this. In the **flat memory model** all segments (segment registers) are generally set to zero, and only offsets are variable.

See also: Long mode

See also [edit]

- Base address
- Endianness
- Low-level programming language
- Memory address register
- Memory allocation
- Memory management unit (MMU)
- Memory model (programming)
- Memory protection
- Memory segmentation
- Offset (computer science), also known as a *displacement*
- Page table

References [edit]

- ↑ Abrahamson, Karl R. (Aug 20, 2022). "5.10.1. The Memory and Memory Addresses" ↗. *East Carolina University*. Retrieved Feb 3, 2023.

V · T · E	Data types	[hide]
Uninterpreted	Bit · Byte · Trit · Tryte · Word · Bit array	
Numeric	Arbitrary-precision or bignum · Complex · Decimal · Fixed point · Floating point (Reduced precision (Minifloat · Half precision · bfloat16) · Single precision · Double precision · Quadruple precision · Octuple precision · Extended precision (Long double)) · Integer (signedness) · Interval · Rational	
Pointer	Address (physical · virtual) · Reference	
Text	Character · String (null-terminated)	
Composite	Algebraic data type (generalized) · Array · Associative array · Class · Dependent · Equality · Inductive · Intersection · List · Object (metaobject) · Option type · Product · Record or Struct · Refinement · Set · Union (tagged)	
Other	Boolean · Bottom type · Collection · Enumerated type · Exception · Function type · Opaque data type · Recursive data type · Semaphore · Stream · Strongly typed identifier · Top type · Type class · Empty type · Unit type · Void	
Related topics	Abstract data type · Boxing · Data structure · Generic · Kind (metaclass) · Parametric polymorphism · Primitive data type · Interface · Subtyping · Type constructor · Type conversion · Type system · Type theory · Variable	

Category: Computer memory

This page was last edited on 27 September 2023, at 18:18 (UTC).

Text is available under the **Creative Commons Attribution-ShareAlike License 4.0**; additional terms may apply. By using this site, you agree to the **Terms of Use** and **Privacy Policy**. Wikipedia® is a registered trademark of the **Wikimedia Foundation, Inc.**, a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Code of Conduct Developers Statistics Cookie statement Mobile view

