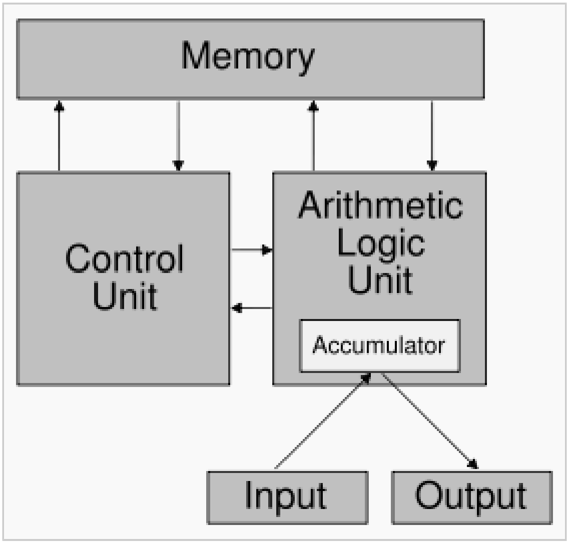
# Von Neumann architecture

From Wikipedia, the free encyclopedia

The **von Neumann architecture** is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data. It is named after mathematician and early computer scientist John von Neumann. Such a computer implements a universal Turing machine, and the common "referential model" of specifying sequential architectures, in contrast with parallel architectures. The term "stored-program computer" is generally used to mean a computer of this design, although as modern computers are usually of this type, the term has fallen into disuse.



Design of the von Neumann architecture

## Contents

## History

The earliest computing machines had fixed programs. Some very simple computers still use this design, either for simplicity or training purposes. For example, a desk calculator (in principle) is a fixed program computer. It can do basic mathematics, but it cannot be used as a word processor or to run video games. To change the program of such a machine, you have to re-wire, re-structure, or even re-design the machine. Indeed, the earliest computers were not so much "programmed" as they were "designed". "Reprogramming", when it was possible at all, was a laborious process, starting with flow charts and paper notes, followed by detailed engineering designs, and then the often-arduous process of physically re-wiring and re-building the machine.

The idea of the stored-program computer changed all that. By creating an instruction set architecture and detailing the computation as a series of instructions (the program), the machine becomes much more flexible. By treating those instructions in the same way as data, a stored-program machine can easily change the program, and can do so under program control.

The terms "von Neumann architecture" and "stored-program computer" are generally used interchangeably, and that usage is followed in this article. However, the Harvard architecture concept should also be mentioned, as a design which stores the program in a modifiable form, but without using the same physical storage or format as for general data.

A stored-program design also lets programs modify themselves while running, effectively allowing the computer to program itself. One early motivation for such a facility was the need for a program to increment or otherwise modify the address portion of instructions, which had to be done manually in early designs. This became less important when index registers and indirect addressing became customary features of machine architecture. Self-modifying code has largely fallen out of favor, since it is very hard to understand and debug, as well as inefficient under modern processor pipelining and caching schemes.

On a large scale, the ability to treat instructions as data is what makes assemblers, compilers and other automated programming tools possible. One can "write programs which write programs".[1] On a smaller scale, I/O-intensive machine instructions such as the BITBLT primitive used to modify images on a bitmap display, were once thought to be impossible to implement without custom hardware. It was shown later that these instructions could be implemented efficiently by "on the fly compilation" technology, e.g. code-generating programs — one form of self-modifying code that has remained popular.

There are drawbacks to the von Neumann design. Aside from the von Neumann bottleneck described below, program modifications can be quite harmful, either by accident or design. In some simple stored-program computer designs, a malfunctioning program can damage itself, other programs, or the operating system, possibly leading to a crash. This ability for programs to create and modify other programs is also frequently exploited by malware. A buffer overflow is one very common example of such a malfunction. Malware might use a buffer overflow to smash the call stack, overwrite the existing program, and then proceed to modify other program files on the system to further propagate the malware to other machines. Memory protection and other forms of access control can help protect against both accidental and malicious program modification.

## First designs

The term "von Neumann architecture" arose from mathematician John von Neumann's paper, *First Draft of a Report on the EDVAC*.[2] Dated June 30, 1945, it was an early written account of a general purpose stored-program computing machine (the EDVAC). However, while von Neumann's work was pioneering, the term *von Neumann architecture* does somewhat of an injustice to von Neumann's collaborators, contemporaries, and predecessors.

A patent application of Konrad Zuse mentioned this concept in 1936.

The idea of a stored-program computer existed at the Moore School of Electrical Engineering at the University of Pennsylvania before von Neumann even knew of the ENIAC's existence. The exact person who originated the idea there is unknown.

Herman Lukoff credits Eckert (see References).

John William Mauchly and J. Presper Eckert wrote about the stored-program concept in December 1943 during their work on ENIAC. Additionally, ENIAC project administrator Grist Brainerd's December 1943 progress report for the first period of the ENIAC's development implicitly proposed the stored program concept (while simultaneously rejecting its implementation in the ENIAC) by stating that "in order to have the simplest project and not to complicate matters" the ENIAC would be constructed without any "automatic regulation."

When the ENIAC was being designed, it was clear that reading instructions from punched cards or paper tape would not be fast enough, since the ENIAC was designed to execute instructions at a much higher rate. The ENIAC's program was thus wired into the design, and it had to be rewired for each new problem. It was clear that a better system was needed. The initial report on the proposed EDVAC was written during the time the ENIAC was being built, and contained the idea of the stored program, where instructions were stored in high-speed memory, so they could be quickly accessed for execution.

Alan Turing presented a paper on February 19, 1946, which included a complete design for a stored-program computer, the Pilot ACE.

## Von Neumann bottleneck

The separation between the CPU and memory leads to the *von Neumann bottleneck*, the limited throughput (data transfer rate) between the CPU and memory compared to the amount of memory. In modern machines, throughput is much smaller than the rate at which the CPU can work. This seriously limits the effective processing speed when the CPU is required to perform minimal processing on large amounts of data. The CPU is continuously forced to wait for vital data to be transferred to or from memory. As CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem.

The term "von Neumann bottleneck" was coined by John Backus in his 1977 ACM Turing award lecture. According to Backus:

> "Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it."

The performance problem is reduced by a cache between CPU and main memory, and by the development of branch prediction algorithms. It is less clear whether the *intellectual bottleneck* that Backus criticized has changed much since 1977. Backus's proposed solution has not had a major influence. Modern functional programming and object-oriented programming are much less geared towards "pushing vast numbers of words back and forth" than earlier languages like Fortran, but internally, that is still what computers spend much of their time doing.

## Early stored-program computers

The date information in the following chronology is difficult to put into proper order. Some dates are for first running a test program, some dates are the first time the computer was demonstrated or completed, and some dates are for the first delivery or installation.

- The IBM SSEC was a stored-program electromechanical computer and was publicly demonstrated on January 27, 1948. However it was partially electromechanical, thus not fully electronic.
- The Manchester SSEM (the *Baby*) was the first fully electronic computer to run a stored program. It ran a factoring program for 52 minutes on June 21, 1948, after running a simple division program and a program to show that two numbers were relatively prime.
- The ENIAC was modified to run as a stored-program computer (using the Function Tables for program ROM) and was demonstrated as such on September 16, 1948, running a program by Adele Goldstine for von Neumann.
- The BINAC ran some test programs in February, March, and April 1949, although it wasn't completed until September 1949.
- The Manchester Mark I grew out of the SSEM project. An intermediate version

of the Mark I was available to run programs in April 1949, but it wasn't completed until October 1949.

- The EDSAC ran its first program on May 6, 1949.
- The EDVAC was delivered in August 1949, but it had problems that kept it from being put into regular operation until 1951.
- The CSIR Mk I ran its first program in November 1949.
- The SEAC was demonstrated in April 1950.
- The Pilot ACE ran its first program on May 10, 1950 and was demonstrated in December 1950.
- The SWAC was completed in July 1950.
- The Whirlwind was completed in December 1950 and was in actual use in April 1951.
- The first ERA Atlas (later the commercial ERA 1101/UNIVAC 1101) was installed in December 1950.

## References

1. ^ "MFTL" entry, Jargon File 4.4.7 (http://catb.org/~esr/jargon/html/M/MFTL.html)
2. ^ First Draft of a Report on the EDVAC (http://www.virtualtravelog.net/entries/2003-08-TheFirstDraft.pdf) (PDF, 420 KB)

- *The First Computers: History and Architectures*, edited by Raúl Rojas and Ulf Hashagen, MIT Press, 2000. ISBN 0-262-18197-5.
- *From Dits to Bits... : A Personal History of the Electronic Computer*, Herman Lukoff, 1979. Robotics Press, ISBN 978-0-89661-002-6
- Martin Davis (2000), Chapter 8, "Making the first Universal computers", *Engines of Logic: Mathematicians and the origin of the Computer*, W. W. Norton & Company, Inc. New York. ISBN 0-393-32229-7 pbk.
- *Can Programming be Liberated from the von Neumann Style?*, John Backus, 1977 ACM Turing Award Lecture. Communications of the ACM, August 1978, Volume 21, Number 8. Online PDF (http://www.stanford.edu/class/cs242/readings/backus.pdf)
- C. Gordon Bell and Allen Newell (1971), *Computer Structures: Readings and Examples*, McGraw-Hill Book Company, New York. Massive (668 pages).

## See also

- Harvard architecture
- Modified Harvard architecture
- Turing machine
- Random access machine
- Little man computer
- CARDboard Illustrative Aid to Computation
- Von Neumann syndrome

Retrieved from "http://en.wikipedia.org/wiki/Von_Neumann_architecture"

Categories: Computer architecture | Flynn's Taxonomy | Reference models