5. What von Neumann Knew: Computer Architecture

The term **Computer Architecture** may refer to the entire hardware level of the computer. However, it is often used to refer to the design and implementation of the digital processor part of the computer hardware, and we focus on the computer processor architecture in this chapter.

The **central processing unit** (CPU, or processor) is the part of the computer that executes program instructions on program data. Program instructions and data are stored in the computer's Random Access Memory (RAM). A particular digital processor implements a specific **Instruction Set Architecture (ISA)**, which defines the set of instructions and their binary encoding, the set of CPU registers, and the effects of executing instructions on the state of the processor. There are many different ISAs, including: SPARC, IA32, MIPS, ARM, ARC, PowerPC, and x86 ISAs (which include IA32 and x86-64). **Microarchitectures** define the circuitry of an implementation of a specific ISA. Microarchitecture implementations of the same ISA can differ as long as they implement the ISA definition. For example, Intel and AMD produce different microprocessor implementations of IA32 ISA.

Some ISAs define a **Reduced Instruction Set Computer (RISC)**, and others define a **Complex Instruction Set Computer (CISC)**. RISC ISAs have a small set of basic instructions that execute quickly; each instruction executes in about a single processor clock cycle, and compilers combine sequences of several basic RISC instructions to implement higher-level functionality. RISC ISAs are also load-store architectures, which means that they provide explicit instructions to load (read) values from RAM into CPU registers and to store (write) values in CPU registers to RAM. All other RISC instructions use CPU registers for source and destination operands. For example, a RISC ADD instruction reads its operand values from registers and writes the result of their sum to a register. RISC ISAs also typically support a small number of different addressing modes (ways to express memory addresses). Additionally, all RISC instructions are fixed-length (the same number of bytes), which simplifies parts of instruction execution.

In contrast, CISC ISA's instructions provide higher-level functionality than RISC instructions. For example, a single CISC instruction may perform a sequence of low-level functionality and take several clock cycles to execute. CISC ISAs also export a larger set of instructions than RISC, support more complicated addressing modes, and allow for variable-length instructions. CISC ISAs typically support instruction operands that explicitly refer to memory locations. For example, a CISC ADD instruction may specify the memory address of an operand value to read or the memory address of the result to write. As a result, a single CISC ADD instruction can perform a sequence of low-level operations that would require several RISC instructions to perform (load value from memory, add it to a value in a register, and store the result to memory).

RISC versus CISC

In the early 1980s, researchers at Berkeley and Stanford Universities developed RISC through the Berkeley RISC project and the Stanford MIPs project. David Paterson of Berkeley and John Hennessy of Stanford won the 2017 Turing Award[1] (the highest award in computing) for their work developing RISC architectures.

At the time of its development, the RISC architecture was a radical departure from the commonly held view that ISAs needed to be increasingly complex to achieve high performance. "The RISC approach differed from the prevailing complex instruction set computer (CISC) computers of the time in that it required a small set of simple and general instructions (functions a computer must perform), requiring fewer transistors than complex instruction sets and reducing the amount of work a computer must perform."[2]

CISC ISAs express programs in fewer instructions than RISC, often resulting in smaller program executables. On systems with small main memory, the size of the program executable is an important factor in the program's performance, since a large executable leaves less RAM space available for other parts of a running program's memory space. Microarchitectures based on CISC are also typically specialized to efficiently execute the CISC variable length and higher-functionality instructions. Specialized circuitry for executing more complex instructions may result in more efficient execution of specific higher-level functionality, but at the cost of requiring more complexity for all instruction execution.

In comparing RISC to CISC, RISC programs contain more instructions to execute, but each instruction executes much more efficiently than most CISC instructions, and RISC allow for simpler microarchitecture designs than CISC. CISC programs contain fewer instructions, and CISC microarchitectures are designed to execute more complicated instructions efficiently, but they require more complex microarchitecture designs and faster clock rates. In general, RISC processors result in more efficient design and better performance. As computer memory sizes have increased over time, the size of the program executable is less important to a program's performance. CISC, however, has been the dominant ISA due in large part to it being implemented by and supported by industry.

Today, CISC remains the dominant ISA for desktop and many server-class computers. For example, Intel's x86 ISAs are CISC-based. RISC ISAs are more commonly seen in high-end servers (e.g. SPARC) and in mobile devices (e.g. ARM) due to their low power requirements. A particular microarchitecture implementation of a RISC or CISC ISA may incorporate both RISC and CISC design under the covers. For example, most CISC processors use microcode to encode some CISC instructions in a more RISC-like instruction set that the underlying processor executes, and some modern RISC instruction sets contain a few more complex instructions or addressing modes than the initial MIPS and Berkeley RISC instruction sets.

All modern processors, regardless of their ISA, adhere to the von Neumann Architecture model. The general-purpose design of the von Neumann architecture allows it to execute any type of program. It uses a stored-program model, meaning that the program instructions reside in computer memory along with program data, and both are inputs to the processor.

This chapter introduces the von Neumann architecture and the ancestry and components that underpin modern computer architecture. We build an example digital processor (CPU) based on the von Neumann architecture model, design a CPU from digital circuits that are constructed from logic gate building blocks, and demonstrate how the CPU executes program instructions.

References

1. ACM A. M. Turing Award Winners. https://amturing.acm.org/
2. "Pioneers of Modern Computer Architecture Receive ACM A.M. Turing Award", ACM Media Center Notice, March 2018. https://www.acm.org/media-center/2018/march/turing-award-2017

5.1. The Origin of Modern Computing Architectures

When tracing the ancestry of modern computing architecture, it is tempting to consider that modern computers are part of a linear chain of successive transmutations, with each machine simply an improvement of the one that previously existed. While this view of inherited improvements in computer design may hold true for certain classes of architecture (consider the iterative improvements of the iPhone X from the original iPhone), the root of the architectural tree is much less defined.

From the 1700s until the early 1900s, mathematicians served as the first *human* computers for calculations related to applications of science and engineering[1]. The word "computer" originally referred to "one who computes". Women mathematicians often served in the role of computer. In fact, the use of women as human computers was so pervasive that computational complexity was measured in "kilo-girls", or the amount of work a thousand human computers could complete in one hour[2]. Women were widely considered to be better at doing mathematical calculations than men, as they tended to be more methodical. Women were not allowed to hold the position of Engineer. As such, they were relegated to more "menial" work, such as computing complex calculations.

The first general-purpose digital computer, the Analytical Engine, was designed by British mathematician Charles Babbage, who is credited by some as the father of the computer. The Analytical Engine was an extension of his original invention, the Difference Engine, a mechanical calculator that was capable of calculating polynomial functions. Ada Lovelace, who perhaps should be known as the mother of computing, was the very first person to develop a computer program and the first to publish an algorithm that could be computed using Charles Babbage's Analytical Engine. In her notes is included her recognition of the general-purpose nature of the Analytical Engine: "[t]he Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.[3]" However, unlike modern computers, the Analytical Engine was a mechanical device and was only partially built. Most of the designers of what became the direct forerunners to the modern computer were unaware of the work of Babbage and Lovelace when they developed their own machines.

Thus, it is perhaps more accurate to think about modern computer architecture rising out of a primordial soup of ideas and innovations that arose in the 1930s and 1940s. For example, in 1937, Claude Shannon, a student at MIT, wrote what would go on to be perhaps the most influential masters thesis of all time. Drawing upon the work of George Boole (the mathematician who developed Boolean algebra), Shannon showed that Boolean logic could be applied to circuits and could be used to develop electrical switches. This would lead to the development of the binary computing system, and much of future digital circuit design. While men would design many early electronic computers, women (who were not allowed to be engineers) became programming pioneers, leading the design and development of many early software innovations, such as programming languages, compilers, algorithms, and operating systems.

While a comprehensive discussion of the rise of computer architecture is not possible in this book (see *Turing's Cathedral*[4] by George Dyson and *The Innovators*[6] by Walter Isaacson for more detailed coverage), we briefly enumerate several significant innovations that occurred in the 1930s and 1940s that were instrumental in the rise of modern computer architecture.

5.1.1. The Turing Machine

In 1937, British mathematician Alan Turing proposed[7] the "Logical Computing Machine", a theoretical computer. Turing used this machine to prove that there exists no solution to the decision problem (in German, the *Entscheidungsproblem*), posed by mathematicians David Hilbert and Wilhelm Ackermann in 1928. The decision problem is an algorithm that takes a statement as input and determines if the statement is universally valid. Turing proved that no such algorithm exists by showing that Halting Problem ("will machine *X* halt on input *y*?") was undecidable for Turing's machine. As part of this proof, Turing described a universal machine that is capable of performing the tasks of any other computing machine. Alonzo Church, Turing's dissertation advisor at Princeton University, was the first to refer to the "Logical Computing Machine" as the "Turing machine", and its universal form as the universal Turing machine.

Turing later returned to England and served his country as part of the code breaking unit in Bletchley Park during World War II. He was instrumental in the design and construction of the Bombe, an electro-mechanical device that helped break the cipher produced by the Enigma machine, which was commonly used by Nazi Germany to protect sensitive communication during World War II.

After the War, Turing designed the Automatic Computing Engine (ACE). The ACE was a stored-program computer, meaning that both the program instructions and its data are loaded into the computer memory and run by the general-purpose computer. His paper, published in 1946, is perhaps the most detailed description of such a computer[8].

5.1.2. Early Electronic Computers

World War II accelerated much of the development of early computers. However, due to the classified nature of military operations in World War II, much of the details about innovations that occurred as a result of the frenetic activity during the war was not publicly acknowledged until years later. A good example of this is Colossus, a machine designed by British engineer Tommy Flowers to help break the Lorenz Cipher, which was used by the Nazi Germany to encode high-level intelligence communication. Some of Alan Turing's work aided in its design. Built in 1943, the Colossus is arguably the first programmable, digital, and fully electronic computer. However, it was a special-purpose computer, designed specifically for code breaking. The Women's Royal Naval Service (WRNS) (known as "wrens") served as operators of Colossus. In spite of the General Report of the Tunny[14] noting that several of the wrens showed ability in cryptographic work, none of them were given the position of crytographer, and instead were delegated more menial Colossus operation tasks[5],[15].

On the other side of the of the Atlantic, American scientists and engineers were hard at work creating computers of their own. Harvard professor Howard Aiken (who was also a Naval Commander in the U.S. Navy Reserves) designed the Mark I, an electro-mechanical, general purpose programmable computer. Built in 1944, it aided in the design of the atomic bomb. Aiken built his computer largely unaware of Turing's work and was motivated by the goal of bringing Charles Babbage's analytical engine to life[6]. A key feature of the MARK I is that it was fully automatic and able to run for days without human intervention[6]. This would be a foundational feature in future computer design.

Meanwhile, American engineers John Mauchly and Presper Eckert of the University of Pennsylvania designed and built the Electronic Numerical Integrator and Computer (ENIAC) in 1945. The ENIAC is arguably the forerunner of modern computers. It was digital (though it used decimal rather than binary), fully electronic, programmable, and general purpose. While the original version of the ENIAC did not have stored-program capabilities, this feature was built into the ENIAC before the end of the decade. ENIAC was financed and built for the U.S. Army's Ballistic Research Laboratory and was designed primarily to calculate ballistic trajectories. Later the ENIAC would be used to aid in the design of the hydrogen bomb.

As men were drafted into the Armed Forces during WWII, women were hired to help in the war effort as human computers. With the arrival of the first electronic computers, women became the first programmers, as programming was considered secretarial work. It should come as no surprise that much of the early innovations in programming, such as the first compiler, the notion of modularizing programs, debugging, and assembly language are credited to women inventors. Grace Hopper, for example, developed the first high-level and machine-independent programming language (COBOL) and its compiler. Hopper was also a programmer for the Mark I and wrote the book that described its operation. The ENIAC programmers were six women: Jean Jennings Bartik, Betty Snyder Holberton, Kay McNulty Mauchly, Frances Bilas Spence, Marlyn Wescoff Meltzer, and Ruth Lichterman Teitelbaum. Unlike the wrens, the ENIAC women were given a great deal of autonomy in their task; given just the wiring diagrams of the ENIAC, they were told to figure out how the ENIAC works and how to program it. In addition to their innovation in solving how to program (and debug) one of the world's first electronic general-purpose computers, the ENIAC programmers also developed the idea of algorithmic flow charts, and developed important programming concepts such as subroutines, and nesting. Like Grace Hopper, Jean Jennings Bartik and Betty Snyder Holberton would go on to have long careers in computing, and are some of the early computing pioneers. Unfortunately, the full extent of women's contributions in early computing are not known. Unable to advance, many women left the field after WWII. To learn more about early women programmers, we encourage readers to check out *Recoding Gender* [11] by Janet Abbate, *Top Secret Rosies*, a PBS documentary[12] directed by LeAnn Erickson, and *"The Computers"* by Kathy Kleiman[13].

The British and the Americans were not the only ones interested in the potential of computers. In Germany, Konrad Zuse developed the first electromechanical general-purpose digital programmable computer, the Z3, which was completed in 1941. Zuse came up with his design independently of the work of Turing and others. Notably, Zuse's design used binary (rather than decimal), the first computer of its kind to use the binary system. However, the Z3 was destroyed during aerial bombing of Berlin, and Zuse was unable to continue his work until 1950. His work largely went unrecognized until years later. He is widely considered the father of computing in Germany.

5.1.3. So What Did von Neumann Know?

From our discussion the origin of modern computer architecture, it is apparent that in the 1930s and 1940s there were several innovations that led to the rise of the computer as we know it today. In 1945, Jon von Neumann published a paper called the "First Draft of a Report on the EDVAC"[9], which describes an architecture on which modern computers are based. The EDVAC was the successor of the ENIAC. It differed from the ENIAC in that it was a binary computer instead of decimal, and it was a stored-program computer. Today, this description of the EDVAC's architectural design is known to as the von Neumann architecture.

The **von Neumann architecture** describes a general-purpose computer, one that is designed to run any program. It also uses a stored program model, meaning that program instructions and data are both loaded onto the computer to run. In the von Neumann model there is no distinction between instructions and data; both are loaded into the computer's internal memory, and program instructions are fetched from memory and executed by the computer's functional units that execute program instructions on program data.

John von Neumann's contributions weave in and out of several of the previous stories in computing. A Hungarian mathematician, he was a professor at both the Institute of Advanced Study and Princeton University and served as an early mentor to Alan Turing. Later, von Neumann became a research scientist on the Manhattan Project, which led him to Howard Aiken and the Mark I, and would later serve as a consultant on the ENIAC project, and correspond regularly with Eckert and Mauchly. His famous paper describing the EDVAC came from his work on the the Electronic Discrete Variable Automatic Computer (EDVAC), proposed to the U.S. Army by Eckert and Mauchly, and built at the University of Pennsylvania. The EDVAC included several architectural design innovations that form the foundation of almost all modern computers. Namely it was general-purpose, used the binary numeric system, had internal memory, and was fully electric. In large part because von Neumann was the sole author of the paper[9], the architectural design the paper describes is primarily credited to von Neumann and has become known as the von Neumann architecture. It should be noted that Turing described in great detail the design of a similar machine in 1946. However, since von Neumann's paper was published before Turing's, von Neumann received the chief credit for these innovations.

Regardless of who "really invented" the von Neumann architecture, von Neumann's own contributions should not be diminished. He was a brilliant mathematician and scientist. His contributions to mathematics span the areas of set theory to quantum mechanics and game theory. In computing, he is regarded also as the inventor of the merge sort algorithm. Walter Isaacson, in his book *The Innovators*, argued that one of von Neumann's greatest strengths lay in his ability to collaborate widely and intuitively see the importance of novel concepts[6]. A lot of the early designers of the computer worked in isolation from each other. Isaacson argues that by witnessing the slowness of the Mark I computer, von Neumann was able to intuitively realize the value of a truly electronic computer, and the need to store and modify programs in memory. It could therefore be argued that von Neumann, even more than Eckert and Mauchly, grasped and fully appreciated the power of a fully electronic stored-program computer[6].

5.1.4. References

1. David Alan Grier, *"When Computers Were Human"*, Princeton University Press, 2005.
2. Megan Garber, *"Computing Power Used to be Measured in 'Kilo-Girls'"*. The Atlantic, October 16, 2013. https://www.theatlantic.com/technology/archive/2013/10/computing-power-used-to-be-measured-in-kilo-girls/280633/
3. Betty Alexandra Toole, *"Ada, The Enchantress of Numbers"*. Strawberry Press, 1998.
4. George Dyson, *Turing's Cathedral: the origins of the digital universe*. Pantheon. 2012.
5. Jack Copeland, *"Colossus: The Secrets of Bletchley Park's Code-breaking Computers"*.
6. Walter Isaacson. *"The Innovators: How a group of inventors, hackers, genius and geeks created the digital revolution"*. Simon and Schuster. 2014.
7. Alan M. Turing. *"On computable numbers, with an application to the Entscheidungsproblem". Proceedings of the London mathematical society* 2(1). pp. 230—265. 1937.
8. Brian Carpenter and Robert Doran. *"The other Turing Machine". The Computer Journal* 20(3) pp. 269—279. 1977.
9. John von Neumann. *"First Draft of a Report on the EDVAC (1945)"*. Reprinted in *IEEE Annals of the history of computing* 4. pp. 27—75. 1993.
10. Arthur Burks, Herman Goldstine, John von Neumann. *"Preliminary discussion of the logical design of an electronic computing instrument (1946)"*. Reprinted by *The Origins of Digital Computers* (Springer), pp. 399—413. 1982.
11. Janet Abbate. *"Recoding gender: Women's changing participation in computing"*. MIT Press. 2012.
12. LeAnn Erickson. *"Top Secret Rosies: The Female Computers of World War II"*. Public Broadcasting System. 2010.
13. Kathy Kleiman, *"The Computers"*. http://eniacprogrammers.org/
14. *"Breaking Teleprinter Ciphers at Bletchley Park: An edition of I.J. Good, D. Michie and G. Timms: General Report on Tunny with Emphasis on Statistical Methods (1945)"*. Editors: Reeds, Diffie, Fields. Wiley, 2015.
15. Janet Abbate, *"Recoding Gender"*, MIT Press, 2012.

5.2. The von Neumann Architecture

The von Neumann architecture serves as the foundation for most modern computers. In this section, we briefly characterize the architecture's major components.

The von Neumann Architecture (depicted in Figure 43 below) consists of five main components:

1. The **processing unit** executes program instructions.
2. The **control unit** drives program instruction execution on the processing unit. Together, the processing and control units make up the CPU.
3. The **memory unit** stores program data and instructions.
4. The **input unit(s)** load program data and instructions on the computer and initiate program execution.
5. The **output unit(s)** store or receive program results.

Buses connect the units, and are used by the units to send control and data information to each other. A **bus** is a communication channel that transfers binary values between communication endpoints (the senders and receivers of the values). For example, a data bus that connects the memory unit and the CPU could be implemented as 32 parallel wires that together transfer a 4 byte value, one bit transferred on each wire. Typically, architectures have separate buses for sending data, memory addresses, and control between units. The units use the control bus for sending control signals that request or notify other units of actions, use the address bus to send the memory address of a read or write request to the memory unit, and use the data bus to transfer data between units.

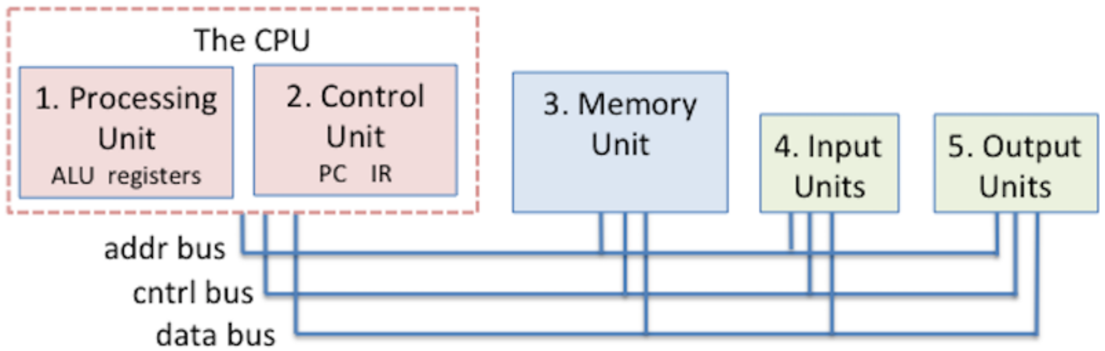

Figure 43. The von Neumann Architecture consists of the processing, control, memory, input, and output units. The control and processing units make up the CPU, which contains the ALU, the general-purpose CPU registers, and some special-purpose registers (IR and PC). The units are connected by buses used for data transfer and communication between the units.

5.2.1. The CPU

Together the control and processing units implement the CPU, which is the part of the computer that executes program instructions on program data.

5.2.2. 1. The Processing Unit

The **Processing Unit** of the von Neumann machine consists of two parts. The first is the **Arithmetic/Logic unit (ALU)** that performs mathematical operations, such as addition, subtraction, and logical or, to name a few. Modern ALUs typically perform a large set of arithmetic operations. The second part of the processing unit is a set of registers. A **register** is a small, fast unit of storage used to hold the program data and instructions that are being executed by the ALU. Crucially, there is no distinction between instructions and data in the von Neumann architecture. For all intents and purposes, instructions *are* data. Each register therefore is capable of holding one data word.

5.2.3. 2. The Control Unit

The **Control Unit** drives the execution of program instructions by loading them from memory and feeding instruction operands and operations through the processing unit. The control unit also includes some storage to keep track of execution state and to determine its next action to take: the **Program Counter** (PC) keeps the memory address of the next instruction to execute; and the **Instruction Register** (IR) stores the instruction, loaded from memory, that is currently being executed.

5.2.4. 3. The Memory Unit

Internal memory is a key innovation of the von Neumann architecture. It provides program data storage that is close to the processing unit, significantly reducing the amount of time to perform calculations. The **Memory Unit** stores both program data and program instructions — storing program instructions is a key part of the stored program model of the von Neumann Architecture.

The size of memory varies from system to system. However, a system's ISA limits the range of addresses that it can express. In modern systems, the smallest addressable unit of memory is one byte (8 bits), and thus each address corresponds to a unique memory location for one byte (8 bits) of storage. As a result, 32-bit architectures support a maximum address space size of 2^{32}, which corresponds to 4 gigabytes (GiB) of addressable memory[1].

The term **Memory** sometimes refers to an entire hierarchy of storage in the system. It can include registers in the processing unit, as well as secondary storage devices like hard disk drives (HDD) or solid state drives (SSD). In the Storage and Memory Hierarchy Chapter we discuss the memory hierarchy in detail. For now, we use the term "memory" interchangeably with internal **Random Access Memory (RAM)** — memory that can be accessed by the central processing unit. RAM storage is random access because all RAM storage locations (addresses) can be accessed directly. It is useful to think of RAM as a linear array of addresses, where each address corresponds to one byte of memory.

Word sizes through history

Recall that **word size**, defined by an ISA, is the number of bits of the standard data size that a processor handles as a single unit. The standard **word size** has fluctuated over the years. For EDVAC, the word size was proposed at 30 bits. In the 1950s, 36-bit word sizes were common. With the innovation of the IBM 360 in the 1960s, word sizes became more or less standardized, and started to expand from 16-bits, to 32 bits, to today's 64 bits. If you examine the Intel Architecture in more detail, you may notice the remnants of some of these old decisions, as 32-bit and 64-bit architectures were added as an extension of the original 16-bit architecture.

5.2.5. 4, 5. The Input and Output (I/O) units

While the control, processing, and memory units form the foundation of the computer, the input and output units enable it to interact with the outside world. In particular, they provide mechanisms for loading a program's instructions and data into memory, storing its data outside of memory, and displaying its results to users.

The **Input Unit** consists of the set of devices that enable a user or program to get data from the outside world onto the computer. The most common forms of input devices today are the keyboard and mouse. Cameras and microphones are other examples.

The **Output Unit** consists of the set of devices that relay results of computation from the computer back to the outside world or that store results outside internal memory. For example, the monitor is a common output device. Other output devices include speakers and haptics.

Some modern devices, such as the touchscreen, act as both input and output, enabling users to both input and receive data from a single unified device.

Solid state and hard drives are another example of devices that act as both input and output devices. These storage devices act act as input devices when they store program executable files that the operating system loads into computer memory to run, and they act as output devices when they store files to which program results are written.

5.2.6. The von Neumann Machine in action: executing a program

The five units that make up the von Neumann Architecture work together to implement a Fetch-Decode-Execute-Store cycle of actions that together execute program instructions. This cycle starts with a program's first instruction, and is repeated until the program exits:

1. **The control unit *Fetches* the next instruction from memory**. The control unit has a special register, the **Program Counter (PC)**, that contains the address of the next instruction to fetch. It places that address on the *Address Bus* and places a *Read* command on the *Control Bus* to the memory unit. The memory unit then reads the bytes stored at the specified address and sends them to the control unit on the *Data Bus*. The **Instruction Register (IR)** stores the bytes of the instruction received from the memory unit. The control unit also increments the PC's value to store the address of the new next instruction to fetch.
2. **The control unit *Decodes* the instruction stored in the IR**. It decodes the instructions bits that encode which operation to perform and bits that encode where the operands are located. It also fetches the data operand values from their locations (from CPU registers or memory), as input to the processing unit.
3. **The processing unit *Executes* the instruction**. The ALU performs the instruction operation on instruction data operands.
4. **The control unit *Stores* the result to memory**. The result of the processing unit's execution of the instruction is stored to memory. The control unit writes the result to memory by placing the result value on the *Data Bus*, placing the address of the storage location on the *Address Bus*, and placing a *Write* command on the *Control Bus*. When received, the memory unit writes the value to memory at the specified address.

The input and output units are not directly involved in the execution of program instructions. Instead, they participate in the program's execution by loading a program's instructions and data on the computer and by storing or displaying the results of the program's computation.

Footnotes

1. The operating system can add support for a 32-bit processor to make use of larger main memory than it can address. However, from the processor's point of view, it can only address up to 2^{32} bytes of memory.

5.3. Logic Gates

Logic gates are the building blocks of the digital circuitry that implements arithmetic, control, and storage functionality in a digital computer. Designing complicated digital circuits involves employing a high-degree of abstraction: a designer creates simple circuits that implement basic functionality from a small set of basic logic gates; these simple circuits, abstracted from their implementation, are used as the building blocks for creating more complicated circuits (simple circuits are combined together to create new circuits with more complicated functionality); these more complicated circuits may be further abstracted and used as a building block for creating even more complicated functionality; and so on to build complete processing, storage, and control components of a processor.

Transistors

Logic gates are created from transistors that are etched into a semiconductor material (e.g. silicon chips). Transistors act as switches that control electrical flow through the chip. A transistor can switch its state between on or off (between a high or low voltage output). Its output state depends on its current state plus its input state (high or low voltage). Binary values are encoded with these high (1) and low (0) voltages, and logic gates are implemented by arrangements of a few transistors that perform switching actions on the inputs to produce a logic gate's output. The number of transistors that can fit on an integrated circuit (a chip), is a rough measure of its power; with more transistors per chip, there are more building blocks to implement more functionality or storage.