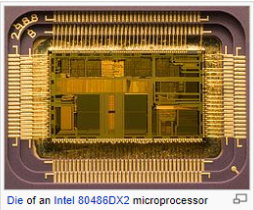
Central processing unit

From Wikipedia, the free encyclopedia

"CPU" redirects here. For other uses, see CPU (disambiguation).

A **Central Processing Unit** (**CPU**) is a machine that can execute computer programs. This broad definition can easily be applied to many early computers that existed long before the term "CPU" ever came into widespread usage. The term itself and its initialism have been in use in the computer industry at least since the early 1960s (Weik 1961). The form, design and implementation of CPUs have changed dramatically since the earliest examples, but their fundamental operation has remained much the same.

Early CPUs were custom-designed as a part of a larger, sometimes one-of-a-kind, computer. However, this costly method of designing custom CPUs for a particular application has largely given way to the development of mass-produced processors that are suited for one or many purposes. This standardization trend generally began in the era of discrete transistor mainframes and minicomputers and has rapidly accelerated with the popularization of the integrated circuit (IC). The IC has allowed increasingly complex CPUs to be designed and manufactured to tolerances on the order of nanometers. Both the miniaturization and standardization of CPUs have increased the presence of these digital devices in modern life far beyond the limited application of dedicated computing machines. Modern microprocessors appear in everything from automobiles to cell phones to children's toys.



Die of an Intel 80486DX2 microprocessor (actual size: 12×6.75 mm) in its packaging.

Contents [hide]

History of CPUs [edit]

Main article: History of general purpose CPUs

Prior to the advent of machines that resemble today's CPUs, computers such as the ENIAC had to be physically rewired in order to perform different tasks. These machines are often referred to as "fixed-program computers," since they had to be physically reconfigured in order to run a different program. Since the term "CPU" is generally defined as a software (computer program) execution device, the earliest devices that could rightly be called CPUs came with the advent of the stored-program computer.

EDVAC, one of the first electronic stored program computers.

The idea of a stored-program computer was already present during ENIAC's design, but was initially omitted so the machine could be finished sooner. On June 30, 1945, before ENIAC was even completed, mathematician John von Neumann distributed the paper entitled "First Draft of a Report on the EDVAC." It outlined the design of a stored-program computer that would eventually be completed in August 1949 (von Neumann 1945). EDVAC was designed to perform a certain number of instructions (or operations) of various types. These instructions could be combined to create useful programs for the EDVAC to run. Significantly, the programs written for EDVAC were stored in high-speed computer memory rather than specified by the physical wiring of the computer. This overcame a severe limitation of ENIAC, which was the large amount of time and effort it took to reconfigure the computer to perform a new task. With von Neumann's design, the program, or software, that EDVAC ran could be changed simply by changing the contents of the computer's memory. [1]

While von Neumann is most often credited with the design of the stored-program computer because of his design of EDVAC, others before him such as Konrad Zuse had suggested similar ideas. Additionally, the so-called Harvard architecture of the Harvard Mark I, which was completed before EDVAC, also utilized a stored-program design using punched paper tape rather than electronic memory. The key difference between the von Neumann and Harvard architectures is that the latter separates the storage and treatment of CPU instructions and data, while the former uses the same memory space for both. Most modern CPUs are primarily von Neumann in design, but elements of the Harvard architecture are commonly seen as well.

Being digital devices, all CPUs deal with discrete states and therefore require some kind of switching elements to differentiate between and change these states. Prior to commercial acceptance of the transistor, electrical relays and vacuum tubes (thermionic valves) were commonly used as switching elements. Although these had distinct speed advantages over earlier, purely mechanical designs, they were unreliable for various reasons. For example, building direct current sequential logic circuits out of relays requires additional hardware to cope with the problem of contact bounce. While vacuum tubes do not suffer from contact bounce, they must heat up before becoming fully operational and eventually stop functioning altogether.[2] Usually, when a tube failed, the CPU would have to be diagnosed to locate the failing component so it could be replaced. Therefore, early electronic (vacuum tube based) computers were generally faster but less reliable than electromechanical (relay based) computers.

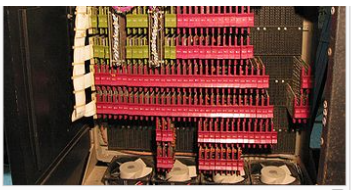
Tube computers like EDVAC tended to average eight hours between failures, whereas relay computers like the (slower, but earlier) Harvard Mark I failed very rarely (Weik 1961:238). In the end, tube based CPUs became dominant because the significant speed advantages afforded generally outweighed the reliability problems. Most of these early synchronous CPUs ran at low clock rates compared to modern microelectronic designs (see below for a discussion of clock rate). Clock signal frequencies ranging from 100 kHz to 4 MHz were very common at this time, limited largely by the speed of the switching devices they were built with.

Discrete transistor and IC CPUs [edit]

The design complexity of CPUs increased as various technologies facilitated building smaller and more reliable electronic devices. The first such improvement came with the advent of the transistor. Transistorized CPUs during the 1950s and 1960s no longer had to be built out of bulky, unreliable, and fragile switching elements like vacuum tubes and electrical relays. With this improvement more complex and reliable CPUs were built onto one or several printed circuit boards containing discrete (individual) components.

- اردو
- ייִדיש
- 粵語
- 中文

During this period, a method of manufacturing many transistors in a compact space gained popularity. The integrated circuit (**IC**) allowed a large number of transistors to be manufactured on a single semiconductor-based die, or "chip." At first only very basic non-specialized digital circuits such as NOR gates were miniaturized into ICs. CPUs based upon these "building block" ICs are generally referred to as "small-scale integration" (**SSI**) devices. SSI ICs, such as the ones used in the Apollo guidance computer, usually contained transistor counts numbering in multiples of ten. To build an entire CPU out of SSI ICs required thousands of individual chips, but still consumed much less space and power than earlier discrete transistor designs. As microelectronic technology advanced, an increasing number of transistors were placed on ICs, thus decreasing the quantity of individual ICs needed for a complete CPU. **MSI** and **LSI** (medium- and large-scale integration) ICs increased transistor counts to hundreds, and then thousands.



CPU, core memory, and external bus interface of a DEC PDP-8/I. made of medium-scale integrated circuits

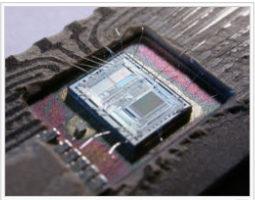
In 1964 IBM introduced its System/360 computer architecture which was used in a series of computers that could run the same programs with different speed and performance. This was significant at a time when most electronic computers were incompatible with one another, even those made by the same manufacturer. To facilitate this improvement, IBM utilized the concept of a microprogram (often called "microcode"), which still sees widespread usage in modern CPUs (Amdahl *et al.* 1964). The System/360 architecture was so popular that it dominated the mainframe computer market for the decades and left a legacy that is still continued by similar modern computers like the IBM zSeries. In the same year (1964), Digital Equipment Corporation (DEC) introduced another influential computer aimed at the scientific and research markets, the PDP-8. DEC would later introduce the extremely popular PDP-11 line that originally was built with SSI ICs but was eventually implemented with LSI components once these became practical. In stark contrast with its SSI and MSI predecessors, the first LSI implementation of the PDP-11 contained a CPU composed of only four LSI integrated circuits (Digital Equipment Corporation 1975).

Transistor-based computers had several distinct advantages over their predecessors. Aside from facilitating increased reliability and lower power consumption, transistors also allowed CPUs to operate at much higher speeds because of the short switching time of a transistor in comparison to a tube or relay. Thanks to both the increased reliability as well as the dramatically increased speed of the switching elements (which were almost exclusively transistors by this time), CPU clock rates in the tens of megahertz were obtained during this period. Additionally, while discrete transistor and IC CPUs were in heavy usage, new high-performance designs like SIMD (Single Instruction Multiple Data) vector processors began to appear. These early experimental designs later gave rise to the era of specialized supercomputers like those made by Cray Inc.

Microprocessors [edit]

Main article: Microprocessor

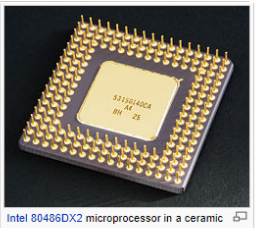
The introduction of the microprocessor in the 1970s significantly affected the design and implementation of CPUs. Since the introduction of the first microprocessor (the Intel 4004) in 1970 and the first widely used microprocessor (the Intel 8080) in 1974, this class of CPUs has almost completely overtaken all other central processing unit implementation methods. Mainframe and minicomputer manufacturers of the time launched proprietary IC development programs to upgrade their older computer architectures, and eventually produced instruction set compatible microprocessors that were backward-compatible with their older hardware and software. Combined with the advent and eventual vast success of the now ubiquitous personal computer, the term "CPU" is now applied almost exclusively to microprocessors.



The integrated circuit from an Intel 8742, an 8-bit microcontroller that includes a CPU running at 12 MHz, 128 bytes of RAM, 2048 bytes of EPROM, and I/O in the same chip.

Previous generations of CPUs were implemented as discrete components and numerous small integrated circuits (ICs) on one or more circuit boards. Microprocessors, on the other hand, are CPUs manufactured on a very small number of ICs; usually just one. The overall smaller CPU size as a result of being implemented on a single die means faster switching time because of physical factors like decreased gate parasitic capacitance. This has allowed synchronous microprocessors to have clock rates ranging from tens of megahertz to several gigahertz. Additionally, as the ability to construct exceedingly small transistors on an IC has increased, the complexity and number of transistors in a single CPU has increased dramatically. This widely observed trend is described by Moore's law, which has proven to be a fairly accurate predictor of the growth of CPU (and other IC) complexity to date.

While the complexity, size, construction, and general form of CPUs have changed drastically over the past sixty years, it is notable that the basic design and function has not changed much at all. Almost all common CPUs today can be very accurately described as von Neumann stored-program machines. As the aforementioned Moore's law continues to hold true, concerns have arisen about the limits of integrated circuit transistor technology. Extreme miniaturization of electronic gates is causing the effects of phenomena like electromigration and subthreshold leakage to become much more significant. These newer concerns are among the many factors causing researchers to investigate new methods of computing such as the quantum computer, as well as to expand the usage of parallelism and other methods that extend the usefulness of the classical von Neumann model.



Intel 80486DX2 microprocessor in a ceramic PGA package.

CPU operation [edit]

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions called a program.The program is represented by a series of numbers that are kept in some kind of computer memory. There are four steps that nearly all von Neumann CPUs use in their operation: **fetch**, **decode**, **execute**, and **writeback**.

The first step, **fetch**, involves retrieving an instruction (which is represented by a number or sequence of numbers) from program memory. The location in program memory is determined by a program counter (PC), which stores a number that identifies the current position in the program. In other words, the program counter keeps track of the CPU's place in the current program. After an instruction is fetched, the PC is incremented by the length of the instruction word in terms of memory units.[3] Often the instruction to be fetched must be retrieved from relatively slow memory, causing the CPU to stall while waiting for the instruction to be returned. This issue is largely addressed in modern processors by caches and pipeline architectures (see below).

The instruction that the CPU fetches from memory is used to determine what the CPU is to do. In the **decode** step, the instruction is broken up into parts that have significance to other portions of the CPU. The way in which the numerical instruction value is interpreted is defined by the CPU's instruction set architecture(**ISA**).[4] Often, one group of numbers in the instruction, called the opcode, indicates which operation to perform. The remaining parts of the number usually provide information required for that instruction, such as operands for an addition operation. Such operands may be given as a constant value (called an immediate value), or as a place to locate a value: a register or a memory address, as determined by some addressing mode. In older designs the portions of the CPU responsible for instruction decoding were unchangeable hardware devices. However, in more abstract and complicated CPUs and ISAs, a microprogram is often used to assist in translating instructions into various configuration signals for the CPU. This microprogram is sometimes rewritable so that it can be modified to change the way the CPU decodes instructions even after it has been manufactured.

After the fetch and decode steps, the **execute** step is performed. During this step, various portions of the CPU are connected so they can perform the desired operation. If, for instance, an addition operation was requested, an arithmetic logic unit (**ALU**) will be connected to a set of inputs and a set of outputs. The inputs provide the numbers to be added, and the outputs will contain the final sum. The ALU contains the circuitry to perform simple arithmetic and logical operations on the inputs (like addition and bitwise operations). If the addition operation produces a result too large for the CPU to handle, an arithmetic overflow flag in a flags register may also be set .

The final step, **writeback**, simply "writes back" the results of the execute step to some form of memory. Very often the results are written to some internal CPU register for quick access by subsequent instructions. In other cases results may be written to slower, but cheaper and larger, main memory. Some types of instructions manipulate the program counter rather than directly produce result data. These are generally called "jumps" and facilitate behavior like loops, conditional program execution (through the use of a conditional jump), and functions in programs.[5] Many instructions will also change the state of digits in a "flags" register. These flags can be used to influence how a program behaves, since they often indicate the outcome of various operations. For example, one type of "compare" instruction considers two values and sets a number in the flags register according to which one is greater. This flag could then be used by a later jump instruction to determine program flow.

After the execution of the instruction and writeback of the resulting data, the entire process repeats, with the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter. If the completed instruction was a jump, the program counter will be modified to contain the address of the instruction that was jumped to, and program execution continues normally. In more complex CPUs than the one described here, multiple instructions can be fetched, decoded, and executed simultaneously. This section describes what is generally referred to as the "Classic RISC pipeline," which in fact is quite common among the simple CPUs used in many electronic devices (often called microcontroller). It largely ignores the important role of CPU cache, and therefore the **access** stage of the pipeline.

Design and implementation [edit]

Main article: CPU design

Integer range

[edit]

Prerequisites
Computer architecture
Digital circuits

The way a CPU represents numbers is a design choice that affects the most basic ways in which the device functions. Some early digital computers used an electrical model of the common decimal (base ten) numeral system to represent numbers internally. A few other computers have used more exotic numeral systems like ternary (base three). Nearly all modern CPUs represent numbers in binary form, with each digit being represented by some two-valued physical quantity such as a "high" or "low" voltage.[6]

MOS 6502 microprocessor in a dual in-line package, an extremely popular 8-bit design.

Related to number representation is the size and precision of numbers that a CPU can represent. In the case of a binary CPU, a **bit** refers to one significant place in the numbers a CPU deals with. The number of bits (or numeral places) a CPU uses to represent numbers is often called "word size", "bit width", "data path width", or "integer precision" when dealing with strictly integer numbers (as opposed to floating point). This number differs between architectures, and often within different parts of the very same CPU. For example, an 8-bit CPU deals with a range of numbers that can be represented by eight binary digits (each digit having two possible values), that is, 2^8 or 256 discrete numbers. In effect, integer size sets a hardware limit on the range of integers the software run by the CPU can utilize.[7]

Integer range can also affect the number of locations in memory the CPU can **address** (locate). For example, if a binary CPU uses 32 bits to represent a memory address, and each memory address represents one octet (8 bits), the maximum quantity of memory that CPU can address is 2^{32} octets, or 4 GiB. This is a very simple view of CPU address space, and many designs use more complex addressing methods like paging in order to locate more memory than their integer range would allow with a flat address space.

Higher levels of integer range require more structures to deal with the additional digits, and therefore more complexity, size, power usage, and general expense. It is not at all uncommon, therefore, to see 4- or 8-bit microcontrollers used in modern applications, even though CPUs with much higher range (such as 16, 32, 64, even 128-bit) are available. The simpler microcontrollers are usually cheaper, use less power, and therefore dissipate less heat, all of which can be major design considerations for electronic devices. However, in higher-end applications, the benefits afforded by the extra range (most often the additional address space) are more significant and often affect design choices. To gain some of the advantages afforded by both lower and higher bit lengths, many CPUs are designed with different bit widths for different portions of the device. For example, the IBM System/370 used a CPU that was primarily 32 bit, but it used 128-bit precision inside its floating point units to facilitate greater accuracy and range in floating point numbers (Amdahl *et al.* 1964). Many later CPU designs use similar mixed bit width, especially when the processor is meant for general-purpose usage where a reasonable balance of integer and floating point capability is required.

Clock rate

[edit]

Main article: Clock rate

Most CPUs, and indeed most sequential logic devices, are synchronous in nature.[8] That is, they are designed and operate on assumptions about a synchronization signal. This signal, known as a **clock signal**, usually takes the form of a periodic square wave. By calculating the maximum time that electrical signals can move in various branches of a CPU's many circuits, the designers can select an appropriate period for the clock signal.

This period must be longer than the amount of time it takes for a signal to move, or propagate, in the worst-case scenario. In setting the clock period to a value well above the worst-case propagation delay, it is possible to design the entire CPU and the way it moves data around the "edges" of the rising and falling clock signal. This has the advantage of simplifying the CPU significantly, both from a design perspective and a component-count perspective. However, it also carries the disadvantage that the entire CPU must wait on its slowest elements, even though some portions of it are much faster. This limitation has largely been compensated for by various methods of increasing CPU parallelism (see below).

However architectural improvements alone do not solve all of the drawbacks of globally synchronous CPUs. For example, a clock signal is subject to the delays of any other electrical signal. Higher clock rates in increasingly complex CPUs make it more difficult to keep the clock signal in phase (synchronized) throughout the entire unit. This has led many modern CPUs to require multiple identical clock signals to be provided in order to avoid delaying a single signal significantly enough to cause the CPU to malfunction. Another major issue as clock rates increase dramatically is the amount of heat that is dissipated by the CPU. The constantly changing clock causes many components to switch regardless of whether they are being used at that time. In general, a component that is switching uses more energy than an element in a static state. Therefore, as clock rate increases, so does heat dissipation, causing the CPU to require more effective cooling solutions.

One method of dealing with the switching of unneeded components is called clock gating, which involves turning off the clock signal to unneeded components (effectively disabling them). However, this is often regarded as difficult to implement and therefore does not see common usage outside of very low-power designs.[9] Another method of addressing some of the problems with a global clock signal is the removal of the clock signal altogether. While removing the global clock signal makes the design process considerably more complex in many ways, asynchronous (or clockless) designs carry marked advantages in power consumption and heat dissipation in comparison with similar synchronous designs. While somewhat uncommon, entire asynchronous CPUs have been built without utilizing a global clock signal. Two notable examples of this are the ARM compliant AMULET and the MIPS R3000 compatible MiniMIPS. Rather than totally removing the clock signal, some CPU designs allow certain portions of the device to be asynchronous, such as using asynchronous ALUs in conjunction with superscalar pipelining to achieve some arithmetic performance gains. While it is not altogether clear whether totally asynchronous designs can perform at a comparable or better level than their synchronous counterparts, it is evident that they do at least excel in simpler math operations. This, combined with their excellent power consumption and heat dissipation properties, makes them very suitable for embedded computers (Garside *et al.* 1999).

Parallelism

[edit]

Main article: Parallel computing

The description of the basic operation of a CPU offered in the previous section describes the simplest form that a CPU can take. This type of CPU, usually referred to as **subscalar**, operates on and executes one instruction on one or two pieces of data at a time.

Model of a subscalar CPU. Notice that it takes fifteen cycles to complete three instructions.

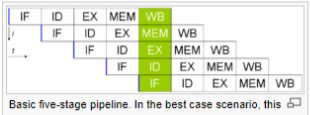
This process gives rise to an inherent inefficiency in subscalar CPUs. Since only one instruction is executed at a time, the entire CPU must wait for that instruction to complete before proceeding to the next instruction. As a result the subscalar CPU gets "hung up" on instructions which take more than one clock cycle to complete execution. Even adding a second execution unit (see below) does not improve performance much; rather than one pathway being hung up, now two pathways are hung up and the number of unused transistors is increased. This design, wherein the CPU's execution resources can operate on only one instruction at a time, can only possibly reach **scalar** performance (one instruction per clock). However, the performance is nearly always subscalar (less than one instruction per cycle).

Attempts to achieve scalar and better performance have resulted in a variety of design methodologies that cause the CPU to behave less linearly and more in parallel. When referring to parallelism in CPUs, two terms are generally used to classify these design techniques. Instruction level parallelism (ILP) seeks to increase the rate at which instructions are executed within a CPU (that is, to increase the utilization of on-die execution resources), and thread level parallelism (TLP) purposes to increase the number of threads (effectively individual programs) that a CPU can execute simultaneously. Each methodology differs both in the ways in which they are implemented, as well as the relative effectiveness they afford in increasing the CPU's performance for an application.[10]

Instruction level parallelism

[edit]

Main articles: Instruction pipelining and Superscalar

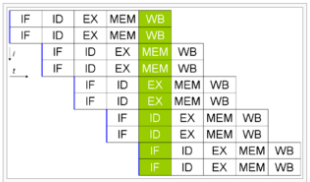


Basic five-stage pipeline. In the best case scenario, this pipeline can sustain a completion rate of one instruction per cycle.

One of the simplest methods used to accomplish increased parallelism is to begin the first steps of instruction fetching and decoding before the prior instruction finishes executing. This is the simplest form of a technique known as **instruction pipelining**, and is utilized in almost all modern general-purpose CPUs. Pipelining allows more than one instruction to be executed at any given time by breaking down the execution pathway into discrete stages. This separation can be compared to an assembly line, in which an instruction is made more complete at each stage until it exits the execution pipeline and is retired.

Pipelining does, however, introduce the possibility for a situation where the result of the previous operation is needed to complete the next operation; a condition often termed data dependency conflict. To cope with this, additional care must be taken to check for these sorts of conditions and delay a portion of the instruction pipeline if this occurs. Naturally, accomplishing this requires additional circuitry, so pipelined processors are more complex than subscalar ones (though not very significantly so). A pipelined processor can become very nearly scalar, inhibited only by pipeline stalls (an instruction spending more than one clock cycle in a stage).

Further improvement upon the idea of instruction pipelining led to the development of a method that decreases the idle time of CPU components even further. Designs that are said to be **superscalar** include a long instruction pipeline and multiple identical execution units. [Huynh 2003] In a superscalar pipeline, multiple instructions are read and passed to a dispatcher, which decides whether or not the instructions can be executed in parallel (simultaneously). If so they are dispatched to available execution units, resulting in the ability for several instructions to be executed simultaneously. In general, the more instructions a superscalar CPU is able to dispatch simultaneously to waiting execution units, the more instructions will be completed in a given cycle.



Simple superscalar pipeline. By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed.

Most of the difficulty in the design of a super scalar CPU architecture lies in creating an effective dispatcher. The dispatcher needs to be able to quickly and correctly determine whether instructions can be executed in parallel, as well as dispatch them in such a way as to keep as many execution units busy as possible. This requires that the instruction pipeline is filled as often as possible and gives rise to the need in superscalar architectures for significant amounts of CPU cache. It also makes hazard-avoiding techniques like branch prediction, speculative execution, and out-of-order execution crucial to maintaining high levels of performance. By attempting to predict which branch (or path) a conditional instruction will take, the CPU can minimize the number of times that the entire pipeline must wait until a conditional instruction is completed. Speculative execution often provides modest performance increases by executing portions of code that may or may not be needed after a conditional operation completes. Out-of-order execution somewhat rearranges the order in which instructions are executed to reduce delays due to data dependencies.

In the case where a portion of the CPU is superscalar and part is not, the part which is not suffers a performance penalty due to scheduling stalls. The original Intel Pentium (P5) had two superscalar ALUs which could accept one instruction per clock each, but its FPU could not accept one instruction per clock. Thus the P5 was integer superscalar but not floating point superscalar. Intel's successor to the Pentium architecture, P6, added superscalar capabilities to its floating point features, and therefore afforded a significant increase in floating point instruction performance.

Both simple pipelining and superscalar design increase a CPU's ILP by allowing a single processor to complete execution of instructions at rates surpassing one instruction per cycle (**IPC**).[11] Most modern CPU designs are at least somewhat superscalar, and nearly all general purpose CPUs designed in the last decade are superscalar. In later years some of the emphasis in designing high-ILP computers has been moved out of the CPU's hardware and into its software interface, or ISA. The strategy of the very long instruction word (VLIW) causes some ILP to become implied directly by the software, reducing the amount of work the CPU must perform to boost ILP and thereby reducing the design's complexity.

Thread level parallelism [edit]

Another strategy of achieving performance is to execute multiple programs or threads in parallel. This area of research is known as parallel computing. In Flynn's taxonomy, this strategy is known as Multiple Instructions-Multiple Data or MIMD.

One technology used for this purpose was multiprocessing (MP). The initial flavor of this technology is known as symmetric multiprocessing (SMP), where a small number of CPUs share a coherent view of their memory system. In this scheme, each CPU has additional hardware to maintain a constantly up-to-date view of memory. By avoiding stale views of memory, the CPUs can cooperate on the same program and programs can migrate from one CPU to another. To increase the number of cooperating CPUs beyond a handful, schemes such as non-uniform memory access (NUMA) and directory-based coherence protocols were introduced in the 1990s. SMP systems are limited to a small number of CPUs while NUMA systems have been built with thousands of processors. Initially, multiprocessing was built using multiple discrete CPUs and boards to implement the interconnect between the processors. When the processors and their interconnect are all implemented on a single silicon chip, the technology is known as a multi-core microprocessor.

It was later recognized that finer-grain parallelism existed with a single program. A single program might have several threads (or functions) that could be executed separately or in parallel. Some of earliest examples of this technology implemented input/output processing such as direct memory access as a separate thread from the computation thread. A more general approach to this technology was introduced in the 1970s when systems were designed to run multiple computation threads in parallel. This technology is known as multi-threading (MT). This approach is considered more cost-effective than multiprocessing, as only a small number of components within a CPU is replicated in order to support MT as opposed to the entire CPU in the case of MP. In MT, the execution units and the memory system including the caches are shared among multiple threads. The downside of MT is that the hardware support for multithreading is more visible to software than that of MP and thus supervisor software like operating systems have to undergo larger changes to support MT. One type of MT that was implemented is known as block multithreading, where one thread is executed until it is stalled waiting for data to return from external memory. In this scheme, the CPU would then quickly switch to another thread which is ready to run, the switch often done in one CPU clock cycle. Another type of MT is known as simultaneous multithreading, where instructions of multiple threads are executed in parallel within one CPU clock cycle.

For several decades from the 1970s to early 2000s, the focus in designing high performance general purpose CPUs was largely on achieving high ILP through technologies such as pipelining, caches, superscalar execution, Out-of-order execution, etc. This trend culminated in large, power-hungry CPUs such as the Intel Pentium 4. By the early 2000s, CPU designers were thwarted from achieving higher performance from ILP techniques due to the growing disparity between CPU operating frequencies and main memory operating frequencies as well as escalating CPU power dissipation owing to more esoteric ILP techniques.

CPU designers then borrowed ideas from commercial computing markets such as transaction processing, where the aggregate performance of multiple programs, also known as throughput computing, was more important than the performance of a single thread or program.

This reversal of emphasis is evidenced by the proliferation of dual and multiple core CMP (chip-level multiprocessing) designs and notably, Intel's newer designs resembling its less superscalar P6 architecture. Late designs in several processor families exhibit CMP, including the x86-64 Opteron and Athlon 64 X2, the SPARC UltraSPARC T1, IBM POWER4 and POWER5, as well as several video game console CPUs like the Xbox 360's triple-core PowerPC design, and the PS3's 8-core Cell microprocessor.

Data parallelism [edit]

Main articles: Vector processor and SIMD

A less common but increasingly important paradigm of CPUs (and indeed, computing in general) deals with data parallelism. The processors discussed earlier are all referred to as some type of scalar device.[12] As the name implies, vector processors deal with multiple pieces of data in the context of one instruction. This contrasts with scalar processors, which deal with one piece of data for every instruction. Using Flynn's taxonomy, these two schemes of dealing with data are generally referred to as SISD (single instruction, single data) and SIMD (single instruction, multiple data), respectively. The great utility in creating CPUs that deal with vectors of data lies in optimizing tasks that tend to require the same operation (for example, a sum or a dot product) to be performed on a large set of data. Some classic examples of these types of tasks are multimedia applications (images, video, and sound), as well as many types of scientific and engineering tasks. Whereas a scalar CPU must complete the entire process of fetching, decoding, and executing each instruction and value in a set of data, a vector CPU can perform a single operation on a comparatively large set of data with one instruction. Of course, this is only possible when the application tends to require many steps which apply one operation to a large set of data.

Most early vector CPUs, such as the Cray-1, were associated almost exclusively with scientific research and cryptography applications. However, as multimedia has largely shifted to digital media, the need for some form of SIMD in general-purpose CPUs has become significant. Shortly after floating point execution units started to become commonplace to include in general-purpose processors, specifications for and implementations of SIMD execution units also began to appear for general-purpose CPUs. Some of these early SIMD specifications like Intel's MMX were integer-only. This proved to be a significant impediment for some software developers, since many of the applications that benefit from SIMD primarily deal with floating point numbers. Progressively, these early designs were refined and remade into some of the common, modern SIMD specifications, which are usually associated with one ISA. Some notable modern examples are Intel's SSE and the PowerPC-related AltiVec (also known as VMX).[13]

See also [edit]

- Addressing mode
- CISC
- Computer bus
- Computer engineering
- CPU cooling
- CPU core voltage
- CPU design
- CPU power dissipation
- CPU socket
- Floating point unit
- Instruction pipeline
- Instruction set
- Notable CPU architectures
- RISC
- Wait state
- Ring (computer security)
- Stream processing

Notes [edit]

1. ^ While EDVAC was designed a few years before ENIAC was built, ENIAC was actually retrofitted to execute stored programs in 1948, somewhat before EDVAC was completed. Therefore, ENIAC became a stored program computer before EDVAC was completed, even though stored program capabilities were originally omitted from ENIAC's design due to cost and schedule concerns.
2. ^ Vacuum tubes eventually stop functioning in the course of normal operation due to the slow contamination of their cathodes that occurs when the tubes are in use. Additionally, sometimes the tube's vacuum seal can leak, which accelerates the cathode contamination. See vacuum tube.
3. ^ Since the program counter counts *memory addresses* and not *instructions*, it is incremented by the number of memory units that the instruction word contains. In the case of simple fixed-length instruction word ISAs, this is always the same number. For example, a fixed-length 32-bit instruction word ISA that uses 8-bit memory words would always increment the PC by 4 (except in the case of jumps). ISAs that use variable length instruction words,increment the PC by the number of memory words corresponding to the last instruction's length.
4. ^ Because the instruction set architecture of a CPU is fundamental to its interface and usage, it is often used as a classification of the "type" of CPU. For example, a "PowerPC CPU" uses some variant of the PowerPC ISA. A system can execute a different ISA by running an emulator.
5. ^ Some early computers like the Harvard Mark I did not support any kind of "jump" instruction, effectively limiting the complexity of the programs they could run. It is largely for this reason that these computers are often not considered to contain a CPU proper, despite their close similarity as stored program computers.
6. ^ The physical concept of voltage is an analog one by its nature, practically having an infinite range of possible values. For the purpose of physical representation of binary numbers, set ranges of voltages are defined as one or zero. These ranges are usually influenced by the circuit designs and operational parameters of the switching elements used to create the CPU, such as a transistor's threshold level.
7. ^ While a CPU's integer size sets a limit on integer ranges, this can (and often is) overcome using a combination of software and hardware techniques. By using additional memory, software can represent integers many magnitudes larger than the CPU can. Sometimes the CPU's ISA will even facilitate operations on integers larger than it can natively represent by providing instructions to make large integer arithmetic relatively quick. While this method of dealing with large integers is somewhat slower than utilizing a CPU with higher integer size, it is a reasonable trade-off in cases where natively supporting the full integer range needed would be cost-prohibitive. See Arbitrary-precision arithmetic for more details on purely software-supported arbitrary-sized integers.
8. ^ In fact, all synchronous CPUs use a combination of sequential logic and combinatorial logic. (See boolean logic)
9. ^ One notable late CPU design that uses clock gating is that of the IBM PowerPC-based Xbox 360. It utilizes extensive clock gating in order to reduce the power requirements of the aforementioned videogame console it is used in. (Brown 2005)
10. ^ Neither ILP nor TLP is inherently superior over the other; they are simply different means by which to increase CPU parallelism. As such, they both have advantages and disadvantages, which are often determined by the type of software that the processor is intended to run. High-TLP CPUs are often used in applications that lend themselves well to being split up into numerous smaller applications, so-called "embarrassingly parallel problems." Frequently, a computational problem that can be solved quickly with high TLP design strategies like SMP take significantly more time on high ILP devices like superscalar CPUs, and vice versa.
11. ^ Best-case scenario (or peak) IPC rates in very superscalar architectures are difficult to maintain since it is impossible to keep the instruction pipeline filled all the time. Therefore, in highly superscalar CPUs, average sustained IPC is often discussed rather than peak IPC.
12. ^ Earlier the term scalar was used to compare the IPC (instructions per cycle) count afforded by various ILP methods. Here the term is used in the strictly mathematical sense to contrast with vectors. See scalar (mathematics) and Vector (geometric).
13. ^ Although SSE/SSE2/SSE3 have superseded MMX in Intel's general purpose CPUs, later IA-32 designs still support MMX. This is usually accomplished by providing most of the MMX functionality with the same hardware that supports the much more expansive SSE instruction sets.

References [edit]

- [a] [b] Amdahl, G. M., Blaauw, G. A., & Brooks, F. P. Jr. (1964). "*Architecture of the IBM System/360*". IBM Research.
- [a] Brown, Jeffery (2005). "Application-customized CPU design". IBM developerWorks. Retrieved on 2005-12-17.
- [a] Huynh, Jack (2003). "The AMD Athlon XP Processor with 512KB L2 Cache" pp. 6–11. University of Illinois - Urbana-Champaign. Retrieved on 2007-10-06.
- [a] Digital Equipment Corporation (November 1975). "LSI-11 Module Descriptions". *LSI-11, PDP-11/03 user's manual* (2nd edition ed.). Maynard, Massachusetts: Digital Equipment Corporation. pp.4–3.
- [a] Garside, J. D., Furber, S. B., & Chung, S-H (1999). "*AMULET3 Revealed*". University of Manchester Computer Science Department.
- Hennessy, John A.; Goldberg, David (1996). *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers. ISBN 1-55860-329-8.
- [a] Gary D. Knott (1974) *A proposal for certain process management and intercommunication primitives* ACM SIGOPS Operating Systems Review. Volume 8 , Issue 4 (October 1974). pp. 7 - 44
- [a] MIPS Technologies, Inc. (2005). "*MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set*". MIPS Technologies, Inc..
- [a] Smotherman, Mark (2005). "History of Multithreading". Retrieved on 2005-12-19.
- [a] von Neumann, John (1945). "*First Draft of a Report on the EDVAC*". Moore School of Electrical Engineering, University of Pennsylvania.
- [a] [b] Weik, Martin H. (1961). "*A Third Survey of Domestic Electronic Digital Computing Systems*". Ballistic Research Laboratories.

External links [edit]

Microprocessor designers

- Advanced Micro Devices - Advanced Micro Devices, a designer of primarily x86-compatible personal computer CPUs.
- ARM Ltd - ARM Ltd, one of the few CPU designers that profits solely by licensing their designs rather than manufacturing them. ARM architecture microprocessors are among the most popular in the world for embedded applications.
- Freescale Semiconductor (formerly of Motorola) - Freescale Semiconductor, designer of several embedded and SoC PowerPC based processors.
- IBM Microelectronics - Microelectronics division of IBM, which is responsible for many POWER and PowerPC based designs, including many of the CPUs utilized in late video game consoles.
- Intel Corp - Intel, a maker of several notable CPU lines, including IA-32, IA-64, and XScale. Also a producer of various peripheral chips for use with their CPUs.
- MIPS Technologies - MIPS Technologies, developers of the MIPS architecture, a pioneer in RISC designs.
- NEC Electronics - NEC Electronics, developers of the 78K0 8-bit Architecture, 78K0R 16-bit Architecture, and V850 32-bit Architecture.
- Sun Microsystems - Sun Microsystems, developers of the SPARC architecture, a RISC design.
- Texas Instruments - Texas Instruments semiconductor division. Designs and manufactures several types of low-power microcontrollers among their many other semiconductor products.
- Transmeta - Transmeta Corporation. Creators of low-power x86 compatibles like Crusoe and Efficeon.
- VIA Technologies - Taiwanese maker of low-power x86-compatible CPUs.

Further reading

- How Microprocessors Work

Listen to this article (2 parts) · (info)
Part 1 · *Part 2*
This audio file was created from a revision dated 2006-06-13, and does not reflect subsequent edits to the article. (Audio help)
More spoken articles

v · d · e	CPU technologies [hide]	
Architecture	ISA : CISC · EDGE · EPIC · MISC · OISC · RISC · VLIW · ZISC · Harvard architecture · Von Neumann architecture · 32 bit · 64 bit · 128 bit	
Parallelism	Pipeline	Instruction pipelining · In-Order & Out-of-Order execution · Register renaming · Speculative execution
	Level	Bit · Instruction · Superscalar · Data · Task
	Threads	Multithreading · Simultaneous multithreading · Hyperthreading · Superthreading
	Flynn's taxonomy	SISD · SIMD · MISD · MIMD
Types	Digital signal processor · Microcontroller · System-on-a-chip · Vector processor	
Components	Arithmetic logic unit (ALU) · Floating point unit (FPU) · Backside Bus · Demultiplexer · Registers · Memory management unit (MMU) · Multiplexer · Translation lookaside buffer (TLB) · Cache	
Power management	APM · ACPI (states) · Dynamic frequency scaling · Dynamic voltage scaling · Clock gating	

Categories: Spoken articles | Central processing unit