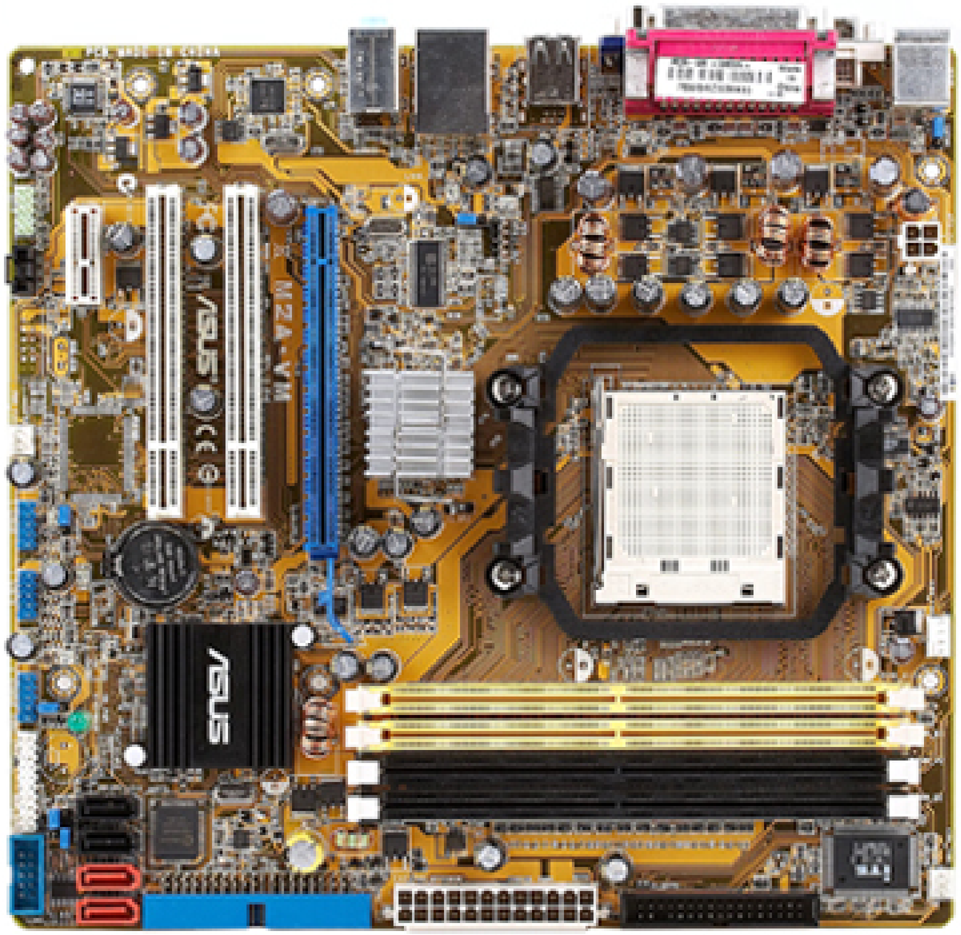
Handbook of Central Processing Unit and its Sockets



Odalys Morrill

Arturo Himes

First Edition, 2012

ISBN 978-81-323-1274-1

Published by:
College Publishing House
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1 - Central Processing Unit

Chapter 2 - History of General Purpose CPUs

Chapter 3 - Microprocessor and CPU Design

Chapter 4 - Instruction Pipeline and Superscalar

Chapter 5 - Vector Processor and SIMD

Chapter 6 - CPU Socket

Chapter 7 - Dual in-line Package

Chapter 8 - Socket 370 and Socket 478

Chapter 9 - Socket 479 and Zero Insertion Force

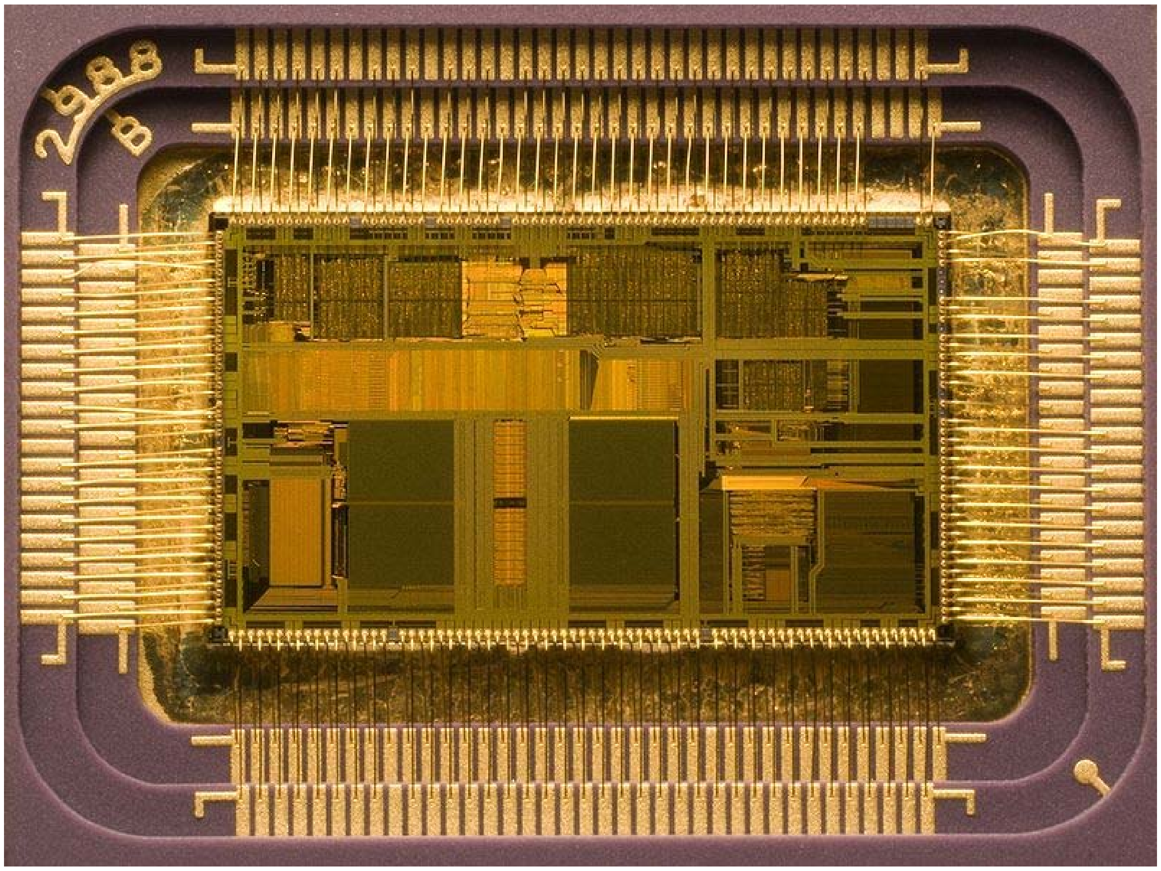
Chapter 10 - Slot 1 and Slot 2

Chapter 11 - LGA 775, LGA 1156 and LGA 1366

Chapter 12 - Plastic Leaded Chip Carrier, LGA 771 and Socket M

Chapter 1

Central Processing Unit

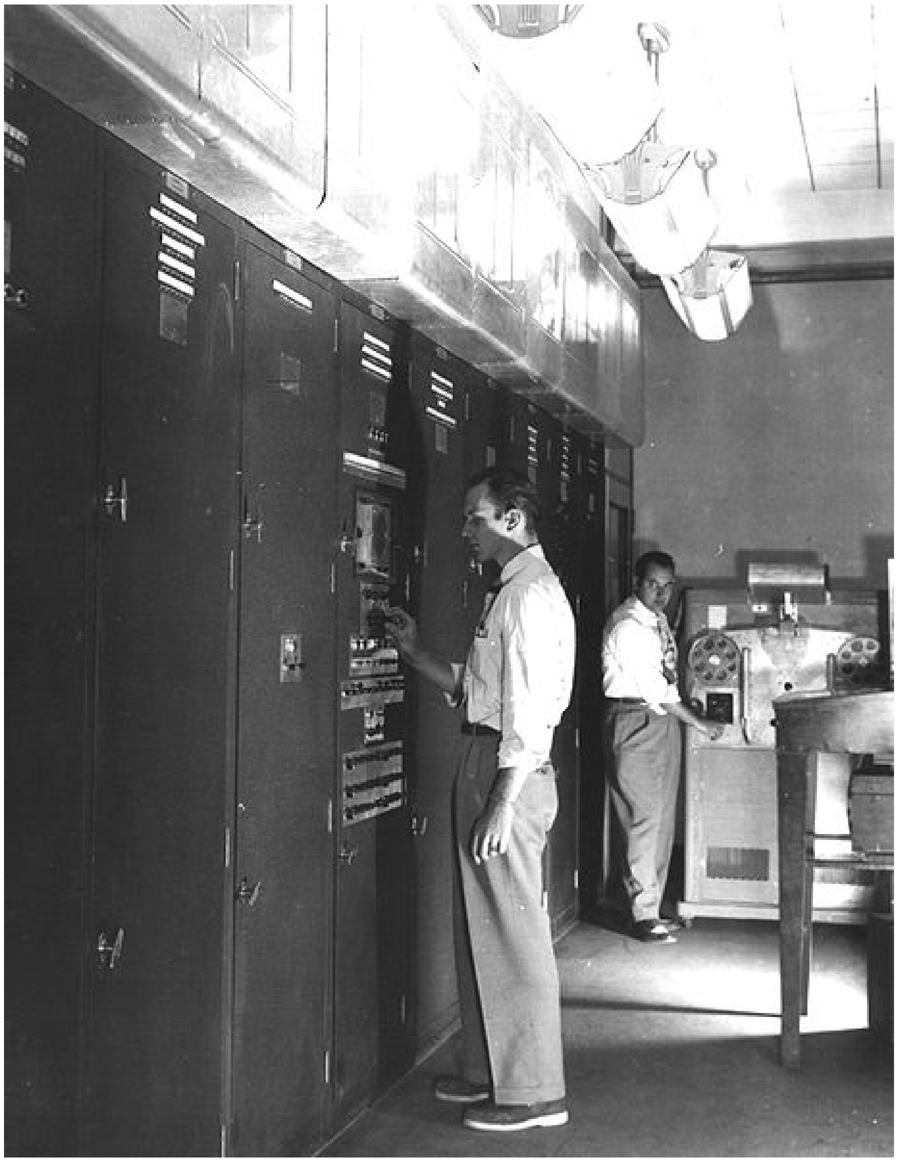


Die of an Intel 80486DX2 microprocessor (actual size: 12×6.75 mm) in its packaging

The **central processing unit** (**CPU**) is the portion of a computer system that carries out the instructions of a computer program, and is the primary element carrying out the computer's functions. The central processing unit carries out each instruction of the program in sequence, to perform the basic arithmetical, logical, and input/output operations of the system. This term has been in use in the computer industry at least since the early 1960s. The form, design and implementation of CPUs have changed dramatically since the earliest examples, but their fundamental operation remains 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 made 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 and children's toys.

History



EDVAC, one of the first electronic stored program computers

Computers such as the ENIAC had to be physically rewired in order to perform different tasks, which caused these machines to be called "fixed-program computers." 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.

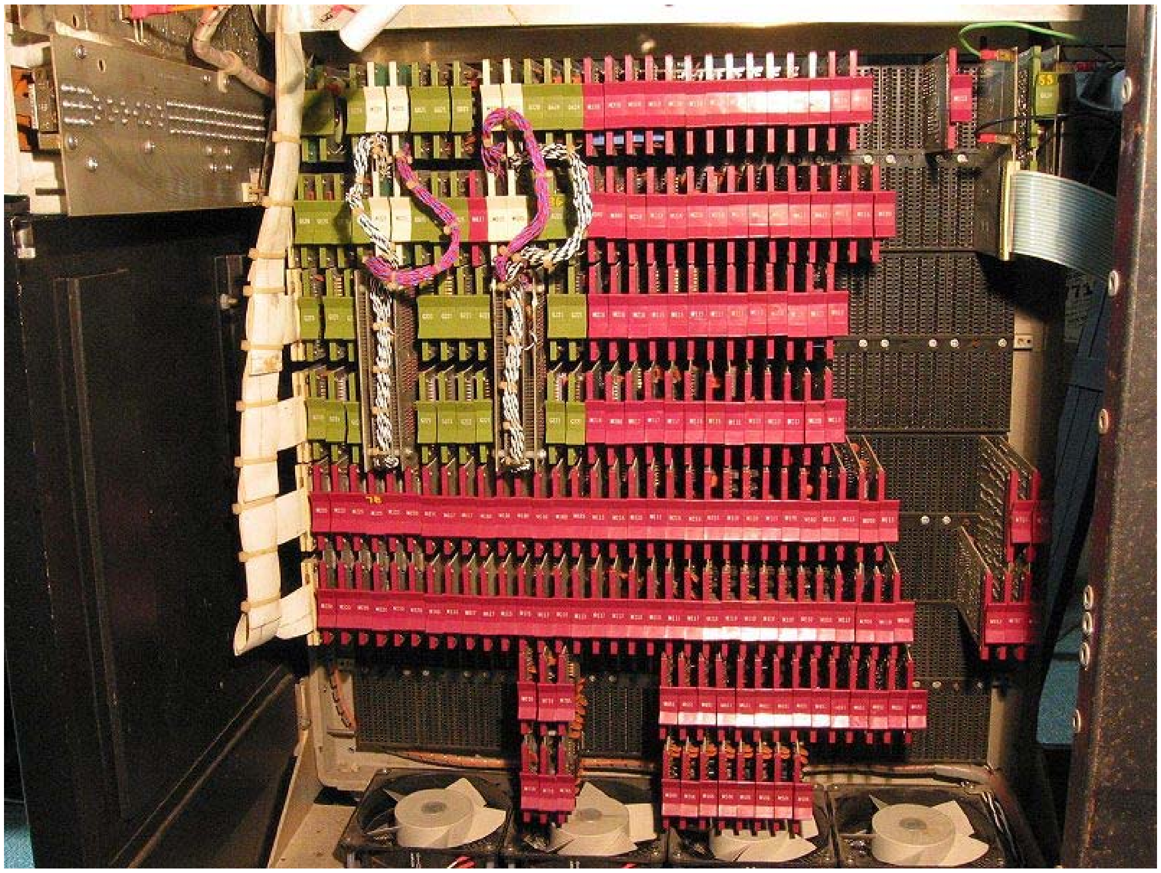
The idea of a stored-program computer was already present in the design of J. Presper Eckert and John William Mauchly's ENIAC, but was initially omitted so the machine could be finished sooner. On June 30, 1945, before ENIAC was completed, mathematician John von Neumann distributed the paper entitled 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. 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 considerable time and effort required 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.

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 and implemented similar ideas. 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.

As a digital device, a CPU is limited to a set of discrete states, and requires some kind of switching elements to differentiate between and change states. Prior to commercial development 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 they eventually cease to function due to slow contamination of their cathodes that occurs in the course of normal operation. If a tube's vacuum seal leaks, as sometimes happens, cathode contamination is accelerated. Usually, when a tube failed, the CPU would have to be diagnosed to locate the failed 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. 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. 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 integrated circuit CPUs



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

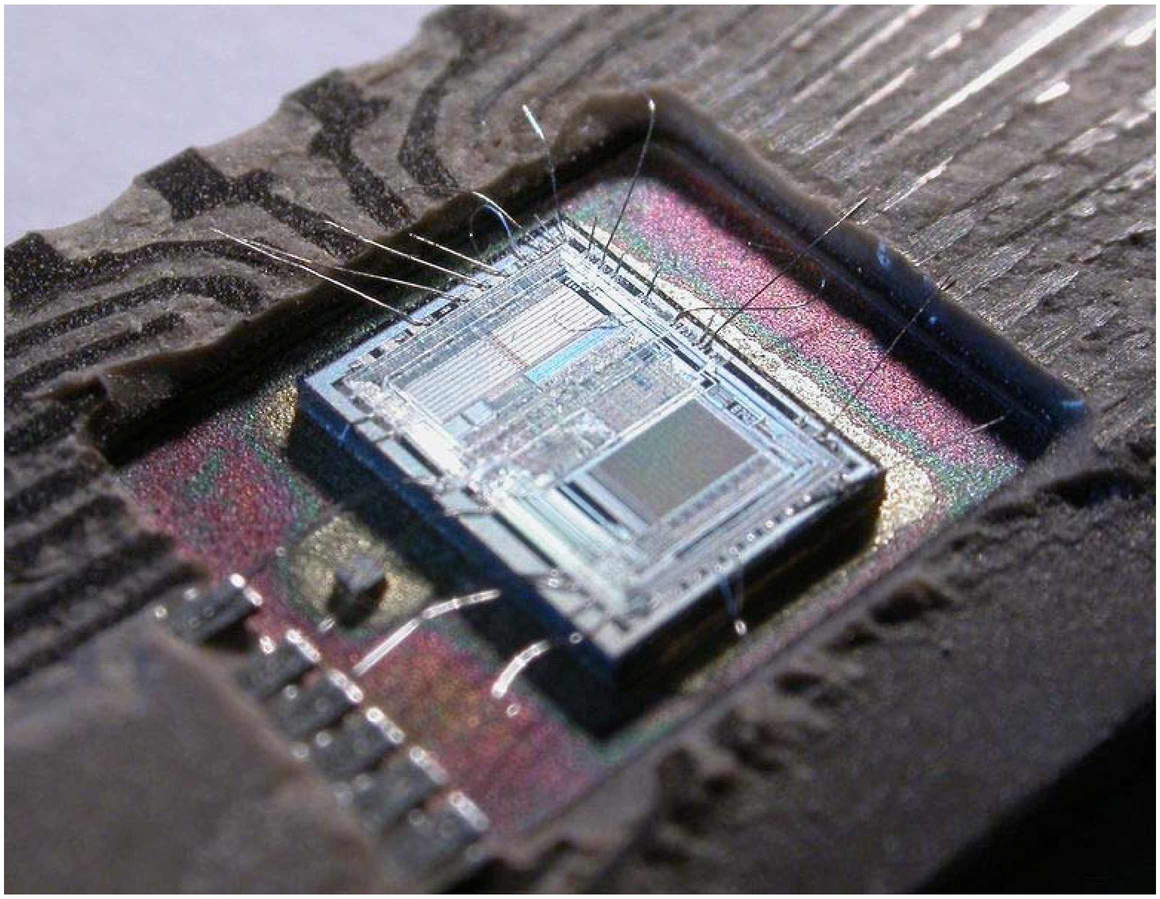
The design complexity of CPUs increased as various technologies facilitated building smaller and more reliable electronic devices. The first such improvement came with the advent of the transistor. Transistorized CPUs during the 1950s and 1960s no longer had to be built out of bulky, unreliable, and 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.

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.

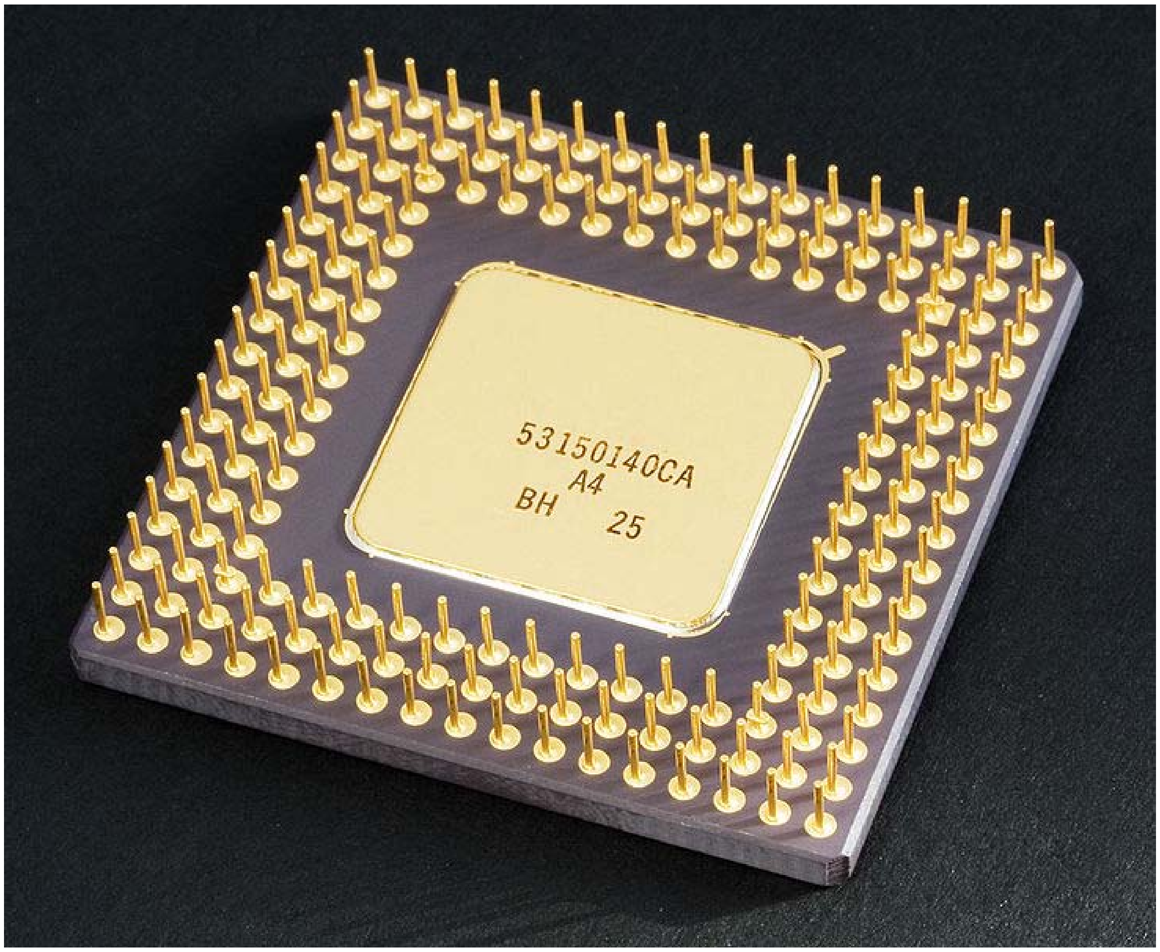
In 1964 IBM introduced its System/360 computer architecture which was used in a series of computers that could run the same programs with different speed and performance. This was significant at a time when most electronic computers were incompatible with one another, even those made by the same manufacturer. To facilitate this improvement, IBM utilized the concept of a microprogram (often called "microcode"), which still sees widespread usage in modern CPUs. The System/360 architecture was so popular that it dominated the mainframe computer market for 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.

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



The die from an Intel 8742



Intel 80486DX2 microprocessor in a ceramic PGA package

The introduction of the microprocessor in the 1970s significantly affected the design and implementation of CPUs. Since the introduction of the first commercially available 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.

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.

Operation

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 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 program. After an instruction is fetched, the PC is incremented by the length of the instruction word in terms of memory units. 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**). 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. 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

Integer range

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.

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.

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. 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

The clock rate is the speed at which a microprocessor executes instructions. Every computer contains an internal clock that regulates the rate at which instructions are executed and synchronizes all the various computer components. The CPU requires a fixed number of clock ticks (or clock cycles) to execute each instruction. The faster the clock, the more instructions the CPU can execute per second.

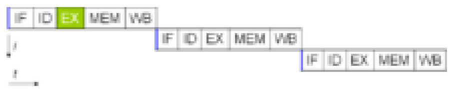
Most CPUs, and indeed most sequential logic devices, are synchronous in nature. That is, they are designed and operate on assumptions about a synchronization signal. This signal, known as a **clock signal**, usually takes the form of a periodic square wave. By calculating the maximum time that electrical 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. 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 in which it is used. 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.

Parallelism



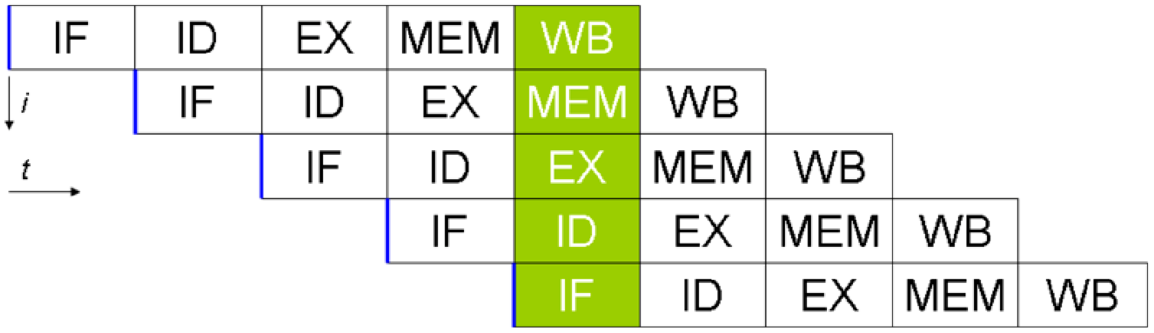
Model of a subscalar CPU. Notice that it takes fifteen cycles to complete three instructions.

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.

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.

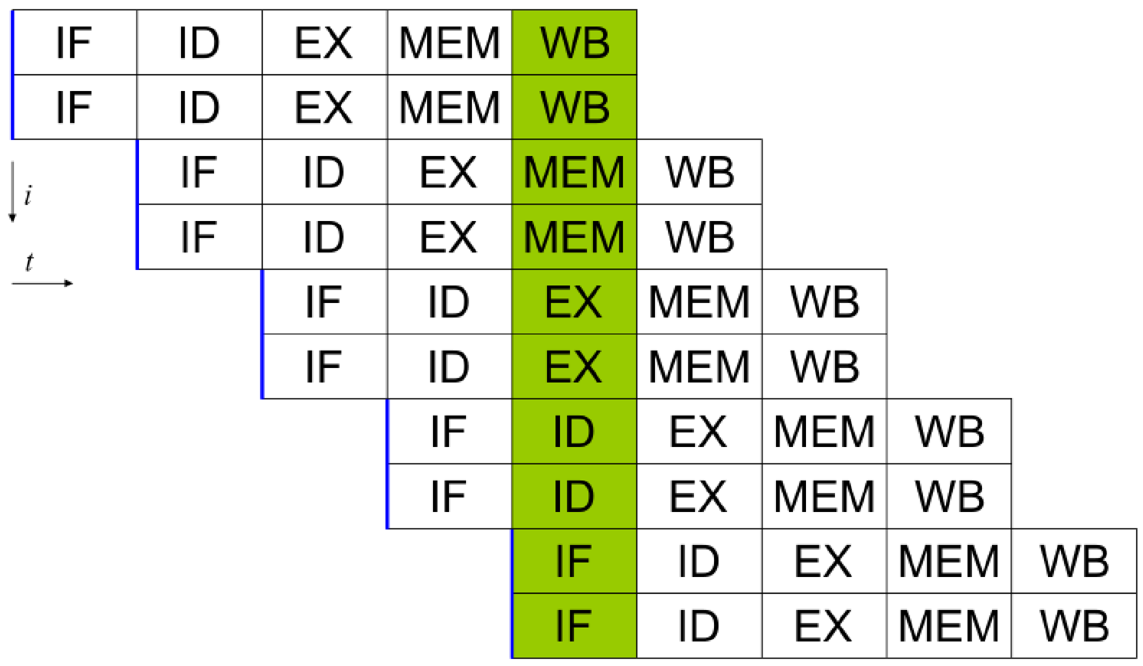
Instruction level parallelism



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).



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

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. 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.

Most of the difficulty in the design of a superscalar 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 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. Also in case of Single Instructions Multiple Data — a case when a lot of data from the same type has to be processed, modern processors can disable parts of the pipeline so that when a single instruction is executed many times, the CPU skips the fetch and decode phases and thus greatly increasing performance on certain occasions, especially in highly monotonous program engines such as video creation software and photo processing.

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 Intel P5 Pentium 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 P5 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**). 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

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, such as the UltraSPARC Technology. 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 7-core Cell microprocessor.

Data parallelism

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. 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 SIMD (single instruction, multiple data) and SISD (single instruction, single 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 HP's Multimedia Acceleration eXtensions (MAX) and 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).

Performance

The performance or speed of a processor depends on the clock rate and the instructions per clock (IPC), which together are the factors for the instructions per second (IPS) that the CPU can perform. Many reported IPS values have represented "peak" execution rates on artificial instruction sequences with few branches, whereas realistic workloads consist of a mix of instructions and applications, some of which take longer to execute than others. The performance of the memory hierarchy also greatly affects processor performance, an issue barely considered in MIPS calculations. Because of these problems, various standardized tests such as SPECint have been developed to attempt to measure the real effective performance in commonly used applications.

Processing performance of computers is increased by using multi-core processors, which essentially is plugging two or more individual processors (called cores in this sense) into one integrated circuit. Ideally, a dual core processor would be nearly twice as powerful as a single core processor. In practice, however, the performance gain is far less, only about fifty percent, due to imperfect software algorithms and implementation.

Chapter 2

History of General Purpose CPUs

The **history of general purpose CPUs** is a continuation of the earlier history of computing hardware.

1950s: early designs

Each of the computer designs of the early 1950s was a unique design; there were no upward-compatible machines or computer architectures with multiple, differing implementations. Programs written for one machine would not run on another kind, even other kinds from the same company. This was not a major drawback at the time because there was not a large body of software developed to run on computers, so starting programming from scratch was not seen as a large barrier.

The design freedom of the time was very important, for designers were very constrained by the cost of electronics, yet just beginning to explore how a computer could best be organized. Some of the basic features introduced during this period included index registers (on the Ferranti Mark 1), a return-address saving instruction (UNIVAC I), immediate operands (IBM 704), and the detection of invalid operations (IBM 650)

By the end of the 1950s commercial builders had developed factory-constructed, truck-deliverable computers. The most widely installed computer was the IBM 650, which used drum memory onto which programs were loaded using either paper tape or punched cards. Some very high-end machines also included core memory which provided higher speeds. Hard disks were also starting to become popular.

Computers are automatic abaci. The type of number system affects the way they work. In the early 1950s most computers were built for specific numerical processing tasks, and many machines used decimal numbers as their basic number system – that is, the mathematical functions of the machines worked in base-10 instead of base-2 as is common today. These were not merely binary coded decimal. Most machines actually had ten vacuum tubes per digit in each register. Some early Soviet computer designers implemented systems based on ternary logic; that is, a bit could have three states: +1, 0, or -1, corresponding to positive, zero, or negative voltage.

An early project for the U.S. Air Force, BINAC attempted to make a lightweight, simple computer by using binary arithmetic. It deeply impressed the industry.

As late as 1970, major computer languages were unable to standardize their numeric behavior because decimal computers had groups of users too large to alienate.

Even when designers used a binary system, they still had many odd ideas. Some used sign-magnitude arithmetic (-1 = 10001), or ones' complement (-1 = 11110), rather than modern two's complement arithmetic (-1 = 11111). Most computers used six-bit character sets, because they adequately encoded Hollerith cards. It was a major revelation to designers of this period to realize that the data word should be a multiple of the character size. They began to design computers with 12, 24 and 36 bit data words.

In this era, Grosch's law dominated computer design: Computer cost increased as the square of its speed.

1960s: the computer revolution and CISC

One major problem with early computers was that a program for one would not work on others. Computer companies found that their customers had little reason to remain loyal to a particular brand, as the next computer they purchased would be incompatible anyway. At that point, price and performance were usually the only concerns.

In 1962, IBM tried a new approach to designing computers. The plan was to make an entire family of computers that could all run the same software, but with different performances, and at different prices. As users' requirements grew they could move up to larger computers, and still keep all of their investment in programs, data and storage media.

In order to do this they designed a single reference computer called the **System/360** (or **S/360**). The System/360 was a virtual computer, a reference instruction set and capabilities that all machines in the family would support. In order to provide different classes of machines, each computer in the family would use more or less hardware emulation, and more or less microprogram emulation, to create a machine capable of running the entire System/360 instruction set.

For instance a low-end machine could include a very simple processor for low cost. However this would require the use of a larger microcode emulator to provide the rest of the instruction set, which would slow it down. A high-end machine would use a much more complex processor that could directly process more of the System/360 design, thus running a much simpler and faster emulator.

IBM chose to make the reference instruction set quite complex, and very capable. This was a conscious choice. Even though the computer was complex, its "control store" containing the microprogram would stay relatively small, and could be made with very fast memory. Another important effect was that a single instruction could describe quite a complex sequence of operations. Thus the computers would generally have to fetch fewer instructions from the main memory, which could be made slower, smaller and less expensive for a given combination of speed and price.

As the S/360 was to be a successor to both scientific machines like the 7090 and data processing machines like the 1401, it needed a design that could reasonably support all forms of processing. Hence the instruction set was designed to manipulate not just simple binary numbers, but text, scientific floating-point (similar to the numbers used in a calculator), and the binary coded decimal arithmetic needed by accounting systems.

Almost all following computers included these innovations in some form. This basic set of features is now called a "complex instruction set computer," or CISC (pronounced "sisk"), a term not invented until many years later.

In many CISCs, an instruction could access either registers or memory, usually in several different ways. This made the CISCs easier to program, because a programmer could remember just thirty to a hundred instructions, and a set of three to ten addressing modes rather than thousands of distinct instructions. This was called an "orthogonal instruction set." The PDP-11 and Motorola 68000 architecture are examples of nearly orthogonal instruction sets.

There was also the BUNCH (Burroughs, UNIVAC, NCR, Control Data Corporation, and Honeywell) that competed against IBM at this time though IBM dominated the era with S/360.

The Burroughs Corporation (which later merged with Sperry/Univac to become Unisys) offered an alternative to S/360 with their B5000 series machines. In 1961, the B5000 had virtual memory, symmetric multiprocessing, a multi-programming operating system (Master Control Program or MCP), written in ALGOL 60, and the industry's first recursive-descent compilers as early as 1963.

1970s: Large Scale Integration

In the 1960s, the Apollo guidance computer and Minuteman missile made the integrated circuit economical and practical.

Around 1971, the first calculator and clock chips began to show that very small computers might be possible. The first microprocessor was the Intel 4004, designed in 1971 for a calculator company (Busicom), and produced by Intel. In 1972, Intel introduced a microprocessor having a different architecture: the 8008. The 8008 is the direct ancestor of the current Intel Core 2, even now maintaining code compatibility (every instruction of the 8008's instruction set has a direct equivalent in the Intel Core 2's much larger instruction set, although the opcode values are different).

By the mid-1970s, the use of integrated circuits in computers was commonplace. The whole decade consists of upheavals caused by the shrinking price of transistors.

It became possible to put an entire CPU on a single printed circuit board. The result was that minicomputers, usually with 16-bit words, and 4k to 64K of memory, came to be commonplace.

CISCs were believed to be the most powerful types of computers, because their microcode was small and could be stored in very high-speed memory. The CISC architecture also addressed the "semantic gap" as it was perceived at the time. This was a defined distance between the machine language, and the higher level language people used to program a machine. It was felt that compilers could do a better job with a richer instruction set.

Custom CISCs were commonly constructed using "bit slice" computer logic such as the AMD 2900 chips, with custom microcode. A bit slice component is a piece of an ALU, register file or microsequencer. Most bit-slice integrated circuits were 4-bits wide.

By the early 1970s, the PDP-11 was developed, arguably the most advanced small computer of its day. Almost immediately, wider-word CISCs were introduced, the 32-bit VAX and 36-bit PDP-10.

Also, to control a cruise missile, Intel developed a more-capable version of its 8008 microprocessor, the 8080.

IBM continued to make large, fast computers. However the definition of large and fast now meant more than a megabyte of RAM, clock speeds near one megahertz , and tens of megabytes of disk drives.

IBM's System 370 was a version of the 360 tweaked to run virtual computing environments. The virtual computer was developed in order to reduce the possibility of an unrecoverable software failure.

The Burroughs B5000/B6000/B7000 series reached its largest market share. It was a stack computer whose OS was programmed in a dialect of Algol.

All these different developments competed for market share.

Early 1980s: the lessons of RISC

In the early 1980s, researchers at UC Berkeley and IBM both discovered that most computer language compilers and interpreters used only a small subset of the instructions of a CISC. Much of the power of the CPU was simply being ignored in real-world use. They realized that by making the computer simpler and less orthogonal, they could make it faster and less expensive at the same time.

At the same time, CPU calculation became faster in relation to the time for necessary memory accesses. Designers also experimented with using large sets of internal registers. The idea was to cache intermediate results in the registers under the control of the compiler. This also reduced the number of addressing modes and orthogonality.

The computer designs based on this theory were called Reduced Instruction Set Computers, or RISC. RISCs generally had larger numbers of registers, accessed by simpler instructions, with a few instructions specifically to load and store data to memory. The result was a very simple core CPU running at very high speed, supporting the exact sorts of operations the compilers were using anyway.

A common variation on the RISC design employs the Harvard architecture, as opposed to the Von Neumann or Stored Program architecture common to most other designs. In a Harvard Architecture machine, the program and data occupy separate memory devices and can be accessed simultaneously. In Von Neumann machines the data and programs are mixed in a single memory device, requiring sequential accessing which produces the so-called "Von Neumann bottleneck."

One downside to the RISC design has been that the programs that run on them tend to be larger. This is because compilers have to generate longer sequences of the simpler instructions to accomplish the same results. Since these instructions need to be loaded from memory anyway, the larger code size offsets some of the RISC design's fast memory handling.

Recently, engineers have found ways to compress the reduced instruction sets so they fit in even smaller memory systems than CISCs. Examples of such compression schemes include the ARM's "Thumb" instruction set. In applications that do not need to run older binary software, compressed RISCs are coming to dominate sales.

Another approach to RISCs was the MISC, "niladic" or "zero-operand" instruction set. This approach realized that the majority of space in an instruction was to identify the operands of the instruction. These machines placed the operands on a push-down (last-in, first out) stack. The instruction set was supplemented with a few instructions to fetch and store memory. Most used simple caching to provide extremely fast RISC machines, with very compact code. Another benefit was that the interrupt latencies were extremely small, smaller than most CISC machines (a rare trait in RISC machines). The Burroughs large systems architecture uses this approach. The B5000 was designed in 1961, long before the term "RISC" was invented. The architecture puts six 8-bit instructions in a 48-bit word, and was a precursor to VLIW design.

The Burroughs architecture was one of the inspirations for Charles H. Moore's Forth programming language, which in turn inspired his later MISC chip designs. For example, his f20 cores had 31 5-bit instructions, which were fit four to a 20-bit word.

RISC chips now dominate the market for 32-bit embedded systems. Smaller RISC chips are even becoming common in the cost-sensitive 8-bit embedded-system market. The main market for RISC CPUs has been systems that require low power or small size.

Even some CISC processors (based on architectures that were created before RISC became dominant), such as newer x86 processors, translate instructions internally into a RISC-like instruction set.

These numbers may surprise many, because the "market" is perceived to be desktop computers. x86 designs dominate desktop and notebook computer sales, but desktop and notebook computers are only a tiny fraction of the computers now sold. Most people in industrialised countries own more computers in embedded systems in their car and house than on their desks.

Mid-to-late 1980s: exploiting instruction level parallelism

In the mid-to-late 1980s, designers began using a technique known as "instruction pipelining", in which the processor works on multiple instructions in different stages of completion. For example, the processor may be retrieving the operands for the next instruction while calculating the result of the current one. Modern CPUs may use over a dozen such stages. MISC processors achieve single-cycle execution of instructions without the need for pipelining.

A similar idea, introduced only a few years later, was to execute multiple instructions in parallel on separate arithmetic logic units (ALUs). Instead of operating on only one instruction at a time, the CPU will look for several similar instructions that are not dependent on each other, and execute them in parallel. This approach is called superscalar processor design.

Such techniques are limited by the degree of instruction level parallelism (ILP), the number of non-dependent instructions in the program code. Some programs are able to run very well on superscalar processors due to their inherent high ILP, notably graphics. However more general problems do not have such high ILP, thus making the achievable speedups due to these techniques to be lower.

Branching is one major culprit. For example, the program might add two numbers and branch to a different code segment if the number is bigger than a third number. In this case even if the branch operation is sent to the second ALU for processing, it still must wait for the results from the addition. It thus runs no faster than if there were only one ALU. The most common solution for this type of problem is to use a type of branch prediction.

To further the efficiency of multiple functional units which are available in superscalar designs, operand register dependencies was found to be another limiting factor. To minimize these dependencies, out-of-order execution of instructions was introduced. In such a scheme, the instruction results which complete out-of-order must be re-ordered in program order by the processor for the program to be restartable after an exception. Out-of-Order execution was the main advancement of the computer industry during the 1990s. A similar concept is speculative execution, where instructions from one direction of a branch (the predicted direction) are executed before the branch direction is known. When the branch direction is known, the predicted direction and the actual direction are compared. If the predicted direction was correct, the speculatively-executed instructions and their results are kept; if it was incorrect, these instructions and their results are thrown out. Speculative execution coupled with an accurate branch predictor gives a large performance gain.

These advances, which were originally developed from research for RISC-style designs, allow modern CISC processors to execute twelve or more instructions per clock cycle, when traditional CISC designs could take twelve or more cycles to execute just one instruction.

The resulting instruction scheduling logic of these processors is large, complex and difficult to verify. Furthermore, the higher complexity requires more transistors, increasing power consumption and heat. In this respect RISC is superior because the instructions are simpler, have less interdependence and make superscalar implementations easier. However, as Intel has demonstrated, the concepts can be applied to a CISC design, given enough time and money.

Historical note: Some of these techniques (e.g. pipelining) were originally developed in the late 1950s by IBM on their Stretch mainframe computer.

1990 to today: looking forward

VLIW and EPIC

The instruction scheduling logic that makes a superscalar processor is just boolean logic. In the early 1990s, a significant innovation was to realize that the coordination of a multiple-ALU computer could be moved into the compiler, the software that translates a programmer's instructions into machine-level instructions.

This type of computer is called a **very long instruction word** (VLIW) computer.

Statically scheduling the instructions in the compiler (as opposed to letting the processor do the scheduling dynamically) can reduce CPU complexity. This can improve performance, reduce heat, and reduce cost.

Unfortunately, the compiler lacks accurate knowledge of runtime scheduling issues. Merely changing the CPU core frequency multiplier will have an effect on scheduling. Actual operation of the program, as determined by input data, will have major effects on scheduling. To overcome these severe problems a VLIW system may be enhanced by adding the normal dynamic scheduling, losing some of the VLIW advantages.

Static scheduling in the compiler also assumes that dynamically generated code will be uncommon. Prior to the creation of Java, this was in fact true. It was reasonable to assume that slow compiles would only affect software developers. Now, with JIT virtual machines for Java and .NET, slow code generation affects users as well.

There were several unsuccessful attempts to commercialize VLIW. The basic problem is that a VLIW computer does not scale to different price and performance points, as a dynamically scheduled computer can. Another issue is that compiler design for VLIW computers is extremely difficult, and the current crop of compilers (as of 2005) don't always produce optimal code for these platforms.

Also, VLIW computers optimise for throughput, not low latency, so they were not attractive to the engineers designing controllers and other computers embedded in machinery. The embedded systems markets had often pioneered other computer improvements by providing a large market that did not care about compatibility with older software.

In January 2000, a company called Transmeta took the interesting step of placing a compiler in the central processing unit, and making the compiler translate from a reference byte code (in their case, x86 instructions) to an internal VLIW instruction set. This approach combines the hardware simplicity, low power and speed of VLIW RISC with the compact main memory system and software reverse-compatibility provided by popular CISC.

Intel's Itanium chip is based on what they call an Explicitly Parallel Instruction Computing (EPIC) design. This design supposedly provides the VLIW advantage of increased instruction throughput. However, it avoids some of the issues of scaling and complexity, by explicitly providing in each "bundle" of instructions information concerning their dependencies. This information is calculated by the compiler, as it would be in a VLIW design. The early versions are also backward-compatible with current x86 software by means of an on-chip emulation mode. Integer performance was disappointing and despite improvements, sales in volume markets continue to be low.

Multi-threading

Current designs work best when the computer is running only a single program, however nearly all modern operating systems allow the user to run multiple programs at the same time. For the CPU to change over and do work on another program requires expensive context switching. In contrast, multi-threaded CPUs can handle instructions from multiple programs at once.

To do this, such CPUs include several sets of registers. When a context switch occurs, the contents of the "working registers" are simply copied into one of a set of registers for this purpose.

Such designs often include thousands of registers instead of hundreds as in a typical design. On the downside, registers tend to be somewhat expensive in chip space needed to implement them. This chip space might otherwise be used for some other purpose.

Multi-core

Multi-core CPUs are typically multiple CPU cores on the same die, connected to each other via a shared L2 or L3 cache, an on-die bus, or an on-die crossbar switch. All the

CPU cores on the die share interconnect components with which to interface to other processors and the rest of the system. These components may include a front side bus interface, a memory controller to interface with DRAM, a cache coherent link to other processors, and a non-coherent link to the southbridge and I/O devices. The terms multi-core and MPU (which stands for **Micro-Processor Unit**) have come into general usage for a single die that contains multiple CPU cores.

Intelligent RAM

One way to work around the Von Neumann bottleneck is to mix a processor and DRAM all on one chip.

- The Berkeley Intelligent RAM (IRAM) project
- eDRAM
- computational RAM

Reconfigurable logic

Another track of development is to combine reconfigurable logic with a general-purpose CPU. In this scheme, a special computer language compiles fast-running subroutines into a bit-mask to configure the logic. Slower, or less-critical parts of the program can be run by sharing their time on the CPU. This process has the capability to create devices such as software radios, by using digital signal processing to perform functions usually performed by analog electronics.

Open source processors

As the lines between hardware and software increasingly blur due to progress in design methodology and availability of chips such as FPGAs and cheaper production processes, even open source hardware has begun to appear. Loosely-knit communities like OpenCores have recently announced completely open CPU architectures such as the OpenRISC which can be readily implemented on FPGAs or in custom produced chips, by anyone, without paying license fees, and even established processor manufacturers like Sun Microsystems have released processor designs (e.g. OpenSPARC) under open-source licenses.

Asynchronous CPUs

Yet another possibility is the "clockless CPU" (asynchronous CPU). Unlike conventional processors, clockless processors have no central clock to coordinate the progress of data through the pipeline. Instead, stages of the CPU are coordinated using logic devices called "pipe line controls" or "FIFO sequencers." Basically, the pipeline controller clocks the next stage of logic when the existing stage is complete. In this way, a central clock is unnecessary.

It might be easier to implement high performance devices in asynchronous logic as opposed to clocked logic:

- components can run at different speeds in the clockless CPU. In a clocked CPU, no component can run faster than the clock rate.
- In a clocked CPU, the clock can go no faster than the worst-case performance of the slowest stage. In a clockless CPU, when a stage finishes faster than normal, the next stage can immediately take the results rather than waiting for the next clock tick. A stage might finish faster than normal because of the particular data inputs (multiplication can be very fast if it is multiplying by 0 or 1), or because it is running at a higher voltage or lower temperature than normal.

Asynchronous logic proponents believe these capabilities would have these benefits:

- lower power dissipation for a given performance level
- highest possible execution speeds

The biggest disadvantage of the clockless CPU is that most CPU design tools assume a clocked CPU (a synchronous circuit), so making a clockless CPU (designing an asynchronous circuit) involves modifying the design tools to handle clockless logic and doing extra testing to ensure the design avoids metastable problems.

Even so, several asynchronous CPUs have been built, including

- the ORDVAC and the identical ILLIAC I (1951)
- the ILLIAC II (1962), the fastest computer in the world at the time
- The Caltech Asynchronous Microprocessor, the world-first asynchronous microprocessor (1988)
- the ARM-implementing AMULET (1993 and 2000)
- the asynchronous implementation of MIPS R3000, dubbed MiniMIPS (1998)
- the SEAforth multi-core processor from Charles H. Moore

Optical communication

One interesting possibility would be to eliminate the front side bus. Modern vertical laser diodes enable this change. In theory, an optical computer's components could directly connect through a holographic or phased open-air switching system. This would provide a large increase in effective speed and design flexibility, and a large reduction in cost. Since a computer's connectors are also its most likely failure point, a busless system might be more reliable, as well.

In addition, current (2010) modern processors use 64- or 128-bit logic. Wavelength superposition could allow for data lanes and logic many orders of magnitude higher, without additional space or copper wires.

Optical processors

Another farther-term possibility is to use light instead of electricity for the digital logic itself. In theory, this could run about 30% faster and use less power, as well as permit a direct interface with quantum computational devices. The chief problem with this approach is that for the foreseeable future, electronic devices are faster, smaller (i.e. cheaper) and more reliable. An important theoretical problem is that electronic computational elements are already smaller than some wavelengths of light, and therefore even wave-guide based optical logic may be uneconomic compared to electronic logic. The majority of development effort, as of 2006 is focused on electronic circuitry.

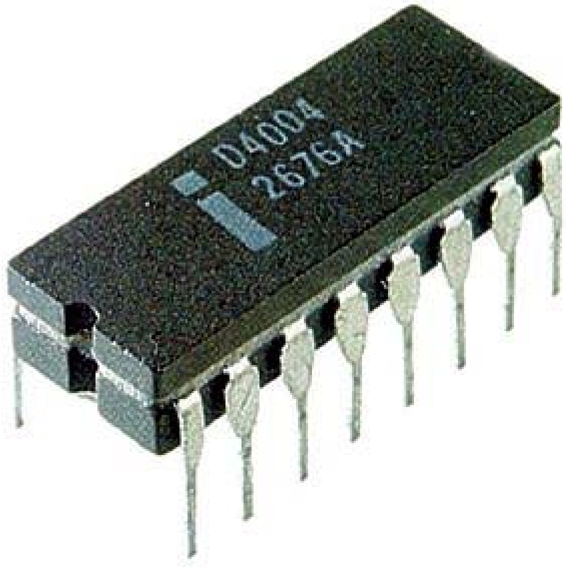
Time of events

- 1971. Intel released the Intel 4004, the world's first commercially available microprocessor.
- 1977. First VAX sold, a VAX-11/780.
- 1978. Intel introduces the Intel 8086 and Intel 8088, the first x86 chips.
- 1981. Stanford MIPS introduced, one of the first RISC designs.
- 1982. Intel introduces the Intel 80286, which was the first Intel processor that could run all the software written for its predecessors, the 8086 and 8088.
- 1985. Intel introduces the Intel 80386, which adds a 32-bit instruction set to the x86 microarchitecture.
- 1989. Intel introduces the Intel 80486
- 1993. Intel launches the original Pentium microprocessor, the first processor with a x86 superscalar microacrhitecture.
- 1995. Intel introduces the Pentium Pro which becomes the foundation for the Pentium II, Pentium III, Pentium M, and Intel Core Architectures.
- 2000. AMD announced x86-64 extension to the x86 microarchitecture.
- 2000. Analog Devices introduces the Blackfin architecture.
- 2000. Intel introduces Netburst which would become the foundation of all Pentium 4 variants and later be phased out in favor of the Core architecture.
- 2002. Intel releases a Pentium 4 with Hyper-Threading, the first modern desktop processor to implement simultaneous multithreading (SMT).
- 2003. Intel introduced the Pentium M, a low power mobile derivative of the Pentium Pro architecture.
- 2005. AMD announced Athlon 64 X2, the first x86 dual-core processor.
- 2006. Intel introduces the Core line of CPUs based on a modified Pentium M design.
- 2008. About ten billion CPUs were manufactured in 2008.

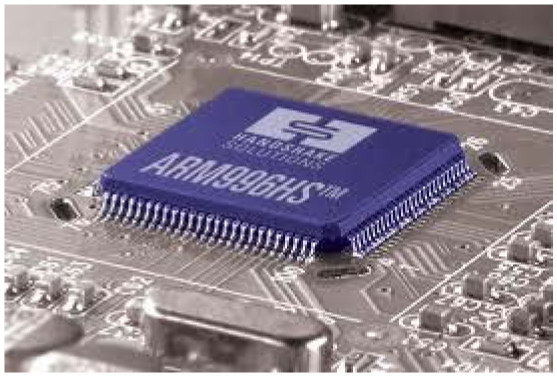
Chapter 3

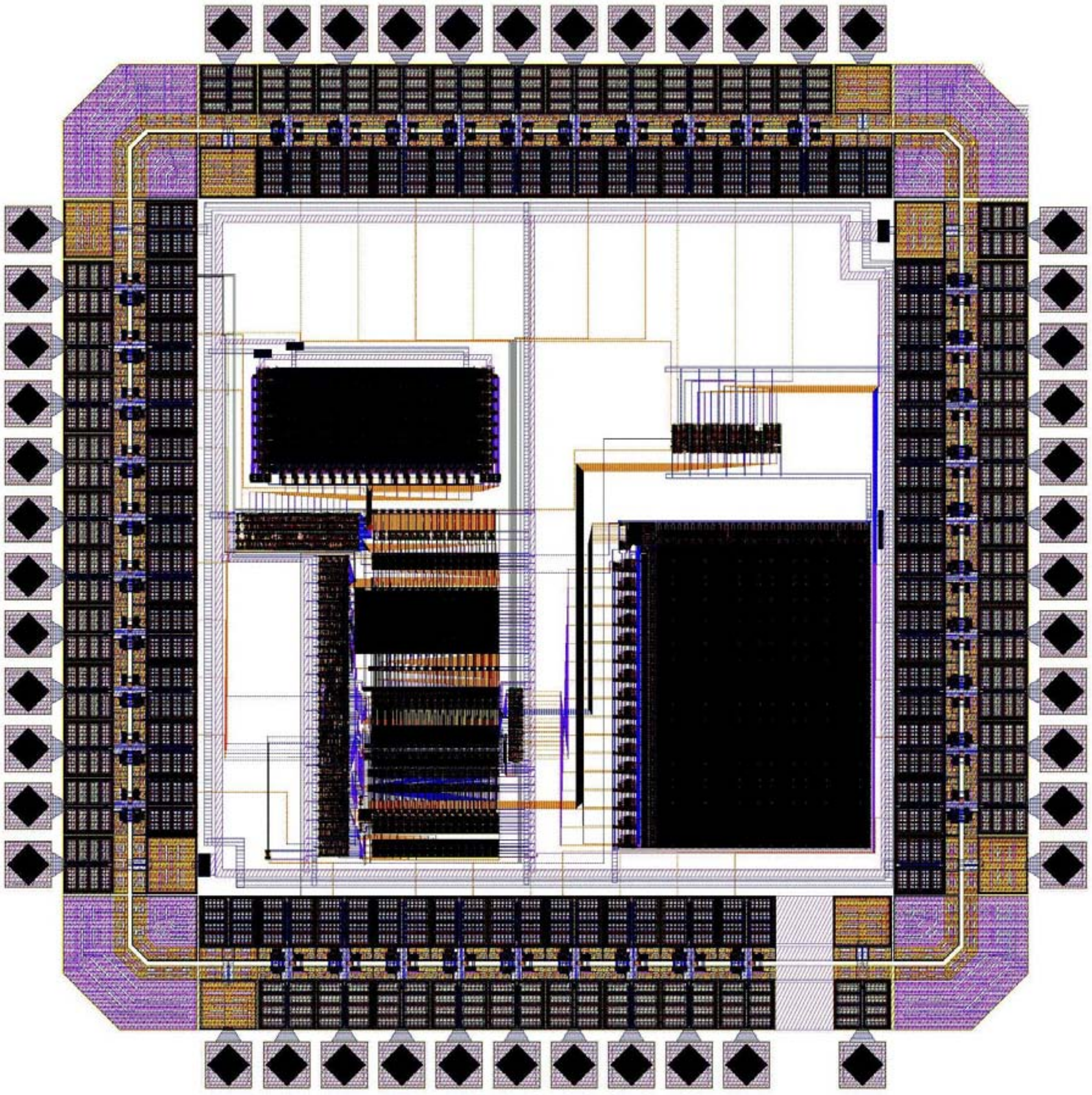
Microprocessor and CPU Design

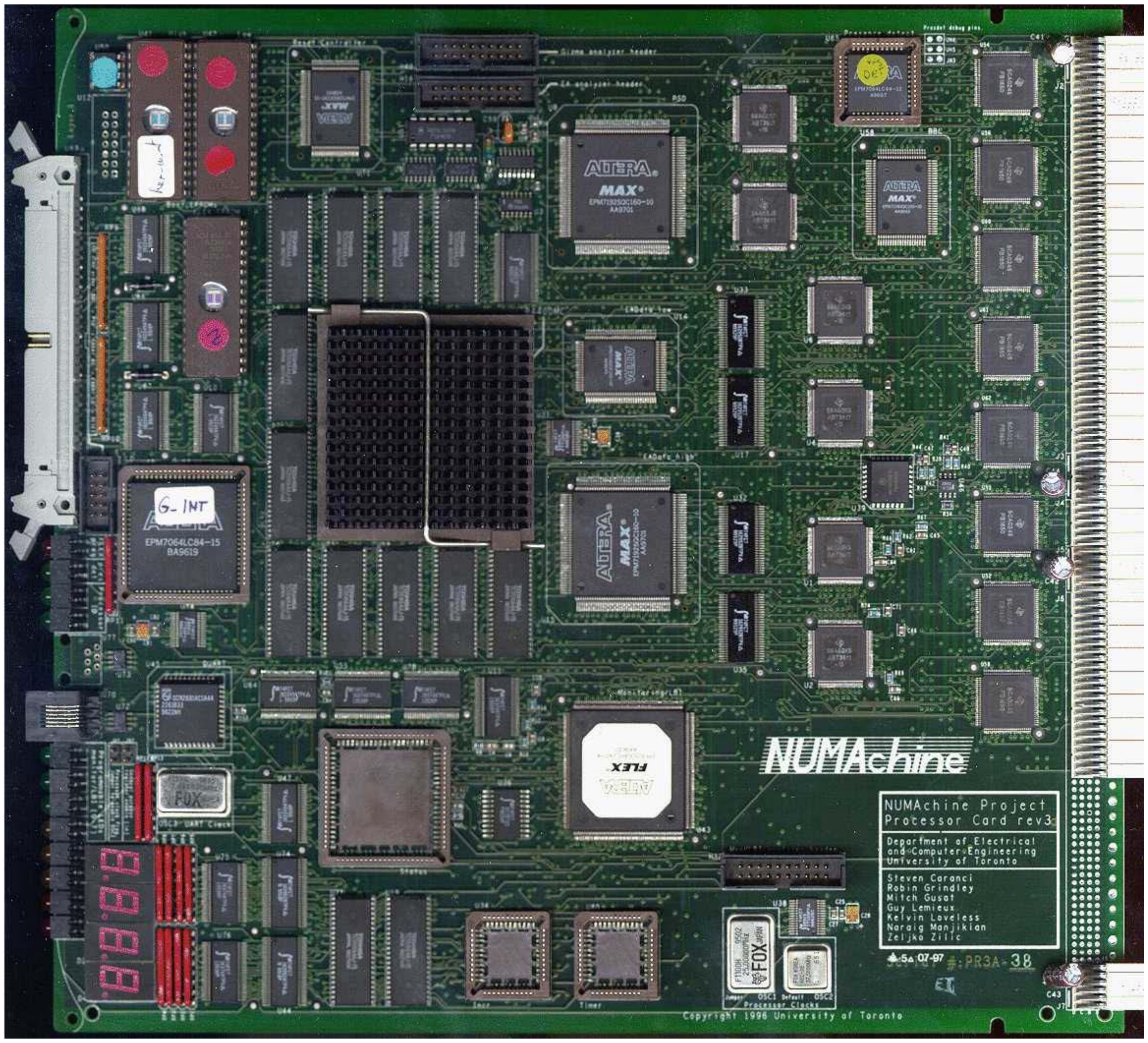
Microprocessor



Intel 4004, the first general-purpose, commercial microprocessor







A **microprocessor** incorporates most or all of the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC, or microchip). The first microprocessors emerged in the early 1970s and were used for electronic calculators, using binary-coded decimal (BCD) arithmetic on 4-bit words. Other embedded uses of 4-bit and 8-bit microprocessors, such as terminals, printers, various kinds of automation etc., followed soon after. Affordable 8-bit microprocessors with 16-bit addressing also led to the first general-purpose microcomputers from the mid-1970s on.

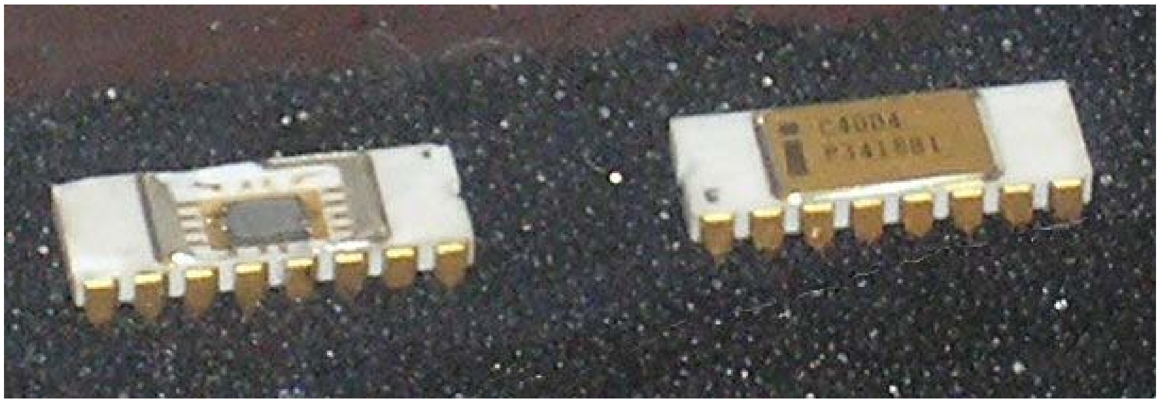
During the 1960s, computer processors were often constructed out of small and medium-scale ICs containing from tens to a few hundred transistors. The integration of a whole CPU onto a single chip greatly reduced the cost of processing power. From these humble beginnings, continued increases in microprocessor capacity have rendered other forms of computers almost completely obsolete, with one or more microprocessors used in everything from the smallest embedded systems and handheld devices to the largest mainframes and supercomputers.

Since the early 1970s, the increase in capacity of microprocessors has been a consequence of Moore's Law, which suggests that the number of transistors that can be fitted onto a chip doubles every two years. Although originally calculated as a doubling every year, Moore later refined the period to two years. It is often incorrectly quoted as a doubling of transistors every 18 months.

Firsts

Three projects delivered a microprocessor at about the same time: Intel's 4004, Texas Instruments (TI) TMS 1000, and Garrett AiResearch's Central Air Data Computer (CADC).

Intel 4004



The 4004 with cover removed (left) and as actually used (right)

The Intel 4004 is generally regarded as the first microprocessor, and cost thousands of dollars. The first known advertisement for the 4004 is dated November 1971 and appeared in Electronic News. The project that produced the 4004 originated in 1969, when Busicom, a Japanese calculator manufacturer, asked Intel to build a chipset for high-performance desktop calculators. Busicom's original design called for a programmable chip set consisting of seven different chips. Three of the chips were to make a special-purpose CPU with its program stored in ROM and its data stored in shift register read-write memory. Ted Hoff, the Intel engineer assigned to evaluate the project, believed the Busicom design could be simplified by using dynamic RAM storage for data, rather than shift register memory, and a more traditional general-purpose CPU architecture. Hoff came up with a four–chip architectural proposal: a ROM chip for storing the programs, a dynamic RAM chip for storing data, a simple I/O device and a 4-bit central processing unit (CPU). Although not a chip designer, he felt the CPU could be integrated into a single chip. This chip would later be called the 4004 microprocessor.

The architecture and specifications of the 4004 came from the interaction of Hoff with Stanley Mazor, a software engineer reporting to him, and with Busicom engineer Masatoshi Shima, during 1969. In April 1970, Intel hired Federico Faggin to lead the design of the four-chip set. Faggin, who originally developed the silicon gate technology

(SGT) in 1968 at Fairchild Semiconductor and designed the world’s first commercial integrated circuit using SGT, the Fairchild 3708, had the correct background to lead the project since it was SGT that made it possible to implement a single-chip CPU with the proper speed, power dissipation and cost. Faggin also developed the new methodology for random logic design, based on silicon gate, that made the 4004 possible. Production units of the 4004 were first delivered to Busicom in March 1971 and shipped to other customers in late 1971.

TMS 1000

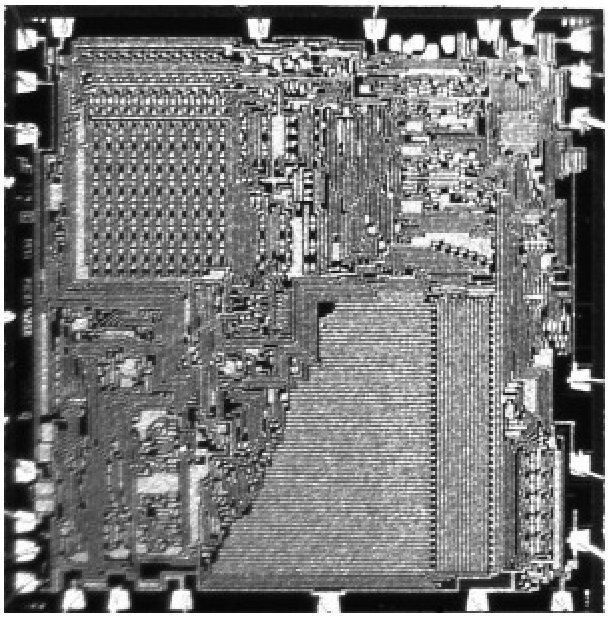
The Smithsonian Institution says TI engineers Gary Boone and Michael Cochran succeeded in creating the first microcontroller (also called a microcomputer) in 1971. The result of their work was the TMS 1000, which went commercial in 1974.

TI developed the 4-bit TMS 1000 and stressed pre-programmed embedded applications, introducing a version called the TMS1802NC on September 17, 1971 which implemented a calculator on a chip.

TI filed for the patent on the microprocessor. Gary Boone was awarded U.S. Patent 3,757,306 for the single-chip microprocessor architecture on September 4, 1973. It may never be known which company actually had the first working microprocessor running on the lab bench. In both 1971 and 1976, Intel and TI entered into broad patent cross-licensing agreements, with Intel paying royalties to TI for the microprocessor patent. A nice history of these events is contained in court documentation from a legal dispute between Cyrix and Intel, with TI as intervenor and owner of the microprocessor patent.

A computer-on-a-chip is a variation of a microprocessor that combines the microprocessor core (CPU), some program memory and read/write memory, and I/O (input/output) lines onto one chip. The computer-on-a-chip patent, called the "microcomputer patent" at the time, U.S. Patent 4,074,351, was awarded to Gary Boone and Michael J. Cochran of TI. Aside from this patent, the standard meaning of microcomputer is a computer using one or more microprocessors as its CPU(s), while the concept defined in the patent is more akin to a microcontroller.

Pico/General Instrument



The PICO1/GI250 chip introduced in 1971. This was designed by Pico Electronics (Glenrothes, Scotland) and manufactured by General Instrument of Hicksville NY

In 1971 Pico Electronics and General Instrument (GI) introduced their first collaboration in ICs, a complete single chip calculator IC for the Monroe/Litton Royal Digital III calculator. This chip could also arguably lay claim to be one of the first microprocessors or microcontrollers having ROM, RAM and a RISC instruction set on-chip. The layout for the four layers of the PMOS process was hand drawn at x500 scale on mylar film, a significant task at the time given the complexity of the chip.

Pico was a spinout by five GI design engineers whose vision was to create single chip calculator ICs. They had significant previous design experience on multiple calculator chipsets with both GI and Marconi-Elliott. The key team members had originally been tasked by Elliott Automation to create an 8 bit computer in MOS and had helped establish a MOS Research Laboratory in Glenrothes, Scotland in 1967.

Calculators were becoming the largest single market for semiconductors and Pico and GI went on to have significant success in this burgeoning market. GI continued to innovate in microprocessors and microcontrollers with products including the PIC1600, PIC1640 and PIC1650. In 1987 the GI Microelectronics business was spun out into the very successful PIC microcontroller business.

CADC

In 1968, Garrett AiResearch (which employed designers Ray Holt and Steve Geller) was invited to produce a digital computer to compete with electromechanical systems then under development for the main flight control computer in the US Navy's new F-14

Tomcat fighter. The design was complete by 1970, and used a MOS-based chipset as the core CPU. The design was significantly (approximately 20 times) smaller and much more reliable than the mechanical systems it competed against, and was used in all of the early Tomcat models. This system contained "a 20-bit, pipelined, parallel multi-microprocessor". The Navy refused to allow publication of the design until 1997. For this reason the CADC, and the MP944 chipset it used, are fairly unknown. Ray Holt graduated California Polytechnic University in 1968, and began his computer design career with the CADC. From its inception, it was shrouded in secrecy until 1998 when at Holt's request, the US Navy allowed the documents into the public domain. Since then several have debated if this was the first microprocessor. Holt has stated that no one has compared this microprocessor with those that came later. According to Parab et al. (2007), "The scientific papers and literature published around 1971 reveal that the MP944 digital processor used for the F-14 Tomcat aircraft of the US Navy qualifies as the first microprocessor. Although interesting, it was not a single-chip processor, and was not general purpose – it was more like a set of parallel building blocks you could use to make a special-purpose DSP form. It indicates that today's industry theme of converging DSP-microcontroller architectures was started in 1971." This convergence of DSP and microcontroller architectures is known as a Digital Signal Controller.

Gilbert Hyatt

Gilbert Hyatt was awarded a patent claiming an invention pre-dating both TI and Intel, describing a "microcontroller". The patent was later invalidated, but not before substantial royalties were paid out.

Four-Phase Systems AL1

The Four-Phase Systems AL1 was an 8-bit bit slice chip containing eight registers and an ALU. It was designed by Lee Boysel in 1969. At the time, it formed part of a nine-chip, 24-bit CPU with three AL1s, but it was later called a microprocessor when, in response to 1990s litigation by Texas Instruments, a demonstration system was constructed where a single AL1 formed part of a courtroom demonstration computer system, together with RAM, ROM, and an input-output device.

8-bit designs

The Intel 4004 was followed in 1972 by the Intel 8008, the world's first 8-bit microprocessor. According to A History of Modern Computing, (MIT Press), pp. 220–21, Intel entered into a contract with Computer Terminals Corporation, later called Datapoint, of San Antonio TX, for a chip for a terminal they were designing. Datapoint later decided not to use the chip, and Intel marketed it as the 8008 in April, 1972. This was the world's first 8-bit microprocessor. It was the basis for the famous "Mark-8" computer kit advertised in the magazine Radio-Electronics in 1974.

The 8008 was the precursor to the very successful Intel 8080 (1974), Zilog Z80 (1976), and derivative Intel 8-bit processors. The competing Motorola 6800 was released August

1974 and the similar MOS Technology 6502 in 1975 (designed largely by the same people). The 6502 rivaled the Z80 in popularity during the 1980s.

A low overall cost, small packaging, simple computer bus requirements, and sometimes the integration of extra circuitry (e.g. the Z80's built-in memory refresh circuitry) allowed the home computer "revolution" to accelerate sharply in the early 1980s. This delivered such inexpensive machines as the Sinclair ZX-81, which sold for US$99.

The Western Design Center, Inc. (WDC) introduced the CMOS 65C02 in 1982 and licensed the design to several firms. It was used as the CPU in the Apple IIe and IIc personal computers as well as in medical implantable grade pacemakers and defibrilators, automotive, industrial and consumer devices. WDC pioneered the licensing of microprocessor designs, later followed by ARM and other microprocessor Intellectual Property (IP) providers in the 1990s.

Motorola introduced the MC6809 in 1978, an ambitious and thought-through 8-bit design source compatible with the 6800 and implemented using purely hard-wired logic. (Subsequent 16-bit microprocessors typically used microcode to some extent, as CISC design requirements were getting too complex for purely hard-wired logic only.)

Another early 8-bit microprocessor was the Signetics 2650, which enjoyed a brief surge of interest due to its innovative and powerful instruction set architecture.

A seminal microprocessor in the world of spaceflight was RCA's RCA 1802 (aka CDP1802, RCA COSMAC) (introduced in 1976), which was used onboard the Galileo probe to Jupiter (launched 1989, arrived 1995). RCA COSMAC was the first to implement CMOS technology. The CDP1802 was used because it could be run at very low power, and because a variant was available fabricated using a special production process (Silicon on Sapphire), providing much better protection against cosmic radiation and electrostatic discharges than that of any other processor of the era. Thus, the SOS version of the 1802 was said to be the first radiation-hardened microprocessor.

The RCA 1802 had what is called a static design, meaning that the clock frequency could be made arbitrarily low, even to 0 Hz, a total stop condition. This let the Galileo spacecraft use minimum electric power for long uneventful stretches of a voyage. Timers and/or sensors would awaken/improve the performance of the processor in time for important tasks, such as navigation updates, attitude control, data acquisition, and radio communication.

12-bit designs

The Intersil 6100 family consisted of a 12-bit microprocessor (the 6100) and a range of peripheral support and memory ICs. The microprocessor recognised the DEC PDP-8 minicomputer instruction set. As such it was sometimes referred to as the **CMOS-PDP8**. Since it was also produced by Harris Corporation, it was also known as the **Harris HM-**

6100. By virtue of its CMOS technology and associated benefits, the 6100 was being incorporated into some military designs until the early 1980s.

16-bit designs

The first multi-chip 16-bit microprocessor was the National Semiconductor IMP-16, introduced in early 1973. An 8-bit version of the chipset was introduced in 1974 as the IMP-8.

Other early multi-chip 16-bit microprocessors include one used by Digital Equipment Corporation (DEC) in the LSI-11 OEM board set and the packaged PDP 11/03 minicomputer, and the Fairchild Semiconductor MicroFlame 9440, both of which were introduced in the 1975 to 1976 timeframe.

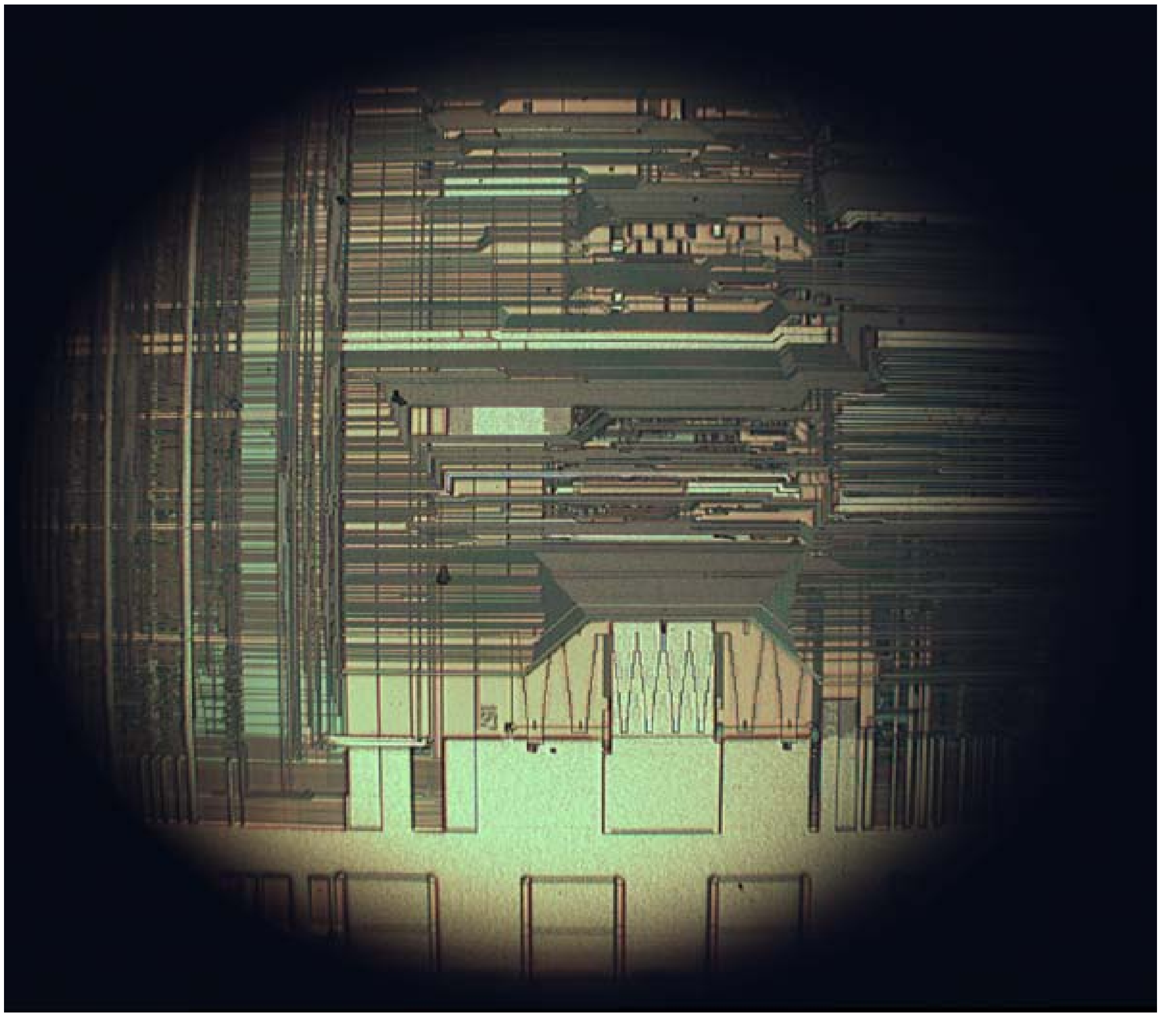
In 1975, National introduced the first 16-bit single-chip microprocessor, the National Semiconductor PACE, which was later followed by an NMOS version, the INS8900.

Another early single-chip 16-bit microprocessor was TI's TMS 9900, which was also compatible with their TI-990 line of minicomputers. The 9900 was used in the TI 990/4 minicomputer, the TI-99/4A home computer, and the TM990 line of OEM microcomputer boards. The chip was packaged in a large ceramic 64-pin DIP package, while most 8-bit microprocessors such as the Intel 8080 used the more common, smaller, and less expensive plastic 40-pin DIP. A follow-on chip, the TMS 9980, was designed to compete with the Intel 8080, had the full TI 990 16-bit instruction set, used a plastic 40-pin package, moved data 8 bits at a time, but could only address 16 KB. A third chip, the TMS 9995, was a new design. The family later expanded to include the 99105 and 99110.

The Western Design Center, Inc. (WDC) introduced the CMOS 65816 16-bit upgrade of the WDC CMOS 65C02 in 1984. The 65816 16-bit microprocessor was the core of the Apple IIgs and later the Super Nintendo Entertainment System, making it one of the most popular 16-bit designs of all time.

Intel followed a different path, having no minicomputers to emulate, and instead "upsized" their 8080 design into the 16-bit Intel 8086, the first member of the x86 family, which powers most modern PC type computers. Intel introduced the 8086 as a cost effective way of porting software from the 8080 lines, and succeeded in winning much business on that premise. The 8088, a version of the 8086 that used an external 8-bit data bus, was the microprocessor in the first IBM PC, the model 5150. Following up their 8086 and 8088, Intel released the 80186, 80286 and, in 1985, the 32-bit 80386, cementing their PC market dominance with the processor family's backwards compatibility. The 8086 and 80186 had a crude method of segmentation, while the 80286 introduced a full-featured semgented memory management unit (MMU), and the 80386 introduced a flat 32-bit memory model with paged memory management.

32-bit designs



Upper interconnect layers on an Intel 80486DX2 die

16-bit designs had only been on the market briefly when 32-bit implementations started to appear.

The most significant of the 32-bit designs is the MC68000, introduced in 1979. The 68K, as it was widely known, had 32-bit registers but used 16-bit internal data paths and a 16-bit external data bus to reduce pin count, and supported only 24-bit addresses. Motorola generally described it as a 16-bit processor, though it clearly has 32-bit architecture. The combination of high performance, large (16 megabytes or 2^{24} bytes) memory space and fairly low cost made it the most popular CPU design of its class. The Apple Lisa and Macintosh designs made use of the 68000, as did a host of other designs in the mid-1980s, including the Atari ST and Commodore Amiga.

The world's first single-chip fully-32-bit microprocessor, with 32-bit data paths, 32-bit buses, and 32-bit addresses, was the AT&T Bell Labs BELLMAC-32A, with first samples in 1980, and general production in 1982 After the divestiture of AT&T in 1984, it was renamed the WE 32000 (WE for Western Electric), and had two follow-on generations, the WE 32100 and WE 32200. These microprocessors were used in the AT&T 3B5 and 3B15 minicomputers; in the 3B2, the world's first desktop supermicrocomputer; in the "Companion", the world's first 32-bit laptop computer; and in "Alexander", the world's first book-sized supermicrocomputer, featuring ROM-pack memory cartridges similar to today's gaming consoles. All these systems ran the UNIX System V operating system.

Intel's first 32-bit microprocessor was the iAPX 432, which was introduced in 1981 but was not a commercial success. It had an advanced capability-based object-oriented architecture, but poor performance compared to contemporary architectures such as Intel's own 80286 (introduced 1982), which was almost four times as fast on typical benchmark tests. However, the results for the iAPX432 was partly due to a rushed and therefore suboptimal Ada compiler.

The ARM first appeared in 1985. This is a RISC processor design, which has since come to dominate the 32-bit embedded systems processor space due in large part to its power efficiency, its licensing model, and its wide selection of system development tools. Semiconductor manufacturers generally license cores such as the ARM11 and integrate them into their own system on a chip products; only a few such vendors are licensed to modify the ARM cores. Most cell phones include an ARM processor, as do a wide variety of other products. There are microcontroller-oriented ARM cores without virtual memory support, as well as SMP applications processors with virtual memory.

Motorola's success with the 68000 led to the MC68010, which added virtual memory support. The MC68020, introduced in 1985 added full 32-bit data and address busses. The 68020 became hugely popular in the Unix supermicrocomputer market, and many small companies (e.g., Altos, Charles River Data Systems) produced desktop-size systems. The MC68030 was introduced next, improving upon the previous design by integrating the MMU into the chip. The continued success led to the MC68040, which included an FPU for better math performance. A 68050 failed to achieve its performance goals and was not released, and the follow-up MC68060 was released into a market saturated by much faster RISC designs. The 68K family faded from the desktop in the early 1990s.

Other large companies designed the 68020 and follow-ons into embedded equipment. At one point, there were more 68020s in embedded equipment than there were Intel Pentiums in PCs. The ColdFire processor cores are derivatives of the venerable 68020.

During this time (early to mid-1980s), National Semiconductor introduced a very similar 16-bit pinout, 32-bit internal microprocessor called the NS 16032 (later renamed 32016), the full 32-bit version named the NS 32032. Later the NS 32132 was introduced which allowed two CPUs to reside on the same memory bus, with built in arbitration. The NS32016/32 outperformed the MC68000/10 but the NS32332 which arrived at approximately the same time the MC68020 did not have enough performance. The third generation chip, the NS32532 was different. It had about double the performance of the

MC68030 which was released around the same time. The appearance of RISC processors like the AM29000 and MC88000 (now both dead) influenced the architecture of the final core, the NS32764. Technically advanced, using a superscalar RISC core, internally overclocked, with a 64 bit bus, it was still capable of executing Series 32000 instructions through real time translation.

When National Semiconductor decided to leave the Unix market, the chip was redesigned into the Swordfish Embedded processor with a set of on chip peripherals. The chip turned out to be too expensive for the laser printer market and was killed. The design team went to Intel and there designed the Pentium processor which is very similar to the NS32764 core internally The big success of the Series 32000 was in the laser printer market, where the NS32CG16 with microcoded BitBlt instructions had very good price/performance and was adopted by large companies like Canon. By the mid-1980s, Sequent introduced the first symmetric multiprocessor (SMP) server-class computer using the NS 32032. This was one of the design's few wins, and it disappeared in the late 1980s. The MIPS R2000 (1984) and R3000 (1989) were highly successful 32-bit RISC microprocessors. They were used in high-end workstations and servers by SGI, among others. Other designs included the interesting Zilog Z80000, which arrived too late to market to stand a chance and disappeared quickly.

In the late 1980s, "microprocessor wars" started killing off some of the microprocessors. Apparently, with only one major design win, Sequent, the NS 32032 just faded out of existence, and Sequent switched to Intel microprocessors.

From 1985 to 2003, the 32-bit x86 architectures became increasingly dominant in desktop, laptop, and server markets, and these microprocessors became faster and more capable. Intel had licensed early versions of the architecture to other companies, but declined to license the Pentium, so AMD and Cyrix built later versions of the architecture based on their own designs. During this span, these processors increased in complexity (transistor count) and capability (instructions/second) by at least three orders of magnitude. Intel's Pentium line is probably the most famous and recognizable 32-bit processor model, at least with the public at large.

64-bit designs in personal computers

While 64-bit microprocessor designs have been in use in several markets since the early 1990s, the early 2000s saw the introduction of 64-bit microprocessors targeted at the PC market.

With AMD's introduction of a 64-bit architecture backwards-compatible with x86, x86-64 (also called **AMD64**), in September 2003, followed by Intel's near fully compatible 64-bit extensions (first called IA-32e or EM64T, later renamed **Intel 64**), the 64-bit desktop era began. Both versions can run 32-bit legacy applications without any performance penalty as well as new 64-bit software. With operating systems Windows XP x64, Windows Vista x64, Windows 7 x64, Linux, BSD and Mac OS X that run 64-bit native, the software is also geared to fully utilize the capabilities of such processors. The move to 64 bits is more than just an increase in register size from the IA-32 as it also doubles the number of general-purpose registers.

The move to 64 bits by PowerPC processors had been intended since the processors' design in the early 90s and was not a major cause of incompatibility. Existing integer registers are extended as are all related data pathways, but, as was the case with IA-32, both floating point and vector units had been operating at or above 64 bits for several years. Unlike what happened when IA-32 was extended to x86-64, no new general purpose registers were added in 64-bit PowerPC, so any performance gained when using the 64-bit mode for applications making no use of the larger address space is minimal.

Multi-core processor

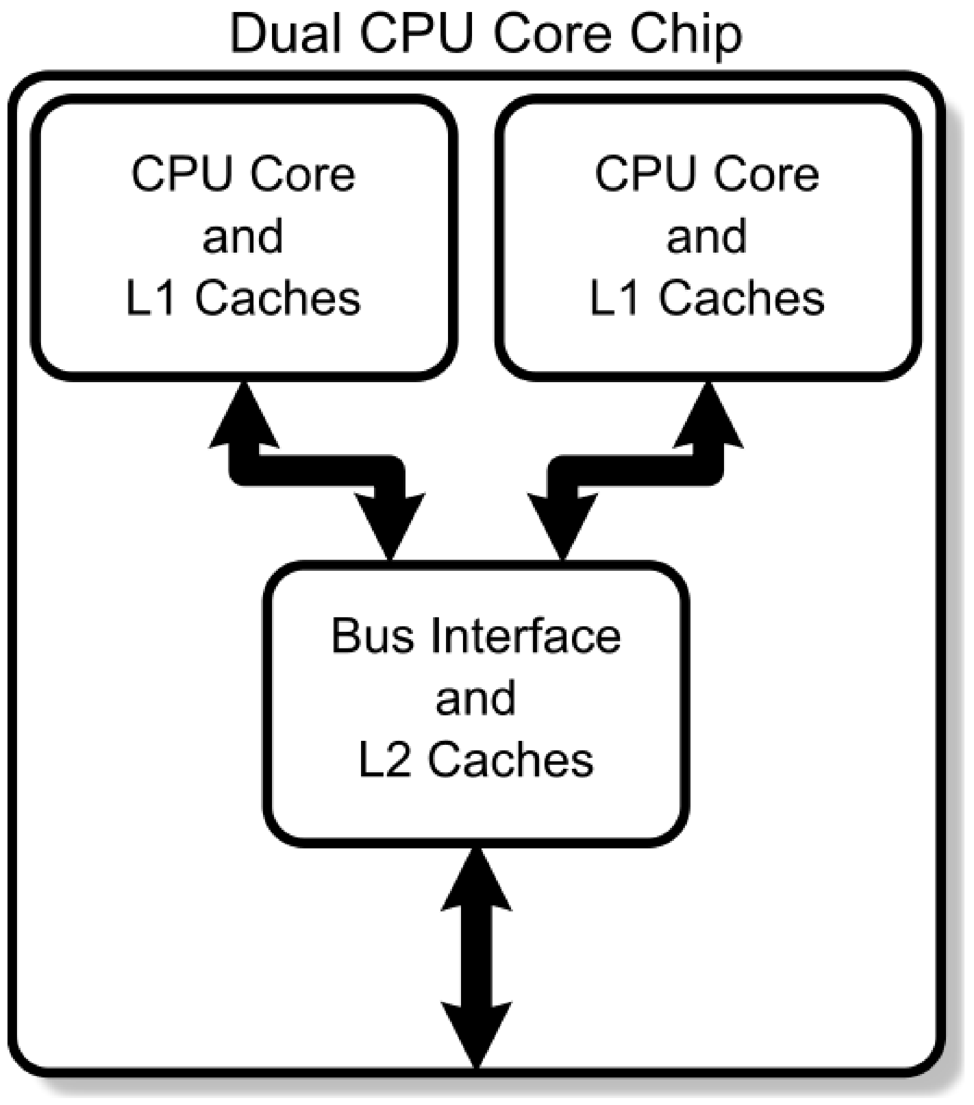
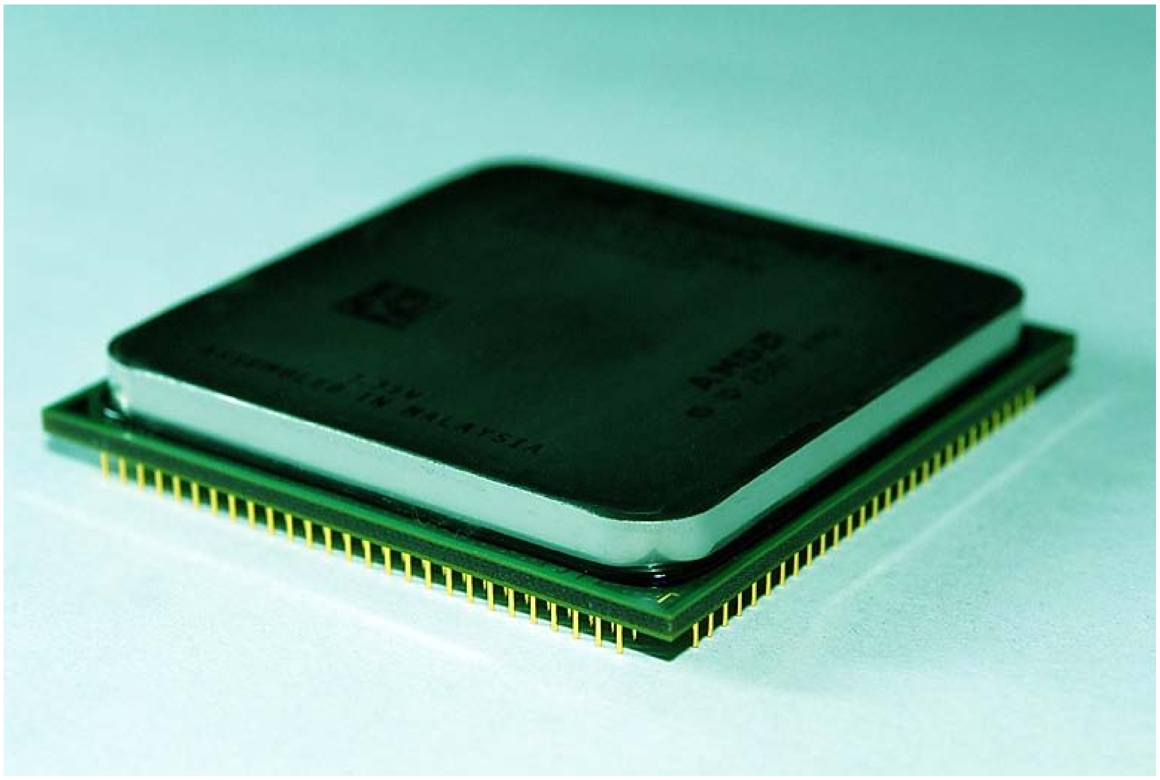


Diagram of a generic dual-core processor, with CPU-local level 1 caches, and a shared, on-die level 2 cache.

An Intel Core 2 Duo E6750 dual-core processor



An AMD Athlon X2 6400+ dual-core processor

In computing, a **processor** is the unit that reads and executes program instructions, which are fixed-length (typically 32 or 64 bit) or variable-length chunks of data. The data in the instruction tells the processor what to do. The instructions are very basic things like reading data from memory or sending data to the user display, but they are processed so rapidly that we experience the results as the smooth operation of a program.

Processors were originally developed with only one **core**. The core is the part of the processor that actually performs the reading and executing of instructions. Single-core processors can process only one instruction at a time. (To improve efficiency, processors commonly utilize pipelines internally, which allow several instructions to be processed together; however, they are still consumed into the pipeline one at a time.)

A **multi-core processor** is composed of two or more independent cores. One can describe it as an integrated circuit which has two or more individual processors (called cores in this sense). Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package. A **many-core** processor is one in which the number of cores is large enough that traditional multi-processor techniques are no longer efficient — largely due to issues with congestion supplying sufficient instructions and data to the many processors. This threshold is roughly in the range of several tens of cores and probably requires a network on chip.

A **dual-core processor** contains two cores (Such as AMD Phenom II X2 or Intel Core Duo), a **quad-core processor** contains four cores (Such as the AMD Phenom II X4 and Intel 2010 core line, which includes 3 levels of quad core processors), and a **hexa-core processor** contains six cores (Such as the AMD Phenom II X6 or Intel Core i7 Extreme Edition 980X). A multi-core processor implements multiprocessing in a single physical package. Designers may couple cores in a multi-core device tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared memory inter-core communication methods. Common network topologies to interconnect cores include bus, ring, 2-dimensional mesh, and crossbar. Homogeneous multi-core systems include only identical cores, unlike heterogeneous multi-core systems. Just as with single-processor systems, cores in multi-core systems may implement architectures like superscalar, VLIW, vector processing, SIMD, or multithreading.

Multi-core processors are widely used across many application domains including general-purpose, embedded, network, digital signal processing (DSP), and graphics.

The amount of performance gained by the use of a multi-core processor depends very much on the software algorithms and implementation. In particular, the possible gains are limited by the fraction of the software that can be parallelized to run on multiple cores simultaneously; this effect is described by Amdahl's law. In the best case, so-called embarrassingly parallel problems may realize speedup factors near the number of cores, or beyond even that if the problem is split up enough to fit within each processor's or core's cache(s) due to the fact that the much slower main memory system is avoided.

Many typical applications, however, do not realize such large speedup factors. The parallelization of software is a significant on-going topic of research.

Terminology

The terms multi-core and dual-core most commonly refer to some sort of central processing unit (CPU), but are sometimes also applied to digital signal processors (DSP) and system-on-a-chip (SoC). Additionally, some use these terms to refer only to multi-core microprocessors that are manufactured on the same integrated circuit die. These people generally refer to separate microprocessor dies in the same package by another name, such as multi-chip module. Here we uses both the terms "multi-core" and "dual-core" to reference microelectronic CPUs manufactured on the same integrated circuit, unless otherwise noted.

In contrast to multi-core systems, the term multi-CPU refers to multiple physically separate processing-units (which often contain special circuitry to facilitate communication between each other).

The terms many-core and massively multi-core sometimes occur to describe multi-core architectures with an especially high number of cores (tens or hundreds).

Some systems use many soft microprocessor cores placed on a single FPGA. Each "core" can be considered a "semiconductor intellectual property core" as well as a CPU core.

Development

While manufacturing technology improves, reducing the size of single gates, physical limits of semiconductor-based microelectronics have become a major design concern. Some effects of these physical limitations can cause significant heat dissipation and data synchronization problems. The demand for more-capable microprocessors encourages CPU designers to use various methods with a view to increasing performance. Some instruction-level parallelism (ILP) methods like superscalar pipelining are suitable for many applications, but are inefficient for others that tend to contain difficult-to-predict code. Many applications are better suited to thread level parallelism (TLP) methods, and multiple independent CPUs is one common method used to increase a system's overall TLP. A combination of increased available space (due to refined manufacturing processes) and the demand for increased TLP led to the development of multi-core CPUs.

Commercial incentives

Several business motives drive the development of dual-core architectures. For decades, it was possible to improve performance of a CPU by shrinking the area of the integrated circuit which drove down the cost of the silicon used per device. Alternatively, for the same circuit area, more transistors could be utilized in the design, which increased functionality, especially for CISC architectures. The CPUs could also be driven at faster clock rates, to the point where microwave-frequency circuit layout techniques had to be employed.

Eventually these techniques failed to improve the CPU performance. Multiple processors had to be employed to gain speed in computation. This led to the use of multiple cores on the same chip to improve performance, which could then lead to better sales of CPU chips which had dual cores, and more (multi-core). One manufacturer has produced a 48-core for research in processors used for cloud computing.

The terminology "dual-core" (and other multiples) lends itself to marketing differentiation from single-CPU processors.

Technical factors

Since computer manufacturers have long implemented symmetric multiprocessing (SMP) designs using discrete CPUs, the issues regarding implementing multi-core processor architecture and supporting it in software are well known.

Additionally:

- Utilizing a proven processing-core design without architectural changes reduces design risk significantly.
- For general-purpose processors, much of the motivation for multi-core processors comes from greatly diminished gains in processor performance from increasing the operating frequency. This is due to three primary factors:
    1. The memory wall; the increasing gap between processor and memory speeds. This effect pushes cache sizes larger in order to mask the latency of memory. This helps only to the extent that memory bandwidth is not the bottleneck in performance.
    2. The ILP wall; the increasing difficulty of finding enough parallelism in a single instructions stream to keep a high-performance single-core processor busy.
    3. The power wall; the trend of consuming exponentially increasing power with each factorial increase of operating frequency. This increase can be mitigated by "shrinking" the processor by using smaller traces for the same logic. The power wall poses manufacturing, system design and deployment problems that have not been justified in the face of the diminished gains in performance due to the memory wall and ILP wall.

In order to continue delivering regular performance improvements for general-purpose processors, manufacturers such as Intel and AMD have turned to multi-core designs, sacrificing lower manufacturing-costs for higher performance in some applications and systems. Multi-core architectures are being developed, but so are the alternatives. An especially strong contender for established markets is the further integration of peripheral functions into the chip.

Advantages

The proximity of multiple CPU cores on the same die allows the cache coherency circuitry to operate at a much higher clock-rate than is possible if the signals have to travel off-chip. Combining equivalent CPUs on a single die significantly improves the performance of cache snoop (alternative: Bus snooping) operations. Put simply, this means that signals between different CPUs travel shorter distances, and therefore those signals degrade less. These higher-quality signals allow more data to be sent in a given time period, since individual signals can be shorter and do not need to be repeated as often.

The largest boost in performance will likely be noticed in improved response-time while running CPU-intensive processes, like antivirus scans, ripping/burning media (requiring file conversion), or searching for folders. For example, if the automatic virus-scan runs while a movie is being watched, the application running the movie is far less likely to be starved of processor power, as the antivirus program will be assigned to a different processor core than the one running the movie playback.

Assuming that the die can fit into the package, physically, the multi-core CPU designs require much less printed circuit board (PCB) space than do multi-chip SMP designs. Also, a dual-core processor uses slightly less power than two coupled single-core processors, principally because of the decreased power required to drive signals external to the chip. Furthermore, the cores share some circuitry, like the L2 cache and the interface to the front side bus (FSB). In terms of competing technologies for the available silicon die area, multi-core design can make use of proven CPU core library designs and produce a product with lower risk of design error than devising a new wider core-design. Also, adding more cache suffers from diminishing returns.

Disadvantages

Maximizing the utilization of the computing resources provided by multi-core processors requires adjustments both to the operating system (OS) support and to existing application software. Also, the ability of multi-core processors to increase application performance depends on the use of multiple threads within applications. The situation is improving: for example the Valve Corporation's Source engine offers multi-core support, and Crytek has developed similar technologies for CryEngine 2, which powers their game, Crysis. Emergent Game Technologies' Gamebryo engine includes their Floodgate technology which simplifies multicore development across game platforms. In addition, Apple Inc.'s latest OS, Mac OS X Snow Leopard has a built-in multi-core facility called Grand Central Dispatch for Intel CPUs.

Integration of a multi-core chip drives chip production yields down and they are more difficult to manage thermally than lower-density single-chip designs. Intel has partially countered this first problem by creating its quad-core designs by combining two dual-core on a single die with a unified cache, hence any two working dual-core dies can be used, as opposed to producing four cores on a single die and requiring all four to work to produce a quad-core. From an architectural point of view, ultimately, single CPU designs may make better use of the silicon surface area than multiprocessing cores, so a development commitment to this architecture may carry the risk of obsolescence. Finally, raw processing power is not the only constraint on system performance. Two processing cores sharing the same system bus and memory bandwidth limits the real-world performance advantage. If a single core is close to being memory-bandwidth limited, going to dual-core might only give 30% to 70% improvement. If memory bandwidth is not a problem, a 90% improvement can be expected. It would be possible for an application that used two CPUs to end up running faster on one dual-core if communication between the CPUs was the limiting factor, which would count as more than 100% improvement.

Hardware

Trends

The general trend in processor development has moved from dual-, tri-, quad-, hexa-, octo-core chips to ones with tens or even hundreds of cores. In addition, multi-core chips mixed with simultaneous multithreading, memory-on-chip, and special-purpose "heterogeneous" cores promise further performance and efficiency gains, especially in processing multimedia, recognition and networking applications. There is also a trend of improving energy-efficiency by focusing on performance-per-watt with advanced fine-grain or ultra fine-grain power management and dynamic voltage and frequency scaling (i.e. laptop computers and portable media players).

Architecture

The composition and balance of the cores in multi-core architecture show great variety. Some architectures use one core design repeated consistently ("homogeneous"), while others use a mixture of different cores, each optimized for a different, "heterogeneous", role .

The article CPU designers debate multi-core future by Rick Merritt, EE Times 2008, includes comments:
"Chuck Moore [...] suggested computers should be more like cellphones, using a variety of specialty cores to run modular software scheduled by a high-level applications programming interface.
[...] Atsushi Hasegawa, a senior chief engineer at Renesas, generally agreed. He suggested the cellphone's use of many specialty cores working in concert is a good model for future multi-core designs.
[...] Anant Agarwal, founder and chief executive of startup Tilera, took the opposing view. He said multi-core chips need to be homogeneous collections of general-purpose cores to keep the software model simple."

Software impact

An outdated version of an anti-virus application may create a new thread for a scan process, while its GUI thread waits for commands from the user (e.g. cancel the scan). In such cases, a multicore architecture is of little benefit for the application itself due to the single thread doing all heavy lifting and the inability to balance the work evenly across multiple cores. Programming truly multithreaded code often requires complex co-ordination of threads and can easily introduce subtle and difficult-to-find bugs due to the interleaving of processing on data shared between threads (thread-safety). Consequently, such code is much more difficult to debug than single-threaded code when it breaks. There has been a perceived lack of motivation for writing consumer-level threaded applications because of the relative rarity of consumer-level multiprocessor hardware. Although threaded applications incur little additional performance penalty on single-processor machines, the extra overhead of development has been difficult to justify due to the preponderance of single-processor machines. Also, serial tasks like decoding the entropy encoding algorithms used in video codecs are impossible to parallelize because each result generated is used to help create the next result of the entropy decoding algorithm.

Given the increasing emphasis on multicore chip design, stemming from the grave thermal and power consumption problems posed by any further significant increase in processor clock speeds, the extent to which software can be multithreaded to take advantage of these new chips is likely to be the single greatest constraint on computer performance in the future. If developers are unable to design software to fully exploit the resources provided by multiple cores, then they will ultimately reach an insurmountable performance ceiling.

The telecommunications market had been one of the first that needed a new design of parallel datapath packet processing because there was a very quick adoption of these multiple-core processors for the datapath and the control plane. These MPUs are going to replace the traditional Network Processors that were based on proprietary micro- or pico-code.

Parallel programming techniques can benefit from multiple cores directly. Some existing parallel programming models such as Cilk++, OpenMP, FastFlow, Skandium, and MPI can be used on multi-core platforms. Intel introduced a new abstraction for C++ parallelism called TBB. Other research efforts include the Codeplay Sieve System, Cray's Chapel, Sun's Fortress, and IBM's X10.

Multi-core processing has also affected the ability of modern computational software development. Developers programming in newer languages might find that their modern languages do not support multi-core functionality. This then requires the use of numerical libraries to access code written in languages like C and Fortran, which perform math computations faster than newer languages like C#. Intel's MKL and AMD's ACML are written in these native languages and take advantage of multi-core processing.

Managing concurrency acquires a central role in developing parallel applications. The basic steps in designing parallel applications are:

Partitioning

The partitioning stage of a design is intended to expose opportunities for parallel execution. Hence, the focus is on defining a large number of small tasks in order to yield what is termed a fine-grained decomposition of a problem.

Communication

The tasks generated by a partition are intended to execute concurrently but cannot, in general, execute independently. The computation to be performed in one task will typically require data associated with another task. Data must then be transferred between tasks so as to allow computation to proceed. This information flow is specified in the communication phase of a design.

Agglomeration

In the third stage, development moves from the abstract toward the concrete. Developers revisit decisions made in the partitioning and communication phases with a view to obtaining an algorithm that will execute efficiently on some class of parallel computer. In particular, developers consider whether it is useful to combine, or agglomerate, tasks identified by the partitioning phase, so as to provide a smaller number of tasks, each of greater size. They also determine whether it is worthwhile to replicate data and/or computation.

Mapping

In the fourth and final stage of the design of parallel algorithms, the developers specify where each task is to execute. This mapping problem does not arise on uniprocessors or on shared-memory computers that provide automatic task scheduling.

On the other hand, on the server side, multicore processors are ideal because they allow many users to connect to a site simultaneously and have independent threads of execution. This allows for Web servers and application servers that have much better throughput.

Licensing

Typically, proprietary enterprise-server software is licensed "per processor". In the past a CPU was a processor and most computers had only one CPU, so there was no ambiguity.

Now there is the possibility of counting cores as processors and charging a customer for multiple licenses for a multi-core CPU. However, the trend seems to be counting dual-core chips as a single processor: Microsoft, Intel, and AMD support this view. Microsoft have said they would treat a socket as a single processor.

Oracle counts an AMD X2 or Intel dual-core CPU as a single processor but has other numbers for other types, especially for processors with more than two cores. IBM and HP count a multi-chip module as multiple processors. If multi-chip modules count as one processor, CPU makers have an incentive to make large expensive multi-chip modules so their customers save on software licensing. It seems that the industry is slowly heading towards counting each die as a processor, no matter how many cores each die has.

Embedded applications

Embedded computing operates in an area of processor technology distinct from that of "mainstream" PCs. The same technological drivers towards multicore apply here too. Indeed, in many cases the application is a "natural" fit for multicore technologies, if the task can easily be partitioned between the different processors.

In addition, embedded software is typically developed for a specific hardware release, making issues of software portability, legacy code or supporting independent developers less critical than is the case for PC or enterprise computing. As a result, it is easier for developers to adopt new technologies and as a result there is a greater variety of multicore processing architectures and suppliers.

As of 2010, multi-core network processing devices have become mainstream, with companies such as Freescale Semiconductor, Cavium Networks, Wintegra and Broadcom all manufacturing products with eight processors.

In digital signal processing the same trend applies: Texas Instruments has the three-core TMS320C6488 and four-core TMS320C5441, Freescale the four-core MSC8144 and six-core MSC8156 (and both have stated they are working on eight-core successors). Newer entries include the Storm-1 family from Stream Processors, Inc with 40 and 80 general purpose ALUs per chip, all programmable in C as a SIMD engine and Picochip with three-hundred processors on a single die, focused on communication applications.

RISC

In the mid-1980s to early-1990s, a crop of new high-performance Reduced Instruction Set Computer (RISC) microprocessors appeared, influenced by discrete RISC-like CPU designs such as the IBM 801 and others. RISC microprocessors were initially used in special-purpose machines and Unix workstations, but then gained wide acceptance in other roles.

In 1986, HP released its first system with a PA-RISC CPU. The first commercial RISC microprocessor design was released either by MIPS Computer Systems, the 32-bit R2000 (the R1000 was not released) or by Acorn computers, the 32-bit ARM2 in 1987. The R3000 made the design truly practical, and the R4000 introduced the world's first commercially available 64-bit RISC microprocessor. Competing projects would result in the IBM POWER and Sun SPARC architectures. Soon every major vendor was releasing a RISC design, including the AT&T CRISP, AMD 29000, Intel i860 and Intel i960, Motorola 88000, DEC Alpha.

As of 2007, two 64-bit RISC architectures are still produced in volume for non-embedded applications: SPARC and Power ISA.

Special-purpose designs

A microprocessor is a general purpose system. Several specialized processing devices have followed from the technology. Microcontrollers integrate a microprocessor with periphal devices for control of embedded system. A digital signal processors (DSP) is sepcialized for signal processing. Graphics processing units may have no, limited, or general programming facilities. For example, GPUs through the 1990s were mostly non-programmable and have only recently gained limited facilities like programmable vertex shaders.

Market statistics

In 2003, about $44 billion (USD) worth of microprocessors were manufactured and sold. Although about half of that money was spent on CPUs used in desktop or laptop personal computers, those count for only about 0.2% of all CPUs sold.

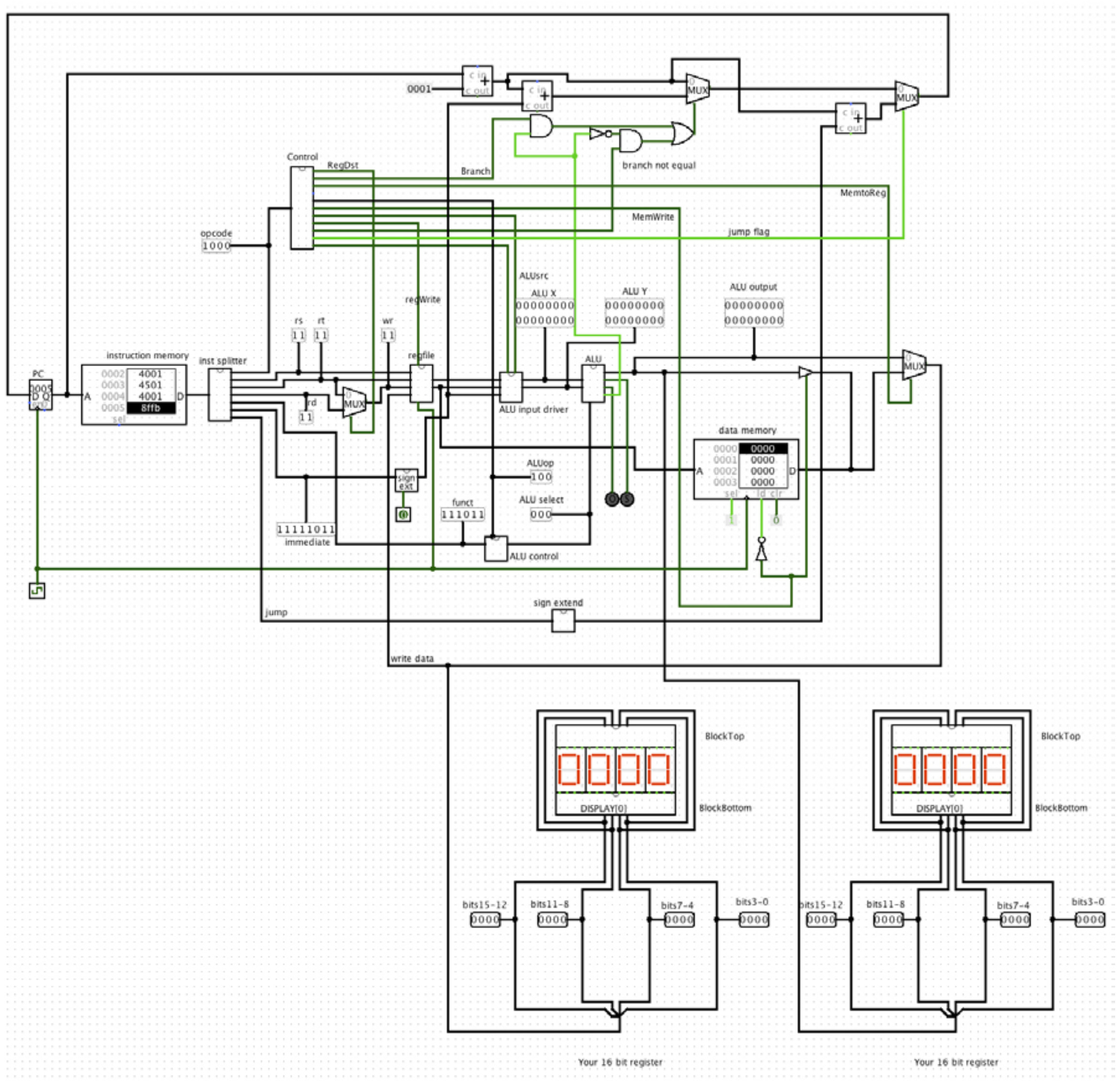
About 55% of all CPUs sold in the world are 8-bit microcontrollers, over two billion of which were sold in 1997.

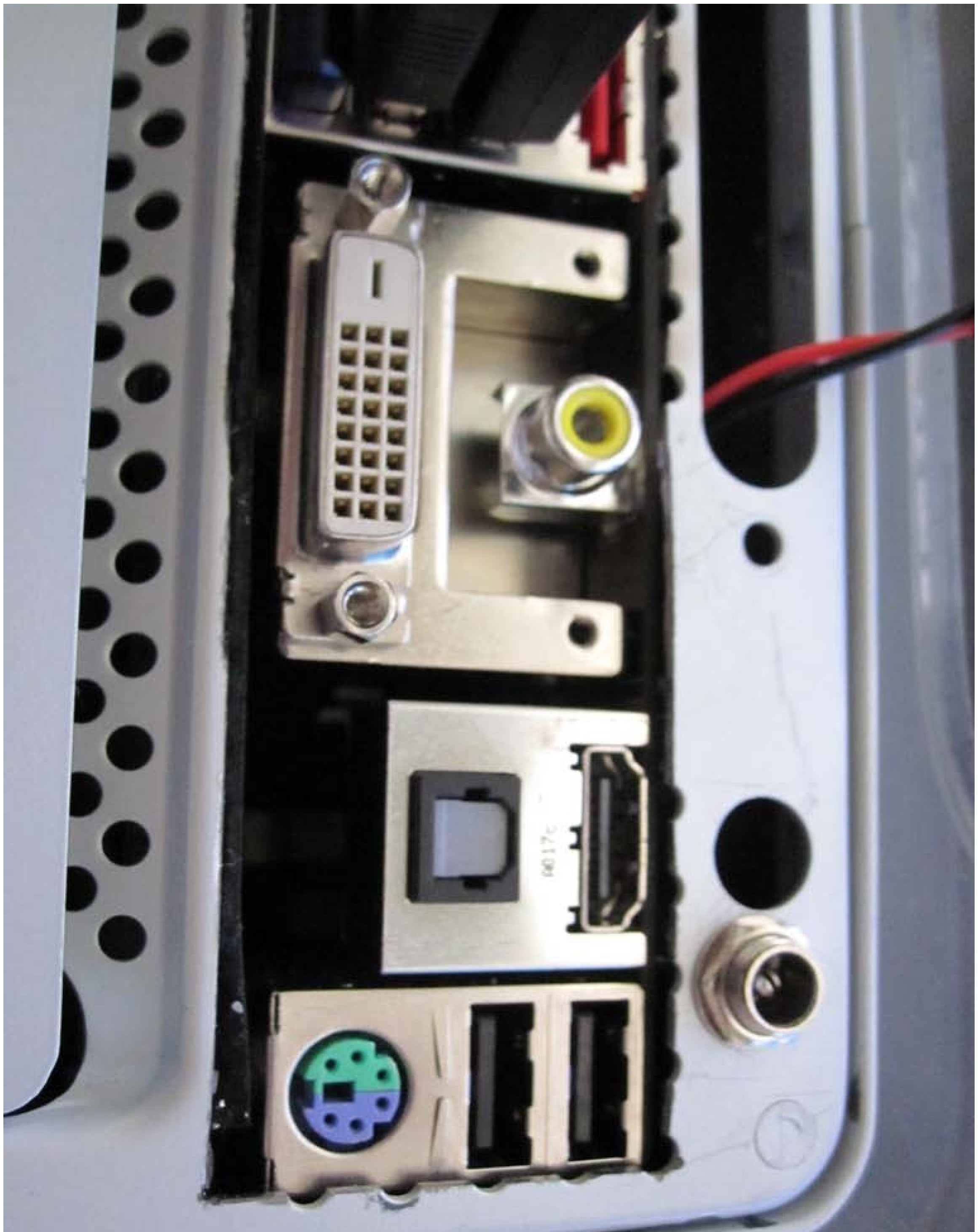
As of 2002, less than 10% of all the CPUs sold in the world are 32-bit or more. Of all the 32-bit CPUs sold, about 2% are used in desktop or laptop personal computers. Most microprocessors are used in embedded control applications such as household appliances, automobiles, and computer peripherals. Taken as a whole, the average price for a microprocessor, microcontroller, or DSP is just over $6.

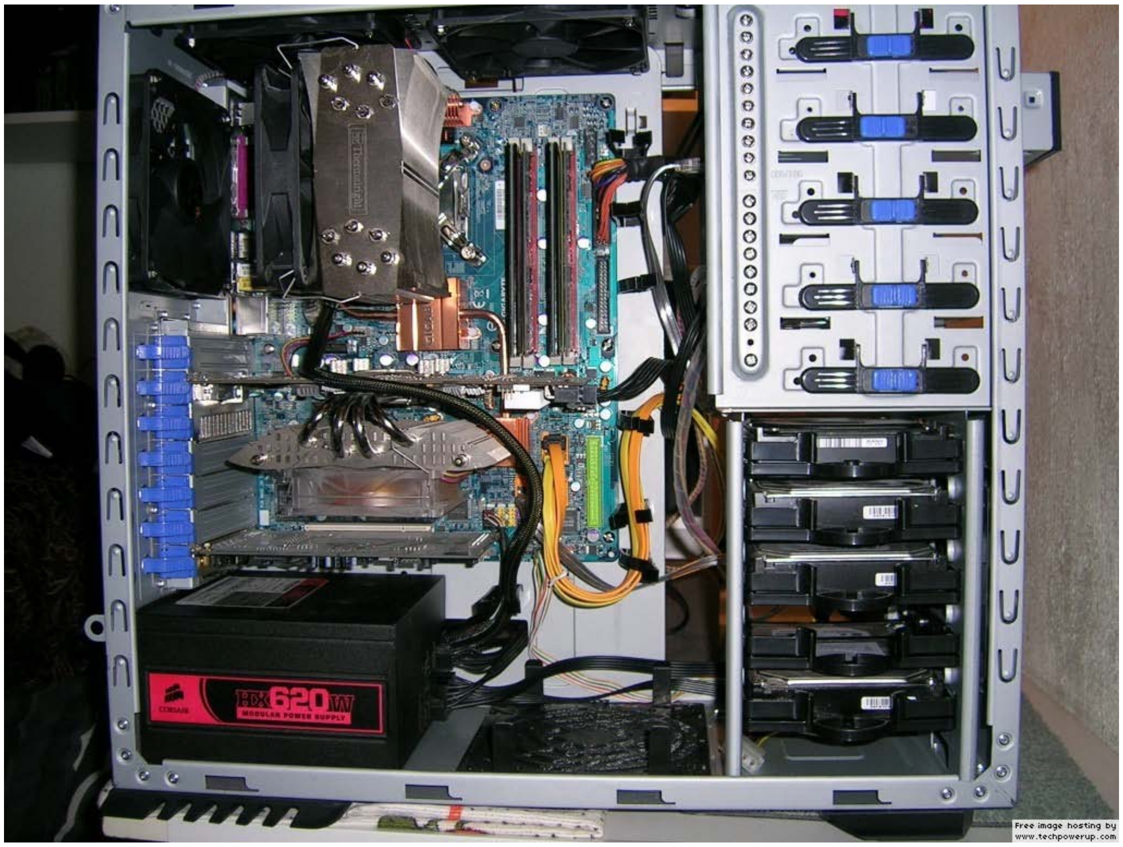
About ten billion CPUs were manufactured in 2008. About 98% of new CPUs produced each year are embedded.

CPU design

CPU design is the design engineering task of creating a central processing unit (CPU), a component of computer hardware. It is a subfield of electronics engineering and computer engineering.






HX620W
MODULAR POWER SUPPLY
CORSAIR
Free image hosting by www.techpowerup.com

Overview

CPU design focuses on these areas:

1. datapaths (such as ALUs and pipelines)
2. control unit: logic which controls the datapaths
3. Memory components such as register files, caches
4. Clock circuitry such as clock drivers, PLLs, clock distribution networks
5. Pad transceiver circuitry
6. Logic gate cell library which is used to implement the logic

CPUs designed for high-performance markets might require custom designs for each of these items to achieve frequency, power-dissipation, and chip-area goals.

CPUs designed for lower performance markets might lessen the implementation burden by:

- Acquiring some of these items by purchasing them as intellectual property
- Use control logic implementation techniques (logic synthesis using CAD tools) to implement the other components - datapaths, register files, clocks

Common logic styles used in CPU design include:

- Unstructured random logic
- Finite-state machines
- Microprogramming (common from 1965 to 1985)
- Programmable logic array (common in the 1980s, no longer common)

Device types used to implement the logic include:

- Transistor-transistor logic Small Scale Integration logic chips - no longer used for CPUs
- Programmable Array Logic and Programmable logic devices - no longer used for CPUs
- Emitter-coupled logic (ECL) gate arrays - no longer common
- CMOS gate arrays - no longer used for CPUs
- CMOS ASICs - what's commonly used today, they're so common that the term ASIC is not used for CPUs
- Field-programmable gate arrays (FPGA) - common for soft microprocessors, and more or less required for reconfigurable computing

A CPU design project generally has these major tasks:

- Programmer-visible instruction set architecture, which can be implemented by a variety of microarchitectures

- Architectural study and performance modeling in ANSI C/C++ or SystemC
- High-level synthesis (HLS) or RTL (eg. logic) implementation
- RTL Verification
- Circuit design of speed critical components (caches, registers, ALUs)
- Logic synthesis or logic-gate-level design
- Timing analysis to confirm that all logic and circuits will run at the specified operating frequency
- Physical design including floorplanning, place and route of logic gates
- Checking that RTL, gate-level, transistor-level and physical-level representations are equivalent
- Checks for signal integrity, chip manufacturability

As with most complex electronic designs, the logic verification effort (proving that the design does not have bugs) now dominates the project schedule of a CPU.

Key CPU architectural innovations include index register, cache, virtual memory, instruction pipelining, superscalar, CISC, RISC, virtual machine, emulators, microprogram, and stack.

Goals

The first CPUs were designed to do mathematical calculations faster and more reliably than human computers.

Each successive generation of CPU might be designed to achieve some of these goals:

- higher performance levels of a single program or thread
- higher throughput levels of multiple programs/threads
- less power consumption for the same performance level
- lower cost for the same performance level
- greater connectivity to build larger, more parallel systems
- more specialization to aid in specific targeted markets

Re-designing a CPU core to a smaller die-area helps achieve several of these goals.

- Shrinking everything (a "photomask shrink"), resulting in the same number of transistors on a smaller die, improves performance (smaller transistors switch faster), reduces power (smaller wires have less parasitic capacitance) and reduces cost (more CPUs fit on the same wafer of silicon).
- Releasing a CPU on the same size die, but with a smaller CPU core, keeps the cost about the same but allows higher levels of integration within one VLSI chip (additional cache, multiple CPUs, or other components), improving performance and reducing overall system cost.

Performance analysis and benchmarking

Because there are too many programs to test a CPU's speed on all of them, benchmarks were developed. The most famous benchmarks are the SPECint and SPECfp benchmarks developed by Standard Performance Evaluation Corporation and the ConsumerMark benchmark developed by the Embedded Microprocessor Benchmark Consortium EEMBC.

Some important measurements include:

- Instructions per second - Most consumers pick a computer architecture (normally Intel IA32 architecture) to be able to run a large base of pre-existing pre-compiled software. Being relatively uninformed on computer benchmarks, some of them pick a particular CPU based on operating frequency.
- FLOPS - The number of floating point operations per second is often important in selecting computers for scientific computations.
- Performance per watt - System designers building parallel computers, such as Google, pick CPUs based on their speed per watt of power, because the cost of powering the CPU outweighs the cost of the CPU itself.
- Some system designers building parallel computers pick CPUs based on the speed per dollar.
- System designers building real-time computing systems want to guarantee worst-case response. That is easier to do when the CPU has low interrupt latency and when it has deterministic response. (DSP)
- Computer programmers who program directly in assembly language want a CPU to support a full featured instruction set.
- Low power - For systems with limited power sources (e.g. solar, batteries, human power).
- Small size or low weight - for portable embedded systems, systems for spacecraft.
- Environmental impact - Minimizing environmental impact of computers during manufacturing and recycling as well during use. Reducing waste, reducing hazardous materials.

Some of these measures conflict. In particular, many design techniques that make a CPU run faster make the "performance per watt", "performance per dollar", and "deterministic response" much worse, and vice versa.

Markets

There are several different markets in which CPUs are used. Since each of these markets differ in their requirements for CPUs, the devices designed for one market are in most cases inappropriate for the other markets.

General purpose computing

The vast majority of revenues generated from CPU sales is for general purpose computing. That is, desktop, laptop and server computers commonly used in businesses and homes. In this market, the Intel IA-32 architecture dominates, with its rivals PowerPC and SPARC maintaining much smaller customer bases. Yearly, hundreds of millions of IA-32 architecture CPUs are used by this market.

Since these devices are used to run countless different types of programs, these CPU designs are not specifically targeted at one type of application or one function. The demands of being able to run a wide range of programs efficiently has made these CPU designs among the more advanced technically, along with some disadvantages of being relatively costly, and having high power consumption.

High-end processor economics

In 1984, most high-performance CPUs required four to five years to develop.

Developing new, high-end CPUs is a very costly proposition. Both the logical complexity (needing very large logic design and logic verification teams and simulation farms with perhaps thousands of computers) and the high operating frequencies (needing large circuit design teams and access to the state-of-the-art fabrication process) account for the high cost of design for this type of chip. The design cost of a high-end CPU will be on the order of US $100 million. Since the design of such high-end chips nominally takes about five years to complete, to stay competitive a company has to fund at least two of these large design teams to release products at the rate of 2.5 years per product generation.

As an example, the typical loaded cost for one computer engineer is often quoted to be $250,000 US dollars/year. This includes salary, benefits, CAD tools, computers, office space rent, etc. Assuming that 100 engineers are needed to design a CPU and the project takes 4 years.

Total cost = $250,000 / Engineer-Man/Year x 100 engineers x 4 years = $100,000,000 USD.

The above amount is just an example. The design teams for modern day general purpose CPUs have several hundred team members.

Scientific computing

A much smaller niche market (in revenue and units shipped) is scientific computing, used in government research labs and universities. Previously much CPU design was done for this market, but the cost-effectiveness of using mass markets CPUs has curtailed almost all specialized designs for this market. The main remaining area of active hardware design and research for scientific computing is for high-speed system interconnects.

Embedded design

As measured by units shipped, most CPUs are embedded in other machinery, such as telephones, clocks, appliances, vehicles, and infrastructure. Embedded processors sell in the volume of many billions of units per year, however, mostly at much lower price points than that of the general purpose processors.

These single-function devices differ from the more familiar general-purpose CPUs in several ways:

- Low cost is of utmost importance.
- It is important to maintain a low power dissipation as embedded devices often have a limited battery life and it is often impractical to include cooling fans.
- To give lower system cost, peripherals are integrated with the processor on the same silicon chip.
- Keeping peripherals on-chip also reduces power consumption as external GPIO ports typically require buffering so that they can source or sink the relatively high current loads that are required to maintain a strong signal outside of the chip.
    - Many embedded applications have a limited amount of physical space for circuitry; keeping peripherals on-chip will reduce the space required for the circuit board.
    - The program and data memories are often integrated on the same chip. When the only allowed program memory is ROM, the device is known as a microcontroller.
- For many embedded applications, interrupt latency will be more critical than in some general-purpose processors.

Embedded processor economics

As of 2009, more CPUs are produced using the ARM architecture instruction set than any other 32-bit instruction set. The ARM architecture and the first ARM chip were designed in about one and a half years and 5 man years of work time.

The 32-bit Parallax Propeller microcontroller architecture and the first chip were designed by two people in about 10 man years of work time.

It is believed that the 8-bit AVR architecture and first AVR microcontroller was conceived and designed by two students at the Norwegian Institute of Technology.

The 8-bit 6502 architecture and the first MOS Technology 6502 chip were designed in 13 months by a group of about 9 people.

Research and educational CPU design

The 32 bit Berkeley RISC I and RISC II architecture and the first chips were mostly designed by a series of students as part of a four quarter sequence of graduate courses. This design became the basis of the commercial SPARC processor design.

For about a decade, every student taking the 6.004 class at MIT was part of a team—each team had one semester to design and build a simple 8 bit CPU out of 7400 series integrated circuits. One team of 4 students designed and built a simple 32 bit CPU during that semester.

Some undergraduate courses require a team of 2 to 5 students to design, implement, and test a simple CPU in a FPGA in a single 15 week semester.

Soft microprocessor cores

For embedded systems, the highest performance levels are often not needed or desired due to the power consumption requirements. This allows for the use of processors which can be totally implemented by logic synthesis techniques. These synthesized processors can be implemented in a much shorter amount of time, giving quicker time-to-market.

Chapter 4

Instruction Pipeline and Superscalar

Instruction pipeline

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Basic five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

An **instruction pipeline** is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput (the number of instructions that can be executed in a unit of time).

The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step. This allows the computer's control circuitry to issue instructions at the processing rate of the slowest step, which is much faster than the time needed to perform all steps at once. The term pipeline refers to the fact that each step is carrying data at once (like water), and each step is connected to the next (like the links of a pipe.)

The origin of pipelining is thought to be either the ILLIAC II project or the IBM Stretch project though a simple version was used earlier in the Z1 in 1939 and the Z3 in 1941. .

The IBM Stretch Project proposed the terms, "Fetch, Decode, and Execute" that became common usage.

Most modern CPUs are driven by a clock. The CPU consists internally of logic and memory (flip flops). When the clock signal arrives, the flip flops take their new value and the logic then requires a period of time to decode the new values. Then the next clock pulse arrives and the flip flops again take their new values, and so on. By breaking the logic into smaller pieces and inserting flip flops between the pieces of logic, the delay before the logic gives valid outputs is reduced. In this way the clock period can be reduced. For example, the classic RISC pipeline is broken into five stages with a set of flip flops between each stage.

1. Instruction fetch
2. Instruction decode and register fetch
3. Execute
4. Memory access
5. Register write back

Hazards: When a programmer (or compiler) writes assembly code, they make the assumption that each instruction is executed before execution of the subsequent instruction is begun. This assumption is invalidated by pipelining. When this causes a program to behave incorrectly, the situation is known as a hazard. Various techniques for resolving hazards such as forwarding and stalling exist.

A non-pipeline architecture is inefficient because some CPU components (modules) are idle while another module is active during the instruction cycle. Pipelining does not completely cancel out idle time in a CPU but making those modules work in parallel improves program execution significantly.

Processors with pipelining are organized inside into stages which can semi-independently work on separate jobs. Each stage is organized and linked into a 'chain' so each stage's output is fed to another stage until the job is done. This organization of the processor allows overall processing time to be significantly reduced.

A deeper pipeline means that there are more stages in the pipeline, and therefore, fewer logic gates in each stage. This generally means that the processor's frequency can be increased as the cycle time is lowered. This happens because there are fewer components in each stage of the pipeline, so the propagation delay is decreased for the overall stage.

Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing. If 4 instructions that depend on the output of the first instruction are not available, the pipeline control logic must insert a stall or wasted clock cycle into the pipeline until the dependency is resolved. Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required. While pipelining can in theory increase performance over an unpipelined core by a factor of the number of stages (assuming the clock frequency also scales with the number of stages), in reality, most code does not allow for ideal execution.

Advantages and Disadvantages

Pipelining does not help in all cases. There are several possible disadvantages. An instruction pipeline is said to be fully pipelined if it can accept a new instruction every clock cycle. A pipeline that is not fully pipelined has wait cycles that delay the progress of the pipeline.

Advantages of Pipelining:

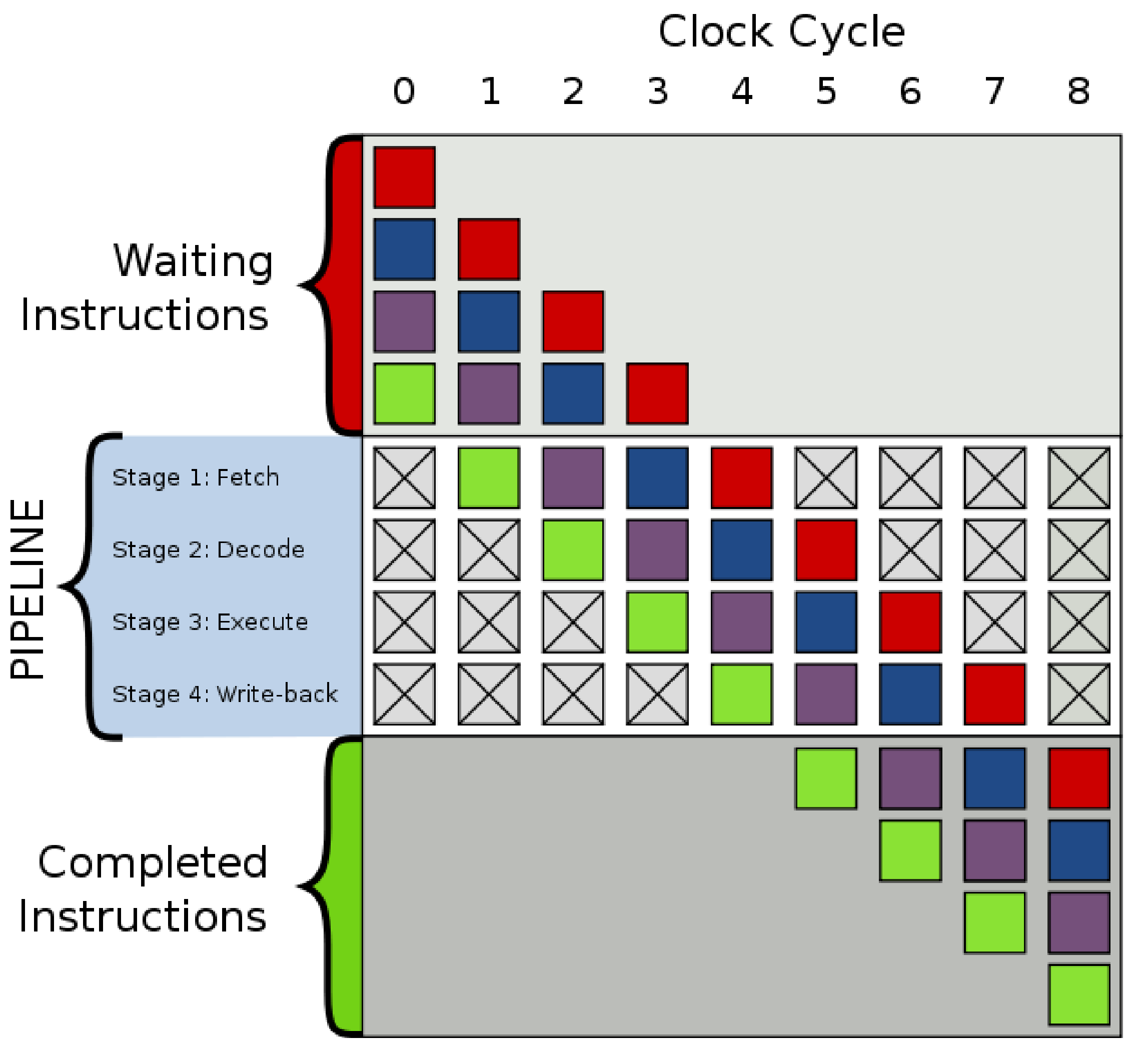
1. The cycle time of the processor is reduced, thus increasing instruction issue-rate in most cases.
2. Some combinational circuits such as adders or multipliers can be made faster by adding more circuitry. If pipelining is used instead, it can save circuitry vs. a more complex combinational circuit.

Disadvantages of Pipelining:

1. A non-pipelined processor executes only a single instruction at a time. This prevents branch delays (in effect, every branch is delayed) and problems with serial instructions being executed concurrently. Consequently the design is simpler and cheaper to manufacture.
2. The instruction latency in a non-pipelined processor is slightly lower than in a pipelined equivalent. This is because extra flip flops must be added to the data path of a pipelined processor.
3. A non-pipelined processor will have a stable instruction bandwidth. The performance of a pipelined processor is much harder to predict and may vary more widely between different programs.

Examples

Generic pipeline



Generic 4-stage pipeline; the colored boxes represent instructions independent of each other

To the right is a generic pipeline with four stages:

1. Fetch
2. Decode
3. Execute
4. Write-back

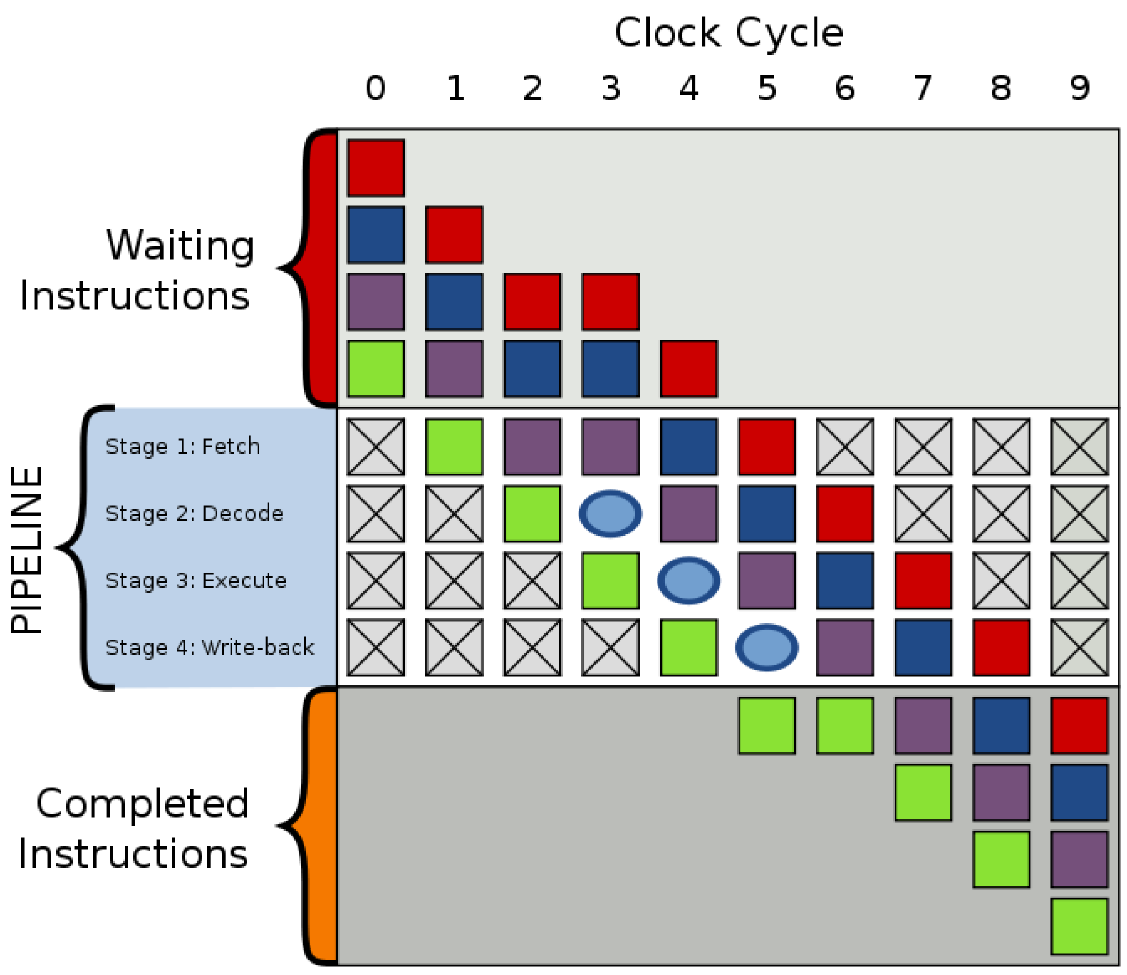
(for lw and sw memory is accessed after execute stage)

The top gray box is the list of instructions waiting to be executed; the bottom gray box is the list of instructions that have been completed; and the middle white box is the pipeline.

Execution is as follows:

Time	**Execution**
0	Four instructions are awaiting to be executed
1	• the green instruction is fetched from memory
2	• the green instruction is decoded • the purple instruction is fetched from memory
3	• the green instruction is executed (actual operation is performed) • the purple instruction is decoded • the blue instruction is fetched
4	• the green instruction's results are written back to the register file or memory • the purple instruction is executed • the blue instruction is decoded • the red instruction is fetched
5	• the green instruction is completed • the purple instruction is written back • the blue instruction is executed • the red instruction is decoded
6	• The purple instruction is completed • the blue instruction is written back • the red instruction is executed
7	• the blue instruction is completed • the red instruction is written back
8	• the red instruction is completed
9	All instructions are executed

Bubble



A bubble in cycle 3 delays execution

When a "hiccup" in execution occurs, a "bubble" is created in the pipeline in which nothing useful happens. In cycle 2, the fetching of the purple instruction is delayed and the decoding stage in cycle 3 now contains a bubble. Everything "behind" the purple instruction is delayed as well but everything "ahead" of the purple instruction continues with execution.

Clearly, when compared to the execution above, the bubble yields a total execution time of 8 clock ticks instead of 7.

Bubbles are like stalls, in which nothing useful will happen for the fetch, decode, execute and writeback. It can be completed with a nop code.

Example 1

A typical instruction to add two numbers might be `ADD A, B, C`, which adds the values found in memory locations A and B, and then puts the result in memory location C. In a pipelined processor the pipeline controller would break this into a series of tasks similar to:

```
LOAD R1, A
LOAD R2, B
ADD R3, R1, R2
STORE C, R3
LOAD next instruction
```

The locations 'R1', 'R2' and 'R3' are registers in the CPU. The values stored in memory locations labeled 'A' and 'B' are loaded (copied) into the R1 and R2 registers, then added, and the result (which is in register R3) is stored in a memory location labeled 'C'.

In this example the pipeline is three stages long- load, execute, and store. Each of the steps are called **pipeline stages**.

On a non-pipelined processor, only one stage can be working at a time so the entire instruction has to complete before the next instruction can begin. On a pipelined processor, all of the stages can be working at once on different instructions. So when this instruction is at the execute stage, a second instruction will be at the decode stage and a 3rd instruction will be at the fetch stage.

Pipelining doesn't reduce the time it takes to complete an instruction; it increases the number of instructions that can be processed at once and reduces the delay between completed instructions. The more pipeline stages a processor has, the more instructions it can be working on at once and the less of a delay there is between completed instructions. Every microprocessor manufactured today uses at least 2 stages of pipeline. (The Atmel AVR and the PIC microcontroller each have a 2 stage pipeline.) Intel Pentium 4 processors have 20 stage pipelines.

Example 2

To better visualize the concept, we can look at a theoretical 3-stage pipeline:

Stage	Description
Load	Read instruction from memory
Execute	Execute instruction
Store	Store result in memory and/or registers

and a pseudo-code assembly listing to be executed:

```
LOAD  A, #40      ; load 40 in A
```

```
MOVE  B, A        ; copy A in B
ADD   B, #20      ; add 20 to B
STORE 0x300, B    ; store B into memory cell 0x300
```

This is how it would be executed:

Clock 1

Load Execute Store

LOAD

The LOAD instruction is fetched from memory.

Clock 2

Load Execute Store

MOVE LOAD

The LOAD instruction is executed, while the MOVE instruction is fetched from memory.

Clock 3

Load Execute Store

ADD MOVE LOAD

The LOAD instruction is in the Store stage, where its result (the number 40) will be stored in the register A. In the meantime, the MOVE instruction is being executed. Since it must move the contents of A into B, it must wait for the ending of the LOAD instruction.

Clock 4

Load Execute Store

STORE ADD MOVE

The STORE instruction is loaded, while the MOVE instruction is finishing off and the ADD is calculating.

And so on. Note that, sometimes, an instruction will depend on the result of another one (like our MOVE example). When more than one instruction references a particular location for an operand, either reading it (as an input) or writing it (as an output), executing those instructions in an order different from the original program order can lead to hazards (mentioned above). There are several established techniques for either preventing hazards from occurring, or working around them if they do.

Complications

Many designs include pipelines as long as 7, 10 and even 20 stages (like in the Intel Pentium 4) The later "Prescott" and "Cedar Mill" Pentium 4 cores (and their Pentium D derivatives) had a 31-stage pipeline, the longest in mainstream consumer computing. The Xelerator X10q has a pipeline more than a thousand stages long . The downside of a long pipeline is that when a program branches, the processor cannot know where to fetch the next instruction from and must wait until the branch instruction finishes, leaving the pipeline behind it empty. In the extreme case, the performance of a pipelined processor could theoretically approach that of an un-pipelined processor, or even slightly worse if all but one pipeline stages are idle and a small overhead is present between stages. Branch prediction attempts to alleviate this problem by guessing whether the branch will be taken or not and speculatively executing the code path that it predicts will be taken. When its predictions are correct, branch prediction avoids the penalty associated with branching. However, branch prediction itself can end up exacerbating the problem if branches are predicted poorly, as the incorrect code path which has begun execution must be flushed from the pipeline before resuming execution at the correct location.

In certain applications, such as supercomputing, programs are specially written to branch rarely and so very long pipelines can speed up computation by reducing cycle time. If branching happens constantly, re-ordering branches such that the more likely to be needed instructions are placed into the pipeline can significantly reduce the speed losses associated with having to flush failed branches. Programs such as gcov can be used to examine how often particular branches are actually executed using a technique known as coverage analysis; however such analysis is often a last resort for optimization.

Self-Modifying Programs: Because of the instruction pipeline, code that the processor loads will not immediately execute. Due to this, updates in the code very near the current location of execution may not take effect because they are already loaded into the Prefetch Input Queue. Instruction caches make this phenomenon even worse. This is only relevant to self-modifying programs.

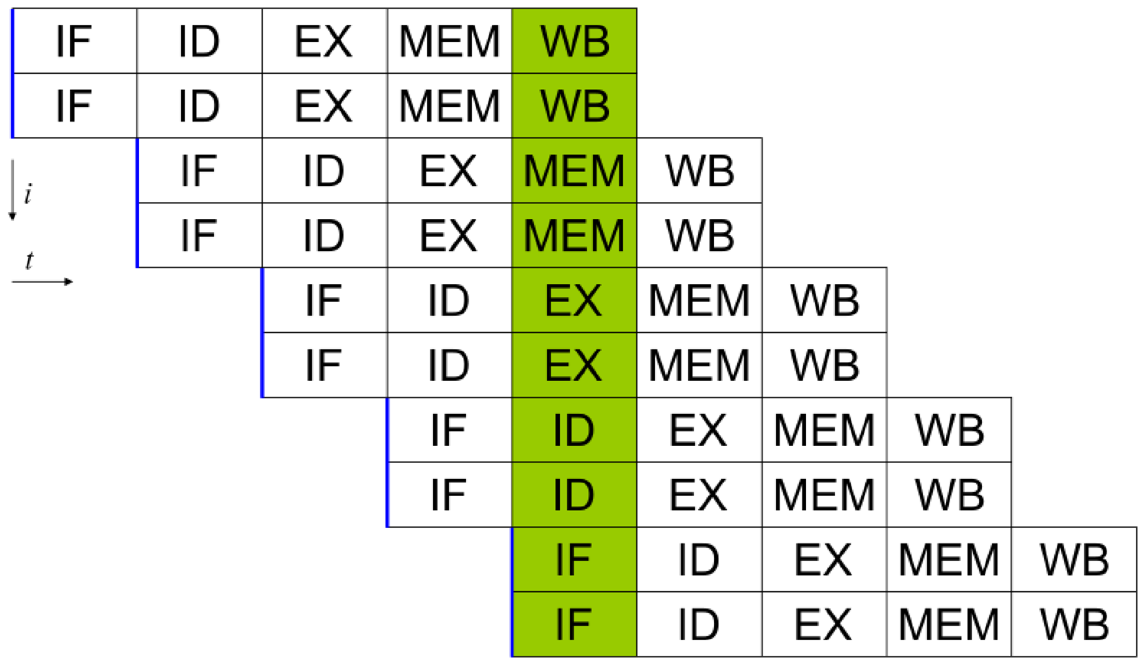
Mathematical pipelines: Mathematical or arithmetic pipelines are different from instructional pipelines, in that when mathematically processing large arrays or vectors, a particular mathematical process, such as a multiply is repeated many thousands of times. In this environment, an instruction need only kick off an event whereby the arithmetic logic unit (which is pipelined) takes over, and begins its series of calculations. Most of these circuits can be found today in math processors and math processing sections of CPUs like the Intel Pentium line.

History

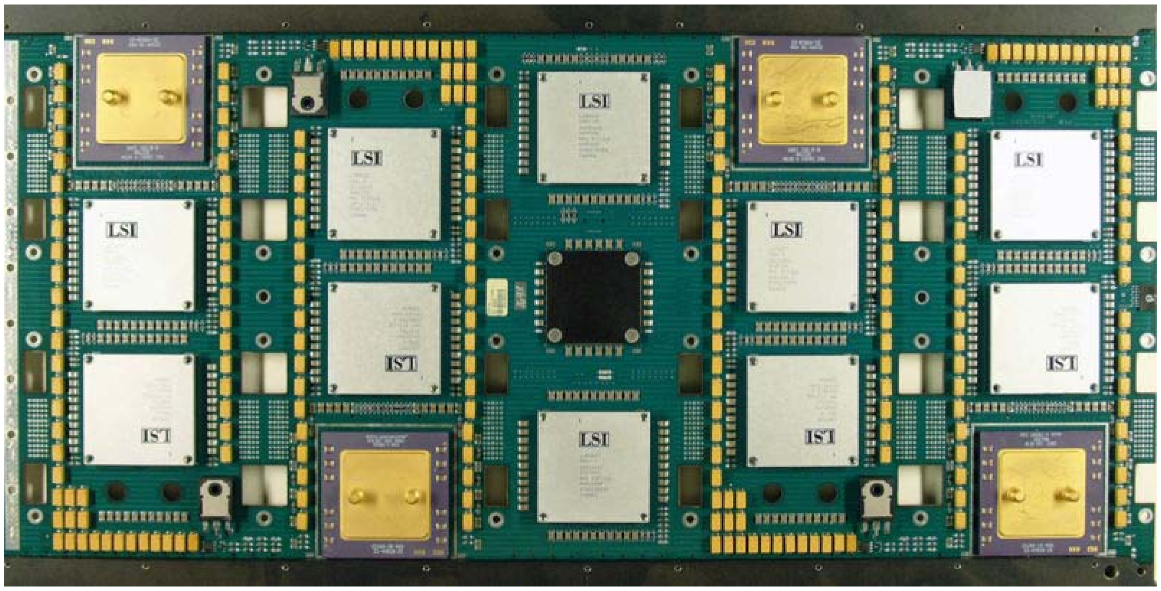
Math processing (super-computing) began in earnest in the late 1970s as Vector Processors and Array Processors. Usually very large bulky super-computing machines that needed special environments and super-cooling of the cores. One of the early super computers was the Cyber series built by Control Data Corporation. Its main architect was

Seymour Cray, who later resigned from CDC to head up Cray Research. Cray developed the XMP line of super computers, using pipelining for both multiply and add/subtract functions. Later, Star Technologies took pipelining to another level by adding parallelism (several pipelined functions working in parallel), developed by their engineer, Roger Chen. In 1984, Star Technologies made another breakthrough with the pipelined divide circuit, developed by James Bradley. By the mid 1980s, super-computing had taken off with offerings from many different companies around the world.

Superscalar



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



Processor board of a CRAY T3e supercomputer with four superscalar Alpha 21164 processors

A **superscalar** CPU architecture implements a form of parallelism called instruction level parallelism within a single processor. It therefore allows faster CPU throughput than would otherwise be possible at a given clock rate. A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier.

While a superscalar CPU is typically also pipelined, pipelining and superscalar architecture are considered different performance enhancement techniques.

The superscalar technique is traditionally associated with several identifying characteristics (within a given CPU core):

- Instructions are issued from a sequential instruction stream
- CPU hardware dynamically checks for data dependencies between instructions at run time (versus software checking at compile time)
- The CPU accepts multiple instructions per clock cycle

History

Seymour Cray's CDC 6600 from 1965 is often mentioned as the first superscalar design. The Intel i960CA (1988) and the AMD 29000-series 29050 (1990) microprocessors were the first commercial single-chip superscalar microprocessors. RISC CPUs like these brought the superscalar concept to microcomputers because the RISC design results in a simple core, allowing straightforward instruction dispatch and the inclusion of multiple functional units (such as ALUs) on a single CPU in the constrained design rules of the time. This was the reason that RISC designs were faster than CISC designs through the 1980s and into the 1990s.

Except for CPUs used in low-power applications, embedded systems, and battery-powered devices, essentially all general-purpose CPUs developed since about 1998 are superscalar.

The P5 Pentium was the first superscalar x86 processor; the Nx586, P6 Pentium Pro and AMD K5 were among the first designs which decode x86-instructions asynchronously into dynamic microcode-like micro-op sequences prior to actual execution on a superscalar microarchitecture; this opened up for dynamic scheduling of buffered partial instructions and enabled more parallelism to be extracted compared to the more rigid methods used in the simpler P5 Pentium; it also simplified speculative execution and allowed higher clock frequencies compared to designs such as the advanced Cyrix 6x86.

From scalar to superscalar

The simplest processors are scalar processors. Each instruction executed by a scalar processor typically manipulates one or two data items at a time. By contrast, each instruction executed by a vector processor operates simultaneously on many data items. An analogy is the difference between scalar and vector arithmetic. A superscalar processor is sort of a mixture of the two. Each instruction processes one data item, but there are multiple redundant functional units within each CPU thus multiple instructions can be processing separate data items concurrently.

Superscalar CPU design emphasizes improving the instruction dispatcher accuracy, and allowing it to keep the multiple functional units in use at all times. This has become increasingly important when the number of units increased. While early superscalar CPUs would have two ALUs and a single FPU, a modern design such as the PowerPC 970 includes four ALUs, two FPUs, and two SIMD units. If the dispatcher is ineffective at keeping all of these units fed with instructions, the performance of the system will suffer.

A superscalar processor usually sustains an execution rate in excess of one instruction per machine cycle. But merely processing multiple instructions concurrently does not make an architecture superscalar, since pipelined, multiprocessor or multi-core architectures also achieve that, but with different methods.

In a superscalar CPU the dispatcher reads instructions from memory and decides which ones can be run in parallel, dispatching them to redundant functional units contained inside a single CPU. Therefore a superscalar processor can be envisioned having multiple parallel pipelines, each of which is processing instructions simultaneously from a single instruction thread.

Limitations

Available performance improvement from superscalar techniques is limited by three key areas:

1. The degree of intrinsic parallelism in the instruction stream, i.e. limited amount of instruction-level parallelism.
2. The complexity and time cost of the dispatcher and associated dependency checking logic.
3. The branch instruction processing.

Existing binary executable programs have varying degrees of intrinsic parallelism. In some cases instructions are not dependent on each other and can be executed simultaneously. In other cases they are inter-dependent: one instruction impacts either resources or results of the other. The instructions `a = b + c; d = e + f` can be run in parallel because none of the results depend on other calculations. However, the instructions `a = b + c; b = e + f` might not be runnable in parallel, depending on the order in which the instructions complete while they move through the units.

When the number of simultaneously issued instructions increases, the cost of dependency checking increases extremely rapidly. This is exacerbated by the need to check dependencies at run time and at the CPU's clock rate. This cost includes additional logic gates required to implement the checks, and time delays through those gates. Research shows the gate cost in some cases may be n^k gates, and the delay cost $k^2 \log n$, where n is the number of instructions in the processor's instruction set, and k is the number of simultaneously dispatched instructions. In mathematics, this is called a combinatoric problem involving permutations.

Even though the instruction stream may contain no inter-instruction dependencies, a superscalar CPU must nonetheless check for that possibility, since there is no assurance otherwise and failure to detect a dependency would produce incorrect results.

No matter how advanced the semiconductor process or how fast the switching speed, this places a practical limit on how many instructions can be simultaneously dispatched. While process advances will allow ever greater numbers of functional units (e.g., ALUs), the burden of checking instruction dependencies grows so rapidly that the achievable superscalar dispatch limit is fairly small. -- likely on the order of five to six simultaneously dispatched instructions.

However even given infinitely fast dependency checking logic on an otherwise conventional superscalar CPU, if the instruction stream itself has many dependencies, this would also limit the possible speedup. Thus the degree of intrinsic parallelism in the code stream forms a second limitation.

Alternatives

Collectively, these two limits drive investigation into alternative architectural performance increases such as Very Long Instruction Word (VLIW), Explicitly Parallel Instruction Computing (EPIC), simultaneous multithreading (SMT), and multi-core processors.

With VLIW, the burdensome task of dependency checking by hardware logic at run time is removed and delegated to the compiler. Explicitly Parallel Instruction Computing (EPIC) is like VLIW, with extra cache prefetching instructions.

Simultaneous multithreading, often abbreviated as SMT, is a technique for improving the overall efficiency of superscalar CPUs. SMT permits multiple independent threads of execution to better utilize the resources provided by modern processor architectures.

Superscalar processors differ from multi-core processors in that the redundant functional units are not entire processors. A single processor is composed of finer-grained functional units such as the ALU, integer multiplier, integer shifter, floating point unit, etc. There may be multiple versions of each functional unit to enable execution of many instructions in parallel. This differs from a multi-core processor that concurrently processes instructions from multiple threads, one thread per core. It also differs from a pipelined CPU, where the multiple instructions can concurrently be in various stages of execution, assembly-line fashion.

The various alternative techniques are not mutually exclusive—they can be (and frequently are) combined in a single processor. Thus a multicore CPU is possible where each core is an independent processor containing multiple parallel pipelines, each pipeline being superscalar. Some processors also include vector capability.

Chapter 5

Vector Processor and SIMD

Vector processor

A **vector processor**, or **array processor**, is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors. This is in contrast to a scalar processor, whose instructions operate on single data items. The vast majority of CPUs are scalar.

Vector processors first appeared in the 1970s, and formed the basis of most supercomputers through the 1980s and into the 1990s. Improvements in scalar processors, particularly microprocessors, resulted in the decline of traditional vector processors in supercomputers, and the appearance of vector processing techniques in mass market CPUs around the early 1990s. Today, most commodity CPUs implement architectures that feature instructions for some vector processing on multiple (vectorized) data sets, typically known as SIMD (**S**ingle **I**nstruction, **M**ultiple **D**ata). Common examples include MMX, SSE, and AltiVec. Vector processing techniques are also found in video game console hardware and graphics accelerators. In 2000, IBM, Toshiba and Sony collaborated to create the Cell processor, consisting of one scalar processor and eight vector processors, which found use in the Sony PlayStation 3 among other applications.

Other CPU designs may include some multiple instructions for vector processing on multiple (vectorised) data sets, typically known as MIMD (**M**ultiple **I**nstruction, **M**ultiple **D**ata). Such designs are usually dedicated to a particular application and not commonly marketed for general purpose computing.

History

Vector processing was first worked on in the early 1960s at Westinghouse in their **Solomon** project. Solomon's goal was to dramatically increase math performance by using a large number of simple math co-processors under the control of a single master CPU. The CPU fed a single common instruction to all of the arithmetic logic units (ALUs), one per "cycle", but with a different data point for each one to work on. This allowed the Solomon machine to apply a single algorithm to a large data set, fed in the form of an array. In 1962, Westinghouse cancelled the project, but the effort was re-started at the University of Illinois as the ILLIAC IV. Their version of the design originally called for a 1 GFLOPS machine with 256 ALUs, but, when it was finally delivered in 1972, it had only 64 ALUs and could reach only 100 to 150 MFLOPS. Nevertheless it showed that the basic concept was sound, and, when used on data-intensive applications, such as computational fluid dynamics, the "failed" ILLIAC was the fastest machine in the world. The ILLIAC approach of using separate ALUs for each data element is not common to later designs, and is often referred to under a separate category, massively parallel computing.

The first successful implementation of vector processing appears to be the Control Data Corporation STAR-100 and the Texas Instruments Advanced Scientific Computer (ASC). The basic ASC (i.e., "one pipe") ALU used a pipeline architecture that supported both scalar and vector computations, with peak performance reaching approximately 20 MFLOPS, readily achieved when processing long vectors. Expanded ALU configurations supported "two pipes" or "four pipes" with a corresponding 2X or 4X performance gain. Memory bandwidth was sufficient to support these expanded modes. The STAR was otherwise slower than CDC's own supercomputers like the CDC 7600, but at data related tasks they could keep up while being much smaller and less expensive. However the machine also took considerable time decoding the vector instructions and getting ready to run the process, so it required very specific data sets to work on before it actually sped anything up.

The vector technique was first fully exploited in the famous Cray-1. Instead of leaving the data in memory like the STAR and ASC, the Cray design had eight "vector registers," which held sixty-four 64-bit words each. The vector instructions were applied between registers, which is much faster than talking to main memory. The Cray design used pipeline parallelism to implement vector instructions rather than multiple ALUs. In addition the design had completely separate pipelines for different instructions, for example, addition/subtraction was implemented in different hardware than multiplication. This allowed a batch of vector instructions themselves to be pipelined, a technique they called vector chaining. The Cray-1 normally had a performance of about 80 MFLOPS, but with up to three chains running it could peak at 240 MFLOPS – a respectable number even as of 2002.

Other examples followed. Control Data Corporation tried to re-enter the high-end market again with its ETA-10 machine, but it sold poorly and they took that as an opportunity to leave the supercomputing field entirely. In the early and mid-1980s Japanese companies (Fujitsu, Hitachi and Nippon Electric Corporation (NEC) introduced register-based vector machines similar to the Cray-1, typically being slightly faster and much smaller. Oregon-based Floating Point Systems (FPS) built add-on array processors for minicomputers, later building their own minisupercomputers. However Cray continued to be the performance leader, continually beating the competition with a series of machines that led to the Cray-2, Cray X-MP and Cray Y-MP. Since then, the supercomputer market has focused much more on massively parallel processing rather than better implementations of vector processors. However, recognising the benefits of vector processing IBM developed Virtual Vector Architecture for use in supercomputers coupling several scalar processors to act as a vector processor.

Vector processing techniques have since been added to almost all modern CPU designs, although they are typically referred to as SIMD. In these implementations, the vector unit runs beside the main scalar CPU, and is fed data from programs that know it is there.

Description

In general terms, CPUs are able to manipulate one or two pieces of data at a time. For instance, many CPUs have an instruction that essentially says "add A to B and put the result in C". The data for A, B and C could be—in theory at least—encoded directly into the instruction. However things are rarely that simple. In general the data is rarely sent in raw form, and is instead "pointed to" by passing in an address to a memory location that holds the data. Decoding this address and getting the data out of the memory takes some time. As CPU speeds have increased, this memory latency has historically become a large impediment to performance.

In order to reduce the amount of time this takes, most modern CPUs use a technique known as instruction pipelining in which the instructions pass through several sub-units in turn. The first sub-unit reads the address and decodes it, the next "fetches" the values at those addresses, and the next does the math itself. With pipelining the "trick" is to start decoding the next instruction even before the first has left the CPU, in the fashion of an assembly line, so the address decoder is constantly in use. Any particular instruction takes the same amount of time to complete, a time known as the latency, but the CPU can process an entire batch of operations much faster than if it did so one at a time.

Vector processors take this concept one step further. Instead of pipelining just the instructions, they also pipeline the data itself. They are fed instructions that say not just to add A to B, but to add all of the numbers "from here to here" to all of the numbers "from there to there". Instead of constantly having to decode instructions and then fetch the data needed to complete them, it reads a single instruction from memory, and "knows" that the next address will be one larger than the last. This allows for significant savings in decoding time.

To illustrate what a difference this can make, consider the simple task of adding two groups of 10 numbers together. In a normal programming language you would write a "loop" that picked up each of the pairs of numbers in turn, and then added them. To the CPU, this would look something like this:

```
execute this loop 10 times
  read the next instruction and decode it
  fetch this number
  fetch that number
  add them
  put the result here
end loop
```

But to a vector processor, this task looks considerably different:

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
add them
put the results here
```

There are several savings inherent in this approach. For one, only two address translations are needed. Depending on the architecture, this can represent a significant savings by itself. Another saving is fetching and decoding the instruction itself, which has to be done only one time instead of ten. The code itself is also smaller, which can lead to more efficient memory use.

But more than that, a vector processor may have multiple functional units adding those numbers in parallel. The checking of dependencies between those numbers is not required as a vector instruction specifies multiple independent operations. This simplifies the control logic required, and can improve performance by avoiding stalls.

As mentioned earlier, the Cray implementations took this a step further, allowing several different types of operations to be carried out at the same time. Consider code that adds two numbers and then multiplies by a third; in the Cray, these would all be fetched at once, and both added and multiplied in a single operation. Using the pseudocode above, the Cray did:

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
fetch another 10 numbers
add and multiply them
put the results here
```

The math operations thus completed far faster overall, the limiting factor being the time required to fetch the data from memory.

Not all problems can be attacked with this sort of solution. Adding these sorts of instructions necessarily adds complexity to the core CPU. That complexity typically makes other instructions run slower—i.e., whenever it is **not** adding up many numbers in a row. The more complex instructions also add to the complexity of the decoders, which might slow down the decoding of the more common instructions such as normal adding.

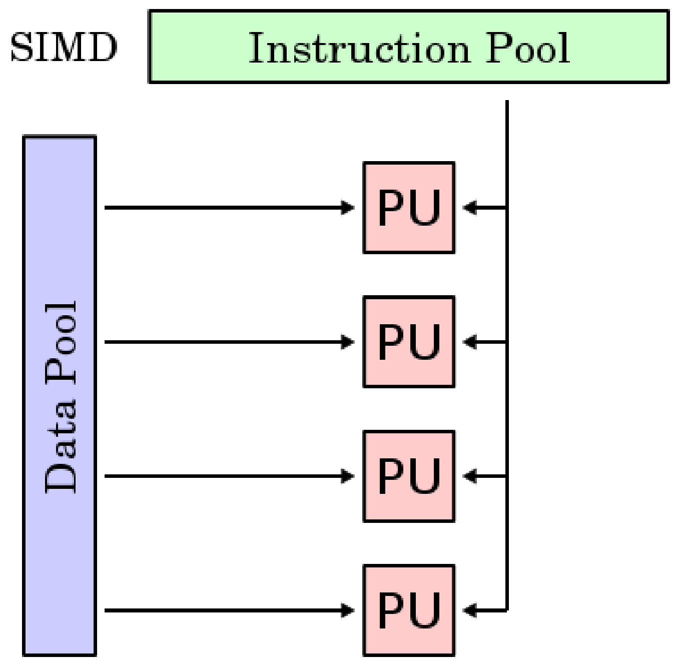
In fact, vector processors work best only when there are large amounts of data to be worked on. For this reason, these sorts of CPUs were found primarily in supercomputers, as the supercomputers themselves were, in general, found in places such as weather prediction centres and physics labs, where huge amounts of data are "crunched".

Real world example: vector instructions usage with the x86 architecture

Shown below is a actual x86 architecture example for vector instruction usage with the SSE instruction set. The example multiplies two arrays of single precision floating point numbers. It's written in the C language with inline assembly code parts for compilation with GCC (32bit).

```
//SSE simd function for vectorized multiplication of 2
arrays with single-precision floatingpoint numbers
//1st param pointer on source/destination array, 2nd param
2. source array, 3th param number of floats per array
 void mul_asm(float* out, float* in, unsigned int leng)
 {    unsigned int count, rest;

      //compute if array is big enough for vector operation
      rest  = (leng*4)%16;
      count = (leng*4)-rest;

     // vectorized part; 4 floats per loop iteration
      if (count>0){
      __asm __volatile__  (".intel_syntax noprefix\n\t"
      "loop:                 \n\t"
      "movups xmm0,[ebx+ecx] ;loads 4 floats in first
register (xmm0)\n\t"
      "movups xmm1,[eax+ecx] ;loads 4 floats in second
register (xmm1)\n\t"
      "mulps xmm0,xmm1       ;multiplies both vector
registers\n\t"
      "movups [eax+ecx],xmm0 ;write back the result to
memory\n\t"
      "sub ecx,16            ;increase address pointer by 4
floats\n\t"
      "jnz loop              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest):
"xmm0","xmm1");
      }

      // scalar part; 1 float per loop iteration
      if (rest!=0)
      {
       __asm __volatile__  (".intel_syntax noprefix\n\t"
      "add eax,ecx           \n\t"
      "add ebx,ecx           \n\t"

      "rest:                 \n\t"
      "movss xmm0,[ebx+edx]  ;load 1 float in first
register (xmm0)\n\t"
      "movss xmm1,[eax+edx]  ;load 1 float in second
register (xmm1)\n\t"
      "mulss xmm0,xmm1       ;multiplies both scalar parts
of registers\n\t"
      "movss [eax+edx],xmm0  ;write back the result\n\t"
      "sub edx,4             \n\t"
      "jnz rest              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest):
"xmm0","xmm1");
      }
      return;
 }
```

SIMD



Flynn's taxonomy

Single Instruction	**Multiple Instruction**

Single Data	SISD	MISD
Multiple Data	**SIMD**	MIMD

Single instruction, multiple data (SIMD), is a class of parallel computers in Flynn's taxonomy. It describes computers with multiple processing elements that perform the same operation on multiple data simultaneously. Thus, such machines exploit data level parallelism.

History

The first era of SIMD machines was characterized by supercomputers such as the Thinking Machines CM-1 and CM-2. These machines had many limited functionality processors that would work in parallel. For example, each of 64,000 processors in a Thinking Machines CM-2 would execute the same instruction at the same time so that you could do 64,000 multiplies on 64,000 pairs of numbers at a time.

Supercomputing moved away from the SIMD approach when inexpensive scalar MIMD approaches based on commodity processors such as the Intel i860 XP became more powerful, and interest in SIMD waned. Later, personal computers became common, and became powerful enough to support real-time gaming. This created a mass demand for a particular type of computing power, and microprocessor vendors turned to SIMD to meet the demand. Sun Microsystems introduced SIMD integer instructions in its "VIS" instruction set extensions in 1995, in its UltraSPARC I microprocessor. The first widely-deployed SIMD for gaming was Intel's MMX extensions to the x86 architecture. IBM and Motorola then added AltiVec to the POWER architecture, and there have been several extensions to the SIMD instruction sets for both architectures. All of these developments have been oriented toward support for real-time graphics, and are therefore oriented toward vectors of two, three, or four dimensions. When new SIMD architectures need to be distinguished from older ones, the newer architectures are then considered "short-vector" architectures. A modern supercomputer is almost always a cluster of MIMD machines, each of which implements (short-vector) SIMD instructions. A modern desktop computer is often a multiprocessor MIMD machine where each processor can execute short-vector SIMD instructions.

DSPs

A separate class of processors exists for this sort of task, commonly referred to as Digital Signal Processors, or **DSP**s. The main difference between DSP and other SIMD-capable CPUs is that the DSPs are self-contained processors with their own instruction set, while SIMD-extensions rely on the general-purpose portions of the CPU to handle the program details, and the SIMD instructions handle the data manipulation only. DSPs also tend to include instructions to handle specific types of data, sound or video for instance, while

SIMD systems are considerably of more generic purpose. DSPs generally operate in Scratchpad RAM driven by DMA transfers initiated from the host system and are unable to access external memory.

Some DSPs include SIMD instruction sets. The inclusion of SIMD units in general purpose processors has supplanted the use of DSP chips in computer systems, though they continue to be used in embedded applications. A sliding scale exists - the Cell's SPUs and Ageia's PhysX Physics Processing Unit could be considered half way between CPUs and DSPs, in that they are optimized for numeric tasks and operate in local store, but they can autonomously control their own transfers thus are in effect true CPUs.

Advantages

An application that may take advantage of SIMD is one where the same value is being added (or subtracted) to a large number of data points, a common operation in many multimedia applications. One example would be changing the brightness of an image. Each pixel of an image consists of three values for the brightness of the red (R), green (G) and blue (B) portions of the color. To change the brightness, the R, G and B values are read from memory, a value is added (or subtracted) from them, and the resulting values are written back out to memory.

With a SIMD processor there are two improvements to this process. For one the data is understood to be in blocks, and a number of values can be loaded all at once. Instead of a series of instructions saying "get this pixel, now get the next pixel", a SIMD processor will have a single instruction that effectively says "get lots of pixels" ("lots" is a number that varies from design to design). For a variety of reasons, this can take much less time than "getting" each pixel individually, as with traditional CPU design.

Another advantage is that SIMD systems typically include only those instructions that can be applied to all of the data in one operation. In other words, if the SIMD system works by loading up eight data points at once, the `add` operation being applied to the data will happen to all eight values at the same time. Although the same is true for any super-scalar processor design, the level of parallelism in a SIMD system is typically much higher.

Disadvantages

- Not all algorithms can be vectorized. For example, a flow-control-heavy task like code parsing wouldn't benefit from SIMD.
- It also has large register files which increases power consumption and chip area.
- Currently, implementing an algorithm with SIMD instructions usually requires human labor; most compilers don't generate SIMD instructions from a typical C program, for instance. Vectorization in compilers is an active area of computer science research. (Compare vector processing.)
- Programming with particular SIMD instruction sets can involve numerous low-level challenges.

  - SSE (Streaming SIMD Extension) has restrictions on data alignment; programmers familiar with the x86 architecture may not expect this.
  - Gathering data into SIMD registers and scattering it to the correct destination locations is tricky and can be inefficient.
  - Specific instructions like rotations or three-operand addition aren't in some SIMD instruction sets.
  - Instruction sets are architecture-specific: old processors and non-x86 processors lack SSE entirely, for instance, so programmers must provide non-vectorized implementations (or different vectorized implementations) for them.
  - The early MMX instruction set shared a register file with the floating-point stack, which caused inefficiencies when mixing floating-point and MMX code. However, SSE2 corrects this.

Chronology

The first use of SIMD instructions was in vector supercomputers of the early 1970s such as the CDC Star-100 and the Texas Instruments ASC. Vector processing was especially popularized by Cray in the 1970s and 1980s.

Later machines used a much larger number of relatively simple processors in a massively parallel processing-style configuration. Some examples of this type of machine included:

- ILLIAC IV, circa 1974
- ICL Distributed Array Processor (DAP), circa 1974
- Burroughs Scientific Processor, circa 1976
- Geometric-Arithmetic Parallel Processor, from Martin Marietta, starting in 1981, continued at Lockheed Martin, then at Teranex and Silicon Optix
- Massively Parallel Processor (MPP), from NASA/Goddard Space Flight Center, circa 1983-1991
- Connection Machine, models 1 and 2 (CM-1 and CM-2), from Thinking Machines Corporation, circa 1985
- MasPar MP-1 and MP-2, circa 1987-1996
- Zephyr DC computer from Wavetracer, circa 1991
- Xplor, from Pyxsys, Inc., circa 2001.

There were many others from that era too.

Hardware

Small-scale (64 or 128 bits) SIMD has become popular on general-purpose CPUs in the early 1990s and continuing through 1997 and later with Motion Video Instructions (MVI) for Alpha. SIMD instructions can be found, to one degree or another, on most CPUs, including the IBM's AltiVec and SPE for PowerPC, HP's PA-RISC Multimedia Acceleration eXtensions (MAX), Intel's MMX and iwMMXt, SSE, SSE2, SSE3 and SSSE3, AMD's 3DNow!, ARC's ARC Video subsystem, SPARC's VIS and VIS2, Sun's

MAJC, ARM's NEON technology, MIPS' MDMX (MaDMaX) and MIPS-3D. The IBM, Sony, Toshiba co-developed Cell Processor's SPU's instruction set is heavily SIMD based. NXP founded by Philips developed several SIMD processors named Xetal. The Xetal has 320 16bit processor elements especially designed for vision tasks.

Modern graphics processing units (GPUs) are often wide SIMD implementations, capable of branches, loads, and stores on 128 or 256 bits at a time.

Future processors promise greater SIMD capability: Intel's AVX instructions will process 256 bits of data at once, and Intel's Larrabee graphic microarchitecture promises two 512-bit SIMD registers on each of its cores (VPU - Wide Vector Processing Units) [although as of early 2010, the Larrabee project was canceled at Intel].

Software

SIMD instructions are widely used to process 3D graphics, although modern graphics cards with embedded SIMD have largely taken over this task from the CPU. Some systems also include permute functions that re-pack elements inside vectors, making them particularly useful for data processing and compression. They are also used in cryptography. The trend of general-purpose computing on GPUs (GPGPU) may lead to wider use of SIMD in the future.

Adoption of SIMD systems in personal computer software was at first slow, due to a number of problems. One was that many of the early SIMD instruction sets tended to slow overall performance of the system due to the re-use of existing floating point registers. Other systems, like MMX and 3DNow!, offered support for data types that were not interesting to a wide audience and had expensive context switching instructions to switch between using the FPU and MMX registers. Compilers also often lacked support requiring programmers to resort to assembly language coding.

SIMD on x86 had a slow start. The introduction of 3DNow! by AMD and SSE by Intel confused matters somewhat, but today the system seems to have settled down (after AMD adopted SSE) and newer compilers should result in more SIMD-enabled software. Intel and AMD now both provide optimized math libraries that use SIMD instructions, and open source alternatives like libSIMD and SIMDx86 have started to appear.

Apple Computer had somewhat more success, even though they entered the SIMD market later than the rest. AltiVec offered a rich system and can be programmed using increasingly sophisticated compilers from Motorola, IBM and GNU, therefore assembly language programming is rarely needed. Additionally, many of the systems that would benefit from SIMD were supplied by Apple itself, for example iTunes and QuickTime. However, in 2006, Apple computers moved to Intel x86 processors. Apple's APIs and development tools (XCode) were rewritten to use SSE2 and SSE3 instead of AltiVec. Apple was the dominant purchaser of PowerPC chips from IBM and Freescale Semiconductor and even though they abandoned the platform, further development of AltiVec is continued in several Power Architecture designs from Freescale, IBM.

SIMD within a register, or SWAR, is a range of techniques and tricks used for performing SIMD in general-purpose registers on hardware that doesn't provide any direct support for SIMD instructions. This can be used to exploit parallelism in certain algorithms even on hardware that does not support SIMD directly.

Commercial applications

Though it has generally proven difficult to find sustainable commercial applications for SIMD-only processors, one that has had some measure of success is the GAPP, which was developed by Lockheed Martin and taken to the commercial sector by their spin-off Teranex. The GAPP's recent incarnations have become a powerful tool in real-time video processing applications like conversion between various video standards and frame rates (NTSC to/from PAL, NTSC to/from HDTV formats, etc.), deinterlacing, image noise reduction, adaptive video compression, and image enhancement.

A more ubiquitous application for SIMD is found in video games: nearly every modern video game console since 1998 has incorporated a SIMD processor somewhere in its architecture. The PlayStation 2 was unusual in that its vector-float units could function as autonomous DSPs executing their own instruction streams, or as coprocessors driven by ordinary CPU instructions. 3D graphics applications tend to lend themselves well to SIMD processing as they rely heavily on operations with 4-dimensional vectors. Microsoft's Direct3D 9.0 now chooses at runtime processor-specific implementations of its own math operations, including the use of SIMD-capable instructions.

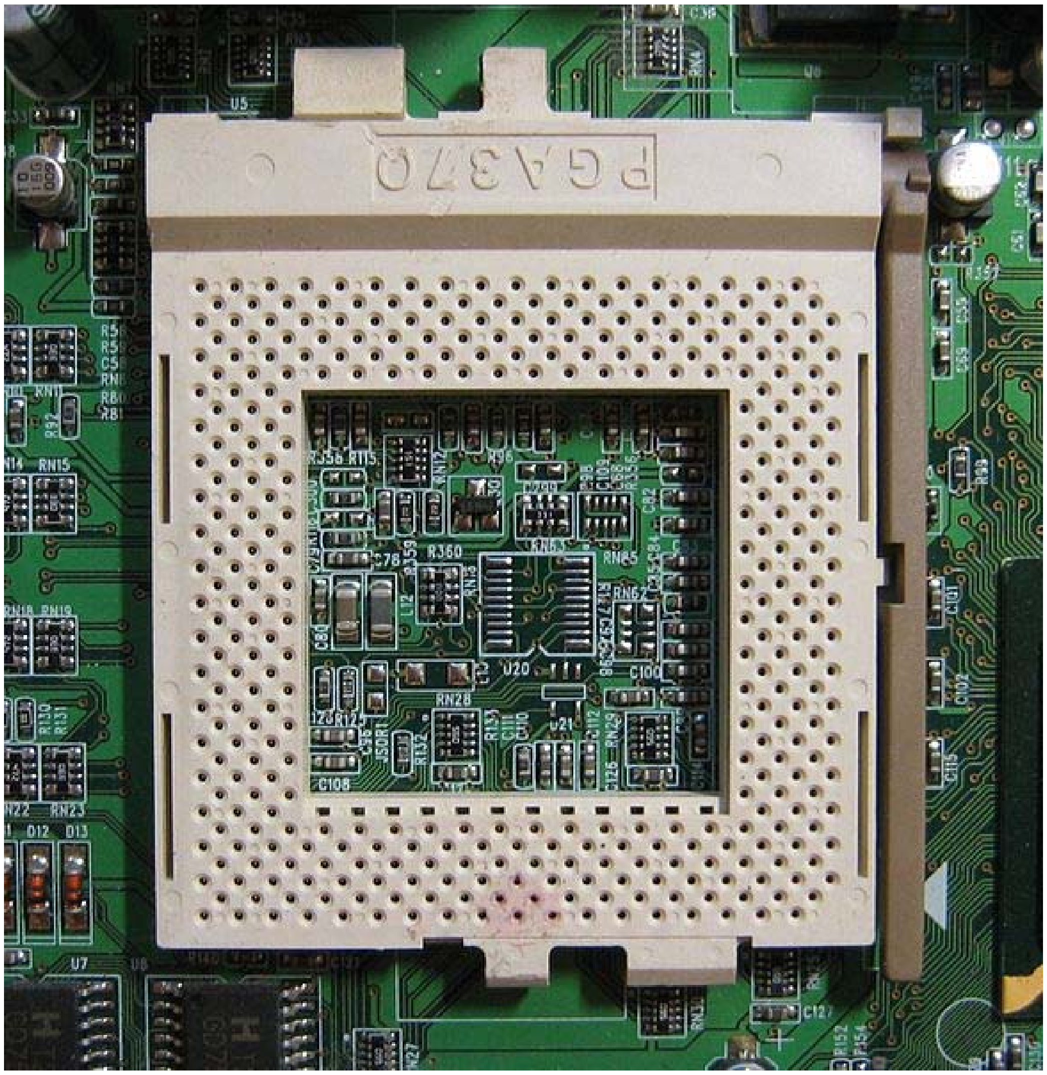
One of the recent processors to use vector processing is the Cell Processor developed by IBM in cooperation with Toshiba and Sony. It uses a number of SIMD processors (each with independent RAM and controlled by a general purpose CPU) and is geared towards the huge datasets required by 3D and video processing applications.

A recent advancement by Ziilabs was the production of an SIMD type processor which can be used on mobile devices, such as media players and mobile phones.

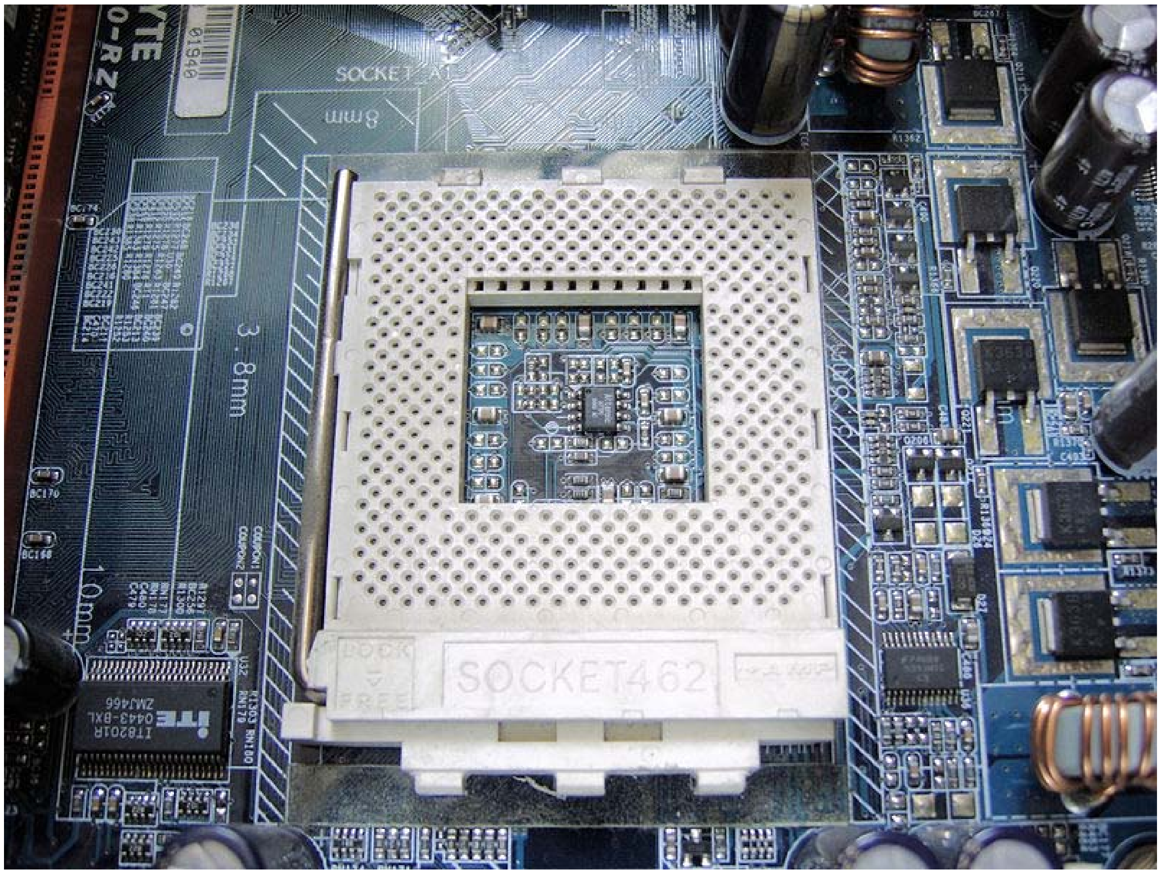
Larger scale commercial SIMD processors are available from ClearSpeed Technology, Ltd. and Stream Processors, Inc. ClearSpeed's CSX600 (2004) has 96 cores each with 2 double-precision floating point units while the CSX700 (2008) has 192. Stream Processors is headed by computer architect Bill Dally. Their Storm-1 processor (2007) contains 80 SIMD cores controlled by a MIPS CPU.

Chapter 6

CPU Socket



The Socket 370 processor socket, a ZIF type PGA socket



Socket A (also known as Socket 462)

A **CPU socket** or **CPU slot** is a mechanical component that provides mechanical and electrical connections between a device (usually a microprocessor) and a printed circuit board (PCB). This allows the CPU to be replaced without risking the damage typically introduced when using soldering tools.

Common sockets utilize retention clips that are designed to apply a constant force, which must be overcome when a device is inserted. For chips that sport a high number of pinouts, either zero-insertion force (ZIF) sockets or land grid array (LGA) sockets are used instead. These designs apply a compression force once either a handle (for ZIF type) or a surface plate (LGA type) is put into place. This provides superior mechanical retention while avoiding the added risk of bending pins when inserting the chip into the socket.

CPU sockets are used in desktop and server computers (laptops typically use surface mount CPUs) because they allow easy swapping of components, they are also used for prototyping new circuits.

Function

A CPU socket is often made up of plastic, a metal lever or latch, and metal contacts for each of the pins or lands on the CPU. Most packages are keyed to ensure the proper insertion of the CPU. CPUs with a PGA package are inserted into the socket and the latch is closed.

List of sockets and slots

Socket name	CPU families	Package	Pin count	Pin pitch	Bus speed	Notes
DIP	Intel 8086 Intel 8088	DIP	40	2.54mm	5/10 MHz	
PLCC	Intel 80186 Intel 80286 Intel 80386	PLCC	68, 132	1.27mm	6-40 MHz	
Socket 1	Intel 80486	PGA	169		16-50 MHz	
Socket 2	Intel 80486	PGA	238		16-50 MHz	
Socket 3	Intel 80486	PGA	237		16-50 MHz	
Socket 4	Intel Pentium	PGA	273		60-66 MHz	
Socket 5	Intel Pentium AMD K5 IDT WinChip C6 IDT WinChip 2	PGA	320		50-66 MHz	
Socket 6	Intel 80486	PGA	235			
Socket 7	Intel Pentium Intel Pentium MMX AMD K6	PGA	321		50-66 MHz	
Super Socket 7	AMD K6-2 AMD AMD K6-III Rise mP6 Cyrix MII	PGA	321		66-100 MHz	
Socket 8	Intel Pentium Pro	PGA	387		60-66 MHz	
Slot 1	Intel	Slot	242		66-133 MHz	Celeron

	Pentium II Intel Pentium III					(Covington, Mendocino) Pentium II (Klamath) Pentium III (Katmai)- all versions Pentium III (coppermine)
Slot 2	Intel Pentium II Xeon	Slot	330		100-133 MHz	
Socket 463/ Socket NexGen	NexGen Nx586	PGA	463			
Socket 499	Alpha 21164A	Slot	587			
Slot A	AMD Athlon	Slot	242		100 MHz	
Slot B	Alpha 21264	Slot	587			
Socket 370	Intel Pentium III Intel Celeron VIA Cyrix III VIA C3	PGA	370	1.27mm	66-133 MHz	
Socket 462/ Socket A	AMD Athlon AMD Duron AMD Athlon XP AMD Athlon XP-M AMD Athlon MP AMD Sempron	PGA	462		100-200 MHz This is a double data rate bus having a 400 MT/s (megatransfers/second) fsb in the later models	
Socket 423	Intel Pentium 4	PGA	423	1mm	400 MT/s (100 MHz)	Willamette core only
Socket 478/ Socket N	Intel Pentium 4 Intel Celeron Intel	PGA	478	1.27mm	400-800 MT/s (100-200 MHz)	

	Pentium 4 EE Intel Pentium 4 M					
Socket 495	Intel Celeron	PGA	495	1.27mm		
PAC418	Intel Itanium	PGA	418		133 MHz	
Socket 603	Intel Xeon	PGA	603	1.27mm	400-533 MT/s (100-133 MHz)	
PAC611	Intel Itanium 2 HP PA-8800, PA-8900	PGA	611			
Socket 604	Intel Xeon	PGA	604	1.27mm	400-1066 MT/s (100-266 MHz)	
Socket 754	AMD Athlon 64 AMD Sempron AMD Turion 64	PGA	754	1.27mm	200-800 MHz	
Socket 940	AMD Opteron Athlon 64 FX	PGA	940	1.27mm	200-1000 MHz	
Socket 479	Intel Pentium M Intel Celeron M	PGA	479		400-533 MT/s (100-133 MHz)	
Socket 939	AMD Athlon 64 AMD Athlon 64 FX AMD Athlon 64 X2 AMD Opteron	PGA	939	1.27mm	200-1000 MHz	Support of Athlon 64 FX to 1 GHz Support of Opteron limited to 100-series only
LGA 775/ Socket T	Intel Pentium 4 Intel Pentium D Intel Celeron	LGA	775	1.09mm x 1.17mm	1600 MHz	

	Intel Celeron D Intel Pentium XE Intel Core 2 Duo Intel Core 2 Quad Intel Xeon					
Socket 563	AMD Athlon XP-M	PGA	563			
Socket M	Intel Core Solo Intel Core Duo Intel Dual-Core Xeon Intel Core 2 Duo	PGA	478		533 - 667 MT/s (133-166 MHz)	For notebook platform Replaces Socket 479
LGA 771/ Socket J	Intel Xeon	LGA	771	1.09mm x 1.17mm	1600 MHz	
Socket S1	AMD Turion 64 X2	PGA	638	1.27mm	200-800 MHz	
Socket AM2	AMD Athlon 64 AMD Athlon 64 X2	PGA	940	1.27mm	200-1000 MHz	Replaces Socket 754 and Socket 939
Socket F	AMD Athlon 64 FX AMD Opteron	LGA	1207	1.1mm		Replaces Socket 940
Socket AM2+	AMD Athlon 64 AMD Athlon X2 AMD Phenom AMD Phenom II	PGA	940	1.27mm	200-2600 MHz	Separated power planes Replaces Socket AM2 AM2+ Pkg. CPUs can work in Socket AM2 AM2 Pkg. CPUs can work in Socket AM2+

Socket P	Intel Core 2	PGA	478		533-1066 MT/s (133-266 MHz)	For notebook platform Replaces Socket M
Socket 441	Intel Atom	PGA	441		400-667 MHz	
LGA 1366/ Socket B	Intel Core i7 (900 series) Intel Xeon (35xx, 36xx, 55xx, 56xx series)	LGA	1366		4.8-6.4 GT/s	Replaces server-oriented Socket J (LGA 771) in the entry level.
Socket AM3	AMD Phenom II AMD Athlon II AMD Sempron	PGA	941	1.27mm	200-3200 MHz	Separated power planes Replaces Socket AM2+ AM3 Pkg. CPUs can work in Socket AM2/AM2+ Sempron 140 Only
LGA 1156/ Socket H	Intel Core i7 (800 series) Intel Core i5 (700, 600 series) Intel Core i3 (500 series) Intel Xeon (X3400, L3400 series) Intel Pentium (G6000 series) Intel Celeron (G1000 series)	LGA	1156		2.5 GT/s	DMI bus is a (perhaps modified) PCI-E x4 v1.1 interface
Socket G34	AMD Opteron (6000	LGA	1974		200-3200 MHz	Replaces Socket F

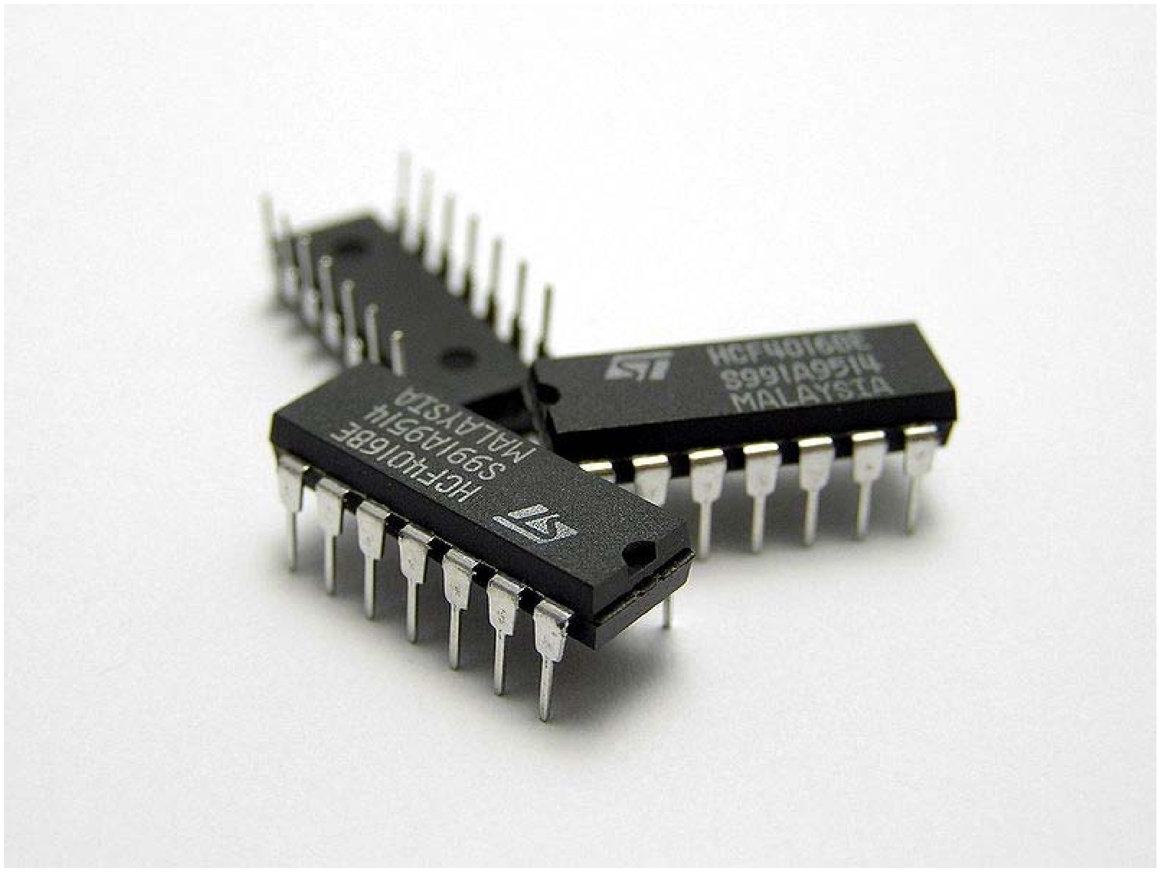
Socket name	CPU families	Package	Pin count	Pin pitch	Bus speed	Notes
	series)					
Socket C32	AMD Opteron (4000 series)	LGA	1207		200-3200 MHz	Replaces Socket F, Socket AM3
LGA 1248	Intel Intel Itanium 9300-series	LGA	1248		4.8 GT/s	
LGA 1567	Intel Intel Xeon 6500/7500-series	LGA	1567		4.8-6.4 GT/s	
LGA 1155/ Socket H2	Intel Sandy Bridge-DT	LGA	1155		5 GT/s	Supports 20 PCI-E 2.0 lanes.
LGA 2011/ Socket R	Intel Sandy Bridge B2	LGA	2011		4.8-6.4 GT/s	Supports 40 PCI-E 3.0 lanes.

Slotkets

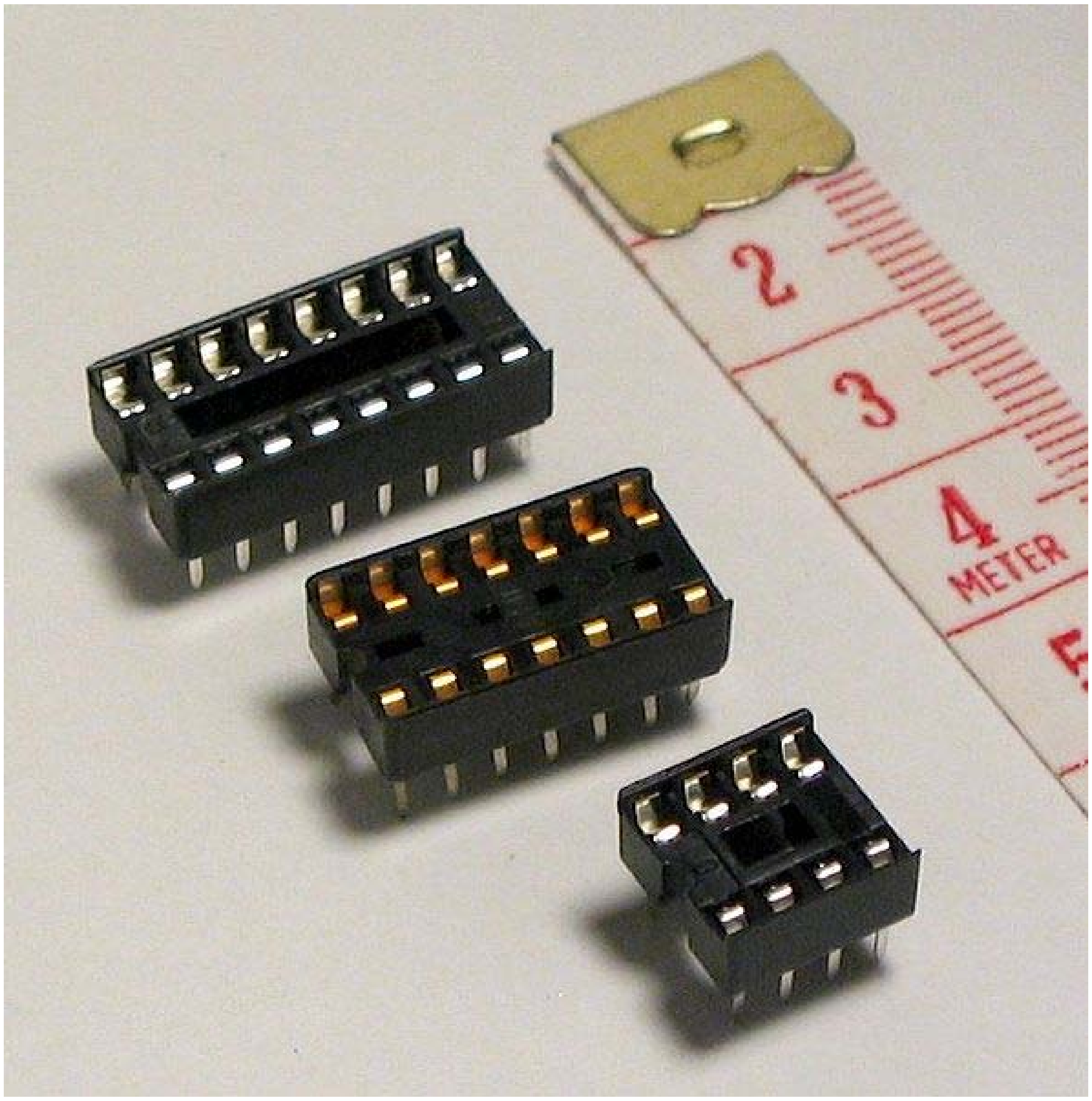
Slotkets are special adapters for using socket processors in bus-compatible slot motherboards.

Chapter 7

Dual in-line Package



Three 14-pin (DIP14) plastic dual in-line packages containing IC chips

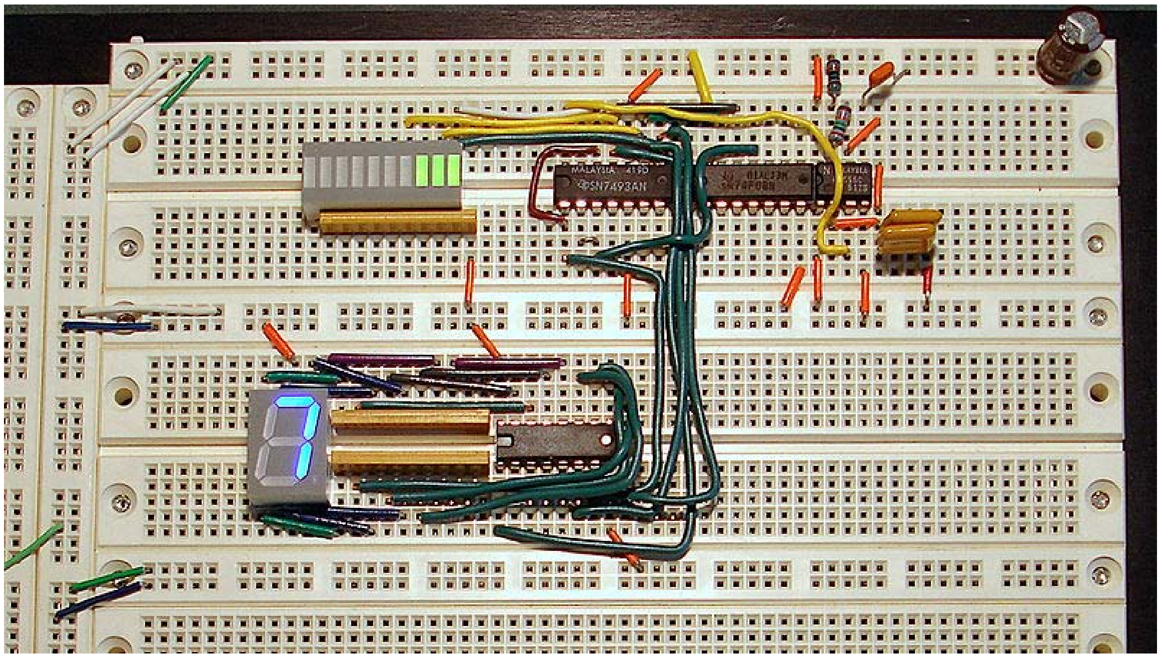


Sockets for 16-, 14-, and 8-pin packages

In microelectronics, a **dual in-line package (DIP)**, sometimes called a **DIL**-package (for **D**ual **I**n **L**ine-package), is an electronic device package with a rectangular housing and two parallel rows of electrical connecting pins. The pins are all parallel, point downward, and extend past the bottom plane of the package at least enough to be through-hole mounted to a printed circuit board (PCB), i.e. to pass through holes on the PCB and be soldered on the other side. DIP is sometimes incorrectly considered to stand for dual in-line pin, in an effort to justify the redundant term "DIP package" (Dual in-line pin would imply one line of two pins). Generally, a DIP is relatively broadly defined as any rectangular package with two uniformly spaced parallel rows of pins pointing downward, whether it contains an IC chip or some other device(s), and whether the pins emerge from the sides of the package and bend downwards or emerge directly from the bottom of the package and are completely straight. In more specific usage, the term refers only to an IC package of the former description (with bent leads at the sides.) A DIP is usually referred to as a **DIPn**, where n is the total number of pins. For example, a microcircuit package with two rows of seven vertical leads would be a DIP14. The photograph at the upper right shows three DIP14 ICs.

Applications

Types of devices



An operating prototyped circuit incorporating four DIP ICs, a DIP LED bargraph display (upper left), and a DIP 7-segment LED display (lower left).

DIPs may be used for semiconductor integrated circuits (ICs, "chips"), like logic gates, analog circuits, and microprocessors, which is by far their most common use. They may also be used for other types of devices including arrays of discrete components such as resistors (often called resistor packs), arrays of miniature rocker or slide switches known as DIP switches, various LED arrays including segmented and bargraph displays and light bars, miniature rotary encoder switches, and electromechanical relays. Integrated circuits and resistor arrays usually have bent leads (leads are one type of IC package connector; other types are pins, and more recently balls) which extend from the sides of the package and turn to point downward; the IC packages tend to be black, and DIP resistor networks tend to be dark yellow or white plastic. The other types of DIP components, particularly LED devices, usually have completely straight leads extending directly from the bottom/back of the package, which is usually molded plastic and can be any color.

DIP plugs for ribbon cable, to connect to DIP sockets, have also been made (and can be found in some Apple II computers, as well as on some electronics test equipment used by technicians.) Dallas Semiconductor manufactured integrated DIP real-time clock (RTC) modules which contained an IC chip and a non-replaceable 10-year lithium battery. Both of these were examples of devices with leads emerging from the bottom; the construction of the RTC modules was very similar to the LED displays described below, with epoxy in a plastic shell. The RTC modules are an example of a component that could not be easily converted to surface-mount, as their size and weight with the contained battery would push the limits of mechanical strength of surface solder pads, and the thicker leads of a DIP as compared to most SMT packages were probably necessary to mechanically support the battery. Dallas also sold the same RTC chips without a battery in standard side-lead DIP packages; both versions would physically fit the same DIP sockets.

Some older equipment (circa the 1970s) used DIP terminal blocks onto the top of which discrete components could be soldered; these blocks (typically DIP16 size) would then be plugged into sockets on circuit boards and could be easily removed and swapped out, or test equipment connected between them and their sockets, for repair or testing of the machines they were a part of. These saw use mainly in industrial, commercial, and prototype equipment, not consumer electronics.

Uses

Dual in-line packages were invented at Fairchild in 1965 and, by allowing integrated circuits to be packaged more densely than previous round packages did, made it possible to build complex systems such as sophisticated computers. The package was well-suited to automated assembly equipment; a printed circuit board could be populated with scores or hundreds of ICs, then have all devices soldered at once (e.g. on a wave soldering machine) and passed on to automated testing machines, with very little human labor required. However, the packages were still large with respect to the integrated circuits within them. By the end of the 20th century, most ICs were packaged in surface-mount packages, which allowed further reduction in the size and weight of systems. DIP chips are still popular for circuit prototyping on a breadboard because of how easily they can be inserted and utilized there.

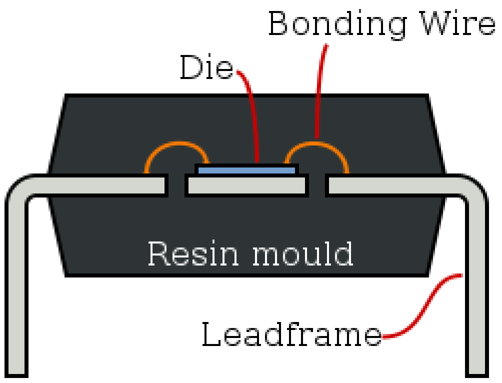
DIPs were the mainstream of the microelectronics industry in the 1970s and 80s. Their use has declined in the first decade of the 21st century due to the emerging new surface-mount technology (SMT) packages such as plastic leaded chip carrier (PLCC) and small-outline integrated circuit (SOIC), though DIPs continued in extensive use through the 1990s, and still continue to be used substantially as the year 2011 passes. Because some modern chips are available only in surface-mount package types, a number of companies sell various prototyping adapters to allow those SMT devices to be used like DIP devices with through-hole breadboards and soldered prototyping boards (such as stripboard and perfboard). (SMT can pose quite a problem, at least an inconvenience, for prototyping in general; most of the characteristics of SMT that are advantages for mass production are difficulties for prototyping.)

For programmable devices like EPROMs and GALs, DIPs remained popular for many years due to their easy handling with external programming circuitry (i.e., the DIP devices could be simply plugged into a socket on the programming device.) However, with In-System Programming (ISP) technology now state of the art, this advantage of DIPs is rapidly losing importance as well. Through the 1990s, devices with lead counts below 20 were manufactured in a DIP format in addition to the newer formats. Since about 2000, newer devices are often unavailable in the DIP format, though many still are.

Mounting

DIPs can be mounted on a circuit board either directly using through-hole soldering or using inexpensive spring-contact sockets. Using a DIP socket allows for easy replacement of a device and eliminates the risk of damage from overheating during soldering (as the device is inserted into the soldered socket only after it has cooled.) These are by far the most common mounting types for DIP components. DIPs can also be easily and conveniently used with standard protoboards/breadboards, whose design includes extensive consideration for them; this is a temporary mounting arrangement for circuit design development or device testing. Some hobbyists, for one-off construction or permanent prototyping, use point-to-point wiring with DIPs, and their appearance when physically inverted as part of this method inspires the informal term "dead bug style" for the method.

Construction



Side view of a dual in-line package (DIP) IC

The body (housing) of a DIP containing an IC chip is usually made from molded plastic or ceramic. The hermetic nature of a ceramic housing is preferred for extremely high reliability devices. However, the vast majority of DIPs are manufactured via a thermoset molding process in which an epoxy mold compound is heated and transferred under pressure to encapsulate the device. Typical cure cycles for the resins are less than 2 minutes and a single cycle may produce hundreds of devices.

The leads emerge from the longer sides of the package along the seam, parallel to the top and bottom planes of the package, and are bent downward approximately 90 degrees (or slightly less, leaving them angled slightly outward from the centerline of the package body.) (The SOIC, the SMT package that most resembles a typical DIP, appears essentially the same, notwithstanding size scale, except that after being bent down the leads are bent upward again by an equal angle to become parallel with the bottom plane of the package.) In ceramic (CERDIP) packages, an epoxy or grout is used to hermetically seal the two halves together, providing an air and moisture tight seal to protect the IC die inside. Plastic DIP (PDIP) packages are usually sealed by fusing or cementing the plastic halves around the leads, but a high degree of hermeticity is not achieved because the plastic itself is usually somewhat porous to moisture and the process cannot ensure a good microscopic seal between the leads and the plastic at all points around the perimeter. However, contaminants are usually still kept out well enough that the device can operate reliably for decades with reasonable care in a controlled environment.

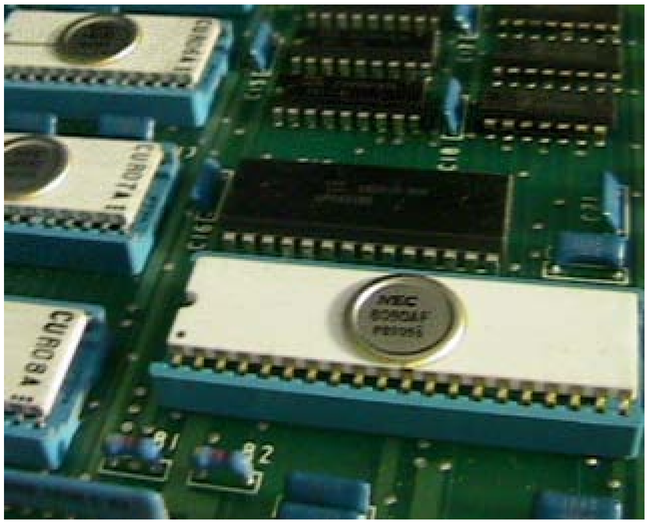
Inside the package, the lower half has the leads embedded, and at the center of the package is a rectangular space, chamber, or void into which the IC die is cemented. The leads of the package extend diagonally inside the package from their positions of emergence along the periphery to points along a rectangular perimeter surrounding the die, tapering as they go to become fine contacts at the die. Ultra-fine bond wires (barely visible to the naked human eye) are welded between these die periphery contacts and bond pads on the die itself, connecting one lead to each bond pad, and making the final connection between the microcircuits and the external DIP leads. The bond wires are not usually taut but loop upward slightly to allow slack for thermal expansion and contraction of the materials; if a single bond wire breaks or detaches, the entire IC may become useless. The top of the package covers all of this delicate assemblage without crushing the bond wires, protecting it from contamination by foreign materials. Semiconductor materials must be ultra pure, and doping (the controlled addition of impurities) must be very precise, to make a working device, so the slightest chemical contamination must be avoided. Usually, a company logo, alphanumeric codes and sometimes words are printed on top of the package to identify its manufacturer and type, when it was made (usually as a year and a week number), sometimes where it was made, and other proprietary information (perhaps revision numbers, manufacturing plant codes, or stepping ID codes.) (In contrast, most SMT packages, because of their micro-miniature size, bear very little printed information, usually just four- or eight-character codes identifying their functional type.)

The necessity of laying out all of the leads in a basically radial pattern in a single plane from the die perimeter to two rows on the periphery of the package is the main reason that DIP packages with higher lead counts must have wider spacing between the lead rows, and it effectively limits the number of leads which a practical DIP package may have. Even for a very small die with many bond pads (e.g. a chip with 15 inverters, requiring 32 leads), a wider DIP would still be required to accommodate the radiating leads internally. This is one of the reasons that four-sided and multiple rowed packages, such as PGAs, were introduced (around the early 1980s.)

Also, a large DIP package (such as the DIP64 used for the Motorola 68000 CPU) incorporates substantial lead length inside the package between the leads located toward the ends of the package and the chip die at its center, making such a package unsuitable for high-speed microarchitectures (above a few hundred megahertz), which require lead length to be kept to a minimum. The 68000 ran at no more than 20 MHz, so this was not an issue for it.

Some other types of DIP devices are built very differently. Most of these have molded plastic housings and straight leads or leads that extend directly out of the bottom of the package. For some, LED displays particularly, the housing is usually a hollow plastic box with the bottom/back open, filled (around the contained electronic components) with a hard translucent epoxy material from which the leads emerge. Others, such as DIP switches, are composed of two (or more) plastic housing parts snapped, welded, or glued together around a set of contacts and tiny mechanical parts, with the leads emerging through molded-in holes or notches in the plastic.

Variants



Several PDIPs and CERDIPs. The large CERDIP in the foreground is an Intel 8080 microprocessor.

Several DIP variants for ICs exist, mostly distinguished by packaging material:

- **Ceramic Dual In-line Package (CERDIP or CDIP)**
- **Plastic Dual In-line Package (PDIP)**
- **Shrink Plastic Dual In-line Package (SPDIP)** – A denser version of the PDIP with a 0.07 in. (1.778 mm) lead pitch.
- **Skinny Dual In-line Package (SDIP or SPDIP)** – Sometimes used to refer to a 0.3 in. wide DIP, normally when clarification is needed e.g. for a 24 or 28 pin DIP.

For EPROMs, which can be erased by UV light, some DIPs, generally ceramic CERDIPs, were manufactured with a circular window of clear quartz in the center of the top of the package, over the chip die. This enabled the packaged chips to be erased by UV irradiation in an EPROM eraser. Often, the same chips were also sold in less expensive windowless PDIP or CERDIP packages as one-time programmable (OTP) versions. (These were actually the same erasable chips, but there was no way to get UV radiation to them to erase them.) The same windowed and windowless packages were also used for microcontrollers, and perhaps other devices, containing EPROM memory; in this context, the OTP nature of the windowless versions was sometimes a needed requirement of the customer (i.e., to prevent their end users from modifying the stored information, which might include access control bits to disable read-out of proprietary code or factory test modes which were disabled after final test qualification.) Windowed CERDIP-packaged PROMs were used for the BIOS ROM of many early IBM PC clones (which were manufactured in limited enough quantities to make PROM an economical choice) often with a foil-backed (or regular paper) adhesive label covering the window to prevent inadvertent erasure through exposure to ambient light.

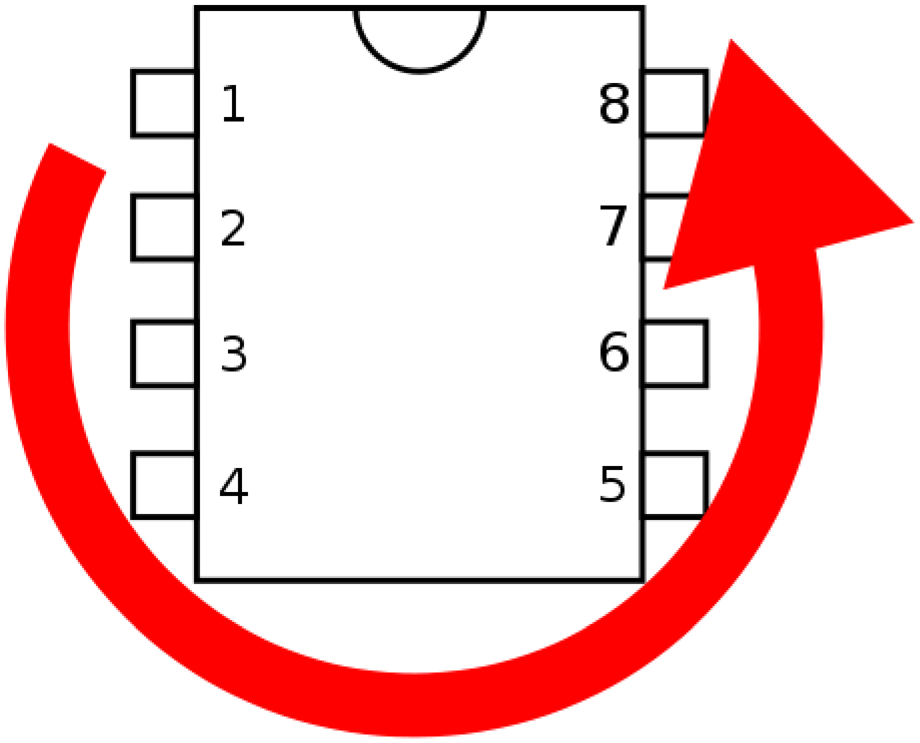
Lead count and spacing

Commonly found DIP packages that conform to JEDEC standards use an inter-lead spacing (lead pitch) of 0.1 inch (2.54 mm). Row spacing varies depending on lead counts, with 0.3 in. (7.62 mm) or 0.6 inch (15.24 mm) the most common. Less common standardized row spacings include 0.4 inch (10.16 mm) and 0.9 inch (22.86 mm), as well as a row spacing of 0.3 inch, 0.6 inch or 0.75 inch with a 0.07 inch (1.778 mm) lead pitch.

The former Soviet Union and Eastern bloc countries used similar packages, but with a metric inter-lead spacing of 2.5 mm rather than 2.54 mm (0.1 inch).

The number of leads is always even. For 0.3 inch spacing, typical lead counts are 8 to 24; less common are 4 or 28 lead counts. For 0.6 inch spacing, typical lead counts are 24, 28, 32 or 40; less common are 36, 48 or 52 lead counts. Some microprocessors, such as the Motorola 68000 and Zilog Z180, used lead counts as high as 64; this is typically the maximum number of leads for a DIP package.

Orientation and lead numbering



Pin numbering is counter-clockwise

When a DIP is viewed from the top with the in-line lead rows horizontal and the orientation indicator on the left side, the leads are sequentially numbered counterclockwise, with lead numbers increasing from left to right across the bottom edge and from right to left across the top edge. Finding any given lead is simply a matter of finding lead 1 and then counting counterclockwise. Because the dual-inline lead arrangement has radial symmetry, the layout looks the same if the package is rotated 180 degrees; therefore, to resolve this ambiguity, one end of every DIP is marked with an orientation notch, a dot, or both. The notch is centered between the lead rows and may extend all the way through the end of the package or may be just cut into the top of the end, but it is almost always present. The dot will be on top of the corner of the package at the same end as the notch, if both are present, and it will usually be a molded depression, though it may be raised or even, rarely (except for ceramic packages), merely a printed mark. When the package is viewed from the top side, lead 1 is the lead at the corner with the dot, or it is the lead counterclockwise from the notch (that is, counterclockwise around the center of the package). Lead 1 is always in the same inline row as lead 2.

Merely knowing that lead 1 is a corner lead on the notched or dotted end, that leads 1 and 2 are always adjacent in the same inline row of leads, and that the lead numbering is always counterclockwise is sufficient to figure out the numbers of all the leads for any DIP package. Another way to remember the numbering, also sufficient for all DIPs, is that lead 1 and the highest numbered leads are the two corner leads at the notched and/or dotted end, and that the lead numbering is counterclockwise. By either of these rule sets, if lead 1 is misidentified at the wrong corner of the marked end, it is impossible to number the leads counterclockwise with increasing numbers.

Using basic math and logic, it can be concluded that for a package with any number of leads n (where n is always even, of course), if lead 1 is at the lower left corner (which implies that the lead rows are horizontal), then the bottom row is numbered 1 to n / 2 from left to right, and the top row is numbered n / 2 + 1 to n from right to left. So, for example, for a 14-lead DIP, n = 14 and n / 2 = 7; with the notch at the left, the bottom row leads are numbered from 1 to 7 (left to right) and the top row leads are numbered 8 to 14 (right to left). The diagram at the right above shows the DIP package rotated 90 degrees clockwise relative to this description (with lead 1 at the upper left instead of the lower left and the lines of leads vertical columns instead of horizontal rows), so the left column is numbered 1 to n / 2 from top to bottom, and the right column is numbered n / 2 + 1 to n from bottom to top.

Some DIP devices, such as segmented LED displays, have some lead positions skipped (i.e. leads omitted). In that case, often the existing leads will still be numbered according to the corresponding positions on a standard DIP that has no gaps in the lead rows, so the numbers of the present leads will not be contiguous. This makes it easier for experienced engineers and technicians to identify leads on these devices using their knowledge of DIP lead numbering, though it may be initially confusing for amateur hobbyists, who may expect the physical leads of every device to be numbered sequentially. This highlights an important point: when naming a package according to the number of leads, e.g. DIP14, the number should be chosen not according to the number of actual leads but according to the number of lead positions on the package, or, in other words, according to the size of the standard DIP socket that the package will fit into. So, a 7-segment LED display with ten leads arranged in two rows 7 x 0.1 in. long and 0.3 in. apart (which leaves two missing lead-spaces on each side) would fit a DIP14 socket, but not a DIP12 or DIP10 socket, and therefore it should be identified as a DIP14; it has ten leads but 14 lead positions.

In addition to providing for human visual identification of the orientation of the package, the notch allows automated chip-insertion machinery to ensure correct orientation of the chip by mechanical sensing. A physical notch is also more durable than a printed mark, as it cannot wear off or fade, and, no less, it gains this benefit at a small saving of some material.

Descendants

The SOIC (Small Outline IC), a surface-mount package which is currently very popular (in 2009), particularly in consumer electronics and personal computers, is essentially a shrunk version of the standard IC PDIP, the fundamental difference which makes it an SMT device being a second bend in the leads to flatten them parallel to the bottom plane of the plastic housing. The SOJ (Small Outline J-lead) and other SMT packages with

"SOP" (for "Small Outline Package") in their names can be considered further relatives of the DIP, their original ancestor.

Pin grid array (PGA) packages may be considered to have evolved from the DIP. PGAs with the same 0.1 inch pin centers as most DIPs were popular for microprocessors from the early-mid 1980s through the 1990s. Owners of personal computers containing Intel 80286 through P5 Pentium processors may be most familiar with these PGA packages, which were often inserted into ZIF sockets on motherboards. The similarity is such that a PGA socket may be physically compatible with some DIP devices, though the converse is rarely true.

History

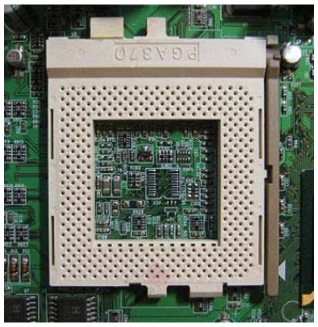
The original Dual-in-line package was invented by Bryant "Buck" Rogers in 1964 while working for Fairchild Semiconductor. The first devices had 14 pins and looked much like they do today.

Chapter 8

Socket 370 and Socket 478

Socket 370

Socket 370



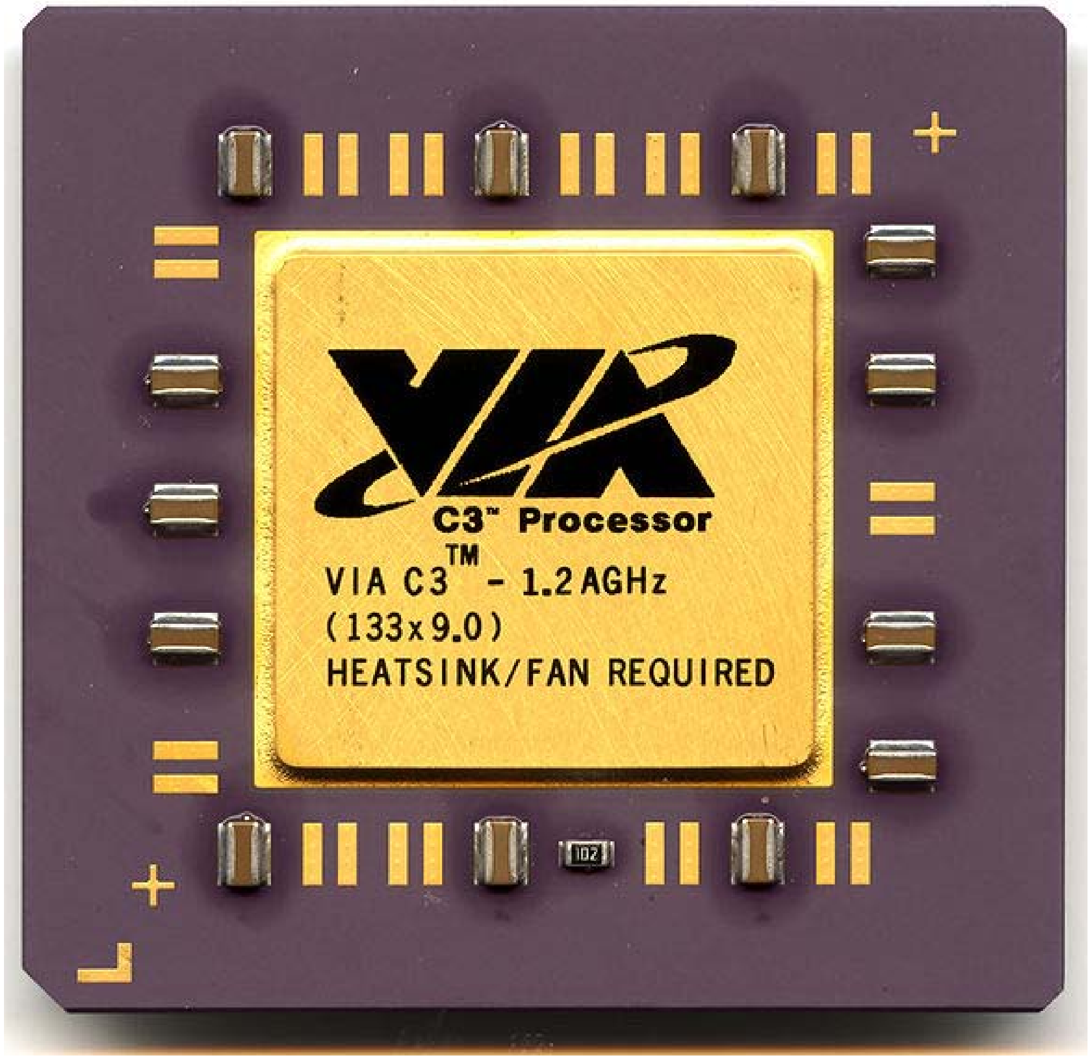
Type	PGA-ZIF
Chip form factors	Plastic pin grid array (PPGA) and Flip-chip pin grid array (FC-PGA and FC-PGA2)
Contacts	370
FSB protocol	GTL+
FSB frequency	66, 100 and 133 MHz
Voltage range	1.05–2.1 V
Processor dimensions	1.95 × 1.95 inches
Processors	Intel Celeron Mendocino (PPGA, 300–533 MHz, 2.0 V) Intel Celeron Coppermine (FC-PGA, 533–1100 MHz, 1.5–1.75 V) Intel Celeron Tualatin (FC-PGA2, 900–1400 MHz, 1.475–1.5 V) Intel Pentium III Coppermine (FC-PGA, 500–1133 MHz, 1.6–1.75 V)

Intel Pentium III Tualatin (FC-PGA2, 1000–1400 MHz, 1.45–1.5 V)

VIA Cyrix III/C3 (500–1200 MHz, 1.35–2.0 V)

Socket 370 (also known as the **PGA370 socket**) is a common format of CPU socket first used by Intel for Pentium III and Celeron processors to replace the older **Slot 1** CPU interface on personal computers. The "370" refers to the number of pin holes in the socket for CPU pins. Modern Socket 370 fittings are usually found on Mini-ITX motherboards and embedded systems.

Technical specifications



Microprocessor VIA C3 1.2 GHz Nehemiah C5XL CPGA socket-370

Socket 370 was originally used for the Intel Celeron, but later became the socket/platform for the Coppermine and Tualatin Pentium III processors, as well as the Via-Cyrix Cyrix III, later renamed the VIA C3. Some motherboards that used Socket 370 support Intel processors in dual CPU configurations. Others allowed the use of a Socket 370 or Slot 1 CPU, although not at the same time.

The weight of a Socket 370 CPU cooler should not exceed 180 grams (6.3 ounces). Heavier coolers may result in damage to the die when the system is not properly handled.

Socket 370 Intel processors mechanical load limits

Most Intel Socket 370 processors (Pentium III and Celeron) have the following mechanical maximum load limits which should not be exceeded during heat sink assembly, shipping conditions, or standard use. Load above those limits will crack the processor die and make it unusable.

Location	Dynamic	Static
Die Surface	890 N (200 lb_f)	222 N (50 lb_f)
Die Edge	667 N (100 lb_f)	53 N (12 lb_f)

Those load limits are quite small compared to the load limits of Socket 478 processors. Indeed they were so small that many users ended up with cracked processors while trying to remove or attach a heatsink to their processor. On the other hand, Socket A processors had even smaller load limits.

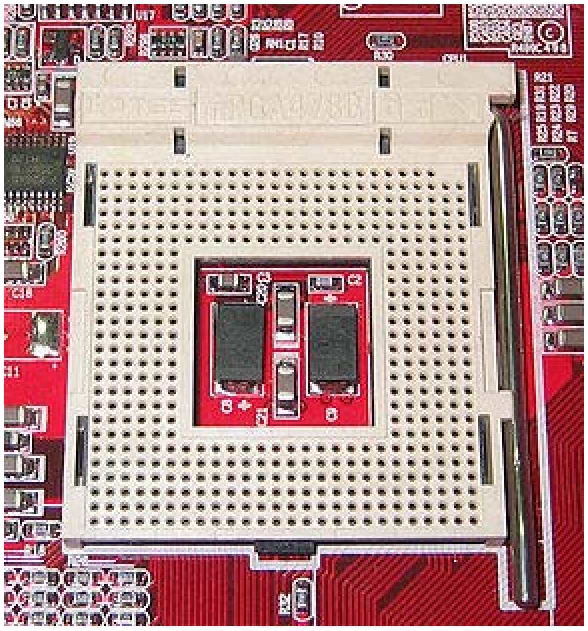
Socket 370 Intel processors with IHS mechanical load limits

All Intel Socket 370 processors with IHS (Pentium III and Celeron 1.13–1.4 GHz) have the following mechanical maximum load limits which should not be exceeded during heatsink assembly, shipping conditions, or standard use. Load above those limits will crack the processor die and make it unusable.

Location	Dynamic	Static
IHS Surface	890 N (200 lb_f)	667 N (100 lb_f)
IHS Edge	556 N (125 lb_f)	N/A
IHS Corner	334 N (75 lb_f)	N/A

Socket 478

Socket 478



Type	PGA-ZIF
Chip form factors	Flip-chip pin grid array (FC-PGA2 or FC-PGA4)
Contacts	478 (not to be confused with the new Socket P that also uses 478-pins)
FSB protocol	AGTL+
FSB frequency	400 MT/s 533 MT/s 800 MT/s
Processor dimensions	1.38 × 1.38 inches
Processors	Intel Pentium 4 (1.4 - 3.4 GHz) Intel Celeron (1.7 - 2.8 GHz) Celeron D (2.13 - 3.2 GHz) Intel Pentium 4 Extreme Edition (3.2, 3.4 GHz)

Socket 478 is a 478-contact CPU socket used for Intel's Pentium 4 and Celeron series CPUs.

Socket 478 was launched with the Northwood core to compete with AMD's 462-pin Socket A and their Athlon XP processors. Socket 478 was intended to be the replacement for Socket 423, a Willamette-based processor socket which was on the market for only a short time. Socket 478 was phased out with the launch of LGA 775.

Technical specifications

Socket 478 was used for all Northwood Pentium 4 and Celeron processors. It supported the first Prescott Pentium 4 processors and all Willamette Celerons, along with several of the Willamette-series Pentium 4s. Socket 478 also supported the newer Prescott-based Celeron D processors, and early Pentium 4 Extreme Edition processors with 2 MiB of L3 cache.

Celeron D processors were also available for Socket 478 and were the last CPUs made for the socket.

While the Intel mobile CPUs are available in 478-pin packages, they in fact only operate in a range of slightly differing sockets, Socket 479, Socket M, and Socket P, each incompatible with the other two.

Mechanical load limits

All socket (Pentium 4 and Celeron) have the following mechanical maximum load limits which should not be exceeded during heatsink assembly, shipping conditions, or standard use. Load above those limits will crack the processor die and make it unusable.

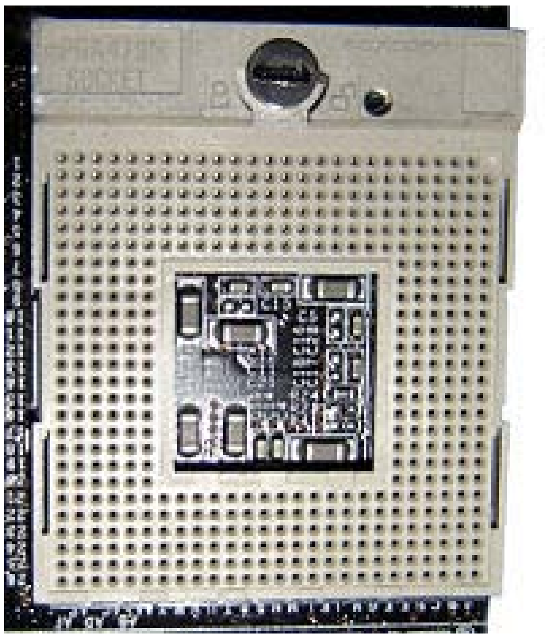
Location	**Dynamic**	**Static**	**Transient**
IHS Surface	890 N(200 lb_f)	445 N(100 lb_f)	667 N(150 lb_f)

Chapter 9

Socket 479 and Zero Insertion Force

Socket 479

Socket 479



Type	PGA-ZIF
Chip form factors	Flip-chip pin grid array (FC-PGA2)
Contacts	479 on the socket, 478 on the processor
FSB protocol	AGTL+
FSB frequency	400 MT/s, 533 MT/s
Processors	Intel Pentium M Intel Celeron M VIA C7-M

Socket 479 is the CPU socket for the Intel Pentium M and Celeron M, mobile processors. Normally used in laptops, but has also been used with Tualatin-M Pentium III processors. The official naming by Intel is mFCPGA and mPGA479M. Despite the fact that Socket 479 has 479 pin holes, the Pentium M Processors for this socket have only 478 pins.

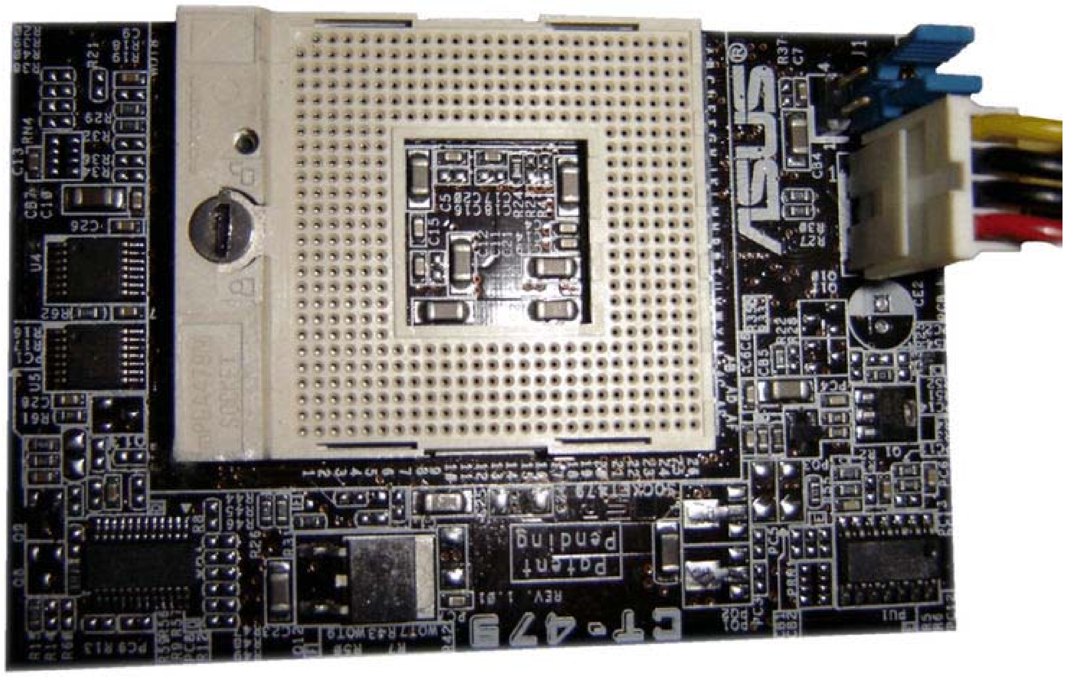
There existed three electrically incompatible, but physically identical, versions of this socket:

- for Pentium III-M (released in 2001);
- for Pentium M and Celeron M 3xx (this was the most common version of the socket, and was released in 2003); and
- a version compatible with Socket M for Intel Core, Core 2 and Celeron M 4xx and 5xx processors.

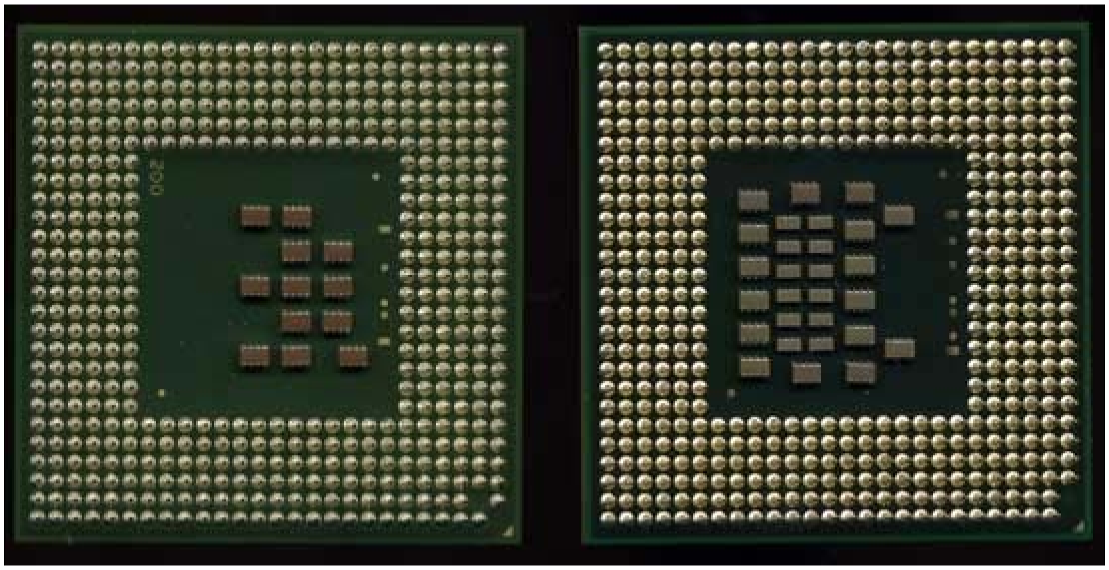
Technical specifications

This socket is physically similar to, yet different from Socket 478. The Socket 479 has a different electrical pin-arrangement from Socket 478, making it impossible to use a Pentium M in a normal 478 board although the Pentium M fits mechanically in a Socket 478. For this reason manufacturers like Asus have made drop-in boards (e.g. CT-479) which let you use Socket 479 processors.

Chipsets which employ this socket for the Pentium M are the Intel 855GM/GME/PM and Intel 915GM/GMS/PM. While the Intel 855GME chipset supports all Pentium M CPU's, the Intel 855GM chipset does not officially support 90 nm 2MB L2 cache (Dothan core) models (even though it works, it only works at 400FSB, some 3rd party/user was able to overclock the FSB on 855GM/GME/PM to support 533FSB Dothan Core)... The other difference is the 855GM chipset gaphics core runs at 200 MHz while the 855GME is runs at 250 MHz.



Asus CT-479 adapter

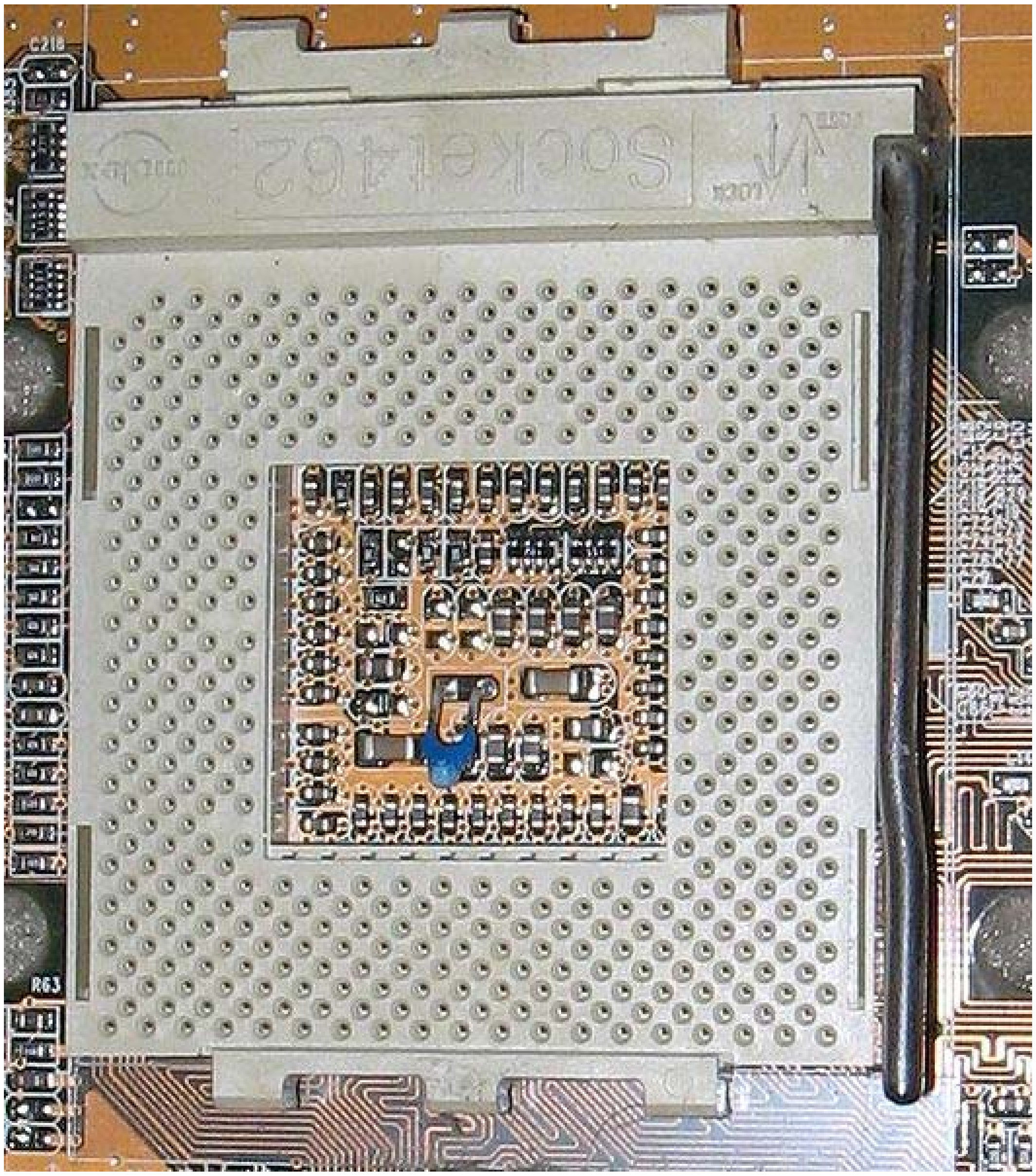


A comparison of the Dothan (Pentium M SL7SM) (left) and Yonah (Core Duo) (right). Both chips have 478 pins, but the placement of one pin has been changed.

In 2006, Intel released the successor to Socket 479 with a revised pinout for its Core processor, called Socket M. This socket has the placement of one pin changed from the Pentium M version of Socket 479; Socket M processors will physically fit into a Socket

479, but are electrically incompatible with most versions of the socket. Socket M supports a 667 MT/s FSB with the Intel 945PM/945GM chipsets.

Zero insertion force



A large ZIF socket (socket A)

ZIF is an acronym for **zero insertion force**, a concept used in the design of IC sockets, invented to avoid problems caused by applying force upon insertion and extraction.

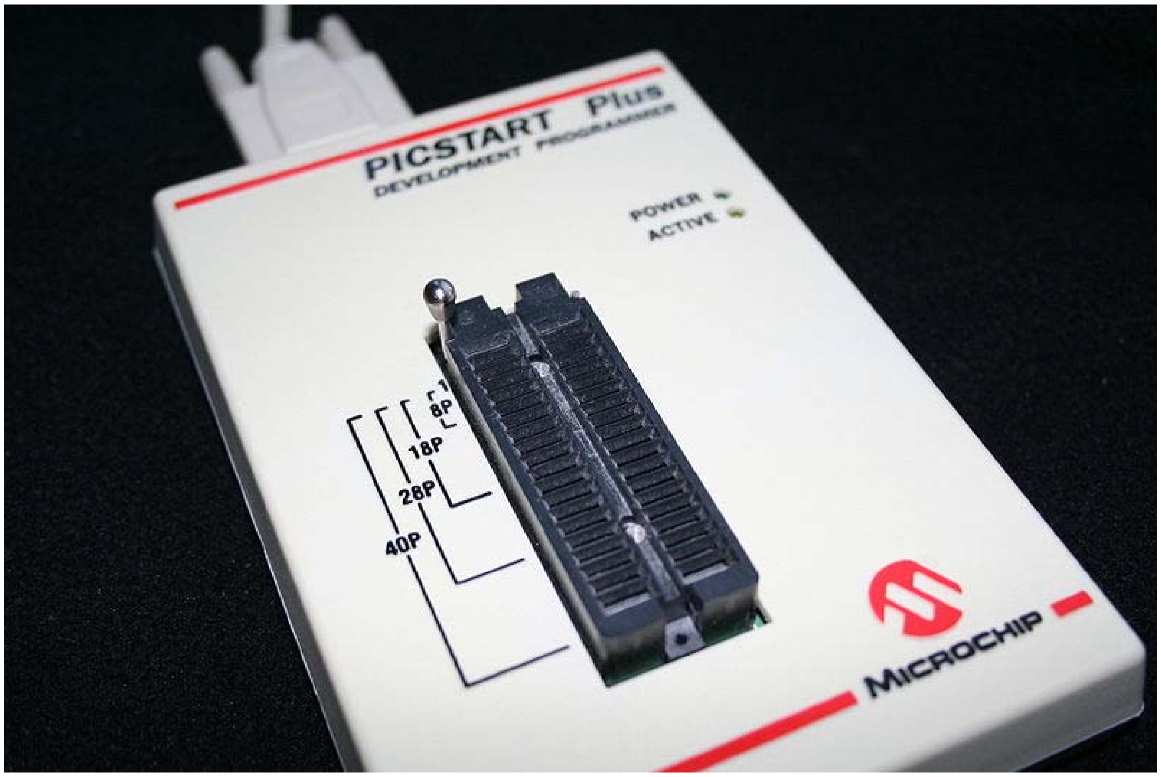
A normal integrated circuit (IC) socket requires the IC to be pushed into sprung contacts which then grip by friction. For an IC with hundreds of pins, the total insertion force can be very large (tens of newtons), leading to a danger of damage to the device or the PCB. Also even with relatively small pin counts each extraction is fairly awkward and carries a significant risk of bending pins (particularly if the person performing the extraction hasn't had much practice or the board is crowded). Low insertion force (LIF) sockets reduce the issues of insertion and extraction but the lower the insertion force of a conventional socket, the less reliable the connection is likely to be.

With a ZIF socket, before the IC is inserted, a lever or slider on the side of the socket is moved, pushing all the sprung contacts apart so that the IC can be inserted with very little force (generally the weight of the IC itself is sufficient with no external downward force required). The lever is then moved back, allowing the contacts to close and grip the pins of the IC. ZIF sockets are much more expensive than standard IC sockets and also tend to take up a larger board area due to the space taken up by the mechanism. Therefore they are only used when there is a good reason to do so.

Large ZIF sockets are only commonly found mounted on PC motherboards (from about the mid 1990s forward). These CPU sockets are designed to support a particular range of CPUs, allowing computer retailers and consumers to assemble motherboard/CPU combinations based on individual budget and requirements. The rest of the electronics industry has largely abandoned sockets and moved to surface mount components soldered directly to the board.

Smaller ZIF sockets are commonly used in chip-testing and programming equipment, e.g. programming and testing on EEPROMs, Microcontrollers, etc.

Universal test sockets



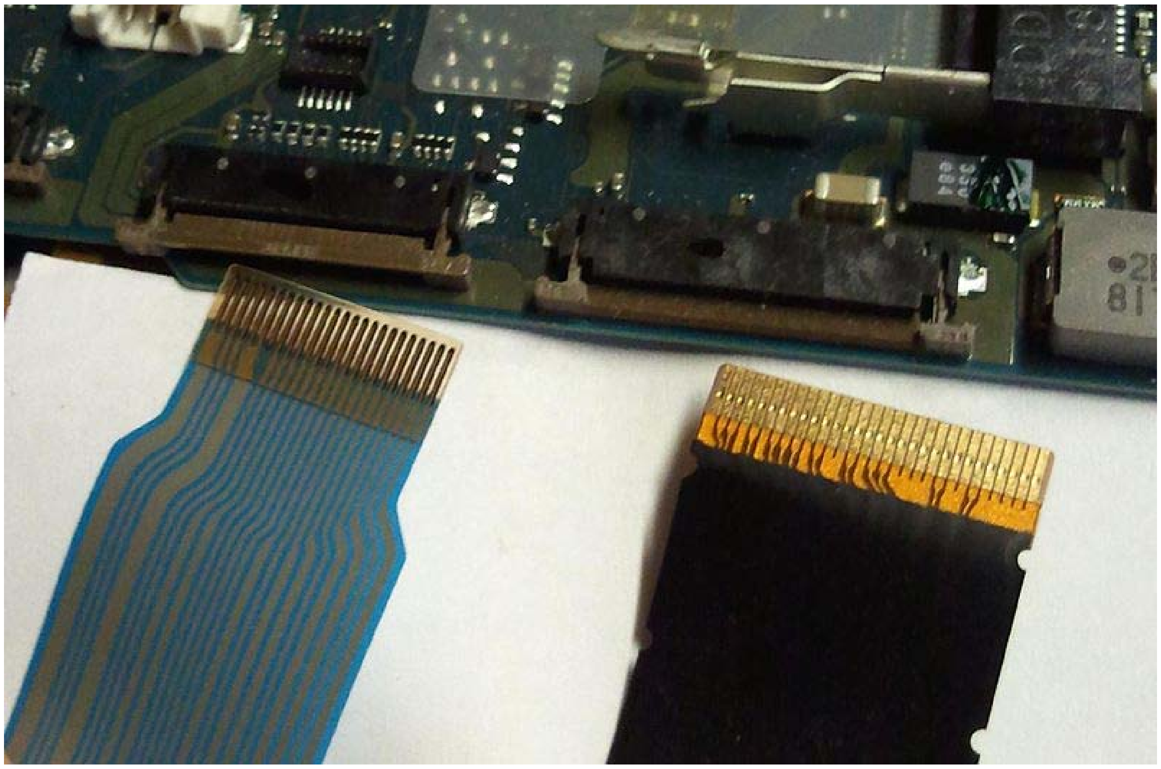
Programming device for a PIC microcontroller, with a dual in-line ZIF socket

Standard DIP packages come in two widths (measured between pin centers), 0.3 in (7.62 mm) (skinny dip) for smaller devices (8-28 pin) and 0.6 in (15.24 mm) for larger devices (24-40 pin). To allow design of programmers and similar devices that supported a range of devices some in skinny dip and some in full width dip universal test sockets are produced. These have wide slots into which the pins drop allowing both 0.3 in and 0.6 in devices to be inserted.

Ball grid array sockets

ZIF sockets can be used for ball grid array chips, particularly during development. These sockets tend to be unreliable, failing to grab all the balls. Another type of BGA socket, also free of insertion force but not a "ZIF socket" in the traditional sense, does a better job by using spring pins to push up underneath the balls.

ZIF wire-to-board connectors



A pair of ZIF connectors, with the flexible flat cable that connects to them

ZIF wire-to-board connectors are used for attaching wires to printed circuit boards inside electronic equipment. The wires, often formed into a ribbon cable, are pre-stripped and the bare ends placed inside the connector. The two sliding parts of the connector are then pushed together, causing it to grip the wires. The most important advantage of this system is that it does not require a mating half to be fitted to the wire ends, therefore saving space and cost inside miniaturised equipment.

Hard disk drives

ZIF tape connections are used for connecting IDE and SATA disk drives (mostly 1.8" factor). ZIF-style connectors for IDE hard drives were used primarily in the design of Ultra-Portable notebooks, and has since been phased out, as SATA has a relatively small-form-factor connector by default. Also, nearly all hard drives use ZIF tape to connect their circuit board to their platter motor.

ZIF tape connections are also heavily used in the design of the Apple Inc. iPod range of portable media players.

Chapter 10

Slot 1 and Slot 2

Slot 1

Slot 1

Type	Slot
Chip form factors	• Single Edge Contact Cartridge (Pentium II) • Single Edge Contact Cartridge 2 (Pentium II, Pentium III) • Single Edge Processor Package (Celeron)
Contacts	242
FSB protocol	GTL+
FSB frequency	66, 100, and (on third-party chipsets) 133 MHz
Voltage range	1.3 to 3.50 V
Processors	Pentium II: 233–450 MHz Celeron: 266–433 MHz Pentium III: 450–1.133 GHz (A Slotket makes following Socket 370 CPUs usable: Celeron and Pentium III to 1,400 MHz,

VIA Cyrix III: 350–733 MHz,
VIA C3: 733–1,200 MHz

Slockets also made it possible to use some Pentium Pro CPUs for Socket 8 using the same method.)

Slot 1 refers to the physical and electrical specification for the connector used by some of Intel's microprocessors, including the Pentium Pro, Celeron, Pentium II and the Pentium III. Both single and dual processor configurations were implemented.

General

With the introduction of the Pentium II CPU, the transition from socket to slot had become necessary. With the Pentium Pro, Intel had combined processor and cache dies in the same package, connected by a full-speed bus, resulting in significant performance benefits. Unfortunately, this method required that the two components be bonded together early in the production process, before testing was possible. As a result, a single, tiny flaw in either die made it necessary to discard the entire assembly, causing low production yield and high cost.

Intel subsequently designed a circuit board where the CPU and cache remained closely integrated, but were mounted on a printed circuit board, called a Single-Edged Contact Cartridge (SECC). The CPU and cache could be tested separately, before final assembly into a package, reducing cost and making the CPU more attractive to markets other than that of high-end servers. These cards could also be easily plugged into a Slot 1, thereby eliminating the chance for pins of a typical CPU to be bent or broken when installing in a socket.

The form factor used for Slot 1 was a 5-inch-long, 242-contact edge connector named SC242. To prevent the cartridge from being inserted the wrong way, the slot was keyed to allow installation in only one direction. The SC232 was later used for AMD's Slot A as well, and while the two slots were identical mechanically, they were electrically incompatible. To discourage Slot A users from trying to install a Slot 1 CPU, the connector was rotated 180 degrees on Slot A motherboards.

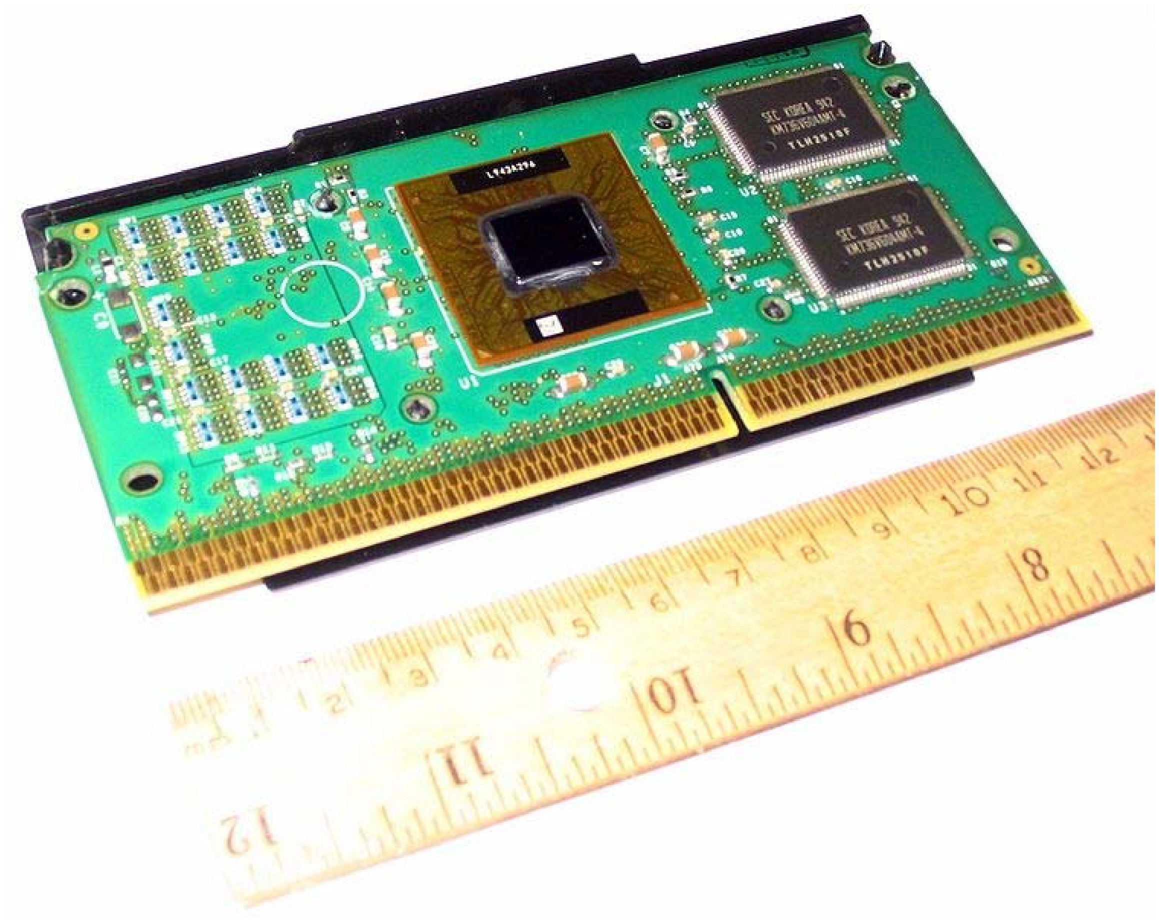
With the new Slot 1, Intel added support for symmetric multiprocessing (SMP). A maximum of two Pentium II or Pentium III CPUs can be used in a dual slot motherboard. The Celeron does not have official SMP support.

There are also converter cards, known as Slotkets, which hold a Socket 8 so that a Pentium Pro CPU can be used with Slot 1 motherboards. These specific converters, however, are rare. Another kind of slotket allows using a Socket 370 CPU in a Slot 1. Many of these latter devices are equipped with own voltage regulator modules, in order to supply the new CPU with a lower core voltage, which the motherboard would not otherwise allow.

Form Factors

Intel Pentium II CPU in SECC form factor

Celeron in SEPP: CPU at center (under heat spreader), surrounding chips are resistors



Pentium III (Katmai) in SECC2: CPU at center, two chips at right are cache

The Single Edge Contact Cartridge, or "**SECC**", was used at the beginning of the Slot 1-era for Pentium II CPUs. Inside the cartridge, the CPU itself is enclosed in a hybrid plastic and metal case. The back of the housing is plastic and has several markings on it: the name, "Pentium II"; the Intel logo; a hologram; and the model number. The front consists of a black anodized aluminum plate, which is used to hold the CPU cooler. The SECC form is very solid, because the CPU itself is resting safely inside the case. As compared to socket-based CPUs, there are no pins that can be bent, and the CPU is less likely to be damaged by improper installation of a cooler.

Following SECC, the **SEPP**-form (Single Edge Processor Package) appeared on the market. It was designed for lower-priced Celeron CPUs. This form lacks a case entirely, consisting solely of the printed-circuit board holding the components.

The Pentium III processors also use a protective case, renamed **SECC2**. The only item to be carried forward from the old SECC form is the backplate. The CPU lies free and, as with Celerons, the heatsink lies directly on the core.

History

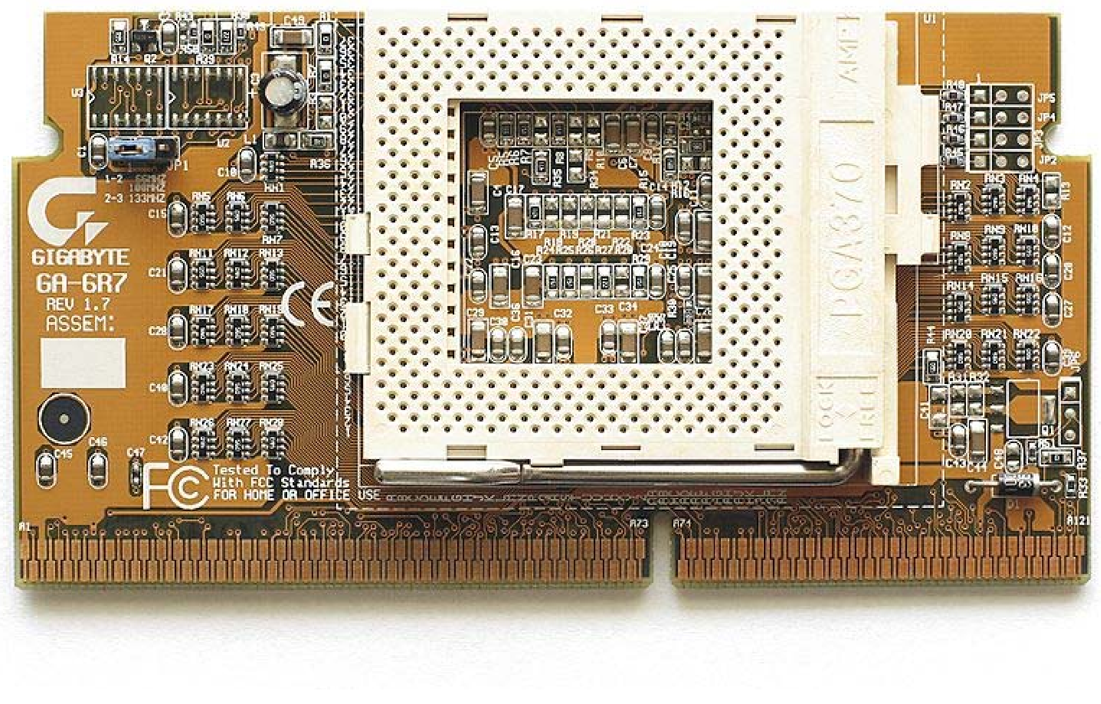
Historically, there are three platforms for the Intel P6-CPUs: Socket 8, Slot 1 and Socket 370.

Slot 1 is a successor to Socket 8. While the Socket 8 CPUs (Pentium Pro) directly had the L2-cache embedded into the CPU, it is located (outside of the core) on a circuit board shared with the core itself. The exception is later Slot 1 CPUs with the Coppermine core which have the L2-Cache embedded into the die.

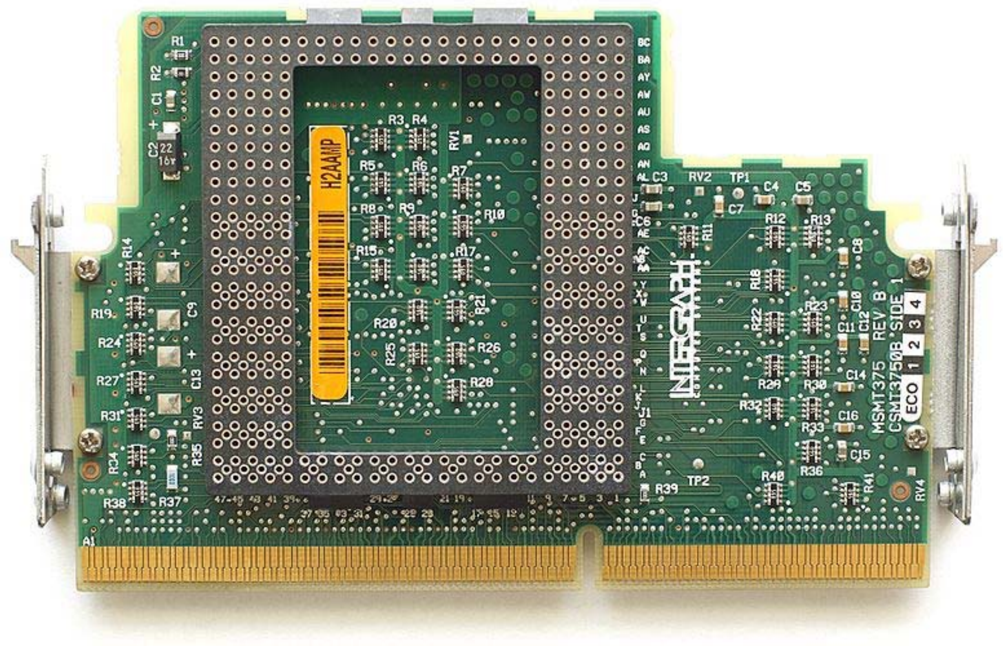
In the beginning of 2000, while the Pentium-III-CPUs with FC-PGA-housing appeared, Slot 1 was slowly succeeded by Socket 370, after Intel had already offered Socket 370 and Slot 1 at the same time since the beginning of 1999. Socket 370 was initially made for the low-cost Celeron processors, while Slot 1 was thought of as a platform for the expensive Pentium II and early Pentium III models. Cache and core were both embedded into the die.

Slot 1 also obsoleted the old Socket 7, at least regarding Intel, as the standard platform for the home-user. After abandoning the Intel P5 Pentium MMX CPU, Intel completely left the Socket 7 market to the manufacturers AMD, Cyrix and IDT.

Chipsets and officially supported CPUs



Slot 1/Socket 370 Converter



Slot 1/Socket 8 Converter

Intel 440FX

- Introduced in: May 6, 1996
- FSB: 66 MHz
- PIO/WDMA
- Supported RAM type: EDO-DRAM
- Supported CPUs:
    - Pentium Pro
    - Pentium II with 66 MHz FSB
    - Celeron (Covington, Mendocino)
- Used in both Socket 8 (Pentium Pro) and Slot 1 (Pentium II, early Celerons)
- Does not support AGP or SDRAM

Intel 440LX

- Introduced in: August 27, 1997
- FSB: 66 MHz
- Supported RAM type: SDRAM
- Supported CPUs: Pentium II, Celeron
- AGP 1x Mode
- UDMA/33
    - Pentium II with 66 MHz FSB
    - Celeron (Covington, Mendocino)
- Introduced support for AGP and SDRAM

Intel 440BX

- Introduced in: April 1998
- FSB: 66 and 100 MHz (some motherboards supported overclocking to 133 MHz, allowing usage of Socket 370 CPUs using a Slocket)
- AGP 2x Mode
- UDMA/33
- Supported RAM types: SDRAM (PC66 and PC100, PC133 with overclocking)
- Supported CPUs:
  - Pentium II with 66 and 100 MHz FSB
  - Pentium III with 100 MHz FSB (133 with overclocking)
  - Celeron (Covington, Mendocino, Coppermine)

Intel 440ZX

- Introduced in: November 1998
- FSB: 66 and 100 MHz (some motherboards supported overclocking to 133 MHz, allowing usage of Socket 370 CPUs using a Slocket)
- AGP 2x Mode
- UDMA/33
- Supported RAM types: SDRAM (PC66 and PC100, PC133 with overclocking)
- Supported CPUs:
  - Pentium II with 66 and 100 MHz FSB
  - Pentium III with 100 MHz FSB (133 with overclocking)
  - Celeron (Covington, Mendocino, Coppermine)

Intel 820/820E (Camino)

- Introduced in: November 1999
- FSB: 66, 100, and 133 MHz
- AGP 4x Mode
- UDMA/66 (i820), UDMA/100 (i820E)
- Supported RAM types: RDRAM, SDRAM (PC133)
- Supported CPUs: All Slot 1 CPUs
- Allowed up to two CPUs for SMP

Via Apollo Pro / Pro+

- Introduced in: 1998?
- FSB: 66, 100 MHz (some motherboards supported overclocking to 133 MHz, allowing usage of Socket 370 CPUs using a Slocket)
- AGP 2x Mode
- UDMA/33
- Supported CPUs:
  - Pentium II with 66 and 100 MHz FSB
  - Pentium III with 100 MHz FSB (133 with overclocking)

    - Celeron (Covington, Mendocino, Coppermine)

Via Apollo Pro 133

- Introduced in: July 1999
- FSB: 66, 100, and 133 MHz
- AGP 2x Mode
- UDMA/66
- Supported CPUs: All Slot 1 CPUs

Via Apollo Pro 133A

- Introduced in: Oct 1999
- FSB: 66, 100, and 133 MHz
- AGP 4x Mode
- UDMA/100
- Supported CPUs: All Slot 1 CPUs

Pentium III EB

A special series of Pentium III for Slot 1 had an addition to its name (Pentium III EB). The "E" indicates a Coppermine core, and the "B" a 133 MHz FSB. Ironically, no consumer-oriented Intel chipset (except select few i820-based motherboards) officially supports a FSB of 133 MHz. To use this CPU, one must either use the VIA Apollo Pro 133A chipset, an Intel 800-series chipset (which was relatively expensive and often required RDRAM, limiting it to high-end workstations or servers), or the Intel 440BX outside of its specifications (which, on some motherboards, is indeed possibly stable).

Slot 2

Slot 2

Type	Slot
Chip form factors	Single Edge Contact Cartridge
Contacts	330

FSB protocol	GTL+, later AGTL+
FSB frequency	100 MT/s, 133 MT/s
Voltage range	1.3 to 3.3 V
Processors	• Intel Pentium II Xeon (400-450 MHz) • Intel Pentium III Xeon (500-1000 MHz)

Slot 2 refers to the physical and electrical specification for the 330-lead Single Edge Contact Cartridge (or edge-connector) used by some of Intel's Pentium II Xeon and certain models of the Pentium III Xeon.

When first introduced, Slot 1 Pentium IIs were intended to replace the Pentium and Pentium Pro processors in the home, desktop, and low-end SMP markets. The Pentium II Xeon, which was aimed at multiprocessor workstations and servers, was largely similar to the later Pentium IIs, being based on the same P6 Deschutes core, aside from a wider choice of L2 cache ranging from 512 to 2048 KB and a full-speed off-die L2 cache (the Pentium 2 used cheaper 3rd party SRAM chips, running at 50% of CPU speed, to reduce cost).

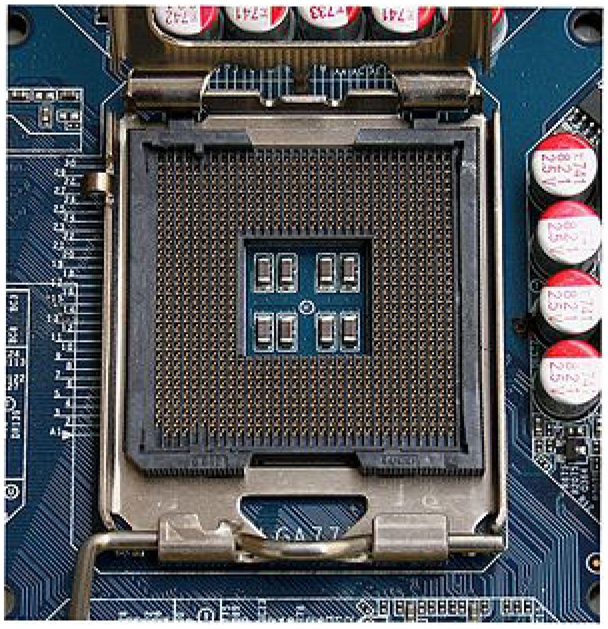
Because the design of the 242-lead Slot 1 connector did not support the full-speed L2 cache of the Xeon, an extended 330-lead connector was developed. This new connector, dubbed 'Slot 2', was used for Pentium 2 Xeons and the first two Pentium III Xeon cores, codenamed 'Tanner' and 'Cascades'. Slot 2 was finally replaced with the Socket 370 with the Pentium III Tualatin; some of the Tualatin Pentium IIIs were packaged as 'Pentium III' and some as 'Xeon', despite the fact they were identical.

Chapter 11

LGA 775, LGA 1156 and LGA 1366

LGA 775

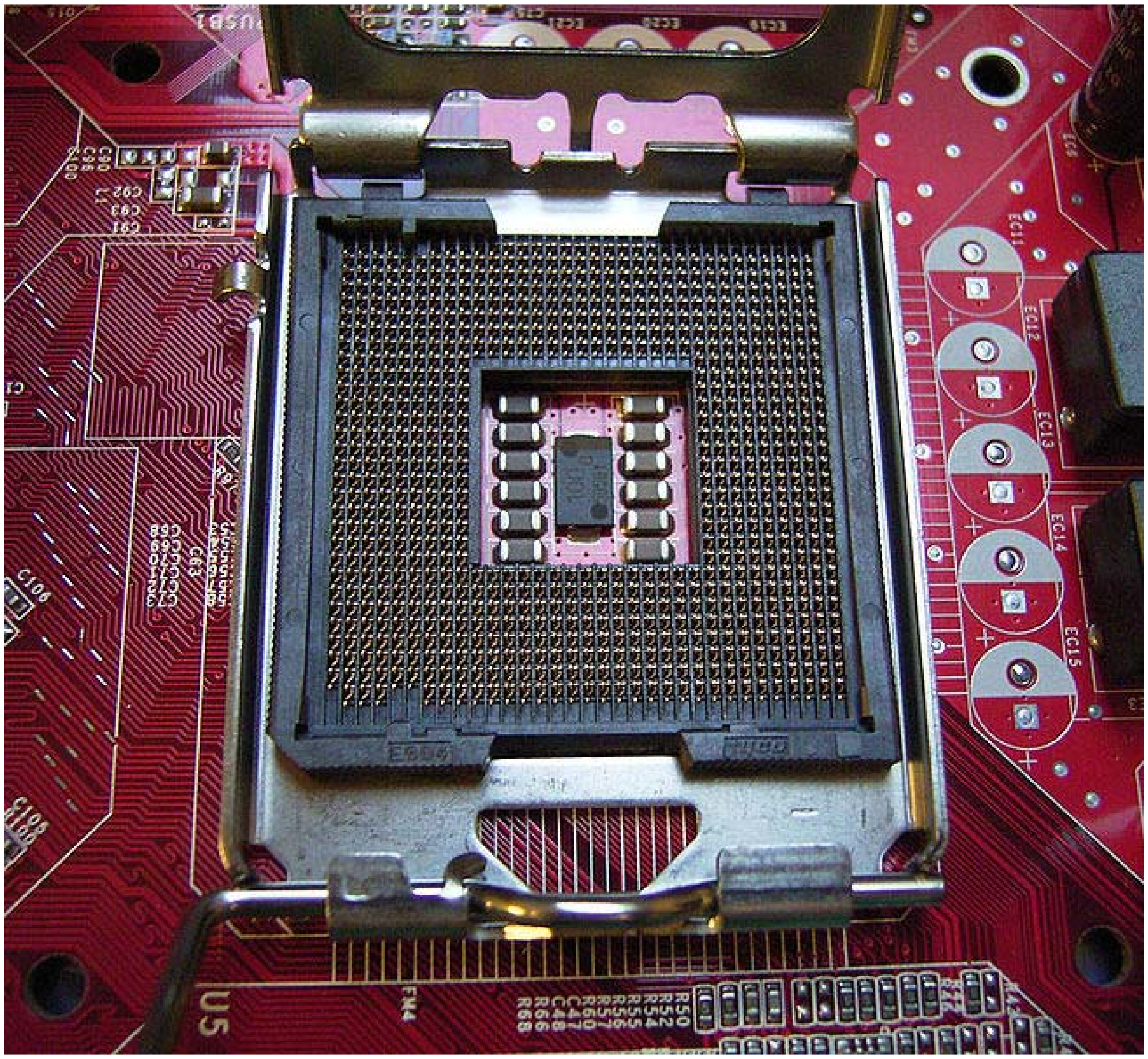
LGA 775



Type	LGA
Chip form factors	Flip-chip land grid array
Contacts	775
FSB protocol	AGTL+
FSB frequency	133 MHz (533 MT/s) 200 MHz (800 MT/s) 266 MHz (1066 MT/s) 333 MHz (1333 MT/s) 400 MHz (1600 MT/s)
Processor dimensions	1.47 × 1.47 inches (37.5mm)

Processors	Intel Pentium 4 (2.60 - 3.80 GHz)
	Intel Celeron D (2.53 - 3.60 GHz)
	Intel Pentium 4 Extreme Edition (3.20 - 3.73 GHz)
	Intel Pentium D (2.66 - 3.60 GHz)
	Pentium Extreme Edition (3.20 - 3.73 GHz)
	Pentium Dual-Core (1.40 - 3.33 GHz)
	Intel Core 2 Duo (1.60 - 3.33 GHz)
	Intel Core 2 Extreme (2.66 - 3.20 GHz)
	Intel Core 2 Quad (2.33 - 3.00 GHz)
	Intel Xeon (1.86-3.40 GHz)
	Intel Celeron (1.60 - 2.40 GHz)

Core 2 Q6600



Sockel 775

LGA 775, also known as **Socket T**, is an Intel desktop CPU socket. LGA stands for Land Grid Array. Unlike earlier common CPU sockets, such as its predecessor Socket 478, the LGA 775 has no socket holes; instead, it has 775 protruding pins which touch contact points on the underside of the processor (CPU).

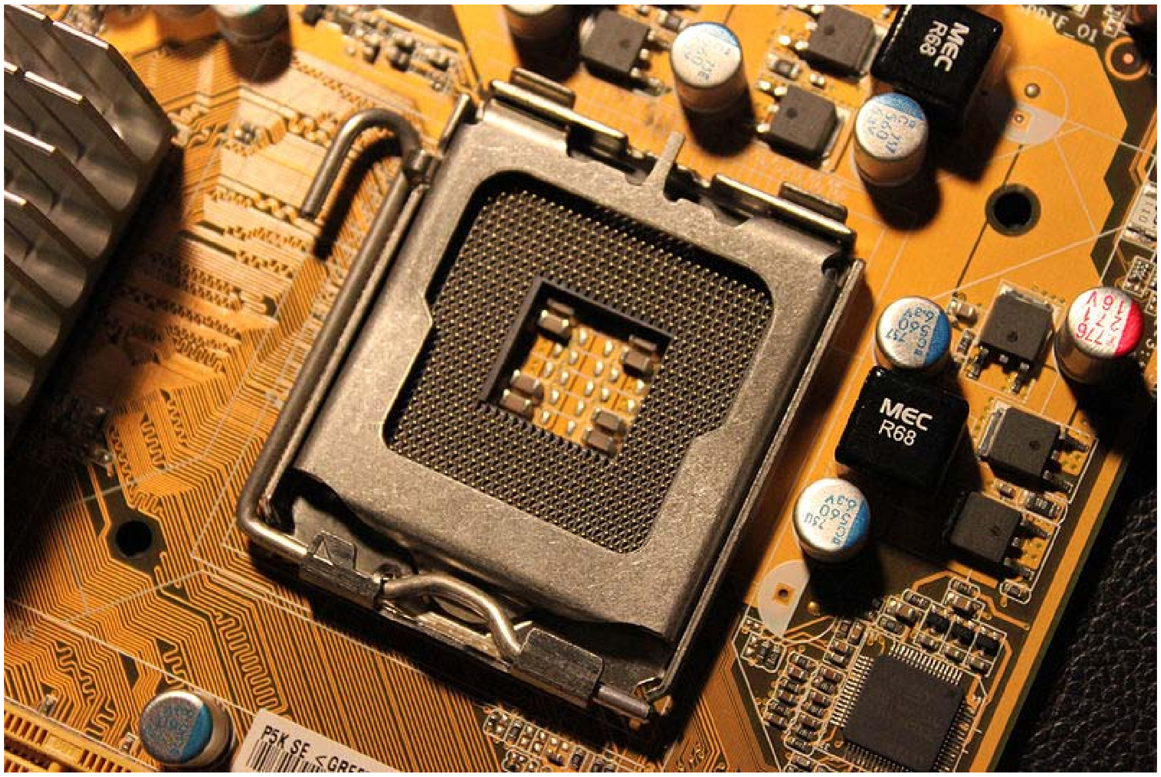
Technical specifications

The Prescott and Cedar Mill Pentium 4 cores, as well as the Smithfield and Presler Pentium D cores, used the LGA 775 socket. In July 2006, Intel released the desktop version of the Core 2 Duo (codenamed Conroe), which also uses this socket, as does the subsequent Core 2 Quad. Intel changed from Socket 478 to LGA 775 because the new pin type offers better power distribution to the processor, allowing the front side bus to be raised to 1600 MT/s. The 'T' in Socket T was derived from the now cancelled Tejas core, which was to replace the Prescott core. Another advantage for Intel with this newer architecture is that it is now the motherboard which has the pins, rather than the CPU, transferring the risk of pins being bent from the CPU to the motherboard.

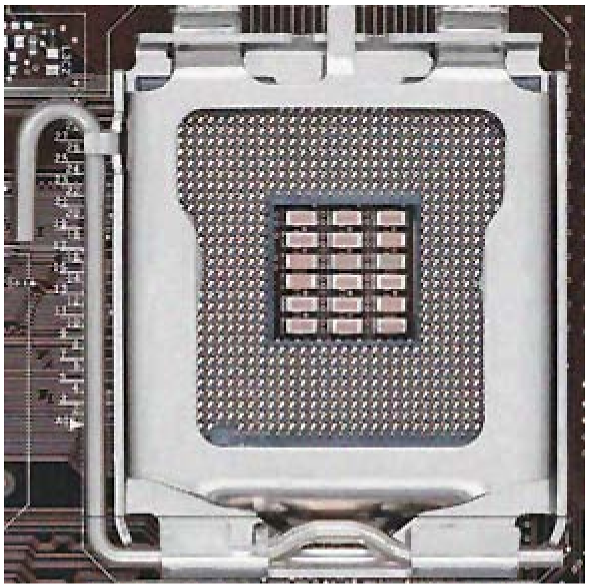
The CPU is pressed into place by a "weld plate", rather than human fingers directly. The installing technician lifts the hinged "weld plate", inserts the processor, closes the weld plate over the top of the processor, and pushes down a locking lever. The pressure of the locking lever on the weld plate clamps the processor's 775 copper contact points firmly down onto the motherboard's 775 pins, ensuring a good connection. The weld plate only covers the edges of the top surface of the CPU (processor heatspreader). The center is free to make contact with the cooling device placed on top of the CPU.

An examination of the relevant Intel data sheets shows that LGA 775 which is used for consumer level desktops and LGA 771 used for (Xeon based) workstation and server class computers appear to differ only in the placement of the indexing notches and the swap of two address pins. Many pins devoted to functions such as interfacing multiple CPUs are not clearly defined in the LGA 775 specifications, but from the information available appear to be consistent with those of LGA 771. Considering that LGA 775 predated LGA 771 by nearly a year and a half, it would seem that LGA 771 was adapted from LGA 775 rather than the other way around.

The socket has been superseded by the LGA 1156 (Socket H) and LGA 1366 (Socket B) sockets.



Socket 775

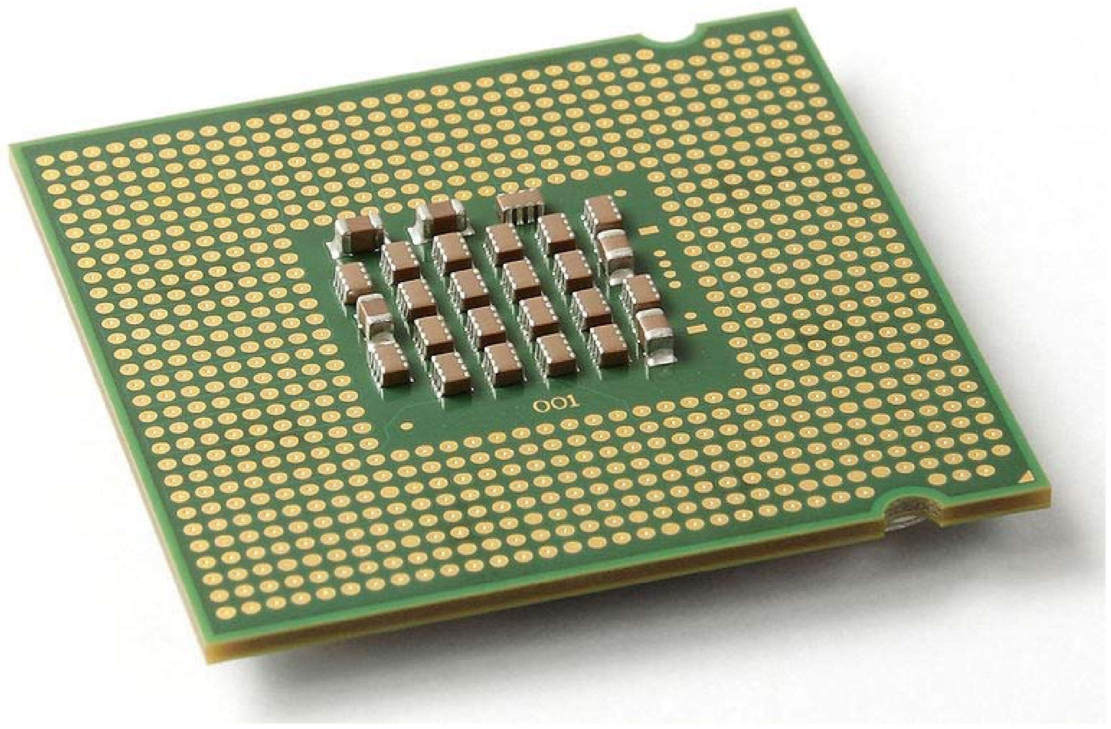


Socket 775

Improvements in heat dissipation

The force from the load plate ensures that the processor is completely level, giving the CPU's upper surface optimal contact with the heat sink or cold-water block fixed onto the top of the CPU to carry away the heat generated by the CPU. This socket also introduces a new method of connecting the heat dissipation interface to the chip surface and motherboard. With LGA 775, the heat dissipation interface is connected directly to the motherboard on four points, compared with the two connections of the Socket 370 and the "clamshell" four-point connection of the Socket 478. This was done to avoid the reputed danger of the heat sinks/fans of pre-built computers falling off in transit. LGA 775 was announced to have better heat dissipation properties than the Socket 478 it was designed to replace, but the Prescott core CPUs (in their early incarnations) ran much hotter than the previous Northwood-core Pentium 4 CPUs, and this initially neutralized the benefits of better heat transfer. However, modern Core 2 processors run at lower temperatures than the Prescott CPUs they replace.

LGA 775 mechanical load limits



The LGA 775 contact points on the underside of a Pentium 4 Prescott CPU

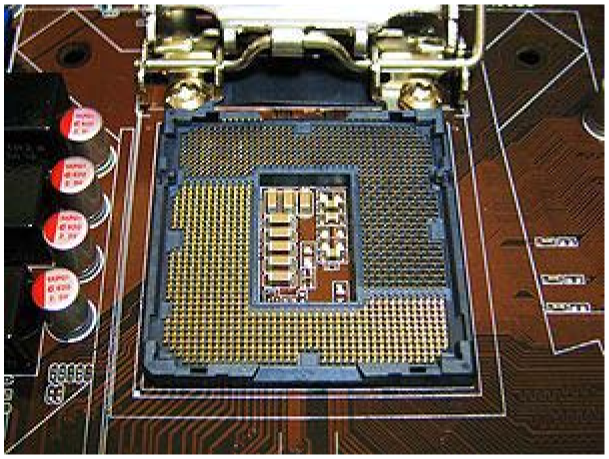
All LGA 775 processors have the following mechanical maximum load limits which should not be exceeded during heat sink assembly, shipping conditions, or standard use. Load above those limits will crack the processor die and make it unusable.

Location	Dynamic	Static
IHS Surface	756 N (170 lb$_f$) (77 kp)	311 N (70 lb$_f$) (31 kp)

The transition to the LGA packaging has lowered those load limits, which are smaller than the load limits of Socket 478 processors but they are bigger than Socket 370, Socket 423 and Socket A processors, which were fragile. They are large enough to ensure that processors will not crack.

LGA 1156

LGA 1156



Type	LGA
Chip form factors	Flip-chip land grid array
Contacts	1156
FSB protocol	PCIe 16x + 4x (DMI) + 2 DP (FDI), 2 DDR3 channels
Processor dimensions	37.5 × 37.5 mm
Processors	Intel Celeron Intel Core i3 Intel Core i5 Intel Core i7 Intel Pentium Intel Xeon

LGA 1156, also known as **Socket H** or **H1**, is an Intel desktop CPU socket. LGA stands for Land Grid Array.

LGA 1156, along with LGA 1366, was designed to replace LGA 775. LGA 1156 is very different from LGA 775. LGA 775 processors were connected to a northbridge using the Front Side Bus. With LGA 1156, the features that were traditionally on a northbridge are integrated onto the processor. The LGA 1156 socket allows the following connections to be made from the processor to the rest of the system:

- PCI-Express 2.0 x16 for communication with a graphics card. Some processors allow this connection to be divided into two x8 lanes to connect two graphics cards. Some motherboard manufacturers use Nvidia's NF200 chip to allow even more graphics cards to be used.

- DMI for communication with the Platform Controller Hub (PCH). This consists of a PCI-Express 2.0 x4 connection.
- FDI for communication with the PCH. This consists of two DisplayPort connections.
- Two memory channels for communication with DDR3 SDRAM. The clock speed of the memory that is supported will depend on the processor.

Supported processors

Code name	Brand name	Model (list)	Frequency	Cores/Threads	Max Memory Speed
Lynnfield	Core i5	i5-7xx	2.66-2.8 GHz	4/4	DDR3-1333
	Core i7	i7-8xx	2.8-2.93 GHz	4/8	
	Xeon	L34xx	1.86 GHz	4/4 or 4/8	
		X34xx	2.4-2.93 GHz		
Clarkdale	Celeron	G1xxx	2.26 GHz	2/2	DDR3-1066
	Pentium	G6xxx	2.80 GHz	2/2	
	Core i3	i3-5xx	2.93-3.2 GHz	2/4	DDR3-1333
	Core i5	i5-6xx	3.2-3.6 GHz	2/4	

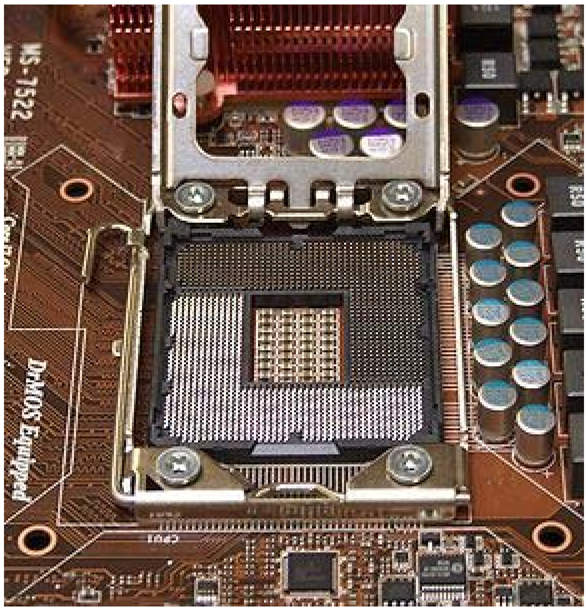
All LGA 1156 processors and motherboards made to date are interoperable, making it possible to switch between a Celeron, Pentium, Core i3 or Core i5 with integrated graphics and a Core i5 or Core i7 without graphics. However, using a chip with integrated graphics on a P55 motherboard will (in addition to likely requiring a BIOS update) not allow use of the on-board graphics processor, and likewise, using a chip without integrated graphics on a H55, H57 or Q57 motherboard will not allow use of the motherboard's graphics ports.

Supported chipsets

The Desktop chipsets that support LGA 1156 are Intel's H55, H57, P55, and Q57. Server chipsets supporting the socket are Intel's 3400, 3420 and 3450

LGA 1366

Socket B



Type	LGA
Chip form factors	Flip-chip land grid array
Contacts	1366
FSB protocol	Intel QuickPath Interconnect
FSB frequency	1× to 2× QuickPath
Processor dimensions	1.77 × 1.67 inches
Processors	Intel Core i7 (9xx series) Intel Xeon (35xx, 36xx, 55xx, 56xx series) Intel Celeron P1053

LGA 1366, also known as **Socket B**, is an Intel CPU socket. This socket supersedes Intel's LGA 775 (Socket T) in the high-end and performance desktop segments. It also replaces the server-oriented LGA 771 (Socket J) in the entry level. LGA stands for land grid array. This socket has 1,366 protruding pins which touch contact points on the underside of the processor (CPU) and accesses up to three channels of DDR3 memory via the processor's internal memory controller.

Socket 1366 (Socket B) uses QPI to connect the CPU to a reduced-function northbridge that serves mainly as a PCI-Express controller. A slower DMI is used to connect Intel's most recent northbridge and southbridge components. By comparison, Intel's socket 1156 (Socket H) moves the QPI link and PCI-Express controller onto the processor itself, using DMI to interface a single-component "chipset" (now called PCH) that serves traditional southbridge functions. The difference in pin number is mostly a reflection of the number of memory channels served.

In November 2008, Intel released Core i7, which was the first processor requiring this socket.

Socket B mechanical load limits

Socket B processors have the following mechanical maximum load limits which should not be exceeded during heatsink assembly, shipping conditions, or standard use. Load above those limits will crack the processor die and make it unusable.

Location	Dynamic	Static
IHS Surface	890 N (200 lb_f)	266 N (60 lb_f)

Processors using this socket have a lower static load limit than previous models using LGA 775.

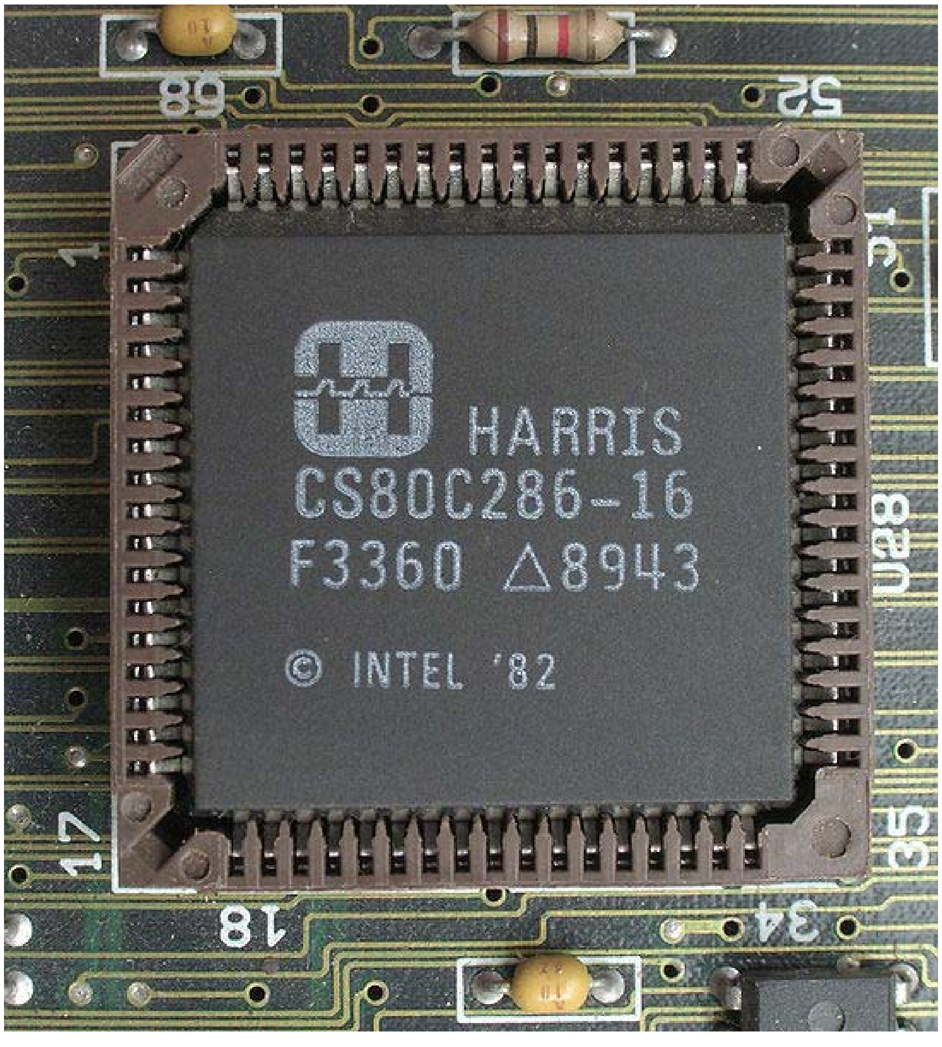
Supported chipsets

The desktop chipset that supports LGA 1366 is Intel's X58.

Chapter 12

Plastic Leaded Chip Carrier, LGA 771 and Socket M

Plastic leaded chip carrier



The Harris CS80C286-16 CPU, an application of the PLCC68 package, in a CPU socket

Micro-controller Motorola MC68HC711E9CFN3 in QFJ52 / PLCC52

Gigabyte DualBIOS in QFJ32 / PLCC32

A **plastic leaded chip carrier** (**PLCC**) is a plastic, four-sided chip carrier, with a "J"-lead and pin spacings of 0.05" (1.27 mm). Lead counts range from 20 to 84. PLCC packages can be square or rectangular. Body widths range from 0.35" to 1.15". PLCCs conform to the JEDEC standard. The PLCC “J” Lead configuration requires less board space versus equivalent gull leaded components, and is a less expensive version of the leadless chip carrier, which is a housing with flat contacts instead of pin connectors, on each side.

The heatspreader versions are identical in form factor to the standard non-heatspreader versions. Both versions are JEDEC compliant in all respects. The heatspreader versions give the system designer greater latitude in thermally enhanced board level and / or system design. RoHs compliant, lead-free & green material sets are now qualified standards.

A PLCC circuit may either be installed in a PLCC socket or surface-mounted. PLCC sockets may in turn be surface mounted, or use through-hole technology. The motivation for a surface-mount PLCC socket would be when working with devices that cannot withstand the heat involved during the reflow process, or to allow for component replacement without reworking. Using a PLCC socket may be necessary in situations where the device requires stand-alone programming, such as some flash memory devices. Some through-hole sockets are designed for prototyping with wire wrapping.

A specialized tool called a PLCC extractor facilitates the removal of a PLCC from a socket.

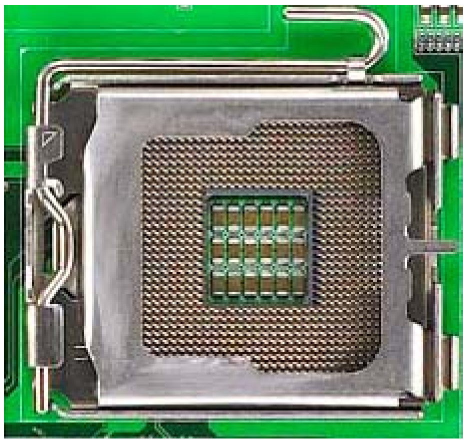
This package is still used for a wide variety of device types, which would include memory, processors, controllers, ASIC, DSP, etc. Applications range from consumer products through automotive and aerospace.

Variants

- QFJ20 (PLCC20) - (10-0-10-0)

LGA 771

Socket J (LGA 771)



Type	LGA
Chip form factors	Flip chip land grid array
Contacts	771
FSB frequency	667 MT/s, 1066 MT/s, 1333 MT/s, 1600 MT/s

Voltage range	Varies
Processors	Intel Dual-Core Xeon E/X/L 5xxx Intel Quad-Core Xeon E/X/L 5xxx Intel Core 2 Extreme QX9775

LGA 771, also known as Socket J, is a CPU interface introduced by Intel in 2006. It is used in Intel Core microarchitecture based DP-capable server processors, the Dual-Core Xeon is codenamed Dempsey, Woodcrest, and Wolfdale and the Quad-Core processors Clovertown, Harpertown. It is also used for the Core 2 Extreme QX9775.

The new Nehalem-based Xeon processors use LGA 1366 instead.

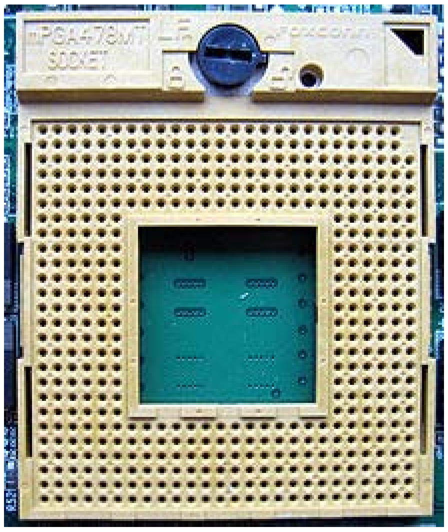
Technical specifications

As its name implies, it is a land grid array with 771 contacts. The word "socket" in this instance is a misnomer, as the processor interface has no pin holes. Instead, it has 771 protruding lands which touch contact points on the underside of the microprocessor.

The "J" in "Socket J" refers to the now-canceled processor codenamed "Jayhawk", which was expected to debut alongside this interface. It is intended as a successor to Socket 604 and takes much of its design from LGA 775.

Socket M

Socket M



Type	PGA-ZIF
Chip form factors	Flip-chip pin grid array
Contacts	478 (not to be confused with the previous Socket 479)
FSB frequency	533 MT/s, 667 MT/s, 800MT/s
Processors	Intel Core Solo T1300, T1350, T1400 Intel Core Duo T2050, T2250, T2300, T2300E, T2350, T2400, T2450, T2500, T2600, T2700 Intel Core 2 Duo T5200, T5300, T5450, T5470, T5500, T5600, T7200, T7400, T7500, T7600 Intel Celeron M Intel Celeron 1.66 GHz

Socket M (mPGA478MT), also known as Socket 479 mPGA is a CPU interface introduced by Intel in 2006 for the Intel Core line of mobile processors .

Technical specifications

It is used in all Intel Core products, as well as the Core-derived Dual-Core Xeon codenamed Sossaman. It was also used in the first generation of the mobile version of Intel's Core 2 Duo, specifically, the T5x00 and T7x00 Merom lines (referred to as Napa Refresh), though that line switched to Socket P (Santa Rosa) in 2007. It typically uses the Intel 945PM/945GM chipsets which support up to 667 MHz FSB and the Intel PM965/GM965 which allows 800 MHz FSB support, though the Socket M, PM965/GM965 combination is less common. The "Sossaman" Xeons use the E7520 chipset.

Relation to other sockets

Although it may seem identical, Socket M is not pin-compatible with the older desktop Socket 478 or the newer mobile Socket P; it is also incompatible with most versions of the older mobile Socket 479. Pentium III-M processors designed for the first version of Socket 479 will physically fit into a Socket M, but are electrically incompatible with it. Although conflicting information has been published, no 45nm Penryn processors have been released for Socket M.