1

EVOLUTION OF MICROPROCESSORS

1.1 INTRODUCTION

A microprocessor is the chip containing some control and logic circuits that is capable of making arithmetic and logical decisions based on input data and produces the corresponding arithmetic or logical output. The word 'processor' is the derivative of the word 'process' that means to carry out systematic operations on data. The computer we are using to write this page of the manuscript uses a **microprocessor** to do its work. The microprocessor is the **heart** of any computer, whether it is a **desktop** machine, a **server or a laptop.** The microprocessor we are using might be a **Pentium, a K6, a PowerPC, a Sparc** or any of the many other brands and types of microprocessors, but they all do approximately the same thing in approximately the same way. No logically enabled device can do any thing without it. The microprocessor not only forms the very basis of computers, but also many other devices such as cell phones, satellites, and many other hand held devices. They are also present in modern day cars in the form of microcontrollers.

A microprocessor is also known as a **CPU** or central processing unit, which is a complete computational engine that is fabricated on a single chip. Here we will discuss the history of the 80x86 CPU family and the major improvements occurring along the line. The historical background will help us to better understand the design compromises they made as well as to understand the legacy issues surrounding the CPU's design. We are discussing the major advances in computer architecture that Intel employed while improving the x86.

1.2 HISTORICAL DEVELOPMENT OF THE MICROPROCESSORS

Intel developed and delivered the first commercially viable microprocessor way back in the early 1970's: the 4004 and 4040 devices. The 4004 was not very powerful and all it could do was **add** and **subtract** with 4-bit data only at a time. But it was amazing those days that everything was on one chip. Prior to the 4004, engineers built computers either from collections of chips or from discrete components (Transistor wired one at a time). The machines then, were not portable and were very bulky, and power hungry. The 4004 changed the scene with all its circuitry on a single chip. The 4004 powered one of the first portable electronic calculators named **'Busicom'.** These 4-bit microprocessors, intended for use in calculators, required very little power Nevertheless, they demonstrated the future potential of the microprocessor - an entire CPU on a single

piece of silicon. Intel rapidly followed their 4-bit offerings with their 8008 and 8080 eight-bit CPUs. A small outfit in Santa Fe, New Mexico, incorporated the 8080 CPU into a box they called the Altair 8800. Although this was not the world's first "personal computer" (there were some limited distribution machines built around the 8008 prior to this), the Altair was the device that sparked the imaginations of hobbyists of the world and the personal computer revolution was born.

The trends in processor design had impact of historical development of microprocessors from different manufacturers. Intel started facing competition from Motorola, MOS Technology, and an upstart company formed by disgruntled Intel employees, Zilog. To compete, Intel produced the 8085 microprocessor. To the software engineer, the 8085 was essentially the same as the 8080. However, the 8085 had lots of hardware improvements that made it easier to design into a circuit. Unfortunately, from software perspective the other manufacturer's offerings were better. Motorola's 6800 series was easier to program, MOS Technologies' 65xx family was also easier to program but very inexpensive, and Zilog's Z80 chip was upward compatible with the 8080 with lots of additional instructions and other features. By 1978 most personal computers were using the 6502 or Z80 chips, not the Intel offerings.

The first microprocessor to make a real splash in the market was the Intel 8088, introduced in 1979 and incorporated into the IBM PC (which appeared around 1982 for the first time). If we are familiar with the PC market and its history, we know that the PC market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. Intel makes all of these microprocessors and all of them are improvements of design base of the 8088. The Pentium 4 can execute any piece of code that ran on the original 8088, but it does it about 5,000 times faster!

Sometime between 1976 and 1978 Intel decided that they needed to *leap-frog* the competition and produced a 16-bit microprocessor that offered substantially more power than their competitor's eight-bit offerings. This initiative led to the design of the 8086 microprocessor. The 8086 microprocessor was not the world's first 16-bit microprocessor (there were some oddball 16-bit microprocessors prior to this point) but it was certainly the highest performance single-chip 16-bit microprocessor when it was first introduced.

During the design timeframe of the 8086 *memory* was very *expensive*. Sixteen Kilobytes of RAM was selling above $200 at the time. One problem with a 16-bit CPU is that programs tend to consume more memory than their counterparts on an 8-bit CPU. Intel, ever cogniscent of the fact that designers would reject their CPU if the total system cost was too high, made a special effort to design an instruction set that had a high memory density (that is, packed as many instructions into as little RAM as possible). Intel achieved their design goal and programs written for the 8086 were comparable in size to code running on 8-bit microprocessors. However, those design decisions still haunt us today.

At the time Intel designed the 8086 CPU the average lifetime of a CPU was only a couple of years. Their experiences with the 4004, 4040, 8008, 8080, and 8085 taught them that designers would quickly ditch the old technology in favour of the new technology as long as the new stuff was radically better. So Intel designed the 8086 assuming that whatever compromises they made in order to achieve a high instruction density would be fixed in newer chips. Based on their experience, this was a reasonable assumption.

Intel's competitors were not standing still. Zilog created their own 16-bit processor that they called it Z8000, Motorola created the 68000, their own 16-bit processor, and National Semiconductor introduced the 16032 device (later to be renamed the 32016). The designers of these chips had different design goals than Intel. Primarily, they were more interested in providing a reasonable instruction set for programmers even if their code density was not anywhere near as high as the 8086. The Motorola and National offers even provided 32-bit integer registers, making programming the chips even easier. All in all, these chips were much better (from a software development standpoint) than the Intel chip.

Intel was not resting on its laurels with the 8086. Immediately after the release of the 8086 they created an eight-bit version, the 8088. The purpose of this chip was to reduce system cost (a minimal system was possible with half the memory chips and cheaper peripherals since the 8088 had an 8-bit data bus). In the very early 1980's, Intel also began work on their intended successor to the 8086—the iAPX432 CPU.

In 1980 a small group at IBM got the go-ahead to create a "personal computer" along the lines of the Apple II and TRS-80 computers (the most popular PCs at the time). IBM's engineers probably evaluated lots of different CPUs and system designs. Ultimately, they settled on the 8088 chip. Most likely they chose this chip because they could create a minimal system with only 16 kilobytes of RAM and a set of cheap eight-bit peripheral devices. So Intel's design goals of creating CPUs that worked well in low-cost systems landed them a very big "design win" from IBM.

Intel was still hard at work on the (ill-fated) iAPX432 project, but a funny thing happened - IBM PCs started selling far better than anyone had ever dreamed. As the popularity of the IBM PCs increased (and as people began "cloning" the PC), lots of software developers began writing software for the 8088 (and 8086) CPU, mostly in assembly language. In the meantime, Intel was pushing their iAPX432 with the Ada programming language (which was supposed to be the next big thing after Pascal, a popular language at that time). Unfortunately for Intel, no one was interested in the '432. Their PC software, written mostly in assembly language would not run on the '432 and the '432 was notoriously slow. It took a while, but the iAPX432 project eventually died off completely and remains a black spot on Intel's record to this day.

Intel was not sitting pretty on the 8086 and 8088 CPUs, however. In the late 1970's and early 1980's they developed the 80186 and 80188 CPUs. These CPUs, unlike their previous CPU offerings, were fully upwards compatible with the 8086 and 8088 CPUs. In the past, whenever Intel produced a new CPU it did not necessarily run the programs written for the previous processors. For example, the 8086 did not run 8080 software and the 8080 did not run 4040 software. Intel, recognizing that there was a tremendous investment in 8086 software, decided to create an upgrade to the 8086 that was superior (both in terms of hardware capability and with respect to the software it would execute). Although the 80186 did not find its way into many PCs, it was a very popular chip in embedded applications (i.e., non-computer devices that use a CPU to control their functions). Indeed, variants of the 80186 are in common use even today.

The unexpected popularity of the IBM PC created a problem for Intel. This popularity obliterated the assumption that designers would be willing to switch to a better chip when such

a chip arrived, even if it meant rewriting their software. Unfortunately, IBM and tens of thousands of software developers were not willing to do this to make life easy for Intel. They wanted not only to stick with the 8086 software they had written but they also wanted something a little better than the 8086. If they were going to be forced into jumping ship to a new CPU, the Motorola, Zilog, and National offerings were starting to look pretty good. So Intel did something that saved their bacon and has infuriated computer architects ever since: they started creating upwards compatible CPUs that continued to execute programs written for previous members of their growing CPU family while adding new features.

As noted earlier, memory was very expensive when Intel first designed the 8086 CPU. At that time, computer systems with a megabyte of memory usually cost megabucks. Intel was expecting a typical computer system employing the 8086 to have somewhere between 4 kilobytes to 64 kilobytes of memory. So when they designed in a one-megabyte limitation, they feared no one would ever install that much memory in a system. Of course, by 1983 people were still using 8086 and 8088 CPUs in their systems and memory prices had dropped to the point where it was very common to install 640 kilobytes of memory on a PC (the IBM PC design effectively limited the amount of RAM to 640 kilobytes even though the 8086 was capable of addressing one megabyte). By this time software developers were starting to write more sophisticated programs and users were starting to use these programs in more sophisticated ways. The bottom line was that everyone was bumping up against the one-megabyte limit of the 8086. Despite the investment in existing software, Intel was about to lose their cash cow if they did not do something about the memory addressing limitations of their 8086 family (the 68000 and 32016 CPUs could address up to 16 megabytes at the time and many system designers [e.g., Apple] were defecting to these other chips). So Intel introduced the 80286 that was a big improvement over the previous CPUs. The 80286 added lots of new instructions to make programming a whole lot easier and they added a new “protected” mode of operation that allowed access to as much as 16 megabytes of memory. They also improved the internal operation of the CPU and bumped up the clock frequency so that the 80286 ran about 10 times faster than the 8088 in IBM PC systems.

IBM introduced the 80286 in their IBM PC/AT (AT = “advanced technology”). This change proved enormously popular. PC/AT clones based on the 80286 started appearing everywhere and Intel’s financial future was assured.

Realizing that the 80x86 (x = “1”, “2”, “3” or “4”) family was a big money maker, Intel immediately began the process of designing new chips that continued to execute the old code while improving performance and adding new features. Intel was still playing catch-up with their competitors in the CPU arena with respect to features, but they were definitely the king of the hill with respect to CPUs installed in PCs. One significant difference between Intel’s chips and many of their competitors was that their competitors (notably Motorola and National) had a 32-bit internal architecture while the 80x86 family was stuck at 16-bits. Again, concerned that people would eventually switch to the 32-bit devices their competitors offered, Intel upgraded the 80x86 family to 32-bits by adding the 80386 to the product line.

The 80386 was truly a remarkable chip. It maintained almost complete compatibility with the previous 16-bit CPUs while fixing most of the real complaints people had with those older chips. In addition to supporting 32-bit computing, the 80386 also bumped up the maximum addressability

to four gigabytes as well as solving some problems with the "segmented" organization of the previous chips (a big complaint by software developers at the time). The 80386 also represented the most radical change to ever occur in the 80x86 family. Intel more than doubled the total number of instructions, added new memory management facilities, added hardware debugging support for software, and introduced many other features. Continuing the trend they set with the 80286, the 80386 executed instructions faster than previous generation chips, even when running at the same clock speed plus the new chip ran at a higher clock speed than the previous generation chips. Therefore, it ran existing 8088 and 80286 programs faster on these older chips. Unfortunately, while people adopted the new chip for its higher performance, they did not write new software to take advantage of the chip's new features.

Although the 80386 represented the most radical change in the 80x86 architecture from the programmer's view, Intel was not done wringing all the performance out of the 80x86 family. By the time the 80386 appeared, computer architects were making a big noise about the so-called RISC (Reduced Instruction Set Computer) CPUs. While there were several advantages to these new RISC chips, an important advantage of these chips is that they purported to execute one instruction every clock cycle. The 80386 instructions required a widely varying number of cycles to execute ranging from a few cycles per instruction to well over a hundred. Although comparing RISC processors directly with the 80386 was dangerous (because many 80386 instructions actually did the work of two or more RISC instructions), there was a general perception that, at the same clock speed, the 80386 was slower since it executed fewer instructions in a given amount of time.

The 80486 CPU introduced 2-major advances in the 80x86 design. First, the 80486 integrated the *floating-point unit* (or FPU) directly onto the CPU die. *Prior* to this point Intel supplied a separate, *external,* chip to provide floating-point calculations (these were the 8087, 80287, and 80387 devices). By incorporating the FPU with the CPU, Intel was able to speed up floating-point operations and provided this capability at a lower cost (at least on systems that required floating-point arithmetic). The second major architectural advance was the use of *pipelined instruction execution.* This feature (which we will discuss in detail a little later) allowed Intel to overlap the execution of two or more instructions. The end result of pipelining is that they effectively reduced the number of cycles each instruction required for execution. With pipelining, many of the simpler instructions had an aggregate throughput of one instruction per clock cycle (under ideal conditions) so the 80486 was able to compete with RISC chips in terms of clocks per instruction cycle.

While Intel was busy adding pipelining to their 80x86 family, the companies building RISC CPUs were not standing still. To create ever faster and faster-CPU offerings, RISC designers began creating *superscalar* CPUs that could actually execute more than one instruction per clock cycle. Once again, Intel's CPUs were perceived as following the leaders in terms of CPU performance. Another problem with Intel's CPU is that the integrated FPU, though faster than the earlier models, was significantly slower than the FPUs on the RISC chips. As a result, those designing high-end engineering workstations (that typically require good floating-point hardware support) began using the RISC chips because they were faster than Intel's offerings.

From the programmer's perspective, there was very little difference between an 80386 with an 80387 FPU and an 80486 CPU. There were only a handful of new instructions (most of which

had very little utility in standard applications) and not much in the way of other architectural features that software could use. The 80486, from the software engineer's point of view, was just a really fast 80386/80387 combination.

So Intel went back to their CAD tools and began work on their next CPU. This new CPU featured a superscalar design with vastly improved floating-point performance. Finally, Intel was closing in on the performance of the RISC chips. Like the 80486 before it, this new CPU added only a small number of new instructions and most of those were intended for use by operating systems, not application software.

Intel did not designate this new chip the 80586. Instead, they called it the *Pentium Processor.* The reason they discontinued referring to processors by number and started naming them was because of confusion in the market place. Intel was not the only company producing 80x86 compatible CPUs. AMD, Cyrix, and a host of others were also building and selling these chips in direct competition with Intel. Until the 80486 came along, the internal design of the CPUs were relatively simple and even small companies could faithfully reproduce the functionality of Intel's CPUs. The 80486 was a different story altogether. This chip was quite complex and taxed the design capabilities of the smaller companies. Some companies, like AMD, actually licensed Intel's design and they were able to produce chips that were compatible with Intel's (since they were, effectively, Intel's chips). Other companies attempted to create their own version of the 80486 and fell short of the goal. Perhaps they did not integrate an FPU or the new instructions on the 80486. Many did not support pipelining. Some chips lacked other features found on the 80486. In fact, most of the (non-Intel) chips were really 80386 devices with some very slight improvements. Nevertheless, they called these chips 80486 CPUs.

This created massive confusion in the market place. Prior to this, if we would purchase a computer with an 80386 chip we knew the capabilities of the CPU. All 80386 chips were equivalent. However, when the 80486 came along and we purchased a computer system with an 80486, we did not know if we were getting an actual 80486 or a remarked 80386 CPU. To counter this, Intel began their enormously successful "Intel Inside" campaign to let people know that there was a difference between Intel CPUs and CPUs from other vendors. This marketing campaign was so successful that people began specifying Intel CPUs even though some other vendor's chips (i.e., AMD) were completely compatible.

Not wanting to repeat this problem with the 80586 generation, Intel ditched the numeric designation of their chips. They created the term "Pentium Processor" to describe their new CPU so they could trademark the name and prevent other manufacturers from using the same designation for their chip. Initially, of course, savvy computer users griped about Intel's strong-arm tactics but the average user benefited quite a bit from Intel's marketing strategy. Other manufacturers release their own 80586 chips (some even used the "586" designation), but they could not use the Pentium Processor name on their parts so when someone purchased a system with a Pentium in it, they knew it was going to have all the capabilities of Intel's chip since it had to be Intel's chip. This was a good thing because most of the other '586 class chips that people produced at that time were, not as powerful as the Pentium.

The Pentium cemented Intel's position as champion of the personal computer. It had near RISC performance and ran tons of existing software. Only the Apple Macintosh and high-end

UNIX workstations and servers went the RISC route. Together, these other machines comprised less than 10% of the total desktop computer market. Intel still was not satisfied. They wanted to control the server market as well. So they developed the Pentium Pro CPU. The Pentium Pro had a couple of features that made it ideal for servers. Intel improved the 32-bit performance of the CPU (at the expense of its 16-bit performance), they added better support for multiprocessing to allow multiple CPUs in a system (high-end servers usually have 2 or more processors), and they added a handful of new instructions to improve the performance of certain instruction sequences on the pipelined architecture. Unfortunately, most application software written at the time of the Pentium Pro's release was 16-bit software that actually ran slower on the Pentium Pro than it did on a Pentium at equivalent clock frequencies. So although the Pentium Pro did wind up in a few server machines, it was never as popular as the other chips in the Intel line.

The Pentium Pro had another big strike against it: shortly after the introduction of the Pentium Pro, Intel's engineers introduced an upgrade to the standard Pentium chip, the MMX (multimedia extension) instruction set. These new instructions (nearly 60 in all) gave the Pentium additional power to handle computer video and audio applications. These extensions became popular overnight, putting the last nail in the Pentium Pro's coffin. The Pentium Pro was slower than the standard Pentium chip and slower than high-end RISC chips, so it did not see much of Sunrise.

Intel corrected the 16-bit performance in the Pentium Pro, added the MMX extensions and called the result the Pentium II. The Pentium II demonstrated an interesting point. Computers had reached a point where they were powerful enough for most people's everyday activities. Prior to the introduction of the Pentium II, Intel (and most industry pundits) had assumed that people would always want more power out of their computer systems. Even if they did not need the machines to run faster, surely the software developers would write larger (and slower) systems requiring more and more CPU power. The Pentium II proved this idea wrong. The average user needed e-mail, word processing, Internet access, multimedia support, simple graphics editing capabilities, and a spreadsheet now and then. Most of these applications, at least as home users employed them, were fast enough on existing CPUs. The applications that were slow (e.g., Internet access) were generally beyond the control of the CPU (i.e., the modem was the bottleneck not the CPU). As a result, when Intel introduced their pricey Pentium II CPUs, they discovered that system manufacturers started buying other people's 80x86 chips because they were far less expensive and quite suitable for their customer's applications. This nearly stunned Intel since it contradicted their experience up to that point.

Realizing that the competition was capturing the low-end market and stealing sales away, Intel devised a low-cost (lower performance) version of the Pentium II that they named *Celeron.* The initial Celerons consisted of a Pentium II CPU without the on-board level two caches. Without the cache, the chip ran only a little bit better than half the speed of the Pentium II part. Nevertheless, the performance was comparable to other low-cost parts so Intel's fortunes improved once more.

While designing the low-end Celeron, Intel had not lost sight of the fact that they wanted to capture a chunk of the high-end workstation and server market as well. So they created a third version of the Pentium II, the Xeon Processor with improved cache and the capability of multiprocessor more than two CPUs. The Pentium II supports a two CPU multiprocessor system

but it is not easy to expand it beyond this number; the Xeon processor corrected this limitation. With the introduction of the Xeon processor (plus special versions of Unix and Windows NT), Intel finally started to make some serious inroads into the server and high-end workstation markets.

We can probably imagine what followed the Pentium II. The Pentium III introduced the SIMD (pronounced SIM-DEE) extensions to the instruction set. These new instructions provided high performance floating point operations for certain types of computations that allow the Pentium III to compete with high-end RISC CPUs. The Pentium III also introduced another handful of integer instructions to aid certain applications.

With the introduction of the Pentium III, nearly all-serious claims about RISC chips offering better performance were fading away. In fact, for most applications, the Intel chips were actually faster than the RISC chips available at the time. Next, of course, Intel introduced the Pentium 4 chip (it was running at 2 GHz as this was being written, a much higher clock frequency than its RISC contemporaries). An interesting issue concerning the Pentium 4 is that it does not execute code faster than the Pentium III when running at the same clock frequency (it runs slower, in fact). The Pentium IV makes up for this problem by executing at a much higher clock frequency than was possible with the Pentium III. One would think that Intel would soon own it all. Surely by the time of the Pentium V, the RISC competition would not be a factor anymore.

There is one problem with this theory: even Intel is admitting that they have pushed the 80x86 architecture about as far as they can. For nearly 20 years, computer architects have blasted Intel's architecture as being gross and bloated having to support code written for the 8086 processor way back in 1978. Indeed, Intel's design decisions (like high instruction density) that seemed so important in 1978 are holding back the CPU today. So-called "clean" designs, that do not have to support legacy applications, allow CPU designers to create high-performance CPUs with far less effort than Intel's. Worse, those decisions Intel made in the 1976-1978 time frame are beginning to catch up with them and will eventually stall further development of the CPU. Computer architects have been warning everyone about this problem for twenty years; it is a testament to Intel's design effort (and willingness to put money into R&D) that they have taken the CPU as far as they have.

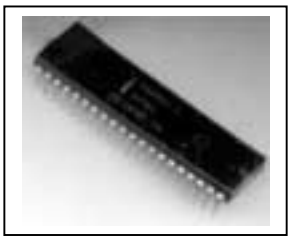
The biggest problem on the horizon is that most RISC manufacturers were extending their architectures to 64-bits. This had 2-important impacts on computer systems. First, arithmetic calculations will be somewhat faster as will many internal operations and second, the CPUs will be able to directly address more than 4-gigabytes of main memory. This last factor is probably the most important for server and workstation systems. Already, high-end servers have more than 4-gigabytes installed. In future, the ability to address more than 4-gigabytes of physical RAM will become essential for servers and high-end workstations. As the price of a gigabyte or more of memory drops below $100, we will see low-end personal computers with more than 4-gigabytes installed. To effectively handle this kind of memory, Intel needs a 64-bit processor to compete with the RISC chips.

Perhaps Intel has seen the light and decided to give up on the 80x86 architecture. Towards the middle to end of the 1990's Intel announced that they in partnership with Hewlet-Packard are going to create a new 64-bit processor based around HP's PA-RISC architecture. This new

64-bit chip would execute 80x86 code in a special "emulation" mode and run native 64-bit code using a new instruction set. It was too early to tell if Intel would be successful with this strategy, but there were some major risks (pardon the pun) with this approach. The first such CPUs (just becoming available as this is being written) ran 32-bit code far slower than the Pentium III and IV chips. Not only does the emulation of the 80x86 instruction set slow things down, but the clock speeds of the early CPUs were half the speed of the Pentium IVs. This is roughly the same situation Intel had with the Pentium Pro running 16-bit code slower than the Pentium. Second, the 64-bit CPUs (the IA64 family) rely heavily on compiler technology and are using a commercially untested architecture. This is similar to the situation with the iAPX432 project that failed quite miserably. Hopefully Intel knows what they are doing and ten years from then we will all be using IA64 processors and wondering why anyone ever stuck with the IA32.

Intel is betting that people will move to the IA64 when they need 64-bit computing capabilities. AMD, on the other hand, is betting that people would rather have a 64-bit 80x86 processor. Although the details are sketchy, AMD has announced that they will extend the 80x86 architecture to 64-bits in much the same way that Intel extends the 8086 and 80286 to 32-bits with the introduction of the 80386 microprocessor. Only time will tell if Intel or AMD (or both) are successful with their visions.

Intel 4004 chip



Intel 8080

Intel Pentium 4 processor

1.2.1 Microprocessor Progress: Intel

Table 1.2.1 helps us to understand the differences between the different processors that Intel has introduced over the years.

Table 1.2.1 Comparison of different microprocessors

Name	*Date*	*Transistors used*	*Microns*	*Max clock speed*	*Address lines*	*Data width*	*MIPS*	*Max addressable memory*
8080	1974	6 K	6	2 MHz	16-bit	8-bit	0.64	64KB, no cache
8086	1978	29 K	6	8 MHz	20-bit	16-bit	0.80	1MB, no cache
8088	1979	29 K	3	5 MHz	20-bit	8-bit bus/ 16-bit	0.33	1MB, no cache
80286	1982	134 K	1.5	12.5 MHz	24-bit	16-bit	2.7	16MB, no cache
80386	1985	275 K	1.5	20 MHz	32-bit	32 bit	6	4GB, no cache
80486	1989	1.2 M	1	25 MHz	32-bit	32 bit	20	4GB, 8K level 1

Pentium	1993	3.1M	0.8	100 MHz	32-bit	64-bit/32-bit	100	4GB, 16K level 1
Pentium Pro	1995	5.5M	0.8	440 MHz	32-bit	64-bit/32-bit	440	64GB, 16K level 1 256K/512 level 2
Pentium II	1997	7.5M	0.35	266 MHz	32-bit	64-bit/32-bit	446	64GB, 32K level 1 256K/512 level 2
Pentium III	1999	9.5M	0.25	500 MHz	32-bit	64-bit/32-bit	1000	64GB, 32K level 1 256K/512 level 2
Pentium 4	2000	42M	0.18	1.5 GHz	32-bit	64-bit/32-bit	1,700	64GB, 32K level 1 256K/512 level 2
Pentium 4 Prescott	2004	125M	0.09	3.6 GHz	32-bit	64-bit/32-bit	7,000	64GB, 32K level 1 256K/512 level 2

Information about this table

- **The date** is the year that the processor was 1st introduced. Many processors are re-introduced at higher clock speeds for many years after the original release date.
- **Transistors mean** the number of transistors used on the chip. We can see that the number of transistors on a single chip has risen steadily over the years.
- **Microns** indicate the width, in microns, of the thinnest wire on the chip. For comparison, a human hair is 100 microns thick. As the feature size on the chip goes down, the number of transistors rises.
- **Clock speed** stands for the maximum rate that the chip can be clocked at. Clock speed will make more sense later.
- **Data Width** is the width of the ALU in bits. A 8-bit ALU can add/ subtract/ multiply/ etc. two 8-bit numbers, while a 32-bit ALU can manipulate 32-bit numbers. A 8-bit ALU would have to execute 4-instructions to add two 32 -bit numbers, while a 32-bit ALU can do it in one instruction. In many cases, the external data bus is of the same width as the ALU, but not always. The 8088 had a 16-bit ALU and a 8-bit external data bus, while the modern Pentiums fetch data 64-bits at a time for their 32-bit ALUs.
- **MIPS** stands for ‘millions of instructions per second’ and is a rough measure of the performance of a CPU. Modern CPUs can do so many different things that MIPS ratings lose a lot of their meaning, but we can get a general sense of the relative power of the CPU from this column.

From Table 1.2.1 we can infer that, in general, there is a relationship between clock speed and MIPS. The maximum clock speed is a function of the manufacturing process and delays within the chip. There is also a relationship between the number of transistors and MIPS. For example, the 8088 clocked at 5MHz but only executed at 0.33 MIPS (about one instruction per 5/0.33 = 15 clock cycles). Modern processors can often execute at a rate of 3.6/7 = 0.5 clocks/ instruction i.e., 2-instructions per clock cycle. That improvement is directly related to the number of transistors on the chip and will make more sense later.

1.3 HOW MICROPROCESSOR WORKS

To understand how a microprocessor works, it is helpful to look inside and learn about the logic used to create one. In the process we can also learn about **assembly language** → the native language of a microprocessor → and many of the things that engineers can do to boost the speed of a processor.

A microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does 3-basic things:

- Using its ALU (Arithmetic Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating-point processors that can perform extremely sophisticated operations on large floating-point numbers.
- A microprocessor can move data from one memory location to another.
- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its 3-basic activities. The diagram shown in Fig.1.3 is an extremely simple representation of the microprocessor capable of doing those 3-things.

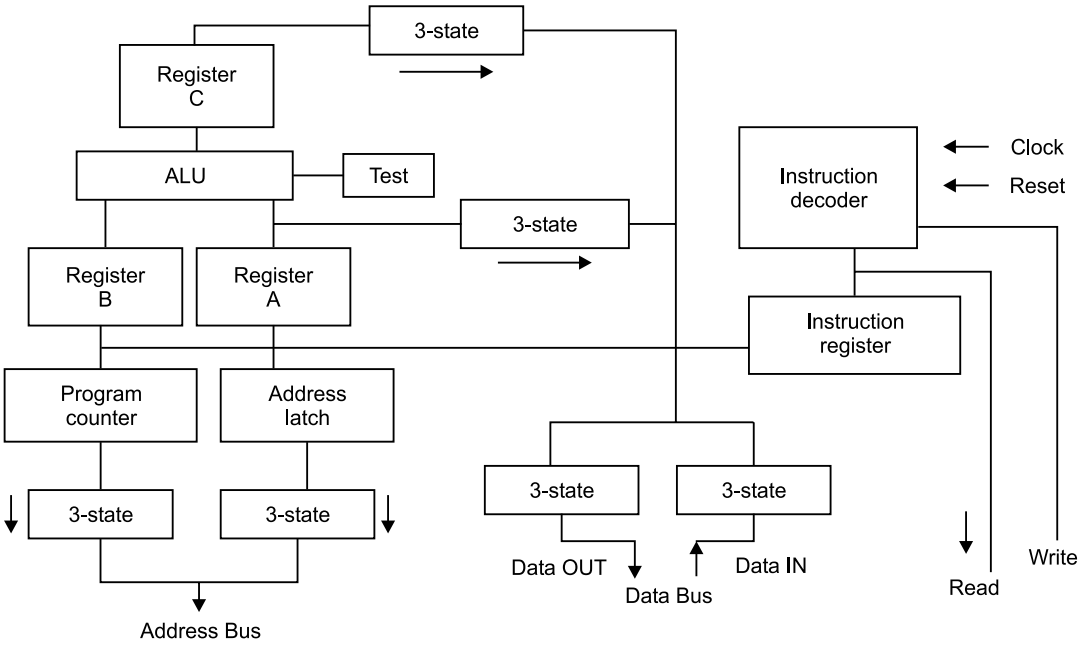


Fig. 1.3 Block diagram of microprocessor

This is about as simple as a microprocessor gets. This microprocessor has:

- **An address bus** (it may be 8, 16, 32 or 64-bits wide) that sends an address to memory or I/O devices.

- **A data bus** (it may be 8, 16, 32 or 64-bits wide)'that can send data to memory or receive data from memory.
- An $\overline{RD}$ (read bar) and $\overline{WR}$ (write bar) line to tell the memory whether it wants to receive or deliver to it.
- **A clock line** that sequences the processor clock pulses.
- **A reset line** that resets the program counter to zero (or whatever) and restarts execution.

Let us assume that both the address and data buses are 8-bits wide in this example. Here are the components of this simple microprocessor:

- Registers A, B and C are simply registers (latches) made out of 8-flip-flops.
- The address registers (latch) is just like registers A, B and C.
- The program counter is a register with the extra ability to increment by 1 and also to reset to zero when told to do so.
- The ALU could be as simple as an 8-bit adder, or it might be able to add, subtract, multiply and divide 8-bit data.
- The test register is a special latch that can hold values from comparisons performed in the ALU. A ALU can normally compare 2-numbers and determine if they are equal, if one is greater than the other, etc. The test register can also normally hold a carry bit from the last stage of the adder. It stores these values in flip-flops and then the instruction decoder can use the values to make decisions.
- There are 6-boxes marked '3-state' in the diagram. These are **tri-state buffers.** A tri-state buffer can pass a 1, a 0 or it can essentially disconnect its output (imagine a switch that totally disconnects the output line from the wire that the output is heading toward). A tri-state buffer allows multiple outputs to connect to a wire, but only one of them to actually drive a 1 or a 0 onto the line.
- The instruction register and instruction decoder are responsible for controlling all of the other components.

Although they are not shown in this diagram, there would be control lines from the instruction decoder that would:

- Tell the A register to latch the value currently on the data bus
- Tell the B register to latch the value currently on the data bus
- Tell the C register to latch the value currently output by the ALU
- Tell the program counter register to latch the value currently on the data bus
- Tell the address register to latch the value currently on the data bus
- Tell the instruction register to latch the value currently on the data bus
- Tell the program counter to increment
- Tell the program counter to reset to zero
- Activate any of the 6-tri-state buffers (6-separate lines)
- Tell the ALU what operation to perform

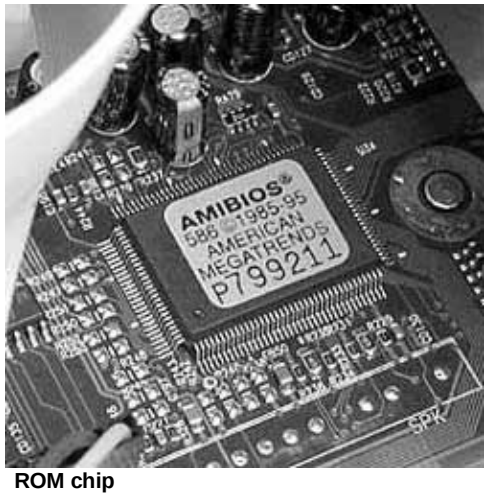
- Tell the test register to latch the ALU's test bits
- Activate the $\overline{RD}$ line
- Activate the $\overline{WR}$ line

Coming into the instruction decoder are the bits from the test register and the clock line, as well as the bits from the instruction register.

1.3.1 ROM and RAM

The previous section talked about the address and data buses, as well as the $\overline{RD}$ and $\overline{WR}$ lines. These buses and lines connect either to RAM or ROM, generally both. In our sample microprocessor, we have an address bus 8-bit wide and data bus 8-bit wide. That means that the microprocessor can address (2^8) 256 bytes of memory, and it can read or write 8-bit of the memory at a time. Let us assume that this simple microprocessor has 128D location of ROM starting at address 0D (0000H) to 127D(007FH) locations and RAM starting at address 128D (0080H) to 257D (00FFH) locations. Typical look of the ROM and RAM chips is shown in Fig.1.3.1.

By the way, nearly all computers contain some amount of ROM (it is possible to create a simple computer that contains no RAM. Many microcontrollers do this by placing a handful of RAM bytes on the processor chip itself, but generally impossible to create one that contains no ROM). On a PC, the ROM is called the BIOS (Basic Input/Output System). When the microprocessor starts, it begins executing instructions found in the BIOS. The BIOS instructions do things like **test** the hardware in the machine, and then it goes to the **hard disk** to fetch the **boot sector.** This boot sector is another small program, and the BIOS **stores** it in RAM after reading it off the disk. The microprocessor then begins executing the boot sector's instructions from RAM. The boot sector program will tell the microprocessor to fetch something else from the hard disk into RAM that the microprocessor then executes, and so on. This is how the microprocessor loads and executes the entire operating system.



ROM chip


RAM chip

Fig.1.3.1 Typical ROM and RAM chips

1.3.2 Microprocessor Instructions

Even the incredibly simple microprocessor shown in the previous example will have a fairly large set of instructions that it can perform. The collection of instructions is implemented as bit patterns, each one of which has a different meaning when loaded into the instruction register. Humans are not particularly good at remembering bit patterns, so sets of short words are defined to represent the different bit patterns. This collection of words is called the **assembly language** of the processor. An **assembler** can translate the words into their bit patterns very easily, and then the output of the assembler is placed in memory for the microprocessor to execute.

How Instructions Work

The instruction decoder needs to turn each of the op codes into a set of signals that drive the different components inside the microprocessor. Let us take the ADD instruction as an example and look at what it needs to do:

1. During the 1st clock cycle, we need to actually load the instruction. Therefore the instruction decoder needs to:
   - Activate the tri-state buffer for the program counter
   - Activate the $\overline{\text{RD}}$ line
   - Activate the data-in tri-state buffer
   - Latch the instruction into the instruction register
2. During the 2nd clock cycle, the add instruction is decoded. It needs to do very little:
   - Set the operation of the ALU to addition
   - Latch the output of the ALU into the C register
3. During the 3rd clock cycle, the program counter is incremented (in theory this could be overlapped into the 2nd clock cycle).

Every instruction can be broken down as a set of sequenced operations like these that manipulate the components of the microprocessor in the proper order. Some instructions, like this ADD instruction, might take 2 or 3-clock cycles. Others might take 5 or 6-clock cycles.

1.3.3 Microprocessor Performance

The number of **transistors** available has a huge effect on the performance of a processor. As seen earlier, a typical instruction in a processor like an 8088 took 15-clock cycles to execute. Because of the design of the multiplier, it took approximately 80-cycles just to do one 16-bit multiplication on the 8088. With more transistors, much more powerful multipliers capable of single-cycle speeds become possible.

More transistors also allow for a technology called **pipelining.** In a pipelined architecture, instruction execution overlaps. So even though it might take 5-clock cycles to execute each instruction, there can be 5-instructions in various stages of execution simultaneously. That way it looks like 1-instruction completes every clock cycle.

Many modern processors have multiple instruction decoders, each with its own pipeline. This allows for multiple instruction streams, which means that more than 1-instruction can complete during each clock cycle. This technique can be quite complex to implement, so it takes lots of transistors.

1.4 CONVENTIONAL/UNCONVENTIONAL ARCHITECTURE

The conventional von Neumann computer architecture continued for over 30-years. Its main features were

- Single computing unit incorporating processor, memory, and communication units.
- Linear memory of a fixed size
- Central control
- Machine language

The conventional von Neumann architecture reached the limit for achieving higher processing speed because of their sequential fetching of instructions and data through a common memory interface. The advanced microprocessors had to choose a completely different philosophy of architecture design to increase the processing power: parallelism of some form to avoid bottleneck of von Neumann architecture.

A different method of classifying the computer architecture is ***Flynn's*** taxonomy based on how the computer relates its instructions to the data being processed. Figure 1.4 explains this.

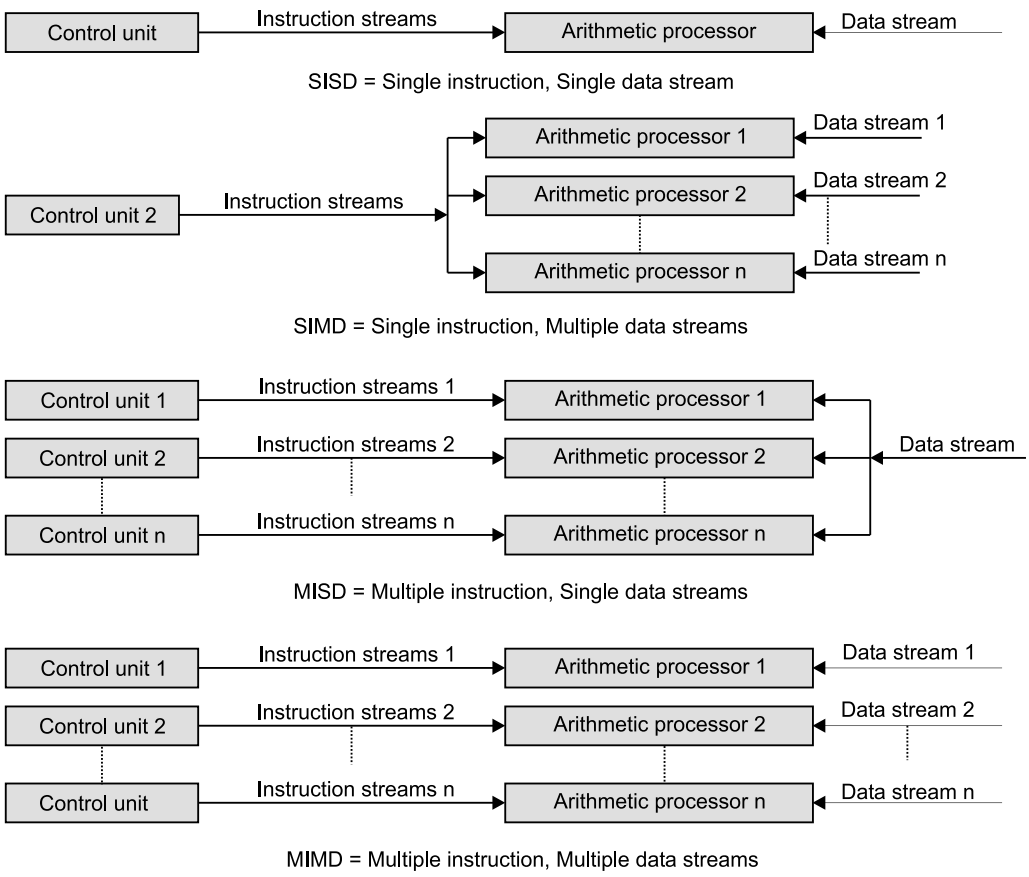


Fig. 1.4 SISD, SIMD, and MIMD Architecture

SISD

This stands for single instruction and single data streams. This is based on the conventional von Neumann architecture. Here, there is only one stream of instruction and each arithmetic instruction initiates one operation. Techniques like pipelining and cache memory may be used to speed up the performance.

SIMD

This stands for single instruction and multiple data streams. This architecture has single instruction stream. But this single instruction is vectored type that initiates many operations. Each element of a vector is regarded as a member of a separate data stream resulting into multiple data streams. For an instance, one instruction can operate on array of data.

MIMD

The word MIMD stands for multiple instruction and multiple data stream. The multiple instruction streams necessarily requires the presence of multiple processors and therefore several data streams are also required. The simple example of such configuration is the multiprocessor and array of microprocessors.

The MIMD architecture is not in common use.

1.5 POPULAR MICROPROCESSOR MANUFACTURERS

The 4-most popular 32-bit microprocessors with conventional architecture available in the market are

- Motorola MC68020,
- National Semiconductor NS32032,
- Zilog Z80000, and
- Intel 80386.

These chips offered significant advantages of their 16-bitters counterparts as

- Higher data thruput having 32-bit data path
- Large direct addressing (2^{32} = 4GB)
- Higher clock frequencies
- Higher processing speeds because larger registers require fewer calls to memory and register-to-register transfer is typically 5-times faster than register-to-memory transfers.
- More instructions and addressing modes to improve the software efficiencies,
- More extensive memory management
- Robust built-in cache
- On-chip coprocessors
- More registers to support high-level language.

The basic requirements for the construction of a 32-bit microprocessor are

- ALU,
- Efficient memory management unit (MMU),

- Floating-point coprocessor,
- Interrupt controller,
- Accurate timing control unit.

Some manufacturers have included several of these feature on the microprocessor chips whereas others have included provisions for interfacing individual chips for such elaborate functions. Each manufacturer approaches the design in different way that is suitable from their point of view. After a new processor is introduced, it is modified, diced, and scaled down or scaled up for long time and then sold under different brand names.

Pentium 4 : Prescott

Prescott is the working title for the next generation of the Pentium 4-architecture. Prescott is manufactured with a line-width of 90nm compared to the 1st generation (Willamette) that was manufactured width of 180nm. The previous generation (Northwood) had a line width of 130nm. At every new generation Intel has been able to fit twice as many transistors as the previous generation. The processor also becomes cheaper to manufacture by decreasing the circuit surface. Usually the manufacturer chooses to do both. The 1st generation had a L2 cache of 256KB while today's generation has a cache of 512KB. Prescott will have a 1MB cache and an 800MHz front side bus (FSB). In conjunction, this will improve the processor performance significantly. Fewer transistors also allow higher clock frequencies and thus even better performance. The 1^{st} model of Prescott is expected to have a clock frequency of 3.5GHz. An integrated security function will also prevent intruders from reading or writing to the hard drive.

Designed for multimedia

Prescott is complemented with 13-additional processor instructions unavailable in Pentium-4. According to Intel these instructions are of significant use for the future use of PCs. This might indicate that the instructions are intended to improve video performance.

Rumors have previously claimed that the new instructions would be 64-bit instructions in order to compete with the upcoming 32/64-bit processor "Claw hammer" from AMD. Instructions in 64-bits makes it possible to address larger quantities of data than what today's 32-bit processor can handle. The "Hammer" from AMD is expected to be much faster than today's processor and compete with Intel's 64-bit Itanium-processors while being much cheaper. This extra 64-bit instruction only takes up a few percent of the circuit size, thus making manufacturing cheap for Intel.

64-bit processors

The 64-bit processors have been with us since 1992, and in 21st century they have started to become mainstream. Both Intel and AMD have introduced 64-bit chips, and the Mac G5 supports a 64-bit processor. The 64-bit processors have 64-bit ALUs, 64-bit registers, 64-bit buses and so on.

One reason why the world needs 64-bit processors is because of their **enlarged address spaces.** The 32-bit chips are often constrained to a maximum of $2^{32} = 2^{30} \times 2^2 = 4$GB of RAM access. That sounds like a lot, given that most home computers currently use only 256MB to 512MB of RAM. However, a 4GB limit can be a severe problem for **server** machines and machines **running large databases.** Even home machine will start bumping up against the 2GB

or 4GB limit pretty soon if current trends continue. A 64-bit chip has none of these constraints because a 64-bit RAM address space is essentially infinite for the foreseeable future. The $2^{64} = 2^{60} \times 2^4$ bytes of RAM is something on the order of a quadrillion gigabytes of RAM.

With a 64-bit wide address bus and high-speed data buses on the motherboard, 64-bit machines also offer faster I/O (Input/Output) speeds to things like hard disk drives and video cards. These features can greatly increase system performance.

Servers can definitely benefit from 64-bits, but what about normal users? Beyond the RAM solution, it is not clear that a 64-bit chip offers 'normal users' any real, tangible benefits at the moment. They can process data (very complex data features lots of real numbers) faster. People doing video editing and people doing photographic editing on very large images benefit from this kind of computing power. High-end games will also benefit, once they are re-coded to take advantage of 64-bit features. But the average user who is reading e-mail, browsing the Web and editing Word documents is not really using the processor in that way. In addition, operating system like Windows XP has not yet been upgraded to handle 64-bit CPUs. Because of the lack of tangible benefits, it will be 2010 or so before we see 64-bit machines on every desktop.

A processor for 4GHz and more

Prescott is expected to use clock frequencies of 4GHz and more. Internal units to handle integers, registers, address generator and more will be added. So is it a 4GHz Pentium 4 or an 8GHz Pentium 5?

Will the CPU become more expensive?

Will Prescott become more expensive than today's Pentium 4? The answer is probably no since the size of the circuit used is less than that for Pentium 4. If AMD will manage to compete successfully with Intel also next year the Prescott will not become more expensive than today's processor while offering much better performance.

Pentium: Dual Core

Intel tests on its first dual-core desktop processors show that it would deliver some real benefit when we run software on them that is designed to take advantage of the two cores, or when we are performing multiple tasks simultaneously → running a virus check while surfing the Web, for instance.

As the name implies, dual-core processors incorporate two physical processors and two L2 memory caches into one piece of silicon, functioning, in theory, like two separate processors. Intel's first dual-core chip, the 3.2 GHz Pentium Extreme Edition 840 (which carries 1MB of L2 cache per core), goes one step further by including Intel's Hyper-Threading technology in each core, which theoretically brings us a "virtual second processor per core."

Multiply your multiple experience

Combined with a digital media adaptor and home network, an Intel Pentium D processor-based PC allows two people to share the content from one PC – in the same room, or even from different places in the house. For example, one person can check e-mail while the other uses a remote control to access digital photos stored on the same PC and view them on the TV from the comfort of their living room.