

**MICROPROCESSORS**

**HISTORY OF COMPUTING**

**NOUF ASSAID**

## Table of Contents

<b>Introduction</b>	<b>2</b>
<b>Brief History</b>	<b>2</b>
<b>Microprocessors</b>	<b>7</b>
<b>Instruction Set Architectures</b>	<b>8</b>
<b>Von Neumann Machine</b>	<b>9</b>
<b>Microprocessor Design</b>	<b>12</b>
<b>Superscalar</b>	<b>13</b>
<b>RISC</b>	<b>16</b>
<b>CISC</b>	<b>20</b>
<b>VLIW</b>	<b>23</b>
<b>Multiprocessor</b>	<b>24</b>
<b>Future Trends in Microprocessor Design</b>	<b>25</b>

## **Introduction**

If we take a look around us, we would be sure to find a device that uses a microprocessor in some form or the other. Microprocessors have become a part of our daily lives and it would be difficult to imagine life without them today. From digital wrist watches, to pocket calculators, from microwaves, to cars, toys, security systems, navigation, to credit cards, microprocessors are ubiquitous.

All this has been made possible by remarkable developments in semiconductor technology enabling in the last 30 years, enabling the implementation of ideas that were previously beyond the average computer architect's grasp.

In this paper, we discuss the various microprocessor technologies, starting with a brief history of computing. This is followed by an in-depth look at processor architecture, design philosophies, current design trends, RISC processors and CISC processors. Finally we discuss trends and directions in microprocessor design.

## **Brief Historical Overview**

### **Mechanical Computers**

A French engineer by the name of Blaise Pascal built the first working mechanical computer. This device was made completely from gears and was operated using hand cranks. This machine was capable of simple addition and subtraction, but a few years later, a German mathematician by the name of Leibniz made a similar machine that could multiply and divide as well. After about 150 years, a mathematician at Cambridge, Charles Babbage made his Difference Engine. This was primarily used to computer navigational charts, but could only add or subtract. This machine was designed to run a single algorithm and output the result by punching it into a copper engravers

plate. He then began to design the successor, the Analytical machine. This device had memory, a computational unit, input subsystems (card reader) and output systems (card puncher and printed output). The advantage of this machine was that it was the first general-purpose machine. It could read instructions from punched cards and then carried them out. Since this was one of the first rudimentary programmable machines, it needed assembly software. (Tanenbaum)

### **Vacuum Tubes**

The first vacuum tube machine was the ENIAC (Electronic Numerical Integrator and Computer). It consisted of 18,000 vacuum tubes and 1500 relays. Architecturally, the machine had 20 registers, each capable of holding a 10 digit decimal number.

Programming this behemoth was a herculean task; one had to set up 6000 multi-position switches and connect a multitude of sockets with jumper cables. This was followed by a myriad of other such machines including EDSAC, EDVAC. However, one of the most important inventions of the time was by Jon Von Neumann, who figured out that programming computers with huge number of switches and cables was slow and inflexible. He came to realize that the program could be represented in digital form in the computers memory, along with the data. He also figured that data could be manipulated in parallel rather than in series. The basic design, which he first described, is known as the Von Neumann machine. It was first used in the EDSAC, the first stored program computer, and is still the basis for nearly all digital computers, even now, more than half a century later. This concept will be discussed further later in the paper. (Brey)

## **Transistors**

The transistor was invented at Bell labs in 1948. Within 10 years the transistor revolutionized computers, and by the late 50's, vacuum tube computers were obsolete. The first of these devices were built at Lincoln Lab, MIT, called the TX-0 (Transistorized Experimental Computer – 0). Ken Olsen, one of the engineers working at this lab, formed a company, Digital Equipment Corporation in '57 and in '61 rolled out the PDP-1. This machine had 4K of 18 bit words and a cycle time of 5  $\mu$ sec. The performance was half that of IBM 7090, the transistorized successor to the 709, the fastest computer in the world at the time. The PDP-1 cost \$120,000; the 7090 cost millions. DEC sold dozens of PDP-1's and the minicomputer industry was born. A direct result of the PDP-1 was a visual display and the ability to plot points any where on its 512 by 512 screen. (Tanenbaum).

Within a few years, DEC released the PDP-8, a 12-bit machine, much cheaper than its predecessor, but with one important new invention: a single bus, or 'omnibus' as they called it, a collection of parallel wires used to connect the components of the machine. This was a revolutionary new concept and DEC sold over fifty thousand PDP-8's, establishing themselves as leaders in the minicomputer business. (Reference)

Seymour Cray, a legendary figure, in the same league as Von Neumann started the CDC or Control Data Corporation and introduced the 6600; a highly parallel machine capable of executing several instructions simultaneously. Cray went on to produce such ultra fast machines, now called supercomputers, including the 6600, 7600 and Cray-1. (Reference)

## **Integrated Circuits**

With the invention of the IC, it was possible to make processors smaller, faster and cheaper than with transistors. IBM introduced its System 360 series of machines based on integrated circuits. These systems were designed so that they could perform both scientific as well as business calculations, since the entire family shared the same assembly language. Therefore commercial computing could be performed on low end System /360 Model 30's while scientific computing could be performed on higher end System /360 Model 75's. Theoretically it was possible to use the code written for a Model 30 on a Model 75. This concept was one of the crucial steps in bringing about legacy computer code and led eventually to the computer companies seeking reverse compatibility with their prior generation processors. This gave rise to Instruction Set Architecture. The term ISA was coined by IBM to refer to the family of computers sharing the same assembly code. (Short, Hennessy & Patterson).

## **VLSI**

When the IC was invented, it was feasible to put dozens of transistors on a single chip, but as time passed by, a few dozen became thousands, and tens of thousands and soon millions of transistors could be fit on a single chip. Advancement in technology allowed for smaller traces on the silicon, making it possible to put more and more transistors on a single chip. This allowed for more complex microprocessor design and faster processors.

By the 80's, the prices for the processors had dropped so low, that even individuals could afford small personal computers or PCs. Initially sold as kits, PCs started to become increasingly popular when IBM decided that it should get into the

market. Intel's 8088 was the processor of choice and they built the machine from commercially available parts. This was introduced in 1981 and instantly became one of the best selling computer in history. However, in an effort to push the sales of PCs even further by allowing other companies to make plug in boards for the IBM PC, they committed one of the biggest blunders in computing history. They published the entire plans and circuit diagrams and sold it for \$49. Since the design was now completely public, other small companies started to build clone machines and thus the IBM PC clone market was born.

Other companies also started to sell machines; some of the interesting ones have been by Apple, Amiga, Commodore and Atari. The Apple Lisa was the predecessor to the Macintosh, and also used a 68000. It also included a fully multitasking, GUI based operating system, highly integrated software, high capacity (but incompatible) 'twiggy' 5 1/4" disk drives, and a large workstation-like monitor. It was better than the Macintosh in almost every way, but was correspondingly more expensive. (Hennessy, Motorola)

The Macintosh was to include the best features of the Lisa, but at an affordable price - in fact the original Macintosh came with only 128K of RAM and no expansion slots. Cost was such a factor that the 8 bit Motorola 6809 was the original design choice, and some prototypes were built, but they quickly realized that it didn't have the power for a GUI based OS, and they used the Lisa's 68000, borrowing some of the Lisa low level functions (such as graphics toolkit routines) for the Macintosh. (Apple)

Competing personal computers such as the Amiga and Atari ST, and early workstations by Sun, Apollo, NeXT and most others also used 680x0 CPUs (including one of the earliest workstations, the Tandy TRS-80 Model 16, which used a 68000 CPU

and Z-80 for I/O and VM support (the 68000 could not restart an instruction stopped by a memory exception, so it was suspended while the Z-80 loaded the page)).

However, the IBM PC was so popular, that others found it difficult to compete. The IBM PC's came with MS DOS installed, supplied by a then small company called Microsoft. Since then Intel has taken off, producing increasingly powerful chips and Microsoft has become one of the biggest software companies in history, together forming an alliance that has revolutionized home computing, as we know it.

From the above outline of computer history, one can fairly say that the definition of a computer has been evolving. What was earlier an entire machine can now be fit into a small part of a microprocessor. Computer systems have undergone many changes; machines that once filled large areas have been reduced to small desktop computer systems because of the microprocessor. Although these desktop machines are compact, they possess computing power that was only dreamed of a few years ago. Million dollar mainframe computer systems developed in the early 1980's are not as powerful as the 486, Pentiums or Pentium III based computers of today. In 1965, Gordon Moore had postulated that the power of these microprocessors would grow exponentially, doubling every two years. This law, better known as Moore's law has, has more or less been accurate till this date in predicting microprocessor complexity and power.

## **Microprocessors**

Microprocessor development has always been spurred by the need for faster computers: The faster a processor runs, the better it would sell in the market place. The task of the microprocessor designer is not easy as they are often constrained by marketing concerns such as price and the necessity for backward software compatibility with a

previous CPU design and current technology. Therefore current-day chip designers are hard pressed to use more and more exotic architectural tricks to get the most speed possible. Sometimes, given the expected target applications, the designer is motivated to make design trade-off: favoring an improvement in a certain area of application while sacrificing others. All this makes the field of processor architecture a very fascinating one to study.

As a start, let's introduce a term often used when talking about CPUs: The *price/performance ratio*. Obviously, it is meant to compare different CPUs. One can immediately see that a CPU with a lower price and/or a higher performance would be a better one. This ratio is not always quoted with numbers; possibly because: (a) the price is never a constant for a long time. For example, Intel prices goes down every 6 months almost like a clockwork, (b) the performance is measured by different incompatible benchmark programs, and (c) price is often more a factor of volume than technology.

Before we get into the intricacies of microprocessor design, it would be prudent for us to review some of the basic concepts involved and themes common to most processor design.

## **Instruction Set Architecture**

The term “computer architecture” was first used by IBM to describe a programming interface to their System 360 series of computers, specifically the instructions, the registers, and flags seen by the assembly programmer. This interface has become known as the instruction set architecture or ISA. ISA defines a set of instructions that can be implemented efficiently in current and future technologies enabling programs

to be executed on hardware using specific sets of instructions. A specific ISA is usually created to run a particular set of applications, or work load, at some performance level cost-effectively. However, sometimes the ability to run particular applications often constrains the ISA through the need to use legacy code, the Intel ISA being the most obvious example, and the availability of applications is heavily influenced by the marketplace. However, one overriding factor is clear: microprocessor architecture is to a significant degree an engineering act of providing a solution to a computing problem within a cost constraint. (Flynn)

### **Basic Processor Design: Von Neumann Machine.**

The operations of most modern computers are modeled based on the Von Neumann machine. The microprocessor executes each instruction in a series of small steps.

1. Fetch the instruction from memory into the instruction register
2. Change the program counter to point to the following instruction.
3. Determine the type of instruction fetched.
4. If the instruction uses a word in memory, determine where it is.
5. Fetch the word, into a register if needed.
6. Execute the instruction.
7. Go to step 1.

This sequence is referred to as the fetch-decode-execute cycle, central to the operation of most modern processors. (Hennessy & Patterson)

## **Microprocessor Design**

Producing a faster processor is usually pursued through the following strategies:

### **Direct Execution by Hardware**

All common instructions are directly executed in hardware, not interpreted by microinstructions thus making the common instructions execute at high speed.

### **Easy Instruction Decoding**

A critical limit on the rate of issue of instructions is decoding individual instructions to determine the required resources. In order to aid this process, instructions should be made regular and have fixed length with a small number of fields.

### **Higher clock rate**

All else being equal, the higher the clock rate, a processor can process the more instructions in a given time. So it is in the last decade we have seen new processors coming out with higher and higher clock rate. While 25Mhz was said to have 'rocket like speed' in 1988, nowadays 1GHz or more can be expected from top-of-the-line models, up to 1.7 GHz for Pentium 4 from Intel.

Actually, achieving a high clock rate has little to do with advancement in processor architecture. It is primarily determined by the currently available fabrication technology, which is usually specified in terms of how far apart are the traces in an integrated circuit. Currently, most companies are moving their production facilities from 0.18 micron to the 0.13-micron production line. To understand the relation between achievable clock rate and trace width, remember that the speed of electricity in a copper wire is limited. Assuming it is about 20cm per nanosecond, then a CPU with 1GHz clock will have 1ns time between cycles, which is just enough time to travel 20cm (we are

ignoring lost time due to the time needed to switch gates polarity etc.). It should now be clear that CPUs on which the transistor elements are more tightly packed can accomplish the same amount of work in less time than otherwise. In short, the smaller the traces are, the higher the clock rate can be.

Another way to decrease the distance between components and therefore increase the speed of communication between them is to provide multiple levels of metal for interconnection. Current processors use between 4 and 6 *layers* of metal interconnections. Its down side is it is more expensive to fabricate.

Additional benefit that comes from a smaller trace and more layers is a higher *chip density*: more transistors can be fitted inside the same chip area. Chips designers gladly use this available area to add more functionality into the chip, such as bigger caches, more registers, or more complex logic supports.

On the other hand, the combination of smaller traces, more transistors, and high clock rate give rise to a more severe heat problem. To combat this (and prevent a CPU meltdown), most CPU nowadays requires huge heat sinks and a dedicated 'CPU fan'. More effectively, to produce less heat, current processors are designed to work at a lower voltage than their predecessor, 2.0 or even 1.6 volt compared to the 5.0 or 3.3 volt of older processors. But going down in voltage also produces a new problem: it is more difficult to detect the difference between logical 0 and logical 1. Which means the electrical tolerances for system components are more sensitive (equates to costlier system). As a way out of this, some processors work with 'dual-voltage', a lower internal one for the CPU core and a higher external one for I/O.

### **Higher instructions per cycle rate**

This is where most of the new ideas in CPU architecture come into being. The basic idea is to get more program instructions done in a fixed number of clock cycles. Alternately, the same goal can be expressed as “lowering the average CPI” (Cycles Per Instruction).

### **Maximize Instruction Issue rate**

Parallelism plays a critical role in improving performance. Processors should be able to process multiple instructions simultaneously, thus large number of slow instructions can be issued in a short time, improving the processors MIPS rating.

### **Superscalar**

*Superscalar processor* architecture allows multiple instructions to be issued simultaneously during each cycle. This is in contrast to the previous generation (so-called *base scalar*) processor that is only able to process one instruction per cycle. In a superscalar processor, a fetch/decode-instruction unit would issue multiple instructions on each cycle to separate pipelines.

The number of instructions which can be executed in parallel for a certain processor is called its *superscalar degree* [Hwang93], although most literature used the term *n*-issue, as in 2-issue, 3-issue, or 4-issue, etc. (other literature uses the term *n*-way instead of *n*-issue). Current processors have a degree between 2 and 5.

In theory, a 4-issue superscalar processor would be 4 times as fast as the scalar implementation. But this is assuming that all the instruction pipelines are busy all the

time, and this is certainly not the case. The difficulty lies in the fact that all the instructions must be independent of each other, and whether there are free *functional unit* to do the task.

Processor	Degree	Units	Note
Pentium Pro	3-issue	?	X86 instruction
PowerPC 620	4-issue	6	3 Integer ALUs, 1 FPU, 1 Load/Store, 1 Branch
PA-8000	4-issue	10	2 Integer ALUs, 2 Integer Shift/Merge, 2 FP Multiply/Accumulate (MAC), 2 FP Divide/Square-root, 2 Load/Store
UltraSPARC-I	2-issue	9	Pre-fetch and Dispatch, Integer Execution, FP, Load/Store
Alpha 21164	4-issue	4	2 Integer, 2 FP
R10000	4-issue	5	Although the CPU has the potential to issue 5 execution instructions per cycle, it can fetch and retire only 4, a fifth cannot complete on the same cycle. Nevertheless, the excess of dispatch bandwidth offers more flexible opportunities for instruction scheduling, which means that the R10000 can complete four instructions per cycle for more of the time.

## Pipelining and Superpipeline

The execution of a program instruction inside the CPU usually happens in stages: first the instruction is fetched (either from memory or from the cache), then it is decoded (analyzed), then if the instruction needs any data as operand, the memory address of the data must be resolved, before the data can be fetched, and then the instruction can be executed, and finally the result might need to be written somewhere. By implementing these stages in separate processing units and make it so that a unit for a stage immediately begin with the next program instruction after finished processing the current one, we have what is called *pipelining*. Much like a car being built on an assembly line.

The number of pipeline stages employed in current processors varies between 4 and 9. More stages meant for more opportunities to optimize the pipeline utilization, but also incur a more severe penalty in case of a (mispredicted) branch and the pipeline has to be *flushed*.

In a *superpipelined* processor, the pipeline begins with the next instruction at a faster rate than the base cycle (which is equal to the time needed by a stage to complete its processing).

Superscalar designs rely primarily on spatial parallelism - multiple operations running concurrently on separate hardware - achieved by duplicating hardware resources, such as execution units and register file ports. Superpipelined designs, on the other hand, emphasize temporal parallelism - overlapping multiple operations on a common piece of hardware - achieved through more deeply pipelined execution units with faster clock cycles.

To realize the speedup potential of a pipelined design, it is imperative that the pipeline is kept busy all the time: an idle pipeline will not contribute to the work being done. The difficulty here is in feeding the pipeline(s) with the appropriate instructions so a *stall* does not happen.

An instruction that stalls in a pipeline meant that further processing of that instruction is held back, and it can only be continued if the cause has disappeared; usually when an earlier instruction completes an action that will eliminate the stall condition.

Hazards that cause pipeline stalls come in three classes:

1. **Structural hazards** occur when the processor does not have enough resources to handle a particular combination of instructions.
2. **Control hazards** can occur when a branch instruction executes. If the branch jumps to an address that cannot be found in the on-chip cache, the execution pipelines will stall until the jumped-to instruction can be fetched from memory. This is a particularly severe problem because of the high ratio of

jumps in an average program. To combat this many techniques have been developed (and implemented), partly in software - by creating smart compilers that exploit parallelism opportunities - and partly in hardware - by techniques such as *out-of-order* execution scheme, *branch prediction* units, and *speculative execution*.

3. **Data hazards** that occurs when one instruction is dependent on a preceding instruction. This can be differentiated into three types:
  - a. RAW (read-after-write) dependency: an instruction tries to read the value from a register when the previous instruction is not yet finished writing a value to it.
  - b. WAR (write-after-read) dependency: an instruction tries to write a value to a register when the previous instruction is not yet finished reading from it.
  - c. WAW (write-after-write) dependency: an instruction tries to write a value to a register when the previous instruction is not yet finished writing to it.

In the efforts to keep pipelines at full utilization (and therefore realize the maximum processing speedup), chip designers put more and more "smarts" into the chip itself. This is somewhat against the earlier trend seen in RISC chips that placed the burden on to the compiler. On one hand, this is a good thing: Older software can run on a new hardware and enjoy a better performance without the need for recompiling on a new optimizing compiler. Also, optimization tricks only need to be designed once instead of separately for each language compiler. On the other hand, it is a bad thing: These kinds of smarts require a whole lot chip area for the supporting logic. To cite examples:

Instruction Reorder Buffer, Branch Processing Unit, Branch Prediction Table, Branch Misprediction Recovery, Rename Buffers, Instruction Completion Units, and so on.

Processor	Pipeline stages int/ldst/fp
PA-8000	7/9/9
Pentium Pro	14/14/16
PowerPC 604e	4/5/6
R10000	5/6/7
21164A	7/7/9
UltraSPARC-I	9/9/9

### Reduced Instruction Set (RISC) architecture

The root idea of RISC (Reduced Instruction Set Computer, intended as a contrast to CISC, which is Complex Instruction Set Computer) is quite simple: Instead of providing a rich instruction set to make compiler writers happy, take only the most used processor instructions and discard the rest. Allow only a few addressing schemes and make sure that all the instructions chosen to be implemented to run **fast**.

That idea is workable because experience has shown that in an average program, a very few of the instructions do most of the work. Concentrating on few instructions and providing as much hardwired logic as possible, almost all instructions can be executed in just one clock cycle.

The exception to the above is the load and store instructions, which have to access the relatively slow memory. Therefore in a RISC machine load and store operations is restricted to a few instructions, while other instructions work exclusively with registers.

To deal with the fact that not all instructions can be expected to finish in one clock cycle, the idea of *pipelining*, as elaborated further in a section below, is introduced. Pipelining proved to be so successful that nowadays all processors use this strategy.

One thing to note about RISC architecture with pipelines, is that a much heavier burden is placed on the compiler to produce an optimized binary code suitable for pipelining.

For a time, there were many heated debates between the RISC and the CISC camp, but nowadays the distinction is quite blurred because both camps use proven good ideas from the other. As an example, Intel processors, which are a classic example of CISC architecture, now use a RISC core in their Pentium Pro processors. All the incoming x86 instructions are first translated to a RISC-like microcode (which is easier to be optimized for the superscalar superpipelined core) before being further processed.

One of the more interesting RISC processor implementations has been the MIPS series of processors, used in Silicon Graphics machines. The R2000 design came from the Stanford MIPS project, which stood for Microprocessor without Interlocked Pipeline Stages, and was arguably the first commercial RISC processor. It was intended to simplify processor design by eliminating hardware interlocks between the five pipeline stages. This meant that only single execution cycle instructions could access the thirty-two 32 bit general registers, so that the compiler can schedule them to avoid conflicts. This also meant that LOAD/STORE and branch instructions have a 1-cycle delay to account for. However, because of the importance of multiply and divide instructions, a special HI/LO pair of multiply/divide registers exist which do have hardware interlocks, since these take several cycles to execute and produce scheduling difficulties.

The R2000 had no condition code register considering it a potential bottleneck. The PC (program counter) was user readable. The CPU included an MMU unit that could also control a cache, and the CPU was one of the first that could operate as a big or little

endian processor. An FPU, the R2010, is also specified for the processor.

Newer versions included the R3000 (1988), with improved cache control, and the R4000 (1991) (expanded to 64 bits and is superpipelined (twice as many pipeline stages do less work at each stage, allowing a higher clock rate and twice as many instructions in the pipeline at once, at the expense of increased latency when the pipeline can't be filled, such as during a branch, (and requiring interlocks added between stages for compatibility, making the original "I" in the "MIPS" acronym meaningless))). The R4400 and above integrated the FPU with on-chip caches. The R4600 and later versions abandoned superpipelines.

The superscalar R8000 (1994) was optimized for floating point operation, issuing two integer or load/store operations (from four integer and two load/store units) and two floating-point operations simultaneously (FP instructions sent to the independent R8010 floating point coprocessor (with its own set of thirty-two 64-bit registers and load/store queues)).

The R10000 and R12000 versions (early 1996 and May 1997) added multiple FPU units, as well as almost every advanced modern CPU feature, including separate 2-way I/D caches (32K each) plus on-chip secondary controller (and high speed 8-way split transaction bus (up to 8 transactions can be issued before the first completes)), superscalar execution (load four, dispatch five instructions (may be out of order) to any of two integer, two floating point, and one load/store units), dynamic register renaming (integer and floating point rename registers (thirty two in the R10K, forty eight in the R12K)), and an instruction cache where instructions are partially decoded when loaded into the cache, simplifying the processor decode (and register rename/issue) stage.

Branch prediction and target caches are also included.

The 2-way (int/float) superscalar R5000 (January, 1996) was added to fill the gap between R4600 and R10000, without any fancy features (out of order or branch prediction buffers). For embedded applications, MIPS and LSI Logic added a compact 16 bit instruction set which can be mixed with the 32 bit set (same as the ARM Thumb 16 bit extension), implemented in a CPU called TinyRISC (October 1996), as well as MIPS V and MDMX (MIPS Digital Multimedia Extensions, announced October 1996). MIPS V adds parallel floating point (two 32 bit fields in 64 bit registers) operations (compared to similar HP MAX integer or Sun VIS and Intel MMX floating point unit extensions), MDMX adds integer 8 or 16 bit subwords in 64 bit FPU registers and a 24 and 48 bit subwords in a 192 bit accumulator for multimedia instructions (a MAC instruction on an 8-bit value can produce a 24-bit result, hence the large accumulator). Vector-scalar operations (ex: multiply all subwords in a register by subword 3 from another register) are also supported. These extensive instructions are partly derived from Cray vector instructions (Cray is owned by SGI, the parent company of MIPS), and are much more extensive than the earlier multimedia extensions of other CPUs. Future versions are expected to add Java virtual machine support.

SGI's financial problems meant that the R10000 and derivatives (R12K and R14K) were the end of the high performance line for the MIPS architecture. SGI scaled back high-end development in favor of the promised IA-64 architecture announced by HP and Intel. MIPS was sold off by SGI, and the MIPS processor was retargeted to embedded designs where it's more successful. The R20K implemented the MDMX extensions, and increased the number of integer unit to six. SiByte introduced a less

parallel, high clock rate 64-bit MIPS CPU (SB-1, mid 2000) exceeding what marketing people enthusiastically call the "1GHz barrier", which has never been an actual barrier of any sort.

Nintendo used a version of the MIPS CPU in the N64 (along with SGI-designed 3-D hardware), accounting for around 3/4 of MIPS embedded business in 1999 until switching to a custom IBM PowerPC, and a graphics processor from ArtX (founded by ex-SGI engineers) for its successor named Game Cube. (SGI / MIPS)

### **Complex Instruction Set Computers (CISC)**

As high-level languages started to become popular, so computer architects attempted to make the machine's capabilities match constructs used by programmers. As an example, complex addressing modes were added so that an expression like:

$$x = a[j++];$$

Could be transformed to a single machine instruction.

At that time, memory was relatively expensive and slow, so that compacting a program into as few instructions as possible was seen as a desirable goal. Smaller programs meant that they loaded faster and that more (scarce and expensive) memory was available for data.

In 1968, Robert Noyce, inventor of the silicon integrated circuit, Gordon Moore, of Moore's law fame, and Arthur Rock, a San Francisco venture capitalist, formed the Intel Corporation to make memory chips. In its first year of operation, Intel sold only \$3000 worth of chips, but business has picked up since then.

In the late 1960s, calculators were large electromechanical machines the size of a modern laser printer and weighing 20 kg. In Sept. 1969, a Japanese company, Busicom, approached Intel with a request for It to manufacture 12 custom chips for a proposed electronic calculator. The Intel engineer assigned to this project, Ted Hoff, looked at the plan and realized that he could put a 4-bit general-purpose CPU on a single chip that would do the same thing and be simpler and cheaper as well. Thus in 1970, the first single-chip CPU, the 2300-transistor 4004 was born (Faggin et al., 1996).

It is worth noting that neither Intel nor Busicom had any idea what they had just done. When Intel decided that it might be worth a try to use the 4004 in other projects, it offered to buy back all the rights to the new chip from Busicom by returning the \$60,000 Busicom had paid Intel to develop it. Intel's offer was quickly accepted, at which point it began working on an 8-bit version of the chip, the 8008, introduced in 1972.

Intel did not expect much demand for the 8008, SO it set up a low-volume production line. Much to everyone's amazement, there was an enormous amount of interest, so Intel set about designing a new CPU chip that got around the 8008's 16K memory limit (imposed by the number of pins on the chip). This design resulted in the 8080, a small, general-purpose CPU, introduced in 1974. Much like the PDP-8, this product took the industry by storm and instantly became a mass-market item. Only instead of selling thousands, as DEC had, Intel sold millions.

In 1978 came the 8086, a true 16-bit CPU on a single chip. The 8086 was designed to be somewhat similar to the 8080, but it was not completely compatible with the 8080. The 8086 was followed by the 8088, which had the same architecture as the 8086, and ran the same programs but had an 8-bit bus instead of a 16-bit bus, making it

both slower and cheaper than the 8086. When IBM chose the 8088 as the CPU for the original IBM PC, this chip quickly became the personal computer industry standard. Neither the 8088 nor the 8086 could address more than 1 megabyte of memory. By the early 1980s this became more and more of a serious problem, so Intel designed the 80286, an upward compatible version of the 8086. The basic instruction set was essentially the same as that of the 8086 and 8088, but the memory organization was quite different, and rather awkward, due to the requirement of compatibility with the older chips. The 80286 was used in the IBM PC/AT and in the midrange PS/2 models. Like the 8088, it was a huge success, mostly because people viewed it as a faster 8088.

The next logical step was a true 32-bit CPU on a chip, the 80386, brought out in 1985. Like the 80286, this one was more-or-less compatible with everything back to the 8080. Being backward compatible was a boon to people for whom running old software was important, but a nuisance to people who would have preferred a simple, clean, modern architecture unencumbered by the mistakes and technology of the past.

Four years later the 80486 came out. It was essentially a faster version of the 80386 that also had a floating-point unit and 8K of cache memory on chip. **Cache** memory is used to hold the most commonly used memory words inside or close to the CPU, to avoid (slow) accesses to main memory. The 80386 also had built-in multiprocessor support, to allow manufacturers to build systems containing multiple CPUs.

At this point, Intel found out the hard way (by losing a trademark infringement lawsuit) that numbers (like 80486) cannot be trademarked, so the next generation got a name: Pentium. Unlike the 80486, which had one internal pipeline, the Pentium had two

of them, which helped make it twice as fast. When the next generation appeared, people who were hoping for the Sexium were disappointed. The name Pentium was now so well known that the marketing people wanted to keep it, and the new chip was called the Pentium Pro. Despite the small name change from its predecessor, this processor represented a major break with the past. Instead of having two or more pipelines, the Pentium Pro had a very different internal organization and could execute up to five instructions at a time.

Another innovation found in the Pentium Pro was a two-level cache memory. The processor chip itself had 8 KB of memory to hold commonly used instructions and 8 KB of memory to hold commonly-used data. In the same cavity within the Pentium Pro package (but not on the chip itself) was a second cache memory of 256 KB.

The next new Intel processor was the Pentium II, essentially a Pentium Pro with special multimedia extensions (called **MMX**) added. These instructions were intended to speed up computations required to process audio and video, making the addition of special multimedia coprocessors unnecessary. These instructions were also available in later Pentiums, but not in the Pentium Pro, so the Pentium II combined the strengths of the Pentium Pro with multimedia.

## **VLIW architecture**

The VLIW (Very Long Instruction Word) architecture is generalized from two well-established concepts: horizontal microcoding and superscalar processing. A typical VLIW machine has instruction words hundreds of bits in length that is processed in parallel by multiple functional units. Thus, a VLIW instruction consists of different fields that carry the opcodes to be dispatched to the functional units. As a consequence,

programs written in conventional short instruction words (say 32 bits) must be compacted together to form the VLIW instruction. This compaction needs to be done by a smart compiler which knows the extent of available functional units, can detect flow and data dependency, and can predict branch outcomes (using elaborate heuristics or run-time statistics).

As such, the success of a VLIW processor depends heavily on the efficiency in code compaction. Furthermore, VLIW architecture is totally incompatible with that of any conventional general-purpose processor, and each implementation require different compaction strategy, thus precluding binary compatibility even within processors of the same family.

The result is, while VLIW is conceptually interesting, it has not gained acceptance in the marketplace. That is why our examples do not include processors using this type of architecture.

## **Multi processor support**

While the power of current processors is not something to be sneered at, there are jobs heavy enough that a single processor system is overburdened by them. One way to handle such jobs in a reasonable amount of time is a multi processor system that can divide the task between the available processors.

These systems require special support from the hardware and OS, so that the processors can share resources without conflict. For the processors, some of the issues that must be resolved are: interaction during system reset, system initialization, and OS boot. Some manner of arbitration would also be needed for interrupt processing and shared access to the bus and memory and last, but not least, a protocol to ensure cache

coherency. Thus, most processors today have a built-in support for a multi processor implementation.

Another related issue which needs to be addressed by the designers as systems goes multi-processor, is providing a bus with enough bandwidth to cope with the much higher data traffic.

Processor	Number
Pentium II	4
UltraSPARC	4

### **Future Trends in Microprocessor Design**

Trends in computer architecture fall into two broad classes; those that result from the progress in the implementation technologies, and those that result from popular applications.

#### **Technology driven trends**

The technology driven trends are primarily driven by the remarkable growth in the semiconductor industry. The constant reductions in the size of electronic and magnetic devices that can be manufactured inexpensively drive these trends. In the not so distant future, we expect to see processor traces separated by 0.05 microns and chips that contain over a 100 million transistors operating at several GHz. But beyond that, semiconductor industry will reach physical limits, at which point we may start to see computing based on different primitives.

Another obstacle is memory latency. Intra-chip latencies are decreasing and bandwidth is increasing, both at a rate much greater than the corresponding inter-chip figures. While the CPU speed curve has increased at about 50% per year, RAM speed has increased only 10% per year. Further, disk access time improvement is at an even lower

rate. Bridging these performance gaps is imperative for a computer system to have significant improvements.

### **Application driven trends**

New applications that make use of voice recognition, and image recognition use a lot of computational power. These kinds of applications have not been the research focus in the past in academia, where the focus is more on scientific and heavy engineering workloads. Indeed, ENIAC, one of the earliest computers was built to calculate missile trajectories, but now the focus of computing has shifted to nonscientific areas such as multimedia and graphics. The thinking in academia is still influenced by this early history and has led to the development of research areas that may not be the most relevant today.

Future research in microprocessor architecture should span a wide variety of subjects, from incremental improvement in design and fabrication to revolutionary new designs and architectures.

Parallelism, an area of research that has been already explored widely, has had mixed results. Although these machines are not difficult to build, the difficulty in programming has presented a formidable barrier to the use of massively parallel processors.

## References

- [Christy97] Peter Christy: "IA-64 and Merced - What and Why", *Microprocessor Report v10 #17, Dec 1996*
- [Gwennap96a] Linley Gwennap: "Digital 21264 Sets New Standard", *Microprocessor Report v10 #14, Oct 1996*
- [Gwennap96b] Linley Gwennap: "Editorial: Clouds Darken Over RISC", *Microprocessor Report v10 #15, Nov 1996*
- [Gwennap97a] Linley Gwennap: "Alpha Sails, PowerPC Flails", *Microprocessor Report v11 #1, Jan 1997*
- [Gwennap97b] Linley Gwennap: "Pentium II Debuts at 300Mhz", *Microprocessor Report v11 #6, May 1997*
- [Halfhill95] Tom R. Halfhill: "Intel's P6", *Byte magazine, Apr 1995*
- [Halfhill96] Tom R. Halfhill: "x86 Enters the Multimedia Era", *Byte magazine, Jul 1996*
- [HP94] "PA-RISC 1.1 Architecture and Instruction Set Reference Manual, Third Edition", *Hewlett-Packard Company, Feb 1994*
- [Hwang93] Kai Hwang: *Advanced Computer Architecture*, McGraw-Hill *Computer Science Series*, 1993
- [Hyde96] John Hyde: "How to Make Pentium Pros Cooperate", *Byte magazine, Apr 1996*
- [IBM/Motorola96] "PowerPC 604 RISC Microprocessor Technical Summary", *IBM / Motorola, 1996*

- [Lesartre96] Gregg Lesartre and Doug Hunt: "PA-8500: The Continuing Evolution of the PA-8000 Family", *Hewlett-Packard Company*, 1997
- [MIPS96] "R10000 Microprocessor User's Manual Version 2.0", *MIPS Technologies, Inc.*, Dec 1996
- [Pountain94] Dick Pountain: "A Different Kind of RISC", *Byte magazine*, Aug 1994
- [Pountain95] Dick Pountain: "HP's Speedy RISC", *Byte magazine*, Jul 1995
- [Pountain96] Dick Pountain and Tom R. Halfhill: "CPU Scorecards", *Byte magazine*, Nov 1995
- [Ryan94a] Bob Ryan: "M1 Challenges Pentium", *Byte magazine*, Jan 1994
- [Ryan94b] Bob Ryan: "Pipeline Hazards", *Byte magazine*, Jan 1994
- [SUN95] "The UltraSPARC processor - Technology White Paper", *Sun Microelectronics*, Nov 1995.
- [Tanenbaum90] A. S. Tanenbaum: *Structured Computer Organization*, 3rd ed., Prentice Hall International, 1990
- [Thompson95] Tom Thompson: "Two Turbocharged PowerPCs", *Byte magazine*, Dec 1995
- Hennessy & Patterson: *Computer Architecture A Quantitative Approach*, 2<sup>nd</sup> ed., 1996
- Goldstine: *The Computer: From Pascal to Von Neumann*, 2<sup>nd</sup> printing, 1973
- Ceruzzi: *A History of Modern Computing*, Second Printing, 1999
- Brey: *The Intel x86 Architecture*, 1998