Processor Types

ARM 1 (v1)

This was the very first ARM processor. Actually, when it was first manufactured in April 1985, it was the very first commercial RISC processor. Ever.
As a testament to the design team, it was "working silicon" in it's first incarnation, it exceeded it's design goals, and it used less than 25,000 transistors.

The ARM 1 was used in a few evaluation systems on the BBC micro (Brazil - BBC interfaced ARM), and a PC machine (Springboard - PC interfaced ARM).
It is believed a large proportion of Arthur was developed on the Brazil hardware.
In essence, it is very similar to an ARM 2 - the differences being that R8 and R9 are not banked in IRQ mode, there's no multiply instruction, no LDR/STR with register-specified shifts, and no co-processor gubbins.

ARM evaluation system for BBC Master
(original picture source not known - downloaded from a website full of BBC-related images this version created by Rick Murray to include zoomed-up ARM down the bottom...)

ARM 2 (v2)

Experience with the ARM 1 suggested improvements that could be made. Such additions as the MUL and MLA instructions allowed for real-time digital signal processing. Back then, it was to aid in generating sounds. Who could have predicted exactly how suitable to DSP the ARM would be, some fifteen years later?
In 1985, Acorn hit hard times which led to it being taken over by Olivetti. It took two years from the arrival of the ARM to the launch of a computer based upon it...

...those were the days my friend, we thought they'd never end.
When the first ARM-based machines rolled out, Acorn could gladly announce to the world that they offered the fastest RISC processor around. Indeed, the ARM processor kicked ass across the computing league tables, and for a long time was right up there in the 'fastest processors' listings. But Acorn faced numerous challenges. The computer market was in disarray, with some people backing IBM's PC, some the Amiga, and all sorts of little itty-bitty things. Then Acorn go and launch a machine offering Arthur (which was about as nice as the first release of Windows) which had no user base, precious little software, and not much third party support. But they succeeded.

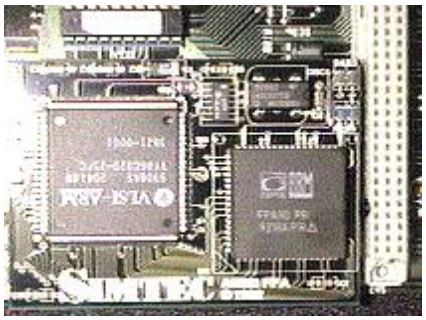
The ARM 2 processor was the first to be used within the RISC OS platform, in the A305, A310, and A4x0 range. It is an 8MHz processor that was used on all of the early machines, including the A3000. The ARM 2 is clocked at 8MHz, which translates to approximately four and a half million instructions per second (0.56 MIPS/MHz).

No current image - can you help?

ARM 3 (v2as)

Launched in 1989, this processor built on the ARM 2 by offering 4K of cache memory and the SWP instruction. The desktop computers based upon it were launched in 1990.
Internally, via the dedicated co-processor interface, CP15 was 'created' to provide processor control and identification.
Several speeds of ARM 3 were produced. The A540 runs a 26MHz version, and the A4 laptop runs a 24MHz version. By far the most common is the 25MHz version used in the A5000, though those with the 'alpha variant' have a 33MHz version.
At 25MHz, with 12MHz memory (a la A5000), you can expect around 14 MIPS (0.56 MIPS/MHz).
It is interesting to notice that the ARM3 doesn't 'perform' faster - both the ARM2 and the ARM3 average 0.56 MIPS/MHz. The speed boost comes from the higher clock speed, and the cache.

Oh, and just to correct a common misunderstanding, the A4 is not a squashed down version of the A5000. The A4 actually came first, and some of the design choices were reflected in the later A5000 design.



ARM3 with FPU
(original picture downloaded from Arcade BBS, archive had no attribution)

ARM 250 (v2as)

The 'Electron' of ARM processors, this is basically a second level revision of the ARM 3 design which removes the cache, and combines the primary chipset (VIDC, IOC, and MEMC) into the one piece of silicon, making the creation of a cheap'n'cheerful RISC OS computer a simple thing indeed. This was clocked at 12MHz (the same as the main memory), and offers approximately 7 MIPS (0.58 MIPS/MHz).
This processor isn't as terrible as it might seem. That the A30x0 range was built with the ARM250 was probably more a cost-cutting exercise than intention. The ARM250 was designed for low power consumption and low cost, both important factors in devices such as portables, PDAs, and organisers - several of which were developed and, sadly, none of which actually made it to a release.

No current image - can you help?

ARM 250 mezzanine

This is not actually a processor. It is included here for historical interest. It seems the machines that would use the ARM250 were ready before the processor, so early releases of the machine contained a 'mezzanine' board which held the ARM 2, IOC, MEMC, and VIDC.

ARM 4 and ARM 5

These processors do not exist.

More and more people began to be interested in the RISC concept, as at the same sort of time common Intel (and clone) processors showed a definite trend towards higher power consumption and greater need for heat dissipation, neither of which are friendly to devices that are supposed to be running off batteries.
The ARM design was seen by several important players as being the epitome of sleek, powerful RISC design.
It was at this time a deal was struck between Acorn, VLSI (long-time manufacturers of the ARM chipset), and Apple. This lead to the death of the Acorn RISC Microprocessor, as Advanced RISC Machines Ltd was born. This new company was committed to design and support specifically for the processor, without the hassle and baggage of RISC OS (the main operating system for the processor and the desktop machines). Both of those would be left to Acorn.

In the change from being a part of Acorn to being ARM Ltd in it's own right, the whole numbering scheme for the processors was altered.

ARM 610 (v3)

This processor brought with it two important 'firsts'. The first 'first' was full 32 bit addressing, and the second 'first' was the opening for a new generation of ARM based hardware.
Acorn responded by making the RiscPC. In the past, critics were none-too-keen on the idea of slot-in cards for things like processors and memory (as used in the A540), and by this time many people were getting extremely annoyed with the inherent memory limitations in the older hardware, the MEMC can only address 4Mb of memory, and you can add more by daisy-chaining MEMCs - an idea that not only sounds hairy, it *is* hairy!

The RiscPC brought back the slot-in processor with a vengeance. Future 'better' processors were promised, and a second slot was provided for alien processors such as the 80486 to be plugged in. As for memory, two SIMM slots were provided, and the memory was expandable to 256Mb. This does not sound much as modern PCs come with half that as standard. However you can get a lot of milage from a RiscPC fitted with a puny 16Mb of RAM.

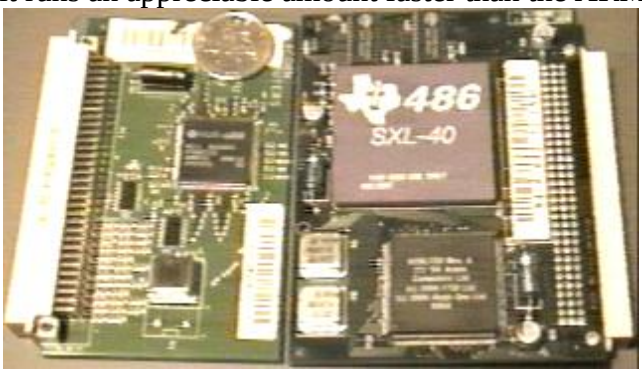
But, always, we come back to the 32 bit. Because it has been with us and known about ever since the first RiscPC rolled out, but few people noticed, or cared. Now as the new generation of ARM processors drop the 26 bit 'emulation' modes, we RISC OS users are faced with the option of getting ourselves sorted, or dying.
Ironically, the other mainstream operating systems for the RiscPC hardware - namely ARMLinux and netbsd/arm32 are already fully 32 bit.

Several speeds were produced; 20MHz, 30Mhz, and the 33MHz part used in the RiscPC.
The ARM610 processor features an on-board MMU to handle memory, a 4K cache, and it can even switch itseld from little-endian operation to big-endian operation. The 33MHz version offers around 28MIPS (0.84 MIPS/MHz).

The RiscPC ARM610 processor card
(original picture by Rick Murray, © 2002)

ARM 710 (v3)

As an enhancement of the ARM610, the ARM 710 offers an increased cache size (8K rather than 4K), clock frequency increased to 40MHz, improved write buffer and larger TLB in the MMU. Additionally, it supports CMOS/TTL inputs, Fastbus, and 3.3V power but these features are not used in the RiscPC.
Clocked at 40MHz, it offers about 36MIPS (0.9 MIPS/MHz); which when combined with the additional clock speed, it runs an appreciable amount faster than the ARM 610.



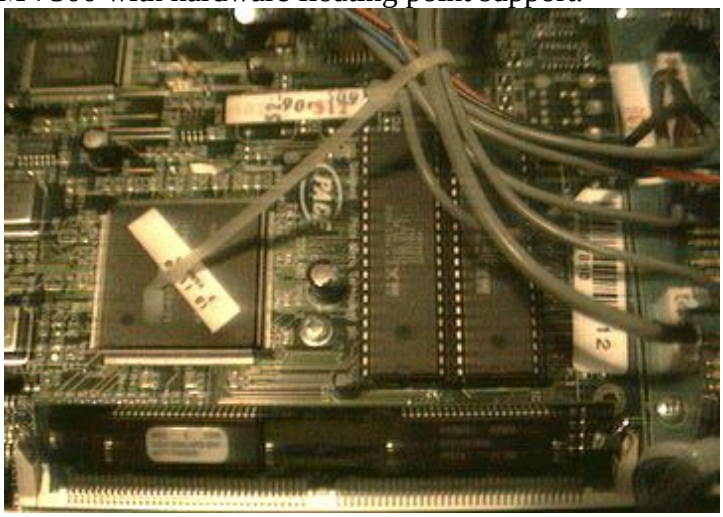
ARM710 side by side with an 80486, the coin is a British 10 pence coin.
(original picture by Rick Murray, © 2001)

ARM 7500

The ARM7500 is a RISC based single-chip computer with memory and I/O control on-chip to minimise external components. The ARM7500 can drive LCD panels/VDUs if required, and it features power management. The video controller can output up to a 120MHz pixel rate, 32bit sound, and there are four A/D convertors on-chip for connection of joysticks etc.
The processor core is basically an ARM710 with a smaller (4K) cache.
The video core is a VIDC2.
The IO core is based upon the IOMD.
The memory/clock system is very flexible, designed for maximum uses with minimum fuss.
Setting up a system based upon the ARM7500 should be fairly simple.

ARM 7500FE

A version of the ARM 7500 with hardware floating point support.



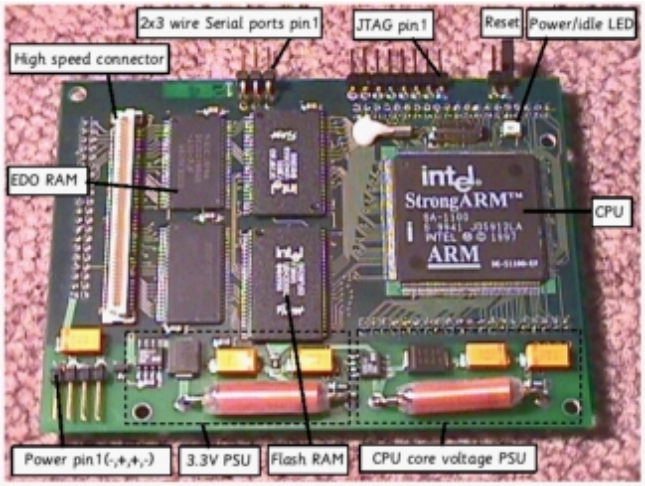
ARM7500FE, as used in the Bush Internet box.

(original picture by Rick Murray, © 2002)

StrongARM / SA110 (v4)

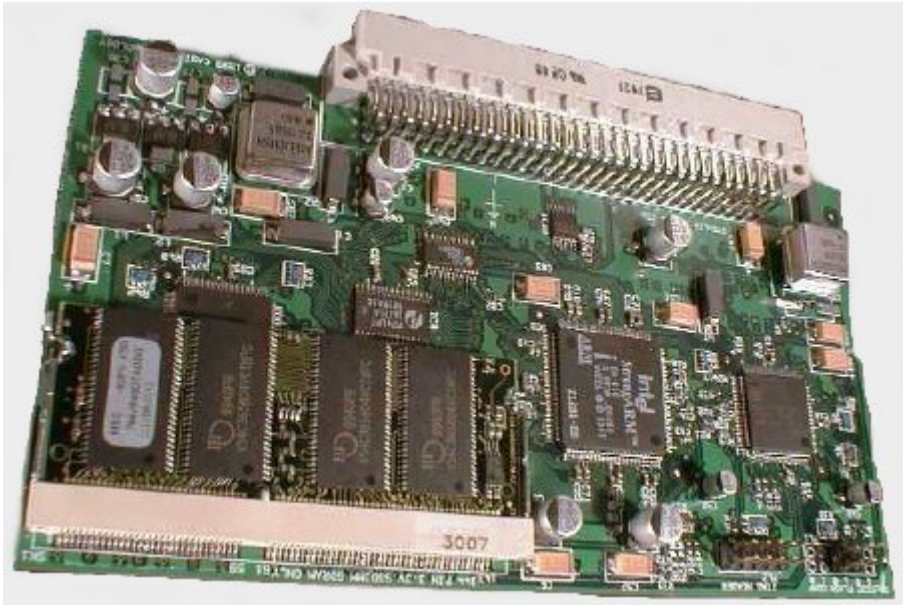
The StrongARM took the RiscPC from around 40MHz to 200-300MHz and showed a speed boost that was more than the hardware should have been able to support. Still severely bottlednecked by the memory and I/O, the StrongARM made the RiscPC fly. The processor was the first to feature different instruction and data caches, and this caused quite a lot of self-modifying code to fail including, amusingly, Acorn's own runtime compression system. But on the whole, the incompatibilities were not more painful than an OS upgrade (anybody remember the RISC OS 2 to RISC OS 3 upgrade, and all the programs that used SYS OS_UpdateMEMC, 64, 64 for a speed boost froze the machine solid!).
In instruction terms, the StrongARM can offer half-word loads and stores, and signed half-word and byte loads and stores. Also provided are instructions for multiplying two 32 bit values (signed or unsigned) and replying with a 64 bit result. This is documented in the ARM assembler user guide as only working in 32-bit mode, however experimentation will show you that they work in 26-bit mode as well. Later documentation confirms this.

The cache has been split into separate instruction and data cache (Harvard architecture), with both of these caches being 16K, and the pipeline is now five stages instead of three.
In terms of performance... at 100MHz, it offers 114MIPS which doubles to 228MIPS at 200MHz (1.14 MIPS/MHz).



A StrongARM mounted on a LART board.

In order to squeeze the maximum from a RiscPC, the Kinetic includes fast RAM on the processor card itself, as well as a version of RISC OS that installs itself on the card. Apparently it *flies* due to removing the memory bottleneck, though this does cause 'issues' with DMA expansion cards.



A Kinetic processor card.

SA1100 variant

This is a version of the SA110 designed primarily for portable applications. I mention it here as I am reliably informed that the SA1100 is the processor inside the 'faster' Panasonic satellite digibox. It contains the StrongARM core, MMU, cache, PCMCIA, general I/O controller (including two serial ports), and a colour/greyscale LCD controller. It runs at 133MHz or 200MHz and it consumes less than half a watt of power.

Thumb

The Thumb instruction set is a reworking of the ARM set, with a few things omitted. Thumb instructions are 16 bits (instead of the usual 32 bit). This allows for greater code density in places where memory is restricted. The Thumb set can only address the first eight registers, and there are no conditional execution instructions. Also, the Thumb cannot do a number of things required for low-level processor exceptions, so the Thumb instruction set will always come alongside the full ARM instruction set. Exceptions and the like can be handled in ARM code, with Thumb used for the more regular code.

Other versions

These versions are afforded less coverage due, mainly, to my not owning nor having access to any of these versions.
While my site started as a way to learn to program the ARM under RISC OS, the future is in embedded devices using these new systems, rather than the old 26 bit mode required by RISC OS...
...and so, these processors are something I would like to detail, in time.

M variants

This is an extension of the version three design (ARM 6 and ARM 7) that provides the extended 64 bit multiply instructions.
These instructions became a main part of the instruction set in the ARM version 4 (StrongARM, etc).

T variants

These processors include the Thumb instruction set (and, hence, no 26 bit mode).

E variants

These processors include a number of additional instructions which provide improved performance in typical DSP applications. The 'E' standing for "Enchanced DSP".

The future

The future is here. Newer ARM processors exist, but they are 32 bit devices.
This means, basically, that RISC OS won't run on them until all of RISC OS is modified to be 32 bit safe. As long as BASIC is patched, a reasonable software base will exist. However all C programs will need to be recompiled. All relocatable modules will need to be altered. And pretty much all assembler code will need to be repaired. In cases where source isn't available (ie, anything written by Computer Concepts), it will be a tedious slog.
It is truly one of the situations that could make or break the platform.

I feel, as long as a basic C compiler/linker is made FREELY available, then we should go for it. It need not be a 'good' compiler, as long as it will be a drop-in replacement for Norcroft CC version 4 or 5. Why this? Because RISC OS depends upon enthusiasts to create software, instead of big corporations. And without inexpensive reasonable tools, they might decide it is too much to bother with converting their software, so may decide to leave RISC OS and code for another platform.

I, personally, would happily download a freebie compiler/linker and convert much of my own code. It isn't plain sailing for us - think of all of the library code that needs to be checked. It will be difficult enough to obtain a 32 bit machine to check the code works correctly, never mind all the other pitfalls. Asking us for a grand to support the platform is only going to turn us away in droves. Heck, I'm still using ARM 2 and ARM 3 systems. Some of us smaller coders won't be able to afford such a radical upgrade. And that will be VERY BAD for the platform. Look how many people use the FREE user-created Internet suite in preference to commercial alternatives. Look at all of the support code available on Arcade BBS. Much of that will probably go, yes. But would a platform trying to re-establish itself *really* want to say goodbye to the rest?
I don't claim my code is wonderful, but if only one person besides myself makes good use of it - then it has been worth it.

Basics of computer processors

Summary: Processors are the actual working part of a computer, performing arithemetic, logic, and other operations needed to run computer programs.

A **computer** is a programmable machine. There are two basic kinds of computers: analog and digital.

Analog computers are analog devices. That is, they have continuous states rather than discrete numbered states. An analog computer can represent fractional or irrational values exactly, with no round-off. Analog computers are almost never used outside of experimental settings.

A **digital computer** is a programmable clocked sequential state machine. A digital computer uses discrete states. A binary digital computer uses two discrete states, such as positive/negative, high/low, on/off, used to represent the binary digits zero and one.

The classic crude oversimplication of a computer is that it contains three elements: processor unit, memory, and I/O (input/output). The borders between those three terms are highly ambigious, non-contiguous, and erratically shifting.

processors

The **processor** is the part of the computer that actually does the computations. This is sometimes called an **MPU** (for main processor unit) or **CPU** (for central processing unit or central processor unit).

A processor typically contains an arithmetic/logic unit (**ALU**), control unit (including processor flags, flag register, or status register), internal buses, and sometimes special function units (the most common special function unit being a floating point unit for floating point arithmetic).

Some computers have more than one processor. This is called **multi-processing**.

The major kinds of digital processors are: CISC, RISC, DSP, and hybrid.

CISC

CISC stands for Complex Instruction Set Computer. Mainframe computers and minicomputers were CISC processors, with manufacturers competing to offer the most useful instruction sets. Many of the first two generations of microprocessors were also CISC.

"In many computer applications, programs written in assembly language exhibit the shortest execution times. Assembly language programmers often know the computer architecture more intimately, and can write more efficient programs than compilers can generate from high level language (HLL) code. The disadvantage of this method of increasing program performance is the diverging cost of computer hardware and software. On one hand, it is now possible to constrcut an entire computer on a single chip of semiconductor material, its cost being very small compared to the cost of a programmer's time. On the other hand, assembly language programming is perhaps the most time-consuming method of writing software.

"One way to decrease software costs is to provide assembly language instructions that perform complex tasks similar to those existing in HLLs. These tasks, such as the *character select* instruction, can be executed in one powerful assembly language instruction. A result of this philosophy is that computer instruction sets become relatively large, with many complex, special purpose, and often slow instructions. Another way to decrease software costs is to program in a HLL, and then let a compiler translate the program into assembly language. This method does not always produce the most efficient code. It has been found that it is extremely difficult to write an efficient optimizing compiler for a computer that has a very large instruction set.

"How can the HLL program execute more quickly? One approach is to narrow the semantic distance between the HLL concepts and the underlying architectual concepts. This closing of the semantic gap supports lower software costs, since the computer more closely matches the HLL,and is therefore easier to program in an HLL.

"As instruction sets become more complex, significant increases in performance and efficiency occurred." —Charles E. Gimarc and Veljko M. Milutinovic[b3a]

RISC

RISC stands for Reduced Instruction Set Computer. RISC came about as a result of academic research that showed that a small well designed instruction set running compiled programs at high speed could perform more computing work than a CISC running the same programs (although very expensive hand optimized assembly language favored CISC).

"Some designers began to question whether computers with complex instruction sets are as fast as they could be, having in mind the capabilities of the underlying technology. A few designers hypothesized that increased performance should be possible through a streamlined design, and instruction set simplicity. Thus, research efforts began in order to investigate how processing performance could be increased through simplified architectures. This is the root of the reduced instruction set computer (RISC) design philosophy.

"Seymour Cray has been credited with some of the very early RISC concepts. In an effort to design a very high speed vector processor (CDC 6600), a simple instruction set with pipelined execution was chosen. The CDC 6600 computer was register based, and all operations used data from registers local to the arithmetic units. Cray realized that all operations must be simplified for maximal performance. One complication or bottleneck in processing can cause all other operations to have degraded performance.

"Starting in the mid 1970s, the IBM 801 research team investigated the effect of a small instruction set and optimizing compiler design on computer performance. They performed dynamic studies of the frequency of use of different instructions in actual application programs. In these studies, they found that approximately 20 percent of the available instructions were used

80 percent of the time. Also, complexity of the control unit necessary to support rarely used instructions, slows the execution of all instructions. Thus, through careful study of program characteristics, one can specify a smaller instruction set consisting only of instructions which are used most of the time, and execute quickly.

"The first major university RISC research project was at the University of California, Berkeley (UCB), David Patterson, Carlos Séquin, and a group of graduate students investigated the effective use of VSLI in microprocessor design. To fit a powerful processor on a single chip of silicon, they looked at ways to simplify the processor design. Much of the circuitry of a modern computer CPU is dedicated to the decoding of instructions and to controlling their execution. Microprogrammed CISC computers typically dedicate over half of their circuitry to the control section. However, UCB researchers realized that a small instruction set requires a smaller area for control circuitry, and the area saved could be used by other CPU functions to boost performance. Extensive studies of application programs were performed to determine what kind of instructions are typically used, how often they execute, and what kind of CPU resources are needed to support them. These studies indicated that a large register set enhanced performance, and pointed to specific instruction classes that should be optimized for better performance. The UCB research effort produced two RISC designs that are widely referenced in the literature. These two processors developed at UCB can be referred to as UCB-RISC I and UCB-RISC II. The mnemonics RISC and CISC emerged at this time.

"Shortly after the UCB group began its work, researchers at Stanford University (SU), under the direction of John Hennessy, began looking into the relationship between computers and compilers. Their research evolved into the design and implementation of optimizing compilers, and single-cycle instruction sets. Since this research pointed to the need for single-cycle instruction sets, issues related to complex, deep pipelines were also investigated. This research resulted in a RISC processor for VSLI that can be referred to as the SU-MIPS.

"The result of these initial investigations was the establishment of a design philosophy for a new type of von Neumann architecture computer. Reduced instruction set computer design resulted in computers that execute instructions faster than other computers built of the same technology. It was seen that a study of the target application programs is vital in designing the instruction set and datapath. Also, it was made evident that all facets of a computer design must be considered together.

"The design of reduced instruction set computers does not rely upon inclusion of a set of required features, but rather, is guided by a design philosophy. Since there is no strict definition of what constitutes a RISC design, a significant controversy exists in categorizing a computer as RISC or CISC.

"The RISC philosophy can be stated as follows: *The effective speed of a computer can be maximized by migrating all but the most frequently used functions into software, thereby simplifying the hardware, and allowing it to be faster. Therefore, included in hardware are only those performance features that are pointed to by dynamic studies of HLL programs. The same philosophy applies to the instruction set design, as well as to the design of all other on-chip resources. Thus, a resource is incorporated in the architecture only if its incorporation is justified by its frequency of use, as seen from the language studies, and if its incorporation does not slow down other resources that are more frequently used.*

"Common features of this design philsophy can be observed in several examples of RISC design. The instruction set is based upon a load/store approach. Only *load* and *store* instructions access memory. No arithmetic, logic, or I/O instruction operates directly on memory contents. This is the key to single-cycle execution of instructions. Operations on register contents are always faster than operations on memory contents (memory references usually take multiple cycles; however, references to cached or buffered operands may be as rapid as register references, if the desired operand is in the cache, and the cache is on the CPU chip). Simple instructions and simple addressing modes are used. This simplificiation results in an instruction

decoder that is small, fast, and relatively easy to design. It is easier to develop an optimizing compiler for a small, simple instruction set than for a complex instruction set. With few addressing modes, it is easier to map instructions onto a pipeline, since the pipeline can be designed to avoid a number of computation related conflicts. Little or no microcode is found in many RISC designs. The absence of microcode implies that there is no complex micro-CPU within the instruction decode/control section of a CPU. Pipelining is used in all RISC designs to provide simultaneous execution of multiple instructions. The depth of the pipeline (number of stages) depends upon how execution tasks are subdivided, and the time required for each stage to perform its operation. A carefully designed memory hierarchy is required for increased processing speed. This hierarchy permits fetching of instructions and operands at a rate that is high enough to prevent pipeline stalls. A typical hierarchy includes high-speed registers, cache, and/or buffers located on the CPU chip, and complex memory management schemes to support off-chip cache and memory devices. Most RISC designs include an optimizing compiler as an integral part of the computer architecture. The compiler provides an interface between the HLL and the machine language. Optimizing compilers provide a mechanism to prevent or reduce the number of pipeline faults by reorganizing code. The reorganization part of many compilers moves code around to eliminate redundant or useless statements, and to present instructions to the pipeline in the most efficient order. All instructions typically execute in the minimum possible number of CPU cycles. In some RISC designs, only load/store instructions require more than one cycle in which to execute. If all instructions take the same amount of time, the pipeline can be designed to recover from faults more easily, and it is easier for the compiler to reorganize and optimize the instruction sequence." —Charles E. Gimarc and Veljko M. Milutinovic[b3a]

DSP

DSP stands for Digital Signal Processing. DSP is used primarily in dedicated devices, such as MODEMs, digital cameras, graphics cards, and other specialty devices.

Hybrid

Hybrid processors combine elements of two or three of the major classes of processors.

assembly language

There are four general classes of machine instructions. Some instructions may have characteristics of more than one major group. The four general classes of machine instructions are: computation, data transfer, sequencing, and environment control.

Different RAM Types and its uses

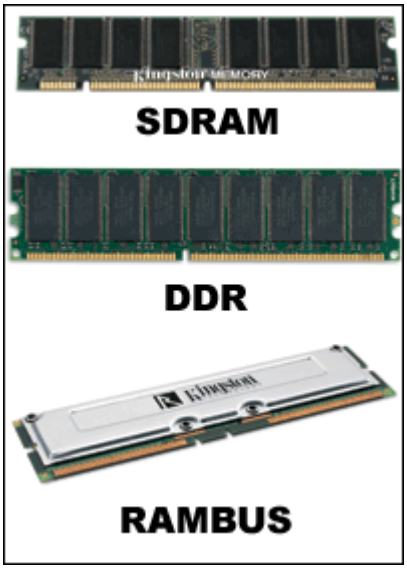
Introduction:

The type of RAM doesn't matter nearly as much as how much of it you've got, but using plain old SDRAM memory today will slow you down. There are three main types of RAM: SDRAM, DDR and Rambus DRAM.

SDRAM (Synchronous DRAM)
Almost all systems used to ship with 3.3 volt, 168-pin SDRAM DIMMs. SDRAM is not an extension of older EDO DRAM but a new type of DRAM altogether. SDRAM started out running at 66 MHz, while older fast page mode DRAM and EDO max out at 50 MHz. SDRAM is able to scale to 133 MHz (PC133) officially, and unofficially up to 180MHz or higher. As processors get faster, new generations of memory such as DDR and RDRAM are required to get proper performance.

DDR (Double Data Rate SDRAM)
DDR basically doubles the rate of data transfer of standard SDRAM by transferring data on the up and down tick of a clock cycle. DDR memory operating at 333MHz actually operates at 166MHz * 2 (aka PC333 / PC2700) or 133MHz*2 (PC266 / PC2100). DDR is a 2.5 volt technology that uses 184 pins in its DIMMs. It is incompatible with SDRAM physically, but uses a similar parallel bus, making it easier to implement than RDRAM, which is a different technology.

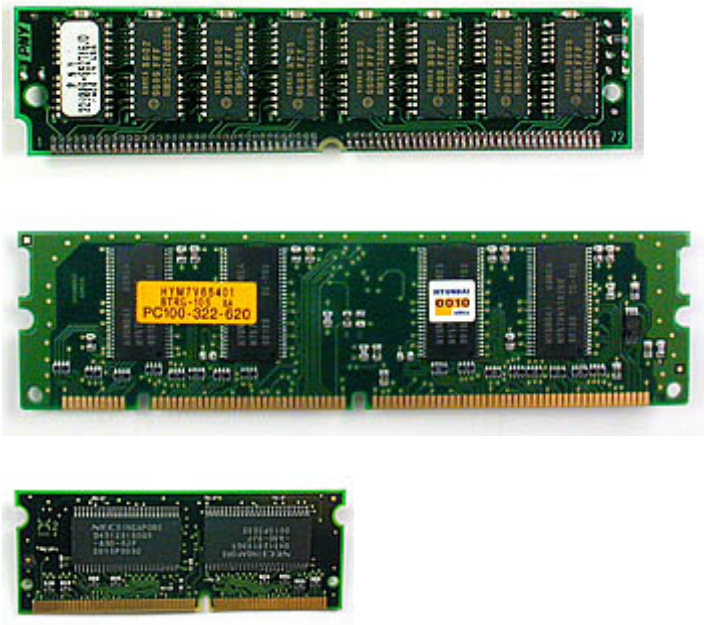
Rambus DRAM (RDRAM)
Despite it's higher price, Intel has given RDRAM it's blessing for the consumer market, and it will be the sole choice of memory for Intel's Pentium 4. RDRAM is a serial memory technology that arrived in three flavors, PC600, PC700, and PC800. PC800 RDRAM has double the maximum throughput of old PC100 SDRAM, but a higher latency. RDRAM designs with multiple channels, such as those in Pentium 4 motherboards, are currently at the top of the heap in memory throughput, especially when paired with PC1066 RDRAM memory.



DIMMs vs. RIMMs
DRAM comes in two major form factors: DIMMs and RIMMS.

DIMMs are 64-bit components, but if used in a motherboard with a dual-channel configuration (like with an Nvidia nForce chipset) you must pair them to get maximum performance. So far there aren't many DDR chipset that use dual-channels. Typically, if you want to add 512 MB of DIMM memory to your machine, you just pop in a 512 MB DIMM if you've got an available slot. DIMMs for SDRAM and DDR are different, and not physically compatible. SDRAM DIMMs have 168-pins and run at 3.3 volts, while DDR DIMMs have 184-pins and run at 2.5 volts.

RIMMs use only a 16-bit interface but run at higher speeds than DDR. To get maximum performance, Intel RDRAM chipsets require the use of RIMMs in pairs over a dual-channel 32-bit interface. You have to plan more when upgrading and purchasing RDRAM.



From the top: SIMM, DIMM and SODIMM memory modules

Memory Speed

SDRAM initially shipped at a speed of 66MHz. As memory buses got faster, it was pumped up to 100MHz, and then 133MHz. The speed grades are referred to as PC66 (unofficially), PC100 and PC133 SDRAM respectively. Some manufacturers are shipping a PC150 speed grade. However, this is an unofficial speed rating, and of little use unless you plan to overclock your system.

DDR comes in PC1600, PC2100, PC2700 and PC3200 DIMMs. A PC1600 DIMM is made up of PC200 DDR chips, while a PC2100 DIMM is made up of PC266 chips. PC2700 uses PC333 DDR chips and PC3200 uses PC400 chips that haven't gained widespread support. Go for PC2700 DDR. It is about the cost of PC2100 memory and will give you better performance.

RDRAM comes in PC600, PC700, PC800 and PC1066 speeds. Go for PC1066 RDRAM if you can find it. If you can't, PC800 RDRAM is widely available.

CAS Latency

SDRAM comes with latency ratings or "CAS (Column Address Strobe) latency" ratings. Standard PC100 / PC133 SDRAM comes in CAS 2 or CAS 3 speed ratings. The lower latency of CAS 2 memory will give you more performance. It also costs a bit more, but it's worth it.

DDR memory comes in CAS 2 and CAS 2.5 ratings, with CAS 2 costing more and performing better.

RDRAM has no CAS latency ratings, but may eventually come in 32 and 4 bank forms with 32-bank RDRAM costing more and performing better. For now, it's all 32-bank RDRAM.

Understanding Cache

Cache Memory is fast memory that serves as a buffer between the processor and main memory. The cache holds data that was recently used by the processor and saves a trip all the way back to slower main memory. The memory structure of PCs is often thought of as just main memory, but it's really a five or six level structure:

The first two levels of memory are contained in the processor itself, consisting of the processor's small internal memory, or **registers**, and **L1 cache**, which is the first level of cache, usually contained in the processor.

The third level of memory is the **L2 cache**, usually contained on the motherboard. However, the Celeron chip from Intel actually contains 128K of L2 cache within the form factor of the chip. More and more chip makers are planning to put this cache on board the processor itself. The benefit is that it will then run at the same speed as the processor, and cost less to put on the chip than to set up a bus and logic externally from the processor.

The fourth level, is being referred to as **L3 cache**. This cache used to be the L2 cache on the motherboard, but now that some processors include L1 and L2 cache on the chip, it becomes L3 cache. Usually, it runs slower than the processor, but faster than main memory.

The fifth level (or fourth if you have no "L3 cache") of memory is the **main memory** itself.

The sixth level is a piece of the hard disk used by the Operating System, usually called **virtual memory**. Most operating systems use this when they run out of main memory, but some use it in other ways as well.

This six-tiered structure is designed to efficiently speed data to the processor when it needs it, and also to allow the operating system to function when levels of main memory are low. You might ask, "Why is all this necessary?" The answer is cost. If there were one type of super-fast, super-cheap memory, it could theoretically satisfy the needs of this entire memory architecture. This will probably never happen since you don't need very much cache memory to drastically improve performance, and there will always be a faster, more expensive alternative to the current form of main memory.

Memory Redundancy

One important aspect to consider in memory is what level of redundancy you want. There are a few different levels of redundancy available in memory. Depending on your motherboard, it may support all or some of these types of memory:

The cheapest and most prevalent level of redundancy is **non-parity memory**. When you have non-parity memory in your machine and it encounters a memory error, the operating system will have no way of knowing and will most likely crash, but could corrupt data as well with no way of telling the OS. This is the most common type of memory, and unless specified, that's what you're getting. It works fine for most applications, but I wouldn't run life support systems on it.

The second level of redundancy is **parity memory** (also called true parity). Parity memory has extra chips that act as parity chips. Thus, the chip will be able to detect when a memory error has occurred and signal the operating system. You'll probably still crash, but at least you'll know why.

The third level of redundancy is **ECC** (Error Checking and Correcting). This requires even more logic and is usually more expensive. Not only does it detect memory errors, but it also corrects 1-bit ECC errors. If you have a 2-bit error, you will still have some problems. Some motherboards enable you to have ECC memory.

Older memory types

Fast Page Mode DRAM

Fast Page Mode DRAM is plain old DRAM as we once knew it. The problem with standard DRAM was that it maxes out at about 50 MHz.

EDO DRAM

EDO DRAM gave people up to 5% system performance increase over DRAM. EDO DRAM is like FPM DRAM with some cache built into the chip. Like FPM DRAM, EDO DRAM maxes out at about 50 MHz. Early on, some system makers claimed that if you used EDO DRAM you didn't need L2 cache in your computer to get decent performance. They were wrong. It turns out that EDO DRAM works along with L2 cache to make things even faster, but if you lose the L2 cache, you lose a lot of speed.