BIOS

Definitions

BIOS

Basic Input Output System. All computer hardware has to work with software through an interface. The BIOS gives the computer a little built-in starter kit to run the rest of softwares from floppy disks (FDD) and hard disks (HDD). The BIOS is responsible for booting the computer by providing a basic set of instructions. It performs all the tasks that need to be done at start-up time: POST (Power-On Self Test, booting an operating system from FDD or HDD). Furthermore, it provides an interface to the underlying hardware for the operating system in the form of a library of interrupt handlers. For instance, each time a key is pressed, the CPU (Central Processing Unit) perform an interrupt to read that key. This is similar for other input/output devices (Serial and parallel ports, video cards, sound cards, hard disk controllers, etc...). Some older PC's cannot co-operate with all the modern hardware because their BIOS doesn't support that hardware. The operating system cannot call a BIOS routine to use it; this problem can be solved by replacing your BIOS with an newer one, that does support your new hardware, or by installing a device driver for the hardware.

CMOS

Complementary Metal Oxide Semiconductor. To perform its tasks, the BIOS need to know various parameters (hardware configuration). These are permanently saved in a little piece (64 bytes) of CMOS RAM (short: CMOS). The CMOS power is supplied by a little battery, so its contents will not be lost after the PC is turned off. Therefore, there is a battery and a small RAM memory on board, which never (should...) loses its information. The memory was in earlier times a part of the clock chip, now it's part of such a highly Integrated Circuit (IC). CMOS is the name of a technology which needs very low power so the computer's battery is not too much in use.

Actually, there is not a battery on new boards, but an accumulator (Ni_Cad in most cases). It is recharged every time the computer is turned on. If your CMOS is powered by external batteries, be sure that they are in good operating condition. Also, be sure that they do not leak. That may damage the motherboard. Otherwise, your CMOS may suddenly "forget" its configuration and you may be looking for a problem elsewhere. In the monolithic PC and PC/XT, this information is supplied by setting the DIP (Dual-In-line Package) switches at the motherboard or peripheral cards. Some new motherboards have a technology

named the Dallas Nov-Ram. It eliminates having an on-board battery: There is a 10 year lithium cell epoxyed into the chip.

Chipset

A PC consists of different functional parts installed on its motherboard: ISA (Industry Standard Architecture), EISA (Enhanced Industry Standard Architecture) VESA (Video Enhanced Standards Association) and PCI (Peripheral Component Interface) slots, memory, cache memory, keyboard plug etc... Not all of these are present on every motherboard. The chipset enables a set of instructions so the CPU can work (communicate) with other parts of the motherboard. Nowadays most of the discrete chips; PIC (Programmable Interrupt Controller), DMA (Direct Memory Access), MMU (Memory Management Unit), cache, etc... are packed together on one, two or three chips; the chipset. Since chipsets of a different brand are not the same, for every chipset there is a BIOS version. Now we have fewer and fewer chipsets which do the job. Some chipsets have more features, some less. OPTi is such a commonly used chipset. In some well integrated motherboards, the only components present are the CPU, the two BIOS chips (BIOS and Keyboard BIOS), one chipset IC, cache memory (DRAMs, Dynamic Random Access Memory), memory (SIMMs, Single Inline Memory Module, most of the time) and a clock chip.

Setup

Setup is the set of procedures enabling the configure a computer according to its hardware caracteristics. It allows you to change the parameters with which the BIOS configures your chipset. The original IBM PC was configured by means of DIP switches buried on the motherboard. Setting PC and XT DIP switches properly was something of an arcane art. DIP switches/jumpers are still used for memory configuration and clock speed selection. When the PC-AT was introduced, it included a battery powered CMOS memory which contained configuration information. CMOS was originally set by a program on the Diagnostic Disk, however later clones incorporated routines in the BIOS which allowed the CMOS to be (re)configured if certain magic keystrokes were used.

Unfortunately as the chipsets controlling modern CPUs have become more complex, the variety of parameters specifiable in SETUP has grown. Moreover, there has been little standardization of terminology between the half dozen BIOS vendors, three dozen chipset makers and large number of motherboard vendors. Complaints about poor motherboard documentation of SETUP parameters are very common.

To exacerbate matters, some parameters are defined by BIOS vendors, others bychipset designers, others by motherboard designers, and others by various combinations of the above. Parameters intended for use in Design and Development, are intermixed with parameters intended to be adjusted by technicians -- who are frequently just as baffled by this stuff as everyone else is. No one person or organization seems to understand all the parameters available for any given SETUP.

Hardware Performance

· System Timing

Although computers may have basic similarities (they all look the same on a shelf), performance will differ markedly between them, just the same as it does with cars. The PC contains several processes running at the same time, often at different speeds, so a fair amount of coordination is required to ensure that they don't work against each other.

Most performance problems arise from **bottlenecks** between components that are not necessarily the best for a particular job, but a result of compromise between price and performance. Usually, price wins out and you have to work around the problems this creates.

The trick to getting the most out of any machine is to make sure that each component is giving of its best, then eliminate potential bottlenecks between them. You can get a bottleneck simply by having an old piece of equipment that is not designed to work at modern high speed - **a computer is only as fast as its slowest component**, but bottlenecks can also be due to badly written software.

System Timing

The clock is responsible for the speed at which numbers are crunched and instructions executed. It results in an electrical signal that switches constantly between high and low voltage several millions times a second.

The **System Clock**, or CLKIN, is the frequency used by the processor; on 286s and 386s, this will be half the speed of the main crystal on the motherboard (the CPU devides it by two), which is often called CLK2IN. 486 processors run at the same speed, because they use both edges of the timing signal. A clock generator chip (82284 or similar) is used to synchronize timing signals around the computer, and the data bus would be run at slower speed synchronously with the CPU, e.g. CLKIN/4 for an ISA bus with a 33 MHz CPU.

ATCLK is a separate clock for the bus, when it's run asynchronously, or not derived from CLK2IN. There is also a 14.138 MHz crystal which was used for all system timing on XTs. Now it's only used for the colour frequency of the video controller (6845).

The Year 2000 problem

As far as the PC hardware is concerned, the problem lies with the *Real Time Clock*, or RTC, and its relation to the internal DOS clock device driver (CLOCK$), which is actually a counter and not a real clock at all. You can verify if your system has this problem by setting the date just before midnight 1999 and leaving the machine running to see what happen when it reaches 2000. DOS copes with the problem quite easily. However, if you turn the power off and reboot, you might see a system date starting somewhere in 1980 (do not log on to the network, if it is the case, otherwise you might get the server

time). The date 01/01/1980 is usually set if your CMOS contents are lost, and 01/04/1980 if an out-of-range date is encountered.

The reason for this discrepancy is the interaction between the two clocks mentioned above. The RTC lives inside the CMOS chip that keeps the BIOS settings and is kept alive by the battery. Some of these cannot keep track of the centuries by themselves, so a byte is used in the CMOS to do the job instead. Also, they are trimmed at the factory to a certain tolerance, typically plus or minus 20 seconds a month, which will only be kept to if the desired operating environment is kept (temprerature, humidity, etc.).

The DOS device driver (CLOCK$), on the other hand, only interrogates the RTC (via the BIOS) when the machine starts, then proceeds to ignore it as long as the PC is running. This is to save going as far as the RTC for the time, which is a slower process. The date supplied is converted to the number of days since January 1st 1980, and the number of seconds since midnight of the current day. The latter is stored in the counter by the BIOS, and when DOS needs to read the clock, the BIOS is called to read the counter and the number of ticks is converted back to seconds. If the counter goes past midnight, it is reset to zero by the BIOS, and the first caller after that is told that the day has advanced. As a result, if more thant 24 hours has elapsed between calls, there is no way that DOS can tell which day it is.

This is why there is often a time difference between your watch and your PC at the end of the day; the system clock has to compete for attention with other devices, and is often reprogrammed by games, etc, who use it for their own timing purposes, mostly to do with running the video faster. In short, being **interrupt driven**, its accuracy depends on system activity.

As DOS operates between 1980 and 2099, it can figure out that 00 equates to 2000, although it may have problems if the RTC specifically hands it a date of 1900, or any other incompatible date. In practice, the BIOS converts it as well; some correct the time automatically at boot and some will supply DOS with 2000 instead of a hardware date of 1900. Others cannot produce a date later than 1999 anyway; Award BIOS 4.5G prior to November 1995 can only accept dates between 1994 and 1999. A good way to solve the 2000 problem (for the BIOS aspect of it at least) is the **change your motherboard**. Have a look at www.year2000.com <http://www.year2000.com> or www.sbhs.com/y2k <http://www.sbhs.com/y2k>.

The Motherboard

This is a large circuit board to which are fixed the **Central Processor** (CPU), the data bus, memory and various support chips, such as those that control speed and timing the keyboard, etc. The CPU does all the thinking, and is told what to do by instructions contained in memory, so there will be a direct two-way connection between them. The data bus is actually a part of the CPU, although it's treated separately.

Extra circuitry in the form of **expansion cards** is placed into **expansion slots** on the data bus, so the basic setup of the computer can be changed easily (for example, you can connect more disk drives or a modem there).

A math co-processor is frequently fitted alongside the main processor, which is specially built to cope with non-integer arithmetic (e.g. decimal points). The main processor has to convert decimals and fractions to whole numbers before calculating them, and then has to convert them back again.

The Central Processor

The chip that was the brains of the original IBM PC was called the 8088, manufactured by Intel. In the days Intel developed the 8088, microprocessors were classified by their external databus. Intel has thus officially classified their 8088 as an 8-bit HMOS microprocessor. Although it was internally classified as being 16-bit, it spoke to the data bus and memory with 8 bits in order to keep the costs down and keep in line with the capabilities of the support chips. Then when it wanted to send two characters to the screen over the data bus, it had to send them one at a time, rather than both together, so there was an idle state where nothing was done every time the data was sent.

In addition, it could only talk to 1 Mb of memory at any time; there were 20 physical connections between it and the CPU. On a binary system this represents 220, or 1,048,576.

The 80286

The 80286 was introduced in response to competition from manufacturers who were cloning the IMB PC. The connections between the various parts of the motherboard became 16-bit throughout, thus increasing efficiency 4 times. It also has 24 memory address lines, so it could talk to 16Mb of physical memory. However, DOS could not use it since it had to be addressed in protected mode. DOS can only run in real mode, which is restricted to the 1 Mb that can be seen by the 8088. Therefore, a Pentium running DOS is just a fast XT. Just as the 8088, the 80286 CPU is limitted to 1MB (+ 64KB) when running in real mode. The 80286 CPU has to run in protected mode to access extended memory. On a 80286 system, DOS' extended memory manager (himem.sys) uses BIOS service INT 15h/AH=87h to move data from/to extended memory. INT 15h enters and leaves protected mode.

The 80386

Compaq was the first company to use the 80386 (the DX version, as opposed to the SX-see below), which uses 32 bits between itself and memory, but 16 towards the data bus, which has not really been developed in tandem wit the rest of the machine. This is partly to ensure backwards compatibility and partly due to the plumbing arrangements of running a fast CPU with faster memory and a slow bus (8 MHz).

The 386 can run multiple copies of real mode (that is, it can create several virtual 8088s). It uses paging to remap memory so that these machines are brought to the attention of the CPU when the programs in them require it; this is done on a timeslice basis, around 60 times a second, which is how we get multitasking in Windows of OS/2 (in 95 the slice is every 20 ns).

The 386 can also switch out of protected mode on the fly, or at least in a more elegant way that the 286; in order to get the hard disk and other parts of the computer, protected-mode software hat to get DOS to perform real mode services, so the CPU has to switch in and out of protected mode continually. The goal is therefore to use real mode as little as possible and to run in protected mode. Windows does this by using 32-bit instructions. Because of 32-bit addressing, the 80386 and above CPU's are not limitted to 1MB when running in real mode. Because the address bus is also 32-bit, any address can be reached using 0000:<32-bit offset>. Also, DOS segments larger than 64KB can be handled as a whole instead using chunks of 64KB, or by using normalized (huge) pointers. On 80386+ systems, DOS' extended memory manager simply uses a 32-bit block move instruction to move data from/to extended memory.

The 386 uses pipelining to help streamline memory access-the idea is that they are done independently of each other (at the same time) while other units get on with their jobs; a form of primitive parallel processing. The 386 also has a pre-fetch unit for instructions, that tries to speed things up by guessing which ones the processor will use next.

Although the 386 is 32-bit and has certain benefits, like the ability to manipulate memory and switch in and out of protected mode more readily, replacing a 286 with a 386 does not automatically give you performance benefits it you are running 16-bit 80286 code (most DOS programs).

The 80386SX

The 80386Sx is a 32-bit chip internally, but 16-bit externally to both memory and the data bus, so you get bottlenecking. It is a cut-down version of the 80386DX, created to both cut costs and give the impression the 286 was obsolete (true), because at the time other manufacturers could make the 286 under license. Although it can run 836-specific software, it looks like a 286 to the machine it is in, so existing 286 motherboards could be used to plug in 386SX CPUs. At the same clock speed, a 386SX is around 25% slower than a 386DX.

The 80486

To non-technical people, the 80486 is a fast 80386DX with an on-board math co-processor and 8K of cache memory. It is not really a newer technology as such (only second generation), but better use is made of its facilities. For example, it takes fewer instruction cycles to do the same job, and is optimized to keep as many operations inside the chip as possible. The 386 pre-fetch unit was replaced by 8K of SRAM cache, and pipelining was replaced by burst mode, which works on the theory that most of the time spent getting data concerns its address. Burst allows a device to send large amounts of data in a short time without interruption. Pipelining on the 386 requires 2 clocks per transfer; only one is needed with 486 Burst Mode. Memory parity checks also take their own path at the same time as the data they relate to. The 486 has an on-board clock, and both edges of the square wave signal are used to calculate the clock signal, so the motherboard runs at the same speed as the CPU. In addition, the bus system uses a single pulse cycle. Generally speaking, at the same clock speed, a 486 will deliver between 2-3 times the performance of a 386.

The 80486SX

The 486SX is as above, but with the math co-processor facility disabled, therefore you should find no significant difference between it and a 386; a 386/40 is broadly equivalent to a 486/25.

Clock Doubling

The DX/" chip runs at double speed of the original, but it is not the same as having a proper high speed motherboard because the bus will still be running at the normal speed. Unfortunately, high speed motherboards are more expensive because of having to design out RF emissions, and the like.

Actual performance depends on how many accesses are satisfied from the chip's cache, which is how the CPU is kept busy, rather than waiting for the rest of the machine. If the CPU has to go outside the cache, effective speed is the same as the motherboard or, more properly, the relevant bus (memory or data), so best performance is obtained when all the CPU's needs are satisfied from inside itself. However, performance is still good if it has to use cache, as the hit rate is around 90%. The DX4 has a larger cache

(16K) to cope with the higher speed.

The Pentium

Essentially two 486s in parallel (or rather an SX and a DX), so more instructions are processed at the same time; typically two at once. This, however, depends on whether software can take advantage of it, and get the timing of the binary code just right. It has separate 8K caches, for instructions and data, split into banks which can be accessed alternately. It has a 64-bit bus, to cope with 2 32-bit chips.

The Pentium Pro

This is a RISC chip with a 486 hardware emulator on it, running at 200 MHz of below. Several techniques are used by this chip to produce more performance than its predecessors; speed is achieved by dividing processing into more stages, and more work is done within each clock cycle; three instructions can be decoded in each one, as opposed to two for the Pentium.

In additions, instruction decoding and execution are decoupled, which means that instructions can still be executed if one pipeline stops (such as when one instruction is waiting for data from memory; the Pentium would stop all processing at this point). Instructions are sometimes executed out of order, that is, not necessarily as written down in the program, but rather when information is available, although they won't be much out of sequence; just enough to make things run smoother.

It has a 8K cache for programs and data, but it will be a two chip set, with the processor and a 256K L2 cache in the same package. It is optimized for 32-bit code, so will run 16-bit code no faster than a Pentium.

Cyrix

The 6x86 is a Pentium-type chip with Pentium Pro characteristics, as it can execute faster instructions out of sequence. They use a *P-Rating* to determine performance relative to the Pentium, so a 6x86-166 is equivalent to a Pentium 166, even if running at 133 MHz.

Summing up

In principle, the faster the CPU the better, but only if your applications do a lot of logical operations and calculation (where the work is centered around the chip) rather than writing to disk. For example, when a typical word processing task, replacing a 16 MHz 386 with a 33 MHz one (doubling the speed) will only get you something like a 5-10% increase in practical performance, regardless of what the benchmarks might say. **It is often a better idea to spend money on a faster hard disk**.

Also, with only 8 Mb RAM in your computer, you won't see much performance increase from a DX2/66 until you get a Pentium 90 (none at all between a DX4/100 and a Pentium 75). With Windows, this is because the hard disk is used a lot for virtual memory (swap files), which means more activity over the data bus. Since motherboards below the 90 run at 33Mhz (only the chips run faster), the bottleneck is the disk I/O, running at much the same speed on them all. This is especially true if you use Programmed I/O (PIO), where the CPU must scrutinize every bit to and from the hard drive (although Multi-sector I/O or EIDE will improve things). As the Pentium 90's motherboard runs faster (60 MHz), the I/O can proceed at a much faster pace, and performance will more than double (a more sophisticated chipset helps).

With 16 Mb, on the other hand, performance will be almost double anyway, regardless of the processor,

because the need to go to the hard disk is so much reduced, and the processor can make a contribution to performance. The biggest jump is from a DX2/66 to a DX/4, with the curve flattening out progressively up to the Pentium 90.

Processor	MB Speed	Clock X	Bus Speed
P60	60	1	30
P66	66	1	33
P75	50	1.5	25
P90	60	1.5	30
P100	66	1.5	33
P120	60	2	30
P133	66	2	33
P150	60	2.5	30
P166	66	2.5	33
P200	66	3	33
6x86-120	50	2	25
6x86-133	55	2	?
6x86-150	60	2	30
6x86-166	66	2	33

Memory

Memory Types

Unlike the good old days of mainframe computers when there were only two types of memory to deal with, the PC uses five, again for historical reasons. The memory contains the instructions that tell the Central Processor what to do, as well as the data created by its activities. Since the computer works with the binary system, memory chips work by keeping electronic switches in one state or the other for however long they are required. Actually, they consist of a capacitor and a transistor; the capacitor stores a charge (data), which represents a 1, and the transistor acts as a switch that turns the charge on or off. Where these states can be changed at will, it is called **Random Access Memory**, or RAM. The term derives from when magnetic tapes were used for data storage, and the information could only be accessed sequentially. A ROM, on the other hand, is a memory chip with its electronic switches permanently on of off, so they can't be changed, hence **Read Only Memory**.

· **Static RAM** (SRAM) is the fastest available, with a typical access time of 25 nanoseconds. Static RAM is expensive and can only store a quarter of the data that DRAM is able to in the same given area, although it does retain it for as long as the chip is powered. The transistors are connected so that only one is either in or out at any time; whichever one is in stands for a 1 bit. Synchronous SRAM allows a faster data stream to pass through it; which is needed when used for cacheing on 90 and 100 MHz Pentium.

· **Dynamic RAM** (DRAM) uses internal capacitors to store data (a single transistor turns it on of off) which lose their charge over time, so they need constant refreshing to retain information, otherwise 1s will turn to 0s. The end result is that between every memory access is sent an electrical charge that refreshes the chip's capacitors to keep data in a fit state, which cannot be reached whilst recharging is going on. Reading a DRAM discharges its contents, so they have to be written back to immediately to keep the sane information.

· **Enhanced DRAM** (EDRAM) replaces standard DRAM and the SRAM in the level 2 cache on the motherboard, typically combining 256 bytes of 15ns SRAM inside 35ns DRAM. Since the SRAM can take a whole 256 byte page of memory at once, it gives an effective 15ns access speed when you get a hit (35ns otherwise). The level 2 cache is replaced with an SIC chip to sort out chipset vs. memory requirements. System performance is increased by around 40%. EDRAM has a separate write path that accepts and completes requests without the rest of the chip.

· **WRAM** (Windows RAM), created by Samsung, is dual ported, but costs about 20% less than VRAM and is 50% faster. It runs at 50 MHz and is optimized for acceleration and can transfer blocks and supports text and pattern fills. Mostly used for video cards.

· **Synchronous DRAM** (SDRAM) takes memory access away from the CPU's control; internal registers in the chips accept the request, and let the CPU do something else while the data requested is assembled for the next time the CPU talks to the memory. As they work on their own clock, the rest of the system can be clocked faster. There is a version optimized for video cards.

· **EDO** (Extended Data Output) is an advanced version of fast page mode (often called Hyper Page

Mode), which can be up to 30% better and only cost 5% more. Single-cycle EDO will carry out a complete memory transaction in 1 clock cycle; otherwise, each sequential RAM access inside the same page takes 2 clock cycles instead of 3, once the page has been selected. As it replaces level 2 cache and doesn't need a separate controller, space on the motherboard is saved, which is good for notebooks. It also saves battery power. In short, EDO gives and increased bandwidth due to shortening of the page mode cycle, but it doesn't appear to be that much faster in practice.

Memory Access

The **cycle time** is the time it takes to read from and write to a memory cell, and it consists of two stages; precharge and access. **Precharge** is where the capacitor in the memory cell is able to recover from a previous access and stabilize. **Access** is where a data bit is actually moved between memory and the bus or the CPU. Total **access time** includes the finding of data, data flow and recharge, and parts of the access time can be eliminated or overlapped to improve performance. The combination of precharge and access equals cycle time, which is what you should use to calculate wait states from.

There are ways of making refreshes happen so that the CPU doesn't notice (i.e. Concurrent and Hidden), which is helped by the 486 being able to use its on-board cache and not needing to use memory so often anyway. In addition, you can affect the **Row Access Strobe** (RAS), or have **Column Access Strobe** (CAS) before RAS (see Advanced Chipset Setup <advchip.htm>).

The fastest DRAM commonly available is rated at 60ns. As these chips need alternate refresh cycles, under normal circumstances data will actually be obtained every 120ns, giving you and effective speed of around 8 MHz for the whole computer, regardless of the CPU speed, assuming no action is taken to compensate. Memory chips therefore need to be operating at something like 20ns to keep up, assuming that the CPU needs only one clock cycle for each one from the memory bus; one internal cycle for each external one. Intel processors mostly use two for one, so the 33 MHz CPU is actually ready to use memory every 60ns, but you need to allow a little more for overheads, such as data assembly and the like. One way of matching the capacities of components with different speeds includes the use of wait states.

Clock Speed (MHz)	Cycle Time (ns)
1	1000
5	200
8	125
12	83
16	63
20	50
25	40
33	30

40 25

Wait States

A wait state indicates how many ticks of the system clock the CPU has to wait for memory to catch up-it will generally be 0 or 1, but can be up to 3 if you're using slower memory chips. Ways of avoiding wait states include:

· **Page-mode memory**. This will cut-down address cycles to retrieve information form one general area, based on the fact that the second access to a memory location on the same page takes around half the time as the first; addresses are normally in two halves, with high bits (for row) and low bits (for column) being multiplexed onto one set of address pins. The page address of data is noted, and if the next data is in the same area, a second address cycle is eliminated as a whole row of memory cells can be read in one go; that is, once a row access has been made, you can get to subsequent column addresses in that row in the time available (you should therefore increase row access time for best performance). Otherwise data is retrieved normally, which will take twice as long. **Fast Page Mode** is a quicker version of the same thing; the DRAMs concerned have a faster CAS access speed. Memory capable of running in page mode is different from normal bit-by-bit type, and the two don't mix. It's unlikely that low capacity SIMMs are so capable.

· **Interleaved memory**, which divides memory into two or four portions that process data alternately; that is, the CPU sends information to one section while another goes through a refresh cycle; a typical installation will have odd addresses on one side and even on the other (you can have word or block interleave). If memory accesses are sequential, the precharge of one will overlap the access time of the other. To put interleaved memory to best use, fill every socket you've got (that is, eight 1 Mb SIMMs are better than two 4 Mb ones). The SIMM types must be the same. As an example, a machine in non-interleaved mode (say a 386SX/20) may need 60ns or faster DRAM for 0ws access, where 80ns chip could do if interleaving were enabled.

· A processor **RAM cache**, which is a bridge between the CPU and slower main memory; it consists of anywhere between 32-512K of (fast) Static RAM chips and is designed to retain the most frequently accessed code and data from main memory. It can make 1 wait state RAM look like that with 0 wait states, without physical adjustments, assuming that the data the CPU wants is in the cache when required (known as a **cache hit**). To minimize the penalty of a cache miss, cache and memory access are often in parallel, with one being terminated when not required. On a 486, how much cache you need really depends on the amount of memory; Dell say that jumping from 128K to 256K only increases the hit rate by around 5% and Viglen say you only need more than 256K if you have more than 32 Mb RAM. A cache should be fast and capable of holding the contents of several different parts of main memory. Software plays a part as well, since cache operation is based on the assumption that programs access memory where they have done so already, or are likely to next, maybe through looping (where code is reused) or code is organized to be next to other relevant parts. A basic cache design will look up an address for the CPU and return the data inside one clock cycle, or 20ns at 50 MHz. *Asynchronous* SRAM will be used for this. As the round trip from the CPU to cache and back again takes up a certain amount of time, only the remainder is available to retrieve data, which gets smaller as the motherboard speed is increased. *Synchronous* SRAM uses a buffer to keep the whole routine inside one clock cycle, even though it may use two (or more) clock cycles the first time round. The address from the CPU is stored,

and while the next is coming in to the buffer, the data for the first is retrieved, and the cycle continues. Pipeline SRAM uses more clock cycles, typically three, the first time round, and Burst SRAM will deliver 4 words (blocks of data) over for consecutive cycles if the request from the CPU is for the first; there will be no waiting for the CPU to request each one individually. Note the level 2 cache can be unreliable, so be prepared to disable it in the interests of reliability. For maximum efficiency, or minimum access time, a cache may be subdivided into smaller blocks that can be separately loaded, so the chances of a different part of memory being requested and the time needed to replace a wrong section are minimized. There are three mapping schemes that assist with this:

· **Fully Associative**, where the whole address is kept with each block of data in the cache (in tag RAM), needed because it is assumed there is no relationships between the blocks. This can be inefficient, as an address comparison needs to be made with every entry each time the CPU presents the address for its next instruction.

· **Direct Mapped**, where every block can only be in one place in the cache, so only one address comparison is needed to see if the data required is there. Although simple, the cache controller must go to main memory more frequently if program code needs to jump between locations with the sane index, which defeats the object somewhat, as alternate references to the same cache cell mean cache misses for other processes. The "index" comes form the lower order addresses presented by the CPU.

· **Set Associative**, a compromise between the above two. Here, an index can select several entries, so in a *2 Way Set Associative* cache, 2 entries can have the same index, so two comparisons are needed to see if the data required is in the cache. Also, the tag field is correspondingly wider and needs larger SRAMs to store address information. As there are two locations for each index, the cache controller has to decide which one to update or overwrite, as the case may be. The most common methods used to make these decisions are *Random Replacement*, *First In First Out* (FIFO) and *Least Recently Used* (LRU). The latter is the most efficient. It the cache is large enough (e.g. 64K), performance from this over direct-mapping may not be much. A *Write Thru Cache* means that every write access is immediately passed on to memory; although it means that cache contents are always identical to main memory, it is slow, as the CPU then has to wait for DRAMs. Buffers can be used to provide a variation on this, where data is written into a temporary buffer so the CPU is released quickly before main memory is updated. A *Write Back Cache*, on the other hand, exists where changed data is temporarily stored in the cache and written to memory when the system is quiet, or when absolutely necessary. This will give better performance when main memory is slower than the cache, or when several writes are made in a very short space of time, but is more expensive. A "dirty bit" is used as a mental note that the cache and main memory contents are different, and that the cache contains the most up to date data. This bit will be checked if the cache needs to be written to, and main memory updated first if this bit is set. Some motherboards don't have the required SRAM for the dirty bit, but it's still faster than Write Thru.

Shadow RAM

ROMs are used by components that need their own instructions to work properly, such as video card of cacheing disk controller. ROMs are 8-bit devices, so only one byte is accessed at a time; also, they typically run between 150-400ns, so using them will be slow relative to 32-bit memory at 60-80ns, which is capable of making four accesses at once.

Shadow RAM is the process of copying the contents of a ROM directly into extended memory which is given the same address as the ROM, from where it will run much faster. The original ROM is then disabled, and the new location write protected. If your applications execute ROM routines often enough, enabling Shadow RAM will make a difference in performance of around 8%, assuming a program spends about 10% of its time using instructions from ROM, but theoretically as high as 300%. The drawback is that the RAM set aside for shadowing cannot be used for anything else, and you will lose a corresponding amount of extended memory, The remainder of Upper Memory, however, can usually be remapped to the end of extended memory and used there.

With some VGA cards, if video shadow is disabled, you might get DMA errors, because of timing when code is fetched from the VGA BIOS, when the CPU cannot accept DMA requests. Some programs don't make use of the video ROM, preferring to directly address the card's registers, so you may want to use extended memory for something else. If you machine hangs during the startup sequence for no apparent reason, check that you haven't shadowed an area of upper memory containing a ROM that doesn't like it-particularly one on a hard disk controller, or that you haven't got two in the same 128K segment.

Base Memory

The first 640K available, which traditionally contains DOS, device drivers, TSRs and any programs to be run, plus their data, so the less room DOS takes up, the more there is for the rest. Different versions of DOS were better or worse in this respect. In fact, under normal circumstances, you can expect the first 90K or so to consist of:

· An *Interrupt Vector Table*, which is 1K in size, including the name and address of the program providing the interrupt service. Interrupt vectors point to routines in the BIOS or DOS that programs can use to perform low level hardware access. DOS uses **io.sys** and **msdos.sys** for the BIOS and DOS, respectively.

· ROM BIOS tables, which are used by system ROMs to keep track of what's going on. This will include I/O addresses and possibly user-defined hard disk data.

· DOS itself, plus any associated data files it needs to operate with (e.g. buffers, etc.).

DOS was written to run applications inside the bottom 640K block simply because the designers of the original IBM PC decided to. Memory at the time was expensive, and most CP/M machines only used 64K anyway (the PC with 128K was $10,000!). Other machines of the same era used more; the Sirius allowed 896K for programs. Contrary to popular belief, Windows 3.1 uses memory below 1Mb, for administration purposes; although it pools all memory above and below 1 Mb (and calls it the *Global Heap*), certain essential Windows 3.1 structures must live below 1 Mb, such as the *Task DataBase* (TDB) which is necessary for starting new tasks.

Every Windows 3.1 application needs 512 bytes of memory below 1 Mb to load, but some will take much more if they can, even all that's available, thus preventing others from loading, which is one source of "Out of Memory" messages. There are programs that will purposely fragment base memory so it can't be hogged by any one program.

Rather than starting at 0 and counting upwards, memory addressing on the PC uses a two-step

segment:offset addressing scheme. The *segment* specifies a 16-byte paragraph of RAM; the *offset* identifies a specific byte within it. The CPU finds a particular byte in memory by using two registers. One contains the starting segment value and the other the offset. The maximum that can be stored in one is 65,535 (FFFF in hex). The CPU calculates a physical address by taking the contents of the segment register, shifting it one character to the left, and adding the two together (see High Memory).

Sometimes, you will see both values separated by a colon, as with FFFF:000F, meaning the sixteenth byte in memory segment FFFF; this can also be represented as the effective address 0FFFFFh. When referring only to 16-byte paragraph ranges, the offset value is often left out. The 1025KB of DOS memory is divided into 16 segments of 64KBeach. Conventional memory contains the ten segments from 0000h to 9FFFh (bytes 0 to 6555,167), and Upper memory contains the six segments ranging from A000h to FFFFh.

Upper Memory

The next 384K is reserved for private use by the computer, so that any expansion cards with their own memory or ROMs can operate safely there without interfering with programs in base memory, and *vice versa*. Typical examples include network interface cards or graphics adapters. **There is no memory in it**; the space is simply reserved. This is why the memory count on older machines with only 1 Mb was 640 + 384K of *extended memory*; the 384K was remapped above 1 Mb so it could be used. When upper memory blocks are needed, that memory is remapped back again, so you lose a bit of extended memory.

This area is split into regions, A-F, which in turn are split into areas numbered from 0000 to FFFF hexadecimally (64K each). With the right software, this area can be converted in *Upper Memory* for use by TSR (memory-resident programs) to make more room downstairs. The amount of upper memory available varies between computers, and depends on the amount of space taken up by the System BIOS and whether you have a separate VGA BIOS (on board video sometimes has its BIOS integrated in the system BIOS). It also depends on the number of add-in cards you have, e.g. disk controllers, that normally take up around 16K.

Some chipsets will always reserve this 384K area for shadowing, so it will not appear in the initial memory count on power-up, the system configuration screen, or when using MEM. Other chipsets have a *Memory Relocation* option which will re-address it above 1 Mb as extended memory. Occasionally, some ROM space is not needed once the machine has booted, and you might be able to use it. A good example is the first 32K of the System BIOS, at F000 in ISA machines. It's only used in the initial stages of booting up, that is, before DOS gets to set up device drivers, so this area is often useable.

Extended Memory

Memory above 1 Mb is known as **extended memory**, and is not normally useable under DOS, except to provide RAM disks or caches, because DOS runs in *real mode*, and it can't access extended memory in protected mode which OS/2 and Windows 95 do. Some programs are able to switch the CPU from one to the other by using the *DOS Protected Mode Interface* (DPMI). Although extended memory first appeared on the 286, and some software was written to take advantage of it, the 286 was used mostly as a fast XT, because DOS wasn't rewritten. It wasn't until the 386, with its memory paging capability, that extended

memory came to be used properly.

High Memory

The first 64K (less 16 bytes) of extended memory, which is useable only by 286, 386 or 486 based computers that have more than 1Mb of memory. It is the result of having more than 20 address linees that can be exploited by DOS to use that portion of extended memory as if it were below 1Mb, leaving yet more available for programs in base memory. In other words, it is extended memory that can be accessed in real mode. It is activated with **himem.sys**.

HMA access is possible because of the **segment:offset** addressing scheme of the PC. Memory addresses on a PC are 20 bits long, and are calculated by shifting the contents of a 16-bit register 4 bits to the left, and adding it to a 16-bit offset. The 8088, with only 20 address lines, cannot handle the address carry bit, so the processor simply wraps around to address 0000:0000 after FFFF:000F; in other words, the upper 4 bits are discarded.

On a 286 or later, the is a 21st memory address line that was left open by accident and which can be operated by software, which gives you a dirty bit. If the system activates this bit while in 8088 (real) mode, the wraparound doesn't happen, and the high memory area becomes available. IBM has lead the A20 line through a switch in the keyboard controller to (de)activate this address line.

Expanded Memory

This is the most confusing one of all, because it sounds so much like expansion memory, which is what extended memory is sometimes called. Once the PC was on the market, it wasn't long before 640K wasn't enough, particularly for people using Lotus, the top-selling application of the time, who were creating large spreadsheets and not having enough memory to load them, especially when version 2 needed 60K more memory than the original. It wasn't entirely their fault; Lotus itself in its early days was very inefficient in its use of memory anyway.

Users got onto Lotus, Intel and Microsoft for a workaround, and they came up with LIM memory, also known as **Expanded**. It's a system of physical bank-switching, were several extra banks of memory can be allocated to a program, but only one will be in the address space of the CPU at any time, as that bank switched, or **paged**, as required. In other words, the program code stays in the physical cells, but the electronic address of those cells is changed, either by software or circuitry.

In effect, LIM (4.0) directly swaps the contents of any 16K block of expanded memory with a similar one inside upper memory; no swapping takes place, but the pages have their address changed to look as if it does. Once the page frame is mapped to a page on the card, the data of that page can be seen by the CPU. Points to note about LIM:

- It's normally only available for data (no program code).
- Programs need to be specially written to use it.

In theory, LIM 4 doesn't need a page frame, the programs you run may well expect to see one. In

addition, there could be up to 64 pages, so you could bank switch up to a megabyte at a time, effectively doubling the address space of the CPU, and enabling program code to be run and multitasking. This was called *large-frame EMS*, but it still used only four pages in upper memory; the idea was to remove most of the memory on the motherboard. The memory card *backfilled* conventional memory and used the extra pages for banking.

On an 8086 or 286-based machine, expanded memory is usually provided by circuitry on an expansion card, but there are some software solutions. 386 (and 486) -based machines have memory management built in to the central processor, so all that's needed is the relevant software to emulate LIM **(emm386.exe** or similar).

Virtual Memory

"Virtual" in the computer industry is a word meaning that something is other than what it appears to be. Many people have difficulties to understand what virtual memory actually means. Virtual memory is memory that does not exist. Several contradictory definitions about virtual memory exist. For some, virtual memory is not disk space. Disk (swap) space is used for backing allocated (committed) memory (global data, heap and stack) when the OS ran out of system memory (RAM). Example: You create a program and define a stack of 64KB. Because your program really doesn't require 64KB (only 2KB), the OS will allocate only one page (4K) during run-time. The other 64KB - 4KB = 60KB is virtual memory; memory that does not exist, not in system memory, not on disk. Only when your program runs out of stack, another page will be allocated unless a total of 64KB is used already.

From another point of view, Virtual Memory isn't memory at all, but hard disk space made to look like it; the opposite of a RAM disk. Windows (and System 7) uses virtual memory for *swap files*, used when physical memory runs out (you need protected mode on a PC to do that). Like disk cacheing, VM was used on mainframes for some time before migrating to the PC; VMS, the OS used on DEC VAXes, actually stands for *Virtual Memory System*. There is a speed penalty, of course, as you have to access the hard disk to use it, but Virtual Memory is a good stopgap when you're running short.

Bus Types

The purpose of an expansion bus is to provide a way for users to **add hardware devices** (slot-in cards) to a PC using standardised connectors. The advances of technology, especially advancements in developing extremely fast micro- processors, has given birth to more computer bus options than ever before. The blame for not attaining high throughput levels **no longer lies with the processor**, but rather with **Input/Output devices** such as hard disks, LAN cards and video cards. It is for this reason that manufacturers are devising better and more efficient ways of interfacing the expansion slots to the processor itself.

The expansion bus (where expansion cards go) is an extension of the Central Processor, so when adding cards to it, you are extending the capabilities of the CPU itself. The relevance of this regard to the BIOS is that older cards are less able to cope with modern buses running at higher speeds than the original

design of 8 or so MHz. Also, when the bus is accessed, the **whole computer** slows down to the bus speed, so it's often worth altering the speed of the bus or the wait states between it and the CPU to speed things up. The PC actually has four buses; the **processor bus** connects the CPU to its support chips, the **memory bus** connects it to its memory, the **address bus** is part of both of them, and the **I/O (or expansion) bus** is what concerns us here.

Background

Before the days of the original IBM PC, there were no standards. The first agreed upon bus specification which was drawn up was called the S-100. Steven Jobs and Steven Wosniak, the originators of Apple computers, are generally credited as been the first to implement expansion slots to allow the possibility of adding slot-in cards. This caught the attention of IBM and produced the first IBM PC with expansion slots using the 8088 16 bit CPU. The 8088 had an external bus of 8 bits running at 4.77 MHz. The expansion slots were connected directly to the CPU and could therefore run at the full CPU clock speed of 4.77 MHz.

IBM then produced the AT computer which had an external 16 bit bus with a bus speed of 6 MHz and later 8 MHz. The expansion slots were made to cope with the 16 bit bus. In order to maintain compatibility with existing 8 bit cards, the 16 bit slot was made to be backwards compatible. IBM clone manufacturers then started pushing the CPU speeds up and kept the expansion slots at the same speed as the CPU. At a bus speed of 12 MHz, it was found that expansion cards no longer functioned correctly. Consensus was reached within the industry to keep expansion slots at a speed of 8 MHz irrespective of the CPU speed - the Industry Standard Architecture (ISA) was born. It was at this time when expansion slots were no longer directly connected to the CPU.

The industry was at peace until Intel produced the 80386 processor with a 32 bit bus. IBM also announced its Micro Channel Architecture for 32 bit bus data transfer running at 10 MHz. In response to clone manufacturers, the Extended Industry Standard Architecture (EISA) was born in 1988 with a bus speed of 8.33 MHz. EISA and MCA was considered too expensive for the average user so ISA still remained as an alternative.

The world is now sitting with 32 bit architecture with video graphics cards and hard disk controller cards plugged into ISA expansion slots trying to transfer many megabytes of data per second down a 16 bit bus running at 8 MHz. This can be compared to a four lane high speed highway suddenly coming down to two lanes and still trying to cater for the same high volume of traffic.

ISA

Industry Standard Architecture. The 8-bit version cane on the original PC and the AT, but the latter uses an extension to make it 16-bit. It has a *maximum* data transfer rate of about 8 megabits per second on an AT, which is actually well above the capability of disk drives, or most network and video cards. The *average* data throughput is around a quarter of that. Its design makes it difficult to mix 8- and 16-bit RAM or ROM within the same 128K block of upper memory; an 8-bit VGA card could force all other cards in the same (C000-DFFF) range to use 8 bits as well, which was a common source of inexplicable crashes where 16-bit network card were involved.

System Attributes	PC or XT	AT Classic Bus
Processor	8088/8086	286 and higher
CPU Modes	Real	Real/Protected
Expansion Slot	8 bit	16 bit
Slot Type	ISA	ISA
Interrupts	8 + NMI	16 + NMI
DMA Channels	4	8
Maximum RAM	1 MB	16 MB
Floppy Controller	360K/720K	1.2M/1.44M

EISA

Extended Industry Standard Architecture. An evolution of ISA and (theoretically) backward compatible with it, including the speed (8 MHz), so the increased data throughput is mainly due to the bus doubling in size-but you must use EISA expansion cards. It has its own DMA arrangements, which can use the complete address space. On advantage of EISA (and Micro Channel) is the ease of setting up expansion cards-plus them in and run the configuration software which will automatically detect their settings.

System Attributes	EISA	ISA
Memory Capacity	4 GB	16 MB
Data bus width	32 bit	16 bit
DMA Range	4 GB	16 MB
DMA data path	8/16/32	8/16
Maximum DMA Transfer	33 MB/sec	2 MB/sec

Bus Arbitration	6 Bus Masters	Single
Bus Master data path	16/32	16
Synchronous clock rate	8.33 MHz	8.33 MHz

MCA

Micro Channel Architecture. A proprietary standard established by IBM in 1987 to take over from ISA, and therefore incompatible with anything else. It comes in two versions, 16- and 32-bit and, in practical terms, is capable of transferring around 20 MBps.

It was found that poorly designed ISA bus machines did have fairly high radio frequency emissions. Good ISA bus designs complied with the basic FCC Class A regulations. MCA, with its very strict FCC Class B certification, has a distinct advantage over ISA and EISA as CPU clock speeds go higher and higher. MCA is very well shielded which makes it immune to electrical noise.

MCA is designed to eliminate the hassle associated with setting jumpers and DIP switches on adapter boards. The adapter card booklet describing all possible settings gets invariably lost, making the card unusable if the card's environment changes. IBM's answer is called Programmable Option Selection (POS) which is built into MCA computers. The POS uses a special file called an Adapter Description File (ADF) which is supplied with each adapter board. This ADF contains all the possible setting and configurations for the specific card. The POS will also take care of any conflicting settings.

MCA also corrects any potential timing problems which may exist between adapter boards or even memory. Each expansion slot is supposed to carry the same signals as any other slot but this might not always happen. Effects on the bus, such as capacitance and signal propagation delays, can cause timing windows to appear differently in different slots. These effects cause unexplained problems to occur mostly in a random fashion. MCA has the ability to wait for a response (automatic wait states) until the adapter is ready to continue. This can slow the process down but is considered more important than the "ghost" symptoms found on ISA.

VL Bus

The Video Electronics Standards Association (VESA) formed a committee in June 1992 to study existing and certainly emerging, local bus video systems in terms of connector layout, signal and data structures. In September 1992 they finalised the VESA Local Bus connector which connects devices directly to the local/host bus. There's also an 'Opti local bus connector'. This connector extends the EISA bus where the VLB connector extends the ISA bus. The local bus is one more directly suited to the CPU; it's next door (hence local), has the same bandwidth and runs at the same speed, so the bottleneck is less (ISA was local in the early days). Data is therefore moved along the bus at processor speeds. **VL-BUS**, a 32-bit bus which allows bus mastering, and uses two cycles to transfers a 32-bit word, peaking at 66 Mb/sec. It also supports burst mode, where a single address cycle precedes four data cycles, meaning that 4 32-bit words can move in only 5 cycles, as opposed to 8, giving 105 Mb/sec at 33 MHz. The speed is mainly obtained by allowing VL-Bus adapter cards first choice at intercepting CPU cycles. It's not designed to cope with more than a certain number of cards at particular speeds; e.g. 3 at 33, 2 at 40 and only 1 at 50 MHz, and

even that often needs a wait state inserted. VL-Bus 2 is 64-bit, yielding 320 Mb/sec at 50 MHz. There are two types of slot; Master and Slave. Master boards (e.g. SCSI controllers) have their own CPUs which can do their own things; slaves (i.e. video cards) don't. A slave board will work on a master slot, but not vice versa.

PCI

PCI, which is a mezzanine bus, divorced from the CPU, giving it some independence and the ability to cope with more devices, so it's more suited to cross-platform work. It is time multiplexed, meaning that address and data lines share connections. It has its own burst mode that allows 1 address cycle to be followed by as many data cycles as system overheads allow. At nearly 1 word per cycle, the potential is 264 Mb/sec. It can operate up to 33 MHz, or 66 MHz with PCI 2.1 and can transfer data at 32 bits per clock cycle so you can get up to 132 Mbytes/sec (264 with 2.1). The PCI 2.1 specs include a 64-bit 66MHz PCI bus. Each PCI card can perform up to 8 functions, and you can have more than one busmastering card on the bus. It should be noted, though, that many functions are not available with PCI, such as sound. Not yet, anyway. It is part of the *plug and play* standard, assuming your operating system and BIOS agree, so it is auto configuring (although some cards use jumpers instead of storing information in a chip); it will also share interrupts under the same circumstances. The PCI chipset handles transactions between cards and the rest of the system, and allows other buses to be bridged to it (typically and ISA bus to allow older cards to be used). Not all of them are equal, though; certain features, such as *byte merging*, may be absent. The connector may vary according to the voltage the card uses (3.3 or 5v; some cards can cope with both)

Basic Optimization Tricks

This section is intended for users who have a limited knowledge of BIOS setup to safely alter their settings. It provides a set of fundamental procedures that may help improve the performance of your system. Also, have a look at the important BIOS settings <importst.htm> section.

· Make sure that all **standard settings** correspond to the installed components of your system. For instance, you should verify the date, the time, available memory, hard disks and floppy disks. For more information, go to the Standard CMOS Setup <STANDARD.htm> section.

· Make sure that your **cache memory** (internal and external) is enabled. Of course you must have internal (L1) and external (L2) cache memory present which is always the case for recent systems (less than five years old). For more information, go to the Advanced CMOS Setup <advcmos.htm> section. Recently, some motherboards were found having **fake cache memory**. Some unscrupulous manufacturers are using solid plastics chips containing no memory to lure vendors and customers and then gain extra profits in an highly competitive semiconductors market. Beware! For more information about this scam (mostly occurring in England), please consult this Web site: ~http://www.dfw.net/~sdw/bios.html <http://www.dfw.net/sdw/bios.html>.

· Make sure that your **Wait States** values are at the minimum possible. You must however be careful because if values are too low, your system may freeze (hang up). For more information (notably, to value to choose depending on your memory speed) go to the Advanced Chipset Setup <advchip.htm> section.

· Make sure to **shadow** your Video and System ROM. On older systems, this may improve performance significantly, while on newer it may not make much difference. For more information, go to the Advanced CMOS Setup <advcmos.htm> section.

· Make sure to use a coherent **power management** strategy. Choosing the right timing may increase the life expectancy of your hard disk. See the Power Management <power.htm> section.

· **Hard disk speed is the major bottleneck** for a system performance, notably for those with 16 MB of memory and more. You may have the fastest CPU, lots of memory and a confortable cache, but if you have a crummy hard disk, you may not see improvement in performances.

· There is an utility called CPUBOOST <ftp://ftp.gsilink.com/pub/programs/dos/utils/cpuboost.zip>, which claims to improve performance up to 80%. It was put to our attention by Scoot Wainner who has tested it. You may give it a try, but at your own risks.

Important BIOS Settings

This is a short list of settings important for the appropriate configuration of your system. A long list is available in the index section. **DO NOT** mess with these settings unless you have **read carefully their implications**. Also, please have a look at the basic optimization tricks <optimiz.htm> on how to coordinate some of these settings.

Standard CMOS Setup

· Date and Time <standard.htm>

· Primary Display <standard.htm>

· Keyboard <standard.htm>

· Hard Disk C Type <standard.htm>

· Floppy Drive A <standard.htm>

Advanced CMOS Setup

· Above 1 MB Memory Test <advcmos.htm>

· Memory parity error check <advcmos.htm>

· Numeric Processor Test <advcmos.htm>

· System Boot Sequence <advcmos.htm>

- External Cache Memory <advcmos.htm>
- Internal Cache Memory <advcmos.htm>
- Password Checking Option <advcmos.htm>
- Video ROM Shadow C000,16K <advcmos.htm>
- System ROM Shadow F000,64K <advcmos.htm>

Advanced Chipset Setup

- AT BUS Clock Selection <advchip.htm>
- Memory Read Wait State <advchip.htm>
- Memory Write Wait State <advchip.htm>
- Cache Read Option <advchip.htm>
- IDE Multi Block Mode <advchip.htm>

Plug and Play/PCI

- Latency Timer (PCI Clocks) <pcipnp.htm>
- Slot X Using INT# <pcipnp.htm>
- Xth Available IRQ <pcipnp.htm>
- PCI IDE IRQ Map to <pcipnp.htm>
- AT bus clock frequency <pcipnp.htm>
- PCI Clock Frequency <pcipnp.htm>
- ISA Bus Clock Frequency <pcipnp.htm>

Changing Your Password

Enable you to change the active password. The default is no password.

Remember your password!!! Write it down somewhere!!! Ask yourself: Do I really need to set a password to access my system and/or the BIOS? (is your brother / sister / kid / employee / colleague that dangerous?) If security is of some minor concern to you, **disabled recommended**. Why not only password protect (or encrypt) some critical files (personal finances - things the IRS should not see, juicy

love letters, pornographic images (the thing that Internet is most used for), customer information databases, etc...)? If you **lose your password**, you will have to **erase your CMOS memory** (see the FAQ <biosfaq.htm>). Some systems allow you to choose when the password is needed to change the CMOS settings, to boot the machine, etc.

Exiting BIOS

There are two ways to exit BIOS settings.

· **Write to CMOS and Exit**: Save the changes you made in the CMOS. You must do that to permanently keep your configuration. Several users say they changed the CMOS setup but forgot to exit with this one! A common source of error.

· **Do Not Write to CMOS and Exit**: If you are not sure of the changes you made in the CMOS settings, use this option to exit safely.

Advanced Topics for Experienced Users

Advanced CMOS Setup

Remember…

May vary according to your system, BIOS version and brand. Some functions may **not be present or the order and name may be different** (particularly for different BIOS brand). Know **EXACTLY** what you are doing. Some configurations may keep your computer **off from booting**. If that's the case: Switch the power off. Turn your computer on **WHILE** keeping the **DEL** key pressed. This is supposed to erase the BIOS memory. If it still doesn't boot, consult your motherboard manual. Look for a "forget CMOS RAM" jumper. Set it. Try it again. If it still doesn't boot, ask a friend or post to a computer hardware newsgroup. You are permitted to panic.

· **Typematic Rate Programming**: Disabled recommended. It enables the typematic rate programming of the keyboard. Not all keyboards support this! The following two entries specify how the keyboard is programmed if enabled.

· **Typematic Rate Delay (msec)**: 500 ns recommended. The initial delay before key auto-repeat starts, that is how long you've got to press a key before it starts repeating.

· **Typematic Rate (Chars/Sec)**: 15. It is the frequency of the auto-repeat i.e. how fast a key repeats.

· **Above 1 MB Memory Test**: If you want the system to check the memory above 1 MB for errors. **Disabled recommended** for faster boot sequence. The HIMEM.SYS driver for DOS 6.2 verifies the XMS (Extended Memory Specification), so this test is redundant. It is thus preferable to use the XMS test provided by HIMEM.SYS since it is operating in the real environment (where user wait states and other are operational).

· **Memory Test Tick Sound**: Enabled recommended. It gives an audio record that the boot sequence is working properly. Plus, it is an aural confirmation of your CPU clock speed/Turbo switch setting. An experimented user can hear if something is wrong with the system just be the memory test tick sound. Since systems have now much more memory than before, this setting is not common anymore.

· **Memory Parity Error Check**: **Enabled recommended**. Additional feature to test bit errors in the memory. All (or almost all) PCs are checking their memory during operation.Every byte in memory has another ninth bit, that with every write access is set in such way that the parity of all bytes is odd. With every read access the parity of a byte is checked for this odd parity. If a parity error occurs, the NMI (Non Maskable Interrupt), an interrupt you mostly cannot switch off, so the computer stops his work and displays a RAM failure) becomes active and forces the CPU to enter an interrupt handler, mostly writing something like this on the screen: PARITY ERROR AT 0AB5:00BE SYSTEM HALTED. On some motherboards you can disable parity checking with standard memory. Enabled to be sure data from memory are correct. Disable only if you have 8-bit RAM, which some vendors use because it is 10% cheaper. Also, this setting is no longer necessary on recent computers since the quality and reliance of memory chips has greatly been improved.

About different memory speeds: Be sure to have memory chips of the same speed installed. It is not uncommon to have system crashes simply because memory SIMMS are of different speed. Faster memory may not adapt itself to the speed of slower memory. 60 ns and 80 ns SIMMS will surely make your system crash and yourself wonder what is the problem (I know).

· **Hard Disk Type 47 RAM Area**: The BIOS has to place the HD type 47 data somewhere in memory. You can choose between DOS memory or PC BIOS (or peripheral card) memory area 0:300. DOS memory is valuable, you only have 640KB of it. So you should try to use 0:300 memory area instead. There may be some peripheral card which needs this area too (sound card, network card, whatever). So if there are some fancy cards in your PC, check the manuals if they're using the 0:300 area. But in most cases this will work without checking. This is redundant if BIOS is shadowed (maybe not in very old BIOSes). The RAM area can be verified by checking address of int41h and int46h. These are fixed disk parameters blocks. If they point to the BIOS area, BIOS made modification of parameters before mapping RAM there.

· If Any Error"**Wait for <F1> If Any Error**: When the boot sequence encounter an error it asks you to press F1. Only at 'non-fatal' errors. If disabled, the system prints a warning and continues to boot without waiting for you to press any keys. Enabled recommended. Disabled if you want the system to operate as a server without a keyboard.

· **System Boot Up Num Lock**: Specify if you want the Num Lock key to be activated at boot up. Some like it, some do not. MS-DOS (starting with 6.0, maybe earlier) allows a "NUMLOCK=" directive in config.sys, too; if someone turns the BIOS flag off but has NUMLOCK=ON in their configuration file, they may be a bit perturbed.

· **Numeric Processor Test**: Enabled if you have a math coprocessor (built in for the 486DX, 486DX2, 486DX3 and Pentium - 586 - family). Disabled if you don't (386SX, 386DX, 486SX, 486SLC and 486DLC). If disabled, your FPU (Floating Point Unit, if present) isn't recognized as present by the system and will therefore significantly decrease the performance of your system.

· **Weitek Coprocessor**: If you have Weitek FPU, enable. If you have not, disable. This high performance FPU has 2-3 times the performance of the Intel FPU. Weitek uses some RAM address space, so memory from this region must be remapped somewhere else. This setting is normally found on 386 motherboards.

· **Floppy Drive Seek at Boot**: Power up your A: floppy drive at boot. **Disabled recommended for faster boot sequence and for reduced damage to heads**. Disabling the floppy drive, changing the system boot sequence and setting a BIOS password are good techniques for adding some security to a PC.

· **System Boot Sequence**: What drive the system checks first for an operating system. **C:, A: recommended for faster boot sequence**, or to not allow any user to enter your system by booting from the FDD if your autoexec.bat starts with a login procedure. A:, C: if the person who uses the computer is someone who don't knows how to setup CMOS. Because if something fails and a boot floppy won't work, many users won't know what to do next. However, be careful. You had better know this setting is turned on and be prepared to turn it off if your hard disk boot track becomes corrupted, but not obviously absent, since you otherwise won't be able to boot from floppy. Also, it's easy to fool yourself into thinking you booted from a known virus-free floppy when it actually booted from the (virus-infested) hard drive.

· **System Boot Up CPU speed**: Specify at what processor speed the system will boot from. Usual settings are HIGH and LOW. **HIGH recommended**. If you encounter booting problems, you may try LOW. You may also change the CPU speed with Ctrl-Alt +.

· **External Cache Memory**: Enabled if you have external cache memory (**better known as L2 cache memory**). This is a frequent error in CMOS setup as if Disabled when you have cache memory, the system performance decreases significantly. Most systems have from 64K to 512K of external cache. It is a cache between the CPU and the system bus. Different operating systems may address different levels of cache memory. For instance, DOS and Windows can address up to 64K at one time while Windows 95, OS/2 and Windows NT can address larger memory spaces. So, don't buy 256K of cache is you are using a DOS environment with less than 8MB of memory. It will not improve much the performance of your system. If Enabled when the system does not have cache memory, the system will **freeze** most of the time.

· **Internal Cache Memory**: Enable or disable the internal cache memory of the CPU (**better known as L1 cache memory**). Disabled for 386 and Enabled for 486 (1 to 8KB of internal CPU cache). If the CPU does not have internal cache, the system may freeze if enabled.

In many AMI and AWARD BIOSes, the two previous options are implemented either as separate Internal and External Enable/Disable options, or as a single option (**Cache Memory : Disabled/Internal/Both**).

· **CPU Internal Cache:** same as above.

· **Fast Gate A20 Option**: Enabled recommended. A20 refers to the first 64KB of extended memory (A0 to A19) known as the "high memory area". This option uses the fast gate A20 line, supported in some chipsets, to access memory above 1 MB. Normally all RAM access above 1 MB is handled through the keyboard controller chip (8042 or 8742). Using this option will make the access faster than the normal method. This option is very useful in networking and multitasking operating systems.

· **Turbo Switch Function**: Enables or disables the turbo switch. Disabled recommended. This setting is now removed since there are no need to switch from normal to turbo modes.

· **Shadow Memory Cacheable**: You increase speed by copying ROM to RAM. Do you want to increase it by cacheing it? Yes or no - see Video BIOS Area cacheable. Yes recommended for MS-DOS and OS/2. Linux and other Unix-like operating systems will not use the cached ROMs and will benefit from the additional available memory if they are not cached.

· **Password Checking Option**: Setup password to have access to the system and / or to the setup menu. Good if the computer is to be shared with several persons and you don't want anyone (friends, sister, etc.) to mess up with the BIOS. Default password: AMI (if you have AMI BIOS). Award: BIOSTAR or AWARD_SW for newer versions (Note: I even know a computer store that kept standard AWARD BIOS configuration with their systems because they didn't know what the default password was!).

· **Video ROM Shadow C000, 32K**: Memory hidden under the "I/O hole" from 0x0A0000 to 0x0FFFFF may be used to "shadow" ROM (Read-Only Memory). Doing so, the contents of the ROM are copied into the RAM and the RAM is used instead, which is obviously faster. Video BIOS is stored in slow EPROM (Erasable Programmable Read-Only Memory) chips (120 to 150ns of access time). Also, ROM is 8 or 16 bit while RAM 32 bit wide access. With Shadow on, the EPROM content is copied to RAM (60 to 80ns of access time with 32 bit wide access). Therefore performance increases significantly. Only sensible on EGA/VGA systems. **Enabled recommended**. If you have flash BIOS (EEPROM), you can disable it. Flash BIOS enables access at speeds similar to memory access so you can use the memory elsewhere. However, flash BIOS is still only accessing it at the speed of the bus (ISA, EISA or VLB). On systems where the BIOS automatically steals 384K of RAM anyway, it shouldn't hurt to enable shadowing even on flash ROM. One side effect is that you will not be able to modify the contents of flash ROM when the chip is shadowed. If you reconfigure an adapter which you think might have flash ROM, and your changes are ignored, or of course if it gives you an error message when you try to change them, you'll need to temporarily disable shadowing that adapter. On (S)VGA you should enable both video shadows. Some video cards maybe using different addresses than C000 and C400. If it is the case, you should use supplied utilities that will shadow the video BIOS, in which case you should disable this setting in the CMOS. Video BIOS shadowing can cause software like XFree86 (the free X Window System) to hang. They should be probably be disabled if you run any of the 386 unixes.

Some cards map BIOS or other memory not only in the usual a0000-fffff address range, but also just below the 16MB border or at other places. The BIOS (for PCI buses only?) now allows to create a hole in the address range where the card sits. The hole may be enabled by giving an address, then a size is requested in power of 2, 64k - 1MB.

· **Adaptor ROM Shadow C800,16K**: Disabled. Those addresses (C800 to EC00) are for special cards, e.g. network and controllers. Enable only if you've got an adapter card with ROM in one of these areas. It is a BAD idea to use shadow RAM for memory areas that aren't really ROM, e.g. network card buffers and other memory-mapped devices. This may interfere with the card's operation. To intelligently set these options you need to know what cards use what addresses. Most secondary display cards (like MDA and Hercules) use the ROM C800 address. Since they are slow, shadowing this address would improve their performance. An advanced tip: in some setups it is possible to enable shadow RAM **without** write-protecting it; with a small driver (UMM) it is then possible to use this 'shadow RAM' as UMB (Upper Memory Block) space. This has speed advantages over UMB space provided by EMM386. Some BIOSes have three options per 16KB/32KB/64KB block; e.g. disable - shadow ROM - shadow RAM or disable - shadow/WP - shadow (WP = write protect) the third option is for upper memory.

· **Adaptor ROM Shadow CC00,16K**: Disabled. Some hard drive adapters use that address.

· **Adaptor ROM Shadow D000,16K**: Disabled. D000 is the default Address for most Network Interface Cards.

· **Adaptor ROM Shadow D400,16K**: Disabled. Some special controllers for four floppy drives have a BIOS ROM at D400..D7FF.

· **Adaptor ROM Shadow D800,16K**: Disabled

· **Adaptor ROM Shadow DC00,16K**: Disabled

· **Adaptor ROM Shadow E000,16K**: Disabled. E000 is a good "out of the way" place to put the EMS page frame. If necessary.

· **Adaptor ROM Shadow E400,16K**: Disabled

· **Adaptor ROM Shadow E800,16K**: Disabled

· **Adaptor ROM Shadow EC00,16K**: Disabled. SCSI controller cards with their own BIOS could be accelerated by using Shadow RAM. Some SCSI controllers do have some RAM areas too, so it depends on the brand.

Some SCSI adapters do not use I/O-Addresses. The BIOS address range contains writable addresses, which in fact are the I/O-ports. This means this address must not be shadowed and even not be cached.

· **System ROM Shadow F000, 64K**: Same thing as Video shadow, but according to the system

BIOS (main computer BIOS). **Enabled recommended for improved performance**. System BIOS shadowing and caching should be disabled to run anything but DOS (Windows).

On older BIOS versions the shadow choices are in 400(hex)-byte increments. For instance, instead of one Video ROM Shadow segment of 32K, you will have two 16K segments (C400 and C800). Same thing for Adaptor ROM Shadow segments.

· **BootSector Virus Protection**: It is **not** exactly a **virus protection**. All it does is whenever your boot sector is accessed for writing, it gives a warning to the screen allowing you to disable the access or to continue. Extremely annoying if you use something like OS/2 Boot Manager that needs to write to it. It is completely useless for SCSI or ESDI (Enhanced Small Device Interface) drives as they use their own BIOS on the controller. **Disabled recommended**. If you want virus protection, use a TSR (Terminate and Stay Resident) virus detection (Norton, Central Point, etc...). Viruscan by Macfee <http://www.mcafee.com/> is also a good idea since it is a shareware.

Advanced Chipset Setup

Remember…

Configurations may vary according to your system, BIOS version and brand. So, some settings may be present on your computer, some may not or have a different name. Be sure of what you are doing! If you find a configuration having a different name, please let us know.

· **Refresh**

· **Data Bus**

· **Cacheing**

· **Memory**

· **Automatic Configuration**: Allows the BIOS to set automatically several important settings (e.g. Clock divider, wait states, etc.). Very useful for newbies. **Disabled recommended if you want to play around with the settings**. If you have some special adapter cards, you will also have to disable this option.

· **Keyboard Reset Control**: Enable Ctrl-Alt-Del warm reboot. Enabled recommended for more control over your system.

Refresh

· **Hidden Refresh**: Allows the RAM refresh memory cycles to take place in memory banks not used by your CPU at this time, instead or together with the normal refresh cycles, which are executed every time a certain interrupt (DRQ0 every 15 ms) is called by a certain timer (OUT1). Every time it takes 2 to 4 ms for the refresh. One refresh cycle every ~16 us refreshes 256 rows in ~ 4ms. Each refresh cycle only takes the equivalent of one memory read or less, as CAS (Column Address Strobe) is not needed for a refresh cycle. Some RAM can do it, some not. Try. If the computer fails, turn it off. Enabled recommended. There are typically 3 types of refresh schemes: cycle steal, cycle stretch, or hidden refresh. Cycle steal actually steals a clock cycle from the CPU to do the refresh. Cycle stretch actually delays a cycle from the processor to do the refresh. Since it only occurs every say 4ms or so, it's an improvement from cycle steal. We're not really stealing a cycle, only stretching one. Hidden refresh typically doesn't stretch or steal anything. It's usually tied to DTACK (Data acknowledge) or ALE (Address Latch Enable) or some other signal relating to memory access. Since memory is accessed ALL of the time it is easy to synchronize the refresh on the falling edge of this event. Of course, the system performance is at its optimum efficiency, refresh wise since we're not taking clock cycles away from the CPU.

· **Slow Refresh**: Causes RAM refresh to happen less often than usual, around four times. This increases the performance slightly due to the reduced contention between the CPU and refresh circuitry, but not all DRAM memories necessarily support these reduced refresh rates (in which case you will get parity errors and crashes). It also saves power, a good opportunity for laptop computers. **Enabled recommended**

· **Concurrent Refresh**: Both the processor and the refresh hardware have access to the memory at the same time. If you switch this off, the processor has to wait until the refresh hardware has finished (it's a lot slower). **Enabled recommended**.

· **Burst Refresh:** Performs several refresh cycles at once. Increase the system performance.

· **DRAM Burst at 4 Refresh:** Refresh is occurring at Bursts of four, increasing the system performance.

· **Hi-speed Refresh:** Refreshes are occurring at an higher frequency, which is improving the system performance. Of course, not all types of memory can support it and Slow Refresh is preferred.

· **Staggered Refresh:** Refresh is performed on memory banks sequentially. The advantages are related to less power consumption and less interference between memory banks.

· **Slow Memory Refresh Divider:** The AT refresh cycle occurs normally every 16 ns, straining the CPU. If you can select an higher value, such as 64 ns, you will increase the performance of your system.

· **Decoupled Refresh Option:** Enables the ISA bus and the RAM to refresh separately. Because refreshing the ISA bus is more slow, this causes less strain on the CPU.

· **Refresh Value**: The lower this value is, the best the performance.

· **Refresh RAS Active Time**: The amount of active time needed for Row Address Strobe during refresh. The lower the better.

Data Bus

· **Single ALE Enable**: Address Latch Enable (ALE) is an ISA Bus Signal (Pin B28) that indicates that a valid address is posted on the bus. The bus is used to communicate with 8 and 16 bit peripheral cards. Some chipsets have the capability to support an enhanced mode in which multiple ALE assertions may be made during a single Bus Cycle. Single ALE Enable apparently enables or disables that capability. May slow the video bus speed if enabled. Disabled (No) recommended.

· **AT BUS Clock Selection (or AT Bus Clock Source)**: Gives a division of the CPU clock (or System Clock) so it can reach the ISA - EISA bus clock. An improper setting may cause significant decrease in performance. The settings are in terms of CLK/x, (or CLKIN/x and CLK2/x) where x may have values like 2, 3, 4, 5, etc. CLK represents your processor speed, with the exception that clock-multiple processors need to use the EXTERNAL clock rate, so a 486DX33, 486DX2/66, and 486DX3/99 all count as 33 and should have a divider value of 4. For 286 and 386 processors, CLK is half the speed of the CPU. You should **try to reach 8.33 Mhz** (that's the old bus clock of IBM AT; there may be cards which could do higher, but it's not highly recommended). On some motherboards, the AT bus speed is 7.15 Mhz. On new BIOS versions, there is an AUTO setting that will look at the clock frequency and determine the proper divider. Here are some appropriate settings:

CLK/3 SX/DX16, DX20, DX25, DX2/50, DX4/100

CLK/4 SX/DX33, DX2/66, DX3/99

CLK/5 DX40, DX2/80

CLK/6 DX50, DX2/100

Selecting the right clock divider. You can try other clock settings to **increase performance**. If you choose a **too small divider** (CLK/2 for a DX33) **your system may hang**. For a **too big divider** (CLK/5 for a DX33) **the performance of ISA cards will decrease**. This setting is for data exchange with ISA cards, **NOT VL bus and PCI cards** which run at CPU bus clock speeds: 25Mhz, 33Mhz and higher. If your ISA cards are fast enough to keep up, it is possible to run the bus at 12 Mhz. Note that if you switch crystals to overclock your CPU, you are also overclocking the ISA bus unless you change settings to compensate. Just because you can overclock the CPU doesn't mean you can get away with overclocking the ISA bus. It might just be one card that causes trouble, but one is enough. It might cause trouble even if you aren't using it by responding when it shouldn't.

· **ISA Bus Speed**: As above, but related to PCI.

· **Bus Mode**: It can be set in synchronous and asynchronous modes. In synchronous mode, the CPU clock is used, while in asynchronous mode the ATCLK is used.

· **AT Cycle Wait State**: Whenever an operation is performed with the AT bus, it indicates the number of wait states inserted. You may need some wait states if old ISA cards are used, notably if they are in operation with fast adapter cards.

· **16-bit Memory, I/O Wait State:** The number of wait states before 16-bit memory and I/O operations.

· **8-bit Memory, I/O Wait State**: As above, except this setting is for 8-bit operations.

· **16-bit I/O Recovery Time**: The additional delay time inserted after every 16-bit operations. This value is added to the minimum delay inserted after every AT cycles.

· **Fast AT Cycle**: If enabled, may speed up transfer rates with ISA cards, notably video.

· **ISA IRQ**: Inform the PCI cards of the IRQs used by ISA cards, so they be discarded.

· **DMA Wait States**: The number of wait states inserted before direct memory access (DMA). The lower the better.

· **DMA Clock Source**: The source of the DMA clock for which some peripheral controllers, like floppy, tape, network and SCSI adapters use to address memory, which is 5 MHz maximum.

· **E0000 ROM belongs to ATBUS**: Tells if the E0000 area (upper memory) belongs to the MB DRAM or to the AT bus. Yes recommended.

· **Memory Remapping**: Remaps the memory used by the BIOS (A0000 to FFFF - 384 k) above the 1 Mb limit. If enabled you cannot shadow Video and System BIOS. Disabled recommended.

· **Fast Decode Enable**: Enabled recommended. Refers to some hardware that monitors the commands sent to the keyboard controller chip. The original AT used special codes not processed by the keyboard itself to control the switching of the 286 processor back from protected mode to real mode. The 286 had no hardware to do this, so they actually have to reset the CPU to switch back. This was not a speedy operation in the original AT, since IBM never expected that an OS might need to jump back and forth between real and protected modes. Clone makers added a few PLD chips to monitor the commands sent to the keyboard controller chip, and when the "reset CPU" code was seen, the PLD chips did an immediate reset, rather than waiting for the keyboard controller chip to poll its input, recognize the reset code, and then shut down the CPU for a short period. This "fast decode" of the keyboard reset command allowed OS/2 and Windows to switch between real and protected mode faster, and gave much better performance. (early 286 clones with Phoenix 286 BIOS had this setting to enable/disable the fast decode logic.) On 386 and newer processors, the fast decode is probably not used, since these CPUs have hardware instructions for switching between modes. There is another possible definition of the "Fast Decode Enable" command. The design of the original AT bus made it very difficult to mix 8-bit and 16-bit RAM or ROM within the same 128K block of high address space. Thus, an 8-bit BIOS ROM on a VGA card forced all other peripherals using the C000-Dfff range to also use 8 bits. By doing an "early decode" of the high address lines along with the 8/16 bit select flag, the I/O bus could then use mixed 8 and 16 bit peripherals. It is possible that on later systems, this BIOS flag controls the "fast decode" on these address lines.

· **Extended I/O Decode**: The normal range of I/O addresses is 0-0x3ff; 10 bits of I/O address space. Extended I/O-decode enables wider I/O-address bus. The CPU support a 64K I/O space, 16 address lines. Most motherboards or I/O adapters can be decoded only by 10 address bits.

· **I/O Recovery Time**: I/O recovery time is the number of wait states to be inserted between two

consecutive I/O operations. It is generally specified as a two number pair -- e.g. 5/3. The first number is the number of wait states to insert on an 8 bit operation, the second the number of waits on a 16 bit operation. A few BIOSes specify an I/O Setup time (AT Bus (I/O) Command Delay). It is specified similarly to IO Recovery Time, but is a delay before STARTING an I/O operation rather than a delay BETWEEN I/O operations. 5/3 has been recommended as a value which will often yield a good combination of performance and reliability. When enabled, more I/O wait states are inserted. A transfer from IDE hard drive to memory happens without any handshaking, meaning the data has to be present (in the cache of the hard disk) when the CPU wants to read them from an I/O Port. This is called **PIO** (Programmed I/O) and works with a REP INSW assembler instruction. Now I/O Recovery Time enabled adds some wait states to this instruction. When disabled, the hard drive is a lot faster. Note that there is a connection between I/O Recovery Time and AT BUS Clock Selection. For example, if the AT BUS Clock is set to 8 MHz and you have a normal hard disk, I/O Recovery Time can be turned off, resulting in a higher transfer rate from hard disk.

· **IDE Multi Block Mode**: Enable IDE drives to transfer several sectors per interrupt. According to the hard drive cache size, six modes are possible. **Mode 0** (standard mode transferring a single sector at a time), **Mode 1** (no interrupts), Mode 2 (Sectors are transferred in a single burst), **Mode 3** (32-bit instructions with speeds up to 11.1 Mb/sec.In BIOSes usually abbreviated as "32-bit mode". Not to be confused with 32-bit protected modeinstructions(!) or Windows' 32-bit disk access.), **Mode 4** (up to 16,7 Mb/sec.) and **Mode 5** (up to 20 Mb/sec.). The so-called "PIO mode 5" is completely bogus. It was launched by some controller manufacturers but was never accepted, never absorbed into the standards and you will not find any disk drives supporting it. Nor will you find any such drives in the future. The relevant parameter for block mode is the number of sectors per interrupt. The maximum number of sectors per interrupt is often (but not always) related to the drive's buffer size. If this setting is not set properly, communication with COM ports may not work properly. If the block size (sectors/interrupt) is set to too large a value, you may experience serial port overruns and CRC errors. To fix this, decrease the block size (preferred) or disable block mode altogether.For more info, please have a look at The EIDE FAQ-ATA harddisks <http://thef-nym.sci.kun.nl/cgi-pieterh/atazip/atafq-10.html>.

· **IDE DMA Transfer Mode**: Settings are Disabled, Type B (for EISA) and Standard (for PCI). Standard is the fastest but may cause problems with IDE CD ROMs. The standard type is type F. Note that both are so-called "third party DMA" and should not be confused with first-party (busmastering) DMA offered by many modern boards.

· **IDE Multiple Sector Mode**: When IDE DMA Transfer Mode is enabled, this sets the number of sectors per burst, with a maximum of 64. Problems may occur with COM ports.

· **IDE Block Mode**: Enables multi-sectors transfers. Also known as **IDE HDD Block Mode**.

Warning. This setting is known to cause crashes in Win95. Disabled recommended. Extremely annoying.

· **IDE 32-bit Transfer**: When enabled, the read / write rate of the hard disk is faster. When disabled only 16-bit data transfers is possible. The read/write rate of the harddisk stays the same, but the transfers over the host bus are **maybe** faster. So, don't expect anything really dramatic. Actually, you should ordinarily expect no difference at all, since even with 16-bit transfers, the local bus is fast enough to accomodate just about any disk drive. However, some interface hardware uses faster timing on the ATA

(IDE) bus when 32-bit transfers are used. In those cases you may notice a speedup. Note that ATA (IDE) is a 16-bit bus. The 32-bit transfers referred to here are strictly the transfers between CPU and interface chip.

· **Extended DMA Registers**: Within a AT, DMA occurs for 16 Mb. When enabled, DMA covers the whole 4 Gb of a 32-bit processor.

Cacheing

· **Cache Read Option**: Often referred as **SRAM Read wait state** or **Cache Read Hit Burst** (SRAM: Static Random Access Memory). A specification of the number of clocks needed to load four 32-bit words into a CPU internal cache. Typically specified as clocks per word. 2-1-1-1 indicates 5 clocks to load the four words and is the theoretical minimum for current high end CPUs (486DX, 486SX, 486DX2, 486DX4, Pentium). Conceptually, the m-n-n-n notation is narrowly limited to CPUs supporting burst mode and with caches organized as 4 word "lines". However it would not be a surprise to see it extended to other CPU architectures. It takes simple integer values, such as 2-1-1-1, 3-1-1-1 or 3-2-2-2. This determines the number of wait states for the cache RAM in normal and burst transfers (the latter for 486 only). The lower you computer can support, the better. 4-1-1-1 is usually recommended.

· **Cache Write Option**: Same thing as memory wait states, but according to cache ram.

· **Fast Cache Read/Write**: Enable if you have two banks of cache, 64K or 256K.

· **Cache Wait State:** Like conventional memory, the lower wait states for your cache, the better. 0 will give the optimal performance, but 1 wait state may be required for bus speed higher than 33 MHz.

· **Tag Ram Includes Dirty**: Enabling will cause an increase in performance, because the cache is not replaced during cycles, simply written over. It will usually cut the maximum cachable range in half, as one bit is taken off the address tag in order to be used as a dirty tag bit. So, if you have a lot of memory, you might be better off without dirty tag bit.

· **Non-Cacheable Block-1 Size**: Disabled. The Non-Cacheable region is intended for a memory-mapped I/O device that isn't supposed to be cached. For example, some video cards can present all video memory at 15 Mb - 16 Mb so software doesn't have to bank-switch. If the non-cacheable region covers actual RAM memory you are using, expect a significant performance decrease for accesses to that area. If the non-cacheable region covers only non-existent memory addresses, don't worry about it. If you don't want to cache some memory you can exclude 2 regions of memory. There are good reasons not to cache some memory areas. For example, if the memory area corresponds to some kind of buffer memory on a card so that the card may alter the contents of this buffer without warning the cache to invalidate the corresponding cache lines. Some BIOSes take more options than enabled /disabled, namely Nonlocal /Noncache /Disabled (VLB only?).

· **Non-Cacheable Block-1 Base**: 0KB. Enter the base address of the area you don't want to cache. It must be a multiple of the Non-Cacheable Block-1 Size selected.

· **Non-Cacheable Block-2 Size**: Disabled.

· **Non-Cacheable Block-2 Base**: 0KB.

· **Cacheable RAM Address Range**: Usually chipsets allow memory to be cached just up to 16 or 32 MB. This is to limit the number of bits of a memory address that need to be saved in the cache together with its contents. If you only have 4MB of RAM, select 4MB here. The lower the better, don't enter 16MB if you only have 8MB installed!

· **Video BIOS Area Cacheable**: To cache or not to cache video BIOS, a good question. You should try what is better - video access is faster with 'enabled', but cache has its size. With an "accelerated" video card it may be necessary to make the video RAM region non-cacheable so the CPU can see any changes the drawing engine makes in the frame buffer.

Memory

· **Memory Read Wait State**: (often referred as **DRAM Wait States**) Each wait states adds 30 ns of RAM access speed. The CPU is often much faster than the memory access time. On a 486, 1 or more wait states are often required for RAM with 80ns or higher access time. And, depending on the processor and motherboard, also for lower than 80ns access time. The less wait states, the better. Consult your manual. If wait states are too low, a parity error will occur. For 386 or 486 non-burst memory access cycle takes 2 clock ticks. A **rough** indication of RAM speed necessary for 0 wait states is 2000/Clock[MHz] - 10 [ns]. For a 33Mhz processor, this would give 50ns of access time required, so if you do not have 50ns memories, wait state is required. The number of wait states necessary is **approximately** (RamSpeed[ns] +10) * Clock[MHz] /1000 - 2. For 70ns RAM and a 33Mhz processor (very standard configuration), this would give roughly 1 wait state. But this really is dependent on chipset, motherboard and cache design, CPU type and whether we talk about reads or writes. Take these formulas with a large grain of salt. You can find out the access time of your RAM chips by looking at their product numbers. Mostly at the end there is a 70, 80, 90, or even 60. If 10 stands there, it means 100 ns. Some RAM chips also have an explicitly written speed in ns. The RAM you buy these days mostly have 70ns or 60ns.

· **Memory Write Wait State**: Same as above except for writting (self evident).

In some BIOSes, these two options are combined as **DRAM Wait State**. In that case, the number of read and write wait states is necessarily equal.

· **DRAM CAS Timing Delay**: The default is no CAS delay. DRAM is organized by rows and columns and accessed through strobes. Then a memory read/write is performed, the CPU activates RAS (Row Access Strobe) to find the row containing the required data. Afterwards, a CAS (Column Access Strobe) specifies the column. RAS and CAS are used to identify a location in a DRAM chip. RAS access is the speed of the chip while CAS is half the speed. When you have slow DRAM, you should use 1 state delay.

· **DRAM Refresh Method**: Selects the timing pulse width of RAS from RAS Only or CAS before RAS (which one is better?).

· **RAS Precharge Time**: Technically, this is the duration of the time interval during which the Row

Address Strobe signal to a DRAM is held low during normal Read and Write Cycles. This is the minimum interval between completing one read or write and starting another from the same (non-page mode) DRAM. Techniques such as memory interleaving, or use of Page Mode DRAM are often used to avoid this delay. Some chipsets require this parameter in order to set up the memory configuration properly. The RAS Precharge value is typically about the same as the RAM Access (data read/write) time. The latter can be used as an estimate if the actual value is unavailable. At least one BIOS describes the precharge and access times as RAS LOW and RAS HIGH Times. For a 33 MHz CPU, 4 is a good choice, while lower values should be selected for slower speeds.

· **RAS Active Time**: The amount of time a RAS can be kept open for multiple accesses. High figures will improve performance.

· **RAS to CAS Delay Time**: Amount of time a CAS is performed after a RAS. The lower the better, but some DRAM will not support low figures.

· **CAS Before RAS**: Reduces refresh cycles and power consumption.

· **CAS Width in Read Cycle**: The number of wait states for the CPU to read DRAM. The lower the better.

· **Interleave Mode**: Controls how the CPU access different DRAM banks.

· **Fast Page Mode DRAM**: This speeds up memory access for DRAM capable of handling it (most do). When access occurs in the same memory area, RAS and CAS are not necessary.

Plug and Play/PCI

A system intended to make fitting of expansion cards easier (yes, really!). In this context, ISA cards are known as Legacy Cards, and are switched as normal to make them fit in. Have as few of these as possible, as accesses to them are slow. With Concurrent PCI, The T II (or 430HX/VX) chipset's Multi Transaction Timer allows multiple transfers in one PCI request, by reducing re-arbitration when several PCI processes can take place at once. Passive Release allows the PCI bus to continue working when it's receiving data from ISA devices, which would normally hog the bus. Delayed Transaction allows PCI bus masters to work by delaying transmissions to ISA cards. Write merging combines byte, word and Dword cycles into a single write to memory.

The idea is that plug and play cards get interrogated by the system they are plugged into, and their requirements checked against those of the cards already in there. The BIOS will feed the data as required to the Operating System, typically Windows '95. Here you will be able to assign IRQs, etc to PCI slots and map PCI INT#s to them. Although Windows '95 or a PnP BIOS can do a lot by themselves, you really need the lot, e.g.a Plug and Play BIOS, with compatible devices and an Operating System for the best performance. Be aware that not all PCI (2.0) cards are PnP. PC (PCMCIA) cards are also "Plug and Play", but are not considered here.

PnP itself was originally devised by Compaq, Intel and Phoenix. Your chipset settings may allow you to choose of two methods of operation:

· All PnP devices are configured and activated.

· All PnP ISA cards are isolated and checked, but only those needed to boot the machine are activated. The ISA system cannot produce specific information about a card, so the BIOS has to isolate each one and give it a temporary handle so its requirements can be read. Resources can be allocated once all cards have been dealt with (recommended for Windows '95, as it can use the Registry and its own procedures to use the same information every time you boot).

ESCD (Extended System Configuration Data), a system which is part of PnP (actually a superset of EISA), that can store data on PnP or non-PnP EISA, ISA or PCI cards to perform the same function as the Windows '95 Registry above, that is, provide consistency between sessions. It occupies part of Upper Memory (E000-EDFF), which is not available to memory managers. The default length is 4K, and problems have been reported with EMS buffer addressing when this area has been used.

PCI Slot Configuration

Although an unlimited number of PCI slots is allowed, in practice 4 is the maximum, due to loading considerations.PCI cards and slots use an internal interrupt system, with each slot being able to activate up to 4, labelled either INT#A-INT#D, or INT#1-INT#4. These are nothing to do with IRQs, although they can be mapped to them if the card concerned needs it. Typically IRQs 9 and 10 are reserved for this, but any available ones can be used.

· **Latency Timer (PCI Clocks)**. Controls the length of time an agent on the PCI bus can hold the bus when another has requested it, so everything gets its fair share.Since the PCI bus runs faster than the ISA bus, the PCI bus must be slowed during interactions with it. This setting allows you to define how long the PCI bus will delay for a transaction between the given PCI slot and the ISA bus. This number is dependent on the PCI master device in use and varies from 0 to 255. AMI defaults to 66, but 40 clocks is a good place to start at 33MHz (Phoenix). The shorter the value, the more rapid access to the bus a device gets, with better response times, but the lower becomes the effective bandwidth and hence data throughput. Normally, leave this alone, but you could set it to a lower value if you have latency sensitive cards (e.g. audio cards and/or network cards with small buffers). Increase slightly if I/O sensitive applications are being run.

· Using IRQ. Affected by the Trigger method. With PCI, you assign IRQs, etc to a slot, rather than adjusting the card, but only if the card needs an IRQ. There are two methods of IRQ usage, Level or Edge triggered (see Expansion Cards). Most PCI cards use the former, and ISA the latter.

· PCI Slot x INTx. Assigns PCI INT#s to slots 1/2/3 (or whatever). See Slot X using INT#, overleaf.

· Edge/Level Select. Programs PCI IRQs to single-edge or logic level. Level or Edge sensitivity is programmed per controller. Select Edge for PCI IDE.

· PCI Device, Slot 1/2/3. Enables I/O and memory cycle decoding.

· Enable. As slave

· En Master. Enables PCI device as bus master.

· Use Default Latency Timer Value. If yes, you don't need Latency Timer (above).

· **Slot X Using INT#**. Selects an INT# channel for a PCI Slot, and there are four (A, B, C & D) for each one, that is, each PCI bus slot supports interrupts A, B, C and D. #A is allocated automatically, and you would only use #B, #C, etc if the card needs to use more than one (PCI) interrupt service. For example, select #D if your card needs four. Using Auto is simplest. Most graphics cards don't need this.

· **Xth Available IRQ**. Selects (or maps) an IRQ for one of the available INT#s above. There are ten selections (3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15). 1st available IRQ means the BIOS will assign this IRQ to the first PCI slots (order is 1, 2, 3, 4). NA means the IRQ has been assigned to the ISA bus and is therefore not available to a PCI slot.

· **1st-6th Available IRQ**. As above.

· **PCI IRQ Activated by**. The method by which the PCI bus recognises an IRQ request; Level or Edge (see Expansion Cards). Use the default unless advised otherwise by your manufacturer or if you have a PCI device which only recognizes one of them.

· **Configuration Mode**. Sets the method by which information about legacy cards is conveyed to the system.

· Use ICU--the BIOS depends on information provided by Plug and Play software (e.g. Configuration Manager or ISA Configuration Utility). Only set this if you have the utilities concerned.

· Use Setup Utility. The BIOS depends on information provided by you in the following settings. Don't use the above utilities.

· **ISA Shared Memory Size**. Sets a block of system memory which will not be shadowed. Should be disabled, unless you have an ISA card that uses the upper memory area. If you use this setting you will also get the following:

· ISA Shared Memory Base Address. If you choose 64K, you can only choose D000 or below.

· **IRQ 3-IRQ 15**. Used to indicate what IRQs are in use by ISA Legacy cards. If not used, set to Available. Otherwise, set Used by ISA Card, which means that nothing else can use it.

· **PCI IDE Prefetch Buffers**. Disables a set of prefetch buffers in the PCI IDE controller. You may need to do this with an operating system (like NT) that doesn't use the BIOS to access the hard disk and doesn't disable interrupts when completing a programmed I/O operation. Disabling also prevents errors with faulty PCI-IDE interface chips that can corrupt data on the hard disk (with true 32-bit operating systems). Check if you've got a PC-Tech RZ1000 or a CMD PCIO 640, but disabling is done automatically with later boards.

· **PCI IDE 2nd Channel**. Disable this if you're not using the 2nd channel on the PCI IDE card, or you will lose IRQ 15 on the ISA slots.

· **PCI IDE IRQ Map to**. Allows you to configure your system to the type of IDE disk controller; an ISA device is assumed. The more apparent difference is the type of slot being used. However, if you have a PCI IDE controller, this setting allows you to specify which slot has the controller and which PCI INT# (A, B,C or D) is associated with the connected hard drives. Note that this refers to the hard disk rather than individual partitions. Since each IDE controller supports two drives, you can select the INT# for each. Note also that the primary has a lower interrupt than the secondary, as described in Slot x Using

INT#.

· PCI-Auto. If the IDE is detected by the BIOS on one of the PCI slots, then the apropriate INT# channel will be assigned to IRQ 14.

· PCI-Slot X. If the IDE is not detected, youn can manually select the slot.

· Primary IDE INT#, Secondary IDE INT#. Assigns 2 INT channels for primary and secondary channels, if supported.

· ISA. Assigns no IRQs to PCI slots. Use for PCI IDE cards that connect IRQs 14 and 15 directly from an ISA slot using a table from a legacy paddleboard.

· **PCI Bus Parking**. Sort of bus mastering; a device parking on the PCI Bus has full control of the bus for a short time. Improves performance when that device is being used, but excludes others. Try with NICs and Hard Disk Controllers.

· **IDE Buffer for DOS & Win**. For IDE read ahead and posted write buffers, so you can increase throughput to and from IDE devices by buffering reads and writes. Slower IDE devices could end up slower, though.

· **IDE Master (Slave) PIO Mode**. Changes IDE data transfer speed; Mode 0-4, or Auto. PIO means Programmed Input/Output. Rather than have the BIOS issue commands to effect transfers to or from the disk drive, PIO allows the BIOS to tell the controller what it wants, and then lets the controller and the CPU perform the complete task by themselves. Modes 1-4 are available.

· **HCLK PCICLK**. Host CLK vs PCI CLK divider; AUTO, 1-1, 1-1.5.

· **PCI-ISA BCLK Divider**. PCI Bus CLK vs ISA Bus CLK divider; AUTO, PCICLK1/3, PCICLK1/2, PCICLK1/4.

· **CPU to PCI Byte Merge**. See Byte Merging for explanation (below).

· **PCI Write-byte-Merge**. When enabled, this allows data sent from the CPU to the PCI bus to be held in a buffer. The chipset will then write the data in the buffer to the PCI bus when appropriate.

· **CPU-to-PCI Read Buffer**. When enabled, up to four Dwords can be read from the PCI bus without interrupting the CPU. When disabled, a write buffer is not used and the CPU read cycle will not be completed until the PCI bus signals that it is ready to receive the data. The former is best for performance.

· **PCI-to-CPU Write Buffer**. See above.

· **CPU-to-PCI Read-Line**. When On, more time will be allocated for data setup with faster CPUs. This may only be required if you add an Intel OverDrive processor to your system.

· **CPU-to-PCI Read-Burst**. When enabled, the PCI bus will interpret CPU read cycles as the PCI burst protocol, meaning that back-to-back sequential CPU memory read cycles addressed to the PCI will be translated into fast PCI burst memory cycles. Performance is improved, but some non-standard PCI adapters (e.g. VGA) may have problems.

· **PCI to DRAM Buffer**. Improves PCI to DRAM performance by allowing data to be stored if a

destination is busy.Buffers are needed because the PCI bus is divorced from the CPU.

· **Latency for CPU to PCI write**. Delay time before CPU writes data to the PCI bus.

· **PCI Cycle Cache Hit WS**. Similar to above. With the latter, the CPU has less to do, so performance is better.

· Normal--Cache refresh during normal PCI cycles.

· Fast--Cache refresh without PCI cycle for CAS.

· **Use Default Latency Timer Value**. Whether or not the default value for the Latency Timer will be loaded, or the succeeding Latency Timer Value will be used. If Yes is selected (default), no further programming is needed in the Latency Timer Value option (below).

· **Latency Timer Value**. The maximum number of PCI bus clocks that the master may burst. A longer latency time gives the CPU more of a chance to control the bus. See also Latency Timer (PCI Clocks).

· **Latency from ADS# status**. This allows you to configure how long the CPU waits for the Address Data Status (ADS). It determines the CPU to PCI Post write speed. When set to 3T, this is 5T for each double word. With 2T (default), it is 4T per double word. For a Qword PCI memory write, the rate is 7T (2T) or 8T (3T). The default should be OK, but if you add a faster CPU to your system, you may find it necessary to increase it. The choices are:

· 3T--Three CPU clocks

· 2T--Two CPU clocks (Default)

· **PCI Master Latency**. If your PCI Master cards control the bus for too long, there is less time for the CPU to control it. A longer latency time gives the CPU more of a chance. Don't use zero.

· **Max burstable range**. The maximum bursting length for each FRAME# asserting. FRAME# is an electrical signal. Dunno what it does, yet.

· **CPU to PCI burst memory write**. If enabled, back-to-back sequential CPU memory write cycles to PCI are translated to PCI burst memory write cycles. Otherwise, each single write to PCI will have an associated FRAME# sequence. Enabled is best for performance, but some non-standard PCI cards (e.g. VGA) may have problems.

· **Fast Back To Back**. Possibly as above, but working on it!

· **CPU to PCI post memory write**. Enabling allows up to 4 Dwords of data to be posted to PCI. Otherwise, not only is buffering disabled, completion of CPU writes is limited (e.g. CPU write does not complete until the PCI transaction completes). Enabled is best for performance.

· **CPU to PCI Write Buffer**. As above. Buffers are needed because the PCI bus is divorced from the CPU; they improve overall system performance by allowing the processor (or bus master) to do what it needs without writing data to its final destination; the data is temporarily stored in fast buffers.

· **PCI to ISA Write Buffer**. When enabled, the system will temporarily write data to a buffer so the CPU is not interrupted. When disabled, the memory write cycle for the PCI bus will be direct to the

slower ISA bus. The former is best for performance.

· **DMA Line Buffer**. Allows DMA data to be stored in a buffer so PCI bus operations are not interrupted. Disabled means that the line buffer for DMA is in single transaction mode. Enabled allows it to operate in an 8-byte transaction mode for greater efficiency.

· **ISA Master Line Buffer**. ISA master buffers are designed to isolate the slower ISA I/O operations from the PCI bus for better performance. Disabled means the buffer for ISA master transaction is in single mode. Enabled means it is in 8-byte mode, increasing the ISA master's performance.

· **CPU/PCI Post Write Delay**. Delay time before the CPU writes data into the PCI bus.

· **Post Write CAS Active**. Pulse width of CAS# when the PCI master writes to DRAM.

· **PCI master accesses shadow RAM**. Enables the shadowing of a ROM on a PCI master for better performance.

· **Enable Master**. Enables the selected device as a PCI bus master and checks whether the card is so capable.

· **AT bus clock frequency**. AT bus speed in a PCI system. Choose whatever divisor gives you a speed of 6-8.33 MHz, depending on the speed of the PCI bus.

· **ISA Bus Clock Frequency**. As above.

· **Base I/O Address**. The base of the I/O address range from which the PCI device resource requests are satisfied.

· **Base Memory Address**. The base of the 32-bit memory address range from which the PCI device resource requests are satisfied.

· **Parity**. Allows parity checking of PCI devices.

· **ISA Linear Frame Buffer**. Set to the appropriate size if you use an ISA card that features a linear frame buffer (e.g. a second video card for ACAD). The address will be set automatically.

· **ISA VGA Frame Buffer Size**. This is to help you use a VGA frame buffer and 16 Mb of RAM at the same time; the system will allow access to the graphics card through a hole in its own memory map; in other words, accesses made to addresses within this hole will be directed to the ISA bus instead of main memory. Should be set to Disabled, unless you are using an ISA card with more than 64K of memory that needs to be accessed by the CPU, and you are not using the Plug and Play utilities. If you have less than 8 Mb memory, or use MS-DOS, this will be ignored.

· **Residence of VGA Card**. Whether on PCI or VL Bus.

· **ISA LFB Size**. LFB=Linear Frame Buffer. See above.

· **Memory Map Hole; Memory Map Hole Start/End Address**. See ISA VGA Frame Buffer Size. Where the hole starts depends on ISA LFB Size. Sometimes this is informative only. If you can change it, base address should be 16Mb, less buffer size.

· **Memory Hole Size**. Options include 1 Mb, 2 Mb, 4 Mb, 8 Mb, Disabled. These are the amounts

below 1 Mb assigned to the AT Bus, and reserved for ISA cards.

· **Memory Hole Start Address**. To improve performance, certain parts of memory are reserved for ISA cards, which must be mapped into the memory space below 16 MB for DMA reasons. The selections are from 1-15 with each number in Mb. This is irrelevant if the memory hole is disabled (see above).

· **Memory Hole at 15-16M**. See above.

· **Local Memory 15-16M**. To increase performance, you can map slower device memory (e.g. on the ISA bus) into much faster local bus memory. Local memory is set aside and the start point transferred from the device memory to local memory. The default is enabled.

· **15-16M Memory Location**. The area in the memory map allocated for ISA option ROMs. Choices are Local (default) or Non-local.

· **Byte Merging**. This exists where writes to sequential memory addresses are merged into one PCI-to-memory operation, which increases performance for older applications that write to video memory in bytes rather than words--not supported on all PCI video cards. Enable unless you get bad graphics. See also next for a variation.

· **Byte Merge Support**. 8- or 16-bit data en route from the CPU to the PCI bus is held in a buffer where it is accumulated, or merged, into 32-bit data, giving faster performance. In this case, enabling means that CPU-PCI writes are buffered (Award).

· **Multimedia Mode**. Enables or disables palette snooping for multimedia cards.

· **Video Palette Snoop**. Controls how a PCI graphics card can "snoop" write cycles to an ISA video card's colour palette registers. Snooping essentially means interfereing with a device.Only set to Disabled if:

· An ISA card connects to a PCI graphics card through a VESA connector

· The ISA card connects to a colour monitor, and

· The ISA card uses the RAMDAC on the PCI card, and

· Palette Snooping (RAMDAC shadowing) not operative on PCI card.

· **PCI/VGA Palette Snoop**. Alters the VGA palette setting while graphic signals pass through the feature connector of PCI VGA card and are processed by MPEG card. Enable if you have MPEG connections through the VGA feature connector; this means you can adjust PCI/VGA palettes. VGA snooping is used by multimedia video devices (e.g. video capture boards) to look ahead at the video controller (VGA device) to see what color palette is currently in use. It is only in exceptional circumstances that you might ever need to enable this, so disable for ordinary systems. (Award BIOS).

· **Snoop Filter**. Saves the need for multiple enquiries to the same line if it was inquired previously. When enabled, cache snoop filters ensure data integrity (cache coherency) while reducing the snoop frequency to a minimum.

· **E8000 32K Accessible**. The 64K E area of upper memory is used for BIOS purposes on PS/2s, 32

bit operating systems and Plug and Play. This setting allows the second 32K page to be used for other purposes when not needed, in the same way that the first 32K page of the F range is useable after boot up has finished.

· **P5 Piped Address**. Default is Disabled

· **PCI Arbiter Mode**. Devices gain access to the PCI bus through arbitration. There are two modes, 1 (the default) and 2. The idea is to minimize the time it takes to gain control of the bus and move data. Generally, Mode 1 should be sufficient, but try mode 2 if you get problems.

· **Stop CPU When Flush Assert**. See below.

· **Stop CPU when PCI Flush**. When enabled, the CPU will be stopped when the PCI bus is being flushed of data. Disabling (default) allows the CPU to continue processing, giving greater efficiency.

· **Stop CPU at PCI Master**. When enabled, the CPU will be stopped when the PCI bus master is operating on the bus. Disabling (default) allows the CPU to carry on, giving greater efficiency.

· **I/O Cycle Recovery**. When enabled, the PCI will be allowed a recovery period for back-to-back I/O, which slows back-to-back data transfers; it's like adding wait states, so disable (default) for best performance.

· **I/O Recovery Period**. Sets the length of time of the recovery cycle used above. The range is from 0-1.75 microseconds in 0.25 microsecond intervals.

· **Action When W_Buffer Full**. Sets the behaviour of the system when the write buffer is full. By default the system will immediately retry, rather than wait for it to be emptied.

· **Fast Back-to-Back**. When enabled, the PCI bus will interpret CPU read cycles as the PCI burst protocol, meaning that back-to-back sequential CPU memory read cycles addressed to the PCI will be translated into the fast PCI burst memory cycles. Default is enabled.

· **CPU Pipelined Function**. This allows the system controller to signal the CPU for a new memory address, even before all data transfers for the current cycle are complete, resulting in increased throughput. The default is Disabled, that is, pipelining off.

· **Primary Frame Buffer**. When enabled, this allows the system to use unreserved memory as a primary frame buffer. Unlike the VGA frame buffer, this would reduce overall available RAM for applications.

· **M1445RDYJ to CPURDYJ**. Whether the PCI Ready signal is to be synchronized by the CPU clock's ready signal or bypassed (default).

· **VESA Master Cycle ADSJ**. Allows you to increase the length of time the VESA Master has to decode bus commands. Choices are Normal (default) and Long.

· **LDEVJ Check Point Delay**. This allows you to select how much time is allocated for checking bus cycle comands. These commands must be decoded to determine whether a local bus device access signal (LDEVJ) is being sent, or an ISA device is being addressed. Increasing the delay increases stability, especially the VESA sub-system while very slightly degrading the performance of the ISA sub-system. Settings are in terms of the feedback clock rate (FBCLK2) used in the cache/memory control

interface.

1 FBCLK2=One clock

2 FBCLK2=Two clocks (Default)

3 FBCLK2=Three clocks

· **CPU Dynamic-Fast-Cycle**. Gives you faster access to the ISA bus. When the CPU issues a bus cycle, the PCI bus examines the command to determine if a PCI agent claims it. If not, then an ISA bus cycle is initiated. The Dynamic-Fast-Access then allows for faster access to the ISA bus by decreasing the latency (or delay) between the original CPU command and the beginning of the ISA cycle.

· **CPU Memory sample point**. This allows you to select the cycle check point, which is where memory decoding and cache hit/miss checking takes place. Each selection indicates that the check takes place at the end of a CPU cycle, with one wait state indicating more time for checking to take place than zero wait states. A longer check time allows for greater stability at the expense of some speed.

· **LDEV# Check point**. The VESA local device (LDEV#) check point is where the VL-bus device decodes the bus commands and error checks, within the bus cycle itself.

0 Bus cycle point T1 (Default)

1 During the first T2

2 During second T2

3 During third T2

· **Local memory check point**. Allows you to select between two techniques for decoding and error checking local bus writes to DRAM during a memory cycle.

· Slow=Extra wait state; better checking (default).

· Fast=No extra wait state used.

· **FRAMEJ generation**. When the PCI-VL bus bridge is acting as a PCI Master and receiving data from the CPU, a fast CPU-to-PCI buffer will be enabled if this selection is also enabled. Using the buffer allows the CPU to complete a write even though the data has not been delivered to the PCI bus. This reduces the number of CPU cycles involved and speeds overall processing.

· Normal Buffering not employed (Default)

· Fast Buffer used for CPU-to-PCI writes.

· **PCI to CPU Write Pending**. Sets the behaviour of the system when the write buffer is full. By default, the system will immediately retry, but you can set it to wait for the buffer to be emptied before retrying.

· **Delay for SCSI/HDD (Secs)**. The length of time in seconds the BIOS will wait for the SCSI hard disk to be ready for operation. If the hard drive is not ready, the PCI SCSI BIOS might not detect the hard drive correctly. The range is from 0-60 seconds.

· **Master IOCHRDY**. Enabled, allows the system to monitor for a VESA master request to generate an I/O channel ready (IOCHRDY) signal.

· **VGA Type**. This data is used when the video bios is being shadowed. The BIOS uses this information to determine which bus to use. Choices are Standard (default), PCI, ISA/VESA.

· **PCI Mstr Timing Mode**. This system supports two timing modes, 0 (default) and 1.

· **PCI Arbit. Rotate Priority**. Typically, the system manages or arbitrates access to the PCI bus on a first-come-first-served basis. When priority is rotated, once a device gains control of the bus it is assigned the lowest priority and every other device is moved up one in the priority queue.

· **I/O Cycle Post-Write**. When Enabled (default), data being written during an I/O cycle will be buffered for faster performance.

· **PCI Post-Write Fast**. As in the above I/O Cycle Post-Write, enabling this will allow the system to use a fast memory buffer for writes to the PCI bus.

· **CPU Mstr Post-WR Buffer**. When the CPU operates as a bus master for either memory access or I/O, this item controls its use of a high speed posted write buffer. Choices are NA, 1, 2 and 4 (default).

· **CPU Mstr Post-WR Burst Mode**. When the CPU operates as a bus master for either memory access or I/O, this item controls its ability to use a high speed burst mode for posted writes to a buffer.

· **CPU Mstr Fast Interface**. This enables/disables what is known as a fast back-to-back interface when the CPU operates as a bus master. When enabled, consecutive reads/writes are interpreted as the CPU high-performance burst mode.

· **PCI Mstr Post-WR Buffer**. When a PCI device operates as a bus master for either memory access or I/O, this item controls its use of a high speed posted write buffer. Choices are NA, 1, 2 and 4 (default).

· **PCI Mstr Burst Mode**. When a PCI device operates as a bus master for either memory access or I/O, this item controls its ability to use a high speed burst mode for posted writes to a buffer.

· **PCI Mstr Fast Interface**. This enables/disables what is known as a fast back-to-back interface when a PCI device operates as a bus master. When enabled, consecutive reads/writes are interpreted as the PCI high-performance burst mode.

· **CPU Mstr DEVSEL# Time-out**. When the CPU initiates a master cycle using an address (target) which has not been mapped to PCI/VESA or ISA space, the system will monitor the DEVSEL (device select) pin for a period of time to see if any device claims the cycle. This item allows you to determine how long the system will wait before timing-out. Choices are 3 PCICLK, 4 PCICLK, 5 PCICLK and 6 PCICLK (default).

· **PCI Mstr DEVSEL# Time-out**. When a PCI device initiates a master cycle using an address (target) which has not been mapped to PCI/VESA or ISA space, the system will monitor the DEVSEL (device select) pin for a period of time to see if any device claims the cycle. This item allows you to determine how long the system will wait before timing-out. Choices are 3 PCICLK, 4 PCICLK (default), 5 PCICLK and 6 PCICLK.

· **IRQ Line**. If you have installed a device requiring an IRQ service into the given PCI slot, use this

item to inform the PCI bus which IRQ it should initiate. Choices range from IRQ 3 through IRQ 15.

· **Fast Back-to-Back Cycle**. When enabled, the PCI bus will interpret CPU read or write cycles as PCI burst protocol, meaning that back-to-back sequential CPU memory read/write cycles addressed to the PCI will be translated into fast PCI burst memory cycles.

· **State Machines**. The chipset uses four state machines to manage specific CPU and/or PCI operations. Each can be thought of as a highly optimized process center designed to handle specific operations. Generally, each operation involves a master device and the bus it wishes to employ. The four state machines are:

CPU master to CPU bus (CC)

CPU master to PCI bus (CP)

PCI master to PCI bus (PP)

PCI master to CPU bus (PC)

· Each have the following settings:

· Address 0 WS. This refers to the length of time the system will delay while the transaction address is decoded. Enabled=no delay.

· Data Write 0 WS. The length of time the system will delay while data is being written to the target address. When Enabled, there will be no delay.

· Data Read 0 WS. The length of time the system will delay while data is being read from the target address. When Enabled, there will be no delay.

· **On Board PCI/SCSI BIOS**. You would enable this if your system motherboard had a built-in SCSI controller attached to the PCI bus, and you wanted to boot from it.

· **PCI I/O Start Address**. I/O devices make themselves accessible by occupying an address space. This allows you to make additional room for older ISA devices by defining the I/O start address for the PCI devices.

· **Memory Start Address**. This is for devices with their own memory which use part of the CPU's memory address space, allowing you to determine the starting point in memory where PCI device memory will be mapped.

· **VGA 128k Range Attribute**. When enabled, this allows the chipset to apply features like CPU-TO-PCI Byte Merge, CPU-TO-PCI Prefetch to be applied to VGA memory range A0000H-BFFFFH.

· Enabled=VGA receives CPU-TO-PCI functions.

· Disabled=Retain standard VGA interface.

· **CPU-To-PCI Write Posting**. The Orion chipset maintains its own internal read and write buffers which are used to help compensate for the speed differences between the CPU and the PCI bus. When this is Enabled, writes from the CPU to the PCI bus will be buffered. When Disabled (default), the writes

will not be buffered and the CPU will be forced to wait until the write is completed.

· **CPU Read Multiple Prefetch**. A prefetch occurs during a process (e.g. reading from the PCI or memory) when the chipset peeks at the next instruction and actually begins the next read. The Orion chipset has four read lines. A multiple prefetch means the chipset can initiate more than one prefetch during a process. Default is Disabled.

· **CPU Line Read Multiple**. A line read means that the CPU is reading a full cache line. When a cache line is full it holds 32 bytes (eight DWORDS) of data. Because the line is full, the system knows exactly how much data it will be reading and doesn't need to wait for an end-of-data signal, freeing it to do other things. When this is enabled, the system is allowed to read more than one full cache line at a time. The default is disabled.

· **CPU Line Read Prefetch**. See above. When this is enabled, the system is allowed to prefetch the next read instruction and initiate the next process.

· **CPU Line Read**. This Enables or Disables (default) full CPU line reads.

· **CPU Burst Write Assembly**. The (Orion) chipset maintains four posted write buffers. When this is enabled, the chipset can assemble long PCI bursts from the data held in them. Default is Disabled.

· **VGA Performance Mode**. If enabled, the VGA memory range of A 0000-B 0000 will use a special set of performance features. This has little or no effect using video modes beyond the standard VGA most commonly used for Windows, OS/2, UNIX, etc, but this memory range is heavily used by games such as DOOM.

· **Snoop Ahead**. This is only applicable if the cache is enabled. When enabled, PCI bus masters can monitor the VGA palette registers for direct writes and translate them into PCI burst protocol for greater speed, to enhance the performance of multimedia video.

· **DMA Line Buffer Mode**. This allows DMA data to be stored in a buffer so as not to interrupt the PCI bus. When Standard is selected, the line buffer is in single transaction mode. Enhanced allows it to operate in 8-byte transaction mode.

· **Master Arbitration Protocol**. This is the method by which the PCI bus determines which bus master device gains access to it.

· **PCI Clock Frequency**. Allows you to set the clock rate for the PCI bus, which can operate between 0-33 Mhz. CPUCLK/3 means the PCI bus was operating at 11 Mhz (33/3 = 11).

CPUCLK/1.5 CPU speed / 1.5 (Default)

CPUCLK/3 CPU speed/3

14 Mhz 14 Mhz

CPUCLK/2 CPU speed/2

· **Max, Burstable Range**. Sets the size of the maximum range of contiguous memory which can be addressed by a burst from the PCI bus, a half or one K.

· **ISA Bus Clock Frequency**. Allows you to set the speed of the ISA bus in fractions of the PCI bus

speed, so if the PCI bus is operating at its theoretical maximum, 33 Mhz, PCICLK/3 would yield an ISA speed of 11 Mhz.

7.159 Mhz (default)

PCICLK/4 A quarter speed of the PCI bus

PCICLK/3 One third speed of the PCI bus

· **8 Bit I/O Recovery Time**. The recovery time is the length of time, measured in CPU clocks, which the system will delay after the completion of an input/output request to the ISA bus, needed because the CPU is running faster than the bus, and needs to be slowed down. Clock cycles are added to a minimum delay (usually 5) between PCI-originated I/O cycles to the ISA bus. Choices are from 1 to 7 or 8 CPU clocks. 1 is the default.

· **16 Bit I/O Recovery Time**. As above, for 16 bit I/O. Choices are from 1 to 4 CPU clocks. 1 is the default.

· **I/O Recovery Time**. A programmed delay which allows the PCI bus to exchange data with the slower ISA bus without data errors. Settings are in fractions of the PCI BCL:

2 BCLK=Two BCLKS (default)

4 BCLK=Four BCLKS

8 BCLK=Eight BCLKS

12 BCLK=Twelve BCLKS

· **PCI Concurrency**. Enabled (default) means that more than one PCI device can be active at a time (Award). With Intel Chipsets, it allocates memory bus cycles to a PCI controller while an ISA operation, such as bus mastered DMA, is taking place, which normally requires constant attention. This involves turning on additional read and write buffering in the chipset. The PCI bus can also obtain access cycles for small data transfers without the delays caused by renegotiatiating bus access for each part of the transfer, so is meant to improve performance and consistency.

· **PCI Streaming**. Data is typically moved to and from memory and between devices in discrete chunks of limited sizes, because the CPU is involved. On the PCI bus, data can be streamed, that is, much larger chunks can be moved without the CPU being bothered. Enable for best performance.

· **PCI Bursting**. Consecutive writes from CPU will be regarded as a PCI Burst cycle. Enable = best performance; some cards might not like it.

· **PCI (IDE) Bursting**. As above, but this one enables burst mode access to video memory over the PCI bus. The CPU provides the first address, and consecutive data is transferred at one word per clock. The device must support burst mode.

· **Burst Copy-Back Option**. When enabled, if a cache miss occurs, the chipset will initiate a second, burst cache line fill from main memory to the cache, the object being to maintain the status of the cache.

· **Preempt PCI Master Option**. When enabled, PCI bus operations can be preempted by certain

system operations, such as DRAM refresh, etc. Otherwise, they can take place concurrently.

· **IBC DEVSEL# Decoding**. Allows you to set the type of decoding used by the ISA Bridge Controller (IBC) to determine which device to select. The longer the decoding cycle, the better chance the IBC has to correctly decode the commands. Choices are Fast, Medium and Slow (default).

· **Keyboard Controller Clock**. Sets the speed of the keyboard controller (PCICLKI = PCI bus speed).

7.16 Mhz Default

PCICLKI/2 1/2 PCICLKI

PCICLKI/3 1/3 PCICLKI

PCICLKI/4 1/4 PCICLKI

· **CPU Pipeline Function**. This allows the system controller to signal the CPU for a new memory address even before all data transfers for the current cycle are complete, resulting in increased throughput. Enabled means that address pipelining is active.

· **PCI Dynamic Decoding**. When enabled, this setting allows the system to remember the PCI command which has just been requested. If subsequent commands fall within the same address space, the cycle will be automatically interpreted as a PCI command.

· **Master Retry Timer**. This sets how long the CPU master will attempt a PCI cycle before the cycle is unmasked (terminated). The choices are measured in PCICLKs which the PCI timer. Values are 10 (default), 18, 34 or 66 PCICLKs.

· **PCI Pre-Snoop**. Pre-snooping is a technique by which a PCI master can continue to burst to the local memory until a 4K page boundary is reached rather than just a line boundary.

· **CPU/PCI Write Phase**. Determines the turnaround between the address and data phases of the CPU master to PCI slave writes. Choices are 1 LCLK (default) or 0 LCLK.

· **PCI Preempt Timer**. This item sets the length of time before one PCI master preempts another when a service request has been pending.

Disabled No preemption (default).

260 LCLKs Preempt after 260 LCLKs

132 LCLKs Preempt after 132 LCLKs

68 LCLKs Preempt after 68 LCLKs

36 LCLKs Preempt after 36 LCLKs

20 LCLKs Preempt after 20 LCLKs

12 LCLKs Preempt after 12 LCLKs

5 LCLKs Preempt after 5 LCLKs

· **CPU to PCI POST/BURST**. Data from the CPU to the PCI bus can be posted (buffered by the controller) and/or burst. This sets the methods.

· POST/CON.BURST. Posting and bursting supported (default).

· NONE/NONE. Neither supported.

· POST/NONE. Posting but not bursting supported.

· **PCI CLK**. Whether the PCI clock is tightly synchronized with the CPU clock, or is asynchronous. If your CPU, motherboard and PCI bus are running at multiple speeds of each other, e.g. Pentium 120, 60 MHz m/b and 30 MHz PCI bus, choose synchronise.

· **IRQ 15 Routing Selection**. MISA=Multiplexed ISA for asynchronously interrupting the CPU. IRQ 15 is usually used for Secondary IDE channels or CD-ROMs.

· **CPU cycle cache hit same point**. Working on this.

· **PCI cycle cache hit sam point**. As above.

· **Arbiter timer timeout (PCI CLK) 2 x 32**. Working on this.

Hard Disk Utility

· Hard disk format

· Auto detect hard disk

· Auto interleave

· Media analysis

Hard Disk Format

Will format your hard disk so it can receive new partitions.

IT WILL SMASH EVERYTHING ON YOUR HARD DISK!!! USE WITH CAUTION. A lot of inexperienced users have lost their sanity with this one. Several computer stores have made extra money with it! There's no need to do this unless you experience errors or if you want to change the interleave. **DON'T TOUCH THIS IF YOU HAVE AN IDE DRIVE**. It will perform a low level format and probably SCRAP your IDE hard drive. IDE is the standard drive type that nearly everyone has now. SCSI or ESDI drives shouldn't be low-level formatted. The new drives actually don't perform the low level format, but some old AT-Bus (IDE) drives you can scratch with this... This entry is only sensible

for old MFM or RLL hard disks! Please refer to your hard disk manual to see how or if your hard disk can be low level formatted. Don't tell us we did not warn you.

Many manufacturers provide utilities to low level format their IDE drives (or any other types). Please refer to the comp.sys.ibm.pc.hardware.storage <news:comp.sys.ibm.pc.hardware.storage> FAQ for more technical information about this procedure. If normal (high level) hard disk formatting is required, you can use DOS FDISK to first erase and create partitions and then use FORMAT. It is also a good idea when you hard disk becomes inaccessible to see if it is just the system files that are corrupted. Most of the time, it is the case. SYS will do the job of replacing system files. Several packages (PC-Tools, Norton, etc.) provide utilities for repairing "damaged" HDD and FDD. Therefore, low level format is always of LAST RESORT when you encounter HDD problems.

Auto Detect Hard Disk

Handy when you "forgot" the specs of your hard drive. The BIOS will detect the number of cylinders, heads and sectors on your hard disk. In some BIOS versions, this option in the main SETUP menu.

Auto Interleave

Determines the optimum interleave factor for older hard disks. Some controllers are faster than others, and you don't want the sectors laid out so reading consecutive sectors usually results in just missing the sector you wanted and having to wait a whole disk rotation for it to come around again. On modern ones, it's always 1:1 (and even if it wasn't, you cannot reformat anyway).

Interleaving is specified in a ratio, n:1, for small positive integers n. Basically, it means that the next sector on the track is located n positions after the current sector. The idea is that data on a hard drive might spin past the heads faster than the adapter can feed it to the host. If it takes you more than a certain amount of time to read a sector, by the time you're ready for the next sector, the heads will have passed it already. If this is the case, the interleave is said to be "too tight". The converse, where the CPU spends more time than necessary waiting for the next sector to spin under the heads, is too "loose" of an interleave. Clearly, it is better to have too loose an interleave than too tight, but the proper interleave is better still. Especially since any controller with read-ahead cacheing can pull the whole track into its buffer, no matter how slow the CPU is about fetching the data down.

The 1:1 interleave arranges the sectors on a track as follows:

0 1 2 3 4 5 6 7 8 9 a b c d e f g (17-sectors, using base 17 for convenience, this is clearly the in-order arrangement, one after another)

This is 2:1 interleaving:

0 9 1 a 2 b 3 c 4 d 5 e 6 f 7 g 8

The CPU has a whole sector's worth of time to get the a sector's data taken care of before the next sector

arrives. It shows which logical sector goes in each physical sector.

Anyway, an n:1 interleave restricts the transfer rate to 1/n the speed of a 1:1 interleave (which is better than 1 revolution per sector if the interleave is too tight!). No modern PC should require interleaving. Only MFM and RLL (maybe also ESDI) and floppy drives which are capable of it (you could format a 1.44 meg floppy to 21 sectors/track, which would require a 2:1 interleave to not exceed the 500 mbps speed of the controller...but why?).

Media Analysis

Scan the hard disk for bad blocks. It is performing a **LOW LEVEL FORMAT** on the track where bad sector is encountered to mark that sector as a bad. It could cause damage on user data, even if scanning itself is non-destructive (also on MFM, RLL disks). Therefore, **DON'T USE** this option to on AT-Bus (IDE), SCSI or ESDI drives. These drives store the bad block data themselves, so you don't have to tell them or scan the media! Recommendation: use a media analysis program provided by an utility package or your hard drive manufacturer.

Power Management

This menu appears on computers having the "**Green PC**" specification, an initiative of the EPA (Environmental Protection Agency of the United States) with its Energy Star program. The main purpose is to minimize power usage when the system stays inactive for a while. The standard is still not yet achieved among manufacturers, so expect to see several variations. On most cases, the power management strategies are incremental, meaning that the longer a system stays inactive, the more parts will close down.

There exists three power management schemes: **APM** (Advanced Power Management) proposed by Intel and Microsoft. **ATA** (AT Attachment) for IDE drives. **DPMS** (Display Power Management Signaling) matches video monitors and video cards so they may simultaneously shut down.

· **Green Timer of Main Board**: Allows to setup the time after which the CPU of an idle system will shut down. Disabled or a time interval ranging from 1 to 15 minutes are the usual options. 5 to 10 minutes recommended.

· **Doze Timer**: Amount of time before the system will reduce 80% of its activity.

· **Standby Timer**: Amount of time before the system will reduce 92% of its activity.

· **Suspend Timer**: Amount of time after the system goes in the most inactive state possible, which is 99%. After this state, the system will require a warm up period so the CPU, hard disk and monitor may go online.

· **HDD Standby Timer**: Allows to setup the time after which the hard disk of an HHD idle system (no HDD access) will shut down. A terrific option if you have a somewhat noisy hard drive unit. The choice of a time interval depends on how hard disk intensive is your operating system. This may depends also on the amount of memory available. You should setup a longer time interval, like 10 minutes, if you only have 8MB of RAM and running OS/2 or Windows. For a plain/standard DOS environments, 2 to 5 minutes are recommended. If you have a comfortable 16 MB or more, the time lapse can be shorter. There are some reports that this option may cause problems with slave hard drives (AMI BIOS only?).

· **System Slow Down**. After the specified time interval, the CPU will slowed down to 8 MHz.