**Time Stamp Counter**

The time stamp counter (TSC) is a 64 bit register present on all x-86 processors since the Pentium. It counts the number of cycles since reset.

The instruction RDTSC loads the high order 32 bits of the timestamp register into EDX and the low order 32 bits into EAX. A bitwise OR is performed to reconstruct and store the register values in a local variable.

The time stamp counter until recently was an excellent high resolution, low overhead way of getting CPU timing information. With the advent of multi-core / hyper-threaded CPUs, systems with multiple CPUs and hibernating Operating Systems the TSC cannot be relied on to provide accurate results unless we correct the possible flaws.

The possible flaws that need to be corrected are

1. Rate of tick
2. Whether all cores (processors) have identical values in their time keeping registers

There is no promise that the timestamp counters of multiple CPUs on a single motherboard are all synchronized. In such cases programmers can only get reliable results by locking their code to a single CPU. Even then the CPU speed may change due to power saving features taken by the OS or BIOS.

When the system hibernates and resumes the TSC is reset.

Reliance on TSC also reduces portability as other processors may not have this feature

**Constant Rate TSC** – A feature present in certain processors where the TSC reads at the processor's maximum rate regardless of the actual CPU running rate. While this makes time keeping more consistent it skews benchmarks. With these CPUs the cycle counter updates at a fixed frequency independent of the operating frequency of the CPU. Thus constant rate TSC makes cycle counting straight-forward again. To see if this is a feature on your CPU look at the file

cat /proc/cpuinfo

And check if the flag constant\_tsc is present in the flags section

**CLOCK\_MONOTONIC** – Clock that cannot be set and represents the monotonic time since some unspecified starting point.

Out of order instruction execution affects the clock count.

**Determining Cycle Time**

Converting from cycles to wall clock times starts with determining the cycle time (or, equivalently, the frequency of the CPU cycle counter). The OS reports the CPU frequency in various ways, but the most reliable method is simply to measure the cycle time. So how much time is one cycle? You can compute the cycle time by counting the number of cycles in a known time interval. For instance, measuring multiple executions of the following code:

[record start cycle time]

sleep(10)

[record end cycle time]

Based on the results finding the mean gives the number of cycles in the 10 second window. This is sufficient to find the cycle time.

**Counting Cycles**

Fortunately, modern CPUs make it easy to read the CPU cycle counter. On the x86, "rdtsc" and "rdtscp" are the instructions you want to use. You will find many examples and suggestions for using these instructions when searching the Web. However, a good tutorial is an Intel white-paper by Gabriele Paoloni: [How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures](http://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-32-ia-64-benchmark-code-execution-paper.html) (September 2010)

**How to Benchmark code execution times on Intel?**

Intel CPUs have a timestamp counter to keep track of every cycle that occurs on the CPU. Starting with the Intel Pentium processor, the devices have included a per-core timestamp register that stores the value of the timestamp counter and can be accessed via the RDTSC and RDTSCP instructions.

A serializing instruction is an instruction that forces the CPU to complete every preceding instruction of the C Code before continuing the program execution.

**RDTSCP**

The RDTSCP instruction waits until all previous instructions have been executed before reading the counter. However, subsequent instructions may begin execution before the read operation is performed.

**Statistics definition**

**Mean** – Also known as average, is obtained by dividing the sum of observed values by the number of observations

**Median** – Median is the middle value of a set of data containing an odd number of elements, or the average of the two middle values of a set of data containing an even number of elements

**Mode** – Mode of a set of data is the value that occurs most frequently

**Variance** – Variance measures how far a set of numbers is spread out. A variance of 0 indicates that all the values are identical. A small variance indicates that the data values tend to be very close to the mean and hence to each other. A high variance indicates that the data points are very spread out around the mean and from each other.

**Standard Deviation** – SD is the square root of variance.