PC Benchmarking

PC Benchmarking

A processor and memory testing tool using C

Andrei Onoie, group 30433

Abstract

A benchmark is a widely accepted tool for comparing hardware products. This paper presents a basic outline for designing such a tool and explains the characteristics of benchmark programs in general.

I. Introduction

A good way to study the performance of a hardware product is to run standardized tests on it and compare speed and efficiency to other competing products. The hardware product that we try to analyze in this paper is the CPU of a personal computer, and its Random-Access Memory card. A few of their properties will be measured using low-level function calls and rendered into a human readable format using well-known measurement units such as hertz, bytes and seconds, including their multiples and submultiples (e.g. MHz, GB, ns).

I.1 CPU Definition

A CPU (Central Processing Unit), or more commonly known as the *processor*, is the most important element of a computer system, doing most of the computations needed for programs to run. It communicates with all the other parts of the computer, using them both as input and output.

Some important characteristics of a CPU are its clock frequency, or the speed at which instructions can be executed, its number of cores and the architecture used.

I.2 RAM Definition

RAM stands for Random Access Memory and is the hardware part of a computer that stores application programs and data in current use, that cannot fit inside the small memory of the processor but still need to be reached by it. The data retained in RAM is volatile, meaning that it is lost after the computer is turned off.

Some characteristics of a RAM card are its capacity (usually in gigabytes), its read frequency, and its architecture.

II. Objectives

The objectives of this paper are:

1. Establish the PC characteristics that will be analyzed
2. Research a viable language to analyze the selected characteristics
3. Develop subroutines that tests accurately those PC parts
4. Group subroutines together in a program and write the output in a readable format.

The PC characteristics I have decided to test are the following:

- processor type, e.g. AMD x86_64
- processor frequency, e.g. 2400 MHz
- memory capacity, e.g. 8 GB
- transfer speed of a data block, e.g. 10 ns
- execution time of ALU operations, e.g. 0.2 ns

A viable language to test these parameters is C, since it has easy access to low level instructions.

III. Theoretical foundation

III.1 Benchmark definition

In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. The term benchmark is also commonly utilized for the purposes of elaborately designed benchmarking programs themselves.

Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example CPU performance.

III.2 CPU basics

A PC's CPU is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions.

Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.

- **Clock rate**

  Most CPUs are synchronous circuits, which means they employ a clock signal to pace their sequential operations. The clock signal is produced by an external oscillator circuit that generates a consistent number of pulses each second in the form of a periodic square wave. The frequency of the clock pulses determines the rate at which a CPU executes instructions and, consequently, the faster the clock, the more instructions the CPU will execute each second.

- **Processor architectures**

  The architecture of a processor chip is a description of its basic components and of its basic operations.

  Each processor family has its own architecture. Assembly language is a programming view of the architecture of a particular processor. Each type of processor has its own assembly language.

The instructions supported by a particular processor and their byte-level encodings are known as its instruction-set architecture (ISA). Different “families” of processors, such as Intel IA32, IBM/Freescale PowerPC, and the ARM processor family have different ISAs. A program compiled for one type of machine will not run on another. On the other hand, there are many different models of processors within a single family. Each manufacturer produces processors of ever-growing performance and complexity, but the different models remain compatible at the ISA level.

Some examples of well-known ISAs are the following:

- Intel x86, used by all of AMD and Intel processors, across desktops, laptops and severs
- ARM, used by most mobile devices
- IBM Power/Power PC, microcontrollers and high-end servers
- MIPS, used by embedded and network processors.

- **CPU Cache**

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, L3, L4, etc.).

Data is transferred between memory and cache in blocks of fixed size, called cache lines or cache blocks. When a cache line is copied from memory into the cache, a cache entry is created. The cache entry will include the copied data as well as the requested memory location (called a tag).

- **ALU Operations**

  An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs.

  An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

III.3 The RAM of a PC

Random-access memory (RAM) is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory.

The biggest consideration when buying RAM is how much of it there is. You need a minimum amount of RAM to run a desktop or laptop operating system and many applications have a minimum amount, too. Those requirements are listed in the gigabytes, or GB, and are often between 1GB and 8GB, depending on how demanding the application is. Having more than the minimum is important, as you're unlikely to just be running one application at a time, but having massive amounts doesn't necessarily make your system faster.

One can read and over-write data in RAM. Many computer systems have a memory hierarchy consisting of processor registers, on-die SRAM caches, external caches, DRAM, paging systems and virtual memory or swap space on a hard drive. This entire pool of memory may be referred to as "RAM" by many developers, even though the various subsystems can have very different access times, violating the original concept behind the random-access term in RAM.

The overall goal of using a memory hierarchy is to obtain the highest possible average access performance while minimizing the total cost of the entire memory system (generally, the memory hierarchy follows the access time with the fast CPU registers at the top and the slow hard drive at the bottom).

III.4 The C Programming Language

C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, while a static type system prevents unintended operations. By design, C provides constructs that map efficiently to typical machine instructions and has found lasting use in applications previously coded in assembly language. Such applications include operating systems and various application software for computers, from supercomputers to embedded systems.

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code. The language is available on various platforms, from embedded microcontrollers to supercomputers.

Computing execution time using C

To get the elapsed time, we can get the time using clock() at the beginning, and at the end of the tasks, then subtract the values to get the differences. After that, we will divide the difference by CLOCK_PER_SEC (Number of clock ticks per second) to get the processor time.

However, since C11, the standard way to get a good time estimate is to use the timespec_get function[5], which returns the system clock time:

```c
    struct timespec t0, t1;

    if(timespec_get(&t0, TIME_UTC) != TIME_UTC) {
        printf("Error in calling timespec_get\n");
        exit(EXIT_FAILURE);
    }

    // Do some work ...

    if(timespec_get(&t1, TIME_UTC) != TIME_UTC) {
        printf("Error in calling timespec_get\n");
        exit(EXIT_FAILURE);
    }

    // Calculate the elapsed time
    double diff = (double)(t1.tv_sec - t0.tv_sec) +
((double)(t1.tv_nsec - t0.tv_nsec)/1000000000L);
```

IV. Implementation

IV.1 Finding the architecture of the system

Using the library `Sysinfoapi.h`, we can find a few system attributes using the function `GetNativeSystemInfo()`, which puts the system information into a structure of type `SYSTEM_INFO`. In this structure, we find the attribute `wProcessorArchitecture`, which allows us to check the architecture through some predefined values, detailed below [1].

Value	**Meaning**
`PROCESSOR_ARCHITECTURE_AMD64` `9`	`x64 (AMD or Intel)`
`PROCESSOR_ARCHITECTURE_ARM` `5`	`ARM`
`PROCESSOR_ARCHITECTURE_ARM64` `12`	`ARM64`
`PROCESSOR_ARCHITECTURE_IA64` `6`	`Intel Itanium-based`
`PROCESSOR_ARCHITECTURE_INTEL` `0`	`x86`
`PROCESSOR_ARCHITECTURE_UNKNOWN` `0xffff`	`Unknown architecture.`

Source:

`https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/ns-sysinfoapi-system_info`

IV.2 Finding the CPU clock speed

The Time Stamp Counter [4] (TSC) is a 64-bit register present on all x86 processors since the Pentium. It counts the number of cycles since reset. The instruction `RDTSC` returns the TSC in EDX:EAX.

The Time Stamp Counter was once an excellent high-resolution, low-overhead way for a program to get 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 upon to provide accurate results — unless great care is taken to correct the possible flaws: rate of tick and 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 will be synchronized. Therefore, a program can get reliable results only by limiting itself to run on one specific CPU. Even then, the CPU speed may change because of power-saving measures taken by the OS or BIOS, or the system may be hibernated and later resumed, resetting the TSC.

The `RDTSC` instruction is used in this case to compute the computers clock speed, alongside the functions `QueryPerformanceFrequency()` and `QueryPerformanceCounter`(), found in the library `Profileapi.h.`

`QueryPerformanceFrequency()` returns the current performance-counter frequency, in counts per second.

`QueryPerformanceCounter()` provides the current value of the high resolution timer.

We add the value of frequency into the counter first, then we retrieve the number of clock cycles elapsed for the start value.

After that, we keep retrieving the value of the performance counter until one whole second has gone by, and we retrieve the number of clock cycles after we finished this operation.

Subtracting the start time from this end time and dividing by one million gives us good estimate of the computer's clock speed. [2]

IV.3 Finding the system's physical memory

Using another Windows library function, `GetPhysicallyInstalledSystemMemory()` we can get the number of bytes available in the system's RAM. Dividing by 1024 gives us the memory size in megabytes, or MB. This function is found in the library `Sysinfoapi`.h. [3]

IV.4 Testing the ALU operations cycle count

Here we use the same ASM instruction, RDTSC, and we use inline assembly to test the basic operations: AND, OR, XOR, ADD, MUL, DIV. Here we have a function `GetNoOfCycles(void (*func)())`, and `func` is a function pointer. In another function named `testOperations()` we store all the functions in a function pointer array, and we run each function 1000 times to get an average. We run this for each of the 6 operations described above.

IV.5 Finding transfer speed of a data block

Same as above, we use the RDTSC instruction to compute the cycle count. The function `int moveBlock(int blockSize)` allocates a block of `blockSize` bytes on the heap, and fills it with random values, then allocates another block of the same size, and moves the old block to the new location using the C standard library function `memmove()`.

We count the number of cycles taken by the `memmove()` function call using the `RDTSC` assembly instruction.

IV.6 Putting it all together

The `main()` function of the program compiles together the functionality of the project. First we call the `printArch()` function, which prints the architecture according to the table from section IV.1. After this, we call the `ProcSpeedCalc()` function, which computes the processor speed in the manner described in section IV.2.

Afterwards, the `MemCalc()` function is called, returning the number of megabytes of available memory. The `testOperations()` function tests each of the ALU operations, and then finally the `moveBlock()` function is called 100 times to obtain an average of cycles for moving a block of 10KB size in the heap memory.

V. Testing and validation

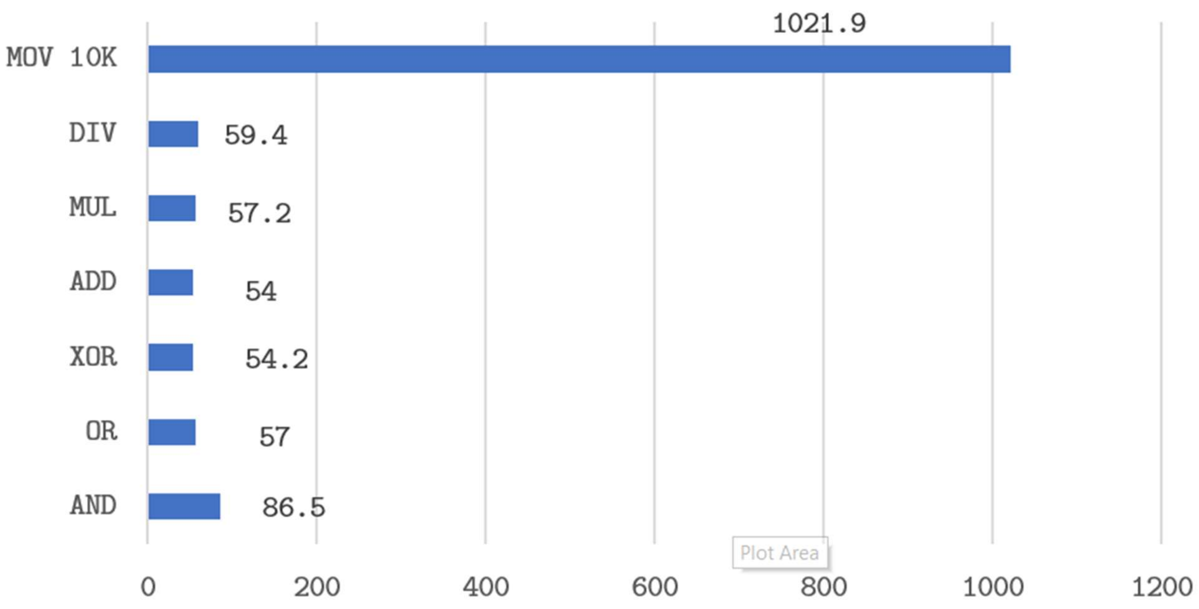
For testing I have used my own laptop computer. The specifications of the PC can be seen in the table below.

Product name	HP Pavilion - 15-au111nq
Microprocessor	Intel® Core™ i7-7500U (2.7 GHz, up to 3.5 GHz with Intel® Turbo Boost Technology, 4 MB cache, 2 cores)
Memory, standard	8 GB DDR4-2133 SDRAM (1 x 8 GB)
Video graphics	NVIDIA® GeForce® 940MX (2 GB DDR3 dedicated)
Hard drive	256 GB M.2 SSD
Optical drive	DVD-Writer

The results from the program developed in this project are the following:

```
C:\> benchmark.exe
Processor architecture: x64 (AMD or Intel)
Clock speed: 2919.24 MHz
Physical memory: 8192MB

Operation testAND took 74.3 cycles on average
Operation testOR took 83.064 cycles on average
Operation testXOR took 125.051 cycles on average
Operation testADD took 96.596 cycles on average
Operation testMUL took 96.177 cycles on average
Operation testDIV took 77.198 cycles on average
Moving a 10k block of memory took 1068.01 cycles.
```



VI. Conclusions and further improvements

The goal of this project was to design and implement a basic benchmark program designed to run on modern personal computers. The parameters tested were the processor architecture, the CPU's clock speed, physical random access memory installed on the motherboard, speed of ALU operations and speed of data transfer inside the RAM. What I liked about the project was the fact that I got a deeper look into computers and understand the operations at the processor level, instruction by instruction. The most challenging task in my opinion was designing the algorithm for testing the clock speed, since it uses concurrency and the RDTSC instruction to compute it mathematically.

Some further improvements would be:

- Testing of GPU memory
- Testing of GPU operations speed
- Testing of GPU clock speed
- Computing a weighted score for the GPU
- Computing a weighted score for the whole computer
- Finding number of cores in CPU
- Testing CPU cache parameters

VII. Bibliography

[1] https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/ns-sysinfoapi-system_info

[2] https://www.codeproject.com/Articles/7340/Get-the-Processor-Speed-in-two-simple-ways

[3] https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getphysicallyinstalledsystemmemory

[4] https://en.wikipedia.org/wiki/Time_Stamp_Counter

[5] https://solarianprogrammer.com/2019/04/17/c17-programming-measuring-execution-time-delaying-program/