University: Technical University of Cluj-Napoca

Faculty: Electronics, Telecommunications and Information Technology

CPU OVERCLOCKING SIMULATOR

Coordinator: Farago Paul

Student: Crehul Vlad

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

This MATLAB project is supposed to simulate the effects of overclocking the CPU of a computer, or simulate it's normal state.

It was inspired by programs such as CPU-Z, or Task Manager performance tab. This idea came to mind simply because I enjoy most things related to computers, and have first hand experience in overclocking.

Many enthusiasts go for extreme overclocking and they often "fry" or destroy their CPUs (expensive ones) in the process. This would allow people to play with the specifications of a CPU with no read danger in this virtual environment.

Table of Contents

Introduction

History

Matlab

MATLAB is a multi-paradigm numerical computing environment and proprietary developed language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python. Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems. As of 2018, MATLAB has more than 3 million users worldwide. MATLAB users come from various backgrounds of engineering, science, and economics.

GUI

A graphical user interface (GUI) is a user interface built with graphical objects -- the components of the GUI -- such as buttons, text fields, sliders, and menus. If the GUI is designed well-designed, it should be intuitively obvious to the user how its components function. For example, when you move a slider, a value changes; when you click an OK button, your settings are applied and the dialog box is closed. By providing an interface between the user and the application's underlying code, GUIs enable the user to operate the application without knowing the commands would be required by a command line interface. For this reason, applications that provide GUIs are easier to learn and use than those that are run from the command line.

CPU

A central processing unit (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. The computer industry has used the term "central processing unit" at least since the early 1960s.Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.

The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. 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.

Most modern CPUs are microprocessors, meaning they are contained on a single integrated circuit (IC) chip. An IC that contains a CPU may also contain memory, peripheral interfaces, and other components of a computer; such integrated devices are variously called microcontrollers or systems on a chip (SoC). Some computers employ a multi-core processor, which is a single chip containing two or more CPUs called "cores"; in that context, one can speak of such single chips as "sockets".

Array processors or vector processors have multiple processors that operate in parallel, with no unit considered central. There also exists the concept of virtual CPUs which are an abstraction of dynamical aggregated computational resources.

Operation

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions that is called a program. The instructions to be executed are kept in

some kind of computer memory. Nearly all CPUs follow the fetch, decode and execute steps in their operation, which are collectively known as the instruction cycle.

After the execution of an instruction, the entire process repeats, with the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter. If a jump instruction was executed, the program counter will be modified to contain the address of the instruction that was jumped to and program execution continues normally. In more complex CPUs, multiple instructions can be fetched, decoded and executed simultaneously. This section describes what is generally referred to as the "classic RISC pipeline", which is quite common among the simple CPUs used in many electronic devices (often called microcontroller). It largely ignores the important role of CPU cache, and therefore the access stage of the pipeline.

Some instructions manipulate the program counter rather than producing result data directly; such instructions are generally called "jumps" and facilitate program behavior like loops, conditional program execution (through the use of a conditional jump), and existence of functions. In some processors, some other instructions change the state of bits in a "flags" register. These flags can be used to influence how a program behaves, since they often indicate the outcome of various operations. For example, in such processors a "compare" instruction evaluates two values and sets or clears bits in the flags register to indicate which one is greater or whether they are equal; one of these flags could then be used by a later jump instruction to determine program flow.

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.

However, architectural improvements alone do not solve all of the drawbacks of globally synchronous CPUs. For example, a clock signal is subject to the delays of any other electrical signal. Higher clock rates in increasingly complex CPUs make it more difficult to keep the clock signal in phase (synchronized) throughout the entire unit. This has led many modern CPUs to require multiple identical clock signals to be provided to avoid delaying a single signal significantly enough to cause the CPU to malfunction. Another major issue, as clock rates increase dramatically, is the amount of heat that is dissipated by the CPU. The constantly changing clock causes many components to switch regardless of whether they are being used at that time. In general, a component that is switching uses more energy than an element in a static state. Therefore, as clock rate increases, so does energy consumption, causing the CPU to require more heat dissipation in the form of CPU cooling solutions.

Performance

The performance or speed of a processor depends on, among many other factors, the clock rate (generally given in multiples of hertz) and the instructions per clock (IPC), which together are the factors for the instructions per second (IPS) that the CPU can perform. Many reported IPS values have represented "peak" execution rates on artificial instruction sequences with few branches, whereas realistic workloads consist of a mix of instructions and applications, some of which take longer to

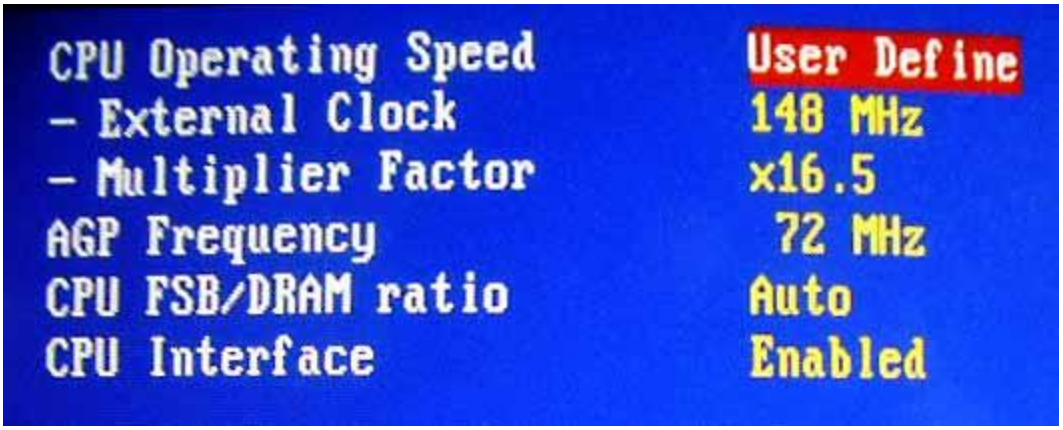
execute than others. The performance of the memory hierarchy also greatly affects processor performance, an issue barely considered in MIPS calculations. Because of these problems, various standardized tests, often called "benchmarks" for this purpose—such as SPECint—have been developed to attempt to measure the real effective performance in commonly used applications.

Processing performance of computers is increased by using multi-core processors, which essentially is plugging two or more individual processors (called cores in this sense) into one integrated circuit. Ideally, a dual core processor would be nearly twice as powerful as a single core processor. In practice, the performance gain is far smaller, only about 50%, due to imperfect software algorithms and implementation. Increasing the number of cores in a processor (i.e. dual-core, quad-core, etc.) increases the workload that can be handled. This means that the processor can now handle numerous asynchronous events, interrupts, etc. which can take a toll on the CPU when overwhelmed. These cores can be thought of as different floors in a processing plant, with each floor handling a different task. Sometimes, these cores will handle the same tasks as cores adjacent to them if a single core is not enough to handle the information.

Due to specific capabilities of modern CPUs, such as hyper-threading and uncore, which involve sharing of actual CPU resources while aiming at increased utilization, monitoring performance levels and hardware use gradually became a more complex task. As a response, some CPUs implement additional hardware logic that monitors actual use of various parts of a CPU and provides various counters accessible to software; an example is Intel's Performance Counter Monitor technology.

Overclocking

The purpose of overclocking is to gain additional performance from a given component by increasing its operating speed. Normally, on modern systems, the target of overclocking is increasing the performance of a major chip or subsystem, such as the main processor or graphics controller, but other components, such as system memory (RAM) or system buses (generally on the motherboard), are commonly involved. The trade-offs are an increase in power consumption (heat) and fan noise (cooling) for the targeted components. Most components are designed with a margin of safety to deal with operating conditions outside of a manufacturer's control; examples are ambient temperature and fluctuations in operating voltage. Overclocking techniques in general aim to trade this safety margin by setting the



device to run in the higher end of the margin, with the understanding that temperature and voltage must be more strictly monitored and controlled by the user. Examples are that operating temperature would need to be more strictly controlled with increased cooling, as the part will be less tolerant of increased temperatures at the higher speeds. Also base operating voltage may be increased to compensate for unexpected voltage drops and to strengthen signalling and timing signals, as low-voltage excursions are more likely to cause malfunctions at higher operating speeds.

While most modern devices are fairly tolerant of overclocking, all devices have finite limits. Generally for any given voltage most parts will have a maximum "stable" speed where they still operate correctly. Past this speed the device starts giving incorrect results, which can cause malfunctions and sporadic behavior in any system depending on it. While in a PC context the usual result is a system crash, more subtle errors can go undetected, which over a long enough time can give unpleasant surprises such as data corruption (incorrectly calculated results, or worse writing to storage incorrectly) or the system failing only during certain specific tasks (general usage such as internet browsing and word processing appear fine, but any application wanting advanced graphics crashes the system).

At this point an increase in operating voltage of a part may allow more headroom for further increases in clock speed, but the increased voltage can also significantly increase heat output. At some point there will be a limit imposed by the ability to supply the device with sufficient power, the user's ability to cool the part, and the device's own maximum voltage tolerance before it achieves destructive failure. Overzealous use of voltage and/or inadequate cooling can rapidly degrade a device's performance to the point of failure, or in extreme cases outright destroy it.

The speed gained by overclocking depends largely upon the applications and workloads being run on the system, and what components are being overclocked by the user; benchmarks for different purposes are published.

Underclocking

Conversely, the primary goal of underclocking is to reduce power consumption and the resultant heat generation of a device, with the trade-offs being lower clock speeds and reductions in performance. Reducing the cooling requirements needed to keep hardware at a given operational temperature has knock-on benefits such as lowering the number and speed of fans to allow quieter operation, and in mobile devices increase the length of battery life per charge. Some manufacturers underclock components of battery-powered equipment to improve battery life, or implement systems that detect when a device is operating under battery power and reduce clock frequency accordingly.

Underclocking is almost always involved in the latter stages of Undervolting, which seeks to find the highest clock speed that a processor will stably operate at a given voltage. That is, while overclocking seeks to maximize clock speed with temperature and power as constraints, underclocking seeks to find the highest clock speed that a device can reliably operate at a fixed, arbitrary power limit. A given device may operate correctly at its stock speed even when undervolted, in which case underclocking would

only be employed after further reductions in voltage finally destabilizes the part. At that point the user would need to determine if the last working voltage and speed have satisfactorily lowered power consumption for their needs – if not then performance must be sacrificed, a lower clock is chosen (the underclock) and testing at progressively lower voltages would continue from that point. A lower bound is where the device itself fails to function and/or the supporting circuitry cannot reliably communicate with the part.

Underclocking and undervolting would be attempted on a desktop system to have it operate silently (such as for a home entertainment center) while potentially offering higher performance than currently offered by low-voltage processor offerings. This would use a "standard-voltage" part and attempt to run with lower voltages (while attempting to keep the desktop speeds) to meet an acceptable performance/noise target for the build. This was also attractive as using a "standard voltage" processor in a "low voltage" application avoided paying the traditional price premium for an officially certified low voltage version. However again like overclocking there is no guarantee of success, and the builder's time researching given system/processor combinations and especially the time and tedium of performing many iterations of stability testing need to be considered. The usefulness of underclocking (again like overclocking) is determined by what processor offerings, prices, and availability are at the specific time of the build.

CPU Locking

CPU locking is the process of permanently setting a CPU's clock multiplier. AMD CPUs are unlocked in early editions of a model and locked in later editions, but nearly all Intel CPUs are locked and recent models are very resistant to unlocking to prevent overclocking by users. AMD ships unlocked CPUs with their Opteron, FX, Ryzen and Black Series line-up, while Intel uses the monikers of "Extreme Edition" and "X-Series." Intel usually has one or two Extreme Edition CPUs on the market as well as X series and K series CPUs analogous to AMD's Black Edition. AMD has the majority of their desktop range in a Black Edition.

Users unlock CPUs to allow underclocking, overclocking, and front side bus speed (on older CPUs) compatibility with certain motherboards, but unlocking invalidates the manufacturer's warranty and mistakes can cripple or destroy a CPU. Locking a chip's clock multiplier does not necessarily prevent users from overclocking, as the speed of the front-side bus or PCI multiplier (on newer CPUs) can still be changed to provide a performance increase. AMD Athlon and Athlon XP CPUs are generally unlocked by connecting bridges (jumper-like points) on the top of the CPU with conductive paint or pencil lead. Other CPU models (determinable by serial number) require different procedures.

Increasing front-side bus or the northbridge/PCI clock can overclock locked CPUs, but this throws many system frequencies out of whack, since the RAM and PCI frequencies are modified as well.

One of the easiest ways to unlock older AMD Athlon XP CPUs was called the pin mod method, because it was possible to unlock the CPU without permanently modifying bridges. A user could simply put one wire (or some more for a new multiplier/Vcore) into the socket to unlock the CPU.

Main Interface

1) Button to close the program

2) Button to close the "welcoming"interface and open the actual simulator.

3) Button that opens the documentation (this doc).

Simulation Interface

1) Edit Text Box that allows the user to type the multiplier.

2)Edit Text Box that allows the user to type the Front Bus Speed/Base Clock.

3)Edit Text Box that allows the user to type the Input Voltage.

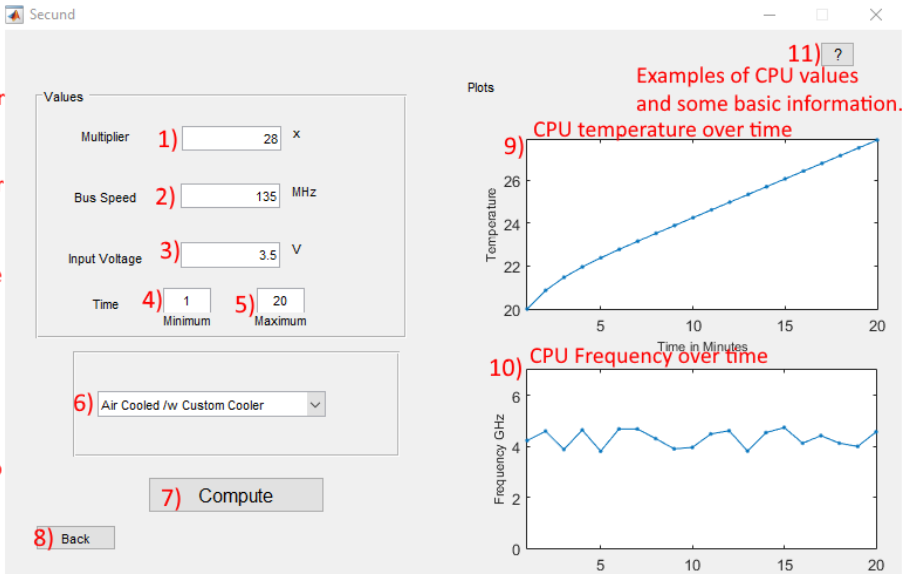
4)Edit Text Box that allows the user to type the starting time from where the resulting graph is plotted.

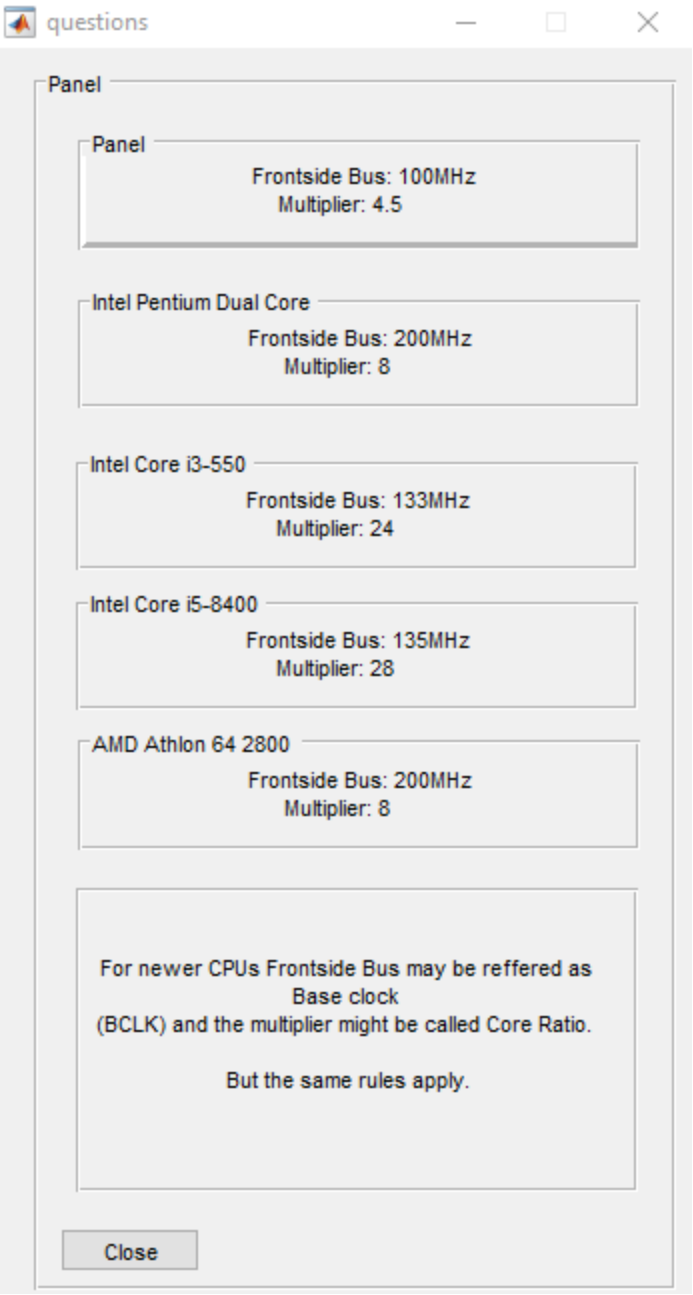
5)Edit Text Box that allows the user to type the ending time from where the resulting graph is plotted.

6)Drop Down Menu that allows to choose a generic cooler.

7)Button that uses the user input to plot.

8) Back Button that opens the "welcome" interface.





This is the basic information page mentioned at “Simulation Interface” 11)

Theoretical Explanation

Formulas

The formula for processor speed is: Frontside Bus/Base Clock x Multiplier/Core Ratio = Processor Speed.

The CPU(Intel Pentium III) runs at 450 million clock cycles per second. The CPU runs at at a speed of 450 megahertz. Using our processor speed equation we have: *100MHz (frontside bus) x 4.5 (multiplier) = 450MHz (processor speed)*

Using our example above, the multiplier is 4.5. Since valid multipliers end in .0 or .5, you could try increasing the multiplier to 5.0 to obtain a performance boost (which would result in 100MHz x 5.0 = 500MHz).

Before we make any changes to the base clock frequency or CPU ratios, there's an additional element to consider: voltage. The voltage applied to your CPU varies continually. Faster operating speeds require additional power for stability. Your motherboard handles this behind the scenes but, should you wish, you can set manual voltage levels to support your overclocking efforts. There is no real formula to calculate the needed voltage, only trial and error.

Goal to be obtained

A simulator for enthusiasts to try overclocking a CPU without risking their real hardware. Making it a safe environment.

My Results

Errors

The biggest problem is what I will also be mentioned at "Did I reach my goal?". I do not have the knowledge to emulate a CPU fully. So I had to resort to some random variables to emulate how a CPU would "react".

How it was done & Code Explanation

The interface was mostly done by using GUIDE. The plotting, errors and simulation implementation was done all in code.

Plotting the temperature code.

```
axes(handles.AxesTemps);
den =
2.^([str2double(get(handles.mintime,'String')):str2double(get(handles.maxtime
,'String'))]-1);
 if str2double(get(handles.VoltageEditText,'String')) > 0.9 &&
str2double(get(handles.VoltageEditText,'String')) <4.0
    Temp =1./den+rand(1)*cool;
    elseif str2double(get(handles.VoltageEditText,'String')) < 0.9
       Temp =1./den+rand(1)*cool*0.1;
    else
         Temp =1+1./den+rand(1)*cool;
 end
Temp(1)=20;

assignin('base','Temp',Temp);
plot(cumsum(Temp),'.-');
fontSize=8;
xlim([str2double(get(handles.mintime,'String'))
str2double(get(handles.maxtime,'String'))]);
xlabel('Time in Minutes', 'FontSize', fontSize);
ylim([20 inf]);
ylabel('Temperature', 'FontSize', fontSize);
```

Cooling Choice, Pop-up Menu

```
popup_sel_index = get(handles.CoolingPop, 'Value');
switch popup_sel_index
    case 1
        err = errordlg('Choose a cooler. You cannot run with no cooling!','No
cooler selected!');
        uiwait(err);
    return
    case 2
        cool=0.8;
        assignin('base','cool',cool);
    case 3
        cool=0.5;
        assignin('base','cool',cool);
    case 4
        cool=0.3;
        assignin('base','cool',cool);
    case 5
        cool=0.1;
        assignin('base','cool',cool);
end
```

Plotting the Frequency code.

```
axes(handles.AxesFreq);
time=str2double(get(handles.mintime,'String')):1:str2double(get(handles.maxti
me,'String'));
 assignin('base','time',time);
for i=1:str2double(get(handles.maxtime,'String'))-
str2double(get(handles.mintime,'String'))+1
    if str2double(get(handles.VoltageEditText,'String')) > 0.9 &&
str2double(get(handles.VoltageEditText,'String')) <4.0

Freq(i)=str2double(get(handles.MultiplierTextEdit,'String'))*(str2double(get(
handles.BusEditText,'String'))/1000)+rand(1);
    elseif str2double(get(handles.VoltageEditText,'String')) < 0.9
        Freq(i)=(str2double(get(handles.BusEditText,'String'))/1000)-rand(1);
    else

Freq(i)=str2double(get(handles.MultiplierTextEdit,'String'))*(str2double(get(
handles.BusEditText,'String'))/1000)+1 + (3-1).*rand(1);
    end
end
 assignin('base','Freq',Freq);
plot(time,Freq,'.-');
fontSize=8;
xlabel('Time in Minutes', 'FontSize', fontSize);
xlim([str2double(get(handles.mintime,'String'))
str2double(get(handles.maxtime,'String'))]);
ylabel('Frequency GHz', 'FontSize', fontSize);
ylim([0 7]);
```

Code for error messages.

```
if str2double(get(handles.MultiplierTextEdit,'String')) < 1
    err = errordlg('Multiplier must be bigger than 0!','Wrong Parameters');
    uiwait(err);
    return
end
if str2double(get(handles.BusEditText,'String')) < 1
    err = errordlg('Bus Frequency must be bigger than 0!','Wrong
Parameters');
    uiwait(err);
    return
end
if
(str2double(get(handles.BusEditText,'String'))/1000)*str2double(get(handles.M
ultiplierTextEdit,'String')) > 6
    err = errordlg('Incorrect Values. Use lower frequency or
multiplier!','Wrong Parameters');
    uiwait(err);
    return
end
if str2double(get(handles.VoltageEditText,'String')) > 5
    err = errordlg('Input Voltage is too high!','Wrong Parameters');
    uiwait(err);
    return
end
```

Conclusion

Did I reach my goal?

Partially. Normally a program such as CPU-Z or any benchmarking programs or even the task manager performance tab simply analyzes the data in real time. There are a few variables I do not know how to simulate and that is why I decided to add in some random data. Each CPU is unique and its overclocking potential or just turbo clock speed is dependent on the quality of the materials used so 2 CPUs that are marked as the same might not perform truly the same and when overclocking the difference will be even more obvious.

Further Developments

The only example of a CPU overclocking simulator I know of is in a game. "PC Building Simulator" and that game does allow you to overclock your CPU, GPU and it seems pretty close to real life, but I do not know yet how they have done that, it might be just a database of benchmarks. But if something could be done is to remove the randomness and make it more accurate.

Appendix

https://www.webopedia.com/DidYouKnow/Computer_Science/overclocking.asp

https://www.tomshardware.com/reviews/cpu-overclocking-guide,4593.html

https://en.wikipedia.org/wiki/Overclocking

https://linustechtips.com/main/topic/780119-cpu-overclocking-guide/

https://en.wikipedia.org/wiki/Central_processing_unit