|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Домашняя работа №3*

*По предмету: «Иностранный язык»*

Студент: Гасанзаде М.А.,

Группа: ИУ7-66

Москва, 2020 г.

CONTENTS

[EXAM TOPIC 3](#_Toc39165168)

[HISTORY 3](#_Toc39165169)

[DEFINITION 4](#_Toc39165170)

[COMPUTER ORGANIZATION 4](#_Toc39165171)

[TOPIC PLAN 5](#_Toc39165172)

[QUESTIONS 5](#_Toc39165173)

[Q/A for article “THE VALUE OF TESTING” 8](#_Toc39165174)

[GRAMMAR: 9](#_Toc39165175)

# EXAM TOPIC

# TOPIC PLAN

1. History
2. Categories
3. Definition
4. Computer organization
5. Conclusion

# QUESTIONS

1. What you can say about Neumann architect?

The Von Neumann architecture, also known as the Princeton architecture, is a computer architecture based on that described in 1945 by the mathematician and physicist John Von Neumann. He described an architecture for an electronic digital computer with parts consisting of a processing unit containing an arithmetic logic unit (ALU) and processor registers, a control unit containing an instruction register and program counter (PC), *a memory to store both data and instructions*, external mass storage, and input and output mechanisms. The meaning has evolved to be any stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus.

1. Definition of computer architecture?

Computer architecture is concerned with balancing the performance, efficiency, cost, and reliability of a computer system. The case of instruction set architecture can be used illustrate the balance of these competing factors.

1. Which categories have architecture?

The discipline of computer architecture has three main subcategories:

* Instruction set architecture (ISA): defines the machine code that a processor reads and acts upon as well as the word size, memory address modes, processor registers, and data type.
* Microarchitecture: also known as "computer organization", this describes how a particular processor will implement the ISA. The size of a computer's CPU cache for instance, is an issue that generally has nothing to do with the ISA.
* Systems design: includes all of the other hardware components within a computing system, such as data processing other than the CPU (e.g., direct memory access), virtualization, and multiprocessing

1. What include Computer hardware?

Computer hardware includes the physical parts of a computer, such as the case, central processing unit (CPU), monitor, keyboard, computer data storage, graphics card, sound card, speakers and motherboard.

By contrast, software is the set of instructions that can be stored and run by hardware. Hardware is so-termed because it is "hard" or rigid with respect to changes, whereas software is "soft" because it is easy to change.

Hardware is typically directed by the software to execute any command or instruction. A combination of hardware and software forms a usable computing system, although other systems exist with only hardware.

1. What do you know about supercomputers?

A supercomputer is a computer with a high level of performance as compared to a general-purpose computer. The performance of a supercomputer is commonly measured in floating-point operations per second (FLOPS) instead of million instructions per second (MIPS). Since 2017, there are supercomputers which can perform over a hundred quadrillion FLOPS (petaFLOPS). Since November 2017, all of the world's fastest 500 supercomputers run Linux-based operating systems. Additional research is being conducted in China, the United States, the European Union, Taiwan and Japan to build faster, more powerful and technologically superior exascale supercomputers.

# Q/A for article “MAKE INVISIBLE MORE VISIBLE”

1. What can you say about source code?

The source code has no innate presence, no innate behavior and does not obey the laws of physics. This is visible when you load it into the editor, but close the editor and it disappears. Think about it a little longer, about how a tree falls, when no one hears it, you begin to wonder if it even existed. A running application has presence and behavior, but shows nothing from the source code from which it was built.

1. What can you say about bugs?

If you’re 90% done and endlessly stuck trying to debug your way through the last 10%, then you’re not 90% done, are you? Fixing bugs is not making progress. You aren’t paid to debug. Debugging is waste. It’s good to make waste more visible so you can see it for what it is and start thinking about trying not to create it in the first place.

1. How to make Invisible visible?

Invisibility can be dangerous. But you can protect yourself from mistakes:

* Using unit tests in each step of your project
* Using bulletin boards and cards makes progress visible and concrete
* Running unit tests provides evidence about the code's behavior. It helps reveal the presence (or absence) of runtime qualities you'd like the appli­cation to exhibit, such as robustness and correctness.

1. What give you visibility?

If your project is apparently on track, and one week later it’s six months late, you have problems—the biggest of which is probably not that it’s six months late, but the invisibility force fields powerful enough to hide six months of lateness! Lack of visible progress is synonymous with lack of progress.

Visibility gives confidence that progress is genuine and not an illusion, deliberate and not unintentional, repeatable and not accidental.

1. What about unit tests?

It is worth paying attention to the start of the project, starting to process the smallest errors, and then you will not have to worry about the invisible consequences. You run the risk of losing more time later on fixing minor issues than if you started debugging at first.

# GRAMMAR:

26.A  
27.A  
28.B  
29.A  
30.D  
31.D  
32.A  
33.D  
34.D  
35.C  
36.B  
37.B  
38.A  
39.C  
40.B