NAME – ISHAN

ROLL NO. – FET-BAML-2022-26-012

SUBJECT – IOT AND AUTOMATION

**ASSINGMENT-4**

The Architecture of Modern Computers:

• How does the architecture of modern CPUs (e.g., x86-64 architecture) handle parallelism, and what are the implications for software design and performance?

• What are the key differences between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) architectures, and how do these differences influence the design of operating systems?

• Explain the differences between the CISC (Complex Instruction Set Computing) and RISC (Reduced Instruction Set Computing) architectures. Additionally, compare the John von Neumann architecture with the Harvard architecture, focusing on their memory organization and instruction processing. How do these differences impact the performance and design of modern processors?

Ans:

Modern CPUs have evolved their architecture to more efficiently handle complex tasks, with less reliance on having humans constitute, via paralleler, instructions to set design and supporting memory organization. Here’s a breakdown of these aspects and their implications:

1. Parallelism in Modern CPU Architectures (e.g., x86-64 Architecture)

• Instruction-Level Parallelism (ILP): Multiple instructions can be processed simultaneously on a single CPU core via pipelining, superscalar execution, and out of order execution.

• Data-Level Parallelism (DLP): On x86-64 SIMD (Single Instruction, Multiple Data) extensions such as AVX, allow the CPU to perform the same operation for each of multiple data points, speeding up jobs like multimedia processing.

• Thread-Level Parallelism (TLP): Because it handles multiple threads at once, multithreading and multicore design of a CPU increases throughput and application responsiveness.

Implications for Software Design: Concurrency Models, Performance Tuning, Scalability

2. Differences Between RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing)

CISC (e.g., x86):

• Complex Instruction Set: Includes a large number of instructions, some of which can perform multi-step operations, reducing the number of instructions per program.

• Variable-Length Instructions: Instructions may vary in length, requiring more complex decoding but potentially fewer instructions per task.

• Memory Access: Supports memory-to-register operations directly within instructions.

RISC (e.g., ARM):

• Reduced Instruction Set: Typically, it emphasizes a smaller set of simpler instructions that each do one operation, resulting in a higher instruction count.

• Fixed-Length Instructions: Simplest decoding and pipelining, increases instruction throughput.

• Load/Store Architecture: It enables memory access only via explicit load and store instructions, making the design simpler and hence more power efficient.

Impact on Operating Systems:

• CISC OS Design: Simple instructions and complex instructions can act to decrease total instruction count for OSs running on CISC architectures; however, the overhead for more cumbersome decoding stages must also be manually managed.

• RISC OS Design: RISC architectures force OSs to use a higher instruction count, but in return for this, there is a simpler, faster pipeline. Typically they have a focus on modularity and high throughput optimization.

Impact on Processor Performance and Design:

• Von Neumann Architecture is simpler and less costly to implement, which makes it pervasive in the design space of general purpose CPUs; however, this architecture is less attractive in the high performance space because of bus contention.

• The Harvard Architecture is more suitable for embedded systems or real time processing where the benefit of keeping separate data and instruction streams outweighs the performance achieved by Harvard Architecture. There are, however, more technical and expensive ways of implementing them.

Microcontrollers (e.g., Arduino Uno R3):

• How does the AVR architecture used in the Arduino Uno R3 differ from the architecture used in more advanced microcontrollers like ARM Cortex-M?

• What is the significance of using an 8-bit microcontroller (like the ATmega328P in Arduino Uno) compared to 16-bit or 32-bit microcontrollers in terms of memory addressing, processing power, and application suitability?

Ans:

AVR vs. ARM Architecture

• AVR (Atmel RISC Architecture): Used in the Arduino Uno R3. The design is a simple 8 bit RISC architecture with a low power consumption and a simple instruction set.

• ARM Cortex-M: A family of 32 bit RISC architectures for more advanced microcontrollers. AVR is basically restricted, uses lower performance, smaller address space, and has fewer features than AVR.

Microcontrollers 8bit vs 16/32bit

• Memory Addressing: Smaller 8bit microcontrollers have less address space and so can only access less memory.

• Processing Power: High processing power microcontrollers come with 16bit and 32 bit microcontrollers which makes them suitable for demanding applications.

• Application Suitability: 8 bit microcontrollers are appropriate for light usage and simplest functions that use small amount of memory and processing requirements while the 16 bit and 32 bit micro controllers are ideal for more complex applications.

Memory Segmentation:

• How does memory segmentation in x86 architecture support backward compatibility, and what are the advantages and disadvantages compared to flat memory models used in modern 64-bit systems?

• What is the difference between User Space and Kernel Space in the virtual memory layout of modern computers, and why is this separation important? Additionally, how does memory segmentation work, and what role does it play in managing memory in older vs. modern computing systems?

Ans:

Memory Segmentation and Backward Compatibility

Memory segmentation in x86 architecture divides the address space into segments, which can be used to isolate different processes or parts of a program. This allows for backward compatibility with older 16-bit software, which used segmented memory. However, segmentation can be inefficient and complex to manage.

User Space vs. Kernel Space

• User Space: The portion of memory to which user applications have access.

• Kernel Space: The space available to the operating system kernel. Security and isolation requires this separation. This prevents user applications from being able to directly access or modify kernel code, thereby posing potential problems such as system instability or security breaches.

Memory Segmentation on Older vs. Modern Systems

• Older Systems: A lot was done with memory segmenting to manage memory and protect processes.

• Modern Systems: Some cases still use segmentation, although the flat memory models are more common. In 64 bit systems, flat memory models are helpful, more simpler and more efficient to manage memory.

Memory Location and Byte Addressability

1-Byte Memory Locations

Each memory location is generally 1 byte in modern computer systems because it is a convenient unit of data. Bytes can represent a wide range of values, from characters to small integers. This design choice simplifies memory addressing and data manipulation.

Endianness

Endianness refers to the order in which bytes are stored in memory.

• Big-Endian: The most significant byte is stored first.

• Little-Endian: The least significant byte is stored first.

Developers must be aware of the endianness of the system they are working with to ensure correct data interpretation. Misunderstanding endianness can lead to data corruption and errors.