2024 International Conference on Science, Engineering and Business for Driving Sustainable Development Goals (SEB4SDG) | 979-8-3503-5815-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/SEB4SDG60871.2024.10630195

Memory Interfacing in Mechatronics Systems: Constraints and Considerations

Snow Ngozi Monye
[1]Department of Information Communication Technology
University of Delta Agbor,
Delta State, Nigeria
ngozi.monye@unidel.edu.ng

Stella Isioma Monye
Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, Nigeria
monyeis@abuad.edu.ng

Joseph F. Kayode
Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, Nigeria
kayodejf@abuad.edu.ng

Sunday Adeniran Afolalu[2,3]
[2]Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, Nigeria
[3]Department of Mechanical and Engineering Science
University of Johannesburg, South Africa
Adeniran.afolalu@abuad.edu.ng

Imhade Princess Okokpujie[2,4]
[2]Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, 360001 Nigeria
[4]Department of Mechanical and Industrial Engineering Technology
University of Johannesburg
Johannesburg, 2028 South Africa
ip.okokpujie@abuad.edu.ng

Aderonke Oluseun. Akinwumi
Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, Nigeria
aderonkeakinwumi@abuad.edu.ng

David Agbemuko
Department of Mechanical and Mechatronics Engineering
Afe Babalola University
Ado Ekiti, Nigeria
davidagbemuko.ad@gmail.com

Kazeem Bello Aderemi
Department of Mechanical Engineering,
Federal University of Oye Ekiti.
Ekiti State, Nigeria
kazeem.bello@fuoye.edu.ng

Abstract**—** Modern computer systems depend heavily on memory interfacing to enable effective data flow between the central processing unit (CPU) and various memory devices. This essay offers a thorough examination of memory interfacing, examining its key components, difficulties, and suggested solutions. The study starts off with a description of memory interfacing and emphasizes its importance in getting the best system performance. It digs into the details of data transfer, highlighting how crucial it is for the CPU and memory modules to communicate effectively and reliably. We explore methods to maximise data transfer rates and reduce latency, including Direct Memory Access (DMA), caching systems, and pipelining. We thoroughly study addressing modes, another crucial component of memory interfaces. The exploration of direct addressing, indirect addressing, and indexed addressing modes highlights their function in gaining access to certain memory locations and obtaining data. The idea of memory hierarchy is examined, demonstrating how memory systems are arranged into several tiers based on speed, cost, and capacity. With a focus on cache memory, main memory (RAM), and secondary storage devices like hard drives and solid-state drives, the effect of memory hierarchy on memory interface is examined. The obstacles associated with memory interface are further discussed in the study, including compatibility problems, timing restrictions, electrical considerations, and the requirement for standardised protocols and standards. The incorporation of compatible components, observance of voltage and timing requirements, and adherence to industry-standard memory interface protocols are suggested as potential solutions to these difficulties. This study paper concludes by offering a thorough grasp of memory interfacing, its difficulties, and suggested remedies. This study adds to the body of knowledge on memory interface by examining data transfer, addressing modes, memory hierarchy, and compatibility difficulties. It also provides helpful insights for scholars and practitioners in the field of computer systems and architecture.

Keywords: Interfacing, Mechatronics, Memory, Constraints, Computer Systems

I. Introduction

In the modern world, computers come in a broad variety of sizes and designs and are used for a wide range of tasks in several industries. Computers are used extensively in a wide range of tasks, from important ones like controlling air traffic and furthering cancer research to more frivolous ones like gaming and improving photos. Computers differ in appearance and have a wide range of uses, yet they are quite similar in terms of their core operation. They rely on a small number of technologies that give them the ability to carry out the numerous miracles we have grown to anticipate. The microprocessor, sometimes referred to as the central processing unit (CPU), is the brain of every modern computer. This tiny, square piece of silicon has a complex system of etched gates and channels that allow electrons to flow across it. This network resembles a more compact form of the familiar circuitry seen in everyday objects like television remote controls and vintage radios because it uses transistors as gates and wires or lines as channels. Therefore, the microprocessor not only serves as the essential "heart" of a contemporary computer, but also functions as a computer unto itself. The basic ideas that underlie all elements of contemporary computing, including the aforementioned air traffic control systems and the silicon brain controlling the brakes of a luxury automobile, will become clear once you understand how this little computer functions. In essence, a computing system takes a sequence of data and instructions (as shown in figure 1), or "code," and produces a sequence of results as its output. For the sake of simplicity, let's say that the data sequence holds the information on which the various mathematical operations are conducted, whereas the code sequence consists of different types of mathematical operations. As a result, the culmination of these processes is the outcomes sequence. When the operators of the code sequence work with the operands of the data sequence, one may also see the beginning of the outcomes series.[1]

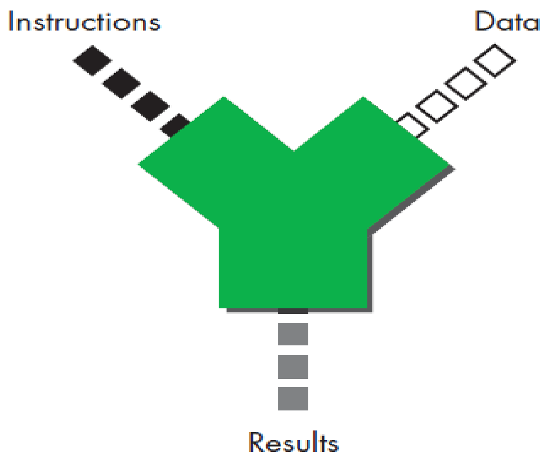


FIGURE 1: A simple representation of the working of a digital computer Source: [1]

A. MICROPROCESSORS

A microprocessor (as described with figure 2) is made up of numerous crucial components that collaborate to carry out computational operations. The Arithmetic Logic Unit (ALU), data bus, memory, and input/output (I/O) ports are a few of these parts. The ALU serves as the microprocessor's computing core and is in charge of carrying out arithmetic operations like addition, subtraction, and multiplication as well as logical operations like AND, OR, and NOT. The data bus acts as a communication channel to allow data to be sent between various components of the CPU and outside devices. Memory, a key component, stores data and instructions for processing, enables the microprocessor to quickly access information. The I/O ports are also used as interfaces for connecting peripheral devices, allowing the CPU and the outside world to exchange data. Together, these essential microprocessor components make sure that activities are carried out effectively and enable fluid communication with the outside world. [2]

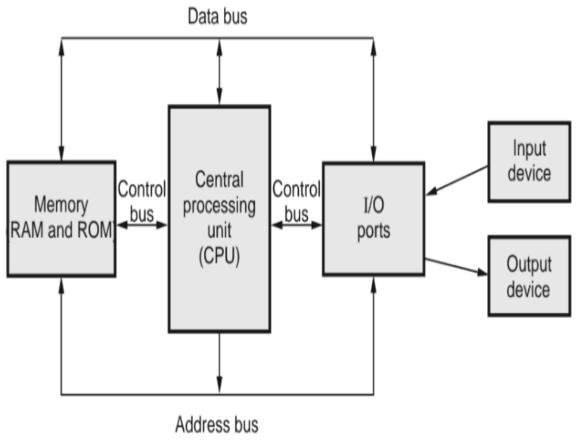
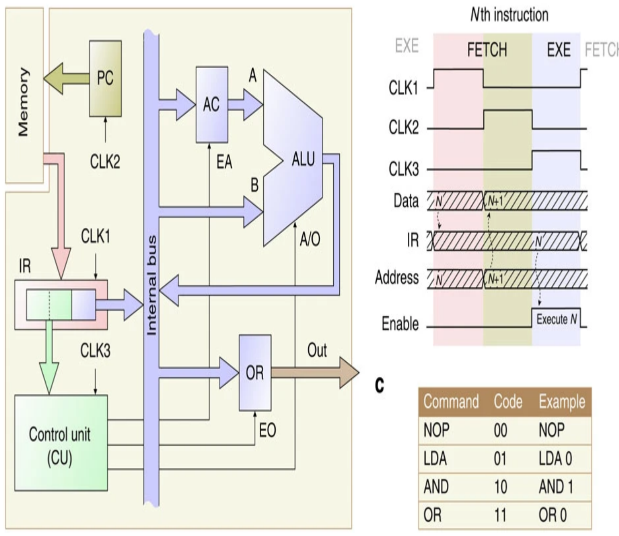


FIGURE 2: A block diagram describing the parts of a microprocessor Source: [2]

Our society has undergone a significant transition since microcomputers first appeared in the 1970s. Since that time, silicon has been used to make microprocessors almost exclusively, but the desire for faster processing speeds, larger integration densities, reduced power consumption, and improved interoperability with ordinary objects has spurred researchers to look for alternatives. In addition, chips based on carbon nanotubes or thin-film plastic technology may make it possible to embed electronic intelligence into any object for the Internet of Things. Germanium and III-V compound semiconductors are being considered as promising candidates for future high-performance processor generations. The architectural block diagram of our microprocessor, for instance, is shown in figure 3 below.



Command	Code	Example
NOP	00	NOP
LDA	01	LDA 0
AND	10	AND 1
OR	11	OR 0

FIGURE 3: The architecture of a microprocessor Source: [3]

B. DESIGN/ARCHETECTURE OF MICROPROCESSORS

The functionality of a computer system is significantly influenced by the microprocessor's design. A microprocessor is fundamentally made up of a number of parts that cooperate to carry out instructions and process data. The strong symbiotic interaction between the microprocessor and memory lies at the heart of this architecture. Memory is used by the microprocessor to store both data and instructions. It retrieves instructions from memory, decodes them, and then decides which operations should be carried out. The microprocessor also keeps temporary information and early findings in memory while doing calculations. Typically, the memory subsystem is divided into many tiers, including cache, RAM, and ROM. Depending on the speed and capacity requirements, the microprocessor accesses these tiers of memory. When compared to RAM and ROM, cache memory, which is situated closest to the CPU, offers quick access to frequently used information and instructions. Address and data buses are used by the CPU to connect to the memory. While the data bus transports the actual data between the CPU and memory, the address bus sends the memory address of the data or instruction being accessed. Microprocessors frequently use memory management strategies including caching, pipelining, and virtual memory to enhance performance. The microprocessor can handle more complicated jobs thanks to these strategies, which also increase memory access rates and overall efficiency. In conclusion, a microprocessor's design closely connects with memory, making it easier to retrieve and store data and instructions. This connection enables the microprocessor to carry out operations, process data, and operate a computer system as a whole. [3]

The status quo is no longer enough to preserve the long-standing heritage of microprocessor innovation in the industry as technology improves, adopting more complex designs, and confronting larger hurdles in technology scaling and power management. An all-encompassing strategy that considers the architecture, microarchitecture, bus memory, and I/O performance of the computing platform is required to allow improvements in system performance and power efficiency. Both general-purpose and networking processor MIPS will increase with the introduction of multithreading and multi-core computer micro-architectures. Extensive on-die caches will be useful for transaction-focused server CPUs. The creation of specialized designs and circuit approaches will be essential for achieving improved performance with more efficiency. Integration of DSP capabilities will be necessary for applications like media-rich communications, computer vision, and voice recognition in the growth of future microprocessors. These developments, which center on handling natural data, will eventually change the existing computer paradigm from one that is data-based and machine-centric to one that is knowledge-based and human-centric. New applications and user communities for microprocessors will appear as the Internet plays a more crucial role for both corporations and consumers. High-bandwidth Internet access, powered by powerful computer servers, is required to meet the needs of the Internet economy. [4]

- Establishing a stable link between the CPU and various memory modules, including as random access memory (RAM), read-only memory (ROM), and peripheral devices, is known as memory interfacing. The effectiveness of this interface has a direct impact on the system's speed, dependability, and overall performance. In order to build and implement effective computing systems, it is crucial to comprehend the complexities of memory interfacing.[5]
- Data transfer is one of the core components of memory interface. The execution of instructions, the storage of data, and the retrieval of information all depend on the efficient and dependable transmission of data between the CPU and memory. This study intends to explore the many methods and protocols used to enhance data transfer speeds and reduce latency, ensuring that the system runs smoothly and satisfies the requirements of contemporary computer applications.[6]
    - In memory interfacing, addressing modes are important. The CPU can successfully access specified memory locations by using various addressing techniques. The CPU locates and retrieves data from memory via direct addressing, indirect addressing, indexed addressing, and other modes. In order to create memory interface designs that satisfy certain computing requirements, it is essential to comprehend the subtleties of different addressing modes.[7]
    - Memory interface and the idea of memory hierarchy are very closely related. A significant tactic to boost system performance is to layer or layer memory systems according to their speed, cost, and capacity. Memory interfacing can improve system efficiency by lowering access latency, increasing data bandwidth, and intelligently managing data across various levels of memory hierarchy.[8]
    - Finally, it should be noted that memory interfacing is essential to the smooth running of computer systems. Creating effective and high-performance computer architectures requires an understanding of the complexities of data transport, addressing modes, memory hierarchy, and overcoming obstacles in memory interface. This research study attempts to provide useful insights into the realm of memory interface and its significance in contemporary computing systems by examining these areas.

II OVERVIEW OF MEMORY INTERFACING

Memory interfacing, which includes the procedures and methods used to provide smooth communication between the central processing unit (CPU) and various memory devices, is a crucial component of computer systems. It acts as the crucial link that facilitates data transfer, archiving, and retrieval, having an impact on a computer system's overall operation and performance. Memory interfacing is crucial in helping to fulfill the rising needs of contemporary computing applications by creating a dependable and effective interface between the CPU and memory modules.[9][10] Memory interfacing in computer systems acts as a link that enables the CPU to communicate with various memory types, including random access memory (RAM), read-only memory (ROM), and peripheral devices. The CPU and memory devices work together to coordinate an interchange of information, instructions, and control signals. Memory interfacing makes ensuring the CPU can access peripheral devices for input and output activities, obtain instructions for execution, and read and write data to memory locations. A computer system would struggle to work at its best without adequate memory interface, which would cause performance bottlenecks, data integrity problems, and decreased overall efficiency.

A. *DATA TRANSFER IN MEMORY INTERFACE*

The core of memory interface is efficient data transfer, which is essential for the smooth running of computer systems. The execution of instructions, storage of data, and information retrieval all depend on the central processing unit's (CPU) ability to transport data quickly and reliably to memory modules. [11]. The intricate details of data transfer within the context of memory interfacing are explored in this section, along with the significance of optimizing data transfer methods to boost system performance and satisfy the expanding demands of contemporary computer applications. We may learn more about the mechanisms that enable effective data transmission and realize the full potential of memory interfacing by examining a variety of techniques and tactics, including direct memory access (DMA), caching systems, and pipelining.

1) IMPORTANCE OF EFFICIENT DATA TRANSFER

For the overall performance and functionality of computer systems, efficient data transfer in memory interfaces is of utmost importance. The system's speed, responsiveness, and efficiency are directly impacted by the central processing unit's (CPU) and memory modules' capacity to transfer data quickly and reliably. In order to enable efficient instruction execution and seamless handling of computational tasks, timely and optimized data transmission ensures that the CPU can access and handle data stored in memory without unnecessarily holding it back. In order to fulfil the rising needs of contemporary computing applications, which frequently entail enormous datasets, real-time processing, and multimedia content, efficient data transport is also essential. Memory interfacing guarantees that the system can perform data-intensive processes successfully by minimizing data transfer latency and maximizing data throughput, enhancing overall user experience and system responsiveness. Additionally, effective data transfer methods assist minimize needless data movement and lower power consumption in memory subsystems, which both contribute to energy efficiency. In conclusion, the significance of effective data transmission in memory interface cannot be understated because it has a direct impact on computer systems' performance, responsiveness, and energy efficiency [12].

2) TECHNIQUES FOR OPTIMIZING DATA TRANSFER

Direct Memory Access (DMA) is a memory interface approach that increases data transfer efficiency by minimizing CPU involvement. In the past, each data transfer between the CPU and memory had to be started by the CPU, using up precious processing cycles. By enabling a peripheral device to directly access the system memory and eschewing the CPU for data transfers, DMA lessens this burden. The CPU may now concentrate on other activities because the peripheral device has taken over control of the memory bus and is performing independent data transfers. When continuous, high-speed data flow is required, such as during disc I/O or network activities, DMA is very advantageous. DMA is an essential part of memory interfacing because it greatly boosts system speed, lowers latency, and increases overall efficiency by taking data transfer duties off the CPU. [13]. Utilizing caching mechanisms is a well-known method for improving data transfer in memory interfaces. Data that is frequently accessed is stored in cache memory, a compact, quick memory located near to the CPU. Caching decreases the requirement for frequent accesses to slower main memory (RAM) by keeping frequently used instructions and data near to the CPU, boosting data transfer rates and lowering latency. Caches take advantage of the "principle of locality," which describes how programmes prefer to retrieve information and instructions that are nearby in time or space. Different cache architectures use various methods for controlling data storage and retrieval, including direct-mapped, set-associative, and completely associative caches. By maximizing the concept of locality and reducing expensive visits to slower memory tiers, memory interfacing can achieve significant performance increases through good cache management and sophisticated caching algorithms.[14]. Pipelining is a memory interface technique used to increase the effectiveness of data flow between the CPU and memory. Pipelining enables overlapping processes and enhanced throughput by segmenting the data transfer process into smaller phases or tasks and carrying them out concurrently. According to this method, the CPU breaks the data transmission process into a series of sequential steps, and each step is handled by a separate pipeline stage. The next step starts processing the following batch of data as soon as the previous one is finished with it. The CPU is used to its fullest potential during this overlapping activity, which increases the rate of data transport overall. Pipelining is particularly useful in situations where several data transfers happen quickly because it makes efficient use of parallelism and the system resources that are at hand. Memory interfacing can produce outstanding improvements in data throughput and system performance by implementing pipelining techniques. [15]

B. *ADDRESSING MODES IN MEMORY INTERFACING*

Through their influence on how the central processing unit (CPU) accesses and retrieves data from memory, addressing modes play a crucial role in memory interface. These modes specify the precise addressing strategies and procedures used to locate data efficiently in memory. Addressing modes are crucial because they permit flexible and adaptable memory access, allowing the CPU to work with various data structures and carry out memory operations quickly. Researchers and designers can choose the best addressing mode for a specific computing activity by having a thorough understanding of the importance of addressing modes in memory interfacing. This will ensure optimal memory utilization and efficient data retrieval inside the computer system. [16]. A fundamental form of memory interface addressing is known as "direct addressing," in which the CPU directly provides the memory address of the information or instruction to be accessed. In this mode, the command directly states the address of the memory region, enabling rapid and simple access. When the CPU uses direct addressing to retrieve or store data, it communicates directly with the memory module corresponding to the given address. The memory access procedure is made easier by this direct connection, which eliminates the need for intricate calculations or further memory lookups. Direct addressing has certain drawbacks, too, one of which is that it limits the CPU's ability to dynamically access data from various memory regions. Direct addressing, however, is still a popular and effective addressing method in many computing systems, particularly when working with fixed data or regularly utilized memory regions. [17][18]. Indirect addressing is a crucial type of addressing mode in memory interfacing that enables flexible and dynamic access to memory locations. Unlike direct addressing, which involves specifying the exact memory address for data retrieval or storage, indirect addressing employs a pointer or reference to access the desired memory location. This approach offers significant advantages in terms of versatility and adaptability, allowing for the manipulation of memory addresses during program execution. By utilizing indirect addressing, programmers can efficiently navigate through data structures, arrays, and linked lists, making it a valuable tool for handling complex data scenarios. Indirect addressing also facilitates the implementation of efficient algorithms and enables dynamic memory allocation, empowering programmers to allocate and manage memory resources dynamically as per the runtime requirements of the program. Overall, the versatility and dynamic nature of indirect addressing make it an indispensable addressing mode in memory interfacing, providing flexibility and power to optimize memory access in various computing applications.. Indexed addressing is a versatile technique employed in memory interfacing that facilitates efficient data access and manipulation. It involves using an index or offset value to calculate the memory address of a desired data element. By combining the base address of a memory block with an offset value stored in a register, indexed addressing allows for quick and convenient retrieval or modification of data elements within an array or a data structure. This addressing mode is particularly valuable in scenarios where repetitive or sequential data processing is required, as it enables rapid access to consecutive memory locations without the need for explicit address calculations in each iteration. By leveraging indexed addressing, memory interfacing achieves improved data organization and access patterns, thereby enhancing the performance and flexibility of computing systems. [19]

C. *MEMORY HIERARCHY AND MEMORY INTERFACING*

The foundation of memory interfacing, which aims to improve the effectiveness and performance of computer systems, is the idea of memory hierarchy. In order to balance performance and cost-effectiveness, memory hierarchy entails classifying various levels or layers of memory according to their speed, cost, and capacity. Memory hierarchy serves two purposes: first, to decrease access latency by placing frequently accessed data in faster, smaller memory levels closer to the CPU, such as cache memory; and second, to increase storage capacity by using slower, more economical memory levels, such as main memory (RAM) and secondary storage devices like hard drives or solid-state drives (SSDs). Memory interfacing guarantees that the CPU may access frequently used data quickly while preserving a bigger pool of memory for storing substantial datasets and programmes by intelligently managing data throughout the memory hierarchy. The system's overall performance is optimized by the hierarchical organization of memory tiers, which makes it possible to retrieve data quickly, respond more quickly, and use resources more efficiently. [20]. A crucial part of a computer system's memory structure is cache memory. Cache memory, which sits in between the CPU and main memory (RAM), functions as a quick buffer for frequently accessed information. Its goal is to reduce the average access time and improve system performance by filling the speed gap between the fast CPU and the relatively slower main memory. Cache memory considerably reduces the latency involved in fetching data from the main memory by keeping copies of frequently used data and instructions closer to the CPU. To maximize the efficiency of data caching, it uses the localization principle to take advantage of the temporal and spatial features of memory access patterns. L1, L2, and L3 caches, which progressively offer increased storage capacity and access speed as they advance closer to the CPU, are examples of different levels of cache memory. Optimizing memory interfacing within the larger memory hierarchy ultimately depends on effective cache memory management, proper caching algorithms, and high cache hit rates. [21]. A computer system's main memory, sometimes referred to as RAM (Random Access Memory), occupies a crucial place in the memory hierarchy. It offers quick and temporary storage for both data and instructions while acting as the CPU's immediate workspace. Main memory is known for being volatile, which means that when power is turned off, its contents are lost. This section emphasizes main memory's significance as a major store medium for actively used data and programme instructions while concentrating on its function in memory interfacing. Information may be accessed and retrieved quickly thanks to the role main memory plays as a link between the CPU and secondary storage. Main memory plays a crucial role in maximizing system performance by lowering latency and facilitating faster data transmission between the CPU and other memory modules thanks to its quicker access times compared to secondary storage devices like hard drives or solid-state drives. [22][23].

FIGURE 4: Random Access Memory Source:[22]

Secondary storage plays a crucial role in memory interface in the hierarchy of memory systems. Contrary to primary memory (such as RAM), which is speedier but more volatile, secondary storage devices offer non-volatile, high-capacity storage for long-term data retention. Hard drives and solid-state drives (SSDs) are examples of secondary storage devices that are crucial for keeping data safe after a single computing session since they can store large amounts of data even when the power is off. When compared to primary memory, these devices have slower access times but bigger store capacities. The fact that secondary storage devices act as repositories for operating systems, applications, user files, and other permanent data, despite their slower speed, greatly contributes to memory interfacing.

FIGURE 5: Secondary Storage Devices Source: [23]

The interaction between memory interfacing and secondary storage extends beyond data persistence and storage capacity considerations. Efficient management of secondary storage is crucial for optimizing data retrieval and input/output (I/O) operations. Techniques such as caching and buffering are employed to mitigate the relatively slower access times of secondary storage devices. Caching mechanisms, such as disk caches or buffer pools, store frequently accessed data in faster primary memory to reduce the need for accessing secondary storage repeatedly. This strategy minimizes I/O latency and enhances overall system performance. Additionally, sophisticated file systems and storage management algorithms are employed to organize and optimize data placement on secondary storage devices, further improving data access times and facilitating seamless integration with memory interfacing. Therefore, the effective utilization and management of secondary storage devices are integral components of memory interfacing, enabling efficient and reliable data storage, retrieval, and I/O operations in computer systems.

Memory interfacing heavily relies on memory hierarchy, a basic idea in computer design. It entails dividing memory systems into a number of tiers or layers, each with unique properties such as price, capacity, and speed. As it directly affects the effectiveness, performance, and general system behavior, memory hierarchy has a significant impact on memory interfacing.. The cache memory, a tiny but incredibly quick storage device placed closer to the CPU, is at the centre of the memory hierarchy. In order to speed up access to frequently used instructions and data, the cache memory serves as a buffer between the CPU and the main memory (RAM). Its presence and its use can significantly decrease the latency associated with accessing data from main memory, improving memory interfacing by accelerating data transfer between the CPU and RAM. The memory hierarchy also includes primary storage components like hard drives and solid-state drives (SSDs), in addition to the cache memory and the main memory itself. The access times, prices, and capacities of the memory hierarchy levels vary. In order to ensure that the CPU can effectively obtain data from the appropriate memory level based on its proximity and access speed, effective memory interfacing entails coordinating the data flow between these various levels. By understanding and effectively managing the memory hierarchy, memory interfacing can be optimized, allowing for improved system performance, reduced latency, and efficient utilization of memory resources.

D. *MEMORY INTERFACING WITH 8085*

An essential component of the Intel 8085 microprocessor's functionality is memory interfacing, which enables the CPU to connect with and access information stored in external memory devices. The 8085 microprocessor uses a simple memory interface strategy, transferring data between the CPU and external memory via address and data buses. The 16-bit address bus of the 8085 microprocessor enables it to address up to 64 KB of memory. By using this address bus, the CPU may tell the external memory devices the precise position from which it wishes to read or write data by sending memory addresses to them. The actual data being transmitted between the CPU and the memory devices is carried via the 8085's 8-bit data bus. The address latch enable (ALE), read (RD), and write (WR) signals are three control signals that the 8085 microprocessor uses to simplify memory interface. In the first clock cycle of each machine cycle, the ALE signal is in charge of latching the bottom byte of the 16-bit address onto the address bus. The memory address that the CPU is now accessing is represented by this latched address. It is possible to tell whether a memory read or write operation is being carried out using the RD and WR signals. The RD signal indicates that the CPU is reading data from the external memory when it is active. The WR signal, on the other hand, shows that the CPU is writing data to the external memory when it is active. By synchronizing the data transmission between the CPU and the memory devices, these control signals guarantee proper coordination and communication. Various memory and input/output (I/O) addressing modes, including direct addressing, indirect addressing, and instantaneous addressing, are also included in the 8085 CPU. These addressing modes offer flexibility in memory interfacing by enabling the CPU to access particular memory regions or I/O ports using various addressing strategies. The address and data buses, along with control signals, are used for memory interfacing in the Intel 8085 microprocessor to establish connection between the CPU and external memory devices. This makes it easier for data and instructions to be sent, which helps the microprocessor run applications and carry out other operations quickly. [24]

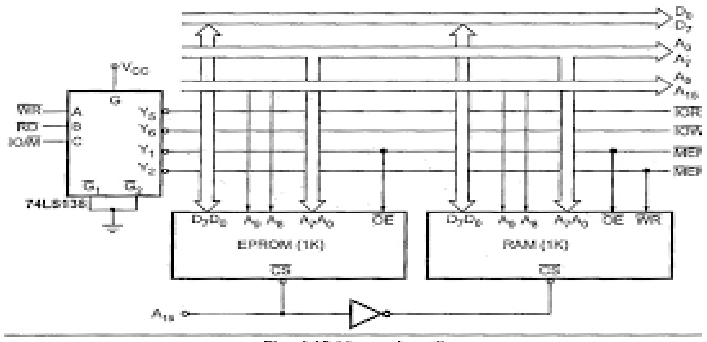


FIGURE 6: 8085 INTERFACING Source: [24]

E. MEMORY INTERFACING WITH 8086

The 8086 microprocessors, a popular 16-bit microprocessor created by Intel, uses memory interfacing to connect and communicate with multiple memory devices. Because the memory address space is divided into segments, each of which may hold up to 64KB of data, the 8086 uses a segmented memory paradigm. The address bus, data bus, and control signals are used in the 8086 to interface with memory. The memory address, which designates the position from which data must be read or written, is transmitted from the CPU to the memory devices through the address bus. Information may be transferred because the data bus connects the CPU and memory with the real data. Control signals regulate memory operations' timing and sequencing, maintaining correct synchronization between the CPU and memory components. Input/output (I/O) devices, random access memory (RAM), read-only memory (ROM), and other memory types are all supported by the 8086. The address decoding logic, which connects the memory devices to the 8086 generally, identifies the precise memory regions the CPU accesses based on the memory address provided. A memory segmentation mechanism is used in the 8086 to simplify memory interfacing. Each segment in the memory address space is designated by a segment register in the CPU. The real physical memory address is created by fusing the offset address and segment base address from the segment register. The 8086 may address a broader memory region than the 64KB limit of a single segment thanks to this segmentation strategy. Various addressing modes, including direct addressing, indirect addressing, and indexed addressing, are also supported by the 8086. These addressing modes enable effective memory interfacing activities by allowing flexible access to and manipulation of data stored in memory.The connection, communication, and synchronization between the CPU and memory devices using the address bus, data bus, and control signals comprise memory interfacing in the 8086 microprocessor. The 8086 uses a segmented memory model and supports a range of memory devices and addressing patterns, making memory access and data transfer quick, easy, and adaptable. [25]

III CONSTRAINTS IN MEMORY INTERFACING AND PROPOSED SOLUTIONS

In the context of memory interfacing, compatibility issues refer to the challenges that arise when different components, such as the CPU and memory modules, have differing specifications or protocols. These compatibility issues can hinder the seamless communication and data transfer between these components. Here, we'll delve into some common compatibility issues and propose potential solutions: Voltage and Timing Compatibility: Memory modules often have specific voltage requirements and timing constraints that must be met for proper operation. Incompatibilities in voltage levels or timing signals can result in data corruption or system instability. To address these issues, memory controllers and interfaces need to support the required voltage levels and timing specifications of the memory modules. Additionally, standardized memory interface protocols, such as DDR (Double Data Rate) standards, help ensure compatibility by defining voltage levels, signaling schemes, and timing parameters .Protocol Compatibility: Different memory modules may utilize different protocols for communication and data transfer. For example, DDR3 and DDR4 memory modules employ different signaling schemes and command protocols. Incompatibilities in protocols can prevent proper data exchange between the memory and CPU. To overcome these challenges, memory controllers and interface designs must be compatible with the specific protocols employed by the memory modules. This involves implementing the necessary signaling circuitry and command decoding logic to ensure accurate protocol handling. Physical Form Factor Compatibility: Memory modules come in various physical form factors, such as DIMMs (Dual Inline Memory Modules) and SODIMMs (Small Outline Dual Inline Memory Modules). These form factors dictate the physical and electrical characteristics of the memory modules. Incompatibilities in form factors can arise when the memory module's physical dimensions or pin configurations do not align with the memory slots on the motherboard. To ensure compatibility, it is important to select memory modules that adhere to the appropriate form factor for the system and verify compatibility with the motherboard specifications. Capacity and Speed Compatibility: In certain cases, compatibility issues may arise when the system's memory controller is not capable of supporting the capacity or speed of the memory modules. For instance, if the memory controller is limited to a maximum memory capacity or does not support high-speed memory modules, compatibility issues can occur. Upgrading or selecting memory modules that are within the supported capacity and speed ranges of the memory controller can help mitigate these compatibility concerns. To address these compatibility issues, it is crucial to perform thorough compatibility testing and verification during the design and implementation of memory interfaces. This includes ensuring adherence to relevant industry standards, utilizing compatible components, and verifying the compatibility of memory modules with the system's memory controller specifications. By addressing compatibility issues in memory interfacing, system designers and engineers can establish reliable and efficient communication between the CPU and memory modules, ensuring seamless data transfer and optimal system performance.

IV CONCLUSION

In conclusion, the investigation of memory interfaces is crucial to the development and use of effective computer systems. This study work has illuminated the crucial elements of memory interface by investigating data transport, addressing techniques, memory hierarchy, and the difficulties involved. It is clear that establishing high-performance computer architectures requires effective data transmission mechanisms, optimised addressing modes, and the tactical application of memory hierarchy. Successful memory interface also requires addressing compatibility difficulties, taking temporal restrictions, electrical considerations, and adhering to protocols and standards into account. The understanding of memory interface will grow as memory technologies continue to advance, spurring the creation of increasingly complex and potent computing systems. By accumulating knowledge about memory interfacing, scientists and engineers are better prepared to handle the difficulties and take advantage of the opportunities given by memory systems, thus advancing the discipline and boosting computing power.

ACKNOWLEDGMENT

The authors acknowledge the management of Afe Babalola University for the financial support received in the publication of this article.

REFERENCES

1. Stokes, J. (2007). *Inside the machine: an illustrated introduction to microprocessors and computer architecture*. No starch press.
2. Godse, A. P., & Godse, D. A. (2020). *Microprocessor and Interfacing*. Technical Publications.
3. Wachter, S., Polyushkin, D. K., Bethge, O., & Mueller, T. (2017). A microprocessor based on a two-dimensional semiconductor. *Nature communications*, *8*(1), 14948.
4. Gelsinger, P. P. (2001, February). Microprocessors for the new millennium: Challenges, opportunities, and new frontiers. In *2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No. 01CH37177)* (pp. 22-25). IEEE.
5. Jacob, J. A., & Chow, P. (1999, February). Memory interfacing and instruction specification for reconfigurable processors. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays* (pp. 145-154).
6. Godse, D. A. (2008). *Microprocessor & Microcontroller*. Technical publications.
7. Saini, M., Kumar, A., & Shankar, V. G. (2020). A study of microprocessor systems using RAMD approach. *Life Cycle Reliability and Safety Engineering*, *9*, 181-194.
8. Tsai, P. A., Gan, Y. L., & Sanchez, D. (2018, October). Rethinking the memory hierarchy for modern languages. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (pp. 203-216). IEEE.
9. Rath, K., Bose, B., & Johnson, S. D. (1993, October). Derivation of a DRAM memory interface by sequential decomposition. In *Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93* (pp. 438-441). IEEE.
10. Kumar, N. S., Saravanan, M., Jeevananthan, S., & Shah, S. K. (2012). Microprocessors and interfacing.
11. Behnam, P., & Bojnordi, M. N. (2020). STFL-DDR: Improving the energy-efficiency of memory interface. *IEEE Transactions on Computers*, *69*(12), 1823-1834.
12. Okuno, H., Inoguchi, Y., & Horiguchi, S. (2001). A new network interface with distributed memory.
13. Recio, R., Metzler, B., Culley, P., Hilland, J., & Garcia, D. (2007). *A remote direct memory access protocol specification* (No. rfc5040).
14. Hu, Z., Zheng, Z., Wang, T., Song, L., & Li, X. (2016). Caching as a service: Small-cell caching mechanism design for service providers. *IEEE Transactions on Wireless Communications*, *15*(10), 6992-7004.
15. Dalvi, N. N., Sanghai, S. K., Roy, P., & Sudarshan, S. (2001, May). Pipelining in multi-query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 59-70).
16. Kumar, K. S., Rao, Y. R., & Manjunathachari, K. Address Mapping In Content Addressable Memory Interface with A Low Power Approach.
17. Kumar, K. S., Rao, Y. R., & Manjunathachari, K. Content Addressable Memory for Multi Page Memory Interface.
18. Milenkovic, A. (2007). Addressing: Direct and Indirect. *Wiley Encyclopedia of Computer Science and Engineering*, 1-11.
19. Dhaliwal, R. S. (2022). On Addressability, or What Even Is Computation?. *Critical Inquiry*, *49*(1), 1-27.

20. Balasubramonian, R., Albonesi, D., Buyuktosunoglu, A., & Dwarkadas, S. (2000, December). Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture* (pp. 245-257).
21. Przybylski, S. A. (1990). *Cache and memory hierarchy design: a performance directed approach*. Morgan Kaufmann.
22. Senni, S., Torres, L., Sassatelli, G., Bukto, A., & Mussard, B. (2014, July). Exploration of magnetic ram based memory hierarchy for multicore architecture. In *2014 IEEE Computer Society Annual Symposium on VLSI* (pp. 248-251). IEEE.
23. Daley, R. C., & Neumann, P. G. (1965, November). A general-purpose file system for secondary storage. In *Proceedings of the November 30--December 1, 1965, fall joint computer conference, part I* (pp. 213-229).
24. Bakrola, V. (2014). Development of 8085 microprocessor based output port and implementation using real components.
25. Triebel, W., & Singh, A. The 8088 and 8086 microprocessors: Programming Interfacing, Software.