Code No.: R22CS58351PE | R22 | H.T.No. | | | 8 | R | | | | | | |

CMR ENGINEERING COLLEGE: : HYDERABAD
UGC AUTONOMOUS
II–M.TECH–I–Semester End Examinations (Regular) - January- 2026
ADVANCED COMPUTER ARCHITECTURE (PE-V)
(CSE)

[Time: 3 Hours] **[Max. Marks: 60]**

Note: This question paper contains two parts A and B.
Part A is compulsory which carries 10 marks. Answer all questions in Part A.
Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

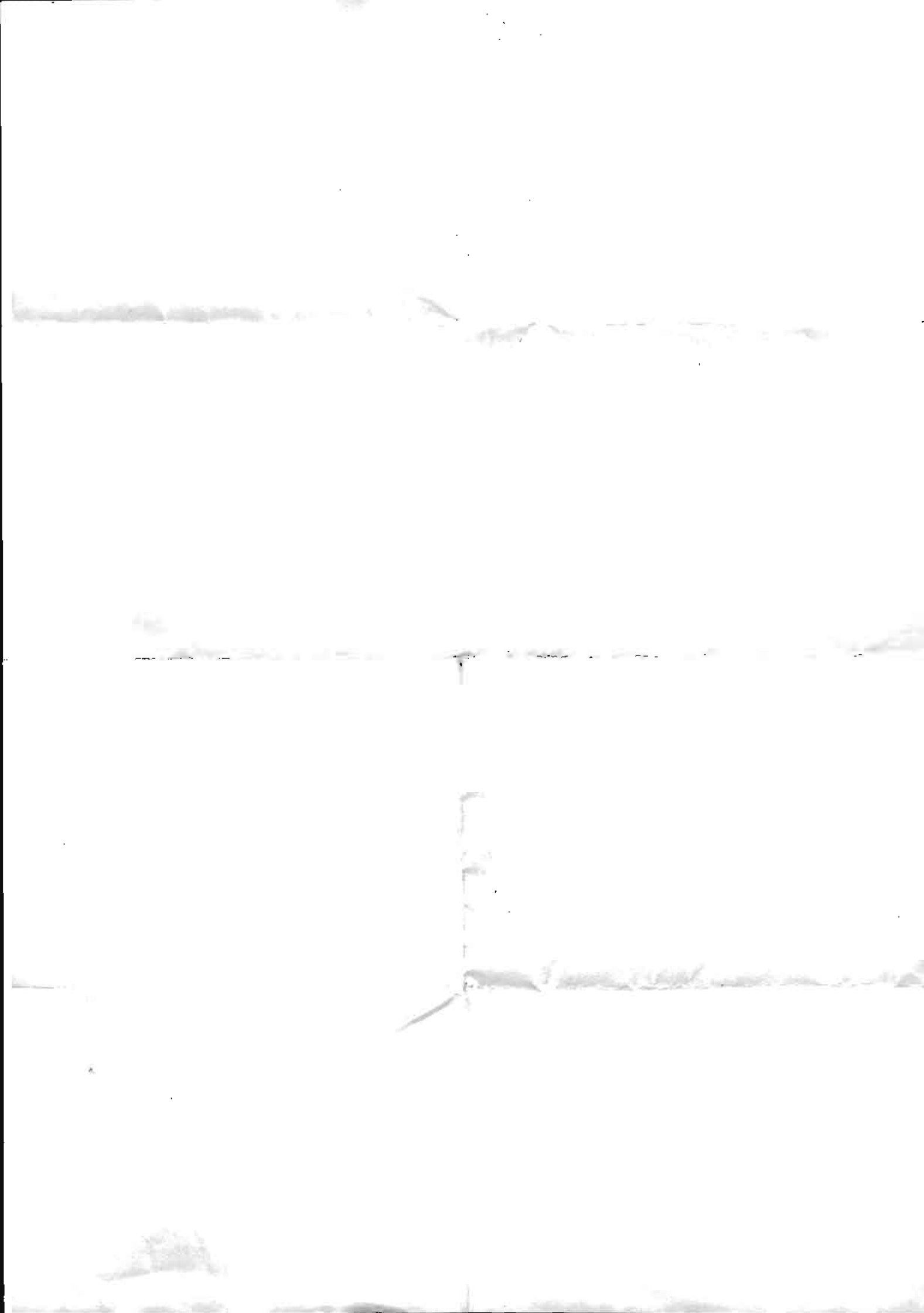
PART-A **(10 Marks)**

1. a)	Difference between multiprocessors and multicomputers.	[1M]
b)	What is the main feature of multivector processors?	[1M]
c)	What is superscalar technology?	[1M]
d)	Explain the role of memory hierarchy in parallel processing.	[1M]
e)	What is arithmetic pipeline design?	[1M]
f)	Mention one challenge of shared memory systems.	[1M]
g)	Mention one difference between SIMD and multivector processors.	[1M]
h)	What is a compound vector processor?	[1M]
i)	What is a multithreaded processor?	[1M]
j)	List one advantage of dataflow architecture.	[1M]

PART-B **(50 Marks)**

2.	Explain program partitioning and scheduling in parallel computing.	[10M]
	OR	
3.	Analyze the role of system interconnection architectures in parallel systems.	[10M]
4.	Discuss the scalability challenges in designing memory hierarchies.	[10M]
	OR	
5.	Explain how performance metrics are used to evaluate scalable systems.	[10M]
6.	Explain the role of superscalar pipeline design in improving performance.	[10M]
	OR	
7.	Discuss the significance of weak consistency models in shared memory systems.	[10M]
8.	Explain the role of synchronization mechanisms in multiprocessor systems.	[10M]
	OR	
9.	Compare SIMD and multivector processors in terms of performance.	[10M]
10.	Explain latency-hiding techniques with real-world examples.	[10M]
	OR	
11.	Discuss the principles of multithreading in parallel systems.	[10M]

****



II m.Tech I Sem End Examinations (Regular) Jan-2026

R22CS5835IPE

PART - A

1 a) Difference between multiprocessors and multicomputers.

- multiprocessors :- multiple processors share a common memory and operate under a single OS
- multicomputers :- processors have private memory and communicate via message passing.

b) what is the main feature of multivector processors?

- Ability to process multiple vector instructions simultaneously, improving data-level parallelism.

c) what is superscalar technology?

- A processor design that can issue and execute multiple instructions per clock cycle using parallel execution units.

d) Explain the role of memory hierarchy in parallel processing

- Reduces memory access latency and bandwidth bottlenecks thereby improving performance and scalability.

e) what is arithmetic pipeline design?

- A technique where arithmetic operations are divided into stages, allowing overlapped execution of multiple operations.

f) mention one challenge of shared memory systems.

- cache coherence problem due to multiple processors accessing shared data.

g) mention one difference between SIMD and multivector processors.

- SIMD : single instruction operates on multiple data elements.
- Multivector : can execute multiple vector instructions concurrently.

h) what is a compound vector processor ?

- A processor that combines vector processing with scalar and pipeline units for higher performance.

i) what is a multithreaded processor ?

- A processor that supports multiple threads of execution to improve utilization and hide latency.

j) List one advantage of dataflow architecture ?

- Enables maximum parallelism by executing instructions as soon as operands are available.

PART - B

② Explain program partitioning and scheduling in parallel computing.

Program partitioning : involves dividing a program into smaller pieces, or "grains", that can potentially be executed in parallel. This can be done manually be executed in parallel. This can be done manually by a programmer or automatically segments that are independent in terms of data, control, and resources. The size of these grains, or "granularity" can vary from fine (instruction level) to coarse (entire jobs), with finer grains generally offering higher potential parallelism but also higher communication and scheduling overhead.

Scheduling :- is the process of assigning these partitioned tasks to available processors or resources to minimize the overall execution time to the program. This involves making trade-offs between exploiting parallelism and the overhead associated with communic-ation and synchronization between tasks.

static scheduling :- Decisions about which tasks run on which Processors are made at compile time when program characteristics are known in advance.

Dynamic scheduling :- Decisions are made at run time which is useful when program behaviour is less predictable though this adds to the scheduling overhead.

3) System interconnection architectures are crucial for communic -ation and data sharing among processors in a parallel system.

system interconnection architectures define how different processor and memory modules are connected in a parallel computing system. Their primary roles include.

- Communication :- They provide the pathways for processors to exchange data and synchronization signals.
- scalability :- The architecture determines how easily the system can be expanded with more processors. some archite -ctures, like a bus, have limited scalability, while others, like a mesh or torus, can scale to thousands of nodes.
- performance :- The architecture's topology affects communicatio latency and bandwidth. A well-designed architecture minimize communication bottlenecks, ensuring that the time spent on data exchange does not dominate computation time.
- Reliability :- The design can incorporate redundant paths to ensure that the failure of a single link or node does not bring down the entire system.

Cost and complexity :- Different architectures involves varying levels of wiring complexity and cost. A simple bus architecture is cheap but has performance limitations, while a crossbar switch offers high performance at a much higher cost and complexity.

4 The primary scalability challenges in designing memory hierarchies revolve around balancing capacity, latency bandwidth, and power consumption as systems grow in size and complexity.

- Latency vs. capacity Trade-off :- As memory hierarchies scale, the gap between the fastest (smallest e.g. L1 cache) and slowest (largest, eg. main memory disk) levels increases.
- Bandwidth limitations :- scaling up the numbers of cores or processors increases the demand for data transfer. the bandwidth of interconnects between different levels of the hierarchy can become a significant bottleneck, limiting overall system performance.
- cache coherence and consistency :- in multi-processor systems maintaining a consistent view of data across multiple caches is complex.
- Power and Energy consumption :- scaling memory hierarchies often involves adding more memory modules and increasing the frequency of data access.
- cost and complexity : Designing and implementing a large, efficient mememory hierarchy is expensive and complex.
- physical constraints :- As systems scale, physical constraints such as pin count limitations on processors board space and cooling requirements become increasingly difficult to manage.

5)

Evaluating scalable systems involves using specific metrics to understand their behaviour as the workload or size of the system increases. key performance metrics include.

- Throughput :- This measures the number of operations or requests processed per unit of time. A scalable system should see throughput increase proportionally with added resources or at least maintain a high level as the load grows.
- Latency/Response Time :- This measure the time taken to respond to a single request. In a scalable system, the goal is often to keep latency relatively constant, or within acceptable bounds, even as throughput increases.
- Resource utilization :- This tracks how effectively resources are being used. High utilization without performance degradation is a sign of good scalability.
- scalability metrics :- Specific metrics like "requests per server" or "cost per transaction" help determine the efficiency of scaling. Linear scalability, where performance increases linearly with added resources is the ideal goal.
- Availability and Reliability :- while not strictly performance these are crucial for scalable systems metrics ensure that scaling does not introduce single points of failure and that the system remains accessible and functional.

⑥ A supercalar pipeline design improve processor performance by allowing the cpu to initiate and execute multiple instruction in parallel during a single clock cycle. while traditional pipelining overlaps instructions, superscalar architecture adds spatial parallelism by duplicating hardware resource to process several instructions concurrently, thereby achieving an instruction throughput of more than one instruction per cycle.

1. Increased Instruction Throughput (IPC) :-
The primary role of a superscalar design is to boost the number of instructions completed per cycle (IPC). A conventional scalar processor can, at best, complete one instruction per cycle (IPC = 1).

2. parallel Execution via multiple functional units :-
superscalar processors contain multiple, independent functional units. The processor fetches a group of instructions and if no independencies exist.

3. Exploiting instruction - Level parallelism (ILP) :- superscalar architectures are designed to automatically identify independent instructions within a sequential program stream.

4. Advanced scheduling and Reduced stalls.
To maximize performance modern superscalar designs often include.
- out of order execution :-
- Register renaming.

5. synergy with other Techniques.
- Pipelining
- Branch prediction.

(7) Weak consistency models are a cornerstone of modern advanced computer architecture.

significance of weak consistency models

The primary significance lies in trading off immediate global data consistency performance gains and enhanced scalability.

- Latency hiding :- in large-scale multiprocessors, memory access is a bottleneck.
- performance optimization :- By reducing constraints on instruction ordering, weak models allow for more.
- enhanced scalability :- As the number of cores increases the overhead of maintaing strict synchronization.
- support for modern Architectures :- modern processor architectures like ARM and RISC.

Weak consistency models :- weak models differentiate between data accesses and synchronization accesses.

1) weak ordering (WO) : programs are expected to use synchronization primitives to protect shared data.

2) Relases consistency (RC) : A more relaxed version of weak ordering. it divides.

3. Eventual consistency :- A often used in distributed systems where if no new updates are made to a data item ; all replicas will eventually converge to the same value.

8. Synchronization mechanisms in multiprocessor systems ensure orderly access to shared resources, preventing data corruption (race conditions) by co-ordinating multiple threads/cores, guarnteeing data consistency, managing resources allocation, and avoiding deadlocks, using tools like locks, semaphores, and hardware support (e.g., cache coherence) for reliable concurrent execution.

Key Roles of synchronization

1. prevent Race conditions & ensure data consistency:
-> When multiple processors try to read/write the same data simultaneously, results are unpredicatable (race condition).

2. Manage Resource Allocation:
-> Fairly distributes access to limited hardware (printer, memory) or software resources among competing processes/threads.

3. Establish Orderly Execution:
-> Ensure processes run in a specific sequence, especial when one depends on another's output (e.g., producer-consumer problem).

4. Avoid Deadlocks:
-> prevents situations where processes get stuck waiting for resources held by each other, critical for system stability.

9 SIMD (Single Instruction, Multiple Data) and multivector (or traditional vector) processors both enhance performance through data-level parallelism, but they differ significantly in flexibility, scalability, and efficiency.

Feature	SIMD (short vector/ packed SIMD)	Multivector (Traditional vector)
Data length	Fixed-width (e.g., 128, 256, 512 bits)	Flexible / variable-length (set via VL register.
performance	High, but limited by fixed width	Higher for large, streaming data sets.
Ideal workload	Multimedia, gaming, DSP	Scientific computing, simulations.
Efficiency	Lower (high instruction overhead)	Higher (due to vector chaining / fewer fetches)
Scalability	Lower (requires ISA change to scale)	High (easy to scale vector length)
Memory Access	Limited, often requires contiguous data	High (gather-scatter, strided access)

10. Latency-hiding techniques mask slow operations (like memory access) by doing useful work concurrently, preventing processor stalls, using strategies like Multithreading (switching threads during waits, common in GPUs/Network processors), prefetching (bringing data ahead of time), caching (keeping data nearby), and Relaxed consistency (allowing reordering), making systems feel faster by overlapping computation and communication, as seen in GPU's processing many threads or CPUs using SMT.

Key Techniques:

1) Multithreading / simultaneous Multithreading (SMT):
=> How it works: When one thread stalls (e.g., waiting for memory), the processor switches to another ready thread, keeping execution units busy.

2) Caching (Memory Hierarchy):
=> How it works: stores frequently used data in faster, closer memory levels (L1, L2, L3 caches) to avoid the much slower main memory.

3) prefetching:
=> How it works: Hardware or software predicts needed data/instructions and fetches them into the cache before they're explicitly requested, overlapping the fetch with other work.

4) Relaxed Memory Consistency / out-of-order Execution:
=> How it works: Allows processors to reorder instruction (if dependencies aren't violated) and buffers memory request, overlapping independent operations and buffering results.

5). Software-Managed prefetching & scheduling:
=> How it works: compilers insert explicit prefetch instructions or reorder code to place memory accesses strategically, hiding latency.

11. Multithreading in parallel systems divides processes into lightweight threads that run concurrently, sharing resources like memory, enabling true parallelism on multi-core CPUs for better performance, responsiveness and resources efficiency, but requires managing complexities like race conditions and deadlocks through synchronization.

Core principles

1) Thread as a lightweight process: A thread is a basic unit of CPU utilization, a subset of a process, sharing its code, data, and resources (like open files) but having its own stack, registers, and program counter.

2) Resource sharing: Threads within the same process share the process's memory space and resources, reducing overhead compared to creating separate processes.

3) Concurrency vs. parallelism.

1. Concurrency (single core): The CPU rapidly switches b/w threads (time-slicing), making them appear to run simultaneously.

2. Parallelism (multi-core): Multiple threads genuinely execute at the exact same time on different processor cores, achieving true simultaneous execution.

4) Performance & Responsiveness: Multithreading improves responsiveness (e.g., a UI thread stays active while another does work) and performance (e.g. complex calculations divided across cores).