Computer Organization and Architecture

Exercise and Exam-Oriented Questions with Answers

Chapter 7

Vector Processing and Array Processor

Covers

Very Short Answer Type Questions (VSAQ)
Short Answer Type Questions (SAQ)
Long Answer Type Questions (LAQ) — Full Model Answers

For Engineering Students — Examination Preparation

Section A — Very Short Answer Type Questions

(1–2 marks each) — suitable for quick revision and viva-voce preparation.

Q1. What is a vector processor?

Ans. A vector processor is a computer specially organised to process an entire vector (ordered set of data elements) using a single instruction, instead of processing one data element at a time as in a scalar processor.

Q2. Define vector processing.

Ans. Vector processing is a mode of computation in which a single instruction operates simultaneously, or in a heavily pipelined and overlapped manner, on an ordered set of data elements (a vector) rather than on one scalar element at a time.

Q3. Name any three techniques used in vector processing.

Ans. Pipelining of the arithmetic unit, use of multiple parallel functional units, and use of high-speed vector registers (memory interleaving and chaining are also acceptable answers).

Q4. What is chaining in vector processing?

Ans. Chaining is the technique of feeding the output of one pipelined functional unit directly into a second pipelined functional unit, element by element, as soon as each result becomes available, instead of waiting for the entire first vector operation to complete.

Q5. Why is main memory interleaved in vector computers?

Ans. Main memory is organised into multiple interleaved modules so that successive, usually consecutive, vector elements can be fetched from different modules in an overlapped manner, supplying operands at the rate the pipeline consumes them.

Q6. State two fields normally present in a vector instruction format.

Ans. Operation code (opcode) and vector length (N) are two typical fields; other acceptable fields are the base addresses of the source and destination vectors, and the address increment.

Q7. What is an array processor?

Ans. An array processor is a computer that uses a single control unit to broadcast one instruction simultaneously to a large number of identical processing elements, each of which executes that instruction on its own local data.

Q8. Under Flynn's classification, where does an array processor fall?

Ans. An array processor is classified as SIMD — Single Instruction stream, Multiple Data stream.

Q9. What is the role of the control unit in an SIMD array processor?

Ans. The control unit fetches and decodes each program instruction and broadcasts the decoded instruction (and any common operands) to all the processing elements over a control/broadcast bus.

Q10. What is an attached array processor?

Ans. An attached array processor is an auxiliary high-speed processing unit connected to a general-purpose host computer, used to accelerate vector/array-intensive computations while the host handles overall program control and I/O.

Q11. Give one example architecture of a true SIMD array processor.

Ans. ILLIAC IV is a classic example of a true, stand-alone SIMD array processor (the Connection Machine, CM-1/CM-2, is also an acceptable example).

Q12. What is an associative (content-addressable) array processor?

Ans. It is a type of SIMD array processor in which each processing element's local memory is content-addressable, so data is retrieved by matching content against a broadcast search pattern rather than by specifying a memory address.

Section B — Short Answer Type Questions

(5 marks each) — expected answer length: 6–10 lines.

Q1. Distinguish between a scalar processor and a vector processor.

Answer:

A scalar processor executes instructions that operate on one pair of data elements at a time; to apply the same operation to n elements, the program must run through a loop n times, incurring loop-control and instruction fetch/decode overhead on every pass. A vector processor, in contrast, executes a single vector instruction that operates on an entire ordered set (vector) of n elements, using pipelined or parallel arithmetic hardware. This removes the repeated loop-control overhead and, once the pipeline is full, produces one result every clock cycle, giving a much higher throughput for large, regular data sets than an equivalent scalar loop.

Q2. Explain the role of pipelining in vector processing.

Answer:

The arithmetic unit used for vector operations (such as a floating-point adder or multiplier) is divided into a sequence of independent stages or segments, each carrying out one small part of the overall computation — for example, comparing exponents, aligning mantissas, performing the addition, and normalising the result. Successive pairs of vector elements are fed into the pipeline on every clock cycle, so that while one pair is in the final normalisation stage, later pairs are simultaneously in earlier stages. Once the pipeline is full, a new result emerges every clock cycle instead of once every several cycles, which is what gives vector processing its principal speed advantage for long vectors.

Q3. What is chaining? Explain its advantage with an example.

Answer:

Chaining connects the output of one pipelined vector functional unit directly to the input of another pipelined functional unit, so that each result element is forwarded onward as soon as it is produced, rather than the whole first vector operation having to complete and be written back to memory or registers before the second operation can start. For example, if a program must compute D = (A + B) * C as vector operations, the elements of the sum (A + B) can be chained straight into the multiply pipeline as they are generated, so the addition and multiplication proceed together in an overlapped manner. This significantly reduces the total execution time of a sequence of data-dependent vector instructions compared with executing them strictly one after another.

Q4. Explain the general format of a vector processing instruction.

Answer:

A vector instruction must specify not just an operation but the location and extent of entire vectors. A typical memory-to-memory format contains: an operation code identifying the action to perform (add, multiply, load, etc.); the base address of the first source vector; the base address of the second source vector, where applicable; the base address of the destination vector where the result is to be stored; and the vector length, which tells the hardware how many elements to process and when to stop. An optional address-increment field may also be included when successive elements are not stored in strictly consecutive locations, such as when accessing a row or column of a matrix. In register-based designs such as the Cray-1, the instruction instead names vector registers that already hold this addressing information, with separate load/store instructions used to move vectors between memory and the registers.

Fig. — General fields of a vector instruction

Q5. Why does vector processing give a greater speed advantage for long vectors than for short vectors?

Answer:

A pipelined vector unit takes a certain fixed number of clock cycles to fill before it starts delivering one result per cycle (the pipeline fill time), and a similar time to drain after the last element enters. For a long vector, this fixed overhead is spread over a large number of elements, so the average time per element approaches the ideal one-cycle-per-result rate, giving a large overall speed-up. For a very short vector, the fill and drain time forms a much larger fraction of the total execution time, so the effective speed-up is much smaller and may even be worse than a direct scalar computation for extremely short vectors.

Q6. Describe the basic organisation of an SIMD array processor.

Answer:

An SIMD array processor consists of a single control unit that fetches and decodes program instructions and broadcasts each decoded instruction to an array of processing elements (PE0, PE1, ..., PE(n-1)) over a control/broadcast bus. Each processing element contains its own ALU, registers, and usually its own local memory holding the data element it is responsible for, so all PEs execute the same instruction simultaneously but on different data. An interconnection network allows data to be exchanged between processing elements when an algorithm requires one PE to use a value held by another, and a masking mechanism allows individual PEs to be selectively disabled so that data-dependent (conditional) behaviour can still be expressed.

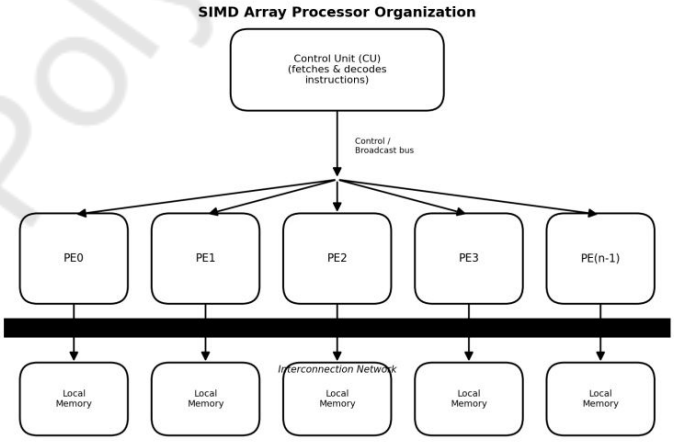

Fig. — General organisation of an SIMD array processor

Q7. Differentiate between an attached array processor and a true SIMD array processor.

Answer:

An attached array processor is an auxiliary unit connected to a separate, general-purpose host computer; the host retains overall program control, operating-system functions and I/O, and simply hands off vector/array-intensive computations to the attached unit for acceleration. A true SIMD array processor, by contrast, is a stand-alone computer built from the outset around the control-unit-plus-processing-element-array organisation; the array structure is itself the main processing hardware of the machine rather than an add-on accelerator to another computer.

Section C — Long Answer Type Questions

(10–15 marks each) — full model answers suitable for descriptive/university examinations.

Q1. What is vector processing? Explain, with suitable examples, the various techniques used to achieve high-speed vector computation.

Answer:

Vector processing is a mode of computation in which a single machine instruction operates on an entire ordered set of data elements — a vector — instead of on one scalar element at a time. Many scientific and engineering problems, such as weather modelling, structural analysis and matrix computation, require the same arithmetic operation to be repeated over large sets of numbers; a vector processor is organised specifically to carry out such repeated operations with much less overhead and much higher throughput than an equivalent scalar program loop. The main techniques used to achieve this are described below.

(a) Pipelining of the Arithmetic Unit

The arithmetic unit is split into a sequence of independent stages, each performing a small part of the total operation. For floating-point addition, typical stages are: compare exponents, align mantissas, add mantissas, and normalise the result. Successive vector-element pairs are fed into the pipeline on every clock cycle, so several pairs are in different stages of computation at the same instant. Once the pipeline is full, one result is produced every clock cycle, instead of one result every several cycles as a non-pipelined unit would require.

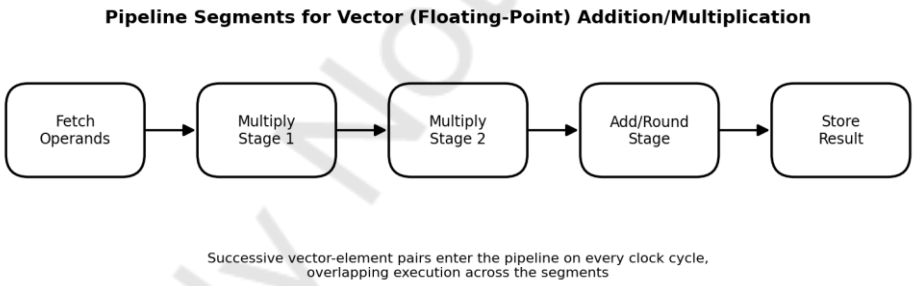

Fig. — Pipeline segments for vector addition/multiplication

(b) Parallel Functional Units

In addition to, or instead of, pipelining a single unit, several identical arithmetic units may be provided so that independent element-pairs are distributed among them and processed during the same time interval, increasing throughput further.

(c) Vector Registers

High-speed vector registers hold an entire vector, or a section of one, close to the pipelined arithmetic units, so that operands can be supplied at very high speed without repeated main-memory accesses for every element. This register-to-register approach is used in machines such as the Cray-1.

(d) Interleaved Memory

Because a pipeline requires a new operand every clock cycle, main memory is organised into several interleaved modules, so that consecutive vector elements can be fetched from different modules in an overlapped manner, matching the demand rate of the pipeline.

(e) Chaining

When the output vector of one pipelined operation is itself required as input to a second vector operation, the results can be chained directly from the output of the first pipeline into the input of the second as each element becomes available, rather than waiting for the first operation to finish completely. This overlaps successive dependent vector instructions and further reduces total execution time.

Together, these techniques allow a vector processor to process large, regular data sets at a rate approaching one result per clock cycle, which is the basis of its large performance advantage over conventional scalar computation for scientific and array-oriented workloads.

Q2. Explain the speed advantage of vector processing over scalar processing. Also describe the general format of a vector processing instruction.

Answer:

Consider adding two vectors of n elements each using a scalar machine: the operation C(I) = A(I) + B(I) must be placed inside a program loop that executes n times. Each pass of the loop, in addition to performing the actual addition, must increment the loop index, compare it against the limit n, compute the effective addresses of A(I), B(I) and C(I), and branch back to the top of the loop — the same instructions must also be re-fetched and re-decoded on every pass. This loop-control and instruction-processing overhead is repeated n times even though the underlying arithmetic operation never changes.

A vector instruction avoids this overhead by specifying the entire operation — both source vectors, the destination vector, and the vector length — in a single instruction. That instruction is fetched and decoded only once; dedicated address-generation hardware and a vector-length counter then step automatically through the n elements, while the pipelined arithmetic unit streams the operations through, producing one result every clock cycle once the pipeline is full. The speed advantage therefore comes from two sources: elimination of repeated loop-control and fetch/decode overhead, and the overlap achieved by the arithmetic pipeline. For long vectors, the fixed pipeline fill/drain time is spread over many elements, so the effective time per element approaches a single clock cycle, giving substantial overall speed-up; for very short vectors this advantage is much smaller because the fill/drain overhead is not well amortised.

Vector Instruction Format

Because a vector instruction must identify entire vectors rather than single operands, its format typically includes the following fields:

- Operation Code (Opcode) — specifies the operation, e.g. vector add, multiply, load.
- Base Address of Source Vector 1 (A1) — starting memory address of the first operand vector.
- Base Address of Source Vector 2 (A2) — starting address of the second operand vector, where applicable.
- Base Address of Destination Vector (A3) — starting address at which the result vector is stored.
- Vector Length (N) — the number of elements to be processed, used by the control hardware to know when to stop.
- Address Increment (optional) — the spacing between successive elements, useful when accessing a row or column of a matrix.

Fig. — General fields of a vector instruction

In register-oriented designs such as the Cray-1, the instruction instead references vector registers, which already hold the base address, length and increment for the currently loaded vector, and separate vector load/store instructions transfer data between memory and these registers. In either format, the essential idea is the same: a single instruction encodes an entire repetitive operation that a scalar architecture would otherwise require an explicit multi-instruction loop to express.

Q3. What is an array processor? Explain its organisation with a neat diagram, and describe the function of each major component.

Answer:

An array processor is a computer that achieves high-speed computation by using a large number of identical processing elements (PEs), all operating under the supervision of a single control unit, so that the same instruction is applied simultaneously, in true spatial parallelism, to many different data elements. This organisation is classified as SIMD (Single Instruction stream, Multiple Data stream) under Flynn's taxonomy: one instruction stream is broadcast to many processing elements, each of which applies it to the data held in its own local memory.

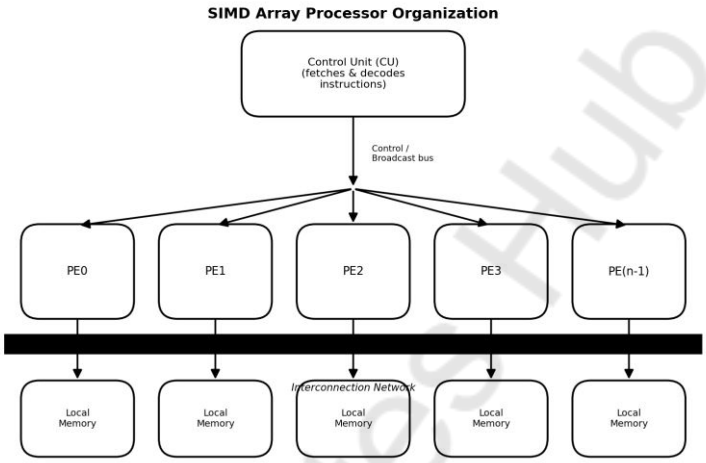

Fig. — General organisation of an SIMD array processor

Major Components

- Control Unit (CU): fetches and decodes program instructions, generates control signals, and broadcasts the decoded instruction (and any common/scalar operands) to every processing element over a control/broadcast bus.
- Processing Elements (PE0 ... PE(n-1)): each PE has its own ALU and registers, and executes the instruction broadcast by the CU on the data available to it, in lock-step with every other PE.
- Local Memory: each PE typically has a private memory module holding the data element(s) assigned to it, so that all PEs can access their own operands simultaneously without memory conflicts.
- Interconnection Network: connects the PEs so that data can be exchanged, shifted, broadcast, or permuted between them, which is necessary whenever an algorithm requires one PE to use a value computed by another.
- Masking Mechanism: allows individual PEs to be selectively enabled or disabled for a given instruction, so that data-dependent (conditional) computation can still be expressed even though every active PE is executing the same instruction.

Because all active PEs compute in the same clock cycle, an array of N processing elements can, ideally, complete a given operation on N data elements in roughly the same time a single processor would take for

one element — that is, close to an N-fold speed-up for suitably regular, data-parallel problems such as image processing, matrix operations, and finite-difference simulations on structured grids.

Q4. Describe the different types of array processors, bringing out clearly the differences between them.

Answer:

Array processors are classified mainly according to where the parallel hardware sits relative to the main control and data path of the overall computer system, and according to how each processing element accesses its data.

(a) Attached Array Processor

An attached array processor is an auxiliary, high-speed unit connected to a general-purpose host computer for the specific purpose of accelerating numerically intensive, vector-oriented computation. The host continues to handle overall program execution, the operating system, and input/output; whenever a vector/array-intensive computation arises, the required data is transferred to the attached processor, which performs the high-speed arithmetic and returns the results. It is typically built from pipelined arithmetic units and connected to the host through an I/O channel or dedicated high-speed interface. Its main purpose is to improve the performance of an existing host inexpensively, without redesigning the host itself.

(b) SIMD Array Processor (True Array Processor)

This is a stand-alone computer built entirely around the array-processing organisation: a single control unit driving a synchronous array of processing elements, each with its own ALU and local memory, connected through an interconnection network. The array structure is the main processing hardware of the machine itself, not an accelerator attached to a separate host. ILLIAC IV and the Connection Machine (CM-1/CM-2) type designs are historically important examples, using large numbers of simple PEs operating in lock-step, and are best suited to problems with a regular, uniform, data-parallel structure such as image and signal processing.

(c) Associative (Content-Addressable) Array Processor

This is a variant of the SIMD array organisation in which each PE's local memory is content-addressable rather than address-based: data is retrieved by matching its content against a pattern broadcast by the control unit, with every PE performing the comparison simultaneously. This organisation is particularly well suited to searching, sorting, and pattern-matching operations across large, uniform data sets.

(d) MIMD-Oriented Processor Arrays

Some multiprocessor systems arrange many independent processors, each with its own control unit, into an array-like interconnection structure such as a mesh or hypercube, so that each processor executes a different instruction stream on its own data — this is Multiple Instruction stream, Multiple Data stream (MIMD) organisation. Although physically similar in having many processing nodes joined by an interconnection network, such systems differ fundamentally from true SIMD array processors because each node has independent control rather than sharing one common control unit.

Type	Control	Typical Use
Attached array processor	Separate host supplies overall control; attached unit only accelerates vector arithmetic	Adding vector-processing speed to an existing general-purpose host system

Type	**Control**	**Typical Use**
SIMD array processor	Single control unit; instruction broadcast to all PEs	Large, regular data-parallel problems: image processing, matrix/grid computation
Associative array processor	Single control unit; PEs use content-addressable local memory	Searching, sorting, and pattern-matching over large uniform data sets
MIMD-oriented processor array	Each processing node has its own independent control unit	General-purpose parallel/distributed computation with independent tasks per node

Q5. Compare pipelined vector processing and SIMD array processing as two approaches to exploiting data-level parallelism.

Answer:

Both pipelined vector processors and SIMD array processors are designed to exploit the same underlying data-level parallelism present in vector and matrix computation — the fact that the same operation must be applied to many independent data elements — but they achieve this in fundamentally different ways.

A pipelined vector processor uses one, or a small number of, pipelined arithmetic units. The arithmetic unit is divided into stages, and successive element-pairs are fed in on every clock cycle so that many elements are in different stages of computation simultaneously; parallelism is achieved in time, through overlap, using a single (or few) physical arithmetic pipeline. An SIMD array processor, by contrast, uses many identical, independent processing elements, each with its own ALU and local memory, all executing the same broadcast instruction in the same clock cycle on different data; parallelism here is achieved in space, by duplicating the arithmetic hardware itself rather than by overlapping operations in a single pipeline.

Aspect	**Pipelined Vector Processor**	**SIMD Array Processor**
Parallelism achieved	In time (temporal overlap within one pipeline)	In space (many ALUs operate at the same instant)
Hardware duplicated	Pipeline stages of one (or few) arithmetic units	Entire ALU and local memory in every processing element
Result rate	One result per clock cycle after pipeline fills	N results per clock cycle for N active processing elements
Efficiency for short vectors	Reduced, due to pipeline fill/drain overhead	High, provided N or fewer elements are needed
Representative examples	Cray-1 (register-to-register vector pipeline)	ILLIAC IV, Connection Machine

In practice, many real high-performance systems combine both ideas — building an array of pipelined processing elements — so as to obtain both forms of parallelism (spatial replication and temporal pipelining) together, achieving even greater throughput than either technique alone.