

Internal Assessment Test III

November 2019

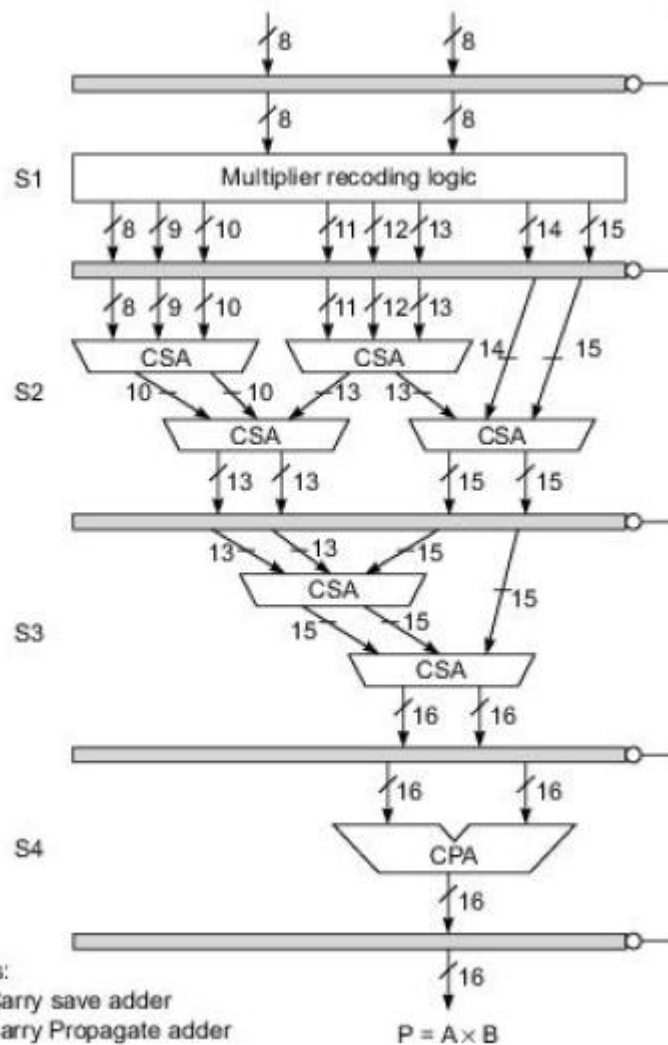
Sub:	Advanced Computer Architecture	Sub Code	15CS72	Branch:	CSE
Date:	18/11/2019	Duration:	90 mins	Max Marks:	50
		Sem / Sec:	VII		OBE
		A,B,C			

Answer any FIVE FULL Questions

MARK S
CO RB
T

1 (a)	<p>Explain multiply pipeline design to multiply two 8-bit integers.</p> <p>1. Consider as an example the multiplication of two 8-bit integers. $A * B = P$, where P is the 16 bit product. This fixed-point multiplication can be written as the summation of eight partial products as shown below:</p> <p>$P = A * B = P_0 + P_1 + P_2 + P_3 + P_4 + \dots + P_7$ where * and + are arithmetic multiply and add operations, respectively.</p> <div style="text-align: center;"> <pre> 1 0 1 1 0 1 0 1 = A ×) 1 0 0 1 0 0 1 1 = B ----- 1 0 1 1 0 1 0 1 = P₀ 1 0 1 1 0 1 0 1 0 = P₁ 0 0 0 0 0 0 0 0 0 = P₂ 0 0 0 0 0 0 0 0 0 0 = P₃ 1 0 1 1 0 1 0 1 0 0 0 = P₄ 0 0 0 0 0 0 0 0 0 0 0 = P₅ 0 0 0 0 0 0 0 0 0 0 0 0 = P₆ +) 1 0 1 1 0 1 0 1 0 0 0 0 0 0 = P₇ ----- 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 = P </pre> </div> <p>2. Note that the partial product P_j is obtained by multiplying the multiplicand A by the jth bit of B and then shifting the result j bits to the left for $j = 0, 1, 2, \dots, 7$.</p> <p>3. The summation of the eight partial products is done with a Wallace tree of CSAs plus a CPA at the final stage, as shown in figure given below.</p> <p>4. The first stage S1 generates all eight partial products, ranging from 8 bits to 15 bits simultaneously. The second. Stage S2 is made up of two levels of four CSA's and it essentially merges eight numbers into four numbers ranging from 13 to 15 bits. The third stage S3 consists of two CSAs and it merges four numbers from S2 into two 16-bit numbers. The final stage</p>	(06)	CO2	L2
-------	---	------	-----	----

(S4) is a CPA, which adds up the last two numbers to produce the final product P.



1 (b) Explain the concept of Hazard Avoidance.

1. If the instructions are not executed in order, incorrect results may be read or written, thereby producing hazards.
2. Consider two instructions I and J in program order such that J follows I. We use the notation D(I) and R(I) for the domain and range of an instruction I. The domain contains the input set to be used by instruction I. The range corresponds to the output set of instructions I.
3. Listed below are the conditions in which the hazards can occur.

$R(I) \cap D(J) \neq \phi$ for RAW hazard

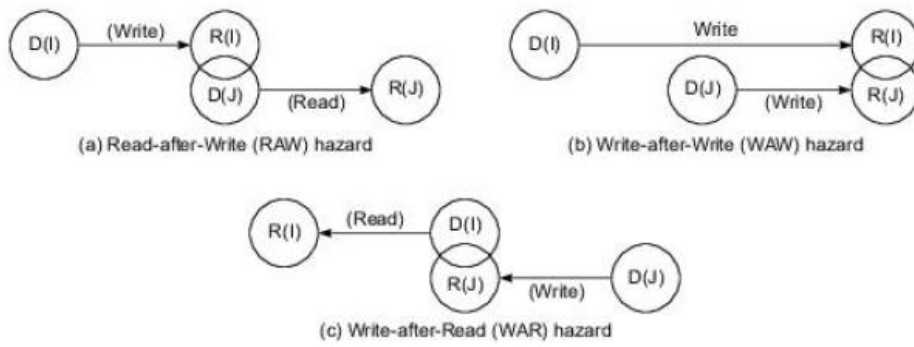
$R(I) \cap R(J) \neq \phi$ for WAW hazard

$D(I) \cap R(J) \neq \phi$ for WAR hazard

(04)

CO3

L1



4. The RAW hazard corresponds to the flow dependence, WAR to the anti-dependence, and WAW to the output dependence.

2 (a) **What is Arbitration? Explain different types of arbitration.**

(08) CO3 L2

The process of assigning the DTB bus to the requesting processor or master is called arbitration. The duration of master's control over the bus is called as bus tenure.

Schemes for arbitration

1. Central Arbitration
2. Distributed Arbitration

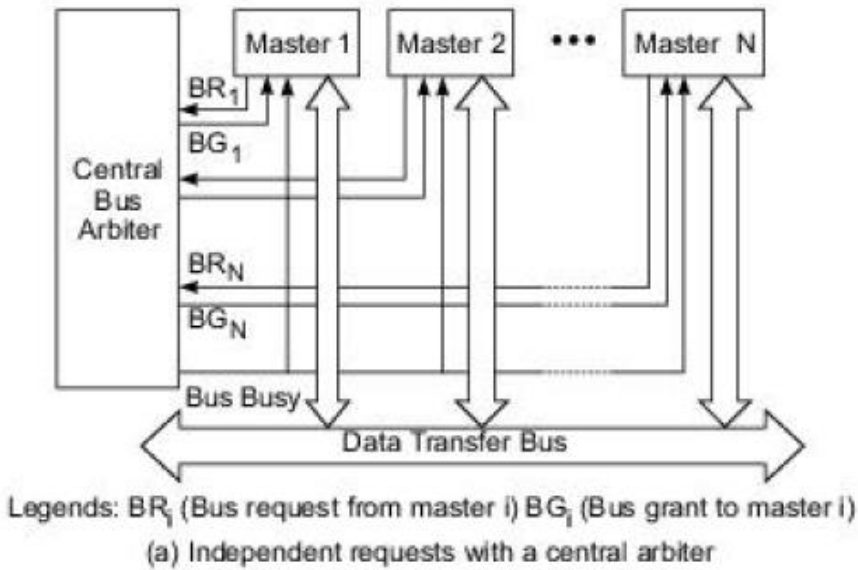
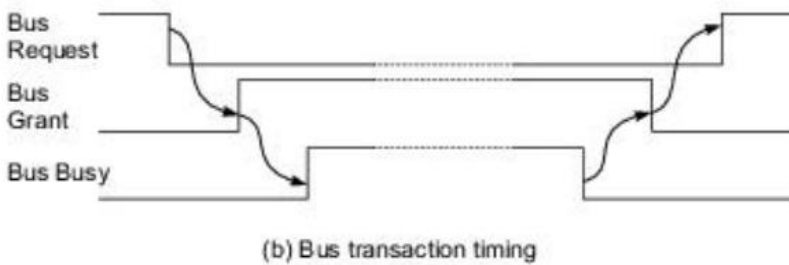
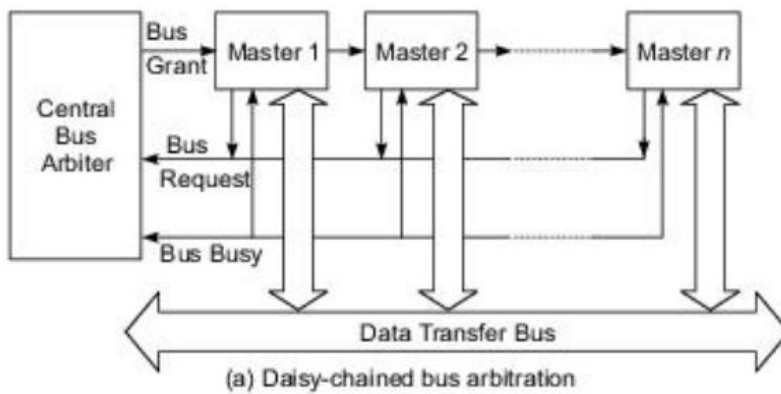
Central Arbitration

1. It uses a central arbiter. The bus grant signal is propagated from first master(slot 1) to last master (slot n).
2. Each master can send bus request. However, all requests share the same bus-request line. As shown in Figure, the bus-request signals the rise of the bus-grant level which in turn raises the bus-busy level.
3. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activates the busy line and gains control of the bus.
4. Fixed priority is set from left to right such that the lower priority devices are at the right. Thus lower priority device on right gets the bus control only when the devices on the left do not request bus control.
5. When the bus transaction is complete, the bus-busy level is lowered, which triggers the falling of the bus grant signal and the subsequent rising of the bus-request signal. Advantage of this arbitration scheme is its simplicity.
6. Disadvantage is fixed-priority sequence violating the fairness practice. Another drawback is its slowness in propagating the bus grant signal

along the daisy chain.

7. Whenever a higher-priority device fails, all the lower-priority devices on the right of the daisy chain cannot use the bus.

Independent Request and Grant: Instead of using shared request and grant lines as in Figure, multiple bus-request and bus-grant signal lines can be independently provided for each potential master as shown in figure. But the total number of signal lines required is larger. The arbitration among potential masters is still carried out by a central arbiter. However, any priority based or fairness-based bus allocation policy can be implemented. The advantage is their flexibility and faster arbitration time compared with the daisy-chained policy. The drawback is the large number of arbitration lines used.

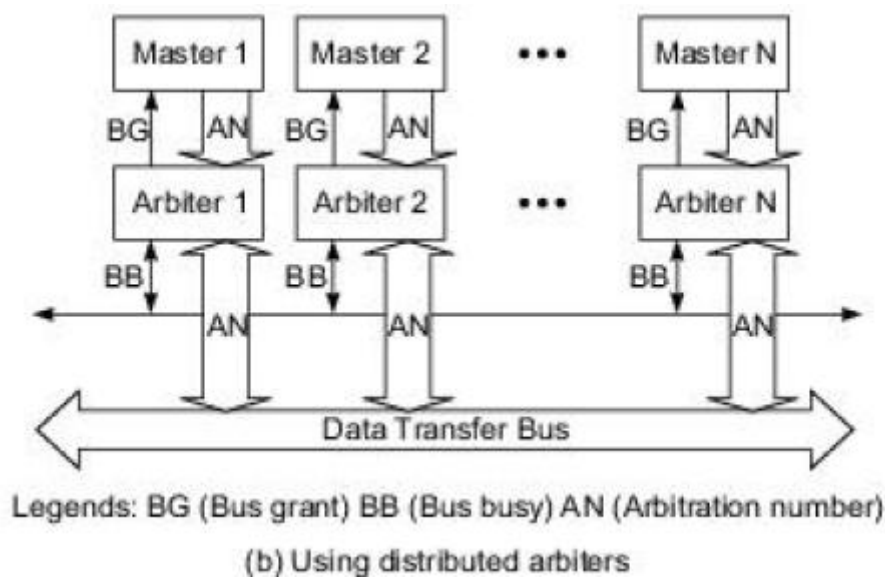


Distributed Arbitration

1. Each potential master is equipped with its own arbiter and a unique

arbitration number.

2. When two or more devices compete for the bus, the winner is the one whose arbitration number is the largest.
3. All potential masters can send their arbitration numbers to the shared-bus request grant (SBRG) lines on the arbitration bus
4. Each arbiter compares the resulting number on the SBRG lines with its own arbitration number. If the SBRG number is greater, the requester is dismissed. At the end, the winner's arbitration number remains on the arbitration bus. After the current bus transaction is completed, the winner seizes control of the bus.
5. It is shown in figure given below



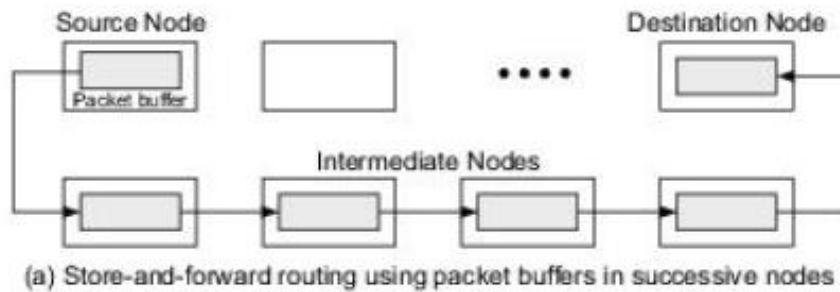
2 (b) Explain the concept of Store and Forward routing.

1. Packets are the basic unit of information flow in store and forward network.
2. A packet is transmitted from a source node to a destination node through a sequence of intermediate nodes.
3. When a packet reaches an intermediate node, it is first stored in the buffer. Then it is forwarded to the next node if the desired output channel and a packet buffer in the receiving node are both available.
4. The latency in store-and-forward networks is directly proportional to the distance (the number of hops) between the source and the destination, this routing scheme was implemented in the first generation of multicomputer.

(02)

CO3

L2



3 (a) For the reservation table of nonlinear pipeline shown below.

(08) CO3 L3

	1	2	3	4	5	6
S1	X					X
S2		X			X	
S3			X			
S4				X		
S5		X				X

a) What are forbidden latencies? Write initial collision vector. b) Draw the state transition diagram
 c) List all the simple cycles and greedy cycles. d) Determine MAL

To detect a forbidden latency, one needs simply to check the distance between any two checkmarks in the same row of the reservation table.

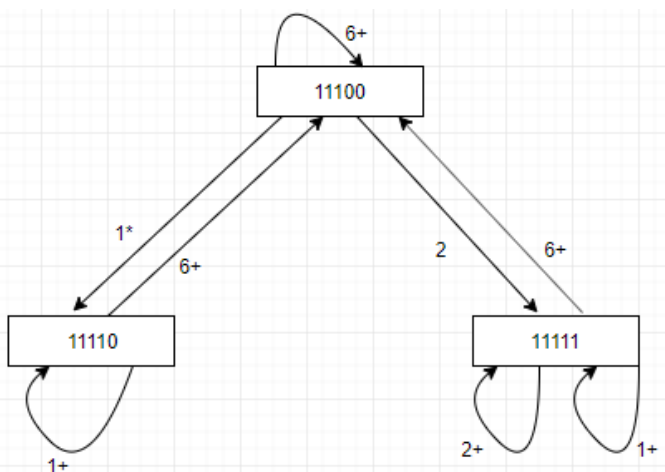
Forbidden Latencies : 5,3,4

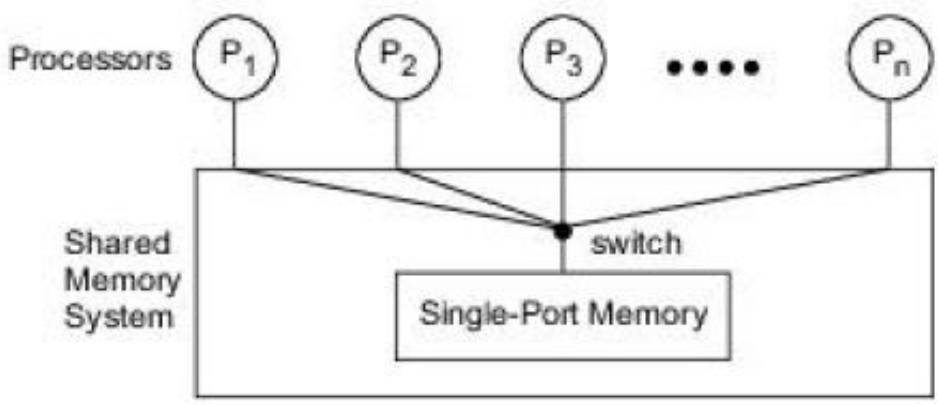
For a reservation table with n columns the maximum forbidden latency $m \leq n-1$.
 The permissible latency p should be as small as possible. Hence $1 \leq p \leq m-1$.

The combined Set of permissible and forbidden latencies can be easily displayed by a collision vector, which is an m -bit binary vector $C = (C_m C_{m-1} \dots C_3 C_2 C_1)$.
 The value of $C_i = 1$ if latency i causes a collision and $C_i = 0$ if latency i is permissible.

Initial Collision Vector : 11100

State Transition Diagram



	Simple Cycles : (1),(1,6),(2),(2,6) Greedy Cycles : (2),(1,6) MAL : 2			
3 (b)	Define following terms 1. Latency Cycle 2. Greedy Cycle Latency Cycle: It is a latency sequence which repeats the same subsequence (cycle) indefinitely. A greedy cycle is one whose edges are all made with minimum latencies from their respective starting states	(02)	CO3	L1
4 (a)	Explain any two Context Switching policies. Four context switching policies are given below. <u>Switch on cache miss:</u> This policy corresponds to the case where a context is preempted when it causes a cache miss. Let R be the average interval between misses and L the time required to satisfy the miss. Here, the processor switches contexts only when it is certain that the current one will be delayed for a significant number of cycles. <u>Switch on every load:</u> This policy allows switching on every load independent of whether it will cause a miss or not. R represents the average interval between loads. <u>Switch on every instruction:</u> This policy allows switching on every instruction, independent of whether it is a load or not. In other words, it interleaves the instructions from different threads on a cycle-by-cycle basis. Successive instructions become independent, which will benefit pipelined execution. However, the cache miss may increase due to breaking of locality. <u>Switch on block of instruction:</u> Blocks of instructions from different threads are interleaved. This will improve the cache-hit ratio due to locality. It will also benefit single-context performance	02	CO4	L2
4 (b)	Explain the sequential and weak consistency model with a neat diagram. <u>Sequential Consistency Model</u> In this model, loads, stores and swaps of all processors appear to execute serially in single global memory order that conforms to the individual program order of the processors.  <p>The diagram illustrates the Sequential Consistency Model. At the top, a row of circles represents processors labeled P₁, P₂, P₃, followed by four dots, and then P_n. Below this row is a large rectangular box labeled 'Shared Memory System'. Inside this box, a central point is labeled 'switch'. Lines connect each processor circle to this switch. Below the switch is a smaller rectangular box labeled 'Single-Port Memory'.</p>	(08)	CO3	L2
	1. There is a single port that is able to service exactly one memory operation at a			

time, and a switch that connects this memory to one of the processors for the duration of each memory operation.

2. The order in which the switch is thrown from one processor to another determines the global order of memory-access operations which is observed by all processors.
3. Also the strong ordering of all the shared memory accesses in the sequential consistency model preserves the program order in all processors.
4. Also another processor is not allowed to access a shared memory until the recent value written to shared memory has been globally performed. This may require the propagation of all shared-memory accesses to all processors, which is rather time-consuming and costly.
5. Also sequential consistency model has poor scalability.

Lamport's Definition: A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

Dubois, Seheurich, and Briggs(DBS) (1986) have provided the following two sufficient conditions to achieve sequential consistency in shared-memory access:

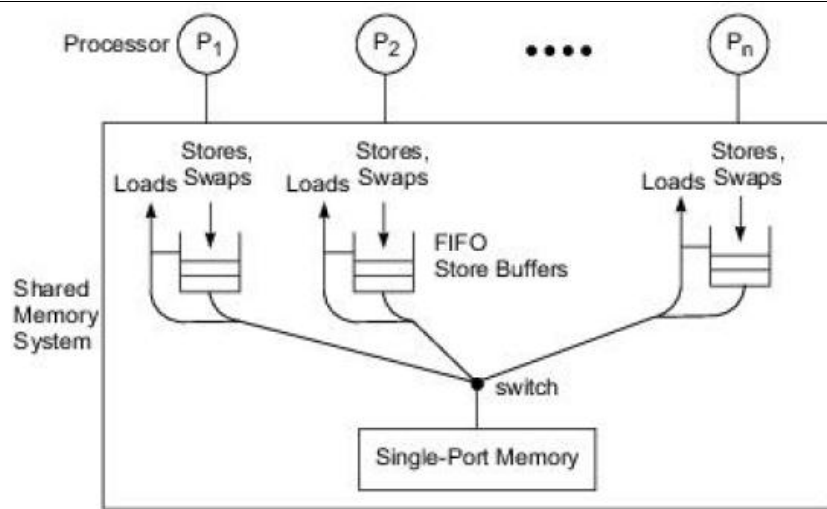
1. Before a load is allowed to perform with respect to any other processor, all previous load accesses must be globally performed and all previous store accesses must be performed with respect to all processors.
2. Before a store is allowed to perform with respect to any other processor, all previous load accesses must be globally performed and all previous store accesses must be performed with respect to all processors.

Strong Ordering introduces unnecessary processor/cache waiting time and reduces the concurrency.

Weak Consistency Model

The weak consistency TSO model developed by Sun Microsystems SPARC architecture is described below. The memory events store and swap are placed in a dedicated store buffer of each processor. Thus the order in which these memory events are performed is the same as the order in which the processor issued them (in program order).

1. The memory order corresponds to the order in which the switch is thrown from one processor to another.
2. A load by a processor first checks its store buffer to see if it contains a store to the same location. If it does, then the load returns the value of the most recent such store. Otherwise, the load goes directly to memory.
3. A swap is placed in the store buffer like a store and it blocks the processor like a load. In other words, swap blocks until the store buffer is empty and then proceeds to the memory.



A TSO Formal Specification

1. A load access is always returned with the latest store to the same memory location issued by any processor in the system.
2. If two stores appear in a particular program order, then they must also appear in the same memory order.
3. If a memory operation follows a load in program order, then it must also follow the load in memory order.
4. A store operation is atomic with respect to other stores.
5. No other store can interleave between the load and store parts of a swap.
6. All stores and swaps must eventually terminate.

The weak consistency model may offer better performance than the sequential consistency model at the expense of more complex hardware software support and more programmer awareness of the imposed restrictions.

5 (a) With a neat diagram explain the architecture of connection machine CM2.

(10)

CO4

L2

The Connection Machine CM-2 produced by Thinking Machines Corporation was a fine-grain MPP computer using thousands of bit-slice PEs in parallel to achieve a peak processing speed of above 10 Gflops.

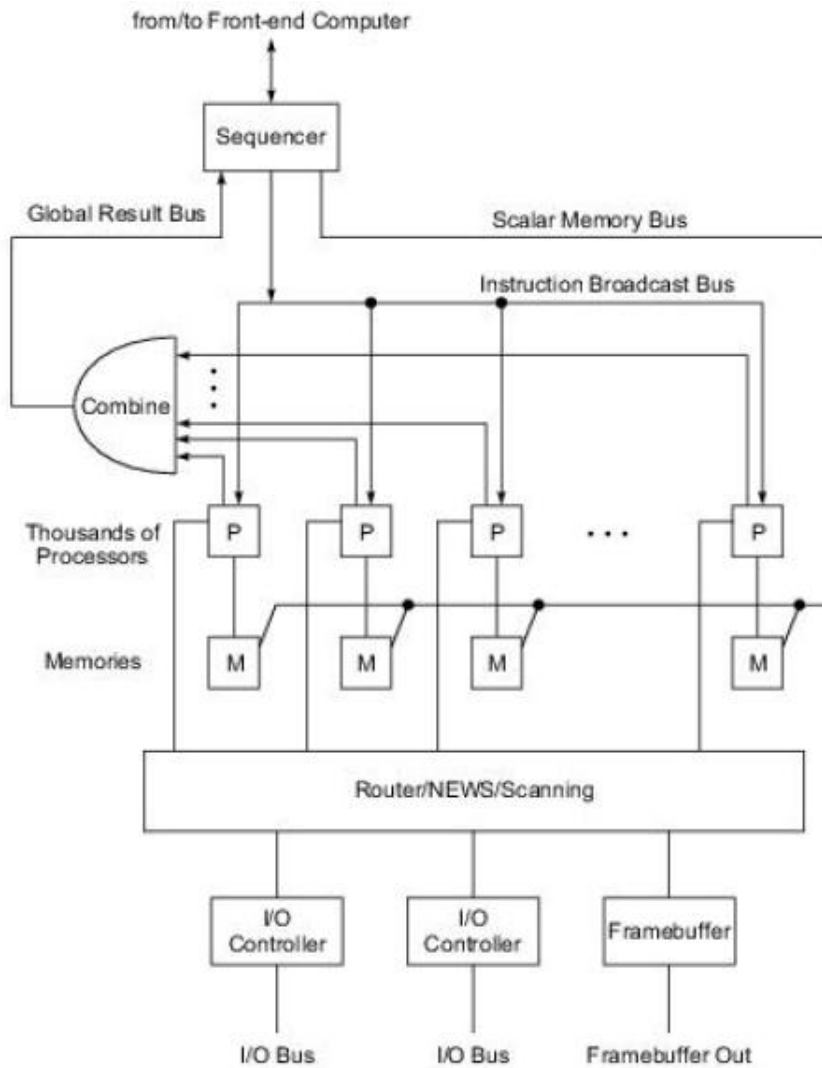
Program execution paradigm

1. The microinstructions are issued from the front end to the backend processing array when data-parallel operations are applied.
2. The sequencer breaks down those microinstructions and broadcast them to all data processors in the array.
3. Data sets and results could be exchanged between the front-end and the processing array in one of three ways i.e. broadcasting, global combining and scalar memory bus.

The Processing Array

1. The processing array contained from 4K to 64K bit-slice data processors (or PEs), all of which were controlled by a sequencer.
2. All processors could access their memories simultaneously. All processors executed the broadcast instructions in a lockstep manner.
3. The processors exchanged data among themselves in parallel through the router,

NEWS grid, or scanning mechanism.



Processing Node

1. A CM2 processing node consists of two processor chips and some memory and floating-point chips.
2. The processor chips were paired in each node sharing a group of memory chips. Each processor chip contained 16 processors. The parallel instruction set, called Paris, included nano-instructions for memory load and store, arithmetic and logical, and control of the router, NEWS grid, and hypercube interface, floating point, I/O, and diagnostic operations.

Hyper-cube Router

1. Special hardware was built on each processor chip for data routing among the processors. The router nodes on all processor chips were wired together to form a Boolean n-cube. A full configuration of CM-2 had 4096 router nodes on processor chips interconnected as a 12-dimensional hypercube.
2. All 16 processors belonging to the same node were equally capable of sending a message from one vertex to any other processor at another vertex of the 12-cube.

The NEWS Grid

1. The "NEWS" grid was based on the fact that each processor has a north, east, west, and south neighbor in the various grid configurations.
2. These flexible interconnections among the processors made it very

efficient to route data on dedicated grid configurations based on the application requirements.

Scanning and Spread Mechanism

1. It is used for fast data combining and spreading in the NEWS grid.
2. Scanning operation could simultaneously scan in every row of a grid along a particular dimension for the partial sum of that row, the largest or smallest value, or bitwise OR, AND, or exclusive OR.
3. Spreading could send a value to all other processors across the chips.

I/O and Data vault

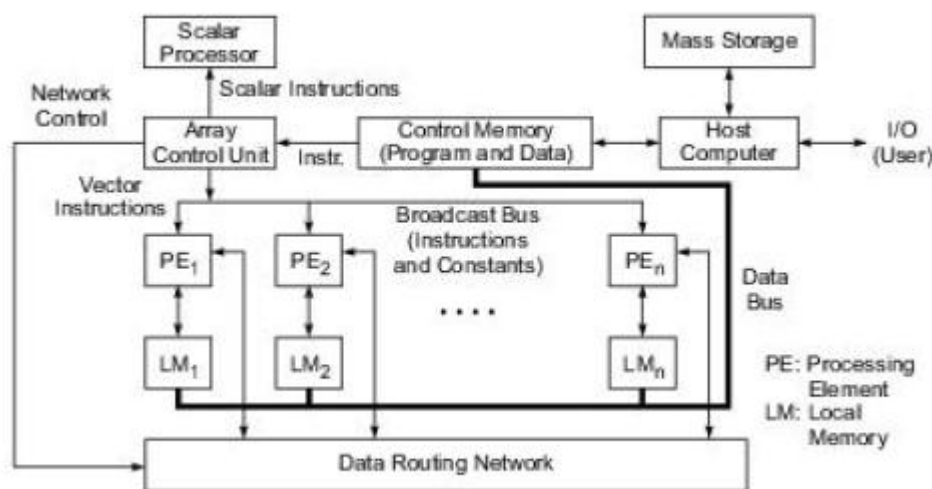
4. High-speed I/O channels were available from 2 to 16 channels for data and for image I/O operations.
5. Peripheral devices attached to I/O channels included a data vault, CM-HIPPI system, CM-IOP system etc.
6. The data vault was a disk-based mass storage system for storing program files and large data bases.

6 (a) **What are the implementation models of SIMD? Explain in detail**

There are two implementation models for SIMD as given below

Distributed-Memory Model

1. A distributed-memory SIMD computer consists of an array of PEs which is controlled by the same array control unit.
2. Program and data are loaded into the control memory through the host computer.
3. An instruction is sent to the control unit for decoding. If it is a scalar or program control operation, it will be directly executed by a scalar processor attached to the control unit. If the decoded instruction is a vector operation, it will be broadcast to all the PEs for parallel execution.



(a) Using distributed local memories (e.g. the Illiac IV)

4. Partitioned data sets are distributed to all the local memories attached to the PEs through a vector data bus.

(10)

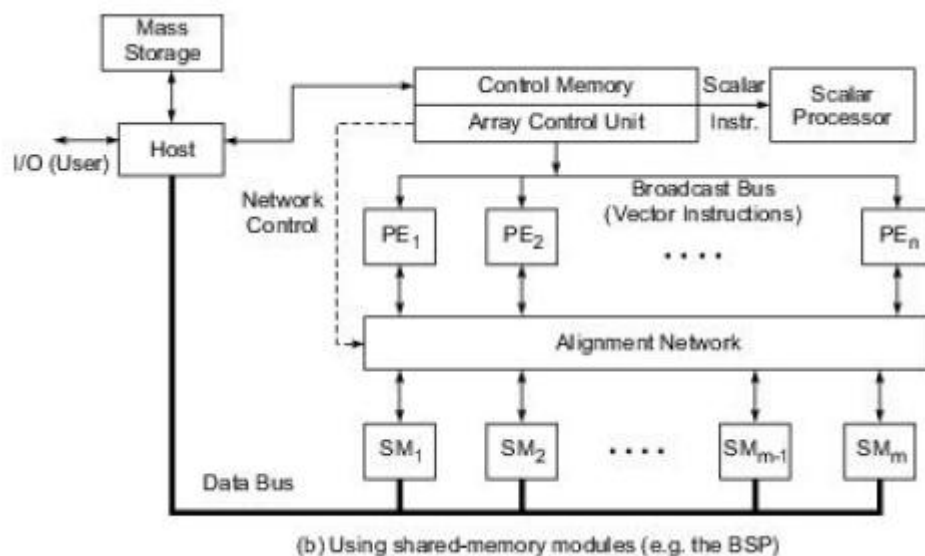
CO4

L2

- The Illiac IV was such an early SIMD machine consisting of 64 PEs with local memories interconnected by an 8*8 mesh.
- The PEs is interconnected by a data-routing network which performs inter-PE data communications such as shifting, permutation, and other routing operations.

Shared Memory Model

- All the PEs are connected to a shared memory.
- An alignment network is used as the inter-PE memory communication network. Again this network is controlled by the control unit.
- The Burroughs Scientific Processor (BSP) had adopted this architecture, with $n = 16$ PEs updating $m = 17$ shared-memory modules through a $16*17$ alignment network.
- The alignment network must be properly set to avoid access conflicts.



7(a) **Explain the Concurrent OOP, Actor Model and Parallelism in COOP.**

(10)

CO5

L2

Object-Oriented Model

- It is parallel programming model.
- In this model, objects are dynamically created and manipulated. Processing is performed by sending and receiving messages among objects.
- Concurrent programming models are built up from low-level objects such as processes and semaphores and high level objects like monitors and program modules.

Concurrent OOP

The Concurrent Object Oriented programming is popular due to three reasons given below

- Due to increased use of interacting processes.
- For resource sharing and distributed problem solving.

3. For providing supercomputing power at a fraction of the traditional cost.

Objects are program entities which encapsulate data and operations into single computational units. The development of concurrent object-oriented programming (COOP) provides an alternative model for concurrent computing on multiprocessors or on multicomputer. Various object models differ in the internal behavior of objects and in how they interact with each other.

An Actor Model

It is developed at MIT as one framework for COOP.

Actors are self-contained, interactive, independent components of a computing system that communicate by asynchronous message passing. Basic actor primitives include:

1. Create: Creating an actor from a behavior description and a set of parameters.
2. Send-to: Sending a message to another actor.
3. Become: An actor replacing its own behavior by a new behavior.

Each message may cause an object (actor) to modify its state, create new objects, and send new messages. The actor model is particularly suitable for multicomputer implementations.

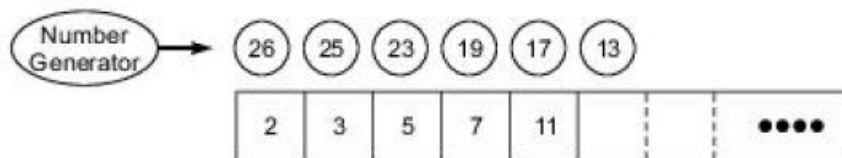
Parallelism in COOP

Two common patterns of parallelism have been found in the practice of COOP.

1. Pipeline Concurrency

First, pipeline concurrency involves the overlapped enumeration of successive solutions and concurrent testing of the solutions as they emerge from an evaluation pipeline.

Example: Prime number generation pipeline. Numbers enter from the left end and is eliminated if it is divisible by the prime number tested at a pipeline stage. All the numbers being forwarded to the right of a pipeline stage are those indivisible by all the prime numbers tested on the left of that stage.



(a) Pipeline concurrency

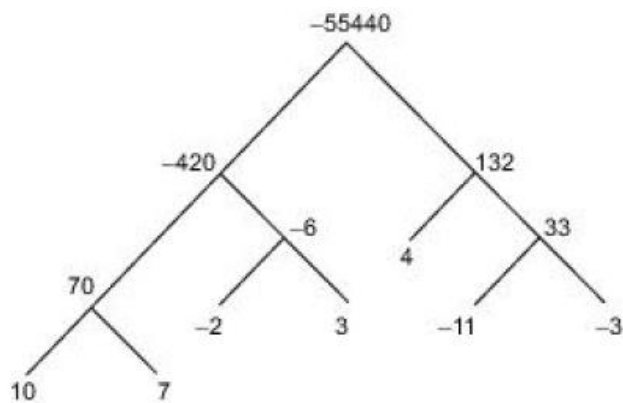
2. Divide-and-conquer concurrency

It involves the concurrent elaboration of different subprograms and the combining of their solutions to produce a solution to the overall problem.

Example: Multiplication of a list of numbers [10, 7, -2, 3, 4, -11, -3] using a divide and- conquer approach. The numbers are re-presented as leaves of a tree.

The problem can be recursively subdivided into sub problems of multiplying two

sub lists, each of which is concurrently evaluated and the results multiplied at the upper node.

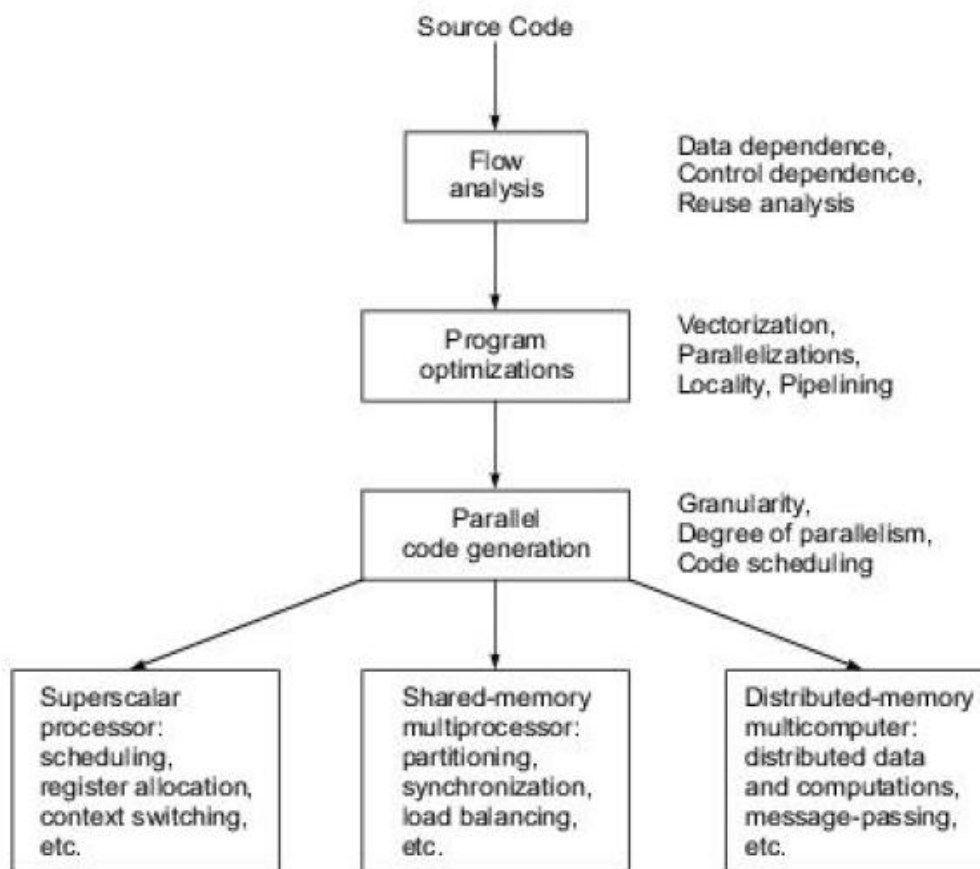


(b) Divide-and-conquer concurrency

8 (a) Explain different phases of the parallelizing compiler with a neat diagram.

(06) CO5 L2

A parallelizing compiler consists of the following three major phases: flow analysts, optimizations, and code generation as shown in figure given below.



Flow Analysis

1. Here the program source code flow analysis is done to determine the data and control dependencies.
2. Flow analysis is conducted at different execution levels on different parallel computers.
3. Instruction-level parallelism is exploited in superscalar or VLSI processors; loop level in SIMD, vector, or systolic computers; and task level in multiprocessors. multicomputer or a network of workstations.

4. The flow analysis must also reveal code/data reuse and memory-access patterns.

Program Optimization

1. The transformation of the program can be done at loop level, locality level or perfecting level with the ultimate goal of reaching global optimization. The optimization often transforms a code into “better” form in the same representation language. These transformations should be machine-independent.
2. The ultimate goal of program optimization is to maximize the speed of code execution, minimization of code length and of memory accesses and the exploitation of parallelism in programs.
3. Both local and global optimizations are needed in most programs. Sometimes the optimization should be conducted at the algorithmic level and must involve the programmer.
4. Various optimization techniques include vectorization using pipelined hardware, parallelization using multiple processors, elimination of unnecessary branches or common expressions.

Parallel Code Generation

1. Code generation usually involves transformation from one representation to another, called an Intermediate form.
2. Parallel code generation is difficult in super scalar processor due to register allocation and synchronization overhead when codes are partitioned for multiprocessor execution.
3. Compiler directives can be used to help generate parallel code when automated code generation cannot be implemented easily.
4. Two well-known exploratory optimizing compilers were developed over mid 1980: one was Parafrase at the University of Illinois, and the other was the PFC (Parallel FORTRAN Converter) at Rice University.

Parafrase and Parafrase 2

1. It transforms sequential Fortran 77 programs into forms suitable for vectorization or parallelization. Parafrase contains more than 100 program transformations which are encoded as passes.
2. The output of Parafrase is the converted concurrent program.
3. Different programs use different pass list and thus go through different sequences of transformations. The pass lists can be optimized for specific machine architectures and specific program constructs. Parafrase 2 was developed for handling programs written in C and Pascal, in addition to convening FORTRAN codes.

The PFC and ParaScope

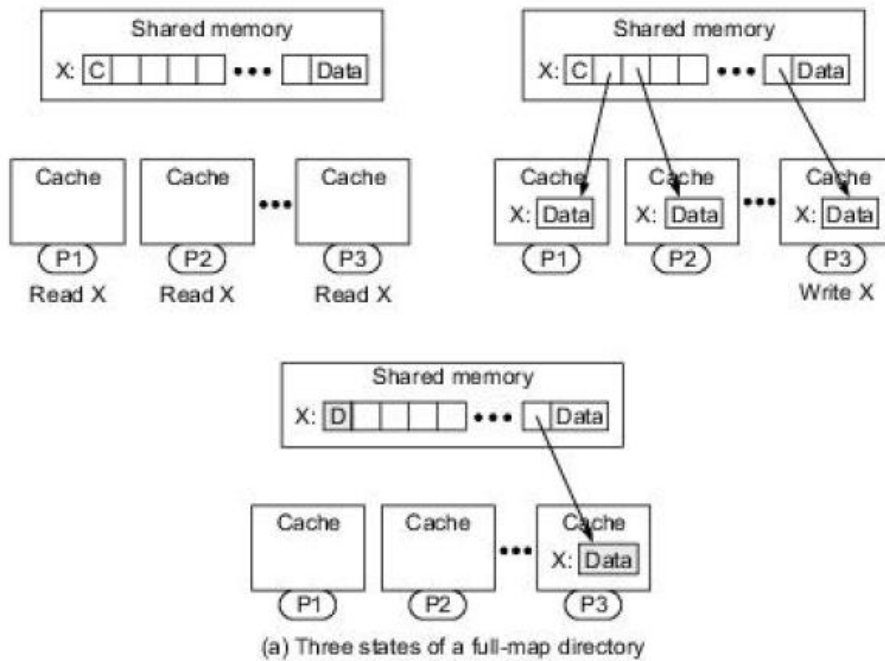
4. It translated FORTRAN 77 code into FORTRAN 90 code.
5. The PFC package was also extended to PFC+ for parallel code generation on shared-memory multiprocessors and also it supports the ParaScope programming environment.

8 (b) **Explain the concept of Full Map Directories with neat diagram for Directory based protocol.**

(04) CO4 L2

Full-MAP Directories

1. For each shared data object the directory entry contains one bit per processor which represents if copy of block is present or absent in the corresponding processor's cache and a dirty bit/clean bit field.
2. If the dirty bit is set, then one and only one processor's bit is set and that processor can write into the block.
3. The figure given below illustrates three different states of a full-map directory.



4. In the first state, location X is missing in all of the caches in the system. The second state results from three caches (C1, C2, and C3) requesting copies of location X. Three pointers [processor bits] are set in the entry to indicate the caches that have copies of the block of data. In the first two states, the dirty bit on the left side of the directory entry is set to clean (C) indicating that no processor has permission to write to the block of data. The third state results from cache C3 requesting write permission for the block. In the final state, the dirty bit is set to dirty {D}, and there is a single pointer to the block of data in cache C3.

- (1) Cache C3 detects that the block containing location X is valid but that the processor does not have permission to write to the block, indicated by the block's write-permission bit in the cache.
- (2) Cache C3 issues a write request to the memory module containing location X and stalls processor P3.
- (3) The memory module issues invalidate requests to caches C1 and C2.
- (4) Caches C1 and C2 receive the invalidate requests, set the appropriate bit to indicate that the block containing location X is invalid, and send acknowledgments back to the memory module.
- (5) The memory module receives the acknowledgments, sets the dirty bit, clears the pointers to caches C1 and C2, and sends write permission to cache C3.
- (6) Cache C3 receives the write permission message, updates the state in the cache, and reactivates processor P3.

Disadvantage

It is not scalable

Lot of memory overhead due to N pointers for each directory entry. Thus, the total memory overhead scales as the square of the number of processors $O(N^2)$.