Fig(*) shows the architectural evolution from sequential scalar computer to vector processor and parallel computer -

Lookahead, Parallelism & Pipelining: Lookahead techniques were introduced to prefetch instructions in order to overlap I/O operations & to enable functional parallelism. Functional parallelism was introdu supported by two approaches: one is multiple functional units and the other is to practice pipelining at various processing levels.
Pipelining has proven especially attractive in performing identical operation over vector data string. Vector computers are divided into implicit & explicit vector.

Explicit Vector: Explicit vector system devided into memory-to memory & register-to-register architecture.

Memory-to-memory architecture: supports the pipelined flow of vector operands directly from the memory to pipelines and then back back to the memory.

Register-to-register architecture: uses vector registers to interface between the memory and functional pipelines. Register-to-register architecture can be classified as SIMD & MIMD.

SIMD: A single instruction multiple data stream is those in which the same instructions are executed, at the same time, by several processors.

Ex: Vector processor & Array processor

MIMD: Multiple Instruction Multiple Data Stream. sets of processor simultaneously execute different instruction sequence on different data sets.

Ex: SMP, NUMA & cluster.

Elements of Modern Computers:

Hardware, software and programming elements of a modern computer system are introduced below –

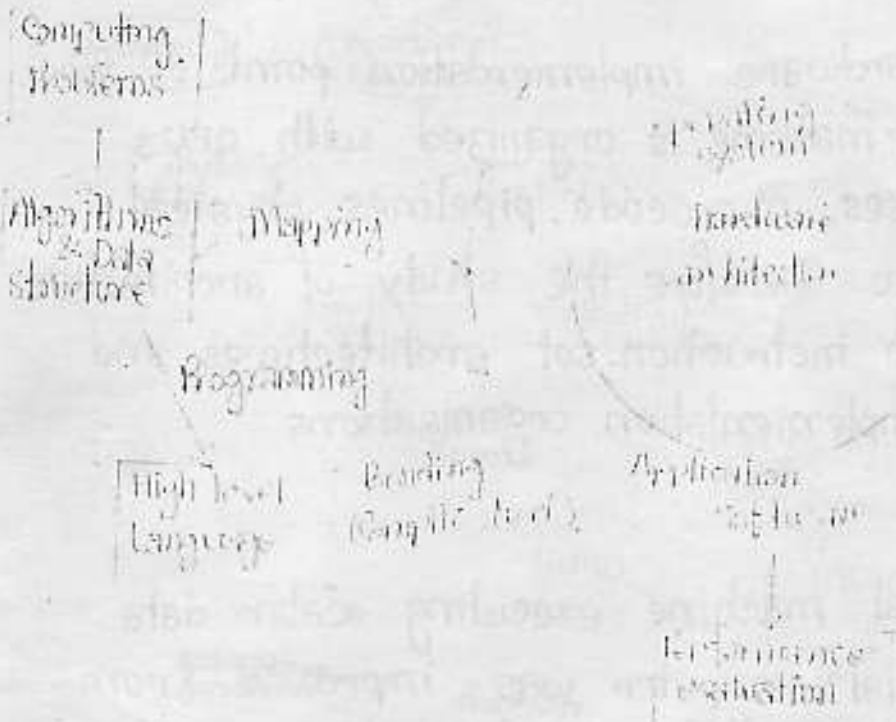


Fig Elements

Evolution of Computer Architecture:

The study of computer architecture involves both hardware organization and programming/software requirements.

From the hardware implementation point of view, the abstract machine is organized with CPU's caches, buses, microcode, pipelines, physical memory etc. Therefore, the study of architecture covers both instruction-set architectures and machine implementation organizations.

A sequential machine executing scalar data. The sequential computer was improved from bit-serial to word-parallel operations, and from fixed-point to floating-point operations.

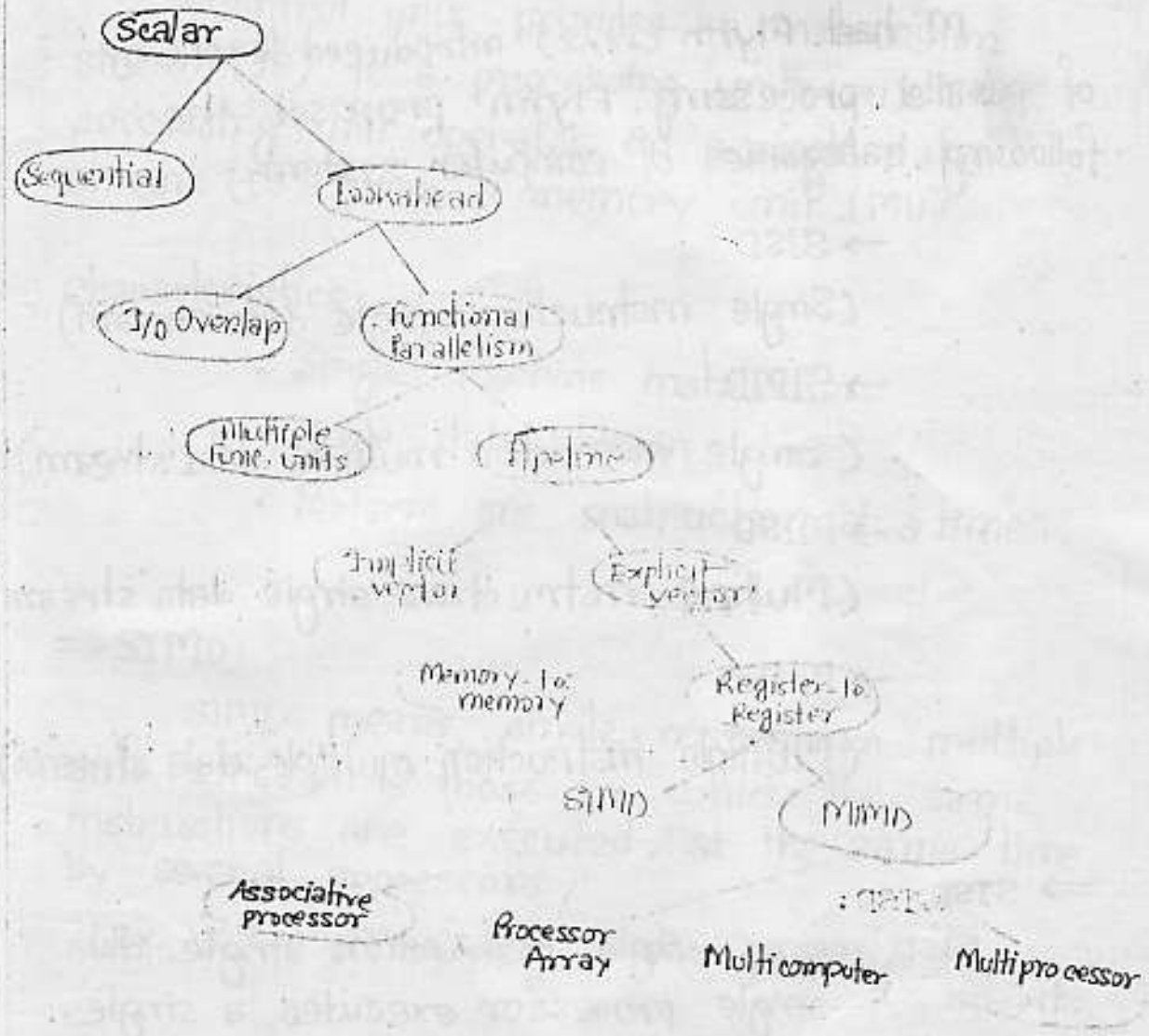


fig. Tree showing architectural evolution from sequential scalar computers to vector processors and parallel computers

Flynn's Classification:

Michael Flynn (1972) introduced taxonomy of parallel processing. Flynn proposed the following categories of computer system.

- → SISD
  (Single instruction single data stream)
- → SIMD
  (Single instruction multiple data stream)
- → MISD
  (Multiple instruction single data stream)
- → MIMD
  (Multiple instruction multiple data stream)

⟹ SISD:

SISD means single instruction single data stream. A single processor executes a single instruction stream to operate on data stored in a single memory.

Ex. Uniprocessor.

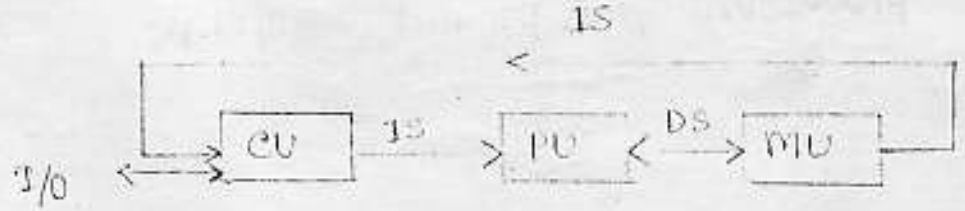


fig. SISD Uniprocessor Architecture

Operation:

Control unit provides an instruction stream (IS) to a processing unit (PU). The processing unit operates on a single data stream (DS) from a memory unit (MU).

Characteristics:

- Single machine instruction.
- Single data stream.
- Perform one instruction at a time.

=> SIMD:

SIMD: means single instruction multiple data stream, is those in which the same instructions are executed, at the same time by several processors.

Ex. Vector processor & Array processor.

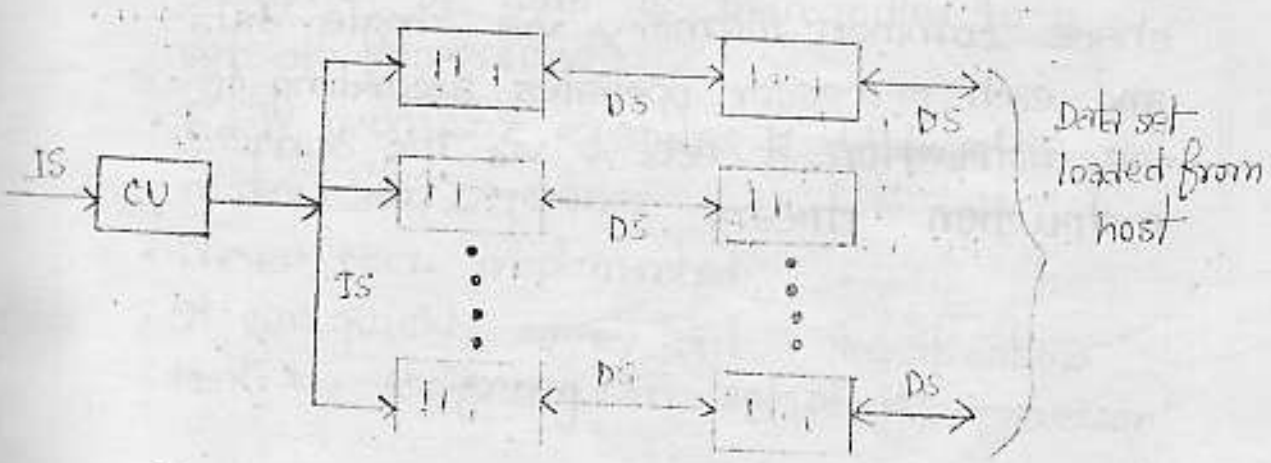


Fig. SIMD Architecture (with distributed Memory)

Operation:

With a SIMD there as a single control unit (CU). Now feeding a single instruction stream to a multiple processor elements (PE). Each processing element may have its own dedicated memory.

characteristics:

- Control simulation execution
- Single machine instruction
- Lockstep basis.
- Number of processing elements.
- Each processing elements has associated data memory.

⇒ MISD:

The multiple instruction single data stream architecture the processors can share common memory via single data and each processor operates according to the instruction it receive via it's own instruction stream.

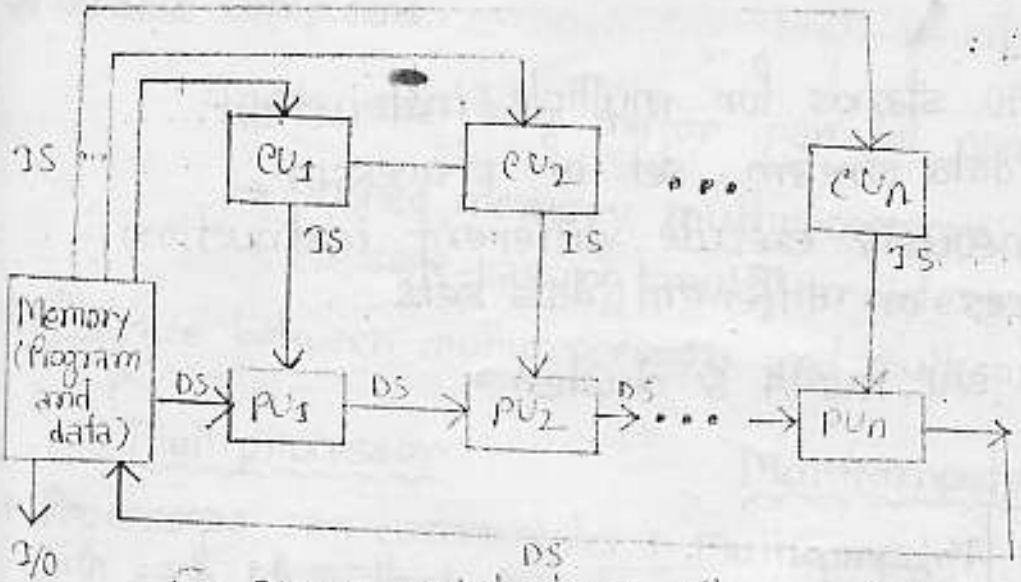


fig. MISD architecture (the cyclotic array)

Operation:

With the MISD, there are several instruction perform on a data item simultaneously. One memory is connecting of several processor which is operates a single data stream.

Characteristics:

- Sequence of data is transmitted to a set of processors.
- Each processor executes a different instruction sequence.
- Never been implemented
- It can quickly carry out a classification task by assigning an act of it's processor.

⇒MIMD:

MIMD stands for multiple instruction multiple data stream. set of processor simultaneously execute different instruction sequences on different data sets.

EX. SAP, NUMA & cluster.

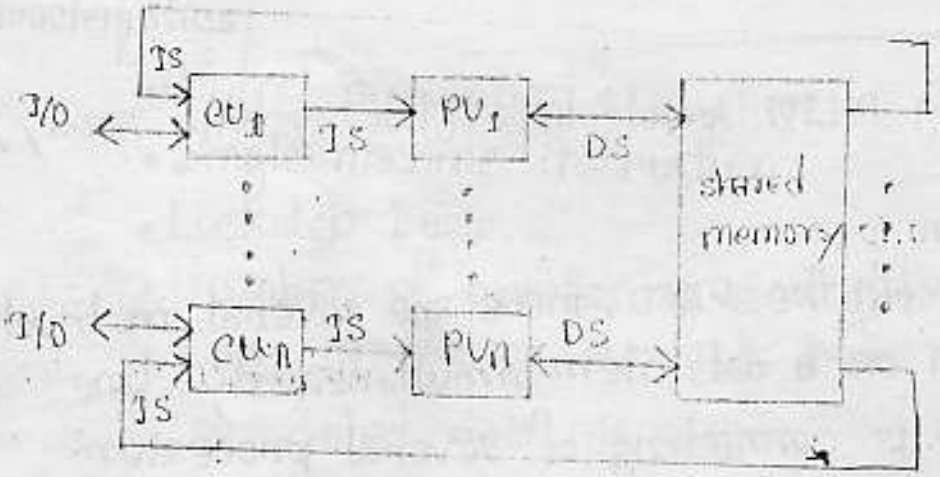


fig. MIMD architecture (with shared memory)

Operation:

With the MIMD, there are multiple control units, each feeding a sequence separate instruction stream to it's own processing elements. Several data stream performs with several memory.

Characteristics:

- Simultaneously execute different instruc sequences on different data sets.

Parallel Computers:

There are two major parallel computers.

→ Shared-memory multiprocessors
→ Message-passing multicomputers.

Difference between multiprocessor and multicomputers.

Multiprocessor	Multicomputer
1. Processors can communicates with each other through shared variables in a common memory.	1. Multicomputer used distributed memory.
2. There exist three shared memory multiprocessors models. These are— • UMA • NUMA & • COMA	2. Message-passing multicomputers are existed.
3. The system interconnects by using a common bus, a crossbar switch or multi-stage network.	3. Multicomputers use hardware routers to pass message. A computer node is attached to cache router.
4. A major shortcoming is the lack of scalability. Latency tolerance for remote memory access is also a major limitation.	4. Distributed multicomputers are more scalable but less programmable due to added communication protocols.
5. Representation multiprocessor system: Sequential symmetry S-81. IBM system/390 IBM ES/9000 BBN TC-2000	5. Representation Multicomputer system: Intel program XP/S Neube/2 6480

shared-Memory Multiprocessors:

There are three shared-memory multiprocessor models:

[a] The uniform-memory-access (UMA)
[b] The nonuniform-memory-access (NUMA)
[c] The cache-only memory architecture model (COMA).

UMA:

In a UMA multiprocessor model, the physical memory is uniformly shared by all the processors. All processors have equal access time to all memory words, which is why it is called uniform memory access. Each processor may use a private cache. Fig shows the following UMA model-

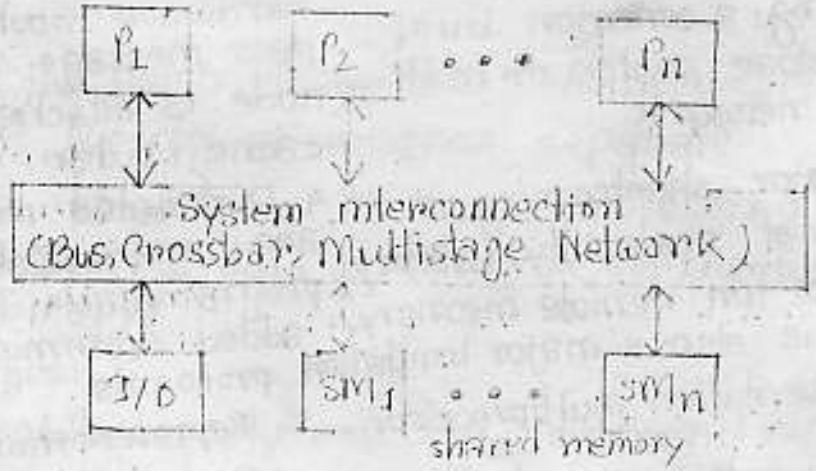


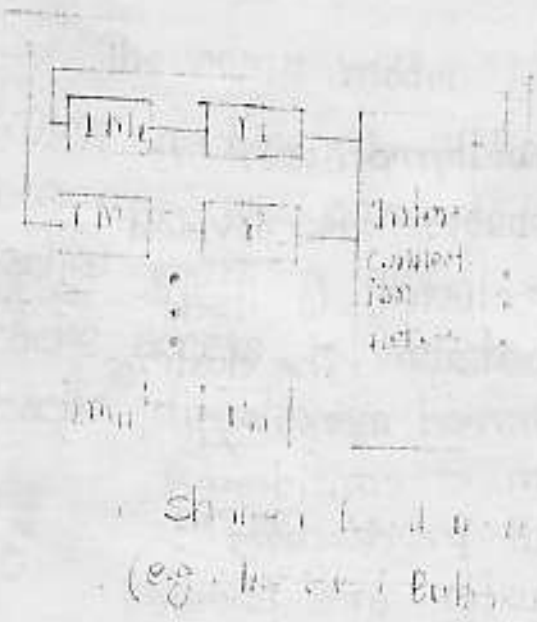
Fig. The UMA multiprocessor model

The UMA model is suitable for general-purpose and time-sharing applications by multiple users. It can be used to speed up the execution of a single-large program in time critical applications.

Ex. The sequent symmetry S-81.

NUMA Model:

A NUMA multiprocessor is a shared-memory system in which the access time varies with the location of the memory word. Two NUMA machine models are depicted in fig. (a) and (b).



The shared memory is physically distributed to all processors, called local memories. The collection of all local memories forms a global address space accessible by all processors, it is faster to access a local memory with a local processor.

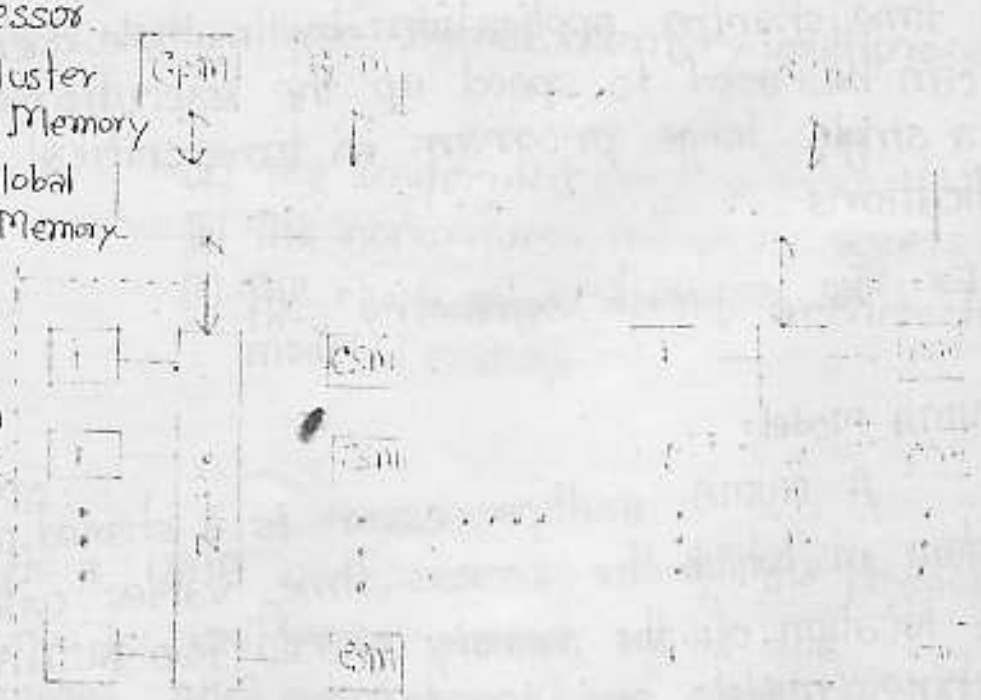
Ex: BBN Butterfly, Cedar System

Regist
regis
and
archit

SIMD
is tho
execu
proce
EX:

MIMD
sets
differ
data
EX

Legends:
P- Processor
CSM- cluster Shared Memory
GSM-Global shared Memory
CIN- cluster Interconnection Network

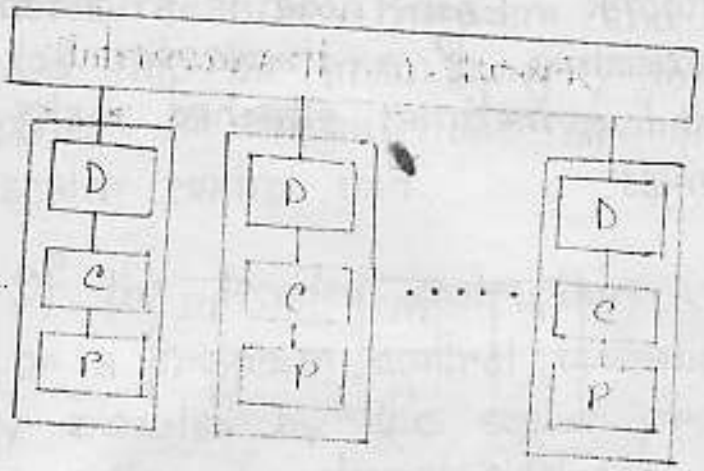


A hierarchically structured multiprocessor is modeled in fig(b). The processors are divided into several clusters. Each cluster is itself an UMA or NUMA multiprocessor. The clusters are connected to global shared memory modules. The entire system is considered a NUMA multiprocessor. All processors belonging to the same cluster are allowed to uniformly access the cluster shared memory modules.

All clusters have equal access to the global memory. The access time to the cluster memory is shorter than that to the global memory.

COMA Model:

A multiprocessor using cache-only memory assumes the COMA model.



Legends:
P = Processor
C = Cache
D = Diredory

Fig. The COMA model of a multiprocessor

The COMA model is a special case of a NUMA machine, in which the distributed main memories are converted to caches. All the caches from a global address space. Remote cache access is assisted by the distributed cache directories.

Ex. KSR-1

Distributed-Memory Multicomputers:

A distributed-memory multicomputers system consists of multiple computers, often called nodes, interconnected by a message-passing network. Each node is an autonomous computer consisting of a processor, local memory, and sometimes attached disks or I/O peripherals.

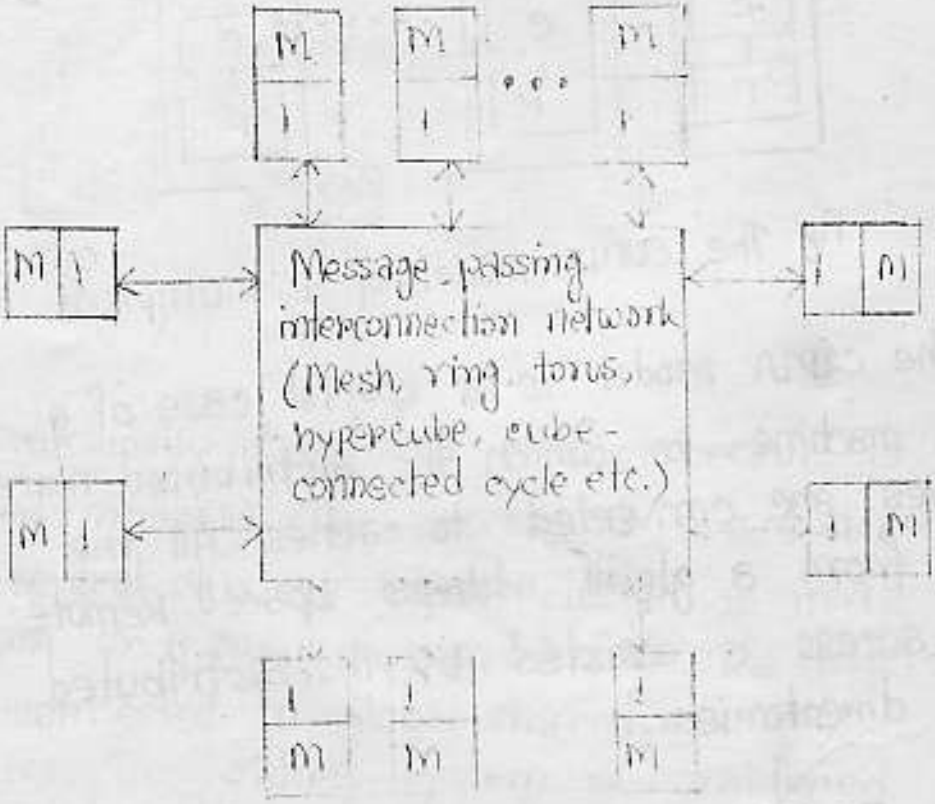


Fig. Generic model of a message-passing multicomputer.

The message-passing network provides point-to-point static connections among the nodes. All local memories are private and are accessible only by local processors. For this reason, traditional multicomputers have been called no-remote-memory-access (NORMA) machines.

Vector Supercomputers:

A vector computer is often built on top of a scalar processor. As shown in fig (a), the vector processor is attached to the scalar processor as an optional feature. Program and data are first loaded into the main memory through a host computer. All instructions are first decoded by the scalar control unit.

1. If the decoded instruction is a scalar operation or a program control operation, it will be directly executed by the scalar processor using the scalar functional pipelines.

2. If the instruction is decoded as a vector operation, it will be sent to the vector control unit. This control unit will supervise the flow of vector data between the main memory and vector functional pipelines. The vector data flow is coordinated by the control unit. A number of vector functional pipelines may be built into a vector processor. Two pipeline vector supercomputer models are described below —

① Register-to-register architecture

② Memory-to-memory architecture.

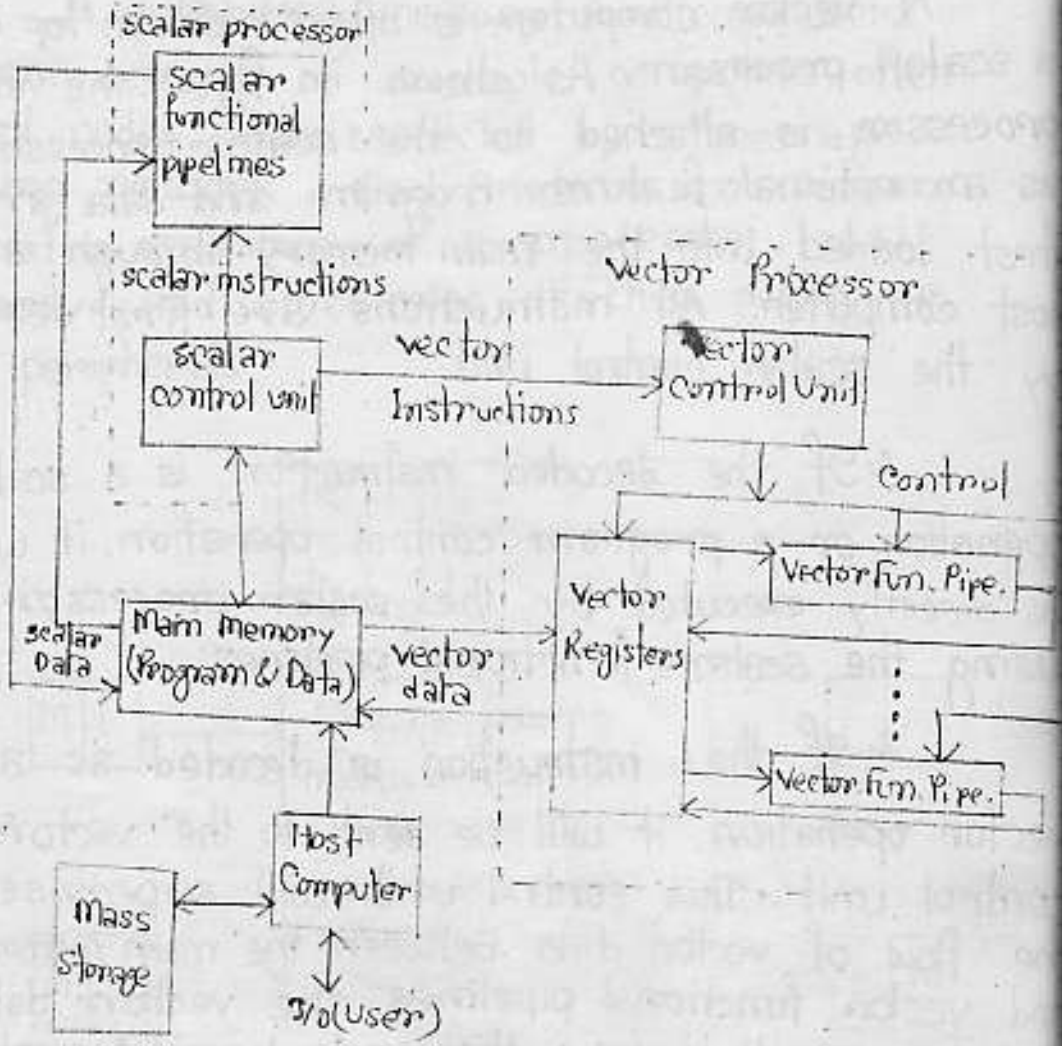


Fig. The architecture of a vector supercomputer

Fig. shows a **register_to_register** architecture. Vector registers are used to hold the vector operands, intermediate and final vector results. The vector functional pipelines retrieve operands from and put the length of each vector register is usually fixed, say, sixty-four 64-bit component registers in a vector

register in a Cray Series supercomputer.

On general, there are fixed numbers of vector registers and functional pipelines in a vector processor.

A memory-to-memory architecture differs from a register-to-register architecture in the use of vector stream unit to replace the vector registers. Vector operands and results are directly retrieved from the main memory in superwords, say, 512 bits as in the Cyber 205.

SIMD Machine Model:

An operational model of an SIMD computer is specified by a 5-tuple:

$$M = (N, C, I, M, R)$$

where,

① N is the number of processing elements (PEs) in the machine. For example, the Illiac IV has 64 PEs and the connection Machine CM-2 uses 65,536 PEs.

② C is the set of instructions directly executed by the control unit (CU), including scalar and program flow control instructions.

③ I is the set of instructions broadcast by the CU to all PEs for parallel execution. These include arithematic, logic, data routing, masking and other local operations executed by each active PE over data within that PE.

④ M is the set of masking schemes, where each mask partitions the set of PEs into enable and disable subsets.

⑤ R is the set of data-routing functions specifying various patterns to be set up in the interconnection network for inter-PE communica

Example 1.3:

Operational specification of the MasPar MP-1 computer.

4. PRAMS Models:

A parallel random-access machine (PRAM) model has been developed by Fortune and Wyllie (1978) for modeling idealized parallel computers with zero synchronization or memory access overhead. It is used for parallel algorithm development and for scalability and complexity analysis.

An n-processor has a globally addressable memory. The shared memory can be distributed among the processors.

The n-processors operate on a synchron read-memory, compute and write memory cycle. With shared memory, the model must specify how concurrent read and concurrent write of memory are handled.

Four memory-update options are possible:

- Exclusive read (ER):

This allows at most one processor to read from any memory location in each cycle.

- Exclusive write (EW):

This allows at most one processor to write into a memory location at a time.

- Concurrent read (CR):

This allows multiple processors to read the same information from the same memory cell in the same cycle.

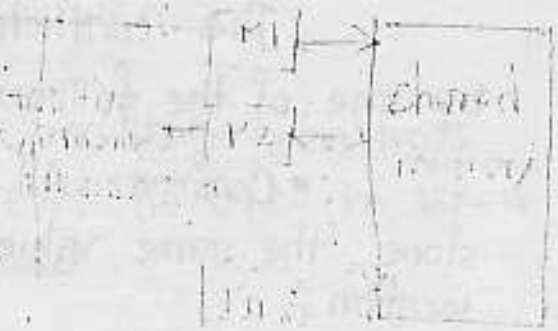
- Concurrent write (CW):

This allows simultaneous writes to the same memory location. In order to avoid confusion, some policy must be set up to resolve the write conflicts.

PRAM variants:

There are four variants of PRAM model depending on how the memory reads and writes handled —

① EREW - PRAM model.
② CREW - PRAM model
③ ERCW - PRAM model
④ CRCW - PRAM model.



① The EREW-PRAM model :

This model forbids more than one processor from reading or writing the same memory cell simultaneously.

② The CREW-PRAM model :

The write conflicts are avoided by mutual exclusion. Concurrent reads to the same memory location are allowed.

③ The ERCW-PRAM model :

This allows exclusive read or concurrent writes to the same memory location.

④ The CRCW-PRAM model :

This model allows either concurrent reads or concurrent writes at the same time.

Resolving of write conflicts :

The conflicting writes are resolved by one of the following four policies :

- Common : All simultaneous writes store the same value to the hot-spot memory location.

- Arbitrary:

Any one of the values written may remain; the others are ignored.

- Minimum:

The value written by the processor with the minimum index will remain.

- Priority:

The values being written are combined using some associative functions, such as summation or maximum.

Lec 7
10-10-11

Last part of Lec-5.

CREW	EREW
① Allows either concurrent read or concurrent writes at the same time.	① Allows either exclusive read or concurrent writes to the same memory location
② Most popular	② Less popular
③ CREW algorithm runs faster.	③ slower
④ Writing conflicts are occured in CREW model.	④ Writting conflicts aren't occured in EREW model.

CHAPTER-2

Program & Network Properties

Bernstein's Conditions:

- In 1966, Bernstein revealed a set of conditions based on which two processes can execute in parallel.

- We define the input set I_i of a process P_i as the set of all input variables needed to execute the process.

- Similarly the output set O_i consists of all output variables generated after execution of the process P_i.

Input variables are essentially operands which can be fetched from memory to registers and output variables are the results to be stored in working registers or memory locations.

- Now, consider two processes P_1 and P_2 with their input sets I_1 and I_2 and output of sets O_1 and O_2, respectively. These two processes can execute in parallel and are denoted $P_1 \parallel P_2$

if they are independent and do not create confusing results.

formally, these conditions are stated as follows:

$$\left.\begin{aligned} I_1 \cap O_2 &= \phi \\ I_2 \cap O_1 &= \phi \\ O_1 \cap O_2 &= \phi \end{aligned}\right\}$$

These three equations are known an Bernstein's conditions.

Lec-8
24.10.11

Data Dependence:

The ordering relationship between statements is indicated by the data dependence. five types of data dependence are define below:

(1) Flow dependence:

A statement S2 is flow-dependent on statement S1 if an execution path exists from S1 to S2 and if at least one output of S1 feeds in as input to S2. Flow dependence is denoted as $S1 \longrightarrow S2$.

(2) Antidependence:

statement S_2 is antidependence on statement S_1 if S_2 follows S_1 in program order and if the output of S_2 overlaps the input to S_1. A direct arrow crossed with a bar as in $S_1 \longleftrightarrow S_2$ indicates antidependence from S_1 to S_2.

(3) Output dependence:

Two statements are output-dependent if they produce (write) the same output variable. $S_1 \circ\!\!\rightarrow S_2$ indicates output dependence from S_1 to S_2.

(4) I/O dependence:

Read and write are I/O statements. I/O dependence occurs not because the same variable is involved but because the same file is referenced by both I/O statements.

(5) Unknown dependence:

The dependence relation between two statements cannot be determined in the following situations:

- The subscript of a variable is itself subscribed (indirect addressing).
- The subscript does not contain the loop index variable.
- A variable appears more than once with subscripts having different coefficients of the loop variable.
- The subscript is nonlinear in the loop index variable.

When one or more of these conditions exist, a conservative assumption is to claim unknown dependence among the statements involved.

Ex 2.1. Cosider the following code fragment of four instructions:

```
S1 :  Load R1, A        / R1 ← Memory (A) /
S2 :  Add R2, R1        / R2 ← (R1) + (R2) /
S3 :  Move R1, R3       / R1 ← (R3) /
S4 :  Store B, R1       / Memory (B) ← (R1) /
```

⇒ S2 is flow-dependent on S1 because the variable A is passed via the register R1.

⇒ S3 is antidependent on S2 because of potential conflicts in register content in R1.

⇒ S3 is output dependent on S1 because they both modify the register R1.

Hardware Parallelism: (HP)

Hardware parallelism refers to the type of parallelism defined by the machine architecture and hardware multiplicity. Hardware parallelism is often a function of cost and performance trade offs. It displays the resource utilization, patterns of simultaneously executable operations. It can also indicate the peak performance of processor resources.

One way to characterize the parallelism in a processor is by the number of instruction issues per machine cycle. If a processor issues k instructions per machine cycle then it is called a k-issue processor.

A conventional processor takes one or more machine cycle to issue a single instruction. These types of processors are called one-issue machines/processors.

If two or more machine instructions can be issued per cycle by processor than that processor is called dual-issue machines.

Ex.①

The Intel i960CA is a three-issue processor with one arithmetic, one memory access and one branch instruction issued per cycle.

② The IBM RISC/System 6000 is a four-issue processor capable of issuing one arithmetic, one memory access, one floating-point & one branch operation per cycle.

Software Parallelism:

This type of parallelism is defined by the control and data dependence of programs. The degree of parallelism is revealed in the program profile or in the program flow graph.

Software parallelism is a function of algorithm, programming style and compiler optimization.

The program flow graph displays the patterns of simultaneously executable operations. Parallelism in a program varies during the execution period.

Example 2.3: Mismatch between software parallelism and hardware parallelism:

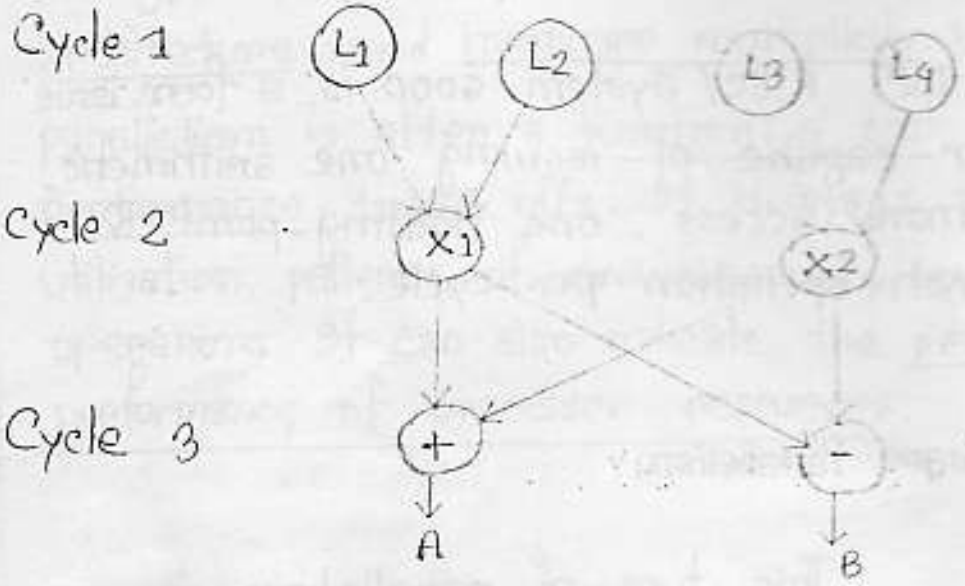


fig. 1(a) Software parallelism.

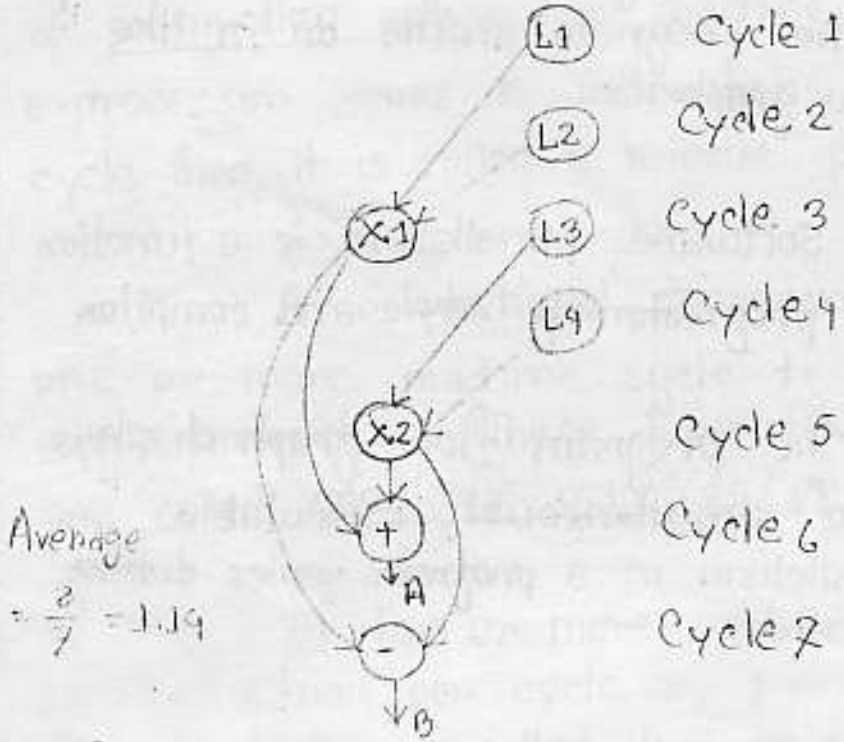


fig. 1(b) Hardware Parallelism.

There are 8 instructions such as four loads and four arithmetic operations to be executed in three consecutive machine cycles. Four load operations are performed in the first cycle, followed by two multiply operations in the second cycle and two add/subtract operations in the third cycle. Therefore, the parallelism varies from 4 to 2 in three cycles. The average software parallelism is equal to $8/3 = 2.67$ instructions per cycle in this example program.

Now, consider execution of ~~per~~ the same program by a two-issue processor which can execute one memory access (load or write) and one arithmetic (add, subtract, multiply, etc) operation simultaneously.

Therefore, the hardware parallelism displays an average value of $8/7 = 1.14$ instructions executed per cycle.

This demonstrates a mismatch between the software parallelism and the hardware parallelism.

Q. How mismatch problem can be solved?

Solution of mismatch problem between software and hardware parallelism:

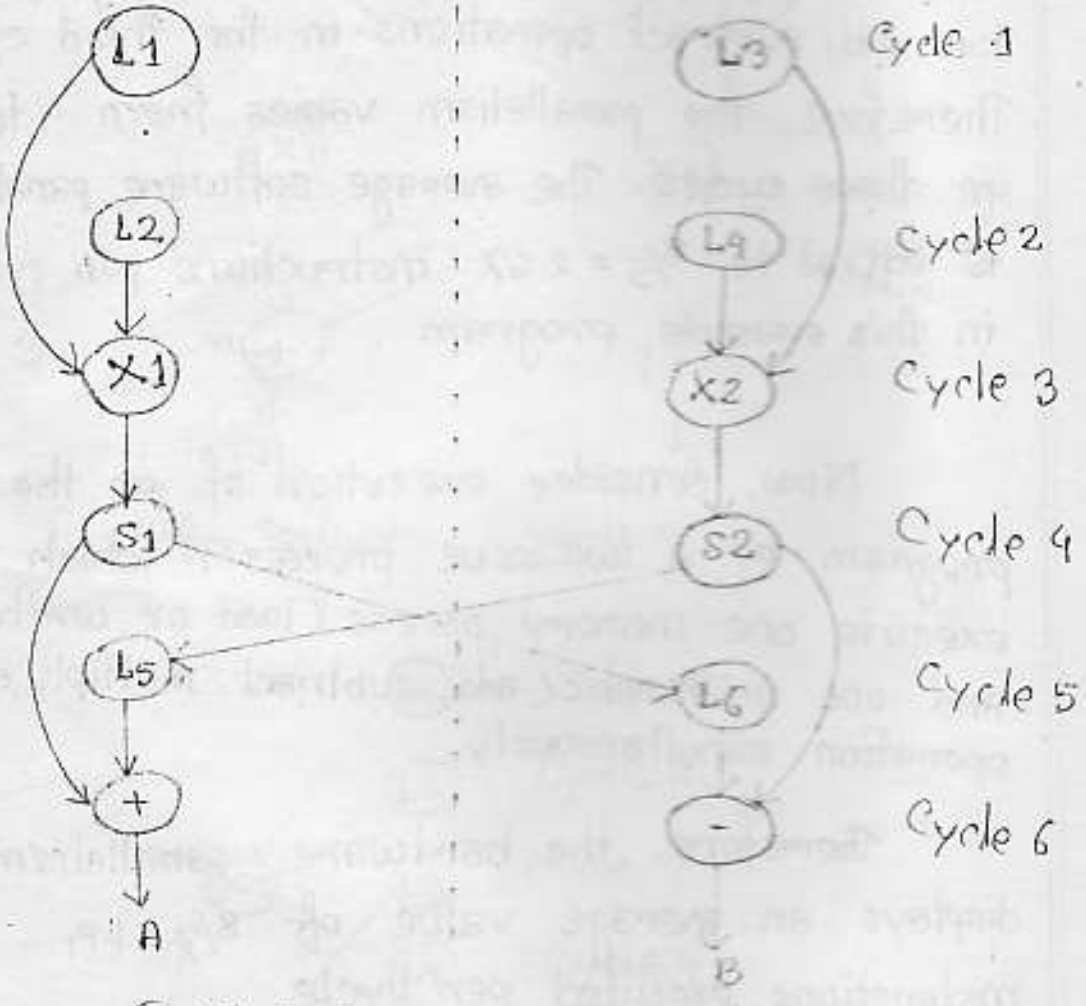


Fig (2) Dual processor execution of the program

L/s : Load / store operation
x : Multiply operation
+/- : Add / subtract operation

Let us try to match the software parallelism shown in fig 1(a) hardware platform of a dual processor system, where single-issue processors are used.

The achievable hardware parallelism is shown in fig (2) where L/S stands for load/store operations. Note that six processor cycles are needed to execute the 12 instructions by two processors. S_1 and S_2 are two inserted store operations and L_5 and L_6 are two inserted load operations. These added instructions are need for interprocessor communication through the shared memory.

The average hardware parallelism in this architecture is equal to $12/6 = 2$ instructions per cycle. So, the mismatch problem was solved by this architecture.

Furthermore, to solve the mismatch problem between software and hardware parallelism, one approach is to develop—

① Compilation support and

② Other is through hardware redesign for more efficient result.

These two approaches must co-operate with each other to produce the best result.

Grain:

When a program is partitionized into little parts, then each part of that program is called grain.

Grain Size:

"Grain Size" or "granularity" is a measure of the amount of computation involved in a software process. The simplist measure is to count the member of instructions in a grain. There are 3 types of grain size are as follows—

1. Fine grain
   (Less than 20 instructions)
2. Medium grain
   (Less than 2000 instructions)
3. Coarse grain
   (As high as tens of thousands of instructions).

Latency:

Latency is a time measure of communication overhead in curred between machine subsystems. Various latencies are—

(i) Memory latency
(ii) Communication Latency
(iii) Synchronization Latency

(i) Memory Latency:

The time required by a processor to access the memory is called memory latency.

(ii) Communication Latency:

(iii) Synchronization Latency:

The time required for two processes to synchronize with each other is called the synchronization latency.

Grain Packing:

When a program is partitionized into grains, then if those grain can not be executed in parallel then those grains brought together. i.e, we pack them. This packing process of grains are called "grain packing".

Ex 2.4: Program graph before and after grain packing.

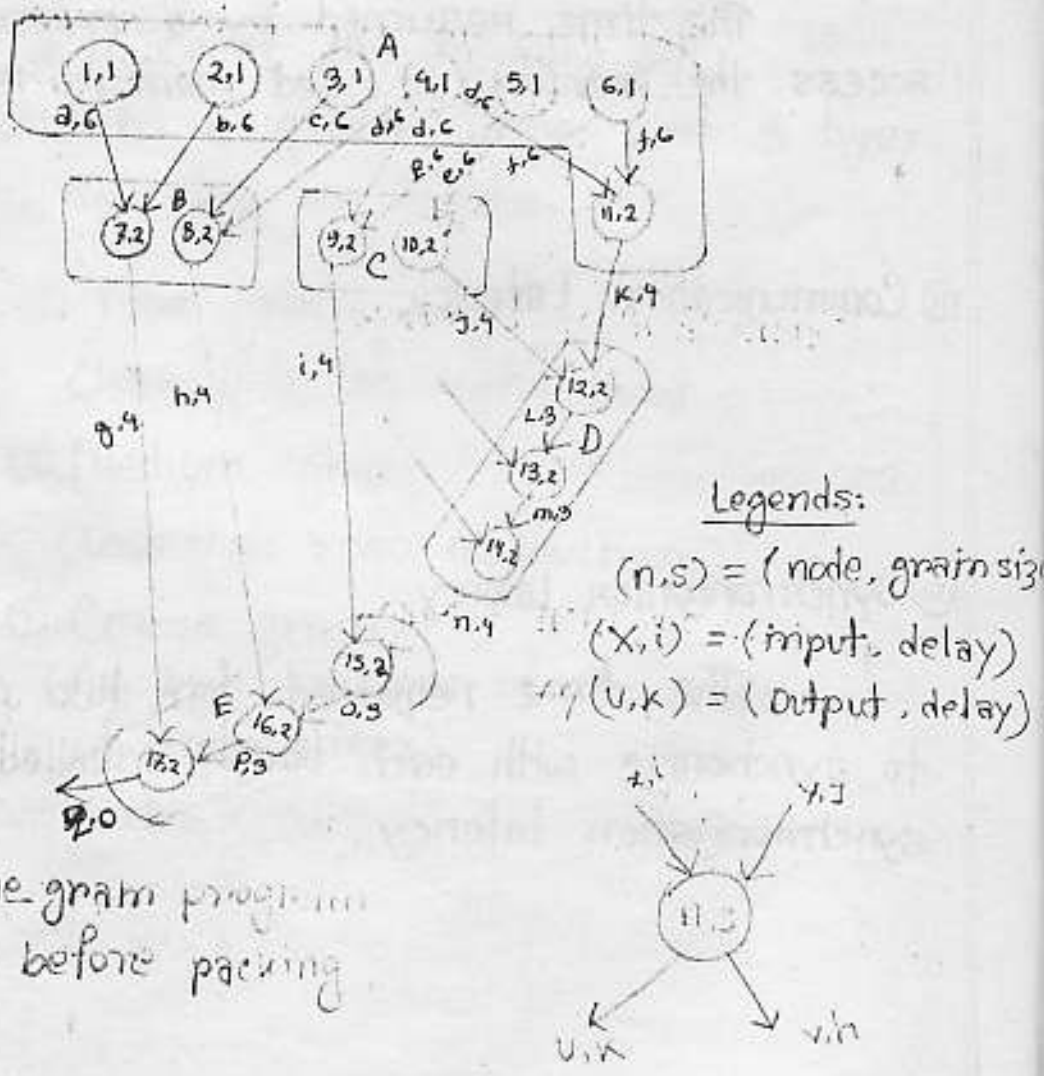


Fig (a) Fine-grain program graph before packing

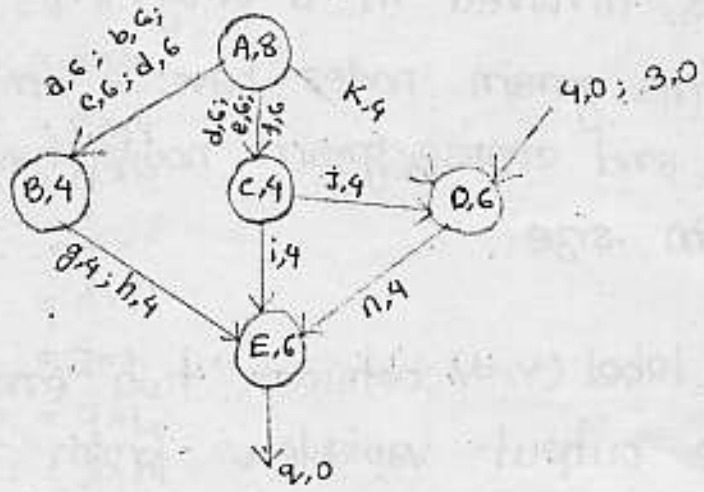


Fig (b): Coarse grain program graph after packing

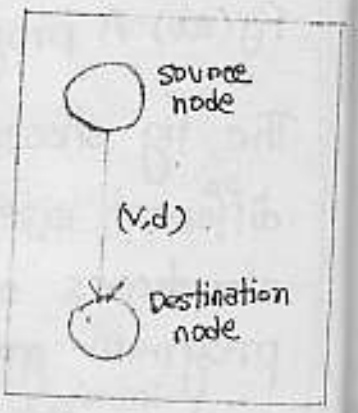
Fig (a,b): A program graph before and after grain packing

The fig shows an example program graph in two different sizes. A program graph shows the structures of a program. Each node in the program graph corresponds to a computation unit in the program. The grain size is measured by number of basic machine cycles (both processor and memory cycles) needed to execute all the operations within the node. We denote each node in fig. by a pair (n,s), where -

n is the node name and
s is the grain size of the node.

Thus grain size reflects the number of computations involved in a program segment. fine grain nodes have a smaller grain size, and coarse grain nodes have a larger grain size.

The edge label (v, d) between two end nodes specifies the output variable v from the source nodes or the input variable to the destination node, and the communication delay d between them.



There are 17 nodes in the fine-grain program graph (fig-a) and 5 in the coarse grain program graph (fig-b). The coarse grain graph node is obtained by combining multiple fine-grain grain nodes. The fine grain corresponds to the following program—

Var a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q

Begin:

1. a := 1
2. b := 2
3. c := 3

4. d: =4
5. e: =5
6. f: =6
7. g: = a×b
8. h: = c×d
9. i: = d×e
10. j: = e×f
11. k: = d×f
12. l: = j×k
13. m: = 4×l
14. n: = 3×m
15. o: = n×i
16. p: = o×h
17. q: = p×g

Nodes 1,2,3,4,5 and 6 are memory reference (data fetch) operation. Each takes one cycle to address and 6 cycles to fetch from memory. All remaining nodes (7 to 17) are CPU operations, each requiring two cycles to complete.

The node (A,8) in fig(b) is obtained by combining the nodes (1,1), (2,1), (3,1), (4,1), (5,1), (6,1) and (11,2) in fig 8. The grain size 8 of node A is the summation of grain sizes (1+1+1+1+1+1+2) = 8 being combined.

Node Duplication:

In order to eliminate the idle time and to further reduce the communication delays among processors, one can duplicate some of nodes in more than one processor.

(fig. a) shows a schedule without duplicating any of the five nodes. This schedule contains idle time as well as long interprocessor delays — (8 units) between P_1 and P_2.

on (fig. b) shows. node A is duplicated into A' and assigned to P_2 besides retaining the original copy A in P_1.

Similarly, c' is copied into P_2 besides the original node C in P_2, that in (fig. a). The reduction in schedule time is caused by elimination of the (a, 8) and (c, 8) delays between the two processors.

The new schedule shown in fig (2) is almo
so/. shor
th

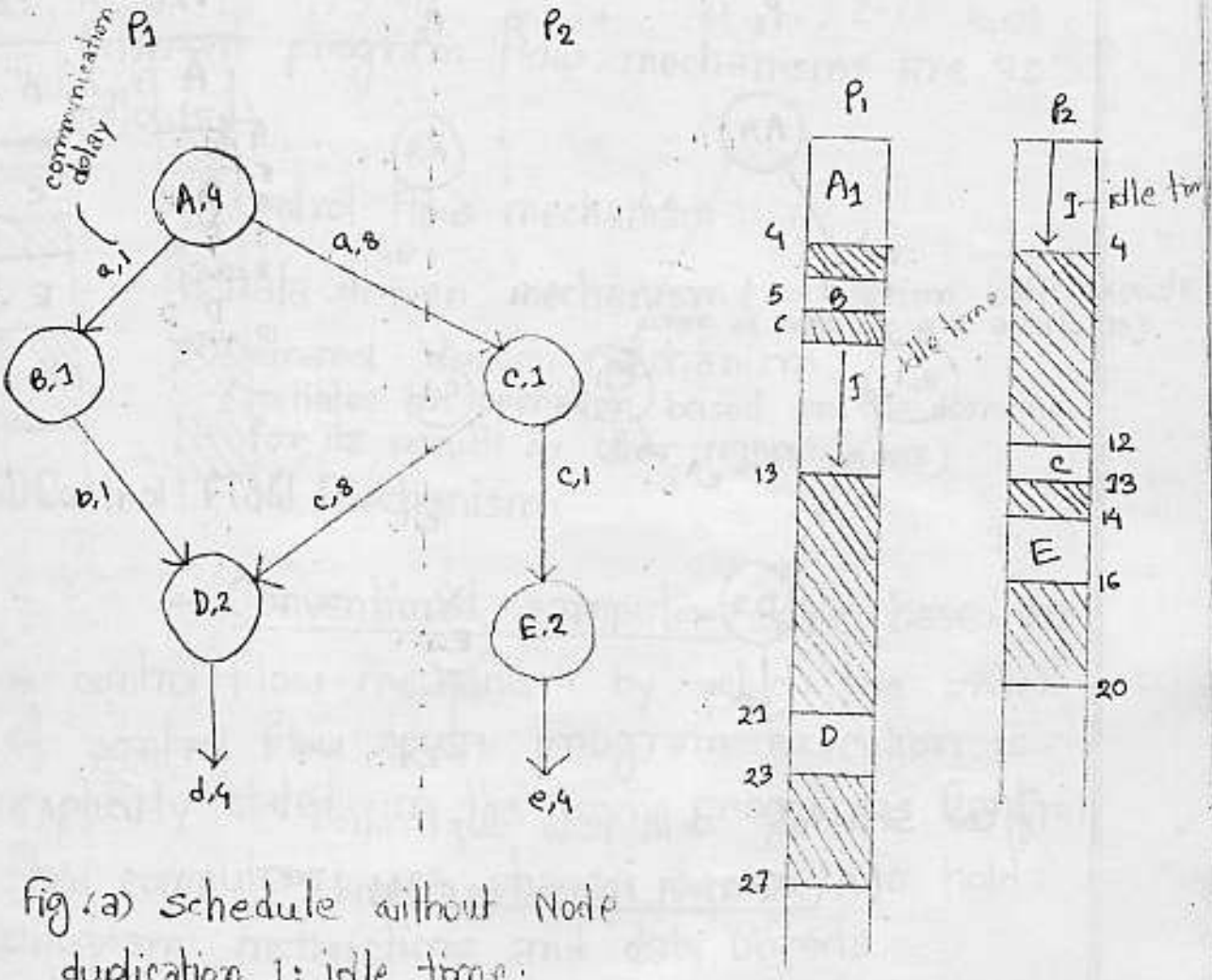


fig:(a) schedule without Node duplication 1: idle time:
shade Area: Communication delays.

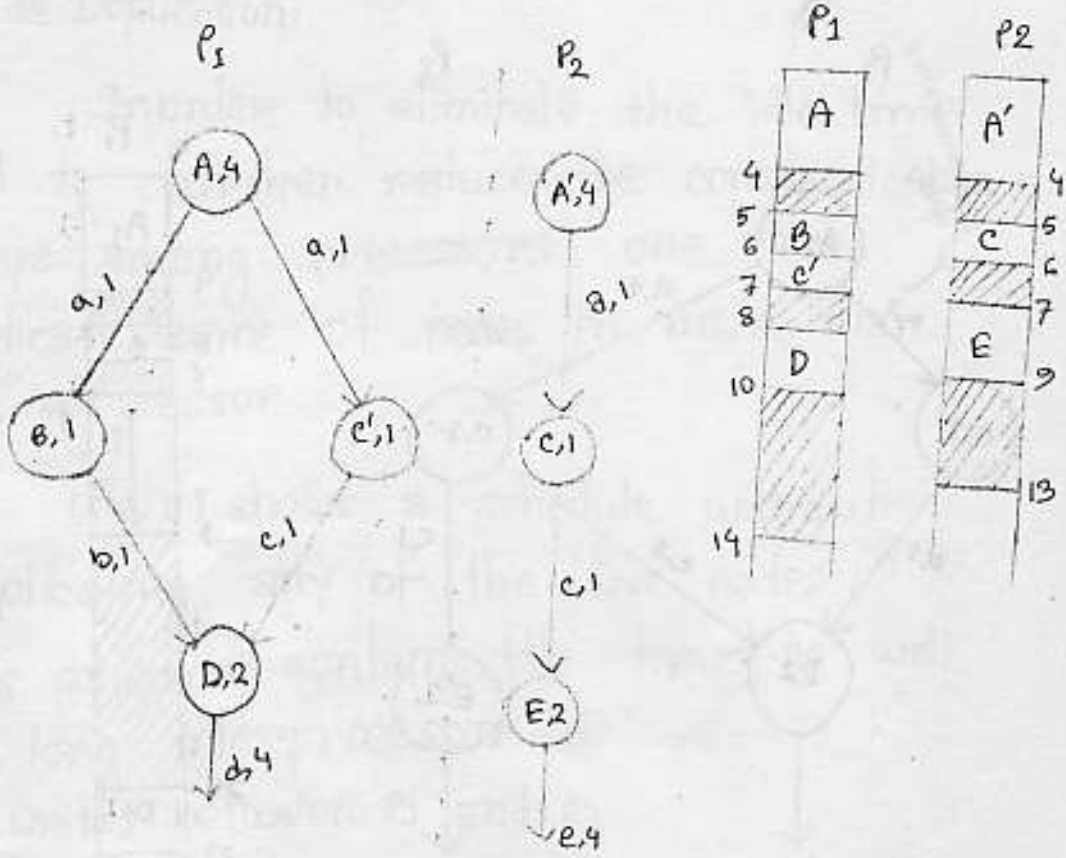


fig (b) Schedule with node duplication
(A->A and A'; c->c and c')

Program Flow Mechanisms:

Different program flow mechanisms are as follows—

(i) Control flow mechanism

(ii) Data driven mechanism (instruction will execute when all operands are available)

(iii) Demand - driven mechanism (initiates an operation based on the demand for its results by other computations)

(i) Control Flow Mechanism:

— Conventional computers are based on a control flow mechanism by which the order of ~~control flow~~ ~~comply~~ program execution is explicitly stated in the user programs. Control flow computers use shared memory to hold program instructions and data objects.

Variables in the shared memory are update by many instructions. The execution of one instruction may produce side effects on other instructions since memory is shared

23.11.11

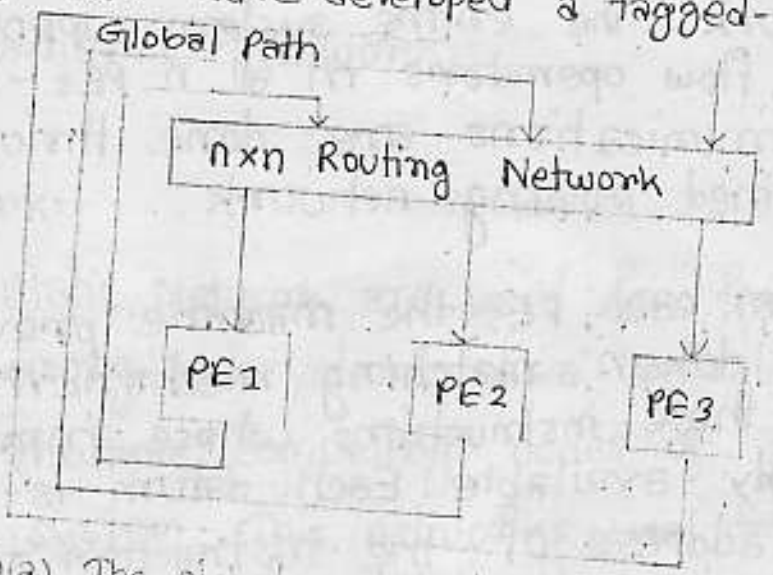
Comparison between Control-flow, data-flow and Reduction (demand-driven) computer architectures:

Machine model	Control Flow	Data Flow (data-driven)	Reduction (Demand-Driven)
Basic defn	Conventional computation token of control indicates when a statement should be executed.	Eager evaluation; statements are executed when all of their operands are available.	Lazy evaluation, statements are executed only when their results is required for another computa
Advantages	full control	Very high potential for parallelism	Only required instructions are executed.
	Complex data and control structures are easily implemented.	High throughput	High degree of parallelism.
		free from side effects.	Easy manipulatio of data structure
Disadvantages	Less efficient	Time lost waiting for unneeded arguments.	Does not support sharing objects with changing local state.
	Difficult in programming.	High control overhead.	Time needed to propagate demand tokens.
	Difficult in preventing runtime error.	Difficult in manipulating data structures.	

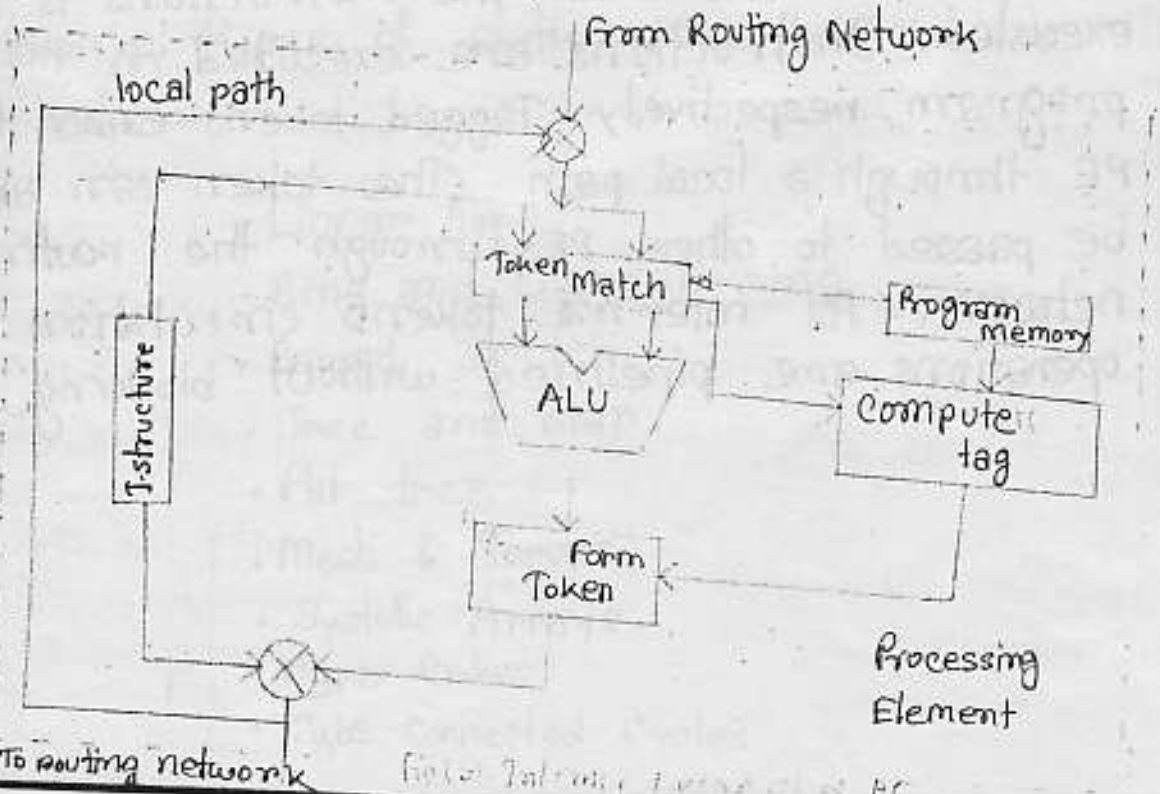
The Architecture of Dataflow Computer:

Data Flow Computer: In a data flow computer, the execution of an instruction is driven by data availability instead of being guided by a program counter.

Data Flow Computer Architecture: Arvind and his associates at MIT have developed a tagged-token



Fig(a). The global



architecture for building a data flow computers fig (a) & (b) shows the global architecture and interior design of a processing element respectively.

The global architecture consists of n processing elements (PEs) interconnected by an n×n routing network. The entire system supports pipelined data flow operators in all n PEs. Inter-PE communications are done through the pipelined routing network.

Within each PE, the machine provides a low-level token matching mechanism which dispatches only those instructions whose input data are already available. Each datum is tagged with the address of the instruction to which it belongs & the context in which the instructions is being executed. Instructions are executed in the program respectively. Tagged tokens enter the PE through a local path. The token can also be passed to other PEs through the routing network. All internal tokens circulation operations are pipelined without blocking.

Network properties & routing:

The topology of an interconnection network can be as follows—

① Static Network.

② Dynamic Network.

Static Network:

static Network are used for fixed connections among subsystems of a centralized system or multiple computing nodes of a distributed system. This networks use direct links which are fixed once built. There are 9 types of static networks are as follows—

- Linear Array
- Ring and chordal ring
- Barrel shifter
- Tree and star
- Fat tree
- Mesh & Torous
- Systolic Arrays
- Hyper Cubes
- Cube Connected Cycles.

•Linear Array:

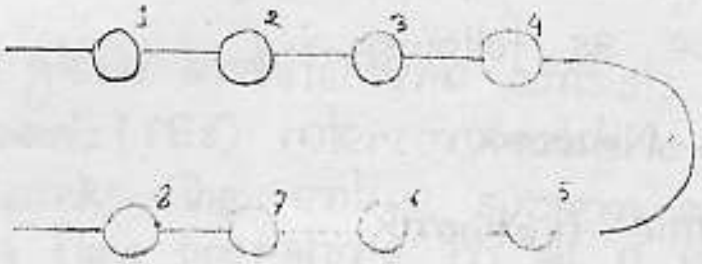


Fig. Linear array

Node no = 8
Node connection
link = (8-1) = 7
Internal node degree = 2
External " " = 1
bisection width b = 1

This is one-dimensional network in which N nodes are connected by N-1 links in a line. fig. shows the depict. Internal nodes have degree 2 and the terminal nodes have degree 1. The diameter is N-1, which is rather long for large N, the bisection width b=1. Linear arrays are the simplest connection topology. The structure is not symmetric and prosses a communication efficiency when N become very large.

• Ring & Chordal ring:

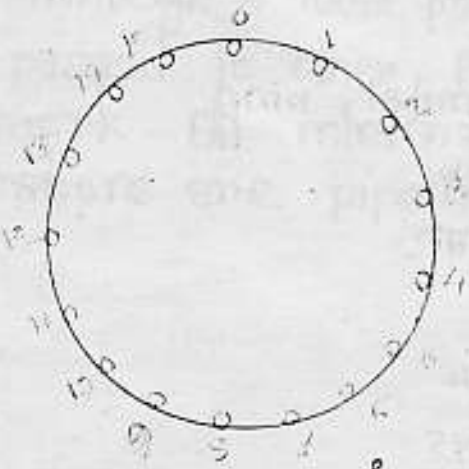


Fig. Ring

A ring is obtained by connecting the two terminal nodes of a linear array with one extra link. (A ring can be unidirectional or bidirectional. It is symmetric with a constant node degree of 2. The diameter is $\lfloor N/2 \rfloor$ for a bi-directional ring and N for unidirectional rings.

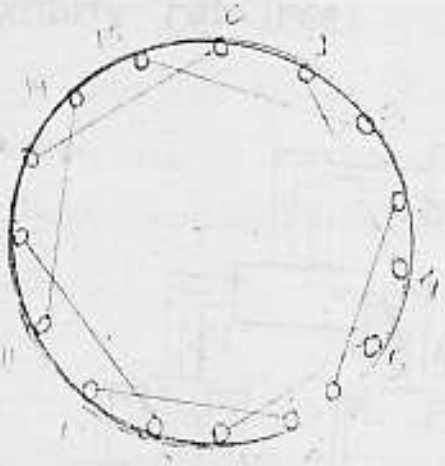


fig. Chordal ring of degree 3

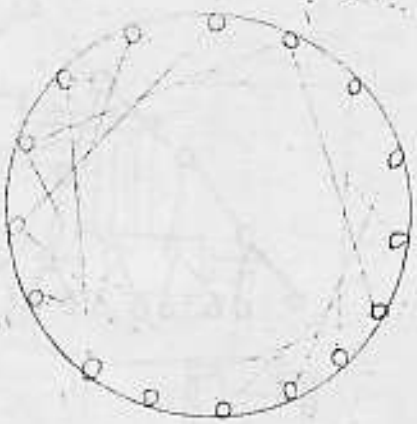


fig. chordal ring of degree 4

By increasing the node degree from 2 to 3 or 4, we obtain two chordal ring which is shown in fig. (One or two extra links are added to produce the two chordal ring).

- Barrel shifter:

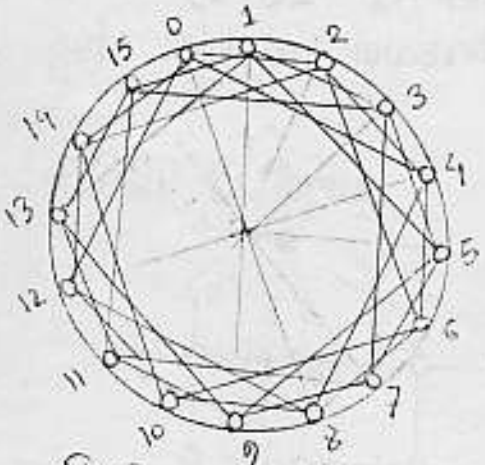


fig. Barrel shifter

fig. shows the barrel shifter network which has $N = 16$ nodes. The barrel shifter is obtained from the ring by adding extra links from the node to those nodes having a distance equal to an integer power of 2.

• Tree:

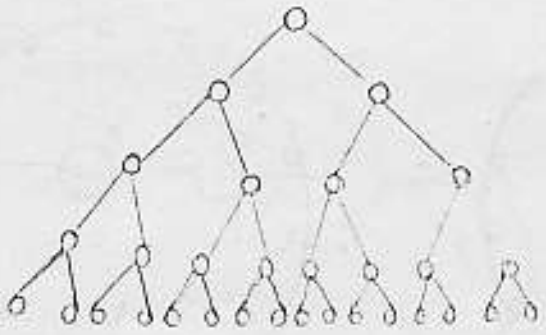


fig Binary Tree

A "binary tree" of 31 nodes in five levels shown in fig. In general a k-level, completely balanced binary tree should have $N = 2^k - 1$ nodes. The maximum node degree is 3 and the diameter is $2(k-1)$.

• star:

The "star" is a two level tree with a high node degree of $d = N-1$ as shown in fig. and a small constant diameter of 2. The star architecture has been used in systems with a centralized supervisor node.

Fig. star

• Binary Fat Tree:

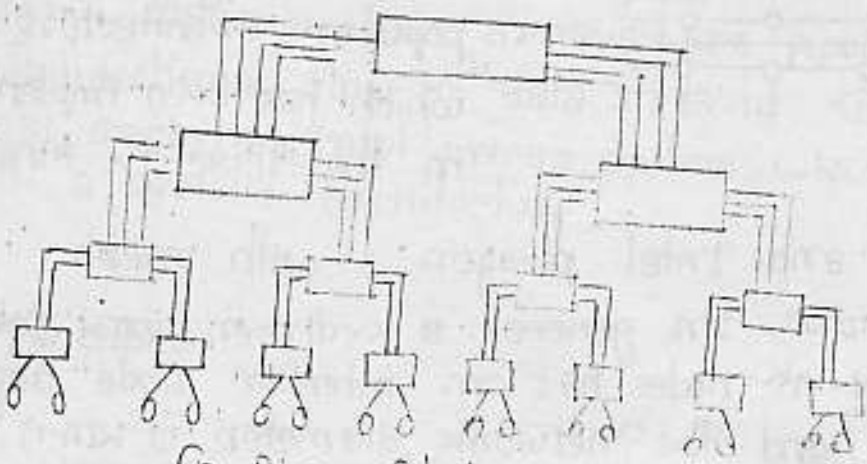


Fig. Binary Fat tree

The "Fat Tree" introduced by Leiserson in 1985. The channel width of a fat tree increases as we ascend from leaves to the root. The fat tree is more like real tree) in that branches get thicker towards the root.

• Mesh & Torus:

Lec-16
28.11.11

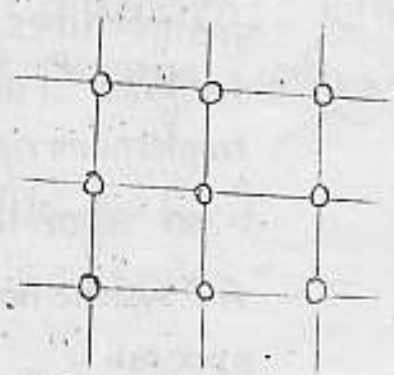


Fig. Mesh

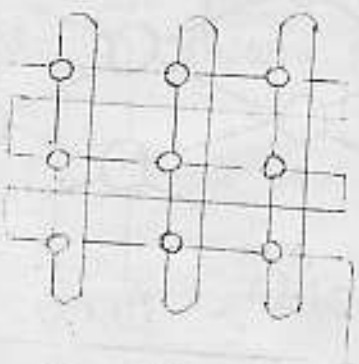


Fig. 3D Mesh

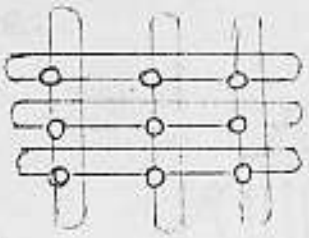


fig. Torus

A 3×3 mesh network is shown in fig. This is a popular architecture which has been implemented in the Illiac IV, MPP, DAP, CM-2 and Intel paragon with mesh variations. In general, a k-dimensional mesh with $N = n^k$ nodes has an interior node degree of $2k$ and the network diameter is $k(n-1)$.

The tropology combines the ring and meshes and extends to higher dimension. fig. shows the torus network.

- Systolic Arrays:

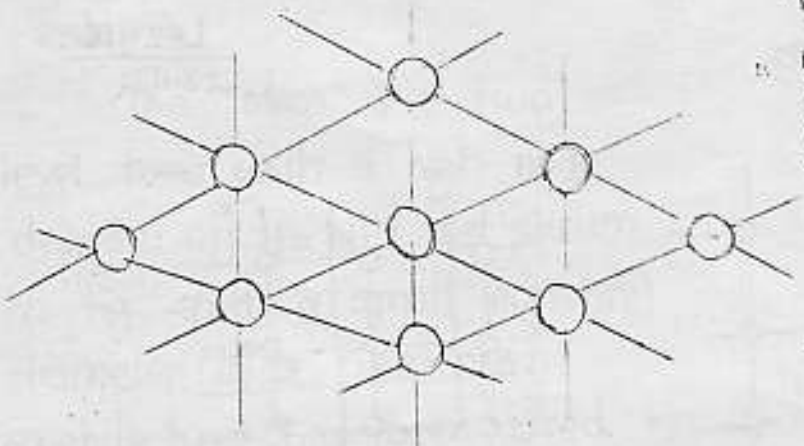


Fig. Systolic Arrays

This is a class of multidimentional pipelined array architectures designed for implementing fixed algorithms. A "systolic Array" specially

designed for performing matrix-matrix multiplication. The interior node degree is 6 in example. In general, static systolic arrays are pipelined with multidirectional flow of data streams. The commercial machine Intel iWrap system was designed with a systolic architecture.

• Hypercubes:

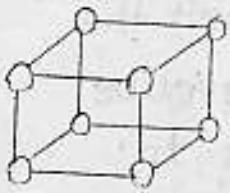
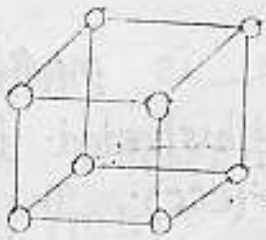


Fig. 3-cube



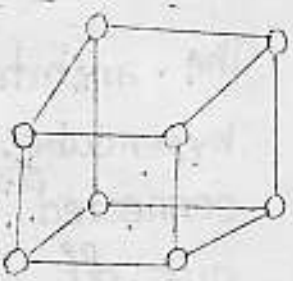


Fig 4 cube

This is a binary A-cube architecture, which has been implemented in the iPSC, nCUBE & CM-2 systems. In general, an n-cube consists of $N = 2^n$ nodes spanning along n dimensions with two nodes per dimensions. A 3-cube with 8 nodes in shown in fig.

Cube-Connected cycles:

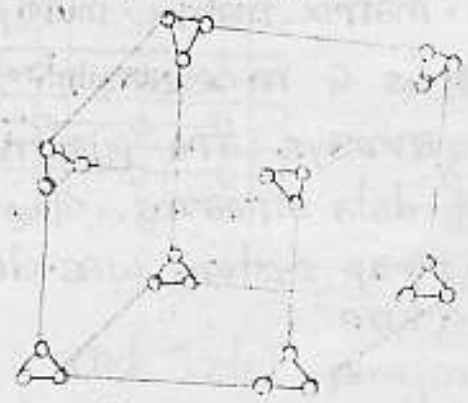


Fig. Cube connected cycle

The architecture is modified from the hypercube. An illustrated fig. a 3-cube connected cycle (ccc). The idea is to cut off the corner nodes (vertices) of the 3 cube and it replaces each by a ring (cycle) of 3 nodes.

② Dynamic Network Connection:

Dynamic networks are implemented with switched channels, which ~~are~~ are dynamically configured to match the communication demand in user programs. In increasing order of cost and performance dynamic connection networks include:

(i) Bus systems
(ii) Multistage interconnection networks (MIN)
(iii) Crossbar switch networks, which are often used in shared memory multiprocessors.

(1) Bus Systems / Digital Bus:

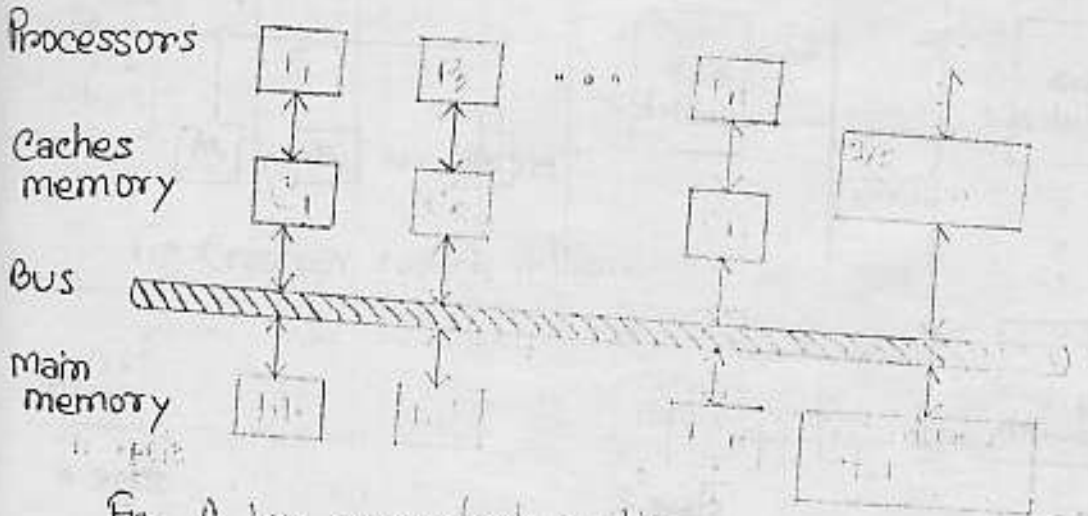
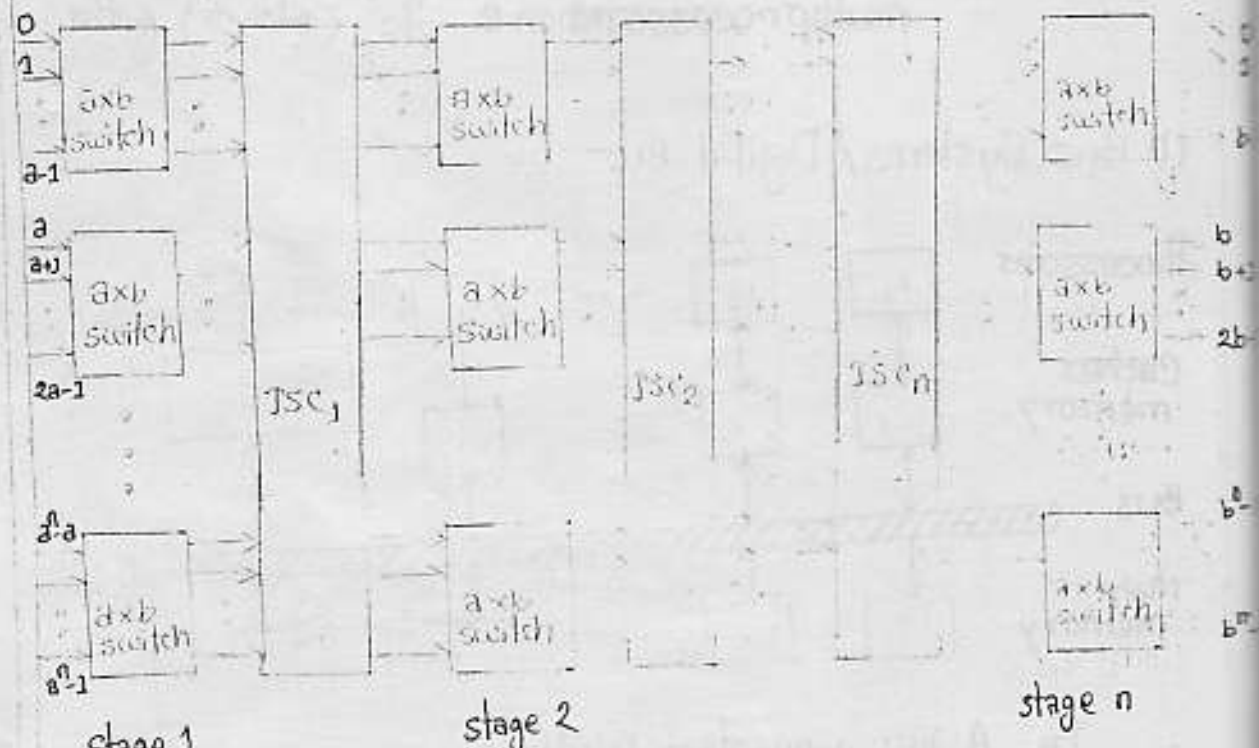


Fig. A bus-connected multiprocessor system

A bus system essentially a collection of wires & connector for data transactions among processors, memory modules, & peripheral devices attached to the bus. The bus is used for only one transaction at a time bet^n a source & a destination. A bus system has a lower cost & provides a limited bandwidth. Many industrial and IEEE bus standards are available. Fig a bus-connected multiprocessor system. The system bus provides a common communication path bet^n the processors or I/O subsystem & the memory modules or secondary storage devices (disks, tape units, etc).

(ii) Multistage Interconnection Network (MIN):



Multistages Interconnection Networks have been used in both MIMD & SIMD computers. A generalized multistage network is illustrated in fig. A number of a x b switches are used in each stage. Fixed interstage connection are used between the switches in adjacent stages. The switches can be dynamically set to establish the desired connections between the inputs & outputs.

(iii) Crossbar Switch Networks:

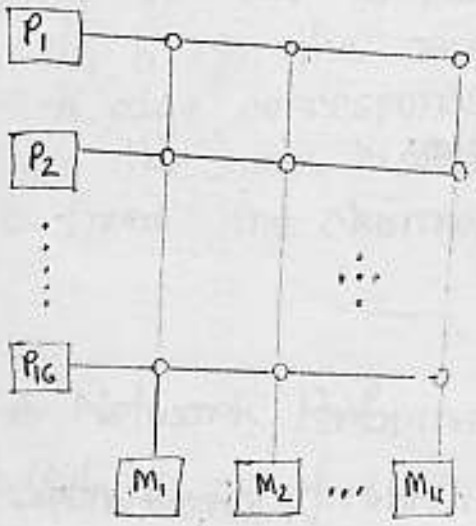


fig. Crossbar switch Network

Network Parameters:

① Node Degree
② Network Diameter
③ Bisection Width

Node Degree:

Number of edges incident on a nodes is called the node degree d

On unidirectional channels, the number of channels into a node is the in degree and that out of the node is the out degree. Then the node degree is the sum of two.

Network Diameter:

The diameter D of a network is the maximum shortest path between any two nodes. The path length is measured by the number of links traversed. The network diameter indicate the maximum number of distinct hops between any two nodes, thus providing a figure of communication merit for the network there fore the networks diameter should be as small as possible from a communication point of view.

Bisection Width:

When a given network is cut into two equal halves the minimum number of edges(channels) along the cut is called the channed bisection width b. In the case of communication network, each edge corresponds to a channel with w bit wires. Then the wire bisection width is $B = bw$. when B is fixed, the channel width $w = B/b$.

Network Performance:

- Functionality
  - (data routing, interrupt handling)
- Network Lateney
  - (Worst-case time delay for message passing)

- Bandwidth
  - (Maximum data transfer rate)
- Hardware Complexity
  - (for wires, switches, connectors, arbitration and interface logic)
- Scalability
  - (ability of a network to be modularly expandable with a scalable performance)

Difference

Static Connection Network	Dynamic Con. Net
1. Static networks are formed of point-to-point direct connection which will not change during program execution.	1. Dynamic networks are implemented with switched channels, which are dynamically configured to match the communication demand is user program
(2) It is used for fixed connection.	(2) used shared memory multiprocessors.
(3) Categories as- - one dimensional - Two » - Three »	(3) Categories as- - digital bus - single stage - multi »
(4) More suitable for building computer.	(4) multi purpose or general purpose application.

Design Space of Processor families:

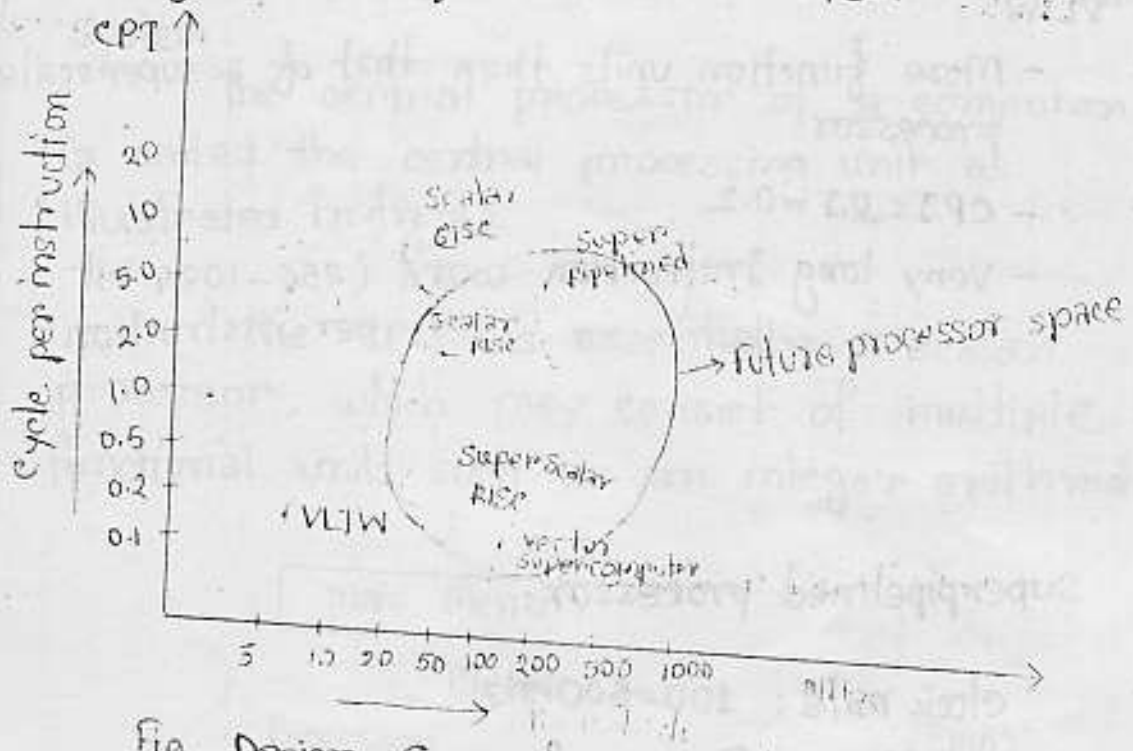


Fig. Design Space of Modern processor families

CISC:

clock rate : 33 - 50 MHz
CPI : 1 to 20

RISC:

clock rate : 20 - 120 MHz
CPI : 1 to 2
EX: Intel i860, SPARC

Superscalar RISC:

Clock Rate : 20 - 120 MHz
CPI : 02 - 0.5

VLIW:

- More function units than that of a superscalar processor.
- CPI : 0.1 – 0.2
- Very long Instruction word (256 – 1024 bit per instruction)

Superpipelined processor:

Clock rate : 100 – 500 MHz
CPI : 1 – 7

Vector Supercomputer:

Clockrate : 100 – 1000 MHz
CPI : 0.1 – 0.2

→ use multiple functional units.

Architectural Models of a basic scalar computer System:

The central processor of a computer is called the central processing unit as illustrated in fig (a).

The CPU is essentially a scalar processor, which may consist of multiple functional units such as an integer arithmetic

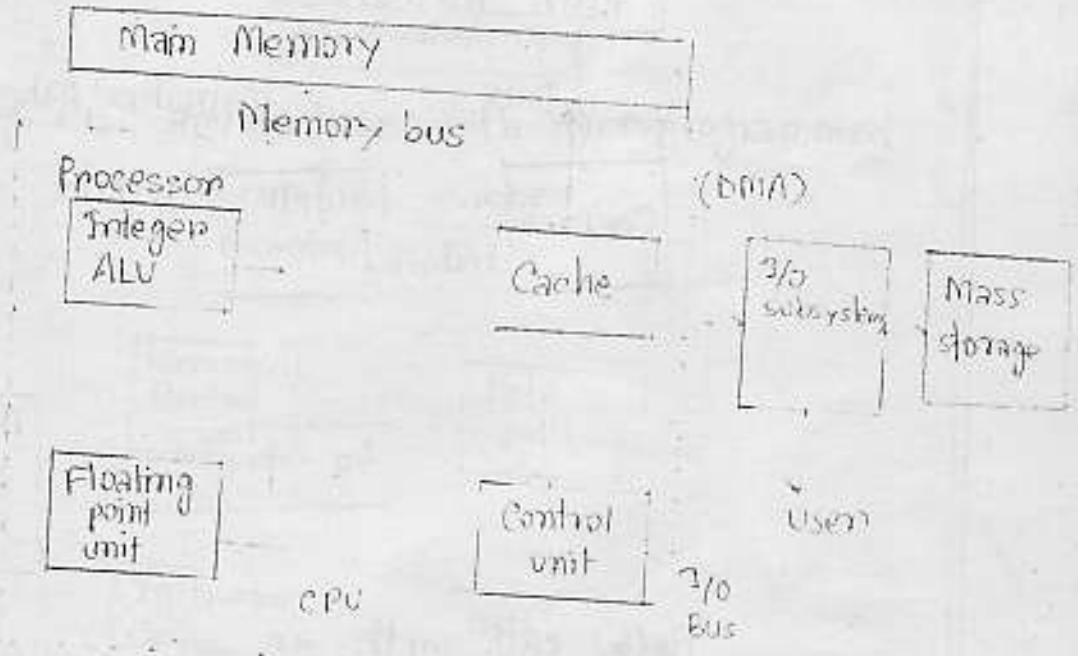
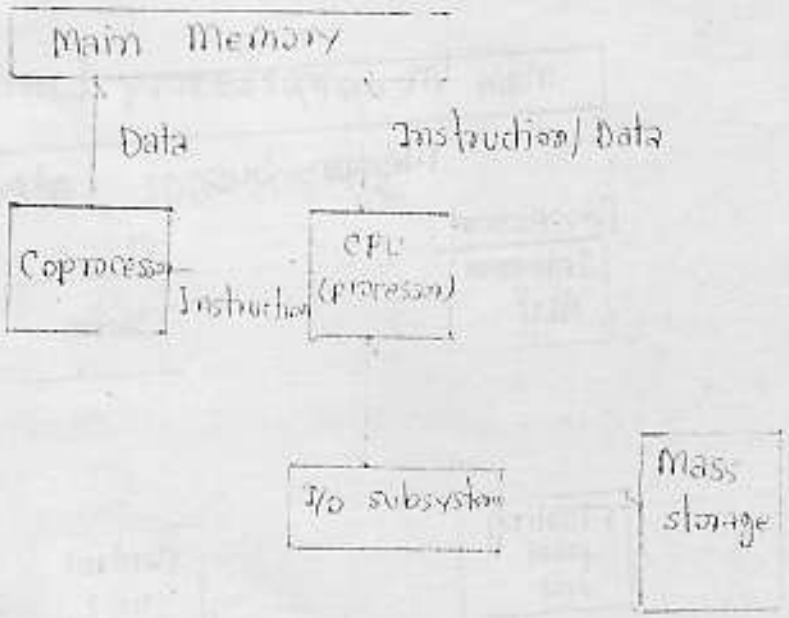


Fig (a) - CPU with built in floating unit

and logic unit, a floating-point accelerator etc. The floating point can be built on a coprocessor, which is attached to the CPU. Fig (b) shows this depict—

The coprocessor executes instructions dispatched the cpu. A coprocessor may be a floating-point accelerator executing scalar data, vector processor executing vector operands, a digital signal processor, or a lisp processor executing AI program. Coprocessor cannot handle I/O operations.



Fig(b) cpu with an attached processor

Coprocessor cannot be used alone. The processor and coprocessor operate with a host-back-end relationship between them is a necessity.

Architectural Distinction betⁿ RISC & CISC processor:

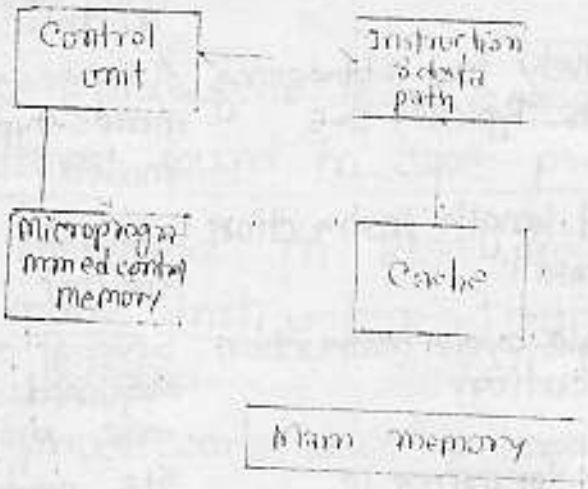


fig. CISC architecture with microprogrammed control & unified cache

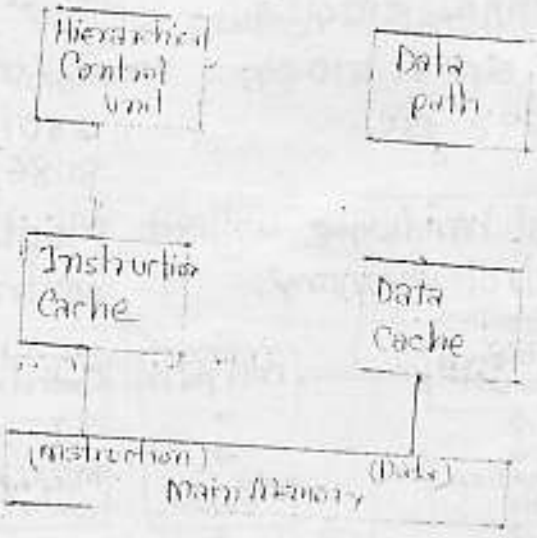
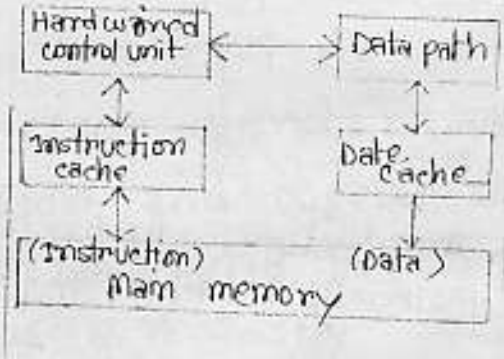
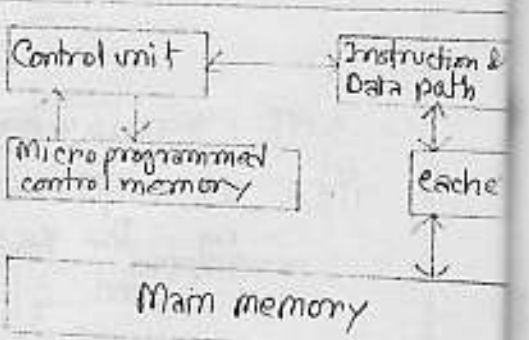


fig. The RISC architecture with hardwired control and split instruction cache and data cache.

21-12-11

Difference bet^n RISC & CISC.

Architectural characteristics	RISC processor	CISC processor
1. Instructions set	Relatively few instruction	A large number of instructions.
2. Addressing modes	Relatively few addressing modes. Typically 3-5.	A large variety of address modes-typically 5-20 differe modes.
3. Instructions formats	fixed-length instruction format.	Variable-length instructio format.
4. Instructions executions	Single cycle instruction execution.	Several cycles may be required to execute one instruction.
5. Performance	Its performance is generally high.	Its performance is generally low.
6. Clock rate & CPI	50-150 MHz in 1993 with one cycle for almost all instructions and an average CPI < 1.5.	33-50 MHz in 1992 with a CPI between 2 & 15.
7. Example	DEC alpha: 21064, 21064a etc.	Motorola-6800, 68000, 68010 etc. Intel-8085, 8086, 8088.
8. CPU control	Most hardwire without control memory.	Most micro coded using controls memory (ROM)
9. Block Diagram		

Disadvantages of RISC:

(i) Converting from a CISC program to an equivalent RISC program increased the code length by 40%.

(ii) A RISC processor lacks some sophisticated instructions found in CISC processors.

(iii) The increase in RISC program length implies more instruction traffic and greater memory demand.

(iv) RISC processors use a larger register file where the register decoding system will be more complicated.

(v) Another short coming of RISC lies in its hardwired control, which is less flexible and more error-prone.

Q. 1. Digital equipment VAX 8600 processor architecture (CISC processor) (fig-4.5)
P-166, 167

* Q. 2 Intel i860 processor architecture (RISC processor) fig (4.9)
P=174-178

Q. 3 The IBM RS/6000 architecture (RISC processor) fig (4.13)
P=181-183

Q 4 The VLIW architecture (fig -4.14)

Q.5 Memory Hierarchy. P=188-189

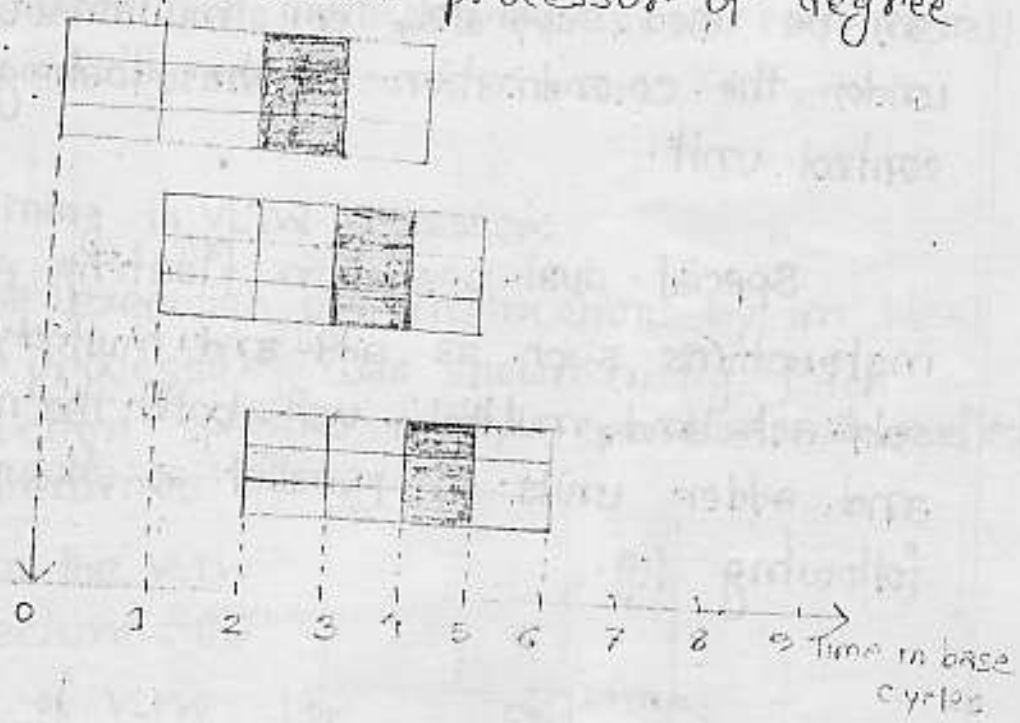
Q.6 Virtual Memory. P=196-201

Pipelining in Super Scalar processor:

Super scalar processor is special subclass of RISC processor. In a super scalar processor, multiple instruction pipelines are used. This implies that multiple instructions are issued per cycle and multiple results are generated per cycle.

Super scalar processors are designed to exploit more instruction-level parallelism in user programs. Only independent can

be executed in parallel without causing a wait state. The amount of instruction-level parallelism varies widely depending on the type of code being executed. Fig. shows a super scalar processor of degree m=3.



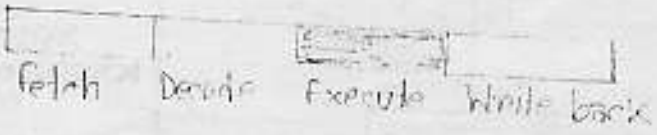


Fig. A super scalar processor.

Example:

Intel i960 CA, IBM RS/6000 & DEC 21064

04.01.12

[2] Dual operations in i860 microprocessor:

In the intel i860 RISC architecture, there are two floating point units, namely, the "multiplier unit" and the "adder unit", which can be used separately or simultaneously under the co-ordination of the floating point control unit.

Special dual-operation floating-point instructions such as add-and-multiply and subtract-and-multiply use both the multiplier and adder units in parallel as shown in following fig.

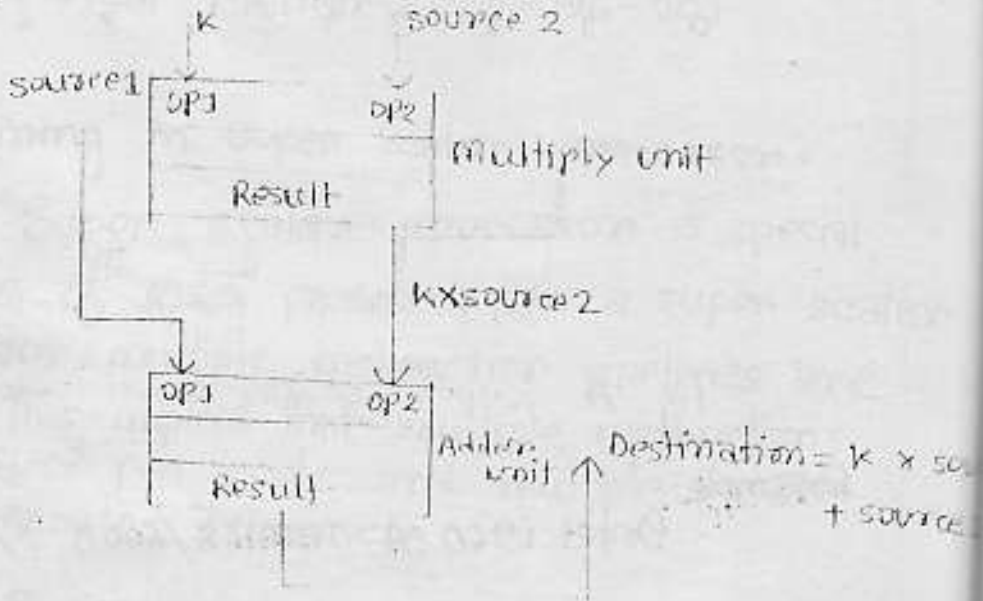


fig. Dual operations in i860 processor

04.01.12

3 Applications of IBM RS/6000 architecture:

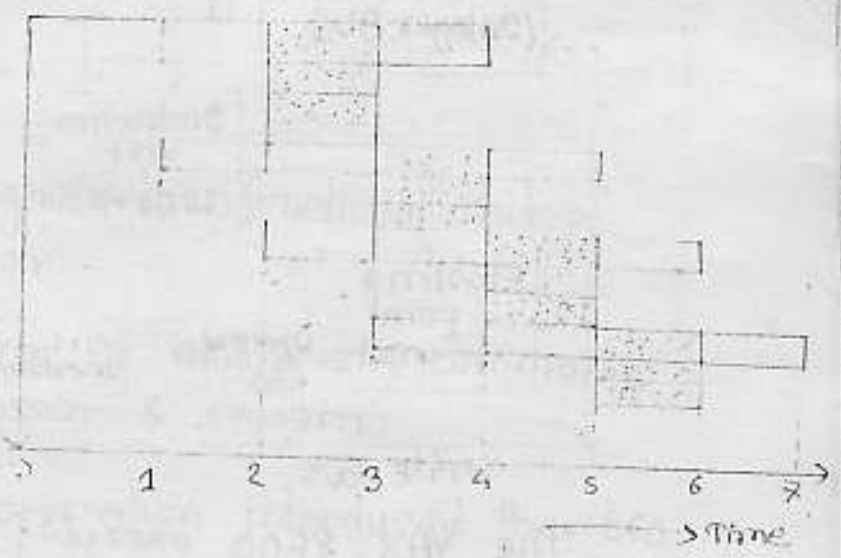
① Perform numerically intensive scientific applications.

② Perform Engineering applications.

③ Used for multiuser comltiuser commercial environments.

4 Pipelining in VLIW processor:

The execution of instructions by an ideal VLIW processor was shown in fig. Each instruction specifies multiple operations. The effective CPI becomes 0.33 in the VLIW architecture. The degree of VLIW processor is 3 as 3 instructions can be issued at a time, VLIW machines behaves much like superscalar machine.



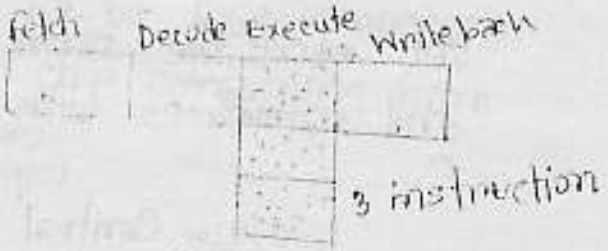
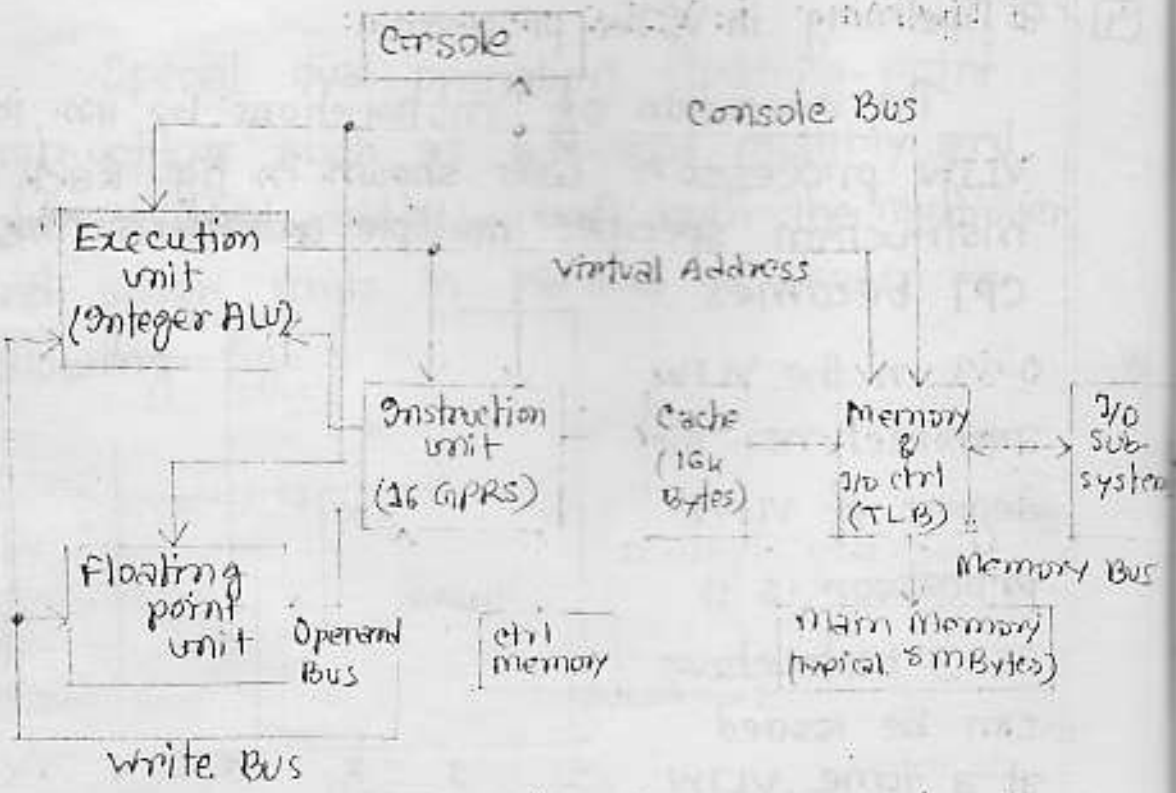


fig. VLIW execution with degree = 3
(Pipelined Operation)

Q. Describe in brief Digital equipment VAX 8600 processor architecture.

The VAX 8600 was introduction by Digital Equipment Corporation in 1985. This machine implements a typical CISC architecture with micro programmed control. The instruction set contains about 30 instructions with 20 different addressing modes. As shown in fig-



The VAX, 8600 executes the same instruction set, runs the same VMS operating system, and interfaces with the same I/O buses.

Captions:

CPU = Central Processing Unit
TLB = Translation Lookaside Buffer
GPR = General Perpose Register

The CPU in the VAX 8600 consists of two functional units for concurrent execution of integer and floating-point instruction. The unified cache is used for holding both instruction and data. There are 16 GPRs in the institution unit. Institution pipeline has been built with six stages in the VAX 8600, as in most CISC machines.

A translation lookaside buffer (TLB) is used the memory control unit for fast generation of a physical address from a virtual address. Both integer and floating point units are pipelined.

Q Explain with diagram the functional unit of Intel i860 processor.

or, Draw the diagram of ~~major~~ any commonly used RISC processor & explain.

⇒ In 1989 Intel Corporation introduced the i860 microprocessor. It is a 64-bit RISC processor fabricated on a single chip containing more then 1 million transistors.

A schemantic block diagram of major components in the i860 is shown in fig-

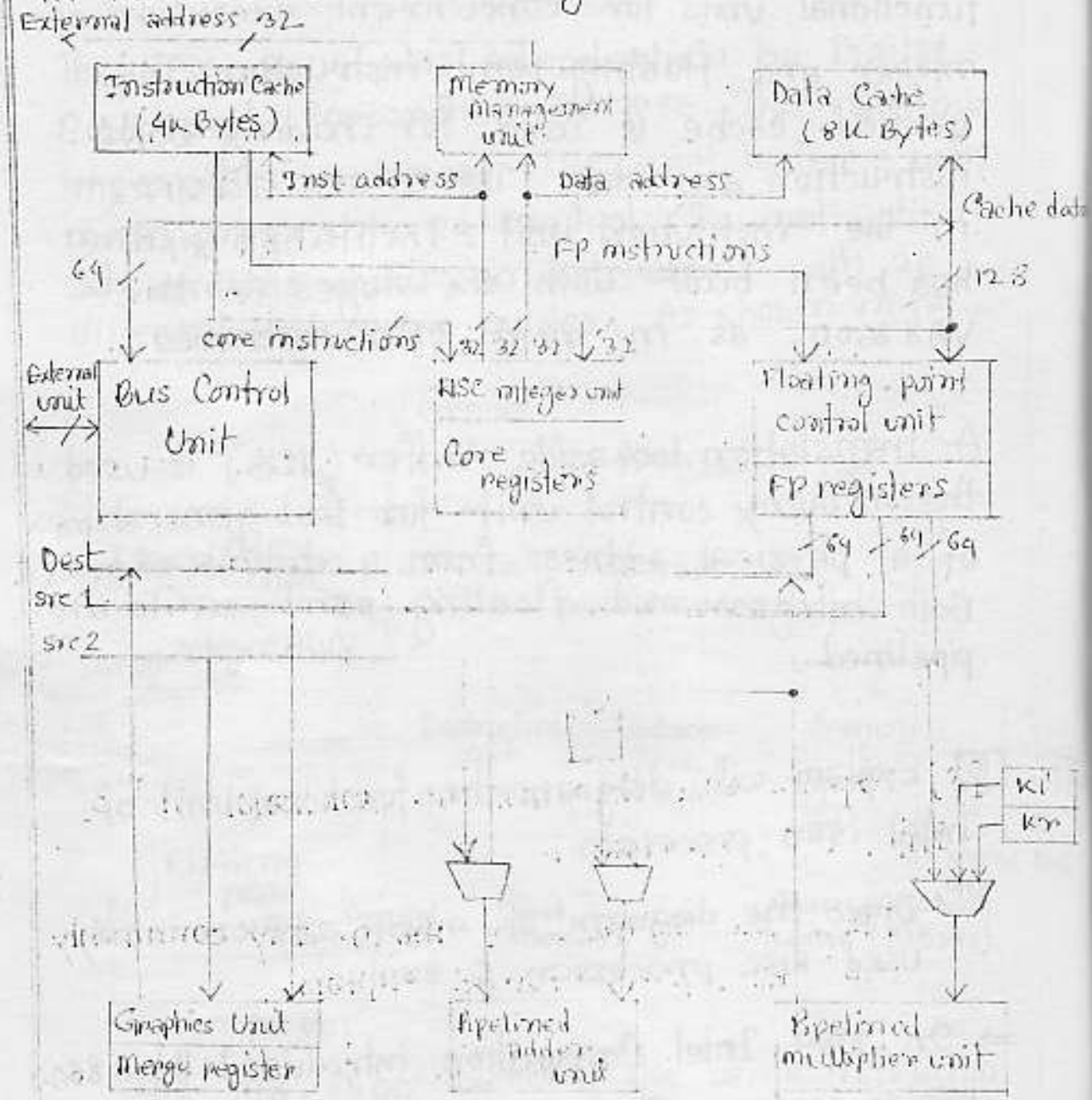


fig. functional units & data paths of the Intel i860 RISC microprocessor

The i860 has the following parts –

- Bus unit / bus control unit
- Cache unit (instruction and data cache)
- Memory management unit (MMU)
- Integer unit
- Floating point unit
- Graphics unit
- Pipelined adder and multiplier unit.

⇒

Bus unit:

There are nine functional units interconnected by multiple data paths with widths ranging from 32 to 128 bits.

All external or internal address buses are 32-bit wide, and external data path or internal data bus is 64-bit wide.

Cache unit:

The instruction cache has 4 Kbytes organized as a two-way set-associative memory with 32 byte per cache block. It transfers 64 bits per clock cycle, equivalent to 320 Mbytes/s at 40 MHz.

MMU: The MMU implements protected 4KB paged virtual memory of 2^{32} bytes via TLB.

Integer unit: The RISC integer unit consists of 32-bit RISC core & GPRs is 32. It executes load, store, bit and control instructions and fetches instructions for the floating point control units as well.

Floating-point unit: There are two floating-point units. – The multiplier unit and
– The adder unit.

Graphics unit: The graphics unit executes integer operations corresponding to 8-, 16-, or 32-bit pixel data types.

Pipelined adder & multiplier unit: The i860 execution 82 instruction, including 42 RISC integer, 24 floating-point, 10 graphics and 6 assembler pseudo operations.

Claimed performance & successor to watch: 40 MIPS & 60 Mflops for 40 MHZ, i860/XP announced in 1992 with 2.5 M transistors.

(3) Q Describe in brief the IBM RS/6000 architecture.
or,
Draw the diagram of any commonly used RISC super scalar processor & explain its principle operations.

⇒ It is a Super scalar processor as illustrate in fig. There are three functional units called the branch processor, fixed-point unit and floating-point units, which can operate parallel.

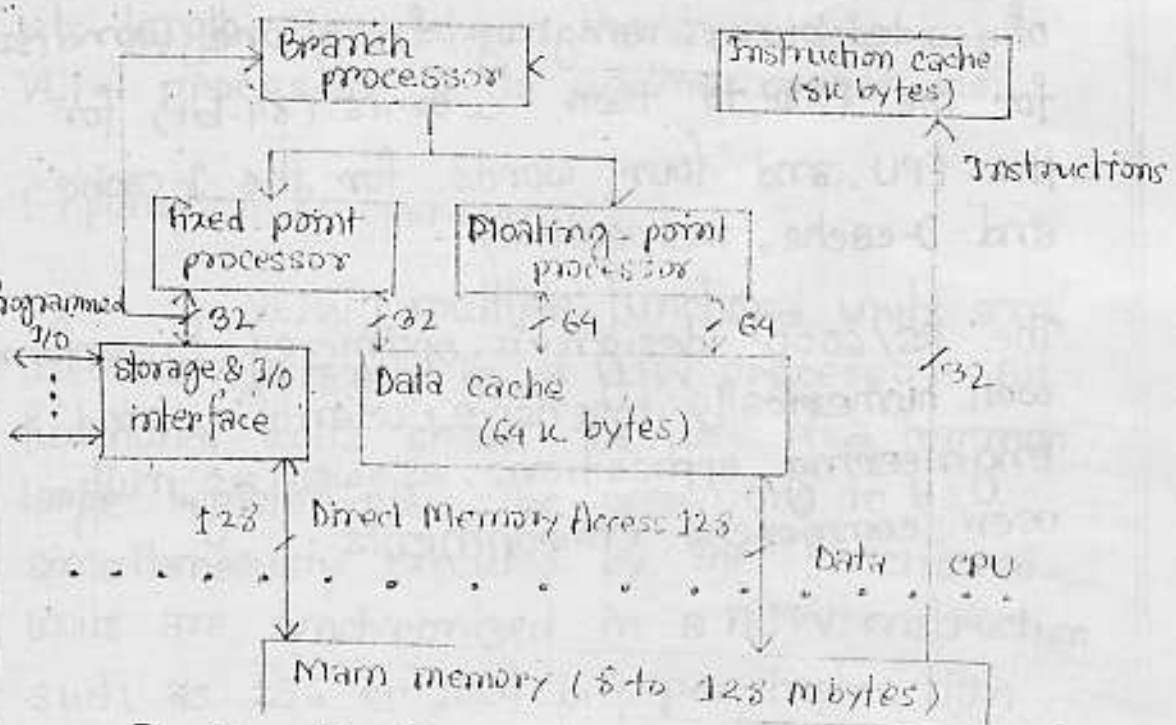


fig. The POWER architecture of the IBM RISC system/6000 super scalar processor

The branch processor can arrange the execution of up to 4 instructions per cycle

These include one branch instruction in the branch processor, one fixed-point instruction in the FXU, one condition-register instruction in the branch processor and one floating-point multiply-add instruction in the FPU, which can be counted as two floating-point operations.

The RS/6000 is hardwired rather then microcoded. The system uses a number of wide buses ranging from one word (32-bits) for the FXU to two words (64-bit) for the FPU, and four words for the I-cache and D-cache, respectively.

The RS/6000 design is optimized to perform well numerically intensive scientific and engineering applications, as well as multi-user commercial environments.

[9] Describe the architecture and operation of a VLIW machine.

⇒ Very Long Instruction Word (VLIW) architecture:

The VLIW architecture is generalized from two well-established concepts.

- Horizontal microcoding and
- Super scalar processing.

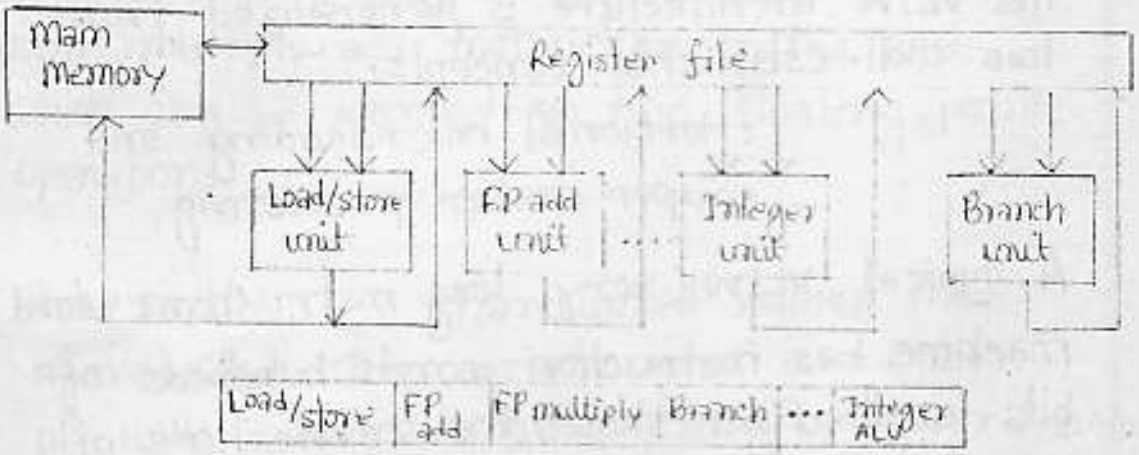
A typical VLIW (very long instruction word) machine has instruction words hundreds of bits length. Fig-1 shows the architecture of VLIW processor & its pipeline operations.

Explanation & operations:

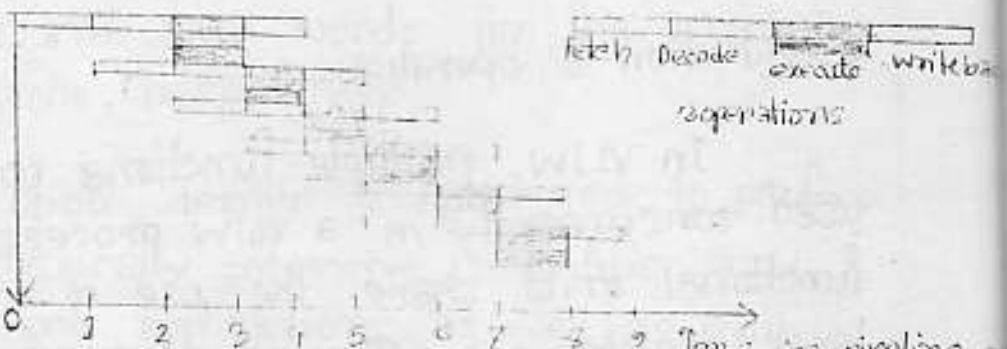
In VLIW, multiple functional units are used concurrently in a VLIW processor. All functional units share the use of a common large register file. The operations to be simultaneously executed by the functional units are synchronized in a VLIW instruction such as 256 or 1024 bits per instruction word. Each instruction specifies multiple operations. It serves—

- The decoding of VLIW instructions is very easy.
- The code density of VLIW is normal.

- VLIW machine exploiting different amounts of parallelism would require different instruction sets.



fig(a) A typical VLIW processor & instruction format



fig(b) VLIW execution with degree m = 3.

fig 1 The architecture of VLIW processor & its pipline operation.

Advantages:

- VLIW processor can eliminate the hardware & software needed to detect parallelism.
- The VLIW is simplicity in hardware structure & instruction set.
- The VLIW processor can potentially perform well in scientific applications.

5. Q Give a brief description of hierarchical memory technology / memory hierarchy technology with necessary diagram??

Hierarchical Memory Technology:

Storage devices such as registers, caches, main memory, disk device and tape units are often organized as a hierarchy as depicted in fig (*). The memory technology and storage organization at each level are characterized by five parameters—

- The access time (t_i)
- Memory size (s_i)
- Cost per byte (c_i)
- Transfer bandwidth (b_i) and
- Unit of transfer (x_i).

The access time (t_i), refers to the round-trip time from the CPU to the ith-level memory. The memory size is (s_i), is the number of bytes or words in level i. The cost of the ith-level memory is estimated by the product $c_i s_i$.

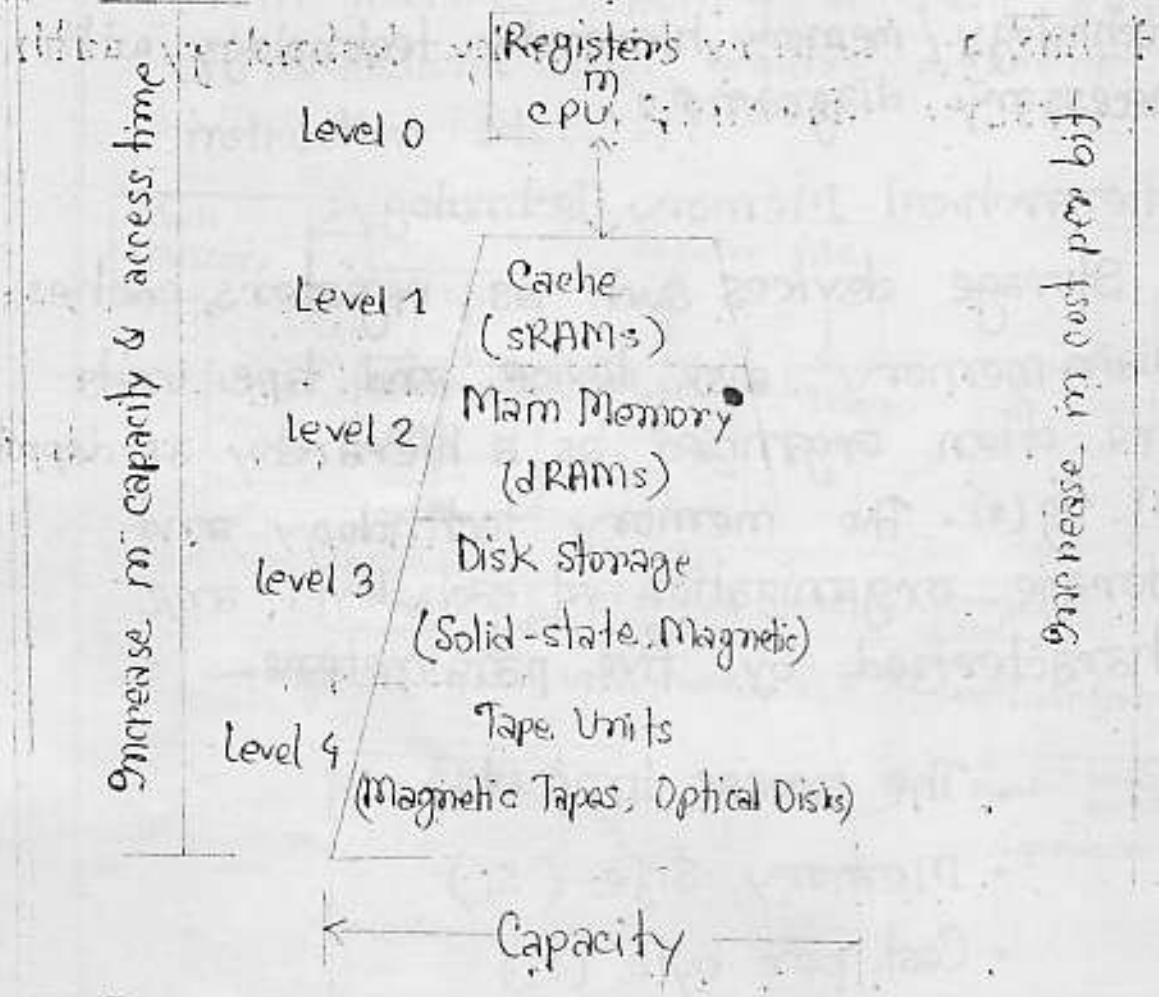


Fig: A 5-level memory hierarchy

The bandwidth (b_i) refers to the rate at which information is transfer between adjacent levels. The unit of transfer (x_i) refers to the grain size for data transfer between levels i and $i+1$.

Memory devices at a lower level are faster access, ~~numbe~~ smaller in size & more expensive per byte having a higher bandwidth & using a smaller unit of transfer as compared with those at a higher level. In other words, we have.

$t_{i-1} < t_i$, $s_{i-1} < s_i$, $c_{i-1} < c_i$, $b_{i-1} < b_i$ &
$x_{i-1} < x_i$ for $i = 1, 2, 3$ & 4 in the hierarchy where $i = 0$.

Distributed

CHAPTER-1

(2)

Lec-1
12-10-11

Distributed Computing System:

Distributed computing system is bassically a collection of processors interconnected by a communication network in which each processor has it's own local memory and other peripherals and the communication between any two processors of the system takes place by message passing over the communication network.

→ for a particular processor, its own resources are local, whereas the other processors and their resources are remote.

→ Together a processor and its resources are usually referred to as a node or site or machine of the distributed computing system.

Example:

- Consider a network of workstations in a university or company department.
- Consider a factory full of robots, each containing a powerful computer for handling vision, planning, communication and other tasks.

2. Tightly & Loosely Coupled Multiprocessor system:

Computer architecture consisting of interconnected multiple processors are basically two types.

(i) Tightly coupled and
(ii) Loosely coupled

(i) Tightly Coupled System: (Parallel processor)

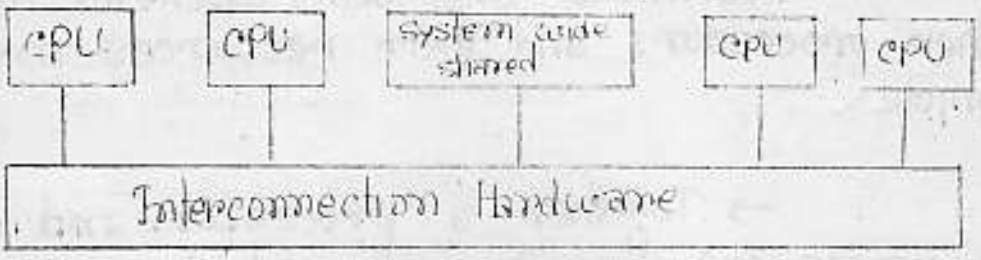


Fig. A tightly coupled multiprocessor system

On this system, there is a single system wide primary memory that is shared by all the processors.

Therefore, any communication between the processors usually takes place through the shared memory.

Compare between—

Tightly Coupled	Loosely Coupled
1. Tightly coupled systems are reffered to as a parallel processing system.	1. Loosely coupled systems are reffered to as a distributed computing system or simply distributed system.
2. It's working on a single problem.	2. Working on a many unrelated problem.
3. The delay time short and the data rate is high.	3. The inter machine message delay is large and the data rate is low.
4. The number of bits per second that can be transferred is large.	4. The number of bits per second that can be transferred is short.
5. Example - Two CPU chips on the same printed circuit board and connected by wires etched onto the board are likely to be tightly coupled.	5. Example - Two computers connected by a 2400 bit/sec modem over the telephone system are certain to be loosely coupled.

3. Advantage of distributed system over centralized system.

4 ⊡ Advantages of Distributed System over Independent PC:

1. Data Sharing:

Allow many users access to a common database.

2. Device Sharing:

Allow many users to share expensive peripherals like color printer.

3. Communication:

Make human-to-human communication easier, for example by electronic mail.

4. Flexibility:

Spread the workload over the available machines in the most cost-effective way.

5 ⊡ Disadvantage of Distributed System:

1. Software - Little software exists at present for distributed system.

2. Networking - It can lose message (which requires to recovering mechanism) and consequently it can become overload when the network saturats it must be required.

3. Security - It is the main concern for distributed system.

6. Distributed Computing System Model:

Various models are used for building distributed computing systems. These models can be broadly classified into five categories.

(i) Mini Computer Model.
(ii) Workstation Model.
(iii) Workstation server model.
(iv) Processor pool model.
(v) Hybrid Model.

① Mini Computer Model:

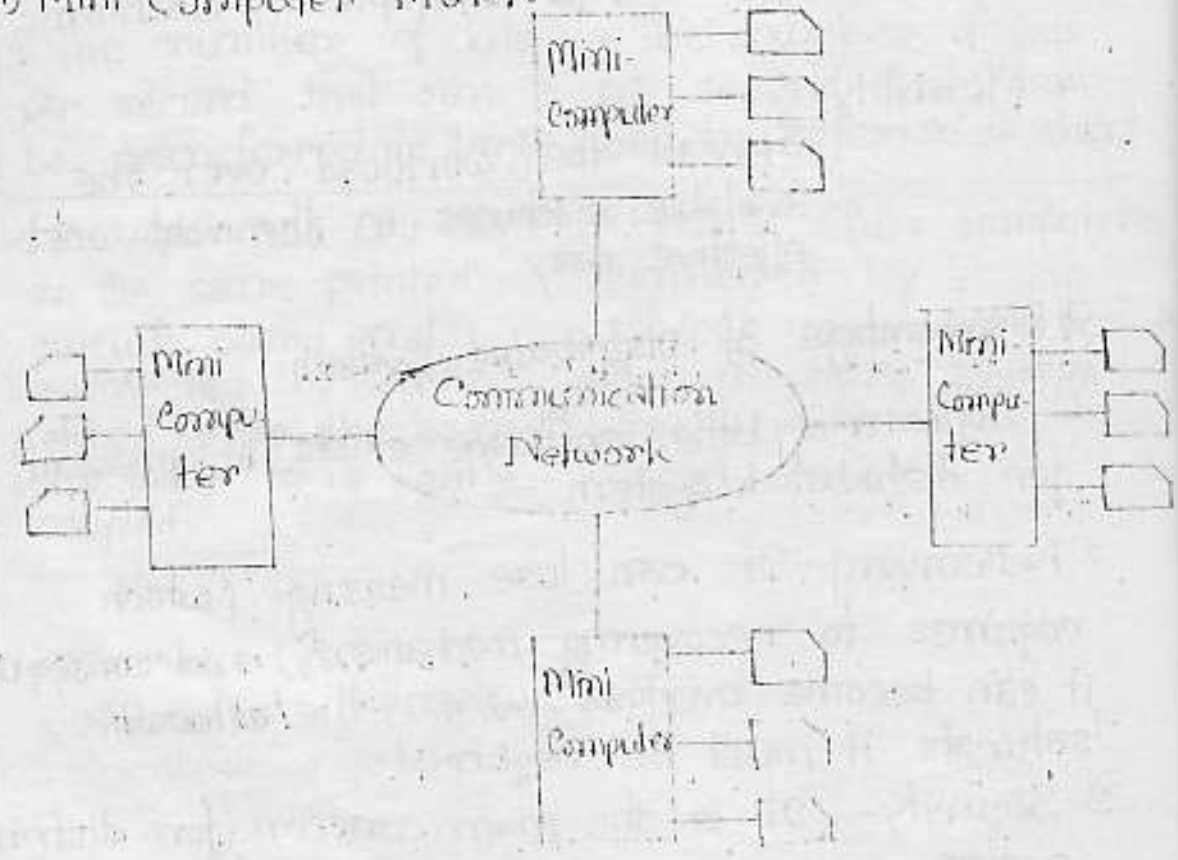


Fig Mini Computer Model

→ The minicomputer model is a simple extension of the centralized time-sharing system. A distributed computing system based on this model consists of a few minicomputers interconnected by a communication network.

→ Each minicomputer usually has multiple users simultaneously logged on it.

→ The network allows a user to access remote resources that are available on some machine other than the one to which the user is currently logged.

→ The minicomputer model may be used when resources sharing with remote users is desired.

(ii) Workstation Model:

→ A distributed computing system based on workstation model consists of several workstations interconnected by a communication network.

→ A companies office may have several workstations scattered throughout a building, each workstations equipped with its own disk and serving single-user computer.

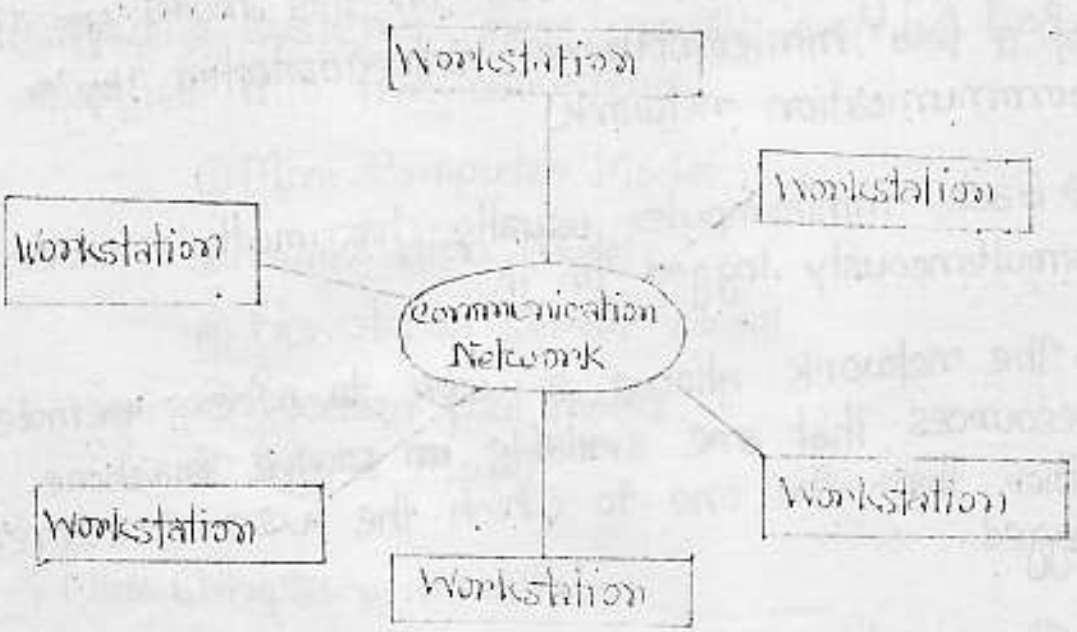


fig Workstation Model

→ In such as environment at any time, a significant proportion of the workstations are idle, resulting in the waste of large amount of CPU time.

→ The idea of the workstation model is to interconnect all these workstation by a high speed LAN so that idle workstations may be used to process.

(iii) Workstation-Server Model:

→ A distributed computing system based on the workstation model consists of a few minicomputers and several workstations interconnected by a communication network. These—

• Diskful Workstation – A workstation with its own local disk is usually called a diskful workstation.

• Diskless Workstation – A workstation without a local disk is called a diskless workstation.

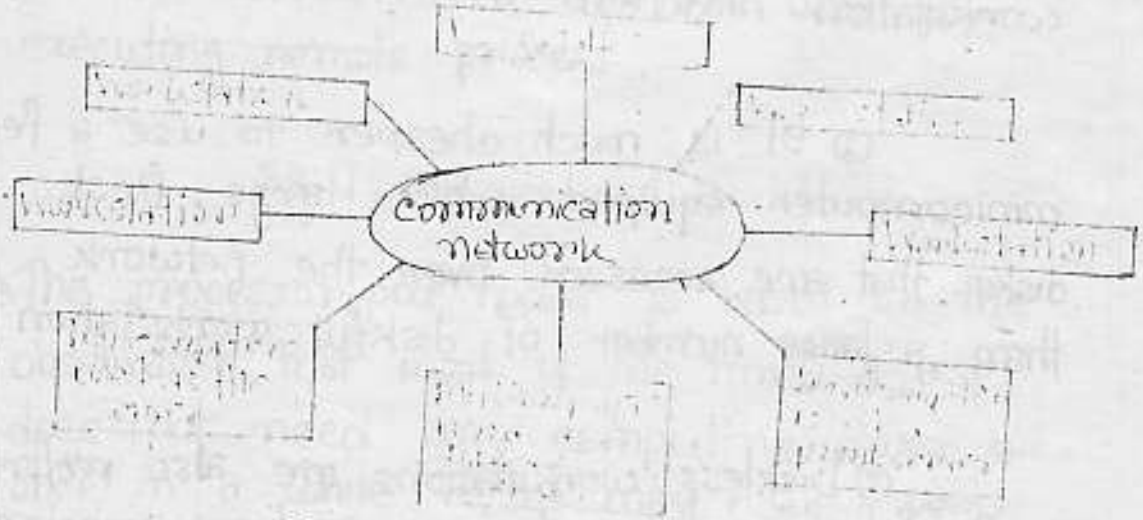


Fig. Workstation server model

→ One or more of the minicomputers are used for implementing the file systems, other minicomputer may be used for providing other types of services such as database services and print service.

→ For better overall system performance, the local disk of a diskful workstation is normally used for such purpose of storage of temporary files.

storage of unshared files
storage of shared files
paging activity and
Caching of remote access data.

Advantages of workstation-server model over workstation model:

① It is much cheaper to use a few minicomputer equiped which large, first disks that are accessed over the network than a large number of diskful workstation.

② Diskless workstations are also performed to diskful workstation from a system maintenance point of view.

③ In the workstation-server model, since all files are managed by the file servers, users have the flexibility to access the files irrespective of which workstation the user is currently logged on.

But in workstation model, each workstation has it's local file system, so different mechanisms are required to access local and remote files.

④ Unlike the workstation model, this model does not need a process migration facility, which is difficult to implement.

⑤ A user has guaranted response time because workstations are not used for executing remote process.

(iv) Processor-pool Model:

→ The processor-pod model is based on the observation that most of the time, a user does not need any computing power but once in a while he/she may need a very large amount of computing power for a short time.

→ Run server is a special server which manages and allocates the processors in the pool to different users on a demand basis.

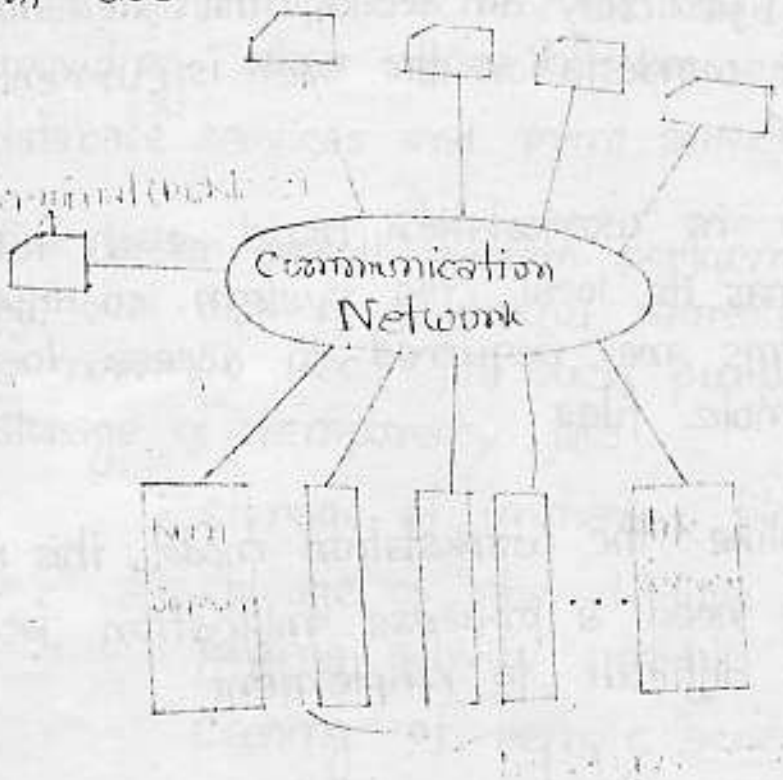


Fig. Processor Pool Model

→ The pool of processors consists of a large number of minicomputers and microcomputers attached to the network.

Applications of pool-processor model

Massive Computation

⑤ Hybrid Model:

Workstation-server model and processor-pool model both are together called Hybrid model.

7. [Q.] Why are distributed computing system (DCS) gaining popularity?

⇒ Distributed computing systems are much more complex and difficult to build than traditional centralized approach.

Despite the increased complexity and difficult building distributed computing systems, the installation are used of distributed computing systems are rapidly increasing.

This is mainly because the advantages of distributed computing system outweight them disadvantages.

Popularity of DCS's are -

① Inherently Distributed Application.
② Information Sharing Distributed users.
③ Resource Sharing.
④ Better Price-performance ratio.

(v) Shorter response time and higher throughput.

(vi) Higher reliability.

(vii) Extensibility and Incremental Growth.

(viii) Better Flexibility in meeting user needs.

Advantages of DS Over Centralized System:

(1) The leading reason is that distributed systems have a much better price/performance ratio than a single large centralized system would have.

(2) May have more total computing power than a main frame / Centralized system.

(3) A next reason for building a distributed system is that some applications involve spatially separated machine.

(4) If one machine crashes, the system as a whole can still service, i.e; a distributed system provides higher reliability.

(5) finally, incremental growth is also potentially a big plus.

Issues in designing a distributed OS: (DOS)

→ Despite the complexities & difficulties of building distributed computing system, the users should be able to view a distributed system as a virtual centralized system.

That is, flexible, efficient, reliable, secure and easy to use. To meet this challenges the designers of a DOS must deal with several design issues. These are—

1. Transparency
2. Reliability
3. Flexibility
4. Performance
5. Scalability
6. Security

Describing:

Transparency: A distributed operating system (DOS) must be designed in such a way that a collection of distric machines connected by a communication subsystem

appears to its users as a virtual uniprocessor. The eight forms of transparency identified by the ISO are –

• Access Transparency – means that users should not or be able to recognize whether a resource is remote or local.

• Location Transparency – means of the resource should be independent of the physical connectivity or topology of the system.

• Replication Transparency – Almost all DDS have the provision to create replicas of files and other resources on different nodes of the distributed system.

• Failure Transparency – deals with masking from the users partial in the system, such as communication link failure, a machine failure or a storage device crash. Failure transparency will continue to function perhaps is a degraded form.

• Migration Transparency – The aim of Migration Transparency is to create that the movement of the object is manipulated automatically by the system.

• Concurrency Transparency (CT):

→ In a distributed system, multiple users who are spatially seperated use the system concurrently.

→ Concurrency transparency means that each user has a feeling that he/she is the sole user of the system.

• Performance Transparency (PT):

The aim of PT is to allow the system to be automatically reconfigured to improve performance as loads very dynamically in the system.

• Scaling Transparency (ST):

The aim of scaling transparency is to allow the system to expand in scale without distrupting the activities of the users.

② Reliability :

→ In general DS are accepted to be more reliable than centralized systems due to the existence of multiple instance of resources.

→ The DOS which manages these resources, must be designed properly to increase the systems reliability by taking full advantage.

→ A fault is a mechanical or algorithmic defect that may generate an error. A fault in a system causes system failure.

→ For higher reliability, the fault-handling mechanisms of a DOS must be designed properly to avoid faults, to tolerate faults, and to detect and recover from fault

Fault Tolerance: F.T. is the ability of a system to continue functioning in the event of partial system failure. The performance of the system might be degraded due to the partial failure but otherwise the system function properly.

③ Flexibility—

Another important issue in the design of DOS is flexibility. The design of a DOS should be flexible due to the following reasons.

i) Ease of Modification:

It will be more easy to incorporate changes if the system is highly flexible.

ii) Ease of Enhancement:

In the every system, new functionalities have to be added from time to time, to make it more powerful and easy to add new services to the system.

④ Performance—

If a distributed system is to be used, its performance must be at least as good as a centralized system. To achive this goals, it is important that various components of the OS of a DOS be designed properly.

Some design principles considered useful for better performance as follows.

① Bach if possible
⑪ Cache whenever possible.

⑤. Scalability -

→ Refers to the capability of a system to adapt to increase service load. Therefore, a DOS should be designed to easily cope with the growth of nodes and users in the system.

⑥ Security -

→ In order that the users can trust the system and rely on it, the various resources of a computer system must be protected against destruction & unauthorized access.

→ DS has the following additional requirements

① Sender should know that the message has sent by the authorized sender.

(ii) Sender should know that the message has received by the intended receiver.

(iii) Both the sender & receiver should know that the contents of message has not been changed while it is transfer.

CH-6 (P.K Sinha)
Q 1 What are the synchronization mechanism

⊞ Synchronization:

A client process and a file server process must co-operate when performing file access operations. Both co-operative and competitive sharing require adherence to certain rules of behavior that guarantee that correct interaction occurs. The rules for enforcing correct interaction are implemented in the form of synchronization mechanisms. Synchronization mechanisms that are suitable for distributed systems. The following issues are—

① Clock Synchronization

② Event Ordering

③ Mutual Exclusion

④ Deadlock

⑤ Election Algorithm.

(2) ①Clock Synchronization:

How computer clocks are implemented?

A computer clock usually consists of three components:

① A quartz crystal that oscillates at well defined frequency.

(ii) A Counter Register

(iii) A Constant Register

The constant register is used to store a constant value that is decided based on the frequency of oscillation of the quartz crystal.

The counter register is used to keep track of the oscillations of the quartz crystal. That is, the value in the counter register is decremented by 1 for each oscillation of the quartz crystal.

When the value of the counter register becomes zero an interrupt is generated and it's value is reinitialized to the value in the constant register. Each interrupt is called clock tick

To make the computer clock function as an ordinary clock used by us in our day-to-day life, the following things are done.

① The value in the constant register is choosen so that 60 clock ticks occur in a second.

(ii) The computers clock is synchronized with real time (external clock).

Q.3 Desc the type of clock synchronization in DS

Types of Clock Synchronization:

A distributed system requires two types of clock synchronization.

(i) Synchronization of the computer with real time (external) clocks:

This type of synchronization is mainly required for real-time application. That is, external clock synchronization allows the system to exchange information about the timing of events with other system and users. An external time source that is often used as a reference for synchronization computer clocks with real time is the co-ordinated univesal time (UTC).

The UTC is an international standard. Many standard bodies dissemmate UTC signals by radio, telephone & satellite. Commercial devices (time providers) are available to receive and interpret these signals.

Computers equipped with time provider devices can synchronize their clocks with these timing signals.

(ii) Mutual (Internal) synchronization of the clocks of different nodes of the system.

This types of synchronization of the is mainly required for those applications that required a consistent view of time across all nodes of a distributed system as well as for the measurement of the duration of distributed activities that terminate on a node different from the one on which they start.

[9] Externally synchronized clocks are also internally synchronized.

Every computer needs a timer mechanism (called a computer clock) to keep track of current time and also for various accounting purposes such as calculating the time spent by a process in CPU utilization, disk I/O and so on, so that the corresponding user can be changed properly.

☐ Externally Synchronized clocks are also internal synchronized, but the converser is not true, Explain

⇒ Note that, externally synchronized clocks are also internally synchronized. However, the converser is not true because with the passes of time internally synchronized clocks may drift arbitrary far from external time.

☐ Clock synchronized Algorithm:

Clock synchronization algorithms may be broadly classified as—

① Centralized and
② Distributed Algorithm

Q.4 Desc the centralized clock syn algorithm

4. 1) Centralized Algorithm:

The goal of the algorithm is to keep the clocks of all other nodes synchronized with the clock time of the time server node. Centralized Clock synchronization algorithms are two types.

⇒ Passive Time Server Centralized Algorithm

In these method, each node periodically send

a message ("time=?") to the time server. When the time server receives the message, it quickly responds with a message ("time=T"), where, T is the current time in the clock of the time server node.

Let us assume that, when the client node sends the "time=?" message, it's clock time is T_0, and when it receives the "time=T" message, its clock time is T_1. Since T_0 and T_1 are measured using the same clock. The propagation of the message "time=T" from the time server node to the client's node is $-\left(\frac{T_1 - T_0}{2}\right)$. When the reply is received at the client's node, it's clock is readjusted to $-T + \frac{(T_1 - T_0)}{2}$. It has two such methods—

- The availability of some additional information —

It is assumed that the approximate time taken by the time server to handle the interrupt and process a "time=?" request message is known.

A better estimate of the time taken by for propagation of the message "time = T", the client's node —

$$\left(\frac{T_1 - T_0 - I}{2}\right).$$

When the reply is received at the client's node it's clock is readjusted to —

$$T + \left(\frac{T_1 - T_0 - I}{2}\right)$$

• No additional information is available:

This method was proposed by Cristian (1989). In this method, several measurements of $T_1 - T_0$ are made, and those measurements for which $T_1 - T_0$ exceeds some threshold value, are considered to be unreliable and discarded.

⇒ Active Time Server (centralized Algorithm:

In active time server approach, the time server periodically broadcasts its clock time ("time = T"). The other nodes receive the broadcast message and use the clock time in the message for correcting their own clocks. Each node has a priori knowledge of the approximate time (T_B) required for the propagation of the message "time = T" from the time server node to its own node. Therefore, when the broadcast message is received at a node, the node's clock is readjusted to the time $T + T_B$.

A major drawback —

It is not fault tolerant.

6. Centralized Clock Synchronization algorithms suffer from two major drawbacks:

① They are subject to single-point failure. If the time server node fails, the clock synchronization operation cannot be performed.

② From a scalability point of view it is generally not acceptable to get all the time requests serviced by a single time server. In a large system, such a solution puts a heavy burden on that one process.

5. ② Distributed Algorithm:

We know that externally synchronized clocks are also internally synchronized. That is, if each node clock is independently synchronized with real time, all the clocks of the system remain mutually synchronized.

Therefore, a simple method for clock synchronization may be to equip with each node of the system with a real time receiver. so that, each node's clock can be independently synchronized with real-time.

• Multiple real-time clocks (one for each node) are normally used for the purpose.

• different real-time clock produce different time. Therefore, internal synchronization is normally performed for better performance.

Two approaches are used for internal synchronization:

① Global Averaging Distributed Algorithm.

⑪ Localized Averaging Distributed Algorithm.

9. What is event ordering?

Event Ordering:

keeping the clocks in a distributed system synchronized to within 5 or 10 msec is an expensive.

Lamport observed that for most applications it is not necessary to keep the clocks in a Distributed system synchronized.

Rather, it is sufficient to ensure that all events that occur in a distributed system be totally ordered in a manner that is consistent with an observed behavior.

For partial ordering of events, Lamport defined a new relation called happened-before introduced the concept of logical clocks for ordering of event based on the happened-before relation.

Q. Desc happened before relation with space time diagram.

Happened Before Relation:

The happened-before relation (denoted by $\rightarrow$) on a set of events satisfies the following conditions —

(i) If a and b are events in the same process and a occurs before b, then ... $a \rightarrow b$

(ii) If a is the event sending a message by one process and b is the event of the receipt of the same message by another process,

then $a \rightarrow b$

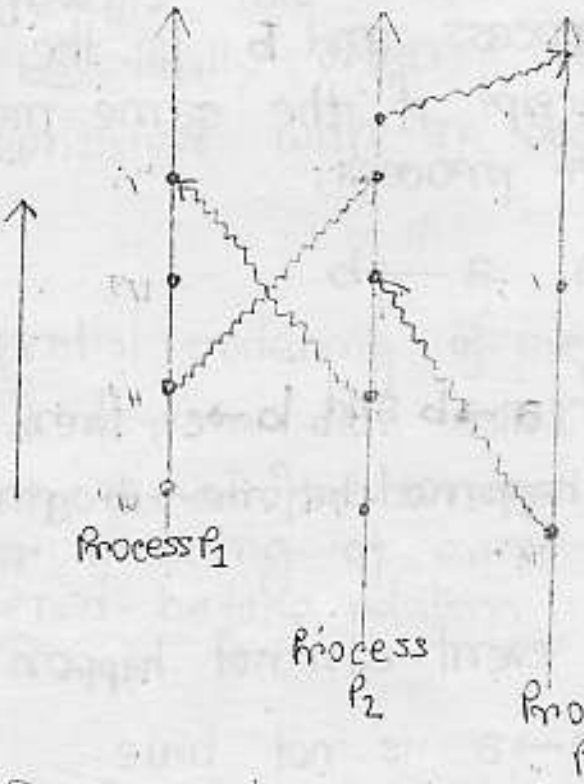
(iii) If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.

That is, happened-before is a transitive relation.

N.B (i) an event can not happen itself, that is $a \rightarrow a$ is not true.

③ In terms of happend-before relation, two events a and b are said to be concurrent if they are not related by the happend before relation. That is, neither $a \rightarrow b$ or $b \rightarrow a$ is true.

☐ Space Time Diagram of Three Process:



legends:
Vertical line = P
each dot = an even
wavy line = a mess
transfer from
process to anothe
in the direction
of the arrow

Fig. space time diagram for three process

A space time diagram is often used to illustrate the concepts of happened before relation and concurrent events. On this diagram, each vertical line denoted a process, each dot on a vertical line denotes an event in the corresponding process, and each wavy line denotes a message transfer from one process to another in the direction of the arrow.

From this space-time diagram it is easy to see that for two events and , is true if and only if there exists a path from to , by moving forward in time along process and message lines in the direction of the arrows.

For example, Some of the events of fig that we related by the happened-before relation are

$e_{10} \rightarrow e_{11}$.

$e_{20} \rightarrow e_{24}$.

$e_{21} \rightarrow e_{13}$.

$e_{30} \rightarrow e_{24}$ (since $e_{30} \rightarrow e_{22}$ and $e_{22} \rightarrow e_{24}$)

$e_{11} \rightarrow e_{32}$ (Since $e_{11} \rightarrow e_{23}$, $e_{23} \rightarrow e_{24}$, & $e_{24} \rightarrow e_{32}$)

On the otherhand, two events and are concurrent if no path exists either from a to 1 and 1 to 2.

For example: Some of concurrent event of fig are –

e_{12} and e_{20},
e_{21} and e_{30},
e_{10} and e_{30},
e_{11} and e_{31},
e_{12} and e_{31} &
e_{13} and e_{22}

Lec-7
18.01.12

Q.8) Desc concept of logical clock (lamport) algo.

2. Logical Clocks Concept:

To determine that an event a happened before an event b, either a common clock or a set of perfectly synchronized clocks is needed. We have seen that neither of these is available in a distributed system. Therefore, in a distributed system the happened-before relation must be defined without the use of globally synchronized physical clocks.

The lamport solution:

- The logical clocks concept is a way to associate a timestamp with each system event so that events that are related to each other by the happend-before relation (directly or indirectly) can be properly ordered in that sequence.

- Under this concept, each process P_i has a clock C_i associated with it that assigns a number $C_i(a)$ to any event a in that process. The clock of each process is called a logical clock.

In fact, the logical clocks may be implemented by counters with no actual timing mechanism.

The logical clocks of a system can be considered to be correct if the events of the system that are related to each other by the happened-before relation can be properly ordered using this clocks. The following clock condition—
for any two events a and b,
if $a \longrightarrow b$, then

$$C(a) < C(b)$$

Q.9] What is clock condition? What are the condition that satisfy the clock condition?

Implementation of logical clocks:

From the definition of the happened-before relation, is satisfied the following conditions —

[C1] If a and b are two events within the same process P_i and a occurs before b, then—

$$C_i(a) < C_i(b)$$

[C2] If a is the sending of a message by process P_i and b is the receipt of that

message by process P_j, then

$C_i(a) < C_j(b)$

[C3] A clock C_i associated with a process P_i must always go forward, never backward, That is, corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting value.

Q.10] What are the implementation of Lamport algorithm?

The algorithm proposed by Lamport is given— To meet conditions C1, C2 and C3. Lamport's algorithm use the following implementation rules:

[IR1] Each process P_i increments C_i between any two successive events.

[IR2] If event a is the sending of a message m by process P_i, the message m contains a timestamp $T_m = C_i(a)$, and upon receiving the message m a process P_j sets C_j greater than or equal to its present value but greater than T_m.

message by process P_j, then

$C_i(a) < C_j(b)$

[C3] A clock C_i associated with a process P_i must always go forward, never backward, That is, corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting value.

Q.10] What are the implementation of Lamport algorithm?

The algorithm proposed by Lamport is given— To meet conditions C1, C2 and C3, Lamport's algorithm use the following implementation rules:

[IR1] Each process P_i mements C_i between any two successive events.

[IR2] If event a is the sending of a message m by process P_i, the message m contains a timestamp $T_m = C_i(a)$, and upon receiving the message m a process P_j sets C_j greater than or equal to its present value but greater than T_m.

Rule IR1 ensures that condition C1 is satisfied and rule IR2 ensures that condition C2 is satisfied.

9.11 What are the implementation technique of logical clock

The logical clocks can be implemented —

① By using counters with no actual timing mechanism or

② By using physical clocks.

Lec-8
22.01.12

① Implementation of logical clocks by using counter:

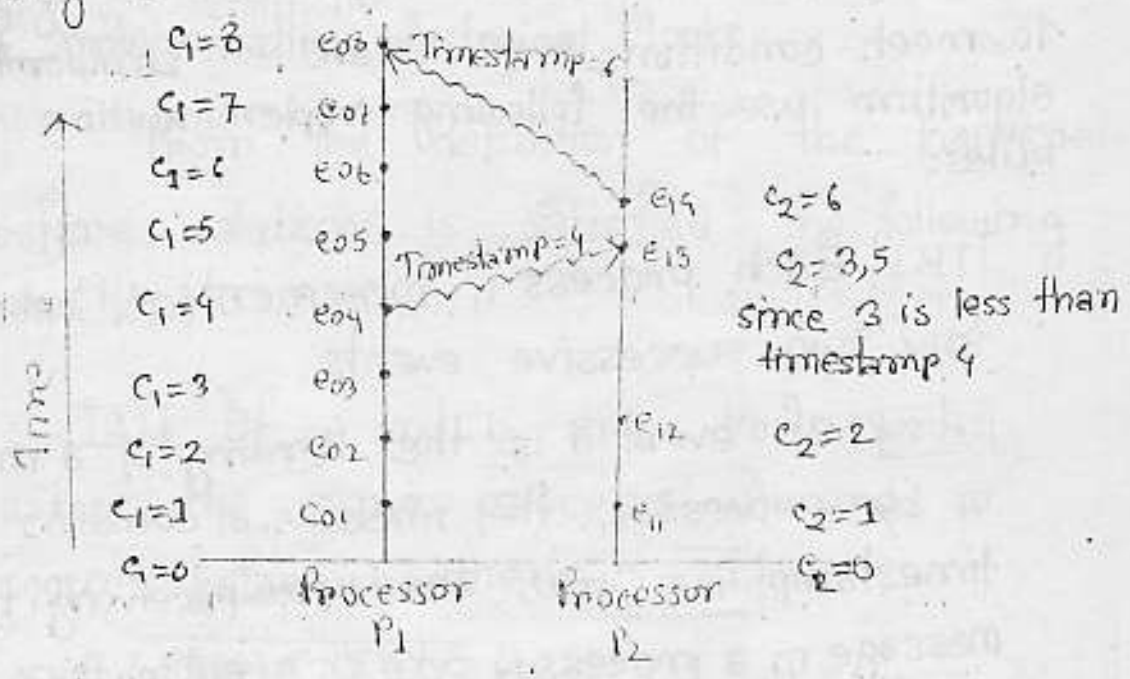


fig. shows two processor P_i & P_j each have a counter C_i and C_j respectively. The counter act as logical clocks. At the beginning the counter are initialized to zero and a process increments its counter by 1 whenever an events occurs in that process. If the event is sending of a message (e.g. event e_{04} and e_{14}) the process includes the incremented value of the counter in the message. On the other hand, if the event is receiving of a message (e.g. event e_{13} & e_{08}). instead of simply incrementing the counter by 1, a check is made to see if the incremented counter value is less than or equal to the timestamp in the received message. If so, the counter value is corrected and set to 1 plus the timestamp in the received message (e.g. event e_{13}). If not, the counter value is left as it is (event e_{08}).

(11) Implementation of logical clocks by using physical clocks:

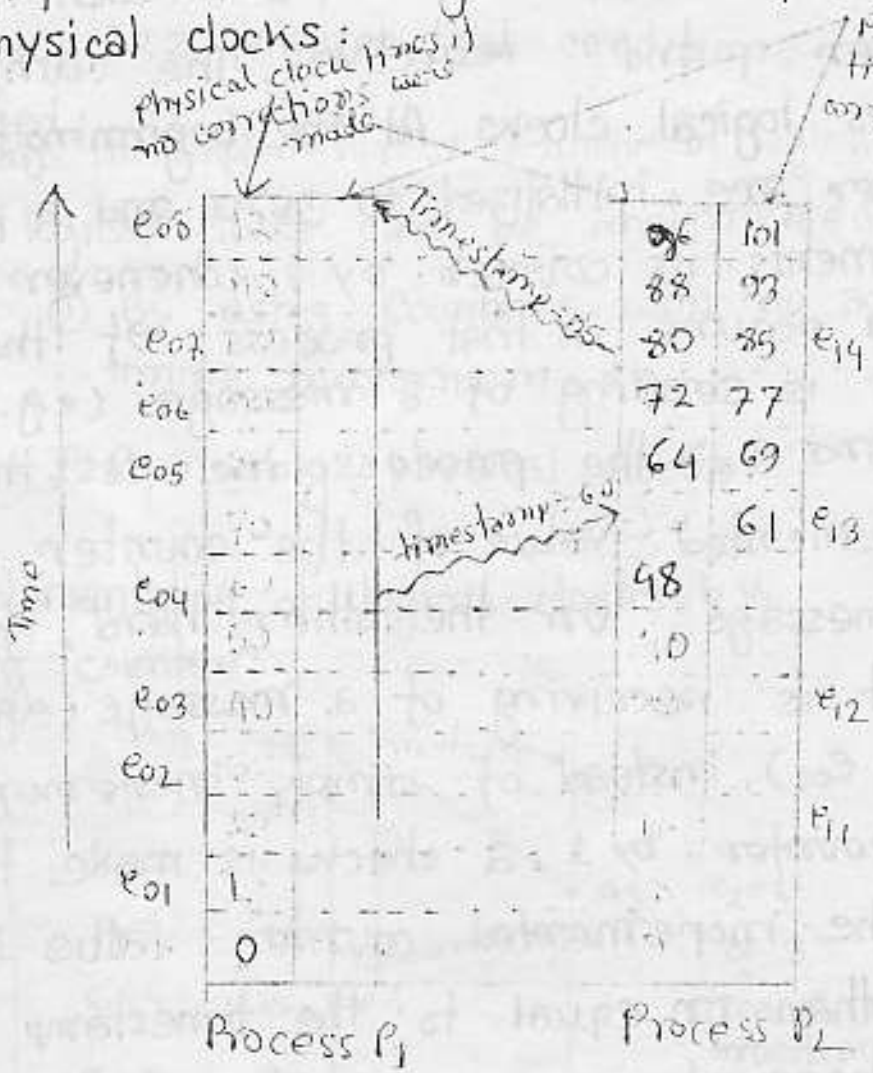


Fig. Implementation of logical clocks by using physical clocks

The implementation of logical clock by using physical clock was shows in the fig. In this case, each process has a physical clock associated with it. Each clock runs

at a constant rate. However, the rates at which different clocks run are different. On the fig, the clock of processor P1 has ticked 10 times, the clock of process P2 has ticked only 8 times.

To satisfy condition C1, the only requirement is that the physical clock of a process must tick at least once between any two events in that process. This is usually not a problem because a computer clock is normally designed to click several times between two events that happen in quick succession.

To satisfy condition C2, for a message sending (e.g. events e_{04} & e_{14}), the process sending the message includes its current physical time in the message. And for a message receiving event (e.g. events e_{13} & e_{08}) a check is made to see if the current time included in the message.

If so, the receivers physical clock is corrected by fast forwarding its clock to be 1 more than the time included in the message (e.g event e_{13}). If not, the receivers clock is left at it is (e.g. event e_{08}).

12. 6.4 MUTUAL EXCLUSION:

6.4.1 Algorithms Example Adv disadv

Q. Define mutual exclusion. What are the req for implementing mutual exclusion?

There are several resources in a system that must not be used simultaneously by multiple processes if program operation is to be correct.
Therefore, exclusive access to such a shared resource by a process must be ensured. This exclusiveness access is called Mutual Exclusion between processes.

Requirements for implementing Mutual Exclusion:

Mutual exclusion: Given a shared resource accessed by multiple concurrent processes, at any time only one process should access the resource. That is, a process that has been granted the resource must release it before it can be granted to another process.

Progress: If every process that is granted the resource eventually release it, every request must be eventually granted.

Q.11] Desc the distributed approach for mutual exclusion.

Distributed Approach:

In distributed approach, the decision making for mutual exclusion is distributed across the entire system. That is all process that want to entire the same critical section cooperate with each other before reaching a decision on which process will enter the critical section next. Later Ricart & Agrawala (1981) proposed a more efficient algorithm that also requires there be a total ordering of all events in the system. Ricart & Agrawala algorithm is discribed below —

When a process wants to enter a critical section, it sends a request message to all other processes. The message contains the following information —

(i) The process identifier of the process.
(ii) The name of the critical section that the process wants to enter.
(iii) A unique timestamp generated by the process for the request message.

On receiving a request message, it defers sending a reply based on the following rules —

① If the receiver process is itself currently executing in the critical section, it simply queues the request message and differs sending a reply.

② If the receiver process is currently no executing in the critical section but is waiting for its turn to enter the critical section, it compare the timestamp in the received request message with the time stamp its own request message that it has sand to other processes. If the time stamp of the received request message is lower, it means that the sender process made a request before the receiver process to enter the critical section.

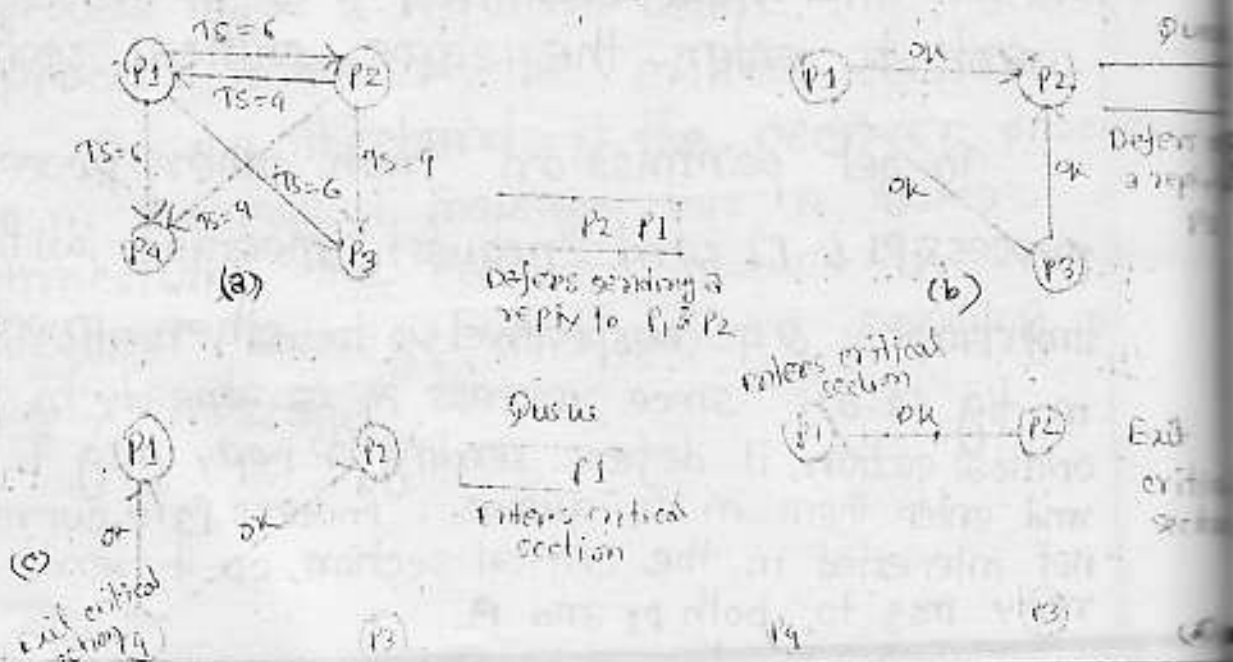
On the otherhand, if the receiver proeesses own the request message has a lower timestamp, the receiver queues the received request message and defers sending a reply message.

(iii) If the receiver process neither is in the critical section nor is waiting for its turn to enter the critical section, it immediately sends message.

A process that sends out a request message keeps waiting for reply message from other processes. It enters the critical section as soon as it has received reply message from all processes. After it finishes executing the critical section.

Ex: To illustrate how the algorithm works. Let us consider in fig(1). There are 4 processes P1, P2, P3 & P4. While process P4 is in a critical section, process P1 & P2 want to enter the same critical section.

To get permission from other processes, process P1 & P2 send request message with timestamps 6 & 4 respectively to other process in fig (1.a). Since process P4 is already in the critical section, it defers sending a reply msg to P1 & P2 and enter them in its queue. Process P3 is currently not interested in the critical section, so it sends a reply msg to both P1 and P2.


TS=6
TS=9
TS=6
TS=9
TS=6
TS=9
P1
P2
P3
P4
(a)
P2 P1
Defers sending a reply to P1 & P2
(b)
OK
OK
OK
Defers
(c)
Exits critical section
Enters critical section
P1
P2
P3
P4

[9] What is election algorithm ?

29.01.12

15. ☐ Election Algorithm:

Election algorithm are ment for electing a co-ordinator process ~~for~~ all from among the currently running process in such a manner that at any instance of time there is a single co-ordinator for all processes in the system. Election algorithms are based on the following assumptions –

(i) Each process in the system has a unique priority number.

(ii) Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the co-ordinator.

(iii) On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

Dese the bully algorithm with example 9.16

☐ The Bully Algorithm:

The algorithm was proposed by Granicia-Molina (1982). This algorithm is assumed that every process knows the priority number of

every other number in the system. The algorithm works as follows—

① when a process (Pi) sends a request message to the co-ordinator and does not receive a reply within a fixed timeout period, it assumed that the co-ordinator has failed. of then initiates an election by sending an election message to every process with a higher priority number than itself. of Pi does not receive any response to its election message within a fixed timeout period, it assums that among the currently active processes it has the highest priority number. Therefore it takes up the job the co-ordinator & sends a message to all processes having lower priority numbers than itself, informing that from how on it is the new co-ordinator.

② when a process Pj receives an election message it sends a response message to the sender informing that it is alive and will take over the election activity. Now Pj holds an election if it is not already holding one.

On this way, the election activity gradually moves on to the process that has the height priority number among the currently active processes and eventually wins the election and becomes the new co-ordinator.

(iii) As part of the recovery action, this method requires that a failed processes (P_k) must initiate an election on recovery. If the current co-ordinators priority number is highest than that of P_k, initiated by P_k & will continue to be co-ordinator. On the otherhand, if P_k's priority number is higher than that of the current co-ordinator it will not receive any response for its election message. So it wins the election & take over the co-ordinator job from the currently active co-ordinator. Therefore the active process having the highest priority number always wins the election. Hence, the algorithm is called the "Bully Algorithm".

Example:

Let five processes P1, P2, P3, P4 & P5. And their priority number are 1, 2, 3, 4 & 5 respectively. Also suppose that at a particular instance of time the system is in a state in which P2 is crashed and P1, P3, P4 & P5 are active. Starting from this state, the functioning of the bully algorithm with the changing system states in illustrated below –

① Obviously, P5 is the co-ordinator in the starting state.

② Suppose P5 crashes.

③ Process P3 sends a request message to P5 and does not receive a reply within the fixed time out period.

④ Process P3 assums that P5 has crashed and initiates an election message to P4 & P5.

⑤ When P4 receives P3's election message, it sends an alive message to P3, informing that it is alive and will take over the election activity. Process P5 cannot respond to P3's election message because it is down.

(vi) Now P4 holds an election by sending an election message to P5.

(vii) Process P5 does not respond to P4's election message because it is down. and therefore, P4 wins the election and sends a co-ordinator message to P1, P2 & P3 informing them that from now on it is the new co-ordinator. Obviously this message is not received by P2 because it is currently down.

(viii) Now. suppose. P2 recovers from failure and initiates an election by sending an election message to P3. P4 & P5 since P2's priority number is lower than that of P4 (current co-ordinator) P4 will win the election initiated P2 and will continue to be the co-ordinator.

(ix) Finally, suppose P5 recovers from failure since P5 is the process with the highest priority number it simply sends a co-ordinator message to P1, P2, P3 & P4 and becomes the new co-ordinator.

[9.17] Compare bet^n bully & ring algorithm.

Defⁿ of Deadlock:

Deadlock is the state of permanent blocking of a set of processes each of which is waiting for an event that only another process in the set can cause..

Necessary Condition of Deadlock:

(i) Mutual-exclusion condition

(ii) Hold and wait condition

(iii) No-preemption condition

(iv) Circular wait condition.

Deadlock:

The sequence of events required to use a resource by a process is as follows —

(i) Request: The process first makes a request for the resource if the request resource is not available, the requesting process must wait until the requested resource is allocated to it by the system.

(ii) Allocate: The system allocates the resource to requesting process as soon as possible. It maintains a table in which it records whether each resource is free or allocated and if it is allocated to which process.

(iii) Release: After the process has finished using the allocated resource, it releases the resource to the system.

(i) Mutual-exclusion condition.

If a resource is held by a process any other process requesting for that resource must wait until the resource has been released.

(ii) Hold and wait condition.

Process are allowed to request for new resources without releasing the resources that they are currently holding.

(iii) No-preemption condition.

A resource that has been allocated to a process becomes available for allocation to another process only after it has been released by the process holding it.

(IV) Circular wait condition:

Two or more processes must form a circular chain in which each process is waiting for a resource that is held by the next number of the chain.

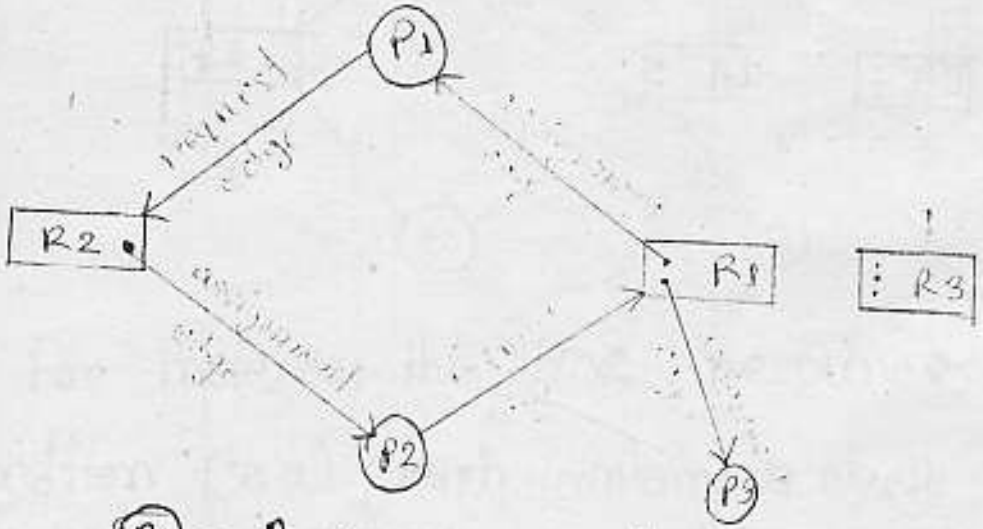
Resource Allocation Graph (for deadlock modeling):

(I) Process nodes

(II) Resource nodes

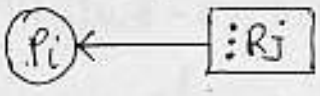
(III) Assignment edges

(IV) Request edges.

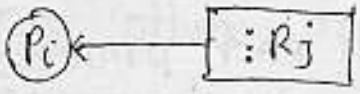


(Pi) = A process named Pi

[Rj] = A resource Rj having 3 units in the system



= Process P_i holding a unit of resource R_j



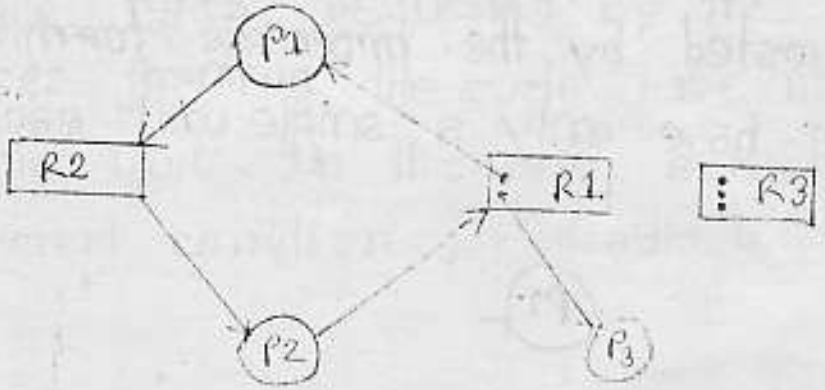
= Process P_i requesting for a unit of resource R_j

Fig. Resource allocation graph

Necessary & Sufficient conditions for deadlock:

On a resource allocation graph, a cycle is a necessary condition for a deadlock exist. That is, if the graph has no cycles, then it represents a state that is free from deadlock.

On the otherhand, if the graph contains a cycle, a deadlock may exist. Therefore, the presence of a cycle in a general resource allocation graph is a necessary but not a sufficient condition for the existence of the deadlock.



Fig(a)

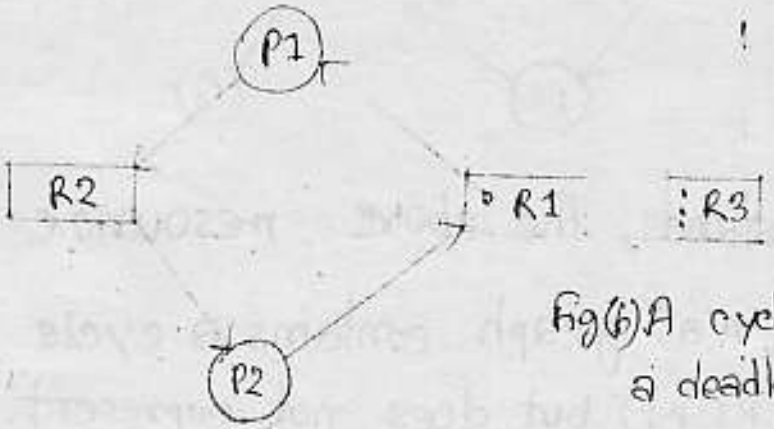
For instance, the above resource allocation (r.a) graph contains a cycle (P1, P2, P2, R1, P1) but does not represent

a deadlock state. Because, when P3 completes using R1 and release it, R1 can be allocated to P2. With both R1 & R2 allocate to it, P2 can now complete its job. As soon as R2 is released it can be allocated to P1.

Therefore, all processes can finish their jobs one by one. The sufficient condition for deadlock is different for the following different cases.

① A cycle in the graph is both a necessary and a sufficient condition for deadlock if all the resource types requested by the processes forming the cycle have only a single unit each.

Ex:



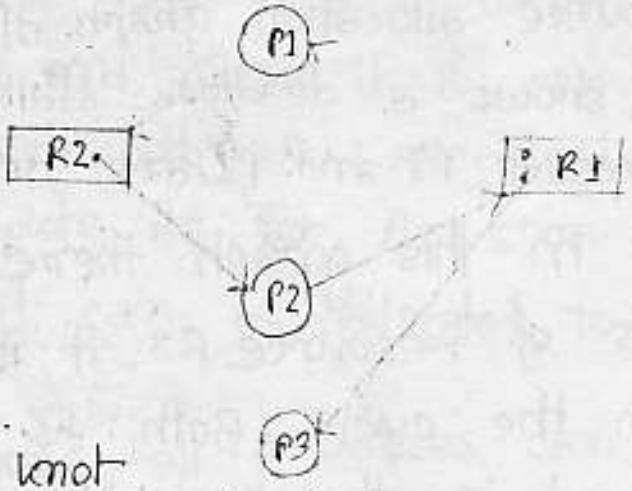
Fig(6) A cycle representing a deadlock.

The resource allocation graph of the above fig shows a deadlock state in which processes P1 and P2 are deadlocked. Notice that in this graph there are three units of resource R3, it is not involved in the cycle. Both R1 & R2 that are involved in the cycle have only one units each. Therefore, the cycle represents a deadlock state.

(ii) A cycle in the graph is a necessary but not a sufficient condition for deadlock if one or more of the resource types requested by the processes forming the cycle have more than one unit. In this case, a knot is a sufficient condition for deadlock.

Ex.

Ex.



fig(c) A knot representing a deadlock

The fig (a) does not represent a deadlock. This is because type R1 has two units and there are no knot in the graph.

Now, suppose, in the same graph P3 requests for R2 and a request edge (P3 P2) is added to the graph. The modified graph is shown in fig(c). This graph has two cycles —

- (P1, R2, P2, R1, P1) &
- (P3, R2, P2, R1, P3) and a knot

{P1, P2, P3, R1, R2} Since the graph contains a knot, it represents a deadlock state in which process P1, P2 & P3 are deadlocked.

Q. What do you mean by fault? Explain different types of fault:

⇒ Fault:

A system is said to fail when it does not meet its specification. A fault is a malfunction, possibly caused by a design error, a manufacturing error, a programming error, physical damage and many other causes.

classification of faults:

(i) Transient fault
(ii) Intermittent fault
(iii) Permanent fault

(i) Transient fault:

It occurs once and then disappears. If the operation is repeated the fault goes away. e.g. A bird flying through the beam of a microwave transmitter may cause lost of bits on some network.

(ii) Intermittent fault:

An intermittent fault occurs, then vanishes of its own occured, then reappears and so on.

e.g. A loose contract on a connector will often cause an intermittent fault.

(iii) Permanent fault:

→ It is one that continue to exists until the faulty component is repaired.
e.g. Burn-out chips, software bugs, and disk head crashes often cause permanent faults.

Processor faults: Two types.

- Fail-silent faults:

With fail-silent faults a faulty processor just stops and does not respond to subsequent input or procedure further output announcing that it is no longer functioning. These are called "Fail-stop" faults.

- Byzantine faults:

With Byzantine fault a faulty processor continue to run issuing wrong answers to questions and possibly working together maliciously with other faulty processor to give the impression that they are all working correctly.

Q Describe a way of fault-tolerance using active replication.

OR, Explain TMR (Triple Modular Redundency) for fault-tolerance.

OR, Write short note on TMR.

⇒ Fault tolerance using Active Replication:

→ Where all the devices are used all the time in order to hide the fault completely.

Physical Redundency:

→ Extra equipment is added to make it possible for the system as a whole to tolerate the malfunctioning of some components.

→ Active Replication is a well-known technique for providing fault-tolerance using physical redundency. it has also been used for fault-tolerance in electronic circuits for years. e.g. consider the following circuit

Here, signal pass through devices A, B & C in sequence. If one of them is faulty, the final result will

probably be wrong.

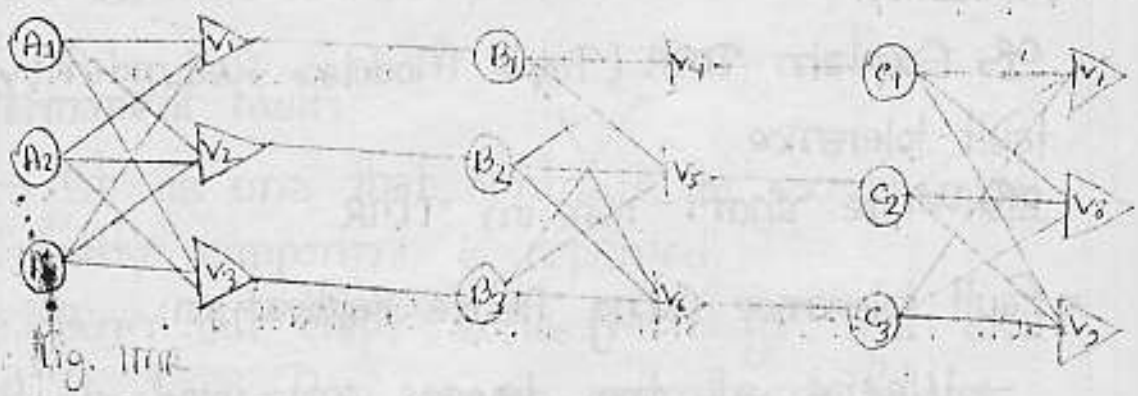


Fig. TMR

→ Each device is replicated three terms
→ Each vector is a circuit that has three inputs and one output.
→ If two or three of the inputs are the same, the output is equal to that input
→ If all these inputs are different, the output is undefined. This word of design is known as TMR.

Suppose elements A2 fails. Each of the ~~vectors~~ V1, V2 and V3 gets two good (identical) inputs and one rough input and each of them outputs the correct value to the second stage.

• In essence, the effect of A2 failing is completely masked. so that the inputs to B_1, B_2 & B_3 are exactly the same as they would have been had no fault occured.

• If B_3 and E_1 also faulty in addition to A_2. These effects are also masked. So, the three final output are still correct.

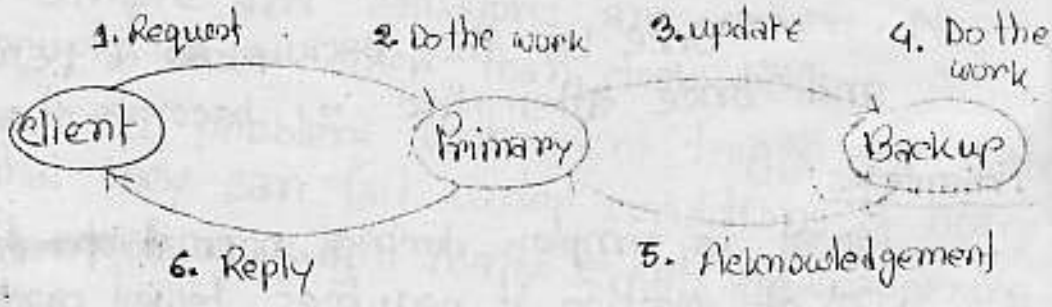
• Afterall, one voter could also detect and pass through the majority view.

e.g. V_1 malfunctions. The input to B_3 will then wrong, but as long as everything also works B_2 and B_3 will produce the same outputs and V_4, V_5 and V_6 will not produce the correct result into stage three.

A fault in V_1 is effective no different than a fault in B1. In both cases, B1 produces incorrect output but in both case it is voted down later.

Explain: Fault tolerance using primary backup

Primary backup method is that at any one instant, one server is the primary and does all the work. If the primary fails the backup takes over. The following fig. shows a write operation.



• The client send a message to the primary which does the work and then send on update message to the backup. When the backup gets the message it does the work and then sends an acknowledgement back to the primary. when the ack. arrives, the primary sends the reply to the client.

• Now, let us consider the effect of primary crash at various moments during an RPC.

① If the primary crashes before doing the work (step 2) no harm is done. The client will time out and retry.

If the p.c

② After doing the work but before sending the update (step 3) when the backup takes over and the result comes again the work will be done a second time.

③ If the primary crashes after step 4, but before step 6. The work may end up being done three-times.

Once by the primary

Once by the backup as a result of step

and once after the " becomes the primary

Advantages:

① It is simpler during normal operation

② It practice it requires fewer machines.

Application in real life:

① Government (The vice president)

This approach is widely used in the world.

② Aviation (co-pilots)

③ Automobiles (spare tires)

and,

diesel-powered electrical generators in hospital operating rooms.

⊞ Difference betⁿ event triggered & time-triggered system:

• In an event-triggered real time system when a significant event in the outside world happens, it is detected by some sensor, which then causes the attached cpu to get an interrupt.

Even triggered systems are thus interrupt driven.

Time Triggered:

In this kind of system, a clock interrupt occurs every AT milliseconds. At each clock tick sensors are sample and actuators are driven. No interrupts occur other than clock ticks.

• The main problem with event triggered systems is that they can fail under conditions of heavy loads, that is, when many events are happening at once.

Distributed file System

Q1 Define distributed file system.

⇒DFS: A key component of any distributed system is the file system. As in single processor systems, in distributed systems the job of the file system is to store programs and data and make them available as needed.

Q: Define & distmgish betn file service & file server.

File service: The file service is the specification of what the file system offers to its clients. It describe the primitive available, what parameters they take and what actions they perform. In effect, the file service specifies the file systems interface to the client.

File server: is a process that runs well on some machine and helps implement the file service. A system may have one file server or several. The clients should not know how many file servers there are and what the location or function of each one is.

Distributed file System Design: (DFS)

It has typically has two distinct components.

(I) The file service interface
e.g. reading, writting, appending etc.

(II) The directory server interface
e.g. creating & managing directories adding & deleting files from directories.

File Service Types:

It can be split into two types.

1. Upload / Download Model
2. Remote Access Model

1) Upload / Download Model:

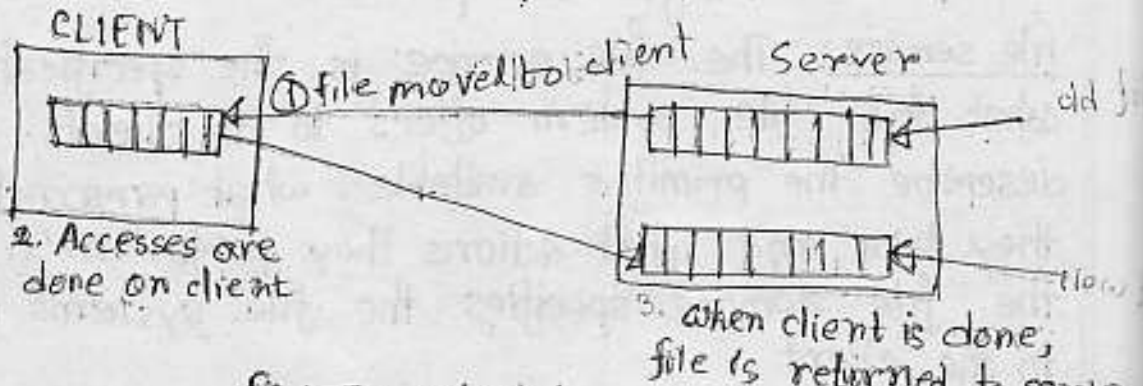


Fig: The upload/download model.

→ In this model, the file services provides any two major operations.

i) Read file: The read operation transfers an entire file from one of the file servers to the requesting client.

ii) Write file: The write operation transfers an entire file from the client to server.

Advantages:

i) Conceptual Simplicity.
i.e., application programs fetch the files that needed, then use them locally

ii) Any modified or newly created files are written back when the program finishes.

Disadvantages:

i) Enough storage must be available on the client to store all the required files.

ii) If only a fraction of a file is needed moving the entire file is wasteful

ii) Remote Access Model:

In this model the file service provides a large number of operations for openning and closing files, reading and writting parts of files, moving around within files examining and changing file attributes and so on.

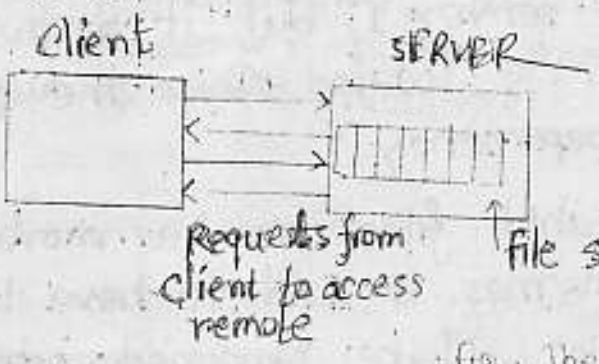


fig: the remote access model

Advantages:

i) Nol requires much space on the clients

ii) Eliminating the need to pull the entire files when only small pieces are needed.

Disadvantages:

i) Implementation complex

ii) No way to use the files on client side

Naming Transparency:

→ The name of the file should give no hint as to where the file is located. Furthermore, a file should be allowed to move from one node to another in a distributed system without having to change the name of the file.

→ It observed in two forms —

① Location Transparency: means that the path name give no hint as to where the file is located. A path like server 1/ dir 1/ x tells everyone that x is located on server 1 but it does not tell where that server is located. Thus the system has location transparency.

② A system in which files can be moved without changing their names is said to have Location Independence. This will be happened when the file x is extremely large and space is tight on server 1. The system might well like to move x to server 2 automatically.

Semantic of file Sharing:

When two or more users share the same file it is necessary to define the semantics of reading and writing precisely to avoid prob

There are 4 approaches for dealing with shared files in a distributed system.

1. UNIX Semantic: Every operation on a file is instantly visible to all processes.
2. Session Semantic: No changes are visible to other processes until the file is closed.
3. Immutable files: No changes are possible, simplifies sharing and replication.
4. Transaction: All changes have the all-or-nothing properly.

Iterative Look-up: What are the ways that file servers and directory servers are organized internally?

On this method, the client should be aware of which server holds which directory and requires more message

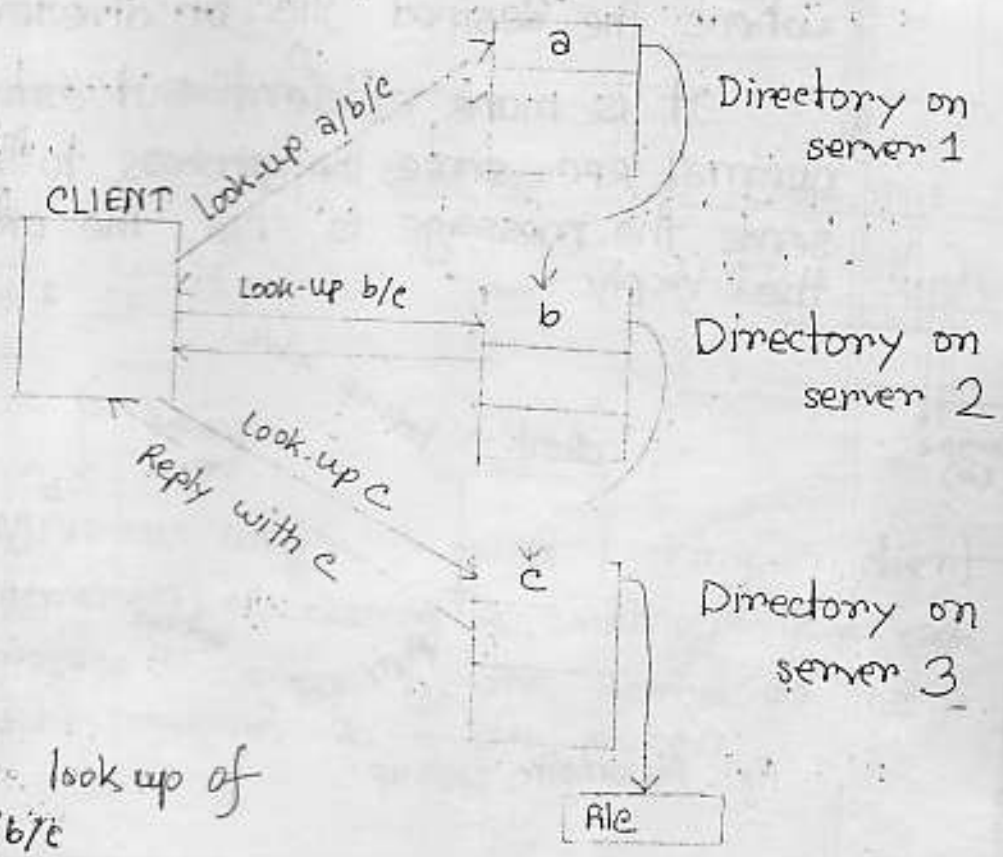


fig. Iterative look up of a/b/c

e.g. A system in which the current directory on server 1; Contains on entry 'a', for another directory on server 2. similarly, this directory contains on entry 'b' for a directory on server 3. This third directory an entry for a file c; along with its binary name. To look-up a/b/c; the client sends a message to server 1, which manages its current directory. The server finds 'a' but sees that the binary name refers to another server. It now has a choice. It can either tell the client which server holds b and have the client look up b/c there itself.

ii) Automatic Look-up:

On this method, the server itself for word the remainder of the request of the next server where the desired file or directory exists.

It is more efficient but cannot maintain normal RPC. since the process to which the client sends the message is not the one that sends the reply.

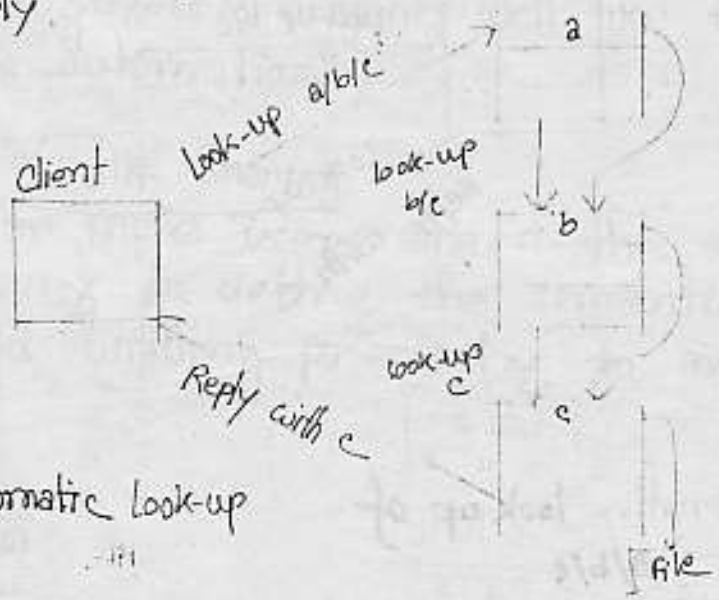
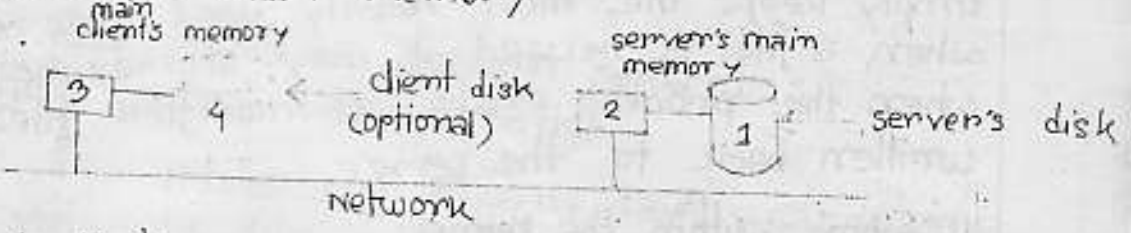


fig. Automatic look-up

Q. What do you mean by caching? Explain the different way of caching in clients, main memory.

Caching: In a client server system, there are 4 potential places to share files or parts of files:

(i) The server's Disk
(ii) The Server's main memory
(iii) The clients Disk (if any) or
(iv) The clients main memory



Caching in clients memory:

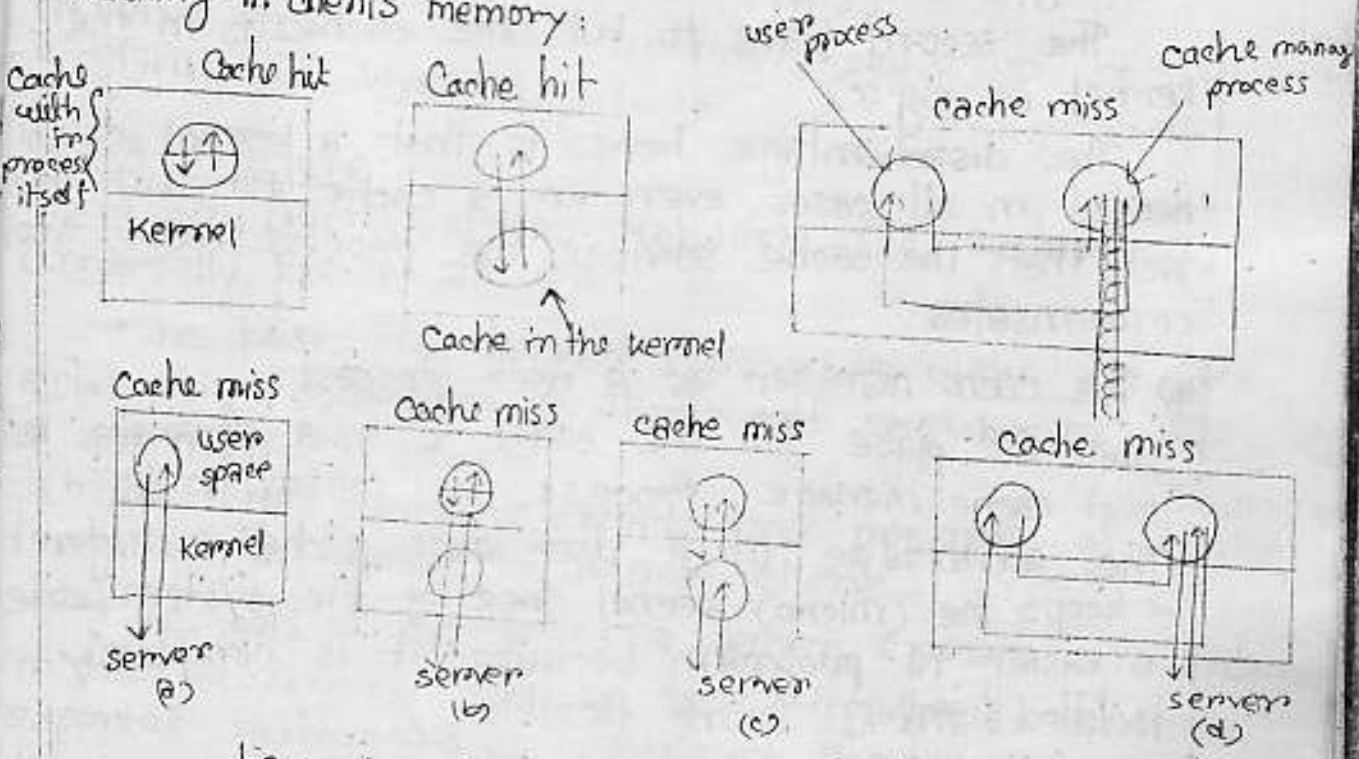


fig: Various ways of doing caching in client memory (a) No caching (b) Caching within each process (c) Caching in the kernel (d) The cache manages as a user process.

→ To put the cache in the clients main memory there are 3 options are open as to precisely where to put it.

i) Caching within each process:

The simplest way is to cache files directly inside each user process own address space, as show in fig (b). typically, the system call library manages the cache. As files opened, closed, read & written the library simply keeps the most heavily used ones around. So that when a file is reused, it may already be available when the process exists, all modified files are written back to the server.

ii) Caching within the kernel:

The second place to but the cache is in the kernel in fig (c).

The disadvantage here is that a kernel call is needed in all cases, even on a cache hit, but the fact that the cache services the process more than compansates.

iii) The cache manager as a user process:

The 3rd place for the cache is in a separate user level cache manage: process, as shown in fig (d). The advantage of a user-level cache manager is that it keeps the (micro) kernel free of file system code, is easier to program because it is completely isolated, and is more flexible.

Cache Consistency: of two clients simultaneously read the same file and the both modify it, several problems occur. For one, when a 3rd process reads the file from the server, it will get the original version not one of the two new ones. This problem is known as cache inconsistency.

There are 4 ways to solve the inconsistency problem. These are —

(i) Write Through: when a ~~cash~~ cache entry is modified, the new value is kept in the cache, but is also sent immediately to the server. so when another process reads the file it gets the most recent value.

ii) Delayed Write: Better performance but possibly ambiguous semantics.

iii) Write on close: Matches session semantics.

iv) Centralized Control: UNIX sematics.

Q] Explain the architectural and layer structure of sun's network file system (NFS).
or, What are the basic architectural chair of NFS? Explain NFS layer structure.

NFS Architecture:

→ Sun Microsystems Network file System universally known as NFS.

→ The basic idea behind NFS is to allow an arbitrary collection of clients and servers to share a common file system.

→ Each NFS servers exports one or more of its directories for access by remote clients.

→ The lists of directories a server exports is maintained in the /etc/exports file, so these directories can be exported automatically whenever the server is booted.

Characteristics of NFS —

① With NFS is that servers export directories and clients mount them remotely.

② If two or more clients mount the same directory at the same time, they can communicate by sharing files in their common directories.

iii) A program on one client can create a file an
a program on a different one can read the file
iv) Once mount have been done, nothing special has
to be done to achieve sharing.

NFS Implementation / NFS layer structure:

→ It consists of 2 layers

① System call layer ② Virtual file System layer (V
iii) Node's OS layer.

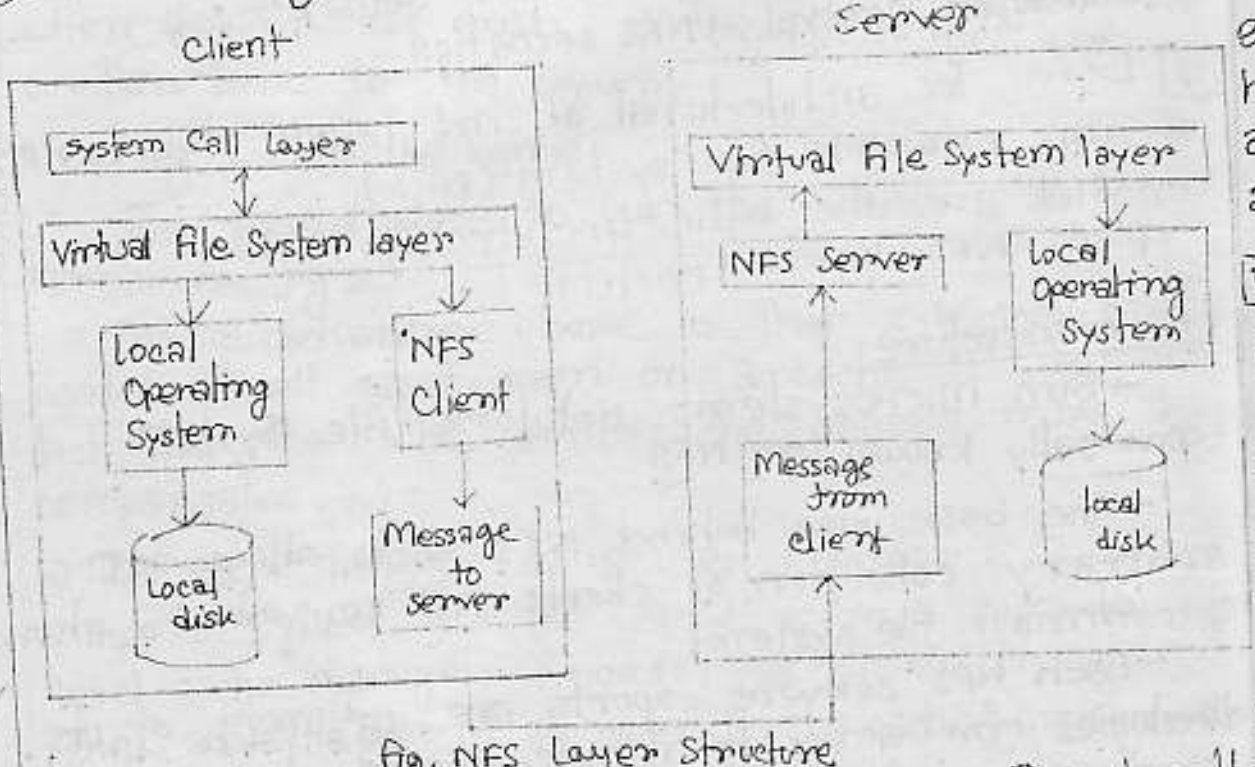


Fig. NFS Layer Structure

The top layer is the system call layer. This handl
calls like OPEN, READ and CLOSE. After parsing the
call and checking the parameters, it invokes the
second layer, the Virtual File System (VFS) Laye

The task of VFS layer is to maintain a table wi
one entry for each open file, instead the VFS
layer has an entry, called a v-node (virtual i-no
for each open file. V nodes are used to tell whether
the file is local or remote. For Remote files, enough
information is provided to be able to access them.

Let, a seq
To mount
calls the
the local
and other
The mount
directory
directory
available
file hand
of them
handle for
and is ava
a file han
① Explain
stateful S
A state
from one
* To illu
we cons
allow the
That is
State inf
for the
is stored
These
the time
client

Let, a sequence of MOUNT, OPEN and READ system calls. To mount a remote file system the system administrator calls the mount program specifying the remote directory, the local directory on which it is to be mounted and other information.

The mount program parses the name of the remote directory to be mounted on which the remote directory is located. of the directory exists and is available for remote mounting, the server returns a file handle for the directory.

of then contacts that machine asking for a file handle for the remote directory. of the directory exists and is available for remote mounting, the server returns a file handle for the directory.

(9) Explain Stateful and Stateless Servers.

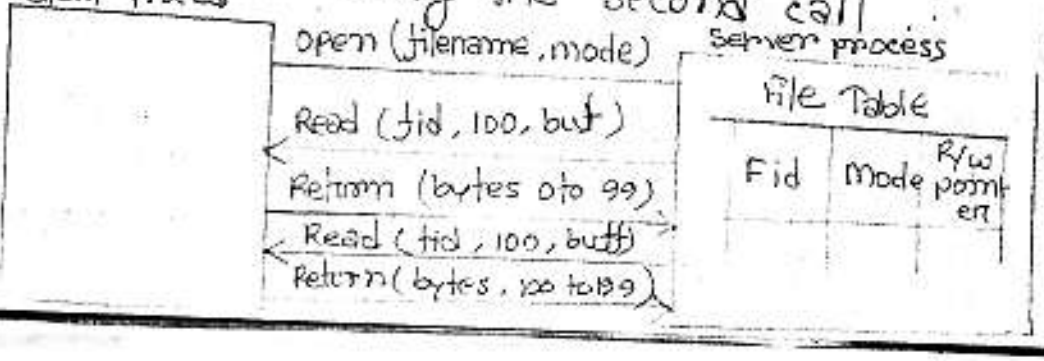
Stateful Server:

A stateful file server maintains clients state information from one access request to the next.

* To illustrate how a stateful file server works. Let us consider a file server for byte-stream files that allows the following operations on files.

That is, in case of two subsequent calls, some state information pertaining to the service performed for the client as a result of the first call execution is stored by the server process.

These client's state info. is subsequently used as the time of executing the second call.



Q. Difference betⁿ Local procedure call and remote procedure call.

①

Difference:

Stateful Server	Stateless Server
① It is ~~more~~ not inherently more fault tolerant.	① It is inherently more fault tolerant.
② With a stateful server, if a client is crashes when a file is open, the server is in a bind.	② No problem if a client is crashes.
③ Reliable communication.	③ Unreliable communicati
④ File locking is possible.	④ File locking is impossil
⑤ If it does nothing, its table will eventually fill up with junk.	⑤ When table is used, if many clients have too ma files open at once, the table can fill up and ne files cannot be opened
⑥ It has limited number of open file.	⑥ no limits on number of open file.
⑦ Server space is wasted on tables.	⑦ No server space waste on tables.
⑧ Better performance is frequently possible.	⑧ Better performance is n found.