B.Tech. TCS-802

Eight Semester Examination 2009-10

Advanced Computer Architecture

Q.1. Attempt any two parts of the following :

Q. 1. (a) Analyse the data dependencies among the following statements in a given program :

S_1 : Load R_1, 1024	$R_1 \leftarrow 1024$
S_2 : Load R_2, M(10)	$R_2 \leftarrow$ Memory (10)
S_3 : Add R_1, R_2	$R_1 \leftarrow (R_1) + (R_2)$
S_4 : Store M(1024), R_1	Memory (1024) $\leftarrow (R_1)$
S_5 : Store M(R_2), 1024	Memory (64) $\leftarrow$ 1024

Note that (Ri) means that the content of register Ri and Memory (10) contains 64 initially.

(i) Draw a dependence graph to show all the dependencies.

(ii) Are there any resource dependencies if only one copy of each functional unit is available in the CPU ?

Ans.

S_1 : Load R_1, 1024	$R_1 \leftarrow 1024$
S_2 : Load R_2, M(10)	$R_2 \leftarrow$ Memory (10)
S_3 : Add R_1, R_2	$R_1 \leftarrow (R_1) + (R_2)$
S_4 : Store M(1024), R_1	Memory (1024) $\leftarrow (R_1)$
S_5 : Store M(R_2), 1024	Memory (64) $\leftarrow$ 1024

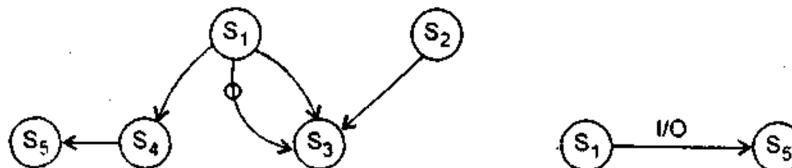
Dependence graph,

flow dependence $\rightarrow S_1 \rightarrow S_2$ o/p of S_1, input of S_2

Antidependence $S_1 \rightarrow S_2$ if S_2 follows S_2 in order and o/p of S_2 overlaps the i/p to S_1.

O/P dependence $S_1 \rightarrow S_2$ if some o/p variable is written.

I/O dependence because same file is referenced.



Resource dependence concerned with the complicts in using shared resources, such circuit units, floating point units, and registors among parallel events.

These will be no resource dependency in the above graph in spite of only one copy of each functional unit is available.

www.btechsamplepapers.in

Q. 1. (b) Explain the types of system performance factors in a parallel architecture.

Ans. Types of system performance factors in parallel architecture Clock Rate : CPU is driven by a clock with a constant cycle time (τ in nanoseconds).

$$\text{Clock rate} = \frac{1}{\tau} = (f) \text{ MHz}$$

$$f \rightarrow \text{clock rate}$$

Instruction Count (I_c) no. of machine instructions to executed in program, it determines the size of program.

CPI (Cycles per Instruction) : Average cycle per instruction provides frequency of appearance in program.

Perameter for measuring the time needed to execute each instruction.

Performance factors : CPU time needed to execute a program

$$T = I_C \times CPI \times \tau$$

Usuall a memory cycle is K-times the processor cycle τ.

$$T = I_c \times (P + m \times K) + \tau$$

$P \rightarrow$ no. of processor cycle needed for instruction decode and execution.

$m \rightarrow$ no. of memory references needed

$K \rightarrow$ ratio between memory cycle and processor cycle.

$I_c \rightarrow$ Instruction count $\tau \rightarrow$ processor cycle time.

MIPS Rate : $C \rightarrow$ total no. of clock cycle needed to execute a given program.

$$\text{CPU Time } T = C \times \tau = \frac{C}{f}$$

$$\text{CPI} = \frac{C}{I_c} \text{ and } T = I_c \times CPI \times \tau$$

$$T = I_c \times \frac{CPI}{f}$$

processor speed often measured in MIPS.

Million instructions per second.

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} = \frac{f \times I_c}{C \times 10^6}$$

Throughput Rate : $W_s \rightarrow$ System throughput (prog./sec)

$W_s < W_p$ (CPU throughput)

$$W_p = (MIPS) \times \frac{10^6}{I_c}$$

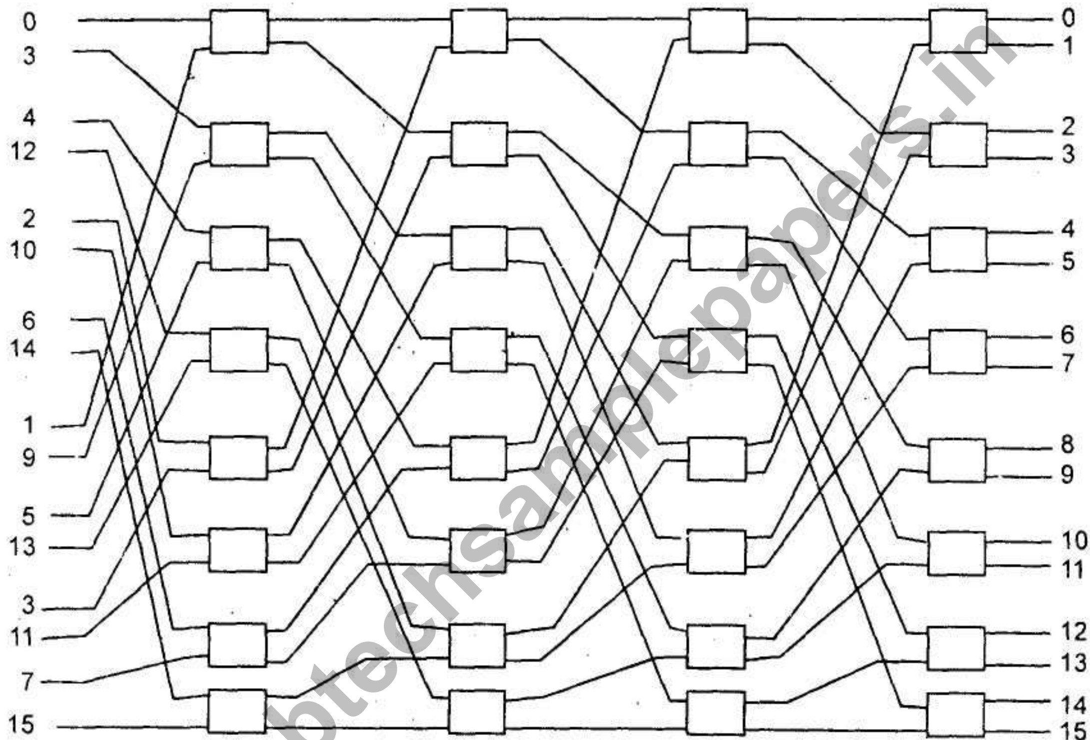
$$W_p = \frac{f}{I_c \times CPI}$$

www.btechsamplepapers.in

Q. 1. (c) Draw a 16 bit omega network using 2×2 switches as building block.

Ans. 16-bit omega N/w using 2×2 switches as building block.

Generally, an n-input omega network requires $\log_2 n$ stages of 2×2 switches. Each stage requires $\frac{n}{2}$ switch modules. In total network uses $n \log_2 \frac{n}{2}$ switches.



www.btechsamplepapers.in

Q. 2. Attempt any two parts of the following :

Q. 2. (a) Consider n level hierarchical memory, let 'h_i' be hit ratio at level M_i. Show that access frequency to 'M_i' is given by :

$$fi = (1 - h_1)(1 - h_2)(1 - h_{i-1}) \cdot hi$$

Further show that effective access time :

$$T_{eff} = \Sigma fi\, t_i$$

here 't_i' are measured with respect to CPU.

Ans. Consider M_i and M_{i-1} in a hierarchy, $i = 1, 2, ..., n$ hit ratio h_i at M_i is probability that an info. item will be found in M_i. Its a function of characteristics of two adjacent levels M_{i-1} and M_i. The miss ratio is $1 - h_i$ at M_i.

We assume $h_0 = 0$ and $h_n = 1$, and access to outermost memory M_n is always a hil.

The access frequency to M_i is defined as $f_i = (1 - h_1)(1 - h_2), \ldots (1 - h_{i-1}) \cdot h_i$

This is the probability of successfully accessing M_i when these are $i-1$ misses, at lower level and a hit at M_i.

$$\sum_{i=1}^{n} f_i = 1 \text{ and } f_1 = h_1$$

Due to locality property,

$f_1 >> f_2 >> f_3 >> ..... >> f_n$ *i.e.*, inner levels of memory are accessed more often than the outer levels.

Every time a miss occurs, a penalty is paid to occur the next higher level of memory. The misses have been called block misses in cache and page fault in main memory, because blocks and pages are the units of transfer between these levels.

The time penalty for a page fault is much longer than that for a block miss due to the fact that $t_1 < t_2 < t_3$.

Using the access frequencies f_i for $i = 1, 2, ..., n$, we can formally define the effective access time for a memory hierarchy,

$$T_{eff} = \sum_{i=1}^{n} f_i \cdot t_i$$

$$= h_1 t_1 + (1-h_1)h_2 t_2 + (1-h_1)(1-h_2)h_3 t_3 + ... + (1-h_1)(1-h_2) ....(1-h_{n-1})t_n$$

Q. 2. (b) Compare and contrast static interconnection network and dynamic interconnection network ?

Ans. Interconnection networks : Interconnection networks make a major factor to differentiate modern multiprocessor architectures. They can be categorized according to a number of criteria such as topology, routing strategy and switching technique . Interconnection networks are bulid up of switching elements; topology is the pattern in which the individual switches are connected to other elements, like processors, memories and other switches.

Direct topologies connect each switch directly to a node, while in indirect topologies at least some of the switches connect to other switches.

Using switching technique as a criterion one can mention some classes:

- circuit switching, in which the entire path through the network is reserved before a message is transferred,
- packet switching with virtual cutthrough, in which a packet is forwarded immediately after it determines an appropriate switch output,
- wormhole routing, which relaxes requirements of completely buffering of blocked packets in a single switch, typical for packet switching.

Wormhole routing is currently the most popular technique in commercial parallel machines.

Queuing technique is another important factor in switching network architectures, since it strongly influences the aggregated bandwidth of the network. In the simple input queuing technique the packets queue at the switch input awaiting the availability of the desired switch output; higher performance is offered by output queuing which in turn is difficult to implement. A solution is to form multiple queues at each switch input or to create a central buffer shared among all switch inputs and outputs.

www.btechsamplepapers.in

The networks can be classified as static or dynamic. Static interconnection networks are mainly used in message-passing architectures; the following types are commonly defined:

- completely-connected network.
- star-connected network.
- linear array or ring of processors.
- mesh network (in 2- or 3D). Each processor has a direct link to four/six (in 2D/3D) neighbour processors. Extensions of this kind of networks is a wraparound mesh or torus. Commercial examples are Intel Paragon XP/S and Cray T3D/E. These examples cover also another class, namely the direct network topology.
- tree network of processors. Communication bottleneck likely to occur in large configurations can be alleviated by inreasing the number of communication links for processors closer to the root, which results in the fat-tree topology, efficiently used in the TMC CM5 computer. CM5 could be also an example of indirect network topology.
- hypercube network. Classically this is a multidimensional mesh of processors with exactly two processors in each dimension. An example of such a system is the Intel iPSC/860 computer. Some new projects incorporate the idea of several procesors in each node which results in fat hypercube, i.e. indirect network topology. An example is the SGI/Cray Origin2000 computer.

Dynamics interconnection networks implement one of four main alternatives:

- **Bus-based networks :** the simplest and efficient solution when the cost and moderate number of processors are involved . Its main drawback is a bottleneck to the memory when number of processors becomes large and also a single point of failure. To overcome the problems, sometimes several parallel buses are incorporated . The classical example of such machine is the SGI Power Challenge computer with packet data bus.

Table 1: Properties of various types of multiprocessor interconnections

Property	Bus	Crossbar	Multistage
Speed	Low	High	high
Cost	Low	High	moderate
Reliability	Low	High	high
Configurability	High	Low	moderate
Complexity	Low	High	moderate

- crossbar switching networks, which employ a grid of switching elements. The network is nonblocking, since the connection of a processor to a memory bank does not block the connection of any other processor to any other memory bank. In spite of high speed, their use

www.btechsamplepapers.in

is limited, due to nonlinear complexity ($0(p^2)$, p-number of processors) and the cost (cf. Table 1). They are applied mostly in multiprocessor vector computers (like Cray YMP) and in multiprocessors with multilevel interconnections (e.g. HP/Convex Exemplar SPP). One outstanding example is the Fujitsu VPP500 which incorporates 224x224 crossbar switch.

- multistage interconnection networks formulate the most advanced pure solution, which lies between the two extremes (Table 1). A typical example is the omega network, which constists of stages, where p is number of inputs and outputs (usually number of processor and of memory banks). Its complexity is , less than for the crossbar switch. However, in the omega network some memory accesses can be blocked. Although machines of this kind of interconnections offer virtual global memory model of programming and ease of use they are still not much popular. Examples from the past cover BBN Butterfly and IBM RP-3 computers, at present IBM RS6K SP incorporates multistage interconnections with Vulcan switch.
- multilevel interconnection network seems to be a relatively recent development. The idea comes directly from clusters of computers and consists of two or more levels of connections with different aggregated bandwidths. Typical examples are: SGI/Cray Origin2000, IBM RS6K SP with PowerPC604 SMP nodes and HP/Convex Exemplar. This kind of architecture is getting the most interest at present.

Q. 2. (c) What do you mean by control flow and Data flow computers ? State advantage and disadvantage of data flow computing.

Ans. Control flow : Control flow (or alternatively, flow of control) refers to the order in which the individual statements, instructions, or function calls of an imperative or a declarative program are executed or evaluated.

Within an imperative programming language, a control flow statement is a statement whose execution results in a choice being made as to which of two or more paths should be followed. For non-strict functional languages, functions and language constructs exist to achieve the same result, but they are not necessarily called control flow statements.

The kinds of control flow statements supported by different languages vary, but can be categorized by their effect:

- continuation at a different statement (unconditional branch or jump),
- executing a set of statements only if some condition is met (choice - i.e. conditional branch),
- executing a set of statements zero or more times, until some condition is met (i.e. loop - the same as conditional branch),
- executing a set of distant statements, after which the flow of control usually returns (subroutines, coroutines, and continuations),
- stopping the program, preventing any further execution (unconditional halt).

Dataflow : Dataflow is a term used in computing, and may have various shades of meaning. It is closely related to message passing.

www.btechsamplepapers.in

Dataflow programming embodies these principles, with spreadsheets perhaps the most widespread embodiment of dataflow. For example, in a spreadsheet you can specify a cell formula which depends on other cells; then when any of those cells is updated the first cell's value is automatically recalculated. It's possible for one change to initiate a whole sequence of changes, if one cell depends on another cell which depends on yet another cell, and so on.

The dataflow technique is not restricted to recalculating numeric values, as done in spreadsheets. For example, dataflow can be used to redraw a picture in response to mouse movements, or to make a robot turn in response to a change in light level.

One benefit of dataflow is that it can reduce the amount of coupling-related code in a program. For example, without dataflow, if a variable Y depends on a variable X, then whenever X is changed Y must be explicitly recalculated. This means that Y is coupled to X. This means that the update operation must be explicitly contained in the program and eventually checking must be added to avoid cyclical dependencies. Dataflow improves this situation by making the recalculation of Y automatic, thereby eliminating the coupling from X to Y. Dataflow makes implicit a significant amount of computation that must be expressed explicitly in other programming paradigms.

Dataflow is also sometimes referred to as reactive programming.

Properties of dataflow programming languages : Dataflow programming focuses on how things connect, unlike imperative programming, which focuses on how things happen. In imperative programming a program is modeled as a series of operations (thing that "happen"), the flow of data between these operations is of secondary concern to the behavior of the operations themselves. However, dataflow programming models programs as a series of (sometimes interdependent) connections, with the operations between these connections being of secondary importance.

Q. 3. Attempt any two parts of the following :

Q. 3. (a) Explain the Flynn's classification for Computer Architectures based on the nature of the instruction flow executed by the computer with diagram.

Ans. Flynn's taxonomy

	Single Data	SISD	MISD
Multiple Data		SIMD	MIMD

www.btechsamplepapers.in

Classifications : The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

Single Instruction, Single Data stream (SISD) : A sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are the traditional uniprocessor machines like a PC (currently manufactured PCs have multiple processors) or old mainframes.

Single Instruction, Multiple Data streams (SIMD) : A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

Multiple Instruction, Single Data stream (MISD) : Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

Multiple Instruction, Multiple Data streams (MIMD) : Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

Q. 3. (b) What does an array processor mean ? What are the different SIMD computer organizations ?

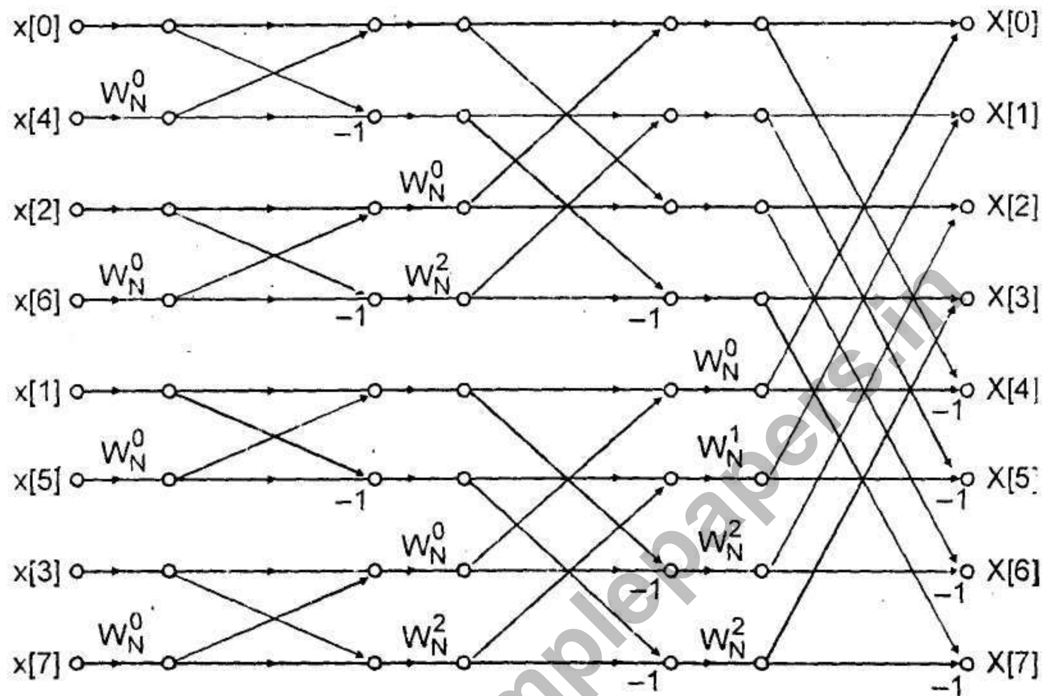
Ans. Vector processor : A vector processor, or **array processor,** is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors. This is in contrast to a scalar processor, whose instructions operate on single data items.

CPUs are able to manipulate one or two pieces of data at a time. For instance, many CPUs have an instruction that essentially says "add A to B and put the result in C". The data for A, B and C could be-in theory at least-encoded directly into the instruction. However things are rarely that simple. In general the data is rarely sent in raw form, and is instead "pointed to" by passing in an address to a memory location that holds the data. Decoding this address and getting the data out of the memory takes some time. As CPU speeds have increased, this memory latency has historically become a large impediment to performance, see Memory wall.

In order to reduce the amount of time this takes, most modern CPUs use a technique known as instruction pipelining in which the instructions pass through several sub-units in turn The first sub-unit reads the address and decodes it, the next "fetches" the values at those addresses, and the next does the math itself. With pipelining the "trick" is to start decoding the next instruction even before the first has left the CPU, in the fashion of an assembly line, so the address decoder is constantly in use. Any particular instruction takes the same amount of time to complete, a time known as the latency, but the CPU can process an entire batch of operations much faster than if it did so one at a time.

Vector processors take this concept one step further. Instead of pipelining just the instructions, they also pipeline the data itself. They are fed instructions that say not just to add A to B, but to add all of the numbers "from here to here" to all of the numbers "from there to there". Instead of constantly having to decode instructions and then fetch the data needed to complete them, it reads a single instruction from memory, and "knows" that the next address will be one larger than the last. This allows for significant savings in decoding time.

www.btechsamplepapers.in



Q. 3. (c) Vectorizing compilers generally detect loops that can be executed on a pipelined vector computer. Are the vectorization algorithms used by vectorizing compilers suitable for MIMD machine parallelization.

Ans. As the vectorizing compiler are used or we can say that vectorizing compilers generally easily detects loops that can be executed on a pipelined vector computer. So we can used them for MIMD machine parallelization.

In MIMD, the multiple instructions are executed upon multiple data streams. Multiple data are used and multiple instruction perform multiple operations for multiple instructions to be executed parallelaly, the data and the instructions should be used as independent as possible. The data should not be dependent on each other.

As vectorizing compiler detects the loop, these compilers easily find the dependency between the data used in machines as it find the dependency, these algorithms are useful in applying parallelism.

Q. 4. Attempt any two parts of the following :

Q. 4. (a) A hierarchical cache main memory subsystem has following specifications :

(i) Cache access time of 50 n sec.

(ii) Main storage access time of 500 n sec.

(iii) 80% of request are for read.

(iv) Hit ratio of 0.9 for read access and for write through scheme is used,

Determine :

(A) Average access time of the system considering only memory read cycle.

www.btechsamplepapers.in

(B) Average access time of the system both for read and write requests.

(C) Hit ratio taking into considerations the write cycle.

(1) 100 n sec **(2) 180 n sec**

(3) 0.72 n sec

Ans. Cache access time of 50 n sec

main storage access time = 500 n sec

80% of requests are for read

Hit ratio of 0.9 for access

→ average accesstime of the system only for read cycle

(50 (0.1) + 500) × 0.8

(5 + 500) × 0.8

(550) × 0.8 = 404 m sec

→ 505 m sec only for both cycle

→ 100 n sec = 0.8

180 n sec = 0.9

0.72 n sec = 0.6

Q. 4. (b) What are the different hazards that occur in instruction pipeline and how these are resolved ?

Ans. Hazard : A hazard is a situation that poses a level of threat to life, health, property, or environment. Most hazards are dormant or potential, with only a theoretical risk of harm; however, once a hazard becomes "active", it can create an emergency situation. A hazard does not exist when it is happening. A hazardous situation that has come to pass is called an incident. Hazard and vulnerability interact together to create risk.

Modes of a hazard: Hazards are sometimes classified into three modes

- **Dormant :** The situation has the potential to be hazardous, but no people, property, or environment is currently affected by this. For instance, a hillside may be unstable, with the potential for a landslide, but there is nothing below or on the hillside that could be affected.
- **Armed :** People, property, or environment are in potential harm's way.
- **Active :** A harmful incident involving the hazard has actually occurred. Often this is referred to not as an "active hazard" but as an accident, emergency, incident, or disaster.

Classifying hazards : By its nature, a hazard involves something that could potentially be harmful to a person's life, health, property, or the environment. One key concept in identifying a hazard is the presence of stored energy that, when released, can cause damage. Stored energy can occur in many forms: chemical, mechanical, thermal, radioactive, electrical, etc. Another class of hazard does not involve release of stored energy, rather it involves the presence of hazardous situations. Examples include confined or limited egress spaces, oxygen-depleted atmospheres, awkward positions, repetitive motions, low-hanging or protruding objects, etc.

There are several methods of classifying a hazard, but most systems use some variation on the factors of "likelihood" of the hazard turning into an incident and the "seriousness" of the incident if it were to occur. (This discussion moved away from hazard to a discussion of risk.)

www.btechsamplepapers.in

A common method is to score both likelihood and seriousness on a numerical scale (with the most likely and most serious scoring highest) and multiplying one by the other in order to reach a comparative score.

Risk = Likelihood of Occurrence x Seriousness if incident occurred : This score can then be used to identify which hazards may need to be mitigated. A low score on likelihood of occurrence may mean that the hazard is dormant, whereas a high score would indicate that it may be an "active" hazard.

An important component of "seriousness if incident occurred" is "serious to whom?" Different populations may be affected differently by accidents. For example, an explosion will have widely differing effects on different populations depending on the distance from the explosion. These effects can range from death from overpressure or shrapnel to inhalation of noxious gases (for people downwind) to being exposed to a loud noise.

Causes of hazards : There are many causes, but they can broadly be classified as below. See the linked articles for comprehensive lists of each type of hazard.

- Natural hazards include anything that is caused by a natural process, and can include obvious hazards such as volcanoes to smaller scale hazards such as loose rocks on a hillside
- Man-made hazards are created by humans, whether long-term (such as global warming) or immediate (like the hazards present at a construction site). These include activity related hazards (such as flying) where cessation of the activity will negate the risk.

Deadly force or retribution is that hazard involving any protective and responsive ready threat of harm or punishment that becomes active in the event of a breach of security, or violation of a boundary or barrier (physical, legal, moral) intended to prevent unauthorized or unsafe access or entry or exposure to a situation, to something, or to someone.

Q. 4. (c) What is meant by Cache-Coherency ? Explain with the help of a suitable example.

Ans. Cache coherence : Cache coherence (also cache coherency) refers to the consistency of data stored in local caches of a shared resource. Cache coherence is a special case of memory coherence.

When clients in a system maintain caches of a common memory resource, problems may arise with inconsistent data. This is particularly true of CPUs in a multiprocessing system. Referring to the "Multiple Caches of Shared Resource" figure, if the top client has a copy of a memory block from a previous read and the bottom client changes that memory block, the top client could be left with an invalid cache of memory without any notification of the change. Cache coherence is intended to manage such conflicts and maintain consistency between cache and memory.

Definition

Coherence defines the behavior of reads and writes to the same memory location. The coherence of caches is obtained if the following conditions are met:

1. A read made by a processor P to a location X that follows a write by the same processor P to X, with no writes of X by another processor occurring between the write and the read instructions made

www.btechsamplepapers.in

by P, X must always return the value written by P. This condition is related with the program order preservation, and this must be achieved even in monoprocessed architectures.

2. A read made by a processor P1 to location X that follows a write by another processor P2 to X must return the written value made by P2 if no other writes to X made by any processor occur between the two accesses. This condition defines the concept of coherent view of memory. If processors can read the same old value after the write made by P2, we can say that the memory is incoherent.

3. Writes to the same location must be sequenced. In other words, if location X received two different values A and B, in this order, by any two processors, the processors can never read location X as B and then read it as A. The location X must be seen with values A and B in that order.

These conditions are defined supposing that the read and write operations are made instantaneously. However, this doesn't happen in computer hardware given memory latency and other aspects of the architecture. A write by processor P1 may not be seen by a read from processor P2 if the read is made within a very small time after the write has been made. The memory consistency model defines when a written value must be seen by a following read instruction made by the other processors.

Cache coherence mechanisms

- **Directory-based coherence :** In a directory-based system, the data being shared is placed in a common directory that maintains the coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed the directory either updates or invalidates the other caches with that entry.
- **Snooping** is the process where the individual caches monitor address lines for accesses to memory locations that they have cached. When a write operation is observed to a location that a cache has a copy of, the cache controller invalidates its own copy of the snooped memory location.
- **Snarfing** is where a cache controller watches both address and data in an attempt to update its own copy of a memory location when a second master modifies a location in main memory. When a write operation is observed to a location that a cache has a copy of, the cache controller updates its own copy of the snarfed memory location with the new data.

Coherency protocol : A coherency protocol is a protocol which maintains the consistency between all the caches in a system of distributed shared memory; the protocol maintains memory coherence according to a specified consistency model. Most of the cache protocols in multiprocessors are supporting sequential consistency model, while in software distributed shared memory more popular are models supporting release consistency or weak consistency.

Transitions between states in any specific implementation of these protocols may vary. For example, an implementation may choose different update and invalidation transitions such as update-on-read, update-on-write, invalidate-on-read, or invalidate-on-write. The choice of transition may affect the amount of inter-cache traffic, which in turn may affect the amount of cache

www.btechsamplepapers.in

bandwidth available for actual work. This should be taken into consideration in the design of distributed software that could cause strong contention between the caches of multiple processors.

Various models and protocols have been devised for maintaining cache coherence, such as :

- MSI protocol
- MESI protocol aka Illinois protocol
- MOSI protocol
- MOESI protocol

Q. 5. Attempt any two parts of the following :

Q. 5. (a) Consider the following reservation table for a four stage pipeline with a clock cycle $i = 20$ ns.

	1	2	3	4
S1	×			×
S2		×		
S3			×	

(i) What are the forbidden latencies and initial collision vector ?

(ii)Draw the state transition diagram for scheduling the pipeline.

(iii) List all the simple cycle and greedy cycle.

(iv) Determine the optimal constant latency and minimal average latency (MAL).

(v) Determine the throughput of this pipeline. Lower bound on the MAL for this pipeline.

Ans.

	1	2	3	4
S1	×			×
S2		×		
S3			×	

(i) Forbidden latencier are 0, 3

initial collision vector is (1 0 0 1)

(ii) State diagram

State 1 : 1001

Reacher state 2 (1011) after 1 cycle

Reacher state 3 (1101) after 2 cycles

Reacher state 1 (1001) after 4 or more cycles.

State 2 : 1011

Reacher state 4 (1111) after 1 cycle

Reacher state 1 (1001) after 4 or more cycle.

www.btechsamplepapers.in

State 3 : 1101

Reacher state 3 (1101) after 2 cycle

Reacher state 1 (1001) after 4 or more cycles

State 4 : 1111

Reacher state 1 (1001) after 4 or more cycles

Fair state

1st state → 1001

State 2 → 1011

State 3 → 1101

State 4 → 1111

(iii) Greedy cycle is $(1, 1, 4)^x$

(iv) The value of minimum average latency for your data MAL = 2

(v) Throughtput = 0.5 instruction per cycle.

Q. 5. (b) What are the properties of the Vector Processors ? Explain each component of Vector-Register Processors with diagram.

Ans. Properties of Vector Processors :

- Each result independent of previous result

(i) long pipeline, compiler ensures no dependencies

(ii) high clock rate

- Vector instructions access memory with known pattern

(i) highly interleaved memory

(ii) amortize memory latency of over " 64 elements no (data) caches required! (Do use instruction cache)

- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (" loop)

(i) fewer instruction fetches

- Single vector instruction implies lots of work (loop)

(i) Fewer instruction fetches

- Each result independent of previous result

(i) Multiple operations can be executed in parallel

(ii) Simpler design, high clock rate

(iii) Compiler (programmer) ensures no dependencies

- Reduces branches and branch problems in pipelines
- Vector instructions access memory with known pattern

(i) Effective prefetching

(ii) Amortize memory latency of over large number of elements

(iii) Can exploit a high bandwidth memory system - No (data) caches required!

Q. 5. (c) Discuss the superscalar and superpipelined processing. Also estimate the performance of superpipelined superscalar processor of degree (*m*, *n*).

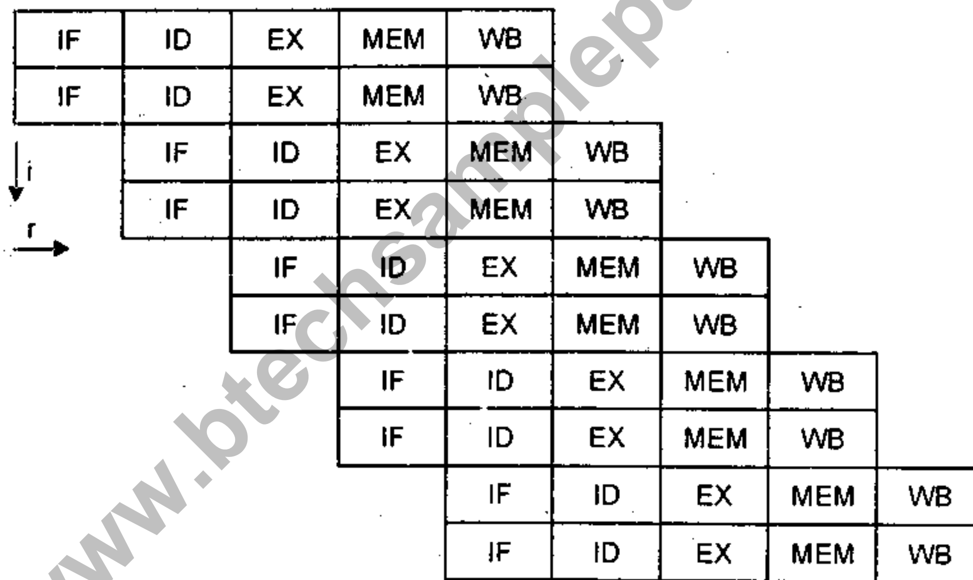
www.btechsamplepapers.in

Ans. A superscalar CPU architecture implements a form of parallelism called instruction level parallelism within a single processor. It therefore allows faster CPU throughput than would otherwise be possible at a given clock rate. A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier.

While a superscalar CPU is typically also pipelined, pipelining and superscalar architecture are considered different performance enhancement techniques.

The superscalar technique is traditionally associated with several identifying characteristics (within a given CPU core):

- Instructions are issued from a sequential instruction stream
- CPU hardware dynamically checks for data dependencies between instructions at run time (versus software checking at compile time)
- The CPU accepts multiple instructions per clock cycle



www.btechsamplepapers.in

Simple superscalar pipeline. By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed.

Limitations : Available performance improvement from superscalar techniques is limited by three key areas:

1. The degree of intrinsic parallelism in the instruction stream, i.e. limited amount of instruction-level parallelism.

2. The complexity and time cost of the dispatcher and associated dependency checking logic.

3. The branch instruction processing.

Existing binary executable programs have varying degrees of intrinsic parallelism. In some cases instructions are not dependent on each other and can be executed simultaneously. In other cases they are inter-dependent: one instruction impacts either resources or results of the other. The

instructions a = b + c; d = e + f can be run in parallel because none of the results depend on other calculations. However, the instructions a = b + c; b = e + f might not be runnable in parallel, depending on the order in which the instructions complete while they move through the units.

When the number of simultaneously issued instructions increases, the cost of dependency checking increases extremely rapidly. This is exacerbated by the need to check dependencies at run time and at the CPU's clock rate. This cost includes additional logic gates required to implement the checks, and time delays through those gates. Research shows the gate cost in some cases may be nk gates, and the delay cost k2logn, where n is the number of instructions in the processor's instruction set, and k is the number of simultaneously dispatched instructions. In mathematics, this is called a combinatoric problem involving permutations.

Even though the instruction stream may contain no inter-instruction dependencies, a superscalar CPU must nonetheless check for that possibility, since there is no assurance otherwise and failure to detect a dependency would produce incorrect results.

No matter how advanced the semiconductor process or how fast the switching speed, this places a practical limit on how many instructions can be simultaneously dispatched. While process advances will allow ever greater numbers of functional units (e.g., ALUs), the burden of checking instruction dependencies grows so rapidly that the achievable superscalar dispatch limit is fairly small. -- likely on the order of five to six simultaneously dispatched instructions.

However even given infinitely fast dependency checking logic on an otherwise conventional superscalar CPU, if the instruction stream itself has many dependencies, this would also limit the possible speedup. Thus the degree of intrinsic parallelism in the code stream forms a second limitation.

www.btechsamplepapers.in