B.TECH.

SEVENTH SEMESTER EXAMINATION, 2006-2007

COMPUTER ARCHITECTURE

Time – 3 hours **Total Marks – 100**

Note: (i) Attempt *ALL* questions.
(ii) All questions carry equal marks.
(iii) Be precise in your answer.

Q. 1. Attempt any four parts of the following– **(5×4=20)**

Q.(a). Discuss various classifications of parallel processing mechanisms in uniprocessor computers.

Ans. Charecteristics of Parallel Processing Mechanism in Uniprocessor Computers

Parallel Processing refers to the concept of speeding-up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across n processors might execute n times faster than it would using a single processor.

Despite rough times for some high-end parallel processing hardware companies, parallel processing is not dead, but is moving in some new directions. Traditionally, multiple processors were provided within a specially designed "parallel computer"; along these lines, Linux now supports SMP Pentium systems in which multiple processors share a single memory and bus interface within a single computer. It is also possible for a group of machines each running Linux to be interconnected by a network to form a parallel-processing cluster.

Parallel processing makes a program run faster because there are more engines (CPUs) running it. In practice, it is often difficult to divide a program in such a way that separate CPUs can execute different portions without interfering with each other.

Q.1.(b). Comment on balancing of system bandwidth.

Ans. Balancing of System Bandwidth

The ratio of cpu speed to memory speed in current high-performance computers is growing rapidly, with significant implications for the design and implementation of algorithms in scientific computing. I present the results of a broad survey of memory bandwidth and machine balance for a large variety current computers, including uniprocessors, vector processors, shared-memory systems, and distributed-memory systems.

It has been estimated that the cpu speed of the fastest available microprocessors is increasing at approximately 80% per year, while the speed of memory devices has been growing at only about 7% per year. The ratio of the cpu to memory performance is thus also growing exponentially,

suggesting the need for fundamental re-thinking of either the design or computer systems or the algorithms that scientific users employ on them. For example, 10 years ago, floating-point operations were considered quite expensive, often costing 10 times as much as an uncached memory reference. Today the situation is dramatically reversed, with the fastest current processors able to perform 100 or more floating-point operations in the time required to service a single cache miss Because of this fundamental change in the balance of the underlying technology, this report presents a survey of the memory bandwidth and machine balance on a variety of currently available machines.

Q.1.(c). Differentiate between parallel and distributed processing.

Ans. Parallel Processing Vs Distributed Processing:

In Parallel Processing, simultaneous use of more than one CPU to execute a program. Ideally, parallel processing makes a program run faster because there are more engines (CPUs) running it. In practice, it is often difficult to divide a program in such a way that separate CPUs can execute different portions without interfering with each other. Most computers have just one CPU, but some models have several. There are even computers with thousands of CPUs. With single-CPU computers, it is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software. Note that parallel processing differs from multitasking, in which a single CPU executes several programs at once. Parallel processing is also called parallel computing.

In Distributed Processing a variety of computer systems that use more than one computer, or processor, to run an application. This includes parallel processing, in which a single computer uses more than one CPU to execute programs. More often, however, distributed processing refers to local-area networks (LANs) designed so that a single program can run simultaneously at various sites. Most distributed processing systems contain sophisticated software that detects idle CPUs on the network and parcels out programs to utilize them.

Q.1.(d). Why array computers are termed as parallel computers?

Ans. Array Computers are turned as Parallel Computers: array computer is formed by individual tiles, each typically containing a display surface, a processor, memory, and communication devices for providing communications with adjoining tiles or the support structure upon which the tiles are placed. Array computers on a single chip is demonstrated with two example realizations. The key idea in the method is that an array computer can be built using reusable, relatively simple building blocks. All necessary memory is distributed among processing units. The number of processing units in the array computer is selected based on speed requirements, yielding a low power, small die area realization.

At least two computer tiles disposed adjacent to each other, each comprising: a display surface having elements thereon for displaying images, the display surface being affixed to a substrate having a predefined shape; a processor element coupled to the display, coupled to a memory element,

and coupled to an information transducer element, each of the processor element and the memory element being mounted on the substrate to enable the processor element to communicate with the display surface to control the presentation of information thereon, wherein the transducer element is disposed to permit communications from the processor element to an external receiver, the transducer providing information to the external receiver through a region of the layer other than the display surface; and at least one strain gauge disposed on the substrate in communication with the processor element, wherein the transducer of each of the computer tiles is disposed in proximity to an edge of the substrate to enable information from the first computer tile to be communicated to the second computer tile.

Q.1.(e). Discuss static and dynamic dataflow architecture models in short.

Ans. Static and Dynamic dataflow architecture model

Static dataflow architecture : For a dataflow multiprocessor is the algorithm by which the nodes of a program graph are allocated for execution to its processors. In the case of the static type of architecture one must consider pipelining as well as spatial concurrency. it uses a graph partitioning scheme to aid in the design of such algorithms and selection of the associated interconnection topology. The scheme first refines the usual dataflow program graph and then partitions it into "trees." Our study shows that the hypercube interconnection accommodates these trees in a way that allows as to develop an algorithm that places producer nodes (of a dataflow graph) nears their consumers to keep the message path very short.

Dynamic data flow architecture : it is hybrid architecture model which employs conventional architecture techniques to achieve fast pipelined operation, while exploiting finegrain parallelism by data-driven instruction scheduling. A mechanism for supporting concurrent operations of multiple instruction threads on the hybrid architecture model is presented and a compiling paradigm for dataflow software pipelining which efficiently exploits loop parallelism

Q.1.(f). Elaborate Flynn's classification.

Ans. Flynn's classification

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 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)

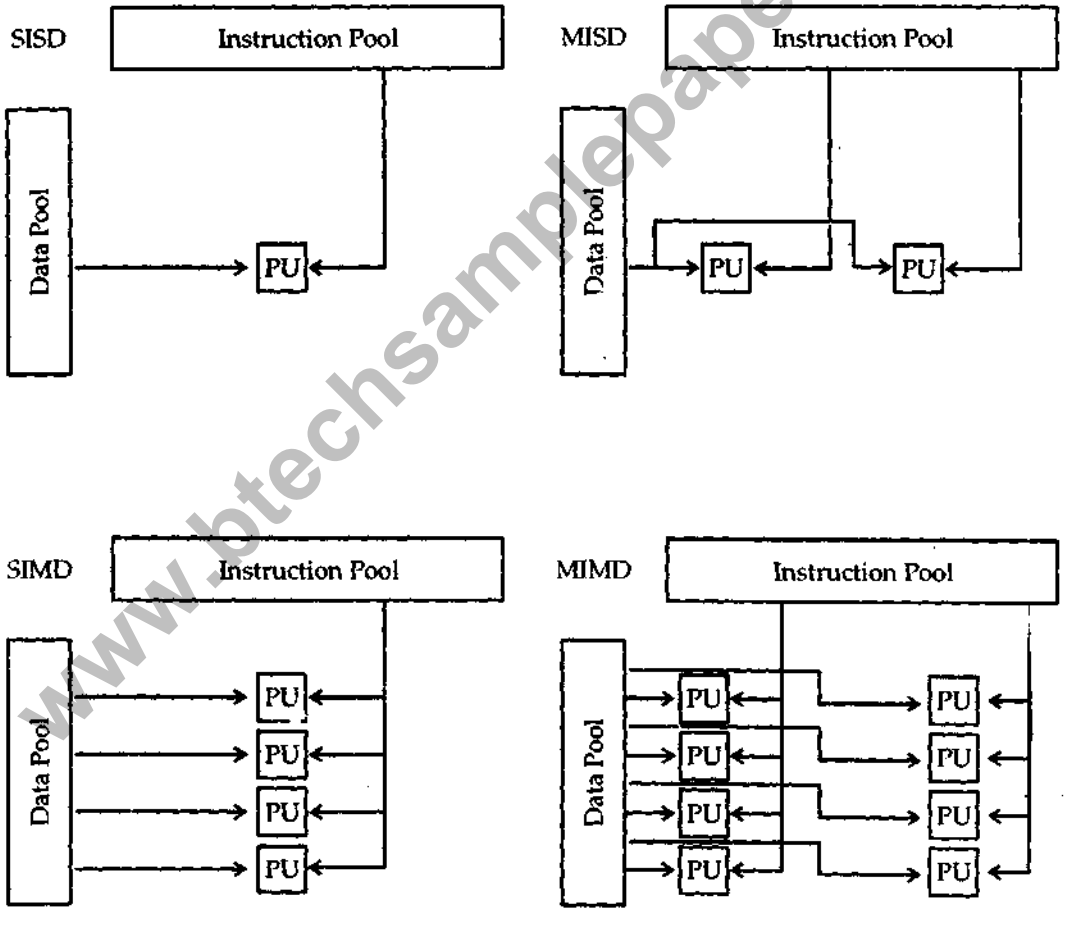
Unusual due to the fact that multiple instruction streams generally require multiple data streams

to be effective. However, this type is used when it comes to redundant parallelism, as for example on airplanes that need to have several backup systems in case one fails. Some theoretical computer architectures have also been proposed which make use of MISD, but none have entered mass production.

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.

These four architectures are shown below where each "PU" is a processing unit:

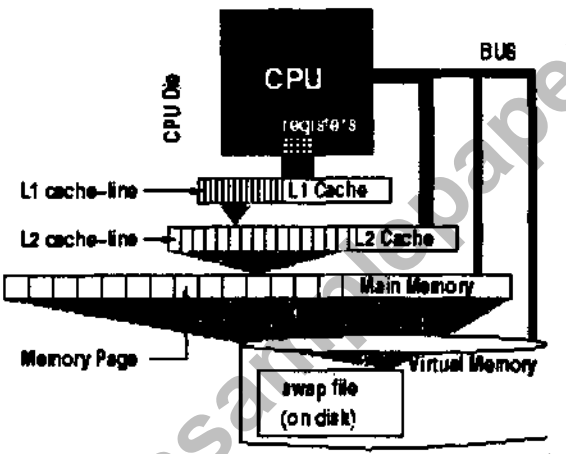


Q. 2. Attempt any fourn parts of the following— **(5×4=40)**

Q.(a). Explain hierarchical memory system. What do you mean by hit ratio and failure ratio?

Ans. Hierarchical Memory Model

With the fast development in CPU speeds, the relative performance of the DRAM memory chips (which develop much slower) rapidly decreases. To accelerate access to the data additional cache memories made of SRAM have been introduced. First they were placed on the motherboard, now they often are a part of a CPU chip. A hierarchical memory scheme found in the modern CPUs.



Hierarchical Memory System

The memory performance in this model may be influenced by multiple hardware parameters—

Capacity

Sizes of memory chips at different levels can be used as a tuning information for many algorithms

Cache line size and number

Caches can differ not only in size, but also in partitioning strategy. Each cache-memory is divided into cache lines. Their size and count give total cache size. These values can have great influence on other factors - big cache lines usually increase memory latency, yet big number of cache lines may result in less cache misses.

Latency

In general, latency is an amount of time needed to transfer a single byte from sender to receiver. For our purposes, the most important are memory latencies. Since the memory system in modern CPUs is hierarchical, we can define various latencies. L_1 latency (l_{L1}) is the time needed by CPU to access data from L1 cache. When CPU tries to access data which is not in L1, L1 miss occurs. The data has to be transferred from L2 to L1 - the time consumed is called L2 latency (l_{L2}). If the data is not in L2 then an (L2 miss), it has to be transferred from main memory, adding memory latency

(l_{Mem}). Therefore, access to data which is not in any cache costs $l_{L1} + l_{L2} + l_{Mem}$. One should notice thatmemory is not transferred in single bytes, but in full cache-lines.

Cache associativity

Modern caches usually keep a few (e.g. 4) possible positions for each memory address. When a new cache line is fetched, its hash value is calculated by taking a few bits from the physical address, and an LRU (least-recently used) strategy is used to find final position in the cache. This strategy can be exploited by algorithms, for example to increase the probability of reusing cached data.

Address translation

An important phase of memory access is translating an address from application's virtual memory area into physical addresses in computer's main memory. To speed up this mechanism modern CPUs contain a translation lookahead buffer (TLB) - a kind of cache for recently used translations. Usually it consist of 64 entries. When the logical address is not found in the TLB (TLB miss), an address translation has to be performed. It can be either done in hardware (e.g. Athlon, Pentium 4) or in software via operating system (e.g. Origin2000, Sun Ultra). The former case is usually fast, yet the software approach can even exceed 50 CPU cycles.

Throughput

In general, throughput is as amount of data transferred from sender to receiver in a specified period of time. For our purposes, we concentrate on the main memory throughput, additionally checking this factor for the caches. Throughput strongly depends on the latency. However, in many cases it can provide better performance than expected by simply dividing cache line size by latency, because many cache lines can be transferred in parallel.

Note that the time-related values may be measured using either real-time units (seconds) or relatively to the CPU frequency (cycles). We emphasize the importance of the latter, to better realize the imbalance in the various hardware parts development. If we look at the performance values in seconds, we see they are improving. However, measuring them in cycles often shows a dramatical decrease, leading to wasting a lot of CPU power.

Part-II

Hit ratio is a request to a memory for a file. When a file is uploaded from a memory the number of "Hits" is equal to the number of blocks requested, therefore one page load does not always equal one hit because often pages are made up of other images and other files which stack up the number of hits counted.

Failure ratio is the frequency with which an system or component fails, expressed for example in failures per second. Failure rate is usually time dependent, and an intuitive corollary is that both rates change over time versus the expected life cycle of a system.

Q.2.(b). What is m-way interleaving? Discuss different types of memory interleaving.

Ans. m – way Memory Interleaving

Interleaving is a way to arrange data in a non-contiguous way in order to increase performance.Historically, interleaving has also been used in ordering block storage on hard disks. Interleaving is also used for multidimensional data structures.

Main memory divided into two or more sections. The CPU can access alternate sections immediately, without waiting for memory to catch up (through wait states). Interleaved memory is one technique for compensating for the relatively slow speed of dynamic RAM (DRAM). Other techniques include page-mode memory and memory caches.

Types of Memory Interleaving

i) High-Order Interleaving

Arguably the most "natural" arrangement would be to use bus lines A26-A27 as the module determiner. In other words, we would feed these two lines into a 2-to-4 decoder, the outputs of which would be connected to the Chip Select pins of the four memory modules. If we were to do this, the physical placement of our system addresses would be as follows:

address	module
0-64M	0
64M-128M	1
128M-192M	2
192M-256M	3

Note that this means consecutive addresses are stored within the same module, except at the boundary. The above arrangement is called high-order interleaving, because it uses the high-order, i.e. most significant, bits of the address to determine which module the word is stored in.

ii) Low-Order Interleaving

An alternative would be to use the low bits for that purpose. In our example here, for instance, this would entail feeding bus lines A0-A1 into the decoder, with bus lines A2-A27 being tied to the address pins of the memory modules. This would mean the following storage pattern:

address	module
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	3
8	0
9	1

etc. etc. In other words, consecutive addresses are stored in consecutive modules, with the understanding that this is mod 4, i.e. we wrap back to M0 after M3.

Q.2.(c). Differentiate between super scalar and vector processors.

Ans. SuperScalar and vector processors

Superscalar CPU architecture implements a form of parallelism called Instruction-level parallelism within a single processor. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate. A superscalar architecture executes more than one instruction during a single pipeline stage by pre-fetching multiple instructions and simultaneously dispatching them to redundant functional units on the processor.

A superscalar processor usually sustains an execution rate in excess of one instruction per machine cycle. But merely processing multiple instructions concurrently does not make an architecture superscalar, since pipelined, multiprocessor or multi-core architectures also achieve that, but with different methods.

In a superscalar CPU the dispatcher reads instructions from memory and decides which ones can be run in parallel, dispatching them to redundant functional units contained inside a single CPU. Therefore a superscalar processor can be envisioned having multiple parallel pipelines, each of which is processing instructions simultaneously from a single instruction thread.

Vector processor, or array processor, is a CPU design that is able to run mathematical operations on multiple data elements simultaneously. This is in contrast to a scalar processor which handles one element at a time. The vast majority of CPUs are scalar (or close to it). Vector processors were common in the scientific computing area, where they formed the basis of most supercomputers through the 1980s and into the 1990s, but general increases in performance and processor design saw the near disappearance of the vector processor as a general-purpose CPU.

Today most commodity CPU designs include some vector processing instructions, typically known as SIMD (Single Instruction, Multiple Data), common examples include SSE and AltiVec. Modern video game consoles and consumer computer-graphics hardware rely heavily on vector processing in their architecture.

Q.2.(d). Differentiate between DMA and I/O channel.

Ans. DMA Vs I/o Channel

Direct memory access (DMA) is a feature of modern computers that allows certain hardware subsystems within the computer to access system memory for reading and/or writing independently of the central processing unit. Many hardware systems use DMA including disk drive controllers, graphics cards, network cards, and sound cards. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel.

Without DMA, using programmed input/output (PIO) mode, the CPU typically has to be occupied for the entire time it's performing a transfer. With DMA, the CPU would initiate the transfer, do

other operations while the transfer is in progress, and receive an interrupt from the DMA controller once the operation has been done. This is especially useful in real-time computing applications where not stalling behind concurrent operations is critical.

A typical usage of DMA is copying a block of memory from system RAM to or from a buffer on the device. Such an operation does not stall the processor, which as a result can be scheduled to perform other tasks. DMA transfers are essential to high performance embedded systems. It is also essential in providing so-called zero-copy implementations of peripheral device drivers as well as functionalities such as network packet routing, audio playback and streaming video.

The I/O channel is a line of communication between the input/output bus and/or memory to the CPU and/or computer peripherals. channel I/O is a generic term that refers to an advanced, high-performance input/output architecture that is implemented in various forms on a number of computer architectures, especially on mainframe computers. In the past they were generally implemented with a custom processor, known alternately as peripheral processors, I/O processors, I/O controllers or DMA controllers.

Many input/output tasks can be fairly complex and require logic to be applied to the data to convert formats and other similar duties. In these situations the computer's CPU would normally be asked to handle the logic, but due to the fact that the I/O devices are very slow, the CPU would end up spending a huge amount of time (in computer terms) sitting idle waiting for the data from the device. A channel I/O architecture avoids this problem by using a low-cost processor with enough logic and memory onboard to handle these sorts of tasks. They are typically not powerful or flexible enough to be used as a computer on their own, and are actually a form of co-processor. The CPU sends small programs to the controller to handle an I/O job, which the channel controller can then complete without any help from the CPU. When it is complete, or there is an error, the channel controller communicates with the CPU using a selection of interrupts. Since the channel controller has direct access to the main memory of the computer, they are also often referred to as DMA Controllers (where DMA means direct memory access), but that term is somewhat more loose in definition and is often applied to non-programmable devices as well.

Q.2.(e). How memory contention problem can be reduced in multiprocessor systems?

Ans. Memory contention problem reduced in multiprocessor system

Memory interleaving considerably increases memory bandwidth in vector processor systems. The concurrent operation of the processors can produce memory bank conflicts and hence alter the memory bandwidth. Total or steady state performance for vector operations in a memory system is studied. Many methods of resolving memory bank conflicts are proposed and compared. Some nonintuitive results are obtained on how conflicts depend on the size of the architecture, the number, the stride and the length of the vectors, the register length assigned by each processor to vector components

Content-addressable memory (CAM) is a special type of computer memory used in certain very high speed searching applications. It is also known as associative memory, associative storage, or associative array, although the last term is more often used for a programming data structure.

Q.2.(f). How multi cache coherence problem is handled?

Ans. Multi cache coherence

Memory architecture with a multiple cache coherency includes at least one processor with a storage area in communication with a cache memory. A main bus transmits and receives data to and from the cache memory and the processor. A coherency control in communication with the cache memory and the processor is configured to determine an existence or location of data in the cache memory or the storage area in response to a data request from the main bus. The coherency control dispatches an existence or location result to the main bus.

The performance of a memory-coherent single-stage shuffle-exchange network based multiprocessor. Each network node contains a small global cache memory and a routing directory. The routing directory is used to implement an adaptive single copy coherence scheme which regulates the dynamic movement of the shared data blocks according to the access patterns exhibited in memory reference streams.

Q. 3. Attempt any two parts of the following— **(10×2=20)**

Q.(a). What are pipeline hazards? Elaborate on various types of hazards and indicate how each is controlled.

Ans. Pipeline Hazards

A pipeline hazard refers to a situation in which a correct program ceases to work correctly due to implementing the processor with a pipeline. There are three fundamental types of hazard: data hazards, branch hazards, and structural hazards. Data hazards can be further divided into Write After Read, Write After Write, and Read After Write hazards.

Types of Hazards:

i) Structural Hazards

These occur when a single piece of hardware is used in more than one stage of the pipeline, so it's possible for two instructions to need it at the same time.

So, for instance, suppose we'd only used a single memory unit instead of separate instruction memory and data memories. A simple (non-pipelined) implementation would work equally well with either approach, but in a pipelined implementation we'd run into trouble any time we wanted to fetch an instruction at the same time a lw or sw was reading or writing its data.

In effect, the pipeline design we're starting from has anticipated and resolved this hazard by adding extra hardware.

Interestingly, the earlier editions of our text used a simple implementation with only a single memory, and separated it into an instruction memory and a data memory when they introduced

pipelining. This edition starts right off with the two memories.

Also, the first Sparc implementations (remember, Sparc is almost exactly the RISC machine defined by one of the authors) did have exactly this hazard, with the result that load instructions took an extra cycle and store instructions took two extra cycles.

ii) Data Hazards

This is when reads and writes of data occur in a different order in the pipeline than in the program code. There are three different types of data hazard (named according to the order of operations that must be maintained):

RAW

A Read After Write hazard occurs when, in the code as written, one instruction reads a location after an earlier instruction writes new data to it, but in the pipeline the write occurs after the read (so the instruction doing the read gets stale data).

WAR

A Write After Read hazard is the reverse of a RAW: in the code a write occurs after a read, but the pipeline causes write to happen first.

WAW

A Write After Write hazard is a situation in which two writes occur out of order. We normally only consider it a WAW hazard when there is no read in between; if there is, then we have a RAW and/or WAR hazard to resolve, and by the time we've gotten that straightened out the WAW has likely taken care of itself.

(The text defines data hazards, but doesn't mention the further subdivision into RAW, WAR, and WAW. Their graduate level text mentions those)

iii) Control Hazards

This is when a decision needs to be made, but the information needed to make the decision is not available yet. A Control Hazard is actually the same thing as a RAW data hazard (see above), but is considered separately because different techniques can be employed to resolve it - in effect, we'll make it less important by trying to make good guesses as to what the decision is going to be.

Q.3.(b). Discuss various advanced pipelined techniques.

Ans. Advanced Pipelined techniques

1. Dynamic Scheduling : where the hardware rearranges the instruction execution to reduce the stalls. Dynamic scheduling offers several advantages:

It enables handling some cases when dependencies are unknown at compile time (e.g., because they may involve a memory reference);

It simplifies the compiler;

It allows code that was compiled with one pipeline in mind to run efficiently on a different pipeline.

These advantages are gained at a cost of a significant increase in hardware complexity

2. **Loop Unrolling:**To keep a pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline. To avoid stalls, a dependent instruction must be separated from the source instruction by a distance in clock cycles equal to the pipeline latency of that source instruction.

3. **Software Pipelining:** software pipelining is a technique used to optimize loops, in a manner that parallels hardware pipelining. Software pipelining is a type of out-of-order execution, except that the reordering is done by a compiler (or in the case of hand written assembly code, by the programmer) instead of the processor.

4. **Dynamic Branch Prediction Units:**dynamically predicting the outcome and the tar address of a multiple-target branch instruction, where the multiple-target branch instruction contains at least two potential target addresses, not including the fall through address. In addition, this method and apparatus can also be used to predict multiple single-target branches simultaneously. The apparatus stores information indicating the outcome of previous executions and predictions of the multiple-target branch instruction in a branch prediction table. In addition, multiple target addresses (at least two) are associated with the multiple-target branch instruction. Using the information indicating the outcome of the previous execution of the multiple-target branch instruction, the apparatus predicts the outcome of a next execution of the multiple-target branch instruction, and predicts which, if any, of the target addresses associated with the multiple-target branch instruction.

5. **Register Renaming:** Register renaming refers to a technique used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations.

Register renaming is a technique used to allow multiple execution paths without conflicts between different execution units trying to use the same registers. Instead of just one set of registers being used, multiple sets are put into the processor. This allows different execution units to work simultaneously without unnecessary pipeline stalls.

6. **Superscalar Processors:** superscalar CPU architecture implements a form of parallelism called Instruction-level parallelism within a single processor. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate. A superscalar architecture executes more than one instruction during a single pipeline stage by pre-fetching multiple instructions and simultaneously dispatching them to redundant functional units on the processor.

7. **VLIW (Very Large Instruction Word) Processors:** Very Long Instruction Word (VLIW) is one particular style of processor design that tries to achieve high levels of instruction level parallelism by executing long instruction words composed of multiple operations. The long instruction word called a MultiOp consists of multiple arithmetic, logic and control operations each of which would probably be an individual operation on a simple RISC processor. The VLIW processor concurrently executes the set of operations within a MultiOp thereby achieving instruction level parallelism.

o Generalized from horizontal microprogramming and superscalar processing

o Multiple functional units use one register file

o Each part of an instruction controls different functional units

o Parallelism determined at compile-time (static scheduling), so success heavily depends on compiler

8.EPIC (Explicitly Parallel Instruction Computers): Explicitly Parallel Instruction Computing (EPIC) is a computing paradigm that began to be researched in the early 1980s resulting in a U.S. patent 4,847,755 (Gordon Morrison, et. al).[1] This paradigm is also called Independence architectures. It was used by Intel and HP in the development of Intel's IA-64 architecture, and has been implemented in Intel's Itanium and Itanium 2 line of server processors. The goal of EPIC was to increase the ability of microprocessors to execute software instructions in parallel, by using the compiler, rather than complex on-die circuitry, to identify and leverage opportunities for parallel execution. This would allow performance to be scaled more rapidly in future processor designs, without resorting to ever-higher clock frequencies, which have since become problematic due to associated power and cooling issues.

The EPIC architecture also includes a grab-bag of architectural concepts to increase ILP:

• Predicated execution is used to decrease the occurrence of branches and to increase the speculative execution of instructions. In this feature, branch conditions are converted to predicate registers which are used to kill results of executed instructions from the side of the branch which is not taken.

• Delayed exceptions (using a Not-A-Thing bit within the general purpose registers) also allow more speculative execution past possible exceptions.

• Very large architectural register files avoid the need for register renaming.

• Multi-way branch instructions

Q. 4. Attempt any two parts of the following— **(10×2=20)**

Q.(a). Write down parallel algorithms for SIMD matrix multiplication. Compare time complexities of SISD and SIMD matrix multiplication algcrithms.

Ans. Parallel algorithm for SIMD Matrix multiplication

This algorithm should be visualized as an extension of the familiar matrix vector multiplication formula

$$v_i = \sum_k A_{ik} x_k$$

Since the instruction set is repeated over all processors, but the data is distributed independently amongst them, we have a standard form of SIMD. The efficiency overhead amounts to distributing and later collecting the data accuratley and is thus minimal in this case. Thus the speedup is significant and in line with the number of processors.

In standard format then, we need to find the new matrix C:

$$C_{ij} = \sum_{k} A_{ik} \cdot B_{jk}$$

And the SIMD parallel algorithm is as follows:

note: (0<=j<=n-1) implies an operation using all processors.

For i:= 0 step 1 until N-1

// Find the i^{th} row of C

begin

C[i,j] := 0; (0<=j<=n-1)

For k:= 0 step 1 until N-1

// Find and add the k^{th} product term over all columns

C[i,j]:= C[i,j] + A[i,k] * B[k,j]; (0<=j<=n-1)

end;

Q.4.(b). In context of language features to exploit parallelism by way of concurrency, define the following terms—

-Join

-FORK

-Cobegin-end

-Fair Scheduling

-Crect

Ans. JOIN and FORK—It is used to denote concurrency is to use FORK and JOIN Statements. Fork spawns a new process and join waits for a previously created process to terminate.

Generally, Fork operation may be specified in three ways—

FORK A;

FORK A, J;

and FORK A, J, N.

* The execution of FORK A statement initiates another process at address A and continues the current process.

* The execution of the FORK A, J statement causes the same action as FORK A and also increments a counter at address J. FORK A, J, N causes the same action of FORK A and set the counter at address J to N.

. In all usages of the FORK statements, the corresponding JOIN statement is expressed as JOIN J. The execution of this statement decrements the counter at J by one.

It the result is 0, the process at address J+1 is initiated, otherwise the processor executing the JOIN statement is released. Hence all process execute join terminals, except the very last one.

Cobegin-end—It is an equivalent extension of the FORK-JOIN concept is the block-structured

language originally proposed by Dijkstra. In this case, each process in a set of n processes S_1, S_2 ..... S_n can be executed concurrently by using the following cobegin-coend constructs—

```
begin
S_d;
Cobegin  S_1; S_2; ..... S_n; coend
  S_n+1;
  end
```

To cobegin declares explicitly the parts of a program that may execute concurrently. This makes it possible to distinguish between shared and local variables, which in turn makes clear from the program text the potential source of interference.

* The block of statements between the cobegin coend are executed concurrently only after the execution of statement so statement S_{n+1} is executes any after all executions of the statements S_1, S_2 ..... S_n have been terminate.

Fair Scheduling—A Process attempting to enter the critical section will eventually do so in a finite time.

The mutually exclusive access to a set of shared variables can be accomplished by a number of constructs. An examples is the MUTEXBEGIN and MUTEXEND constructs. Each constructs alone does not enable a programmer to indicate whether a variable v should be private to a single process or shared by several processes. A compiler must recognize and guard any process interaction involving a shared variable v.

Crect—A critical section may be defined by C set do S. The definition associated a statement S with a common variable v and indicates that the statements S should have exclusive access to v. Critical sections referring to the statement S should have exclusive access to v. Critical sections referring to the same variable v exclude one another in time. By explicitly associating a critical section with the shared variable, the programmer informs the complier of the sharing of this variable among concurrent processes, which is a deliberate exception to the rule of disjointness. At the same time, the compiler can check that a shared variable is used only inside critical sections and can generate code that implements mutual exclusion correctly. Critical sections referring to differing variables can be executed in parallel as show in following example.

```
var v :  shared V;
var w   :  shared W;
cobegin
csect v do P;
cosect w do Q;
coend
```

Q.4.(c). Discuss the functional architecture of SIMD multiprocessor systems.

Ans. Functional Architecture of SIMD Multiprocessor System:

SIMD (Single Instruction, Multiple Data) is a technique employed to achieve data level parallelism, like with vector processor. First made popular in large-scale supercomputers (contrary to MIMD parallelization), smaller-scale SIMD operations have now become widespread in personal computer hardware. Today the term is associated almost entirely with these smaller units.

Application that may take advantage of SIMD is one where the same value is being added (or subtracted) to a large number of data points, a common operation in many multimedia applications. One example would be changing the brightness of an image. Each pixel of an image consists of three values for the brightness of the red, green and blue portions of the color. To change the brightness, the R G and B values are read from memory, a value is added (or subtracted) from them, and the resulting values are written back out to memory.

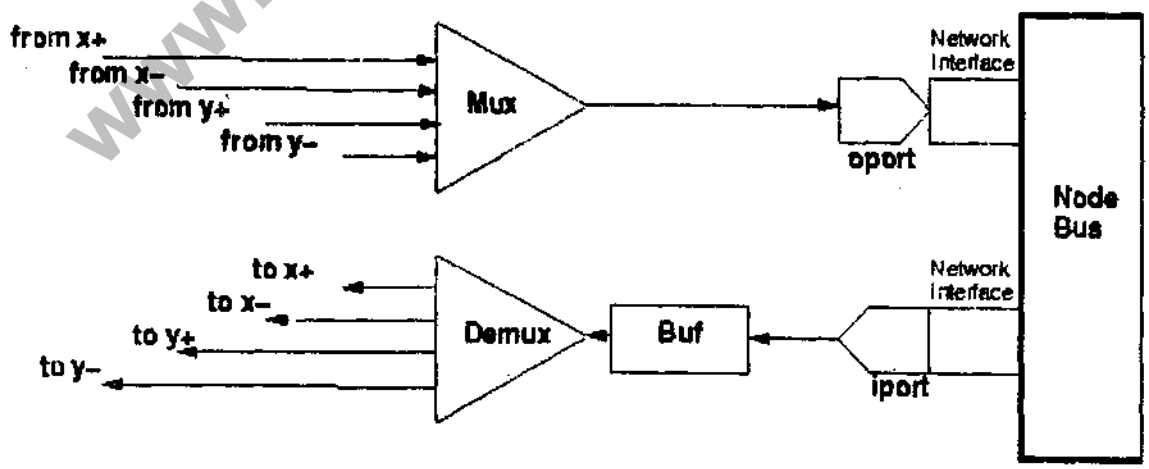
With a SIMD processor there are two improvements to this process. For one the data is understood to be in blocks, and a number of values can be loaded all at once. Instead of a series of instructions saying "get this pixel, now get the next pixel", a SIMD processor will have a single instruction that effectively says "get lots of pixels" ("lots" is a number that varies from design to design). For a variety of reasons, this can take much less time than "getting" each pixel individually, like with traditional CPU design.

Another advantage is that SIMD systems typically include only those instructions that can be applied to all of the data in one operation. In other words, if the SIMD system works by loading up eight data points at once, the add operation being applied to the data will happen to all eight values at the same time. Although the same is true for any superscalar processor design, the level of parallelism in a SIMD system is typically much higher.

Q. 5. Attempt any two parts of the following— **(10×2=20)**

Q.(a). Explain various types of interconnect networks used in multiprocessor systems.

Ans. Interconnection networks used in multiprocessor system



The interconnection network includes separate request and reply networks for deadlock-avoidance. Unlike the other subsystems discussed in this chapter, the network is not built using the standard module framework.

The network flit delay, arbitration delay, width, and buffer sizes can be configured as described in Chapter 4. Additionally, the system can be directed to simulate pipelined switches, by which the flit delay of multiple flits can be incurred in a pipelined manner. To model this behavior, a system with pipelined switches uses a flit delay equal to the granularity of pipelining and adds the remainder of the originally specified flit delay to the arbitration delay of the multiplexers. With these adjustments, the latency of the head flit of a packet remains the same, but subsequent flit delays are based on the degree of pipelining.

The processor connection to the network of each node is depicted in Figure 15.1. Similar components connect the node to the interconnection network and neighboring processors along the X and Y axes. Messages are injected into the network using the SmnetSend event and are received from the interconnection network using the ReqRcvSemaWait and ReplyRcvSemaWait events, corresponding to the request and reply networks, respectively.

Processor side 2D-mesh switch connection

The network routes packets using dimension-ordered routing, and each switch provides wormhole routing. At each buffer, port, multiplexor or demultiplexor, the packet's head flit determines its next destination in the network. When moving from one buffer to the next, the head flit encounters a delay of flitdelay cycles, which corresponds to the flit latency, possibly adjusted for pipelining as described above. In addition, the head flit consumes arbitration delays (possibly adjusted for pipelining) at each multiplexor. (NETSIM allows the head flit to experience routing delays at demultiplexors, but RSIM does not currently use this feature; these delays are set to 0.) A packet's remaining flits are moved when the tail flit is allowed to move. A tail flit is allowed to move in the network as long as it does not share a buffer with the head flit since the tail is never allowed to overtake the head of a packet. Once the tail is allowed to move, the simulator moves all intermediate flits in a pipelined fashion every flitdelay cycles until the tail flit itself moves. The various flits in the packet may thus span several network buffers at any time. This tail movement process continues until the packet has reached the destination output port or until the tail has caught up with its head flit.

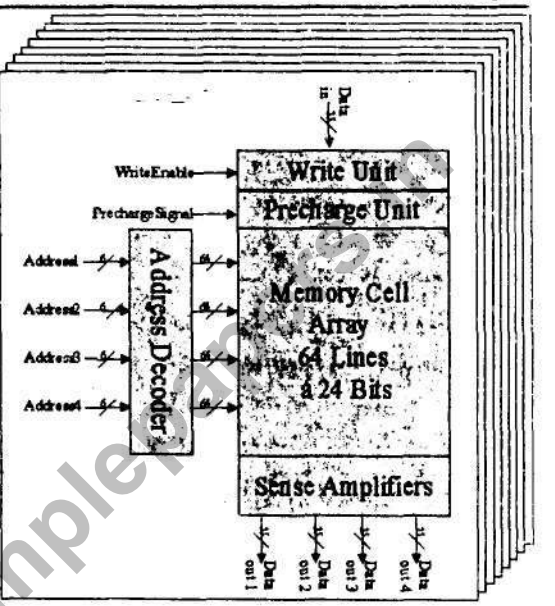
Q.5.(b). Differentiate multiport memory and multistage network.

Ans. Multiport Memory and Multistage network

A memory shared by many processors to communicate among themselves. Multi-port memories offer a great way to interface between two busses, particularly between processors that need to communicate together. Yet these simple devices offer incredible power, internally arbitrating between the two devices with interrupts, semaphores or busy logic. Integrated Device Technology,

Multiport Memory

- 4 Port SRAM
- Delivers/receives data to/from 4 CPUs simultaneously
- Target frequency is 133MHz
- Organised in blocks @ 64 lines
- 8 blocks instruction mem.
- 4 blocks data memory



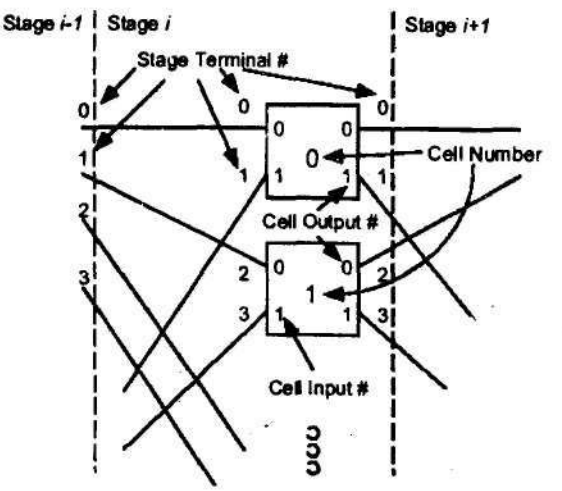
the number one supplier of multi-port memories has put together answers to engineers' most common questions about these products, giving you a great way to become an expert on multi-port products.

Multistage network:

- A MIN consists of cells and links.
- Cells arranged in stages.

Links connect cells in adjacent stages

Multistage-Network Terminology



The following numbering will be used:

Stages, consecutively from 0 (input) to s ¡ 1 or n ¡ 1.

Cells within a stage, consecutively from 0.

Cell inputs and outputs, consecutively from 0 for each cell.

Cell inputs and outputs also given stage terminal numbers.

Terminal numbers consecutively from 0, for all cells.

Link Patterns

A link pattern is an arrangement of links that could connect adjacent stages in a MIN.

More Formally:

Let T1 and T2 be equal-cardinality sets of terminal numbers.

A link pattern is a bijection from T1 to T2.

Example:

Let T1 = f0; 1; 2g and T2 = f0; 1; 2g.

Let ¼(0) = 1, ¼(1) = 0, and ¼(2) = 2.

Mapping ¼ is a bijection from T1 to T2, and therefore a link pattern.

Let ½0(0) = 1, ½0(1) = 1, and ½0(2) = 2.

Mapping ½0 is not a bijection, and so not a link pattern.

Q.5.(c). What is the memory arbiter? Discuss arbitration policies. Give the circuit diagram of the simple arbiter.

Ans. Memory arbiter:

Memory arbiter design that gives dynamic IP cores interfaced to multiple internal networks on the programmable chip concurrent access to multiple banks of the SRAM. The arbiter which uses a new fast version of round robin that we call idle skip, has a small instruction set which is invoked by applications allowing them to read and write multiple memory locations, read and write multiple memory locations in a streaming fashion and perform inter application communication with and without access to external SRAM.

Arbitration policies:

An atomic test and set instruction is provided that allows applications on the FPGA to lock regions of memory in arbitrary sized blocks to enable fine grained producer consumer style interaction between the dynamic IP cores and the host, and between dynamic IP cores of the FPGA.

Memory arbiter for the interface card by specifying two different problems of logic model checking. In the process, we minimise the amount of memory used for intermediate buffering of data streams by augmenting the model with cost variables and applying a guided model checker — Uppaal CORA. It is verified that the resultant arbiter does not deadlock and never starves nor overflows any of the buffers.

Circuit diagram of the simple arbiter:

MEMORY ARBITER SYNTHESIS AND VERIFICATION