

Computer organization and Architecture

Q.No.1.a. Main memory has three pages and processor requires page from virtual memory in the following order 2 3 2 1 5 2 4 5 3 2 5 2 show the implementation of FIFO, LRU, LFU.

10

Ans:

2 3 2 1 5 2 4 5 3 2 5 3

1. FIFO

2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2

F F F F F F F

2. LRU

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2

F F F F

3. LFU

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	3
			1	1	1	4	4	4	2	2	2

Q.No.1.b) Explain SPARC processor in detail?

10

Ans:

The SPARC processor usually contains as many as 160 general purpose register. At any point, only 32 of them are immediately visible to software - 8 are a set of global registers and the other 24 are from the stack of registers.

These 24 registers form what is called a register window, and at function call/return, this window is moved up and down the register stack.

Each window has 8 local registers and shares 8 registers with each of the adjacent windows. The shared registers are used for passing function parameters and returning values and the local registers are used for retaining local values across function calls.

The "Scalable" in SPARC comes from the fact that the SPARC specification allows implementations to scale from embedded processors up through large server processors, all sharing the same core instruction set.

One of the architectural parameters that can scale is the number of implemented register windows; the specification allows from 3 to 32 windows to be implemented, so the implementation can choose to implement all 32 to provide maximum call stack efficiency, or to implement only 3 to reduce context switching time, or to implement some number between them. Other

Computer organization and Architecture

architectures that include similar register file features include Intel i960, IA-64, and AMD 29000.

The architecture has gone through a few revisions. It gained hardware multiply and divide functionality in Version 8. The most substantial upgrade resulted in Version 9, which is a 64-bit SPARC specification published in 1994.

In SPARC Version 8, the floating point register file has 16 double precision registers. Each of them can be used as two single precision registers, providing a total of 32 single precision registers.

Q.No.2.a

Explain Systolic processor with suitable example.

10

Ans:

A systolic array is composed of matrix-like rows of data processing units called cells. Data processing units DPUS are similar to Central Processing Unit. Each cell shares the information with its neighbors immediately after processing.

The systolic array is often rectangular where data flows across the array between neighbour DPUs, often with different data flowing in different directions.

The data streams entering and leaving the ports of the array are generated by Auto Sequencing Memory units, ASMs. Each ASM includes a Data Counter. In Embedded System a data stream may also be input from and/or output to an external source.

An example of a systolic algorithms might be designed for Matrix Multiplication . One matrix is fed in a row at a time from the top of the array and is passed down the array, the other matrix is fed in a column at a time from the left hand side of the array and passes from left to right.

Dummy values are then passed in until each processor has seen one whole row and one whole column. At this point, the result of the multiplication is stored in the array and can now be output a row or a column at a time, flowing down or across the array.

Systolic arrays are arrays of DPUs which are connected to a small number of nearest neighbour DPUs in a mesh-like topology. DPUs perform a sequence of operations on data that flows between them.

Because the traditional systolic array synthesis methods have been practiced by algebraic algorithms, only uniform arrays with only linear pipes can be obtained, so that the architectures are the same in all DPUs.

The consequence is, that only applications with regular data dependencies can be implemented on classical systolic arrays. Like SIMD machines, clocked systolic arrays compute in "lock-step" with each processor undertaking alternate compute | communicate phases.

Q.No.2.b).

What is cache memory? Explain cache coherence strategy in single and multiprocessor system

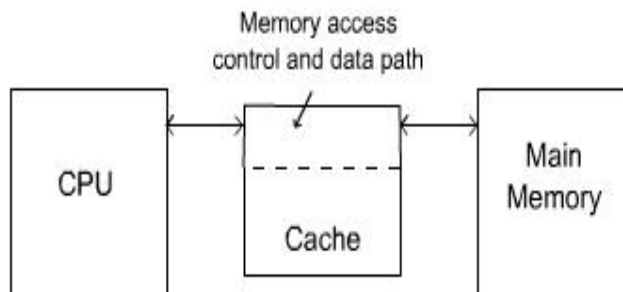
10

Ans:

Analysis of large number of programs has shown that a number of instructions are executed repeatedly. This may be in the form of a simple loops, nested loops, or a few procedures that repeatedly call each other.

Computer organization and Architecture

It is observed that many instructions in each of a few localized areas of the program are repeatedly executed, while the remainder of the program is accessed relatively less. This phenomenon is referred to as locality of reference.



Operation of Cache Memory

The memory control circuitry is designed to take advantage of the property of locality of reference. Some assumptions are made while designing the memory control circuitry:

- The CPU does not need to know explicitly about the existence of the cache.
- The CPU simply makes Read and Write request. The nature of these two operations is same whether cache is present or not.
- The address generated by the CPU always refers to location of main memory.
- The memory access control circuitry determines whether or not the requested word currently exists in the cache.

When a Read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache. When any of the locations in this block is referenced by the program, its contents are read directly from the cache.

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

Computer organization and Architecture

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.

Q.No.3.a).

Compare and contrast DMA, Programmed IO and Interrupt driven IO.

10

Ans:

Programmed I/O:

In programmed I/O, the data transfer between CPU and I/O device is carried out with the help of a software routine. When a processor is executing a program and encounters an instruction relating to I/O, it executes that I/O instruction by issuing a command to the appropriate I/O module.

The I/O module will perform the requested action and then set the appropriate bits in the I/O status register. The I/O module takes no further action to alert the processor. It is the responsibility of the processor to check periodically the status of the I/O module until it finds that the operation is complete.

In programmed I/O, when the processor issues a command to a I/O module, it must wait until the I/O operation is complete. Generally, the I/O devices are slower than the processor, so in this scheme CPU time is wasted. CPU is checking the status of the I/O module periodically without doing any other work.

Interrupt driven I/O

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module.

This type of I/O operation, where the CPU constantly tests a part to see if data is available, is polling, that is, the CPU Polls (asks) the port if it has data available or if it is capable of accepting data. Polled I/O is inherently inefficient.

The solution to this problem is to provide an interrupt mechanism. In this approach the processor issues an I/O command to a module and then go on to do some other useful work. The I/O module then interrupt the processor to request service when it is ready to exchange data with the processor.

The processor then executes the data transfer. Once the data transfer is over, the processor then resumes its former processing.

Direct Memory Access

In above both the methods require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

Computer organization and Architecture

The I/O transfer rate is limited by the speed with which the processor can test and service a device.

The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer. To transfer large block of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access or DMA.

DMA transfers are performed by a control circuit associated with the I/O device and this circuit is referred as DMA controller. The DMA controller allows direct data transfer between the device and the main memory without involving the processor.

Q.No.3.b)

What is RAID? Explain different RAID level in detail.

10

Ans:

RAID, an acronym for *Redundant Array of Independent Disks* (Changed from its original term *Redundant Array of Inexpensive Disks*), is a technology that provides increased storage functions and reliability through redundancy. This is achieved by combining multiple disk drive components into a logical unit, where data is distributed across the drives in one of several ways called "RAID levels".

A number of standard schemes have evolved which are referred to as *levels*. There were five RAID levels originally conceived, but many more variations have evolved, notably several nested level and many non standard levels. RAID levels and their associated data formats are standardized by SNIA in the Command RAID disc drive format.

RAID LEVELS:

1) In **RAID 0** (block-level striping without parity or mirroring) has no (or zero) redundancy. It provides improved performance and additional storage but no fault tolerance. Hence simple stripe sets are normally referred to as RAID 0.

Any disk failure destroys the array, and the likelihood of failure increases with more disks in the array (at a minimum, catastrophic data loss is almost twice as likely compared to single drives without RAID).

2) In **RAID 1** (mirroring without parity or striping), data is written identically to multiple disks (a "mirrored set"). Although many implementations create sets of 2 disks, sets may contain 3 or more disks. Array provides fault tolerance from disk errors or failures and continues to operate as long as at least one drive in the mirrored set is functioning. With appropriate operating system support, there can be increased read performance, and only a minimal write performance reduction. Using RAID 1 with a separate controller for each disk is sometimes called *duplexing*.

3) In **RAID 2** (bit-level striping with dedicated Hamming-code parity), all disk spindle rotation is synchronized, and data is striped such that each sequential bit is on a different disk. Hamming-code parity is calculated across corresponding bits on disks and stored on one or more parity disks.

4) In **RAID 3** (byte-level striping with dedicated parity), all disk spindle rotation is synchronized, and data is striped so each sequential byte is

Computer organization and Architecture

on a different disk. Parity is calculated across corresponding bytes on disks and stored on a dedicated parity disk.

5) In **RAID 4** (block-level striping with dedicated parity) is identical to RAID 5 but confines all parity data to a single disk, which can create a performance bottleneck. In this setup, files can be distributed between multiple disks. Each disk operates independently which allows I/O requests to be performed in parallel, though data transfer speeds can suffer due to the type of parity. The error detection is achieved through dedicated parity and is stored in a separate, single disk unit.

6) **RAID 5** (block-level striping with distributed parity) distributes parity along with the data and requires all drives but one to be present to operate; drive failure requires replacement, but the array is not destroyed by a single drive failure. Upon drive failure, any subsequent reads can be calculated from the distributed parity such that the drive failure is masked from the end user. The array will have data loss in the event of a second drive failure and is vulnerable until the data that was on the failed drive is rebuilt onto a replacement drive.

7) **RAID 6** (block-level striping with double distributed parity) provides fault tolerance from two drive failures; array continues to operate with up to two failed drives. This makes larger RAID groups more practical, especially for high-availability systems. This becomes increasingly important as large-capacity drives lengthen the time needed to recover from the failure of a single drive.

Q.No.4.a)

Explain different bus arbitration scheme

10

Ans:

Bus Arbitration

In single bus architecture when more than one device requests the bus, a controller called bus arbiter decides who gets the bus, this is called the bus arbitration. Arbitration is mostly done in favour of a master micro processor with the highest priority.

The simplest page-replacement algorithm is a FIFO algorithm. The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little book-keeping on the part of the Operating System.

The idea is obvious from the name - the operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the earliest arrival in front. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected. While FIFO is cheap and intuitive, it performs poorly in practical application. Thus, it is rarely used in its unmodified form. This algorithm experiences Belady's Anomaly

The least recently used page (LRU) replacement algorithm, though similar in name to NRU, differs in the fact that LRU keeps track of page usage over a short period of time, while NRU just looks at the usage in the last clock interval.

Computer organization and Architecture

LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. While LRU can provide near-optimal performance in theory, it is rather expensive to implement in practice. There are a few implementation methods for this algorithm that try to reduce the cost yet keep as much of the performance as possible.

One important advantage of the LRU algorithm is that it is amenable to full statistical analysis. It has been proven, for example, that LRU can never result in more than N -times more page faults than OPT algorithm, where N is proportional to the number of pages in the managed pool.

On the other hand, LRU's weakness is that its performance tends to degenerate under many quite common reference patterns. For example, if there are N pages in the LRU pool, an application executing a loop over array of $N + 1$ pages will cause a page fault on each and every access. As loops over large arrays are common, much effort has been put into modifying LRU to work better in such situations. Many of the proposed LRU modifications try to detect looping reference patterns and to switch into suitable replacement algorithm, like Most Recently Used (MRU).

Q.No.4.b).

What is pipelining? Explain six stage instruction pipeline with suitable diagram.

10

Ans:

Pipelining :

It is observed that organization enhancements to the CPU can improve performance. The use of multiple registers rather than a single accumulator, and use of cache memory improves the performance considerably. Another organizational approach, which is quite common, is instruction pipelining.

Pipelining is a particularly effective way of organizing parallel activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly line operation.

By laying the production process out in an assembly line, product at various stages can be worked on simultaneously. This process is also referred to as pipelining, because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end. The processing of an instruction need not be divided into only two steps. To gain further speed up, the pipeline must have more stages.

Consider the following decomposition of the instruction execution:

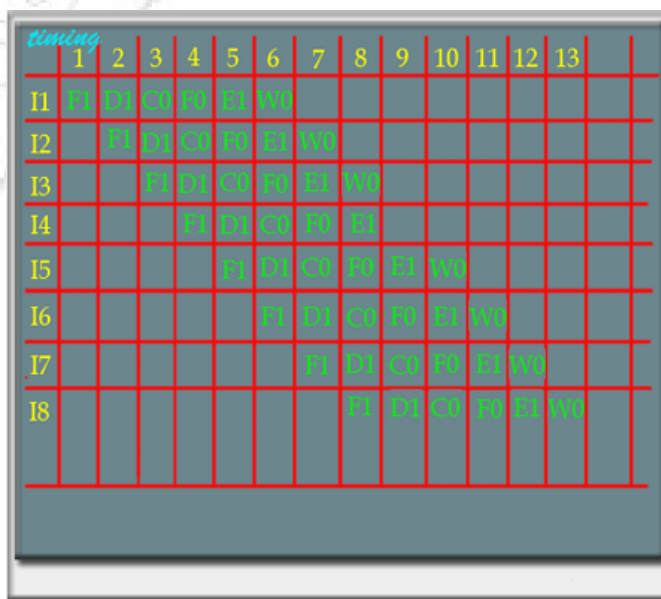
- Fetch Instruction (FI): Read the next expected instruction into a buffer.

Computer organization and Architecture

- Decode Instruction ((DI): Determine the opcode and the operand specifiers.
- Calculate Operand (CO): calculate the effective address of each source operand.
- Fetch Operands (FO): Fetch each operand from memory.
- Execute Instruction (EI): Perform the indicated operation.
- Write Operand (WO): Store the result in memory.

There will be six different stages for these six subtasks. For the sake of simplicity, let us assume the equal duration to perform all the subtasks. If the six stages are not of equal duration, there will be some waiting involved at various pipeline stages.

The timing diagram for the execution of instruction in pipeline is shown in the Figure



From this timing diagram it is clear that the total execution time of 8 instructions in this 6 stages pipeline is 13-time unit. The first instruction gets completed after 6 time unit, and there after in each time unit it completes one instruction.

Without pipeline, the total time required to complete 8 instructions would have been 48 (6 X 8) time unit. Therefore, there is a speed up in pipeline processing and the speed up is related to the number of stages.

Q.No.5.a) Explain with suitable example Booths algorithm.

10

Ans:

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1951 while doing research on crystallography at Birkbeck

Computer organization and Architecture

College in Bloomsbury, London. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed..

Procedure

Booth's algorithm involves repeatedly adding one of two predetermined values A and S to a product P , then performing a rightward arithmetic shift on P . Let \mathbf{m} and \mathbf{r} be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in \mathbf{m} and \mathbf{r} .

- Determine the values of A and S , and the initial value of P . All of these numbers should have a length equal to $(x + y + 1)$.
 - A : Fill the most significant (leftmost) bits with the value of \mathbf{m} . Fill the remaining $(y + 1)$ bits with zeros.
 - S : Fill the most significant bits with the value of $(-\mathbf{m})$ in two's complement notation. Fill the remaining $(y + 1)$ bits with zeros.
 - P : Fill the most significant x bits with zeros. To the right of this, append the value of \mathbf{r} . Fill the least significant (rightmost) bit with a zero.
- Determine the two least significant (rightmost) bits of P .
 - If they are 01, find the value of $P + A$. Ignore any overflow.
 - If they are 10, find the value of $P + S$. Ignore any overflow.
 - If they are 00, do nothing. Use P directly in the next step.
 - If they are 11, do nothing. Use P directly in the next step.
- Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.
- Repeat steps 2 and 3 until they have been done y times.
- Drop the least significant (rightmost) bit from P . This is the product of \mathbf{m} and \mathbf{r} .

Example

Find $3 \times (-4)$, with $\mathbf{m} = 3$ and $\mathbf{r} = -4$, and $x = 4$ and $y = 4$:

- 1) $A = 0011\ 0000\ 0$
- 2) $S = 1101\ 0000\ 0$
- 3) $P = 0000\ 1100\ 0$

- Perform the loop four times :
 - $P = 0000\ 1100\ 0$. The last two bits are 00.
 - $P = 0000\ 0110\ 0$. Arithmetic right shift.
 - $P = 0000\ 0110\ 0$. The last two bits are 00.
 - $P = 0000\ 0011\ 0$. Arithmetic right shift.
 - $P = 0000\ 0011\ 0$. The last two bits are 10.
 - $P = 1101\ 0011\ 0$. $P = P + S$.
 - $P = 1110\ 1001\ 1$. Arithmetic right shift.
 - $P = 1110\ 1001\ 1$. The last two bits are 11.
 - $P = 1111\ 0100\ 1$. Arithmetic right shift.

1. The product is 1111 0100, which is -12 .

Computer organization and Architecture

The above mentioned technique is inadequate when the multiplicand is the largest negative number that can be represented (e.g. if the multiplicand has 4 bits then this value is -8). One possible correction to this problem is to add one more bit to the left of A, S and P. Below, we demonstrate the improved technique by multiplying -8 by 2 using 4 bits for the multiplicand and the multiplier:

- A = 1 1000 0000 0
- S = 0 1000 0000 0
- P = 0 0000 0010 0
- Perform the loop four times :
 - P = 0 0000 0010 0. The last two bits are 00.
 - P = 0 0000 0001 0. Right shift.
 - P = 0 0000 0001 0. The last two bits are 10.
 - P = 0 1000 0001 0. P = P + S.
 - P = 0 0100 0000 1. Right shift.
 - P = 0 0100 0000 1. The last two bits are 01.
 - P = 1 1100 0000 1. P = P + A.
 - P = 1 1110 0000 0. Right shift.
 - P = 1 1110 0000 0. The last two bits are 00.
 - P = 1 1111 0000 0. Right shift.
- The product is 11110000 (after discarding the first and the last bit) which is -16 .

Q.No.5.b)

Explain Microinstruction, Micro operation and Micro program in detail

10.

Ans:

Micro program:

In hardwired control all the control signals required inside the CPU can be generated using a state counter and a PLA circuit. There is an alternative approach by which the control signals required inside the CPU can be generated. This alternative approach is known as micro programmed control unit.

In micro programmed control unit, the logic of the control unit is specified by a microprogram. A micro program consists of a sequence of instructions in a microprogramming language. These are instructions that specify micro operations.

A micro programmed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstructions and (2) generating control signals to execute each microinstruction.

The concept of micro program is similar to computer program. In computer program the complete instructions of the program is stored in main memory and during execution it fetches the instructions from main memory one after another. The sequence of instruction fetch is controlled by program counter (PC).

Microinstruction:

A microinstruction is an instruction that controls data flow and instruction-execution sequencing in a processor at a more fundamental level than machine

Computer organization and Architecture

instructions. A series of microinstructions is necessary to perform an individual machine instruction.

Micro operation:

In computer central processing units, micro-operations (also known as microops or μ ops) are controlled by detailed low-level instructions (micro-instructions) used in some designs to implement complex machine instructions. Various forms of μ ops have long been the basis for traditional microcode routines used to simplify the implementation of a particular CPU design or perhaps just the sequencing of certain multi-step operations or addressing modes. More recently, μ ops have also been employed in a different way in order to let modern "CISC" processors more easily handle asynchronous parallel and speculative execution: As with traditional microcode, one or more table lookups (or equivalent) is done to locate the appropriate μ op-sequence based on the encoding and semantics of the machine instruction; however, instead of having rigid μ op-sequences controlling the CPU directly from a microcode-ROM, μ ops are here dynamically issued, that is, buffered in rather long sequences before being executed.

Q.No.6.a)

Explain RISC and CISC architecture in detail.

10

Ans:

Reduced instruction set computing, or **RISC** is a CPU design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction. A computer based on this strategy is a **reduced instruction set computer**.

Typical characteristics of RISC

For any given level of general performance, a RISC chip will typically have far fewer transistors dedicated to the core logic which originally allowed designers to increase the size of the register set and increase internal parallelism.

Other features, which are typically found in RISC architectures, are:

- Uniform instruction format, using a single word with the opcode in the same bit positions in every instruction, demanding less decoding;
- Identical general purpose registers, allowing any register to be used in any context, simplifying compiler design (although normally there are separate floating point registers)
- Simple addressing modes. Complex addressing performed via sequences of arithmetic and/or load-store operations;
- Few data types in hardware, some CISCs have byte string instructions, or support complex numbers; this is so far unlikely to be found on a RISC.

A **complex instruction set computer (CISC)** is a computer where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) and/or are capable of multi-step operations or addressing modes within single instructions. The term was retroactively coined in contrast to reduced instruction set computer (RISC).

Computer organization and Architecture

Design issues

While many designs achieved the aim of higher throughput at lower cost and also allowed high-level language constructs to be expressed by fewer instructions, it was observed that this was not *always* the case.

For instance, low-end versions of complex architectures (i.e. using less hardware) could lead to situations where it was possible to improve performance by *not* using a complex instruction (such as a procedure call or enter instruction), but instead using a sequence of simpler instructions.

One reason for this was that architects (microcode writers) sometimes "over-designed" assembler language instructions, i.e. including features which were not possible to implement efficiently on the basic hardware available. This could, for instance, be "side effects".

Such as the setting of a register or memory location that was perhaps seldom used; if this was done via ordinary (non duplicated) internal buses, or even the *external* bus, it would demand extra cycles every time, and thus be quite inefficient.

Q.No.6.b)

Explain Flynn's classification with suitable diagram. Also comment on design issues of pipeline architecture.

10

Ans:

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.

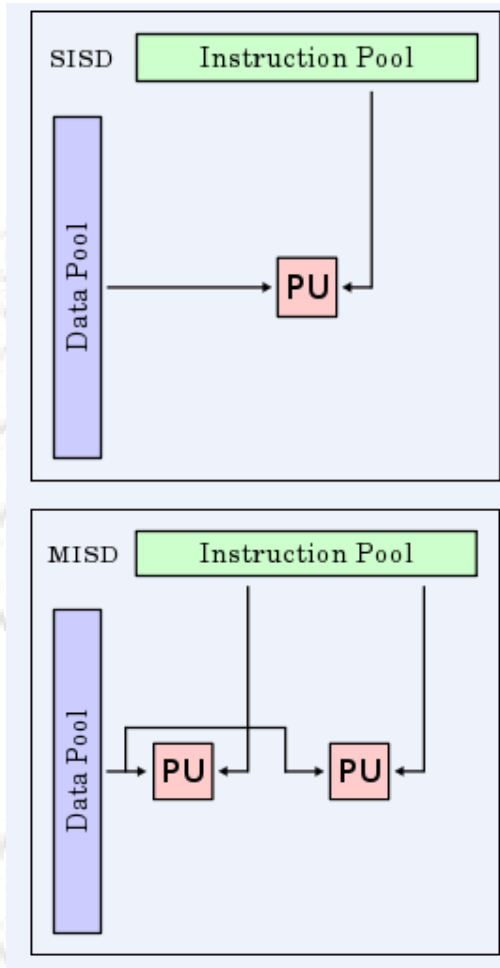
Multiple Instruction, Multiple Data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized

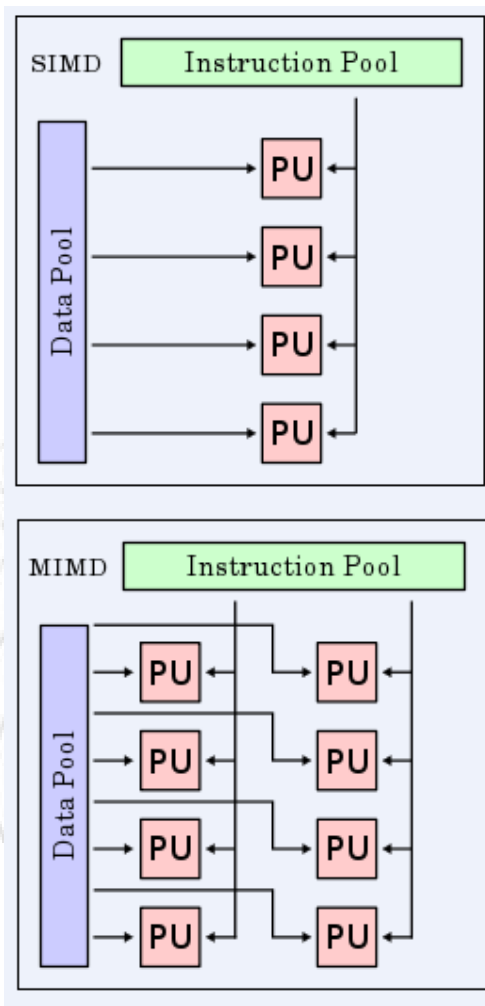
Computer organization and Architecture

to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

DIAGRAMS:



Computer organization and Architecture



Q7

Write a short note on

20

1. PCI BUS

Conventional PCI (PCI is an initialism formed from **Peripheral Component Interconnect** part of the **PCI Local Bus** standard and often shortened to **PCI**) is a computer bus for attaching hardware devices in a computer.

These devices can take either the form of an integrated circuit fitted onto the motherboard itself, called a *planar device* in the PCI specification, or an expansion card that fits into a slot.

The PCI Local Bus is common in modern PCs, where it has displaced ISA and VESA Local Bus as the standard expansion bus, and it also appears in many other computer types. Despite the availability of faster interfaces such as PCI-X and PCI Express, conventional PCI remains a very common interface.

The PCI specification covers the physical size of the bus (including the size and spacing of the circuit board edge electrical contacts), electrical characteristics, bus timing, and protocols. The specification can be purchased from the PCI Special Interest Group (PCI-SIG).

Computer organization and Architecture

Typical PCI cards used in PCs include: network cards, sound cards, modems, extra ports such as USB or serial, TV tuner cards and disk controllers. Historically video cards were typically PCI devices, but growing bandwidth requirements soon outgrew the capabilities of PCI. PCI video cards remain available for supporting extra monitors and upgrading PCs that do not have any AGP or PCI Express slots.

2. Memory Characteristics-

- Performance of main memory-
- storage capacity
- unit of transfer
- Access modes
- Access time-time between address latched and data is available
- Cycle time-time between requests
- total access time- from ld to REG valid
- Refresh and precharge
- cache memory is built from SRAM
- Main memory is built from DRAM

3. INTERLEAVED MEMORY-

Interleaved memory is a technique for compensating the relatively slow speed of DRAM. The CPU can access alternative sections immediately without waiting for memory to be cached. Multiple memory banks take turns supplying data.

An interleaved memory with "n" banks is said to be *n-way interleaved*. If there are "n" banks, memory location "i" would reside in bank number $i \bmod n$.

One way of allocating virtual addresses to memory modules is to divide the memory space into contiguous blocks. 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.

Interleaved memories are the implementation of the concept of accessing more words in a single memory access cycle. This can be

Computer organization and Architecture

achieved by partitioning the memory e.g. into N separate memory modules. Thus, N accesses can be carried out to the memory simultaneously

4) **Loop Buffer-**

A **buffer** is a region of memory used to temporarily hold data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a Mouse) or just before it is sent to an output device (such as Speakers).

However, a buffer may be used when moving data between processes within a computer. This is comparable to buffers in telecommunication. Buffers can be implemented in either hardware or software, but the vast majority of buffers are implemented in software.

Buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler or in online video streaming.

A buffer often adjusts timing by implementing a queue (or FIFO) algorithm in memory, simultaneously writing data into the queue at one rate and reading it at another rate.

Buffers are often used in conjunction with I/O to hardware, such as disk drives, sending or receiving data to or from a network, or playing sound on a speaker.

A line to a rollercoaster in an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded).

The queue area acts as a buffer: a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a FIFO (first in, first out) method, outputting data in the order it arrived.

5) **Microinstruction Sequencing And Execution-**

The micro programmed control unit has to perform two basic task

1. Microinstruction sequencing: Get the next microinstruction from the control memory
2. Microinstruction Execution: Generate the control signals needed to execute the microinstruction

In designing a control unit, these tasks must be considered together, because both affect the format of the microinstruction and timing of control unit.

Design Consideration

Two concerns are involved in the design of a microinstruction sequencing technique: The size of microinstruction and the address generation time

