Lab 4

3 Task: Memory System

1.
Associative arrays are an abstract data type, similar to HashMap in Java and Dictionary in Python, which unlike an array can have any arbitrary thing as its subscript thus providing a great deal of flexibility to work with data.

Its implementation is based on a simple array structure where instead of providing indices 0,1,2, or so on we could simply make the index as a String.

Its internal implementation consists of a table with entries having particular addresses. If we want to find the location of a particular element associated with some subscript, we could simply use the hashing functions which after some manipulations create some number and that number in the table having indices from 0 to N-1 would contain all the associated entries.

But to be cautious the entry is not directly stored in the table. That particular location maps to the element with desired subscript.

Since the table is made of linked lists so by any chance if the programme allots the same location to be mapped to two different entries of the associative arrays then a new node is created containing the entry of the subscript and linked to the previous entry of the same location.

Also, the hash table should be adjusted in such a way that all the locations are uniform with respect to each other. In other words, the linked list associated with all the columns of the table should be of approximately the same length but it should not be huge. Usually the hash table consists of entries with approximately two nodes in each linked list.

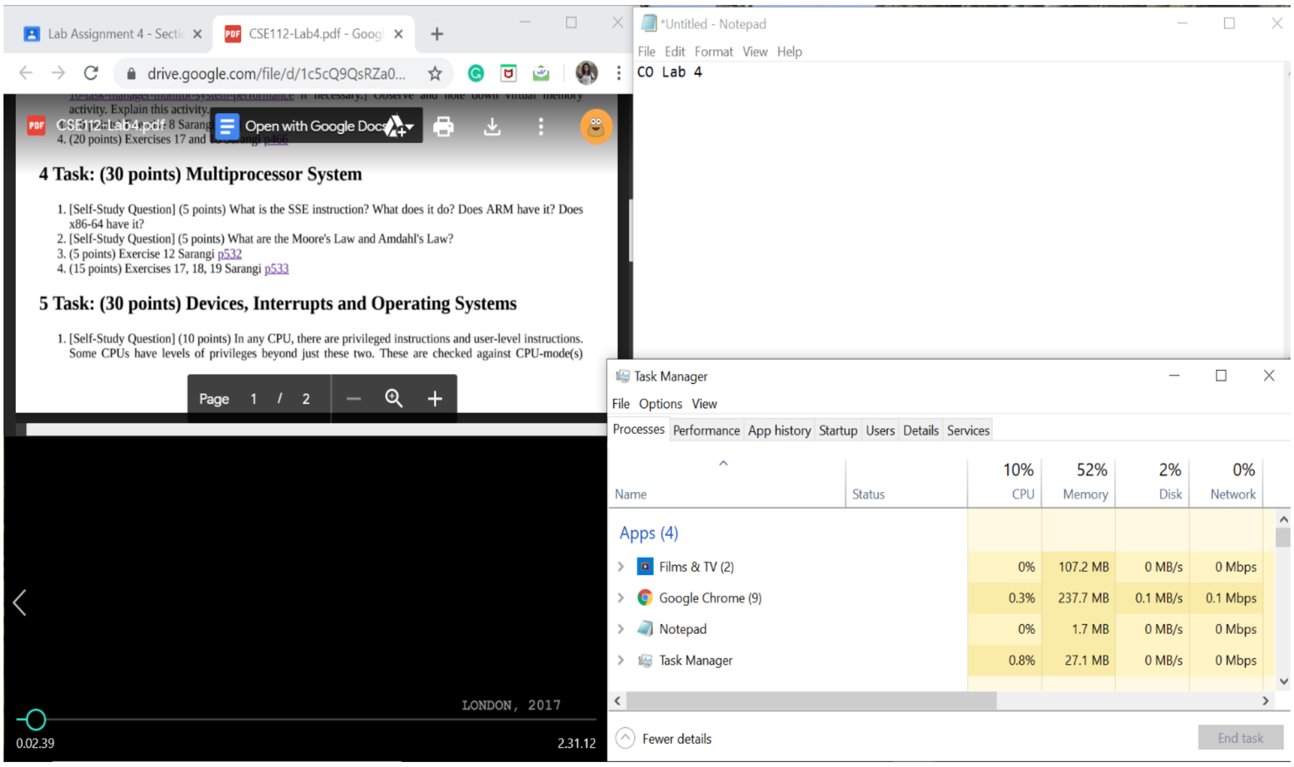
And if all the entries are of constant length then the access to information will be of the constant time. So, if we use a hash function it will tell us where to go and the elements having the same hash will usually be two or less.

Now, if the number of entries is much greater than the value of N, the columns of the hash table then the size of linked list will increase beyond two which is undesirable so to tackle this we will need to rehash everything so that we can get a bigger table and then the possible entries having same hash value could be reduced to two or less. So again, the time to access the information becomes constant.

There is a special hash function with unique properties. This function takes the input of a string or any other object, converts it into integer and then to find the place for a particular entry, integer mod N is calculated to get an integer within the range of that array.

While rehashing a new array of size 2N is created so that the properties of the hash table are not disturbed. Thus, it is a helpful data structure which has inbuilt modules in programming languages named dictionary in Python, associative arrays in PHP etc.

2.



activity. Explain this activity.

4. (20 points) Exercises 17 and ... Sarangi p466

4 Task: (30 points) Multiprocessor System

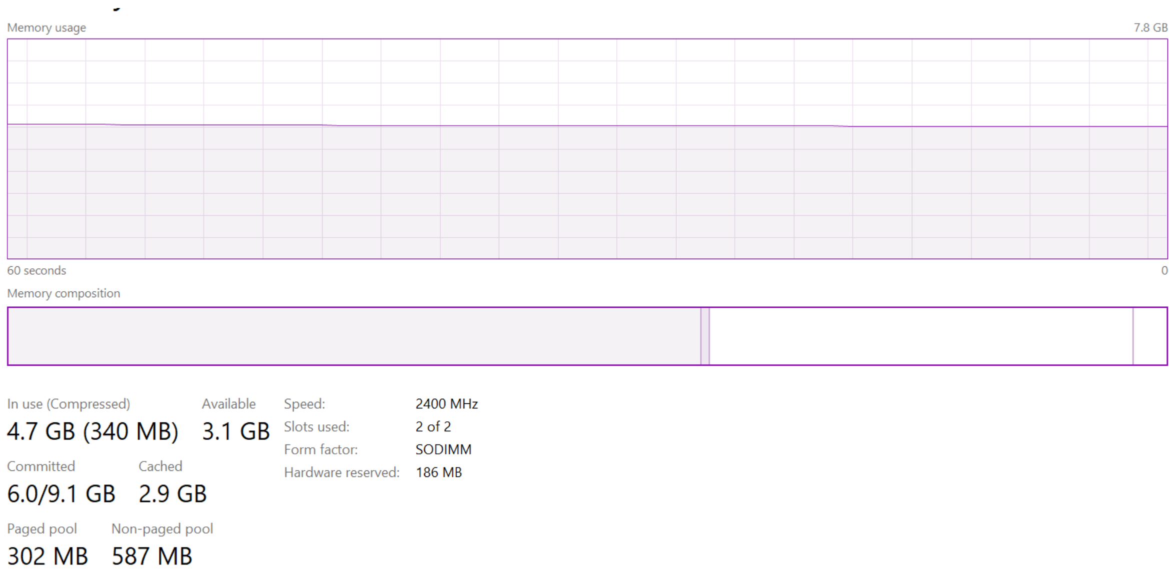
1. [Self-Study Question] (5 points) What is the SSE instruction? What does it do? Does ARM have it? Does x86-64 have it?
2. [Self-Study Question] (5 points) What are the Moore's Law and Amdahl's Law?
3. (5 points) Exercise 12 Sarangi p532
4. (15 points) Exercises 17, 18, 19 Sarangi p533

5 Task: (30 points) Devices, Interrupts and Operating Systems

1. [Self-Study Question] (10 points) In any CPU, there are privileged instructions and user-level instructions. Some CPUs have levels of privileges beyond just these two. These are checked against CPU-mode(s)

```
CO Lab 4
```

Name	Status	10% CPU	52% Memory	2% Disk	0% Network
Apps (4)					
Films & TV (2)		0%	107.2 MB	0 MB/s	0 Mbps
Google Chrome (9)		0.3%	237.7 MB	0.1 MB/s	0.1 Mbps
Notepad		0%	1.7 MB	0 MB/s	0 Mbps
Task Manager		0.8%	27.1 MB	0 MB/s	0 Mbps



In use (Compressed)	4.7 GB (340 MB)
Available	3.1 GB
Committed	6.0/9.1 GB
Cached	2.9 GB
Paged pool	302 MB
Non-paged pool	587 MB
Speed:	2400 MHz
Slots used:	2 of 2
Form factor:	SODIMM
Hardware reserved:	186 MB

6 GB (xGB) is the virtual memory in use. Virtual memory is used because there is not enough space in RAM to support multitasking like above where there are three different programs running simultaneously. Also, there are internal programs and those also require space due to which the virtual memory is required. So, the OS needs xGB to run all the programs together. The OS will use Virtual Memory Manager program to manage xGB of the virtual memory by creating a file on the hard disk of size for the extra memory needed initially. Virual Memory Manager will acquire a memory block from the real memory and do the required operations when the OS will need memory to run different programs, and after doing all the tasks it will

finally move it to RAM in place of the old block of memory. This is how multitasking is accommodated using virtual memory.

3.
In LRU replacement scheme we replace that block that has the lowest chance of being accessed in the future.

In the **write back** policy we take help of the modified bit so that we do not need to write to the lower levels of the memory whenever we get a write request. Here modified is **M**.

	Address	Content	Modified Bit
L1	22	401	1
	23	501	1
L2	20	201	1
	21	301	1
	22	400	1
	23	500	1

Since all the addresses are modified during the sequence of the programme, the modified bit of all the addresses would be 1 as the memory will still have the old data and thus will need to be updated once the block gets evicted.

In **write through** policy there is no need to maintain any modified bit as any write request leads to the change in data at all the levels of memory.

	Address	Content
L1	22	401
	23	501
L2	20	201
	21	301
	22	401
	23	501

Here the content of all the caches of particular addresses is same.

4.

Exercise 17
We have a 32-bit machine system thus there will be 2^{32} different memory locations.
If we consider a 4 KB page then the lower 12 bits specify the address of a byte in a page which is the offset. The upper 20 bits will specify the page number.

Thus, there will be 2^{20} entries in a single level page table and the size of each entry in the page table will be 20 bits.

Exercise 18
4 KB = 4*1024 bytes = 2^{12} bytes

A 32-bit system has 232 different memory locations.
Therefore, no of blocks = $2^{32}/2^{12} = 2^{20}$

Now, it is given that we are using 12 bits to address the primary page table so the number of the bits left is 8.

Thus, in each secondary table there will 2^{8} entries.

4 Task: Multiprocessor System

1.

SSE is the extension of SIMD (Single Instruction Multiple Data) of some microprocessors. It contains several instructions that mainly operate on single precision floating point data. It is helpful when the same operations are to be performed on multiple data objects.

It enables the instruction to handle multiple data elements making the processor faster in certain applications. It is useful in intensive applications. For example, when the 3D graphics are needed or video games.

No, SSE is not present in ARM.

X86 has SSE.

2.

Moore's law: The observation by Gordon Moore states that the number of transistors in a dense integrated circuit doubles every two years. But the growth later slowed down and doubled every 18 months. Now, the number of transistors doubles every two year.

Amdahl's law: This law determines the theoretical speedup in the latency of the execution of a task. It states that the programme's parallel parts can be sped up and not the sequential parts.
Thus, the speed up factor is mainly decided by the fraction of sequential portion of the code.

Tnew = Told *(fseq + (1-fseq)/P)

Where,
Tnew = time taken after the use of parallel processing;
Told = time taken without the use of parallel processing;
fseq = the fraction of code that is sequential;
P = the number of parallel processors

Speed up factor is Tnew/Told = 1/ (fseq + (1 - fseq)/P)

3. Exercise 12

In **sequentially consistent** system, the programme order is maintained. Thus, second instruction must be executed after the first in both the threads. This tells that t1 can take value only 1. Another possibility can be 0 but since thread 2 goes in infinite loop when y=0 and thus even if y=1, t1=x will never be executed hence, t1 can never take value 0.

t1 = 1.

In **weakly consistent** system, it is possible that due to some reasons thread 1 updates the value of y to 1 before x = 1. In this situation, if the thread 2 by any reason runs after the updating of y then thread 2 can assign t1 a value of 0 as x is not updated yet.

t1 = 0 or 1.

4.

Exercise 17

We need to write back to the memory upon M to S transition because M state denotes that the particular block has an updated value about which the main memory is not aware of. While evicting a particular block, we do not want any loss of information, so whenever the block goes from M to S we need to write back to the main memory.

Exercise 18

In all cases, the bus allows only one request per cycle. In case of simultaneous requests, the bus will accept only one request. When there is a clashing request from two nodes to turn from S to M, one additional cycle will be needed for bus arbitration in case of read-miss, write-hit, and write-miss. Thus, the snoopy protocol will let only one node go from S to M in the first clock cycle, and the other node will go from S to M in either the next clock cycle or after some clock cycles.
Reference from Advanced Computer Architecture, 2 E, Tata McGraw Hill, page number 336.

Exercise 19

Due to a large number of broadcasts the snoopy protocol fails because the overhead is too much. Thus, we need to reduce the number of broadcasts to overcome this problem.

We can use directly protocol as a solution to the problem. Here, we use a dedicated structure called a directory to maintain a list of sharers for each block address so that we can reduce the number of broadcasts. Instead of broadcasting data on the bus, a cache sends all of its messages to the directory. And then only a list of sharers needs to be updated when a cache inserts or evicts a block. Thus, it is more scalable.

5 Task: Devices, Interrupts and Operating Systems

1.

PSW is a collection of information that encapsulates the basic execution state of a program at any instant. It permits an interrupted process to resume operation after the interrupt has been handled. The Program Status Word (PSW) is a collection of data which is 64 bits long. Now, we need to ensure that there is a proper execution of the operating system as error in one program can affect other processes. There are two modes of operation to ensure proper execution:

> User Mode: In User mode, the executing code has no ability to directly access hardware or reference memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfil the requests. To switch from kernel to user mode the mode bit should be 1.
>
> Kernel Mode: In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. If user attempt to run privileged instruction in user mode then it will treat instruction as illegal and traps to OS.

There are two types of instructions based on the above mentioned two modes.

> Privileged Instructions: The instructions that can run only in Kernel Mode are called Privileged Instructions. They cannot be executed in user mode as the hardware traps it to the OS. Before transferring the control to any user program, the OS ensures that the timer is set to interrupt. These instructions are used by the OS for correct operation. For example, I/O instructions, set the timer and turn of all interrupts.
>
> Non-Privileged Instructions: The instructions that can only run in user mode are called non-privileged instructions. We require a non-privileged instruction that does not generate any interrupt in order to change the mode from privileged to non-privileged. For example, generate any trap instruction, sending the final printout of printer, etc.

In processors like ARM there are following processor modes:

- SVC Mode: Privileged mode is reserved for the OS.
- Abort Mode: Signalled by the memory system as a result of a failure to load either data or instruction.
- User Mode: Mode in which user application tasks should run.
- IRQ Mode: It is regular interrupt mode. Only r13, r14, and CSPR are shadowed.
- FIQ Mode: This fast interrupt can interrupt the regular interrupts.
- Undefined Mode: When an undefined instruction is encountered, the ARM will wait for a coprocessor to acknowledge that it can deal with the instruction. If no coprocessor responds, or the instruction is one that is not defined, then the undefined instruction vector is taken.

2.

a. Tseek = 50 ms

600 rpm = 10 rps

Time taken for one revolution = 0.1 s

Average rotational latency is time taken for half a revolution,
Rotational Latency = 0.1 / 2 = 0.05 s

Now, time taken to read 25 MB = 25/Bandwidth = 25/100 = 0.25 s

Total time taken to read 25 MB = 0.05 + 0.05 + 0.25 = 0.35 s

0.35s

b. Tseek = 50 ms
600 rpm = 10 rps
Time taken for one revolution = 0.1 s
Rotational Latency = 0.1 / 2 = 0.05 s

Time taken to read 5 MB = 5/Bandwidth = 5/100 = 0.05 s

There will be a delay of 0.1s after each 5MB, due to rotational latency. So, for reading 5 times, we will have to rotate the platter 4 times.

Total time taken to read 25 MB = 0.05 + 0.05 + (0.05) *5 + 0.1*4 = 0.1 + 0.25 + 0.4 = 0.75 s

0.75s

3.

P = D0 ^ D1 ^ D2 ^ D3, (where ^ is xor operator)

= 0xFF00 ^ 0x3421 ^ 0x32FF ^ 0x98AB

= $(1111\ 1111\ 0000\ 0000)_2$ ^ $(0010\ 0100\ 0010\ 0001)_2$ ^ $(0011\ 0010\ 1111\ 1111)_2$ ^ $(1001\ 1000\ 1010\ 1011)_2$

= $(0110\ 0001\ 0111\ 0101)_2$

= 0x6175

Thus, the parity bit is **0x6175.**

Let P’ be the new parity bit.

P’ = P ^ D0 ^ D0’ (D0’ is the new value of D0)
= 0x6175 ^ 0xFF00 ^ 0xABD1
= $(0110\ 0001\ 0111\ 0101)_2$ ^ $(1111\ 1111\ 0000\ 0000)_2$ ^ $(1010\ 1011\ 1101\ 0001)_2$
= $(0011\ 0101\ 1010\ 0100)_2$
= 0x35A4

Thus, the new parity bit is **0x35A4.**

6 Bonus Optional Task: Self Study on Interrupts

1. [Learn Interrupts]

OS is an event driven programme that works only when there is an interrupt or system call. Since OS is crucial to any system it needs to guarantee fairness to user processes. It should not happen that only one process is running. There are two types of events: Interrupts and Exceptions. Interrupt is further divided into two categories software interrupts and hardware interrupts. Interrupts are basically raised by hardware or software to get OS attention whereas exceptions are the illegal operations. **[12:05 11-05-2020].**
Event view of CPU: there is a loop working inside this which works till there is a next instruction. If the CPU gets the next instruction, the instruction starts to execute, if the instruction happens to be an event then it is executed in the event handler. Had the instruction been a simple instruction there would have been no other things but if the instruction happens to be an event then some different things go inside the system. **[12:15 11-05-2020].**

So, to keep track of all the events there is an Interrupt Descriptor Table (IDT) which is used to provide an entry point to every interrupt/exception. Though there are 0 to 255 vectors possible but only 0 to 31 are used, rest is defined by the OS itself. The IDT consists of other information too like the mnemonic, its source and its code. **[12:26 11-05-2020].**

Hardware interrupts can occur due to any device as these events are asynchronous and we do not know when they will happen so these devices need to be occasionally serviced by the CPU which can be done through polling. In polling CPU determines if the device needs attention and if not, it wastes CPU time. **[13:00 11-05-2020].**

Now, during interrupts the device signals to the CPU as it demands attention but the problem arises when two devices simultaneously demand attention. For this we have 8259 Programmable Interrupt Controller which can relay up to 8 interrupts. Now, the interrupt is raised by the interrupt request (IRQ). CPU acknowledges and queries the 8259 to determine which device interrupted. 8259s can be cascaded to support more interrupts. **[14:09 11-05-2020].**

There are 15 IRQs in a legacy CPU. But there are also limitations to this type because it has limited IRQs and it faces spurious interrupts by 8259. There are two types of interrupts in a legacy CPU: edge and level. The level triggered interrupt stays as long as the IRQ line is asserted. It remains active even when the interrupt service is complete. To stop the interrupt, we need to physically deactivate the interrupt. But for an edge triggered interrupt exactly one interrupt occurs when the IRQ line is inserted. Active high interrupts denote logic 1 which symbolises the particular interrupt which is active. These interrupts can be better explained by **crying baby analogy**. **[14:50 11-05-2020].**

There is another kind of interrupt called spurious interrupt. In this case the message that there is an interrupt is sent to the CPU but then the device de-asserts. To prevent this the PIC sends a fake vector number called spurious IRQ. This is the lowest priority IRQ. **[15:08 11-05-2020].**

In an Advanced Programmable Interrupt Controller (APIC), external interrupts are routed from peripherals to CPUs in multi-processor systems through APIC. APIC distributes and prioritizes interrupts to processors. Interrupts can be configured as edge or level triggered. It comprises two components Local APIC and I/O APIC. They communicate through a special three wire APC bus. **[18:03 11-05-2020].**

LAPIC receives interrupts from I/O APIC and routes it to the local CPU It can also receive local interrupts (such as from thermal sensors, internal timer, etc). IPIs used to distribute interrupts between processors or execute system wide functions like booting, load distribution, etc. I/O APIC is present in chipset (north bridge). It is used to route external interrupts to local APIC. The mechanism that carries on when there is an interrupt: the device asserts IRQ of I/O APIC, I/O APIC transfers interrupt to LAPIC, LAPIC asserts CPU interrupts, when the instruction from the CPU completes it obtains IRQ number from the LAPIC and switches to kernel stack if necessary. **[19:00 11-05-2020].**

Now, each process has two stacks, a user space stack and a kernel stack. When an event is executed by the OS, the privilege changes from low to high, if it is already in OS then there is no privilege change. Now, to switch the stack CPU should know locations of the new SS and ESP. It is done by a task segment descriptor. **[21:05 12-05-2020].**

To decide whether we need a switch or not we need to see if it is executing in kernel space or user space. If it is executing in kernel space then there is no stack switch and the current switch is used. However, if it is executing in user space then stack is switched to kernel switch. There is a task state segment that is a specialised segment for hardware supporting multitasking. The important contents of the TSS is used to find new stack and then stack can be switched. **[21:28 12-05-2020].**

We need to save the program state so that the current program being executed must be able to resume after interrupt service is completed. There is also a routine to find the interrupt/ exception. To do this there is an Interrupt descriptor table which is also called interrupt vectors. It is initialised by OS at boot. This routine is done automatically by the CPU. **[22:00 12-05-2020].**

Typical Interrupt Handler saves additional CPU context and invokes kernel scheduler. It also restores CPU context and return. The interrupt latency holds importance in the real time systems as OS should ‘guarantee’ interrupt latency is less than a specified value. The minimum interrupt latency occurs mostly due to the interrupt controller and maximum interrupt latency happens due to the OS and occurs when interrupt handler cannot be serviced immediately. There are some atomic operations too that goes on inside. When the kernel code gets an interrupt, the interrupt is transferred to the interrupt handler and then the programme goes back to the kernel code. To improve responsiveness, enabling Interrupts within handlers causes nested interrupts and makes them more responsive but difficult to develop and validate. **[22:40 12-05-2020].**

Small interrupt handlers do as little as possible in the interrupt handler. There is a top and bottom half technique in Linux. Top half: do minimum work and return from interrupt handler. Bottom half: deferred processing. In the keyboard interrupt handler, there are service special characters which get pushed into the circular buffer. **[22:50 12-05-2020].**

Software Interrupt takes place when an instruction causes an interrupt. It is used for implementing system calls. Its system call processing in the kernel is almost similar to hardware interrupts. The system call number is used to distinguish between system calls. Based on the system call number function syscall invokes the corresponding syscall handler. Passing parameters to system calls not similar to passing parameters in function calls. The typical methods used are pass by register, pass via user mode stack and pass via a designated memory region. In pass by registers System calls with fewer than 6 parameters are passed in registers. If 6 or more arguments, pass pointer to block structure containing argument list. **[10:24 13-05-2020].**

The various exception sources are program-error exceptions, software generated exceptions and machine check exceptions. There are three types of exception faults, traps and aborts. Faults Exception that generally can be corrected. Once corrected, the program can continue execution. Traps are reported immediately after the execution of the trapping instruction. Aborts Severe unrecoverable errors. **[12:00 13-05-2020].**

2. [Critique the Lecture Videos on Interrupts]

Problem Statement
This is a critique of the lectures on Interrupt from three different IITs. It will consist of the reviews made after deeply analysing the videos. All the lectures are from NPTEL site and need to be examined according to the details in the lecture. It will be based upon the different topics discussed, the way of discussion and the completeness of the lectures. Other parameters will also be taken account of. The lecture from IIT Kharagpur is delivered by Prof. Indranil Sengupta, Prof. J. K. Deka from IIT Guwahati, and the lecture from IIT Madras is delivered by Prof S. Raman.

Prof. Indranil Sengupta.
He says that he will be discussing interrupt handling in the lecture. He goes on to discuss what happens when an interrupt request arrives in a detailed manner. He explains that in between the execution cycle the interrupts are not acknowledged. He shows a flowchart on general interrupt processing. However, he should have used diagrams too for a better explanation. He briefs about the interrupt vector concept but does not explain in detail. Then he goes on to show the diagram for a better understanding. He then explains in detail how the priority is set for the interrupt. However, he does not explain the various types of interrupts like software and hardware interrupt for a better overview. He does not explain in detail the different cases that cause difficulty in interrupt handling: fault, trap and abort. He only discusses examples. He goes deep into handling multiple devices. There he explains about device identification with help of a diagram. The explanation about interrupt nesting in handling multiple devices is vivid and clear. Now, he explains about the software interrupt and hardware interrupt which I believe should have been explained before as it gives a better overview and how they are managed. He also explains maskable and non-maskable interrupt. The examples make to delve into the concept and excites us about the notion. He then briefs about exception extending from interrupts which provides a vivid overview. However, due to lack of diagrams and only block diagrams the lecture looked monotonous.

Prof. S. Raman
His lecture looks lively in the starting. He starts with CPU polling which I think should have been explained later and in more detail. I feel that the lecture structure lacks organisation which hampers healthy learning. He goes on to explain about the peripheral devices.

However, his discussion on different modes of data transfer is informative and detailed providing a broad overview. The lecture looks interactive as he asks questions in between thus making the learner more involved. He explains the concept of interrupt inspired by story personifying CPU which makes the concentration. However, it is difficult to take notes as he is speaking and making diagrams due to which it becomes difficult to have organised notes. Then he jumps to multiple interrupt handling. He then explains about vector which points to different vectors. He discusses how different codes can be used to identify devices. He first asks some question prompting us to think and then tells the answer which is a nice strategy followed in the entire video. He brings in the concept of stack switching without sophistication and how CPU switches between programmes. Though he weaves the lecture in a story yet it sounds a bit unorganised as he did not explain the software and hardware parts of the programme of the mechanism completely confusing the learner sometimes. He missed some topics which I believe he should have explained in the lecture like the difficulty in the interrupt handling. Trap, fault and abort are some important concepts which he did not explain in the video. However, he did discuss interrupt latency time and overheads of the mechanism.

Prof. J. K. Deka
His lecture commences with clearly defining the objectives of the lecture giving a broad overview. His voice is not clear which can hamper the concentration. He explains how the interrupt is handled mainly at the end of the instruction and not in the middle. He tells about the program status word and its requirement. The lecture notes are not descriptive and no diagrams making it difficult to keep track with the prof. However, the instruction cycle state diagram is quite informative and clears all the doubts. He explains the mechanism simple interrupt processing in a detailed manner by a flow chart and dividing the procedure into hardware and software. He took so time in explaining a single concept with only one flowchart that it became boring. He then discusses design issues but the lecture slide still look incomplete. The he tells about software poll which he has already given an idea about in the beginning. This way it helped to relate the concepts. He explains various ways of identifying different interrupt modules like daisy chain and hardware poll. He reiterates on multiple interrupts which brings out its significance. Different examples about PC bus gives a better overlook of on different OS. The diagram of 82C59A at last is helpful to clear the example given. Now, he aims to test the learning by asking some questions on the objectives which he had initially mentioned in the lecture. He even gives questions on the topic combining from two objectives. He then gives answers to the question so that the students are not left pondering over. However, he missed some points to discuss like the difficulties in the interrupt handling, how to pass parameters in system calls. He did not discuss interrupt latency, the vectors and the concept of switching stack by the CPU.

Conclusion
All the lectures covered more or less all the topics in somewhat detail. However, all the videos lacked on some aspects making video either monotonous or less clear. But the discussion went proper smooth and lectures tried to interact with the learners by presentation, diagrams and questions too. These lectures can be helpful to learn the topic but for a better understanding one must read a book to supplement the lectures. The lectures were slow paced. Had they been a bit fast it would have consumed less time. The topics discussed in every lecture were greatly emphasised on and relatable with the course outline. The profs tried to stick to the outline and did it greatly. However, the sequence of the topics and way of delivering lectures differed in all the three lectures. Each instructor made the best use of the slides. However, some lectures looked unorganised. Some used the concept of interrupt to

extend to exception which was a great idea to provide better path to the upcoming lectures but some lacked in this aspect. Traps, fault and interrupt are important topics which was not discussed in any of the lectures.