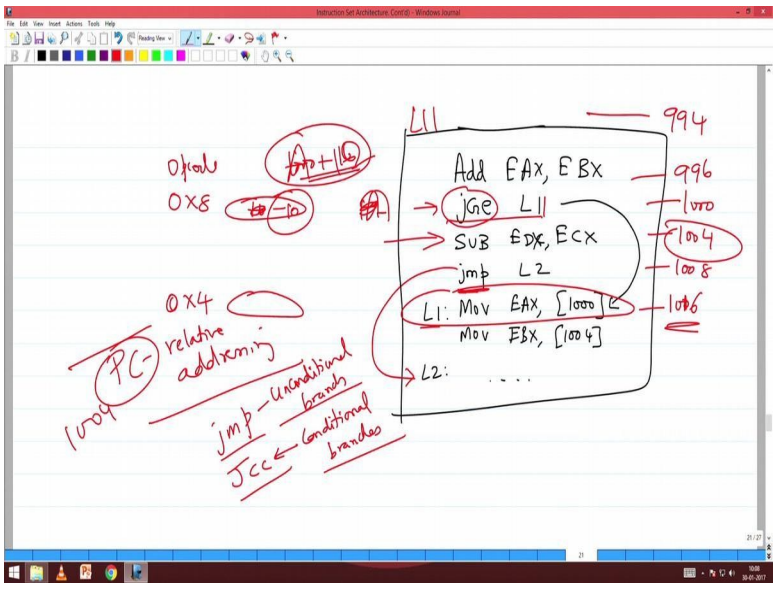
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 13
Atomic and Predicated Instructions

We have been talking about instruction set architecture, and last class we saw about addressing modes different verities of addressing modes. So, we will just see one more addressing mode and then see two very important classes of instructions that are in modern processor and the needs shall. So, we will take two very important classes of instructions in additions to one more additional addressing mode. I hope we have started writing the assembly programs. So, we have written a couple of assembly programs if it is no please make it yes by writing it today. So, let us see some instructions like you know ok.

(Refer Slide Time: 01:06)



Now you see this simple program this is how you write the assembly code and you would have seen such instructions. Now you will see that there is a jump from this instruction to this label and another jump right now how will this instruction be decoded. So, there is an assembly equivalent of this how will it be assembled. So, I am sure all of

you know that from IIT madras, let us say there is some code for jump one greater than or equal to let me say let me say it is 0 x 8 something, and normal jump say this is 0 x, 0 x 4 this is the opcode and what will be stored after that opcode. So, what you store here is basically the offset from the, suppose this instruction is stored at 1000, this is 1004 just assume 1008.

This is 1016actually you go and store 16 here right. So, essentially it means that if this jump is conditioned this condition is successful the jump will happen and what you need to do is from the current program counter from the address of the current instruction which is 1000 add 16. So, this will basically take you to and this is what you will load this will this will be the address of the next instruction that you are going to execute. So, if you just store 16 if this instruction is stored at a then what will be loaded again for the next instruction will be fetched from a plus 16 which would be this correct. So, this is actually called as program counter relative addressing PC relative addressing. So, normally program counter will always point to the next instruction right. So, you may store here 12. So, it will after this is fetched your a will be pointing to this plus t12 will be this.

So, depending upon the architecture you can the compiler or the assembler will do that and what you are storing here is the offset right. So, offset relative to the program counter right. So, what is the advantage of doing this? I can load the program anywhere this is not. So, if l 1 I need not fix l 1. So, this is another one more manifestation of you know the segmentation in some sense.

So, in this 2 instructions we have introduced here the jump and what you call as J c c jump on condition. So, jGe j l we have seen lot more this. So, these are called just to recap these are called conditional branch branches, this is unconditional branch fair enough. So, this is another one more addressing mode and that is favourable in 2 ways for us to actually the compiler cannot compile unless we have PC relative to addressing. So, the architecture has to support PC relative addressing for the compiler to compile it properly otherwise if there is no PC relative addressing please note that compiler cannot compile a code at all cannot model the jump because while it is compiling it will not know where this l 1 will be actually loaded in the memory right. So, this is mandatory

this is the most important thing for that right any doubts yes.

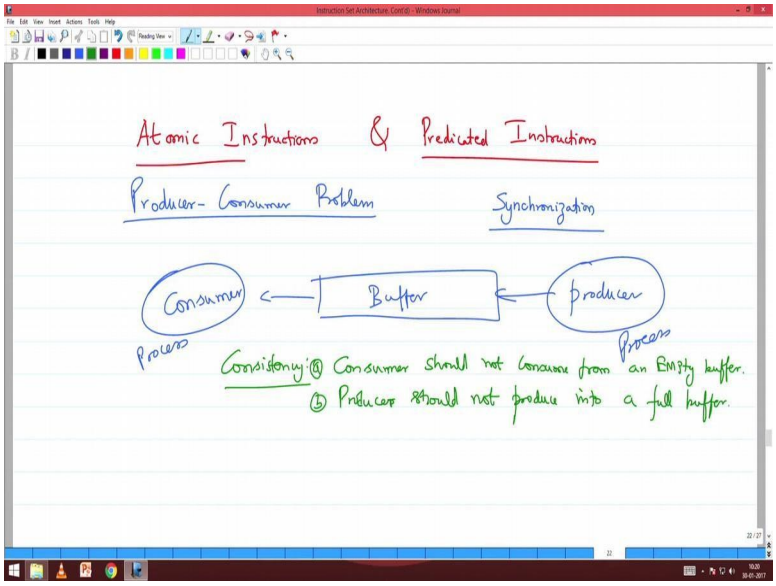
Student: (Refer Time: 06:24) j G e

Then it will be a negative value minus 16.

Student: But it will go the negative side right.

No it will not, no if they were suppose your I one sorry I 11, I 11 was here and this was 990 996? 996 994 then you will put. So, 1004 minus 10, I will put minus 10 here because your program PC will be 1004 after 1000 is fetched, so minus 10. So, I will put minus ten. So, it is PC minus 10 not just minus 10, right. So, whatever is stored in the PC that is added with the number that is in the offset. So, if that number is negative number then you go back backward jump otherwise you do a forward jump got it.

(Refer Slide Time: 07:34)



So, then we will now go in to 2 important instructions that form you know its mandatory in the modern instruction set architecture, and these are all demanded by the operating

system, those 2 classes of instructions are atomic instructions and predicated instructions. So, the x 8 6 also has these atomic instructions and predicate instructions, what do you

mean by atomic? In general where you will see this is an atomic unit basic what do you mean by basic?

Student: (Refer Time: 08:18).

Undivisible indivisible ok we cannot spit it up into smaller parts right. So, you should now say now atom contains electron, proton, neutron, now neutronic unit or protonic unit or whatever electronic unit, but it is not divisible anymore. Now why do we need such type of instruction what do we mean by an indivisible instruction what do you mean by an instruction that cannot be split right these are all questions that need to be answered.

Now, let us take one interesting problem that you will encounter in operating system left and right, whatever you touch in operating system close your eyes and touch you will have this problem and that problem is called producer consumer problem. What is a producer consumer problem right. So, when will there be any conflict at all any problem at all in world, I am asking a philosophical question when there will be a conflict I walk east you walk north south or you walk west will there be a conflict no. So, when will there be a conflict, when will there be an accident, when will there be an argument, when will there be see let us take everything.

Student: (Refer Time: 09:54).

When will there be a misunderstanding. So, if we share a resource right if there is relative grading there will be more conflict than absolute grading right correct right. So, when we share a resource there will be conflict right and that is what and the whole operating system when you study operating system in the next semester, you will see that everywhere there is some entity sharing a resource to some other entity right. So, broadly if you look at operating system there will be some part which will be doing managing the memory, there will be some part which will be managing the CPU there will be some part which will be managing the devise devises, some part which will be managing the files. So, file management process management memory management I o management all these things are there and every fellow will be touching every other fellow. So, take any 2 random entities x and some management and y in another management somewhere

they will share some resource and that makes the entire operating system complex.

So, when 2 fellow share the resource right there should be some consistent way in which they share correct right if that consistency is not maintained then the resource becomes obsolete correct. So, the way by which you will enforce that consistency is by a process called synchronization. Now I will take the simplest of the examples that you will further learn there in your operating system course, probably you spend 2 weeks learning this alone and I will give you an example of 2 fellows trying to share a resource and what do you mean by a consistency policy there, and what it means to enforce that policy I will give you an example where 2 follows are sharing a resource, and we will define what I mean by consistency and then we will see how to implement that consistency are you getting this 3 important things.

So, one of this one of the important problems that you face when you operate with their system is that there is see there is a keyboard, there is a processor. Keyboard is some million times faster than the processor sorry slower than the processor, then you have an disc which is which is say not million times, but you know considerably slower than the processor then the ram which is faster than the disc, but slower than the processor. So, you have verities of devises working and they are interacting together with each other by actually exchanging information right.

So, when there is a fast entity and a slow entity both of them transferring data between each other, the only way to see that there is no data loss is by having a shared buffer. So, there should be a shared buffer to which the producer will be consume the consumer will be consuming from this shared buffer, and the producer will be producing in to this shared buffer. If there is a feed mismatch wherever there is a speed mismatch in computer systems immediately you will see a buffer right. When you see the file systems you will learn in great detail in your operating system course, there will be something called a buffer cache.

A buffer cache is basically responsible for storing some data from the disc when data is transferred from the disc and consumed by the c p u it lies in one cache called a buffer cache. Similarly when you type on a keyboard you will see which gets stored in a

keyboard buffer and the process consumes it from the buffer right when you transfer things over the network again network will be slower than this. So, then there is a buffer. So, everywhere there is a speed mismatch between devise a and devise b there is a shared buffer.

Now, let us look at the simplest version of this problem where there is one consumer consuming in to this buffer, another consumer who sorry one producer producing in to this buffer and one consumer consuming away from this buffer. Now are you able to understand. So, this is this is one process; process means what a program in execution (Refer Time: 14:54) just note hello world dot c is a C program when you compile it a dot out is the executable program, when I press dot slash a dot out in your shell right then it becomes an executable program in execution right a program in execution then it becomes a process. So, a process is always a program in execution, please remember this for eternity very good now there is a producer process there is a consumer process. Now let us try and make a consistency here what is the rule of consistency I am always selecting can you guess what could be the rules for consistency here simple rules I am putting something in I am taking something out what is consistency.

Student: (Refer Time: 16:12).

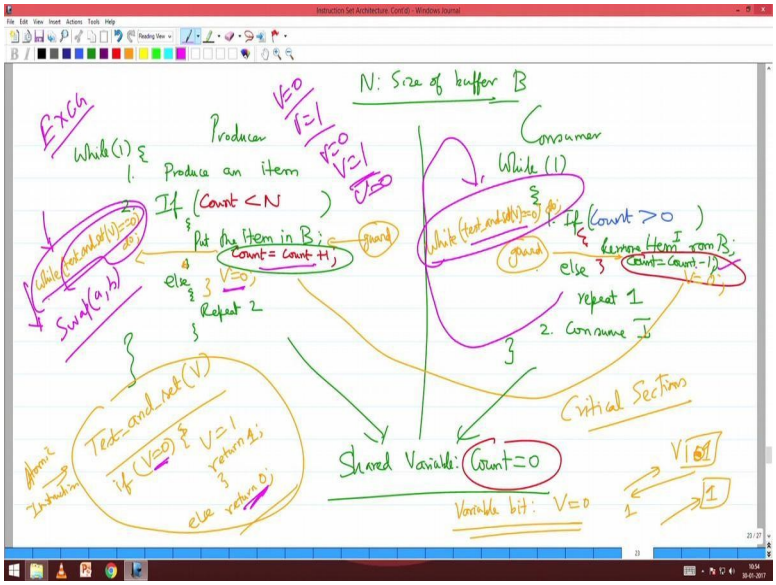
The same, but these are 2 different processes on the same processes n queueing and d queueing since this buffer is a queue. But what is consistency there when will I go what is consistency generally something which should stop me from doing something wrong right that is what we call by consistency in this context. So, what are the wrong things taken exactly beautiful excellent. So, consuming consumer should not consume from an empty buffer, this is rule a then the next rule somebody other that her.

Student: (Refer Time: 17:06).

Producer should not produce in to a full buffer correct. So, these 2 are the consistency policies, now we will write a program right we will write a program to basically ensure right that these consistency is maintained, you are able to follow very simple things. Now let us go to the let us write a program for this, now who is good who is you were

seemed to be a little radioactive to. So, can you just Kavya can you tell me some program can you dictate anybody else. So, what should be the producer program and what should be the consumer program.

(Refer Slide Time: 18:08)



The producer should know it is full consumer should know it is empty. So, how will I know? So, let us know that this is there are N buffer is of size N, N is size of 4. Now what should the producer and consumer producer, producer produce first step is produce an item then he is going to go somewhere and say put the item in let me call that buffer as B put the item in B. So, at this point he needs to do some action check what check for buffer full.

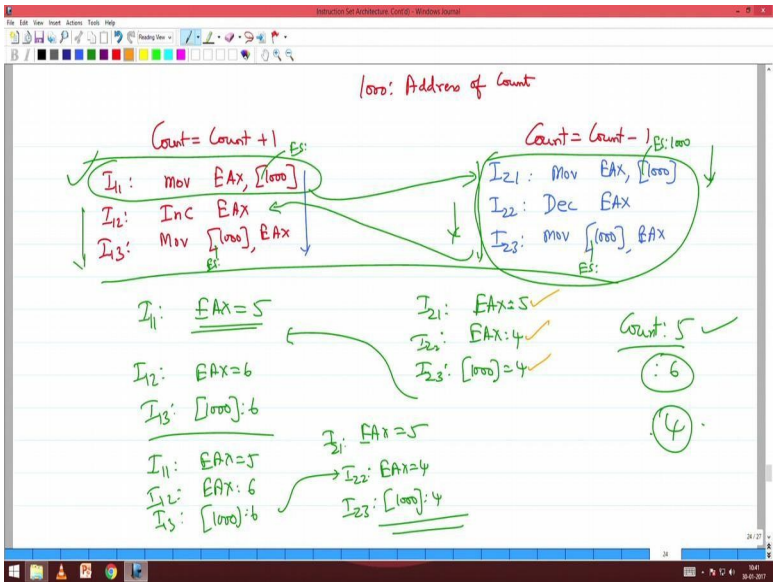
So, we can say if buffer not full put the item in B and this will be repeating the producer will be doing for his life time while one. Similarly consumer what will do while one if buffer not empty remove item form B consume remove item I from B consume I. So, this is basically what you know producer and consumer will do, and they will keep on doing it. So, I will be a producer that is why while loop infinite loop what will the infinite loop do? The producer will keep on producing item and then the next step if buffer not full he will go and put the item if buffer is full what you will do? He has to wait right if buffer not

full else you will be repeating this second statement itself else, if we to again you

will go and see check again here else repeat. So, he will be checking it again. So, this is basically the producer consumer problem.

Now, how will I how will I check buffer is not full or how will the consumer say buffer is not empty, what is the easiest way of doing it. Maintain a count of how many elements are there. So, I will have a shared variable this is where the problem comes because I have a shared variable called count which was initialized to 0 when these program. So, both these producer and consumer will share this. So, what the program now becomes a slightly when count less than N put the item in B and I will put count equal to count plus 1 and similarly I will put here when count greater than 0 remove item I from B. So, here count equal to count minus 1 there is er count equal to count plus 1 and there is a count equal to count minus 1. So, this will be inside a bracket correct. So, now, this thing works perfect count less than n means buffer is not full, count greater than 0 means buffer is not empty and whenever I remove a item I decrement count whenever I add an item I increment count right, but now count is a shared variable, it is shared by the producer process and the consumer process, the same count here is incremented it is decremented got this now.

(Refer Slide Time: 23:38)



Now, let us go to the next n c suppose I want to do count equal to count plus 1 and count

equal to count minus 1 let us take these 2 programs. Count is in memory let us say it is in

some 1000 this is where is. So, 1000 is address of count now what I will do I will say move EAX comma 1000 increment EAX move 1000 comma a x. So, let me say this is I 11, I 12, I 13 what I will do here I 21 is move EAX comma 1000 decrement. So, are you able to follow these 2 programs. Now what will happen is as we had seen in the lab class in multiple process is are executing there is no guarantee that all these 3 instructions will be executed together; what could happen is when I 11 finished, and it was about to execute I 12 the operating system may decide that enough for this process let some other process start. So, there could be a context switch which we have been talking of to this. So, and now I can finish this entire thing and then I will come back and start executing I 12 correct.

So, let me say count was count is say 5 I am producing and consuming. So, that after this action both produce production and consuming the value of count should remain at 5 right it should remain at 5 that means, when this executes once and this executes once count is count plus 1 and again it becomes count minus 1 or count is count minus 1 and then count plus 1. So, it should remain at 5 now let us see what is going to happen. So, 5 now what will happen here first I 11 executes it makes EAX equal to 5 correct. So, let us say I start here then there was a context switch then I 21 starts executing again e a it will also make EAX is equal to 5, but this EAX is different from this EAX because there is a context switch the same register is used, but the content of these registers are stored and the content of those registers are open ok, when I move from one context to the next context right.

So, EAX is 5 this is again EAX will be 5 now I 22 EAX should be decremented. So, EAX becomes 4 then I 2 3 what happens the content of 1000 actually becomes 4 then there is a context switch here, then I one 2 executes it will increment EAX, but that EAX in this context is 5. So, EAX becomes 6 and I 13 now becomes. So, 1000 actually gets a value 6 then both completes. So, at the end of this action of one production and one consumption the value of the count will be 6 rather than 5. The same thing I could demonstrate where in I 2 and executes then there is a context switch and then I 12 I 13 executes then again it goes back to I 2 to I 2 3. So, in that context what would have happened first I 21. So, EAX will be 5 then I 11 EAX will again be 5, I 12 EAX will be 6, I 1 3 1000 will be 6 and then I go back here I 2 2 EAX will be 4 for this context and

then I 2 3 1000 will be. So, I can get a value 4 also are you able to follow right yes.

Student: (Refer Time: 29:00).

Ah.

Student: (Refer Time: 29:03) explain what is context switch.

Context switch is I am executing I am a different process you are a different process when I am executing, I will have some memory allocated for me I will have some values in my registers, when I am moved out all those values need to be saved so that when I restart I should start from exactly where I left where I left all right so.

Student: The process you need to execute in order.

They are executing in order, but of between them they can go and do ping pong any time that is what I explained you in did you attend the lab class yes. So, what did I tell you in the lab class you have millions of fellows logging in to Gmail server? So, I will give you some time I will pull you out, unfortunately for me you are the red process and I pulled you exactly after you finished the and then gave you to rather he starts executing that 3 and then the control come backs comes back to you and when it comes back what is the value of a x will have you will have a 6 right you will have s 5 and then you do the increment and then you will store right.

Student: Count is independent to each of it right.

Count is a shared variable dependent on both of them I am using the same count, count is represented by 1000 did you start writing assembly programs then you should not ask these questions.

Student: (Refer Time: 30:27).

right. So, 1000 is the address where count is stored.

Student: (Refer Time: 30:33) s will be different right.

Ah d s and all, no that is what if I have a shared variable I should have one segment which is focusing on the shared variable which is shared by both correct. So, EAX f s g s one of them I can use for storing that 1000 I could even the same d s for both.

Student: (Refer Time: 31:7).

Nobody stops you it is just a selector that I need to put right. So, I can put 2 for this 2 for this, but in in essence I could have some other segment which is sharing this. So, I can say move E s colon 1000 and here also say some E s colon 1000 here also E s colon 1000 one minute. So, I could have one extra segment dedicated for sharing these you go back to the assembly language model understood.

Student: Yes

Understood it can even can happen on a register I think it can happen on m 8, m 16, m 32.

Student: The first process is 5 when you went to next process it became four.

No it is not became 1000 became a 5, then here no see look at it also started as 5 then it became 4 then you went and stored 4 into 1000.

Student: So, when we go back to the first process.

The EAX is different right the context that is what you are EAX you both of you use the same hardware resource, but when you are using you are value will be there when you are moved out this value will be stored in your process control block next time you come

again it will come back. So, this is a problem. So, I have a shared variable one fellow is

tries to increment it one fellow tries to decrement it, and then if both of them messes up then you will land up with an inconsistent value correct. So, what should you do here either when I start incrementing nobody else should do anything right it can also. So, here we have only 2 process when I start incrementing the consumer should not touch that variable when he starts decrementing I should not touch that variable. So, somehow I have a put a guard right I have to put a guard. So, that only one fellow can touch it. So, let us go back to our other example and I see now let us use this I put a guard here.

Right I put a guard here and that guard should basically you know give me this. So, we need some and who see the architecture should enable you with some instruction which can help you implement this guard. Because this part this count equal to count plus 1 and count equal to count minus 1 are very critical. If I go and start executing both together then we will land up with problem. So, this is.

So, with respect to producer and with respect to consumer there are 2 critical sections one each one for each right such that when one process is executing its critical section the other process should not execute its critical section are you able to understand when another problem one process is executing the critical section the other process should not execute its critical section. So, there should be somebody who is ensuring this. So, please understand critical section is that there are 2 processes each one of it has its own critical section, why we call it critical because if both of them execute them execute the critical section at the same time then there could be some you know mixed. So, that the whole consistency may go to.

What would have happened in the previous in the next the example that we saw just before. I will still have 5 elements in my buffer because I produce something I consume it off, but the count will be either showing 6 or the count will be either showing 4 what is the consistent state? Count should be equal to exactly the number of element that is there in the buffer, but we may get some wrong numbers that correct. So, in this case there are 2 fellows each having its own critical section, and they should share the it is called the critical section because when producer is executing its critical section if the consumer also starts executing its critical section both are different; right count equal to count plus 1 is a critical section of producer count equal to count minus 1 is the critical section of

consumer.

When this fellow is executing this other fellow tries to execute then there is a problem there is a scope for a problem. So, somehow when producer is in its critical section I should stop the consumer from going in to the critical section, for that there is an instruction called test and set of some variable V. Vis a bit in addition to count I will also share a variable bit called V. So, what we will test and set do it will go and test if V equal to 0 right if V equal to 0 then make V equal to 1 and return 1 else return 0 this is the instruction. So, let me see there is a bit V if it is 0 and they execute test and set what will happen it will make it one and return one. If this V is actually already one then I do test and set it will return 0 ok. So, what should we do here at this point while test and set of V is equal to 0 do while test and set of V is equal to 0 do or any way while test and set of V equal to 0 do right. Then I will enter here then I will make V equal to I will V equal to 0 before this point and similarly here the same thing while test and set of V is equal to 0 do and I will again make V equal to 0.

And this test and set of V is an automatic instruction that means, when you start executing no other instruction can stop it or no other instruction can execute till it finishes, and this is ensured by hardware right it will be ensured by hardware. So, initially that variable V will be 0, now I come here let producer come and start executing test and set V it will find V is 0. So, essentially it will get one; the moment it sets to one right when the consumer comes to this point you will be doing test and set V and since V is 1 you will be returning back 0 test and set will be returning back 0.

So, the consumer will be rotating in this loop till when will he be rotating in this loop. Till this fellow finishes count equal to count plus 1 and he makes V equal to 0 are you getting this. So, initially V is 0 to start with producer starts executing he comes to this point while test and set V equal to 0 what will test and set V do? It if you try and see V is equal to 0 then it will make V as one and it will return one. So, this will cross this while loop and it will start now making count equal to count plus 1, while it is trying to do count equal to count plus 1 let us assume that this consumer wants to enter he will come to test and set V, V is already 1. So, every time he is executes test and set V he will be getting only 0 because if V is not 0 I will return 0 if V equal to 1 I will be returning 0

here please understand.

So, I will be getting 0 always. So, I will be rotating on this while loop right and till when will I rotating on this while loop till this V actually becomes 0, only when V becomes 0 I will get test and set as one. So, it will be rotating. So, this fellow cannot go and cross this point till this count equal to count plus 1 is completed, and this fellow makes V as 0. Once this fellow makes V as 0 then he will do test and set V right test and set V 0. So, he will make.

So, this fellow will make V as 0 now this fellow will make V equal to one and then he will now start doing count equal to count minus 1 by the time he enters count equal to equal to count minus 1 this follows would have finished count equal to 1 plus 1 now when he is doing count equal to count minus 1 what is the value of V? 1 correct the value of V is 1 because test and set will make 1. So, V is 1 right now when this fellow is making count equal to count minus 1 the producer cannot enter here because he will come and look at while test and set V; and since V is 1 this fellow will repeatedly be returning 0. So, he will be rotating on this while loop and when can he enter? When this fellow finishes count equal to count minus 1 and then makes V equal to 0; then what will happen? V becomes 0 this fellow will execute test and set again with V as 0 then he will get one then. So, at the same point of time both count equal to count plus 1 and count equal to count minus 1 cannot execute in concurrently.

When producer is executing count equal to count plus 1 your consumer should be away from this loop it should you should be somewhere else in this loop not at the count equal to count minus 1 he cannot and when this fellow is executing count equal to count minus 1 the producer should be should cannot be within this this orange spots can be anywhere in the loop, but not in this part by that I ensure that both of them do not execute the critical section at that same time right the reason that this fellow and this fellow can succeed this while loop is not possible because test and set is a atomic instruction if test and set starts executing it will finish then only the next instruction it cannot be interrupted like your count equal to count plus 1 correct. So, that is the reason that is why that is why the hardware need to implement it in an atomic way meaning one cycle it should finish execution are you getting this. So, atomic instructions form a major part of

you know process synchronization or any synchronization that will be doing in a operating system. So, it is very important that you understand this instruction right we will have. So, there is another like test and set there is another atomic instruction called swap a comma b which can exchange the value of a, with b in one cycle atomically.

The same thing here V is exchanged with 0 and 1 there you can exchange it. So, swap can also be used for synchronization right you can easily think of a way by which you can substitute swap instead of test and set right. So, this is very very important. So, if you go to Intel manual you will see swap or exchange e x EXCG I think EXCG or something exchange or swap and then EXCHG I think something like that one thing and test and set right. So, this is these are atomic instructions for synchronization and this forms the crucial part of your operating systems implementation.

Thank you very much.