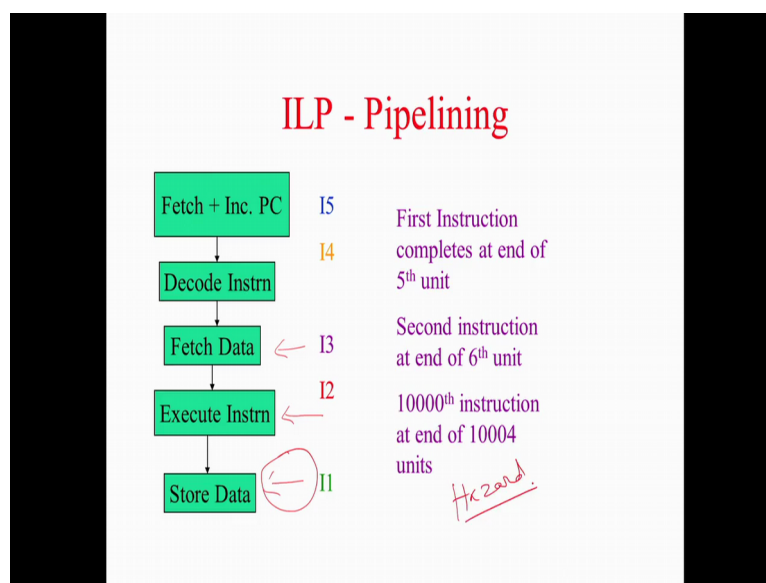


**Computer Organization and Architecture**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 18**  
**Data Hazards**

[FL] start. Good morning. So, last class we saw about pipelining and. So, this was the figure where we stopped with respect to pipelining.

(Refer Slide Time: 00:28)



So, what we see here is that five instructions can be executed concurrently and each instruction is in a different stage of a execution of the instruction. So, there are five stages as we have seen here. So, every instruction is in a different stage now the question remains the 2 questions here. Now question number one is it possible to have such a free flow is there something that will hinder. Question number 2 is that will every execution take the same time for example, we have seen floating point addition floating point multiplication to a large extent you know our Wallace tree multiplier addition, they all take almost same time, but there are operations like square root of a floating point number you know  $\sin x \cos x$  there are some lot of functions that are implemented divisions for example, we will take much more time than a normal operation. So, the assumption that every instruction will take the same time is incorrect. So, that is also something that we need to keep in mind.

Keeping all these thing mind what you will now do is that you will see how to go away from

this ideal case right the ideal case is that the instructions can flow what are the things that can stop a free flow of instruction, what are the things that could stop I 2 to flow immediately after I 1 is there some conditions which will stop I K to immediately follow I K minus one right. So, we will look at those details and finally, come out with what we call as the super scalar architecture.

So, what how to build a super scalar architecture what is inside a super scalar architecture is what we will finally, see. Now before going forward right why do we call some architecture as super scalar what do you mean by super scalar there. Now we will see this architecture can also do more than one instruction per cycle. So, the cycles per instruction CPI we call cycle per instruction is less than one. Number of cycles average number of cycles I spent per instruction if it is less than one then; that means, I am doing I am a super scalar architecture what you see here is that every instruction one unit I finished one instruction right. So, what it seen pipelining in 10,000th instruction comes at 1004 10004 unit the first instruction finish at fifth unit, second instruction at sixth unit and so on. So, every unit I get one instruction what is that unit basically cycle right.

Every cycle I get one instruction. So, cycle per instruction is one this is not super scalar can I get something like CPI is less than one; then you become a super scalar architecture. So, let us that means, what there should be you know multiple issues we will come to that. So, how do you build such an architecture and that is very important for you to understand, because modern architectures today are super scalar architecture right. So, let us hope. So, we all saw this slides that they.

Now, we will go to.

(Refer Slide Time: 04:04)

**ILP Continues....**

- **Data Hazards**

*I<sub>1</sub> - LOAD [R2 + 10], R1 // Loads into R1*  
*I<sub>2</sub> - ADD R3, R1, R2 // R3 = R1 + R2*

- This is the "Read After Write (RAW)" Data Hazard for R1

*I<sub>1</sub> - LD [R2+10], R1 ← R1 is written*  
*I<sub>2</sub> - ADD R3, R1, R2 ← R1 is Read, R3 is written, R2 is read*  
*I<sub>3</sub> - LD [R2 + 14], R1 ← R1 is written*  
*I<sub>4</sub> - ADD R12, R1, R2 ← R1 is read, R2 is read, R12 is written*

- This shows the **WAW** for R1 and **WAR** for R12

*Stalled*

Now, ILP is called instruction level parallelism pipelining is an example of instruction level parallelism because I execute more than one instruction at the same point of time that is why we call it instruction level parallelism. Now let us see what will happen that will hinder the free flow of instructions right and that is what we need to address now. Let us take these 2 instructions load there is a small variation in syntax load R 2 plus ten comma R 1; that means, it the content of R 2 plus 10 is loaded in to R 1 right we will have a small change in syntax that is. So, R 1 gets the content of load R 2 plus R 2 plus 10. So, from memory I am loading in to a register. Now what add does it reads the value of R 1 and R 2 then it sees R 3. Now unless load completes I cannot fetch the data please note let us go back to our pipelining here before the pervious instruction finishes execution I cannot fetch the data for the next instruction.

Before the previous instruction finishes that is when the previous instruction stores back the value in to R 3 then only or R 2 then only this the fetch data of the next instruction can happen. So, the next instruction cannot follow the previous instruction because there is some dependency this dependency is called a hazard. Now we will go an look at this hazard in a now what happens here now I see R 1 is being returned here returned in to and unless it finishes the writing I cannot read that value if I read the value before this finishes writing I will get some previous value of R 1 and that is incorrect.

When I am executing the program one after another, this add needs this value of R 1. So, I

cannot fetch the data for this add unless this instruction finishes execution are you able to understand this. So, if this is I 1 and this is I 2, this I 2 cannot follow I 1 the I 2 will be fetched it will be decoded then I have to wait for I 1 to go and finish the results store back the results then only I can fetch back the fetch the data for this.

So, 2 cycles this pipeline is stalled s t a l l e d; this stalling is because there is a hazard there is something undecidable that is why I call it as an hazard and this hazard is because of what? This hazard is because of data right that fellow is you go into dump some data which I need since that fellow has not finished storing the data I cannot fetch the data because the latest data is in the hands of the previous instruction correct. So, I am stalling this pipeline because there is a data dependency hence this hazard is called a data followed yes or no. Now this hazard is also called read after write hazard I am reading add is reading after load write wrote in to this right. So, this hazard is also called as raw hazard or read after write hazard followed. Now there will be others type of hazards we will see as we proceed. So, this is read after write hazard now let us see what is happening here R 1 let us take the next example R 1 is written R 1 is read R 3 is written R 12 is read 3 thing R 1 is read, R 3 is written and R 12 is read here again R 1 is written here again R 1 is read R 2 is read R 12 is written ok.

You are seeing this now between one and 2 there is a read after write hazard because R 1 is read and R 1 is written. So, there is read after write hazard between one and 2 let me call it I 1, I 2, I 3, I 4 R between 3 and 1 please note that there is something called write after write hazard. Why is this a problem right if I have what I mean by a write after write hazard; for some reason since I have now what I want to have super scalar architecture; that means, more than one instruction should finish at the same point of time, when I am constructing such an architecture what happens if I 3 finishes before I 1 if I 4 is going to follow I 3 I 1, what will I 4 get is the value of I 1 and not the value of I 3 right. So, when I start executing more than one instruction in parallel right then what happens I have to have some control over how the instructions complete because I allow all the instructions to execute.

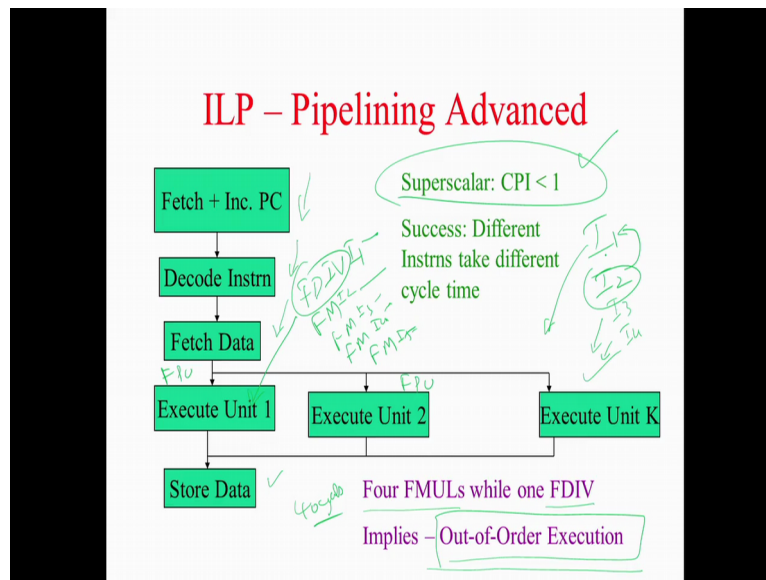
Some reason we will come out how why that possibility can be if I 3 finishes before I 1 and then I 1 finishes last, I 4 will get I one's R 1 and not I three's R 1 you got this. So, that is why we call it as a write after write because I 1 is also writing I 3 also writing, we call it as a write after write hazard. Similarly there is also a write after read hazard between I 2 and I 4 because R 12 is re read here while R 1 is R 12 is returned here. So, here write after read there is a write after read here R 12 is read here while R 12 is written here why again this is a

problem because if I 4 finishes before I 2 or I 4 finishes before I 2 starts executing because anything we are we are introducing parallelism here and I 4 finishes before I 2 then what happens I 2 will get the R 12 of I 4, but I 2 should get the R 12 of some instruction before it you are getting this.

So, the moment I go and say that I will now start I want a super scalar architecture where the number of instructions I finish at the say at they single cycle is less than one; that means, I am pumping more instructions in to the system, and when I pump more instructions in to the system then what will happen all these instructions can start executing concurrently and there can be some who finishes first and who finishes last I do not have a control over if I do not have proper control, then what will happen I will land up with these type of data hazards.

And the data hazards are read after write after write after read. If I have a normal pipeline that is one instruction after another instruction I do not want a super scalar then the only problem that will come is read after write, but if I have a super scalar pipeline then I could have read after write after write and write after read all the 3 possible data hazards. Are you able to follow yes or no? Now we will take a super scalar architecture construction and show how these hazards can come and how the architecture handles it right how the architecture itself handles it. So, the compiler can be dumb compiler can need not take care of all these all these hazards, the compiler just compiles and leaves it gets it just gives us sequential program. The architecture there is lot of hardware that is built inside the architecture which shall take care all of these data dependencies what is that architecture or how can we build one these are things that we are going to discuss.

(Refer Slide Time: 13:49)



Are you able to follow fair enough good now? So, this is our architecture we want to build a super scalar architecture. So, CPI should be less than one, I want CPI should be less than one here and the main issue here is that I can fetch more than one instruction at the same time, I can decode more than one instruction at the same time, I can go and fetch data for more than one instruction at the same time all this I can do then I will go and execute I will have a multiple execution in it instead of one adder I will have 4 adders correct it is a one multiplier it would have 5 multipliers. So, I will have multiple execution units.

So, multiple instructions can execute at a same time and multiple is instructions can store back that data at the same time if I am going to enable all these things then I can achieve this CPI less than and the more important thing is different instructions will take different amount of cycles to complete. So, then I can basically make each one of these execution units as pipeline so that I could pump more and more data as in when it comes. So, this is also very interesting part that for example, I can do 4 floating point multiplication while one floating division occurs.

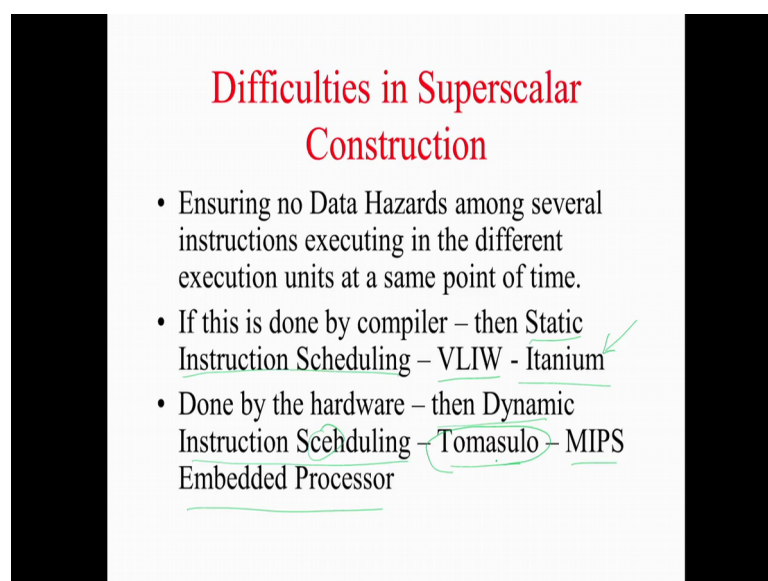
And that is where you will land up with what we call as the out of order execution. So, suppose I have one floating point division followed by 4 floating point multiplications. So, I have 2 floating point units let me say FPU 1 and FPU 2 are 2 floating point units then what will happen the FPU will start executing here it will take say 40 cycles for example, while the f mul which is following let me say this is I 1 I 2 I 3 I 4 I 5 your I 2 will come and start

executing it will finish off in 4 seconds.

So, actually I 2 completes before I 1 right I 2 actually completes before I 1 completes then I 3 will start executing it will also conclude you know it will also complete before I 1 completes. I 4 will start executing it will also complete before I 1 and I five and I 1 will complete. So, there are at least I 1, I 3, I 4 3 instructions which complete at before I 1 completes right. So, in that essence we are doing what we call as out of order execution out of which order there is an order given by the compiler what did the compiler do? It gave one FD which we call it as I 1 followed by I 2 I 3 I 4 I 5 those are all floating point multipliers multiplication instructions. Now what has happened here FD starts executing it takes 40 cycles before FD could complete it is execution which was I 1 completes it is execution I 2 completed I 3 completed I 4 completed and I 5 also completed when I 1 completed.

So, essentially it is not that the previous instruction completes and then the next instruction completes and then. So, the way I am executing is basically out of order which order out of order of the order that was actually given by the compiler are you able to follow. So, this type of an architecture you know this type of an architecture can basically you know get the super scalar behaviour. So, if you want the super scalar behaviour we need this type of a pipeline environment is it fine.

(Refer Slide Time: 17:29)



**Difficulties in Superscalar Construction**

- Ensuring no Data Hazards among several instructions executing in the different execution units at a same point of time.
- If this is done by compiler – then Static Instruction Scheduling – VLIW - Itanium
- Done by the hardware – then Dynamic Instruction Scheduling – Tomasulo – MIPS Embedded Processor

So, what are the difficulties in the super scalar construction? Difficulty one is that I have to ensure no data hazards there will be several instructions that are executing I have to see that

there are no data hazard meaning if there are data hazards I need to handle. Now what I mean by handling data hazard? I will say oh if I 1, I 2; I 2 depends upon I 1 then I will say I 1 I just keep I 2 waiting till I 1 finishes if. So, let me say I 2 I 1, I 2, I 3, I 4, I 1 depends upon I 1 I 2 depends upon I 1 while I 3 and I 4 are independent I now say I allow I 2 to execute and then I 3 and I 4 to execute while I 2 will be still waiting for it right right. So, this is why I again say this is out of order execution because I 3 executes even before I 2 is executed because I 2 is waiting for I 1. So, I 3 and I 4 does not have any dependency it can start executing.

That is how I am getting super scalar processor that we have explained. Now the point is the hardware basically says I 1 now you go and execute I 2 you wait I 3 now you can execute I 4 you can execute. So, what it means this hardware is responsible for allowing an instruction to execute or stopping an instruction from not executing. So, the hardware does instruction scheduling what do you mean by scheduling? Scheduling a class means this class will happen at this point of time scheduling and instruction means this instruction will execute at this point of time are you able to understand what I am saying. So, what is the hardware doing it now schedules the instruction.

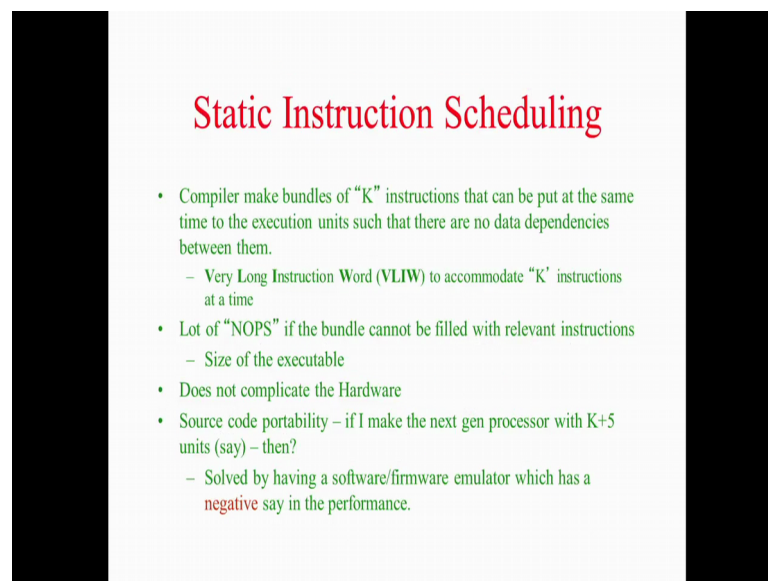
When it is scheduling the instruction when the process when it is a process when the program is in execution it does the scheduling, that is why we call it as dynamic instruction scheduling this entire process is called dynamic instruction scheduling, because this happens when the program is in execution. The hardware sees five or six instructions coming now it analyses the dependency between this instructions dependent instructions it makes them wait independent instructions it start it allows to execute and all these things are done by the hardware the compiler need not care anything about it and so, this type of an instruction scheduling since it is done by the hardware at the time of execution this is called dynamic instruction scheduling are you able to follow any doubts in this.

Now, the other aspect is that this is compiler is heaven compiler does not bother about the architecture the other aspect is that if the compiler does the scheduling. So, compiler sees 10 it groups takes ten instructions it analyses whether there are dependencies if they have no dependencies, then it allows it makes a packet and say now you go and execute right. If the compiler actually does this type of scheduling then it is called static instruction scheduling. So, there are some architectures which we call as VLIW very long instruction word very long instruction word Itanium is one such example of a processor which supported VLIW where static instruction scheduling is done.



In our case whatever we are going to see now this entire thing is done dynamically and for example, MIPS embedded processor or arm embedded processor does have you know dynamic instruction scheduling sorry for the spelling typo and we will learn a technique called tomasulos technique; tomasulo is the name of the scientist who found it. So, we will do this tomasulo technique of constructing an hardware which will do dynamic instruction scheduling meaning it will be responsible for handling all the data hazards namely raw war and WAW followed yes or no.

(Refer Slide Time: 21:55)



**Static Instruction Scheduling**

- Compiler make bundles of “K” instructions that can be put at the same time to the execution units such that there are no data dependencies between them.
  - Very Long Instruction Word (VLIW) to accommodate “K’ instructions at a time
- Lot of “NOPS” if the bundle cannot be filled with relevant instructions
  - Size of the executable
- Does not complicate the Hardware
- Source code portability – if I make the next gen processor with K+5 units (say) – then?
  - Solved by having a software/firmware emulator which has a negative say in the performance.

Before we proceed to this we will have a very very short introduction to static instruction scheduling.

Static instruction scheduling is done by the compiler, that is why the word static is done when the program is not in execution that is why the word static instruction scheduling. So, what will the compiler see it will take a bunch? So, suppose I have K execution units not all the execution units may be symmetric. So, let us go back to this slide I have K execution units not every fellow can do integer addition there can be 4 or 5 who will do integer addition, there can be some 4 or 5 which will do floating point, there can be some who is doing s I m d instructions right now the compiler we will know what the architecture is it knows that there are K instruction K units and each of this is capable of doing this, then it will now go if after it actually comp it compiles and makes the machine language program.

Now it will go and look at this machine language and then it will find machine language

program and it will see if I could assemble K instructions such they are there is no data dependency between them, and these K instruction should adhere to the K units. So, the first instruction if the first unit is integer it should be an integer instruction; the second unit if it is going to be like here right if the second unit is going to be a floating point unit it should be a floating point instruction like that for every unit it will find if there is an instruction that could be executed it will find and the K instructions that I am going to send at one shot none of these instruction should be one dependent there should be no data dependency.

Say it will make one bundle like this; obviously, it cannot find a bundle where all its instructions are productive like I may not find a bundle where I have a floating point and an integer instruction also. So, it may not be successful in finding a bundle with K instructions, each matching with the corresponding execution unit and there is no data dependency it becomes extremely complex are you able to follow right. So, what it does it takes say some R units R less than K, R instructions it schedules for the remaining fellows he will put NOPS n o p s. So, any instruction set architecture in the world we will have something called n o p no operation right.

Even if you go into the Intel manual, you will find there is something called n o p no operation. Why do you need such an instruction at all why do you need a nop this is one example I will show you many more examples, but this is one example in the case of static instruction scheduling if I do not find enough instruction to fill all the execution unit for those units corresponding to those units I will put nop.

So, what is. So, finally, what is static instruction what is static instruction scheduling I make a very long instruction word to accommodate K instructions, and if I do not get K instructions that such that each instruction you know adhere to the word to the execution unit and adheres to the execution unit on which it needs to execute and also that there is no data dependency if I am not able to get those K instruction or one bundle of K instruction then I put it put NOPS. Now as you see the hardware is not complex because the hardware is not going to the dynamic instruction scheduling, hardware I just takes one bundle executes it and throws and next bundle executes it and throws it.

So, the hardware becomes extremely simple, but the compiler the has a night mare not a very big night mare, but it is it is indeed a night mare. So, that is one very important thing. So, what happens is that many times I do not find K instructions in a bundle. So, I land up with

adding lot of nops. So, the size of the execution you uh the executable program the final executable program actually becomes very fat. So, that is one major drawback of this static instruction scheduling. The second important draw of this static instruction scheduling is the portability of the code from one architecture to another architecture. Tomorrow if I put  $K + 1$  execution units or  $2K$  execution units then this one this executable will not run on that. So, I have to go take back from the machine code again I have to do bundling again right. So, so the portability is going to be a major issue right and many times what happens is that they use some software firmware emulator to handle this portability issue.

I move from core which had earlier  $K$  execution unit to  $K + 2$  execution unit, then every bundle that I have generated should have either 2 NOPS or it needs to be rearranged. So, I cannot basically every time a new Itanium sorry a new VLA architecture is announced even in the same series I cannot go out and keep on compiling new versions and say for this you use this version then it becomes commercially unviable.

So, there that was one major drawback in this static instruction scheduling; there were some software firmware emulator solution which basically did not work that effectively today we do not see many of these v l a VLA machines. So, we will not put lot of time on trying to see that architecture, but basically we will have at an introductory of this nature we will have an introduction. Now there is something much more complex that happens in static instruction scheduling and that is this.

(Refer Slide Time: 28:11)

## Thorn in the Flesh for Static Instruction Scheduling

- The famous “Memory Aliasing Problem”
  - ST [R1+20], R2 //Store R2 into
  - LD R3, [R4+40] //Load R3 with*Writing into R1+20  
Loading from R4+40*
- This implies a RAW if  $(R1 + 20 = R4 + 40)$  and this cannot be detected at compile time
- Such combinations of memory operations are not put in same bundle and memory operations are strictly scheduled in program order.

This problem what I am going to describe now is called memory aliasing and this is something which your static instruction scheduling cannot handle. Now what is happening here let us carefully look into now I am trying to store a value of or store the value of or to on to the address R 1 plus 20, and I am reading the value of R 4 plus 40 and storing it in l 3.

So, I am writing in to R 1 plus 20 and I am reading from R 4 plus 40. For some reason if R 1 plus 20 is equal to R 4 plus 40 note that this is your hazard and this hazard the compiler cannot detect it why the compiler cannot detect because at the time of compilation it does not know what is the value of R 1 and what is the value of R 4 correct this is going to be dynamically determined. So, if I see a load and store load following a store it can potentially be a read after write if those 2 addresses become the same this is called the memory aliasing problem.

What the say let R 1 plus 20 be 100. So, hundred can be approved as R 1 plus 20 alias or 4 plus 40 both will give me 100, but at this I cannot find out at the time of compilation are you getting this. So, since I cannot find out at the time of compilation if I see a load and store typically like this what will happen the architecture has to schedule it one after another. So, they cannot put it in the bundle I cannot put your load and store even though there may be no connections at all see this is using some R 1 R 2 register this is using R 3 R 4 register, there is absolutely no connection right as in the in the prime of a c v u, statically if I if I you know look at these 2 instructions I do not find any connection. Dynamically they could land up

right if  $R_1 + 20$  equal to  $R_4 + 40$  which I can find out at the time of compilation.

So, when I have a load and store such that load follows store then better I put them in 2 different packs; that means, if there are a lot of instructions that are dependent upon this store and load those cannot be packed together here. So, load store essentially becomes sequential and that is very very important about static instruction scheduling this is a thorn in the flesh and you will see a lot of load and store in your program because ultimately you have to read from the memory for every variable and write back to the memory. So, when you see a lot of load and store together this is not a by the static instruction schedule if I am going to do static instruction scheduling this should happen one after another.

(Refer Slide Time: 31:57)

The slide is titled "Dynamic Instruction Scheduling" in red. It contains a bulleted list in green: "The data hazards are handled by the hardware". Below this, there are two red handwritten notes: "- RAW using Operand Forwarding Technique" and "- WAR and WAW using Register Renaming Technique". A red arrow points from the "RAW" note to a handwritten box containing "True dependencies". Another red arrow points from the "WAR and WAW" note to a handwritten box containing "Named dependencies" and "False" (with a checkmark).

Now, what you will be covering in our thing is that dynamic instruction scheduling where in the data hazards are handled by the hardware right they are not handled by the software. So, their memory aliasing type of issues will not all be handled by hardware and that is why the hardware becomes a little more complex.

Now there are 2 types of data hazards. So, we will have something called a raw hazard; the raw hazard is basically solved using the operand forwarding technique operand forwarding while the war and WAW hazard will be solved using something called register renaming technique. So, these 2 techniques we will have to implement it in hardware and show that it is working correct. Now also understand this raw we will see that it is actually a dependency, it is a true dependency while this war and WAW are not are virtual dependencies

or whatever you call it as named dependencies, you can even call as false dependencies we will show you why.

So, the hardware that we have going to deal with into tomorrow's class right is one that will solve the of raw and war and WAW hazard completely we will spend 45 minutes to understand this, for the raw we will be using something called an operand forwarding technique while for the war and WAW we will be something called register renaming technique. So, in computer literature of computer architecture there are 2 things one is true dependency another is named dependency, in true dependency there is actually a dependency and I cannot solve the true dependency problem by any means I have to wait right.

So, if there is a if the previous instruction is holding that data that I need I have to wait for it to complete. So, that is why we call it as a true dependency we will elaborate that a little more while the war and WAW are virtual dependencies if I go and go some jugglery, then those these things will disappear your war and WAW can disappear will you something called register renaming that need to do it, both these together we will cover in tomorrow's class.