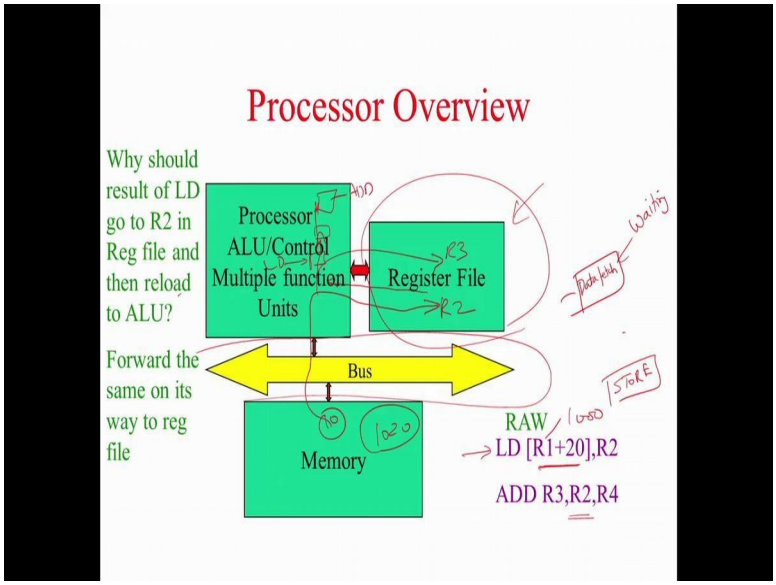
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 19
Lab 2: Instruction Scheduling - Static and Dynamic

(Refer Slide Time: 00:18)



All the peripherals are connected to the CPU through this bus. So, basically for this bus there will be a bus master or what we call as bus control logic. So, if I want to access the bus as a CPU, I basically give a request to the bus control logic then the bus control logic will resolve of all the request I will grant if you make a request it will grant you the request; that means, you become the master of the bus and there will be somebody who is receiving that will be the slave of the bus or you could have multiple slaves if you are trying to send data to multiple units right. So, the bus plays a very crucial part as we see when we move from this is a single core you know

description when we move from a single core to multi core description the bus becomes much and much complex. So, lot of lot of activities will happen over the bus. So, this is a structure of a simple processor any doubt in that now.

Now, let us look at this read after write I am writing in to R 2 the first load institution is

writing into R 2, one minute the first institution is writing into R 2 the second institution is reading from R 2. So, this is a read after write right. So, what is the first instruction do it will takes from the memory address let R 1 be some thousand. So, from thousand twenty it will take some value let it be some value let us call that value as ten this will read this ten and it will store it in the register this is what the load does and what will add do it will read from the register file and perform the addition and then store back to register file. So, the load instruction takes the value from thousand twenty assuming R 1 is storing thousand is takes the value from thousand twenty let me say it is ten and then stores it in R 2 the next instruction that is following it right we will read from R 2 and it will do the addition here and write back the result on to R 3 right. So, the data in the first instruction is actually moving from the memory through the CPU to the register file and in the next instruction it is basically taken from the register file and operated upon and the results are stored back in the register.

So, for this data ten it goes to the register file and comes goes to the register file through the CPU and then comes back to the CPU for next processing when load when load is in the store result stage when load is actually storing the result in R 2 add will be waiting in where in the data fetch stage correct because it needs the value of R 2 to go and do the next step. So, it has to wait how it is going to wait we will see, but the hardware will make it wait in the data fetch stage while load is in the store result stage the add will be waiting in the data fetch stage correct now where is this where it will where is the data fetched inside the CPU right the pipeline is inside the CPU. So, somewhere load is there here while add is waiting here correct now why should. So, the question now is why should the result of why should result of l d load go to R 2 in the register and then come back to the ALE in its way from you know the memory to the register file cannot I take the value keep it why should I spend extra time for it to go to R 2 and then again fetch back from this when the when ten is moving from the memory to the register file through the ALU I can just basically take a copy of this and give it back cannot I right.

Right and this particular concept is called operand forwarding. So, when I am taking the value of R 2 and I am going to store it in a register file of course, it has to go to the register file in that path since it is going through ALU I can basically also take that copy of that value and give it to add are you able to follow. So, very quickly I will tell you

again there is a load instruction which is reading from thousand twenty and loading in to R 2 there is an add institution that immediately follows it which uses the value of R two. So, in our pipeline this add instruction should be waiting inside the CPU in the data fetch stage while load is completing it is store load is completing it is operation in the in the store stage of load what it should do it should take the value in thousand twenty in every time and it will be storing it in R 2 and that moment is through the ALU right in the semester when you actually did the hardware how did the move institution work it came through the ALU and went to the register file do you remember still when you constructed this, right.

When you want to initialize some value it actually moved through the ALU it did not go. So, it came through the CPU and then moved the instruction comes to the CPU right and then you move it inside. So, the point is when the data is going to the register file I can just take that copy of the data and then I can forward it to the waiting instruction rather than I writing in register file and asking this add to go and access the register file and take it there are 2 important things that we save here one thing is accessing the register will also some amount of time the second thing is this store has to load has to complete the load has to complete writing in to R 2 before I start. So, there is again some more time elapsed in that right. So, these 2 time intervals are shortened by just doing this forward. So, this is the simple easy way of understanding operand forwarding are you able to appreciate right.

(Refer Slide Time: 07:39)

Now we will do this entire stuff here. So, you all understood register renaming sorry we all understood operand forwarding now we will do something called register re renaming right.

Now, let us take these instructions. So, there were 3 hazards that we have been talking of the RAW the WAR and WAW now what we did in the previous slide was how to handle a read after write and in the read after write of course, the add will be waiting, but it will not wait for too long we will try and shorten it is waiting period by doing operand forwarding, but the add the subsequent instruction has to wait for the data to complete that is why we call it as a true dependency right I cannot do any jugglery and get away with it on the other hand let us look at the other 2 type of data dependencies namely write after read and write after write now let us take these set of 6 institutions add R 1 R 2, R 3 store the result of R 1 in R 4 plus 50 add R 1 R 5 R 6 subtract R seven R 1 R 8 store the result of R 1 in R 4 plus 54 add R 1 R nine R ten correct now what are all the read after write there is a read after write between one and 2 because one is writing into R 1 store is reading from R 1 storing it in R 4 plus R 5, correct.

So, there is a read after write there is store is reading from R 1 and storing it in R 4 R 5.

So, this is a read after a write similarly one and 4 there is a read after write this is read

writing and this is reading and there is 1 and 5 there is a read after write 3 and 4 there is a read after write this is writing and this is reading 3 and 5 there is a read after write this is writing and this is reading read after write. So, so many dependencies are there between for R 1 within these 5 instructions now let us take write after read what are write after read between 2 and 3 there is a write after read because this is writing and this is reading store reads form R 1 and stores in R 4 plus fifty while add writes into R 1 the sum of R 5 plus R 1 between 2 and 6 there is a write after read right this is writing and this is 2 and six. So, this is reading and this is writing. So, write after read between 4 and 6 this is reading and this is writing and 5 and 6 this is reading and this is writing right. So, so many dependencies we have seen between this and write after write that you see between one and 3 this is writing to R 1 this is also writing to R 1 and one and 6 write after write and 3 and six.

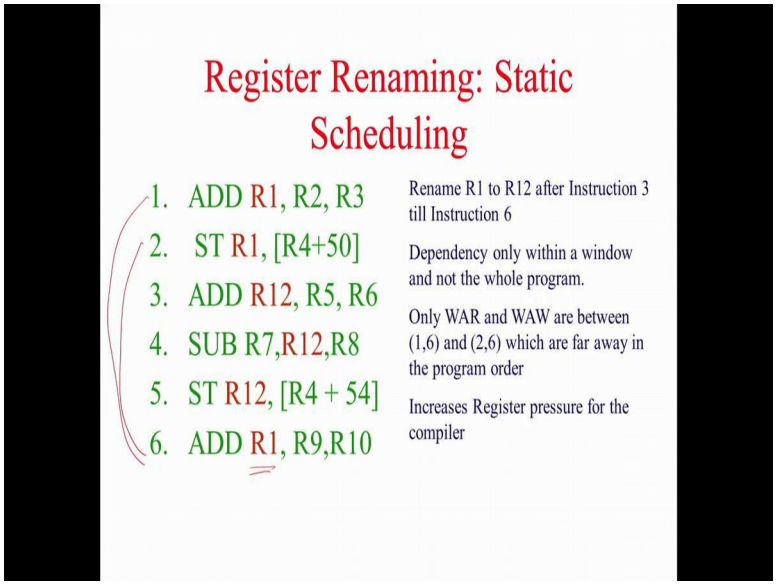
So,. So, many dependencies are there between these instructions of course, it is a cooked out example, but we can get such examples right now why are these actually I again I repeat why are these dependencies extremely important because any of these institutions can execute in any order we want Rajanikanth here right super scalar. So, I can allow every institution can work out of order. So, if I am free for example, R 2 can be an output of some floating point division integer casting. So, that floating point division will be taking so, much of time. So, this add has to wait for that. So, this store also will be waiting for R 1 to complete correct because R 2 somewhere here would be doing lot of floating point operation, but. So, so this add cannot proceed till that R 2 completes, but note here this add and this subtract can go on because R 5 R 6 R seven R eight they are all free you are getting this, right. So, these 2 will start executing even before these 2 because this is waiting for something and these 2 are independent of these and that floating point division can be it will may land up in a floating point exception right. So, it can go on forever while this fellow can start doing something else. So, all these 4 instructions can execute even before these follow touches right are you getting what I am trying to say right.

So, and these type of things have to be enabled. So, that you get that super scalar behaviour CPI is less one right. So, because of out of order if I am allowing 3 and 4 to execute before one and 2 all these hazards become extremely relevant correct right

because if one and 2 are the fellows who are going to finish last the value of R 1 would be the one that is output by this add instruction rather that this add instruction right and that I have to somehow handle it when at this point when I observe what should be the value of R 1 I should get the value of R 1 that is output by instruction number 6 and not instruction number one correct, but at the same time I would love to allow 3 4 5 6 to execute before one and 2 actually starts executing to actually save time and get my cycle per instruction less than one are you able to follow what I am trying to say right. So, so in this context all these dependencies become extremely crucial now RAW we cannot do anything about RAW the only thing we can do is operand forwarding that I explained in the previous slide now what can we do about WAR and WAW let us see this.

Suppose I go and replace please note that the life of R 1 starts at one and ends at 3 because at 3 R 1 is again rewritten with a new value right we have done that liveliness analysis still you remember the interval graph please remember that. So, the life of R 1 starts at one and it stops at 3 the life of again R 1 starts at 3 and ends at 5 because 6 again somebody is writing in to R 1.

(Refer Slide Time: 14:49)



So, what I am doing is for the other [FL] of R 1 other lives of R 1 I will go and replace it to some other value for example, what I have done is there R ones I replace it with R

twelve are you following what I am saying these R ones I replace it with R twelve now where are the WAR and WAW everything went out program is correct whatever this program is going to achieve is exactly what this program is going to achieve functionally both are equivalent when I replace this R 1 by R twelve all your WAR WAW all these hazards that were there sorry this WAR and WAW completely gone except for this six. So, the only WAR and WAW are now between 2 and sorry between 2 and 6 between 2 and 6 and sorry, sorry, 1 and 6 and 2 and 6 now this is very far off 5 is very much far away from one two. So, by the time you reach 6 right when you by the time you reach 6 these instructions should have been completed we expect it to complete.

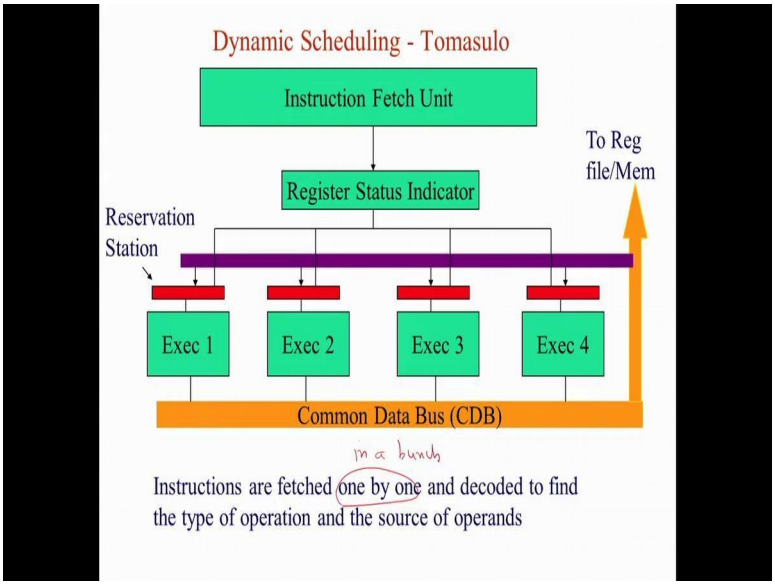
So, this will not hinder this WAR or WAW will not hinder that. So, by doing register renaming what I have done I have renamed the register from R 1 I just replace it by R twelve by doing that and how did I replace by R twelve I took all the instance of R one. So, took WAR R 1 is getting initialized it is getting initialized at 3 and till the next initialization of R 1 I go and replace all instances of R 1 with some other register name namely 3 4 and 5 I replace R 1 with R twelve the moment I do that what happens all the WAR and WAW that existed completely is gone one minute and after doing that there is only one more WAR or WAW that is between and 6 and 2 and 6 and now you see that are this is much far away the window where 6 will enter into the CPU is much far away from the windows of R 1 and R 1.

So, sorry 1 and 2 instructions 1 and 2 and so, I do not rename it because by the time this fellow comes these 2 would have completed. So, this is how we start processing the thing this I am coming to your doubt this register renaming can be done by the compiler if it is done by the compiler of course, the WAR and WAW hazards can be moved there right, but in the case of static scheduling, but in dynamic scheduling who should do the register renaming the hardware should do the register renaming and that is the thrill how does the hardware automatically do register renaming how does the hardware actually do operation forwarding right. So, because compiler is dumb as I told you it will just dump it will just take this code it will write an assembly code and give it any dependency everything is left to the hardware.

Now, how will hardware do it? So, that is a very interesting question that we will answer,

but if it going to be software and we are going to have pipelining like this then register renaming can be done by the compiler, but note that this WAR and WAW just evaporated the moment I went and rename.

(Refer Slide Time: 18:34)



So, that is why we call this data dependency as a false dependency or a named dependency right it is all because of a name nothing more now we will do this is. So, this is the you know the architecture the micro architecture which is the hardware which is going to do your dynamic scheduling which will do operand forwarding and register renaming the operand forwarding is to handle RAW hazards register renaming is to basically handle the WAR and WAW hazards it will automatically do register renaming your compiler is dumb as me and the architecture is as intelligent as you ok.

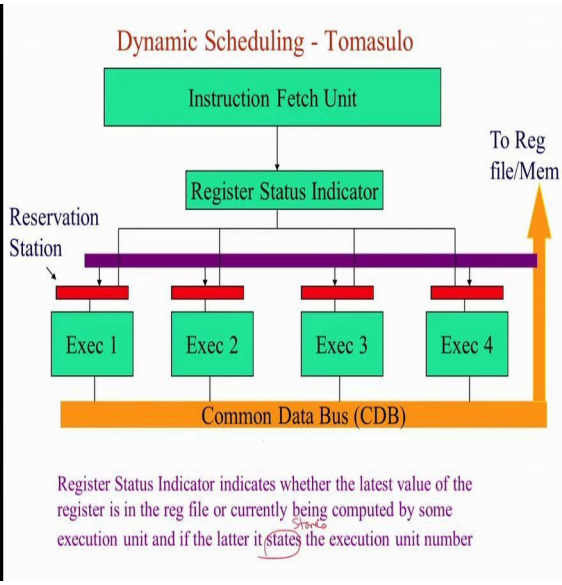
So, it will do very good functions ok now [FL]. So, how does this go right? So, there is an instruction fetch unit it has to fetch the instruction. So, what is the step first step when we are trying to execute an instruction fetch the instruction now I want super scalar right. So, I cannot fetch one instruction. So, I will fetch some n instructions I can fetch 4 instructions I can fetch 2 instructions. So, let us say we will fetch 2 instructions at a time or 4 instructions

at a time now, now let us go back to RISC and CISC I will not just teach this I will teach lot of peripheral things. So, that you know understand this if I go and

look at CISC right the; our famous by x 86 Intel inside what will happen I will have instructions that are one byte 2 instructions that are thirty 4 bytes in length arbitrary some random number right. So, I say fetch 4 instructions I do not know where that 4 instruction will start where it will end correct. So, I will go and fetch some amount of bytes and pray god it is going to give me 4 instructions, but in the case of RISC what will happen I know for sure each instruction is sixteen bit in your you know core c s t cores you have seen right each instruction was sixteen bits. So, I will if I bring forty eight bits I am sure 4 instructions are going to be there. So, that is very important. So, this type of architecture will work extremely well in when it is RISC architecture now I fetch 4 instructions now there is something called a register status indicator one by one please see the captions that are going I spent lot of time preparing this slide.

So, enjoy that my paint. So, instructions are fetched one by one and decoded to find the type of operation and the source of operations this is in a general thing. So, instructions are fetched in many. So, I in a bunch and decoded to find the type of operation and the source of operands right I will have multiple decode units inside this instruction fetch unit.

(Refer Slide Time: 21:35)

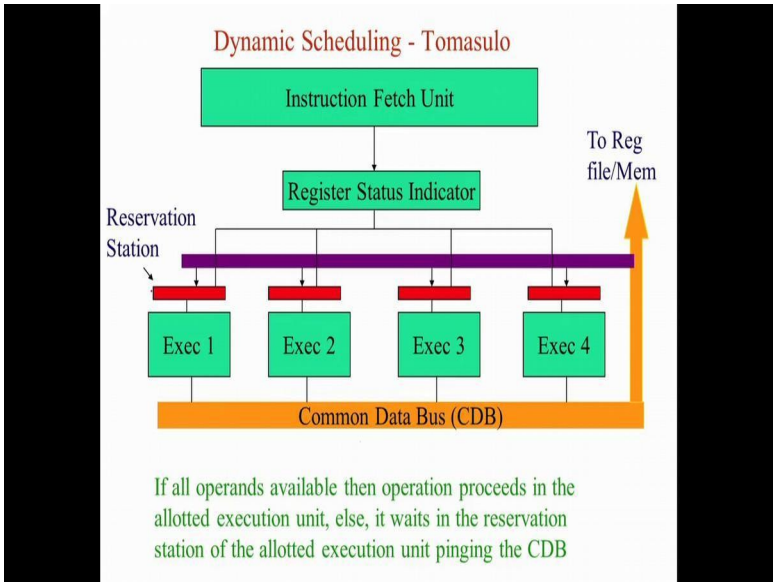


Now what will happen now there is the next thing we need to see is the next thing is I

have something you all understood what is an instruction fetch unit it will fetch many instructions and decode find out what are the operations to be done and what are the operands now there is some registers status indicator what will the register status indicator do if it is 0. So, for every register there is one entry every general purpose register right there is an entry if the latest value that I want if it is there in the register itself then I will have 0 if the latest value I want is currently being computed by one of these execution units then it will have the number of that execution unit let me repeat for every register every general purpose register like your e a x e b x like that R 1 R 2 to R 1 twenty eight etcetera for every register there will be one entry right that will not be just a bit it will be an entry right can be an integer and that entry will be 0 if the latest value I want suppose I want the latest value of R 4 if it is there in the register then that entry will be 0 if it is currently computed getting computed by some elements for example, I just add R 4 R 2 R 3 and that add is executing in execution unit 3 right.

So, in the R 4 entry the entry corresponding to R 4 I will have the value 3. So, the CPU will know when it is trying to dispatch that instruction for execution it will know for each operand each register operand where is that latest value is it there in the register itself then you go and fetch it if it is not there in the register and it is currently being computed by some unit then I know which unit has the latest value of that are you able to follow. So, that is the register status indicator after assuming you have followed this let me read out that you know magenta statement down register status indicator indicates whether the latest value of the register is in the reg file or currently being computed by some execution unit and if the later that is if it is computed by some execution unit it actually states the execution unit number it actually stores the execution unit number.

(Refer Slide Time: 24:21)



Now if all the operands are available right then the operation can proceed if some of the operands are not available then the operation then the operation has to wait and while it will it wait I will wait in the reservation station. So, I am doing I had add R 2 R 3 R 4 R 3 is not in the register file it is being computed by execution unit 4 that I will know in the register status indicator. So, I know that in add R 2 R 3 R 4 R 3 value is not available to me it is currently being computed right. So, that instruction it is an add instruction it can execute on one of these execution unit say one. So, it will come there and it will wait in this reservation station it will be waiting for whom waiting for that R 3 right I am going to give a example for this very full blown example for this, but you please understand what is going to happen here.

So, let me read this green statement if all operands are available then the operation proceeds in the allocated execution unit add I go and allocate it to execution unit one and say go and add use that execution one to do that unit to do that addition. So, I start doing that addition provided all the operands are available if one of the operand or 2 of the operands are not available then what I do I wait in the this in the reservation station I am waiting there for what the operands to become available now where are the operands if the

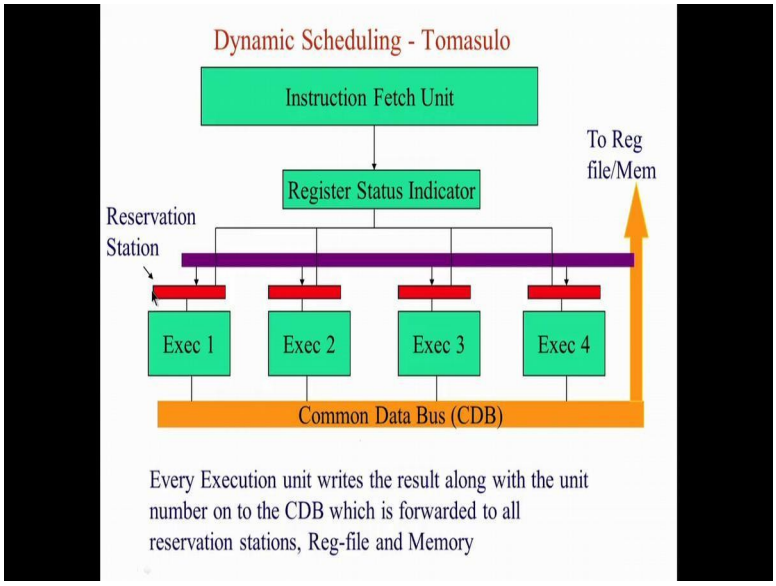
operands are not available where are the operands they are currently being computed by one of these execution units right and I know which execution unit is actually

computing my operand how will I know that the register status indicator will tell me right add R 2 R 3 R 4 R 3 is some 6 means execution unit 6 is basically computing the value of R 3. So, you wait till that 6 fellow finish right. So, I know that. So, I will come here to the reservation station and wait for the sixth fellow to finish right now. So, when the sixth fellow finishes I put till 4 let us assume till 6 he will write the result in this common data bus.

He will not only write the result, but he will also write the unit number that result will go to the register file or memory depending upon whether it is a store instruction store instruction means that go to memory other instructions let us this is a load store architecture other instructions have to write back to register. So, that data will go to the register file and the same data will be fetched back to all these reservation units. So, as the data is moving from the ALU to the register the data will be fetched fetched back thought this violet line it will also come and fetch back in to the reservation station. So, if I am waiting for execution unit 6 to finish I will be pinging this pinging write you all have done ping right it called an execution thing right have you executed the command ping yes, yes or no, no go and execute. So, find what is ping. So, it will basically find out whether an I p address is alive or not. So, you send some packets in the network to find out whether an I p address is alive or not like. So, I will be watching eagerly.

The common data base and if the; and what will happen if 6 you will write the result and you will also put the unit number, so, I will be just watching for the unit number 6 6 6 6 has come. So, I will grab it and I will start executing. So, the result did not go to the register file and then come to me as it was going to the register file it also came and dropped in to my execution unit.

(Refer Slide Time: 28:55)



So, this is operand forwarding followed really now every execution unit writes the result along with the unit number on to the data bus which is forwarded to all reservation station through this violet bus and also through the register file manager. So, if I am waiting for a result that result will come to me when I am waiting through the violet bus are you able to follow right now let now we will go on to an animation we will just spend ten minutes nine fifty 5 I will leave you. So, so to sum up there is an instruction fetch unit the instruction fetch unit will fetch multiple instructions decode them then there is a register status indicator which tells; where is the latest value of the register lies in; is it in the register itself or is it being currently computed by some execution unit. So, when I am proceeding to my computation communication computation I will know for sure whether the data is already available or I should buy it if I need to wait I go and wait in the reservation station and every unit when it finishes it puts it is unit number and also the result and I know from which unit my result is going to come and I am waiting there.

So, when they that unit actually dumps the result on the common data bus it will come to me through the violet bus and I will take it start proceeding. So, operand forwarding is happening and automatically your instruction gets stalled if there is a data that is currently being computed and it is not available both of the things are happening when there is a data

which is currently computed getting computed by a previous instruction which I need automatically I am stalled and when it completes the data gets forwarded to

me automatically are you able to understand this architecture it is very simple very nice architecture.