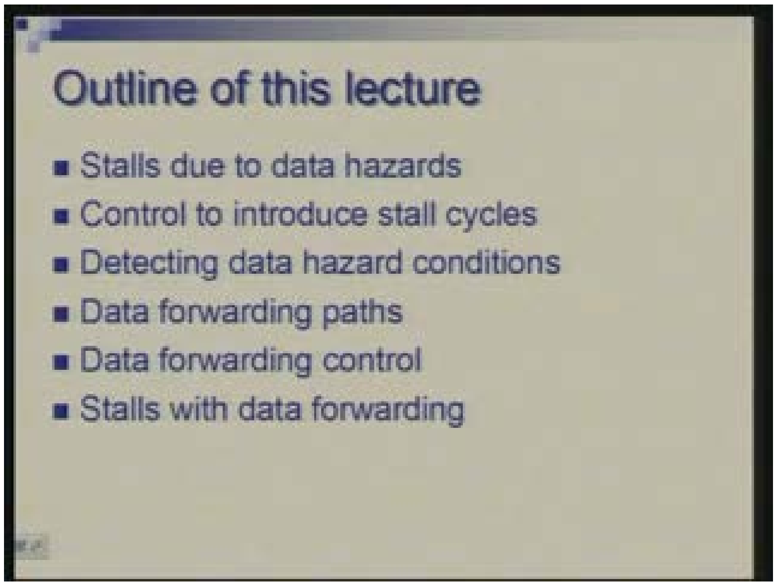
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 26
Pipelined Processor Design: Handling Data Hazards

There are three types of hazards which a pipeline design may face; these are structural hazards, data hazards and control hazards. We started with the design whereby the processor of design itself we took care of structural hazards that means enough resources were introduced so that there are no structural hazards. But the problem of data hazards and control hazards is still there in the design we have talked of.

Today we will see how to handle data hazards in particular. Initially we will see that how we can introduce no ops or bubbles in the pipeline so that the functionality remains correct and then we try to introduce techniques of data forwarding and improve the situation. So first we will examine the question of stalling the pipeline in detail; for different types of instruction sequences what is the number of stall cycles required and which instruction or in which cycle do we need to introduce the stall.

(Refer Slide Time: 00:02:02)



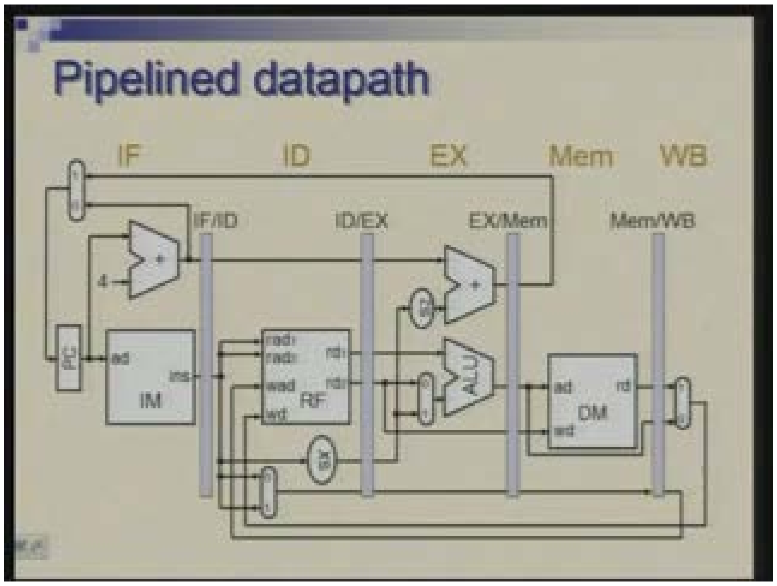
Outline of this lecture

- Stalls due to data hazards
- Control to introduce stall cycles
- Detecting data hazard conditions
- Data forwarding paths
- Data forwarding control
- Stalls with data forwarding

We will see what additional control circuitry is required to actually cause these stalls to be introduced then we will define the conditions which need to be detected in order to exercise this control. After having done that we will talk of data forwarding; that is a technique where you sort of bypass the register file and pass on the data from one instruction to other instruction as early as possible so that requires additional paths; we will see what these are.

Once you introduce additional paths you need additional control and that is what we will see next. Finally we will see that in spite of introducing data forwarding you may sometime need to stall so under those cases how stalling is done.

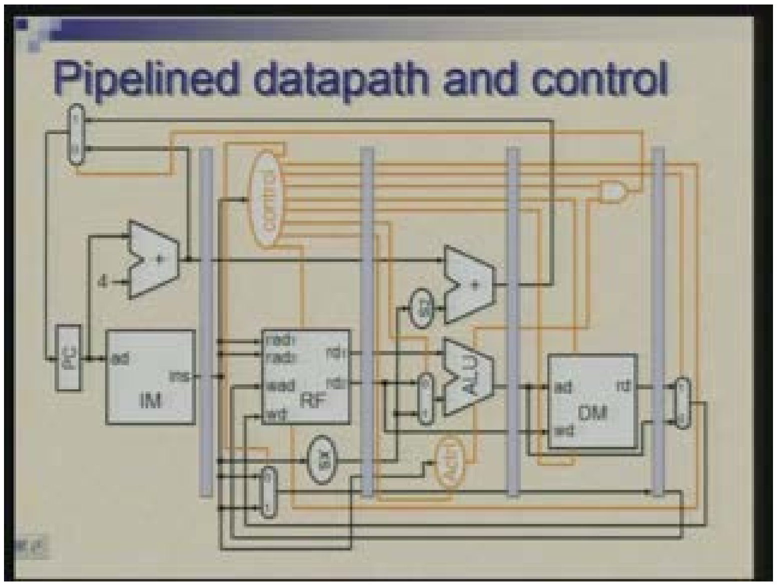
(Refer Slide Time: 00:03:08)



Pipelined datapath

Here is the pipelined datapath which we have designed in the previous lectures and you would recollect that all we have done was taken single cycle datapath and introduced inter stage registers and after introducing controls this is how things looked.

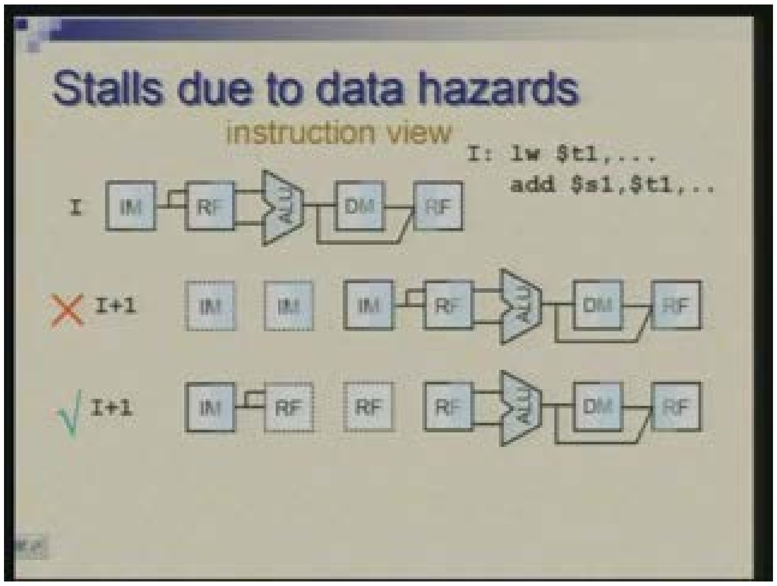
(Refer Slide Time: 00:03:20)



Pipelined datapath and control

You have datapath and control which is much the same as single cycle control but again we make the control signal pass through the inter stage registers so that they are applicable at the right time.

(Refer Slide Time: 00:03:44)



Stalls due to data hazards

instruction view

```
I: lw $t1,...
   add $s1,$t1,..
```

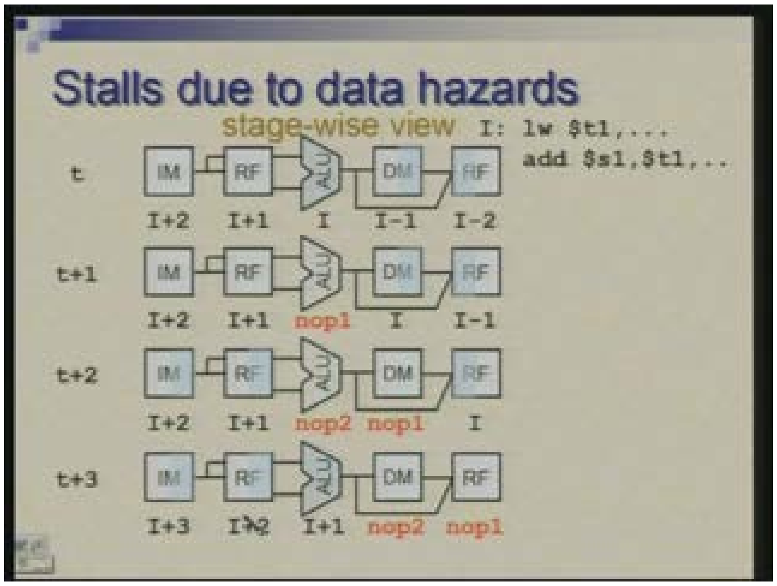
Now let us look at the issue of stalling when you find a pair of instructions which are data dependent. So, suppose you have an instruction which you are calling IF instruction which is the load instruction storing something in t1 and immediately an add instruction follows which is trying to read t1 we had earlier seen that this instruction which is following load instruction I plus 1 would be able to begin with the delay of two cycles and the picture which I had shown earlier was this (Refer Slide Time: 4:22) where we said that this cycle goes waste and this cycle goes waste and this instruction starts after a delay of 2 so that register read for second instruction and register write for the first instruction happen in the same cycle. This makes sense when reading and writing of register file is being done in a single cycle; first half of the cycle you write in second half of the cycle you read; in that case the value which is being written let us say in the middle of the cycle here is getting read in this and the data is correctly forwarded.

But this picture is not quite correct because what will happen is that in this cycle remember that in this the horizontal axis is the time axis. So in this cycle instruction I plus 1 will get fetched so IM cycle will get complete and it is the next cycle which will get actually held up. So the correct picture is to show like this that IM..... this I plus 1 instruction (Refer Slide Time: 5:25) is in the fetch stage here it is in the instruction memory then the next cycle what one would expect is RS cycle but it cannot read correct values here so it is this we want to prolong. So this instruction would remain in RF stage for three cycles and it is only the last or the third one which will be fruitful where actually you will read values and proceed further.

So this is the way you will actually implement delaying of instructions. So it is somewhere in the middle you need to delay. You got the instruction in the pipeline; you notice that you cannot proceed further forward you get stuck at RF stage; this instruction stays there at that stage and then two cycles later it proceeds normally.

Now we can view the same thing. You remember that we discussed that pipeline can be seen in two different views; in one view show instruction by instruction and have horizontal axis indicating the time so for each instruction you put show which stage it is passing through in different cycles. The other approach is to show all stages and have the time time time shot snapshots of pipeline in different clock cycles shown one after another vertically.

(Refer Slide Time: 00:06:52)



So if you do that...... this is the picture which will emerge and it will throw more light into what is happening in the pipeline. So these are the five stages and what we say here is that at time t in cycle t let us say the instruction I is here in the ALU stage then clearly I minus 1 and I minus 2 the previous instruction has gone ahead and they are here. I plus 1 instruction which is our add instruction that is where the trouble is; it has reached this point (Refer Slide Time: 7:23) it has reached RF and I plus 2 has reached IM now. So, from this cycle we will trace what happens in subsequent cycles.

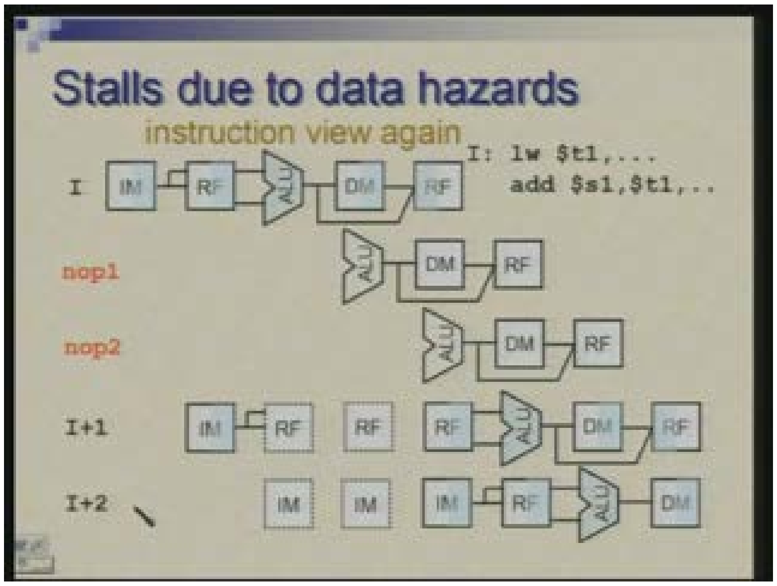
So you would also notice that I have not shaded these two because these ins these stages or these cycles of instruction I plus 1 and I plus 2 will not be able to complete and these two instructions will remain struck. So remember that it is not only I plus 1 which gets struck for two cycles it is the one which is following. So everyone behind the instruction in the queue gets held up.

Now, since in t plus 1 this is instruction is not able to move forward we have to introduce a no op; no op means no operation; it is a null instruction which does nothing and I have not shown ALU shaded which means that instruction here but it is not utilizing anything it is just occupying a cycle.

We go to next cycle; this is still not able to proceed (Refer Slide Time: 8:26) so third cycle will be spent by instruction I plus 1 within RF stage and here it will culminate; it will complete here so another no op gets introduced here because this no op moves forward there is nothing stopping that and that moves forward and you introduce another no op here and I plus 1, I plus 2 are struck there but this time they will complete and then move forward. I plus 1 will move to ALU stage here because here it has got the operands and behind it other instructions will also follow. So now effectively what has happened is that in this pipeline two bubbles have come. So let us say the fluid was flowing and there is some air bubble which is indicating no activity and of course with time this will get passed out.

We therefore need to be sure how to introduce these two no operations at the correct time instant that is crucial. With this understanding let us go back to the instruction wise diagram and see more accurately what is happening.

(Refer Slide Time: 00:09:34)

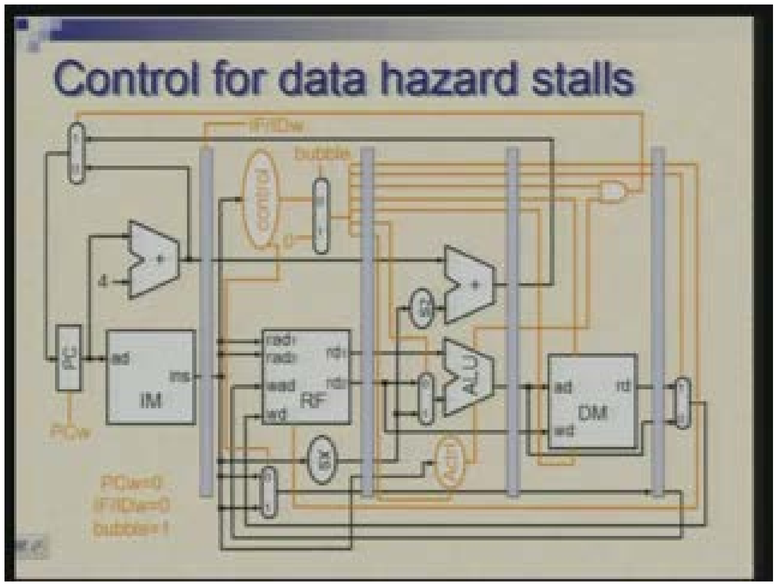


So, instruction I was like this (Refer Slide Time: 9:43) and in instruction I plus 1 we had introduced these two inactive stages I mean instruction is still in RF but not doing anything here; in between we have nop 1 and nop 2 two nops introduced which do not go through fetch stage; no operations get introduced from ALU stage onwards if they are not being fetched from memory as they are not going through RF stage but they get introduced from ALU onwards and then move forward so nop 1 has this profile, nop 2 has this profile then I plus 1, I plus 2. So I plus 1 you would notice is getting stuck at IM for three cycles.

So now what is happening?

Basically we need to check if an instruction which has reached RF stage it can proceed forward or not. If it cannot proceed further then we need to introduce a no operation and this condition....... we will work out a condition; if this condition is true for one cycle one nop gets introduced; if this condition persists for two cycles then two nops gets introduced.

(Refer Slide Time: 00:10:56)



So now let us look at this design where you have datapath and control. It is same thing I might have re-rooted some wires just for the sake of clarity of the diagram. So now where is it how how do we introduce no operations and how do we prevent an instruction from moving forward.

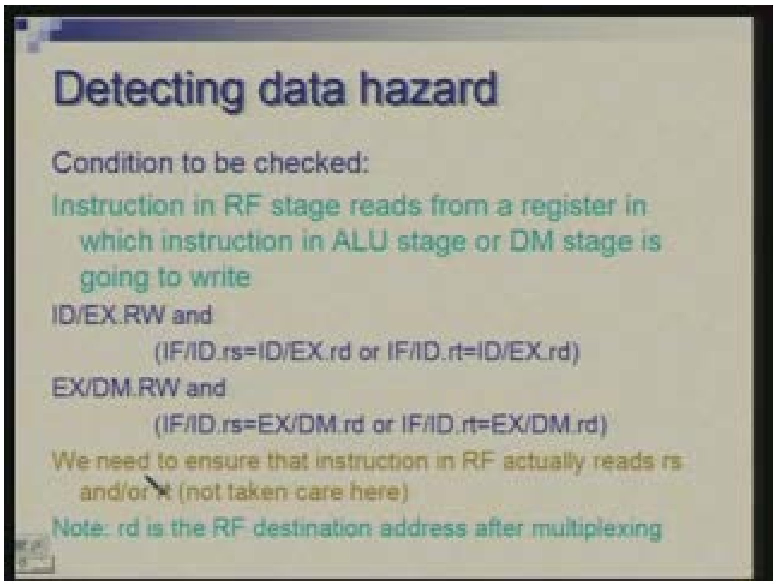
So we have this register and here is PC. If you do not clock this register if you do not put in new information in this (Refer Slide Time: 11:33) then whatever is here in this stage does not change. So let us say in some cycle instruction was fetched, brought into this register; if you do not change it for another cycle this instruction which is here remains as it is it does not move forward and similarly if you do not update pc new instruction does not get fetched the PC also remains struck with the old value.

So, we had not explicitly shown control signals for these two registers. But now to take care of stalling we need to define control signals for these which may be turned on or off depending upon what is the requirement.

Secondly, in order to introduce a null instruction or no operation instruction we could load all zero control signals all null control signals into this register. Remember that control signals which are applicable for subsequent stages gets stored in this register for a cycle then some of them get stored in this register for another cycle and then so on. So assuming that we have defined our control signal such that zero always means inactive situation we have put a multiplexer here through which controls are passing so there are some six seven or whatever number is all those are going as one input of the multiplexer and the other input of multiplexer has all zeros. So, depending upon how we select this multiplexer either these control signals get passed on or all zero values get passed on. So, for introducing a bubble or a nop all we need to do is this control signal which we are calling as bubble needs to be made 1; when you make it 1 then effectively you have introduced a no operation nop instruction in the EX stage.

Now basically to take care of the situation which we had just illustrated if you make pc write as 0, IF ID register write as 0 and make bubble as 1 this combination of control will achieve the desired result. What we need next is to figure out when do we do this. We need to detect that condition we need to basically detect the dependency between two instructions at the right time.

(Refer Slide Time: 00:14:08)



Detecting data hazard

Condition to be checked:

Instruction in RF stage reads from a register in which instruction in ALU stage or DM stage is going to write

ID/EX.RW and

(IF/ID.rs=ID/EX.rd or IF/ID.rt=ID/EX.rd)

EX/DM.RW and

(IF/ID.rs=EX/DM.rd or IF/ID.rt=EX/DM.rd)

We need to ensure that instruction in RF actually reads rs and/or rt (not taken care here)

Note: rd is the RF destination address after multiplexing

Here is the condition which you need to check to know that there is a hazard. First of all stating this condition in words what we need to do is instruction which is in RF stage if that reads from a register in which the instruction, the ALU stage or DM stage is going to write. so we look at instruction which is sitting in the RF stage, we also look at the instructions which are sitting in EX stage and DM stage; see where these instructions are going to write and see where this instruction is going to read from; if there is a match then there is a hazard condition.

Now, suppose at some point of time there was s hazard between instruction in RF stage and ALU stage that will cause instruction RF stage to be held back, instruction ALU stage will proceed to DM stage and we will still see a hazard that condition that means we are still waiting for that instruction to move forward and write. So this conditional will persist for two cycles and two nops will be introduced. The second instruction which is dependent will get delayed by two cycles.

But suppose we notice the hazard only when an instruction is in this stage I am sorry in this case (Refer Slide Time: 15:34) you will always first see it here and then here so actually the way we have described in this there will always be two cycles introduced because any instruction on which something is depending will pass through this stage and this stage and on two occasions you will see this condition.

Now how do you precisely check this condition?
I am introducing some notation here; there is register name this is the inter stage register and followed following the dot there is name of the control signal. So what we are saying is that instruction which is stored in this register ID/EX so that means instruction is actually in EX stage; instruction in EX stage has this RD, RW signal active that means it is an instruction which is going to write into register file; it could be a load instruction, it could be a R class instruction and one of these two conditions is satisfied this or this (Refer Slide Time: 16:41) and this is where we are comparing the register addresses.

So we are comparing RS address in IF ID that means the instruction which is in RF stage has a source register which matches with RD specification of instruction which is in the ALU stage or this destination is matching RT so one of the two may be matching the destination here is matching with RF or RT or the second part of the condition we instead of ID/IF we check with EX/DM so that is instruction setting in the DM stage. So, if that instruction intends to write then there is a match in the source and destination register.

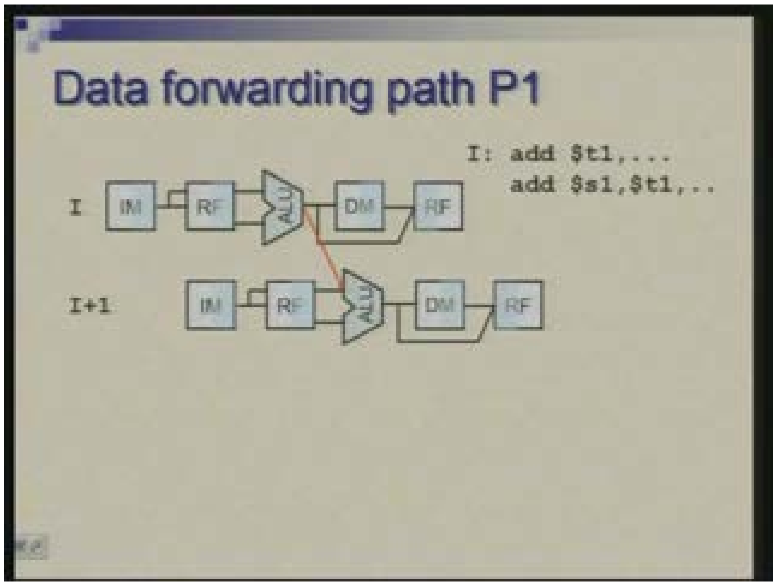
Now there is one little catch here is that we are trying to make sure that the instruction which is ahead is going to write but we have not really put the condition which says that instruction which is behind actually reads the register because we know that some instructions like jump do not read the registers so there is no point in trying to hold back a jump instruction so therefore this condition needs to be refined little bit so that you check it only if the instruction actually reads the registers. So there are some registers which will read only RS but not RT so we need to take care of those but I am omitting those details. So, there are two conditions which are corresponding to two parts of this condition.

Another point which must be noticed is that when we are looking at RD field in the instruction here this one or this one (Refer Slide Time: 18:40) we are assuming that a two alternative destination addresses have been multiplexed. What I mean here is that these two addresses are multiplexed here and then we are checking it here or we are checking it there. For some reason suppose you have placed this multiplexer somewhere else you can do that logically; it will have influence in terms of number of register bits and clock cycle and so on but technically it is possible then in that case you need to check with both. But the advantage of having multiplex is showing up here also that we need to check only one field here or there I am sorry this one; this one or this one or this one, this one (Refer Slide Time: 19:27). So this is the condition which needs to be checked and once this condition is true we need to set those IF/ID write signal, PC write signal and bubble signal.

Any question on this?

So basically this is some combinational logic which should be considered as part of the controller. Now this is a simple solution when you do not have data forwarding paths you are always introducing two cycles of delay if there is dependence between two consecutive instructions.

(Refer Slide Time: 00:20:04)



Data forwarding path P1

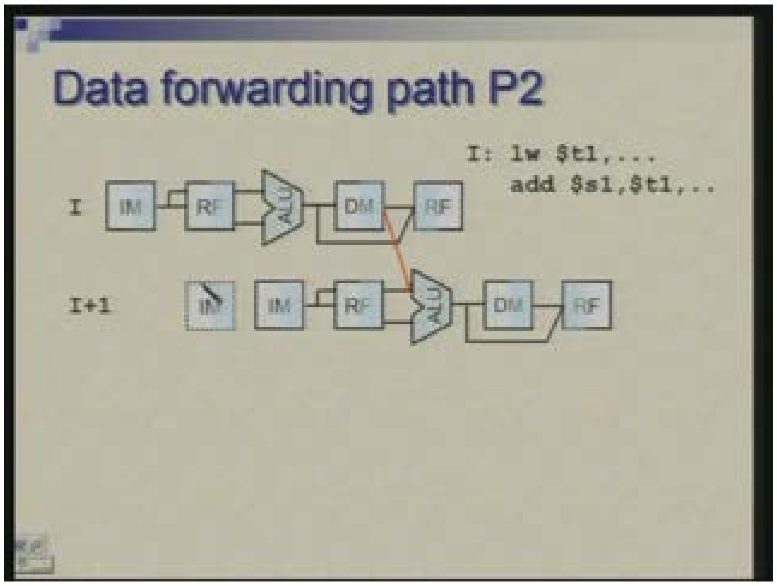
```
I: add $t1,...
   add $s1,$t1,..
```

Yeah I think there is a situation where the delay introduced will be one cycle. The two dependent instructions are separated by one more instruction in between so in that case this match would occur (Refer Slide Time: 20:29) when the earlier instruction has reached DM stage and the one behind is in the RF stage. When the first instruction is in the ALU stage then it will not find any conflict there. So there could be situation where we introduce only one cycle.

Now let us look at the data forwarding techniques. As you recall that idea here is that as soon as a result is generated we pass on to any following instruction if it requires it and we try to do it as early as possible.

There are various scenarios I am going to repeat those slides. So, suppose there is add instruction following an add instruction then the result of first is available at the output of ALU and it needs to be sent to the second instruction at the input of ALU so that is one possible path I am calling this p1; this path....... actually it means two paths leading to this input and another one to that input so we will actually require both.

(Refer Slide Time: 21:45)



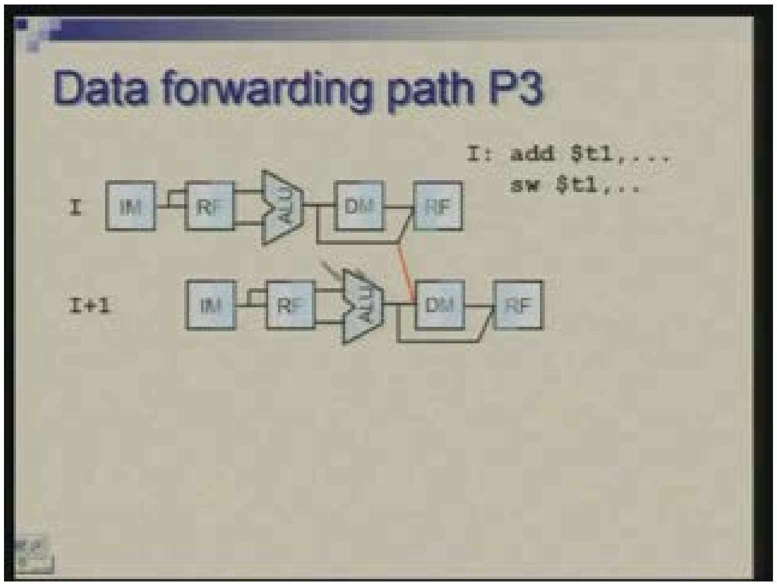
Data forwarding path P2

```
I: lw $t1,...
   add $s1,$t1,..
```

Another possibility is that a load instruction is followed by a dependent instruction like add. So in this case the data needs to be forwarded from DM state to ALU state and here you still need one cycle delay. Again this is not correct actually; we should show IM as a solid thing and RF as a dotted one here (Refer Slide Time: 22:03).

Yet another possibility is that you have add instruction followed by a store instruction. Particularly when....... see store instruction requires two registers; one register has to be read in the third .....is required........ actually both are normally....... they are read in second cycle in RF stage but the actual requirement of the value is in ALU stage for one of the registers and in DM stage for another register; the value as which is to be written in a memory is actually required in fourth cycle. So we are talking of that dependency where the data coming out of ALU is required in this.

(Refer Slide Time: 22:52)

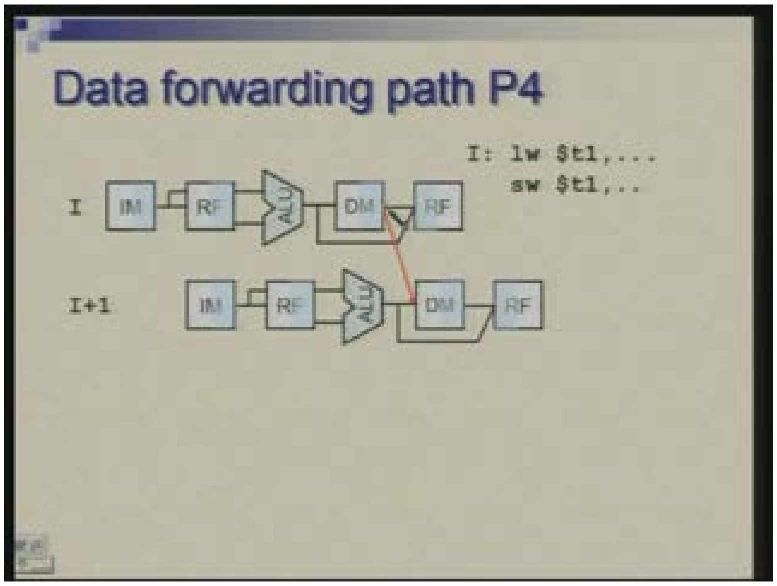


Data forwarding path P3

```
I: add $t1,...
   sw $t1,..
```

Therefore, now we do not pass directly from ALU to DM because that is no longer available that value would have moved to next stage so we will have to tap it after the DM stage and get it to input of the DMA.

(Refer Slide Time: 23:10)

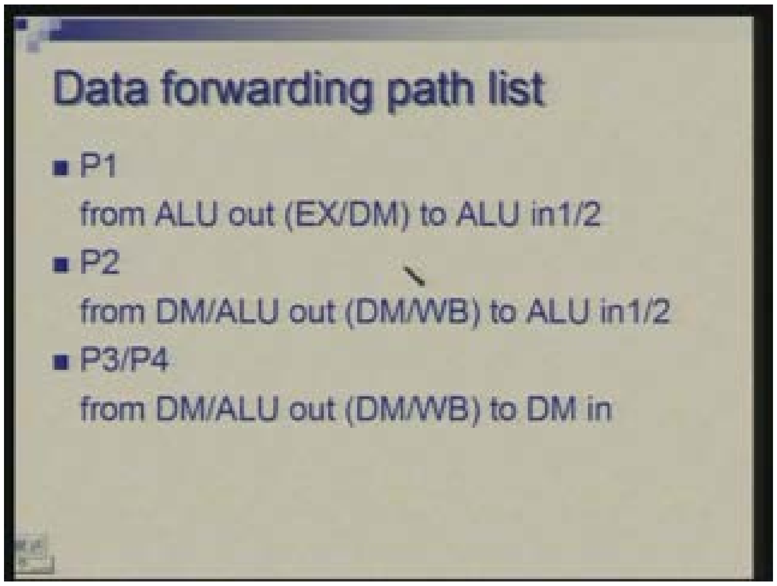


Data forwarding path P4

```
I: lw $t1,...
   sw $t1,..
```

And fourth possibility is that you have load followed by a store so output of DM goes to DM.

(Refer Slide Time: 00:23:18)



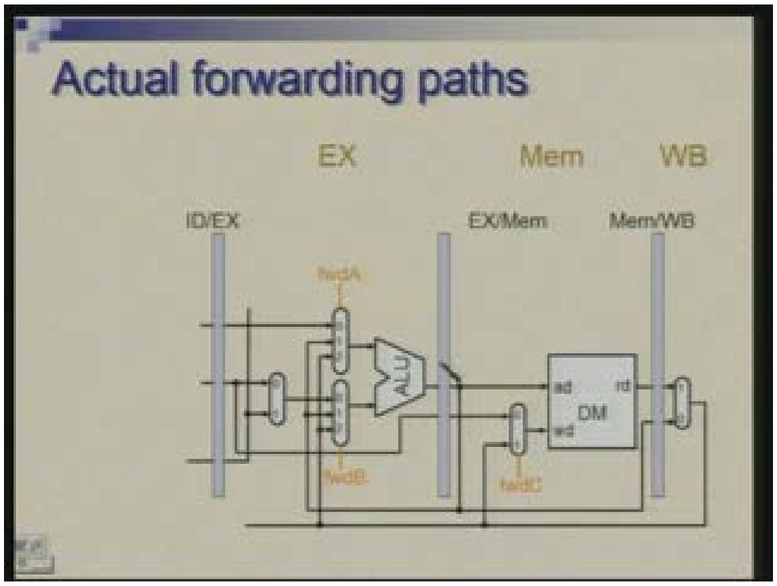
Data forwarding path list

- P1
  from ALU out (EX/DM) to ALU in1/2
- P2
  from DM/ALU out (DM/WB) to ALU in1/2
- P3/P4
  from DM/ALU out (DM/WB) to DM in

So now we have these paths; this slide actually summarizes all of them. You have P1 P2 and I am looking at P3 P4 together. So P1 goes from ALU output; strictly speaking from the output of EX/DM register to ALU input 1 or 2, then P2 goes from.... it is basically output of DM or ALU; we have to tap it from DM/WB register and send it to ALU input 1 or 2. Why I am saying output of DM or ALU is because well it is because we will actually try to tap all these after the multiplexer.

Basically there is a DM/WB register, after that we have a multiplexer which will look at two possibilities and we will take it from there and take it to the input of ALU input 1 or 2 and then from the same multiplexer we take it to DM input.

(Refer Slide Time: 00:25:01)


Actual forwarding paths

I have taken part of the datapath augmented with these forwarding paths and shown here so let us identify these paths.

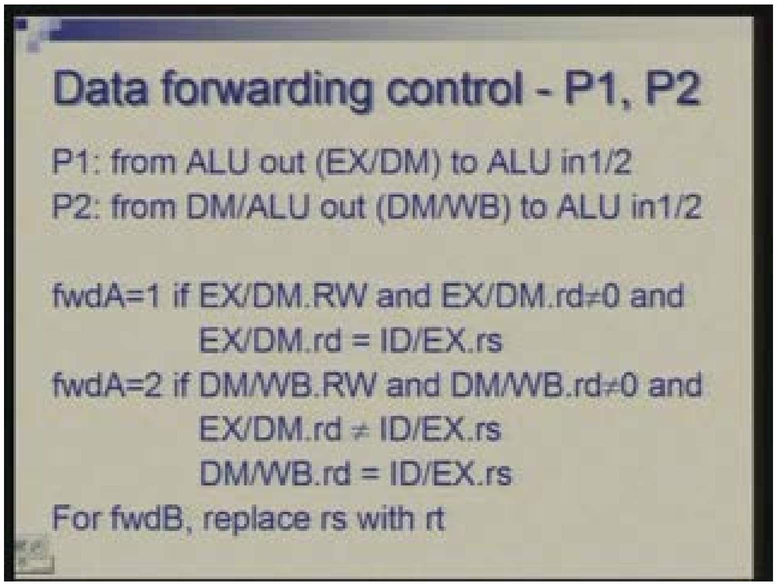
P1 is here (Refer Slide Time: 25:18) this is P1, from output of ALU we are not taking from here we are taking after the register; we are taking it to two inputs of ALU. So you notice that there are additional multiplexers which have been introduced. The normal input is 0 which is either coming from register files through this register or from register file through this register or the offset one of these two gets selected and coming as the normal input. So inputs labeled as 1 are those which are part of P1 so this output (Refer Slide Time: 25:59) is available here and there depending upon where you need or maybe you need at both places.

You could have an instruction which is trying to add you might say add A comma B comma B so that B may be dependent and it is required at both the inputs. So you may situation there you may pass on same thing on both the cases. Then P2 is this, we are taking from this multiplexer; output of this is normally going to register file. We are bringing a copy here at input 2 so this is P2.

For P3 which is from DM to DM we have introduced a multiplexer here. The normal path is which is coming from register file and this forwarding path is a copy of this is brought here. So now we require three control signals forward A, forward B and forward C and we need to work out conditions which will activate these or which will give proper value to these signals. So forward A is actually 2-bit signal but symbolically we will say that it has the value 0, 1 or 2; similarly forward B and forward C will have only one of the two values. Is this clear? This is how exactly the paths are organized and next we need to see the conditions to energize these paths.

So let us look at P1 and P2 the paths which are leading to the multiplexer at the input of ALU.

(Refer Slide Time: 00:27:38)



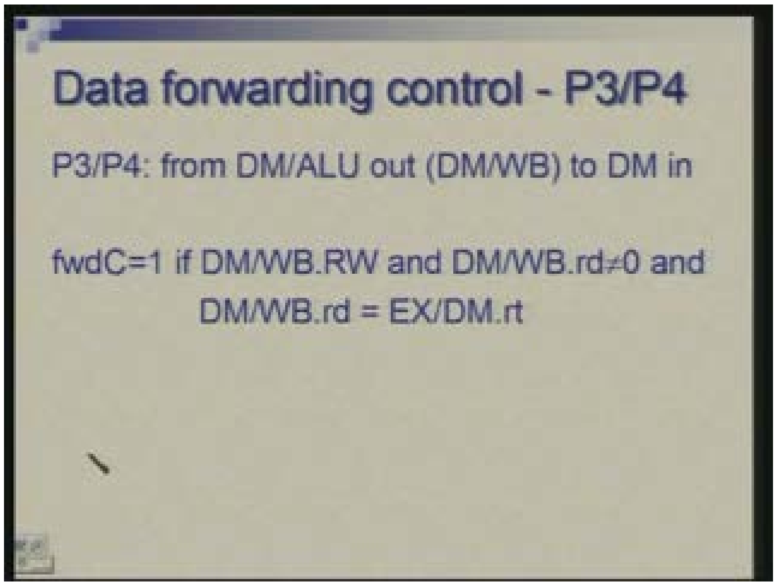
Data forwarding control - P1, P2

P1: from ALU out (EX/DM) to ALU in1/2
P2: from DM/ALU out (DM/WB) to ALU in1/2

fwdA=1 if EX/DM.RW and EX/DM.rd≠0 and
EX/DM.rd = ID/EX.rs
fwdA=2 if DM/WB.RW and DM/WB.rd≠0 and
EX/DM.rd ≠ ID/EX.rs
DM/WB.rd = ID/EX.rs
For fwdB, replace rs with rt

These are coming from different stages. The control signals we require are forward A and forward B would have similar equations. So, when is forward A equal to 1 that is the condition; when is forward A equal to 2 that is a condition.

(Refer Slide Time: 00:27:55)



Data forwarding control - P3/P4

P3/P4: from DM/ALU out (DM/WB) to DM in

fwdC=1 if DM/WB.RW and DM/WB.rd≠0 and
DM/WB.rd = EX/DM.rt

If these conditions do not hold by default you will make sure that forward A has a value 0; none of these holds the value 0. Condition for forward B would be similar but first let us look at this.

What we are saying here (Refer Slide Time: 28:15) is now again the same notation is used that instruction in EX/DM are basically in the......... yeah instruction EX/DM stage intends to write in the register file and it is in intending to write in a register which not addresses a 0 so we are taking special care of zero register because that does not cause dependency. Actually we should have made this check in earlier case also when we were specifying other conditions without forwarding because if one instruction is writing into 0 for whatever reason right or wrong another instruction when trying to read from 0 there is no dependency because the value is going to be 0.

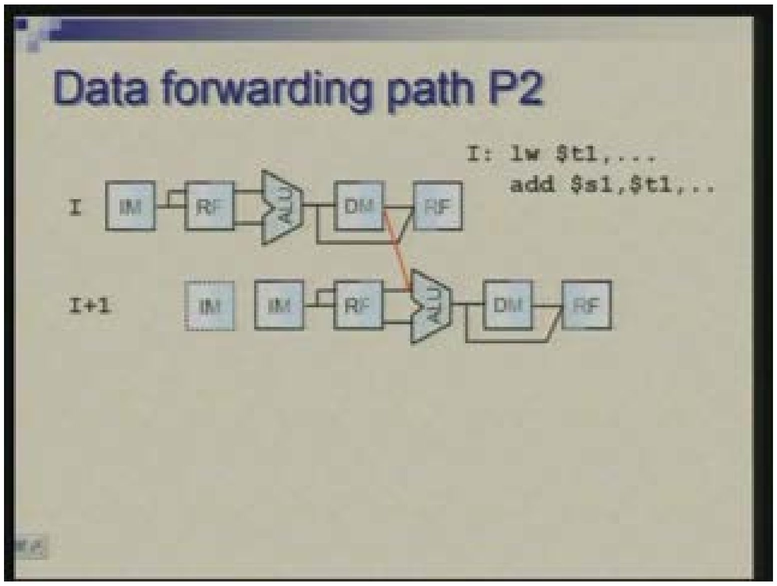
So we are checking here that this instruction intends to write, it intends to write in a register which has non zero address and that address is matching with RS of the instruction which is currently in the ALU stage. So the instruction from where we are picking up the data obviously that has reached here; it is this value we are feeding back. So instruction here (Refer Slide Time: 29:40) intends to write into some register from where the instruction here intends to read that is what we are checking.

Similarly this is a forward A equal to 2 if in DM/WB stage the instruction which has even moved further intends to write into non zero register and that matches the destination matches the source here. Also, we need to make it 2 when it is not 1. Imagine that there are there is some instruction in this stage (Refer Slide Time: 30:18) which is reading from some register where instruction here is trying to write as well as instruction here it is trying to write. So actually we should forward the data from this one because this will be of more current value which is supposed to read. Because if this writes into some register the instruction here will overwrite over that and it is that value which will be latest and should be read by this instruction. So therefore we have introduced this additional check here that this is not matching. That means the instruction which is between the two is not intending to write in the same register; that is required so that we read the latest value only.

Therefore the condition for defining forward B will be exactly similar; all we do is rs is replaced with rt because forward B is nothing but controlling the multiplexer which is below and here the relevant register field is rt in place of rs and rest of it remains same. So this was for data forwarding path P1 and P2.

For P3 P4 we are trying to forward data from WB stage to the DM stage so there is a similar condition that instruction in WB stage intends to write into non zero register and the destination there matches with the instruction in DM stage which actually require rt register. These are the conditions which can be checked by control and energize or enable the right forwarding paths. Having done that now what is still required is that in spite of forwarding there are cases when null needs to be introduced and that may go back to this case.

(Refer Slide Time: 32:22)

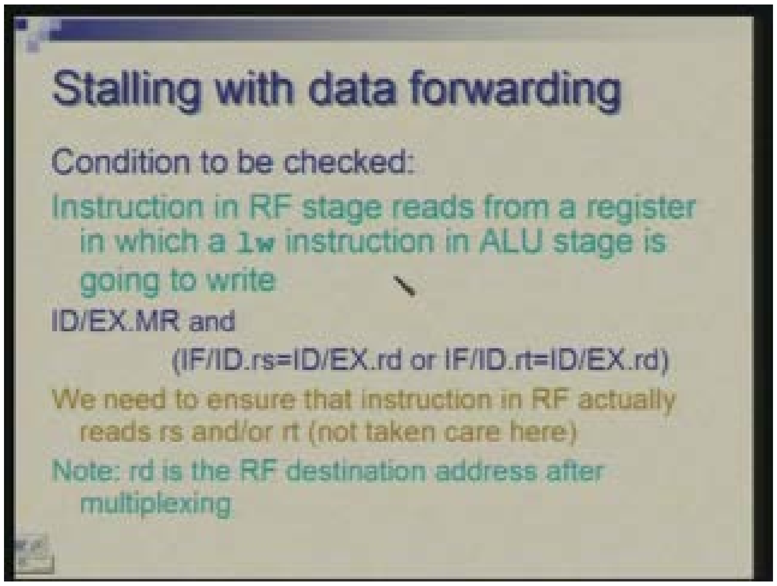


Data forwarding path P2

```
I: lw $t1,...
   add $s1,$t1,..
```

The only time this happen is that the earlier instruction is the load instruction because that is the only one which writes whether the result is ready after the fourth stage and the following instruction needs the result in the ALU stage. So we need to check this condition and we know how to introduce stall cycles; it is only the condition now we need to define freshly.

(Refer Slide Time: 00:32:52)



Stalling with data forwarding

Condition to be checked:

Instruction in RF stage reads from a register in which a `lw` instruction in ALU stage is going to write

ID/EX.MR and
(IF/ID.rs=ID/EX.rd or IF/ID.rt=ID/EX.rd)

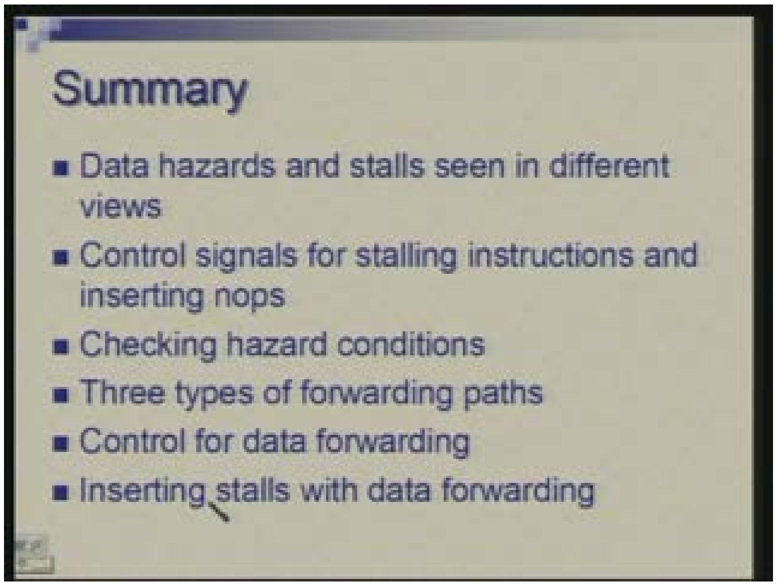
We need to ensure that instruction in RF actually reads rs and/or rt (not taken care here)

Note: rd is the RF destination address after multiplexing

So, stalling condition when data forwarding is happening in the manner we have described the condition we need to check is that instruction in the RF stage reads from a register in which a load instruction, we are not saying any instruction but specifically a load instruction in the ALU stage is going to write. So the check is happening actually again referring to this diagram (Refer Slide Time: 33:18) we have to check when this load instruction is here only it is not that we check when instruction has reached there; it is still at this stage that load is here and the following instruction is here we hold the following instruction at the RF stage for one cycle so that will be the effect we will achieve if we check this condition; that is we are looking at MR signal which is memory read and the only instruction which does memory read is load instruction. So we are looking at the signal; if this signal is active the instruction which is in EX stage and the destination of that matches with either rs or rt of the instruction which is at RF stage then we exercise our controls for holding the instruction back by the cycle and introducing a nop.

We have again.... the same shortcoming is here in the condition that if the instruction which is following is a jump instruction we do not want to do all that.

(Refer Slide Time: 34:30)



Summary

- Data hazards and stalls seen in different views
- Control signals for stalling instructions and inserting nops
- Checking hazard conditions
- Three types of forwarding paths
- Control for data forwarding
- Inserting stalls with data forwarding

I will summarize at this point. Basically we have looked at the data hazards. We analyzed what are the different situations. We saw its effects in two different ways when we are seeing instruction by instruction how the picture looks like we noticed that instructions are getting introduced nops are getting introduced from ALU stage onwards. We also saw in the stage view that is you look at all the stages and take times times time slabs. This was seen in two views to get a clarity of the situation and we identified how we are going to how we are going to introduce stall cycles and nops are inserted, the control signal were defined for that then conditions were defined, we checked the hazards and activated those control signals then to improve the performance we talked about data forwarding paths. they have three types of data forwarding paths actually four but two of them we consider together P1 P2 P3 P4; P3 P4 we consider them together because we took the value after the multiplexing so both were combined and then this requires its own control because there are some multiplexers which need to be activated correctly.

Further there are still situations where stalls are required and we defined conditions for introducing stalls and along with the data forwarding. Basically the conditions which we developed for stall in the end would be used in place of the conditions we developed earlier. If you do not have forwarding paths then there was one set of control conditions; if you have forwarding paths then you need two things: you need control signals to select these multiplexers correctly and also in special cases introduce the stalls. So, if they are any questions I will take that otherwise I will stop here.

No questions? Okay thank you.