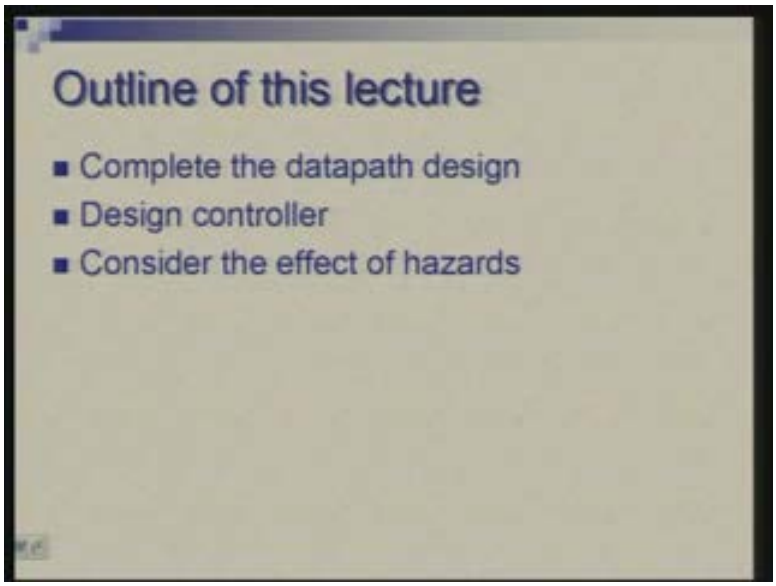
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 25
Pipelined Processor Design: Datapath and Control

In the last lecture we discussed a new type of design called pipelined design where the objective is to get a low cpi as a high frequency of clock. The idea here is that you attempt to initiate one instruction every cycle and at any given time there are number of instructions which are there in the datapath at different stages of execution. Ideally each stage should have one execution to keep the pipeline full and get maximum benefit. But we have seen that there are situations where you cannot keep the pipeline full and delays get introduced which causes loss in the overall performance.

We briefly hinted on some ways of handling those situations these were called hazards. We looked at three different types of hazards: structural hazards, data hazards and control hazards. By design we have tried to eliminate structural hazards. But data hazards and control hazards are somewhat inherent to the whole concept and cannot be always rules out entirely. So we have to do something to reduce their effect as far as possible.

What we will do today is look at the skeleton datapath we had discussed last time, complete this and then see how such a pipeline can be controlled. So, first we will see we will ignore the hazards and see in normal condition in ideal condition how pipeline could be controlled and instructions could be initiated one instruction per clock cycle.

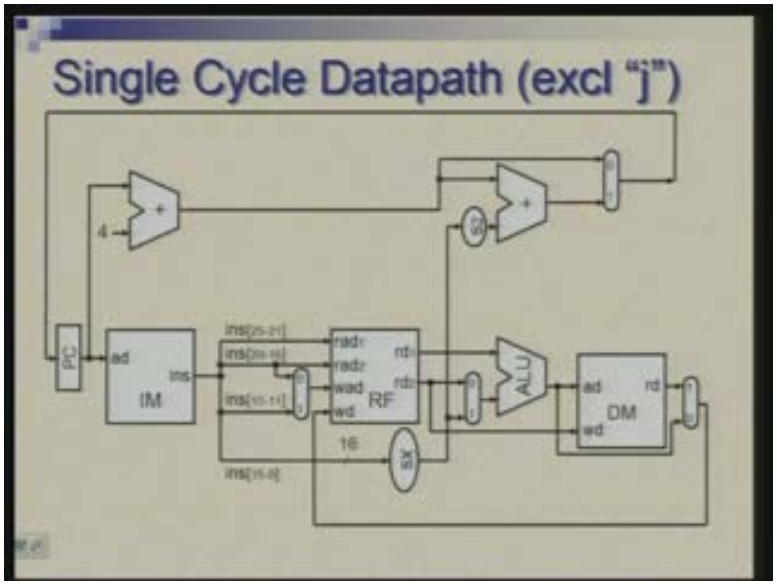
(Refer Slide Time: 00:03:23)



So we will first, as I mentioned, complete the datapath design, introduce all the components, multiplexers and shifters, bit rooters and so on then introduce the controller

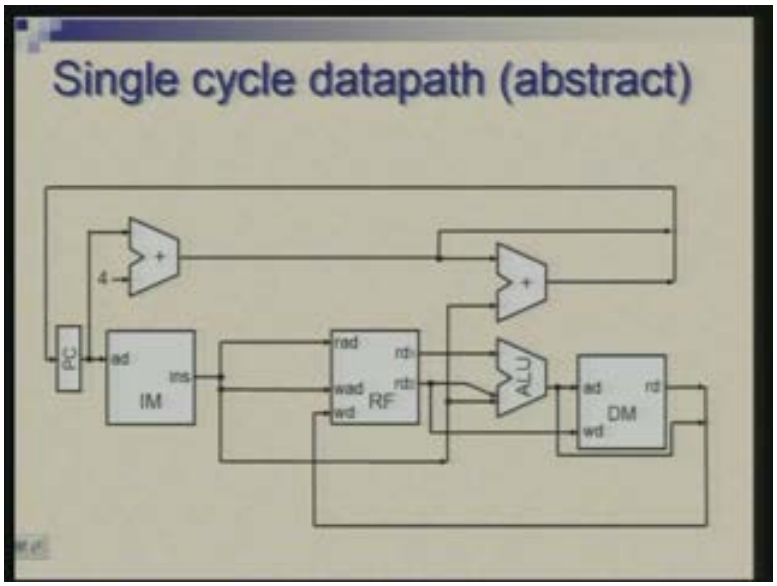
and then finally see that how this simple pipeline design behaves in context of hazards. And possibly, in subsequent lectures, we will augment our design to handle these hazards.

(Refer Slide Time: 00:03:29)



So you recall that we started with a single cycle datapath which was formed which was used as the basis for designing pipelined datapath. So just to simplify things we had omitted instruction j not because it is too complicated but just to reduce the size of the datapath so that it becomes easier for discussions.

(Refer Slide Time: 00:03:53)

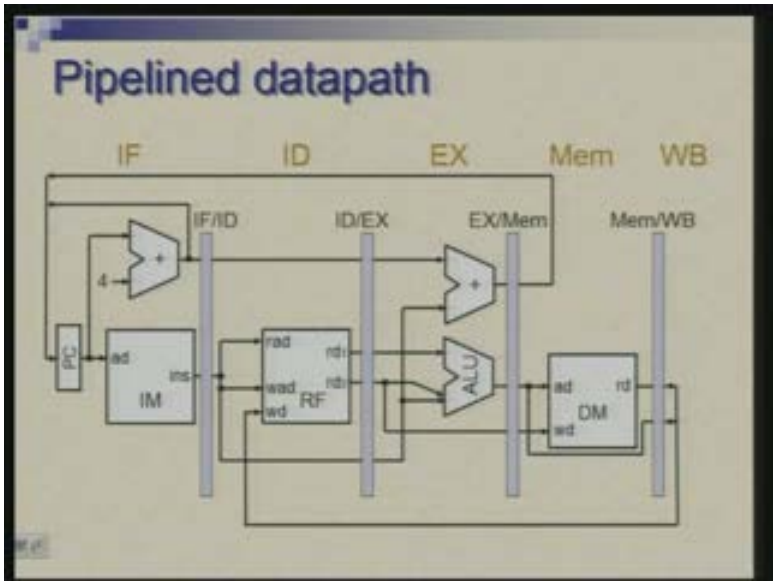


And another simplification we did was that we omitted some of the components like multiplexers and sign extenders and shifters so that again the number of components you see in the diagram is smaller and you can easily analyze and discuss. So the approach to the datapath design was basically to introduce the registers which would separate stages from one and another.

So for example, first stage is considered instruction fetch stage which involves accessing the instruction memory and at the same time updating the PC value to PC plus 4. This is one stage (Refer Slide Time: 4:41) and we introduced a register here. Next stage is where instruction is getting decoded, control signals are getting generated and operands are being fetched from register file so that is the second stage and after that again there is an there is a register. Third stage is where ALU comes into action. It may perform either address calculation or for R class instruction it could do add, subtract AND OR facility and so on and at the same time address calculation for branch instruction may be required to be done and there is an another adder for that and the final stage not the final the fourth stage is memory access for read or write of data and the final stage is writing results back into register file.

What we said was that all instructions will see these as the five stages. If some instruction needs to skip some stage it could skip some stage by simply wasting some cycle so that the whole pipeline operates in a uniform manner and this is the design.

(Refer Slide Time: 00:05:46)



We have…… these registers (Refer Slide Time: 5:54) are basically cutting all the forward going paths. Any line which is going forward is cut by this which means that the size of this register for example which we are seeing as the first register will be 32 bits of instruction plus 32 bits of this address so it is a 64-bit register. The register which you see here is cutting 1 2 3 and 4 paths so it is probably a 128-bit register. This is cutting three paths so it is 96 bits register; this is 64-bit register and so on. These are these are large

registers and what we are saying is that all forward going paths have to pass through. That means the information is available after the register after one cycle of delay.

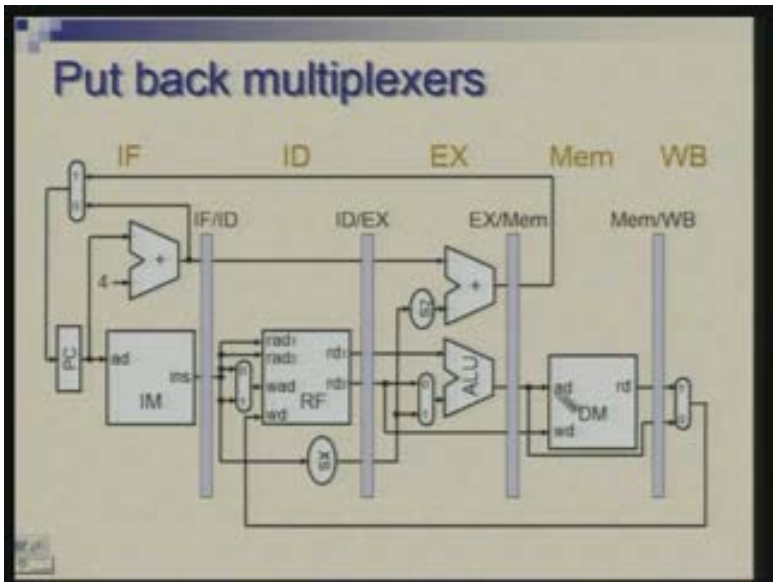
In this there are couple of backward going paths; one is the path which is used to write back to register file so that is not passing through register file through these registers, this branch address (Refer Slide Time: 7:00) is going back, PC plus 4 is going back, they have to go back to PC. Now we deliberately kept this path going from PC plus 4 to PC turning back within the first stage and this is very crucial to be able to get one instruction in every cycle.

You recall that an intermediate design was that this multiplexing which we expect here was actually happening here which would mean that you cannot initiate next instruction unless first instruction has gone through some three stages or so. But that is not in the spirit of what we are trying to do. Therefore as a special case we are turning this back right here (Refer Slide Time: 7:47) and this loop can turn out new PC values every cycle and therefore a new instruction gets pumped into this pipeline in every cycle.

Now what is missing in this datapath are the multiplexer which we had removed for convenience of discussion and we can put these back. So here we have a multiplexer which is computing the next PC value out of these two choices. Now you would recall that there is a peculiar thing happening here that is we are taking this PC plus 4 value generate for instruction which is here but this branch address is coming from an instruction which is here. So what is the consequence of that we will have to analyze but let us leave that there.

We have a multiplexer here because the write address comes from two different sources a multiplexer here because ALU may be adding data coming from register file or from the instruction, the offset, a multiplexer here which decides what goes to the memory output of what goes to this register file, output of memory or output of ALU.

(Refer Slide Time: 9:10)

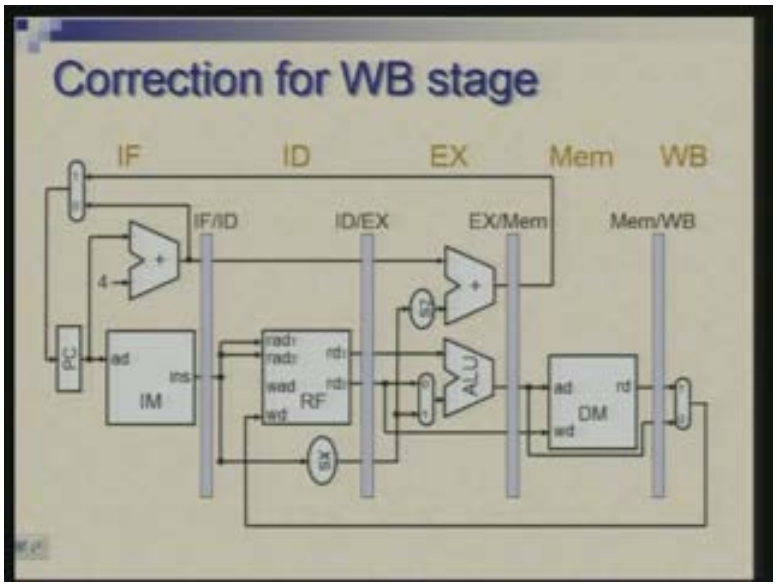


So notice here that if there is an R class instruction where memory stage is not being used then data is simply passing through and experiencing a delay of one cycle here. So essentially we are going through that cycle but not doing anything; idling through one cycle so that pipeline operates in a very homogenous and uniform fashion.

There is a slight problem with this datapath one day if you can notice. Yeah, exactly that is a problem that is normally which we have introduced.

Look at the write back operation. Write back (Refer Slide Time: 9:55) is happening as fifth stage activity so data which is going back to this is coming here after having been delayed to the fifth stage. But when the data comes here the address which is coming is from an instruction which is still here; it is a different instruction. So what we need to do is we also need to take this address through the same amount of delay and basically carry the data and address together let them experience same delay and then when you write they are available to the register file at the same time. So the change we require is as follows.

(Refer Slide Time: 00:10:34)



I have removed the logic here which is feeding the address to the register file and we introduce another path which is actually going through these three register stages and then coming back here. So this multiplexer is same thing I will just reposition it here (Refer Slide Time: 10:55), the output of this multiplexer is going through these three stages so that the data and address all move together and are presented to the register file in the same cycle and therefore they are consistent with each other otherwise there is a mismatch; you are taking data of one instruction and address of another instruction and trying to use them together.

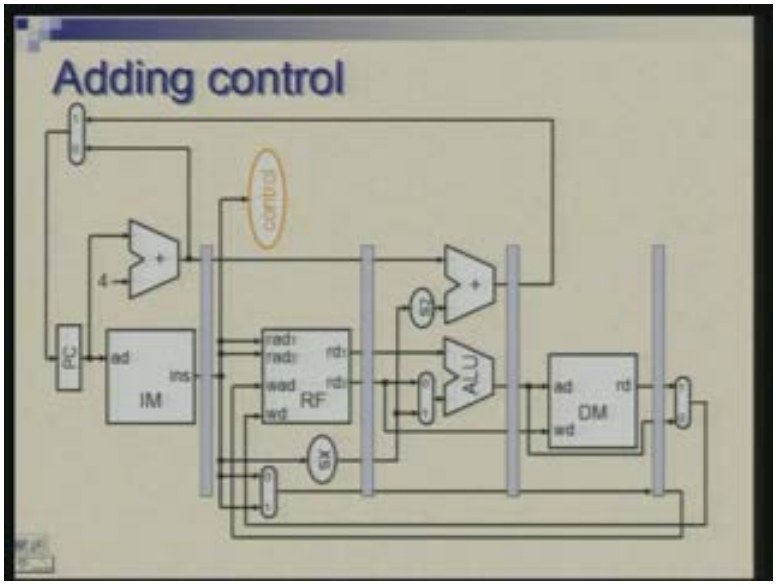
Now there may be lot of options here. For example, should I keep the multiplexer here or should I take both these addresses all the way and then multiplex just before feeding in. So, logically both are equivalent. The consequences of multiplexing later would be that I would here I am consuming five bits in these registers if I do not multiplex I have to pass on two five bit pieces through these registers and therefore my register length will increase so the idea is to multiplex them early and then pass through this.

Using the same argument if you look at the way sign extension is being done just try to imagine the consequence of positioning this sign extension unit after this register. Input of sign extension is sixteen lines, output is thirty two lines. Now, by positioning at here I am actually allowing thirty two lines to pass through this register so I am actually wasting. On the other hand, if I position this afterwards I have positioned it just for more convenience of the diagram but the right solution would be to keep it afterwards so that I am only consuming 16 bits here in the registers. So this is not the best position the best position is here (Refer Slide Time: 12:49). Is that idea clear?

So these are only Positioning these is just a matter of saving a few bits here and there. But more crucial thing is that the address is made to go hand in hand with the data and

reach the register file at the same time so that is the idea; that is the correction which was necessary to do WB operation correctly. Now let us move to the control.

(Refer Slide Time: 13:19)



We will introduce a controller and try to generate control signals for all the components which require for example register files, multiplexers and so on.

[Conversation between Student and Professor: (13:44) which one……this multiplexer; this can be actually this yeah, you are right, this can be brought back before the register. Right now we are passing two things so we are consuming 64 bits here. If we bring this multiplexer before then we will save something here. But while we are discussing this point let us also understand other implications of making such changes. Let us also see, what is the influence of such repositioning of components of multiplexers in particular before register and after register? What is the influence of this on the delay or on the clock period?

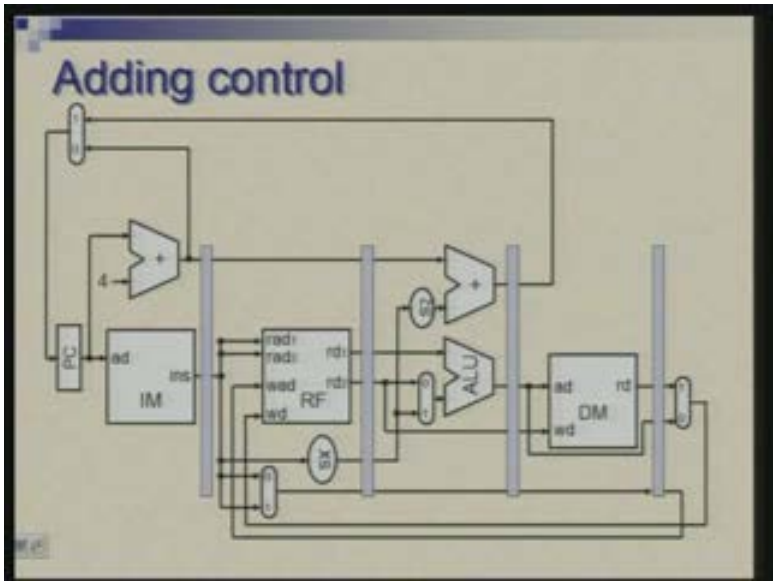
Now, the clock period is dictated by the longest path within any of the stages. So, for example, in the first stage (Refer Slide Time: 14:36) the paths are from PC through this IM through this register or through PC through this adder through this multiplexer back to PC. Basically in each stage we need to consider all paths going from some register to other register or some storage element to other storage element.

Now let us say if this multiplexer is positioned where it is then it is going to form the part of the path which starts from this register goes to this multiplexer goes to register file and ends in this register (Refer Slide Time: 15:15). So, delay of this multiplexer which we have we have been neglecting treating them as zero but in real life there will be some small delay that delay will get combined with this register file when you calculate the maximum path delay or such delays are called critical path delays.

On the other hand, if you keep it before register file then it gets lumped with data memory delay. So you need to see which one of the two which one of the two choices achieves a better balance of the delay. Suppose the delay of data memory was much larger than delay of RF that is how the case would be typically. Then keeping this multiplexer before would add to the DM delay and make things worse.

On the other hand, there is some slack some room here on the RF side to accommodate some more delay and therefore keeping this here may help from that point of view. So here is a trade off. There is an influence on the clock period; there is also influence on the hardware cost in terms of register length, register size and so on. So there is no universal answer here; one has to see, for a specific design what are the things that change and what is that you are trying to optimize.

(Refer Slide Time: 00:16:39)



So, coming back to this we want to add control. Now question is which style of control we are going to follow; are we going to follow the single cycle type of control design or multi-cycle type of control design?
In single cycle the key observation was that the controller was purely a combinational circuit; it looks at the instruction, generates the sig generates the signals whereas in multi cycle design there was a finite state machine which steps through states and in each state controls may be generated differently because different action is required in different states therefore we need to look at the current state also look at the opcode and then decide the control signals.

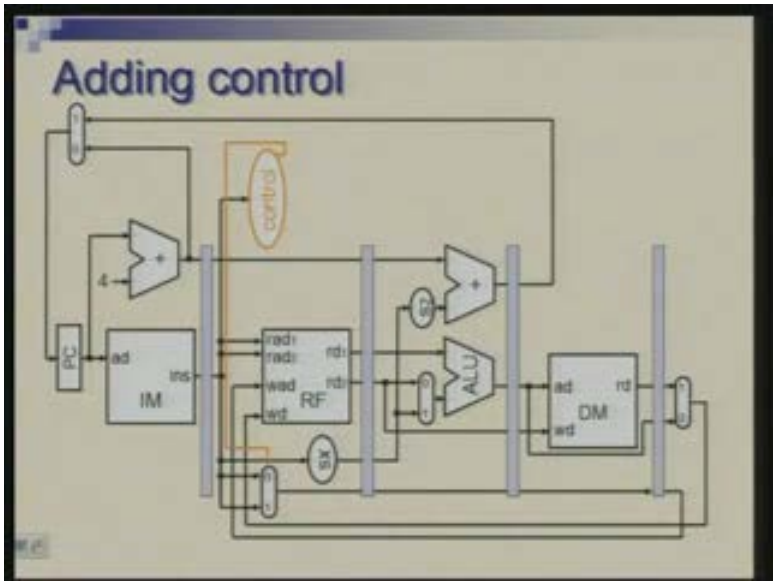
In fact we had organized things such that the control signals gets determined directly from the state and we do not even have to look at the opcode because in some sense the influence of opcode would have been taken care of while making the control transition. In fact there was a question which some students asked later on that can we not reduce the number of control states there and allow the control signals to generate by looking at

opcode as well as control state. We are trying to branch off after the first two states you know; we have fetch state then we have decode state and then we are branching off depending upon the instructions Suppose we do not branch off we go through a sequence of three states, four states or five states just in a chain and repeat that because we have the information about what is the opcode. So, looking at the state and the opcode we can always determine what control signals are there. If you recall your theory of finite state machines this will be a Mealy machine type of approach; what we have followed is a Moore machine type of approach.

Now, coming back to the question of what style of control we can follow what will suit the pipeline design; will the controller be a combinational circuit probably or will it have states; would you need to do something different cycles and therefore remember the states.

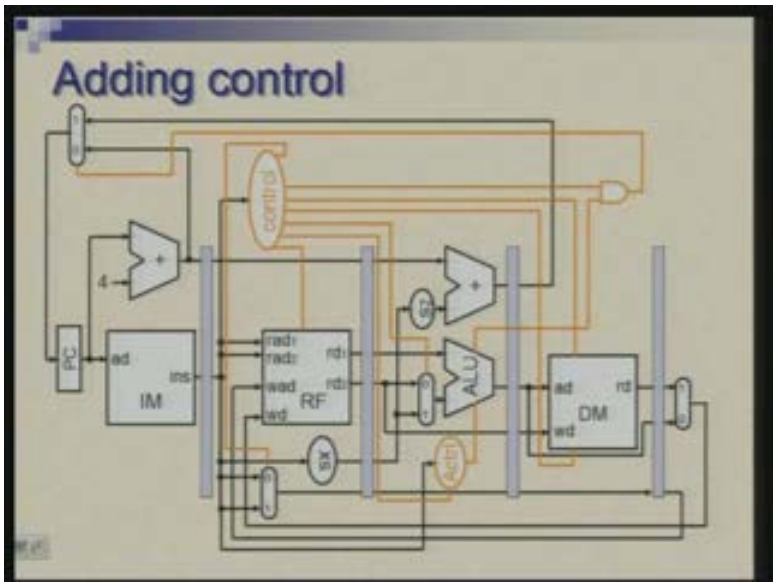
So, the answer is that it cannot be a purely combinational circuit because an instruction is going through multiple cycles and an instruction needs to do different things in different cycles and control need to be generated differently. But remember now that things are even more complicated here that it is not just one instruction which is in flight there are up to say five instructions which can be in different stages and in a given stage we have a particular instruction and we need to generate control for that. So it might appear that you need to look at either you maintain several control states one for each instruction or look at the opcodes and decide control signals differently for different instructions. But fortunately what turns out is that again we can start with the control design of the single cycle approach and by simple changes there we can actually derive the controller for this particular case. So let us reconstruct the controller for the single cycle. Let us superimpose over it and then see what changes are required.

(Refer Slide Time: 00:20:22)



We have control signal going to this multiplexer, next there is a control signal which controls the write operation register file, this is a control signal which goes to the ALU and remember that this has to go through another small controller which will look at some bits of the instruction (Refer Slide Time: 20:46). So here we are going to look at the function bits, the six LSBs of the instruction that will be another input to this; let me connect that as well. So this looks at some control signal which actually has 2 bits and then 6 bits coming from here and generating signal for ALU. Next is control for the multiplexer which is feeding ALU, then write control for data memory, read control for data memory, the AND gate which generates signal for taking care of branch instruction. So there is a zero signal result of comparison coming out of the ALU that is gated with the signal coming from the controller and that together control this multiplexer.

(Refer Slide Time: 00:21:24)

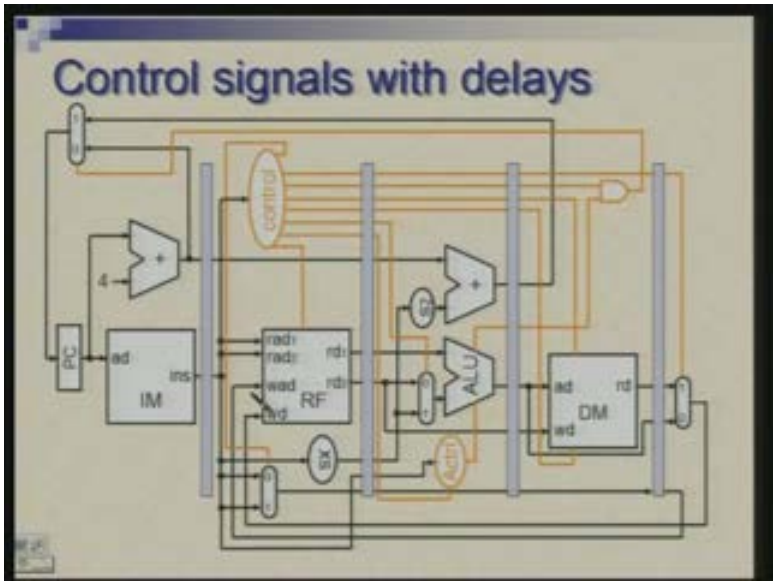


We still have not really seen that what is the influence of mixing of datapath from two different instructions. This is the address coming from this instruction which is here (Refer Slide Time: 21:55) and that is the address coming from instruction which is in that stage. But anyway the signal which is getting generated is found by ANDing a signal coming from controller and Z output of ALU. Finally we need one more for the multiplexer at the output end.

Now if we leave it this way what is going to happen is that this controller will look at the opcode which is sitting here; the instruction which is in decode stage is going to drive the controller and the control signal seem to be going to all stages. The consequence of this is that we are ignoring the identity of other instructions which are sitting at different places. What could be done is that if you recall how we solved the problem of synchronizing address and data while writing to register file all that we did was that address was also delayed the way data was getting delayed. So what can be done for control is exactly the same thing. That is you generate the control signals when the instruction is in second stage or the decode stage but do not apply all the control signals to all the stages

immediately; you apply them as the time comes. So signals which are relevant for the current stage you apply them immediately; signals which are relevant for the next stage you delay them by one cycle; signals which are applicable two stages later you delay them by two cycles and so on. So essentially what needs to be done is that we extend these registers to accommodate the control signals also and all the control signals which are they are except for those which are going in within this stage those which are going forward are passed through these registers the same way as we do for data signals and addresses.

(Refer Slide Time: 00:24:05)



So all that I have done is I have extended these walls to separate control along with the data. So basically the signals which are effective within this cycle are going directly; the signals which are required here in the ALU stage is going through one register so basically you could see these two signals going to ALU control and going to this multiplexer.

Signals which are required in the memory stage are going through two units of delay; they pass through this register, they pass through this register and then they are applicable here (Refer Slide Time: 24:47); signals which are used for write back are going through three stages and getting delayed. And of course also the signal which is going to this multiplexing is also moved later on and passed through these two registers so that this address which is going and the control signal which is going there they all reach the multiplexer together. So basically this is a simple arrangement where we can start from controller of the single cycle datapath and then simply by inserting register extend it to multi cycle datapath.

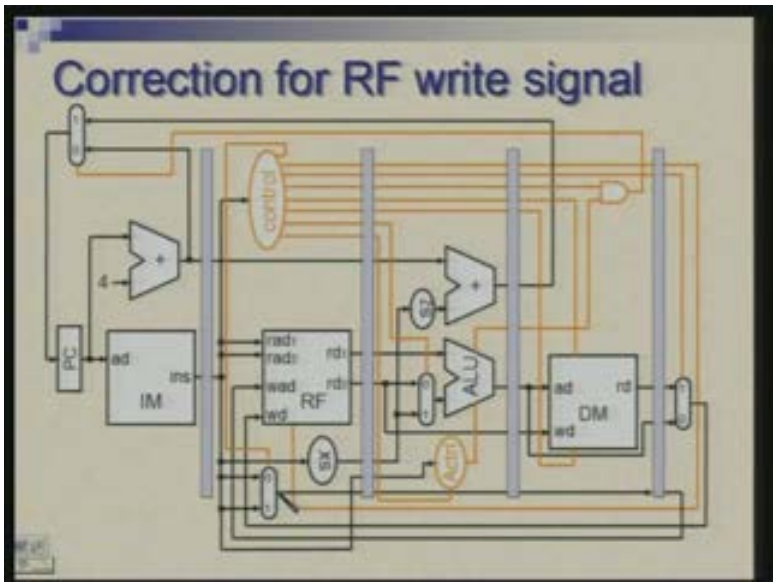
Therefore, now do you see this as the controller which is combinational circuit or it is a sequential circuit?

It is a sequential circuit because these portions of registers (Refer Slide Time: 25:47) through which control signals are passing these are in some sense carrying the control state. So let us see you have 1 plus 2 3 4 5 6 7 bits here, 4 bits here, 11 and 1 bit here 12. So effectively we have a 12-bit register which is remembering in some sense that what are the control signals required for instruction which is going to…… what are the control signals required in the next stage, what are the control signals required in the next to next stage and so on. So this is the storage element here and combinational circuit is identical to what we had for single cycle design.

Well, once again the there is some correction required because some signals have been not correctly timed and again you need to focus your attention on that WB activity. The signal required for WB is…… one is that you need to do multiplexing here so that signal has gone through three walls correctly but the signal which enables write of the register file is being sent directly so that is not correct. This signal (Refer Slide Time: 27:18) also needs to route through all these so go through all these and then get applied here.

Now this multiplexing is also actually relevant for write back but since we are multiplexing here within this stage and then the output of multiplexing is going through this wall so this is fine this can be taken out from controller directly (Refer Slide Time: 27:34) but this signal needs to be modified. So we will remove that and route this; I am just for convenience tapping it from here taking through these three registers and bringing back and applying register file.

(Refer Slide Time: 00:27:41)



So now everything is correctly timed; all the data, addresses and control signals and this is the complete design, datapath and control; of course we have taken out jump instruction but for the remaining eight instructions this is the complete design.

[Conversation between Student and Professor…..... (28:20)…… yeah, yeah yeah……..which……. you are talking of this multiplexer? Let us see what will happen if we do not multiplex it here. Suppose we were to position this multiplexer also here……okay okay…. what you are saying is; yeah that is an interesting point but there is a problem.

So what is being suggested is that instead of delaying the data you let the data be available here directly but you are giving write signal at appropriate time. Is that what you are saying? But see what will happen is that this address which you are computing and giving here will not stay there this would change; this register file is not actually keeping that address and internally holding it. So, if you just leave it like that as instructions change in subsequent cycles this output will also change.

So, while the instruction is here we need to pull out whatever we want out of this instruction code and whatever we need later needs to be carried through the registers. So therefore you need to do something here.

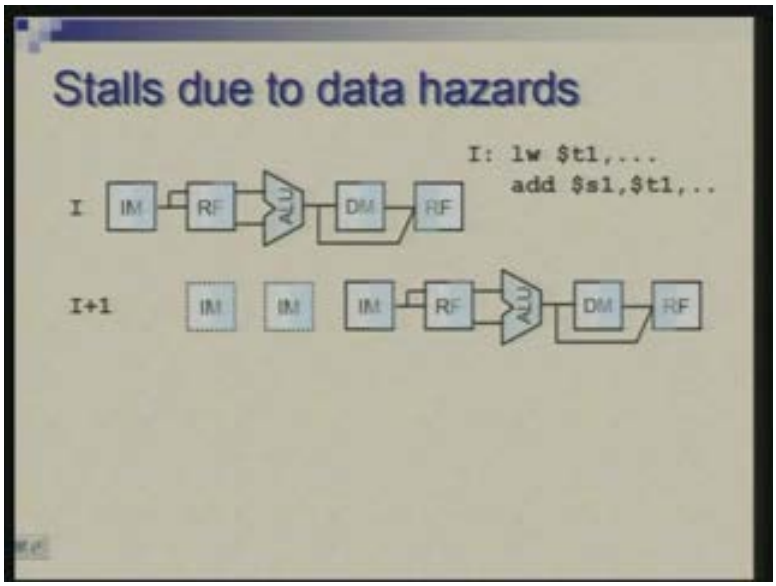
If you were to move this multiplexer let us say here one consequence was that there will be more bits you will have to pass through these registers; the other consequence would be that this control signal will also have to be passed and made available in that particular cycle. So that would actually mean that more bits are being passed through these registers and the other consequence would be where the delay is getting carried. So, if you keep it here that delay will get added to the register file write operation.

Right now this delay is not in series with anything else. From this register (Refer Slide Time: 30:47) you are going through multiplexer and then to this register that is a very small path so it is not bothering us at all.

[Conversation between Student and Professor:…… yeah, yeah, yeah, yeah, that is one possibility that if you if you store this after having generated this address you store it here but also remember that it is not only this address which you need to store. Suppose next instruction comes which also needs to write so that also will have to be stored so two three addresses will have to be stored and effectively what we are doing in these registers is the same thing; these registers are basically doing the same thing; only thing is that we are not storing it at one place; in one cycle we store here next cycle we move and store it elsewhere and so on. So, it is storing in a sort of a first in first out kind of an arrangement.

Now, having done this design we need to understand what is missing here. as you recall that last time we discussed that for datapath for data hazard and control hazards we have two approaches; either we introduce appropriate delays we suffer the delays we suffer the delays or we do something so that these delays are reduced.

(Refer Slide Time: 00:32:20)



You have data hazards which comes because there are two consecutive instructions with dependence among each other. That means the value computed by one instruction is used by other. Actually the instruction which may cause data hazard may not be consecutive they could be with one gap or two gaps depending upon the length of the pipeline. In a very deep pipeline when you have many stages the opportunities for data hazards are many many more.

So here in this case, for example, lw is putting a result in t1 which is required by add instruction. So we need to delay the register read stage of second instruction to the extent that it matches with register write or write back stage of the first instruction. That is possible because we are sharing a cycle between read and write of register file; half the cycle for writing and half the cycle for reading and if you do not do that then there will be one more delay. There are two ideal cycles or two bubbles we have introduced.

Now what will happen in this design is that we have not made any arrangements to check this situation and introduce delays. So, in absence of that what will happen is that an instruction which follows and there is a dependence it is going to read old results which are not valid and computations will go wrong. That is the problem which exists here. So what is required is that we need to first of all detect that there is a dependency. There are two instructions following close to each other we need to match their register fields.

Now what it means is that we would need to look at the register field of more than one instruction and therefore we need to pass on part of the instruction also through these registers. The relevant information……. So, for example, suppose we are trying to compare an instruction which is in this stage and instruction which is in that stage (Refer Slide Time: 34:43) let us say there are two instructions which could be in consecutive cycles we want to see if this instruction which is here is going to write into a register

which the instruction which is here is going to read from and we will not allow this instruction to proceed further if that is happening.

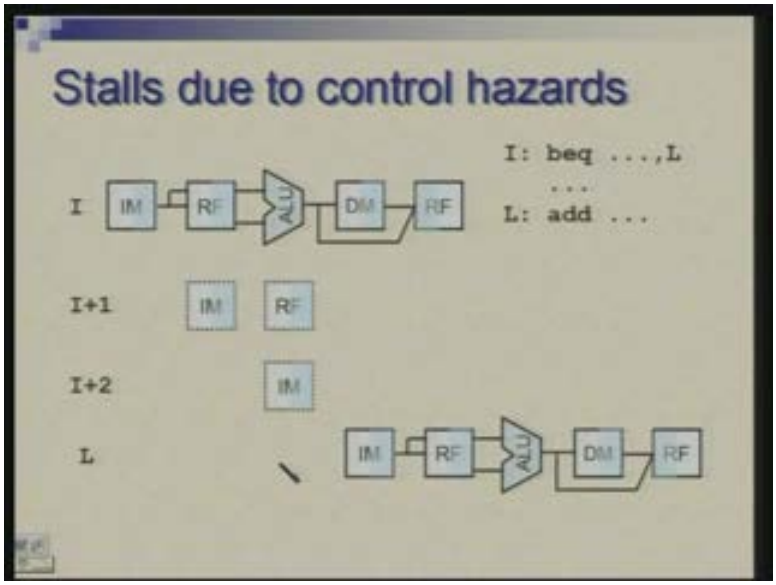
Now the read address of this instruction is available directly because instruction is stored here but the instruction which has reached here we have in this design we have not carried this information forward. So these fields let us say actually write field is here so that information is available here actually.

If we look at this information which is the write address for instruction which is now here compare it with both read addresses of the instruction which is here we can come to know if there is a dependency between these and as a result we would need to take some action. If we have forwarding path we need to enable those paths otherwise we need to halt this instruction and not allow it go through (Refer Slide Time: 35:58).

How do we do that?
Basically we require to control transfer of information into these registers. Every time you clock let us say this register it means you have passed the instruction which is here to the next stage. If you do not clock this here that instruction does not pass. So we need to keep the instruction here so we also need to disable the clock of this so that what is here does not change and what is here also, I mean that is immaterial so this actually…..

What we should do here is that we should put a no op instruction here. we should If we have designed all our control signals so that zero value means inactive what we can do is we can simply zero this zero the contents of this register. We will see how it is to be done perhaps in the next class but I am just giving the basic idea.

(Refer Slide Time: 00:37:10)



The second point was what is happening when control hazards occur. We have an instruction which is taking some decision and then deciding whether to follow

sequentially or go to some address L. So now if this instruction (Refer Slide Time: 37:30) is following through these stages normally there will be a tendency in the pipeline to start with I plus 1 and I plus 2; it is only in this stage you will realize that a mistake has been made by us and then you need to nullify these instructions; you need to flush these instructions and effectively the instruction labeled L is delayed it is starting at this point.

Hence, detecting that there is a branch instruction is very easy you do not need to look at two instructions here you just look at one opcode or you look at the control signal which is coming out indicating that it is a branch and then be ready to flush the instruction. So, as a machine there could be several possibilities here; one possibility was to just the moment you see a branch instruction freeze do not do do not bring in more instructions and at this point decide whether to start with L or I plus 1.

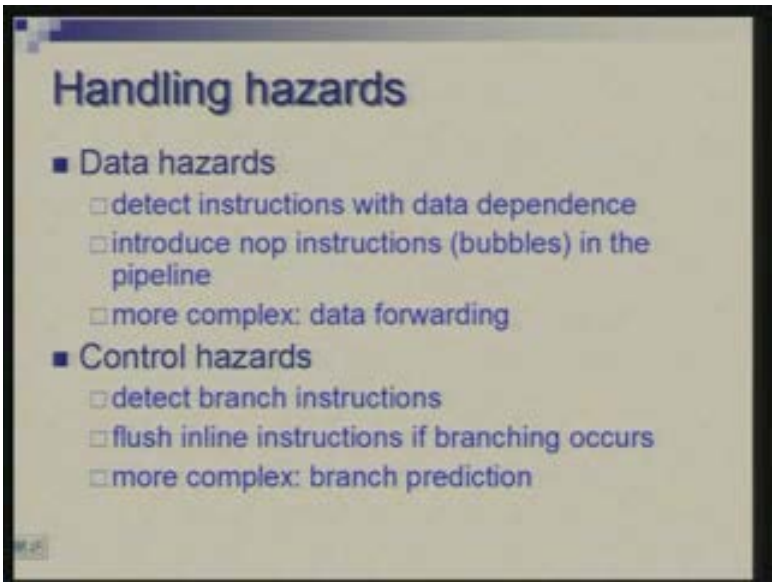
The other approach was that you let I plus 1 I plus 2 etc come through and if necessary you flush them out if you go wrong. Yet another approach was to do a prediction; at this point you decide what is likely to happen; is I plus 1likely to happen or L likely to happen and start with that when you know the correct thing and if you have gone wrong then you flush whatever it is. That is called branch prediction. It may be done statically that means there may be some logic some heuristic which you will use to decide whether to predict the inline operation or the branch operation. It could depend upon the opcode what kind of branch it is, whether you are branching forward or backwards and so on.

Typically what may be done is that if it is backward branch it means that it is a last instruction of a loop typically and most often loops are iterated several times so more likely you take a branch. In such a case you will predict that the next instruction is most likely helped.

Of course such an arrangement also means that you should be able to calculate the address L; either you should have stored somewhere or calculate. This is static prediction. But there is also a dynamic prediction where you keep track of what happened last time when the same instruction was encountered. This instruction could have been part of a loop and will be done many times. So last time…… it is a very simple prediction that last time if it was a branch taken you think that it may be taken is more likely to be taken this time also. Hence, this is a most naive way of dynamic prediction but there are more sophisticated ones than this.

So in any case let us just be clear about the actions we require in light of the hazards. We will work out the design changes to take care of these in the next class.

(Refer Slide Time: 00:40:33)



So, for data hazard the first thing is to detect instructions with dependence and you introduce suitable number of no op instructions. Sometimes a delay of two is required and sometimes a delay of one is required so in the pipeline where things are flowing moving smoothly you insert bubbles or no op instructions.

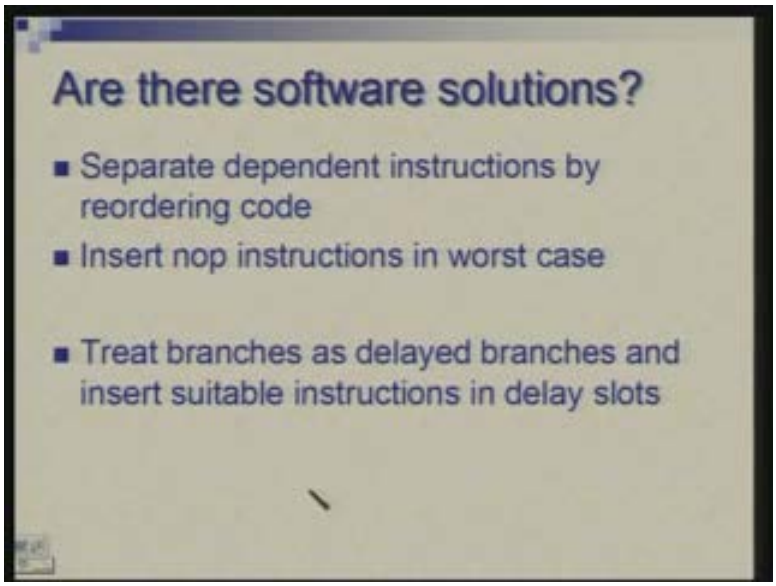
A more complex approach is when you have data forwarding path you have to suitably enable the paths. The introduction of bubbles may be required over and above this. Sometimes data forwarding will eliminate the delay entirely, sometimes it will only reduce. So, instead of two cycles of delay for example you may need only one cycle of delay so that delaying logic is still necessary.

[Conversation between Student and Professor: (41:30)……yes please…….yeah, all that is to be done. basically as the instructions are flowing at some the instruction which has gone beyond a point you will allow them to move whatever is behind that is held up so everything behind that is held up including fetching next instruction and moving it forward.

Yeah, yeah exactly, right; yeah, I am coming to that in the next slide actually….. (41:55).

So control hazards require that you detect, identify branch instructions and there may be need to flush wrong instruction which were brought in; either they may be you may blindly inline instructions or you may do prediction and bring something else. So in any case if the decision is wrong that has to be flushed out and predictions could be fairly straightforward, it could be very complex; dynamic prediction is somewhat involves much more hardware.

(Refer Slide Time: 00:42:37)



The last question where I want to leave is; are there software solutions. Given these hazards can we do something with the program? Because you will call that unlike structural hazards which we took care by introducing more hardware resources these the other hazards are coming because the way program is, the way the program is interacting with the hardware so can we do something in the software; the answer is as was hinted by somebody there is something which can be done in the software; the assembler or compiler whichever is generating the code can do the analysis and rearrange instruction so that the impact of these hazard is minimized.

For example, the data hazard is coming when dependent instructions are close to each other. So, often it is possible to rearrange instruction without changing the meaning; I think that is the key line that any rearrangement if you do you should not make the program go wrong. So, if instruction A is dependent upon instruction B the order cannot be changed. But two instructions are independent then it can be reordered. So, by reordering you can separate out instructions which are dependent; insert something useful between them which is independent and in worst case if you do not find anything else if you find nothing else then you can insert no ops.

Therefore, if instruction sequence is organized in this way that you do insert something if nothing else you insert no op then you can have hardware and not worry about data hazards. So hazards are Data hazards are removed by construction or the program itself.

How do we handle the branch instruction with similar spirit? After a branch instruction again you can put instructions which need to be done in both parts. Let us say you have if then else structure there is something you do in then part something you do in else part. If there is something common which needs to be done both ways then you can put those one or two instructions immediately after the branch

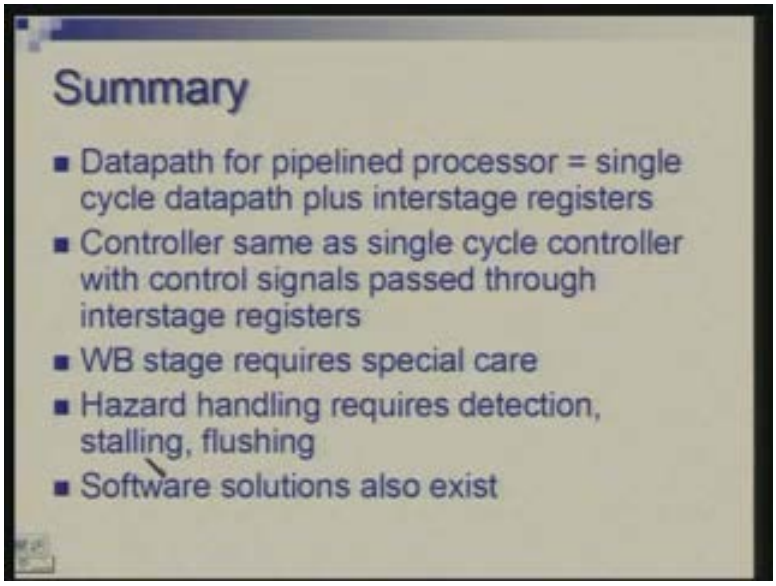
instruction and imagine that the branch has to be effective always two cycles later or three cycles later.

So, if these extra instructions if they are not available after the branch I mean if you cannot if you cannot find anything in common in then and else part then you might move it from the code which is before the branch as long as this is not dependent. You can sometimes bring something before the branch instruction and place it after the branch and these positions are called delay slots of a branch instruction.

So, suppose normally you are able to decide two cycles later what is the next instruction then this delay can be filled up by defining those as two slots two delay slots where you are going to use some hopefully useful instruction; if you do not have useful instructions to be put there you put no op. But the branch is going to be effective after these two; either you go to the instruction which was normally to follow branch here or the targeted instruction.

These transformations in the program can be done by the compiler, assembler or the programmer if programmer is writing the program directly and it simplifies the task of hardware. In the case of data hazard the data hazards need not be checked and in case of branch the hardware needs to just make sure that the branches are effective only after the delay slots.

(Refer Slide Time: 00:46:44)



To summarize what we have learnt today; we completed the datapath design and datapath design which we have is nothing but the single cycle datapath design with interstage registers and we have to make sure that every line goes through write number of registers so that timing is correct so carefully one has to do that.

The next thing was controller and controller also turned out to be same as the single cycle datapath controller but control signals were to be passed through the interstage registers so that they get timed correctly.

We need to take care of WB stage because WB stage involves many things getting fed back namely the data, control and address so they need to be timed carefully.

Handling of hazard requires detection, stalling, flushing. We can also have some support from the software which simplifies the task of hardware. So, often we have some combination of both which is used to minimize their effect. I will stop at that, thank you.