Digital Computer Organization

Prof. P. K. Biswas

Department of Electronic and Electrical Communication Engineering

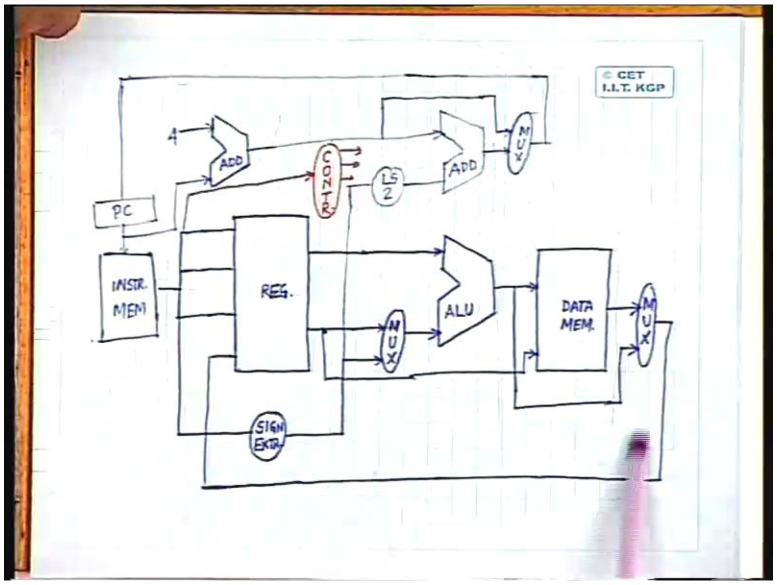
Indian Institute of Technology, Kharagpur

Lecture No. # 12

Pipeline CPU – III

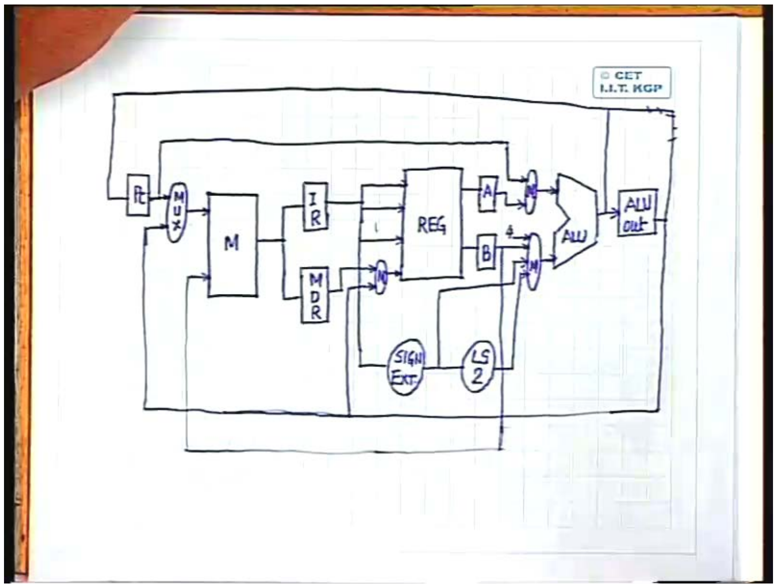
We are discussing about the MIPS processor architecture. We have developed one data path for a single cycle implementation of instructions and the data path as we have seen is like this.

(Refer Slide Time: 00:01:12)



In this one we have said that there are a number of data units which are duplicated. So when you go for single cycle implementation of the MIPS processor, there are duplicated data units. We have seen later that the other disadvantage of having single cycle implementation that is on an average, the CPU performance will become poor. So an improvement over this that we had tried is instead of trying for a single cycle implementation, we can go for multi cycle implementation.

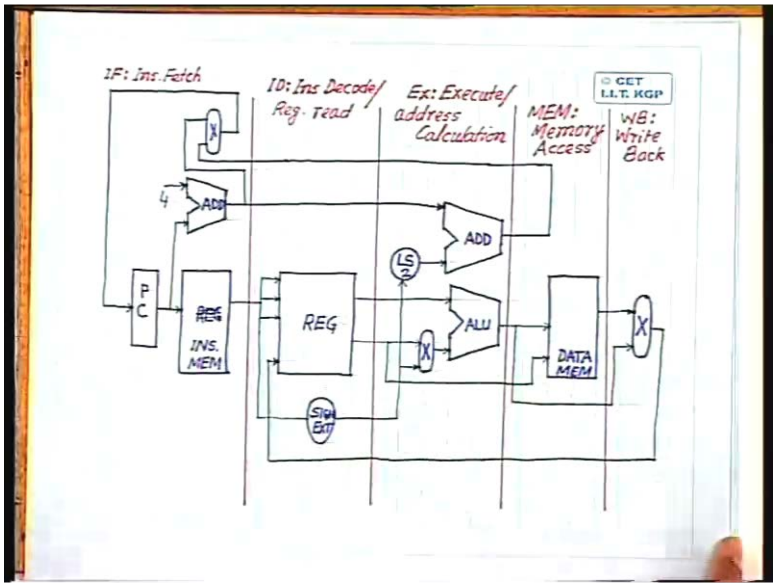
(Refer Slide Time: 00:01:51)



In multi cycle implementation what we have done is we have broken this data path into a number of steps. So there are 5 different steps and in addition to that, we can reduce the duplication of data units. Say for example earlier we had one ALU and a number of adder units and all the add operations are now being done only by this ALU. Of course because we are going for multi cycle implementation, so we have to have a number of intermediate registers for storing the intermediate balance. Then we just mentioned that after discussing this, that we will go for a pipeline data path. That means how this can be converted to a pipelined architecture?

Now as we have said in case of pipelining that more than one instructions will be in the pipeline simultaneously. Some instructions will be fetched, previous instructions will be decoded, the instruction which was fetched before that will be executed, instruction which was fetched before that will finally put the result in memory or in one of the registers. So these are different operations that will be performed by different instructions may be during the same clock period. So because we will have more than one instructions in the pipeline simultaneously, it is quite obvious that for pipeline architecture, it is better that not to go for optimization of the adder. That means instead of replacing all the adders with a single ALU, we will make use the same single cycle implementation architecture as this and try to see how this can be converted to a pipeline data path where we will have duplicated data units. So I have just redrawn this single cycle implementation of the architecture here.

(Refer Slide Time: 00:03:47 min)



So you find that all the units which were there in the single cycle architecture, all of them are present here. At this stage when an instruction is to be fetched, the data units which are involved is obviously the program counter. The program counter has to give the address of the instruction that has to be fetched. We have this instruction memory then we have this adder which increments the program counter after an instruction is fetched. Then we have this multiplex or selector which selects one among this implemented program counter or the branch address which has to be generated during the execution phase. So one of this will be selected by this multiplexer and the next program counter value will be loaded into a program counter where from the next instruction will be fetched.

Once an instruction is fetched, we have said in multi cycle implementation that during the next phase what we have to do is we have to decode the instruction. At the same time we have also said that we can read the registers which are specified by the instructions, the values of the registers which are required because we don't know what is the instruction that is to be executed until and unless the instruction is decoded. So this reading of registers is done in advance before knowing what is the instruction. We have also mentioned that this read operation does not harm the operation because even if the register values which are read, they are not needed. They will be overwritten subsequently by the appropriate values.

So here also during the next phase, the registers will be read, simultaneously the instructions will be decoded. This phase onwards whatever operations that has to performed depends upon the decoder output. So this whole thing as you see that this is nothing but the single cycle implementation of the processor that we have done. only thing that we have done here, only change is now this operations are more organized that is all the operations which are to be performed is broken into 5 pipeline stages. The first stage is responsible for fetching an instruction from the memory and this stage we call as IF or instruction fetch stage. The next stage is instruction decode or ID which we are specifying as instruction decode at the same time register read.

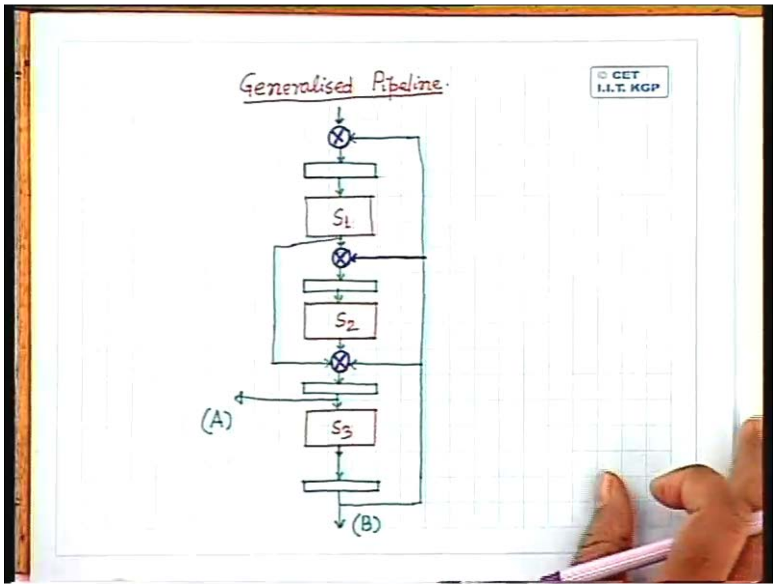
Third stage is in the pipeline which is the execute stage, we are calling it as EX or execute and at the same time if there is any memory reference operation to be performed, the address of the memory is also calculated. So during this phase, it will be the execution of instructions for R-type instructions as well as address calculation for memory if it is an intake instruction.

Fourth stage is actually the memory access stage. I mean this is the stage when either you write into the memory or you read from the memory. So we have to write into the memory if the operation to be performed is a store ward or SW type of operation. The data will be read from the memory it will be, if it is load word or LW type of instructions. The final stage that is the fifth stage in the pipeline we call it WB or write back. Write back means whatever is the output, the data will be written back into one of the registers depending upon whichever is the destination register that is specified within the instructions. When I discussed about the pipeline I said whenever I have so many pipeline stages, in between every stage I have to have pipeline latches. So here again between every stage, I will have a number of pipeline latches. So I will have a pipeline latch here, I will have a pipeline latch here, I will also have a pipeline latch here.

Now coming to this architecture, you will find that as long as the data flows in the forward direction that is from left to right. So this direction, flow of direction from left to right when all the instructions need the data flow from left to right, the pipeline gives maximum efficiency. But there are two instances when the pipeline will suffer that is when the information has to move in the backward direction that is from right to left. So we have two such situations. One is the right backstage. When the data has to be written from the fifth stage, has to move from the fifth stage to the second stage.

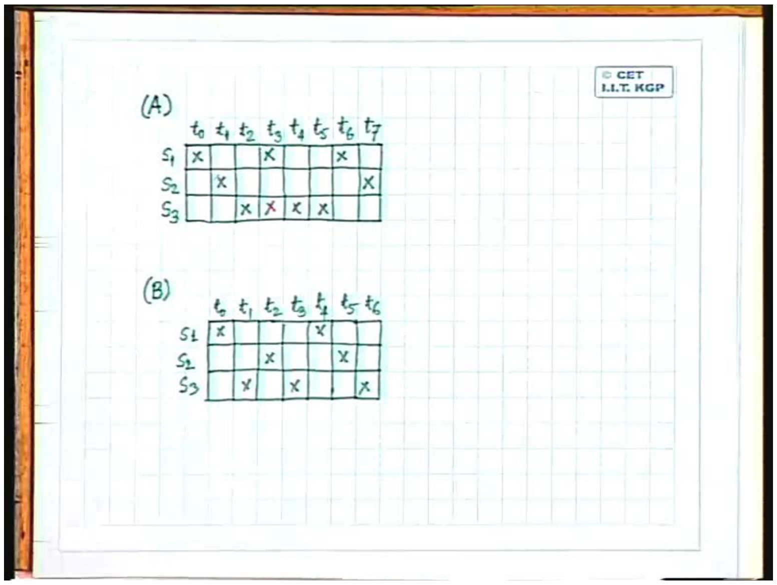
So this is one situation when the data will move from right to left and the second is this one that is case of any branch instruction, you are calculating the branch address here in the execution phase and the branch address that is calculated has to be loaded into the program counter in the first stage that is instruction fetch stage. So these are the two situations when the data has to move or the information has to move backwards in the pipeline. Whenever such situation arises, the pipeline performance is bound to degrade but you have to see that how we can reduce that degradation. Now one way that this can be handled is by means of the reservation table and you find that this kind of architecture is nothing but a special case of generalized pipeline that we have discussed.

(Refer Slide Time: 00:10:01 min)



So if you remember this figure, a generalized pipeline architecture where the data can move from in the forward direction, data can also move in the backward direction. Now for any operation how this different pipeline stages are to be used for execution for completion of a particular function that has to be specified with the help of a reservation table and we had given two such instances of reservation tables.

(Refer Slide Time: 00:10:34 min)



(A)

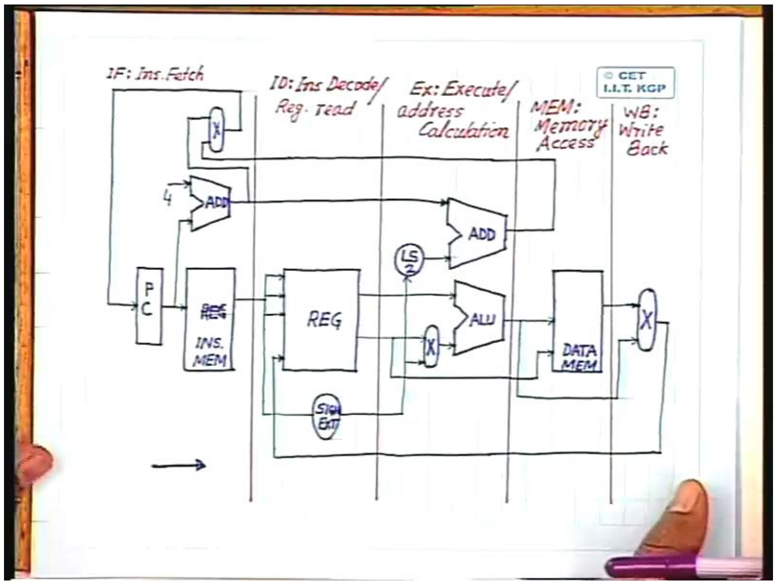
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
S_1	x			x			x	
S_2		x						x
S_3			x	x	x	x		

(B)

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S_1	x				x		
S_2			x			x	
S_3		x		x			x

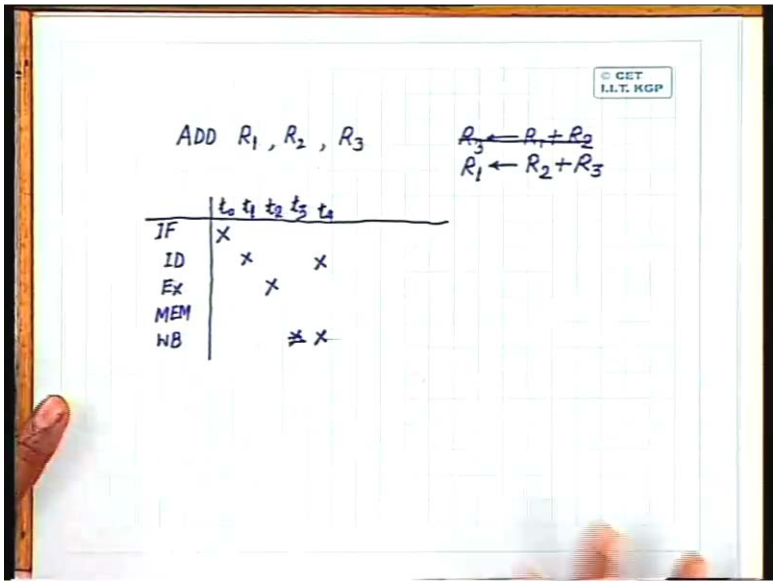
One for computation of function A and the other one for computation of function B. this reservation table is nothing but specifying that during which time stave which of the pipeline stages will be used for performing which operation. So if I go by this type of reservation table type of concept, in that case you find that all the instructions which has to pass through this pipeline will have its own reservation table.

(Refer Slide Time: 00:11:05 min)



Say for example if I go for, if the instruction which is to be executed is simply add R_1 R_2 R_3, if this is the instruction that is to be executed; the operation of this is R_3 gets the value of R_1 plus R_2.

(Refer Slide Time: 00:11:10 min)



ADD R_1 , R_2 , R_3

~~$R_3 \leftarrow R_1 + R_2$~~

$R_1 \leftarrow R_2 + R_3$

	t_0	t_1	t_2	t_3	t_4
IF	X				
ID		X			X
Ex			X		
MEM					
WB				$\cancel{\times}$	X

So what is the first operation? First we have to fetch this instruction. For fetching the instruction, the instruction fetch stage will be used. The second operation that is to be performed is decoding the instruction as well as reading the register values R_1 and R_2 which is to be performed during the ID stage.

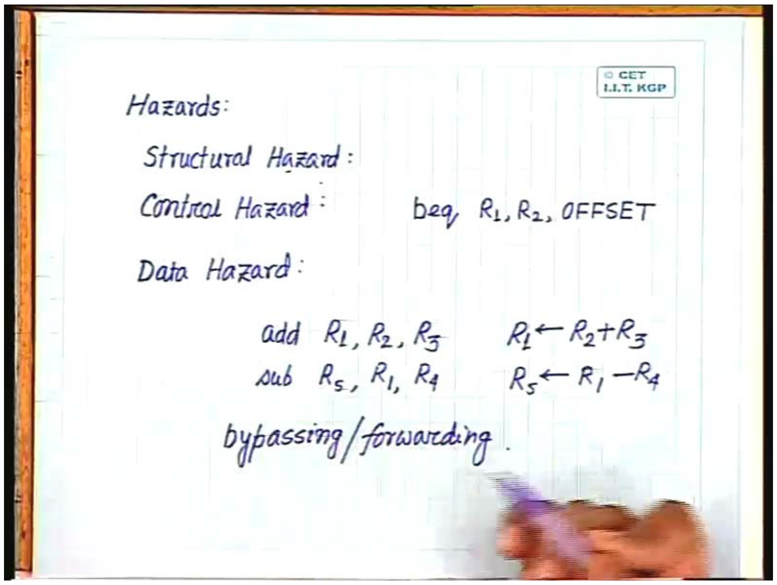
The next operation will be you have to add R_1 R_2 which is to be performed by the execute stage. Next is, the result has to go to R_3. So output of R_1 plus R_2 is available at the ALU output which will pass to this multiplexer. In the fifth stage, the data will be written back into register R_3. So this is how you find that all these pipeline stages will be used. Sorry here the operation will not be R_3 gets R_1 R_2 but it is R_1 gets R_2 plus R_3 because the first register which is specified in the instruction that is the destination register. However the sequence of the operations remain the same. So if I want to design the reservation table for this instruction, what I will do is I will have first pipeline stage which is IF then ID then execute, next is memory, the next is write back. Let us see how the reservation table of this will look like.

During time step T_0 this instruction will make use of IF stage. During T_1 the next time period, the instruction will make use of ID stage. During T_2 it will make use of the execute stage. During T_3 what is the stage that is used? Write back cannot be done during T_3 because now the data is available here. Memory operation is not needed, so during T_3 this instruction does not make use of any of the stages. During T_4 it will make use of this write back stage and the data has to be written into register, so the unit that will be used is ID only. So during T_4 again ID will be used, ID and write back together because you have to select the multiplexer properly. [Conversation between Student and Professor – Not audible ((00:14:45 min))] pardon. Because all the stages of the pipeline, information has to flow through all the stages and that is controlled by the latches. At every stage we have a latch.

So even if I am not making use of this data unit for transferring the data from this latch to this latch, I need one clock cycle and that is what is T_3. During T_3 I am not performing any operation, simply moving the data, the result from one latch to another latch. So this will be the reservation table for this ID instruction. Similarly, I can't (Refer Slide Time: 00:15:30). Sorry, this will be during T_4. Similarly, I can decide that what will be the reservation table for every instruction. Now once you know this reservation table for different instructions, then I can go for pipeline scheduling. That is once an instruction enters the pipeline, I can decide when the next instruction can enter. That is following the pipeline state diagram that we have already discussed. That is one way this can be done but however this can be improved. Let us see how this improvement can be made. When I talk about this pipeline then what are the problems that we can face in the pipeline.

The problems that you face in the pipeline are sometimes called pipeline hazards. One kind of pipeline hazard is called a structural hazard. So this structural hazard arises if the hardware does not permit combination of instructions that we want to execute simultaneously. Say for example when we had converted the single cycle architecture to a multi cycle architecture, you will find that in a single cycle architecture which is nothing but a variation of this without the latches. Here I have used an instruction memory and a data memory, the instruction memory and data memory are different. In multi cycle implementation what we have done? We have combined the instruction memory and the data memory into a single memory unit.

(Refer Slide Time: 00:16:24 min)



Hazards:

Structural Hazard:

Control Hazard: beq R_1, R_2, OFFSET

Data Hazard:

add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

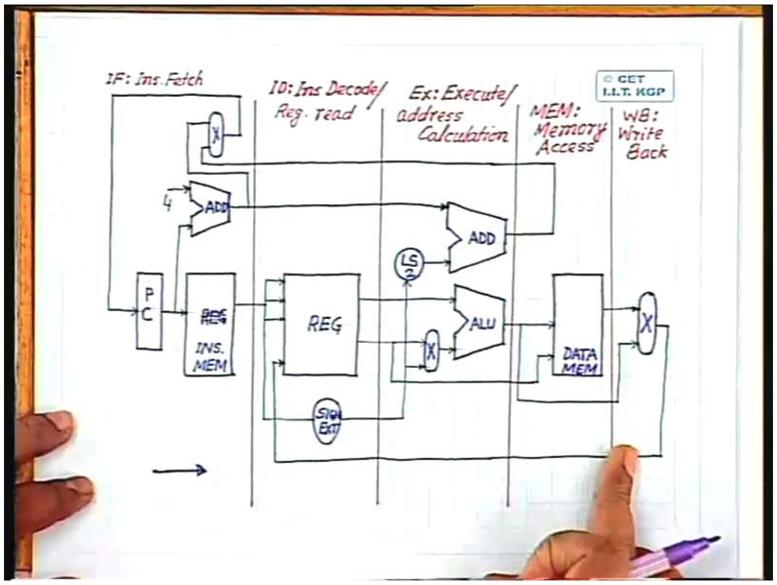
sub R_5, R_1, R_4 $R_5 \leftarrow R_1 - R_4$

bypassing/forwarding.

© CET I.I.T. KGP

Now if I use a single memory unit for storing both the instruction and the data then what problem we will face? Here you find that in this particular case, I can have a situation that one instruction can be fetched. Simultaneously during the same clock period, some other instruction can access the data memory either for reading the data or writing the data. If I combine these two in a single memory unit that is both the instruction and the data into the same memory, in that case such a type of operation will not be possible; either you can fetch the instruction from the memory or you can read data or write data into the memory.

(Refer Slide Time: 00:17:15 min)

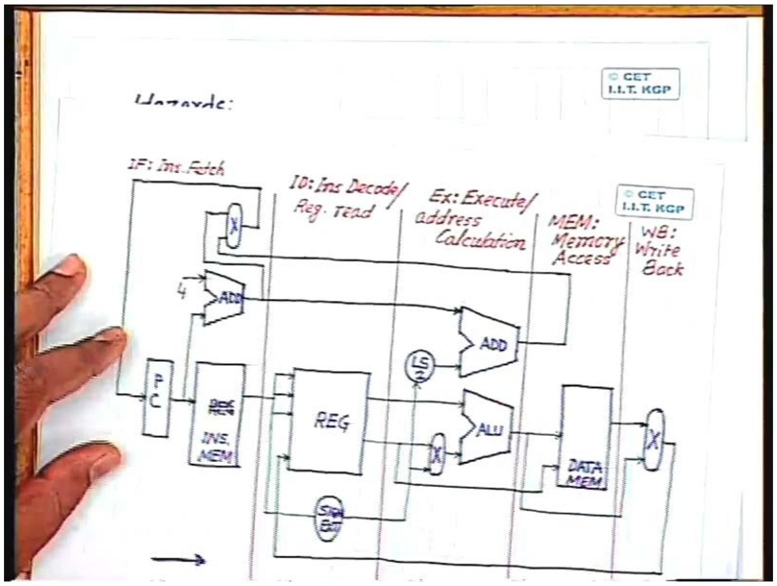


One of the two operations you can do, you cannot do both. That means fetching the instruction and accessing the data memory simultaneously is not possible. Whereas that is possible, if I have this instruction memory and data memory separately. So if I combine these two then obviously that leads to some bottle neck in the pipeline that is simultaneous instruction read and data read is not possible. This is a kind of hazard which is given by the pipeline and that is because of the hardware limitation. This kind of hazard is known as structural hazards. So one such structural hazard in this pipeline architecture is avoided by having separate memories for instruction and data.

The second kind of hazard that we can have in a pipeline is called a control hazard. Control hazard comes due to the fact that some decision has to be taken based on the output of some instruction while other instructions are already in the pipeline. Say for example a branch instruction. We have said, we have taken an instruction like this branch one equal R_1 R_2 then OFFSET. this is an example instruction that we have taken. Now here it says whenever R_1 and R_2 are same then you have to take the branch and the branch address will be the OFFSET value away from the current program counter value.

Now find that whether this branch has to be taken or branch is not to be taken that depends upon the output of this instruction. That means whether R_1 and R_2 they are equal or not that you will come to know only in the execute phase. So if you have to decide that what will be the next program counter value only after execute phase, that means at least for two clock periods the pipeline will not be able to take new instructions. This is the problem which is called pipeline stalling, stalling the pipeline. That means you are not allowing new instructions to enter the pipeline and that problem will always be there because I can never know without executing the instruction whether the branch is to be taken or branch is not to be taken. but we have to find out some optimization. So what is the optimization that can be made? One optimization is branch prediction that means to predict whether the branch is to be taken or branch is not to be taken. In the simplest case, we can assume that branch is not to be taken.

(Refer Slide Time: 00:21:40 min)



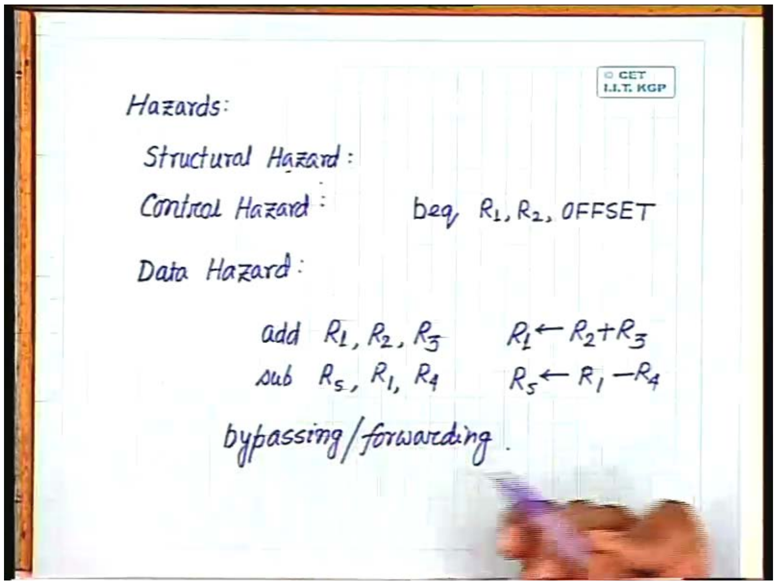
So whenever this branch one equal R_1 R_2 offset that enters this pipeline that goes through all the stages. In case of stalling what we have to do is until and unless this stage is complete, that means until and unless we know that R_1 is equal to R_2 or R_1 is not equal to R_2, we don't allow new instructions instructions to enter the pipeline. In case of this prediction when I am predicting that branch is not taken, I will not stop instructions from entering the pipeline. When the first instruction that is branch one equal that moves from IF to decode stage, the next instruction from the sequence will enter the IF stage. My prediction is branch will not be taken. So the next instruction already enters the IF stage.

When the branch one equal comes to this execute stage, the next instruction which was in IF stage comes to ID and the third instruction will come to IF stage. Only after this, I will come to know whether the branch will be successful or branch will not be successful. In case the branch is not successful then all these instructions will pass through the pipeline as it is. In case the branch is successful in that case, the time taken by these two instructions in this pipeline stages will be wasted because in that case the next instruction that will be fetched into this IF stage that will come from the branch address, not from the same sequence in which the instructions are already there.

In this particular case, possibly it will not lead to any problem because the operations that you are doing in ID is only registered read and we have already said that even if you read the registers, you are not executing on them. You are not doing any performance on them. So possibly the operation that you do even in this ID stage that is not harmful. But there can be situations in case of other pipelines that whatever operation you do in these two stages, that may already modify some of the data which is harmful. So that has to be carefully looked into whenever you design a pipeline, if you have any such data modification during these stages. However for our example that is not harmful, only when the time taken by the instructions in these two stages are wasted. [Conversation between Student and Professor – Not audible ((00:24:16 min)) how do you make sure that those two instructions which we attached will not be executed] No, they have come from where? IF to ID. Yes sir. IF to ID, then the instruction which has come to ID that has to go to execute stage. Yes sir. So I can have the control unit, if the control unit decides that the branch is to be taken that means my prediction was wrong. What I will do is I will not load this to this latch. So that it does not enter the execution stage because the moment it enters the execution stage, it is going to modify the data. So that I will not permit. That is to be taken by the controller. So this is one approach, when you predict that branch is not taken. The other approach can be that branch is taken, that is also another prediction. This can be further defined based on the history.

What happens is whenever some instructions are executed, you maintain the history of the instructions which are executed. The situations become more complicated but more sophisticated. So in the history whenever such a branch instruction is encountered, you go back to the history to check earlier execution of same branch instruction, what happened. Whether the branch was taken or branch was not taken? If you find that in earlier case branch was taken, you assume that branch will be taken. If in the earlier case you find that the branch was not taken again then you assume that branch will not be taken. So accordingly you select that which instruction next has to come to this IF stage. However these are all refinements and obviously in this case you need extra amount of memory, additional memory to maintain the history. So obviously there are some advantages and disadvantages in every approach, so that is about the control hazard.

(Refer Slide Time: 00:26:23 min)



Hazards:

Structural Hazard:

Control Hazard: beq R_1, R_2, OFFSET

Data Hazard:

add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

sub R_5, R_1, R_4 $R_5 \leftarrow R_1 - R_4$

bypassing/forwarding.

The third kind of hazard that we have in case of pipeline is called data hazard. Now what is this data hazard? Suppose I have a sequence of instructions like this add R_1, R_2, R_3 and we have said that this operation performs addition of R_2 and R_3 and the result goes to R_1. Suppose the next instruction is subtract R_5, R_1, R_4. This is the next instruction and this will perform the operation, R_5 gets the data which is R_1 minus R_4. What will be the performance of this pipeline in this case because you find that the second instruction cannot be executed until and unless R_1 is written into because only then the right data is available for this subtract instruction. Now coming to this pipeline, what happens? the first instruction subtract will come here instruction fetch, then it will come to ID, then it will come to execute, then it will pass to memory and only in the write back stage R_1 will be written back into the stage interference.

Coming to the second instruction, you will find when the add instruction is in this stage that is in the memory stage, the second instruction is in the execute stage. So when the add instruction is in the memory stage, the second subtract instruction which is the immediate next that is in the execute stage. and in the execute stage it expects the correct value of R_1. Isn't it? But the correct value of R_1 has not been set yet. that will be set only after the write back that is in the fifth stage. So if we don't take any extra precaution then this subtract stage will utilize the previous value of R_1 because the correct value of R_1 has not yet been restored but in R_1 you have some value. So this subtract operation will make use of the previous value of R_1 which is the wrong one to give you a wrong result into R_5. Isn't it? So naturally this is again another kind of hazard and no compiler will give you this error. This error will come only at the output. We have to take extra precaution to take care of such problem. So how this can be done? [Conversation between Student and Professor – Not audible (Refer Slide Time: 00:29:34 min))] What can be done is you find that when the subtract instruction is in the execute stage, at that time the value of R_1 is available at the out of ALU that means at the output of this latch. So here what we can do is I can bypass these two stages, for R_1 value. What I can have is I can have a feedback from this to this ALU input. That means here I have to have a multiplexer, one more multiplexer. One of the inputs to the multiplexer will come from the register file, the other input to the multiplexer will come from the ALU output itself. Now the controller has to decide that which one to select. So whenever such a type of data hazard situation comes, now the controller has to select the ALU output to be fed to ALU input and the subtraction operation can continue as needed. So while doing this, what I am doing is effectively I am bypassing this memory access stage. I am also bypassing the write back stage, getting the data directly into the ALU input from the ALU output itself. So this is a mechanism which is called a bypass mechanism, sometimes it is also called forwarding mechanism. So solution to this hazard can be by bypassing or forwarding.

We can have these three kinds of hazards in the pipeline architecture. The structural hazard, as we have said that this can be avoided by duplication of resources however it is not possible to avoid it always. There will be some situations in which structural hazards will come and in that case pipeline will suffer but by duplication of resources, we can ensure that the performance degradation will be as minimum as possible. Similarly we can have control hazards and this situations are bound to come, we cannot avoid it. But what can be done is in such cases, I can move this adder which is used for computation of branch address to this place to this place. Now this adder is in the execute stage. If I move this from the execute stage to ID stage because even in ID stage all of them are available, whatever is this offset that is available, whatever is the next program counter value that is also available. So if I move this adder from the execute stage to the ID stage, then i can even save one clock period for taking the branch. So when this control hazard can be removed but it cannot be totally avoided.

The third one is data hazard. For avoiding the data hazard, we can make use of the bypassing or forwarding mechanism. In this case what is needed is in this register to ALU circuit, register to ALU data path I have to make some more modifications. That means I have to put one more multiplexer here, control circuit has to be designed accordingly so that control circuit will identify, can locate such data hazards and generate the control signals accordingly, so that the data to the ALU is available at the appropriate point. Is that okay? So with this, I will stop this pipeline architecture and the processor architecture as such. So next class onwards we will talk about the other components of the computer system.