Advanced VLSI Design

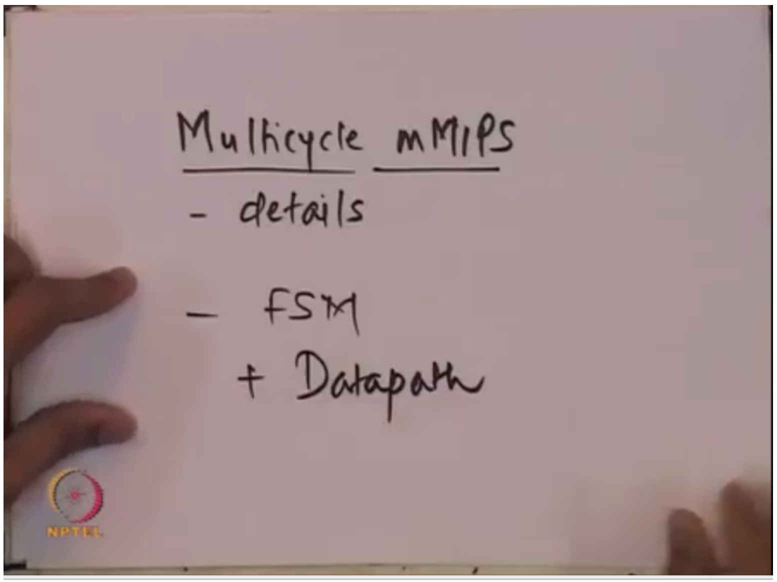
Prof. Sachin Patkar
Department of Electrical Engineering
Indian Institute of Technology- Bombay

Lecture - 28
Multicycle MMIPS? FSM

So now we are ready to discuss the details of multicycle implementation of microMIPS. That means the details of the datapath and details of the controller. Details of the datapath could mean we have already identified the main datapath components that is, the certain registers like program counter and instruction register, memory data register, the register file, memory blocks, ALU plus some more registers that we identified so far.

But in the datapath we also have these router components like multiplexers and some glue logic would be there. That those kind of details will try to get right now. Also so far we have got a good picture of finite state machine that would control this datapath, how the data would be processed, by which datapath component, the direction of the data, the routing of the data, which source will be used on a multiplex connection signal.

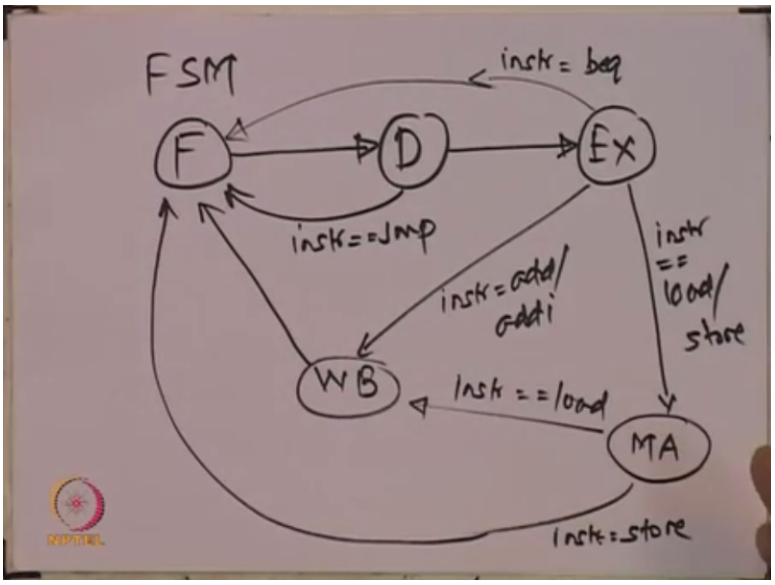
(Refer Slide Time: 01:40)



Multicycle mMIPS
- details
- FSM
+ Datapath

So we are ready for that now. So the topic is multicycle mMIPS the details okay. So there is an FSM whose not all details, but just main details will try to work upon plus the details of the datapath.

(Refer Slide Time: 02:10)



So recall the FSM, the FSM we drew it as rough in a very natural way it evolved, there was a fetch state doing the fetch kind of sub operations related to instruction fetching from the memory and also putting the instruction into the instruction register. That is the kind of activity that would happen during the fetch state of the system. So if controller will have fetch state and it will creates control signals.

So that the datapath would do necessary operations on the memory and like you know (()) (02:46) instruction coming out of memory into the instruction register and so on so forth. There is more to it. After fetch in the next clock cycle, every instruction would go into the decode stage okay. Then if it were a jump instruction after decode jumps we could arrange things so that the jump instruction cycle would be over and would go to the fetch state of the next instruction cycle.

So this is if during the decode we find out that it is jump instruction, then we finish the work and in the next clock cycle we would be in the fetch state for the next instruction cycle. If it were not a jump instruction, then every instruction cycle would be in the Ex state execute, that is where the ALU is involved, address computations take place, arithmetic computations take place you know and so on so forth.

Now again like what next after the Ex state, if it were the branch instruction, branch equal to Let us say, then we could arrange that the Ex state itself computes the like you know finishes the work of the execution of the branch instruction branch if equal to because in this state it would be able to check whether the two operands are equal or not and in the previous clock cycle that is in the decode stage, it would have found the target address with the help of ALU again.

And now it would be ready to commit whether the new target address has to be loaded in the program counter or not so the work of the branch equal to instruction, the instruction cycle of branch equal to could be arranged to be over finished after this state. So we could go to the fetch state of the next instruction alright. If for other instructions, let say if the instruction is of arithmetic type, add or add immediate okay.

In that case, we do not have any role of memory. There are two more states not every instruction cycle goes to all the five states, but there is a write back state that an instruction like add or add immediate would go to after execution. This is where the datapath should be configured so that the result of the ALU would be registered into appropriate register of the register file okay.

And after write back stage, this instruction or any instruction that arrives in this state would be at the end of the instruction cycle and next state would be the fetch state of the next instruction cycle so that is why this arrow. For the other remaining situation, if the instruction where either load or store, then the execute state would have computed the addresses of the locations in the memory from where to load or to which to store okay.

That could have happen in the execute stage for instruction like load and store. Yeah so in the next clock cycle, it would be in the stage of doing the memory access with the help of the address that it has computed in the previous Ex state, but now if it were a store instruction, then there will not be any further work remaining. With the memory access the store instruction cycle would have archived the or stored the whatever it wanted to store that was from the one of the registers that would be stored into the memory location whose address has been found so far is available.

And then it would be at the end of the instruction cycle and the next state would be the fetch state of the next instruction cycle. That could be in case of the store instruction. If it were a load instruction, then after the memory access the load instruction would have the data, which is to be return back into the register file. So it would go to this state okay. So this is how like a state diagram would like for in case we are supporting only this few instructions like JMP unconditional jump, branch equal to, add, add immediate, load, store and so on so.

In fact, you know what is limited choice of instruction that we are describing here is purely for the sake of simplicity of the presentation. The datapath would be capable of supporting many more instructions. The FSM can be easily modified using the same template for many more instructions. In fact, for the later discussion, we will be even removing this jump instruction support for our toy CPU okay and that can be considered as an exercise for you to work upon okay.

So will not see henceforth see the jump picture in the next few slides or next part of the discussion for a while okay. So this is FSM yes I mean you will see a slightly different version of this FSM in different text books or like you know even next I would be describing a small variation of it for the sake of convenience or simplicity, simplifying the implementation logic okay, but the essential FSM is this.

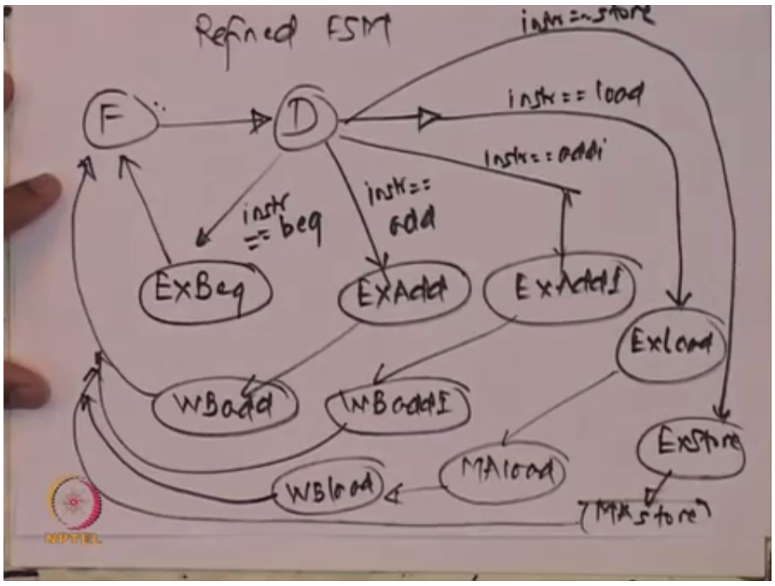
So these are only the states what more do we require on FSM. In fact, the main purpose of FSM from the perspective of the outside world is that FSM should generate control signals based on the inputs that it sees. What are the inputs of this FSM? Clock input, so many of the transitions are being made independent of any like when the clock triggers, the transition is made like you know F to D is a transition that always happens at the triggering edge of the clock.

But D to Ex may or may not happen depending on whether the instruction jump or not and for example other example is WB to F, this will trigger at every triggering edge of the clock. It will not depend on any other input, but many other transitions you see that they depend on what the instruction for the type of the instruction, so this can be regarded as input to this FSM.

What about the outputs? The output of the FSM that is the main thing, that is the main thing from the datapath point of view and the CP implementation point of view. The control signals are going to be generated by this FSM to guide the datapath, to control the datapath differently in different clock cycles and different instruction subcycles. So those are the details, which we need to work out and for that we might choose to simplify this state machine not by reducing the number of states that would complicate a lot, combination logic that one has to implement, but by you know refining the states into sub states.

So that the combination logic as you know it is a heart of FSM implementation, the combination logic that generates the next state as well as the combination logic that generates the output that would be simpler and easier to kind of analyze or make efficient and so on okay. So that is what we will be seeing.

(Refer Slide Time: 11:15)



So the refinement is as follows so I call it refined FSM. The refined FSM would have F state as it is, as I told you we will dropping jump out of the picture F D, this transition is always there at every triggering edge provided you are in any instruction cycle that is in the fetch state would go into the decode stage at the next triggering edge of the clock okay. Then we are not considering jump, but after D we will go to Ex.

Now here we make a separation. We go to different kind of Ex stage because the behavior inside the Ex execute state how the ALU is to be configured, how the datapath is to be configured, would be different for different instructions. So now we make the bifurcation here. So if the instruction were beq then I said I create a refined version of state Ex called Ex for beq, a cute arbitrary name.

Name is not important so it is clear from what I chosen. It is a Ex state of the beq instruction okay. After this so the instruction cycle for beq could be in F state, then D state and then Exbeq state and then it would go here okay. I just mention why are we like in a refining the state Ex into several such states is because that would make our task of describing the control signal outputs mostly like easier to implement, easier to describe, analyze mainly okay.

Then from the decode state, if the instruction where add then we would go to the Ex sub state corresponding to add instruction. Most of behavior in these states will be same or much of the behavior, but there will be difference which is specific to exactly which instruction, so how the arithmetic is being done by the ALU, how the ALU is being controlled and so on. So that is why I am going to have different.

There was nothing wrong with the previous FSM that was probably the most compact in terms of number of states, but when you try to minimize number of states there would be a trade off in terms of the complicated nature of the combination logic that we will using for generating control outputs and so on so. So here that is why we have chosen this approach and several text books will also suggest that or there might be something intermediate between this extreme and the other extreme.

So similarly if it were add immediate then I have a state called ExAddI then if it were load, then Ex load, if it were store then Ex store, this is routinely I am creating states okay. These five states are the refinements of the original Ex state. Now if we are in the Ex state of the add cycle then the next state would be write back state. I call it WB for add. Similarly from here I would be in write back for add immediate.

Now if it were a store instruction or load instruction, the next states would be memory access, memory access for load and memory access for store. In fact, it will turn out that this two states memory access for load and store are absolutely identical because what we do or other Ex load and Ex store would have been identical because there precisely what is happening is not this.

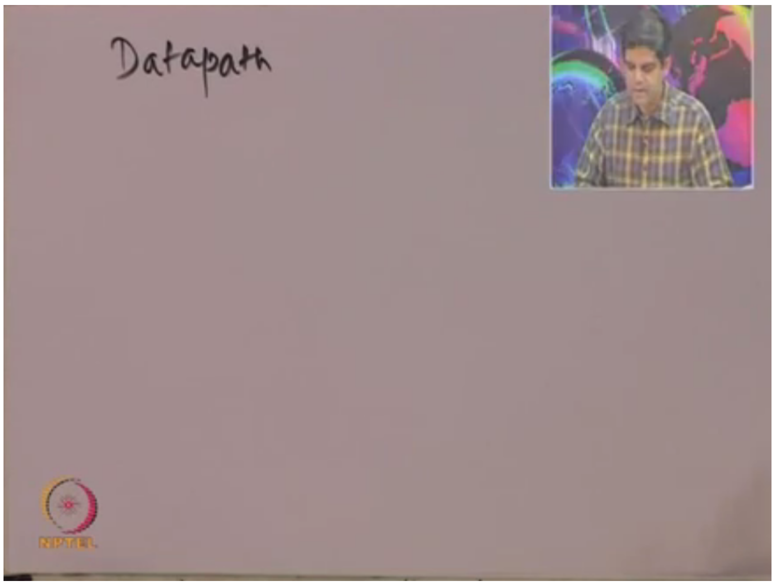
But the Ex load and Ex store, which we have shown separate states here they would be identical because what would be happening in the state is just simply the ALU would configured to generate the memory address, which is the addition of the contents of these first source register that is A and the immediate field of the instruction, which is the sign extended and multiplied by 4 or shifted left by two bits.

So these two states could have been like merging to one that is the kind of FSM minimization that can always be attempted, but that is not the main focus of this so I am like you know taking the easy way out. So I have created so far 11 states. So then most of the states are the terminal states of the instruction cycles like after write back of add immediate will go to the F state of the next instruction cycle.

After memory access of the store where something is going to be stored in the memory that is the end of the store instruction, so we will be at the end of the instruction cycle so we go here. For the load case, one more state is required that is WB load okay and then we will go here okay. So there are 12 states that we have created out as a refinement of that FSM. It will be larger details but slightly simpler to read out or like analyze yeah.

So still these are not, we have to work out details of what control signals would be in these individual states, but we have got a more clearer picture of what should be happening inside. Otherwise in the older FSM what would be happening inside, Ex state would naturally depend on whether the instruction is branch equal to or whether it is add immediate, add, load store and so on so forth.

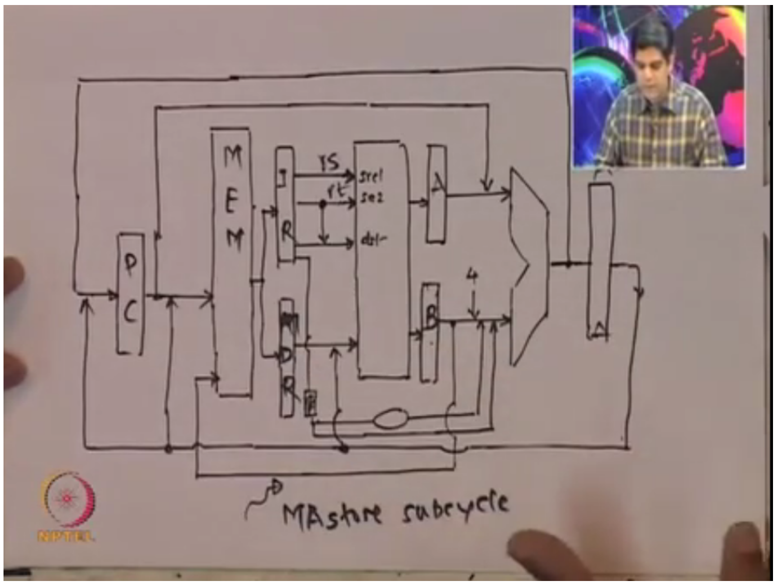
(Refer Slide Time: 18:30)



So now the things have been separated out. Now Let us go to datapath again. So far we have noticed that we could do with instead of two memory blocks we could have just one memory block and we instruct three ALU or adders, we just could do with only a single adder, single ALU. So we could see some resource optimization, but there was a bit of price to pay and that was in terms of some extra registers that we require.

So that the results of the sub operations like fetch decode, execute. Those sub-op results would be archived in some registers like A register, B register IR instruction register that is, memory data register then like ALU result. So few registers had to be brought into the picture and we had more or less evolved that datapath.

(Refer Slide Time: 19:32)



Now we will show a complete picture of it here. Still little abstract without details especially the names and so you do not bother about not being able to exactly see the details of the wires and where they are leading to, where the signals are coming from, but I will just read out otherwise there will be just too much clutter if I put in too many details, which are not really essential for understanding.

So Let us recognize whatsoever here. This must be our program counter right and what is being shown as the input of program counter, there is a signal coming in, but that signal there are two possibilities either that signal comes from output of the ALU that could be ready to go into and to be registered into the program counter or they something that comes from here, which is also a kind of the result of ALU, but a default result of ALU right.

The ALU result, which has been put in the register so what will be coming out of here is the ALU result of the previous clock cycle. What is coming out of here is ALU result of the current clock cycle that is the difference. That is the subtle difference will realize why we need that. So these are the interesting subtleties that we come across when we design such datapath, multicycle and FSM controller and so on so forth.

So this our program counter, yeah I could start writing this okay. So now this clearly is the memory block right and this must be the address input of memory okay and this must be the data output of memory, which can go into either instruction register or the so called memory data register MDR, memory data register it is not clear here, but read it as memory data register.

So this is the data read from the memory, so this is the address port and this must be the like data input of memory where it could come from this B register only from here okay. There will not be any other source from where data arrives, which is to be stored into the memory. So this must be on behalf of the memory access store subcycle or state okay. See that the contents of B register are being brought in over here at the data input port of memory and it will be stored into the appropriate memory location in the memory block okay.

Recall what is in A and B, this is A and this is B, so in A we have the data registered from the register of the register file whose index is specified by RS field of instruction register okay. And in B register in the decode stage, we would have read out the contents of the register in the register file, which is given by the index coming on this 5 bit lines at this okay that means RT and so on.

So we know A and B what the purpose was, so we will come to the details of individual subcycles few typical ones and then will leave a few of them as exercise for you. There are just 12 states, so we need to do this exercise that will be doing about understanding each subcycle separately at most 12 times. We will do it a few times and then we will leave the rest to you.

I will do it. So far I have not yet started that, but I am just trying to help you recognize what is there on this datapath. So the PC, there is memory block, there is instruction register, there is a memory data register, who fills it up, when does it get filled up, it gets filled up in the memory access state of I mean it is to be filled up at the end of the memory access state of load instruction okay.

And then this is register file alright A and B registers. This is the ALU, which you can and this is the register called ALU result. Name is not so important. It is something maybe I should call it really ALU result differed okay. So this is going to store the ALU result not of the current ALU operation, but the ALU operation that happened in the previous clock cycle. So this is registered result of ALU. Please note that I am not showing clock inputs to all these registers.

There will be clock inputs to A, B this register, IR, MDR all of them are clock registers right. So they are registering the loading of data into the registers is happening at a triggering edge of the clock provided the registers are enabled by the enable inputs okay. So more details are to be seen plus we also identified the role of multiplexes. You can see that like over here there are two possible signals to be loaded into program counter.

And depending on the state of computation or state of instruction subcycle appropriate with either this or this will go into it and so there must be a router multiplexer over here okay. There must be a similarly a multiplexer over here at the address input of the memory okay. Here when the ALU result is also possible signal that can be used as an address of the memory input.

You can recognize why, when and all the contents of PC itself could be sometimes used as input to the address port of the memory block when instructions has to be fetched right. So there are some more multiplexers like over here. There is a small multiplexer here that would decide what should be the index of the destination register where the data is to be written into in the write back state of any those few instruction cycle.

So whether the destination register address should it come from the RD field or should come from the RT field okay. So this line corresponds to the 5-bit RD field of instruction register and these five bits correspond to RT field of them. So the destination can sometimes be provided by the RD field or sometimes can be provided by RT field. Again you can quickly think about like when you know when these different things will happen.

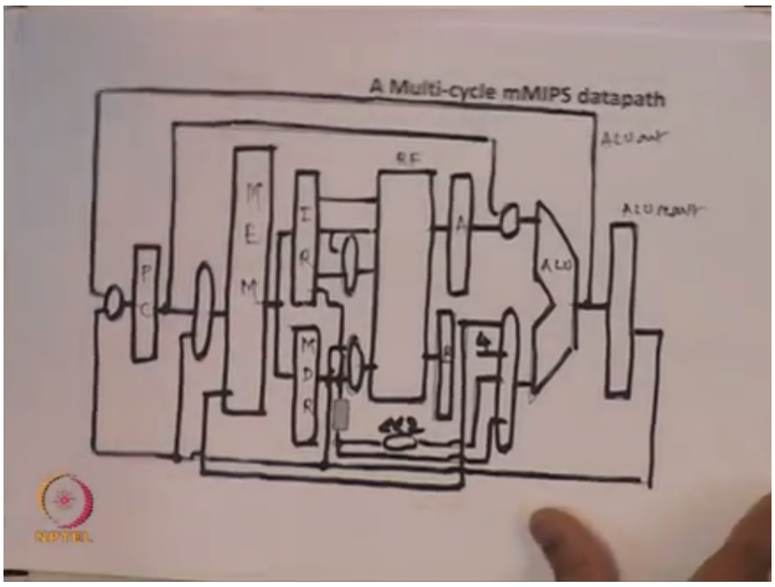
Will come to those examples again. There is one more multiplexer over here okay. So we had already evolved most of this previously when we were discussing trying to evolve a cool picture of FSM and datapath in the earlier portion of this lecture module okay clear I hope.

(Refer Slide Time: 27:28)

Datapath control signals

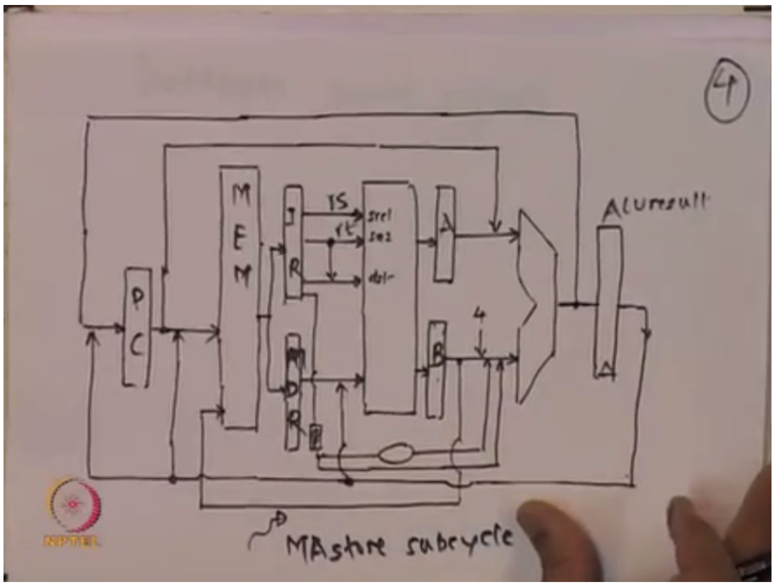
Now Let us try in work towards some more details of this datapath and those details will make us visualize where these control signals controlling the datapath are arriving or where are they like arriving at in the datapath. For that we will have to like you know enhance the picture a bit to get more clear picture of the multiplexers, where the multiplexer are.

(Refer Slide Time: 27:56)



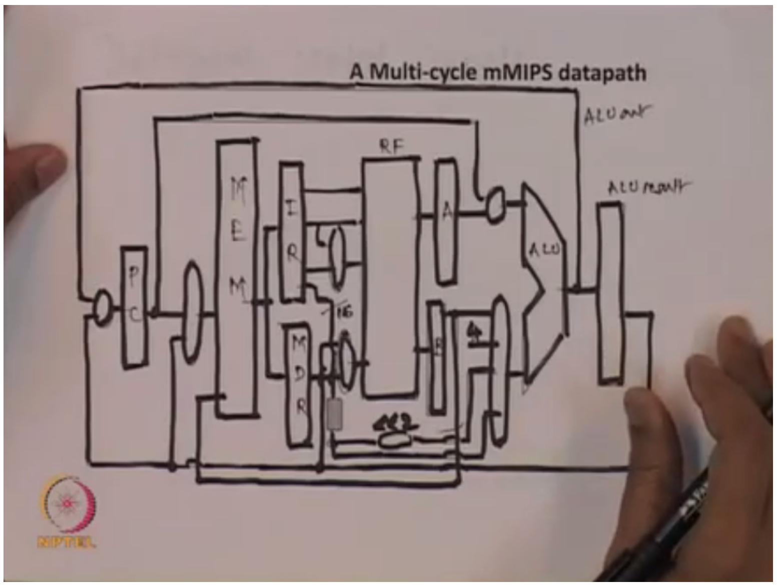
So here in this diagram, which is same as the previous diagram excepting that like over here we had not shown the multiplexers explicitly.

(Refer Slide Time: 28:03)



Say for example over here and big multiplexer over here you see that there are four possible sources to this particular second B port of ALU either it really comes from the B register or the constant four or a pair of signals. You know there are four options over here, there will be four to one multiplexer required over here. So I have not drawn that in this abstract diagram.

(Refer Slide Time: 28:35)



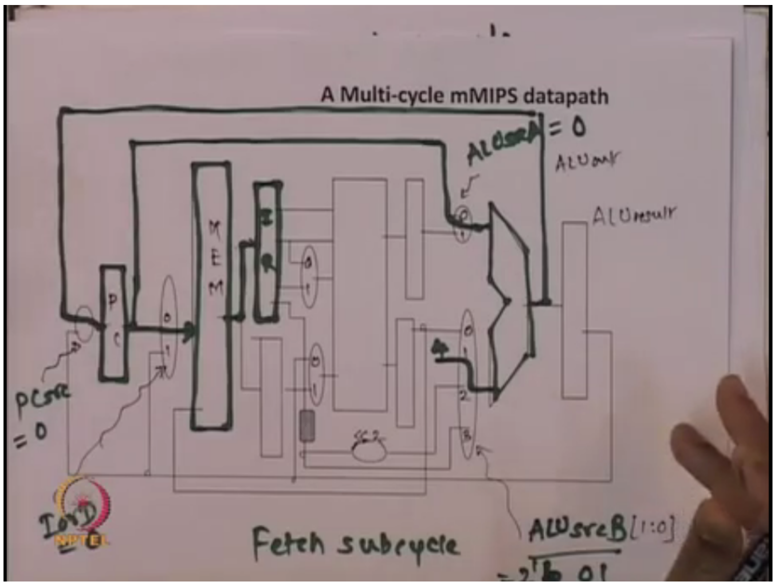
So the next diagram will kind of make that clearer, but with bit more clutter like here same. PC name you recognize all of that, instruction register, register file RF, A, B, ALU, ALU result. Do not worry about the names so much. They are like you will get used to that okay. So now you see that there is a four-way multiplexer over here as I told you about that one of the inputs to the multiplexer is directly coming from the B register, other input is coming feeding constant four.

Other two inputs are coming out of as you see that IR instruction register and these are the 16-bit immediate fields okay. Will tell you the details about it with some bit of operation on that. So there are two possibilities here. In one go, we will not be able to it will not be easy to describe why this four, but as you see like different instruction subcycles, different states, together like you know we super impose them then we get this picture, but individually in pieces will have much cleaner view of the different parts of this datapath okay.

So we will work the such pictures next yeah so in my notation this oval, this oval, this oval, this narrow oval and so on are multiplexers okay. Again I have not drawn the directions, but it is fairly clear from the context of the figure what the inputs are and what the outputs are. I am also not showing the control select signals on these multiplexes as well as other control signals, but now the need of the control signals will become clear.

Which control signals are to be generated and asserted, deasserted in which clock cycle that will become clear okay.

(Refer Slide Time: 30:55)



So Let us take this again just an outline here this okay this is some small insignificant thing relatively insignificant do not worry about it. This outline we just have it as a background and now we show how the datapath is configured for different subcycles. Let us take the first subcycle that is fetch, which is common for all the instruction cycles, all the instructions. So what happens in fetch subcycle.

So this is for fetch subcycle what happens, which registers are involved, which signals have some meaning in this fetch subcycle. Is the ALU involved during fetch subcycle, obviously program counter is involved right, obviously memory is involved because that is where the instruction is stored and instruction is to be brought out from there.

So really like you know program counter will be used and the program counter values will be fed as address input to the memory, memory will be involved in the fetch subcycle because memory contains both instruction as well as data. We want an appropriate instruction at location pointed to by this. Then data output of this memory arise over here and this IR instruction register is enabled to load into itself or register into itself that contains of the memory that are coming out over here and that could be the instruction.

So that in the next state, the instruction is stably available here and that can be decoded, analyzed for future rest of the subcycles for that instruction cycle okay. What else is supposed to be happening in this fetch subcycle. There is an instruction fetch, but there is a little bit more than that. In this subcycle or clock cycle itself will be using the ALU to generate PC+4.

Then is a tentative all typical next titer is to be loaded into program counter and for that will make use of the only ALU available, will configure it to do a simple addition, addition of what, addition of the program counter value, which we sent and with the help of this multiplexer will route it to this first port of this and what is to be added to PC4 because the next instruction 32-bit instruction will be at 4 bytes away from the current program current instruction address that will be this four getting routed over here.

And the output of this ALU in the same clock cycle, we are not taking the registered value of it, is to be arranged to be brought here. So there is a role of this multiplexer, there is a role of this multiplexer, there is a role of this multiplexer involved in this fetch subcycle. So they have to be given appropriate select inputs to this multiplexers, multiplexers have to be appropriately defined.

So that appropriate routing of data occurs, do you get the point. This ALU PC+4 all this is required because in the fetch subcycle, we are also updating program counter with the next typical address of the next instruction PC+4 okay. Of course, we know if it is a jump instruction or a branch equal to or that kind of instruction, then there will be possibility of this value being over written with appropriate value, but that is at a later time. In the fetch subcycle, this is what exactly happens okay.

Is it clear? Okay just for to disambiguate between this two will call this ALU result and this ALU out or you might use a fix like ALU result register and ALU result all unregistered or whatever like you know. So now we will just identify some of this multiplexers and give them some names. So that we can refer to them bit more conveniently. So there is this multiplexer, which is about selecting the okay.

This multiplexer will be for the purpose of selecting the source to program counter PCsrc that is the name I give it. So then this multiplexer I had call it or many text books will also call it I or Data. So that is how we will refer to it means select signal will be called IallD instruction all data okay because the job of this multiplexer is to send either the address of the instruction or the address of the data that we will see later, address of the data and when will it arrive over this line okay.

So these names will be used as the names of the select signals on the multiplexer. They will be one bit select signal, there will be a one bit select signal here, and our another convention that I am adopting is that all this multiplexers the upper one is 0 and port number 0, port number 1, here it is more interesting port number 0, 1, 2, 3. So this multiplexer being four input multiplexer would require 2-bit select signal.

This is the 2 input multiplexer the two ports are 0 and 1. So avoiding the clutter by like not putting this like you know names and whatever symbols inside, but we are having some uniform convention of like you know reading out what these ports are like you know upper one is 0, later ones are 1, 2, 3 whatever okay.

So some more multiplexers, this multiplexer is a big multiplexer, I call it ALUsrcB, I regard this as the B port of the ALU and what is the source to that B port of ALU is determined by the 4 input multiplexer whose select signal is going to be a 2-bit select signal one down to 0, in a Verilog notation 1:0, in VHDL 1 down to 0 or whatever okay.

So I am not showing that those select signals, their implicit and they will be referred to by the names that I am choosing over here. Similarly, this I will call ALUsrcA okay like it is to be read out as the multiplexer select signal, which will decide what is the source to the A port of this ALU whether it is something coming from PC or it is something coming from the A register here. Do not confuse this A with A, but yes there is a correlation okay.

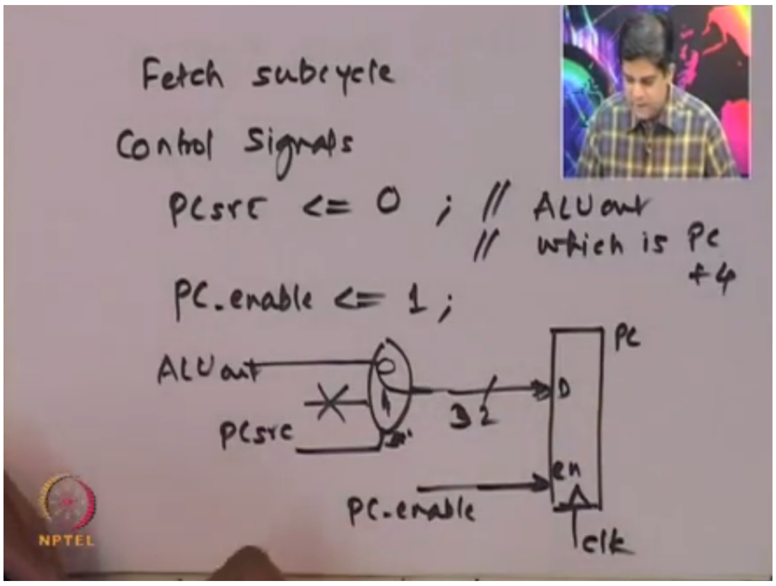
And then this two will come to them a bit later yeah when we like talk about them. There is something more that you see over here. This is the left shift by two bits, which will require because when doing the address arithmetic the offset is given as a word offset and that offset has to be like you know sign extended by this block and then left shifted by two bits. So that it becomes a byte offset okay.

And 32-bit byte offset, 16-bit contents of IR immediate field are to be sign extended and then left shifted by 2. When it is to be interpreted as the byte offset in case of some address calculation yeah. So will come to that later. So please make a note of these names that we have chosen for the multiplexers and we just finished showing the fetch subcycle, the portion the datapath that is active what is happening.

So what we will need to notice is that in this PCsrc will have value 0 because it should allow the upper input to go to PC. IorD should have value again 0 because that is the upper one that is going through this multiplexer at its output okay. What about this ALUsrcB? You see that constant 4, which is on the second port that is port number 1 or input number 1 of this multiplexer that should go through to the output and hence to the B port of ALU.

So this should be 01 or 01 binary rather in Verilog notation two tick b 01 okay that in decimal it is just 1 alright and what about this, this value of the ALU source A should be 0 again because 0th inputs which is coming from program counter is being routed over here. So that is what we mean by the details of the control signals inside the fetch state F state of the controller FSM.

(Refer Slide Time: 42:16)

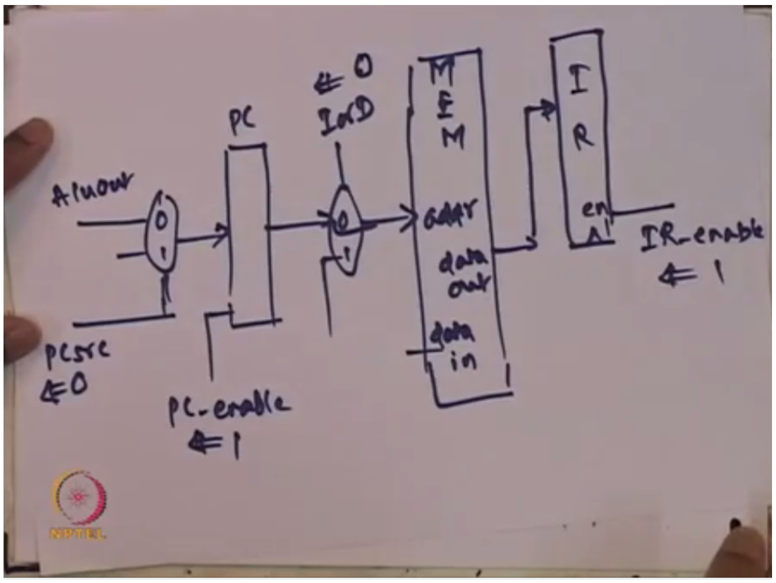


So inside fetch subcycle we will just document or will make a note of one was PCsrc. PCsrc would be said to the values 0, comment is that like ALU out which is PC+4 is to be brought at input of program counter, to be loaded into it and we use the enable signal called PC enable program counter enable we assert it okay. So this is the program counter slightly expanded.

This is the clock and there is a enable input, this is clock, there is enable input and this enable input is going to decide whether the 32-bit data that is coming at the D input, these are D flipflops typically or this is coming out of that multiplexer, which is controlled by PCsrc and this is a signal called PC enable. So this is also coming out of the controller FSM. This says that we should let this ALU out on which PC+4 is computed on with that value is stabilized that should be allowed to go through.

And this one you should not be okay. This is the select input okay this is the enable input to the register program counter for letting this value to be loaded into this or not. So this is going to be loaded okay, but that is not alright. In fetch subcycle, some more control signals are also playing a role. What are those? This is related to update of program counter, which is a kind of auxiliary task of fetch, important but auxiliary.

(Refer Slide Time: 44:57)

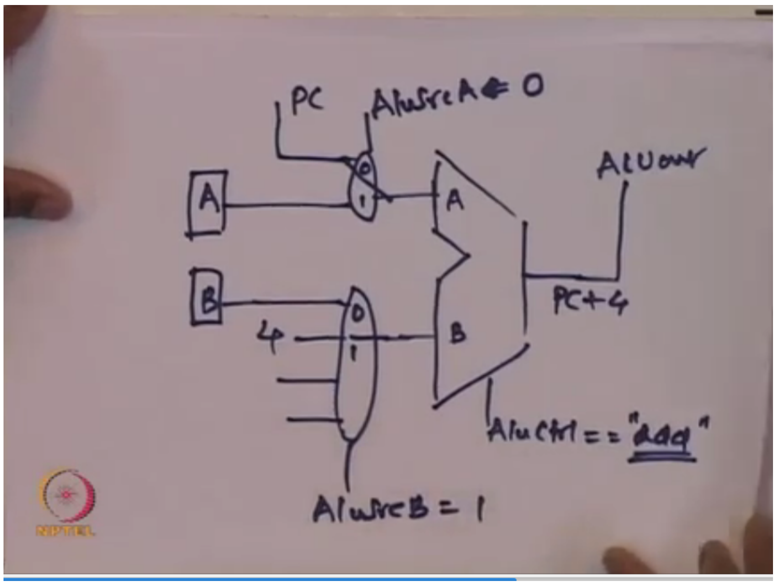


The main task of fetch is to get the instruction from the memory. For that there is a memory (()) (44:52). This we have seen, but now there is a role of this multiplexer, which we call IorD so job in this cycle is to route the content of PC to go as a address of MEM block. This is the address input and this is the data input that we do not need to bother about. We are not writing in the memory.

On the other hand we are reading data out from the memory and that would be there are two places where this thing information can go either IR or there is an MDR, but we will enable this register using as control signal caller IR enable okay. This is IR so we will assert this to 1 so that the contents of memory are to be loaded latched into or registered into IR. We will set this PCsrc to 0 so that ALU out which contains PC+4 is to be loaded into this PC, which is controlled by setting this to 1.

To allow this to go through as address this should be set to 0, this should be enable what else ALU that also remains, exactly what is happening at the ALU, how the ALU's environment is to be controlled, dated the part of the datapath.

(Refer Slide Time: 47:09)



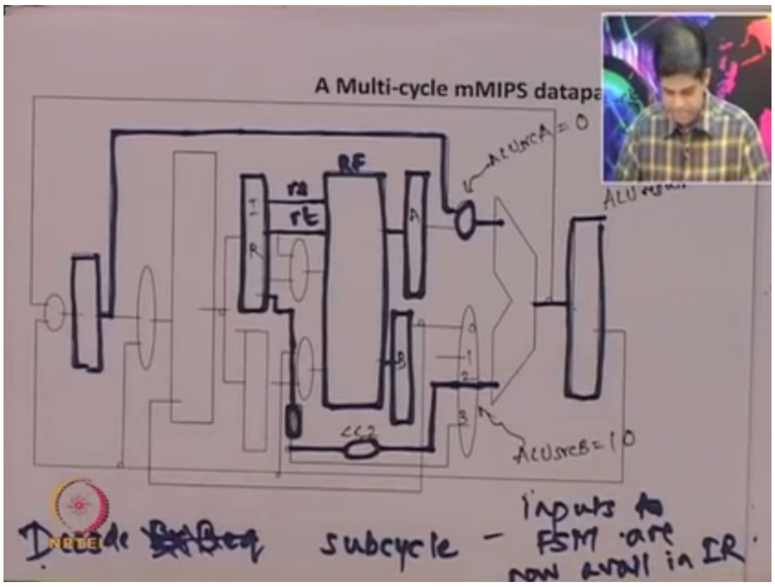
So recall at ALU, again we are still talking about fetch subcycle what is happening at ALU? This is the A port of ALU, can be called port 1 port 2, but let us call it A port B port. There is a multiplexer here, there are two possibilities here, where is this coming from, this is coming from PC remember that and this is coming from four different sources. For example, this was coming from A register, this was coming from B register, this is from some other places not important right now.

But we have to let this pass, we have to let this pass, so for that we have to set it to 0 and this ALUsrcB we have to set it to 01 that is 1 okay. We already remarked on that so that this PC and this 4 go to ALU. We also have to configure the ALU control to say add, add function okay. That is auxiliary detail that you can work out and so on. Because of which will have PC+4 available at the ALU out.

And this ALU out goes, is arranged to be routed to program counter and so that it gets loaded. So this is what is happening in different parts of the datapath inside fetch subcycle okay. It is not complicated, but one has to get a clear picture of what is happening in different parts and should not miss out on something. If you miss out on any one of them, then the controller is not going to control or edit datapath correctly, but careful analysis is quite easy and like HDL description allows us to kind of simulate and test things and debug.

So this is not black magic or black art of doing things okay. So this completes our discussion of fetch subcycle. Similarly let us do couple of more cases.

(Refer Slide Time: 49:44)


A Multi-cycle mMIPS datapa
RF
rs
rt
A
B
I
R
ALUsrcA = 0
ALUsrcB = 10
ALU result
<<2
Decode ~~Beq~~ subcycle - inputs to FSM are now avail in IR
NPTEL

After the study of fetch subcycle and what happens on the datapath during that we will study naturally the next one that is decode subcycle. There are 10 more states to be considered, but we will do a few of the three or four of this and leave the rest to you okay. So again this is the outline, which will start filling up, there is one small thing that is missing here. This is that left shift by 2 yeah I think hopefully now it will bring a picture.

So in the decode subcycle what is happening? The instruction is being decoded that is the major thing. So IR is going to be playing a role. This is instruction register right and also the register operands are being grade out. So this is the register file so that is also involved in this okay. So the decode stage the appropriate registers of this register file are going to be read out and they contains of those registers are going to be stored in A and B.

This is A and this is B okay and how is H, which register chosen to output 2A that will be based on 5-bit signals coming from here, which are the RS field. Will give the details later on, select same as the single cycle datapath details. The register for B is chosen by this rt field okay. So this influence which register is loaded into A, which register contents are loaded into B.

So this is the register read out part of the decode subcycle. What else must be happening looks like on the face of it since we already have now IR like in the previous clock cycle instruction register has been updated at the end of the previous clock cycle. Now we have the whole instruction over here, most of the instructions of core and relevant control information is sent to the controller from here okay.

And these are the inputs of FSM. Note now non trigger input to FSM, FSM are now available in IR okay. Then clock is always in the input to the FSM because like you know that is when the transitions are triggered, but the non-trivial inputs are now available in the instruction register and that would be used in this and future state subcycles.

There is something more that we do in decode subcycle. For example, if it were a branch instruction or jump instruction we have the opportunity, will see that the ALU is not being used like you know arithmetic operations, which are part of these add instruction or the ALUs also not free because load or stored instruction would be using the ALU in the Ex state that is the next clock cycle.

So right now ALU is free so it is a good opportunity to make use of ALU for doing something, which can be done right now. That is calculated tentative branched target address. The branch target address is addition of the contents of PC. The branch target address can be calculated now. Please note that because PC has been updated to PC+4, this is a peculiar thing about MIPS.

The target address will be original program counter plus 4 plus whatever is available specified in the immediate field of the instruction register, sign extended, shifted left by two bits, so that it looks like a word, it reflects the byte address of that instruction word and then this. So we have to configure this multiplexer to let this go through and configure this multiplexer as in the fetch cycle to let this go through.

So this ALUsrc will be 0 and this will be 2. ALUsrcA will be 0 and ALUsrcB will be 10 okay and then this do we feed it back immediately to this, no we do not want to, now program counter because we are in the decode cycle, program counter contains the current instructions address that was earlier there plus 4.

Remember that in fetch cycle we updated this, but now we have calculated at the output of ALU that tentative branch target address, but do we bring it back here, no because we are not going to take a decision on updating the program counter right now in this decode because too early, if it were a jump instruction because it is unconditional we could update this right now, but now we are talking about we have left out from consideration the jump instruction.

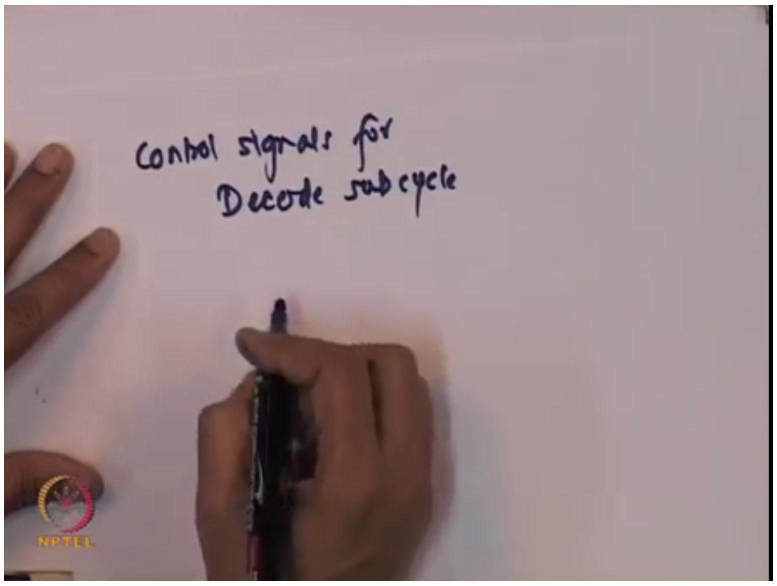
We assume that jump we are not supporting in our current exercise. So the only instruction which would require this branch target address to be loaded in PC is those conditional branch like instructions, but whether to load the PC with branch target address would be decided only in the Ex state when the operands are compared whether they are found to be equal to or not whether something is found to be negative depending on the condition of the operands or arithmetic on the operands.

So since we want to differ the decision of like whether the branch target address is to be loaded here to the next clock cycle what we will do is that the information about the target address that we have calculated will put it in the register. This is the ALU result register, this one will load okay, so it is enable signal to this will be asserted so that the branch target address is going to be stored here.

That is part of the decode subcycle okay. ALU is used, it is not just reading out the registers, pair of operands from the registers, it is not just sending the instruction bits to their controller, but calculating tentative branch target address and archaizing it, storing it in the ALU result register, so that it can be used possibly in the next clock cycle. Otherwise, we will just forget about it, but this is the safe place to keep.

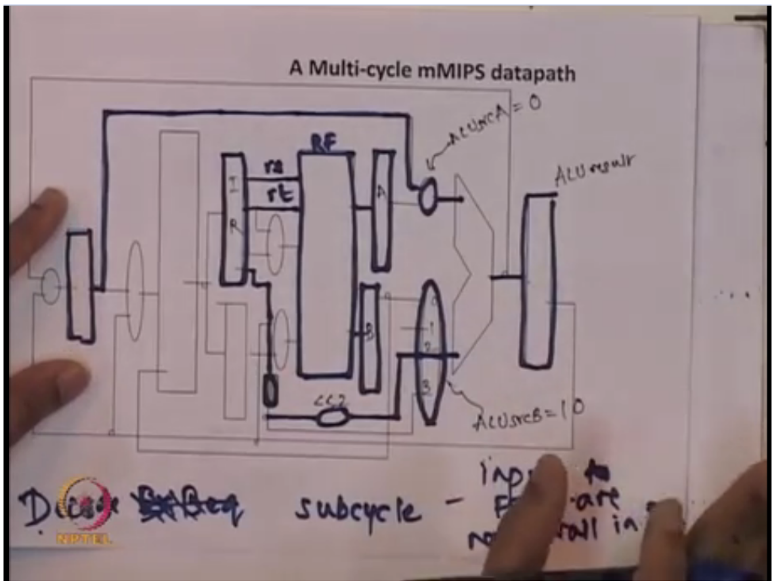
It cannot not be used immediately like in the case of PC+4 was used in the fetch subcycle immediately. In the same clock cycle, the program counter was unable to be updated now there is no such need. If jump were to be handled jump instruction then we would have seen that kind of thing, but now we do not need any enabling of PC and so on and that is why this. Yeah so then we can similarly like you know list out which multiplexers are involved, which other control signals are set to what values.

(Refer Slide Time: 58:33)



So control signals for decode subcycle okay.

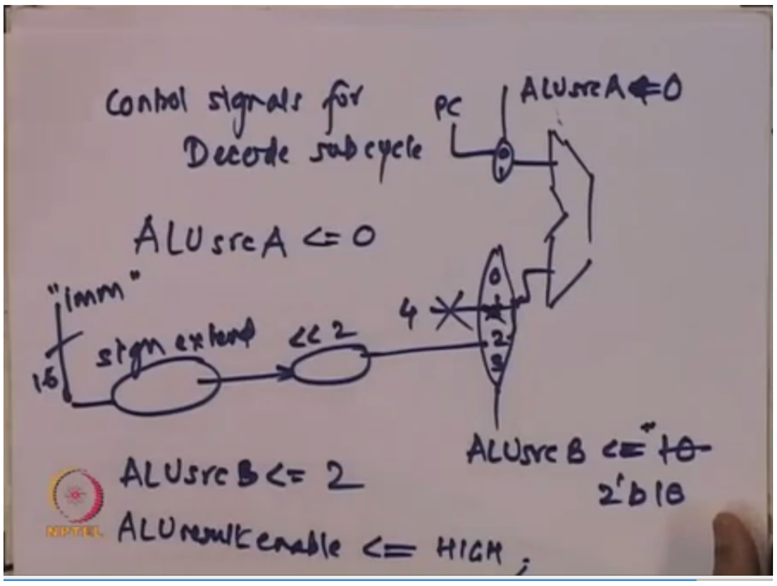
(Refer Slide Time: 58:49)



So as we see here this multiplexer had to yeah there involved, this multiplexers have not involved, this multiplexer is to decide which value gets written into the register that is not needed right now. This like enabling of this register is involved. These two registers they are perpetually enabled, they are enable all the time because you know we will not really bother about what is inside them only when we like you know we need them it will be made sure that we have some like you know correct value there.

Otherwise what is going into is we will not be necessarily worried. So we always let this registers be all the time enabled to be loaded, can verify that that is safe and fine so we do not need additional like you know control logic, we just hardwire this enable signal to this A and B registers to be 1, but here this is we have to be more selective.

(Refer Slide Time: 59:58)

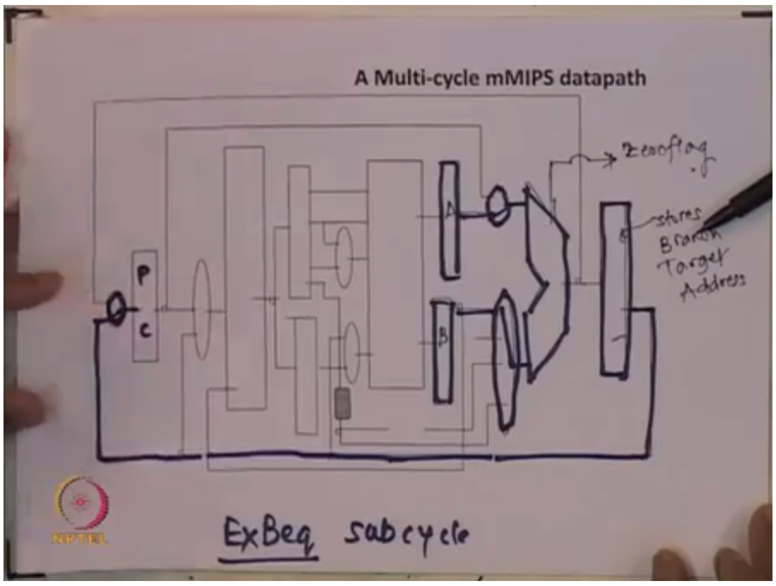


So Let us make a list of things control signals to ALUsrcA that is ALUsrcA 01, we have to let PC which is not PC+4 that go through here and this other big one we have to let the constant 4 go here. So this should be set to 0, this should be set to oh sorry not this right I am sorry that was in the fetch subcycle, it will be this which is basically 16-bit immediate field of instruction is sign extended, make it a 32-bit number is left shifted by 2 that is multiplied by 4, that is fed to the second input of this multiplexer.

And so that is 2 tick b 10 that is equivalent to the second. So this is being connected okay. So ALU source is 0, ALUsrcB is 2 okay and ALU result that registers enable signal, the flipflops enable signal is to be asserted high or 1 or whatever it, anything else I think that is more or less yeah. Other things take the default value that is deasserted or whatever like you know yeah. Only when we need to start writing the Verilog code or HDL, VHDL code, then we will have to be like you know we have to be exhaustive about all these values and so on, but this is more about getting an idea how do we go about it.

I think the picture should be fairly clear. Next, we will Let us take a look at the execute subcycle of the beq instruction because we know something interesting will happen here this is at the end of the instruction cycle for beq branch if equal to instruction so should be interesting okay.

(Refer Slide Time: 01:02:54)



So let us now again take one outline of the datapath and see what is going to happen in. Execute subcycle of Beq okay. ALU is involved right, what will be happening in the branch if equal to the A and B, which now hold the values of appropriate register selected by the instruction so A and B are involved. This values contains of A and B are compared are for equality for that we have to arrange this multiplexers to route this to the ALU okay.

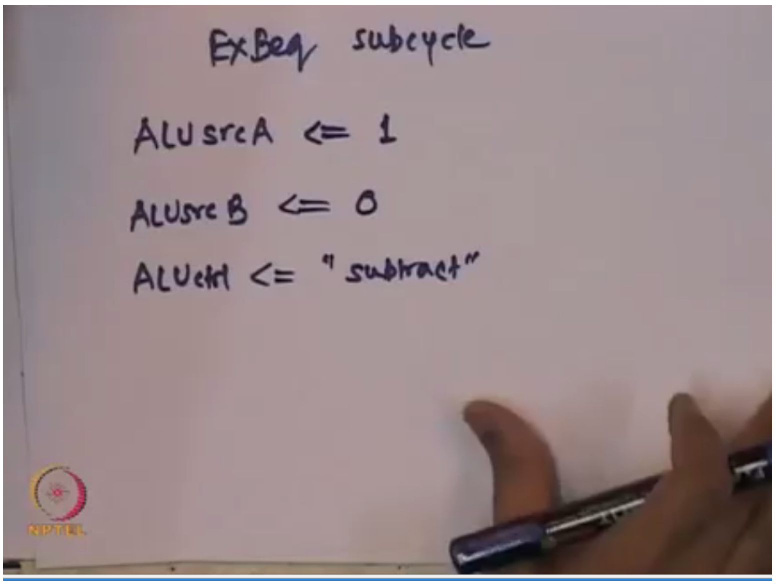
And ALU will may be do subtraction or check for equality by some comparator and put a status flag says zero flag. Put it out zero flag okay. It is an output of the ALU, a status output of ALU. This will go to the controller or some other control circuitry. So that is what will be happening in this okay. More interestingly like you know we also realize that we need the branch target address that was computed during the decode subcycle, which was the previous clock cycle, where is that available that is available inside the ALU result we just remarked on that right.

That we are using ALU in the decode clock cycle for something, which could be possibly used right now and that is the case that we are at. This is ALU result register so what about this? This could possibly be conditionally be what that should get loaded into program counter PC okay, but it is not necessary right. It would depend on whether this zero flag says yes or not okay.

So depending on this zero flag the enable input of this program counter is going to be asserted or not okay, but the point is that the multiplexers have to be configured to allow this comparison of A and B the two operands okay and ALU result, which stores what branch target address, stores branch target address, that branch target address is we have to be ready to load it into the PC in case the conditions have been found to be correct or whatever like you know. So this router this multiplexer also has to be configured appropriately.

So this multiplexer, this multiplexer and this multiplexers are going to and note that ALU out is not of any interest okay. This is the status flag of ALU is of interest, this is of interest so this is how the portion of the datapath, which is relevant to this subcycle. So Let us start like you know exercising our memory like what those names, which multiplexers and what are their names.

(Refer Slide Time: 01:06:39)



ExBeq subcycle

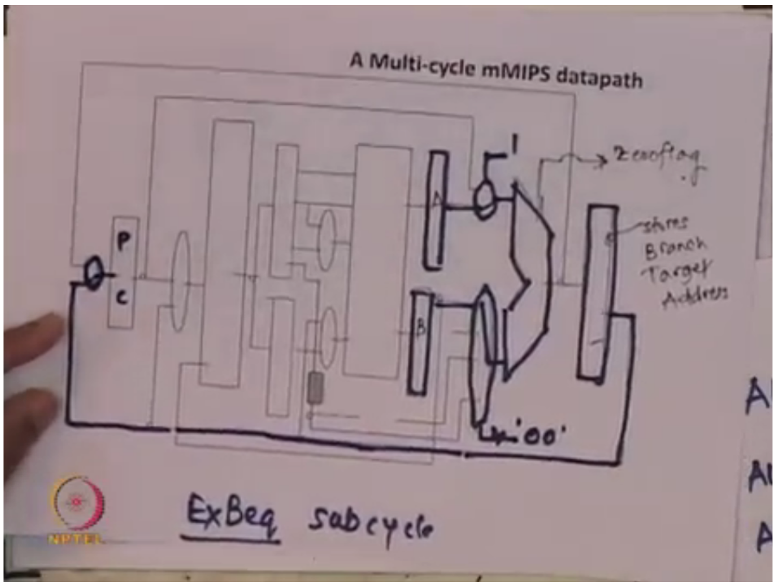
ALUsrcA <= 1

ALUsrcB <= 0

ALUctrl <= "subtract"

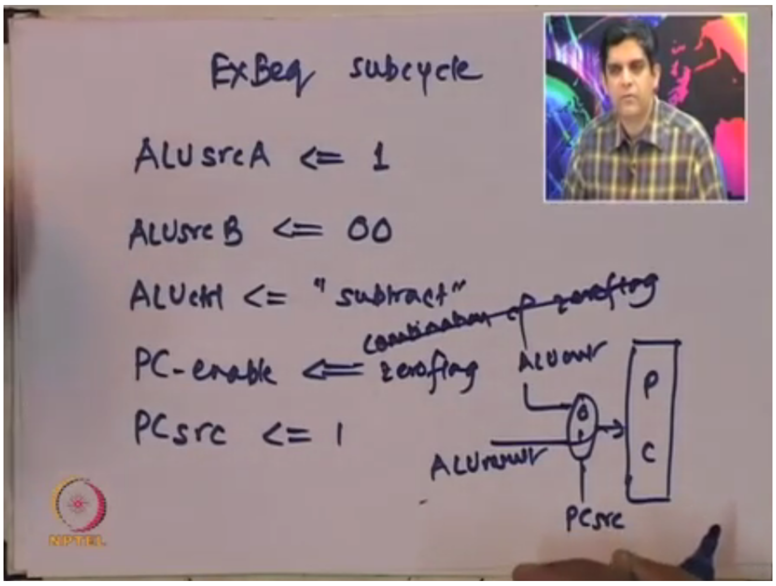
Multiplexer ALUsrcA that is involved, ALUsrcB that is involved. We have ALU controller, ALU control should say compare or subtract right, subtract the two operands and if the subtraction gives zero then the zero flag will be asserted and that is what we are using. ALU source A says that A should be allowed to go through that is the port number 1 of that multiplexer. Here it is a port number 0 of that multiplexer.

(Refer Slide Time: 01:07:23)



We recall note this diagram, so here I will be putting 0 because this is 0 and here in the select signal I will be putting 1. Think of these are select signals, is basically two bit 00 okay, two bits. So this selects this particular input port to go through that is the contents of B. So I will write here ALU source B as 00 and this as 1.

(Refer Slide Time: 01:08:00)



ExBeq subcycle

ALUsrcA <= 1

ALUsrcB <= 00

ALUctrl <= "subtract"

PC-enable <= ~~combination of zeroflag~~ zeroflag

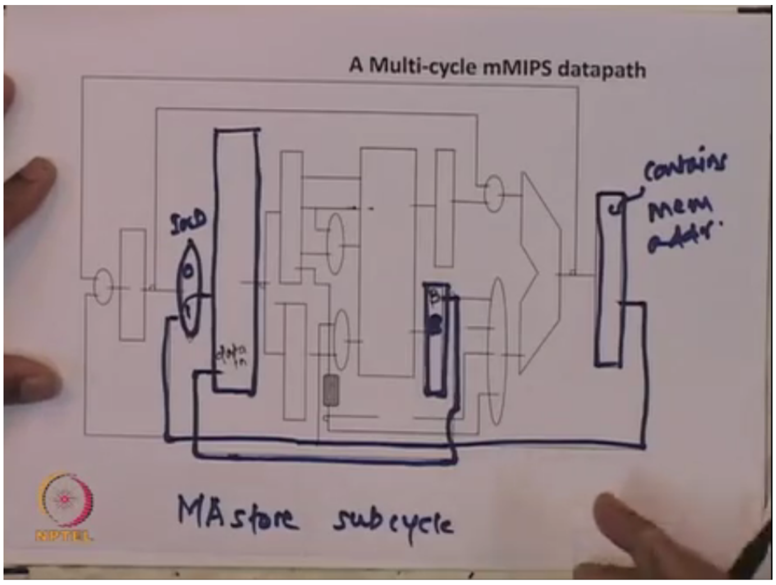
PCsrc <= 1

ALUout
ALUresult
0
1
P
C
PCsrc

What else PC-enable and also that multiplexer before PCsrc, that should say 1 because this is from ALU out and this is from ALU result. So branch target address is in the ALU result, so that is why we should select this to go through okay, but what about PC-enable that would depend on appropriate combination of zero flag okay. Rather it will depend on zero flag, could as well say zero flag, if zero flag is asserted that means we have to kind of load this branch target address, which is available in the ALU result into the PC.

So the logic driving PC-enable will be just whether the ALU status has zero flag is 2 or not okay. So this will mean some combination logic here okay, are we missing anything in the execute Beq subcycle okay more or less if something is missing think of it as an exercise for you to get more practice with this yeah. I think you have got most of the idea.

(Refer Slide Time: 01:10:00)



Let just do a memory access, memory access of store okay. Let just draw the datapath portion and leave the control signal exact definitions as an exercise for you okay. It is in fact the terminal state, last state, it is at the end of the instruction cycle of store instruction. So what must be happening? Of course memory is involved. So where does the address of the memory should come from, where should it come from, it should come from what has happened before memory access of store, it was a execute stage of memory access.

So what happened before this instruction subcycle in store instructions cycle. It would have been the execute state subcycle of store instruction in which the effective address of the memory location would have been computed by the ALU and stored into the ALU result for future use right. So ALU result would contains the memory address okay, which was computed in the Ex store subcycle, execute store subcycle.

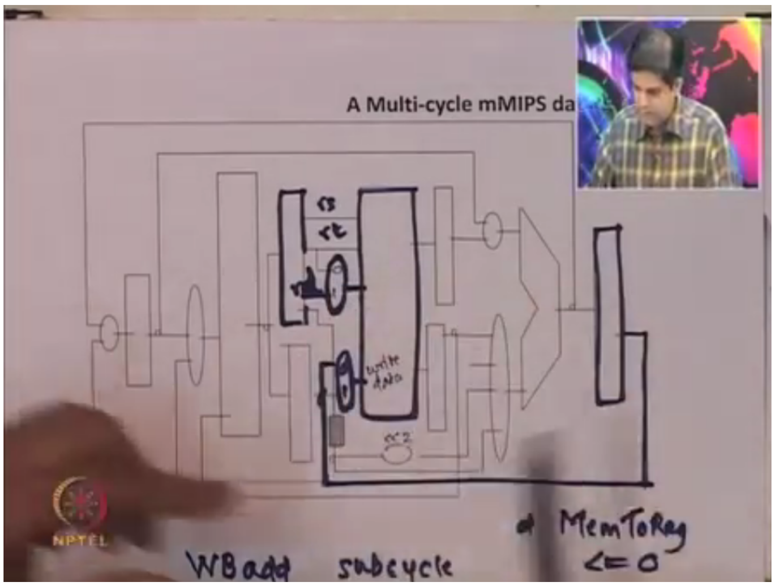
So this is now to be used over here okay. So this multiplexer called IorD is going to play a role and is going to use the select signal value as 1 so that this is allowed to go through at the address input of memory okay. This is the store instruction, so what should have happened in this commitment phase stage of store instruction cycle is, at the data input port of memory, this is the data input port of memory.

At this data input port of memory, where should the information come from, information should be stored into the memory would be available in the B register. This is the B register, so this is playing a role okay. Verify that from the semantics of the store instructions. RT field of IR is going to indicate the register from which the data is to be taken and stored into memory location whose address is given by RS index register and immediate field.

So that address has been computed previously, B register has been loaded previously and that has to be used as data input over here. So this is the B, if it is not clear it is a B thing okay. Remember that in the previous clock cycle, this RT specify field of IR like you know indicated contents of which register have to be loaded into B. So that has already been done, now they are to be put into the appropriate location in the data portion of the memory okay.

Note that there is no role of multiplexer, there is no role of ALU here, there is no role of register file and program counter and so on right. So this is the terminal state or subcycle of store instruction. Just to end things I mean there are still more, but to get an idea of like what happens with register file if some interesting cases remain which we will take them as exercises later on.

(Refer Slide Time: 01:14:34)

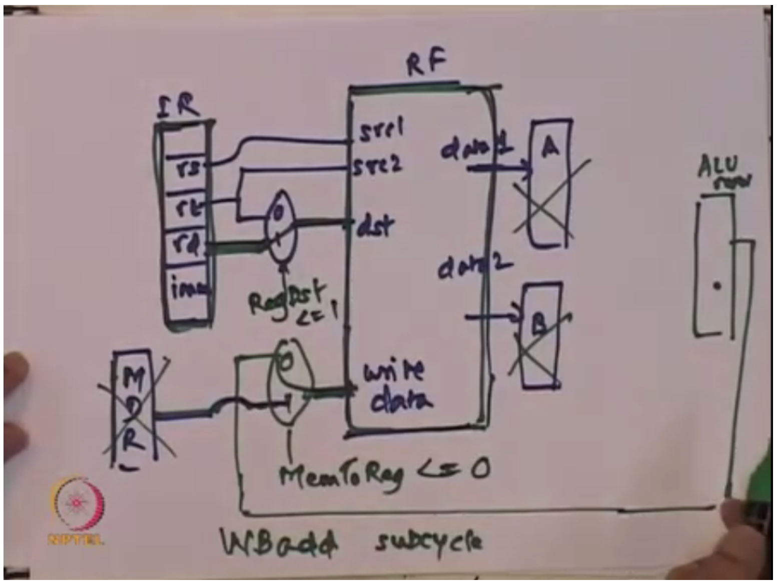


So let us consider write back stage for say add instruction. So here assume that in the ALU result previously at the end of the previous clock cycle the result of the addition has been stored alright. So now ALU is not involved, now it is just the matter of putting this in appropriate place. Where should this go to, this should arrive finally at the write data port of the register file so register file is going to be involved okay. This multiplexer is going to be involved because that decides what should arrive here and what it should allow is this.

ALU result should be allowed to go through the zero port of this multiplexer. So this multiplexer will give it a name, so we can call it MemToReg, this is MemToReg multiplexer is selected with 0 input and so that this ALU result is archived into the register file, which register is it archived into that would be decided by this input of the register file and that is why this MUX will play a role and it is the rd field of this instruction register.

So instruction register is also going to be involved in this picture okay so note that this was rs, which is not playing a role here, rt is also not playing a role, but rd got fluttered here so again we call it 01, so maybe we can do more justice to it by drawing some the part of this separately.

(Refer Slide Time: 01:17:15)



This is IR, R has some portions rs, rt and rd immediate and something else. This is not really is proper to the scale and whatever. So rd is input to this multiplexers, rt is also input to this multiplexer, rt and rs go separately to register file. This is source 1, this is the index of source 2, this is the index of destination okay. Data 1 that goes to A, data read out from the register specified here that goes to B, that we are not concerned about in this case, but just having a complete picture of this.

This is the data input to the register and which register that is decided by this, but in this current instruction it is going to be okay and this normally it would have been from the MDR sorry this is not to be worried about. The green lines are indicating our datapath now. Where is this coming from? This is coming from the ALU result okay. ALU result yeah so I should technically draw this in green, so indicating that register file is in picture in this subcycle, it is getting updated because of this information.

A and B we do not care, ALU also we do not care, but we care about the result of ALU that we obtain in the previous clock cycle, which is stored over here. Instruction IR is involved, MDR is not involved, A and B are not involved in this so called what phase is this, this is the write back of add okay. Before that the Ex of add, execute stage of add instruction had taken place and ALU result has like the result of the addition operation.

The destination field of IR has the information about destination register and this multiplexer allows this to go through, this multiplexer I call it MemToReg, so MemToReg should be set to 0 indicating that it is a register that should go not the MDR. In some other case, in fact it will be in the write back stage of load instruction cycle, the data flow will be like this. While doing that exercise you will be able to use this.

Will see the use of MDR and the other value for MemToReg okay. So this should become 0. This I call it RegDst, give it arbitrary name that should be 1, this control signal okay through that this goes to. So this together gives some picture of good enough picture of I guess the different subcycles and how control signals or output by the finite state machine in those individual clock cycles okay.