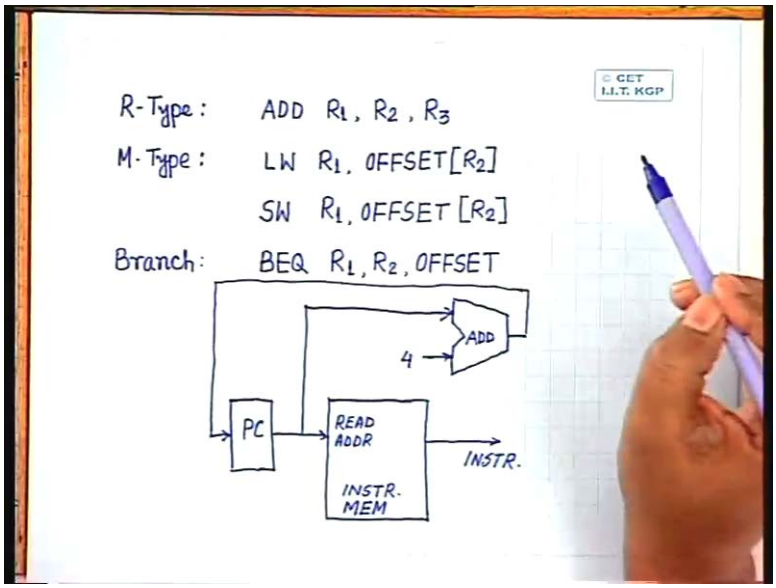
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering,
Indian Institute of Technology, Kharagpur
Lecture No. # 11
Pipeline CPU – II

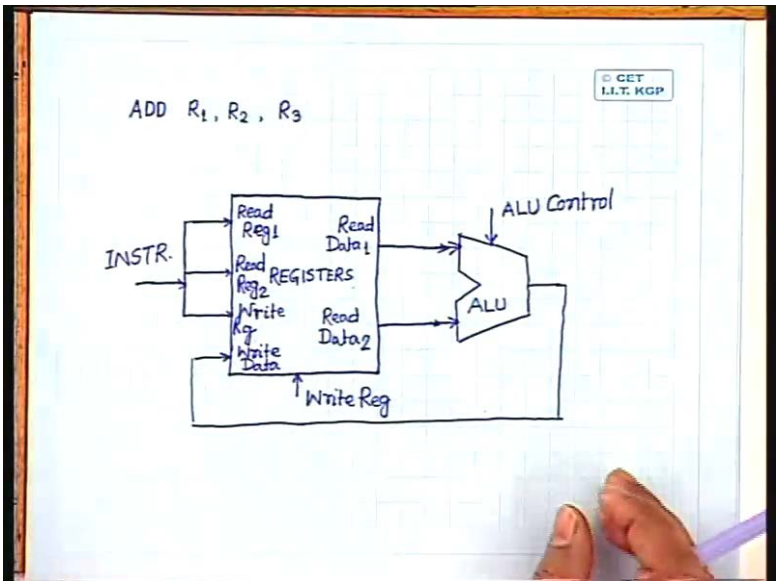
In the last class we have started discussion on MIPS processor architecture and to build up the MIPS processor architecture, we have taken some examples some representative examples from the MIPS instruction set.

(Refer Slide Time: 00:01:15 min)



One of the examples we have taken is add instruction which is a register type or R type instruction. Then two instructions of M type or memory reference instructions we have considered, they are load word and store word and one branch instruction that is branch on equal instruction. We have seen in different steps for execution of these instructions, what are the data elements that will be required and what we will be the data path connecting those data elements? So you have seen that in case of instruction fetch operation which is of course common for any of the instructions, we need program counter and we have said that we have an instruction memory. So program counter output gives address to the instruction memory when from the instruction memory, you get the instruction opcode output. Simultaneously the program counter is incremented by 4 and this incrementation operation is done with the help of a adder and our assumption in this case is all the instructions have the same length that is 4 bytes. This is the data path that will be used for instruction fetch operation.

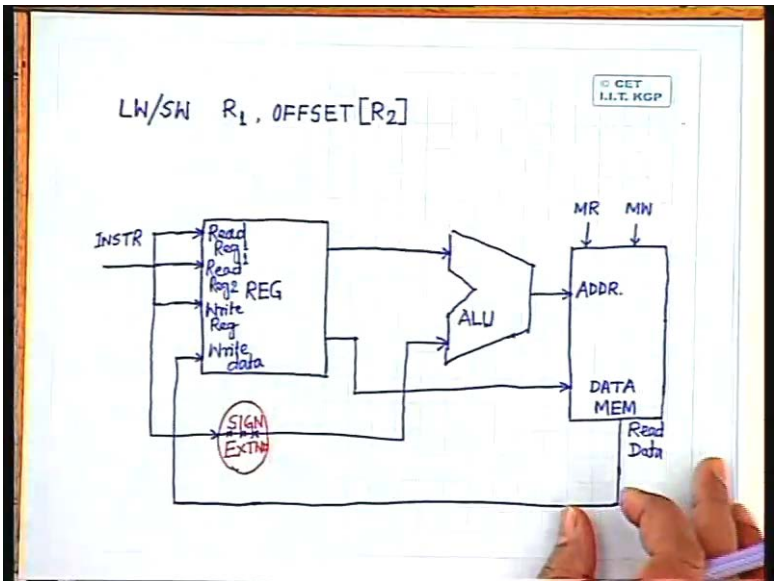
(Refer Slide Time: 00:02:30 min)



The next for an R type operation that is an add instruction, the format is ADD R_1, R_2 and R_3 where the contents of R_1 and R_2 will be added and the result will be stored in R_3 and the data units that are involved for execution of this instruction is set of registers, we put every registers in one block. So this becomes set of registers where the addresses of the registers R_1 R_2 and R_3 will come from the instruction. So we have two registers to be read that is register R_1 and register R_2, so those addresses will be available on these lines and one register to be written that is register R_3, the address will be available on write register address lines. Then output from the register files, the outputs of the registers which are addressed by these two address lines are added through this ALU and the result is written back into the register file through this write data. This data will be written into the register whose address is available on the write register address and this is the data path and the data units which will be needed for execution of this add instruction and this is a typical register type instruction.

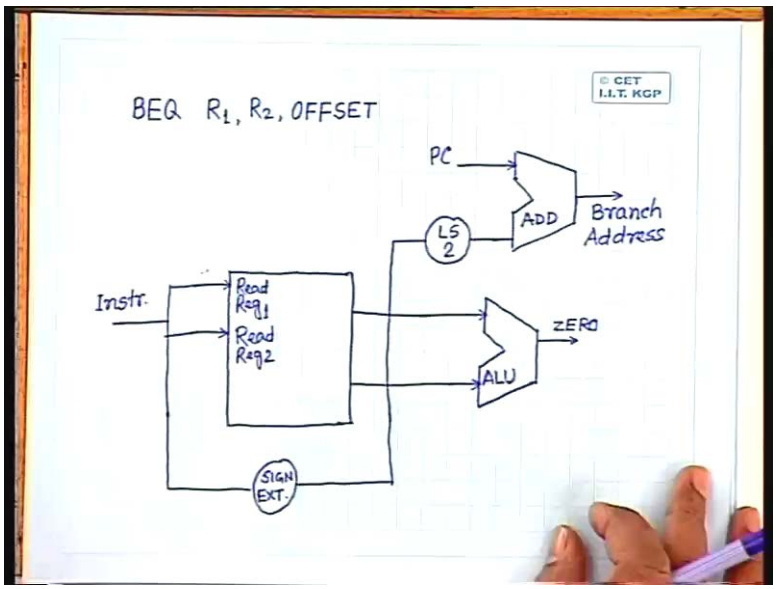
Then for memory type instructions we have considered load word and store word instructions where these instructions have this format load word or store word R_1, offset R_2 where R_2 is the base register. The content of this register R_2 is added with offset, offset after sign extension. To give you the physical address in the memory and for load operation, you have to read that particular memory location and store the data in register R_1. Whereas for store word, the data from register R_1 will be stored into the addressed memory location.

(Refer slide Time: 00:03:56 min)



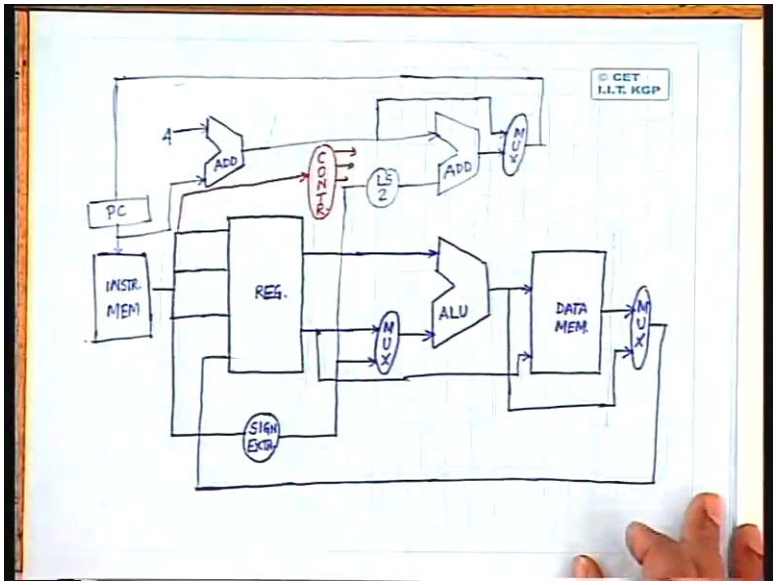
The data path and data units that are involved in this operation are this (Refer Slide Time: 00:04:42). This obviously will have their register files because two registers are involved. Register R_2 is always to be read, R_1 may be read or may be written into depending upon the type of instruction that you execute, whether it is load word or store word. In this case ALU is used for computation of the physical address in the memory, not for performing any add operation like we have seen in case of register type instructions. So this ALU now computes the physical address in the memory location which is to be accessed either for reading purpose or for writing purpose.

(Refer Slide Time: 00:05:22 min)



Then the branch type of instruction that we have seen that is branch on equal R_1 R_2 offset where R_1 and R_2, if these two registers contents are same then the program counter will be loaded with a value offset and this offset is from the current program counter value. So again in this case, this ALU is used for comparing R_1 and R_2 that means ALU has to work in subtract mode. If the ALU output is zero in that case, the effective branch address has to be loaded into the program counter and for that what you have to do is with the current program counter value, you have to add the sign extended offset after giving a left shift by two bits. This left shift of two bits ensures that always the branch address that is generated starts at the starting location of a word of 4 bytes. So that ensures that always this branch address will be the first address of an instruction because every instruction consists of four bytes. These are the data units and the data paths which are involved while execution of BEQ instruction.

(Refer Slide Time: 00:06:40 min)

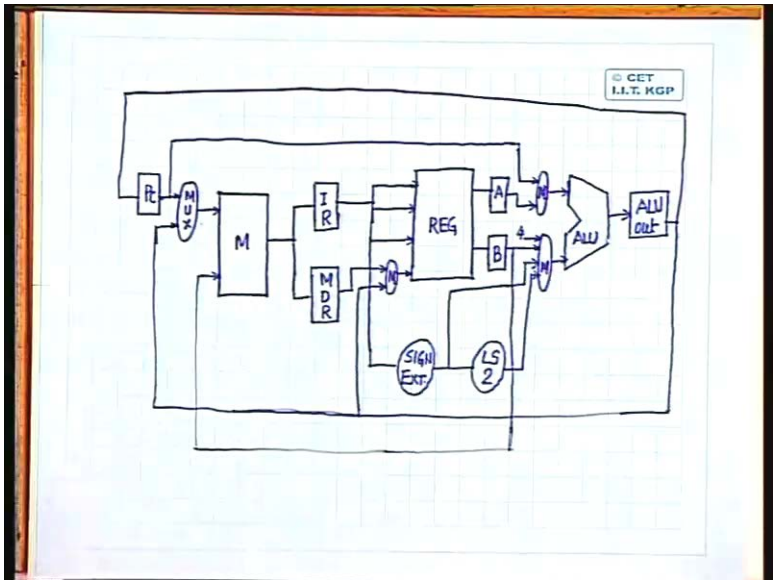


Then you have seen that how to combine all these data paths and the data units by using the multiplexers because in many cases we have that data from various sources has to go to a single destination. So all those source outputs are to be multiplexed and the multiplexer output will go to the destination. Accordingly we have one multiplexer here and another multiplexer here. So this is the basic design of this MIPS CPU.

However you will find that in this particular CPU, we have duplication of data units. For example here we have two adder units and one ALU, though these addition operations can be performed by this same ALU, if we design our data path properly. Not only that, here in we need two different memory units, one is the instruction memory and the other one is data memory. So instruction memory contains only the instructions, the data memory contains the data. So whenever you have to read a data or you have to write a data by using load word or store word instructions, the data will be read from or written into this data memory whereas the instruction will always come from the instruction memory. Now let us see that how we can avoid this duplication because if we can properly design the data path and you can avoid the duplication, a lot of hardware cost can be saved.

So the first thing that you have to do is to avoid duplication is that whether we can combine this data memory and instruction memory. So that we will have a single memory unit which will store both the instruction and data.

(Refer Slide Time: 00:08:28 min)



So for optimization of hardware, the first operation that we will do is we will have a single memory which will store both the data and the instruction. Now we don't have any more separate instruction memory and data memory. We have the memory unit M. Now since we are storing both the instruction and data in the same memory and we have seen before that when an instruction is to be fetched from the memory, the address has to come from the program counter. When a data has to be accessed from the memory, the address has to come from the ALU output because it is the ALU which computes the effective address of the data. There are two sources of addresses from where the address to this memory can be available, one is program counter, other one is ALU.

Naturally we have to have some multiplexer which will multiplex the program counter output and the ALU output, so that the proper address can reach the memory. So we have a multiplexer and the multiplexer output gives the address of the memory. We will have two inputs to this multiplexer, one input will come from the program counter and the other input will come from the ALU output. so we will connect this later. Now output of the memory can either be an instruction or it can also be a data. Now we put it this way that if it is an instruction, we will load the instruction in a register called an instruction register.

In our earlier design, initial design you will find that we did not have anything called an instruction register. Now we will put a register called an instruction register and if the output of the memory is an instruction, the instruction will be stored in the instruction register. Whereas if the output of the memory is a data, we will put that in another register and we will call this as memory data register. So it will be memory data register or MDR.

Now after this stage, as you have seen here previous design that once an instruction is read, you have to access a set of registers. So between this memory and the register, now this memory is combined. It is an instruction memory cum data memory. So between the memory and the register bank or register files, we have put two more registers. one is instruction register and another one is memory data register.

Now the addresses to different registers has to come from the instruction and because we have stored the instruction in instruction register, so all these connections earlier which was coming from the instruction memory directly, now it has to come from the instruction register. So the next component that we will put is the register bank, so a set of registers and all the register addresses to this register will now come from the instruction register. So we will have read register addresses, we will also have the write register address. These are now coming from the instruction register. Now the registers which are read, the outputs will be available on this output. These outputs have to go to the ALU. So we have to put one ALU and before ALU, we will latch this register outputs in two more intermediate registers. Let us call them registers A and B. One is register A and the other register is B and from these registers, the output will go to the ALU. So this is the ALU.

Now our aim is that we were having in our basic design, two adders and one ALU. These two adders were responsible for computing the addresses, instruction addresses and ALU was used for performing the register type instructions, executing the register type instructions and also for calculating the memory data address. Now all these we want to perform with the help of single ALU. So obviously there are a number of sources from which data is to be fed to the ALU. That means at the input of the ALU, we must have some multiplexer to multiplex different data sources. So we will have one multiplexer here, we will have another multiplexer here. Outputs of the multiplexers will give data to the ALU. For this multiplexer one of the input will come from register A and the other input will come from the program counter because the same ALU, we will also use for calculating the instruction address.

Now to this multiplexer, one of the inputs is the output of register B. The other input can come from the instruction because in many cases, the offset address which is specified within the instruction that has to be added with the program counter. So one of the inputs to this multiplexer will come from the instruction register and whatever we get from the instruction register that is from here, the first operation we have said that we will perform on this is sign extension and the second operation that were to be performed on this is left shift by two bits. Now there are operations in which case we need only the sign extension.

There are cases when we need both sign extension and left shift operation. Only sign extension will be needed when we want to compute the address of the data in the memory. We need both of them when this offset will be used for calculation of branch address. That means to this multiplexer, I have to provide this input. One of the multiplexer inputs will be this one that is only sign extension. The other input to the multiplexer can be after the left shift bit operation. Another input to this multiplexer will be a fixed number that is 4 and this fixed number is used whenever after fetching an instruction, you compute the next instruction address. So that sets these multiplexers.

Now ALU output, whatever ALU will compute depending upon what is the control signal that is provided to the ALU and depending upon which of the inputs to this multiplexers are selected. Output of the ALU, we will latch another register which let us call as ALU out. Now as I said that the other input to this multiplexer will come from the program counter. So the second input of the upper multiplexer comes from the program counter. So this completes this part.

The next is register. For registers you find that there is one write input data to the registers and this write data input to the register will be used from the data memory output. When the instruction that is executed is a load word instruction or this can also be from the output of the ALU, when the instruction that is to be executed is an R type instruction. So that says that to this input, again I have to have a multiplexer. The multiplexer output will provide the write data to these set of registers. One of these multiplexer inputs will come from the memory data register and this path will be used when you perform a load word instruction, execute a load word instruction and the other input to this multiplexer will come from the ALU out. This path will be used when the instruction to be executed is an R type instruction like add instruction. So I put this ALU out to the other input of this multiplexer. Not only that this ALU also computes the program counter value.

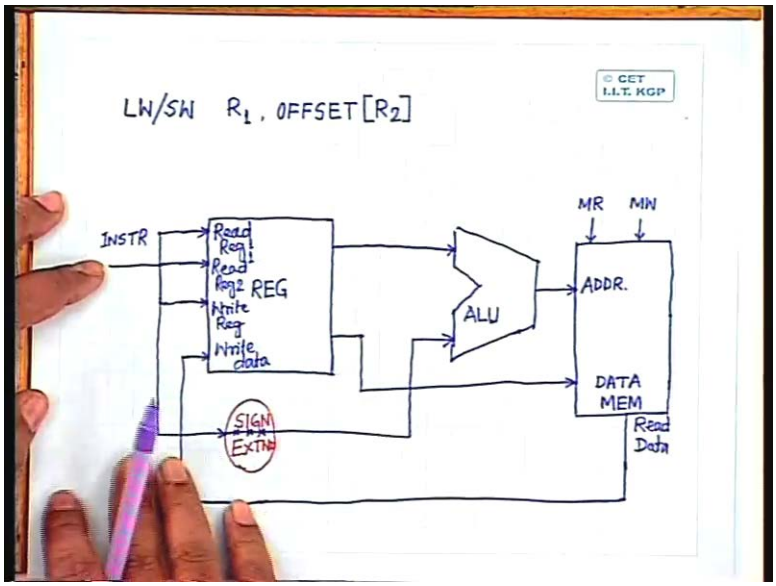
For calculation of the next instruction address, for calculation of the jump address all those things this ALU computes the next program counter value. So the other input of this first multiplexer will also come from the same ALU output. Is it okay? Anything else is left? That is the write data to the memory. So when I want to perform a store word operation, in case of store word the data which is read from the register file that has to come to memory. That means I have to have one memory data input from one of the registers. So that completes the entire data path of this CPU where we have been able to avoid duplication of the data units.

In this case all these are multiplexers, this is a multiplexer, this is also a multiplexer, this is also a multiplexer. Now when we have done this design, you will find that this particular design… One more path that I have missed that is loading the program counter value. The program counter input has to come from the ALU output. So this completes the entire data path design of the CPU. Now this CPU as we said that in our earlier case, the basic design was suitable even for a single clock period execution. Every instruction could be executed in a single clock period in this basic design.

When we convert this design to this design, now we can make use of multiple clock periods to execute an instruction. What is the advantage we gain? That is even the instructions which require shorter delay that can be incorporated. So the CPU will be more efficient. Of course the sequence in which the operations are to be done that has to be decided by the control unit which I have not shown in this diagram. So it is quite obvious that this instruction register output will go to an instruction decoder. The instruction decoder output goes to a timing and control circuit and the timing and control circuit will generate the required control signals to execute any instruction in proper sequence. What are the control signals that will be needed? Load program counter, read memory, write memory, load instruction register, load memory data register, then write register. Then you need control signals for load A or load B, you need the control signals for ALU function. You need the control signals for latching ALU output into this ALU out register. You also need control signals for the select inputs of the multiplexers.

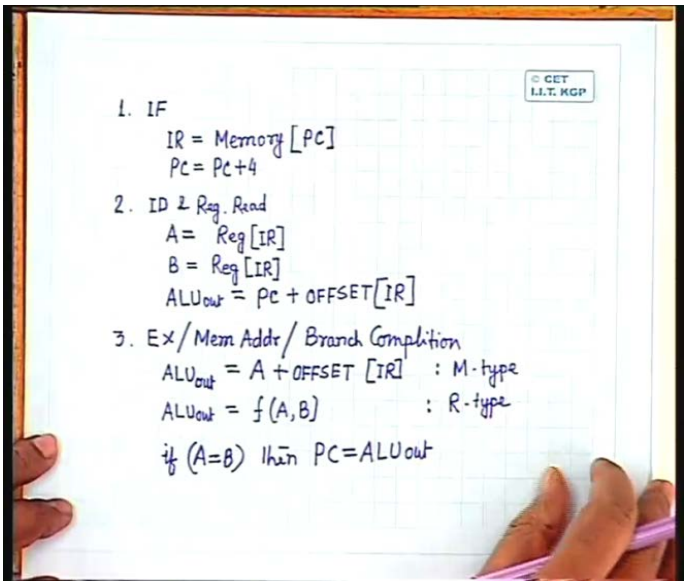
So all these control signals in the required sequence will now be generated by the timing and control unit. The timing and control unit will generate those control signals after it gets the input from an instruction decoder, the instruction decoder will get input from the instruction register. So that part I am not showing in the diagram. Why the output of B going to M? Yeah, this is for store word instruction. What is the operation that we have to perform in case of store word?

(Refer Slide Time: 00:24:06 min)



See in case of store word this was the format, store word R_1 offset R_2 in which case, the address of the memory will be calculated by adding content of R_2 to the offset that is specified in the address. So you will get the physical address in the memory where the data is to be written and which data you have to write? That is the content of register R_1. Here the content of register R_1 will be available in register B, after that the data is read from the register file and that content has to be written back into the memory. Address of the memory will come from this ALU out. Clear? So you can make use of this data unit for multi cycle implementation of different instruction executions. Now once we have this data unit, you will find that the operations that are to be performed in sequence are the first operation is obviously the instruction fetch operation.

(Refer Slide Time: 00:25:10 min)



In instruction fetch what you do? We have said that an instruction is to be loaded into instruction register. So it is instruction register which will get the instruction from memory and the address of the memory will come from the program counter. Simultaneously the program counter also has to be incremented by program counter plus 4, program counter has to be incremented by 4. So you will find that during the first clock period, you are making use of two units in this architecture. One is the program counter output has to be available to the memory address inputs, that means the multiplexer has to be set accordingly so that this input is selected to the output. During the same clock period we also increment the program counter that means we make use of this ALU.

For this ALU, this program counter comes to one input of the multiplexer. So this multiplexer select input has to be set in such a way that this input is passing to the output. For this multiplexer, the select input has to be set in such a way that this fixed input 4 comes to this input of the ALU. ALU has to perform an add operation and this output of the ALU has to come to the program counter. So for that let me put it this way, instead of connecting the program counter input from here let us connect the program counter input from here. So that during the same clock cycle, this ALU output can be loaded into the program counter. Otherwise I need an additional clock cycle because output of the ALU has to be loaded into the ALU out register and then from the ALU out register, it has to go to the program counter. So I connect this program counter input directly from the ALU output. So this is the operation that is to be performed during the first clock period.

During the second clock period. (Conversation between professor and student: Refer Slide Time: 00:27:44) but only that one will be loaded into pc for which the pc write enable is active. During the second clock period, the operation that will be performed is instruction decode. Simultaneously since here you have loaded the instruction into instruction register, so the register addresses are also available.

So what I can do is I can perform instruction decode at the same time register read. These two operations can be performed simultaneously. Now find that because during this period only I am decoding the instructions, so which instruction is going to be executed that is not known. May be the registers that we are reading, they are not needed for execution of that instruction. So anything that we perform before the instruction decoding is complete. We have to ensure that, that operation is not harmful or the operations which are common to every instruction that can be performed. The operation which is common to every instruction is an instruction fetch but that has already been performed. So during this step when we are decoding the instruction, I can only perform that operation which is not harmful.

Even if the result is wrong that can be overwritten by the correct result during the next clock cycles. So register read is such an operation, if I read the register and put the output of the register in say intermediate registers A and B, that is not harmful. May be the data that is read will not be used that will be overwritten by a fresh data during later clock cycles but this is not harmful. So we can read these registers. So during the second clock period, what I can do is I can simply say that intermediate latch A will get the register output. Output from one of the registers where the address of the register will come from some bits in the instruction register. I am simply reading it, without knowing that they will be used.

Similarly I can also read the second register that is register B, the intermediate register B gets the second register output, the address of which also comes from some other bits in the instruction register. Simultaneously i can perform another thing. If it is a branch instruction then I have to compute the branch address. Now what I can do is I can pre-calculate the branch address irrespective of whether it is a branch instruction or not. So what I can do is I can simply put ALU out register to be program counter plus offset fields which comes from the instruction register. Now this instruction may not be a branch instruction at all.

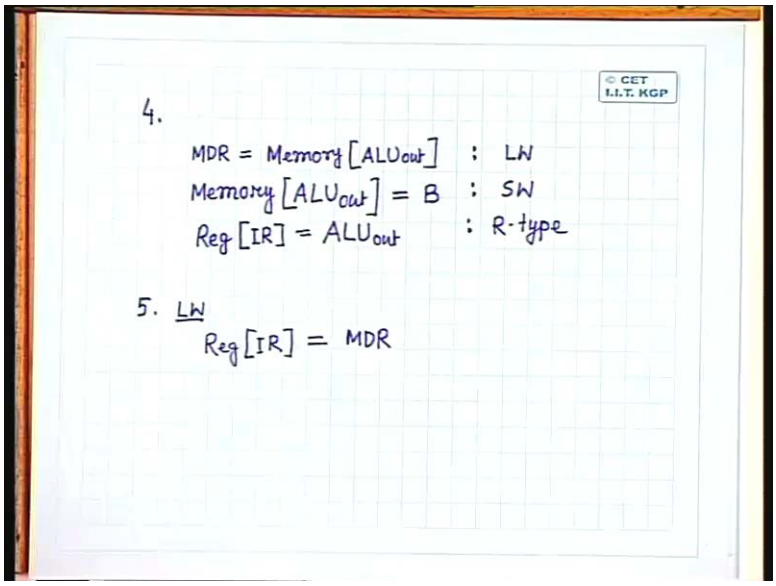
So in this case the ALU out register will be over written by the fresh data during later clock cycles but there is no harm if I pre-calculate this. But what is the advantage that I get? If I pre-calculate this and I find that the instruction is actually branch instruction, the data is already, branch address is already having it. So I can reduce one more clock cycle. During the third clock period, what we can do is we can execute some instruction or we can compute the memory address or if it is a branch instruction, we can complete the branch instruction. So if it is an M type instruction, memory type instruction in that case what we have to do is we have to calculate the memory address. So in that case the ALU out will be simply register A because for calculation of the branch address, this will become the base address plus the offset which comes from the instruction register.

This has to be done for a memory type instruction, if the instruction is a memory type instruction. If it is a register type instruction like add and all those things, in that case we will have ALU out is equal to the function that we have to perform on the registers R_1 and R_2. The contents of R_1 and R_2 are now available in the intermediate registers A and B, so that functional value will be loaded into ALU out register, if the instruction is an R type instruction. If the instruction is a branch instruction in that case branch output is already available in the ALU out from the previous step.

So what we will do is we will simply compare if A is equal to B then the program counter will get the value of ALU out. By loading this ALU out into program counter, your branch instruction execution is complete. So that is the advantage by precalculating the branch address. Now even if it is not a branch address, I don't have any harm because the ALU out is being over written by the proper values in step three. So this earlier value will be over written. So this is not harmful but in the third step, whatever operation you are performing that is instruction dependent. That means in the second step the instruction has been decoded and depending upon the decoder output, one of this operations will be performed.

Then what you do in the fourth step? during the fourth clock period, the operations that will be performed is if it is a memory reference operation then either the content of the memory has to be loaded into memory data register, if it is a load word instruction or the output of the register file has to be stored into memory if it is a store word instruction or output of ALU has to be loaded into the register, one of the register the write register if it is an R type instruction. So what we have to do is we have to do MDR, during step 4 will be memory address of this will come from ALU out, if the instruction that you are executing is a load word instruction or memory ALU out will get the value of B if the add instruction that we are executing is the store word instruction.

(Refer Slide Time: 00:35:24 min)



So that is what is performed through this block, output of B is going to the write input of memory. If it is an R type instruction then a register whose address comes from the instruction register that is the write register address, this gets the value from ALU out if the instruction is an R type instruction. Now here you find that every instruction if it is a branch instruction, the branch instruction was complete in step number three. If it is a store word instruction, the store word instruction is complete in step number 4. If it is an R type instruction that is also complete in step number 4. But if it is a load word instruction, it is not yet complete because only the data from the memory has been loaded into MDR, the memory data register at the end of step 4.

But finally this data has to go to the destination register. That means for a load word instruction during step number 5, during clock period 5 the operation that is to be performed is write register that is of which comes from the instruction register will get the value from memory data register or MDR.

So we find that out of these instructions that we are considering, it is the load word instruction which takes maximum number of clock cycles. All other instruction instructions take less number of clock cycles which was not possible in case of single cycle implementation. In case of single cycle implementation, for every operation we had to give the same amount of time but in this case, only the load word will get maximum amount of time others will get the time as required. At the end of each of them we have to design the controller in such a way that the machine will be put back into clock period zero or time state zero. So by doing this we can save a lot of time and the implementation becomes very efficient. So we have seen that by converting a single cycle implementation to a multi cycle implementation, our design becomes efficient and not only that I mean it will be more efficient so far as execution time is concerned. It is also optimized with respect to hardware because it is the same hardware unit which can be shared by more than one operation at different time instants.

In case of single cycle implementation we had duplicated hardware. So our hardware cost is also reduced. But is it the best design? While converting from a single cycle implementation to a multi cycle implementation, we have reduced the hardware but we have lost the facility of pipelining. In case of single cycle implementation by slight modification we can make that a pipeline processor but this one I cannot covert to a pipeline processor. So if we are not interested in pipelining this is optimum design. If we are interested in pipelining in that case some hardware duplication must be permitted, must be incorporated. So that we will see later.