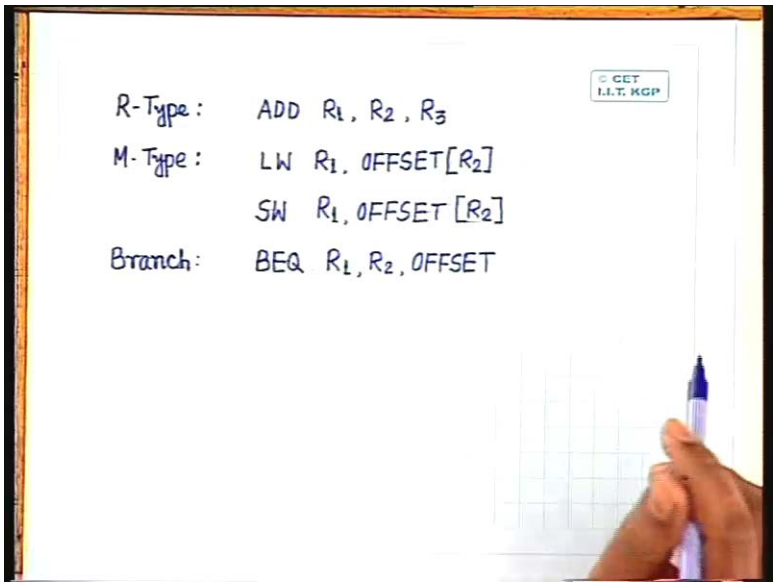
**Digital Computer Organization**
**Prof. P.K. Biswas**
**Department of Electronic & Electrical Communication Engineering,**
**Indian Institute of Technology, Kharagpur**
**Lecture No. # 10**
**Pipeline CPU – 1**

Towards the beginning of our course, we have talked about designing of a particular CPU, a common CPU and there our concentration was on defining the controls circuit of the CPU. Then in the latter part what we have seen is how to design a hardware control unit, how to design a micro program control unit and then we have seen that for improving the performance of the CPU, how we can pipeline different operations so that more than one instruction can be in the pipeline simultaneously but we dint talk about the pipelined CPU as such. We have taken in general pipeline architectures.

Now another aspect which is very important for improvement of the performance of any processor is not only the control unit, it is also the data path that is what are the paths that will be taken for transferring the data from one data unit to another data unit. So before we go to pipelined CPU, let us see for a typical advanced CPU, how the data path is to be decided. So for that we consider a particular architecture which is known as a MIPS processor and we will take few of the MIPS instructions from the MIPS instruction set to decide how we can design the data path. So few of the instructions that we will consider for deciding the data path is, we will consider a data instruction of type register or R type instruction. We will also consider an instruction of type memory or M type instruction.

(Refer Slide Time: 00:02:53 min)



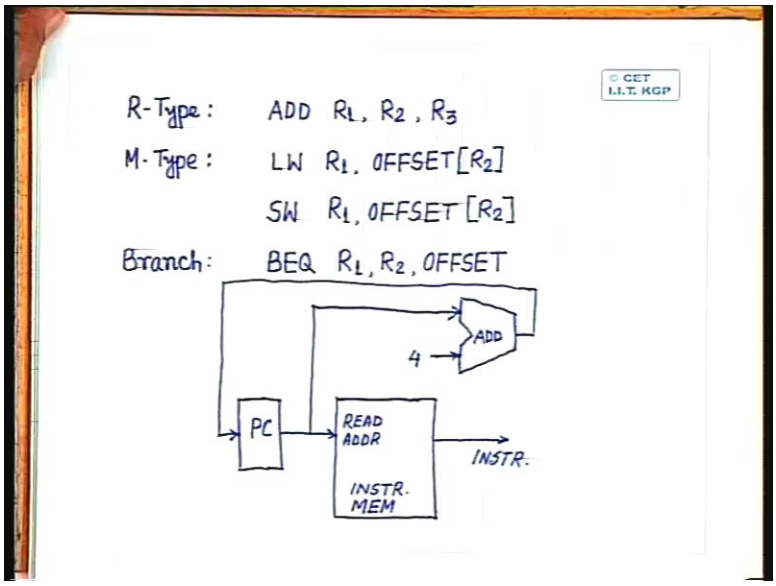

We will consider one branch instruction. So an R-type instruction is typically like this. If we want to perform addition of the contents of two registers, so we have two registers $R_1$ and $R_2$.

We want to add the contents of $R_1$ and $R_2$ and to the result, we will go the destination which in discuss we assume it is the third register that is $R_3$. So an ADD instruction or R type instruction typically will be of this form ADD $R_1$, $R_2$ and $R_3$ where the content of $R_1$ and $R_2$ will be added and the result will be stored in register $R_3$. We will consider 2 M type instructions, one is load and other one is store. So the instructions are like this LW which is load word and LW will work with arguments like this $R_1$ then offset $R_2$. So this means that the content of the memory location… Now the address of the memory location will be decided by content of register $R_2$ and an offset value. So this is also sometimes called relative addressing where $R_2$ will act as the base address. On the content of the base address, you add the offset value to get the physical address in the main memory. This load word will read through the content from that physical address of the memory and load that content into register $R_1$.

Similarly a store instruction, we put this as store word. The format will be similar, $R_1$ offset $R_2$ where you calculate the address of the memory location from the content of register $R_2$ and the offset value that is specified within the instruction to that particular location in the memory, you store the content of register $R_1$. So this is what store word instruction means. In the branch category we consider one instruction, let us say branch on equal or BEQ. The format will be BEQ $R_1$, $R_2$ and an offset. So when this BEQ $R_1$, $R_2$, offset this instruction is executed, what this instruction will do is it will compare the contents of register $R_1$ and register $R_2$. If the contents are same that is $R_1$ equal to $R_2$ then a branch will take place and the branch address will be given by this offset and when you specify this as offset that means this offset value will be added to the present program counter value and that tells you that from the location from where the next instruction will be taken for execution. So this offset is not the physical address of the instruction. It gives you a relative address, relative to the present program counter value.

Now when you execute these instructions as you have seen before also that the instruction fetch operation is common for all the instructions. So to decide the data path, let us see that what are the data components and what is the path that is informed in an instruction fetch operation. Obviously the data components which will be involved is a program counter because the program counter gives you the address of the instruction that is to be fetched and the other component to the data unit which will be informed is memory. Now in this case let us say that we have a memory which is called an instruction memory that stores only the instructions. Later on we will think of the other options. so we will have two components which are involved in any instruction fetch that is the program counter, instruction memory and the other component that is involved is some sort of an ALU which tells you that once an instruction is fetched what is the address of the next instruction to be fetched. So here for instruction fetch, we will have a program counter. We will have a memory and in this case let us say that it is instruction memory.

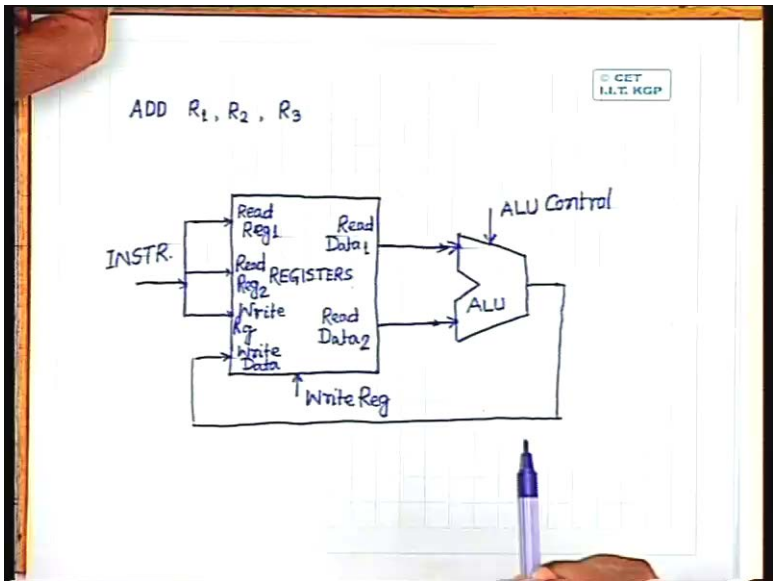
(Refer Slide Time: 00:08:35 min)




The program counter gives the address to the instruction memory that particular address in the instruction memory contains the instruction. So output of this instruction memory will be an instruction and this is the read address. As I said that the other component which will be involved in this instruction fetch operation is some sort of an ALU and the function of this ALU is only addition because the program counter has to be incremented by some amount after an instruction is read. So output of the program counter will also go to this one of the inputs to this ALU. The other input of this ALU, so now instead of calling it an ALU, let us call it just an ADD unit because only function that this ALU will perform is addition of some amount to the current program counter value. Let us assume that every instruction in this instruction set is of fixed length and the length is let us say it is 4 bytes. So we add 4 to this program counter content which gives you the address of the next instruction to be read from this instruction memory.

So these are the components, the data units and this is the data path which is involved in fetching an instruction and this is common for all the instructions. Now after an instruction is fetched, put into program counter, you read the corresponding instruction. the instruction is available at the output of the instruction memory. Then the execution of the instruction starts from this point onwards, once the instruction is available from the output of the instruction memory. Now let us see that for execution of different instructions, what type of units that you need and what is the data path involved in each of them. So firstly we will consider the R-type instruction that is ADD $R_1$, $R_2$ and $R_3$. So the instruction that will consider first is an instruction of the form ADD $R_1$, $R_2$ and $R_3$ where the contents of the registers $R_1$ and $R_2$ will be read and for the result will be written into register $R_3$.

(Refer Slide Time: 00:11:45 min)




So you find that there are three registers involved, two registers will take part in read operation and one register that is $R_3$ will take part in the write operation. Now there can be a number of registers in the CPU. Each of the registers will have an unique address. So if in this field, in the instruction if I specify 0 0 0 0 like this, in that case zeroth register in the registers I will put all the registers together, let us call it a register bank. So zeroth register in the register bank will be accessed. So to decide the data path for this and the units that are involved in execution of this instruction, we will put it like this. We have a register bank, in some book you will find that this is also mentioned as register file. So this is the register bank and three addresses are involved in this. One for register $R_1$, one for register $R_2$ and the other for register $R_3$. All these addresses will be mentioned within the instruction.

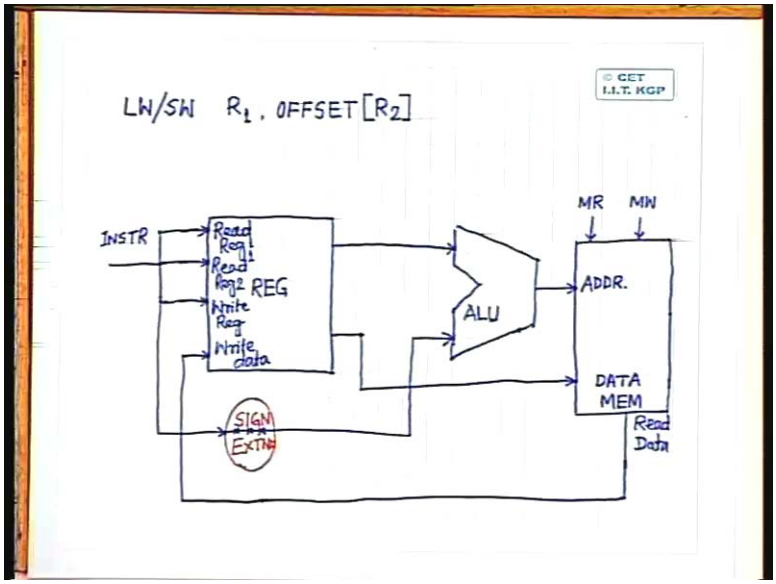
So I have to have address for register $R_1$, I have to have address for register $R_2$, I also have to have address for register $R_3$. So this is for read operation, so read $R_1$ or we put read register one. This address will be for reading register two and the third address is meant for writing into a register, so this is for write register. The other input to this register bank will be the data that is to be written into one of the registers. So the third input will be meant for write data. This register bank will have two outputs, the outputs corresponding to output of register $R_1$, content of register $R_1$ and content of register $R_2$. So I will have two outputs, one is the read data 1 and other one is read data 2.

Now since the operation to be performed is adding the content of register $R_1$ and register $R_2$ which are available on the output of this register bank, they have to be added together and the result has to go to register $R_3$. That means this outputs are to be fed to the input of an ALU. So I have to have an ALU and it is an output of the ALU that has to come to this write data input of the register bank. Is it okay? So you provide three addresses from the instruction address for two registers which are to be read and address of the third register which has to be written and another input that is provided to the register bank is the data input which is to be written into the third register.

All these register addresses will come from the instruction. So it is the instruction which is giving you the addresses of the registers which are to be read. It is also telling you the address of the register which is to be written. Then the data to be written is the output of the ALU which works on these two output data and the control signal that will be needed to this register is a write control signal for writing into the register $R_3$. So whatever data is available on the write data input that will be written into the register on asserting this write register control signal into a register whose address comes on the write register input of this register bank. So this will be the typical components needed for an R-type instruction of which a typical example is ADD $R_1$, $R_2$, $R_3$. [Conversation between Student and Professor – Not audible (Refer Slide Time: 00:17:45 min)] what we can assume is that whenever the inputs, the addresses are available to these inputs the data is always available on this outputs. It can be simply a combinational circuit.

The other set of control signals that you need is the ALU control signals or ALU function select signals because instead of an ADD operation, if we want to perform an AND operation over these two registers $R_1$ and $R_2$ then this ALU has to perform an AND operation other than an ADD operation and which operation this ALU will perform that depends upon what is the control signal that you apply to the ALU input. So this will be the typical organization of different components and their interconnections for an R-type instruction. [Conversation between Student and Professor – Not audible ((00:18:41 min))] We will come to that later.

(Refer Slide Time: 00:19:06 min)




Then the second kind of instruction that you have said is M-type instruction where our instruction format is something like this, either we can have a load word or a store word on $R_1$ then offset $R_2$. In case of load word this memory address will be read and the data will be written into $R_1$. In case of store word the data from $R_1$ will be written into the memory whose address is given by this. Again in this case your register bank will be needed. Isn't it? To identify these two registers, which registers you are using. So again I need the register bank. So these are the registers, I also need the ALU but in this case the ALU is needed not for performing any AND or ADD operation but the ALU is needed to calculate the memory address from where the data is to

be read or from where the data is to be written. So again this register bank, it will have three inputs but out of that we will need two register identification numbers. All three are not needed for this particular instruction.

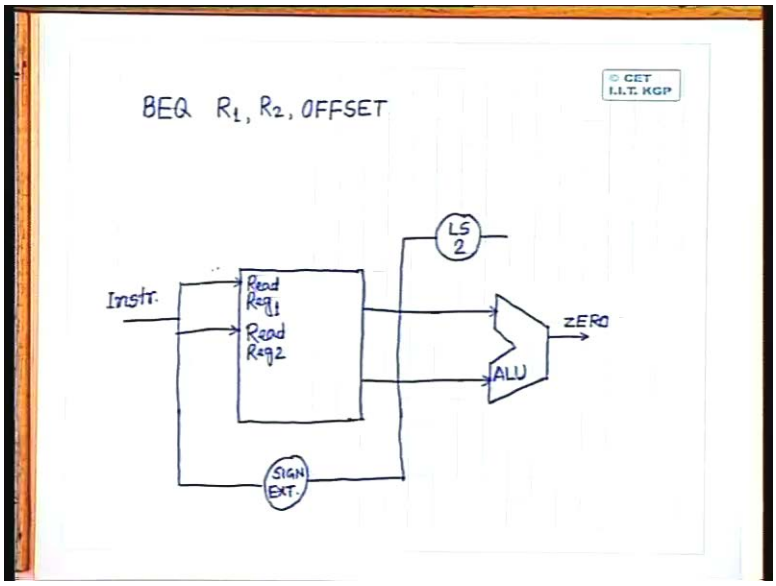
For store operation I need write register and write data. For load operation I need write register and write data because in case of load, the data has to be written into the register. In case of store word, the data is not to be written into the register but the data will be written into the memory and it will be read from the register. So in this particular case I need say read register one, I may also need read register two, I may also need write register, I may also need the write data. I also need the ALU for the calculation of the address. One of the outputs from this registers will directly go to the ALU. Isn't it? Because one of the contents of one of the registers is to be added with offset value. This register addresses, register identification numbers will come from the instruction. All of them will come from the instruction. The instruction also contains the offset value.

So in this case the other input to the ALU will not be from the registers but the other input to the ALU will come from the instruction. So this is my ALU. Is it okay? Now what is usually done is in case of MIPS architecture, this offset is given as a 16 bit watt whereas the total addressing capacity of a MIPS architecture is a 32 bit watt. So when I have a 16 bit offset, this 16 bit data or the address has to be sign extended to 32 bit. So in between we can have one more unit which is for sign extension. Data path will be modified like this. In between we have a sign extension unit, so the offset is inputted to the sign extension unit. So this direct path is no more there and output of the sign extension unit goes to the ALU. ALU output is the physical address of the memory which is to be written or to be read.

In case of load word, the memory will be read. In case of store word, the memory will be written into. So I have a memory block here and in this case let me call it as data memory. This is your data memory, address to the data memory comes from the ALU. So this is the address. Now the data memory gives you the output. Let me take it from here which is the read data. In case of load word, this data output from the data memory has to be given back to the write data input of the register bank because you are reading from the memory location, loading the data into register $R_1$. So in case of load word instruction, the write data of the registers will come from the read data from the memory. Whereas in case of store word, it is the content of one of the registers which is to be written into the data memory.

So other output of the registers will now be connected to the data input of the memory. The control signals that you need for this memory is, one is memory read which is used in case of load word and the other control signal that will be used is memory write which will be used for store word instruction. So you see that how you are deciding the data path, depending upon the type of operation that is to be performed we decide that what kind of data path will be needed. So this is for the second category of instruction that is accessing the memory either for load operation or for store operation.

(Refer Slide Time: 00:27:17 min)

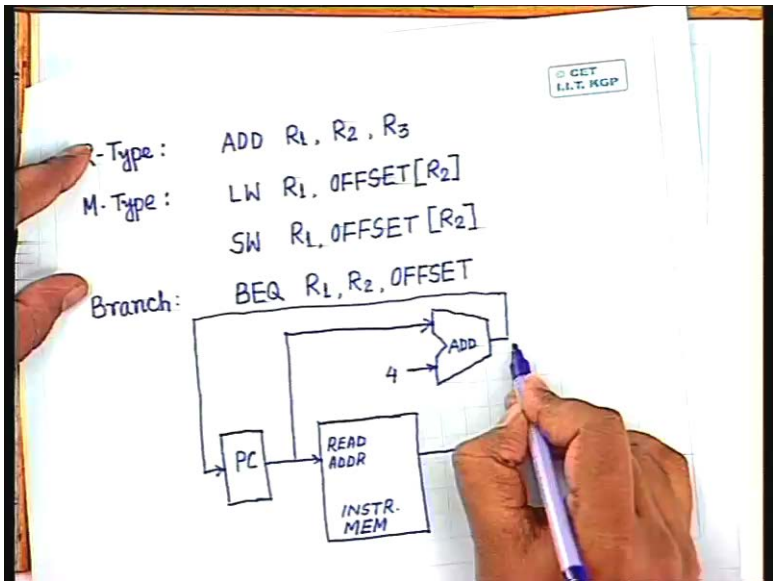



The data path that will be used for third kind of instruction that we are using is for branch on equal that is BEQ and the format is BEQ $R_1$ $R_2$ and an offset. So you compare the contents of registers $R_1$ and $R_2$. If they are same, if they are equal then you jump to a location which is away by this amount offset from the current program counter value. So obviously that register bank is also involved because I have to read the contents of registers $R_1$ and $R_2$. Then they are to be compared, so ALU is also involved. So here again I have this register bank which will have read register 1 and a read register 2 control units. So this is for read register 1 and not control units sorry the address inputs, read register two address. There are other inputs also but in this particular case they are not needed. This output will go to an ALU which will compare the two register outputs. So this will go to ALU.

The ALU will perform a separate operation because that is the way in which we have compare these two data. It will perform a separate operation but the output will not go anywhere but the zero out condition will be used by the control unit to decide what will be the address of the next instruction to be executed. So this ALU in addition to giving you the difference output, it also gives you an output whether the result is zero or not. If the result is zero in that case what we have to do is we have to add the current program counter value with this offset but again this offset is given in the form of 16 bit which is to be sign extended to 32 bit and all of them will come from the instruction. So all this comes from the instruction. The offset is sign extended to say 32 bits so I will have a sign extension unit. The other thing that is done for calculating this branch address is after sign extension, we have to ensure because we have assumed that every instruction is of length 4 bytes that means whatever address we generate that will always be generated at 4 byte offset, at 4 byte bounded. So another block that is put is that after you sign extend, generate the offset address. You give left shift by two bits to the offset address. Whenever giving left shift by 2 bits that is equivalent to multiplying by 4. That ensures that any new instruction that is to be fetched will start from four byte boundary. That means it is the beginning address of a new instruction. So this sign extended bit will now go to a two bit left shifter, so I have left shift by two bits.

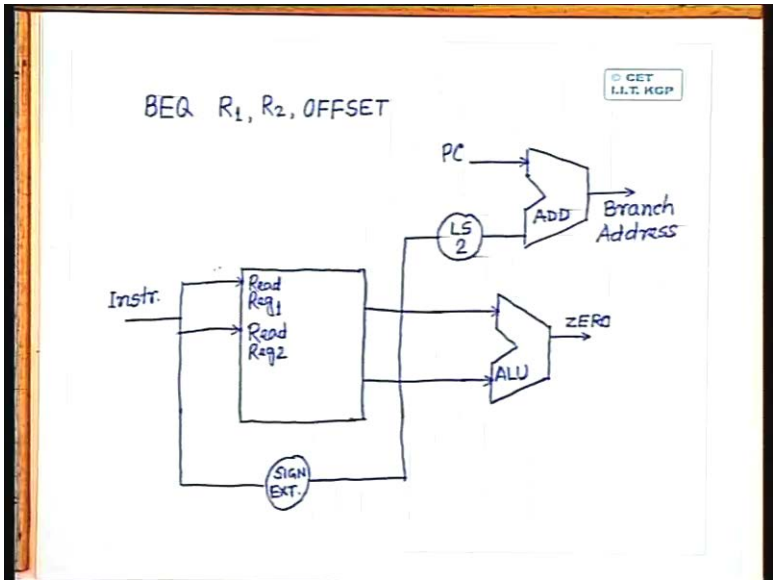
So this gives me the final offset. This final offset is to be added with the current program counter value and the current program counter value, you will get from the instruction fetch unit.

(Refer Slide Time: 00:32:00 min)




This output is the current program counter value.

(Refer Slide Time: 00:32:05 min)

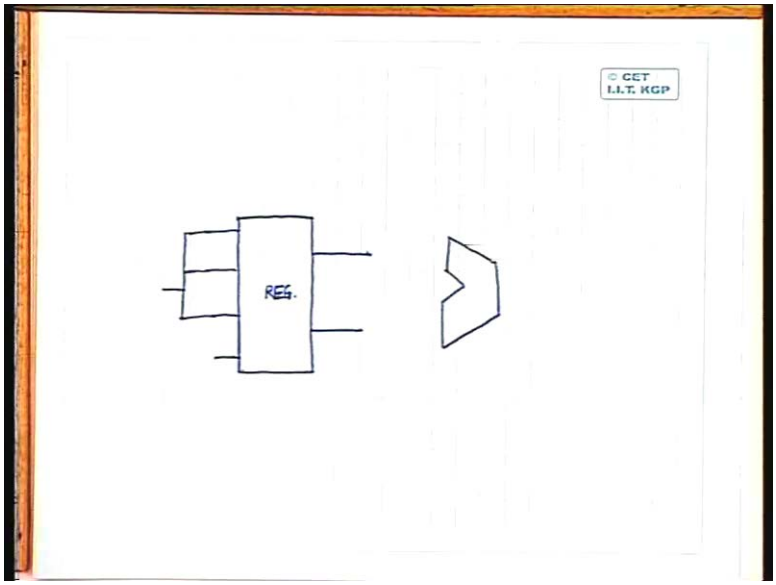



So this left shifted offset is to be added with the current program counter value. So I need an adder unit here. Here what I get is the program counter and this output, this is a simple adder unit. This output gives me the branch address.

So we will find that we have been able to identify that what are the data units and what will be the data path among different data units for the fetch operation and for execution of three different kinds of instructions but obviously in a CPU, I cannot have different units for execution of different instructions. All of them to are to be combined together.

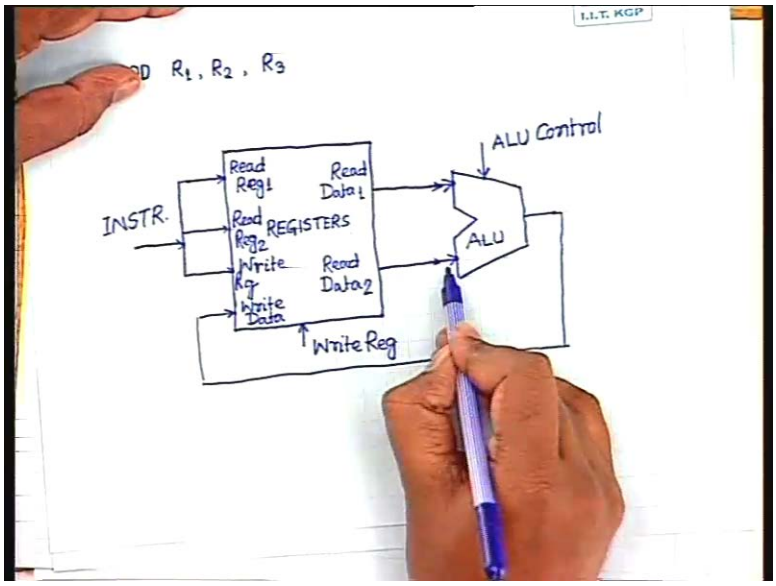
So next what we have to think is how to combine all these pieces together so that we can have a complete CPU. So for that we have to see that how, in case to a particular source the data comes from multiple sources to a particular destination, how to mix those different source outputs. So one way and the simplest way for doing that is to use a multiplexer. So first let us complete the data parts and then we will try to combine this with the instruction fetch part. In all these cases we have found that your register bank is involved, so I put this register bank.

(Refer Slide Time: 00:34:01 min)



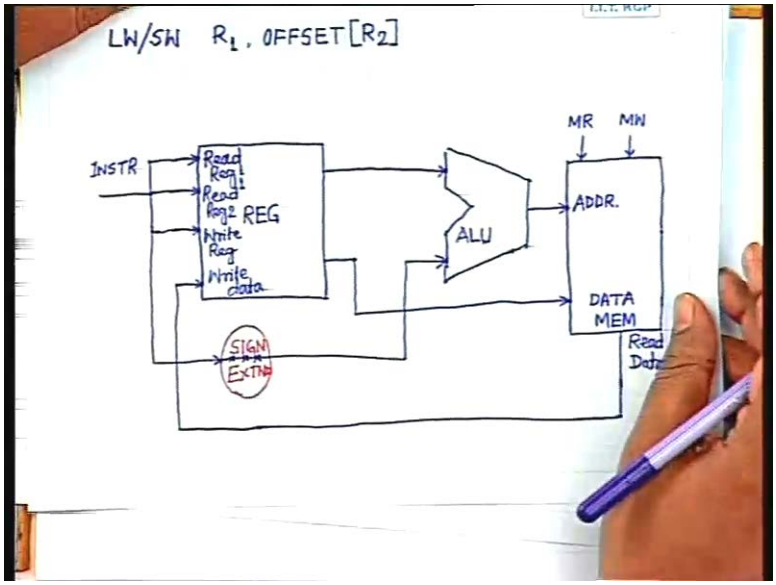

So these are the register bank. Register bank gives you two outputs and it has got four inputs, four inputs are meant for read register one, read register two, write register and write data. So I have four inputs to this register bank. Out of these four inputs, three of the inputs comes from the instruction, so this comes from the instruction. Now these outputs will go to the ALU. When it comes to an ALU, you find that in case of ALU the second input to the ALU can be from various sources. So you find that the first instruction that you have considered is for ADD instruction.

(Refer Slide Time: 00:35:26 min)



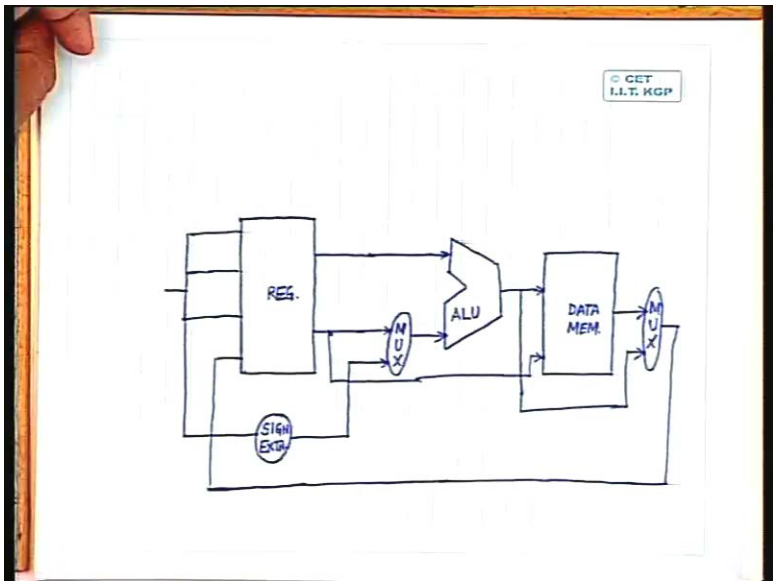

In case of ADD instruction, the second input to the ALU comes directly from the register bank. First input also comes directly from the register bank.

(Refer Slide Time: 00:35:36 min)




In case of load or store instruction, first input to the ALU comes from the register bank whereas the second input comes from the instruction offset. In case of branch on equal instructions, second input also comes from the register bank that is there are two sources from which the second input to the ALU will come. So I have to multiplex those two sources.

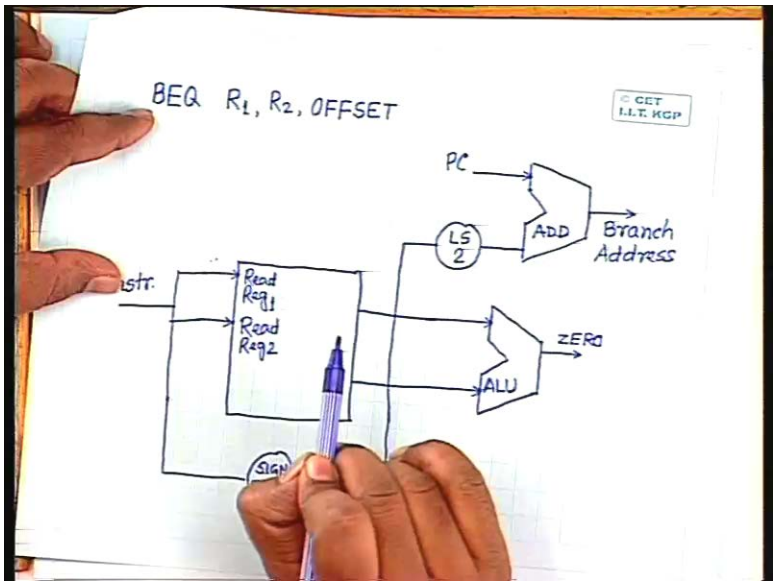
(Refer Slide Time: 00:36:02 min)




So what I need here is a multiplexer. One input to the multiplexer is the second output of the register bank. first output of the register bank can directly go to the ALU. The other input of the multiplexer will be the offset from the instruction after sign extension. So this has to be sign extended and this becomes the second input to the multiplexer. Multiplexer output feeds the second input of the ALU, so this is my ALU. Output of the ALU gives the address to the memory, data memory. So this is data memory and output of the ALU gives you the address. Now coming to the write data, you find that for register type operations the write data comes from the ALU output. Whereas for load type instructions, the write data comes from the data memory. That means here also I have to have a multiplexer, I will put a multiplexer here. One of the inputs to the multiplexer is the data output of the data memory.

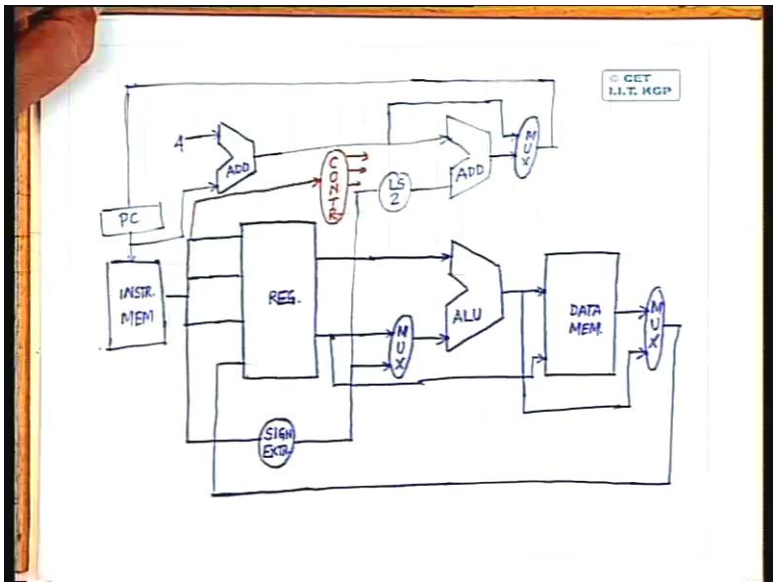
One of the inputs to the multiplexer is data output of the data memory. The other input to the multiplexer is the output of the ALU and multiplexer output can now be connected to the write data input of the register bank. So this path is also complete. Whenever I wanted to write the data into the memory which has to come from one of the register bank outputs because that is needed for store word instruction. So the second output of this register bank can be connected to the data input of the memory unit. Is it okay? So you find that we have combined the data paths within this data unit for all types of instructions. The next operation that is to be performed is how to take care of the branch instructions? In case of branch instruction what you have said is that is this offset value, sign extender offset value is left shifted by two bits and this left shifted output is added with the current program counter.

(Refer Slide Time: 00:39:24 min)




The current program counter value we have generated with the help of an adder like this, so here we had a program counter. I have the instruction memory which gets address from the program counter. The same program counter output goes to this adder to add with a number four and this gives the current program counter value. This sign extender offset is to be left shifted by two bits, so I have to left shift by two bits and after left shifting this by two bits, we have to add this with the current program counter value. So I need another adder. Output of this, it's the second input of this adder.

(Refer Slide Time: 00:39:34 min)



How do I decide the next program counter value? The next program counter value can be either this or output of this adder. If it is a branch instruction that is being executed then the next program counter value will be output of this adder, in case the branch condition is satisfied. So I put one more multiplexer here, this multiplexer will multiplex between the branch address or the normal program counter flow and output of this multiplexer can be loaded to the program counter and this completes my entire data path.

Once this data path is completed then we have to design what will be the control circuit and as we said, as we have discussed earlier the control circuit always takes input from the instruction code. The instruction is decoded and depending upon the decoded instruction, the control circuit generates the control signals. So here we have to have a control unit. The control unit will take a part of the instruction and it will generate all the required control signals. What are the control signals required? Signals for controlling the ALU, for selecting what function the ALU has to perform, selecting memory read or memory write operations, selection of these multiplexers which input of the multiplexer has to be passed to the output. So all these control signals have to be generated by this control unit. So this will be the complete structure or complete organization of an advanced CPU.

Now find that if you analyse this circuit, there is no reason why every instruction cannot be executed in single clock period. You find that whenever an instruction is available here or may be whenever the program counter value is that, the program counter will give the address to the instruction memory, instruction memory output will be the instruction code. All these paths will be set, registers will be read or the write data to the register will be set. The ALU will operate, the data can be read from the data memory or it can be written into the data memory depending upon the control signal that you get. So it is possible that all these operations can be drawn in a single clock period because all these components which are involved in this path they are all combinational logic circuits. There is no sequential logic circuit.

The sequencing is done by the control unit. So if I want the different phases of this instruction execution has to be done in different clock periods then this control circuit will generate those control signals in sequence but it is possible that in a single clock period all these operations can be done. But what is the disadvantage? If I want to perform all these instruction execution of every instruction in a single clock period, there is a problem. The problem is the clock period now will be decided by the largest delay path and that is required for load and store instructions because it is the load and store instructions which makes use of instruction memory, register, ALU, data memory, everything. Whereas for register type instructions, the components which are used is only instruction memory, in any case will be there; only the register bank and the ALU.

So I have a shorter delay path. So if I want to perform this operations in a single clock period, my clock period will be decided by the largest delay path which will also be applied on the instructions which actually require less amount of delay. So effectively you are making your design inefficient. Instead of doing that, what you can do is you simply increase the frequency, the clock frequency and decide that how many clock frequencies, how many clock periods will be allocated for execution of fetch instruction. In that case in general your design will be more efficient. We will come to more on this in the next class.