

**Hardware Modeling using Verilog**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 37**  
**Pipeline Implementation of A Processor (Part 1)**

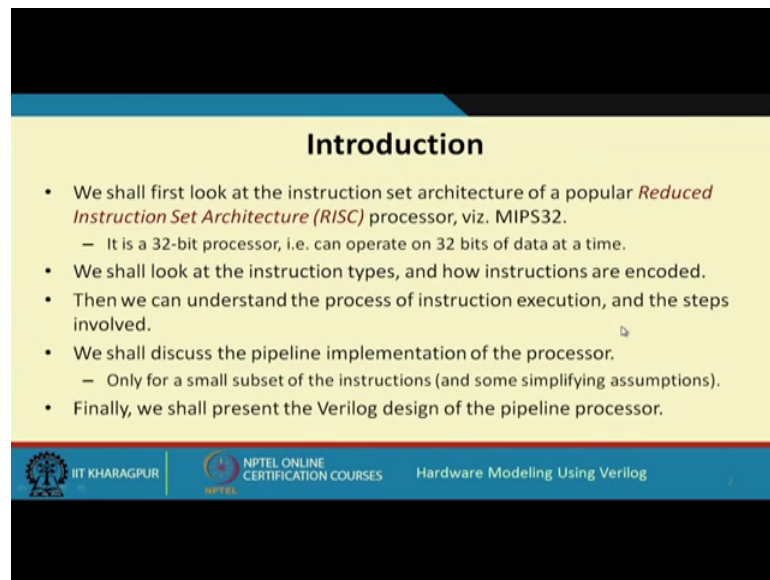
So, in the last few weeks we have seen how we can design and use the various features available in the verilog language and through a number of examples we saw how we can model both combinational and sequential circuits. Towards the end we have seen how we can design efficient digital circuits say using the concept of pipelining let say. So, we saw that if we use pipelining then without investing more in terms of hardware we can gain a significance speed up.

Now, we shall be looking at a much more complex example namely that of designing a complete processor, see when I talk about a processor I talk about something called an instruction set architecture. So, when you are using a processor or programming a processor you have some internal registers and you have an instruction set and using those instructions belonging to the instruction set you can write a program to solve any problem, this is the idea.

So, in this lecture we shall be starting our discussion on the design of a processor in verilog with an utilizing the concept of pipelining and the kind of processor that we look at is something called reduced instruction set architecture or reduced instruction set computer, risk in short R I s c which is rather easy to implement in hardware and in particular in a pipeline ok.

So, the topic of our discussion in this lecture is pipeline implementation of a processor the first part.

(Refer Slide Time: 02:21)



**Introduction**

- We shall first look at the instruction set architecture of a popular *Reduced Instruction Set Architecture (RISC)* processor, viz. MIPS32.
  - It is a 32-bit processor, i.e. can operate on 32 bits of data at a time.
- We shall look at the instruction types, and how instructions are encoded.
- Then we can understand the process of instruction execution, and the steps involved.
- We shall discuss the pipeline implementation of the processor.
  - Only for a small subset of the instructions (and some simplifying assumptions).
- Finally, we shall present the Verilog design of the pipeline processor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

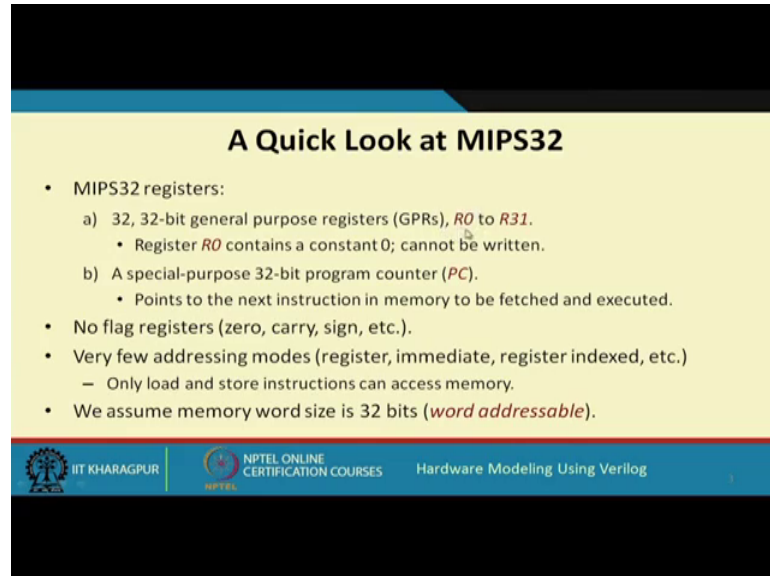
Now, as I have said that we shall be looking at the instruction set architecture of a particular processor which belongs to the risk category reduced instruction set architecture. Now, now as them implies a risk architecture or a risk instruction set will consist of fewer number of instructions simpler instructions and large number of registers mostly general purpose registers the result is that it is much easier to implement this kind of processors in hardware.

So, this is the reason why we have chosen a risk architecture as the, you can say platform for our example. So, a few things about the processor the risk processor we are looking at is MIPS 32 this is a well known processor, but actually we are not considering the entire instruction set of MIPS 32, but rather a small subset of it because our objective is to show you how we can implement the processor in verilog, let us say we start with 50 instructions, but if I give you 50 instructions following the same principal you can also implement them. So, the purpose of taking a small subset is to illustrate how we can design the processor fine.

So, the important thing to look at before we think about implementation is what are the instruction types and how the instructions are encoded and after we have seen this we shall be understanding the different steps of instruction execution right. So, as I have said we shall be looking at the pipeline implementation of this particular processor using a

small subset of MIPS 32 instruction set, not all the instructions right and towards the end we shall be looking at how we can model this pipelined implementation in verilog.

(Refer Slide Time: 04:38)



**A Quick Look at MIPS32**

- MIPS32 registers:
  - a) 32, 32-bit general purpose registers (GPRs), *R0* to *R31*.
    - Register *R0* contains a constant 0; cannot be written.
  - b) A special-purpose 32-bit program counter (*PC*).
    - Points to the next instruction in memory to be fetched and executed.
- No flag registers (zero, carry, sign, etc.).
- Very few addressing modes (register, immediate, register indexed, etc.)
  - Only load and store instructions can access memory.
- We assume memory word size is 32 bits (*word addressable*).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Let us have a quick look at MIPS 32 the main features, first thing is that there are 32 general purpose registers they are called R 0 to R 31 and each of this registers are of 32 bit in size.

Now, we use this registers for temporary storage of data during some computation because there are large number of registers 32 there is lot of flexibility available to the programmer and this register R 0 is a special register which is a assume to always contain the constant 0, means you cannot write any other value in to R 0 because in many application you need the number 0 so you can use R 0 for the purpose. There is a program counter which is also a 32 bit register and a program counter you may be knowing this is a register which always points to the next instruction in memory which is to be fetched.

So, when an instruction is being in executed we have to fetch the instruction from memory decode the instruction what kind of instruction it is and then follow through the steps of execution and while we are doing this we will also be incrementing p c to point to the next instruction. So, that after our current instruction execution is over we can go back and fetch the next instruction right ok.

Now, in MIPS 32 there are no flag registers as are present in many other processors flag registers means 0 flag, carry flag, sign flag some of you may be aware and there is a very limited set of addressing modes register immediate register indexed and not many more and another important thing is that the only instructions that can access memory are load and store and all other instructions they operate only on the c p u registers. They cannot use any data directly that stored in memory you will first have to load the data in to a register and then use it in some competition right and another assumption we are making because everything is 32 bits to makes thing simple we are assuming that memory is word addressable meaning that every word has a unique address and the memory word size is 32 bits.

(Refer Slide Time: 07:23)

**The MIPS32 Instruction Subset Being Considered**

- Load and Store Instructions
  - LW R2,124(R8) // R2 = Mem[R8+124]
  - SW R5,-10(R25) // Mem[R25-10] = R5
- Arithmetic and Logic Instructions (only register operands)
  - ADD R1,R2,R3 // R1 = R2 + R3
  - ADD R1,R2,R0 // R1 = R2 + 0
  - SUB R12,R10,R8 // R12 = R10 - R8
  - AND R20,R1,R5 // R20 = R1 & R5
  - OR R11,R5,R6 // R11 = R5 | R6
  - MUL R5,R6,R7 // R5 = R6 \* R7
  - SLT R5,R11,R12 // If R11 < R12, R5=1; else R5=0

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

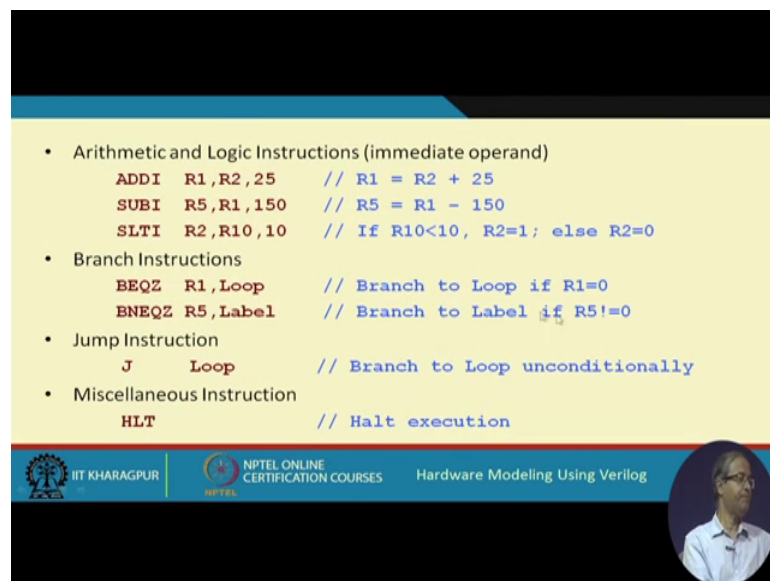
So, the instructions subset, that you are considering are summarized. Firstly, there are load and store instructions. So, I am illustrating with examples how they look like, load instruction is like this load word LW and this R 2 is a register R 8 is a register and this 1 2, this is how we write. The meaning is the value of the R 8 register is added to this number we have specified we call this an offset, we add them up that is treated as a memory address, from that memory address we fetch or read the content of the memory location and the data is loaded into R 2. So, the memory is loaded into R 2.

Similarly, store word, store word is reverse same is a register R 5 its value will be stored in a memory location where same way the content of the register R 25 you add minus 10

to it; that means, minus 10. these numbers will be represented in 2's complement signed form. This will be seen later, there are some arithmetic and logic instructions which will be considered. These few numbers where only register operands are used like `add R1, R2, R3`. So, here the convention is that the first register is our destination, the other 2 are the source.

So, `add R1, R2, R3` means add R2 and R3, store the result in R1. Similarly, `add R1, R2, R0` means you add R2 to R0, which always contains 0, and store it in R1. So, you see this is an indirect way to move the content of a register into another register. This effectively is copying the value of R2 into R1. So, when you need to do this, you can use this instruction with R0. Subtract a similar `sub R10, R8, R12` and bit by bit and similar bit by bit or multiply `R6, R7, R5` and this is a special instruction. This is called set less than, set less than means it checks whether the second operand R11, R12 is less than R. Not if it is less than then the target register is set to one, otherwise the target register is set to 0. OK.

(Refer Slide Time: 10:10)



```
• Arithmetic and Logic Instructions (immediate operand)
  ADDI R1,R2,25 // R1 = R2 + 25
  SUBI R5,R1,150 // R5 = R1 - 150
  SLTI R2,R10,10 // If R10<10, R2=1; else R2=0

• Branch Instructions
  BEQZ R1,Loop // Branch to Loop if R1=0
  BNEQZ R5,Label // Branch to Label if R5!=0

• Jump Instruction
  J Loop // Branch to Loop unconditionally

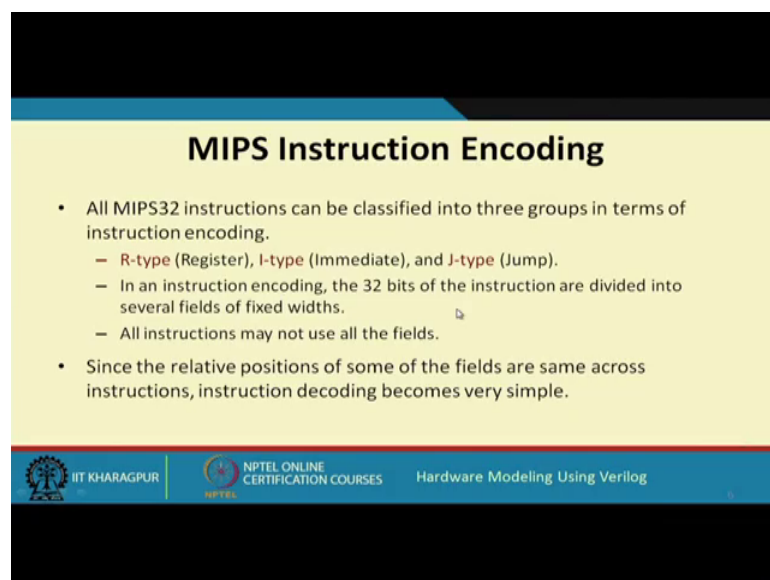
• Miscellaneous Instruction
  HLT // Halt execution
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

These are some of the instructions and there will be some arithmetic and logic instructions where some offset or an immediate operand is given like you have a version of add and subtract instruction we call them add immediate, subtract immediate. So, how do you use it? `add immediate R1, R2`, the last parameter is a number. This means you add the contents of R2 with 25, result stored in R1.

Similarly, sub immediate R 5, R 1 150 so, R 1 minus 150 store in R 5, similarly there is a version of set less than immediate. So, you can right like this R 2, R 10 and a constant 10. So, here you compare with a R 10 is less than 10 or not if it is true then the target R 2 set to 1 else R 2 set to 0. There are 2 types of branch instruction would be considering branch equal to 0 and branch not equal to 0, branch equal to 0 means there is a register specified we check whether this register is 0 or not, if it is 0 then we jump to loop otherwise we proceed with the next instruction similarly not equal to 0. If R 5 is not equal to 0 then you jump to this level and there is some unconditional jump instruction J loop it will always jump, but of course, here we shall not be implementing this jump loop and log the last instruction of our program will be halt instruction. So, it will stop the instruction execution.

(Refer Slide Time: 11:53)



**MIPS Instruction Encoding**

- All MIPS32 instructions can be classified into three groups in terms of instruction encoding.
  - R-type (Register), I-type (Immediate), and J-type (Jump).
  - In an instruction encoding, the 32 bits of the instruction are divided into several fields of fixed widths.
  - All instructions may not use all the fields.
- Since the relative positions of some of the fields are same across instructions, instruction decoding becomes very simple.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

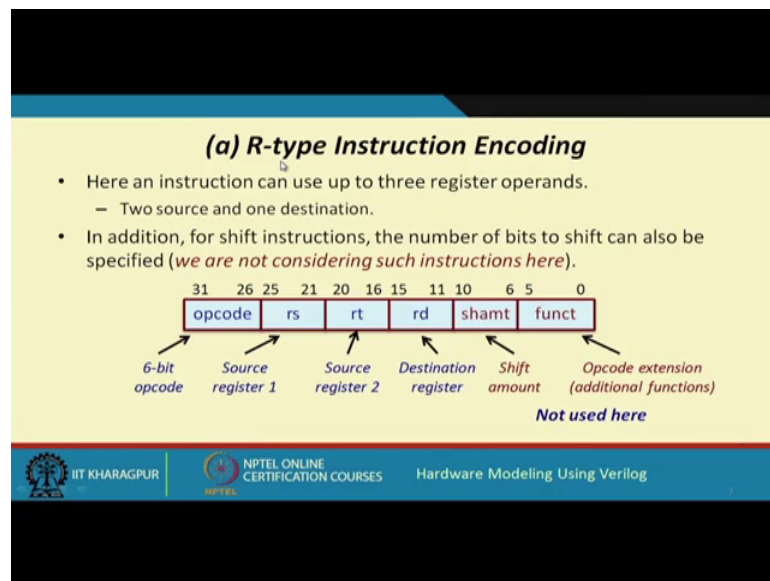
Now, it is important to know that how this instructions are actually encoded; that means, in terms of bits I told that all word size in memory is 32 bits. So, every instruction will also be encoded in 32 bits. So, we will have to know what this 32 bits actually indicate ok.

So, once you know it only then you can proceed with the hardware implementation or the verilog modeling whatever you say. So, we need to understand how this instructions are encoded, broadly speaking MIPS 32 instructions can be classified into either register type, immediate type and jump type of course, we are not considering jump type in our

implementation because in the previous slide we said the J instruction we are not implementing so J instruction uses this type right.

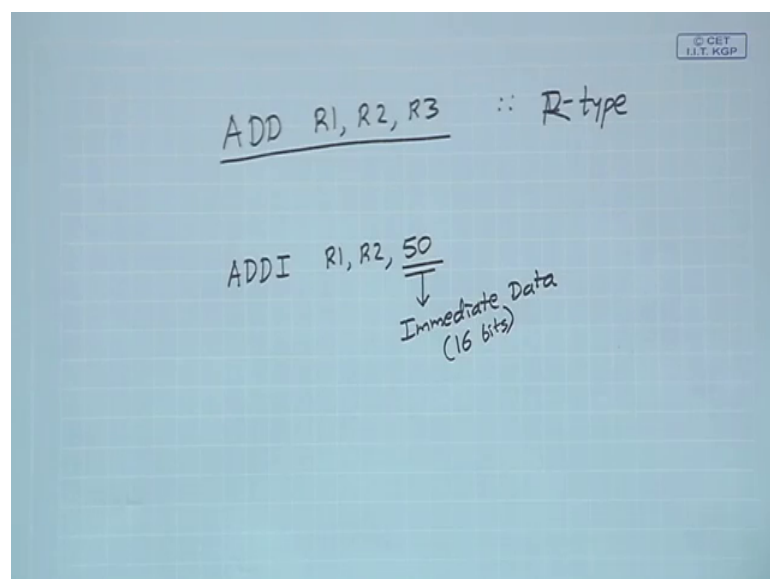
So, when you say encoding as I said the 32 bits of the instruction they will be divided in to some fields and each field will be having some fixed widths and all instructions may not be using all the fields let see that what this means.

(Refer Slide Time: 13:11)



Let start with the R type instruction encoding, say R type instruction encoding is like the add instruction add R 1, R 2, R 3 like the instruction that I have said.

(Refer Slide Time: 13:23)



Some instructions like this where there are 3 register operands. So, you add R 2 and R 3 store the result in R 1 this is an example of I type instruction, not R type R type instruction encoding register type right.

Now, in this type how we are encoding in 32 bits you see the first 6 bits bit number 26 to 31 this is 32 bit word 0 up to 31 this we are calling op code. This op code actually tells you what kind of operation this instruction refers to because we are considering only a small subset of instructions 6 bits are sufficient for us.

So, in 6 bits actually we can represent  $2^6$  or 64 possible instructions, but in our case it is much less. So, 6 bits is more than enough for us, the next 3 fields they indicate some registers in R type instruction there will be 2 source registers and 1 destination register. The source registers are stored in the first 2 fields it is called r s and r t they are stored in bit numbers 21 to 25. So, recall there are 32 registers right R 0 to R 31. So, to address 32 registers uniquely you require 5 bits because  $2^5$  is 32.

So, in all these register fields you need 5 bits each, 5 bits here, 5 bits here and 5 bits for the destination. So, the bits are given 11 to 15, 16 to 20 and 21 to 25. Now the last bits there used in MIPS 32, but for our implementation we are not using them, because this 5 bits is normally used to specify something called shift amount and last 6 bits is used to specify some ALU operations which is sometimes called op code extension, but these 2 we are not using here in this implementation. So, let us ignore this we will be setting all these bits to 0s.




(Refer Slide Time: 15:46)

• R-type instructions considered with opcode:

Instruction	opcode
ADD	000000
SUB	000001
AND	000010
OR	000011
SLT	000100
MUL	000101
HLT	111111

```
SUB    R5, R12, R25
000001 01100 11001 00101 00000 000000
SUB    R12  R25  R5
= 05992800 (in hex)
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

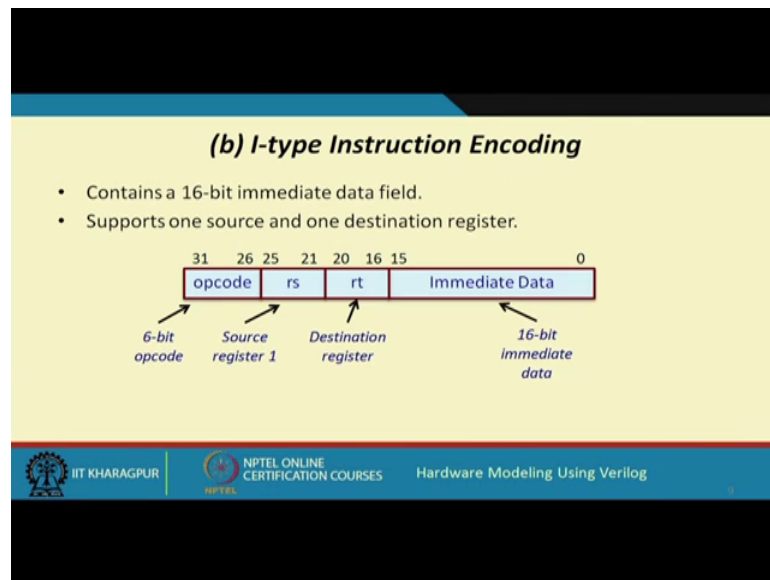


Let us take some examples. So, the instructions that we are considering these instructions can fall in the R type category they require 3, 3 registers most of them halt does not require any 0 registers. So, halt we are also putting in this category because you can fit it in the same thing because op code will be halt other fields will all be blank they will not be required and you are assuming that these are the opcodes, all 0 is for add 1 is first sub 2, 2 in decimal is and 3 for or 4 for s l t, 5 for ml t and all ones this is the actually 30, this is actually 63 in 6 bits this will represent halt.

Now, an example of encoding let us take a subtract instruction sub R 5, R 12, R 25. So, subtract the op code is 00001. So, when you convert it to the instruction format for subtract to write this 0001 then the 2 source registers R 12 and R 25, 12 in binary is 01100, this indicates register 12 and 25 is 11001 this indicate register 25 and destination is 5, 00101 is R 5 and the last bits we are not using as I said these are blank.

So, if you break them up into four bits each and write in hexadecimal this instruction in hexadecimal will be 0599200, right.

(Refer Slide Time: 17:30)



Fine now let us look at something called I type or immediate type instruction encoding, let us take an example what kind of instruction talking about, let say add immediate. Say add immediate we are saying R 1, R 2 then a number 50 this 50 is sometimes known as immediate data, that is why we call it immediate mode or I mode. So, there will be 2 registers the operation and a immediate data now here this immediate data is encoded in 16 bits right.

So, you see so how the instruction encoding is done here again the first 6 bits are for op code the next 5 bits are for the source register, like in this case the source register is R 2 in the example that we have given R 2 is the source register and R 1 is the destination register right. So, the first field is for the source the second field is for the destination because we need only 2 registers and the last 16 bits is left for the immediate data 16 bit immediate data. Sometime this also represents an offset for instructions like branch this we shall see.

(Refer Slide Time: 18:57)

• I-type instructions considered with opcode:

Instruction	opcode
LW	001000
SW	001001
ADDI	001010
SUBI	001011
SLTI	001100
BNEQZ	001101
BEQZ	001110


**LW R20, 84 (R9)**

```
001000 01001 10100 0000000001010100
LW   R9   R20   offset
= 21340054 (in hex)
```

**BEQZ R25, Label**

```
001110 11001 00000 yyyyyyyyyyyyyyyy
BEQZ  R25  Unused  offset
= 3b20YYYY (in hex)
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



Now, the instructions that fall under this I type category are, are these load word, load store they also belong to this category load store add immediate, subtract immediate, set less than immediate, branch not equal to 0 and branch equal to 0 and these are the op codes that are chosen by us 6 bits.

Now, some example encoding take a load instruction. So, I said earlier there are load instruction looks like this. So, R 9 is added to 84 that will be our memory address from there data will be loaded into 20. So, load op code is this source is R 9, 901001 destination is R 20 this is 20 and in the 16 bit offset we write 84, 84 is in decimal if you converted to you convert it to binary this will be the binary equivalent for 84 this you can check.

So, again if you convert this into hexadecimal the hexadecimal form becomes 21340054 this instruction is encoded like this, let us take another example, branch equal to 0, this I said it checks there is only one register here 2 registers are not required only 1. So, if this R 25 is 0 then you jump to level otherwise do not jump. So, here BEQZ the opcode is this 001110 there is only one source register R 25 this is 25, but destination register in this instruction is unused we leave it as all 0s and then the 16 bit is offsets. So, here we writing it y yy this can be anything in fact.

So, the place where you want to jump will be specifying some offset here. So, what will happen is that this offset value will be added to the contents of the program counter and

that way the address of the next instruction will be calculated this is how branch will take place right and again this number if you convert to hexadecimal it will be s 3 b20 the last y s I write it y y y y this is 3 b 2 and 0.

(Refer Slide Time: 21:24)

**(c) J-type Instruction Encoding**

- Contains a 26-bit jump address field.
  - Extended to 28 bits by padding two 0's on the right.

Instruction	opcode
J	010000

Not Implemented

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Fine and the J type instruction that I had said that we are not implementing this, but actually J type J type instruction looks like this it consist of an op code and a 26 bit immediate data, this is only for the jump instructions.

So, here we have also assigned a opcode, but in the implementation we will not be using this, this is not implemented.

(Refer Slide Time: 21:51)

**A Quick View**

R-type: 31 26 25 21 20 16 15 11 10 6 5 0  
opcode rs rt rd shamt funct

I-type: 31 26 25 21 20 16 15 0  
opcode rs rt Immediate Data

J-type: 31 26 25 0  
opcode Immediate Data

- Some instructions require two register operands *rs* & *rt* as input, while some require only *rs*.
- Gets known only after instruction is decoded.
- While decoding is going on, we can prefetch the registers in parallel.
  - May or may not be required later.

• Similarly, the 16-bit and 26-bit immediate data are retrieved and sign-extended to 32-bits in case they are required later.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, a quick view of the 3 instruction encoding types. So, one thing you see you forget J type because you are not implementing this we are not looking at J type, but among the R type and I type 1 thing is very clear you see the first 6 bits are op codes the next 6 bit is always the source the next 6 bit is sometimes a source sometimes the target destination and for R type the next 6 bits is the destination.

So, what you saying is that we really do not know what type of instruction this is unless we decode the instruction, but the strategy that will be following is as follows to speed up things we will be assuming that there are 2 source registers. We will be taking the 2 source register numbers there will be a register bank we will be prefetching 2 registers for those number register numbers. We do not know whether they are actually required or not this will come to know later only after decoding, but in case they are required the registers can be fetched and they will already be available with us this way time is saved right.

So, now what you are saying is that I have just summarize it here, some instructions required 2 registers operands *r s* and *r t* like add subtract while some required only *r s* like add immediate, subtract immediate. But only after decoding is completed; that means, when you decode the opcode you will come to know exactly what kind of instruction is this. So, what I have just now said is that is a while this process of decoding is going on you pre fetch the registers assuming that these are it is an r type

instruction assuming that you need both the registers. So, register numbers already there you access the register bank and pre fetch them, but if later on find that it is an I type instruction then the second register that you have fetched you will simply ignore it only the first one will be used this may or may not be used it to a later on.

Similarly, for the immediate data for I type there can be 16 bit well J type you have not implemented for J type it will be 26 bits. So, we do something called sign extension we will talk about it later we convert it in to a 32 bit number and get ready. So, that later on when you need it already the values available in 32 bits.

(Refer Slide Time: 24:30)

**Addressing Modes in MIPS32**

- Register addressing      *ADD R1,R2,R3*
- Immediate addressing    *ADDI R1,R2,200*
- Base addressing          *LW RS,150(R7)*
  - Content of a register is added to a “base” value to get the operand address.
- PC relative addressing    *BEQZ R3,Label*
  - 16-bit offset is added to PC to get the target address.
- Pseudo-direct addressing   *J Label*
  - 26-bit offset is added to PC to get the target address.

IIT KHARAGPUR      NPTEL ONLINE CERTIFICATION COURSES      Hardware Modeling Using Verilog

So, the addressing modes that are available MIPS, MIPS 32 this is very quickly summarized here. Register addressing an example is this where all the operands are stored in registers R 1, R 2 and R 3 the destination in R 1 that immediate addressing one of the operand is an immediate data 200, base addressing there is a register there is a number you are adding this number to the register to get the effective address or the operand address this base addressing p c relative addressing branch.

So, whatever offset is specified here that is added to the program counter to get the, to get the actual address of branch and jump of course, you are not consider it is very similar this twenty 6 offset will be added to p c. So, with this we come to the end of this lecture where we gave you an over view of the processor that you are trying to design just one thing you remember some of you may or may not be having proper background

in computer architecture and organization for those of you have already done a course on computer architecture or organization you will be understanding much clearly what I am talking about. I am talking about a processor I am talking about a instructions decoding steps of execution and so on.

But if you are a designer wanting or trying to learn the language verilog you just stay with me you need not understand all the things that I am talking about, but ultimately you try to understand what is the problem that you are trying to model and how we are modeling in verilog you are implementing it in pipeline and what is the process involved. In the next lecture we shall be continuing from here we shall be talking in more detail about the steps of instruction execution and how we can design the, you can say controller you can say how the instruction execution is done inside ok.

Thank you.