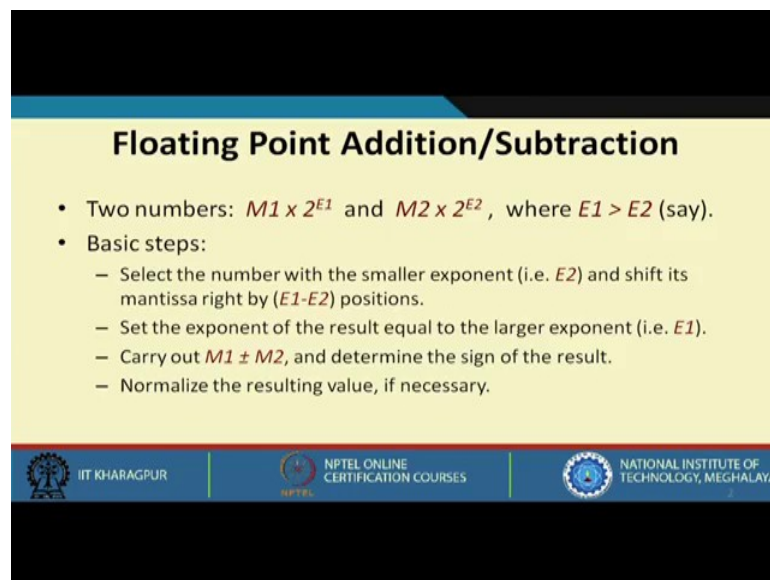


**Computer Architecture and Organization**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 39**  
**Floating - Point Arithmetic**




In this lecture we shall be starting or discussion on Floating - Point Arithmetic. We have seen earlier how a floating point number can be represented. Now we shall see how with this representation, we can carry out addition, subtraction, multiplication, division and for doing that what kind of hardware is required.

(Refer Slide Time: 00:44)



**Floating Point Addition/Subtraction**

- Two numbers:  $M1 \times 2^{E1}$  and  $M2 \times 2^{E2}$ , where  $E1 > E2$  (say).
- Basic steps:
  - Select the number with the smaller exponent (i.e.  $E2$ ) and shift its mantissa right by  $(E1-E2)$  positions.
  - Set the exponent of the result equal to the larger exponent (i.e.  $E1$ ).
  - Carry out  $M1 \pm M2$ , and determine the sign of the result.
  - Normalize the resulting value, if necessary.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES |  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

We start with floating point addition and subtraction. We consider them together because the process is very similar, and the same hardware can be used for performing both addition and subtraction. Let us assume that we have two numbers; for the time being we are not showing this sign, sign is also there.

The first numbers is  $M1 \times 2$  to the power  $E1$ , and the other number is  $M2 \times 2$  the power  $E2$ , and we assume that the exponent value of the first number is larger than that of the second,  $E1 > E2$ . For addition or subtraction the basic steps will be as follows.

Here I have assumed that the second number has the smaller exponent. Select the number of the smaller exponent, here it is  $E2$  and shift its mantissa right by  $E1 - E2$  positions,

such that this E2 becomes equal to E1. That means, we are aligning the mantissa such that the exponents are becoming equal. Suppose this was 2 to the power 10 and this was 2 to the power 5; you shift M2 by 5 positions such that this also becomes 2 to the power 10.

So, both the exponents will become equal. So, after this we will be adding or subtracting M1 and M2 straight away, because M2 is already shifted right. We have already aligned the mantissa.

After shifting we simply carry out addition or subtraction depending on which operation we are trying to carry out. Also we can see the result of addition and subtraction, and you can calculate the sign of the result and after addition or subtraction we may need to normalize the result because the result may not start with a 1. For normalization we have to shift it left, so that the first bit of the mantissa always becomes 1, so that we can represent the mantissa in a suitable way excluding that implied 1.

(Refer Slide Time: 03:14)

**Addition Example**

- Suppose we want to add  $F1 = 270.75$  and  $F2 = 2.375$   
 $F1 = (270.75)_{10} = (100001110.11)_2 = 1.0000111011 \times 2^8$   
 $F2 = (2.375)_{10} = (10.011)_2 = 1.0011 \times 2^1$
- Shift the mantissa of F2 right by  $8 - 1 = 7$  positions, and add:  

$$\begin{array}{r} 1\ 0011\ 0000\ 0000\ 0000\ 0000\ 000\ 000 \\ \hline 1000\ 1000\ 1001\ 0000\ 0000\ 0000\ 0000\ 000 \end{array}$$
- Result:  $1.00010001001 \times 2^8$  ← Residue

Let us take an example. Suppose we are adding two numbers F1 and F2, values are 270.75 and 2.375. 270.75 if you express in binary it becomes this and in a normalized binary floating point representation, you can write it as this. So, it becomes this multiplied the 2 to the power 8. Similarly F2 can be represented in binary as this. So, if you shift this point by one position it becomes this. So, it becomes 2 the power 1.

Now for the two numbers, one of them is 2 to the power 8, other is 2 the power 1; second one is smaller. So, what I do we shift the mantissa of the number with the smaller exponent; that means, F2 by  $8 - 1 = 7$  positions. So, it becomes like this. Then we add the two manissas.


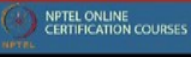

So, you take the first 24 bits of the sum, this will be your actual result, and these remaining bits will be the residue. So, your actual result will be this where it this number is already normalized it starts with 1.

(Refer Slide Time: 05:32)

**Subtraction Example**

- Suppose we want to subtract  $F2 = 224$  from  $F1 = 270.75$   
 $F1 = (270.75)_{10} = (100001110.11)_2 = 1.0000111011 \times 2^8$   
 $F2 = (224)_{10} = (11100000)_2 = 1.111 \times 2^7$
- Shift the mantissa of F2 right by  $8 - 7 = 1$  position, and subtract:  

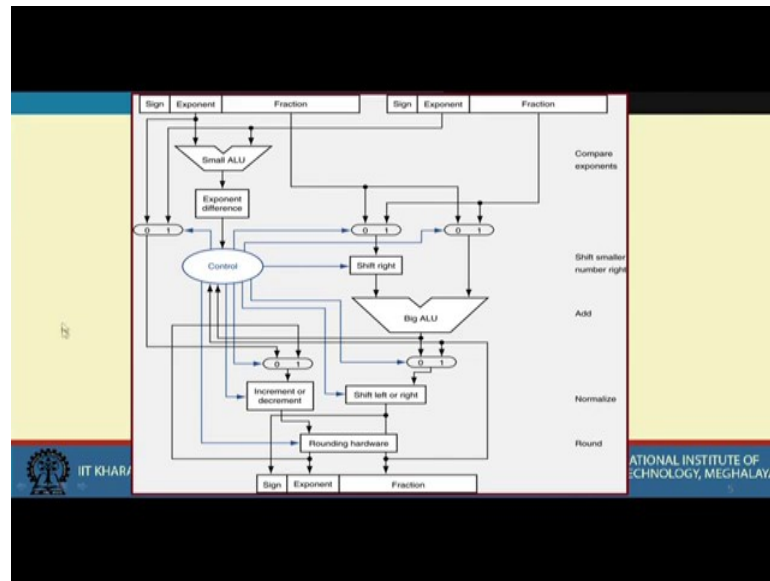
$$\begin{array}{r} 1000\ 0111\ 0110\ 0000\ 0000\ 0000 \\ \underline{111\ 0000\ 0000\ 0000\ 0000\ 0000} \\ 0001\ 0111\ 0110\ 0000\ 0000\ 0000 \end{array}$$
- For normalization, shift mantissa left 3 positions, and decrement E by 3.
- Result:  $1.01110110 \times 2^5$

 IIT KHARAGPUR
  NPTEL ONLINE CERTIFICATION COURSES
  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Similarly, let us take a number 224 and 270.75. We can represent the numbers as follows. Similarly we will have to shift the mantissa of F2 by one position. Then you subtract the mantissas. You need a normalization step here and the result will be this. This adjustment is required at the end.

This is the process of addition or subtraction of floating point numbers; what are the steps that are involved. From this you can directly generate the circuit.

(Refer Slide Time: 07:16)



You look at this schematic. These are the two numbers, this is the first number, this is second number; sign exponent fraction and sign exponent fraction. This is a small 8 bit ALU; this is actually subtracting the two exponents and finding out which one is smaller; also it is calculating the exponent difference because later on we will have to shift the mantissa by that amount.

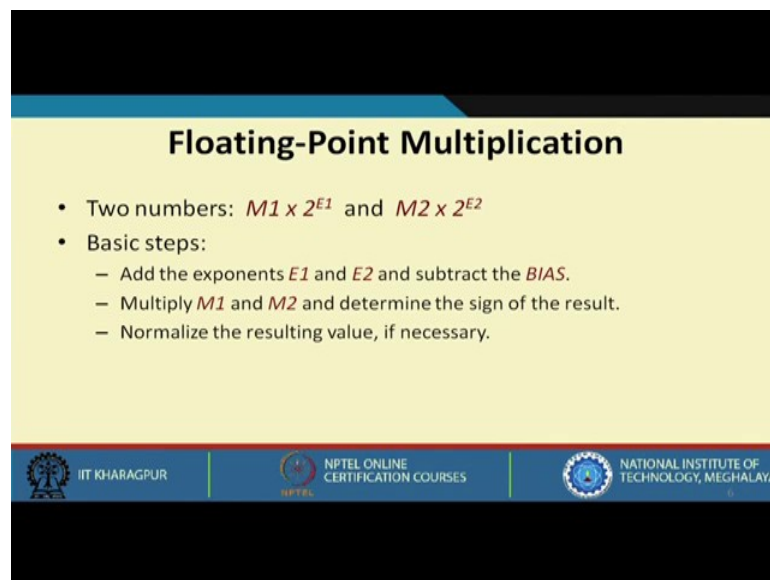
So, there is a control circuit here. This information is going to the control depending on which exponent is smaller. There is another ALU where this smaller of the exponents will be coming to the first input; there is a multiplexer here. Either this fraction or this fraction will be selected; control will be generating the signal.

That will be shifted right by so many positions (exponent difference). After shifting you do the actual addition or subtraction; these two multiplexers are selected by the control unit. After you do the addition or subtraction again similarly you will have to look at the process of normalization. Like in the previous example you have seen that the result was not normalized; you had to shift it left by 3 positions. You do the same thing, you may have to shift it and you see by how many positions you have to shift.

Accordingly you make the adjustment and for IEEE format you need an additional step for rounding. So, if you have enabled rounding, after everything is done the mantissa will be rounded depending on those two bits r and s and you get the final result.

The concept of floating point addition subtraction is very simple because already you have seen the steps involved. You have to compare the exponents, you have to align the mantissa by shifting one of them, then you have to do addition or subtraction, then we have to carry out and normalization of the result, and finally, if required, rounding.

(Refer Slide Time: 09:50)



**Floating-Point Multiplication**

- Two numbers:  $M1 \times 2^{E1}$  and  $M2 \times 2^{E2}$
- Basic steps:
  - Add the exponents  $E1$  and  $E2$  and subtract the  $BIAS$ .
  - Multiply  $M1$  and  $M2$  and determine the sign of the result.
  - Normalize the resulting value, if necessary.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Next let us come to multiplication; floating point multiplication and division are relatively easier in concept, because it involves less number of steps. Suppose I am trying to multiply two numbers like this  $M1 \times 2$  to the power  $E1$ ,  $M2 \times 2$  to the power  $E2$ .

(Refer Slide Time: 10:17)

© CET  
I.T. KGP

$$\begin{array}{r} 1.2 \times 10^5 \\ \times 1.2 \times 10^6 \\ \hline 1.44 \times 10^{11} \end{array}$$
$$\begin{array}{r} E1 (+127) \\ + \\ E2 (+127) \\ \hline \\ \hline -127 \\ \hline = \\ E \end{array}$$

Suppose I have a number  $1.2 \times 10$  to the power 5, and there is another number  $1.2 \times 10$  to the power 6.

When you multiply the two numbers, there is no question of doing any adjustment of mantissa; you simply multiply the two mantissa. So,  $1.2 \times 1.2$  will become 1.44. And you simply add the two exponents; it will be 11, and we get the result. But in floating point number one thing you have to remember that is  $E1$  and  $E2$  are both biased; that means, for single precision number already we have done +127. So, for multiplication if you are adding these two numbers whatever you get, do not forget to subtract a 127 because you have counted 127 twice.





So, you have to subtract it once. What I have mentioned is written here. You add the two exponents and subtract the bias that is 127; multiply  $M1$  and  $M2$  and also determine the sign of the result, and only at the end you need to normalize result if it is required.

So, it is much simpler than addition.

(Refer Slide Time: 12:00)

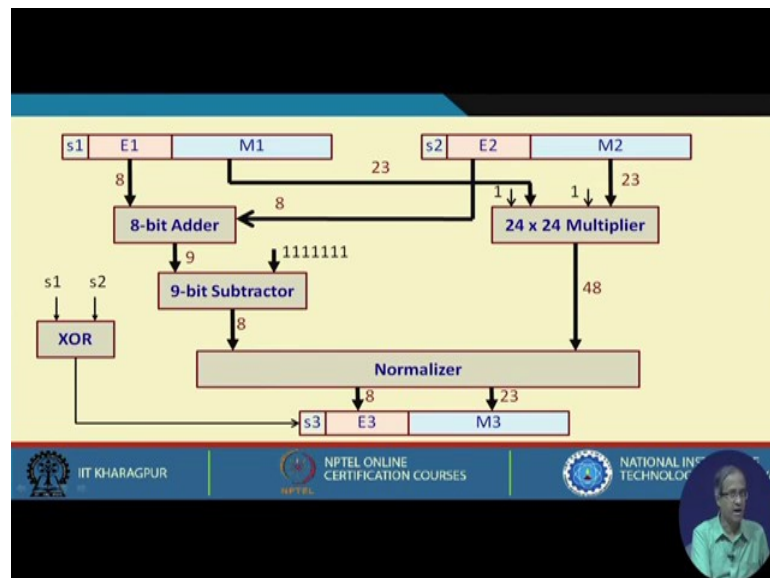
### Multiplication Example

- Suppose we want to multiply  $F1 = 270.75$  and  $F2 = -2.375$   
 $F1 = (270.75)_{10} = (1000011110.11)_2 = 1.00001111011 \times 2^8$   
 $F2 = (-2.375)_{10} = (-10.011)_2 = -1.0011 \times 2^1$
- Add the exponents:  $8 + 1 = 9$
- Multiply the mantissas:  $1.01000001100001$
- Result:  $1.01000001100001 \times 2^9$

IIT KHARAGPURNPTEL ONLINE CERTIFICATION COURSESNATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR

So, let us take an example. Suppose we want to multiply  $F1 = 270.75$  and  $F2 = -2.375$ ; both are normalized. The steps of multiplication are shown.

(Refer Slide Time: 13:08)



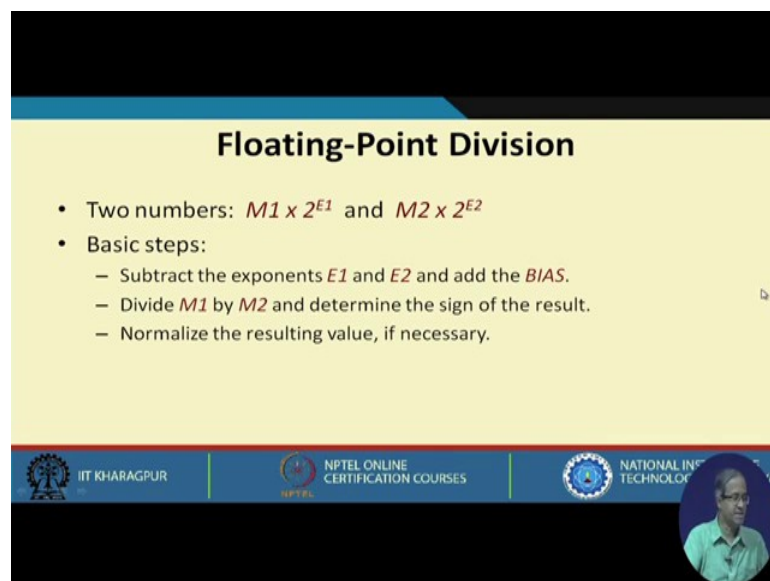
The multiplier hardware will be conceptually simpler. You have the first number here, and the second number here. There is an adder which will be adding  $E1$  and  $E2$ ; you will have to add the two exponents and these are 8 bit numbers, temporarily you generate an 9-bit some because you will have to subtract the bias 127.

After subtracting the bias you get the final 8-bit result which is the final exponent. On the other side you have a 24 x 24 multiplier, because its mantissas are all 23 bit numbers, but there is an implied 1 bit which is hidden. So, the multiplier will also have to be fed with that extra 1 bit, and then the two numbers multiplied.

So, we get 48 bit product; you look at the product whether its starts with 1 or not; if not you will have to do a normalization, shift it and also adjust the exponent accordingly. After that the final exponent will go into the result. The final mantissa 23 bits will go to here, and the sign will be the exclusive or of  $s_1$  and  $s_2$ .

So, if one of the numbers is positive and other is negative then only the result will be negative; otherwise the result will be positive. So, this is the multiplication hardware; floating point division is quite similar.

(Refer Slide Time: 15:02)



The slide is titled "Floating-Point Division" and contains the following content:

- Two numbers:  $M1 \times 2^{E1}$  and  $M2 \times 2^{E2}$
- Basic steps:
  - Subtract the exponents  $E1$  and  $E2$  and add the  $BIAS$ .
  - Divide  $M1$  by  $M2$  and determine the sign of the result.
  - Normalize the resulting value, if necessary.

The slide footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR, along with a small circular portrait of a man in the bottom right corner.





For division let us say we have two numbers  $M1 \times 2$  to the power  $E1$ , and  $M2 \times 2$  to the power  $E2$ . For division, instead of adding the exponents we are subtracting the exponents, but you see in both the exponents we have added a bias. So, when you subtract them, the bias also gets canceled out. So, you will have to additionally add that bias after the subtraction, then you divide  $M1$  and  $M2$ , and finally, normalize.



(Refer Slide Time: 15:40)

### Division Example

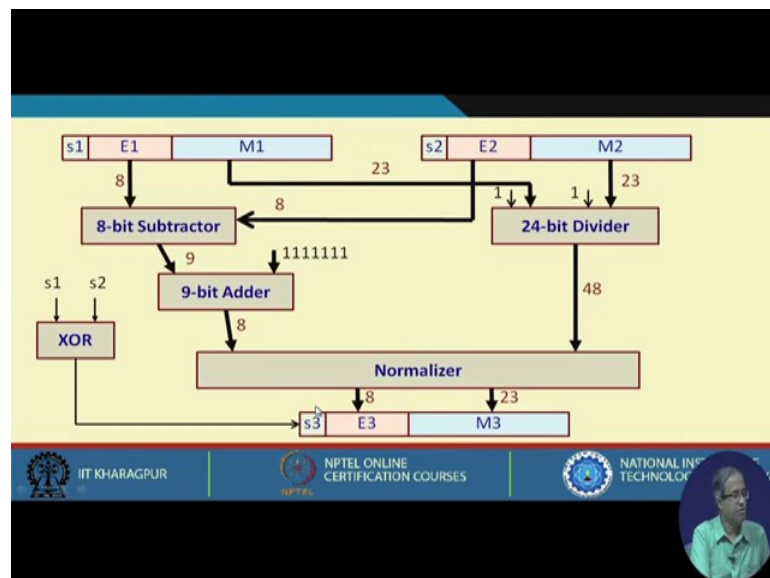
- Suppose we want to divide  $F1 = 270.75$  by  $F2 = -2.375$   
 $F1 = (270.75)_{10} = (100001110.11)_2 = 1.0000111011 \times 2^8$   
 $F2 = (-2.375)_{10} = (-10.011)_2 = -1.0011 \times 2^1$
- Subtract the exponents:  $8 - 1 = 7$
- Divide the mantissas:  $0.1110010$
- Result:  $0.1110010 \times 2^7$
- After normalization:  $1.110010 \times 2^6$

IIT KHARAGPURNPTEL ONLINE CERTIFICATION COURSESNATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR

So, let us take an example; suppose I want to divide this number by this number.

The steps of division are all shown.

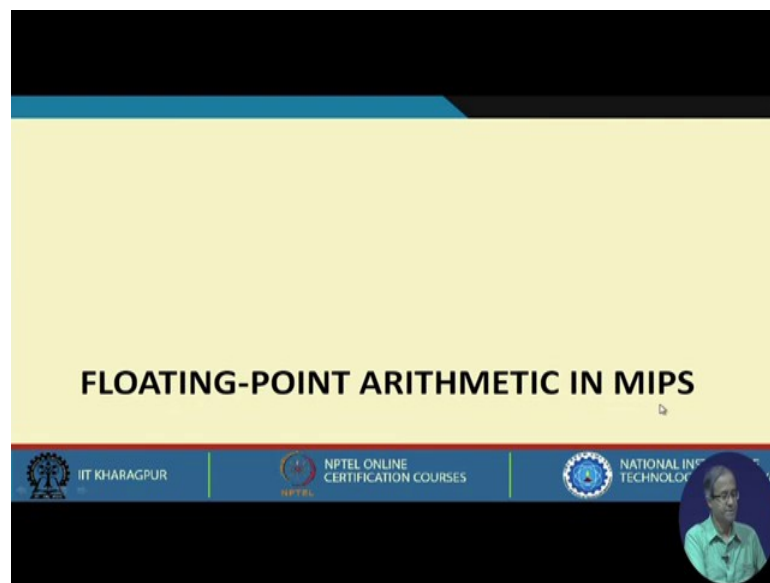
(Refer Slide Time: 16:41)



The hardware will be very similar to multiplication. The only difference is that here instead of addition you do a subtraction; you subtract the two exponents, you get a 9 bit result, and you add the bias to get the final value of the exponent.

Similarly, you have a 24-bit divider. These 23 bit mantissas are coming and the implied 1 bit are here. The divider will be generating the bits of the result. Again you may have to normalize as the previous example showed, and after normalization you take the corrected value of exponent. The first 23 bits of the result will go to mantissa, and just like multiplication for division also you take the XOR of the s1 and s2 that will give you this sign of the result.

(Refer Slide Time: 17:42)



Now, for MIPS32 instructions set architecture there are also floating point arithmetic, which we have not talked about earlier. Very briefly let us look at what are the features that are available in MIPS. The first thing is that in the MIPS architecture we have some floating point registers called F0 up to F31; just like for integer registers we had R0 to R31.

(Refer Slide Time: 18:25)

The MIPS32 architecture defines the following floating-point registers (FPRs).

- 32 32-bit floating-point registers F0 to F31, each of which is capable of storing a single-precision floating-point number.
- Double-precision floating-point numbers can be stored in even-odd pairs of FPRs (e.g., (F0,F1), (F10,F11), etc.).

In addition, there are five special-purpose FPU control registers.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

These registers are all capable of storing a single precision number because they are 32-bit numbers. Now additionally you can also operate on double precision numbers which are 64 bits in size, but how you do it? You will be actually using register pairs (F0, F1), (F2, F3), etc.

(Refer Slide Time: 18:49)

Handwritten diagram illustrating double precision register pairs:

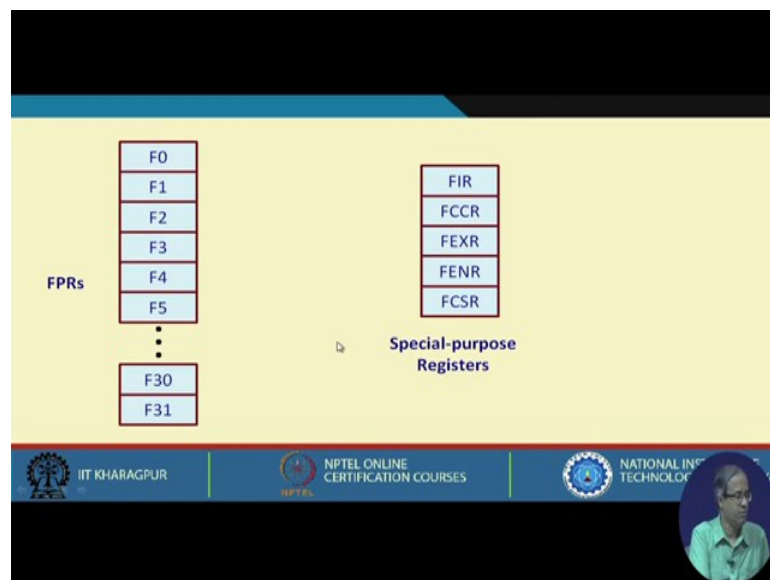
- F0 and F1 are grouped together with a bracket and labeled "64-bit ⇒ F0".
- F2 and F3 are grouped together with a bracket and labeled "64-bit ⇒ F2".

IIT, KGP

If you are using F0 and F1, together this will be a 64-bit register, if you look at F2 and F3 together this will also be a 64-bit register. But when you use it in a program you just refer to as F0 double precision, F2 double precision, etc.

So, when you are into double precision F0 means (F0, F1), F2 means (F2, F3). It is implied that even and odd register pairs will be taken. In addition to this there are some special purpose registers also, but here we are not going into details of them because these are not really required.

(Refer Slide Time: 19:43)



(Refer Slide Time: 19:54)

### Typical Floating Point Instructions in MIPS

- Load and Store instructions
  - Load Word from memory
  - Load Double-word from memory
  - Store Word to memory
  - Store Double-word to memory
- Data Movement instructions
  - Move data between integer registers and floating-point registers
  - Move data between integer registers and floating-point control registers

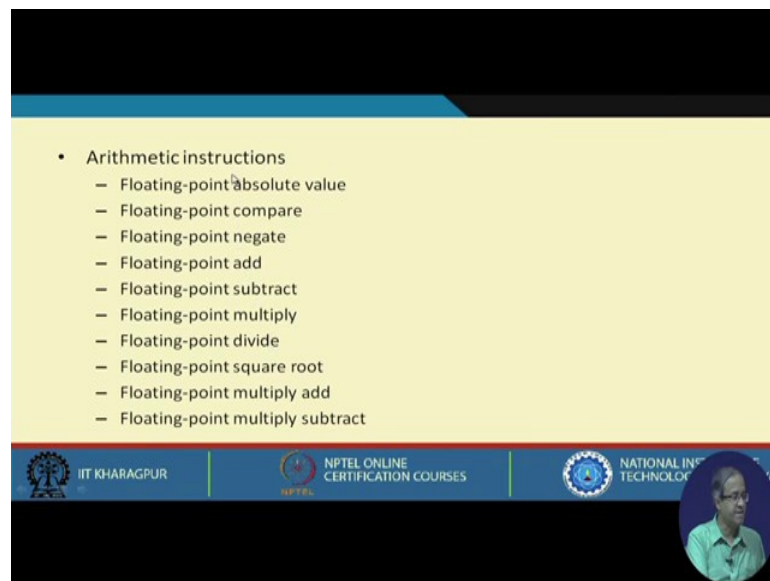
The slide footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY.

And the instructions that relate to the floating point registers are as follows. There are Load and Store instructions; we can load a single precision floating point number from

memory or we can also do a load double. If you have a load double let us say to F0, it will be loaded into F0 and F1.

Similarly, there is a store word, store double word data movement instruction. You can move data from an integer register to a floating point register or vice versa, or into some control registers. Also the arithmetic instructions add, subtract, multiply, divide.

(Refer Slide Time: 20:34)



The slide displays a list of arithmetic instructions under the heading "Arithmetic instructions". The instructions are:

- Floating-point absolute value
- Floating-point compare
- Floating-point negate
- Floating-point add
- Floating-point subtract
- Floating-point multiply
- Floating-point divide
- Floating-point square root
- Floating-point multiply add
- Floating-point multiply subtract

The slide footer includes the logos and names of IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY.

There are some other instructions also. You can calculate the absolute value of a number, you can compare two numbers, you can negate; that means, given  $x$  you find out  $-x$ , square root of a number. Multiply add, multiply subtract means you multiply two numbers and add a third number  $a \times b + c$ . This multiply accumulate kind of instructions or multiply subtract are quite useful in various DSP applications, where this kind of computations appear quite frequently.

(Refer Slide Time: 21:27)

• Rounding instructions:

- Floating-point truncate
- Floating-point ceiling
- Floating-point floor
- Floating-point round

• Format conversions:

- Single-precision to double-precision
- Double-precision to single-precision

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

There are some instructions to round the numbers. Here you can specify what kind of rounding you are doing: truncation, ceiling, floor, or just round. I mentioned these 4 types are there, and also you can sometimes convert from single to double precision or vice versa; those instructions are also there.

(Refer Slide Time: 21:51)

**Example: Add a scalar  $s$  to a vector  $A$**

```
for (i=1000; i>0; i--)  
  A[i]= A[i] + s;
```

**Loop:**

L.D	F0, 0(R1)
ADD.D	F4, F0, F2
S.D	F4, 0(R1)
ADDI	R1, R1, -8
BNE	R1, R2, Loop

R1: initially points to A[1000]  
(F2,F3): contains the scalar  $s$   
R2: initialized such that 8(R2) is the address of A[1]

We assume double precision (64 bits):

- Numbers stored in (F0,F1), (F2,F3), and (F4,F5).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

I just take a simple example of adding a scalar to a vector. Like in C let us say I have a loop like this. The MIPS32 assembly language code is also shown.

This is a very simple example that shows how we can use the floating point registers and instructions. It is quite simple, just an extension of the integer instructions.

So, with this we come to the end of this lecture. If you recall in this lecture we had looked at how we can carry out floating point operations, in particular the addition, subtraction, multiplication and division, and also we had discussed the corresponding hardware requirements.

Thank you.