

Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 9
Floating Point Number - 2

(Refer Slide Time: 00:16)

Best way to represent real numbers



- Scientific notations 0.5×10^6
 – Normalized 5.0×10^5
- Large as well as small numbers can be represented
- Exponential in binary
 $1000001 = 1.000001 \times 2^5$
- Representing – decimal point, exponent in binary?



Digital Logic Design: Introduction

67

How to do that? Let us see.

(Refer Slide Time: 00:22)

Exponential binary numbers



- Significand (could be +/-)
- Fraction or mantissa
- Exponent (could be +/-)
- Typical binary representation
 - 1 bit for sign of significand
 - Few bits for exponent
 - Rest bits for mantissa

-1.000000	x	2^{-26}
1.101001	x	2^8
-1.100010	x	2^{10}
1.000001	x	2^{-18}

How to distribute bits in exponent and mantissa?



Digital Logic Design: Introduction

68

So, if I see this particular number. So, I have taken couple of examples. So where my, some of these numbers are positive, some of these numbers are large, some of these numbers are small.

So, if I have these numbers now, what I see, I see a couple of interesting things. One thing that if I can specify, I can say that out of my n number of bits, let us say out of my 16 bits or out of my 32 bit, I assigned a couple of bits for this, this part and some bits for this part, that would be good and maybe some bits for plus and minus. So, before doing that, let us try to understand how to read these numbers.

So, one thing we call this particular thing we call Significant. Significant would be one point and then these numbers. So, this is called Significant and in this significant because of our normal definition. In the definition of normal scientific notations, we said this a number before decimal point has to be nonzero. So, in this case, in case of decimal exponential numbers, it could be 123456789 but because it is binary, we have only one choice it can be only one.

And the way we define significant is significant is actually this digit dot rest of the digits, they are called significance. Now, the significant could be plus could be positive number could be a negative number. So, this is one particular thing. And now, because in a normal notation, this before the decimal point it is going to be 1. So, I can also define one more part which is a fractional part, which is also called Mantissa.

So, this fractional part could have 0, it could have the combination of 0s and 1s. So, this is called Mantissa. And the third thing is Exponent. In exponent also, I can assume that because it is the binary number, it is always going to be going to be the power of 2. Now, there is another number 2s power what? So, these 2's power is the exponent value again, this exponent value could be positive could be negative.

Essentially, if I combine all of these things, then I can out of let us say my 16 bit or 32 bits, I can represent I can choose one bit for the sign of significant whether significant is positive or negative. And then few bits I can choose I can fix for, for Exponent. And rest of the bits I can choose for the Mantissa. So, I need not to represent decimal point but again decimal point would be implicit here.

Similarly, this 2 or this multiplication sign is also implicit in nature, what we have done is that out of this whole these notations this, this explanation notations are normal, normal notations for binary numbers, normal exponential exponent notations for binary numbers we see that this is a 3

piece of information if we are able to store that means we can regenerate or calculate what was that Floating Point Number.

1 I need to know whether that number is a positive number or a negative number. The second thing what is the value of mantissa see, we are not talking about significant because insignificant this one bit, because in a normal form, this is always going to be one so we can again this whole this we can take it as the implicit information, so we are taking the mantissa part and we only know the, the exponential values.

So, if we know the exponent values, we know the mantissa and we know the sign bit then we can reconstruct this floating-point number order the real number. So, the question now which arises here that how many bits should we assign for exponent how many bits should we assign for Mantissa. Sign would definitely require only one bit we can say 0 means positive 1 means negative.

So, one bit is sufficient to represent sign, but for exponent and mantissa there is always a question that how many bits would be required. And again, for the second for information, I can tell you that when these floating-point numbers were initially invented, or this particular way was converse that this is how we represent floating point numbers. Every manufacturer, people who were designing their calculators or their computers, they were choosing their own number of exponent bits and magnetic tape.

So, it was not standardized. And the problem was faced when 2 different vendors were communicating the information. So, let us say vendor one was writing all the information in magnetic tape the other vendor want to read it now how will they know that what was their format so there comes the standardization. Now, in case of Computer Science and Electrical Engineering,

picked bias method for representing exponent. Why they have used bias method for representing exponent?

Because here we are not supposed to do multiplication mostly it is addition and subtraction. So that is why a bias method is sufficient and good. And bias of 127 and we, right now, for a couple of sites, we just assumed that 0 and 255 are not used. So, the range of numbers which can be represented in exponent is from 1 to 254, they are all valid other than 1 and 254 nothing is varied, they are not valid.

And they are used for some special purpose, which we will see later in the lecture or maybe in the next lecture. The rest of the bits 23 bits for mantissa. So, you will also see these numbers are essentially they are using sign magnitude form because we have one bit which is used only to represent sign. So here, 2's complement method of representing negative numbers are using integers, but rest of the 2 methods like sign magnitude, because we are representing one bit as a sign, and also bias method of representing negative numbers is used in this floating point notation.


So, if we have these 3 pieces of information, I am calling it s, e and m, then I can construct my final number as $minus\ 1\ into\ s\ whole\ minus\ 1\ is\ to\ power\ s$, actually it should be $minus\ 1\ power\ s$ multiplied by $1\ plus\ 0.m\ into\ 2\ is\ to\ power\ e$. So, if I had this, this 32-bit number from 0 bit to 22 bit essentially total of 23 bits would be useful mantissa and the bit number 23, 2 bit number 30 would be used for exponent and 31st bit would be used for sign.

Why this particular way of representing? So, it has certain advantage that like our integers, my MSB here is telling me whether the number is positive or negative. So, I can use the same method to find out whether the number is positive or negative means last bit I can check and there is no particular way why mantissa and exponents are given these particular set of bits, but because exponent is given MSB and mantissa is given lower side or basically lower significant bits.

So that yes, there is no particular reason for that this thing, but this is how it is organized and because it is given in a standard so we can always follow that this particular way what this particular bit would represent mantissa or exponent sector. So, now this is probably not clear


unless we take some examples that how a number would be represented and how mantissa or how exponent would be calculated.

(Refer Slide Time: 12:25)



Floating point example 1

- Decimal number: 24.25
= 16+8+0.25
- Binary number: 11000.01
- Normal scientific form : 1.100001×2^4
- Sign: 0
- Mantinssa: 100 0010 0000 0000 0000 0000
- Exponent: $(127 + 4)_{10} = (131)_{10} = (100\ 0001\ 1)_2$
- Floating point number
0100 0001 1100 0010 0000 0000 0000 0000
0x41C20000



Digital Logic Design Introduction 70

So, let us see with some examples that how do we compute for how do we see a floating-point number. So, to take this example to, to have this understanding, let us take our decimal number as 24.25. So, this 24.25 I want to represent in binary so, I said it is these are all power of 2's so 16 plus 8 is 24 and then 1 by 4 is 0.25, 1 by 4 is 2 to the power minus 2. So, that means I can represent this number in a binary way I can write it like 11000 and then there is a decimal point then 01.

Now, the then I will have to write this binary number in a normal scientific form in a normal scientific form, there would be only one significant digit. So, significant digit means non-zero digit before this decimal point and after that, rest of the digits will be there. So, if I want to write in this form, then I need to calculate how many what is the 2 is to the power what is the exponent value, so 1 2 3 4.

So, that means I can write this whole number as one point I can push this decimal point at this place 1.10001 and 2 is power 4. So, once we have constructed this normal scientific form, after constructing this normal scientific form, then I can segregate or I can find out the 3 values which are required, I require a sign bit mantissa and exponent. So, because this number is a positive number, then we write sign with a 0. Mantissa, mantissa means this significant digit will remove

and after removing significant digit, whatever is left is a fractional part or mantissa part that we write here. So, we are writing all the 23 bits in this fashion. Now, I will give you here one trick that because the number of bits are 23 and finally we have to find out a number which is 32 in, in nature, so it would be better or always better to write these numbers in a hexadecimal form, so we will have only 8, 8 digits.

So, because it is 23 bits, what I am doing is I am grouping them in the, in the groups of first 3, first 3 bits, and after that the group of 4 4 4 bits. So, I have 5 groups of 4 bits. And so essentially, I have 5 nibbles here and one, one group of 3 bits. So, these are the mantissa. Now exponent, we said there is a base of 127. So that means whatever exponent we are getting after this normal form, we have to add 127, we have to add this bias.

So, after adding this bias, it will become 127 plus 4, that means 131. And now this if I want to represent 131, in 8 bits, then it will become 1000001. So again, you see I have made the grouping like this first 3 bits, then 4 bits, then one bit why I am doing so you will see later. So now I would like to construct my floating-point number. And what I will do is I will put my sign bit at 31st place and then exponent bits and then mantissa bits.

So, because 31st bit is a sign and then I would require 3 another bits from the exponent to complete this nibble. That is why the group of 3, 4 and 1 was constructed then a next nibble from the exponent can be written here. Then the next bit last bit of the exponent would be written here and then I can write mantissa. So, first 3 bits of mantissa and then rest 5 nibbles of the mantissa.

Now, all these are now written in 8 different nibbles from these 8 different nibbles. Now I can construct my hexadecimal number which is a quick and which is one to one representation corresponding to binary numbers. So, it is 0 cross 4, 1, C, 2, 0, 0, 0, 0. So, this is how we can represent or we can convert any number into a floating-point number. So, let us take some more example.

(Refer Slide Time: 17:34)

Floating point example 2



- Decimal number: 0.625
= 0.5 + 0.125
- Binary number: 0.101
- Normal scientific form : 1.01×2^{-1}
- Sign: 0
- Mantinssa: 010 0000 0000 0000 0000 0000
- Exponent: $(127 - 1)_{10} = (126)_{10} = (011\ 1111\ 0)_2$
- Floating point number
0011 1111 0010 0000 0000 0000 0000 0000
0x3F200000



Digital Logic Design: Introduction

71

Let us say the number is 0.625. I am taking easier examples. I am leaving the hard examples for the tutorial session and for your exams. Yeah, this is a joke. So, let us see. So, we will try to see at least cover a couple of examples here we can see during the exam that how do we go. So, this 0.625 can also be written like a 0.5. That means 1 by 2, and 0.125. That is 1 by 8. In other words, if I write this in binary, then it would be a 100 point 101.

And making this particular number in a normal scientific form, it would require 1.01 because the first significant digit is 1, so that means it has to be pushed like this. And 1.01 into 2 is 4 minus 1. So, we have now all the information we know the number is positive, we know what is mantissa 01. And we also know what is exponent, now we can construct a floating-point number. So, in a floating point number sign would be 0 and mantissa is 010 and rest for all the nibbles it is going to be 0.

And exponent because we have a bias of 127. So, we will add minus 1 to 127. So that means it will become 126. And then we can convert this 126 into binary, that means 0111111 and 0. You can ask me, how do I do it so quickly? It is not only because I pre calculated but because 127 means all 7 1's it is 1 less than 128. That means 7 1's would mean 127 and now I am subtracting 1 so that means the last bit will become 0.


So, this is the trick to calculate a binary number quickly without going through the divide by 2 procedure. So now I have all the information I can club that information to create my floating-


point numbers. So, sign I will keep here at 0 and then these 3 bits of exponent and then these 4 bits of exponent then last bit of exponent and then my mantissa of 0 1 0 and then rest nibbles of 0s. So, you can construct this, this will become 0 cross 3 f and this is 2 and then 5 0s. So, the hexadecimal representation of this number is 0.625.

(Refer Slide Time: 20:27)

Floating point example 3

- Decimal number: -2.625
= 2 + 0.5 + 0.125
- Binary number: 10.101
- Normal scientific form : 1.0101×2^1
- Sign: 1
- Mantinssa: 010 1000 0000 0000 0000 0000
- Exponent: $(127 + 1)_{10} = (128)_{10} = (100\ 0000\ 0)_2$
- Floating point number
1100 0000 0010 1000 0000 0000 0000 0000
0xC0280000



72



Let us take some example of a negative number. The example is very similar to the previous one, so, I can go a little fast. So, because it is 2, 2 is already a power of 2, so I can write it as 10.101 then this number in a normal scientific way would be written as 1 point basically decimal has shifted towards a left one point it has shifted, so that means 2 to power 1 will become as the exponent factor.

Now, I did the normal scientific after creating my normal scientific form, I know the old the exponent mantissa is sign bit with. So yeah, here sign bit has to be a negative here. So, now sign bit because it was a negative, so I am putting sign as 1 and mantissa is these 4 bits, 0010 I will put in first place and then after that 1, and then the rest of the 0s, I will keep it keep the trailing 0s and rest of the bits are also 0.

For the exponent, 1 would be added to 127 because 127 was the base address or basically the offset so this offset or bias, so in the bias, we add 1, it becomes 128. 128 means 1 followed by 7 0s. Then we can again, combine these 3 vectors signing mantissa and exponent to create our

floating-point number in a hexadecimal way. Our one more example which is which looks quite simple, but it is important to touch that also.


(Refer Slide Time: 22:21)



Floating point example 4

- Decimal number: 1
- Binary number: 1
- Normal scientific form : 1.0×2^0
- Sign: 0
- Mantinssa: 000 0000 0000 0000 0000 0000
- Exponent: $(127 + 0)_{10} = (127)_{10} = (011\ 1111\ 1)_2$
- Floating point number
0011 1111 1000 0000 0000 0000 0000 0000
0x3F800000

Digital Logic Design: Introduction. 73



Let us say I want to represent 1. Now, you see there is no decimal, but because I would represent 1 in a floating-point notation, I have to write it like 1.0×2^0 . So here, my mantissa is 0, my exponent is also 0, my sign is also 0, how do I represent this number for mantissa, it is going to be all bits all bits will be 0 all 23 bits will be 0, sign would be 0 and exponent in 127 0 would be added.

That means it will be 127 only 0 and then 7 1's, then I can combine all of these and then the resultant floating-point number in hexadecimal is 0 cross 3F8 5 0s. So, this also gives us one food of thought that whenever we are converting any decimal number and so whenever you are writing C, C++ code or Java code, and you are simply writing you want to typecast an integer to float, you want to say that this integer should be converted into a float number, then all of these processing steps would happen.

And let us say number was written as 1 there. But in, in a floating-point notation, it would not be written as 1 it is a it is a more sophisticated form. It also requires a lot of steps to convert an integer to floating point number. Now one can again ask that why do I want to convert an integer into a floating point number? Again, my answer is very similar to the examples we are taken for large numbers.

So, let us say the number has some 20 digits, 20 of decimal digits. If number has some decimal 20 digits, so that means I need to convert those 20 decimal digits into binary digits and we should be probably more than 60 or 70. Now because I have to represent them, but again, the lower significant digits may not be useful for all practical purposes. So, if I make that as a floating-point number that can help me in easing of certain operations. So, whenever a large integer need to be represented then those integers are also represented using a floating point number. So, with this, I would like to close this lecture.

(Refer Slide Time: 25:29)

Summary



- Real number can be represented in binary form
 - Both in fixed point format as well as floating point format



And in summary, what we have learned in this lecture is that we learned that how a Real number can be represented in a in a in a binary form, and how decimal point would be represented how exponent would be represented. So, we finally observed that there are two ways of representing a real number either we put it in a fixed-point format or we put it in a Floating-Point format.

Fixed point format gives us certain advantage that it is easy to operate with we can use integer operations of addition, subtraction, multiplication, but it seemingly eliminate the amount of precision and range. On the other hand floating point notation is a complex operation in conversion itself. Where to identify what is mantissa we have to identify what is sign, we have to identify what is exponent and then add the bias in the exponent and then construct a Floating Point Number. So, this is for, this is all for today. Thank you very much, and Good Luck.