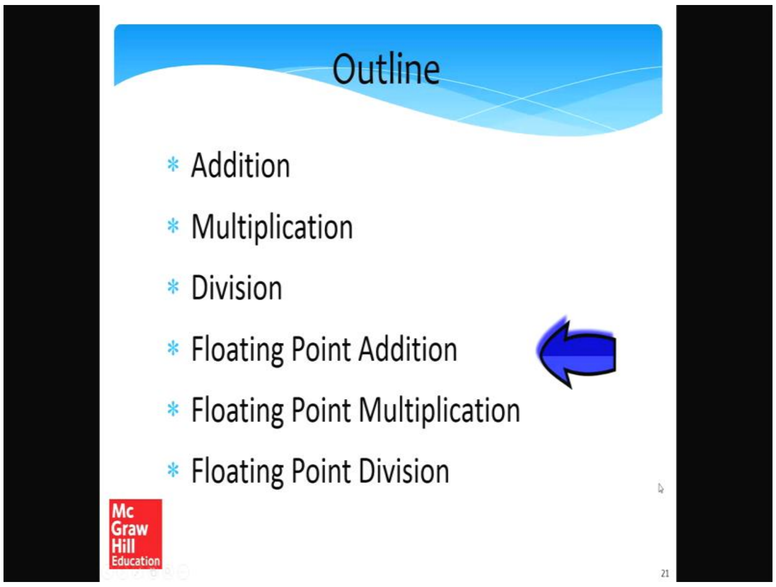
Computer Architecture

Prof. Smruti Ranjan Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 21
Computer Arithmetic Part-[vocalized-noise]

Welcome back. Let us take a look at floating point addition. So, we will take a look at all 3 floating point addition, multiplication and division.

(Refer Slide Time: 00:25)



Outline

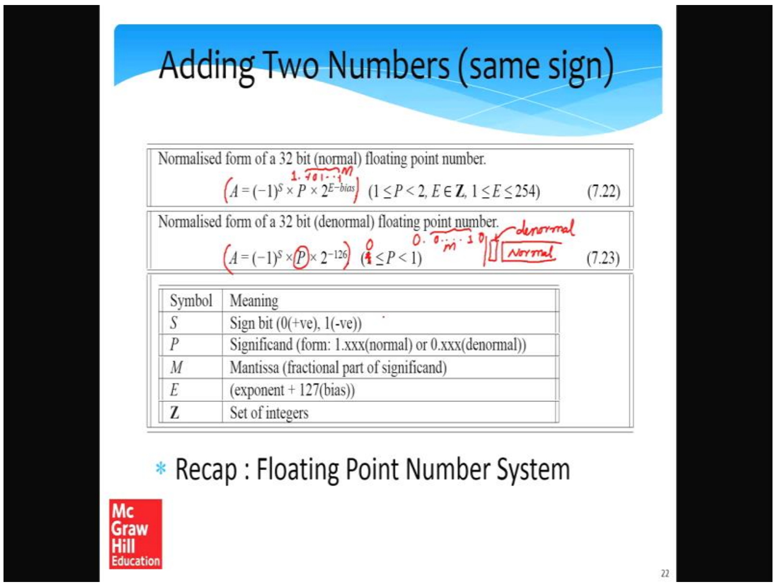
* Addition
* Multiplication
* Division
* Floating Point Addition
* Floating Point Multiplication
* Floating Point Division

Mc Graw Hill Education

21

Let us start with the addition. Additional multiplication is done almost similarly. Division is how a different is very interesting.

(Refer Slide Time: 00:45)



Adding Two Numbers (same sign)

Normalised form of a 32 bit (normal) floating point number.

$$A = (-1)^S \times \overset{1.\overbrace{101\cdots1}^{M}}{P} \times 2^{E-bias} \quad (1 \le P < 2, E \in \mathbf{Z}, 1 \le E \le 254) \qquad (7.22)$$

Normalised form of a 32 bit (denormal) floating point number.

$$A = (-1)^S \times \boxed{P} \times 2^{-126} \quad (\overset{0}{\cancel{1}} \le P < 1) \qquad (7.23)$$

$0.\underbrace{0\cdots1}_{m}0$ denormal Normal

Symbol	Meaning
S	Sign bit (0(+ve), 1(-ve))
P	Significand (form: 1.xxx(normal) or 0.xxx(denormal))
M	Mantissa (fractional part of significand)
E	(exponent + 127(bias))
$\mathbf{Z}$	Set of integers

* Recap : Floating Point Number System

22

So, let us quickly go back to chapter 2, and quickly do a revision of how floating point numbers are stored. So, we will use 30 bits 30-bit number system as the running example, but you know same results holds for 64 bit. So, I will not you know, in a cost of, you know not repeating the same thing. Rather I will not repeat it.

So, in a 32 bit number system we store 1 bit most significant bit is the sign bit. So, any number this has a normalized form. So, normalized form basically says that we take this number minus 1 and to the power S if S is 0, the sign bit is 0 means that this is 1, otherwise is a sign bit is 1 means that minus 1 to the power S is minus 1. So, this number is negative. Then we store a significant. So, the significant is of the form 1 point a sequence of 23 0s are 1s.

So, the significant is less in a greater than equal to 1 or rather 1 is less than equal to P and is less than 2. Then we have 2 times E minus bias where E is a real. So E is the exponent that is stored and bias is 127. So, for example, if we want to represent a number the form let us say 1.5, let me go over here let us say we want to represent a number of the form 1.5. So, 1.5 in this then would translate to 1.1 in these 2. So, in this case the sign bit is 0, and then we will save a 23 bit mantissa. The main reason being that the 1 here is common; so the 23 bits of the mantissa.

So, go back to chapter 2 just in case you have forgotten what these terms are so, the 23 mantissa bit mantissa will say one and all 0s pretty much and after that we will save the exponent. The exponent is stored in the bias notation, and so if the exponent in this case is nothing is 1.1 times it is 1.5 times 2 raise to power 0 right. So the exponent basically is 0. So, what we will do is that we will set E as 127, such that the real exponent is 127 minus the bias which is 0.

So, that is pretty much. So, in a 32 bits the way that we would store is that we will have one bit for the sign. We will have 23 bits for the mantissa. Then we will have 8 bits to store the exponent. And the exponent is stored in the bias notation. The real notation is actually the E minus the bias right that is the real exponent. And we also mentioned that we have. So, since this number can verify from 0 to 255 we also mentioned that 0 and 255 are used to hold special values.

So, in particular if we use the value 0. So, last 8 bits are 0, it meant that it meant 2 things if the significant is 0 then the number is 0. Otherwise if the significant is non-zero then it represents a special kind of number. So, we will this special kind of number is called a denormal number. And we also said that if the exponent is 255 which is the highest value and then if the mantissa is all 0s. Then it can either be positive or negative infinity and if the mantissa is not 0 then it is not a number.

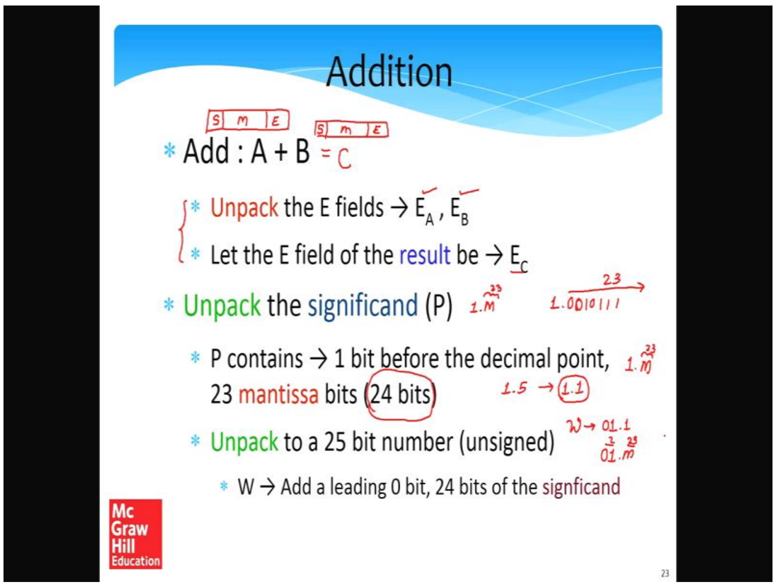
So, I am not discussing these things in detail here, because we already spent a lot of time discussing these issues in chapter 2, but let me nevertheless just summarize, let me just summarize that information that we require. So, what we require is that any floating point number can be expressed in the normal form. Where it will have a sign bit which will determine the sign of the number. Then we will have a significant which is of the form 1 point then 23 0s are 1s. So, this part is called the mantissa.

So, it can be any combination or 0s or 1s, but just 23 of them. And then we will have an exponent the exponent is stored in a bias notation. And the bias is 127. So, we also took into account the fact that when the exponent is 0 then we can have a special class of number called denormal numbers. Where the standard form of a denormal number is like this. I show here there is a small error in this slide. So, instead of one it should be 0. So, 0 is less than equal to P is less than 1.

So, as we can see is that just where the range of normal numbers n. So, let us say these are all the normal number. Of course, the graph is not drawn to scale and let us say this number is 0. So, there is a tiny buffer of numbers here called denormal numbers. And the standard form of a denormal number is that it is of a number of the type 2 raise to the power minus 126 times P times minus 1 to the power S, but here the significant has a different form, is not 1 point something rather it is 0.23 0s are 1s right it is of the form 0 plus the mantissa. Because the mantissa is 23 0s are 1s. Right, in the same number some combination of 0s and 1s.

So, given the fact that we have seen this, we are now ready to in a sense take a look at the way that we shall do floating point arithmetic.

(Refer Slide Time: 08:07)



Addition

- Add : A + B $= C$
  - Unpack the E fields → E_A, E_B
  - Let the E field of the result be → E_C
- Unpack the significand (P) $1.\overset{23}{M}$ $1.\overrightarrow{0010111}^{23}$
  - P contains → 1 bit before the decimal point, 23 mantissa bits $\boxed{\text{24 bits}}$ $1.\overset{23}{M}$ $1.5 \rightarrow \boxed{1.1}$
  - Unpack to a 25 bit number (unsigned) $W \rightarrow 01.1$ $\overset{2}{01}.\overset{23}{M}$
    - W → Add a leading 0 bit, 24 bits of the signficand

So, let us assume that we want to add the numbers A and B. So, if you want to do that let us take a look at the E field and the E field is the value that is stored right. So, it is the real exponent plus127 which is the bias. For example, when we try to save a number of the form 1.5 which is actually 1.1 in binary then the real exponent was 0. So, what we save is 0 plus127 which is the bias.

Let us now take a look at floating point addition. So, let us first consider the inputs. So, we are given 2 32 bit inputs. So, the format of the input is as follows, that we have a one bit sign bit. We have a 23 bit mantissa, and we have an 8 bit exponent field. So, the exponent is stored in the bias notation. So, if the real exponent is x. So, what we are saving is x plus the bias. So, the real exponent is 0 then we are saying 127 because the bias is 127.

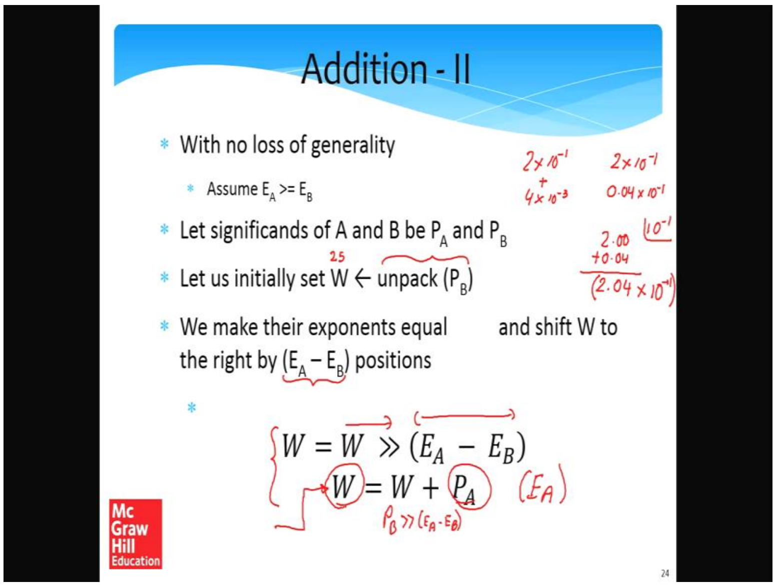
Similarly, the B field is formatted in a same manner. We have a sign bit we have a 23 bit mantissa and we have a 8 bit exponent. So, we need to add these 2 floating point numbers, the result will be another floating point number, which is stored in exactly the same format. And we need to design an algorithm such that given A and B we can compute C. So, what we do is that let me just describe the pseudo code algorithm. So, what is the pseudo code it is basically a list of steps written in a kind of high level description of what the algorithm will do. Then hardware designers can take this algorithm and implement the circuit in hardware.

Let us do one thing. Let us unpack the E fields, and unpack meaning let us extract them. So, let us call the E field of A as E A and the E field the B as E B and let the E field of the result be E C. Similarly let us unpack the significant. So, the significant recall that it is a number for in a normal floating point numbers it is a number of the form 1 point m where m is the mantissa. And mantissa itself contains 23 bits it is a number basically of the form 1 point you know 0 0 1 0 1 1, 23 bits.

So, the one bit before the decimal point and the 23 bit mantissa make it 24 bits let us do one more thing. So, let us then assign it to a number of the form W where we add a leading 0 bit. So, let us say let me consider an example. Let us assume 1.5 in decimal. So, 1.5 in decimal is 1.1 in binary. So, in this case the significant is 1.1 and the mantissa is one followed by 22 0s. So, we will define a number of the form W. So, W is our form 0 1.1.

So, mind you the point cannot be stored in binary. So, we are only using it to enhance readability. So, it is position is implicit. So, W thus becomes a 25 bit number quantity and. So where does this 25 come? Again from once again the significant is of the form 1 point the 23 mantissa bits. So, the significant is 24 bits, and we add a leading 0. So, this will become 0 1 point m where this is 2 bits and this is 23 bits. So, total it is a 25 bit quantity.

(Refer Slide Time: 11:58)



Addition - II

- With no loss of generality
  - Assume $E_A >= E_B$
- Let significands of A and B be P_A and P_B
- Let us initially set $\overset{25}{W} \leftarrow \overbrace{\text{unpack}}\,(P_B)$
- We make their exponents equal and shift W to the right by $\underbrace{(E_A - E_B)}$ positions
- $$W = \overrightarrow{W} \gg \overrightarrow{(E_A - E_B)}$$
  $$\boxed{W} = W + \underset{P_B \gg (E_A - E_B)}{\boxed{P_A}} \quad (E_A)$$

$2 \times 10^{-1} + 4 \times 10^{-3}$

2×10^{-1}
0.04×10^{-1}

$$\begin{array}{r} 2.00 \\ +0.04 \\ \hline (2.04 \times 10^{-1}) \end{array} \quad \boxed{10^{-1}}$$

24

So, with no loss of generality let us assume that E A is greater than equal to E B. And let the let us call the significance of A and B as P B and P B respectively. Furthermore, let us define a temporary variable W which is of the same 25 bit form that we had discussed in the last slide, and let us get it by unpacking the significant of B, which has the smaller exponent. So, let us unpack P B and get W. So, now, let us take a look at the way that we want to do the addition, but before that let me motivate you by saying how we would do it in the case of decimal.

So, let us assume I want to add 2 times 10 to the power minus 1 plus 4 times 10 raise to the power minus 3. So, I would first in a sense unpack it. So, this would be 0.2 this would be 0.004. Subsequently what we will do is I will write it in such a way that the decimal points are aligned. Then we will go forward and do the addition. So, what we need to do in a similar. So, let us there is also another way of looking at this let me may be erase part of it.

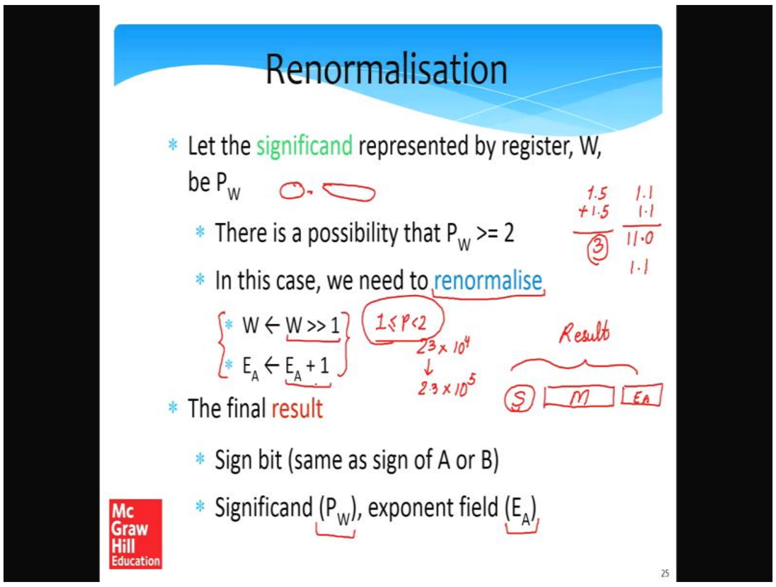
So in this case the exponent of this minus 1 is greater than minus 3. Say instead what I can do is that I can write this as 2 raise to the power 10 to the power minus 1 no problem with that; however, this can be written in a different way. This can be written as 0.04 times 10 to the power minus 1. So, 10 to the power minus 1 part being common I can just you know write it in this side and then I can add, 2.00 and 0.04 to make it 2.04 and then. So, this is the final answer is 2.04 times 10 to the power minus 1.

So, what we can do is that we can do something similar. We can first make their exponents the same and then we can align the decimal points and perform an addition. To make the exponents the same what we would do is that we will make the exponent of B the same as the exponent of A and the exponents can be made equal by shifting W to the right by E A minus E B positions. So, another simple way you know very simple thing. So, E A minus E B is a difference in exponents. And so, what we will do is that if we shift W to the right by these many positions then effectively we will be multiplying 2 to the power E A minus E B with E B to make it to the power E A.

So, then we will have we will be able to align the decimals point and then add right. So, this is the first thing you mean any floating point addition we always align the decimal points. Aligning the decimals points essentially means that we make the exponents the same and we write it in such a way that we can just do a normal addition.

So, we are doing exactly the same thing. We are setting the exponent of A and B to be the same, and this we are doing by right shifting W by E A minus E B positions. Subsequently, now that the exponents are same so their exponent field for both of them is A and the real exponent is of course, E A minus the bias. So that is fine. So, then what we can do is we can do W equals W plus P B, which means that I am adding the significant of P B plus the significant of P B in a sense right shifted by E A minus E B positions.

(Refer Slide Time: 16:46)



Renormalisation

- Let the significand represented by register, W, be P_W
  - There is a possibility that $P_W >= 2$
  - In this case, we need to renormalise
    - $W \leftarrow W >> 1$
    - $E_A \leftarrow E_A + 1$
- The final result
  - Sign bit (same as sign of A or B)
  - Significand (P_W), exponent field (E_A)

1.5
+1.5
3
1.1
1.1
11.0
1.1
$1 \leq P < 2$
23×10^4
2.3×10^5
Result
S
M
E_A
25

So, this is the same as equivalent to aligning the decimals points setting the exponents to be the same and add it now let the significance. So, basically registered W now contains a result of the addition the result of the addition. So, let the significant represented by registered W be P W right. So, this is the significance that is there this is this will be at the form you know something point something right. So, the position of the point is implicit or can be inferred. So, there is a certain possibility that P W can be greater than equal to 2 for example, if you are adding 1.5 and 1.5. So, it is actually 1.1 and 1.1 in binary. So, then what is the result 1 plus 1 is 0 1 plus 1 plus 1 is 11.0.

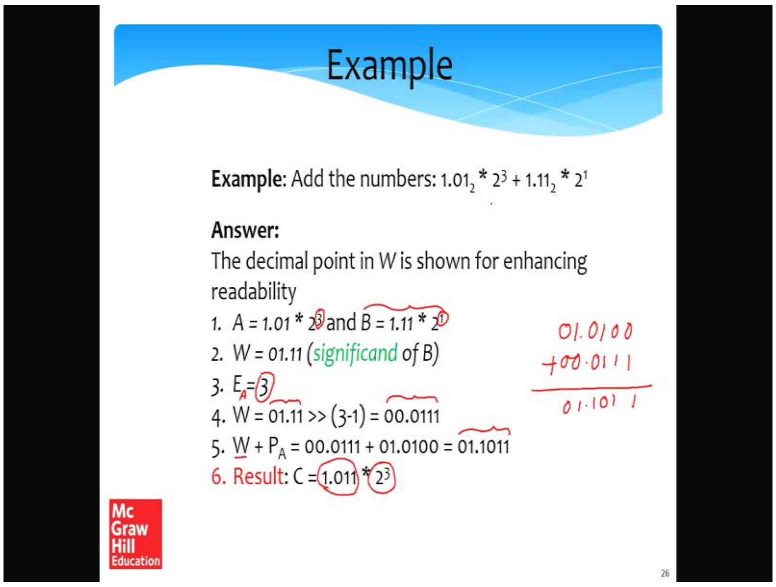
So, where we consider all these points to be aligned; so result is essentially 3 and this is greater than equal to 2. So, this is not a valid significant for us. So, in this case we need to do something called renormalization. Which is basically what we do is that we right shift. So, 11.0 we make it 1.1. This is done by right shifting W by one position. And we increment the exponent E A because E A is now the exponent of both E A is the exponent of the result. So, we increment the exponent by one position.

So, let me give a similar example. In decimal let us assume we have 23 times 10 to the power 4, but we do not want 2 digits to the left of the decimal place decimal point. So, we can alternatively write this as 2.3 times. So, essentially I am right shifting it by one position and times 10 to the power 5. So, increment the exponent. So, we are doing exactly the same in binary to ensure that our significant is between 1 and 2.

So, the final result is like this. Where the sign bit will be same as the sign of A or B. So, mind you we are doing addition and we are thus assuming that the sign bit is the same right. So, how do we define how are at least we defining addition. We are defining addition by the fact that we are either adding 2 positive numbers or 2 negative numbers, the sign bit is different it will essentially become subtraction.

So, sign bit will be the same as sign of A or B right. So, 2 positive numbers we add the result is positive. 2 negatives we add the result is negative. Now we also have the significant P W and we have the exponent field E A. So, given the fact that we have all 3 we can construct a floating point number let me just quickly go back. Given the fact that we know the sign of the result; so let this be the result. So, we know the sign of the result. So, we know the significant. So, from the significant we can find the mantissa right 23 bits after the point. And we also know the final exponent field what it supposed to be. So, we can have E A over here right after these doing these 2 steps.

(Refer Slide Time: 20:10)



Example

Example: Add the numbers: $1.01_2 * 2^3 + 1.11_2 * 2^1$

Answer:

The decimal point in W is shown for enhancing readability

1. $A = 1.01 * 2^{\boxed{3}}$ and $B = \overbrace{1.11 * 2^{\boxed{1}}}$
2. $W = 01.11$ (significand of B)
3. $E_A = \boxed{3}$
4. $W = \overbrace{01.11} >> (3\text{-}1) = \overbrace{00.0111}$
5. $\underline{W} + P_A = 00.0111 + 01.0100 = \overbrace{01.1011}$
6. Result: $C = \boxed{1.011} * \boxed{2^3}$

$$\begin{array}{r} 01.0100 \\ +00.0111 \\ \hline 01.1011 \end{array}$$

26

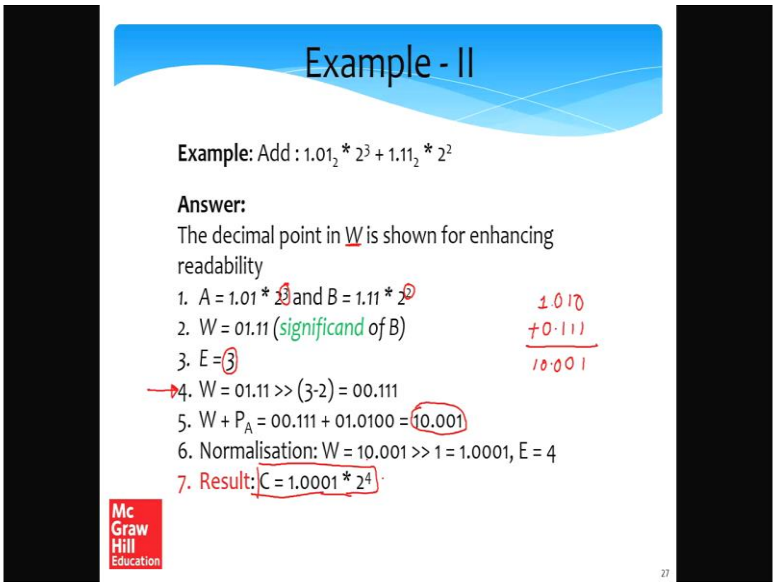
So, let us explain with an example. So, let us consider this example let us add the numbers 1.01 in base 2, times 2 cube plus 1.1 1 again in base 2 times 2 2 raise to the power 1. So, the decimal point will be showing in W to enhance readability, but in the hardware it will not be there. So, let us set A as 1.01 times 2 cube. And let us set B as 1.11 times 2 raise to the power 1. So, the exponent in a will be greater than the exponent of B. And also you know for the sake of readability let us not use the bias notation for E right. So, let us also you know avoid doing that.

So, let us not use the bias notation. Instead let us assume that E is stored as is even though in hardware that is not the case. So, in this case W will be 01.11 which is the significant of B. E is 3 E the exponent of a E A the final exponent is 3. So, to align the decimals points, what we need to do is that we need to right shift W 01.11 by 3 minus 1 places why 3 minus 1 because there is exponent of A and one is the exponent of B. So, we need to right shift W by 2 places. So, this will give us 0 1 1 1 after the decimal point and 0 0 before the decimal point.

So, now the decimal points of both A and B are aligned and the exponents for both of them is E or E A same thing which is 3. So, then we should add W with the significant of P A. So, W is 00.0111 with the significant of P B which is 01.0100. So, if I do the addition. So, let me see plus, I will have 1 1 here 1 plus 1 is 0. It is 1 here and 0 1. So, the final result is 01.1011. There is no reason to re normalize because this is between 1 and 2. So, the final result C is 1.1.01 1 times 2 to the power 3 right because we are taking the exponent of A.

So, this is one example of how we would do it in hardware of course, we have made some simplifications we are not using the bias notation, but this should capture most of the idea, and the idea is simple we are using exactly the same algorithm as we would have used for decimals numbers. So, exactly the same algorithm and the same technique is being used that we first make the exponents the same then we align the decimal points and add.

(Refer Slide Time: 23:20)



Example - II

Example: Add : $1.01_2 * 2^3 + 1.11_2 * 2^2$

Answer:

The decimal point in W is shown for enhancing readability

1. A = 1.01 * $2^{\boxed{3}}$ and B = 1.11 * $2^{\boxed{2}}$
2. W = 01.11 (significand of B)
3. E = $\boxed{3}$
4. W = 01.11 >> (3-2) = 00.111
5. W + P_A = 00.111 + 01.0100 = $\boxed{10.001}$
6. Normalisation: W = 10.001 >> 1 = 1.0001, E = 4
7. Result: $\boxed{C = 1.0001 * 2^4}$

$$\begin{array}{r} 1.010 \\ +0.111 \\ \hline 10.001 \end{array}$$

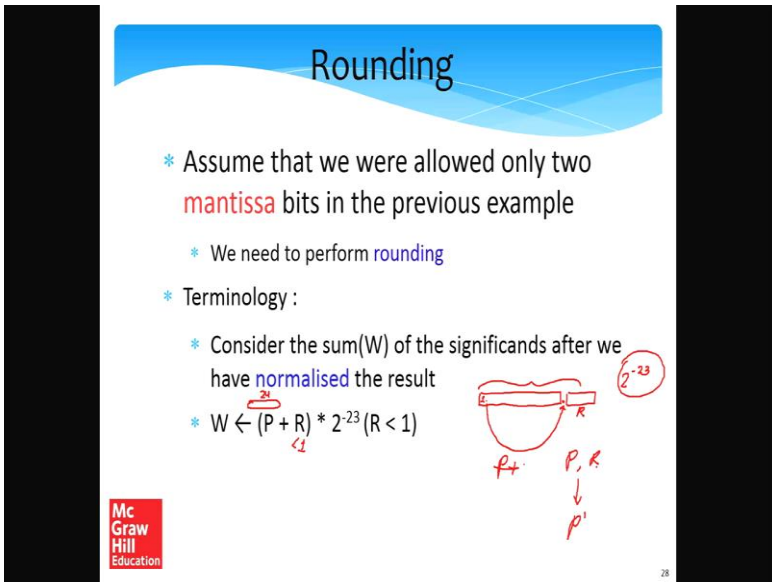
27

So, let me consider one more example. So, in this case we are adding 1.01 in base 2 times 2 cube, plus 1.1 1 in base 2 times 2 square. Again the decimal point in W is shown for enhancing readability and we will not be using the biased scheme right. So that is what we will not be doing. So, a is 1.01 times 2 cube B is 1.1 1 times 2 square and the exponent of A is greater than the exponent of B. So, the significant of B is 01.11 and E the final exponent that at least we are planning at the moment is 3.

So, to make W the first stage would be is that we take the significant of B and shifted to the right by 3 minus 2 positions. So, this will make it 0 0.1 1 1 then we add 0 0.1 1 1 which is the value of W the right shifted version of B significant with the significant of which is 1.01. So, let us take a look at the addition 1.01 0 plus 0.1 1 1. So, 0 plus 1 is 1 1 plus 1 is 0 carry a 1 1 plus 1 is 0 carry of one. So, it is one 0 point 0 0 1.

So, as we can see this number is greater than 2. So, is greater than or equal to 2 and that is a problem. So, as a result this needs to be fixed. The way that we fix it is that we sort of re normalized. So, we shift it to the right by one position. So, then this becomes 1 point 3 0s and 1 and we increment E by 1. So, 3 plus 1 is 4. So, the final result that we have over here is 1.0001 times 2 to the power 4. So, as we see that after doing the addition there was a need to renormalize the result. And so, what we did is that we right shifted W and we added 1 to E. So, we added 3 plus 1 and made it 4.

(Refer Slide Time: 25:46)



Rounding

- Assume that we were allowed only two mantissa bits in the previous example
  - We need to perform rounding
- Terminology :
  - Consider the sum(W) of the significands after we have normalised the result
  - $W \leftarrow (\overset{24}{P} + \underset{<1}{R}) * 2^{-23} (R < 1)$

Let us now discuss a very tricky issue in the word of flooring point math. So, let us discuss rounding. So, assume that we were allowed only 2 mantissa bits in our previous example. So, let us go back. So, assume that here right we are allowed only 2 mantissa bits. So, we can store only 1.00 and we need to discard these 2 bits. So, well if a 0 1 maybe we can discard, but assume that the final result was a they form 1.0011 it might have been inappropriate to discard 1 1 without making a change in the 0 0. So, we it would have been may be more appropriate to round it to 1.01.

So, rounding is an issue especially when the number of mantissa bits is finite. And this issue has to be considered. So, let us first define some normal standard terminology, and then we will talk about rounding. And why does rounding comes there is another reason.

So, the reason is that we are essentially right shifting. So, initially you know if I consider the significant of a it is of the form 1 point you know something for 23 bits that is fine, but this is being added with W which is the significant of B right shifted by certain number of bits. So, for all we know this can be of the form 0 0 0 and then a certain number of bits. Let me just erase and clean this right and then 23 bits.

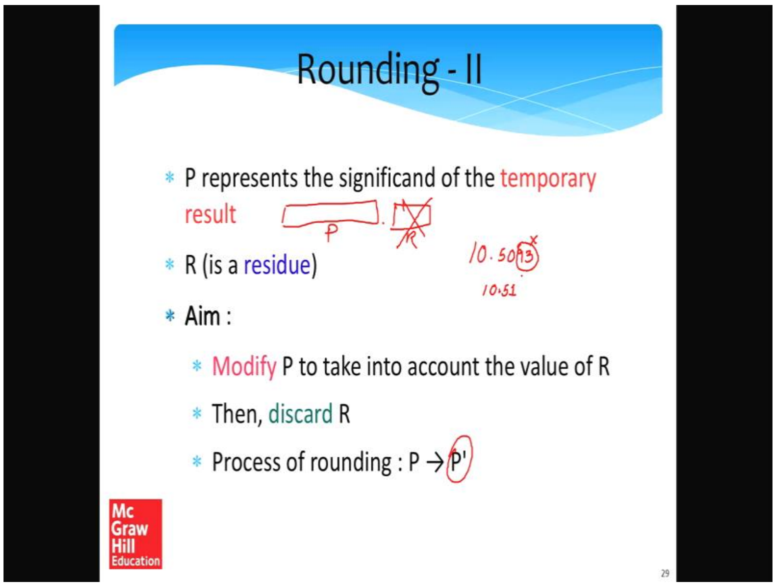
So, when we do an addition it will be 1 point something you know it might be like you know greater than 23 bits. So, there will be a requirement to stop this at 23 bit because that is all that we can store, and the remaining bits have to be discarded, but they should be discarded in a in a way that result is the most accurate, and for that reason we have to do some amount of rounding. So, let us the terminology first. So, consider the some W of the significance that we have, and note that in hardware we are not storing the decimal point it is implicit. So, in a sense that we know where it is, but is not being stored. So, the significant is pretty much being stored as a 24 number in hardware.

So, it is in reality it is 1 point the mantissa which is 23 bits the way that the significant is being stored in hardware is one continuous sequence of 24 bits. So, it can be in a alternatively interpreted as a 24 bit unsigned number. So, let us use that interpretation over here. And so, this is all that we can keep we do not have space to store any more, but it is possible that during our operation there will be some additional bits which actually have to be discarded.

So, let us do one thing let us assume that this is you know let us logically assume that this 24 bit significant is one kind of number. And the bits that are been discarded is another number. And let us say that the final number W is P plus R times 2 to the power minus 23, where if I multiply the significant with 2 raise to the power minus 23 the decimal point will come here just after the first bit. And let us define the R portion the residue as the part that we will ultimately throw out right. So, the residue we can treat it as less than 1 in this particular equation, we are assuming that since we are taking 2 to the power minus 23 out the decimals point from this point is actually moved to this point. So, that is what we are assuming.

So, any number we have your regular significant and residue and the residue part is thrown out. So, we want to round the significant in such a way that it reflects the residue correctly.

(Refer Slide Time: 29:53)



Rounding - II

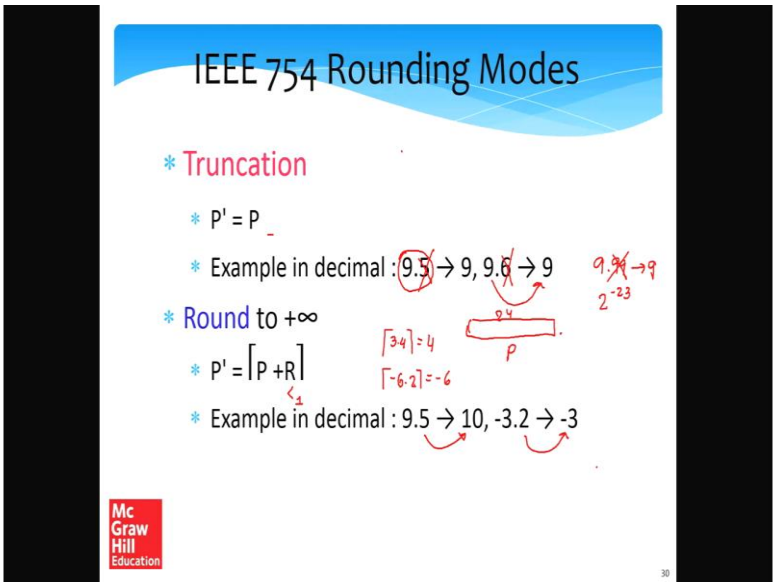
- P represents the significand of the temporary result
- R (is a residue)
- Aim :
  - Modify P to take into account the value of R
  - Then, discard R
  - Process of rounding : P → P'

P
R
10.5093
10.51
Mc Graw Hill Education
29

So, the term terminology again P represents a significant of the temporary result it is stored as a 24 bit unsigned number. Right that is P and we are storing the residue as a number which is strictly less than 1. So, aim is to modify P to take into account the value of R and then we can happily discard R.

For example, if I you know let me give an example in the world of the decimal numbers. So, let us assume we are allowed to store only the 2 digits after the decimal points and the number is of the form 10.5093. So, since these 2 are the residue they can be discarded, but we should update the significant to make it more accurate which will be 10.51. So, that is exactly what we want to do and then. So, in this process we have created a new significant which is P dash and P dash reflects the residue correctly right because the residue cannot be stored we do not have the space for it.

(Refer Slide Time: 31:03)



IEEE 754 Rounding Modes

* Truncation
  * P' = P
  * Example in decimal : $9.\cancel{5} \rightarrow 9$, $9.\cancel{6} \rightarrow 9$
* Round to +∞
  * $P' = \lceil P + R \rceil$
  * Example in decimal : $9.5 \rightarrow 10$, $-3.2 \rightarrow -3$

$9.\cancel{99} \rightarrow 9$
2^{-23}
24
P
$\lceil 3.4 \rceil = 4$
$\lceil -6.2 \rceil = -6$
< 1
30

So, IEEE 754 which is the floating point standard that we have discussed in a fair amount of depth in chapter 2 has several rounding modes. So, let us discuss what these rounding modes are so, one rounding mode is called truncation. So, truncation basically means that P dash is equal to P. So, we just get rid of the residue. So, let us say you know I am not allowed to save a single, even a single digit after the decimal point.

So, in this case what I will do is 9.5 I will just round it to 9. And I will completely discard the residue. In this case I will discard 6 and even 9.6 I will round it to 9 and even 9.99, I will round it to 9. So, I am just truncating I am just discarding the residue and I am not updating the significant. So, this is a very simple mode of rounding and this is typically you know not advisable to use, but you know we can use it if you want to. So, this is called truncation.

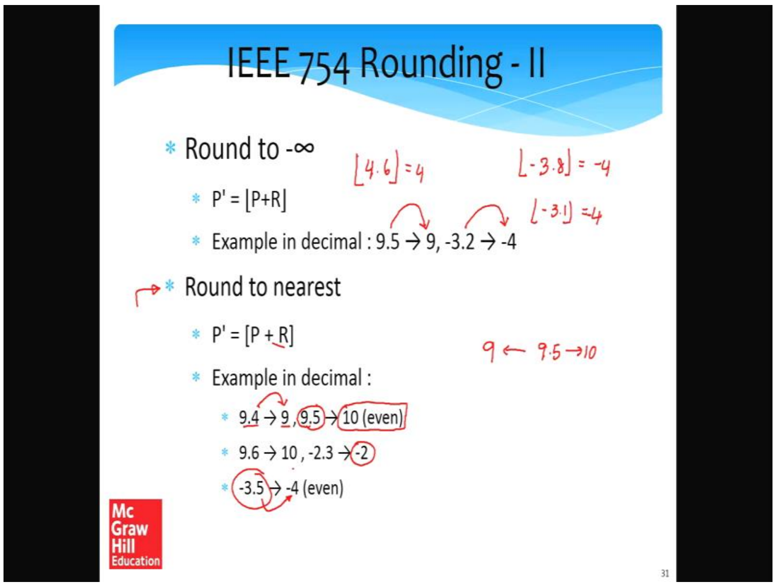
The other is round it to plus infinity which means that we add the residue to the significance. So, mind you once again the significant is a 24 bit consigned number and the residue is less than 1. Basically if we will add both of them, and we will take that ceiling; so the ceiling operator essentially takes you to the next higher integer. For example, the ceiling of 3.4 is 4 the ceiling of minus 6.2 right. The next higher integer is minus 6. So, pretty much we will take the ceiling of the value. So, 9.5 will round it to 10 because that is the next higher integer. Minus 3.2 you know if we have a number of this type we will round it to minus 3, because that is the next higher or greater integer. So, this is rounding to plus infinity.

So, this also is something that can be done is supported by the IEEE 754 format, and the advantage here is that we always choice the higher integer. So, for example, what we are doing is we are considering a 24 bit significant and then we are so, since we have taken 2 to the power minus 23 out of the equation I will just go back. So, this equation is the crux and we need to ensure that all of us are understanding it. Otherwise it will cause a little bit of an issue lot of issue rather. The spirit of this equation is as follows that in the course of our algorithm we will produce intermediate results which are more than again a 24 or 25 bits can even have 40 bits.

So, they will have a certain significant part which is of the form 1 point something and then they will have a certain residue part. See if I take 2 to the power minus 23 out then pretty much what happens is that the decimal point in a sense significant was of the form 1 point something, the decimal point is going to move from this point to this point. Also thus we will have a purely positive number even a pure integer a pure not positive, but a pure integer called P which is 24 bits can be more does not matter it is a pure integer does not have a decimal portion. And a residue which is after the decimal point and the residue is thus less than 1.

So, our aim is to consider P plus R, I sorry, our aim is to consider P and R both of them. And come up with a new significant P dash and discard R. So, to discard R will any way have to be discarded, but we will have to see how can we take the effect of R into account you know R is small we can get rid of it, but ever it is large then P has to be incremented. Some rounding has to be done. So, our aim is to move from P to 1 more significant P dash and discard R and, but take the value of R into account. So, we discussed truncation we have discussed rounding to plus infinity.

(Refer Slide Time: 35:39)



IEEE 754 Rounding - II

- Round to $-\infty$
  - $P' = \lfloor P+R \rfloor$
  - Example in decimal : $9.5 \rightarrow 9$, $-3.2 \rightarrow -4$
- Round to nearest
  - $P' = [P+R]$
  - Example in decimal :
    - $9.4 \rightarrow 9$, $9.5 \rightarrow 10$ (even)
    - $9.6 \rightarrow 10$, $-2.3 \rightarrow -2$
    - $-3.5 \rightarrow -4$ (even)

$\lfloor 4.6 \rfloor = 4$ $\lfloor -3.8 \rfloor = -4$

$\lfloor -3.1 \rfloor = -4$

$9 \leftarrow 9.5 \rightarrow 10$

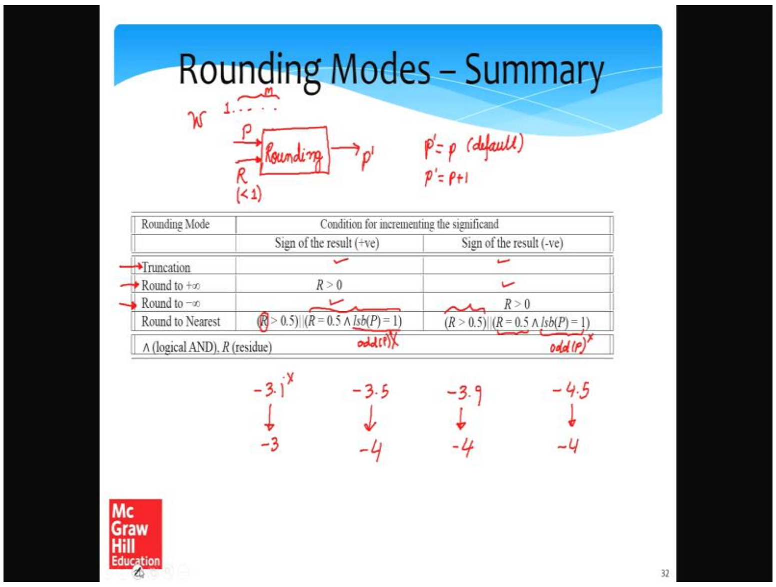
31

So, one more method is rounding to minus infinity or negative infinity, which means we go to the lower integer it is the opposite of rounding to plus infinity. So, in this case what we do is that we actually go down. So, from 9.5 we go down to 9, and from minus 3.2 we will go to the lower integer which is minus 4. So, mathematically represented we will add R to P. And we will take the floor of the value. So, floor means go to the lower integer for example, 4.6 the floor is 4. Minus 3.8 we go to the integer that is lower than it that is minus 4. In fact, even from minus 3.1 we go to the next lower integer which is minus 4.

So, what are the 3 modes that we have discussed right now? Straight forward truncation round to plus infinity and round to minus infinity. Now we will discuss slightly more sophisticated mechanism which is also more popular. So, in this case what we do is that we said P dash is equal to P plus R; actually some function of P plus R. So, we will discuss this function is called the box function, but we will discuss this function in detail. So, what we will do is that we will round it to the nearest is called round to nearest. So, we will round it to the nearest integer. So, 9.4 will become 9 and 9.6 will become 10.

So, 9.4 the nearer one is 9 and 9.6 will become 10 similarly minus 2.3 the closer integer is minus 2. So, that is fine, but the difficulty comes with 9.5 and minus 3.5 because they are in the middle. So, what we will do is we will always prefer we will round it. So, 9.5 can be rounded to either 9 or can either be rounded to 10. So, if equally you know close to both 9 and 10. So, in this case what we will do is as a matter of a convention we will prefer the even number. So, even number is 10. So, we round it to it. Similarly, for minus 3.5 we will round it to the nearest even number which is minus 4. So, what is round to nearest again, round to nearest is that if the number is not of a form x plus 0.5 we round it to the nearest integer; however, if it is a number of the form x plus 0.5 then we choose the integer that is even right the nearest integer that is even.

(Refer Slide Time: 38:26)



Rounding Modes – Summary

Rounding Mode	Condition for incrementing the significand	
	Sign of the result (+ve)	Sign of the result (-ve)
Truncation	✓	✓
Round to $+\infty$	$R > 0$	✓
Round to $-\infty$	✓	$R > 0$
Round to Nearest	$(R > 0.5) \|\| (R = 0.5 \wedge lsb(P) = 1)$	$(R > 0.5) \|\| (R = 0.5 \wedge lsb(P) = 1)$
$\wedge$ (logical AND), R (residue)		

Let us now try to summarize the rounding modes. So, the rounding can be thought of as a function. So, the inputs or the function are like this. So, let us just draw a black box. So, what goes in is the significant as an unsigned number is a 24 or 25 bit it does not matter, but what goes in is a unsigned number which is the significant. And so actually even before this we compute the final output W, and then we extract the significant which is pretty much the amount that we can fit with a 23 bit mantissa.

So, it should be of the form 1 point something and so, mantissa is over here 23 bits. So, that is out significant which we have extracted out of the final result of the mathematical operation right be it addition or subtraction, but we will talk about just addition with the same signs for the time B. Then what we do is that we send this number which is actually 24 bit number as the significant and we also send the residue which is less than 1.

What comes out is a new significant P dash. And so, P dash there are 2 choices one is P dash is equal to P. So, that is a default. So, here basically we set the residue to 0. The other choice is at P dash is P incremented with 1, so P plus 1. So, in this case we have effectively done some rounding. So, in this case we have incrementing the significant. So, let us now try to state the basic rules for different rounding modes. Let us consider truncation first. So, in the case of the truncation what we do is we completely discard the residue. So, P dash is equal to P. So, in both the cases we do not do anything and the default remains and the default when the sign of the result is positive and when the sign of the result is negative is the same, which is that we just discard the residue. So, that is the reason we do not do anything other than mere truncation.

Let us now talk about rounding to plus infinity. So, let us take a look at 2 numbers 3.1 and minus 3.1. So, in the case of 3.1 this needs to be rounded to plus 4, because that is the next greater integer in the case of minus 3.1 this needs to be rounded to minus 3, because that is the next greater integer. So, the conditions are as follows; so if the sign of the result is positive and we have a non-zero residue which is exactly this case. Then what we need to do is we need to increment the significant. So, we need to increment 3 to 4. Otherwise if the residue is 0 anyway we do not need rounding. So, we can leave it as is.

If the sign of the result is negative, as in this case minus 3.1 in that case; so basically if I draw a number line in a minus 3 will lie here minus 4 will lie here and all intermediate numbers will lie over here. So, since we are rounding towards plus infinity all the numbers will be rounded to minus 3. So, what we are essentially doing is that if is this can be any number of the form minus 3 point x you will truncate this x and we will round everything to minus 3. So, essentially we need to let the default happen which is that we said P dash equal to P and discard the residue.

Let us now after a brief cleanup of course, let me clean this part up, let us now talk about and rounding to minus infinity which means choosing the lower integer. So, let us again take a look at 3.1 and minus 3.1. So, in the case of 3.1 we need to round it to 3. So, even if. So, let me draw a number line. So, let me have I am sorry let me have 4 over here and 3 over here. So, all the numbers between 3 and 4 will get rounded to 3. So, what are doing is that for when the sign of the result is positive we are discarding the residue. So, that is a reason we can do a default action here.

However, if the number if negative then; so let me draw a number line. Let me write minus 3 here and minus 4 here, so minus 3.1 will stand somewhere here. So, for all such numbers what we need to do is that they need to be rounded to the lower integers which are minus 4. So, the logic is that if there is any non-zero residue which means that the number is somewhere in this range then we need to perform a rounding which means increment the significant. So, what we need to do is we need to check if the residue is non-zero. If that is the case, then we will increment the significant and from minus 3 will take it to minus 4.

So, once again just to refresh for all the empty entries, we never increment the significant rather we fall back to the default which is to discard the residue and keep the significant as is wherever there is a condition, if the condition is satisfied then only we increment the significant. So, let us now discuss the last rounding mechanism which is round to the nearest which is also by the way the most complicated. So, just give me one second.

So, rounding to the nearest is the most complicated, but it is it is not that hard. So, let us see. So, let us assume that I have a 3 numbers 3.1 3.5 and 3.9. So, 3.1 I need to round to 3, 3.5 I have a choice between 3 and 4, I choose the even number. So, I round it to 4 and 3.9 the closest integer is 4. So, the logic is like this. I will first take a look at the residue if the residue is greater than 0.5 as in the case of 3.9, then I will increment the significance I will increment 3 to 4. Otherwise I will take a look at the other condition which is at the residue should be equal to 0.5. So, in this case the residue is equal to 0.5 and the least significant bit of P which is the significant is 1.

So, what this basically means that P is an odd number. So, recall that what I am trying to say is an odd number implies that it is you know implies and also you know it is implied back, that is least significant bit of that number. So, in this case P is equal to 1 in binary and the reason is very simple that in this binary notation for an even number. Since these sums of powers of 2; for an even number it is least significant position has to contain 0 because the rest of the number are a power of 2 multiples of 2 essentially and for an odd number the least significant bit position should be 1.

So, what we are essentially checking here is that you know we are checking that P is an odd number. So, in this case the significant P is 3 which is an odd number and the residue is 0.5 that is the reason we are incrementing 3.5 we are rounding it to 4. If I would have rather considered 4.5 then the first sub condition would have been true that the residue is 0.5; however, the significant P would not have been odd. So, this condition would have evaluated to false hence we would have not rounded it, and we would have rather truncated it and the answer would have been 4.

So, this is the case when the sign of the result is positive. So, let us now see what happens when the sign of the result is negative. So, let me again consider the 3 numbers minus 3.1 minus 3.5 and minus 3.9 right the negative variants. So, what I do is I will take a look at the residue, if the residue is greater than 0.5 I will do something, or if the residue is 0.5 I will do something else, but both the conditions for these number are false that is the reason I will truncate. So, minus 3.1 will become minus 3.

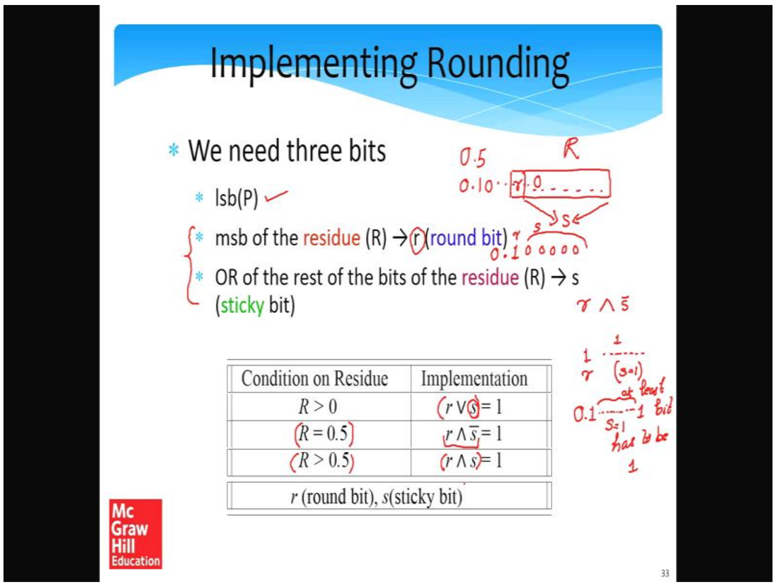
Now, if you take a look at minus 3.9 the first condition is true, the residue point 9 is greater than 0.5. That is the reason we will increment the significant and this will get rounded to minus 4 now we consider minus 3.5 well see in this case the residue is 0.5. So, this sub condition is true again what I need to check is that P is an odd number. If it is an odd number something needs to be done otherwise we can go with the default. So, in this case there is an odd number. So, that is the reason we round it to minus 4, had it been minus 4.5 then this condition would have been false because 4 is an even number.

So, instead we have rounded it to minus 4, which is exactly what we want to do in this particular scheme. That we always want to round it to an even number if the residue is 0.5. So, these are all the conditions for all the 4 rounding modes truncation, which means do nothing. So, do nothing means keep the significant as is get rid of the residue, in all cases we get rid of the residue, but here we keep the significant as is round to plus infinity we check if the residue is non-zero. And of course, if the sign is positive the sign is negative again we do nothing and round to minus infinity is just the reverse of round to plus infinity, round to nearest is somewhat tricky we need to there are 2 cases.

We need to check if the residue is greater than 0.5, we increment the significant if the residue is 0.5 and the significant is odd, then also we increment. And this is the same condition for both positive as well as negative results. So, one very important take away point here is that to implement this in hardware we have to have multiple circuits. So, the first circuit needs to compare the value of the least significant bit of P with 1 right. So, you need to have one comparison circuit that is one circuit that you need to have the second circuit’s needs to compare the residue with 0 and compute this expression. The third circuit needs to compare the residue with 0.5 and see if the residue is greater and forth you need to compare the residue with 0.5 and see whether it is equal.

If you can compute the values of all 4 of these expressions, then we can compute all the rounding conditions.

(Refer Slide Time: 50:51)



Implementing Rounding

- We need three bits
  - lsb(P) ✓
  - msb of the residue (R) → r (round bit)
  - OR of the rest of the bits of the residue (R) → s (sticky bit)

Condition on Residue	Implementation
$R > 0$	$(r \vee s) = 1$
$(R = 0.5)$	$r \wedge \bar{s} = 1$
$(R > 0.5)$	$(r \wedge s) = 1$

r (round bit), s(sticky bit)

0.5
R
0.10..r.0......
s
r
0.100000
$r \wedge \bar{s}$
1
1
r
(s=1)
at least
0.1......1 bit
S=1
has to be
1
McGraw Hill Education
33

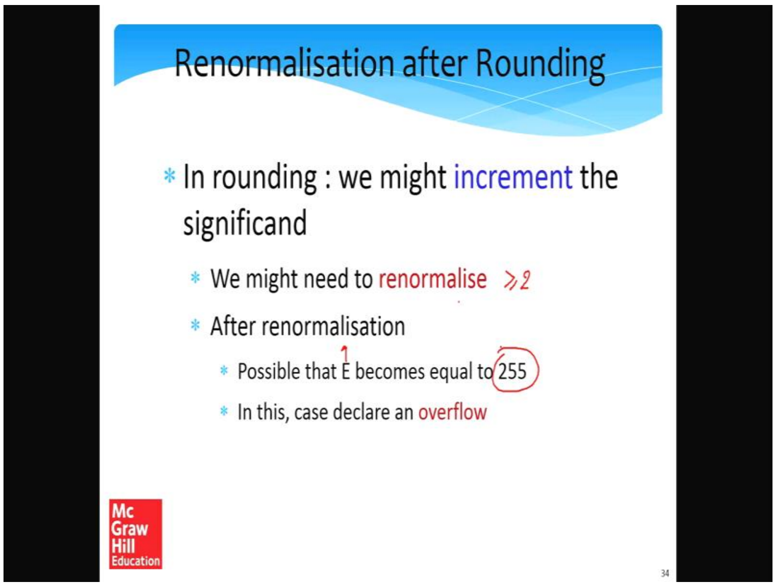
So, this is exactly what we need we need the least significant bit of P. So, this very easy to get I will not discuss this right is just one bit has to be extracted, but to get the rest is somewhat tricky. So, what we will do is we will add a couple of more bits to get what we want. So, let us consider the residue R and let us refer to it is most significant bit as R or the round bit. So, if I consider you know this as the residue. So, let me consider is most significant bit let me call it R. And let the logical or of the rest of the bits of the residue you know or of all of these bits just a simple or operation let me refer to this as s.

So, if s is one it means that at least one of these bits is 1; if s is 0 means that all of these bits are 0. So, I claim that just with R and s bits we will be able to evaluate all the conditions let us see how. So, R greater than 0; if R is greater than 0 it means the at least one of the bit in the residue is 1. So, which means that either R is 1 or S is 1. If S is 1 means one of the bits here should be 1, has to be 1. So, the condition is we just to R or s is equal to 1. If any of if this condition is true it means that one of the bits here is 1 and we can infer that the residue is greater than 0. Now let us consider one more case where the residue is equal to 0.5. If the residue is equal to 0.5 it means that the round bit is 1, because you know it is pretty much less than 1. So, we have 0 point here before it which is implicit.

So this is. So, what is a representation of 0.5 in binary? It is 0.10000. So, basically the R bit will be equal to 1, but the all of these bits will be equal to 0. So, there or will also be 0. So, the condition will be that R is one and s is 0. So, this can be inferred as s bar s compliment is 1. So, this is the condition R and s compliment is equal to 1, the last condition is R is greater than 0.5 this will happen when R is one and at least one of the bits in the at least one of the rests of the bits is equal to 1.

So, then or function will evaluate to 1. So, the s bit will be 1. So, basically the function here is at both R as well as S both of them need to be one for the residue to be greater than 0.5, why is this again because any number is greater than 0.5. It is binary representation has to be at the form 0.1 and then you know can be anything else, but among these bits at least one bit at least one bit has to be 1. So, then we can say that the number is greater than 0.5. And if one of these bits are equal to 1 then it means that their or is 1. So, S is equal to 1 with the condition is at R is 1 and s is 1. So, R and S1 is equal to 1. So, we thus see that keeping the R bit or the round bit and the s bit as the sticky bit you know just computing them when you output is being created, will help us get all the conditions to implement all the 4 rounding modes.

(Refer Slide Time: 54:41)



Renormalisation after Rounding

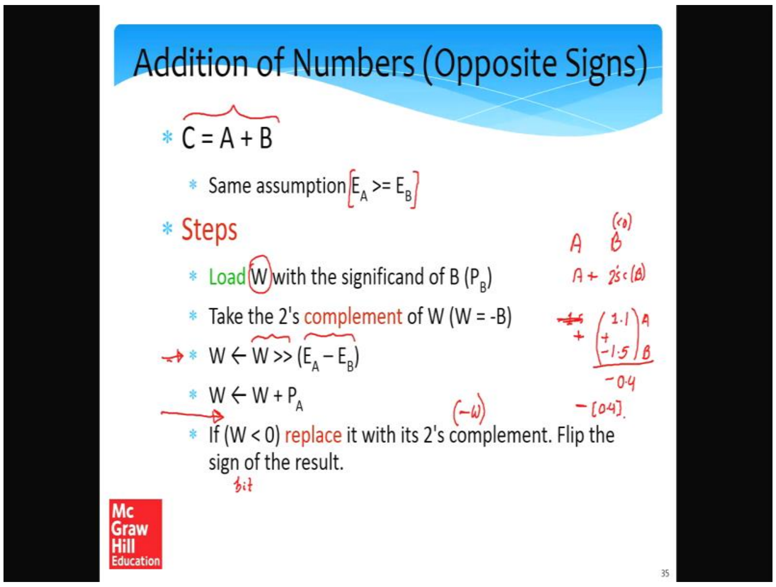
- In rounding : we might increment the significand
  - We might need to renormalise ≥ 2
  - After renormalisation
    - Possible that $\overset{1}{E}$ becomes equal to 255
    - In this, case declare an overflow

34

Let us look at the process of rounding in a little bit more detail one particular aspect that we have not talked about. So, when we increment the significant there is a possibility that we might have to renormalize once again. The reason is that the significant might become greater than equal to 2. After renormalization it is again possible that the exponent field E becomes equal to 255, which is actually an invalid value because 255 can signify 2 things. Either not a number or plus minus infinity in both cases it is not a valid number.

So, we need to declare an over flow which means that the number the result cannot be represented with the number of bits that we have; essentially the number over flows the range of the number system.

(Refer Slide Time: 55:34)



Addition of Numbers (Opposite Signs)

- $C = A + B$
  - Same assumption $[E_A >= E_B]$
- Steps
  - Load W with the significand of B (P_B)
  - Take the 2's complement of W (W = -B)
  - $W \leftarrow W >> (E_A - E_B)$
  - $W \leftarrow W + P_A$
  - If (W < 0) replace it with its 2's complement. Flip the sign of the result.

(<0)
A B
A + 2's c (B)
1.1 A
+
-1.5 B
-0.4
- [0.4].
(-W)
bit
35

So, let us now look at additional numbers with opposite signs. So, in in the world of sign numbers, we technically do not have a difference between addition and subtraction. That is the reason we have classified power discussion differently, we have set that let us consider a numbers with the same sign and let us consider numbers with the opposite sign.

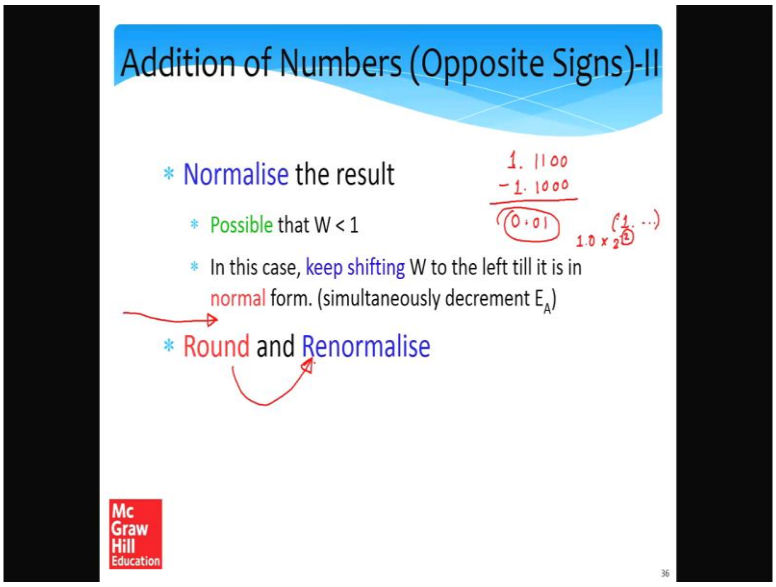
So, if I consider numbers with the opposite sign the algorithm is more or less similar a little bit of difference in some places. So, with no loss of generality let us assume again E A is greater than or equal to E B. So, the first few steps are the same we load the register W with the significant of B P B. So, since A and B do not have the same sign we need to essentially ensure that they have the same sign and this can be done by negating P. So, we take the 2s compliment of W. So, essentially set it minus B and then we will perform an addition. So, the next step would be to align the decimal points. So, to align the decimal points, what we do is we do the same thing that we did in the case of number which had the same sign, which was that we right shift W by E A minus E B positions, such that the exponents are the same and the decimals are aligned.

Subsequently we add W with P A. So, this will complete our addition. So, we do that and so recall that till this point everything is standard. So, even if let us say we where you know to add 2 numbers with opposite signs in regular arithmetic. So, since this significant is positive let one number be A, and let the other number be B and B is in this case less than 0. So, what we would do is we would add a with the 2s compliment of B right. The 2s C 2s complement of B that is exactly what we have done we have taken the 2s complement of W which contains a significant of B. We have aligned the decimal points then we perform the addition.

Now, it is possible that when we perform the addition. The significant the result might be less than 0. So, why would this happen well simple reason that this would happen is that let us assume you know one case where we are adding let us say minus 1 you know minus 1, point sorry let us say we are adding 1.1 with minus 1.5. So, in this case the exponents are the same. So, the first one can be A, and the second number can be B. So, then we will see that the sign of the result will be negative. So, that is a reason. So, it is very easy to handle this case we will essentially replace W with the 2s complement of W.

So, basically we replace it with minus W and we will flip the sign of the result rather we will flip the sign bit of the result to indicate that. So, initially the sign bit will be the sign of A, but if the if after addition the result is negative, then what we will do is we will replace W with minus W and flip the sign of the result. So, in this case initially the sign of the result is positive, but since we get minus 0.4 over here, we will flip the sign of the results. So, it will become minus and pretty much the mantissa significant and exponent will encode 0.4.

(Refer Slide Time: 59:26)


Addition of Numbers (Opposite Signs)-II

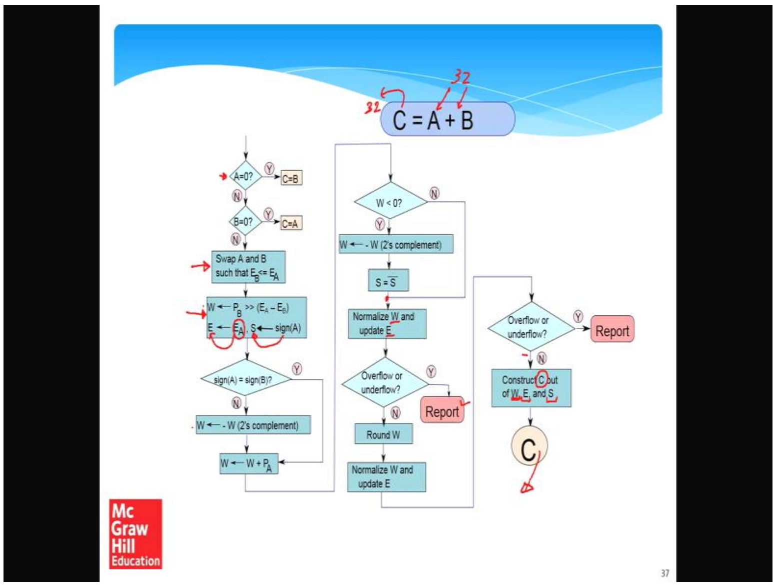
- Normalise the result
  - Possible that W < 1
  - In this case, keep shifting W to the left till it is in normal form. (simultaneously decrement E_A)
- Round and Renormalise

1.1100
-1.1000
$\boxed{0.01}$
1.0×2^{-2}
$(1. \ldots)$
36

So, after we add to perform the addition and adjust for the sign, we need to normalize the result. So, there are some reasons why we need to do that. Let us assume that we are trying to add these numbers. So, the result will pretty much be 0.01. So, this is not valid significant. The reason it is not a valid significant is basically because it is not of the form 1 point something. So this is in a one of the cases where the result W is less than 1. So, what we need to do is we need to keep shifting W to the left till it is in the normal form which is 1 point something. So, in this case the normal for form would 1.0; however, it will be 1.0 times 2 to the power minus 2.

So, what we have essentially have to do is we have to keep left shifting W or we have to keep shifting W to the left till it is in the normal form. And simultaneously we need to decrement the exponent E A till we reach the normal form. So, in this case we will have to decrement the exponent E A by 2. After that the rest of the procedure is the same we need to round number wells. So, rounding in this case is not really required, because if you are shifting to the left then we are essentially shifting in 0s, but otherwise it might be required, especially if the exponents are the same and some bits are remaining. So, we need to round and renormalize alright. So, once we have discussed all of these let us summarize our entire discussion in a flow chart.

(Refer Slide Time: 61:32)



So, the flow chart looks at all possible cases. So, let us assume that we are trying to add A and B to floating point numbers A and B and the result is one more floating point number which is C. So, they can have the same sign or they can have different signs. So, first check that we should probably do is check if A is 0. See if A is 0 it is a simple case the result is equal to B. If that is not the case we can check if B is 0, if B is 0 again it is a simple case, we can set the result to A and mover forward then we need to swap A and B such that E B is less than equal to E A or E A essentially as the higher exponent.

So, we need to ensure with no loss of generality that E B is less than equal to E A, if that is not the case we will just swap A and B. Then what we do is that we will subsequently what we do is that we align the decimals points. So, what we can do is we can perform this step. So, in this step we align. So, we set the exponents the same and align the decimal points and this is achieved by essentially setting W as P B right shifted by E A minus E B positions, and we set E A the exponent of A as the default exponent of the result and the sign of the A as the default sign of the result. So, recall that in our discussion we the order was slightly different, but I will come to it. Then we check if the sign of A is equal to the sign of B if that is the case then well that is great.

So, we just add W we just said W as the old value of W which is the significant of B shifted to the right by some places, plus the significant of A. So, we just perform the addition. Otherwise we take the 2s complement of W and then we perform the addition.

So, recall that we had reversed the order of this step and this step in a in one of our previous slides. So, it technically does not matter if you have enough number of bits because we are not losing precision, and when we are you know inside the hardware we are not limited to have 32 bits we can have more bits because they are after all temporary spaces.

So, the constraint is that the inputs will be 32 bits right, and the output also needs to be 32 bits, but inside the circuit we can definitely represent the significance and the mantissa with more number of bits, and later on discard some bits using a proper rounding policy. So, that is the reason the order of right shifting in 2s complement does not matter, and as long as in a we have sufficient space which any way we have or we are assuming to have.

So, after performing the addition especially if the signs are different there is a possibility that W is less than 0, if this possibility is there that W is less than 0, then in if this is the case the W is less than 0 well then we again replace W with the 2s complement of W, and we flip the sign bit of the result the sign of the result right, as we are showed in the previous slide that when the result was minus 0.4. So, we actually set the significant to 0.4. I mean 0.4 properly adjusted and we flip the sign of the result if it is not the case we can directly move over here.

Then we normalize W and bring it to the standard form, and also update the exponent E. At this point it is possible that we can either have an over flow or underflow what this means is that we might have exceeded the range of the number system. So, this should be reported as an over flow then we need to perform a rounding, because you know the mantissa can only be 23 bits the additional bits have to be discarded. So, we need to round. After rounding we need to renormalize W and update E. And the process of renormalization itself can induce an over flow or under flow. So, this should be reported and if we have reached this step means everything is in order. So, we need to construct the result C out of W which is the final significant. So, we can construct the mantissa out of it the exponent field and the sign bit and then we report the result as C.

So, our advice all the students to take a look at this flow chart in great detail. The flow chart is otherwise very simple as long as all the individual steps are understood, but it is very important to understand each and every individual step. So now, let us discuss floating point multiplication and division.