

Digital System Design
Professor. Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar
Lecture 6
Negative Number Representation 2

(Refer Slide Time: 00:16)

The image displays two screenshots of a digital system design interface. The top screenshot is for a 3-bit system. It shows a circular arrangement of 8 nodes representing bit patterns from 000 to 100. The nodes are labeled with their decimal values: 0 (000), 1 (001), 2 (010), 3 (011), 4 (100), 3 (-3), 2 (-2), and 1 (-1). The nodes for 0, 1, 2, and 3 are green, while the nodes for -1, -2, -3, and 4 are red. A dashed line separates the positive and negative values. The interface includes a control panel with 'Remove Bit' and 'Add Bit' buttons, and an 'Encoding' section with 'Positive', 'Signed', '1s Complement', and '2s Complement' options. The NPTEL logo is visible in the top right corner. A small video inset shows a man speaking.

The bottom screenshot is for a 4-bit system. It shows a circular arrangement of 16 nodes representing bit patterns from 0000 to 1000. The nodes are labeled with their decimal values: 0 (0000), 1 (0001), 2 (0010), 3 (0011), 4 (0100), 5 (0101), 6 (0110), 7 (0111), 8 (1000), 7 (-7), 6 (-6), 5 (-5), 4 (-4), 3 (-3), 2 (-2), and 1 (-1). The nodes for 0 through 7 are green, while the nodes for -1 through -7 are red. A dashed line separates the positive and negative values. The interface includes a control panel with 'Remove Bit' and 'Add Bit' buttons, and an 'Encoding' section with 'Positive', 'Signed', '1s Complement', and '2s Complement' options. The NPTEL logo is visible in the top right corner. A small video inset shows a man speaking.

So, this is the website. In this website, you can see here we can, there is one side where we can say number of bits, number of bits could be 3, number of bits could be 2 or 4. So, let us say if number of bits are 3. So, then it could be represented using 0, 1, 2, 3 and then minus 4, minus 3,

minus 2, minus 1 and similarly using 4 bits. So, there are four ways of representing here which has been shown, let us quickly have a recap here.

(Refer Slide Time: 00:50)

The screenshot shows a presentation slide with a circular diagram of 4-bit binary representations for numbers 0 to 15. The interface includes 'Number of bits: 4', 'Encoding' options (Positive, Signed, 1s Complement, 2s Complement), and an NPTEL logo. A video inset shows a man speaking.

Number	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

These are the positive numbers, which we discussed first that 0, 1, 2 up to 15. So, this is the positive number circle.

(Refer Slide Time: 01:03)

The screenshot shows a presentation slide with a circular diagram of 4-bit binary representations for numbers 0 to 15. The interface includes 'Number of bits: 4', 'Encoding' options (Positive, Signed, 1s Complement, 2s Complement), and an NPTEL logo. A video inset shows a man speaking.

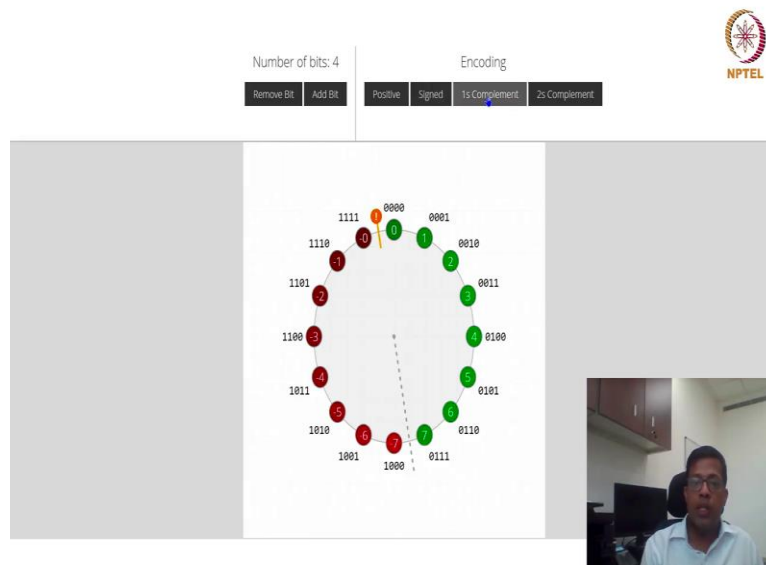
Number	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

If we are doing it in signed manner, then we see this discontinuity is happening at two places. So we are doing 0, 1, 2, 3, 4, 5, 6, 7 and after that, because this 100 is this is a sign notation, sign

magnitude, so basically 1 here means negative number and the rest of the 3 bits represent magnitude.

So, you see, there is a discontinuity from 7, we came here to 0 and similarly here, this is a negative number and if I add 1 to it, then it becomes 000, so there is a discontinuity here also. So essentially in a signed representation, there are 2 discontinuities and representation works like this.

(Refer Slide Time: 1:57)



In 1s complement, you again see there are two issues, one issue is that there is a discontinuity here. So, if I add 1 here, from 7, it became minus 7. This is a continuous thing, minus 7 adding plus 1 is minus 6, adding plus 1 is minus 5. So, that means going in a clockwise direction is addition. But here also this is not discontinuity, but this is a kind of an error that this is also 0, this is also 0, there are two 0s in the same number circle.

(Refer Slide Time: 02:38)

Number of bits: 4

Encoding

Remove Bit Add Bit Positive Signed 1s Complement 2s Complement

1111 0000 0001
1110 0010 0011
1101 0100 0101
1100 0110 0111
1011 1000 1001
1010 1010 1011
1001 1100 1101
1000 1110 1111

And this is how we can see our 2s complement. So, if you want to play a bit, you can play using this particular website and have some fun.

(Refer Slide Time: 02:50)

Number circle

Clockwise – addition
Anticlockwise – subtraction
Positive number
- u clockwise steps from 0
Negative number
- u steps anticlockwise from 0
u clockwise steps =
 $2^N - u$ anticlockwise u steps

<https://thesis.laszlokorte.de/demo/number-circle.html>

Digital Logic Design: Introduction. 44

So, let us get back to the presentation and so, now after understanding it from the number circle perspective, let us have some theoretical background or let us have some mathematical representation of this number circle or this particular type of number representation.

(Refer Slide Time: 03:06)

2's complement



- If u is +ve: $F(u) = |u|$
- If u is -ve : $F(u) = 2^N - |u| = \sim|u| + 1$
- Properties
 - Single zero
 - MSB represents sign
 - Negation rule
 - $F(-u) = 2^N - F(u)$
 - Range (-2^{N-1}) to $(2^{N-1} + 1)$

Digital Logic Design: Introduction.

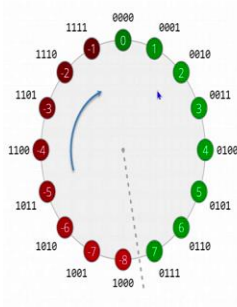
45



Number circle



Clockwise – addition
Anticlockwise – subtraction
Positive number
– u clockwise steps from 0
Negative number
– u steps anticlockwise from 0
 u clockwise steps =
 $2^N - u$ anticlockwise u steps



<https://thesis.laszlokorte.de/demo/number-circle.html>

Digital Logic Design: Introduction.

44



This particular number representation is called 2's complement. Why it is called 2's complement? We will see. So, here I can mathematically define it that if a number is u , then its magnitude is representing 2's complement, if number is negative, then 2's power n minus magnitude of u is actually 2's complement, because it is to 2's power n , so it is called 2's complement.

And since 2's power n , 2's power, we can also represent this using inversion or inverse or logical invert of u and then plus 1. So, this is this is an easier way to find out 2's complement. So, we can invert all the numbers and then add 1 that would be the 2's complement representation of any

negative number. So, as seen from our circle lines or numbers circle, we see couple of properties here.

One property is there is single 0, the other thing is, you can see that MSB, Most Significant Bit if it is 1, then number is negative, if Most Significant Bit is 0, then that number is positive. So, this is also a nice property. So, that means, if I want to see whether a number is a positive number or negative number, I need not to do any other calculation. I need not to do for example, subtract bias or do some other comparison I can directly look at the Most Significant Bit, if that bit is 1 then number is negative, otherwise, number is positive.

The other thing is that I can always find negative of 1 number using 2's power n minus of that number; so which essentially means that negative of a number, I can find out using this 2's power n minus that number, so that means, if I do further negation that the number will become positive. So, let us say if there is a number minus 13, I represented using this 2's complement notation, it will become plus 13 and then again, I do negation then again, it will become 2's power then again it will become negative number.


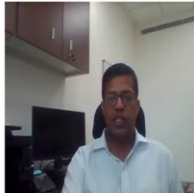
The other thing is range of the number would be minus 2's power n minus 1 to 2's power n minus 1 plus 1. I think there is a mistake here. So, this should have been a minus sign here, not positive sign. So for example, for 4 bit numbers, the range is from minus 2's power 3 to 2's power 3 minus 1. So, that means number ranges from minus 8 to 7, positive 7. So, this is the range we get from 2's complement.

(Refer Slide Time 06:37)

2's complement arithmetic

- Addition
$$F(u+v) = F(u) + F(v)$$

– Normal addition will work for both positive and negative numbers
- Subtraction
$$F(u - v) = F(u) + F(-v)$$
- Multiplication (assume no overflow)
$$F(uxv) = F(u) \times F(v)$$

Digital Logic Design-Introduction. 46

And the other thing, which can be proven easily that addition, subtraction, multiplication all can be done intuitively can be done using a standard binary addition method in 2's complement notation. So, if I want to add 2 numbers, I can take 2's complement of these 2's, these numbers can be represented in 2's complement notation and addition of those numbers using a binary arithmetic will give me correct results, although one of the number could be positive one of the number could be negative or both the numbers could be positive, both the numbers could be negative, it always gave me correct result using the simple binary addition method.

If I want to do subtraction, then the best way is that whatever number I want to subtract, I should take 2's complement of that number. So, I should represent that number as a negative number. Let us say that number was a negative number and I want to subtract that number. So taking 2's complement of that negative number will become the positive number.


So, let us say the number was minus 30 and I wanted to subtract this minus 30. So, after taking 2's complement of that minus 30, it will become plus 30. So that is how this negation rule is helping me that I can do this, the step would be first I need to take 2's complement of that number and then add it like a regular addition.

And interestingly, even if I multiply 2 numbers using 2's complement, then the result will always be correct. But here a caveat is that there is no overflow assume. So, that essentially means that if

I am adding 4 bit number, I am multiplying 4 bit number with another 4 bit number, the output should not be more than 4 bit, then the result would be correct. Otherwise, it may not be correct.

So, there are always ways to fix it, so that is why multiplication is also workable or is can be worked out using this 2's complement arithmetic. So, I can understand that it is because it is all theory or using formula, it does not sink into our mind. So, let us take some examples and see that how addition or multiplication will work.


(Refer Slide Time: 09:17)



Examples

2×-3 (using 4 bit) <pre> 0010 x1101 ----- 0010 0000 0010 0010 ----- 0011010 </pre>	$13 + (-3)$ <pre> 0000 1101 + 1111 1101 ----- 1 0000 1010 </pre>	$13 - 5 = 13 + (-5)$ <pre> 0000 1101 + 1111 1011 ----- 1 0000 1000 </pre>
---	--	---

Digital Logic Design Introduction.
47



So, let us take we will start with an addition. Let us say I want to add a 13 and with minus 3. So, first I will do is I will represent 13 in 2's complement notation, because 13 is a positive number, and here I am assuming 8 bit numbers, this method of 2's complement work for any number of bits, but we I am showing it using 8, 8 bits here. So, it is just an example.

So first, I represent 13 using 2's complement because it is a positive number I need not to do anything for sign bit and 13 I would represent as 1101 and I am (9:59) rest of the most significant bits as 0. Now, I want to represent minus 3. So, for representing minus 3, first I have to calculate 3 that means 1 1 and then or basically 6 0s and then 1 1 and then invert it, inverting it will become six 1s and then 0 0 and then I have to add plus 1. So, that will give me minus 3.

1111 1101. Now because I want to add these 2 numbers, so I am adding 1 plus 1 is 0 and there is a carry of 1. So, this because this is 0 0 carry would come directly 1 here, then again 1 plus 1 is 0 carry is 1, 1 carry plus 1 plus 1 will give me a result sum of 1 and carry is 1, because carry is 1

plus 1 will give me 0, again carry is generated 1 plus 1 0 again carry is generated, 1 plus 1 0, another carry is generated 1 plus 1 0, another carry is generated and we finally will have a carry here.

Now carry here, we have to ignore this carry and we can leave this carry, we can we can simply discard this carry without worrying about the results and the remaining 8 bit gives me correct result. So, minus 13 minus 3 essentially is 10 and that is what we can see from this number. Let us see another number. Let us say I want to, I want to subtract a number 13 minus 5. So again, what I will do is 13 and then take 2's complement of 5 and then add these 2 numbers.

So essentially, all subtraction would be converted into addition and there is no, so because the negative numbers would be represented in 2's complement form, everything is addition here. Now, in the same way because in the same way, if I will do this subtraction, then I would get a correct result here. So, 13 would be represented by 0000 1101 and 5 minus 5 would be represented if I will calculate. So, minus 5 is 1111 1011.

How do I got it, so, basically, 5 is 101. If I inverse it will become 1111 0 this is 1 and then 010, 010 and then I have to add plus 1 out of it, so it becomes 1011. So, this way, this minus 5 2's complement of 5 would be calculated and if I add both of them, then 1 plus 1 is 0, carry of 1 and this carry of 1, I add from to 1 then it is again 0 carry of 1 and this carry is added to 1, so output is 0 here.

Now 1 plus 1 plus 1, sum is 1 and carry is 1 and now carry would be propagated here. So, this will all be 000 and there would be 1 additional carry. Now let us see how will it work for multiplication. So here to make my life simpler, I am taking 4 bit numbers, so there will be only four partial products. So let us say I want to multiply 2 and minus 3, 2 is 0010 and minus 3 is 2's complement of 3 that means 1101.


So if I multiply using this, how do I multiply? So first bit 1, I will multiply with this multiplication, multiplicand. First bit of multiplier is multiplied with multiplicand 0010 and then I will shift, I will put cross here, we remember all our decimal multiplication tricks. So, because multiplier second bit of multiplier is 0, so we can simply say 0000 here. Now the third bit of multiplier is 1, so again, this 1 would be multiplied with 0010 and I leave two places.

That means 0010 and then 2 places are left blank and for the last bit of multiplier, it is 1 again here I will leave three places blank and then I will write the same multiplicand here 0010 and finally, to get a multiplier output, I will add these four partial products. If I add then 0 can be directly come here as 0 and this is 1.

And here in this row these are all 0s, so output is 0. In this row there is only 1, so output is 1. In this row there is only 1 so output is 1. So, similarly we can do this partial addition and because as we mentioned earlier, we are not going to consider overflow only because my input was 4 bit, this was also 4 bit, my output I have to consider only 4 bits. So, the output is going to be 1010.

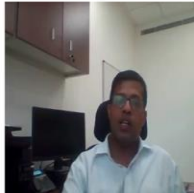
So, if I translate this 1010, it is going to be minus 6 in 2's complement notation. So, this essentially gives us a confidence that yes, using 2's complement, we can represent any numbers plus we can also do this this binary arithmetic addition and subtraction, multiplication using simple operations or whenever we are doing operations, we need not to worry about the sign weight or the underlying representation, under underlying representation of negative numbers, underlined representation, if it is 2's complement, it is always going to give us correct results.

(Refer Slide Time: 16:40)



Overflow and underflow

- Overflow/underflow for addition
 - If sign of both operands are same and result is of opposite sign or result is 0 => overflow
 - If sign of both operands are different => overflow can never occur
- Converting N bit number to M bit number
 - Sign extension
 - 1101 in 4 bit is 1111 1101 in 8 bit

Digital Logic Design: Introduction.48

So, this there is also one thing again, we have to note down that when my results are going to be wrong, that also we should be clear. So, whenever we are going to cross that breakpoint or discontinuity of my number circle, then I am either going in overflow condition or going under

flow conditions. So, this unless we know this overflow, underflow condition, we never know that my results are correct or not.

So, if it is overflow that means number is outside the range of numbers that can be represented using that many number of bits. So, for example, if it is 4 bits, and I can only represent from minus 8 to 7 if number is if my result, addition result is more than 7, I cannot even represent using 4 bits. So, that is why whatever results I am getting, it may be wrong, it will look wrong, it is not actually wrong, but it may look wrong, because this number I cannot represent using that many number of bytes.

So, how to identify this condition that is very important to know and important to practice or see also, whether I am not reaching an overflow condition. So, overflow condition is a very can be laid out or can be put in a very simple words, if addition of my two positive number is resulting in a negative number. That means there is an overflow. So for example, in a number circle, I am adding 5 plus 4, 5 plus 4 and I am representing only using 4 bits.

So, 5 plus 4 will result in 9 and using 4 bits, I can only represent up to minus 8 to 7. So, 7 is the maximum largest positive number that I can represent, but my output is 9. So that means whatever output my binary arithmetic will show will not be correct. I have to raise that there is an overflow, this result is not correct.

So, to check this condition, if signing of, sign of both the operands are same and I am doing addition and result is either of opposite sign or result is 0 that means it is an overflow. So similarly for negative numbers adding negative numbers, so minus 3 minus 7, it is going to be an underflow. So, this is how we can represent overflow and underflow and we also can clear clarify one thing that if the signs are different, one number is positive, another number is negative and we are adding these two numbers, there can never be overflow or underflow condition.

So, the number which is larger is going to be win, going to win. So, for example, I am adding plus number plus 5 and I am adding minus 8, plus 5 and minus 8 I am adding. So, whatever number is larger in magnitude, your number will become of that particular sign, but it is not going to be overflow or underflow. So, this simple condition check can tell me what is the overflow or underflow condition.

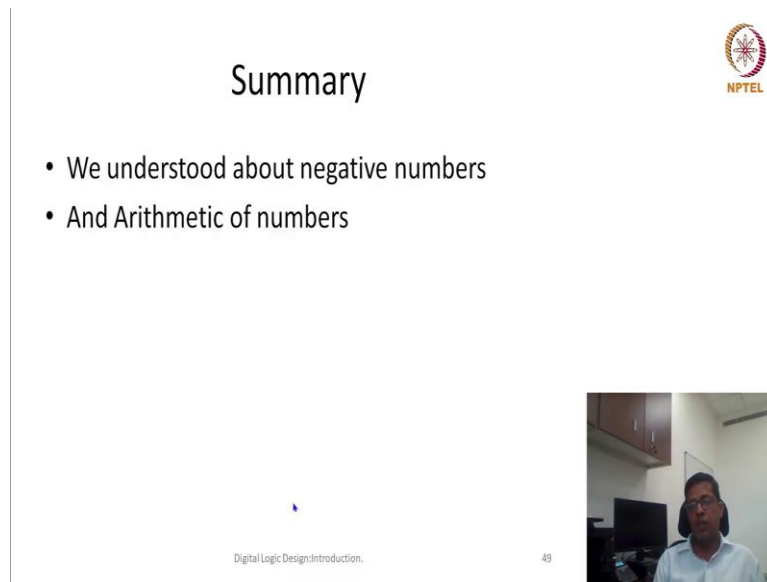
One more topic or one more point which I would like to touch upon here that a negative number, if I have to convert N bit number to M bit number, it is going to be slightly different here. So, let us say I want to convert a 4 bit number into an 8 bit number. If number is positive, it is going to be same, but if number is negative, then it can lead to a different representation, it can lead to a wrong representation that is why we call it as a sign extended number.

So, in a sign extended number, let us say 1101 is a 4 bit number which essentially represents 1, it represents minus 2. So, this minus 2 if I want to represent in 8 bit then I need to remember that because it is minus 2, so its sign bit has to be 1 and minus 2 would in 8 bit would be 2^8 minus 2.

So, that means that this either I need to recalculate this number or the easiest or a trickiest method would be that whatever is the sign bit here is that if I extend that sign bit to the larger like rest of the MSB numbers, let us say M bit is larger and N bit is smaller, then M minus N bit I can extend as a sign bit, then the number representation would be correct.

And this sign extension will work even if my sign bit is 0, because then I would be extending 0s all the way, if sign bit is 1 that means, I am extending it for the negative numbers and it would give me an easy method of converting from 4 bit number to 8 bit number or 8 bit number to 32 bit numbers, the representation is fast and otherwise, the ideally ideal method would be I had to do for example, I need to convert from 8 bit number to 32 bit number I need to subtract that number from 2^32 minus the magnitude part. So, this sign extension trick can make it a little fast.

(Refer Slide Time: 22:56)



Summary

- We understood about negative numbers
- And Arithmetic of numbers

Digital Logic Design: Introduction. 49

So, here I would like to close and I can summarize that overall in last half an hour, 45 minutes we have understood what could be various ways of representing negative numbers. So, one thing you may, you may be thinking at this point that you are not going to use 1's complement, sign magnitude, 2's complement, why we are studying all those things or why was the only motivation to study those things were to find flaws in all those numbers?

We can, here I would say that it is not like that, they each of them has its own purpose or its own qualities. Sometimes we do use sign magnitude number also, sometimes we do use offset or biased method of representing negative numbers as well, we will see in some of the later classes that how do we use them.

The other thing which we have learned in this class is how to do arithmetic operations. So, this class would remain incomplete unless we do practice lot of questions or we do some of these things ourselves. So, I will be posting a tutorial after this class so that you can work on those questions. Some of those questions I would put as old questions and some of the questions which you will try to solve yourself. Thank you very much.