**Computer Architecture**
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 11
Binary Arithmetic, ALU Design

Today we are moving to a different topic. Our discussion is so far was on instruction set architecture and now we will start discussing about micro architecture which means how instructions which we have studied so far how they are realized in hardware. We will begin by looking at how basic operations are done in hardware, we will talk of arithmetic operators, logical comparison and so on and eventually we will see how the heart of the processor called ALU arithmetic logic unit is built.

So, we have so far seen what instructions are, what their usefulness is, what is the purpose of instructions and what is the functionality they achieve. We have also talked about how performance is defined and to some extent we have seen the relationship between performance and instructions. So when we continue with this we develop and understanding micro architecture and relationship between instruction set of architecture and micro architecture and idea of performance will be even more clearer.

(Refer Slide Time: 01:38 min)
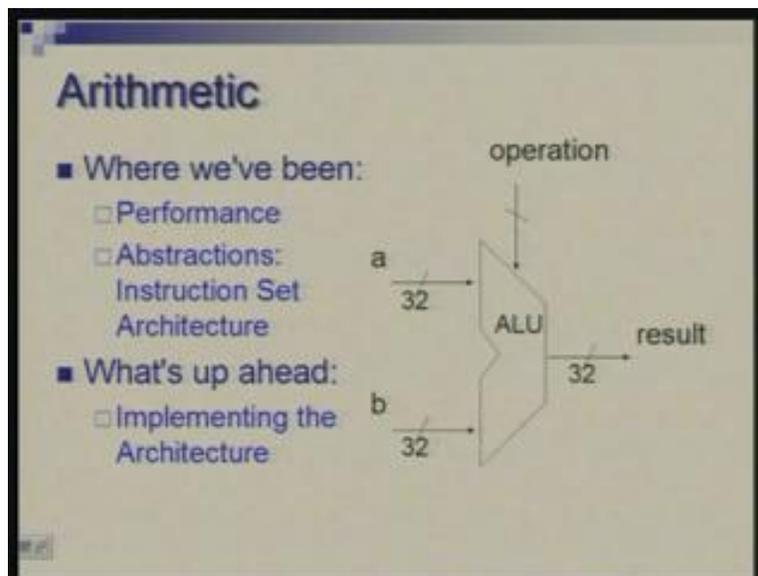
So essentially what we are aiming at is designing something like this an ALU which typically works on two operands; in our case for MIPS processor they will be 32-bit operands and produces a result under control of some signals which are determined from the instruction. So depending upon which instruction you are trying to execute this unit will perform some operation, it might perform addition, subtraction, multiplication, division or do some comparison or do some logical operation.

So, in next few lectures the focus would be design of this. So, talking of this topic, this is the sequence of sub topics you will look at. Today I will talk about binary arithmetic and design of ALU in a very simple situation then I will try to include idea of overflow when a result exceeds the limit what do we do. We will separately talk of multiplier and divider design which are more complex operation than simple add, subtract, AND, OR. We will look at some techniques to speed up the key operation of addition or subtraction then we will move on to non-integer operations floating point how such numbers are represented and what kind of operations are performed over these and finally the hardware to carry out these operations.

(Refer Slide Time: 03:00 min)
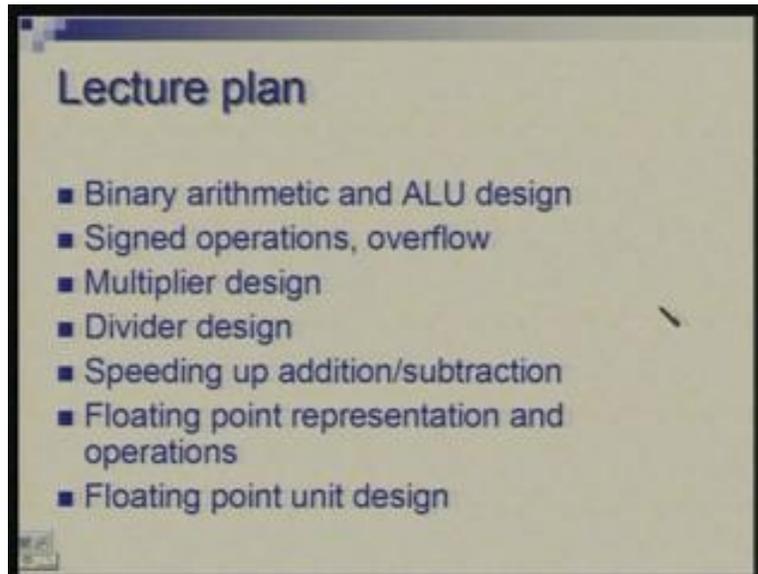
So today, in particular, starting with binary representation, we will see how addition and subtraction are done and how the circuit is built. We will talk of comparison little bit, logical operation and finally come up with ALU design for some of the MIPS instructions.

(Refer Slide Time: 04:01 min)
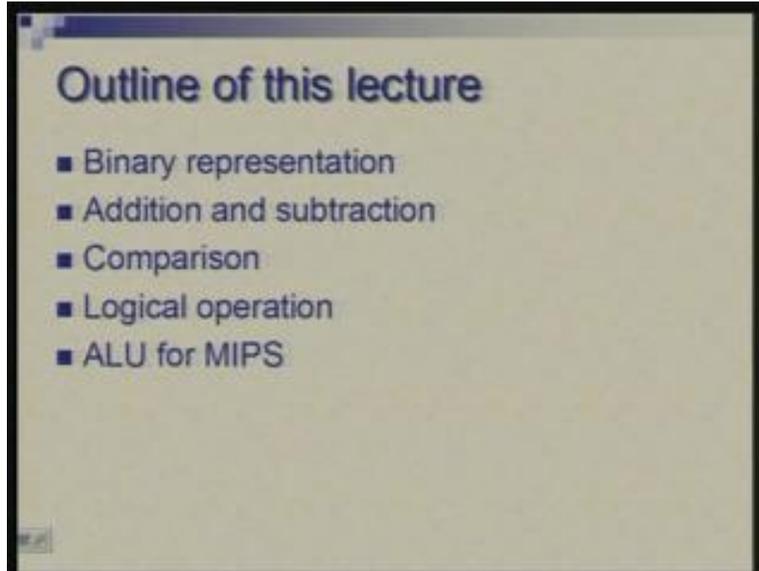
I presume that you are familiar with binary representation to some extent. We will just refresh it so that you are comfortable and you can quickly follow what we discuss later on. so, as you know, inside a computer everything is represented in bits; each bit represents two states 0, 1 and there is no inherent meaning is assigned to a bit, a bit could represent different things, it depends upon the quaintest which you are looking at a string of bits; it could represent an integer, it could represent a fractional number, it could represent a piece of text, it could represent some logical value or some symbols in encoded form. So the context in which you look at set of bits, decides what actually they mean so you have to interpret them in one of the many difference ways depending upon when and in what circumstances you are looking at those.

(Refer Slide Time: 5:39)
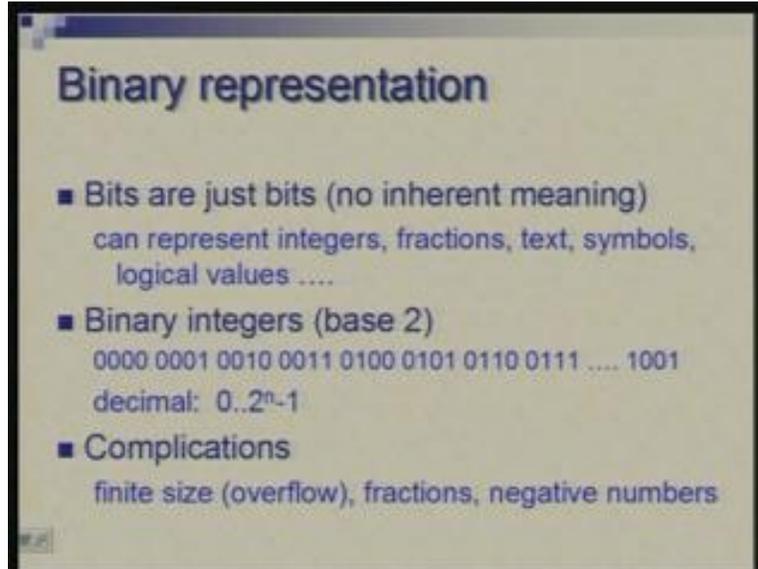
So, to begin with, let us talk of integers represented as strings of bits. Essentially from mathematical point of view we are talking of binary numbers which are essentially with radix 2 or base 2. So a string of n bits can represent a number which will correspond to a number in the range 0 to 2 raised to the power n minus 1 because with n bits you can represent 2 raised to the power n pattern you get a range of numbers 0 to 2 raised to the power n minus 1 if you considering only positive numbers or nonzero numbers I should say. But with this simple idea we need to talk of a few more complications.

For example, in mathematics you talk of integers ranging from minus infinity to plus infinity; here we are talking of a finite range so it is not a strict one-to-one correspondence between integer as we understand in mathematics and integers we are trying to represent here. So there is a question of overflow which we are going to discuss later. we also have to worry about sign of the numbers how to represent positive and negative numbers, number of choices exist and we should understand the implications of those choices, how do you represent fractions and that makes the thing little more complex than simply looking at a string of n bits as a pattern representing numbers from 0 to 2 raised to power n minus 1.

(Refer Slide Time: 07:11 min)

So, starting with the negative numbers there are several representation particularly 3 which has been used commonly; one is called sign magnitude where if you have n bits you keep one bit for the sign and remaining n minus 1 to represent magnitude of the number. So, the first bit typically could represent a sign, typically 0 corresponds to a positive sign and 1 corresponds to a negative sign. So in a three bits situation these eight patterns would corresponds to the integer shown here. So the magnitude can range obviously from 0 to 3 with two bits kept for the magnitude and therefore the range is 0 to 3 on the positive side and 0 to minus 3 on the negative side. interestingly you would notice that there is the plus 0 and minus 0 of course with mathematically mean the same thing but all the same zero magnitude can be prefixed with the positive or negative sign and there are two representations of the same number.

1's compliment is the notation which says that if the number is positive you represent as if you were handling a positive number or without any change but if it is negative then you flip all the bits turn 0s to 1 and 1's to 0s. so if you take 001 for example which represents plus 1 you make it 110 like this it becomes minus 1. So once again the range of number you can have is same 0 to 3 on positive side and minus 0 to minus 3. Once again 0 gets two representations.
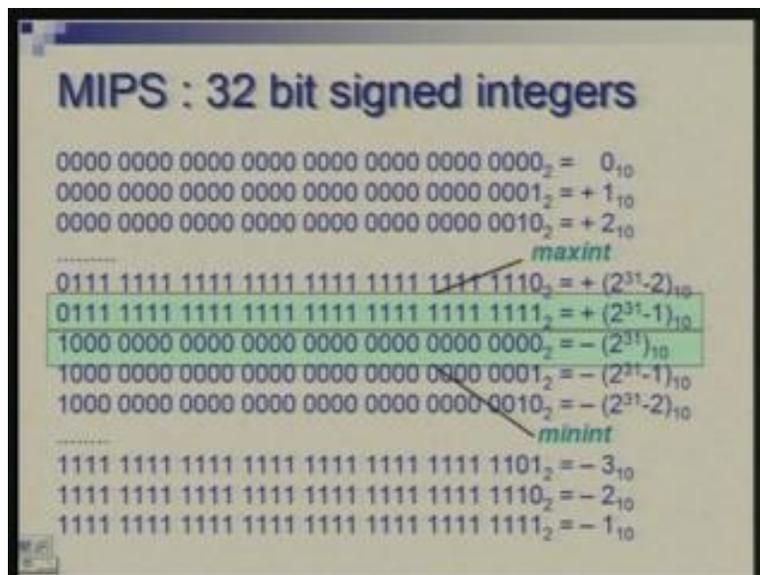
The third column here shows what is called 2's compliment representation. Here positive numbers coincide with what you have seen in other cases. The representation of positive numbers are identical in all cases but negative numbers here or with a one offset from this. so, for example, the largest number which you see here is minus 3 but here we get minus 4; minus 2 becomes minus 3 and minus 1 becomes minus 2 and minus 0 we have got rid of that so it is minus 1. So essentially this representation removes this ambiguity, each number has a unique representation that is a plus point we are getting although the negative points seems to be that the range is unbalanced. Range is unbalanced in sense that on the negative side you can go up to minus 4 but on the positive side you can go up

to plus 3. So in general whatever be the number of bits the range on negative side will be one more than what you have on the positive side. So that is a…. so to say a negative point of this representation. but more significantly an important positive point positive feature here is that it actually eases the arithmetic operations and makes the hardware somewhat simpler.

Now, what is the definition of this representation?
This is basically, for negative number it is in magnitude one more than 1's compliment. 1's compliment is defined as flipping all the bits and if you take say number like minus 2 its 1's compliment is 101 obtained by inverting all the bits and if you add 1 to that you get 110 you get minus 2 so 1's compliment plus 1.

(Refer Slide Time: 11:40 min)

MIPS : 32 bit signed integers

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = +1_{10}$$
$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = +2_{10}$$
maxint
$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = +(2^{31}-2)_{10}$$
$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = +(2^{31}-1)_{10}$$
$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -(2^{31})_{10}$$
$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -(2^{31}-1)_{10}$$
$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -(2^{31}-2)_{10}$$
minint
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$$

This shows the range of number you will get in a 32-bit word as we have in MIPS and following 2's compliment representation. So all zeros is 0, on the side I am writing their equivalent values their decimals base 10. So this is 0 then 1, 2 and so on; the largest number is 0 followed by all 1 that is the largest possible number and it corresponds to 2 raised power 33, 2 raised to power 31 minus 1. This 31 is one less than the number of bits you have.

The smallest negative number is all 1's. so basically if you take 1's compliment of plus 1you will get all 1's with a 0 at the end and add 1 to get this (Refer Slide Time: 12:40) so this represents minus 1 and as you go this way the magnitude is increasing, the largest negative number you will get is 1 followed by all 0s and is minus 2 raised to power 31. So these two numbers are max int maximum positive integer value and min int which is the most negative integer value. So approximately 2 raised to power 31would be about close to…. over 2 million over 2 billion sorry.

(Refer Slide Time: 13:33)

MIPS : 32 bit signed integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = +1_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = +2_{10}$
....   maxint
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = +(2^{31}-2)_{10}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = +(2^{31}-1)_{10}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -(2^{31})_{10}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -(2^{31}-1)_{10}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -(2^{31}-2)_{10}$
                                                       minint
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$

How do you perform addition and subtraction with binary numbers?
Again I presume some familiarity so I will go over this quickly.

(Refer Slide Time: 13:50)

Addition & Subtraction

- Just like in primary school (carry/borrow 1s)

| 0011 [3] | 0101 [5] | 0110 [6] |
| + 0010 [2] | - 0010 [2] | - 0101 [5] |

Basically the method is there is nothing new in the method; you add or subtract exactly the way you do with paper and pencil as you learnt in school, the only thing you have to represent the only thing you have to remember is that numbers are in base 2. So when you add 1 and 1 for example you get a 0 and a carry. So let us look at a few examples. This is adding positive 3 and positive 2, subtracting 2 from 5 subtracting 5 from 6 so you get 5 in first case and you would expect 3 in the second case. Of course now this is

showing as if bits are generated from left to right but when you work it out you actually have to work from right to left because carry a borrow will flow from right to left.

(Refer Slide Time: 15:10 min)

Now I took these examples carefully so that the result is within the range which is permissible by 4 bits we are having at hand. What happens when the number is larger than the word can accommodate, that we will discuss later.

Now, if you bring in the sign if you bring in negative numbers, what we did was adding and subtracting positive numbers. If you have negative numbers and you are representing in 2's compliment form then again it does not complicate the life at all. So when you are adding or subtracting you continue to think of them as positive integers, do not worry about the fact that they are sign 2's compliment number; perform addition or subtraction as you would do without sign. And if your result is within the range then you would have got the correct answer. So the key is that you perform add or subtract operation ignoring the sign. So here are few examples; we were trying to do 7 minus 2, we expect….. basically we are adding minus 2 to 7, so we should get an answer 5 and there is a subtraction of minus 2 from 5 so you should get plus 7 as the answer.

(Refer Slide Time: 16:41)
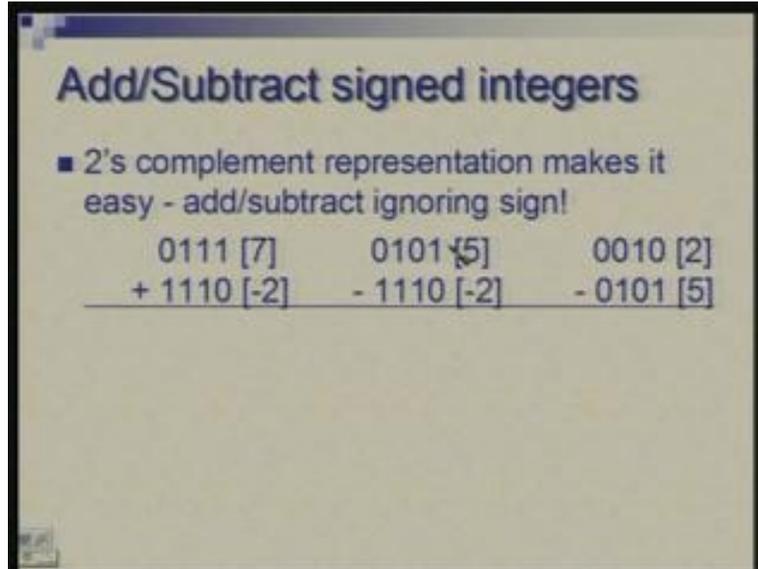
Subtracting 5 from 2 should give a negative answer which is expected to be minus 3. So just quickly make sure that these representations these numbers which I have written is correct, this is representation of minus 2.You can do either way. If you want to find out what is the representation of minus 2 you could take positive 2 take 1's compliment and add. If you are given a negative number like this and you want to find out what it represents again do the same thing. Take its 1's compliment or invert all bits and add 1. So if you compliment you will get triple 0 and 1 and add 1 you get 2. So basically this is representation of minus 2. And you would notice that you get correctly plus 5 when you perform addition. Alright. I have just added without knowing that it is negative or positive, you simply add these are positive numbers there is a carry going out just ignore it. If the result is within the range then you will get correct answer so same thing is happening here, always there is a carry flowing out. We have just thrown it. What you are getting here is plus 7 as we expected.

In this case let us do it step by step so you subtract 0101 from 0010 so here you get a 1 there is a borrow that it is again this one you get a 0; here again there is a borrow you get a 1, there is again a borrow you get a 1 here. So now there is ultimately a borrow from the left side which again like carry we are neglecting. So you have here 1101 what does this represent; you find its compliment 0010 and add 1 to it so you will get 0011 which is 3 so this is the representation of minus 3.

(Refer Slide Time: 19:05)

So these examples just illustrate the point which I mentioned earlier. Question is why does it happen, why is it you are able to ignore the fact that it is a sign. The answer lies in the fact that 2's compliment representation is actually nothing but 2 raised to the power n minus X. So, if you are trying to represent minus X let us X is the magnitude of a number and you are talking about minus X a negative number a negative number with magnitude X so its 2's compliment representation is nothing but 2 raised to the power n minus X so you take this large positive number, X will be always less than 2 n so this positive number minus this positive number (Refer Slide Time: 20:02) could be thought of the positive number. So if you had unsigned number systems representation of this positive number in that is what you are representing minus X as. Therefore, let us look at some example.

(Refer Slide Time: 20:18 min)

Suppose we want to see how we represent minus 3 so it is 2 raised to the power 4 we are talking of 4 bits so of course 2 raised to the power 4 is a 5-bit number but it is just something you are doing on the paper and pencil, out of this you subtract 0010 and what you get is 1101. So this is the same as complimenting these bits getting 1100 and adding 1 because going back to this one 2 raised to the power n can be written as 2 raised to the power n minus 1 plus 1. So 2 raised to the power n minus 1 is....

 (let me write it here) 2 raised to the power n minus 1 is all 1's.

(Refer Slide Time: 20:55 min)

You subtract a binary number from all 1's basically each bit will get flipped and then you are adding a 1 later on. So this just shows the relationship between two ways of looking at it.

(Refer Slide Time: 21:40 min)

So now what we are doing when we are adding and subtracting 2's compliment number is that minus negative number are bringing additional 2 raised to the power n into the picture. So suppose in fact if you have several numbers X, Y, Z, W four numbers to be added some of them are negative so with the negative number we are carrying an extra 2 raised to the power n. So, if you are adding a series of numbers some positive some

negative we would be carrying a few extra 2 raised to the power n which will all go out as superfluous carry on the left side and the result we get is the correct value.

(Refer Slide Time: 23:10 min)

Two's complement representation

- Represent "- 3"

$$10000 \ [2^4] \quad \text{alternatively} \quad 1100 \ [\text{invert 3}]$$
$$- \ \ 0011 \ [3] \qquad\qquad\qquad + \ 0001 \ [1]$$
$$\overline{\quad 1101 \ [-3] \qquad\qquad\qquad\quad 1101 \ [-3]}$$

- Negating a two's complement number (+ve or -ve): invert all bits and add 1
  - remember: "negate" and "invert" are quite different!
- Subtraction using addition: X-Y = X+Y'+1

Now this representation, this way of relating negative numbers to their representation bring us to an easy way of doing subtraction. Suppose you want to carry subtraction of X and Y you want to do X minus Y, Y you can look up on as 1's compliment of Y and sorry minus Y can be looked upon as 1's compliment of Y together with a 1. So basically this subtraction can be looked upon as addition of X and minus Y and minus Y is this. There is one additional factor you need to keep in mind that there are some instructions for example add immediate which require one operand to be 16-bits and one as 32-bits so how do you perform addition or subtraction with one large integer and one small integer.

The hardware which is there inside a processor would be common for add instruction or add immediate instruction. The addition is….. the adder is capable of adding two 32-bit numbers. So basically what is done is the shorter integers 16-bit integers are actually re-represented in 32-bits. So how do you do this conversion? The most significant bit which is actually the sign bit is repeated to fill up those additional sixteen points addition of sixteen places. So if you have positive numbers like 0010 in 4 bits then in 8 bits they get represented as four zeros followed by 0010. But if you have negative numbers it is this 1 (Refer Slide Time: 25:18) which will get extended 1111 then 1010, because this for example what is this number its 1's compliment is 0101 which is 5 and if you add 1 you get 6 so it is minus 6. Now minus 6 when represented as a 4-bit negative number gets as this. If the same minus 6 we had represented as 8-bit negative number we would got all these positions filled with 1's.

(Refer Slide Time: 25:50)
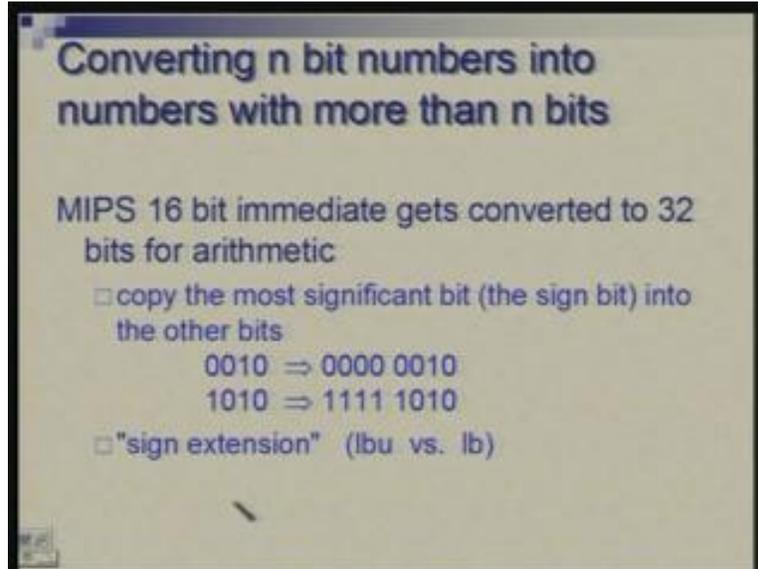
So in simple terms we will look upon this conversion as a sign extension. The sign bit is extended to fill up the additional bit spaces which you have. So, now because of this some instructions like load byte which have only filling up part of the word we have talked about load word which gets one word or 4 bytes from memory and fills up a register. There is also some instruction load byte, there is also one load half word so load byte picks up 1 byte from memory and places in a register.

Now the question is what happens to remaining 3 byte position or remaining 24 bits of the register. If you are looking upon this byte as a sign byte then they should be filled up with the sign. If you are looking at this as unsigned byte they get filled up with signed bit. So there are actually two different instructions load byte and load byte unsigned lbu. So lbu will fill up the remaining 24 bits with 0 and lb fills with the sign of the byte which you have got. So the bit number if you are numbering from 0 to 31 then bit number 7 which is a sign bit replicates and fills up all the positions.

Now let us move towards a circuit design for performing addition. The method is….. the simplest method is to follow what we do on paper and pencil.

(Refer Slide Time: 27:34)
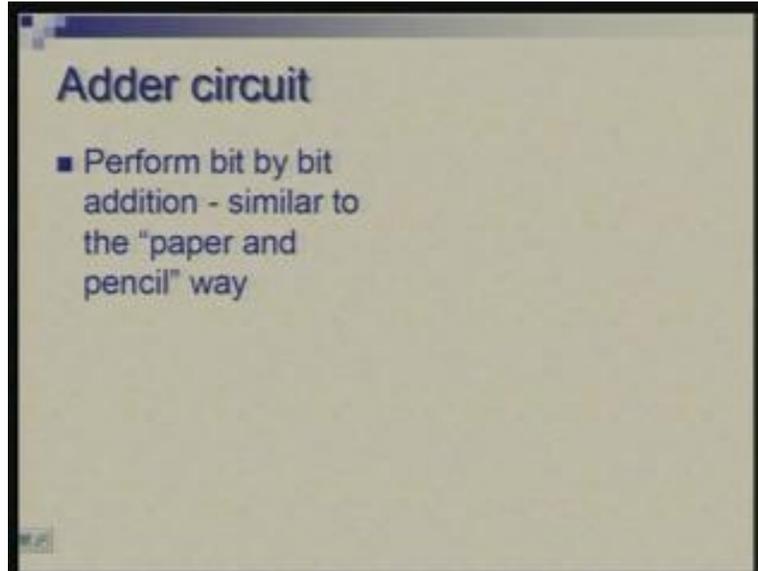
Basically we move from right to left and perform addition bit by bit. So if you know how to perform addition of 1 bit that means at 1 bit position adding 2 bits and also carry coming from right side you can just repeat the same thing for 32 bits. So each bit let us say we are talking of i'th bit of two operands a and b so a i and b i represent the i'th bit of two numbers a and b; Si represents i'th bit of sum; Ci represents the carry coming into this position from right side and Ci plus 1 is a carry going to the left side. So this is a unit this is the module which performs 1 bit addition and a circuit to add two 32-bit numbers nothing but a cascade or an array of thirty two such units. So the carry signals are those which connect one unit to the other unit and form a chain. Initial value of course is 0 which is the initial carry for bit number 0. I am assuming that bit 0 is the least significant bit and bit 31 is the most significant bit.
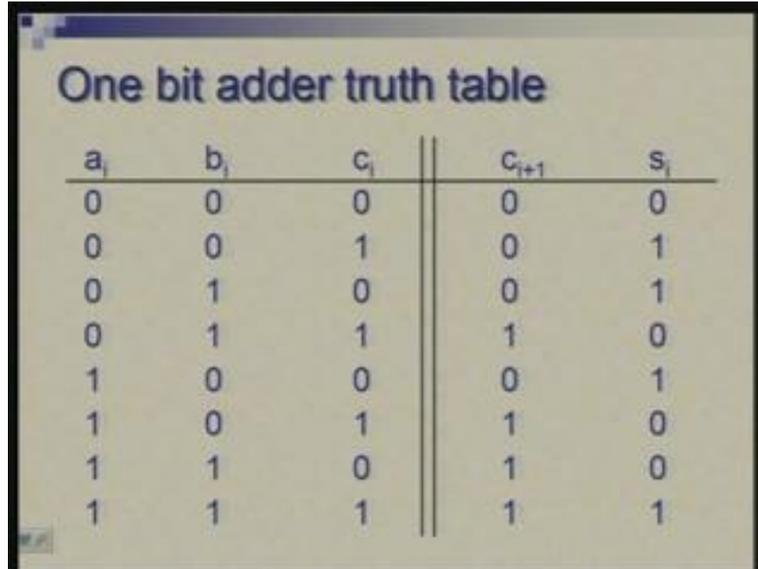
(Refer Slide Time: 28:35 min)
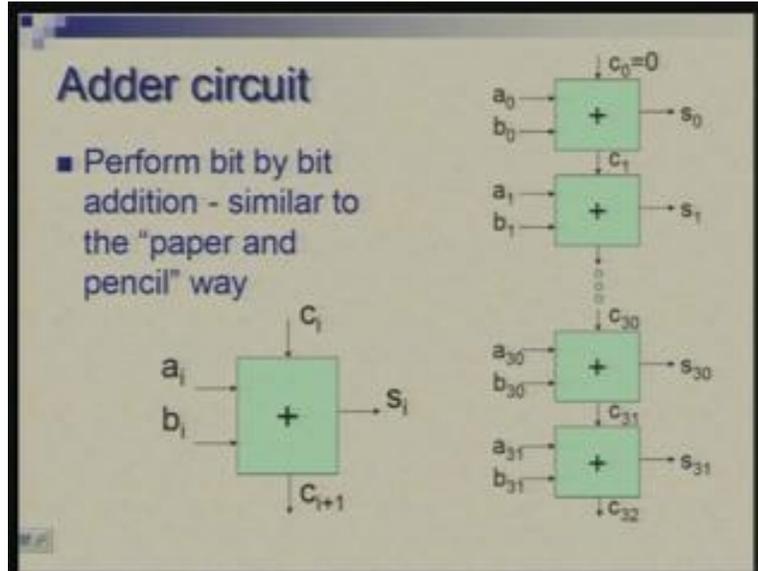
Now, following 2's compliment notation I do not need to worry whether a and b are signed numbers or unsigned numbers; whichever they are, I simply pass them through a circuit like this which to begin with is designed for unsigned numbers.

Now what is this 1-bit adder module?
It can be designed by trying to look at various possible inputs it has to cater for which can be listed in the form of a truth table. This truth table defines, row by row, what happens for each combination of inputs so we have these columns labeled by the inputs a_i, b_i and c_i and here are two columns labeled by the outputs c_i plus 1 and s_i so we have these three inputs and eight different combinations which you can form for these three inputs 0 0 0; 0 0 1 and so on and finally 1 1 1. For each of these we need to exhaustively define what the outputs are. So what we can do is we can find out how many 1's are there, if this is the case when there is no 1 so both sum and carry are 0. In this case and in this case (Refer Slide Time: 30:36) this, this and this these are three cases where you have a single 1 so sum is 1 and carry is 0 like this, like this or like this these three.
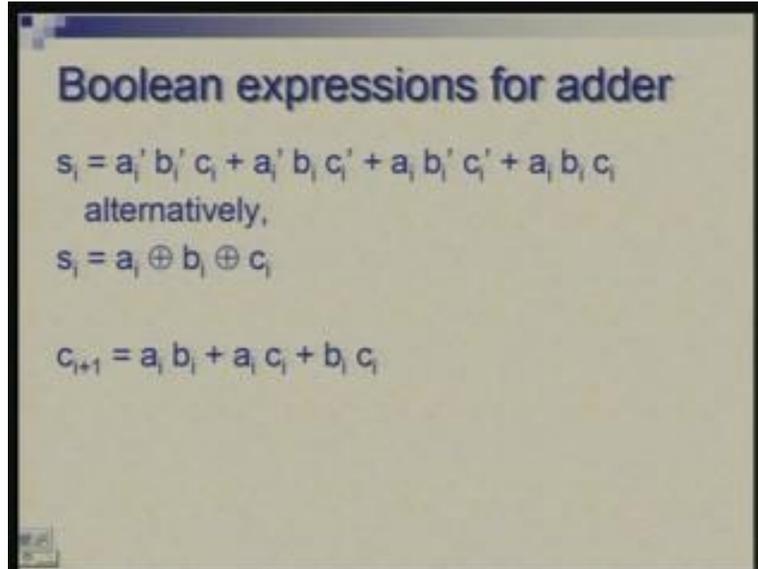
(Refer Slide Time: 29:25 min)

One bit adder truth table

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Then there are some case where you have two 1's. For example, this, here you have two 1's, here you have two 1's, here you have two 1's so basically two 1's make it equivalent of binary 2 which means sum is 0 and there is a carry; same way here, same way here and this is the unique case where you have all three 1's which means number 3 and sum is 0, sum is 1, carry is also 1.

Now this describes the relationship between input and output which can be captured by Boolean equations and Boolean equations then define what circuit you require to implement this. So, as far is sum is concerned it is those four combinations of a, a_i, b_i and c_i where we had sum is 1 so this one where you have a single 1, a single 1, a single 1 and all three 1's (Refer Slide Time: 32:00).

(Refer Slide Time: 32:02)

Boolean expressions for adder

$$s_i = a_i' b_i' c_i + a_i' b_i c_i' + a_i b_i' c_i' + a_i b_i c_i$$

alternatively,

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

Another way of looking at this is an exclusive OR of a_i , b_i and c_i which means that if number of 1's in is odd you have a 1, if number of 1's is even you have a 0 here.

Carry and other hand is 1 when you have two or more inputs as 1 so here the other they could be a_i and b_i , a_i or c_i and b_i these two or these two or these two. In the case of all three being 1's is actually covered by this. you can actually if you are familiar with Karnaugh Map if you have done this in digital electronics or you might be doing it you would know how to get this systematically by drawing a Karnaugh Map and finding a minimal representation in a sum of product form.

So now, once you have got Boolean expressions representing output you can straight away come up with a circuit design which will do this. So you have AND gates and OR gates, you connect them exactly as dictated by this to get the design. I will not go into that.

Refer Slide Time: 33:25 min)
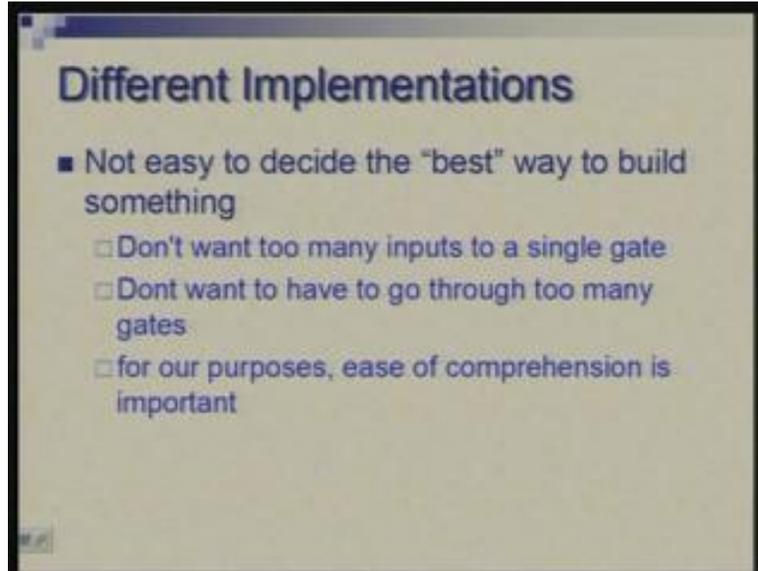
One thing which I would like to point out here is that the number of possibilities is large. There could be several ways of writing these Boolean equations and therefore you can have different circuits. So, for example, in case of some (I showed two) and there are many more ways of representing it depending upon which gates you want to use and there are also variations in different representation in terms of whether you have two input gates or three input gates; if you have terms like a, b and c that means you are talking of AND of 3. So question is you want to work with two input gates or two input and three input gates or whatever your choices are what is the fan-in of a gate you want to limit to; the other issue may be that you have stages of logic; AND OR representation means that you have two stage logic; you have AND terms formed by AND gates and then you are ORing all those. So (more stage is) more gates a signal has to go through more would be the delay. So implication of all these choices is different speeds, different costs, may be different power consumption so that itself is a more complex issue which in course like digital hardware design you will see it more closely more deeply. Therefore, for our purpose we will often try to use circuits which are easy to comprehend, easy to arrive at and comprehend.

Now we have seen how to build an adder, how do we build a subtractor. So basically we are going to use this formula that X minus Y can be represented as an addition of X and Y's compliment and then you can add one more. So now please note that X and Y themselves could be signed numbers. This formula is not just for X and Y as two positive integers; X could be positive or negative; Y could be positive or negative so this will hold.
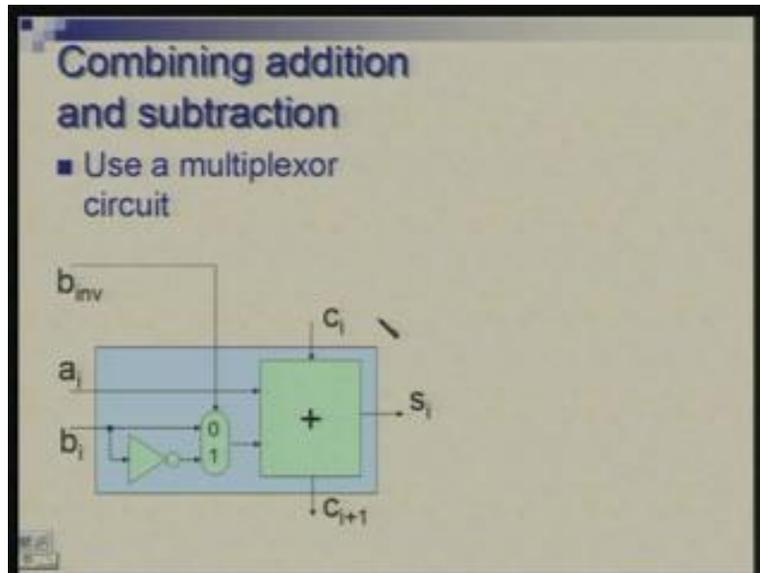
(Refer Slide Time: 35:46)
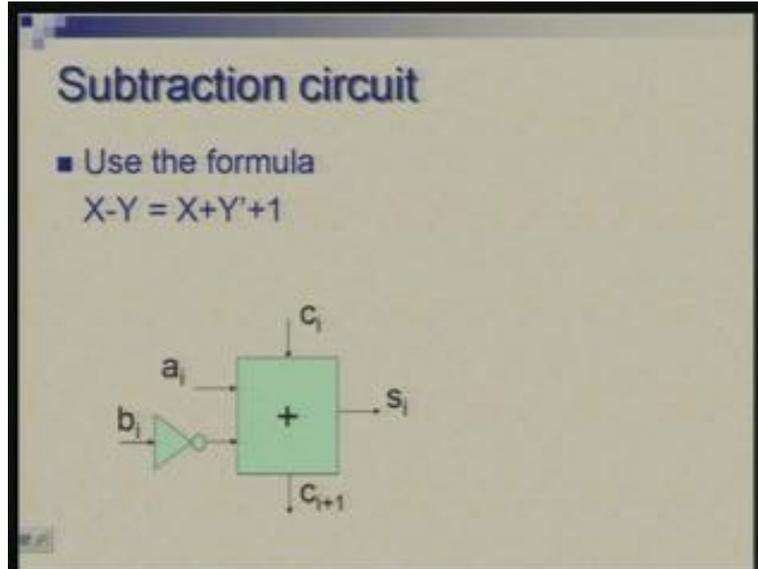
So to get 1's compliment of Y you simply need to put inverter for the second operand so now we had a and b as our operands so b_i is inverted before making it as input to the adder so that takes care of doing X plus Y prime or adding 1's compliment of Y to X. and when you put these modules together to form a 32-bit subtractor; this plus 1 (Refer Slide Time: 36:29) can be taken care by making the initial carry as 1 which is normally 0 in case of an addition. So these are two changes. Each b-bit is inverted and the initial carry is made one.
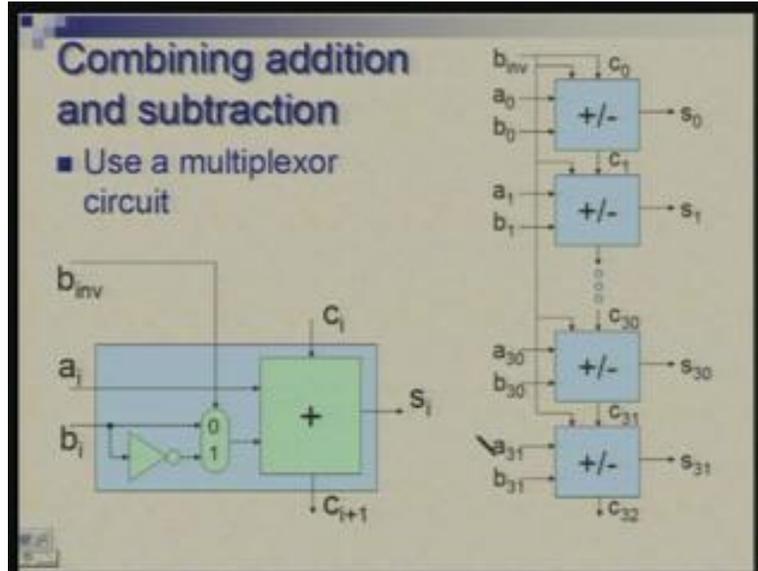
(Refer Slide Time: 36:30 min)
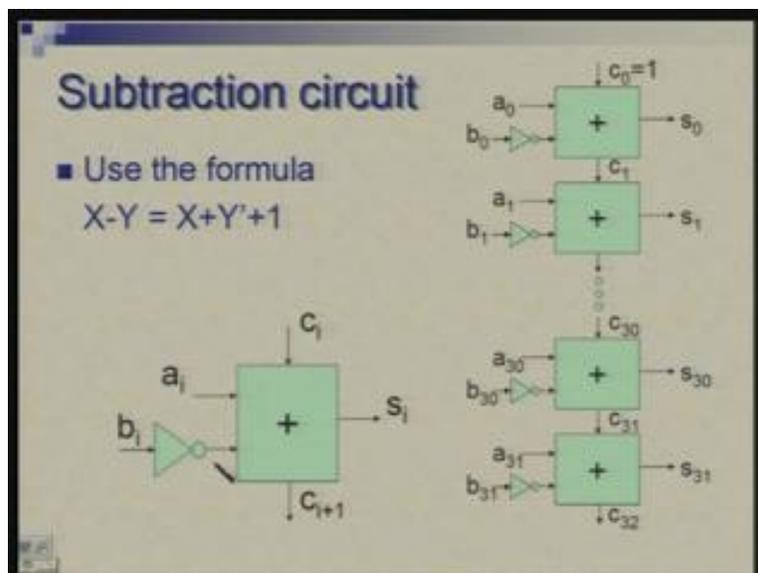
Now with these two circuits for adder and subtractor we can see easily that they can be combined. There is a slight difference; we can combine and come up with a single circuit by using a multiplexor. Multiplex, what multiplexor does is it selects one of the many inputs it has. So in our case the choice is between taking the bi bits directly or through the inverter. If we can put multiplexor to choose one of those two then we have a single circuit which will do addition or subtraction depending upon how we control the multiplexor. So we have the same adder, the bi input can go either directly or through the inverter and this multiplexor (Refer Slide Time: 37:45) is choosing this. It requires a control signal or control input which we call as b inv standing for whether b should be inverted or not so it is controlling. If you give a 0 here the multiplexor will select this input which is labeled with 0 and the circuit will do addition. If you put it as 1 then this inverted input will be taken and circuit will do subtraction.

(Refer Slide Time: 38:19)

So putting these boxes (Refer Slide Time: 38:21) I am not showing the details now. Each of these blue boxes is repeated here; I am calling it adder or subtractor. The carry input also needs to be controlled and it so turns out that you can use the same signal b invso when b inv this control signal is 0 the initial carry is 0 and the bits of number b go without inversion. When this input is 1 initial carry becomes 1 and bits of b (b_o, b_1 etc) go via inverter so this control signal is going to all these modules.

(Refer Slide Time: 38:21 min)

Now these are two arithmetic operations, we have seen their implementation, we have not looked at multiply and divide yet, let us move towards logical operations AND and OR.

(Refer Slide Time: 39:27)
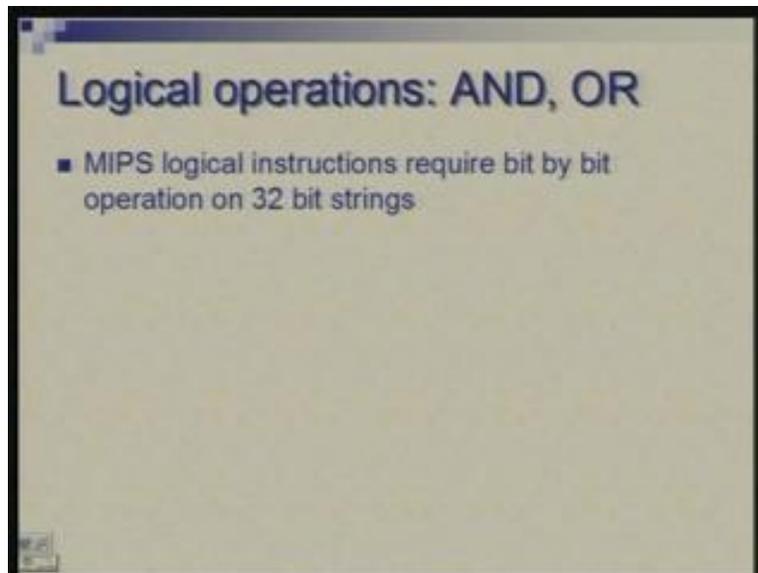
Now the instruction which we have in MIPS AND instruction OR instruction what they do is they perform logical operation bit by bit on 32-bit vectors. For example, if you have one register containing this and another register containing this and you perform AND operation you will get this (Refer Slide Time: 39:51) all that you have done is we have performed bit by bit AND operation so only when both are 1 you get a 1 otherwise you

have 0. Similarly, OR operation over the same two numbers will give you this. Wherever you have either them being 1 you get a 1 and only when both are 0 you get a 0.

(Refer Slide Time: 40:08 min)

So the circuit for doing this very simple; there is no relationship between what you are doing for bit i and what you are doing for bit j. So, for each bit position you have an AND gate so i'th AND gate AND a_i and b_i to give you s_i . Similarly, i'th OR gate out of thirty two OR gates OR a_i and b_i to give you s_i which is OR of these two.

(Refer Slide Time: 40:24 min)
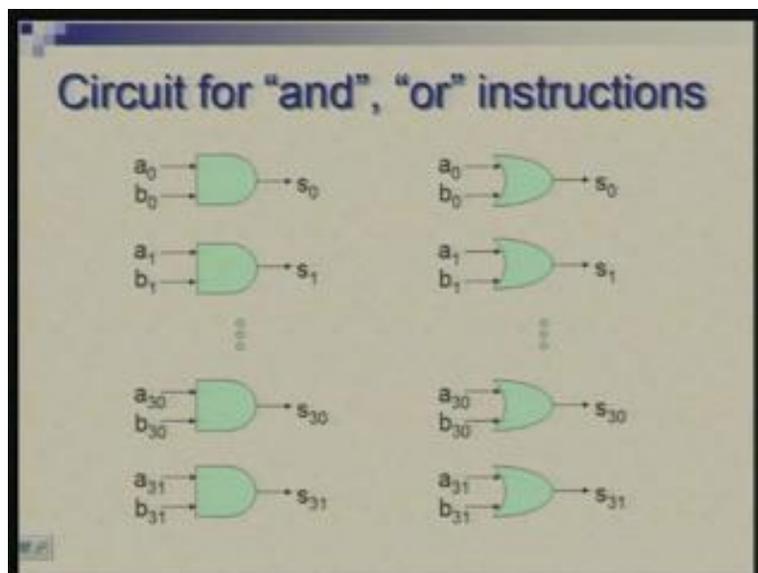
So now let us combine all these four; we have addition, subtraction, ANDing and ORing. These four operations could be combined and let us try to construct a very simple ALU. You would need to enhance it further to bring in more operation but let us keep this as the present target and once again all you need to do is put various thing together and add a multiplexor.

(Refer Slide Time: 41:14)
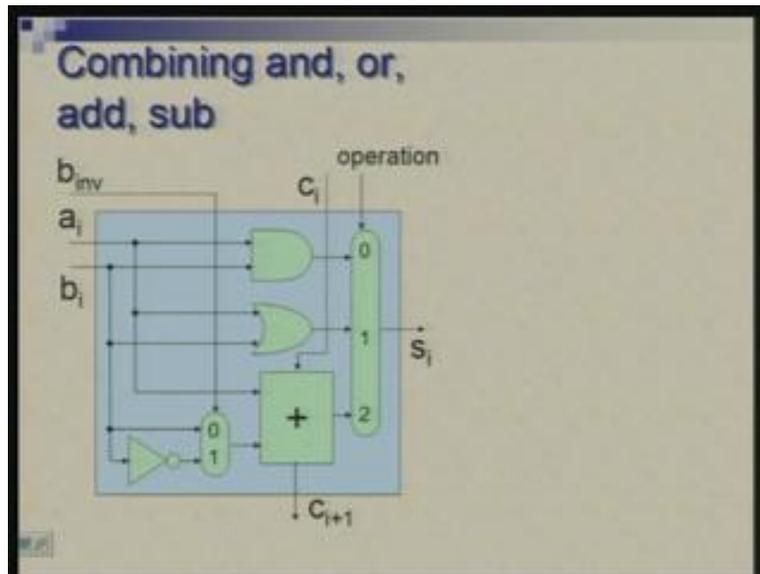
So this multiplexer has now three inputs (Refer Slide Time: 41:18) which I have labeled as 0, 1 and 2 and depending upon a control input which is labeled as an operation, either you will select this or you will select this or select this. So the input a_i and b_i are spread out to all these three components: to AND circuit, to OR circuit and to this adder cum subtractor.

Of course for purpose of subtraction we are actually having this b_i routed through this inverter and multiplexer. So this is one module or one bit unit of ALU which is capable of performing one of the four operations and it needs to be guided by this input which can have three possible values and this input which can have two possible values. Of course this is ignored or it is don't care when you are performing AND and OR operation. Therefore, when you are actually choosing this or this, what you are doing here it does not matter. So this part... effectively what we are doing is the circuit is always there for performing all the operations so you give some a_i and b_i and the circuit would do AND operation as well as OR operation and one of the two among addition and subtraction but you are picking up only one of the results.

So when you are doing AND operation for example, this, whether it does addition or subtraction it does not matter. Therefore the value of this is ignored when this says you pick up 0 or pick up 1; it is significant only when you picking up to here.

(Refer Slide Time: 43:34)
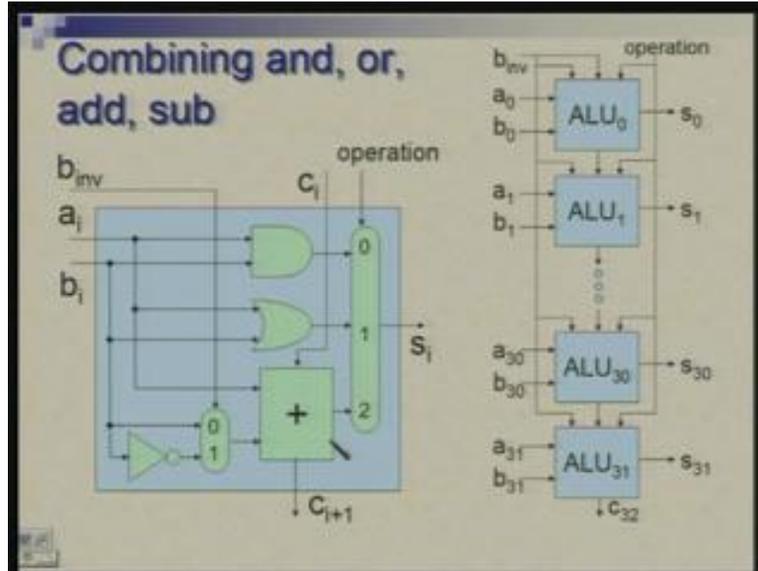
Now, here is a cascade of thirty two or such units which are called ALU_0 , ALU_1 and so on up to ALU_{31} . So you notice that on one hand we have this b_{inv} inverter going to all of them and particularly in the first unit it goes to two positions in the initial carry position and b_{inv} position and this operation control also forms another control input which goes to all of these. So now what we have done here is we have taken care of the core of these four instructions: AND, OR, add, subtract and in fact strictly speaking we have actually accounted for not just these four instructions but additional four instructions which corresponds to their immediate version so there is a mistake so well, subtract has no subtract immediate because the constant….. if you add a negative constant you can effectively do subtract immediate so effectively that means seven instructions: AND OR add subtract four and then AND immediate OR immediate which is ANDI ORI and add immediate. These seven instructions will all utilize this part of the circuit.

Going further…… actually there are…. as far as the add and subtract are concerned, there are two different types of add instructions: add and add unsigned so there is an instruction add and add and addu. Now question would be why we have two separate instructions add and addu when we have said that addition can be done with the sign ignored when you are using 2's compliment representation then what is the purpose of having two different instructions. The reason is there is something which we have ignored here is the detection of overflow.

What happens when the number which is coming out as a result here is not within the range of numbers which can be represented?
When you are adding too very large numbers the result may be beyond the limit or you are adding two large negative numbers still you may exceed the limit on the negative side. Or you are subtracting a large negative number from a large positive number or vice versa subtract a large positive number from a large negative number. Again their range could be exceeded.

And it is here in these the limits were positive and negative numbers differ or signed or unsigned numbers differ. If you are working with signed numbers your range is from something negative to something positive. To be more precise your maximum negative number is minus 2 raised to the power 31 and maximum positive number is 2 raised to the power 31 minus 1 but when you are working with unsigned numbers the range is different 0 to 2 raised to the power 31, 2 raised to the power 32 minus 1. Therefore, detection of overflow is different depending upon whether the numbers are signed or unsigned so the process of addition, subtraction remains unchanged but it is overflow detection which is different and we will look at it separately. So effectively this circuit will cater for these instructions their immediate versions, their unsigned versions of course there is addu and subtractu and there is also unsigned version of add immediate addiu. So all these instructions are taken care of by this small unit.
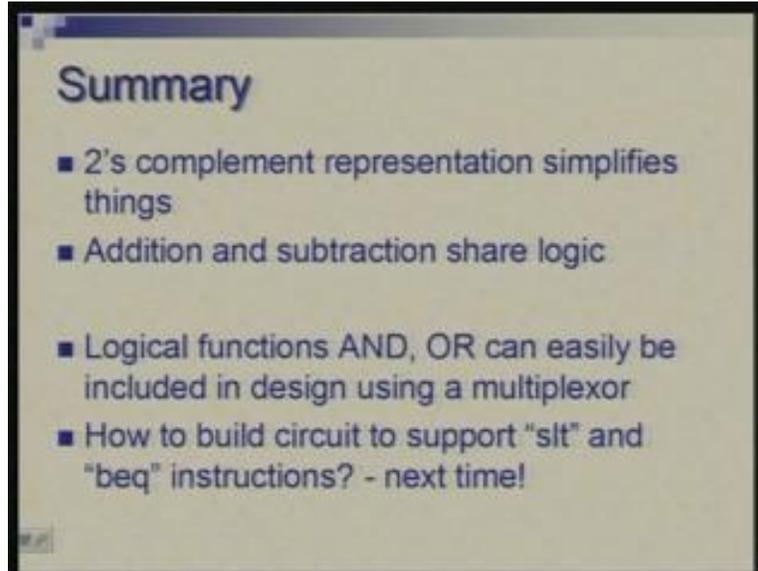
Of course we are not saying that this itself its sufficient to implement the entire instruction; the processor has to have other things: the register file, the program counter, this is the mechanism of getting an instruction from memory, interpreting it, storing the result back and so on but this is the key unit which performs the operation that we have seen.

(Refer Slide Time: 43:31 min)
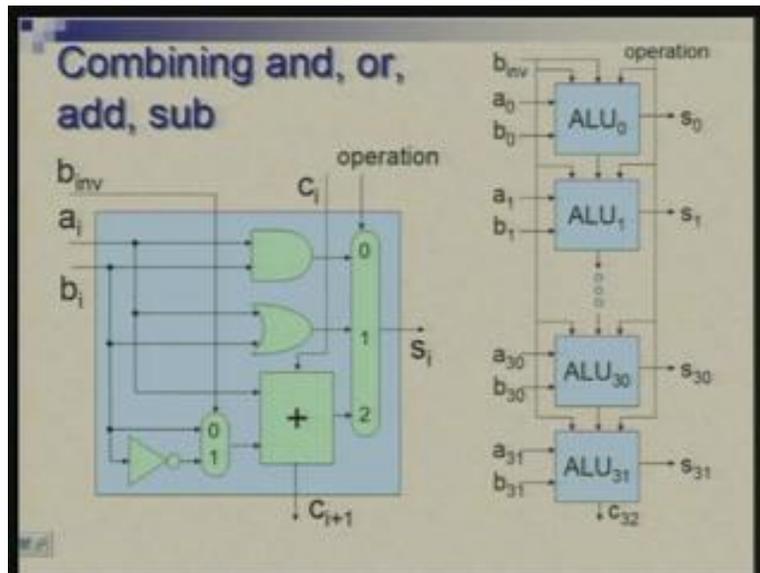
So, to summarize what we have done today we have looked up 2's compliment representation for signed numbers and we have seen that it first of all brings unique representation for 0 which was the problem in other and secondly it simplifies our mechanism for performing operations. So the circuit is simpler we can ignore the sign while we are working with it. We also noticed that addition and subtraction hardware has lot of commonality and since we are either doing addition or subtraction in the processor we can share the hardware.

(Refer Slide Time: 48:10 min)

Summary

- 2's complement representation simplifies things
- Addition and subtraction share logic

- Logical functions AND, OR can easily be included in design using a multiplexor
- How to build circuit to support "slt" and "beq" instructions? - next time!

We have seen that logical functions AND and OR can be easily included in the design of the ALU simply by putting those gates and multiplexor together. Now, one thing which we have left out is of course overflow detection. But secondly, comparison is what we have not looked at yet. So we would now, not today, in the next lecture we will extend the circuit to a accommodate instruction like slt or instruction like beq so beq makes comparison for equality. So we will augment our ALU to do equality check, we will also augment it to do slt operation which means the comparison for less than and generating a result which is either 0 or 1. So these two instructions will be accommodated in the ALU next time.