Similarly ah the number of plus if you see; the number of ones in the answer is 0, the parity flag is set to 1, but again since both the numbers are negative, but the sign is 0 which indicates the answer is positive.

So, overflow flag is 1; so, in this case you see what is an overflow. For example, if both the numbers are negative. So, if you consider a; so, again this one is very important is you are considering signed arithmetic. So, if you are taking a signed arithmetic then 1 and 1; they are two negative numbers.

Since both the numbers are negative sign bit is 1, if we are considering a signed 2's complement arithmetic, but the sign bit of the answer is 0 that is this is a 0. So, had these been the two negative numbers; so, what are the 100 and 100. So, if you are taking a 2's complement arithmetic it will be 00 ah it will be 0 sorry ah this is actually nothing, but 0111, you add a 1 nothing, but -8.

But this is nothing but equal to -8 and -8. So, if you are taking it's a signed arithmetic format this is -8 basically. So, in that case you are adding -8 and -8. So, in this case your answer should be -16, but somehow you are getting the MSB as 0. So, in this case we chose the both the numbers are negative, but the answer is 0 which indicates that the answer is positive.

So, the overflow flag is 0; so, the over flag overflow or overflow flag is set to 0; that means, ah the sorry overflow flag is set to 1 because had this is a signed arithmetic. So, this -8 -8; the answer should have been -16.

But some of the answer is showing MSB as 0 which is basically wrong due to an overflow; so, the overflow flag is set to 1. But in this context as we are using an unsigned arithmetic, you have to totally neglect the overflow flag. So, in this case we are going to neglect the overflow flag, but we are going to take the carry flag because they are both unsigned numbers one has been generated which is nothing, but your carry flag.

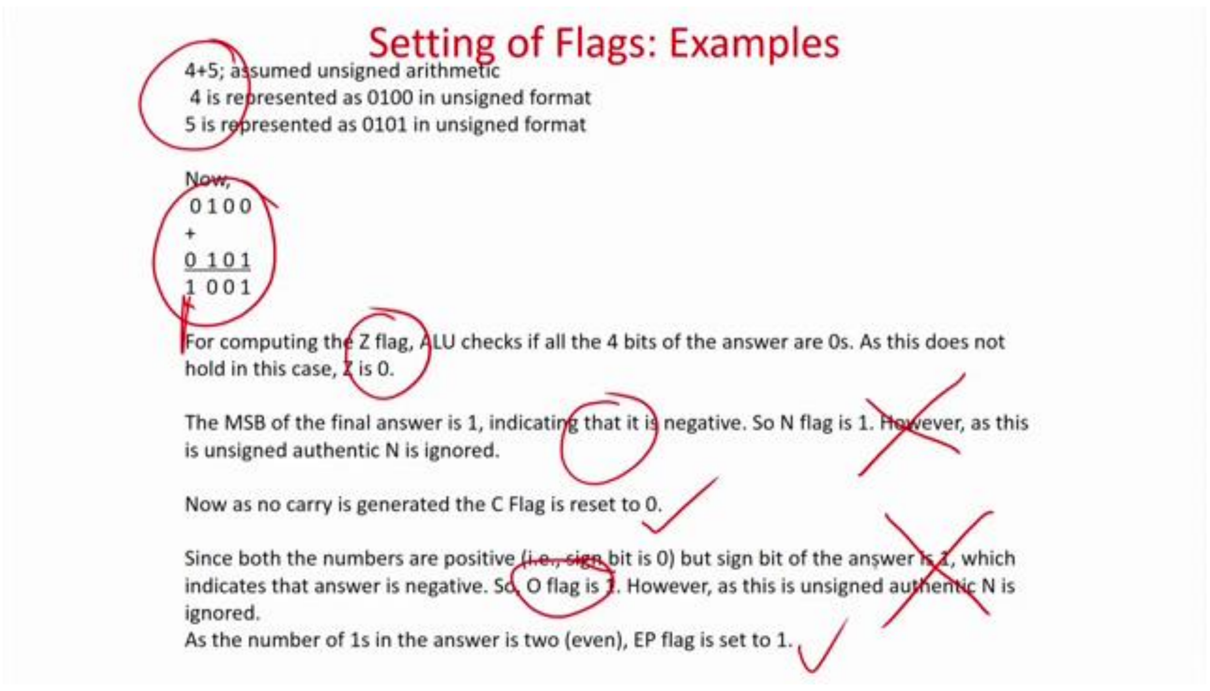
But this is not a mistake there is a no overflow because we are considering only unsigned positive numbers. So, in this case this is +8 and not -8 in the 2's complement format. So, in other words it is very very important to decide what is the context and what flags I have to take and what flags I don't have to take.

Like when I am adding two numbers which are positive immediately you have to think that I will take the zeroth I will take the zeroth flag, I can take the negative flag, I can take the parity flag, I can take the carry flag, but as there are two signed arithmetic's I am adding it. So, the overflow flags can be neglected for the time being like just like this; here it's an unsigned arithmetic two numbers you are generating.

So, the carry flag will be neglected over here because in a signed binary one is negative one is positive or in 2's complement subtraction when you are doing; we always neglect the carry 2's complement positive or negativity whatever when you are doing in 2's complement we neglect the carry.

So, since both the numbers are of different size different symbols like one is positive and one is negative; the overflow can never be generated. So, the overflow flag will always be 0 in this case, but we are not neglecting that is very very important over here ok ah some more very simple examples.

(Refer Slide Time: 45:30)



Setting of Flags: Examples

4+5; assumed unsigned arithmetic
4 is represented as 0100 in unsigned format
5 is represented as 0101 in unsigned format

Now,
```
 0100
+
 0101
 1001
```

For computing the Z flag, ALU checks if all the 4 bits of the answer are 0s. As this does not hold in this case, Z is 0.

The MSB of the final answer is 1, indicating that it is negative. So N flag is 1. However, as this is unsigned authentic N is ignored.

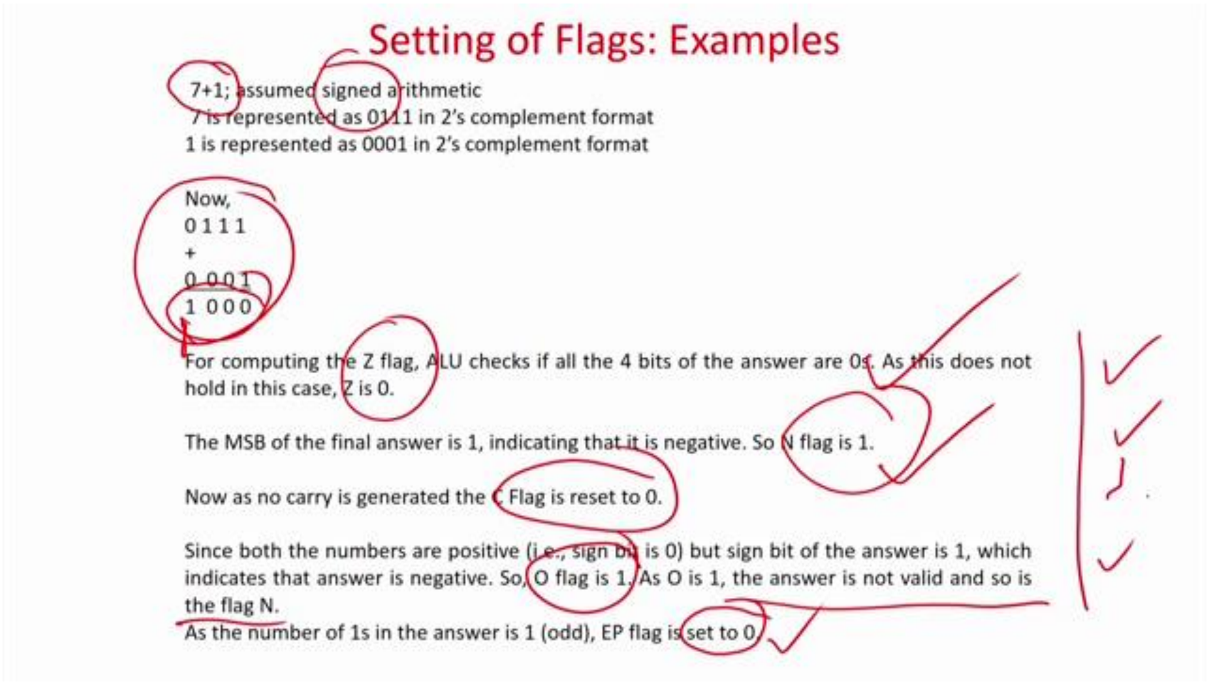
Now as no carry is generated the C Flag is reset to 0. ✓

Since both the numbers are positive ~~(i.e., sign~~ bit is 0) but sign bit of the answer is 1, which indicates that answer is negative. So, O flag is 1. However, as this is unsigned authentic N is ignored.
As the number of 1s in the answer is two (even), EP flag is set to 1. ✓

Like 5 + 4 if you are going to do again this is a unsigned arithmetic. So, you add this you are going to get the answer as this; obviously, the zero flag is reset because the answer is not 0; the MSB is 1 indicating it's a negative number, but again it's an unsigned arithmetic. So, you have to neglect the negative flag there is no carry generated over here. So, the carry flag is reset we have to consider this because 4 + 5 is 9; you are doing the operation in a 4 bit arithmetic.

So obviously, no carry will be generated two numbers you are taking +4 and -4 the MSBs are 1 which is a negative number. So in fact, an overflow flag will be set because two negative numbers who are takes two positive numbers we are taking the answer should always be positive, but the answer is showing an MSB. But again as I told you it's an unsigned arithmetic overflow flag will be set to 1, but it's an unsigned arithmetic. So, it is ignored the number of 1's are 2 over here even parity even parity flag is set and you have to consider this flag.

(Refer Slide Time: 46:28)



Setting of Flags: Examples

7+1; assumed signed arithmetic
7 is represented as 0111 in 2's complement format
1 is represented as 0001 in 2's complement format

Now,
0 1 1 1
+
0 0 0 1
1 0 0 0

For computing the Z flag, ALU checks if all the 4 bits of the answer are 0s. As this does not hold in this case, Z is 0.

The MSB of the final answer is 1, indicating that it is negative. So N flag is 1.

Now as no carry is generated the C Flag is reset to 0.

Since both the numbers are positive (i.e., sign bit is 0) but sign bit of the answer is 1, which indicates that answer is negative. So, O flag is 1. As O is 1, the answer is not valid and so is the flag N.
As the number of 1s in the answer is 1 (odd), EP flag is set to 0.

Similarly, again if I ah take some other example 7 + 1 = 8; same thing is going to be the answer. Answer is not all 0 zeroth flag is reset we have to take it negative flag MSB the final answer is negative. So, negative flag is set to 1; again ok just a minute.

So, let us assume that this slightly different in this case. So, in this case I take 7 +1, but in this case I assume a 2's complement arithmetic like in this case this was just to just to keep your variation. So, in this case is an unsigned arithmetic I could have also taken this in unsigned arithmetic version, but in this case I am using two positive numbers, but still I am using a signed arithmetic version.

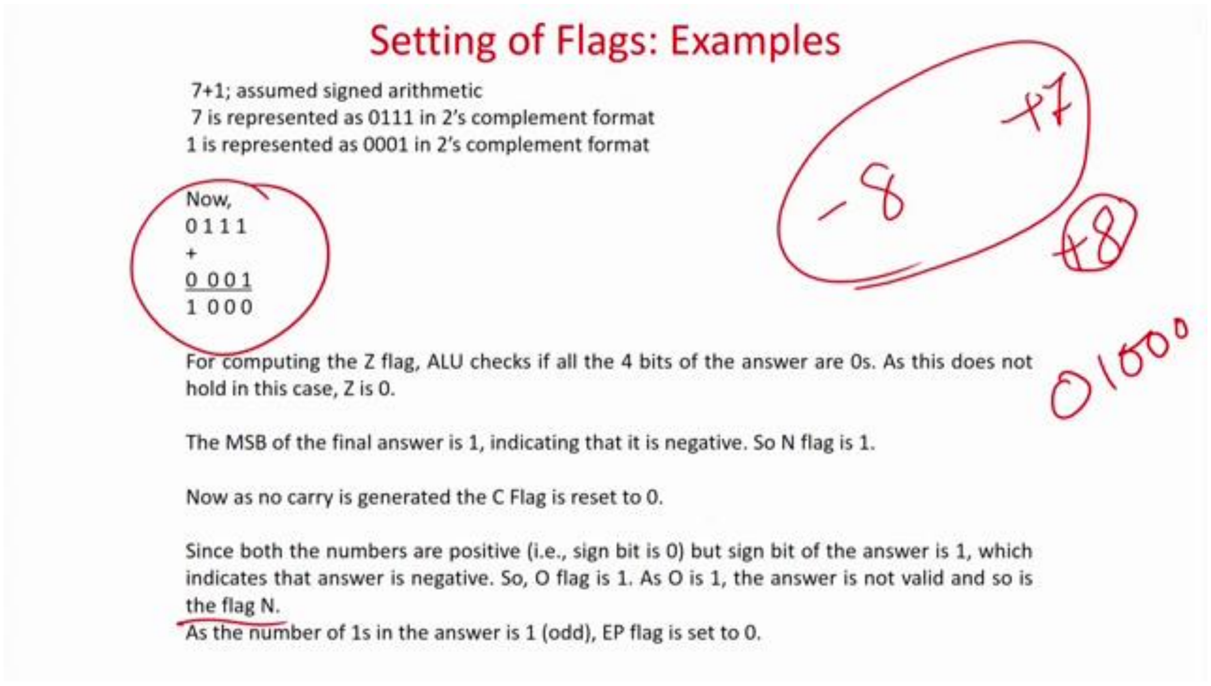
So, just I am trying to see what is the difference? So, in this case; so, I am adding 7 +1. So, it is 8 so this is the case; so, in this case I am using a 2's arithmetic 2's complement arithmetic 2's numbers are positive, but still I am using a signed arithmetic just to give the example. So, MSB is 1; so, indicating the number is negative; so, the negative flag is set to 1.

So, there is no carry generated; so, the carry flag is set to 0. Since both the numbers are positive that is sign bits are 0, but the sign of the answer is one which is a reverse. So, the overflow flag is set to 1 as 1 is also this is as O is 1 the answer is not valid and so is flag N.

Now we are going to see in details only 1 1 in this answer. So, in this case is a odd parity; so, the odd parity flag is set to 0. Again in this case all flags are valid again let us re look what we are doing and what is the difference. So, in this case +5 and +and 4 they are positive numbers we are using a unsigned arithmetic format. So, I have told you what it means in this case I am taking two positive numbers like 7 and 1.

But in fact, I am using a signed arithmetic version that is ah is ah 2's complement arithmetic where the range is from -8 in this 4 bit number -8 to +7 this is the range you all know from digital design fundamentals. But in this case there are 4 bit numbers and this is a unsigned arithmetic. So, you can go from 0 to 50 that is the range difference.

(Refer Slide Time: 48:39)



Setting of Flags: Examples

7+1; assumed signed arithmetic
7 is represented as 0111 in 2's complement format
1 is represented as 0001 in 2's complement format

Now,
0 1 1 1
+
0 0 0 1
1 0 0 0

For computing the Z flag, ALU checks if all the 4 bits of the answer are 0s. As this does not hold in this case, Z is 0.

The MSB of the final answer is 1, indicating that it is negative. So N flag is 1.

Now as no carry is generated the C Flag is reset to 0.

Since both the numbers are positive (i.e., sign bit is 0) but sign bit of the answer is 1, which indicates that answer is negative. So, O flag is 1. As O is 1, the answer is not valid and so is the flag N.
As the number of 1s in the answer is 1 (odd), EP flag is set to 0.

So, in this case I have talking on a 2’s complement rate that is -7 to +7. So, if I do the addition we are going to give this as the answer. So, as if you look at it it’s an invalid answer because 7 +8 is going to be positive 8, but positive 8 cannot be represented in 4 bits in 2’s complement arithmetic; these are very well known thing I think if you are forgot you can just go and revise your digital design fundamentals. Because positive 8 will be 0100 this is actually positive 8 because 1000 is negative 8 in 2’s complement arithmetic. So, this in fact, that is why I told you that it’s going out of range; so, this is going to be incorrect answer.

So, if you do this we are going to get the answer this one; 1000 as a hardware it does not understand much, it will just develop the value of flags based on certain hardware computation. So, 0 zero flag is reset it is taken, MSB is 1 directly indicating that it’s a negative number. So, negative flag is set it’s a negative number. In fact, as I told you the answer should be positive; positive 8, but we are not going to get the answer positive 8 in a 4 bit answer.

So, therefore, you will require a larger space or a 5 bit space to implement; if you could have done in this way then your answer would have been correct would have got this as the answer which is going to give you the correct answer in signed arithmetic.

But as you are using a 4 bit number to do this so in fact, that is why you are going to get an overflow and all the problems have started. So, MSB is 1 the negative flag is 1 there is no carry generated, those carry flag is listed, but here the overflow flag is actually our main role player here. So, it is finding both the answers; both the inputs are 00 that is they are positive numbers, but the answer is a negative number.

So, immediately it is going to say that the overflow flag is 1. So, now, we have to check that when the overflow flag is 1 and the answer is negative. So, both of them are saying that the answer is not valid and so, is the negative flag. So, immediately whenever a overflow is generated; that means, you have to understand that overflow is different from carry.

Carry means some carry has been generated and, but the answer is valid. But in in this case what happens; so, with the carry the whole answer is valid, but in this case when there is a overflow generated means there is no carry generated, but the answer is actually a wrong answer because of the overflow.

So, I could have easily connected by putting it as a 1 bit additional I mean I could have if I would have done with the 5 bit; the answer would have been correct. So, whenever the overflow flag is set so, immediately it will say the answer is invalid as well as the negative flag is also invalid, the sign is also invalid and the answer is also invalid.

But that is why if I just compare again with this one you could have checked the ah see both the numbers are of different size, this is of different size, this is of different size; then immediately the ah overflag is reset; that means, if the two numbers are different like one number is positive one number is negative in a signed arithmetic; you can never generate an overflow.

Because then if two numbers are subtracted; the answer is always less. So, if you can represent one negative number sorry one positive and one negative number in 4 bits; then the answer will always can be represented in 4 bits because you are making the number less due to subtraction.

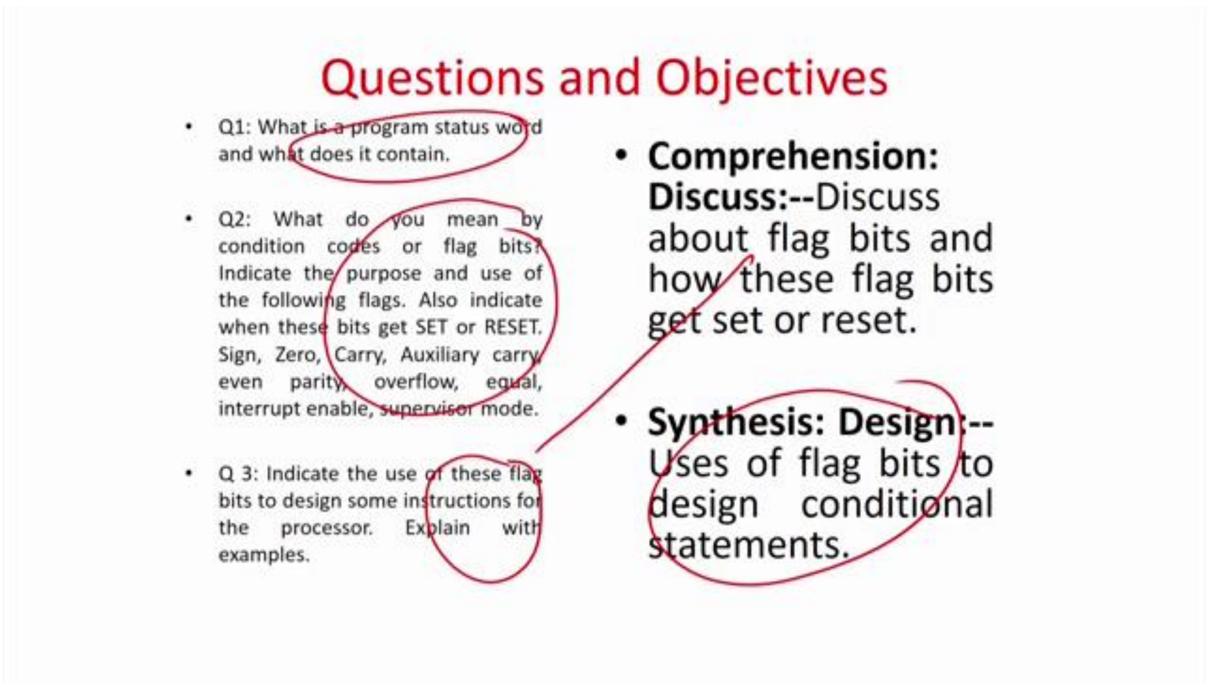
But if there are two numbers of same sign positive number or negative number; then the number can become larger than the two operand itself and it requires to take more number of bits that is what actually has happened. 7 +1 is 8 which is +8 in the signed arithmetic is nothing, but is 01000; 5 bits it cannot be accommodated in 4 bits.

So, overflow has been generated which actually neglects the you have it will not do anything, but just after the operation some flags are set and reset. So, whenever you find out that this overflow flag is set. So, you have to know that the answer is wrong and therefore, in fact, you have to give more precision to this answer. So, that is what is the idea of setting the flag.

So, it is a very very complicated situation and maybe later we will see how to use this overflow flag to set a ah how can you generate an instruction based on the overflow flag to see whether the answer is valid or not. So, for example, if you have a 32 bit machine and you are taking all largest possible number you have fitted up into the memory.

And you are doing the computation; so, every time you have to check whether there is a overflow or not. Since you are getting an overflow bits set by default you have to declare that this is an invalid answer or the precision is an error; so, all these things you have to report. So, this complicated things maybe we will try to see whenever we will be knowing going more in to assembly language coding and micro programming etcetera.

(Refer Slide Time: 53:17)



Questions and Objectives

- Q1: What is a program status word and what does it contain.
- Q2: What do you mean by condition codes or flag bits? Indicate the purpose and use of the following flags. Also indicate when these bits get SET or RESET. Sign, Zero, Carry, Auxiliary carry even parity, overflow, equal, interrupt enable, supervisor mode.
- Q 3: Indicate the use of these flag bits to design some instructions for the processor. Explain with examples.

- **Comprehension: Discuss:--**Discuss about flag bits and how these flag bits get set or reset.
- **Synthesis: Design:--** Uses of flag bits to design conditional statements.

So in fact, what we have done in this class? In this class basically we have seen in the crux of what is the conditional instructions and more importantly; how basically flags are set and reset that is the most difficult part of it.

Once you understand how a flag is set and reset and then accordingly you can easily generate the instruction based on that. And based on the truth and false you go over to the label or you just execute the instructions as usual. So, some typical questions basically which can be asked and let us see how we fix your objective like for example, the first question is what is the program status word and what it contains? Then the second question said what are flag bits, what are different types of flag bits like set, reset, auxiliary set, overflow etcetera.

Indicate the use of this flag registers and with examples how can you do it and basically what instructions can be done or what purpose it can solve. So, if you look at it discuss about flag bits and how this flag bits are set and reset. So, if you are able to answer this and if you are able to answer this of course, this instruction is met.

Use of flag bits to design conditional statements; that is again ah this ah if if indicating use of flag bit should design some instructions; so, it is a synthesis objective. So, after the after doing this unit when you will be able to as if you are answering the question number 3 that you are designing newer instruction some instruction sets for a processor based on these flags, then basically you are able to satisfy the synthesis objectives of designing conditional statements.

And whenever I talk about program status word; so, whenever this flag bits etcetera; that means, flags bit are you can whenever you are synthesizing on the use of flag bits for design conditional statements this implies that you have to know that when I am ah jumping from one context to another then everything has to be set.

And basically what are the values like intermediate registers etcetera which has to be used in a program status word what it contains basically where it is to be saved? This will actually satisfy the objective on comprehension so in fact, ah just after this unit, you will be able to meet these two objectives which we targeted in the beginning.

So, ah this actually completes the ah second one in the seventh unit of this module and next time we will see how to use these instructions these jump instructions or conditional instructions to execute one very important part of your programming paradigm; that is actually your functions and procedures.

Thank you.