Computer Organization
Part – I
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology
Lecture - 12
Addressing Modes

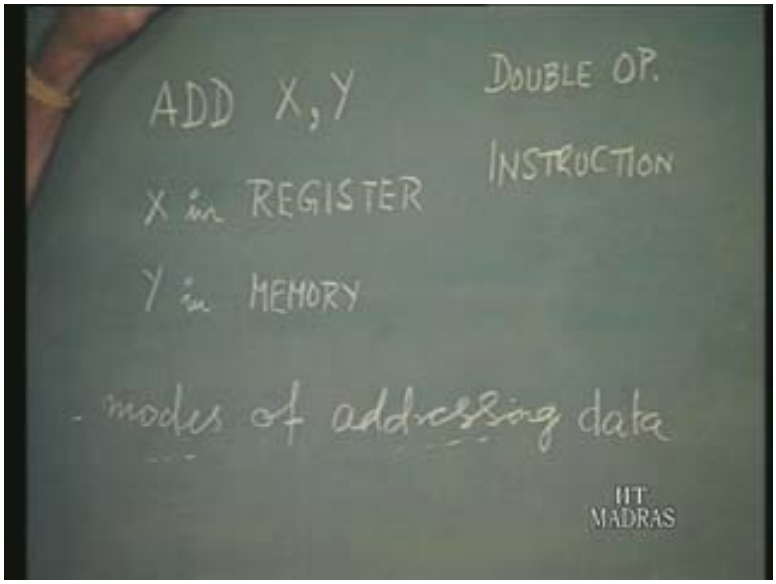
In the previous lectures, we were essentially looking at an example of a double operand instruction. We generally discussed the instruction format and indicated that the instruction, any instruction, may refer to one operand or two operands or sometimes no operand also; so we saw three cases, specifically after taking one example of double operand instruction, that is, an instruction which refers to two operands.

For instance, this particular one, add, is an instruction and it refers to the two operands X and Y. Just for the sake of discussion, we assumed the data X available in some register with the CPU and the data Y available in the memory location. Then, after taking this particular example, we were in fact working out the series of actions we must take. In other words, the micro-instructions involved in the execution of this single instruction were studies. Now this takes us naturally to the next step; we started with an assumption X in register and Y in memory. So naturally, the next step would be to check the various possible ways in which the operands can be referred to. In other words, what we are talking about is the different modes in which we can address the data. That is, modes of addressing the data or referring to the data will usually be called addressing modes also.

What is the addressing mode? Addressing mode is a mode in which we refer or address the data; now here we find in this example which we took, the data X is in register. So this particular mode of addressing is just called register mode. Then, in the second one, the data Y is in memory; that is, the second data Y is in memory. That means we have to give an address of memory location somewhere.

Now we can talk about address of a memory location; the CPU must place its address and then get the data. Recall what was going on earlier. Now for CPU to place that address the address must be made available to the processor. That address may be say in one of the registers of the CPU itself, some register, or that address can come as part of the instruction word itself, and so on. So I will introduce two things right now – one is the register. In this, the register itself holds the data.

(Refer Slide Time 03:50 min)



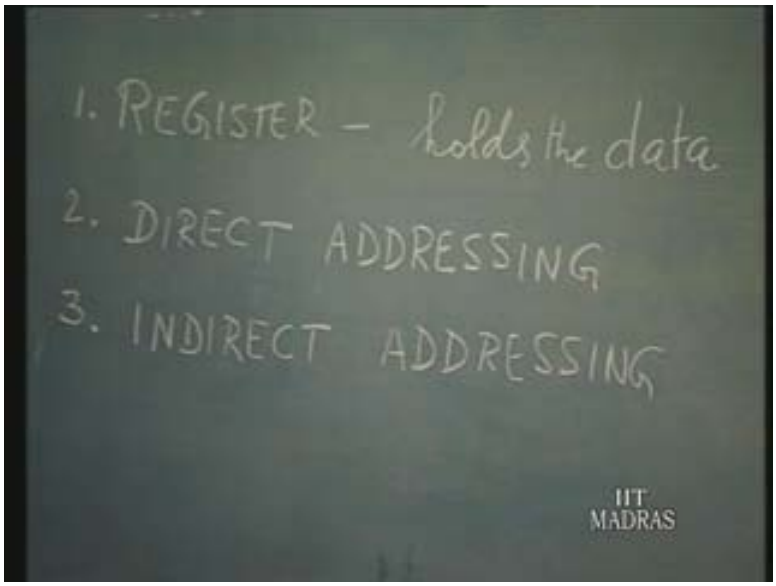
What is the advantage of this particular one? The advantage of this is that the register is usually part of the CPU, and what we mean by register holding data is that the data is already available with the CPU, which means the CPU need not go elsewhere and it saves the data. In the second one also, the data is in memory, so we can call this particular one as direct mode of addressing or direct addressing. Why do we talk about direct addressing? What we are saying here is that the address of the memory location, which contains data, is available somewhere directly. In the sense that the address may be part of the instruction or may be available in some register. For instance, if this address is available in a register, sometimes this is also referred to as register direct mode because the register holds the address.

What did you see here? Suppose this is a register direct mode, it means the register holds the address and what was the previous one? The register holds the data. The previous one is register holds the data and second one is register holds the address. Now from the register, the address will be placed and the data will be got. That is one more step, whereas in this particular one, the first one, register holds the data. In the other one, register holds the address and the address must be placed on the bus. A bus activity must be initiated and then the data must be fetched into the processor for it to proceed.

Now obviously, when we talk about direct addressing, there must be another one by which we may do it, say, indirect. In direct immediately use the data; in indirect, the address is not immediately available, but the register holds something which in turn holds the address. So it is somewhat like this: suppose in the CPU some register holds whatever it is. Now it is not necessary that the register must always hold, but let us say the register holds some number, 1000, and in the memory location 1000, let us say, we have 1010, assuming 1010 will be somewhere here.

Now let us go back and see. In the first one, the register was holding the data; in the second one, the register was holding the address, and now we are discussing in the third one, it is indirect. So register holds a number, 1000, which holds a number 1010, which is the address.

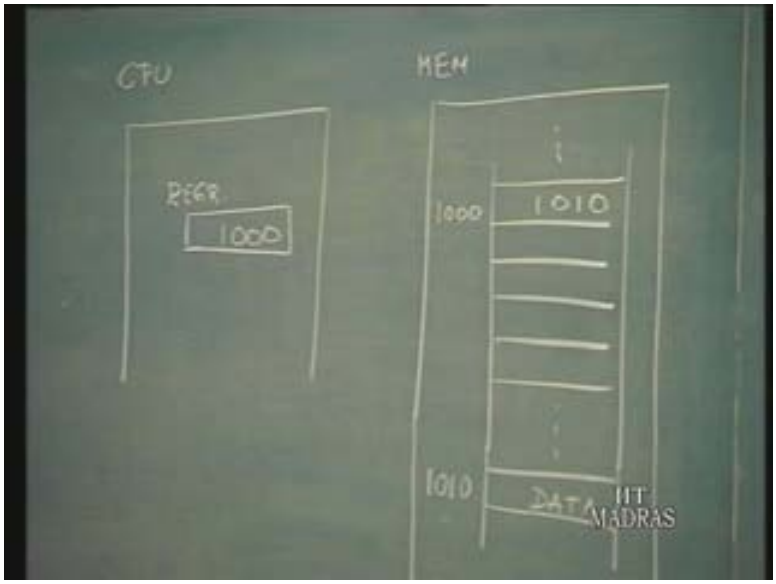
(Refer Slide Time 07:19 min)



So whatever data we are looking for is available in this location. Instead of the register holding the address as in the second case, if it were a register, direct addressing register, holding the address, the register holds a number. In other words, this is also an address. Just remember this is also an address; in other words we say register holds the address of address of data. That is what we mean when we say we are indirectly referring to the data. This particular thing is also called deferred mode, that is, deferring or postponing. So the address is not immediately got; it is got after some deferring; so there is one level of deferral. The register holds an address, which holds the address of the data. Now what happens?

Let us see; in case the register holds the data as in first mode there is no problem. The CPU need not carry out any bus activity and it need not access the memory and so on. It already holds the data. In the second one, the direct addressing, if it is registered direct addressing, we say register holds not the data but the address, in which case the register will hold the address. In this case it will rather hold 1010. In the third one the register is referring to it indirectly. So the register holds an address, which holds address of the data. In this first one, the register holds the data; in the second one, register direct addressing, register holds address of data.

(Refer Slide Time 12:20 min)

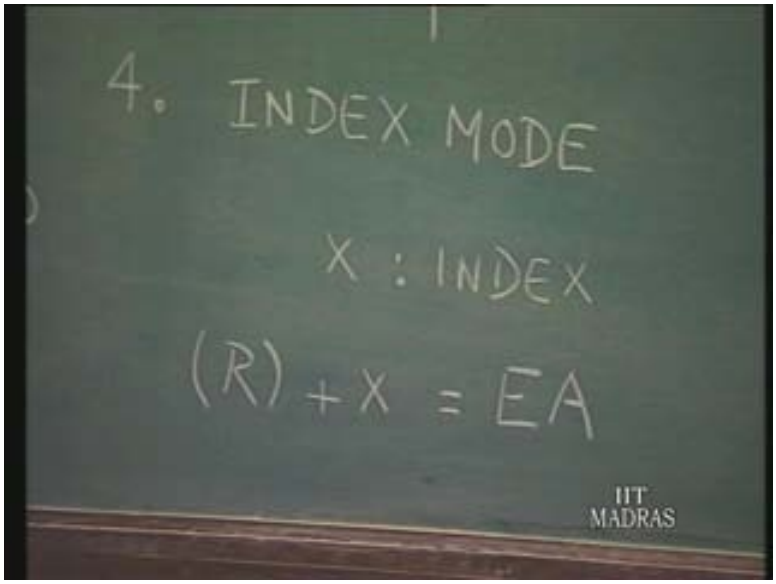


In the third one, the register holds the address of address of data. Now what is the problem? CPU holds the address in the register and there is no bus activity. If CPU holds an address in a register then there is one machine cycle involved. We have seen earlier that the instruction cycle consists of machine cycle.

What is a machine cycle? A machine cycle is one in which some bus activity is initiated to bring some piece of information, some code or whatever; it is to the CPU. So in the second case, the register must place the address and get the data. Now what happens in the third case? The register places the address in the third case; register places this address 1000, and gets into itself the contents of that location 1000, which is 1010. Then again, the register places 1010 over the bus and then gets the data. So it is going to take more time. Going in this particular manner, we are finding that there is more and more calculation involved in getting what we may call as the address of the data.

In other words in calculation of what is called an effective address there is more and more calculation involved. So, more time will be spent in calculating the effective address of the data. It is the final location – what is the address of it? We have to calculate these three. Now we can expand this particular one into the fourth mode; I am not following any specific order in this. We can, for instance, take the fourth mode.

(Refer Slide Time 15:42 min)



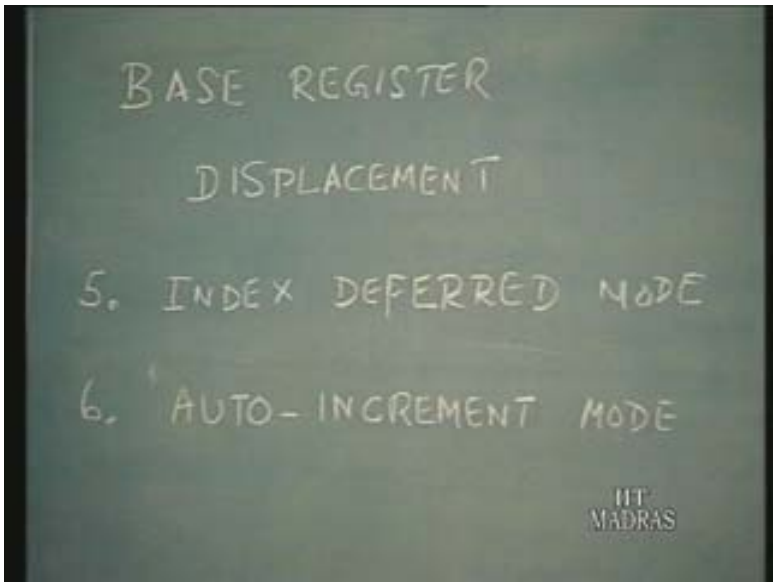
I will just call it an index mode. Sometimes you may find the same thing being referred to in different ways in different books. What is the index? Index is actually some number X, and there is some number X that will be the index and this X must be added to, let us say, some register contents. Some register will be having an address and the contents of that register – an address plus X, you add that and you calculate the effective address. So the effective address is calculated like this. That is, assuming of course; this varies from processor to processor. This mode is referred to as what I said in different ways in different books, but essentially what it is.

Now let us go back. In this first one, the register holds the data; in the second one, register holds the address, in the third one, register holds the address of address. Now what you find? We find that the register holds an address and this X, where does it come from? It must come from that particular instruction, the instruction which refers to the mode. (15:30)

Now it says that instruction will supply this value X, which is the index. There will be some register that should be used; the register contents plus X will be the effective address. Now, what have we arrived at? We arrived at an effective address, which is something like direct addressing. Now you add one more to this itself by deferral; with another indirection you can do it. And index mode is done instantly before we proceed. Normally this particular register will be called a base register. This R could refer to that register, which holds some number; this will be called a base register. Why is it called a base register? It must be able to supply your register because the instructions supply the number X; when it is added to this, you get the effective address. This is some base, which refers to that. Let us say the register is 1000 and index is 10. Suppose this base register, let us say, is the base register. It holds 1000 and supposes index is 10. Then what it means is 1000 plus 10, which is 1010; this refers to 1010 locations away is the base. With reference to these 10 locations away, that's what this index is pointing to; and because of this particular reason it refers to displacing. So this will also be called displacement. Sometimes this index will be called displacement, something that is fixed with reference to which you displace and then get it.

That is one point; so with the first base address we also talk about the displacement. Now these are all different ways in which the address of data can be calculated. After the address is calculated, then the data will be brought in – that aspect we had seen earlier.

(Refer Slide Time 22:17 min)

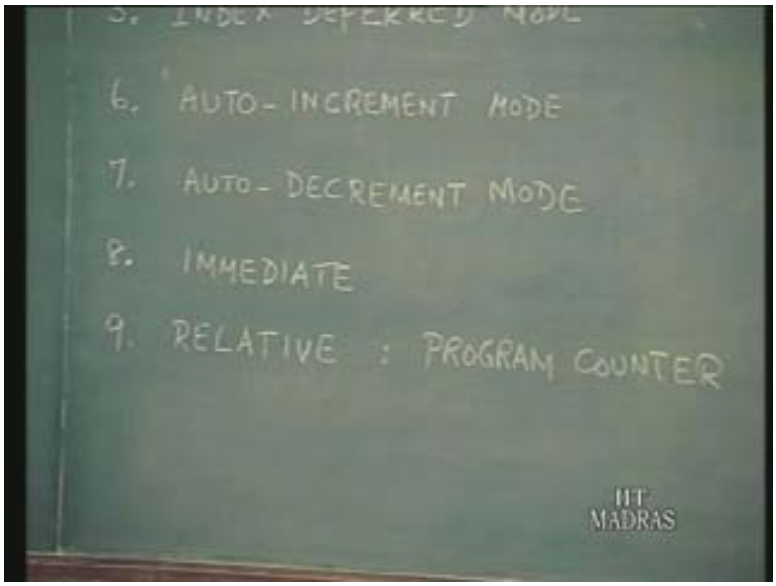


Now to this we can have an index deferred mode. What is an index referred mode? It is deferring as we said earlier. It is not direct addressing but indirect addressing – if there is direct addressing we talk about address, in indirect we talk about address of address. So in index deferred mode, that is, the next mode, let us say, whatever you calculate is not the address but the address of address. That is one more deferring indirection. So the contents of register are whatever R is plus the displacement index; that is not the address but address of address. One can go o in a similar way. Then we can add few more things to this, for instance, let us say in all these things as you have noticed, the register content is not disturbed; it remains the same.

Now in the next one, we will discharge the register content; what is it? We will say you calculate effective address; after that you increment register; that particular thing is called an auto increment mode. What is that? Let us take this particular one, the direct addressing: so the register holds the address. Now in the auto increment mode, again the register will hold the address, but after the address is used the register contents will be incremented. That is what the auto increment mode is. So the register, in auto increment mode also, just like in direct addressing, the register holds the address. After that address issues, what is that address? That address is the effective address; after the address is used to bring the data, the register contents will be automatically incremented. That is what this particular adjective says – auto increment – automatically the contents of the register are incremented. While fetching the instruction, we are fetching an instruction or 1 byte of instruction. After the particular thing is used, the program counter contents are automatically incremented because it must point to the next instruction or next byte of the instruction. So by putting the program counter in this mode, it is automatically achieved. Remember? Go through the steps again – T1 and T2, whatever was going on; similarly, you can go on.

Now you can go to auto increment deferred mode; that is, the register contains not the address but address of address, and after fetching the data from that address the register content will be automatically increment. In the same way when we talk about increment, we can also talk about decrement; so the next is auto decrement mode. In this, what is done is similar to the earlier auto decrement mode; also in the same way the register holds the address. After the address is used, the register contents will be decremented automatically.

(Refer Slide Time 28:01 min)



So it points to the previous location; in some cases this is useful. I already said that you can have an auto increment deferred mode with one more step of deferral. We said earlier that an instruction may be 1 byte or 2 byte or 3 bytes. Now as part of the instructions itself, suppose you are passing on the address. Let us say we have 3-byte instructions, that is, 1000, 1001, and 1002: let us say these three locations hold some instructions. Now that particular instruction can hold the address as part of the instruction. If that were so, then what is the difference? We said earlier the register may hold the address; now we are finding that the instruction itself holds the address. This address is referring to the opcode and the two data, if it is double operand instructions. Now we are talking about address of data; that is what you mean. So the address of data is available as part of the instruction itself, in which case we say that the instruction is available immediately, that is, in the next location or second, in this particular case second or third byte or third location. This immediate mode is not different from the direct addressing mode in the sense that in the immediate mode the address is available as part of the instruction itself, and in the case of this direct addressing what we are saying is the data is referred to by an address, and all along we are saying this will be called register direct.

Now we find that this particular address is not in register but as part of the instruction itself; so this is not really different. Another interesting mode will be with reference to a specific register. That is, in the case of index mode, I was talking about base register and so on, and then index of the displacement. In other words we are referring to a data available in some address, with reference to a register.

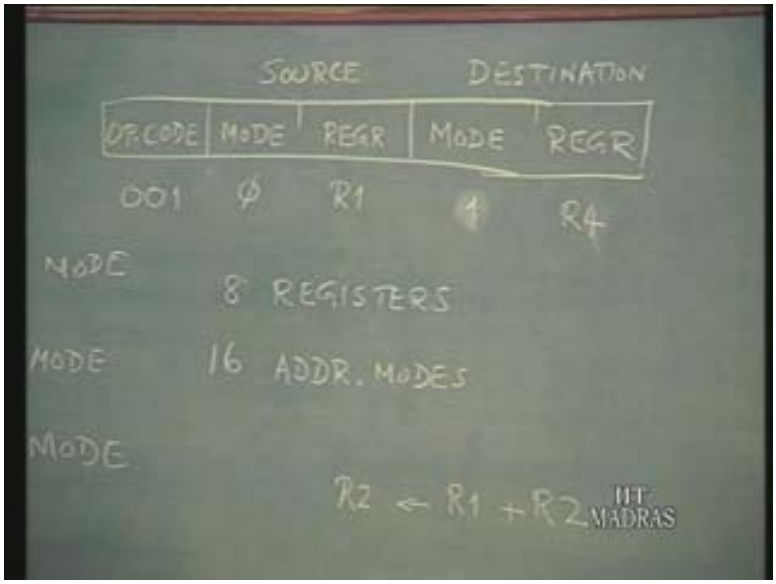
The data is available in an address, which is, let us say X bytes away from what the register points to. So this particular one is giving us the idea but the data can be referred to in a relative way with reference to say some location relatively speaking say 3 bytes away or 2 bytes away or 10 bytes away. It can be in this direction or backwards, you can also say with reference to this you can go back. So this relative addressing is a very general one, and this is a specific example.

In the case of relative addressing we have to know with reference to what; in other words, related to what the address must be calculated. Normally in index mode a specific register may be used; in the relative mode a program counter itself will be the register. PC always holds the address of instruction to be fetched or address of byte to be fetched. The program counter always holds the address of instruction to be fetched. So it is a special case actually. So you can see that relatively it is not very much different from index mode. Index mode is any relative mode program counter specifically supports because the it always points to the current instruction; may be part of the instruction, or the next instruction, depending on which part of the machine in an instruction cycle it is. With reference to what the program counter points to, you say 2 bytes away, 20 bytes away, 200 bytes away – it means this incrementing direction or decrementing direction, both ways.

Here normally in index mode plus will be used in the relative mode; it can be either forward or backward. That is a question of interpretation. Now we got some idea about nine modes of addressing. Going back to original example, we just started with some double operand instruction. We in fact came to this after discussing something about instruction format, and then we said an instruction format in essence refers to two things: one is when there is an opcode, which indicates what the instruction must do, and then it must refer to a set of operands or data. Then we said they can be no operands at all or they can be only one operand or they can be two operands and then we took a specific example of two operand or double operand case.

Now we have seen the mode of referring to the operand or the data, the mode of addressing the data or the operand. Now we must work out the operand field. Any processor will have a uniform instruction format, so in case the processor accommodates no operand, single operand and double operand instruction, the instruction format must have position for referring to two operands. We are seeing now each of these operands can be referred to in one of several modes.
The details of the instruction format evolve like this: we have the opcode no doubt; then we will expand operand part of it and see. So if the opcode part is that, now minimum we must refer to two operands. I will just call one as the source operand and other as the destination operand.

(Refer Slide Time 39:46 min)



So we are saying that is the format in which its return X is the source and Y is the destination, meaning the two will be added and finally the result will be placed somewhere; so that will be the destination. According to this X and Y will be added and the result will be stored in the destination place, which means wherever originally Y was, we will place it. Now we are seeing the details: there are many ways in which this can be organized but I just say the source operand will be referred to by saying what mode, in what mode we are referring to, and then destination is also done the same way. We have seen that the register holds data or register may hold the address or the address of the address or it may hold some address and so on and so forth. These are special and general cases have slight variations. One way in which we can refer to data is we are referring to it as source register or as destination register. Suppose a CPU has eight registers and a CPU has say 16 different modes of addressing. Now here we may refer to any one of those eight registers as source; similarly you may also refer to any one of the eight registers as destination. Then we may refer to any one of the 16 modes; similarly the destination also can be any one of the sixteen modes.

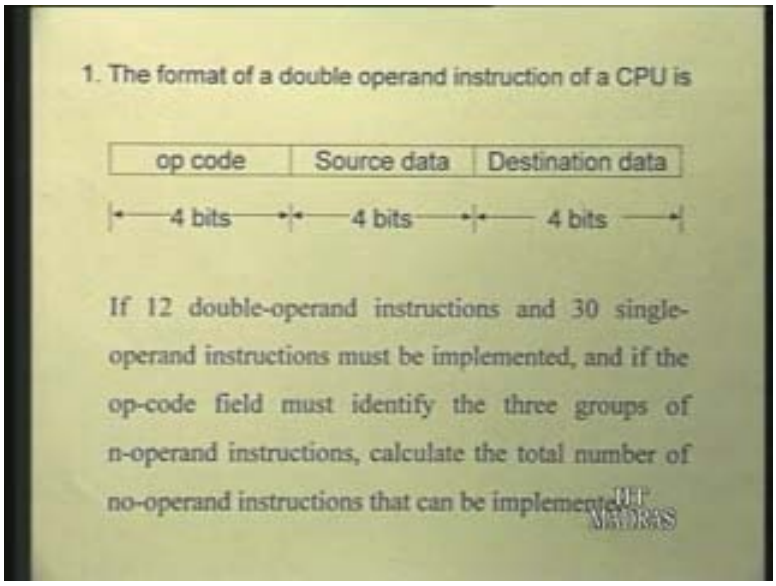
There may be some restrictions. Not all may be possible; some combinations may not be possible let us leave that alone. So in other words, here this instruction has essentially three major fields: one field is opcode; the other one is a source field and the third is the destination field. Source field and destination field respectively have two subfields, one is the mode subfield and other is the register subfield. So the subfield will have to have the code and then identify which of the register subfields must have one of the eight different codes to identify each of the eight registers. Similarly more subfields must have one of 16 different codes to identify which of the 16 modes it is referring to. Now in the very first mode, let us say mode 0 is a register mode. The register itself holds the data assuming. Then let us say you adjust both the registers assuming that is both source and destination data are available in the CPU in two different registers. So for instance, mode 0 with register mode is what we have assumed, and let us say, register 1 holds the source data and register 2 holds the destination data. So the effect of this will be the contents of register 1 will be added to the contents of register 2.

Register 1 contents are to be added to register 2 and the result will be left in the destination, that is, in our case the register 2. Now we must have the appropriate code put in this subfield. So for instance let us say in this instruction add X Y; so X is the source data, source operand, Y is the destination operand, and we have assumed X in register, Y in memory. Now memory means we have to specify where in memory, that is, some address is involved. Let me just arbitrarily say some R4 is the register, which will hold the memory address. So Y in memory is referred to by let us say register R4; register R4 holds the address of Y.

I think we are familiar with this address of Y. Let us say R4 holds the address and X is register from register. I will just say R1 is this particular register. Then, say R1 is the register that holds X; R4 holds the address of Y and the particular address holds the data Y. So R1 is my source register; so in this particular case, R1 is the source register and R4 is the destination register and R1 holds the data. This is the same but in the case of R4, it does not hold the data, but it holds the address; so this is another mode.

Let me repeat: mode 0 is one in which the register holds the data; mode 1 is the one which holds the address. So we do it this way and then whatever is the opcode for add, now let us say some opcode whatever it is say let us say 0; 01 is the opcode. So 001 is some code, which indicates add, and then the source data is referred to this way, that is, R1 holds the source data. In our case it was X, and mode 0 says R1 holds data. Then mode 1 says R4 holds the address. So from that particular address the Y data will be taken. So this is how the instruction format and filling of the various subfields of the instruction code takes place. We have seen quite few things and discussed – now just to see whether we have a grasp of everything, let us work out of few problems.

(Refer Slide Time 40:53 min)



We will start with the simplest of the problem; let us go to the chart. The problem reads the format of a double operand instruction of a CPU and it is given 12 bits. The particular format consists of three sub fields as said earlier: opcode, and then rest of it is operand field, and operand field has been divided into source operand field and destination operand field, each of 4 bits. Now suppose you have a code of 4 bytes, what does it mean? Then you can have as many as 2^4 because we are talking about binary number we can have as many as 2^4 or 16 codes possible; that is minimum. Otherwise also coding is one in which all the things are packed, that is, this is no more 2^4 is not possible. Now we are seeing that that is what is given to us here. Then the problem reads: 12 double operand instructions; so we have two operands, 12 instructions and 30 single operand instructions.

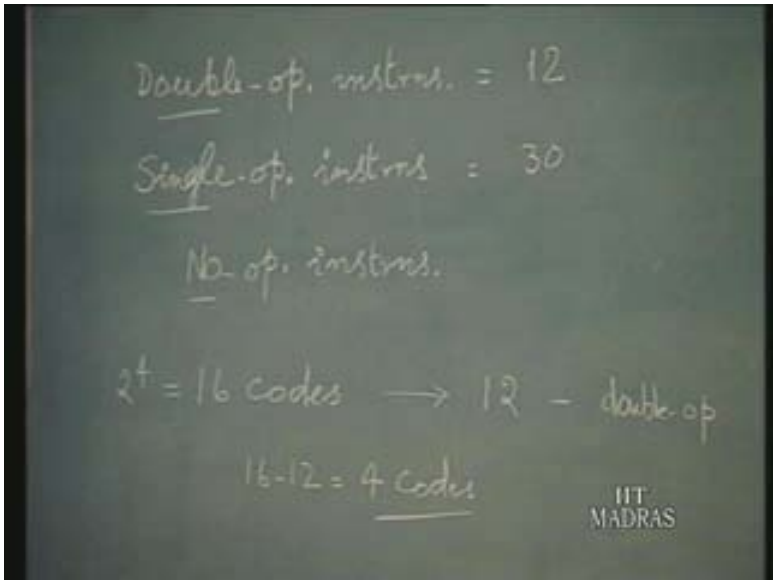
So, 30 single operand instructions must be implemented. And it says if the opcode field must identify the three groups of n operand instructions, in our case, so far we have seen two and one. Now we have three groups; what is the third group? It must be a no operand; let us see. Calculate the total number of no-operand instructions that can be implemented. So we have n as 0 for no operand; 1 for single operand; and 2 for double operand. There is one catch here – it says the opcode field must identify the three groups.

If the opcode field must identify the three groups, now this is important. What do you mean by that? The said opcode field has 4 bits, and we also said that 2^4 16 codes are possible. Now we are saying 12 double operand instructions must be accommodated and also 30 single operand instructions and we have to calculate the number of no operand instructions. Now let us work out the details: 12 of the double operand instructions are there, and the opcode field has 4 bits. So we said two 2^4, that is, 16 codes are possible.

Now this particular opcode field must, as the problem states, identify the three groups of n operand instructions. That is, n is 0, n is 1, n is 2. So first of all we saw that there are 12 double operand instructions. Out of these 16 codes, 12 codes will be used to identify the double operand instructions. So the opcode field will identify 16; of these, 12 are gone. So we are left with 16 − 12 is equal to 4 code set is available now, 4 remaining codes out of the total of 16 codes. These opcode fields must be uniquely identified. Let us go back to the chart and see. We have already considered the 12 cases of double operand; now we have to identify the 30 cases of single operand. In the case of single operand, the operand field need not have two subfields. This has only one operand, that means, only one data; and what is that data? It will be destination data. So this particular 4 bit is available now to extend the opcode.

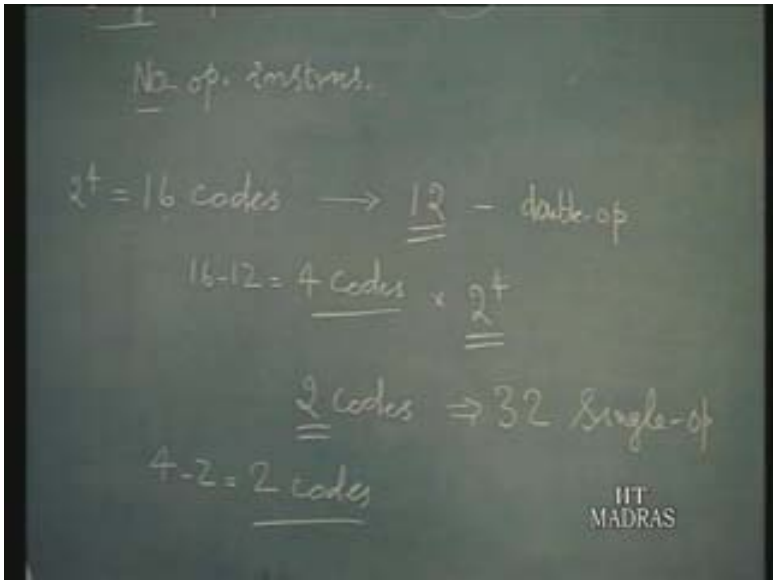
So now, from the opcode subfield, we are extending the opcode into the source subfields. Why? Because the 12 operands and its double operand instruction have already been identified and 12 codes of these 16 possible things have been used. We are only left with 4 codes; so with 4 codes we have to identify 30 single operands and also no operand case.

(Refer Slide Time 44:16 min)



Since for single operand we do not need two operand field, that is, two data field, this source data field now can be used for extending the opcode. That is what we are going to do. The source data is of size 4 bits; now for each of the 4 codes we have been left with, taking the source data field four also, 2^4 or 16 would be possible. For each of these 4 codes, which means for each of these 16, single operand instructions can be identified, which means of these 4 codes we need 2 codes so that we can accommodate these 30 single operand instructions.

(Refer Slide Time 48:39 min)



What is possible with each of the codes – 2^4, which mean 16? With two codes, we can accommodate as many as 32 single operand cases. We have been asked to accommodate for 30 actually. So really, with two codes in the opcode field and making use of the source data field, getting 16 is possible. So in just two of the opcode fields, we can accommodate 32 single operand cases; actually as I said, we are extending the opcode field in to the source data field. So 32 more remain, but then we cannot use because of the condition. Let us see the chart again. The problem says the opcode field must identify the three groups. Now we know that in the opcode field in our case two of the codes in the opcode field will identify the single operand. So now out of 6 codes, 12 have gone for double operand the out of the remaining 4, 2 are going for the single operand, so we are left with 2 codes; 2 codes are available in the opcode.

In the case of no-operand instructions even the destination field can be used. Now what do we have? Two codes in the opcode field; let us see the chart. You have 2 codes available for single operand and for each of those codes, 2^4 or 16 will be possible, for each of these no-operands. So this also can be used for no-operand instructions. For each of these, another 2^4 will be possible. So we have 2 codes from this side and then we used 2 in the other one. Let us see the chart again. There are 2 codes, then for each of these codes we have 2^4 or 16 will be possible for each of those. In the destination also, because it is no-operand, another 2^4 or 16 is possible. This particular 4 comes because of the source data field. And for each of these, there is another one for this concept of the destination data field. Now these 2 codes will uniquely identify the no-operand cases. That is the important thing.

(Refer Slide Time 51:27 min)

Now see that chart again – there is a condition if the opcode field must identify the three groups. They will, because 12 of the codes will identify the double operand cases, 2 of the codes will identify the single operand cases, and the remaining 2 will identify the no-operand cases. Now we find that for each of those 2 codes there is no-operand cases; for each of these 12, there are so many possible combinations. This is how we have to work it out.