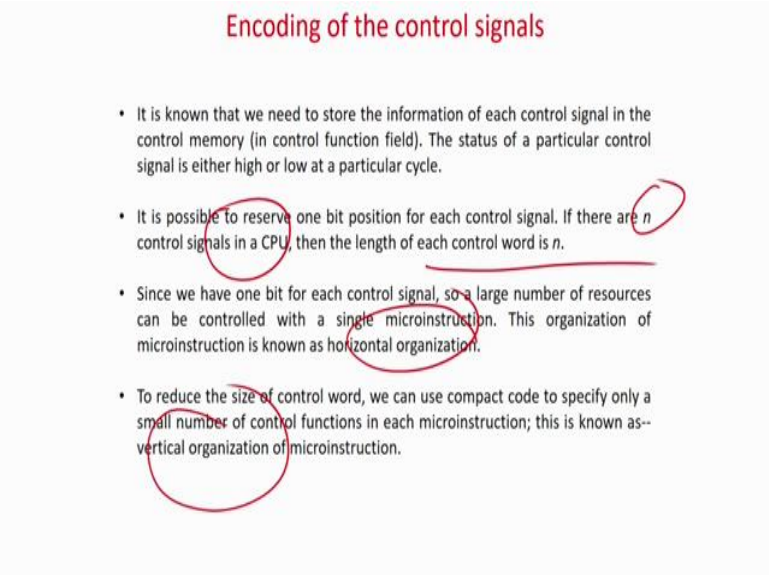
(Refer Slide Time: 15:54)

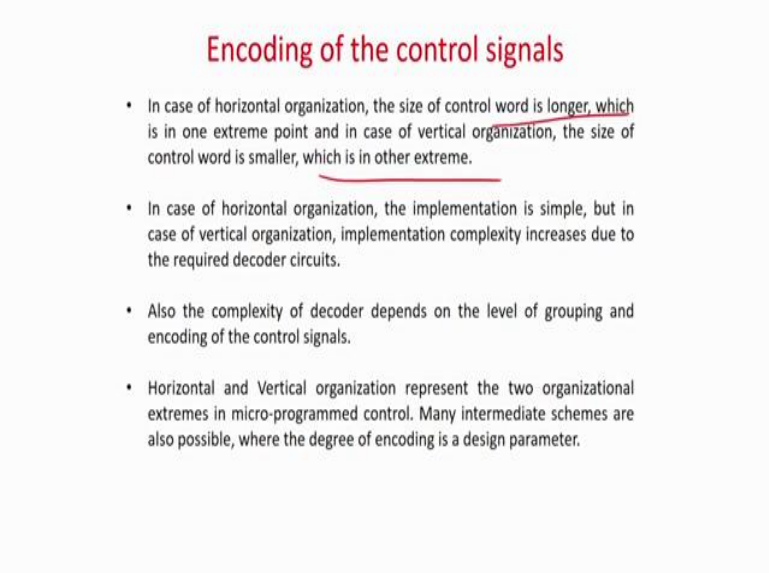


So, this is just again very quickly very quickly let us look at now we are going to take a more elaborate depth with more clear examples that, optimization is done on the signals by encoding. So, last unit we have already dis discussed that encoding is very important. So, if you encode basically what is going to happen? If there are n control signals in the CPU, the length of each control word of the memory will be n very simple there is a horizontal micro program everything is parallel. This is actually a horizontal micro program and it is going to be very fast, but the memory is unoptimized.

To do this we have seen that in the last unit, that we have to optimize by encoding the signals, which is actually called the vertical micro instruction. So, if you just go for a very flat compression that is from 2^n, we just compress it to n or from n to $log\ n$ and using n to 2^n decoder, then actually full compression is there and this is actually called a full vertical micro program. But in this case only one bit signal can be made a 1. So, if simultaneously 3 or 4 points to be a 1 you require 4 cycles for that.

But we have seen that this is not a very very good way of solving the problem, because it makes the memory very small, but it will take very long time to solve the problem.

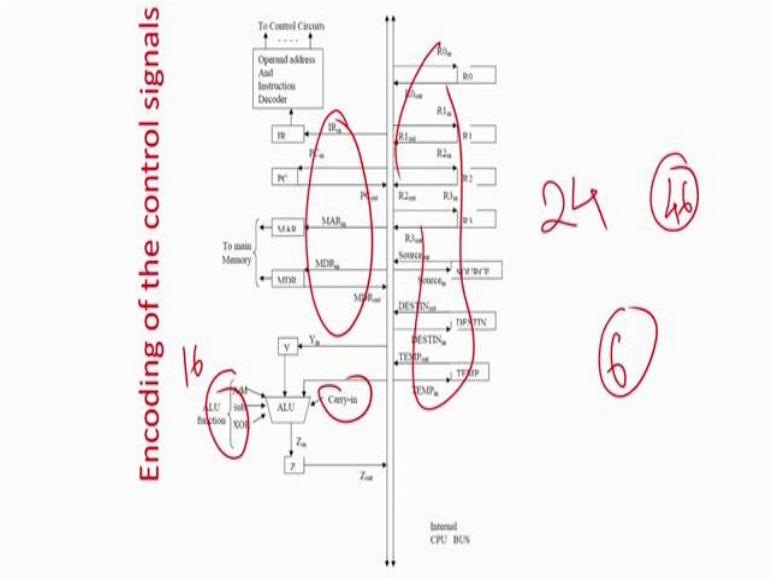
(Refer Slide Time: 17:03)



So, what we basically do is that, we go for basically something called a hybrid approach in hybrid we make clustering and for each clustering we will try to put signals in one cluster, which need not be 1 simultaneously. So, that is one idea of actually called a hybrid micro program.

So, that approach actually we are going to see with a more elaborate example, because in the last unit we have discussed the basics. So, this slide tells about the horizontal micro program that it is one extreme, and it is basically it is longer and in the vertical micro program it is highly compressed. So, it is the other extreme. So, whatever I was discussing is basically written in this slide, you can over go through this the theory part of it, now this is very important.

(Refer Slide Time: 17:46)



So, we are taking a single bus architecture, now we will mainly focusing on example because theory mainly we have covered in the last unit. So, if you look at it, this is a single word single bus architecture. So, in this we are trying to will try to exam try to find out basically how we can optimize the vertical and horizontal micro program taken together, that is your basically a hybrid; hybrid kind of architecture we will consider for the micro program memory and we will be taking a single bus architecture to exam use it.

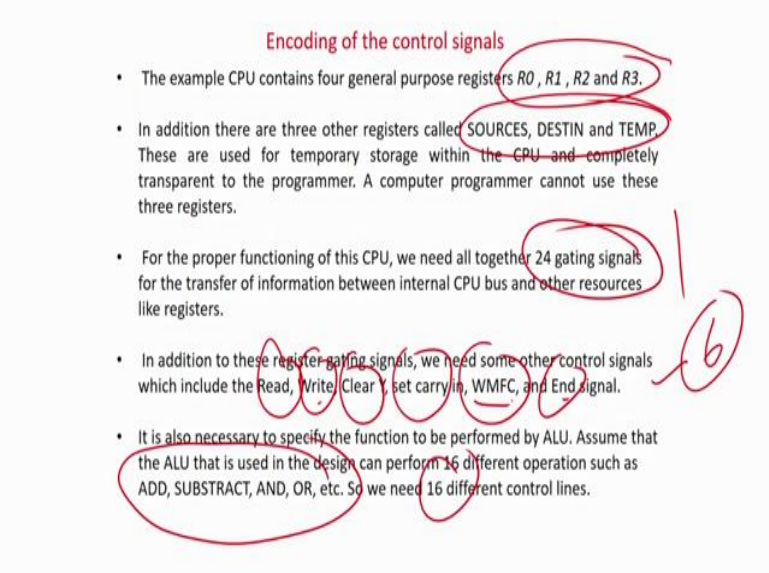
So, what is an assumption here we assume that here there are 4 temporary registers $R0, R1, R2, R3$. So, the number of signals are 0 1 in out, in out, in out. So, 4 registers are there and in and outs are there. So, just remember how many signals are there 8 signals are there then there are 3 system registers, which a user cannot use and they are used for some system storage; so 1, 2, 3. So, that is temp, destination and source some registers are there.

So, how many registers total? 1, 2, 3, 4, 5, 6, 7 and for each register you have a input and output port. So, it is 7 into 2 14 signals are there just keep it in mind, then we have a instruction register. So, instruction register always takes the input. So, IR_{in} is there that is 1. So, 14 plus 115 signals are there, program counter in and out 14, 15, 16, 17; memory address register in. So, it is 18 14, 15, 16, 17, 18 memory data register in and out 18+ 2 20, then this is your CPU. So, you can see Y_{in} then basically your functions that will be again some control lines for the functions and then basically Z_{in} and Z_{out}.

So, let us count how many input and output ports are there. So, 1 2 there are 7; 4 user registers, 3 system registers 7 into 2 14 then 15 16 17 18 19 20 21 22, 23 I miss something 1, 2, 3, 4, 5 6, 7 8, 9, 10 sorry 10; 10 over here, and 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 20 actually there are 24 signals will be there you just count I mean I missed something.

So, there are 24 input output ports you will find out 14 there are 14 over here 15, 16, 17, 18 19, 20, 21, 22 23, 24. So, just count over here so this input output ports if you look at you will find out that there are 24 input output ports the signals I will show you. Along with that; that means, there are 24 ports to be controlled which corresponds to input or output or any register like $MBR, MAR, R0$ to $R4$ source destination Y, Z, Z_{in}, Z_{out} if you count there are 14 sorry 24 signals which correspond to that those values and therefore, also there should be some signals which will tell you; that means, there is to be add, subtract, multiply. So, those signals will also be there.

(Refer Slide Time: 21:07)



So, now, we are we will try to see based on this architecture how we can optimize based on cluster. So, assigning individual bits to each signals leads to a long micro instruction because here we have counted that there are 24 input output ports then for some ports some signals will be there to control the ALU, that also you have to add along with that you have to have some signals like $select, add, select, W$ for read, that is memory read or write there should be one signal then the wait for $WFMC$ will be one signal. So, there are several other signals and the count will be quite large.

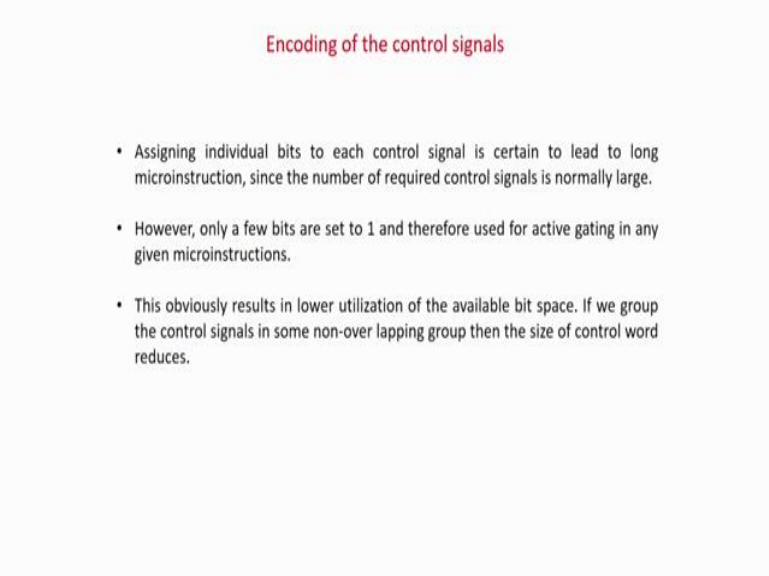
So, let us see what is the count. So, we will see the count in the next slide, but if you take all the counts and put in the flat architecture the size will be very high already we have discussed so many times, which will be a horizontal micro program. So, this will; obviously, lead to lower bit space utilization, which is the case in horizontal micro program.

So, as I told you we are assuming that there are 4 user purpose registers. There are 3 source there are 4 user purpose registers, there are 3 central registers and basically as you have counted corresponding to the like a memory address register, memory data register, Z, Y. So, there are 24 gating signals as we have counted in the figure, along with that basically read, write, clear, set, carry in some signals $WMFC$, End.

So, the signals can be more, but in this example we are assuming that these many other extra signals are there like read, write from the memory. Clear means clearing some value of Y in a register, set carry in that is the carry in of the ALU, wait for the memory and end. So, these also actually count from to 1 2 3 4 5 6 there are another 6 signal are there. 24 signals were there for the input output of the registers which we have counted in the bus architecture.

And let us assume that there are 16 different operations which the ALU can do, like add, subtract, and, or, compare. So, many other signals are there let us assume that 16 different functionalities are possible. So, there will be 16 different lines.

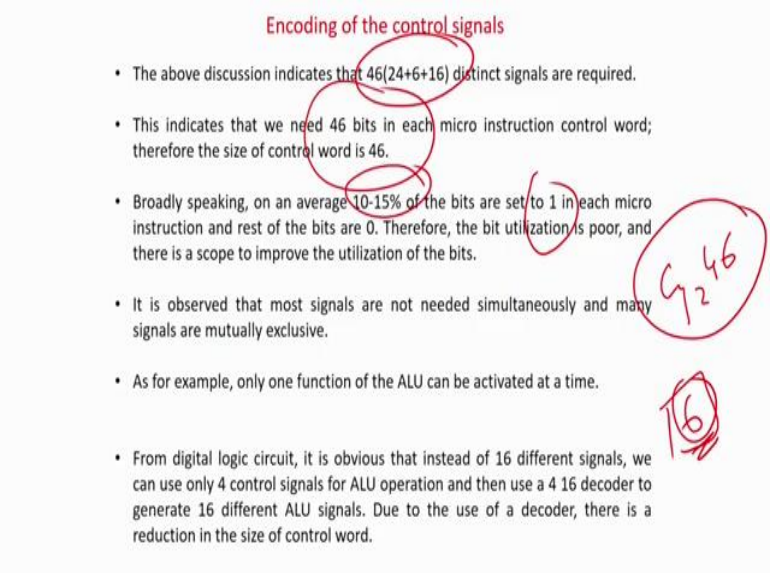
(Refer Slide Time: 23:05)



Just like again I am going back; so there will be 16 lines over here and if you take all these control lines for the registers there will be another 24, and there assume that 6 more control lines are there which corresponds to read, write, carry in ok. So, they are corresponding to $read, write, carry\ in, WMFC$. So, another 6 more signals are required for some other interfaces. So, together there will be 30, 46 signals required if you want do it in parallel.

So, the memory word size will be how much? It will be 46 bits in a horizontal micro program.

(Refer Slide Time: 23:38)



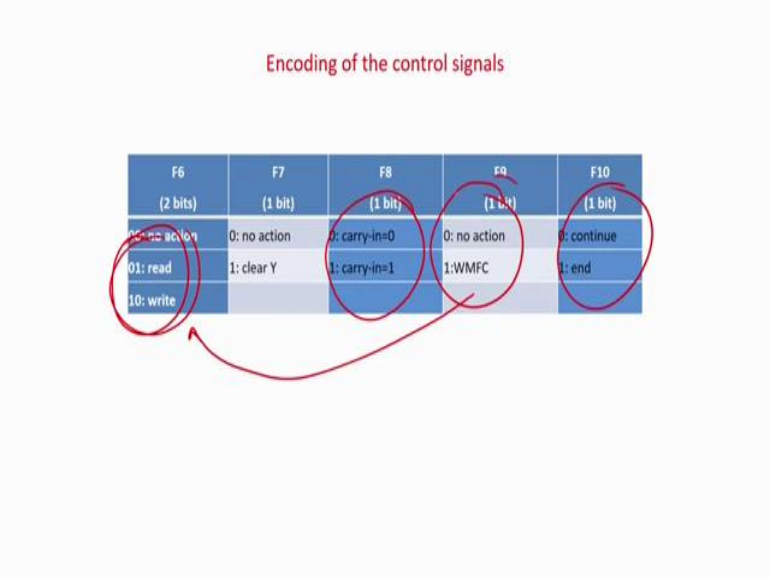
So, therefore, it says that there is 46 different signals are there and therefore, basically 46 will be the length of the micro program and it has been found theoretically that only 10 to 15 % of the memory positions are 1. So, if we have such a big array whose size is 46 and it will be long it will be long depending on how many micro instructions are there, and only 15 % is actually filled with 1 is a huge waste of memory that is why we go for either vertical micro program or a vertical micro program is very slower, because in this case it will be log 2 46. So, I think around 6 bits will 2^5 is 32. So, 2^6 is 64. So, 6 bits will actually solve the problem for you.

So, if I go for a full encoding architecture. So, it will be 6 bits which will encode and then you can use 6 is to the 64 bit architecture using a decoder and the job will be done, but in this case it will be very very slow you have simultaneously only 1 bit can be made 1, so in fact, it is not a very good idea to solve the problem. So, we can may go for a clustering approach which is called the hybrid micro program. So, that is what actually we are going to study.

(Refer Slide Time: 24:45)

Before that how to make a cluster and what will be the idea for that, that we have to properly think and we have to make the cluster. So, if we go for a flat architecture that is each bit can be in one cluster that's horizontal micro program one extreme and if I go for full encoding that is 6 bits will be required to encode all the 46 bits, and it will be a very complex architecture, but at a point only 1 bit can be a 1 which is actually written over here. So, it will be very very slow. So, now, let us try to think how we can make clusters.

(Refer Slide Time: 25:15)



Say for for the timing just assume that there is F1, F2, F3, F4, F5 some 10 clusters they have thought about. So, I will try to see think some basically of some thumb rules we make the cluster. For example PC_{out}, MDR_{out}, this circle if you see this cluster all the outputs of the registers I put in 1 cluster this is because it's single bus architecture. So, simultaneously $R1_{out}, R2_{out}, R3_{out}$, source out cannot be done it will leads to a conflict that is if you look at this if you look at this figure.

So, on it's a single bus architecture. So, all these registers all these 24 story I was talking about only 1 bit can be out at a time, of this single bus that is the problem. So, it is very wise that you put all the basically out signals in 1 cluster; that means, idea is that in 1 cluster only 1 bit can be made 1 at a time. So, there is no problem 1 2 3 4 all these outs you actually put in 1 cluster, then your job is done because simultaneously only 1 bit can be 1 and that is what is required because simultaneously R2 out and R3 out cannot be made 1.

So, actually how many such instructions are there? 1 2 3 4 5 6 7 8 9 10 11. So, if you calculate there will be 1 2 3 4 5 6 or some fifth for 10 of 11 out signals for the registers. So, how many bits are required? You will require a 4 bits to solve the problem. So, you require a 4:16 decoder for this and in this case I mean because the full decoder will not be used because you don't have 16 such signals.

So, you can just say 000 for no transfer; that means, nothing will happen and all the other bits are actually don't cares in all the cases there will be no output, but if it is 001 *PC* out will be 1 if it is 0001 then MDR_{out} it will be 1 and your job is done.

Another important cluster is actually I will put because I told you there are actually 16 different functions of the ALU. So, I can club all the controls in 1 decoder in one cluster because simultaneously I cannot have add and sub simultaneously only one can be high. So I actually cluster everything make a 1 cluster your job is done. So, this one 2 very key rule that all the outputs basically you put in 1 cluster, all the CPU function will be put in 1 cluster because simultaneously 2 cannot be 1 and simultaneously ALU cannot do operation.

But this logic only holds if it's a single bit architecture bus architecture. It is a multiple architecture then 2 bits also can be simultaneously 1 and then again you have to break it up in the cluster. So, in this discussion we are keeping these things simple. So, in a single bus architecture this is one of the very good way of doing.

Now, read. So, read can be multiple, because it can happen that there will be only one output, but simultaneously 2 or 3 guys can take the inputs together. So, based on that you can make some architecture or clustering in this case they have put PC_{in}, IR_{in}, Z_{in} all the registers in 1 cluster; MDR, MAR in 1 cluster; $Y, source\ in$ in 1 cluster. So, this clustering they have made based on some philosophy that is they have thought that like they have said thought that PC_{in}, I $IRin$ they are moving they are keeping it in basically 1 cluster so; that means, they have found out that both reading the value of *PC* that is loading the value of the some loading the value updated value to *PC* and loading the IR generally does not happen in 1 cycle.

So, they have put it in 1 and they have also thought that in the same cycle register when you are updating the *PC*, before or a instruction is loaded into the instruction register $R0, R1, R2\ and\ R3$ basically do not read at the same cycle because when a *PC* is updated, generally the temporary registers do not read the values.

For example when you are dumping the value in of Z that is the output of the ALU goes to Z and then in the next cycle only it goes to the different registers. So, when you are updating the output of the ALU in Z and that cycle basically the registers don't get updated. Because first the output of the ALU goes to Z and then in the next cycle it actually goes to the registers. So, they have kept these all the R_{in}s basically with Z_{in}.

But that is why they have made 1 cluster, and they have found out how many instructions are there 1 2 3 signals are there 1 2 3 4 5 6 7. So, 7 of course, 3 bits are required and 1 may be a don't care case, but they have thought somehow that basically it may happen that some *PC* output basically generally goes to memory address register in. So, that they have put in over here. So, PC_{out} generally goes to memory address register.

But they also have thought that sometimes you may read the value of program counter in terms of temporary registers for some storing the value or starting the current instruction or you can also think that means basically register value out can go to R_{in} and also it may go to basically memory address register in if it is an indirect addressing kind of mode.

So, it may happen that the output of basically 1 register can go to multiple registers or the *PC* can go *PC* output can go to a temporary register, address it can to the memory address register. So, in that philosophy they have decided to put basically the user registers over here and the memory address register or the memory data register or $TEMP_{in}$ in another cluster. That means,

basically simultaneously 1 register can be made 1 and 1 register can be made 1 here and even 1 register can be made 1 here the input.

So, simultaneously some output can be fed to 1 register over here, 1 register over here and 1 register over here that is where you require simultaneous input it's possible more likely you that why divide the cluster. If you would have thought ok just like your output I just think that only 1 register can be input at a time.

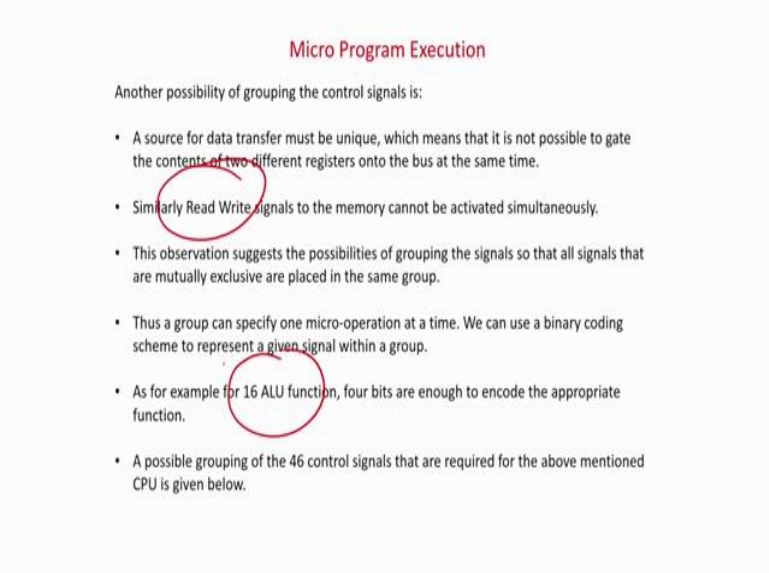
So, you can we could have put all these in another cluster and then you could have saved a space, but here they have assumed that I am making the cluster based on simultaneous inputs which is possible to the registers. So, based on that philosophy they have divided the signals into different clusters corresponding to that single bus architecture. And there were some other control signals as I told you that there is something called read. Again I am putting the read in a single cluster because the memory can be read or write at 1 go and similarly like this is the carry in you are putting in 1 cluster because carry can be either 0 and 1 and so forth that is wait because wait can be simultaneously 1, with anything else you can put it 1 and then end and continue are the 2 signals.

So, these are some of the other controls signals for the memory and wait for memory cycle etcetera because if I want to put this one over here this is not a good idea, because generally we give read and write and simultaneously also we require to wait for the memory signal to be ready. So, we put this read write in 1 cluster and WMFC another.

So, this example actually gives you a broad in the last unit we actually gave you the philosophical idea that why clustering how it can be made as clusters, and why you require to put everything in 1 cluster, if you put it will become a full vertical micro program very slow, if you make everything single signal cluster it will be a horizontal micro program very fast but very inefficient, here we have given a single bus architecture a real example with each bits, signals and registers and ALU.

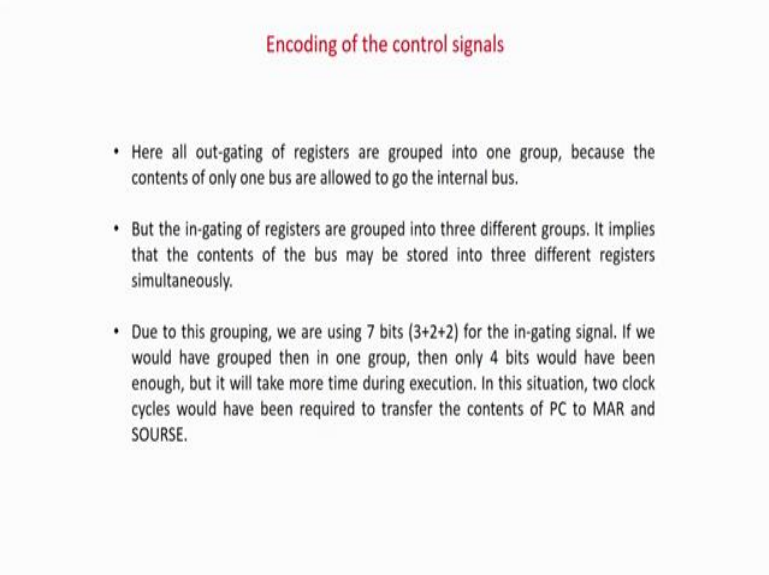
So, 1 pure example or a complete example we have taken for a single bus architecture and then we have made the cluster for all the signals based on the philosophy or clustering; so this one actually examples the theory we discussed in the previous unit in a more practical example.

(Refer Slide Time: 32:38)



So basically so whatever I have discussed is written in this slide, you can go through here basically why I put read write in 1 point, why this 16 ALU functions I put in 1 cluster. So, whatever I was telling things I have kept it in this slide so that you can read it offline.

(Refer Slide Time: 32:55)



So, now, what is the gain? So, if you see the gain. So, how many bits are required 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21. So, 21 bits are required to solve the problem, we saw the 46 bits required in the extreme flat case and if you look at it how the way we have clustered it

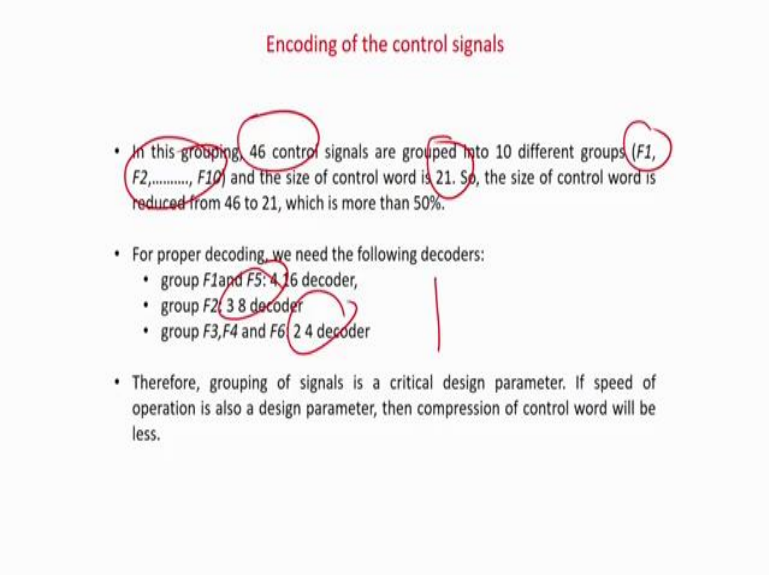
very very few cases there will be contention at very very few cases you can have both R2 and R2 in together in that case you have to split it.

But mostly basically we will have as similar performance as a fully horizontal architecture and still you have saved a lot in the space of the memory. So, basically if you also I mean that is that is what I have to tell here all philosophy is extended. And please go through the slide to find out what is the basically I have told about this grouping and what are the advantages.

So, these 2 slides this slide and this slide tells what are the advantages and what are the gains and what with drawbacks of the basic clustering philosophy we are taught for this example, you can go through this slide offline and get the values.

So in fact, they have find out basically.

(Refer Slide Time: 34:04)

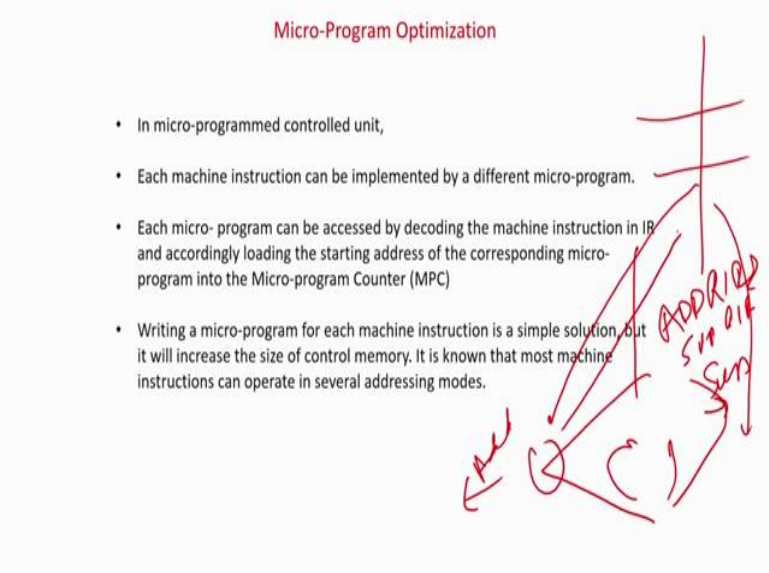


So, I mean instead of 40 signals we are basically now F10 value number of word is 21. So, huge saving space, but of course, some additional things are required we require some decoders over here also some decoders are also required like for some groups basically, we require 4:16 decoder, for some group 3:8 decoder and for some case 2:4 decoders.

So, not only you have to find out the cost of the basically your memory is saved that is true, but also you have to account for the decoders are also taken into picture and also they are maybe slightly slower in speed corresponding to a horizontal micro program, but that will not be very very high, because the way intelligently we have actually placed the signal in different clusters.

So, till now if you have looked actually we were discussing how to optimize the micro programs and in the first part actually we have seen, that how you can optimize it based on actually compressing in the length of the control memory width. So that we can encode some signals based on clustering, how we can optimally select which parts has to be encoded and so forth.

(Refer Slide Time: 35:07)



Now, we are going to see another interesting way of optimizing the micro program how? Basically by compression in this manner; that means, if you have some certain different instructions, you can easily have 10 different micro program micro programs and load them and execute accordingly. But you have might have observed that in most of the cases or in fact, in the all most all cases basically the fetch part is same.

So, and for example if you have a instruction called SUB and you have a instruction called ADD maybe $add\ R1\ or\ R2$ and $subtract\ R1\ and\ R2$. So, you can feel that most of the micro instructions and the signals to be generated will be same except in only 1 case, then in one in case of add the arithmetic logic unit has to be configured to add and in case of subtraction is just to be configured into sub mode. So, only one bit will be changed or one control signal will be changed all others will remain similar.

So, if you have 2 different micro programs for this 2 micro instructions. In fact, that will be wastage in the amount of memory space which will be required in the micro program control memory. So, how can we optimize that? So, basically we can optimize by actually having a

single micro program for most of the common instructions, and then diverging out for the different parts and again coming back from where we can start.

For example if you had a single program called $ADD\ R1, R2$ and maybe $SUB\ R1, R2$. So, most of the fetch etcetera loading from $R1$ to $R2$ saving the value of $R1 + R2$ or $R1 - R2$ to $R1$ will be similar. Only in one case basically you will find out that you have to add signal equal to 1 and in this case you have to make subtract signal equal to 1. So, only may be only 1 line of instruction will be only 1 memory word will be different for the 2 different macro instructions and all other will be similar.

So, what we can do? We can write a common micro program, the fetch part will be similar maybe all other maybe some register loading will be similar just after that there will be 1 signal which one will correspond to $add = 1$ another branch will be for $subtract = 1$ and again after executing then you can again come back here, which will lead to storing the value of the ALU to register $R1$ that is the destination.

So, therefore, the second way of optimization is writing same micro program for different macro instructions which is of similar type, and then moving here and there as the branch instructions based on the difference of the control signal that is required for that macro instruction.

So, in a very broad sense you have some different macro instructions club them together, write a single micro program and then depending on the exact micro program being exact macro program being executed, which will be told to you by basically the instruction decoder that is from the instruction register and the instruction decoder will tell you exactly which is the macro instruction being executed like add or sub, based on that you jump to the location which corresponds to the independent or the different part of micro instructions corresponding to the macro instruction execute the different part and again come back and execute the common part. So, basically that is another way of optimizing the micro program memory, which we are now going to discuss basically.

So, whatever I told you basically is given in this slide, it says that we can implement different instruction by different micro, micro program, but that will be very un optimized. So, basically this is the simple solution, but will increase the number of memory. Because we know that there are so many different addressing modes, so, many different type of basically macro instruction and so forth.