

Computer Organization and Architecture: A Pedagogical Aspect
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 22
Organization and Optimization of Microprogrammed controlled Control Unit

Welcome to the unit 8 on the Module of Control. So, what do we are discussing as of now in this module that how basically we can execute the controls, if the methodology is a micro program control. So, in the last unit basically we have seen the very basic idea of a micro program control unit that give in mainly we have discuss only the fetch part of the instruction. In fetch part of the instruction we have seen that what are the corresponding signals, and how we write it in a micro program control memory, and how we go through them.

Then we have discussed how basically we can optimize it, because in case of a micro program control memory we just give the signals which has to be made 0s and 1 and we write into the memory. And then what we do basically one after another we keep on fetching them and we are putting this those output of the memory are directly given as fed feed to the ports.


But then we have seen that lot of zeroes and lot of zeros in this memory then we have seen how to optimize them using a vertical and hybrid micro program. But in this basically what we will try to do we will be mainly focusing on optimization of micro program control unit in more depth. And by taking more specific examples, and also another focus will be we will try to see the execution of a full instruction. Because in the last unit we just saw about only the fetch part of it, that is this unit is basically an extension of the previous unit. In fact, the micro program is slightly larger topic so we have dedicated two units for that.

So, this is the part we are going to cover and then as a pedagogical method, so we will see what is the summary of this units.

(Refer Slide Time: 02:01)

Units in the Module

- Instruction Cycle and Micro-operations
- Control Signals and Timing sequence
- Control Signals for Complete Instruction execution
- Handling Different Addressing Modes
- Handling Control Transfer Instructions
- Design of Hard-wired Controlled Control Unit
- Micro-instruction and Micro-program
- **Organization and Optimization of Micro-programmed Controlled Control Unit**
- Different Internal CPU bus Organization

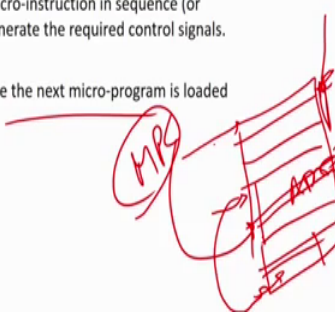


(Refer Slide Time: 02:02)

Unit Summary

A full program written in terms of machine instructions is executed as follows. *Fetch*

- For each machine instruction, when it is in the decoding cycle, based on its OP Code the corresponding micro-program is loaded into the control memory. *OP*
- The micro-program counter fetches each micro-instruction in sequence (or jumps to the required location) and they generate the required control signals.
- Once the present micro-program is complete the next micro-program is loaded based on the new instruction. *MPC*



So, basically a full program which is one of the main emphasis, how to see a full program which is written or and or the idea we will get that how one or after the macro instructions are executed in terms of a micro instructions when it is a micro program concept. So, a full program is written in terms of machine instruction that is macro program how it is executed. For each machine instruction where it is in the decoding cycle based on the Op-Code, the corresponding micro program is loaded into the memory. So, for first stage they have means very simple that is fetch.

So, already we have seen that for each fetch, the corresponding three micro instructions are executed, after that basically it waits; that means what. So, whenever a new instruction has to be executed, the micro program counter PC is pointed to the instruction which corresponds to basically your fetch. So, if you think about this is your micro program memory, maybe these three are dedicated for fetch. So, whenever new instruction is macro instruction is to be executed the micro program counter will point to this. After it has done it will come to the end of the fetch part.

Then what happens? Then the instruction register actually decodes what the exact instruction is if it is add, if it is stored, if it is load accordingly the MPC will be pointed to different part. Maybe this part of the memory corresponds to add. So, it will jump over here. Maybe it is for subtract or multiply. So, if it is multiply, so it will from here it will actually jump to here; that means, based on the instruction register based on the macro instruction being executed, it will actually load the micro program, and PC corresponding to the micro program for that instruction, like this maybe for add this maybe for multiply. Then, it starts actually the real execution of this instruction in terms of it is micro program. So, this one thing we are going to see for some template instructions.

The micro program counter fetches each micro instruction in sequence and the generate the required signal or if required jumps will also be there. So, but basically it goes to this point and to add or respective instruction to be executed corresponding to instruction register which is being decoded, that is the real execution starts, it will keep on doing it till it gets ends.

So, once the present micro program is complete, the next micro program is loaded based on the new instruction. So, once this add has been done then again automatically they it finishes, then automatically the micro program will start pointing to this part, which is actually again fetch. So, it will actually fetch the new instruction.

So, now the MPC this program will be executed. So, from my MPC will be pointing out to this program memory which corresponds to fetch. So, automatically again new instruction, new macro instruction will fetch and the procedural will go.

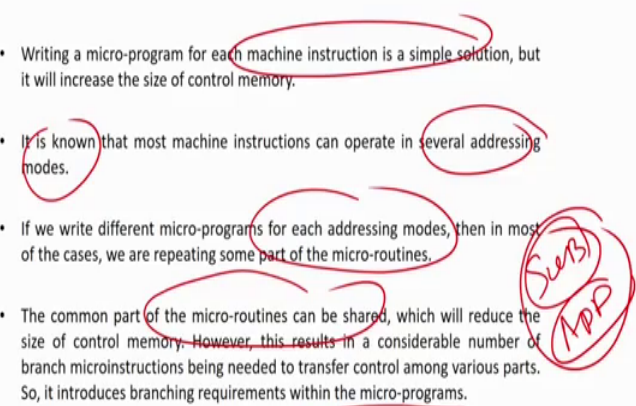
So, there are basically three steps; when a new macro, macro instruction has to be executed; it will first load the MPC correspond to the micro program memory address, control micro program control memory address, which correspond to fetch. Three instructions will be executed. Then again the MPC will wait to get a new value. So, where the value will come from, the instruction register by the time has decoded the macro instruction and corresponding to each macro instruction like add, multiply, store over different address formats it would actually point to the corresponding micro program in the memory.

Then it will keep on executing that, and once it is done the whole instruction is completed and a new macro instruction has to be executed. Then again the fetch part start and keep on going in. So, that is means actually what is the idea of total program execution in terms of a micro program control.

(Refer Slide Time: 05:32)

Unit Summary

- Writing a micro-program for each machine instruction is a simple solution, but it will increase the size of control memory.
- It is known that most machine instructions can operate in several addressing modes.
- If we write different micro-programs for each addressing modes, then in most of the cases, we are repeating some part of the micro-routines.
- The common part of the micro-routines can be shared, which will reduce the size of control memory. However, this results in a considerable number of branch microinstructions being needed to transfer control among various parts. So, it introduces branching requirements within the micro-programs.



So, now you can do it? A very simple solution; for each machine instruction you write micro program. For fetch it is 1, because and then for add you write 1, for load you write 1, for subtract you write 1 and you (Refer Time: 05:46) you can take a very huge micro program memory from the fetch part will be there, after that there will be add, then again for the next instruction they will be again fetch and again the corresponding real execution micro program will be there and you keep on doing in; it will be a huge code with lot of redundancies.

So, what is the solution? That basically because there are so many different type of addressing modes etc. So, and if you write different micro ports for each addressing modes each type of instruction type so it will be a huge memory, but you can do it will solve the problem; depending on the IR decoding you can directly go to that instruction and execute it, but that is actually going to a very very unoptimized solution.

So, what we do basically we try to keep something in common for example, the fetch part is common to all. So, there will be only a 3 bits or 3 word only for fetch and then for example, if we have subtract and say add 2 instructions are there. So, only difference will be basically is abstract we will make the ALU to sub subtraction will require the signal which corresponds to the ALU sub should be mid one and in case of add the control unit control input to the ALU, which is corresponds add will be 1 other than that whole the program will be similar.

So, therefore, basically we can write a single micro program for add and sub with just branching instruction. That if we find out the instruction decoder is saying for the sub,, then slight the slight change will go to one part and if it is add to just go to one part then the other part can be common, so that what we will actually save in the micro program memory size.

So, that is the common part of the micro routines can be shared which will reduce, but in that case we will be required lot of branching part, but that is. However, is that is whatever is common you put together and whenever you have to go to some other part of the other instruction, which is uncommon between them, as I told you in this case only one word will be different, in which case how equal will be equal to 1 and in which case the add will be equal to 1, because they need to configure the ALU in different mode only that that instruction will be different. So, you can easily write a branching program which will do accordingly. So, basically micro programs are written in a branching fashion.

So, that is one of the key idea of these unit which is be starting to look at basically. We will take a full instruction and see how it is executes and also at the same time we are also going to study how basically this branching should happen and at the same time. I will also try to see that basically we will take one practical example of instruction and we will also see how we can practically optimize the instructions based on clustering,

because in the last unit we have just give an idea of clustering, but here we are going to take a much more elaborate look at how the optimization can do. So, therefore, there are the two basic things we are going to focus here.

(Refer Slide Time: 08:34)

Unit Objectives

- **Comprehension: Explain:**--Explain about the branch control mechanism in micro-program.
- **Evaluation: Estimate:**--Estimate the size of control store to implement the control unit.
- **Application: Demonstrate:**--Demonstrate the impact on performance of the control unit depending on the format of control word.

So, what is the objective of this unit because as we are following a pedagogical approach. So, we for each unit we actually define the objectives and in the end we try to see whether the objectives have been made. So, what is the objective is the comprehension objective explain about the branch control mechanism in micro program.

That is very very important we should this should be able to do it because, without branching it will be a very very unoptimized solution, and in fact, if there are branch instruction; obviously, micro program has to be have follow a branching part and not only because of the inherent program should have branching there should be a branching in the micro instructions, also because I told you in micro program lot of parts of different micro programs corresponding to different macro instructions will be similar. So, you have to branch in between them to optimize the memory size.

Then estimate the size of control unit to implement the control store to implement the control unit, that is you have to estimates the sign of the control you have to estimate the sign of the address part also we have to find out the how many signals are there. So, all these bases on a certain architecture of a single bus or multi bus, you should be able to. Mainly we will be concentrating around a single bus architecture, you should be able to

estimate what are the different bits required to store each part of the control memory; that corresponds to the control signal generation, then the condition check as always the branching address check. And as an application objective demonstrate the impact on performance of control word depending on the format of the control word; that means, performance based on the format of control.

So, example if you have an compress format. So, it the control unit will (Refer Time: 10:05) it is a very flat like of horizontal a format, that will be very fast. So, you will be able to demonstrate the efficiency based on the format of the control units, format of the control word basically; that means, whether it is a full horizontal vertical or hybrid. So, these are the basic three objectives.

(Refer Slide Time: 10:19)

Micro-program control for program execution

- For each instruction of the CPU, there is a corresponding micro-program to generate the required control signals.
- Each micro-program is a sequence of micro-instructions. A micro-instruction is nothing but the combination of 0's and 1's which is known as control word.
- Each position of control word specifies a particular control signal. 0 on the control word means that a low signal value is generated for that control signal at that particular control cycle; similarly a 1 indicates a high signal.
- The micro-programs are stored in micro-program memory (also called control memory).
- Since each machine instruction is executed by a corresponding micro-program, it follows that the starting address for the micro-routine must be specified as a function of the contents of the instruction register (IR).

So, let us start with the unit. So, basically what happens as we are discussing, for each instruction of the CPU there is a corresponding micro program for generating the control signal. For each macro instruction there will be a micro instruction there will be a micro program. Each micro program is sequence of micro instructions, and it is nothing but zeros and ones which is in a memory as we have already seen. Each position of control word specify the particular signal and this placed in memory. So, it can be 0 and 1 and you can directly take the memory word out control memory word out and give it to the respective ports. So, that basically they are actually called the control memory and we generally called is a micro program control memory.

Since this machine instruction is executed corresponding to a micro program, it starts to follow the starting address of the micro routine as specified as a function of the contents of IR this is very important; I was as a say that there are different macro instructions what the macro instruction should do will be decoded by the instruction decoder taking the values from IR. So, based on the IR value instruction decoder value we should be able to point out to location in the main in the micro program control memory, which correspond through the micro program for that macro instruction.

And as I told you generally we are trying to have you do not try to keep this try to keep this control memory very long, by giving separate micro program for each macro instruction, whether we try to write a common program and for each of taking set of macro instructions by trying to keep the common part similar and if there is wherever there a diversions we can put jumps, so that how basically words.

(Refer Slide Time: 11:53)

Micro-program control for program execution

- To incorporate the branching instruction, i.e., the branching within the micro-program, a branch address generator unit must be included. To incorporate the conditional branching instruction, it is required to check the contents of condition code and status flag.

A full program written in terms of machine instructions is executed as follows.

- For each machine instruction, when it is in the decoding cycle, based on its OP-Code the corresponding micro-program is loaded into the control memory.
- The micro-program counter fetches each micro-instruction in sequence (or jumps to the required location) and they generate the required control signals.
- Once the present micro-program is complete the next micro-program is loaded based on the new instruction.

Handwritten notes: "branch" with an arrow pointing to "branching instruction"; "main IR" with an arrow pointing to "condition code and status flag"; "Inherent" with an arrow pointing to "OP-Code".

In fact, as I told you jump branch to incorporate branching, here also you will require to check the contents of the code control status flag etc. So, that is already we have seen in the last unit then basically the if we have jump instruction is there, you have to check some control fields as well as many control can be from this status as well as the from some signals actually which are coming from the IO devices. So, that is very similar to a macro program much.

But actually here branching is 2 that is one thing you have to emphasize; one is a normal branch, but none one branch means basically what corresponds to the macro program. So, if the macro program says that you have to do a branch based on the condition, micro program will branch correspondingly.

Another is micro program inherent branch; that means inherent branching. So, why this inherent branching is there? Because I told you because if there is add instruction or sub instruction, we do not required two different micro programs for that we will have say, but depending on the if is either a subtract we actually branch to a part which is the only uncommon part and then again come back and follow the common part it.

. So, there is some inherent nature of optimization of the micro program. So, that you require inherent jumping. So, that we can have a common program and jumps are done only when the instructions are the different part we can do. So, we can have a similar same micro instruction for both add subtract load may be (Refer Time: 13:14) are the common instruction. So, therefore, we require many branches in a micro program execution compare to a macro program.

So, basically a full program written in terms of machine instruction basically execute as follows. For each machine instruction that is the macro instruction where it is in decoding phase, as I already told you the fetch is over first is the fetch phase.

(Refer Slide Time: 13:34)

Micro-program control for program execution

- To incorporate the branching instruction, i.e., the branching within the micro-program, a branch address generator unit must be included. To incorporate the conditional branching instruction, it is required to check the contents of condition code and status flag.

A full program written in terms of machine instructions is executed as follows.

- For each machine instruction, when it is in the decoding cycle, based on its OP-Code the corresponding micro-program is loaded into the control memory.
- The micro-program counter fetches each micro-instruction in sequence (or jumps to the required location) and they generate the required control signals.
- Once the present micro-program is complete the next micro-program is loaded based on the new instruction.

Handwritten notes: FROM, ADDR, M, SUB, IN

The Op-Code actually corresponds to the micro program. The Op-Code corresponding to the micro program is loaded into the control memory. Basically that the idea is that in this case you can just think in this manner that whenever a macro instruction is there has come to this for as for come for execution, immediately you execute the micro instruction for fetch.

After that basically there is a decoding. So, once the decoding is done basically that is a decoding cycle. So, when the decoding is done the corresponding MPC points to the memory of the macro program, which corresponds to that particular instruction. So, if there are as I told if there are multiple instruction like add, sub, multiply, which will be similarly we have made a single port to optimize space, then for any of these instruction you start pointing to the start point of that corresponding instruction.

Now whether it is a add or subtraction or multiply at 1 point of time it will diverge out and again come back. For example, as I told if we have add R 1 M and say add subtract R 1 M. So, only one point that is the control of the ALU will be different. So, only that particular it will diverge way again come back because after doing the computation, again you have to write back the value of because say many part should be common like loading the value of M from the memory then again after doing the computation the values has to be written to R and those parts of the microinstructions will be common. So, that part will be common and only the jump will be based on the signals made to the ALU.

We will take an example which will make the things clear, but just I we are giving the brief theory here and then the micro program counter fetches each micro instruction and they executing sequence or jumps to the required location. So, when there are jumps there will be 2 jumps one is an inherent jump because of the macro program and when because of this optimization of the memory space, the micro program is common to many instructions taken together. So, you have to jump accordingly then the IR actually in instruction register tells where to jump. So, how it exactly implements we will see.

. So, one of the micro program is control the next micro program is loaded based on the new instruction that is very important. So, when add has been done. So, now, add is complete, then the MPC will start pointing 2 again the fetch part of it that is it corresponds through the new instruction or the new macro program macro instruction ok.

(Refer Slide Time: 15:54)

Encoding of the control signals

- It is known that we need to store the information of each control signal in the control memory (in control function field). The status of a particular control signal is either high or low at a particular cycle.
- It is possible to reserve one bit position for each control signal. If there are n control signals in a CPU, then the length of each control word is n .
- Since we have one bit for each control signal, so a large number of resources can be controlled with a single microinstruction. This organization of microinstruction is known as horizontal organization.
- To reduce the size of control word, we can use compact code to specify only a small number of control functions in each microinstruction; this is known as vertical organization of microinstruction.

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 discussed and encoding is very important. So, if you do an 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. These actually a horizontal micro program and it is going to be it is 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 call the vertical micro instruction. So, if you just go for a very flat compression that is from 2 to the power n , we just compress it to n or from n to $\log n$ and using n^2 to the power n decoder, then actually full compression is there and these actually call a micro verti full vertical micro program. But in this case only one bit signal can be made a 1. So, if simultaneously 3 or 4.3 to be a 1 you require 4 cycles from that.

But you have seen that 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)

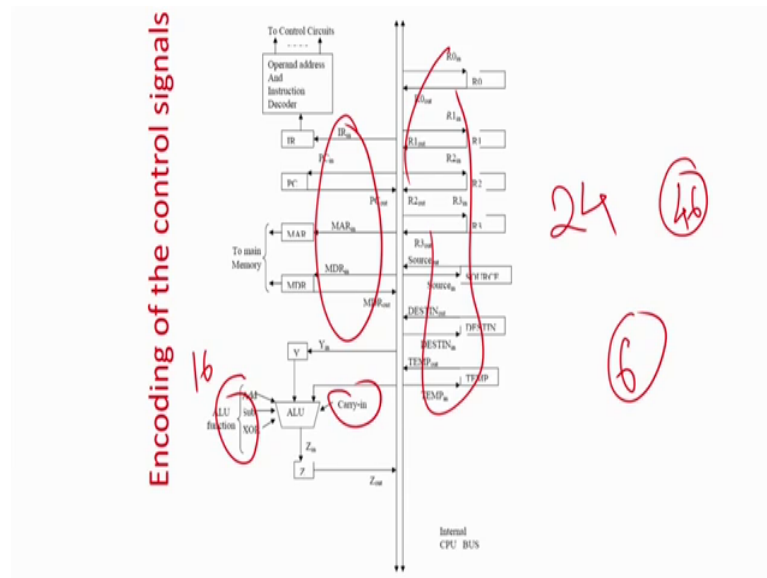
Encoding of the control signals

- In case of horizontal organization, the size of control word is longer, which is in one extreme point and in case of vertical organization, the size of control word is smaller, which is in other extreme.
- In case of horizontal organization, the implementation is simple, but in case of vertical organization, implementation complexity increases due to the required decoder circuits.
- Also the complexity of decoder depends on the level of grouping and encoding of the control signals.
- Horizontal and Vertical organization represent the two organizational extremes in micro-programmed control. Many intermediate schemes are also possible, where the degree of encoding is a design parameter.

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 call 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 discuss the basics. So, this 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 may 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 the 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 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 R 0, R 1, R 2, R 3. So, the number of signals are 0 1 in out, in out, in out. So, 4 registers are there and in an outs are there. So, just remember how many signals are there 8 signals are there then there are 3 system registers, which are user cannot use and they are use for some system storage; so 1 2 3. So, that is temp destination and source some registers are there.

So, how many registers are total? 1, 2, 3, 4, 5, 6, 7 and for each registers you have a input and output port. So, it is 7 into 2 14 signal 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 1 15 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 plus 2 9 18 plus to 20, then this is your CPU. So, you can see why 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 several 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 miss 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 to this input output port if you look at you will find out that the 24 input output ports the signals I will show you. Along with that; that means, there are 24 ports to be controlled which correspond to input or output or any register like MBR MAR R 0 to R 4 source destination Y, Z, Z in Z out specific out 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 are also be there.

(Refer Slide Time: 21:07)

Encoding of the control signals

- The example CPU contains four general purpose registers *R0*, *R1*, *R2* and *R3*.
- In addition there are three other registers called *SOURCES*, *DESTIN* and *TEMP*. These are used for temporary storage within the CPU and completely transparent to the programmer. A computer programmer cannot use these three registers.
- For the proper functioning of this CPU, we need all together 24 gating signals for the transfer of information between internal CPU bus and other resources like registers.
- In addition to these register gating signals, we need some other control signals which include the Read, Write, Clear, set carry in, WMFC, and End signal.
- It is also necessary to specify the function to be performed by ALU. Assume that the ALU that is used in the design can perform 16 different operation such as ADD, SUBSTRACT, AND, OR, etc. So we need 16 different control lines.

So, now, we are we will try to see based on these architecture how we can optimize based on cluster. So, assigning individual bits to each signals these to are long micro instruction here we have counted that the 24 input output ports then for some for some ports signals will be there to control the ALU, that also you have to add along with that

you have to have some signals like we select add select w for read, that is memory read or right there should be one signal then the wait for WMFC will be one signal. So, there are 7 other signals and the count will be quite large.

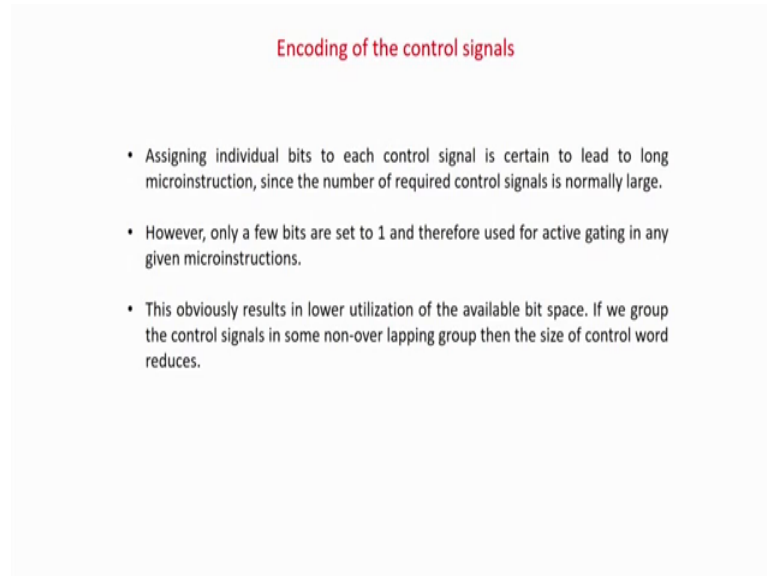
So, let us see what are the counts. So, we will see the count in the next slide, but if you take all the counts and put in the flat architecture the size will be very high already we are discussed so many times, which will be horizontal micro program. So, this will; obviously, lead to lower bit space 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 sources is there 4 user purpose registers there are 3 sip central registers and basically as you have counted corresponding to the like a memory address register, memory reader register, Z Y. So, there are 24 getting signals as we have counted in the figure along with that basically read right clear set carry in some signals is WMFC end.

. So, the signals can be more, but in this example we are assuming that theses many other extra signals are they are 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 weight for the memory and end. So, these also actually count from to 1 2; 1 2 3 4 5 6 there are another 6 signal are there. 24 signals where 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 with ALU can do like add subtract end or compare. So, many other signals are there is assumed that 16 different functionalities are possible. So, there are the 16 different lines.

(Refer Slide Time: 23:05)



Encoding of the control signals

- Assigning individual bits to each control signal is certain to lead to long microinstruction, since the number of required control signals is normally large.
- However, only a few bits are set to 1 and therefore used for active gating in any given microinstructions.
- This obviously results in lower utilization of the available bit space. If we group the control signals in some non-overlapping group then the size of control word reduces.

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 they are as you 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 equate 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)

Encoding of the control signals

- The above discussion indicates that 46(24+6+16) distinct signals are required.
- This indicates that we need 46 bits in each micro instruction control word; therefore the size of control word is 46.
- Broadly speaking, on an average 10-15% of the bits are set to 1 in each micro instruction and rest of the bits are 0. Therefore, the bit utilization is poor, and there is a scope to improve the utilization of the bits.
- It is observed that most signals are not needed simultaneously and many signals are mutually exclusive.
- As for example, only one function of the ALU can be activated at a time.
- From digital logic circuit, it is obvious that instead of 16 different signals, we can use only 4 control signals for ALU operation and then use a 4 to 16 decoder to generate 16 different ALU signals. Due to the use of a decoder, there is a reduction in the size of control word.

Handwritten notes:
46
16

So, therefore, it says that there is 46 different signals are there and therefore, basically 46 will be the length of the micro program end and it has been found theoretically that only 10 to 15 percent of the memory positions are 1. So, if we have such a big array which size is 50 46 and it will be long it will be long depending on how many micro instructions are there, and only 15 percent is actually filled with 1 is a huge waste of memory that is you will 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 2^6 is 32. So, 2^6 is 64. So, 6 bits will actually solve the problem for you.

. So, if I grow for a full encoding architecture. So, it will be 6 bits which will encode and then (Refer Time: 24:26) 2^6 is to the 64 bit architecture using a decoder. And the job will be done, but in that case will be very very slow you have simultaneously only 1 bit can be made 1 so. In fact, is not a very good idea to solve the problem. So, we can may go for a clustering approach which is call the hybrid micro program. So, that is 1 actually we are going to study.

(Refer Slide Time: 24:45)

Encoding of the control signals

F1 (4 bits)	F2 (3 bits)	F3 (2 bits)	F4 (2 bits)	F5 (4 bits)
0000: No Transfer	000: No Transfer	00: No Transfer	00: No Transfer	0000: Add
0001: PCout	001: PCin	01: MARin	01: Yin	0001: Sub
0010: MDRout	001: IRin	10: MDRin	10: SOURCEin	0010: MULT
0011: Zout	011: Zin	11: TEMPin	11: DESTInin	0011: Div
0100: R0out	100: R0in			
0101: R1out	101: R1in			
0110: R2out	110: R2in			
0111: R3out	111: R3in			
1000: SOURCEout				
1001: DESTInout				
1010: TEMPout				
1011: ADDRESSout				1111: XOR

Before that how to we make a cluster and what will be the idea for that that, we have to fairly think and we have to make the cluster. So, if we go for a flat architecture that if each bit can be is in cluster the horizontal micro program one extreme and if I go for full encoding that is 6 bits will be required encode all the 46 bits, and it will be compress 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)

Encoding of the control signals

F6 (2 bits)	F7 (1 bit)	F8 (1 bit)	F9 (1 bit)	F10 (1 bit)
00: no action	0: no action	0: carry-in=0	0: no action	0: continue
01: read	1: clear Y	1: carry-in=1	1: WMFC	1: end
10: write				

Say for the timing this assume that there is F 1 F 2 F 3 F 4 F 5 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 the single bus architecture. So, simultaneously R 1 out, R 2 out, R 3 outsource out cannot be done it will leads to a conflict let is if you look at this if you look at this figure.

So, on if you single bus architectures. So, all these registers all these 24 story I was talking about only 1 bit out at I this single bus there is a 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 100 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 require because simultaneously R 2 out R 3 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 required a 4 bits to solve problem. So, you require 4 is to 16 decoder for this and in this case I mean because the full decoder we will not be used because you not have 16 such signals.

. So, you can just see 0 0 0 for no transfer; that means, nothing will happen and all the other bits are actually do not cares in all the cases there will be no output, but if it is 0 0 1 PC out will be 1 if it is 0 0 0 1 then mdrout 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 cluster because simultaneously I cannot have add and sub simultaneously only one can be high. So, actually cluster everything make a 1 cluster your job is done. So, this one 2 very key rule that all the output basically you put in 1 cluster all the CPU function will it is you put in 1 cluster because simultaneously 2 cannot be 1 and simultaneously ALU cannot do operation.

But this logic only holds if the 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 you 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, so this Y source in 1 cluster. So, this clustering they have may made on some philosophy that is they have thought that like they have say thought that PC in I IR in they are moving they are keeping it in basically 1 cluster so; that means, they are found out that both reading the value of PC that is loading the value of the some loading the value updated value 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 R 0, R 1, R 2 and R 3 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 would not get updated. Because first we 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 Rins 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 do not 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 either thought that that sometimes you may read the value of program counter in terms of important 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 Rin and also it may go to basically memory address register in it is a indirect addressing kind of mode.

So, it may happen that the output are 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

register over here and the memory address register or the memory reader register or TEMP in another cluster. That means, basically simultaneously 1 register can be made 1 and 1 register can be made 1 here and even made 1 register can be made 1 here the input

So, simultaneously some output can feed fed to 1 register over here 1 register over here and 1 register over here that is where you require simultaneous input is possible more likely you that why divide the cluster. If you would have thought 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 put a saved a space, but here they are assume 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 as we can put it 1 and then end and continue are the 2 signal.

. So, these are some of the signal 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 simultaneous also it 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 y clustering how it can be made that cluster, 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 a real example with each bits signals and registers and ALU.

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

(Refer Slide Time: 32:38)

Micro Program Execution

Another possibility of grouping the control signals is:

- A source for data transfer must be unique, which means that it is not possible to gate the contents of two different registers onto the bus at the same time.
- Similarly Read Write signals to the memory cannot be activated simultaneously.
- This observation suggests the possibilities of grouping the signals so that all signals that are mutually exclusive are placed in the same group.
- Thus a group can specify one micro-operation at a time. We can use a binary coding scheme to represent a given signal within a group.
- As for example for 16 ALU function, four bits are enough to encode the appropriate function.
- A possible grouping of the 46 control signals that are required for the above mentioned CPU is given below.

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

(Refer Slide Time: 32:55)

Encoding of the control signals

- Here all out-gating of registers are grouped into one group, because the contents of only one bus are allowed to go the internal bus.
- But the in-gating of registers are grouped into three different groups. It implies that the contents of the bus may be stored into three different registers simultaneously.
- Due to this grouping, we are using 7 bits (3+2+2) for the in-gating signal. If we would have grouped then in one group, then only 4 bits would have been enough, but it will take more time during execution. In this situation, two clock cycles would have been required to transfer the contents of PC to MAR and SOURCE.

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 why we saw we the 46 bits required in the extreme flat keys and if you look at it how the

way we have clustered. In very very few cases there will be contention at very very few cases you can have both R 2 and R 2 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 save a lot in the space of the memory. So, basically if you also I mean that is that is what I have tell here all philosophy is extend. And LP is go through this 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 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 of line and get the values.

So in fact, they have find out basically.

(Refer Slide Time: 34:04)

Encoding of the control signals

- In this grouping, 46 control signals are grouped into 10 different groups (F1, F2,....., F10) and the size of control word is 21. So, the size of control word is reduced from 46 to 21, which is more than 50%.
- For proper decoding, we need the following decoders:
 - group F1 and F5: 4 16 decoder,
 - group F2: 3 8 decoder
 - group F3, F4 and F6: 2 4 decoder
- Therefore, grouping of signals is a critical design parameter. If speed of operation is also a design parameter, then compression of control word will be less.

So, I mean instead of 40 signals we are basically now F 10 value number of word is 21. So, huge saving space, but of course, some additional things are required we acquired some decoders over here also some decoders are also require like for some groups basically, we require 4 is to 16 decoder for some group 3 is to 8 decoder and for some case 2 2 is to 4 decoders.

So, not only have to find out the cost of the basically your memory is save that is true, but also you have to account for the decoders are also taken into picture and also they are

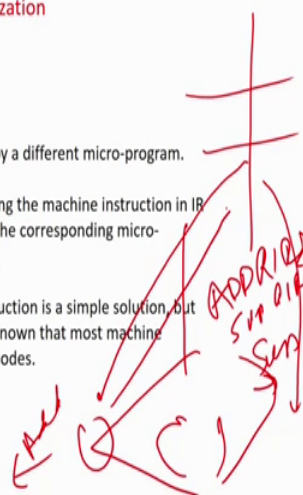
maybe slightly slower in speed corresponding to a horizontal micro program, but that will not be very very high, because the way intelligently we have actually place the signal in different clusters.

So, till now if we are looked actually we will 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)

Micro-Program Optimization

- In micro-programmed controlled unit,
- Each machine instruction can be implemented by a different micro-program.
- Each micro-program can be accessed by decoding the machine instruction in IR and accordingly loading the starting address of the corresponding micro-program into the Micro-program Counter (MPC)
- Writing a micro-program for each machine instruction is a simple solution, but it will increase the size of control memory. It is known that most machine instructions can operate in several addressing modes.



Now, we are going to see another interesting way of optimizing the micro program how? Basically the 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 c.

So, and fore example if we have a instruction call sub and you have a instruction call add maybe add R 1 or R 2 and subtract R 1 and R 2. So, you can feel that most of the micro instructions and the signals to be generated will be same accept in only 1 case, then in one in case of add the arithmetic logic unit has to be configure to add and in case of subtraction is just to be configured into svb mode. So, only one bit will be changed or one controls we never will be change all others remain will similar.

So, if you have 2 different micro programs for this 2 micro instructions. In fact, that will be wastage in the now 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 you can start.

For example if you had a single program call add R 1, R 2 and maybe subtract R 1, R 2. So, most of the phage etcetera loading from R 1 to R 2 saving the value of R 1 plus R 2 or R 1 minus R 2 to R 1 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 re or 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 phage 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 equal to 1 another branch will be for subtract equal to 1 and again after executing then you can again come back here, which will lead to storing the value of the ALU to register R 1 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 is 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 programing executer, 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 micro 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 very un optimized.

So, basically this is the simple solution, but we will increase the number of memory. Because we know that there are so many different addressing modes, so, many different type of basically in macro instruction and so forth.

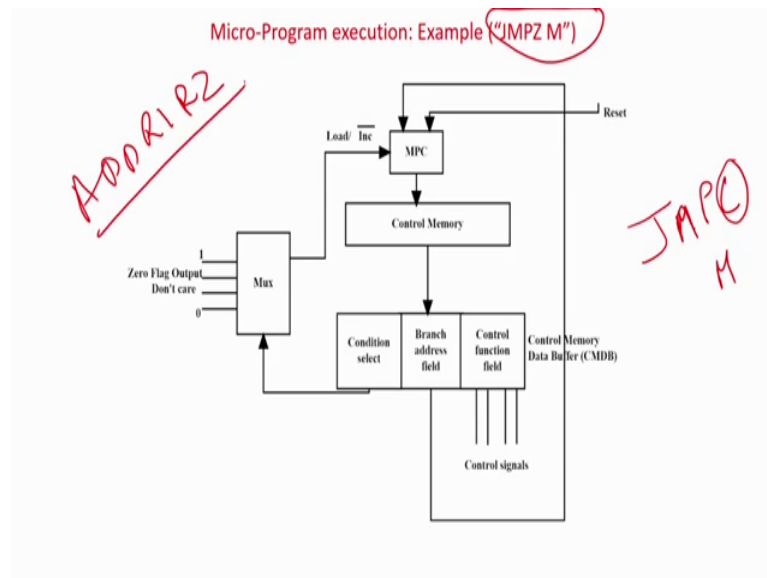
(Refer Slide Time: 38:35)

Micro-Program Optimization

- If we write different micro-programs for each addressing modes, then in most of the cases, we are repeating some part of the micro-routines.
- The common part of the micro-routines can be shared, which will reduce the size of control memory.
- However, this results in a considerable number of branch microinstructions being needed to transfer control among various parts. So, it introduces branching requirements within the micro-programs.
- For example, the fetch micro-routines of all instructions are same.
- Also, for instructions like "ADD R1, R2", "SUB R1, R2" most part of the micro-routines would be the same except the control signal for ALU configuration; in the first case the control signal to the ALU would be Add while in the other case it is Subtract.

So, basically what we can do is that, we can actually write micro routines which can be shared for example, as I told you like add and sub. So, most of the case will be similar excepting the instra excepting 1 bit position or the 1 control signal for corresponding to add or subtract of the ALU. So, you can try to do basically write basically a single micro program for similar type of basically macro micro instruction macro instructions, which are commonly in type this slide actually these 2 slides basically these 2 slides basically tells you what I have discussed. So, you can read in offline.

(Refer Slide Time: 39:10)



Now, let us again now actual I will tell you two things, two things are very important over here. So, as I told you in the last class that in this unit we will also see how a complete macro instruction is executed in terms of micro instructions. Because in the last unit we just saw that how to do a fetch here we will see how a total instruction is executed. Secondly, as I told you we also give some idea and how micro instructions can be macro instruct micro instructions corresponding to different type of similar macro instructions can be club.

Right for example, we are going to show you how a complete macro instruction is executed in terms of micro instruction, and we were going to take the help of jump on 0 M, that is the that is the example of the macro micro instruction we are going to take, and we will try to see how it is implemented in terms of micro instructions. But again as I told you we can optimize based on a single micro routine for difference similar type of macro routines, macro instructions basically.

So, let us take another may be jump on carry and may be you and we say that to jump on M, jump on M another instruction which is very macro instruction which is very similar. So, this is jump on 0 and basically this is on jump on carry to memory location M. So, basically there built to similar macro instructions, they will have almost similar format and similar type of control signals will be generated, accepting the case we will one

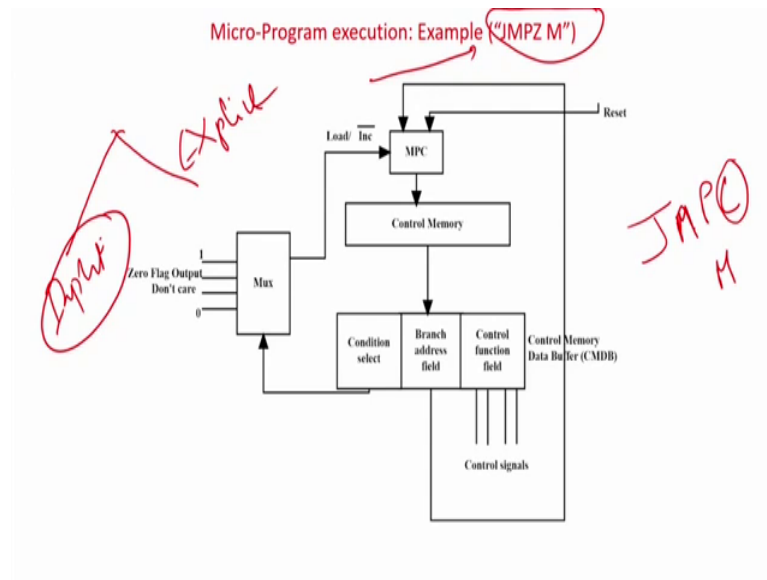
place you are going to check the 0 flag and the other case you are going to check the carry flag that will be the only difference here.

So, we will say how to try to optimize and do this. Again there is very another very important thing you have to note over here. If I say that I will have a similar macro program micro program for different type of macro instructions which are common in type and then you will based on the different part we have to implement, you will jump from the main micro program and again come an execute the different part and then again come back and execute the common routine so; that means, they will be so many branching 2 and fourth in the micro program.

So, this type of branching will be actually called the in implicit jump we have to do, because you are writing common micro program for different macro program. So, this is not a requirement of the macro program. So, for example, if I have add R 1 and R 2. So, if this instruction is there you do not require to explicitly write jump instructions in the micro routine because this is this is not a jump instruction, but as I told you we will not have a single macro micro program for add and sub and may be multiplied, we will have a single routine and based on whether it is add sub or multiply, you will jump.

So, these actually the implicit jump routine, which will be there in the micro program, and then another type of jumps which will be there which are actually explicitly mention in the macro; macro instruction corresponding to that right for example, jmpz. So, it says that if the 0 flag is set you jump to memory location M, that is an explicit jump instruction which has to be in build in the micro routine. So, there are basically 2 parts.

(Refer Slide Time: 42:00)



One is called the explicit that will because of the instruction type if it is then add sub this will not be there, and there will be sub implicit type which will be coming in to the micro routine because you are doing an optimization. So, there are 2 types of things will be required. So, there for I have explicitly chosen these instruction which is called the jump in. So, that I can illustrate both type of jumps which has to be there in the micro routine.

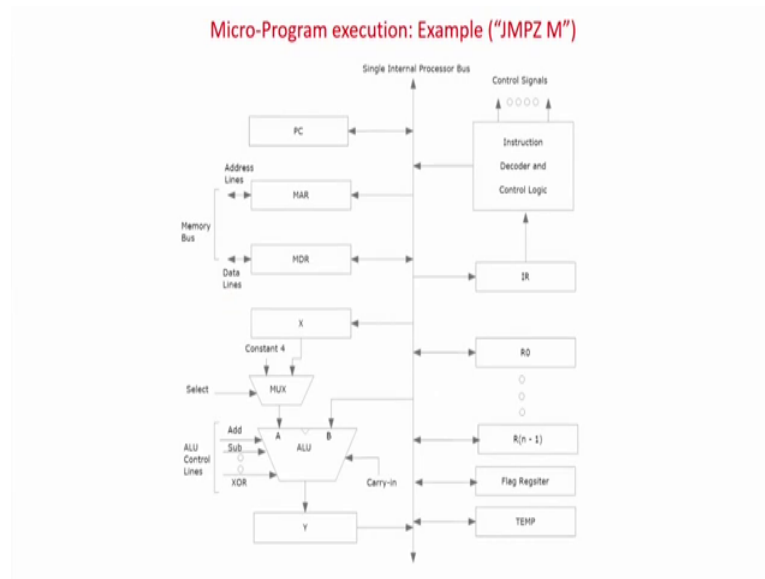
And let us assume that we have we are going to write common routine corresponding to jump on Z to M and jump on carry to M. So, this figure if you look at. So, this is already we have discussing in the last unit just a refresh. So, these are the control signals to be generated, this is the branch field address; that means, if you have to jump to some location that value will be given over here that will be loaded to the micro program control unit if and only if the load value is 1. If the load value is 1 it will take the value of the branch controller address, and if it is 0 then basically it is going to just increment the value already we have seen.

And therefore, who selects whether the you have to jump or whether you have to load the branch address field or just increment depending on the control select. For example, if you are contract select contain condition select is 1 one the last bit of 0 will go and you will always be incrementing mode. If you give 0 0 that is again unconditional jump, if the condition select is 0 0 then basically what happens? The 1 is going the first line of the

marks will be connected which is one. So, this will be a 1 and always it will be load the value from the branch address and you will make unconditional jump.

The other two things basically connected to different conditions like 0 flag instruction register output and so forth. So, accordingly we will we have already seen this thing and accordingly we will fill up the values means these 2 basically things in this case can be reprogrammed based on connectivity and this is the example.

(Refer Slide Time: 43:48).



Of a single bus architecture we have discussed in so many days just have a look at it, because you will be discussing our example for jump on Z on M on this bus architecture.

(Refer Slide Time: 43:58)

Micro-Program execution: Example ("JMPZ M")

The control steps and control signals to fetch and execute the "JMPZ M" instruction are as follows:

1. PC_{out} , MARin, Read, Select=0, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDR_{out}, IR_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in} [If Zero Flag is NOT SET then END]
5. Z_{out}, PCin
6. END

Control Function field of the Control memory for Instruction JMPZ M

Me mLo c	PCou t	MARin	Read	Selec t	Add	Zin	Yin	Zout	PCin	WMFC	MDRou t	IRin	IR- off _{out}	END
1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	1	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	1	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1

I have again come to that.

So, again recollecting some 2 3 lectures back. So, what is jump on 0 does. So, basically if you look at it first will be program counter will be output that will be going to the memory address. That means, whichever instruction you have to read that addresses in mar in memories in read mode select is 0; that means, you are going to take default value of incrementing the PC. So, select equal to 0, ALU is in the add mode which will do PC is equal to PC plus constant, which will be going to the temporary register Z in.

Next is you have to Z out to PC in; that means, PC equal to equal to PC plus constant and basically and it will go to PC and also your waiting basically it will still for the memory to give the read signal. So, these things are very common just already we have discussed so many time, just I am giving out the values and just I am repeating it for ease of readability.

Then you have saying that the memory data register after it has read will go to instruction register now the instruction register has the value of jump on Z to M. Now look at this, this instruction is very important here it is actually the explicit jump command in this program in this macro program. So, this is the micro instructions or micro control signals for these macro instructions. You say that offset field of IR out already we have told you that the already we have discuss some lectures back that offsets field of IR you added with the that value of program counter and there for what you get

basically that we have the PC is also stored in Y in for that reason. So, Y in that is the present value of PC plus the offset if you added, you are going to get the value of the jump which will be actually stored in Z.

So, now by this instruction offset value of IR select and. So, 1 input of the basically of the ALU is basically going to be Y in and the second value is going to be the offset value you add it, and Z actually Z which is connected to the output of the ALU you have to just recollect some 3 4 lectures back. So, Z in the actually is going to have the value of memory location M. So, that is a repetition. So, till now what happened if you exit execute up to 4 Z is going to have the value of memory location M.

Now, it says that if it is jump on 0, if it is 0 then basically you have to load the value of PC equal to M otherwise you do not have to do that. So, what you have to execute the next macro instruction ok. So, what we will do? Basically if 0 flag is set, if the 0 flag is set then you are going to execute this one Z out equal to PC in otherwise basically you have to go to number 6 micro instruction which corresponds to end now gain what is happening maybe after jump on Z M because the 0 flag is not set; that means, 0 0 flag is not set in that case you have to jump to end, after end what happens this micro routine ends and may be after this macro-macro program of this macro instruction after jump on Z maybe say add R 1 M is there in your macro routine.

So, therefore, again the micro program corresponding to this will load or basically what is going to happen is the micro program counter micro program a program counter will start pointing to the micro instructions corresponding to this; that means, the next macro instruction will be executed if the 0 flag is not set. So, if the 0 flag is not set, you are going to jump over here. So, it says that if the 0 flag is not set then go to end basically the; that means, basically you have to load the value of PC with MPC with the value of 6.

Else you do not have to do anything, if the 0 flag is set if the 0 flag is set; that means, it is jump on 0 you do not have to do anything you have to go to memory instruction number 5 micro instruction number 5 Z out will be loaded to PC in, and basically you have an come after that you will go to 6; that means, after at present the value of PC equal to Z out that is equal to M memory location M. So, the in the macro program basically we will jump from this to memory location M which is may be the instruction which one to

execute which may be same MUL R 1 and R 2 and may be just after this maybe we have some call add R 1 and R 2.

(Refer Slide Time: 47:53)

Micro-Program execution: Example ("JMPZ M")

The control steps and control signals to fetch and execute the "JMPZ M" instruction are as follows:

1. PC_{out}, MAR_{in}, Read, Select=0, Add, Z_{in}
2. Z_{out}, PC_{in}, Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}, [If Zero Flag is NOT SET then END]
5. Z_{out}, PC_{in}
6. END

Control Function field of the Control memory for Instruction JMPZ M

Me mLo c	PCou t	MAR _{in}	Read	Sele ct	Add	Z _{in}	Y _{in}	Z _{out}	PC _{in}	WMFC	MDRou t	IR _{in}	IR- off- out	end
1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	1	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	1	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Handwritten notes: MPC → ADD R1, R2; MUL R1, R2; M; ↑

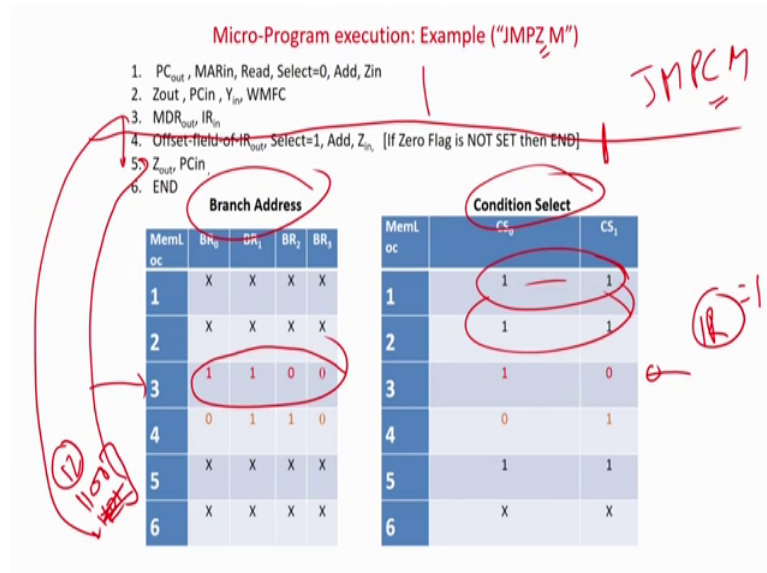
So, what is going to happen? So, if the 0 flag is not set then you go to end; that means, I will jump for directly 6 directly to 6 basically and in the this is the micro micro program and the corresponding macro program what is going to going to happen is that, you will end the micro program for jump on 0 and before the MPC will start pointing to the next instruction basically which is nothing, but R 1 and R 2 that corresponding micro program will start executing.

And if the 0 flag is set and if the 0 flag is set basically if you see then the micro program we will not jump it will go to 5 and in that case what is going to happened? Z out equal to PC in the macro level multiple R 1 and R 2 which is in the memory location M is going to be loaded and after this these a this micro program will end, and the next micro program PC will start pointing to this instruction micro program and not to add R 1 R 2. So, these what is the basic flow.

So, now, let us see how actually it happens in terms of micro program. So, if you look at the first micro instruction PC out mar in read and all these things. So, PC out is 1 mar 1 read 1 and basically add an R 1 is equal to 1 corresponding signals are all 1 2 also if you look at it Z out Y in Z out PC in and so, forth 3 is basically if you see MDR out and this one. So, it is very simple you can just trace out the field and you can find out the

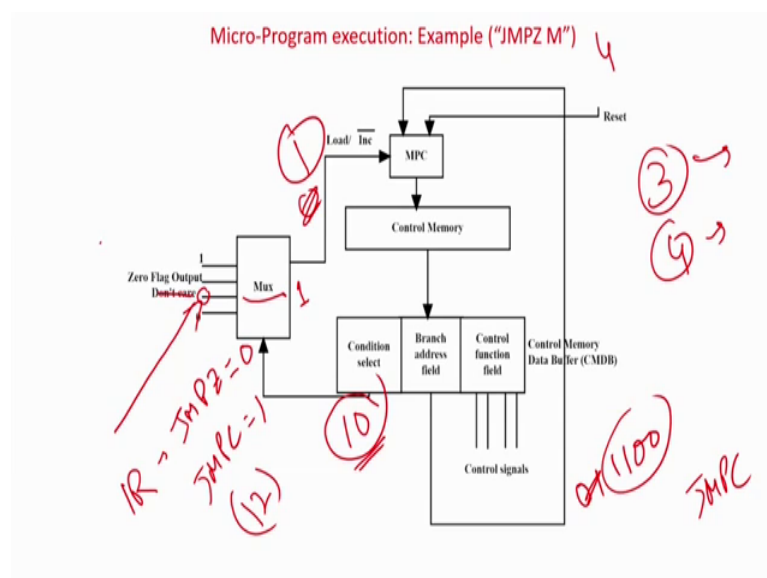
corresponding signals are one. So, this is very simple nothing must true look at the control memory here.

(Refer Slide Time: 49:34)



What is very important is, what we have to see in this slide. So, basically if you see this is your condition select and this is your branch address select now I will have to do some connection in the figure, which I am going to do. So, in this case I am is do not care I remove over here and here I have to put instruction register output.

(Refer Slide Time: 49:50)



Because the instruction register is now going to tell you in this case whether it is jump on zero instruction or maybe it is telling jump on carry because I am just assuming that, I have (Refer Time: 50:07) single macro put sorry micro routine for jump on Z and jump on carry.

So, basically in this case I assume that because if it is the jump on 0. So, basically I assume that the same micro routine has to be executed, and if the instruction register says that it is jump on carry, then I may have to jump to some other memory location micro memory location micro program memory location may be 12, which is actually exactly contrast the value. For example, as I tell you up to 3 is it nothing but it is your basically fetching the instruction, and this is corresponding to the execution of jump on set.

(Refer Slide Time: 50:38)

Micro-Program execution: Example ("JMPZ M")

The control steps and control signals to fetch and execute the "JMPZ M" instruction are as follows:

1. $PC_{out} = MARin$, Read, Select=0, Add, Zin
2. Zout, PCin, Y_{sp}, WMFC
3. MDR_{out}, IR_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}, [If Zero Flag is NOT SET then END]
5. Z_{out}, PCin
6. END

Control Function field of the Control memory for instruction JMPZ M

Me mLo c	PCou t	MARin	Read	Selec t	Add	Zin	Yin	Zout	PCin	WMFC	MDRou t	IRin	IR- off _{out}	END
1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	1	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	1	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Handwritten notes: A red arrow labeled 'JMPZ M' points to step 4. A red arrow labeled '12' points to row 3 of the table. A red circle around the '1' in row 3, column 12. A red circle around the '1' in row 3, column 13. A red circle around the '1' in row 4, column 14.

Maybe I here from here actually after if I am if the instruction is jump on 0, from 3 I have to go to 4 5 and 6.

But assume that if this instruction or the macro instruction is not jump on Z maybe it is on jump on carry to M maybe it start from memory location number 12 micro memory micro instruction memory number 12, then it has to jump and executing corresponding instruction from 12. So, this is actually the implicit jump. So, the micro routine will come up to here, after that it will look at the instruction register decoder output. So, if it says that basically it correspond to jump on Z there will be no jump, but if it correspond to jump on carry it will jump to 12.

So, basically what I will do, I will actually connect instruction register to this the third line of the match and I will say that if jump on 0 is there basically it is going to give you 0 instruction decoder and if it is the jump on carry micro instruction it will give us 1. So, an the condition select at the time is obviously equal to 1 0. So, at this point after a third third micro instruction when the fetch is over, I will give condition select equal to 1 0 and of course, branch address select will be 12. So, it will be 1 sorry 1 1 0 0 I assume that from 12, the jump PC jump on carry micro instruction micro program start.

So, at third program and third microinstruction of this common routine which corresponds to jump on carry and jump on Z, the fetch is over which is common for everything add third instruction it will check the condition select 1 0; that means, is going to check the IR IR line which is the third line and branch address I have kept it has 1 1 0. So, it is checking that if it is jump on 0. So, it will just increment. So, from third it will go to fourth memory location of the micro program, which will be corresponding to jump on Z.

But if the macro program that is the instruction decoder tells that the jump on carry is a jump on carry it will make this line has 1. So, if it is making this line has 1 it is going to load and it is going to load the address of 1 1 0 0; that means, it wills micro program corresponding to jump on carry is going to execute. So, like that in a simple thing that is going to be going to happen over here.

And here I am connecting 0 flag output that is going to check. So, this I can I am repeating that is the implicit jump instruction that has to give in because of 2 micro macro programs 2 macro instructions are club together, that is jump on 0 with basically start from 4 and assuming that jump on carry start from 12. So, that is how it has settled. So, I am just going to look at again.

So, up to 3 has I told you is fetch which correspond to which is very similar to even jump on carry. So, that will be similar. So, there for I do not require to check anything and the condition select. So, I have made both of them 1. So, 1 one means has you can see. So, if I make 1 one means the last bit selected which is 0 this is the condition select is 0 0 then max output is 0. So, it is always basically increment mode. So, it will keep on incrementing. So, again and it will start coming at here. So, here it is very very important.

So, here the condition select as I told you is 1 0; so 1 0 means as I has already shown you. So, 1 0 means your connecting to the third line here I have connected the IR output the instruction decoder output, which actually happens in this way that if it 0 and 1 because here the your making the simple assumption by because I am 2 micro programs common, but of course, if you have large number of micro programs club together, then the marks will also the not be 2 is to 4 d multiplexer, it will be larger size of marks because corresponding to the instruction decoder output the number of input for the marks will rise, because in fact, there are 2 constants 1 is 1 and 1 is 0 1 is for default load and 1 is for default increment, but the other inputs will be corresponding different condition basically.

So, here we have just assume that the 2 instructions of there. So, we have a taken a 4 is 2 two multiplexer, but. In fact, if the number of instructions which are combined and number of IO signals are large. So, generally the practical cells these marks is quiet large in size and the condition select also is larger in width basically. So, for the time being I am just as making simple assumption for you. So, if with the jump on 0 the instruction register will 0 to this and other wise instruction register will give 1 for this.

So, if it is a 1 basically if this is a 1 if this is a 1 as you can see. So, if this is a 1 it will corresponding to basically load which is nothing, but memory location number 12, micro program memory location number 12 which correspond to jump on carry. And if it is jump on Z you are going to get a 0 over here which will load the next value. So, if you look at it. So, it is saying that the condition select is 1 0; that means, you are going to check IR output IR decoder output. And if it is a 0; that means, you are actually talking of this jump on 0. So, you have to just increment from 3 to 4, but it is a jump on carry from 3 you will jump to another location which is memory location number 12.

And giving assume assuming that basically 1 2 3 4 5 6 is common sorry 1 2 3 is common for both because 1 2 3 actually corresponds to jump on Z and jump on carry, but 4 5 6 is very much explicit or in fact, you can say 4 and 5 is very much explicit basically for jump on Z and maybe 11 12 and 13 memory location will be fixed for jump on. In fact, I should say that not even that only this one only this instruction that is offset IR select 1 if 0 flag is not set only that part actually is different from from for jump on carry and jump on Z.

In fact, here we are going to check the 0 flag and the other case maybe you have to just check the carry flag. So, that is the only different part. So, maybe memory location number 12 is the only part which is going to be different. So, if the deco instruction register is actually giving the value of 1 which corresponds with jump on carry. So, in this case it will jump to this memory location because the condition is 1 0. So, it is going to check if it is 1 0; that means, the third row of the multiplexer is selected and if it is a jump on carry, instruction register value will be 1 and it will jump from this 2 memory location number 12.

So, in the 12 we will have just this instruction which is number 4 over here, which is only difference only difference between jump on carry and jump on 0. So, that will be executed and after that you will again jump back and come over here because after that it is very similar again you have to do Z out PC in and M. So, only one bit will be different in this case or only 1 memory what will be different which is the only difference between these 2 micro program that is jump on Z and jump on carry that different will be executed at 12 and again you have to come back.

So, they are be 2 implicit jumps. So, one jump will be jumped from 3 to 12 depending on the type of micro instruction you are considering at the time and after executing 12, you are going to have a basically unconditional jump which will bring you back to 50.

So, basically this is the implicit jump instruction, which you have to put in because you are adding 2 macro micro programs corresponding to 2 macro program macro instructions. So, I have shown you up to 6 here maybe you can put dot dot memory location 12 will also have similar thing which will again correspond to basically the fourth instruction which is the difference. Now interesting now basically till now we have discussed basically the explicit implicit part of it, now this macro routine of the micro routine for this micro instruction is also jump instruction.

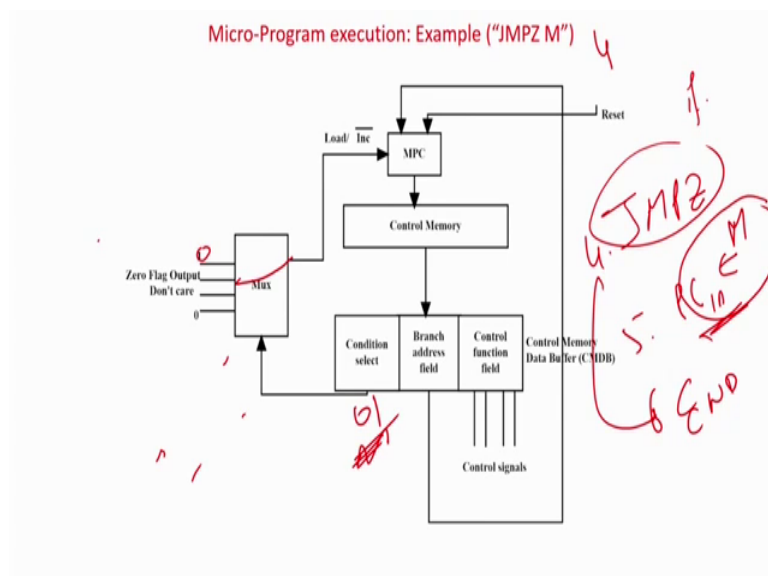
So, what is says that, if the 0 flag is not set then actually you have to go to end and if the 0 flag is set you have to just go an incremented because Z out PC in basically is going to load the program counter if the macro instruction which is located in memory location end. So, this is actually nothing, but basically the jump which has to be explicitly taken in this macro instruction micro instruction. So, this is the explicit jump the micro routine should have corresponding to this macro instruction.

So, of course, again at 4 also there will be condition check, but other than you can see I have put it one. So, they do not correspond of any kind of jump they just correspond to increment, but here again 0 1. So, you should remember that 0 0 and 1 1 they correspond to default load and then they corresponding to they correspond 2 jump or not jump basically, but all other combinations basically like 0 1 and 1 0 in this case, they correspond to shake of the basically condition and they will depend and they will need to jump of some address which is present over in the branch address. So, these 2 are very very important over here.

Because 1 1 corresponds to basically a 0 in the marks, which correspond to just a default increment and basically 0 0 corresponds to default unconditional jump we have already seen in the figure basically again just revisiting. So, 0 0 means this one which actually corresponds to load if there is default jump and 1 1 in the condition actually which corresponds to the fourth line of the marks which is 0, correspond to simple increment, but all other combination in this case 0 1 and 1 0 they correspond to branch.

Now, this is your implicit branch and the last one the third fourth instruction that is 0 1 which corresponds to the second line of the marks basically let me erase it the second line of the marks, now your answer is basically 0 1. Now the condition select in this case is 0 one. So, 0 1 is basically connected to; that means, this line is connected to here. So, what is this it is jump on 0 to some memory location.

(Refer Slide Time: 60:09)

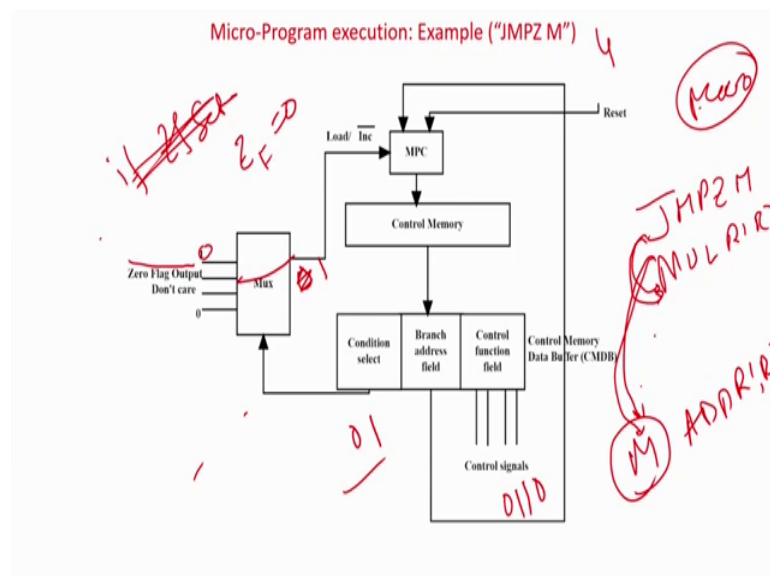


This is say fourth location, this is fifth location and this is sixth location. So, fifth location actually load the value of PC in will be connected to the memory location M that is what is going to happen in PC and sixth location is M we have already seen in the program.

. So, in that is in the fourth instruction you are going to check the 0 flag output, that is for that you are making condition select as 0 1 and if the 0 flag is set if the 0 flag is set in that case you have to basically jump; that means, in the macro program; that means, you have to load instruction number 5, but if the 0 flag is not set in that case you have to directly jump to end because in that case what is going to happen this thing will not happened and the next program in the macro routine will execute so; that means, what if the if 0 flag is not set; that means, if the 0 flag is not set let us look at the routine once again. So, if the 0 flag is not set then you are going to jump to end; that means, what? That means, they will be a jump if the 0 flag is not set.

So, in this case what it says you have to if the 0 flag is not set then you have to have a jump so in fact, that is why you have to make it 0 black flag output bar that is the 0 flag output inversion will have to collected over here because again let us look at this example slightly the more elaborate manner. So, what I have done, I have connect here the condition select is 0 1 in this case which is an explicit condition select.

(Refer Slide Time: 61:43)



Let us try to something assume something here jump on Z is M and this M location may be having called add R 1 and R 2 and the next it is the macro program and here you may have mull R 1 and R 2.

So, what it is say? If it is jump on 0 remember in the macro program this the macro program this is the macro program if it is jump on 0 that is 0 flag is set then basically you have to load the program counter corresponding to this, and if the 0 flag is not set just the program counter will be incrementing that is at the macro program level now you see what happens in the micro program level. In the micro program level will make the condition select as 0 1 because you are going to talk about 0 flag, which connected to the second bit of the marks. And here we are connecting 0 flag output bar now why bar that will be made clear so; that means, if the 0 flag is set. So, you are going to get the answer as a 0 over here because if the 0 flag is set inversion means if the 0 flag is and you will get a 0 over here.

So, if the 0 flag is set over here then what you are going to do? If the 0 flag is set because going to get 0 which is going to be here and which is going increment at the micro program level; so in the micro program level if you look at it so; that means, what just remember if the 0 flag is set if 0 flag set; that means, I am connecting 0 flag bar here. So, you are going to get a 0 over here and we just going to increment this is at the micro program level. So, just see over here. So, what happens? If the 0 flag is set. So, if the 0 flag is set you are going to get a 0 in the micro program control it is just going to increment. So, you are just going to go to memory location number 5. So, we says that Z out equal to PC; that means, the program counter value will now have the value of Z, which means in the macro program level basically the program counter is now counting to M.

Now, if the 0 flag is not set that is 0 flag is 0 that the value of 0 flag is now basically equal to 0. So, if this is the case, when you are going to have a 1 over here. If you have a 1 over here it is the condition select check it will be load and the load value here I have put 0 1 1 0 that is 6 in the micro program level; that means, if the 0 flag is not set; that means, in that case is going to jump and when you are going to jump you are going to jump at memory location number 1 1 0 1 1 0 that is 6 at the micro program level will now come to 6 which is actually end.

So, if it is end, we observe at the macro program level the PC is not updated. So, if the PC is not updated basically what happens you are going to go from jump on 0 to mul. So, in that case in the macro program level the jump has not happened slightly tricky. Whenever it is a jump in a macro program level that is here the micro program does not jump basically it just a basically it what happens, when the micro program oh sorry when the macro program jumps basically in the micro program you go from 4 to 5 which loads the PC with new value and if the macro program jumps that in terms of here to say that is end you are not actually updating the value of PC in that case the macro program does not jump.

So, slightly actually tricky once you look at it carefully it will you can understand not basically is happening slightly tricky means slightly the program has been written as turned about way that is when the macro program has to explicitly jump, the micro program is not jumping and when the micro program is jumping basically the macro program is not jumping anyway that you can look at it, but just look at the philosophy that is very important.

So, what happens in the fourth routine sorry in the fourth case which is corresponding to the explicit jump, which has to do because of the macro instruction that is the macro program the jump we are looking at it. So, this instruction was the implicit jump because of the common micro program corresponding to different macro instructions, that was the a was happening because of optimization, but in this case this is happening because of basically this is explicitly require the fourth one the fourth jump is mandatory, without this your program will not correcc run correctly, but the jump at the third location actually corresponds to optimization.

So, what happened in fourth again I am repeating because this is slightly tricky, we are giving 0 1; that means, you are checking the code corresponding to the second word second line of the marks. The second line of the marks is connected we are connecting it to output of the 0 flag bar. So, if the 0 flag is there, 0 flag bar we are connecting over here there there and in fact, what happens if the 0 flag is not set if the 0 flag is not set; that means, ZF bar is going to be 1; that means, in this case you have to load the address; that means, there is a jump in the micro instruction program

So, in this case where the jump will do? Jump will go to 6 that is going to be end and in that case what happens in the macro program in the micro program there is a jump, but in the macro program what happens basically? This updating is not reflected over there. So, the macro program will not jump it will just execute the next instruction, but if the 0 flag is set. So, we are connecting 0 flag bar there. So, if the 0 flag is set, you are going to get the 0 in the marks output.

So, 0 in the marks output means there will be no jump in the micro program that is it will not jump to 6, but you will just increment then go to 5. So, in go to 5 what happens? Z out is PC in. So, in this micro instruction basically loads the program counter, that is the macro program counter with the new PC value that which is actually offset plus Y in and actually it corresponds to jump in the macro program. So, if this is the PC in is updated. So, it will jump and basically it correspond to jump to the memory address instruction where which end stores. So, there will be jump in the macro level and after that the micro routine will end and then what happens basically? The next end PC that id micro program, program counter in case there is a jump in the macro routine, it will correspond to basically this part. So, if there is a jump jump in the macro instruction macro routine.

So, the MPC will correspond to basically the micro program corresponding to add and in fact, if basically there is the if the 0 flag is not set basically. So, in this case this macro routine macro programm instruction will execute. So, the MPC will start pointing to basically the micro routine corresponding to mul R 1 R 2, but if you think about optimization. So, we will have a single micro routine for both mul and add and in fact, the diversification will happen only when you come to instruction number 4 which will tell whether it is a mul or an add.

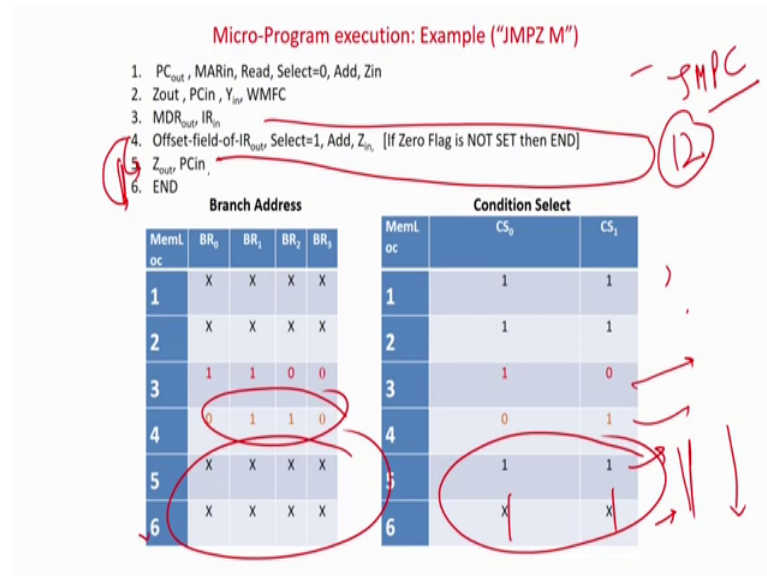
But in fact, this is actually shows you how basically a total routine executes for a given macro macro instruction like jump, and also we have explicitly shown there there can also be jumps based on if you have a single micro routine for similar type of instructions like here we had a single routine for jump on 0 and jump on carry. We are assuming that the different instruction is placed in the memory location number 12 which responds to jump on carry. So, if this is the case there will be a implicit jump from memory location 3 to 12 and again it will come back to routine number or location number 5 and it will execute 5 and 6 which is common to both.

This is an implicit one and this is there for optimization, but the other one it explicit it has to be there for the correct operation of this micro instruction jump 1 end in that case what happens? You use the 0 1 condition which checks 0 output 0 flag bar. So, if the flag is set you basically if the 0th flag is set then what happens basically? You directly execute this one should be 0 flag is set you execute this which loads the new value of updated value of PC plus offset that is your M into PC and in the macro routine there is a jump otherwise you directly come to end and that is basically no jump in the macro routine.

So, this is how it happens basically. So, you can see this one implicit this one explicit and this is the address you have to jump in this case if it is true otherwise it is similar. So, this one just you have to I am erasing out if you just look at it in your easy pace and you have to try to solve it, then you can easily find out what happens. So, slightly roundabout just you have to check. So, and then you can see the other 2 parts add from 5 and 6 basically there is no jump and no com there is nothing difference in between this micro routine these 2 routines are similar for the 2 macro instructions we are considering. So, there is no requirement of any kind of jumps over here. So, you can see that I have put 1 one or in it is a end is 6 is end so, no there is no questions I have put xx or you can again put 1 1. So, basically there is no any kind of jump for that. So, it will be single smooth increment of the micro program counter it will smooth through and that is basically going to happen.

So, this is very very important. So, just go through this example in a very nice and a very slow manner write down yourself on your notebook and see how it executes and you all things will be clear to you. So, only again just repeating only one thing you have to keep in mind the third one is for your implicit jump fourth one is for explicit jump and we are assuming there form memory.

(Refer Slide Time: 70:59)



Location number 12 the jump on carry instruction the explicit different micro instruction involving that will be placed over there. So, there will be a in case there will be a jump from 12 and again it will come back to 5.

Because 1 2 3 5 and 6 are common for both and in the last one this is an explicit jump which is jump corresponding to jump on 0. And again slightly tricky because when there is a jump from here to here there is no jump in the macro instruction macro program, but when there is no jump in the micro routine for 4 2 4 2 5 this is a jump corresponding to the macro routine slightly tricky you solve it and you are going to understand.

So, in a nutshell what we have seen in this? We have seen that how micro programs can be optimized 1 is by compressing the by encoding and compressing the control bits and one way by actually margin most of the common macro instructions the micro programs are merged. So, if you margin there will be lots of jumps here and there implicitly required, because the common parts will be written only once and for the different part of the different macro instructions you have to jump here and there and come back and also there will be some explicit jump instructions like jump 1 Z there is jump on carry etcetera, for which there will be explicit jumps.

So, in that case obviously, the micro routine for jump instruction with carry out those explicit jumps and this is an example by which I have shown how a complete micro routine is executed for a macro-macro instruction and also how if there are 2 macro

instructions for which there is a single micro routine how implicitly is there there will be handled.

So, this is a slightly complicated example, which actually involves both the type of jumps in a single micro routine.

(Refer Slide Time: 72:37)

Questions and Objectives

Q1: To reduce the size of the control work, we generally do some encoding of the control signals of the processor. Mention the factors that we need to consider while encoding the control signals. Explain the encoding techniques by considering a particular CPU organization and its control signals.

Q2: If different micro-programs are used for each different machine instructions then the size of the overall micro-program (for a code) would be very large. How can this issue be addressed so that the length of the overall micro-program is reduced?

Q3: Illustrate the micro-program based controller for executing the conditional jump instruction "ADD R1, M".

- **Comprehension: Explain--** Explain about the branch control mechanism in micro-program.
- **Evaluation: Estimate--** Estimate the size of control store to implement the control unit.
- **Application: Demonstrate--** Demonstrate the impact on performance of the control unit depending on the format of control word.

So before we come to this unit let us see the some 1 or 2 small questions, which will see that how we have made the objectives. So, we said that to reduce the control what we generally do some encoding of the control signal. Mention the factors we need to do the encoding that is clustering etcetera explain the encoding techniques in a particular CPU organization, single bus architecture and it is control signals.

So, basically it will easily satisfy those objective estimate the size of control store to implement the control unit demonstrate and demonstrate the impact of performance of control unit depending on the format of control word. So, these two of this will be satisfied, once you at the enable to solve this problem. In fact, you should be able to do it because you have taken explicit example, to show how it can be optimized both of this level as well as this level. These question number 1 mainly tries to optimize the word sizes based on encoding. So, we will be able to do so, do so, because you are getting some practical example and we will be able to satisfy these 2.

If different micro routines are used for each different instruction then the overall would be very large how this can be addressed that is by merging at this level though these also you have explicitly addressed and I have given you a simple example although not very elaborately explained, then I have taken one jump on 0 and jump on carry, similarly you can take very similar instructions like add, sub, multiply and try to write a common micro routine for that and then see how it how it basically optimizes.

So, in that case you are going to also have to basically satisfy this objective because there will be lot of implicit jumps because of this merging of large number of micro routines for different macro program macro instructions. So, not only you will satisfy this, also you will satisfy the objective because you have to know about basically the branch control. And finally, I have ask you to write a micro program control for execute the instruction add R 1 M.

So, of course, this one will basically requires this 2 objectives because I I would request you to write this add M M add R 1 M and try to optimize the signals in the horizontal level and also try to add another instruction like subtract R 1 M and try to optimize at this level and then try to see the impact of both how many branches you require and also how many you save it both this have action both the horizontal level as well as vertical level. Horizontal means you reduce the one memory called micro program control memory word size and in this case you reduce the number of instructions by merging the common micro routines for different macro be a macro instructions right.

So, with this we come to end of this unit and the next you will be basically we will be trying to focus we will just give you an idea of the whole you whole different units or different lectures we have done in this module, we basically have given an undertaking or assume that everything a single bus architecture. Next will next you it will be dedicated we will try to show how things change if there are more number of buses 3 buses, 4 buses or a multi bus architecture.

So, next will not dealing details that how your micro program will change, how your finite state machine based implementation will change if there are multiple buses you will be able to do it that will be very simple once we completely discussion on the multi bus architecture. So, in multi bus architectures you will find out how these micro control signals will change, what are the difference if you are having a 2 bus or a 3 bus

architecture 1 bus and a 3 bus architecture, and accordingly you will be able to adopt those concepts to develop micro program control routine as well as your hardwired that is finite state machine based program.

So, next unit will be dedicated on looking at how things are going to change if you are having a multi bus architecture. Those simple idea will be will be will enable you to take whatever we have learn till now from single bus architecture to multi bus architecture.

Thank you.