

**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 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 programmed control unit that given a mainly we have discussed 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 it 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 to the ports.

But then we have seen that there are lot of zeros 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 programmed 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 a 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 the unit.

(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

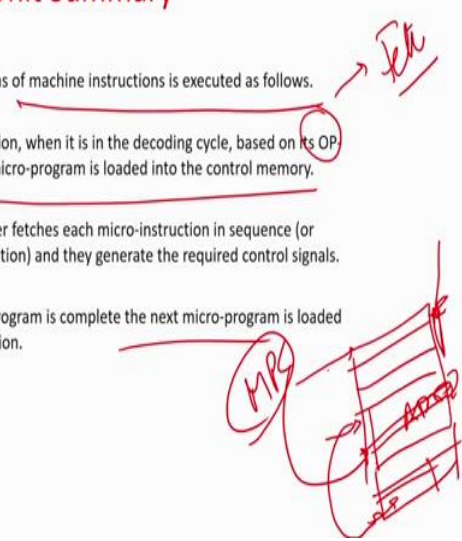


(Refer Slide Time: 02:02)

### Unit Summary

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

- 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.



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 when it is in the decoding cycle based on the Op-Code, the corresponding micro program is loaded into the memory. For first phase they have means very simple that is fetch.

So, already we have seen that for each fetch, the corresponding three micro microinstruction are executed, after that basically it waits; that means what? So, whenever a new instruction has to be executed, the micro program *PC* is pointed to the instruction which corresponds to basically your fetch. So, if you think about this is your micro program memory, may be 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 decode what the exact instruction is. If it is add, if it is store, 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 program or 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 its micro program. So, this is one thing we are going to see for some temp template instructions.

The micro program counter fetches each micro instruction in sequence and then 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 the instruction register which is being decoded, that is the real execution starts, it will keep on doing it till it gets end.

So, one the present micro program is called is complete, the next micro program is loaded based on the new instruction. So, once this add has been done then again automatically 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, my *MPC* will be pointing out to this micro program memory which corresponds to fetch. So, automatically again new instruct, new macro instruction will fetch and the procedure 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 that 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 starts and it will keep on going in. So, that is actually what is the idea of a 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.

Scribble  
APP

So, now how you can do it? A very simple solution; for each machine instruction you write a 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 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 there will be again fetch and again the corresponding real execution micro program will be there and you keep on doing it; 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 etcetera. And if you write different micro codes for each addressing modes, each type of instruction type so it will be a huge memory, but you can do it. 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 be 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 memory for fetch and then for example, if we have subtract and say add 2 instructions are there. So, only difference will be basically is subtract we will make the ALU to sub, subtraction will require the signal which corresponds to the ALU sub should be made 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 a 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 you just go to one part then the other part can be common, so that way 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 require lot of branching parts, but that is however it's ok. 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 sub equal will be equal to 1 and in this 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 the micro programs are written in a branching fashion.

So, that is one of the key idea of this unit which we will 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 an 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, they 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 met. So, what is the objective it's a 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 instructions; obviously, micro program has to be have follow a branching path 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, and also we have to find out the how many signals are there. So, all these based on a certain memory arch certain architecture of a single bus or multi bus, you should be able to. Mainly we will be concentrating around 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 a very compressed format. So, if the control unit will take no time it is a very flat like of horizontal 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's 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 is 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 call it as a micro program control memory.

Since this machine instruction is executed corresponding to a micro program, it starts it 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 a location in the main in the micro program control memory, which corresponds to the micro program for that macro instruction.

And as I told you generally we are trying to have we 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 are diversions we can put jumps, so that how basically it works.

(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 address generator unit" is circled in red. "branch address generator unit" is written in red. "main program" is written in red.

And 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 the status as well as the from the some signals actually which are coming from the I/O devices. So, that is very similar to a macro program part.

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, the micro program will branch correspondingly.



Another is micro program inherent branch; that is 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 require two different micro programs for that we will have say, but depending on the if it's a add or 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 whatever are the common instructions. So, therefore, we require many branches in a micro program execution compared to a macro program.

So, basically a full program written in terms of machine instruction basically executes as follows. For each machine instruction that is the macro instruction where it is in the 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, ADD, SUB, FIN

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 the 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 code to optimize space, then for all any of these instructions you start pointing to the start point of that corresponding instruction.

Now whether it is an 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 R1, M* and say *subtract R1, M*. So, only one point that is the control of the ALU will be different. So, only that particular it will diverge again come back because after doing the computation, again you have to write back the value of because say many parts will 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 execute in 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 the instruction register tells where to jump. So, how it exactly implements we will see.

So, once the micro program is controlled 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 to the new instruction or the new macro program macro instruction ok.