Control unit design

Design of Control Unit

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. As for example, during the fetch phase, CPU has to generate PC_{out} signal along with other required signal in the first clock pulse. In the second clock pulse CPU has to generate PC_{in} signal along with other required signals. So, during fetch phase, the proper sequence for generating the signal to retrieve from and store to PC is PC_{out} and PC_{in}.

To generate the control signal in proper sequence, a wide variety of techniques exist. Most of these techniques, however, fall into one of the two categories,

1. Hardwired Control
2. Microprogrammed Control.

Hardwired Control

In this hardwired control techniques, the control signals are generated by means of hardwired circuit. The main objective of control unit is to generate the control signal in proper sequence.

Consider the sequence of control signal required to execute the ADD instruction that is explained in previous lecture. It is obvious that eight non-overlapping time slots are required for proper execution of the instruction represented by this sequence.

Each time slot must be at least long enough for the function specified in the corresponding step to be completed. Since, the control unit is implemented by hardwire device and every device is having a propagation delay, due to which it requires some time to get the stable output signal at the output port after giving the input signal. So, to find out the time slot is a complicated design task.

For the moment, for simplicity, let us assume that all slots are equal in time duration. Therefore the required controller may be implemented based upon the use of a counter driven by a clock.

Each state, or count, of this counter corresponds to one of the steps of the control sequence of the instructions of the CPU.

In the previous lecture, we have mentioned control sequence for execution of two instructions only (one is for add and other one is for branch). Like that we need to design the control sequence of all the instructions.

By looking into the design of the CPU, we may say that there are various instruction for add operation. As for example,

ADD	NUM	R_1	Add the contents of memory location specified by NUM to the contents of register R_1. $R_1 \leftarrow R_1 + [NUM]$
ADD	R_2	R_1	Add the contents of register R_2 to the contents of register R_1. $R_1 \leftarrow R_1 + R_2$

The control sequence for execution of these two ADD instructions are different. Of course, the fetch phase of all the instructions remain same.

It is clear that control signals depend on the instruction, i.e., the contents of the instruction register. It is also observed that execution of some of the instructions depend on the contents of condition code or status flag register, where the control sequence depends in conditional branch instruction.

Hence, the required control signals are uniquely determined by the following information:

- **Contents of the control counter.**
- **Contents of the instruction register.**
- **Contents of the condition code and other status flags.**

The external inputs represent the state of the CPU and various control lines connected to it, such asMFC status signal. The condition codes/ status flags indicates the state of the CPU. These includes the status flags like carry, overflow, zero, etc.

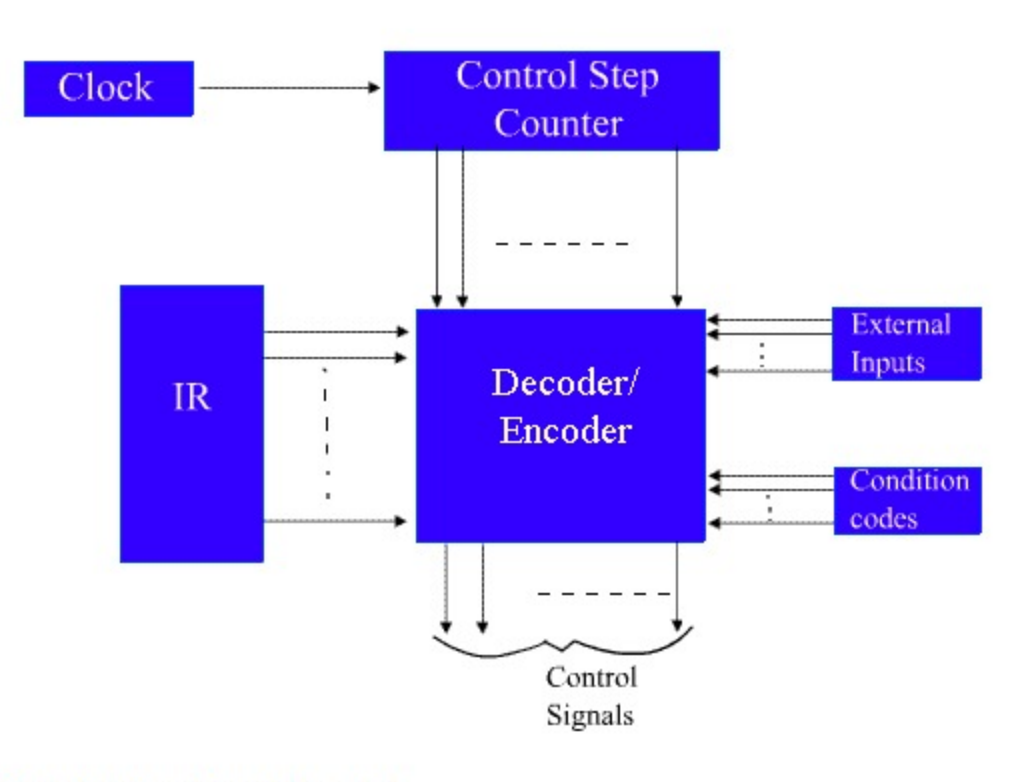


Figure 5.10: Organization of control unit.

The structure of control unit can be represented in a simplified view by putting it in block diagram. The detailed hardware involved may be explored step by step. The simplified view of the control unit is given in the Figure 5.10.

The decoder/encoder block is simply a combinational circuit that generates the required control outputs depending on the state of all its input.

The decoder part of decoder/encoder part provide a separate signal line for each control step, or time slot in the control sequence. Similarly, the output of the instructor decoder consists of a separate line for each machine instruction loaded in the IR, one of the output line INS_1 to INS_m is set to 1 and all other lines are set to 0.

The detailed view of the control unit organization is shown in the Figure 5.11.

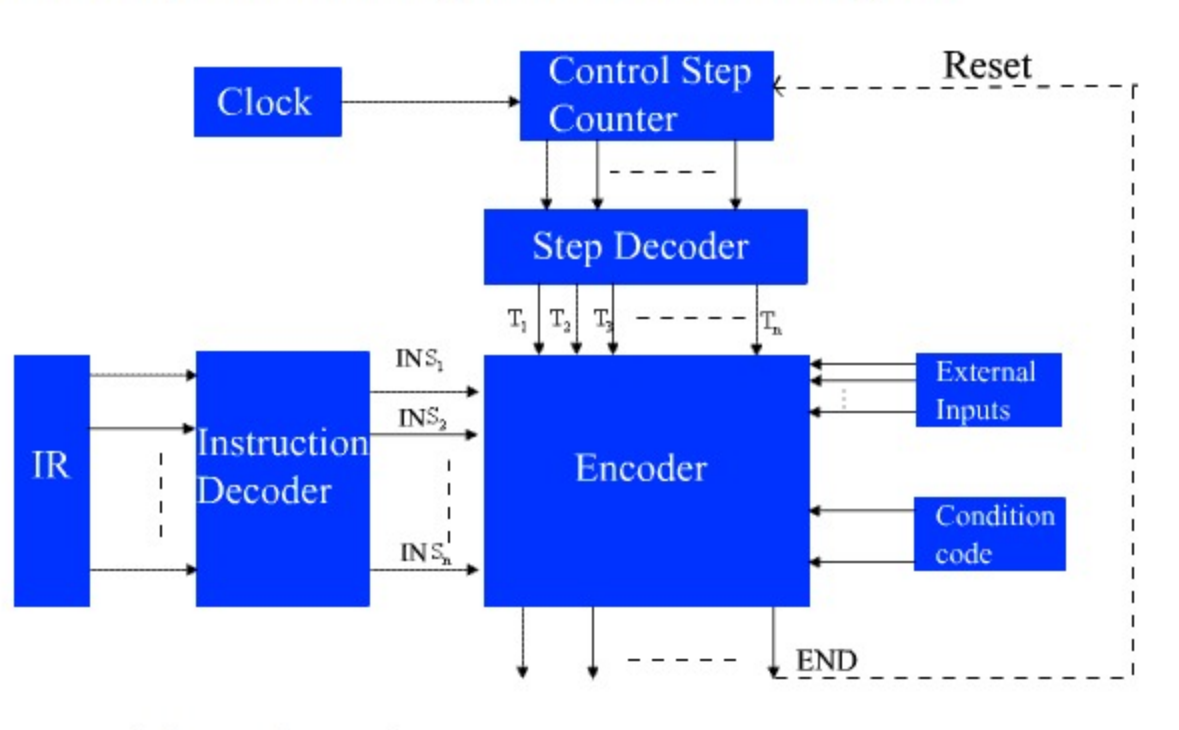


Figure 5.11: Detailed view of Control Unit organization

All input signals to the encoder block should be combined to generate the individual control signals.

In the previous section, we have mentioned the control sequence of the instruction,

"Add contents of memory location address in memory direct made to register R_1 (ADD_MD)",

"Control sequence for an unconditional branch instruction (BR)",

also, we have mentioned about Branch on negative (BRN).

Consder those three CPU instruction ADD_MD, BR, BRN.

It is required to generate many control signals by the control unit. These are basically coming out from the encoder circuit of the control signal generator. The control signals are: PC_{in}, PC_{out}, Z_{in}, Z_{out}, MAR_{in}, ADD, END, etc.

By looking into the above three instructions, we can write the logic function for Z_{in} as :

$Z_{in} = T_1 + T_6 . ADD\_MD + T_5 . BR + T_5 . BRN + \ldots\ldots\ldots\ldots$

For all instructions, in time step1 we need the control signal Z_{in} to enable the input to register Z_{in} time cycle T_6 of ADD_MD instruction, in time cycle T_5 of BR instruction and so on.

Similarly, the Boolean logic function for ADD signal is

$$ADD = T_1 + T_6 . ADD\_MD + T_5 . BR + \ldots\ldots\ldots\ldots$$

These logic functions can be implemented by a two level combinational circuit of AND and OR gates.

Similarly, the END control signal is generated by the logic function :

$END = T_8 . ADD\_MD + T_7 . BR + (T_7 . N + T_4 . \overline{N}) . BRN + \ldots\ldots\ldots\ldots$

This END signal indicates the end of the execution of an instruction, so this END signal can be used to start a new instruction fetch cycle by resetting the control step counter to its starting value.

The circuit diagram (Partial) for generating Z_{in} and END signal is shown in the Figure 5.12 and Figure 5.13 respectively.

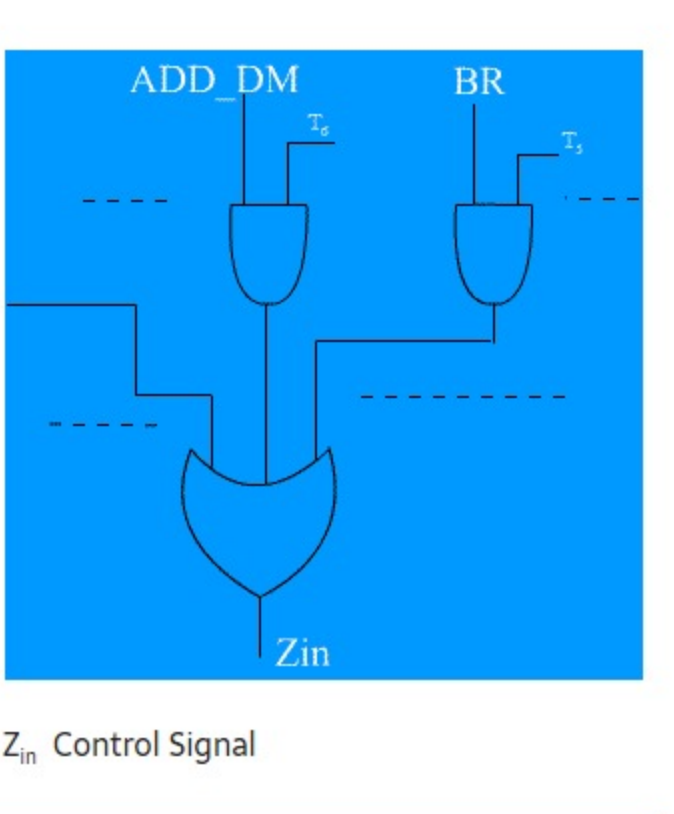


Figure 5.12: Generation of Z_{in} Control Signal

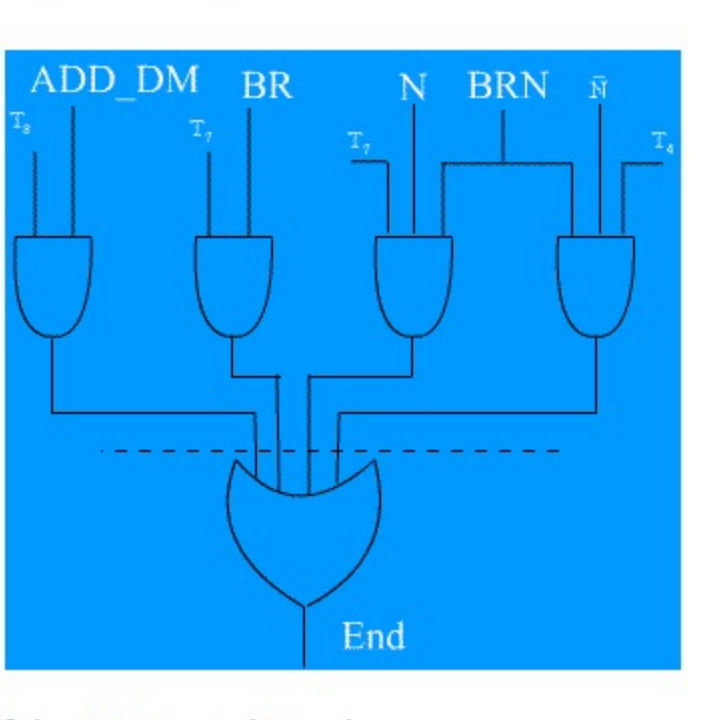


Figure 5.13: Generation of the END Control Signal

The signal ADD_MD, BR, BRN etc. are coming from instruction decoder circuits which depends on the contents of IR.

The signal T_1, T_2, T_3 etc are coming out from step decoder depends on control step counter.

The signal N (Negative) is coming from condition code register.

When wait for MFC (WMFC) signal is generated, then CPU does not do any works and it waits for anMFC signal from memory unit. In this case, the desired effect is to delay the initiation of the next control step until the MFC signal is received from the main memory. This can be incorporated by inhibiting the advancement of the control step counter for the required period.

Let us assume that the control step counter is controlled by a signal called RUN.

By looking at the control sequence of all the instructions, the WMFC signal is generated as:

$WMFC = T_2 + T_5 . ADD\_MD + \ldots\ldots\ldots\ldots$

The RUN signal is generated with the help of WMFC signal and MFC signal. The arrangement is shown in the Figure 5.14.

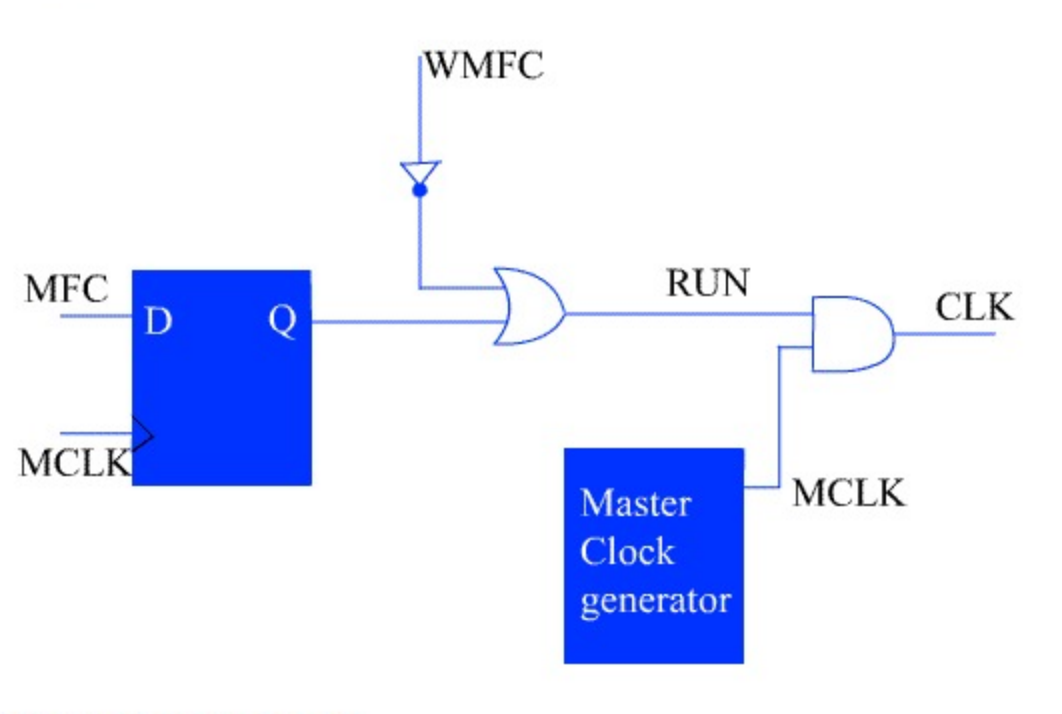


Figure 5.14: Generation of RUN signal

The MFC signal is generated by the main memory whose operation is independent of CPU clock. Hence MFC is an asynchronous signal that may arrive at any time relative to the CPU clock. It is possible to synchronized with CPU clock with the help of a D flip-flop.

When WMFC signal is high, then RUN signal is low. This run signal is used with the master clock pulse through an AND gate. When RUN is low, then the *CLK* signal remains low, and it does not allow to progress the control step counter.

When the MFC signal is received, the run signal becomes high and the CLK signal becomes same with the MCLK signal and due to which the control step counter progresses. Therefore, in the next control step, the WMFC signal goes low and control unit operates normally till the next memory access signal is generated.

Programmable Logic Array

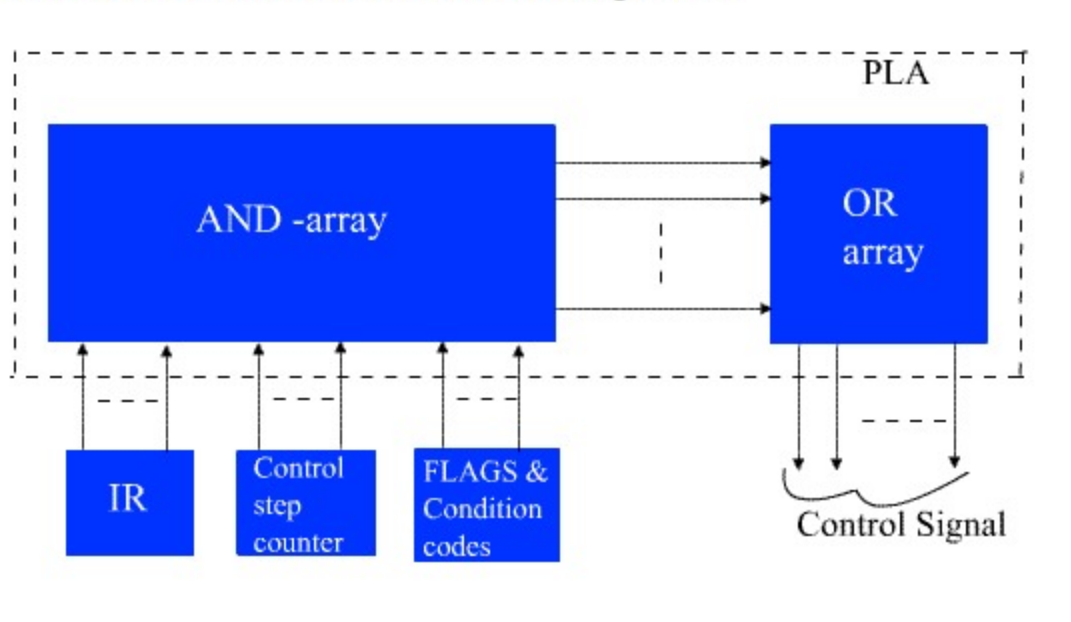
In this discussion, we have presented a simplified view of the way in which the sequence of control signals needed to fetch and execute instructions may be generated.

It is observed from the discussion that as the number of instruction increases the number of required control signals will also increase.

In VLSI technology, structure that involve regular interconnection patterns are much easier to implement than the random connections.

One such regular structure is PLA (programmable logic array). PLAs are nothing but the arrays of AND gates followed by array of OR gates. If the control signals are expressed as sum of product form then with the help of PLA it can be implemented.

The PLA implementation of control unit is shown in the Figure 5.16.



Microprogrammed Control

In hardwired control, we saw how all the control signals required inside the CPU can be generated using a state counter and a PLA circuit.

There is an alternative approach by which the control signals required inside the CPU can be generated . This alternative approach is known as microprogrammed control unit.

In microprogrammed control unit, the logic of the control unit is specified by a microprogram.

A microprogram consists of a sequence of instructions in a microprogramming language. These are instructions that specify microoperations.

A microprogrammed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstructions and (2) generating control signals to execute each microinstruction.

The concept of microprogram is similar to computer program. In computer program the complete instructions of the program is stored in main memory and during execution it fetches the instructions from main memory one after another. The sequence of instruction fetch is controlled by program counter (PC) .

Microprogram are stored in microprogram memory and the execution is controlled by microprogram counter (μ PC).

Microprogram consists of microinstructions which are nothing but the strings of 0's and 1's. In a particular instance, we read the contents of one location of microprogram memory, which is nothing but a microinstruction. Each output line (data line) of microprogram memory corresponds to one control signal. If the contents of the memory cell is 0, it indicates that the signal is not generated and if the contents of memory cell is 1, it indicates to generate that control signal at that instant of time.

First let me define the different terminologies that are related to microprogrammed control unit.

Control Word (CW) :

Control word is defined as a word whose individual bits represent the various control signal. Therefore each of the control steps in the control sequence of an instruction defines a unique combination of 0s and 1s in the CW.

A sequence of control words (CWs) corresponding to the control sequence of a machine instruction constitutes the microprogram for that instruction.

The individual control words in this microprogram are referred to as microinstructions.

The microprograms corresponding to the instruction set of a computer are stored in a aspecial memory which will be referred to as the microprogram memory. The control words related to an instructions are stored in microprogram memory.

The control unit can generate the control signals for any instruction by sequencially reading the CWs of the corresponding microprogram from the microprogram memory. To read the control word sequentially from the microprogram memory a microprogram counter (μ PC) is needed.

The basic organization of a microprogrammed control unit is shown in the Figure 5.17.

The "starting address generator" block is responsible for loading the starting address of the microprogram into the μ PC everytime a new instruction is loaded in the IR.

The μ PC is then automatically incremented by the clock, and it reads the successive microinstruction from memory.

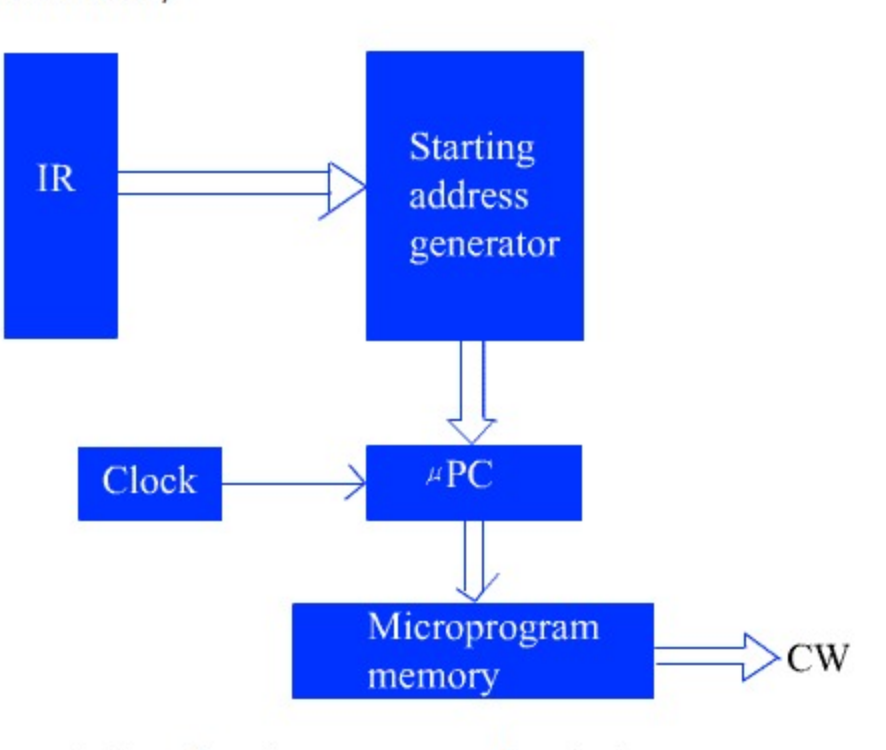


Figure 5.17: Basic organization of a microprogrammed control

Each microinstruction basically provides the required control signal at that time step. The microprogram counter ensures that the control signal will be delivered to the various parts of the CPU in correct sequence.

We have some instructions whose execution depends on the status of condition codes and status flag, as for example, the branch instruction. During branch instruction execution, it is required to take the decision between the alternative action.

To handle such type of instructions with microprogrammed control, the design of control unit is based on the concept of conditional branching in the microprogram. For that it is required to include some conditional branch microinstructions.

In conditional microinstructions, it is required to specify the address of the microprogram memory to which the control must direct. It is known as branch address. Apart from branch address, these microinstructions can specify which of the states flags, condition codes, or possibly, bits of the instruction register should be checked as a condition for branching to take place.

To support microprogram branching, the organization of control unit should be modified to accommodate the branching decision.

To generate the branch address, it is required to know the status of the condition codes and status flag. To generate the starting address, we need the instruction which is present in IR. But for branch address generation we have to check the content of condition codes and status flag.

The organization of control unit to enable conditional branching in the microprogram is shown in the Figure 5.18.

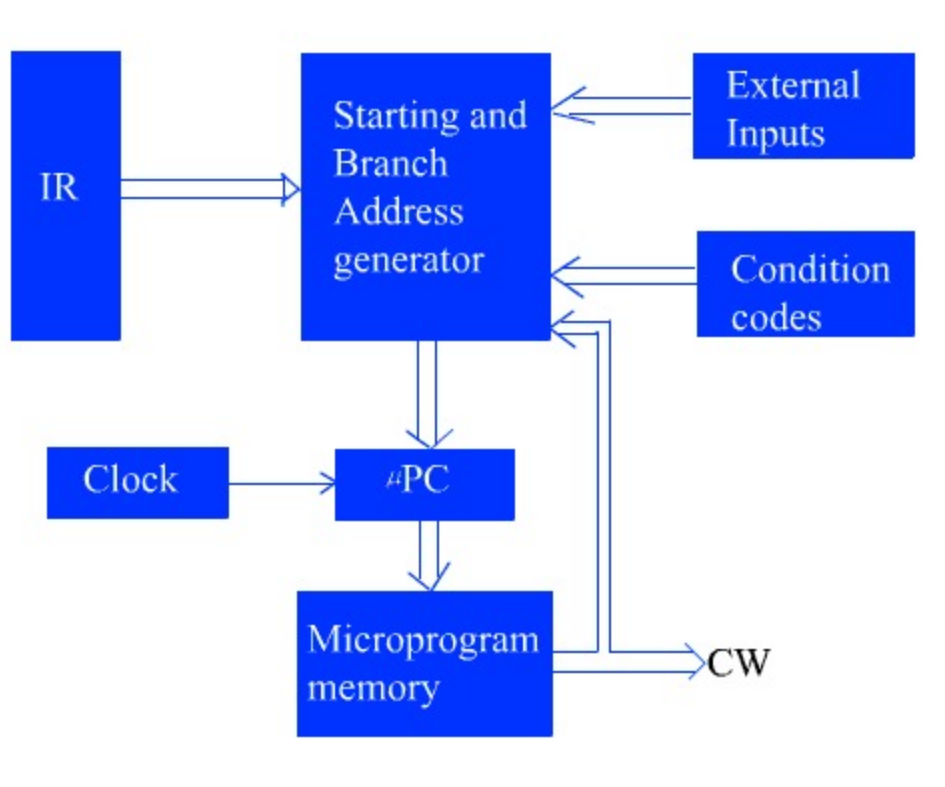


Figure 5.18: Organization of microprogrammed control with conditional branching.

The control bits of the microinstructions word which specify the branch conditions and address are fed to the "Starting and branch address generator" block.

This block performs the function of loading a new address into the μ PC when the condition of branch instruction is satisfied.

In a computer program we have seen that execution of every instruction consists of two part – fetch phase and execution phase of the instruction. It is also observed that the fetch phase of all instruction is same.

In microprogrammed controlled control unit, a common microprogram is used to fetch the instruction. This microprogram is stored in a specific location and execution of each instruction start from that memory location.

At the end of fetch microprogram, the starting address generator unit calculate the appropriate starting address of the microprogram for the instruction which is currently present in IR. After the μ PC controls the execution of microprogram which generates the appropriate control signal in proper sequence.

During the execution of a microprogram, the μ PC is always incremented everytime a new microinstruction is fetched from the microprogram memory, except in the following situations :

1. When an End instruction is encountered, the μ PC is loaded with the address of the first CW in the microprogram for the instruction fetch cycle.
    1. loaded with the branch address.

Let us examine the contents of microprogram memory and how the microprogram of each instruction is stored or organized in microprogram memory. Consider the two example that are used in our previous lecture . First example is the control sequence for execution of the instruction "Add contents of memory location addressed in memory direct mode to register R1".

Steps	Actions
1.	PC_{out}, MAR_{in}, Read, Clear Y, Set carry-in to ALU, Add, Z_{in}
2.	Z_{out}, PC_{in}, Wait For MFC
3.	MDR_{out}, IR_{in}
4.	Address-field-of-IR_{out}, MAR_{in}, Read
5.	$R1_{out}$, Y_{in}, Wait for MFC
6.	MDR_{out}, Add, Z_{in}
7.	Z_{out}, $R1_{in}$
8.	END

Control sequence for Conditional Branch instruction (BRN) Branch on negative)

Steps	Actions
1.	PC_{out}, MAR_{in}, Read, Clear Y, Set Carry-in to ALU, Add , Z_{in}
2.	Z_{out}, PC_{in}, Wait for MFC
3.	MDR_{out}, IR_{in}
4.	PC_{out}, Y_{in}
5.	Address field-of IR_{out}, Add, Z_{in}
6.	Z_{out}, PC_{in}
7.	End

First consider the control signal required for fetch instruction , which is same for all the instruction, we are listing them in a particular order.

PC_{out}	MAR_{in}	Read	Clear Y	Set Carry to ALU	Add	Z_{in}	Z_{out}	PC_{in}	WMFC	MDR_{out}	IR_{in}

The control word for the first three steps of the above two instruction are : (which are the fetch cycle of each instruction as follows):

Step1	1	1	1	1	1	1	1	0	0	0	0	0	- - -
Step2	0	0	0	0	0	0	0	1	1	1	0	0	- - -
Step3	0	0	0	0	0	0	0	0	0	0	1	1	- - -

*We are storing this three CW in memory location 0, 1 and 2. Each instruction starts from memory location 0. After executing upto third step, i.e., the contents of microprogram memory location 2, this control word stores the instruction in **IR**. The starting address generator circuit now calculate the starting address of the microprogram for the instruction which is available in IR.*

Consider that the microprogram for add instruction is stored from memory location 50 of microprogram memory. So the partial contents from memory location 50 are as follows :

Location	**50**	0	1	1	0	0	0	0	0	0	0	0	0	– –	– –	– –
	51	0	0	0	0	0	0	0	0	0	1	0	0	– –	– –	– –
	and so on															

When the microprogram executes the End microinstruction of an instruction, then it generates the End control signal. This End control signal is used to load the μ PC with the starting address of fetch instruction (In our case it is address 0 of microprogram memory). Now the CPU is ready to fetch the next instruction from main memory.

From the discussion, it is clear that microprograms are similar to computer program, but it is in one level lower, that's why it is called microprogram.

For each instruction of the instruction set of the CPU, we will have a microprogram.

While executing a computer program, we fetch instruction by instruction from main memory which is controlled by program counter(PC).

When we fetch an instruction from main memory, to execute that instruction , we execute the microprogram for that instruction. Microprograms are nothing but the collection of microinstrctions. These microinstructions will be fetched from microprogram memory one after another and its sequence is maintained by μ PC. Fetching of microinstruction basically provides the required control signal at that time instant.