

---

# UNIT 4 MICROPROGRAMMED CONTROL UNIT

---

- 4.0 Introduction
- 4.1 Objectives
- 4.2 What is a Micro-programmed control unit ?
- 4.3 Wilkes Control
- 4.4 The Microinstruction
  - 4.4.1 Types of Microinstructions
  - 4.4.2 Control Memory Organisation
  - 4.4.3 Microinstruction formats
- 4.5 A Simple Structure of Control Unit.
- 4.6 Microinstruction Sequencing
- 4.7 Microinstruction Execution
- 4.8 Machine Startup
- 4.9 Summary
- 4.10 Model Answers

---

## 4.0 INTRODUCTION

---

This is the last unit of the block on CPU organisation. We have already discussed in the earlier units about the instruction sets, Register set, ALU organisation and control unit organisation. In this unit, we will discuss about the microprogrammed control unit, which are quite popular in modern computers because of flexibility in designing. We will start the discussion with several definitions about the unit followed by Wilkes Control unit. We will also discuss about microinstructions and a simple structure of such a unit. Finally, we will visit the concepts involved in microinstruction sequencing and execution. Finally, discuss about the aspects of machine startup.

The discussion given in this unit are at conceptual level. For implementation based examples of microprogrammed control unit you can refer to the further readings.

---

## 4.1 OBJECTIVES

---

At the end of this unit you should be able to :

- define the microprogrammed control unit
- define the term microinstruction
- identify types and formats of microinstruction
- discuss about the control memory
- explain the working of a microprogrammed control unit
- define the microinstruction addressing
- explain microinstruction encoding
- identify the process of machine startup

---

## 4.2 WHAT IS A MICROPROGRAMMED CONTROL UNIT ?

---

As discussed earlier in Unit 3, the control unit seems to be a very simple unit, however, if we try to implement such a unit through hardware, we are bound to face problems. The hardwired control unit lack flexibility in design. In addition, it is quite difficult to design, test and implement as in many computers the number of control lines is in hundreds.

Is there any alternate approach of implementing control unit? What about a programming approach for implementing control unit? Can we somehow implement the sequence of execution of micro-operations through a program? Such a program will consist of

instructions, with each of the instruction describing :

- one or more micro-operations to be executed
- and the information about the microinstruction to be executed next.

Such an instruction is termed as **microinstruction** and such a program is termed as a **microprogram** or **firmware**. The firmware is a midway between hardware and software. Firmware, in comparison to hardware is easier to design, whereas in comparison to software, is difficult to write. A control unit provides a set of control signal lines, distributed throughout the CPU, with each of the lines representing a zero or one. Therefore, a microinstruction is made responsible for generating control signals for desired control lines to implement a desired micro-operation. For example, to implement a register to register transfer operation, output of the source register and input of destination register need to be enabled by the respective control signal lines via a microinstruction. Thus, each microinstruction can generate a set of control signals on the control lines which in turn implement one or more micro-operations. A set of control signals with each bit representing a single control line is called a control word.

Thus, a microinstruction can cause execution of one or more micro-operations, and a sequence of microinstructions, that is microprogram, can cause execution of an instruction. The microprograms are mostly stored in read only memory, known as **control store** or **control memory**, as alteration in control store are generally not required after the production of the control unit. What about having read-write control memory? In such a memory instruction set of a computer can be changed by simply modifying the microprograms for various op-codes. Such a computer can be tailored for specific applications on the basis of microprograms. A computer having writable control memory are termed as "dynamically microprogrammable" as the content of control memory for such computers can be changed under program control. The control memory is a word organised unit with each word representing a microinstruction. Please note that the computers which have microprogrammed control unit, have two separate memories; a main memory and the control memory.

How will the microprogrammed control unit control the instruction execution?

Well, the first point of mention here is that each computer instruction fetched from the main memory leads to initiation of a series of microinstructions from the control memory. These microinstructions issued micro-orders to the CPU for instruction fetch, calculation of effective address of operands, and execution of the instruction and then prepares it again for fetching next instruction from the main memory. Thus, for each instruction, one must determine the series of microinstructions which will be needed to get that instruction executed. This series of microinstruction will be different for different instructions.

#### Advantage of Microprogrammed Control Unit

Since the microprograms can be changed relatively easily, therefore, microprogrammed control units are very flexible in comparison to hardwired control units.

#### Disadvantages

- Hardware cost is more because of the control memory and its access circuitry
- This is slower than hardwired control unit because the microinstruction are to be fetched from the control memory which is time consuming.

But, these disadvantages are gradually getting phased out as the new memory technologies are cheap and high-speed memories are gradually becoming common.

---

### 4.3 WILKES CONTROL

---

In 1951 Wilkes had proposed the use of microprogram control unit. In Wilkes design a microinstruction has two major components:

- Control field and
- address field

The control field indicates the control lines which are to be activated and the address field provides the address of the next microinstruction to be executed. Figure 1 shows a simple example of Wilkes control unit design.

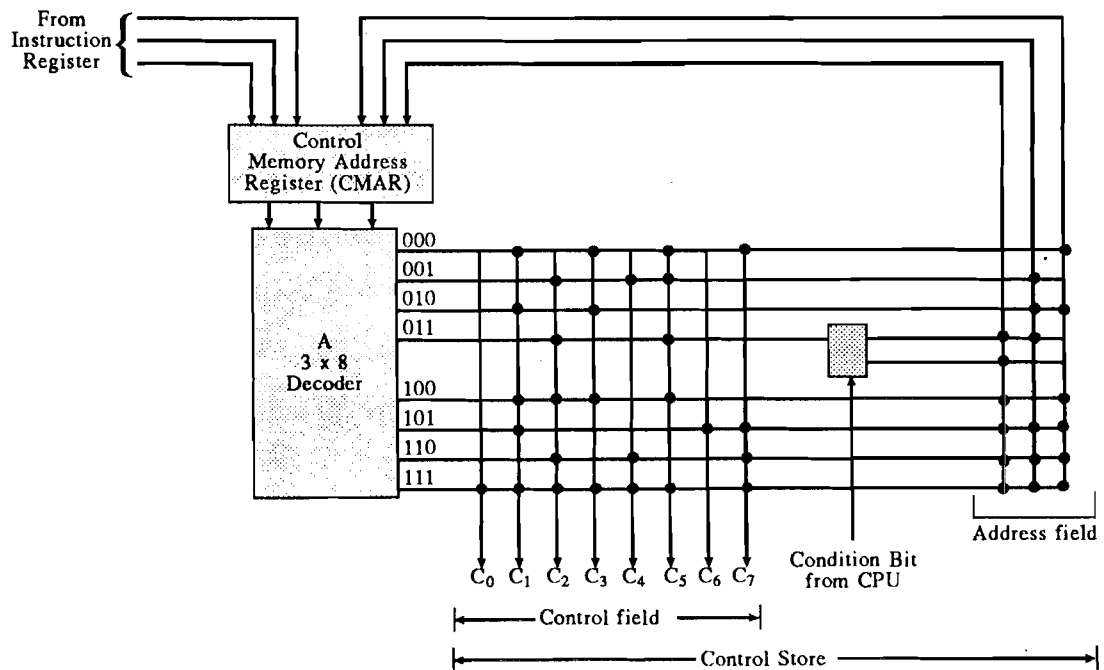


Figure 1 : Wilkes Control Unit

The control memory in this control is organised as a program logic array like matrix made of diodes, a simple electronic device. This is partial matrix and consist of two components, the control signals and the address of next microinstruction. The control memory access register (CMAR) can be loaded by the instruction code register or by the address field of the control matrix. The control memory address register on taking an input from the instruction register provides a 3 bit address to the  $3 \times 8$  decoder. This is an entry point address to the control memory. On the basis of this address, decoder activates one of the eight output lines (horizontal). This activated line, in turn, generates control signals and the address of the next microinstruction to be executed. This address is once again fed to the CMAR resulting in activation of another control line and address field. This cycle is repeated till the execution of the instruction is achieved. For example, in the given figure, instruction register's op-code 000 causes the decoder to have an entry address for a machine instruction in control memory at line 000. The decoder activates the lines in the sequence given below: (Please note this is valid only for the diagram given here which is only an indicative example and not a complete Wilkes Control Unit.)

Decode line activated	Control signals generated	Address of next microinstruction
000	$C_1, C_3, C_5, C_7$	001
001	$C_2, C_4, C_5$	010
010	$C_1, C_3$	011
011	$C_2, C_5$	?

Please note the '?' above. How do we get the address of the next microinstructions to be executed on activating the decode line 011? We have two possible options in the figure 1. A typical requirement of a control unit is that it must respond to an external control condition. Thus, making conditional jumps possible within a microprogram. This is demonstrated in the Wilkes control of the figure 1. The external condition switch causes the control unit to follow one of the two available paths:

EITHER

011	$C_2, C_5$	If external condition is true then 110
110	$C_2, C_4, C_7$	111
111	$C_0, C_1, C_2, C_3, C_4, C_5, C_7$	This may cause loading of next instruction in IR

OR

011	$C_2, C_5$	If external condition is false then 100
100	$C_1, C_2, C_3, C_5$	101
101	$C_1, C_6, C_7$	111
111	$C_0, C_1, C_2, C_3, C_4, C_5, C_7$	This may cause loading of next instruction in IR.

Wilkes not only proposed this control scheme but had proposed an example structure for the

first microprogrammed control processing unit. However, we will not go in the details of this structure here. You can refer to further readings for this structure.

### Check Your Progress 1

1. What is firmware? How is it different from software?

.....

.....

2. State true or false

- (a) A microinstruction can initiate only one micro-operation.

True ☐ False ☐

- (b) A control word is equal to a machine word.

True ☐ False ☐

- (c) A dynamically microprogrammable control memory is writable also.

True ☐ False ☐

- (d) Microprogrammed control is faster than hardwired control.

True ☐ False ☐

- (e) Wilkes control do not provide a branching microinstruction.

True ☐ False ☐

3. What will be the sequence of decode lines activated in the Wilkes control example of figure 1, if the entry address for a machine instruction is 010 and the conditional bit value is true?

.....

.....

---

## 4.4 THE MICROINSTRUCTION

---

After discussing about the Wilkes Control, we have some idea of microprogrammed control unit. Key to a microprogrammed control unit is a microinstruction. So, let us explore more about the micro-instructions in this section.

A microinstruction, as defined earlier, is an instruction of a microprogram. It specifies one or more micro-operations, which can be executed simultaneously. On executing a microinstruction a set of control signals are generated which in turn cause the desired micro-operation/micro-operations to happen.

### 4.4.1 Types of Microinstructions

In general, the micro-instructions can be categorised in two general types. These are branching and non-branching. A non-branching micro-instruction is the one, in which the next micro- instruction which is executed is the one following the current micro-instruction. However, this sequence of micro-instructions is relatively small and lasts only for 3 or 4 micro-instructions.

A conditional branching micro-instruction is a desirable instruction. The condition which is to be tested is a conditional variable or a flag generated by an ALU operation. Normally the branch address is contained in the microinstruction itself.

### 4.4.2 Control Memory Organisation

The next important question about the microinstruction is; how are they organised in the control memory?

One of the simplest way to organise control memory is to arrange microinstructions for various sub cycles of the machine instruction in the memory. Figure 2 shows such an organisation.

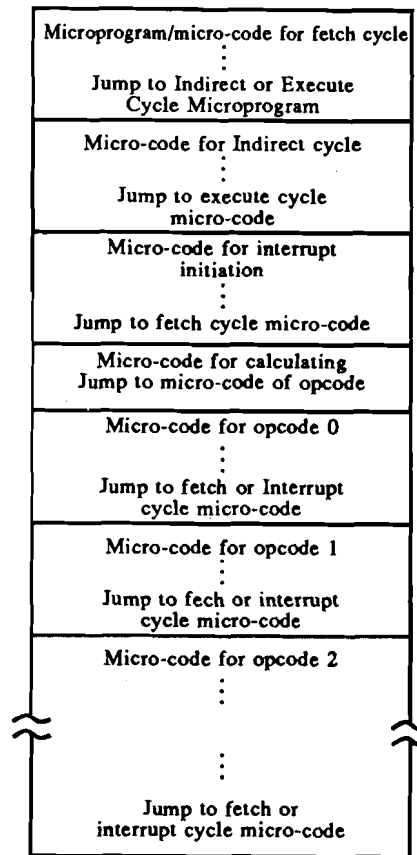


Figure 2 : Control Memory Organisation

Please note, the use of branching microinstructions in the organisation. Let us give an example of control unit organisation. Let us take a machine instruction: Branch on zero. This instruction causes a branch to a specified main memory address in case the result of last ALU operation is zero, that is, the zero flag is set. The pseudocode of the microprogram for this instruction can be:

Test "zero flag". ; If SET branch to ZERO

Unconditional branch to NON-ZERO

Zero : (Microcode which causes replacement of program counter with the address provided in the instruction)

Branch to interrupt of fetch cycle.

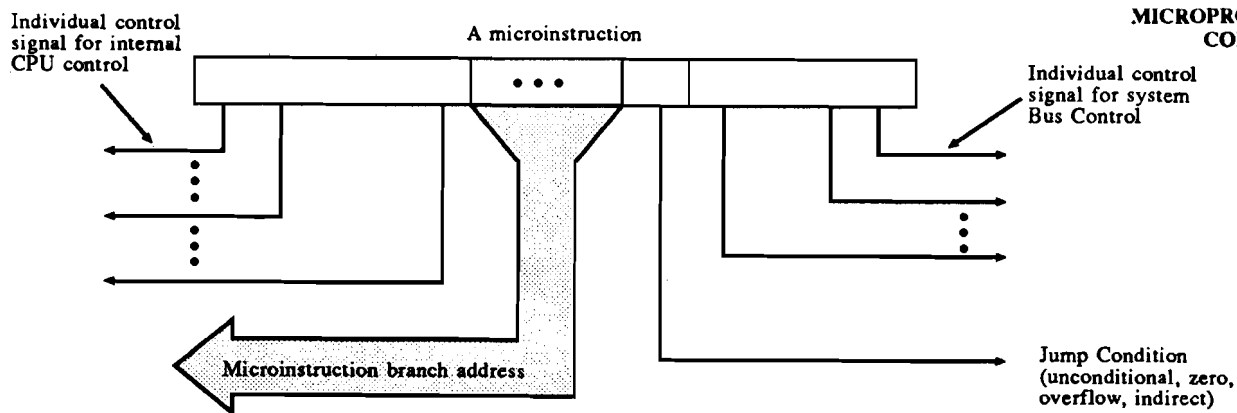
Non-zero : (Microcode which may set flags if desired indicating the branch has not taken place)

Branch to interrupt or fetch cycle.

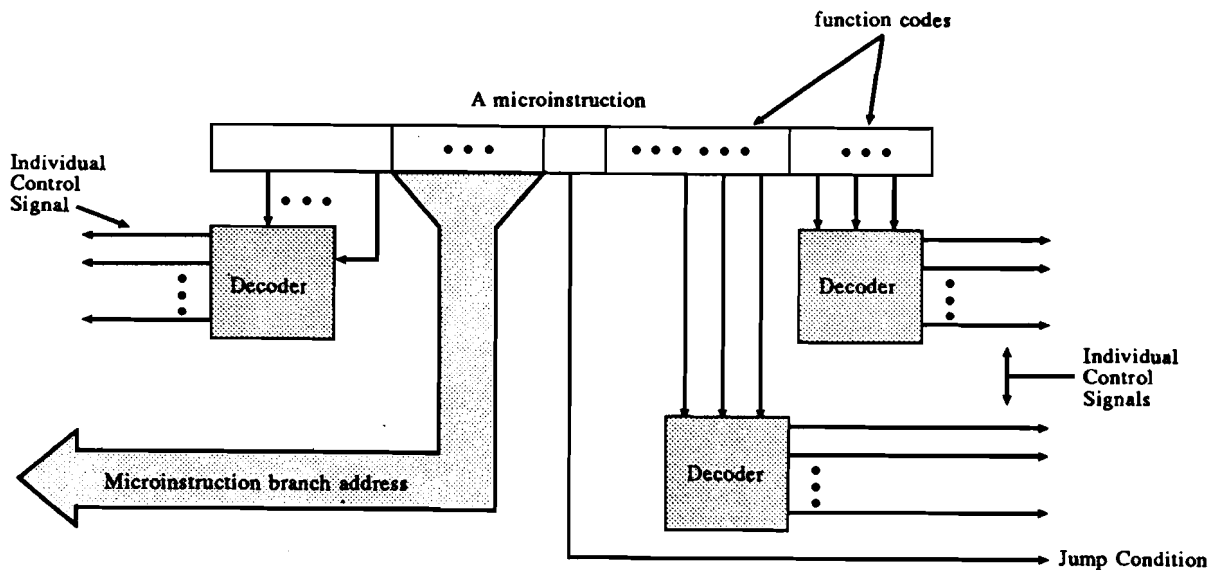
The control memory provides a concise description of the various operations of the control unit. It may also define the sequences of micro-operations which are to be performed during an instruction execution.

#### 4.4.3 Microinstruction formats

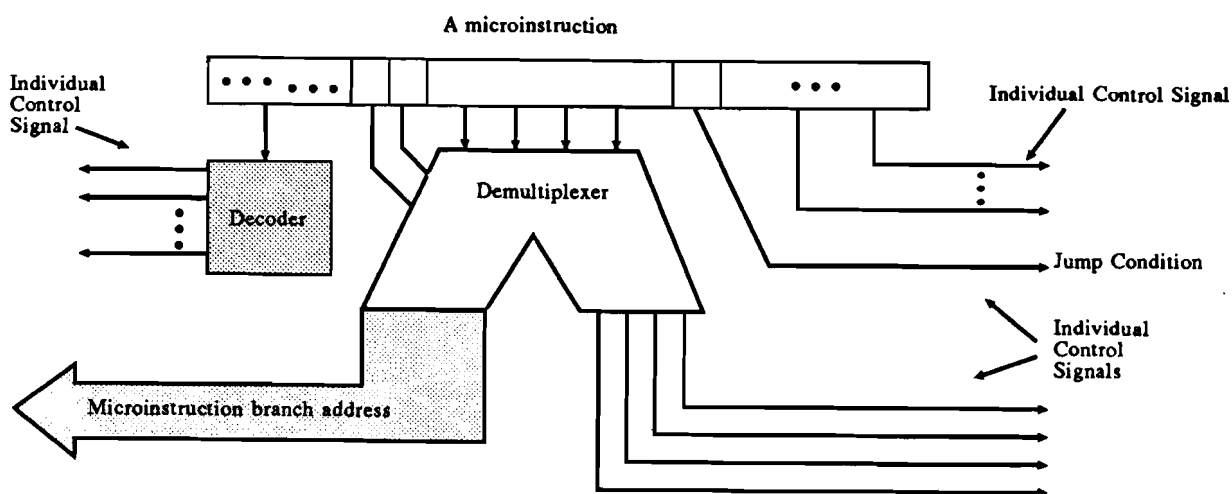
Now, let us focus on how a microinstruction may be organised. The two widely used formats used for microinstructions are horizontal and vertical. In the horizontal microinstruction each bit of the microinstruction represents a micro-order or a control signal which directly controls a single bus line or sometimes a gate in the machine. However, the length of such a microinstruction may be hundred of bits. A typical horizontal microinstruction with its related fields is shown in the Figure 3(a).



(a) Horizontal Microinstruction



(b) Vertical microinstruction using decoders



(c) Dual-use bits

Figure 3 : Microinstruction Formats

In vertical microinstructions many similar control signals can be encoded into few microinstruction bits. For example, for 16 ALU operations which may require 16 individual microcoder in horizontal microinstruction, only 4 encoded bits are needed in vertical microinstruction. Similarly, in a vertical microinstruction only 3 bits are needed to select one of the 8 registers. However, these encoded bits need to be passed from respective decoders to get the individual control signals. This is shown in figure 3(b). Some of the microinstructions may be passed through a de-multiplexer causing selected bits to be used

for few different location in the CPU (Refer to figure 3(c)). For example, a 6 bit field in a microinstruction can be used as branch address in branching microinstruction, however, these bits may be utilised for some other control signals in a non-branching microinstruction. In such a case de-multiplexer can be used. The vertical microinstructions are normally of the order of 32 bits. In certain control units, several levels of control are used. For example, a field in microinstruction or in the machine instruction may hold the address of a read only memory which holds the control signals. This secondary ROM can hold large address constants such as interrupt service routine address.

In general, a horizontal control unit are faster yet require wide instruction words, whereas, vertical control units although require decoder, however, are shorter in length. Most of the systems use neither purely horizontal nor purely vertical microinstructions.

## 4.5 A SIMPLE STRUCTURE OF CONTROL UNIT

After discussing about the microinstructions, in the previous section, let us now discuss about the structure and functioning of a simple microprogrammed control unit. This will also clarify many concepts we have discussed so far. Figure 4 shows a simple structure of such a unit.

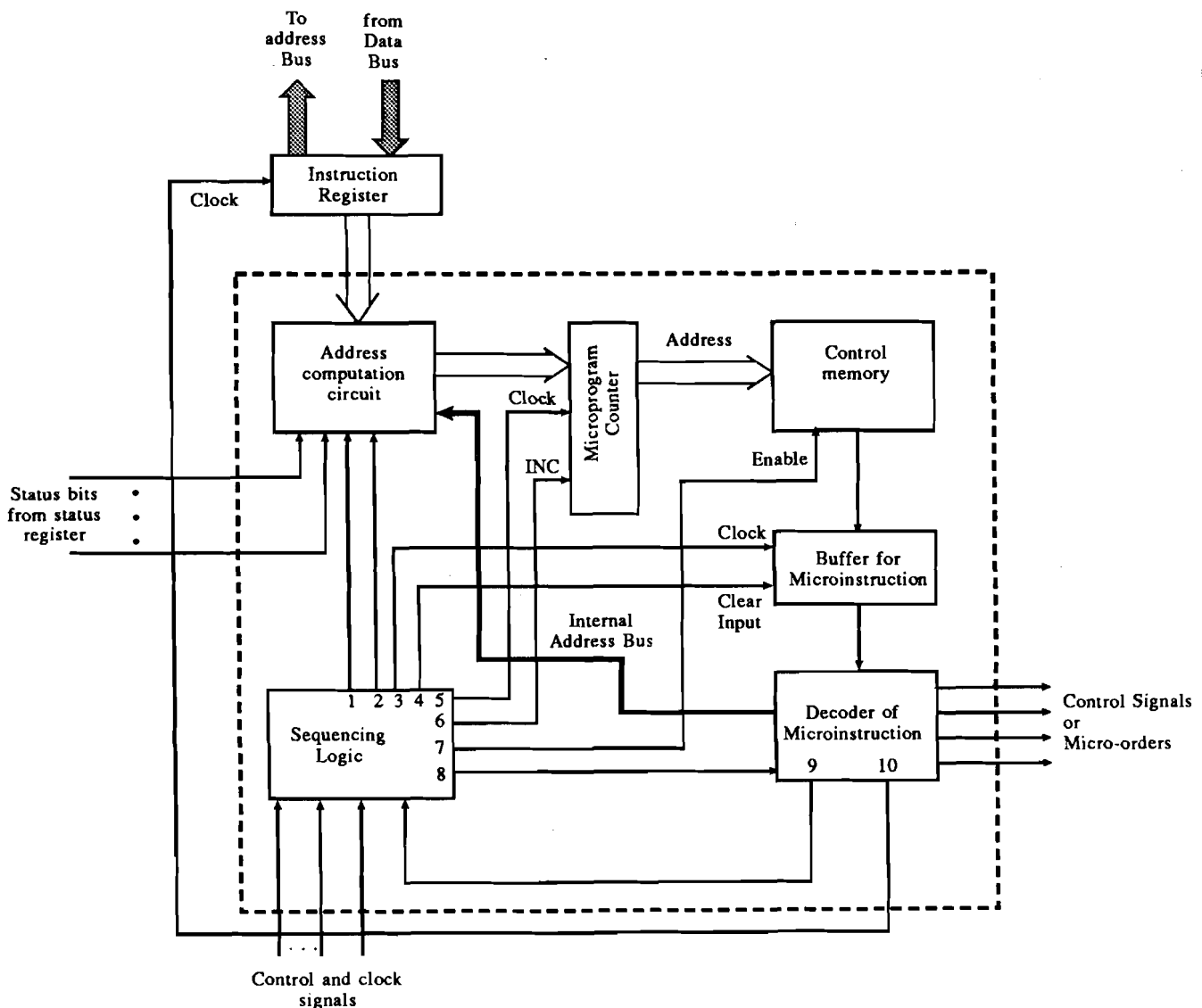


Figure 4 : A simple microprogrammed control unit

A fetched instruction is stored in the Instruction Register (IR). In certain machines IR only holds the op-code of the instruction to be executed. The control memory is normally implemented in the ROM and not in the main memory. This memory is utilised for keeping the microprograms for all op-codes. In addition, control memory may contain routines for

instruction fetch, interrupt initiation and other machine startup routines. The address of the next microinstruction is computed by the address-computation circuit. How? We will answer this question shortly. The role played by microprogram counter register is similar to that of a program counter in CPU. The microprogram counter holds the address of the next microinstruction to be executed. The microinstruction buffer holds the microinstruction which is currently being executed. The decoder of microinstruction generates micro-orders on the basis of microinstruction and op-code of the current instruction. The sequencing logic plays key role in synchronising all these component of microprogrammed control unit. The sequencing logic also plays an important role in the startup of machine. In normal circumstances, sequencing logic controls the control unit.

How does this control unit function? In this control unit address computation circuitry calculates the address of the next microinstruction to be executed. The address of the microinstruction to be executed next is stored in microprogram counter. It can acquire this address from:

- Address computation circuitry when clock signal (No. 5) is enabled.

OR

- Incrementing its own content by enabling INC signal (No. 6). The microinstruction buffer can be cleared during this time by enabling signal (No. 4).

Now, control store can be read. The enable input (signal No. 7) enables the control memory to transfer the microinstruction stored in the location addressed by the microprogram counter into the microinstruction buffer. Please note that for this transfer clock signal (No.3) should also be enabled so that the microinstruction buffer register can be loaded. This microinstruction is then decoded and respective micro-orders are generated in the microinstruction decoder on receiving a control signal (No.8).

This sequence continues till one microprogram is executed, in other words one machine instruction is executed, after this microinstruction decoder issues the micro-order which enables the clock of instruction register, control signal (No. 10). This signal causes loading of the next machine instruction to be executed into the instruction register. One of the responsibilities of control unit, once an instruction is fetched and operand addresses calculated, will be to calculate the entry point address for that op-code in the control memory, that is, the address of the first microinstruction for the microprogram for the given op-code. The responsibility of calculating the entry point address of the microprogram for a given op-code lies with the address computation circuitry. Then, the sequencing logic increments the microprogram counter to acquire the next microinstruction in sequence. After executing the complete microprogram, a branch to another microprogram routine may take place. Thus, the last instruction of a microprogram is a branching microinstruction. This branching microinstruction holds an address and control signals for address computation circuitry. On the basis of these two informations, the address computation circuitry calculates the address of the next microinstruction to be executed, in effect causing a branch.

Now, let us discuss how the branching and non-branching microinstructions can be implemented in the figure 4. The control signal (No. 9) plays a key role in implementing branching and non-branching microinstructions. This control signal indicates to the sequencing logic whether the microinstruction which has been decoded is a non-branching (value of signal No. 9 = 0) or a branching (value = 1) microinstruction. Sequencing logic then generates control signals for calculating the address of next microinstruction in the following way:

- If value of signal 9 = 0  $\Rightarrow$  a non-branching microinstruction  
Sequencing logic activates control signal 6 which increments the microprogram counter.
- Otherwise (that is value of signal 9 = 1),  $\Rightarrow$  a branching microinstruction  
Sequencing logic activates control signal 5 which causes the address calculated in the address computation circuitry to be transferred to microprogram counter.

But, how does address computation circuitry calculates the address of next microinstruction? For answering this question, let us discuss how the address computation circuitry may function.

The address computation circuitry is involved in :

- determining the entry point address of control store depending on the opcode



extracting the branch address indicated by a branching microinstruction. This address is supplied on the internal address bus.

This can be achieved by using two control signals 1 and 2 from the sequencing logic which tells the address computation circuitry to calculate the address of next microinstruction on the basis of opcode or branching microinstruction respectively. In case of conditional branch the address provided by the internal address bus may be modified depending on the status bits supplied by the ALU. Thus, unconditional and conditional jumps can be implemented on the basis of the sequencing logic signals and status bits from ALU. The address of next microinstruction is calculated by using the address obtained on the internal address bus. But, how can the address computation circuitry modify the address it gets from the internal address bus? Well, by using simple logical operations such as masking, ORing, etc.

### Check Your Progress 2

State true or false

1. A branching microinstruction can have only an unconditional jump.  
True ☐ False ☐
2. Control store stores microprograms only for opcodes.  
True ☐ False ☐
3. A true horizontal microinstruction require one bit for every control signal.  
True ☐ False ☐
4. A decoder is needed to find a branch address in the vertical microinstruction.  
True ☐ False ☐
5. The address computation circuit in figure 4, is used for finding the next sequential address of a microinstruction.  
True ☐ False ☐
6. One of the responsibility of sequencing logic (Refer figure 4) is to cause reading of microinstruction addressed by microprogram counter into the microinstruction buffer.  
True ☐ False ☐
7. Status bits supplied from ALU to sequencing logic have no role to play with the sequencing of microinstruction  
True ☐ False ☐

## 4.6 MICROINSTRUCTION SEQUENCING

As discussed earlier, the two basic functions of control unit are microinstruction sequencing and microinstruction execution. We have explained some of the concepts in this regard. Let us discuss about these aspects in greater details in this and subsequent sections. Let us first try to find out the concerns for designing sequencing technique.

The first concern which is applicable in general is "to minimize the size of control memory". The second concern is "to execute a microinstruction as fast as possible". This implies that the address of next microinstruction should be calculated at a fast rate.

Now, let us find out how these two concerns can be achieved. The factors responsible for reducing the size of control memory depend on the length of a microinstruction. The length of microinstruction is greatly influenced by the following three major factors:

- Degree of parallelism which is needed at the microoperation level or in other words "the number of microoperations which can be executed simultaneously".
- Representation/encoding of control information.
- The means of specifying the address of next microinstruction.

The number of microoperations which can be executed simultaneously in a processor may

vary from one to hundred. This degree of parallelism is frequently used for characterising the microprogrammable processors. A highly encoded instruction also tend to be short. We will discuss more about these while we discuss about microinstruction execution.

Let us focus in this section about calculation of the address of next microinstruction. In general, the address of next microinstruction is:

- The address of next microinstruction in sequence
- Calculated on the basis of op-code
- Branch address (conditional or unconditional)

Address is calculated only once from the op-code in one instruction cycle. The machine sequences are not long and branches are common after three or four sequences. Thus, by making branching algorithm better we can make a microinstruction addressing more time efficient. In general, three techniques based on number of addresses have been utilised for sequencing. These are:

- Two address fields in each microinstruction
- Single address field and
- Variable format microinstructions

In the two address fields microinstructions, either of the two addresses or the address generated with the help of op-code is selected using a branch logic which is based on the flags and control signal. In such a case, branching to a desired address can be made very easily. However, in this approach a lot of control memory is wasted as at least one of the address may not be needed in several microinstructions.

With some modified circuitry and added logic we can reduce the number of addresses to one. Here, a new register called microprogram counter, as introduced in Figure 4 can be used. In this case, the next microinstruction address can be the address of :

the next sequential address  
OR  
the address generated using op-code  
OR  
the address stored in the address field of the microinstruction.

The address selection signal which will be based on branch logic can indicate which of the above address is to be selected. Although this scheme saves some space, yet the space provided for even one address is not used very often. Thus, there remains some inefficiency in the coding scheme of the microinstruction. However, this is a commonly used approach.

Another approach which can be used is to provide variable format. In such a case two formats are used. The first format provides the control microinstruction, while the second format provide the branch logic and address. In such a scheme one bit is needed in the microinstruction to indicate whether this is a control microinstruction or a branching microinstruction. In case, the microinstruction contains control signals, then the next microinstruction address is calculated either by using the op-code of instruction register or it is the address of the next microinstruction in sequence. In this approach, an extra cycle is needed for branch microinstruction which is undesirable. You can get more details on sequencing technique in the further readings.

#### Generation of Address:

As discussed earlier, there are three ways of generating the address of next microinstruction to be executed. This address generation once again depends on the number of address fields used in a microinstruction. If we use two address fields then all the addresses are explicit. However, in case of one field and variable format microinstruction, the conditional branch address is calculated on the basis of information like:

- ALU flags
- Address mode indicators
- Sign bits or portion of selected registers
- Status bits of the control unit itself

However, for unconditional branch, address generation is direct. Opcode for each instruction is to be interpreted once per instruction cycle. One of the technique which can be used for opcode interpretation is mapping of opcode into a microinstruction address in the control store. Other techniques in this respect are addition of two portions of address and residual control. In the addition approach, one part of the address which is fixed is added to a variable part to form a complete address. The residual control involves use of an address of microinstruction which has been saved previously at a temporary storage location inside the control unit. More details on these approaches can be obtained from the further readings.

## 4.7 MICROINSTRUCTION EXECUTION

The microinstruction cycle can consist of two basic cycles: the fetch and the execute. Here, in the fetch cycle the address of the microinstruction is generated and this microinstruction is put in a microinstruction register for execution. We have already dealt with this part in the previous sections. The execution of microinstruction simply means generation of control signals. These control signals may drive the CPU (internal control signals) or the system bus. The format of microinstruction and its contents determine the complexity of a logic module which executes a microinstruction.

One of the key features which is incorporated in microinstruction is the encoding of microinstructions. What is encoding of microinstruction? For answering it, let us recall the Wilkes control unit. In Wilkes control unit, each bit of information either generates a control signal or form a bit of next instruction address. Now, let us assume that a machine needs  $N$  total number of control signals. If we follow the Wilkes scheme we require  $N$  bits, one for each control signal in the control unit. Since we are dealing with binary control signals, therefore, a  $N$  bit microinstruction can represent  $2^N$  combinations of control signals.

Do we need all these  $2^N$  combinations?

No, some of these  $2^N$  combinations are not used because:

- (a) Two sources may be connected by respective control signals to a single destination, however, only one of these sources can be used at a time. Thus, the combinations where both these control signals are active for same destination are redundant.
- (b) A register cannot act as source and destination at the same time. Thus, such a combination of control signals is redundant.
- (c) We can provide only one pattern of control signals at a time to ALU, making some of the combinations redundant.
- (d) We can provide only one pattern of control signals at a time to the external control bus also.

Therefore, we do not need all these  $2^N$  combinations. Suppose, we only need  $2^K$  (which is less than  $2^N$ ) combinations then we need only  $K$  encoded bits instead of  $N$  control signals. The  $K$  bit microinstruction is an extreme encoded microinstruction. Let us touch upon the characteristics of the extreme encoded and unencoded microinstructions:

### Unencoded Microinstructions

- One bit is needed for each control signal, therefore, number of bits required in a microinstruction are high.
- It presents a detailed hardware view as control signal need can be determined.
- Since each of the control signal can be controlled individually, therefore, these microinstructions are difficult to program. However, concurrency can be exploited easily.
- Almost no control logic is needed to decode the instruction as there is one to one mapping of control signal to a bit of microinstruction. Thus, execution of microinstruction and hence the microprogram is faster.
- The unencoded microinstruction aims at optimising the performance of a machine.

### Highly Encoded Microinstructions

- The encoded bits needed in microinstructions are small.

- It provided an aggregated view that is a higher view of the CPU as only an encoded sequence can be used for microprogramming.
- The encoding helps in reduction in programming burden, however, the concurrency may not be exploited to the fullest.
- A complex control logic is needed as decoding is a must. Thus, the execution of microinstruction have propagation delay through gates. Therefore, execution of microprogram takes longer time than that of unencoded microinstruction.
- The highly encoded microinstructions are aimed at optimizing programming effort.

In most of the cases, the design is kept between the two extremes. The LSI 11 (highly encoded) and IBM 3033 (unencoded) control units are close examples of these two approaches. You can find the details about these two in the further readings.

In general, in many of the microinstruction designs, more bits are used than absolutely necessary. In this respect, many terms based on different design characteristics were coined. The terms like horizontal or vertical microinstructions which were introduced earlier, can also give information about the length of microinstruction. Typically, horizontal microinstructions are 40 to 100 bits whereas, vertical microinstructions are 16 to 40 bits..

### Encoding of Microinstruction

As mentioned earlier, that microprogrammed control unit designs are neither completely unencoded nor highly encoded. They are slightly coded, in general, to reduce the width of control memory and microprogramming efforts. For encoding, a microinstruction is divided into a set of fields such that:

- A specific control signal can be activated by only one field, thus, making the fields independent of each other.
- Each field represent a pattern of control signals which in turn depicts an action. As the fields are independent, therefore, the actions depicted by different fields can be performed simultaneously, that is, in parallel.
- A specific field specifies actions which are mutually exclusive, that is, only one of these actions can occur at a time.

But, how do we classify control signals in different fields? In this respect two approaches have been identified. When fields are designated by identifying various functions, then we call it functional encoding. For example, a function "transfer of data to accumulator from different sources", if designated by a single field will be encoded such that each code specifies a different source.

However, if we view the machine as a set of independent sources, then we can devote one field for one resource such as Input/Output module, memory, ALU etc. This is termed as resource encoding.

Finally, let us look at one of the responsibility of the control unit which has not been discussed till now, in the next section.

---

## 4.8 MACHINE STARTUP

---

The control unit is also responsible for initialising various registers during the startup of the machine. The control unit loads a hardware generated address in the program counter (PC) and starts execution of the instruction stored in that location. This hardware generated address may be the *address of the first instruction to be executed* (Refer figure 5(a) ). This *address* of the first instruction (The first instruction is executed on machine startup) is known as the *reset vector*. In some machines the hardware generated address points to the reset vector in the memory. This is the case for figure 5(b).

In the first case, the control unit starts its normal operation immediately after loading the reset vector in the program counter. However, in the reset vector address machines, the control unit must first fetch the reset vector into the program counter before beginning its normal operation. What is the need of introducing this extra complexity? The advantage here is that the reset vector address machines allow generality of design to the machine architects, that is, the machine architects will not have to assign the starting address of the startup program to be executed in the memory. They just need to assign the address of the

reset vector in the memory, thus, different reset vector values can be chosen on the same machine allowing more flexibility. For example, if we have two different startup routines for two different operating systems then by changing the reset vector value in the memory we can change the loading of the operating system. Both of these operating systems, however, can reside on the main memory at the same time.

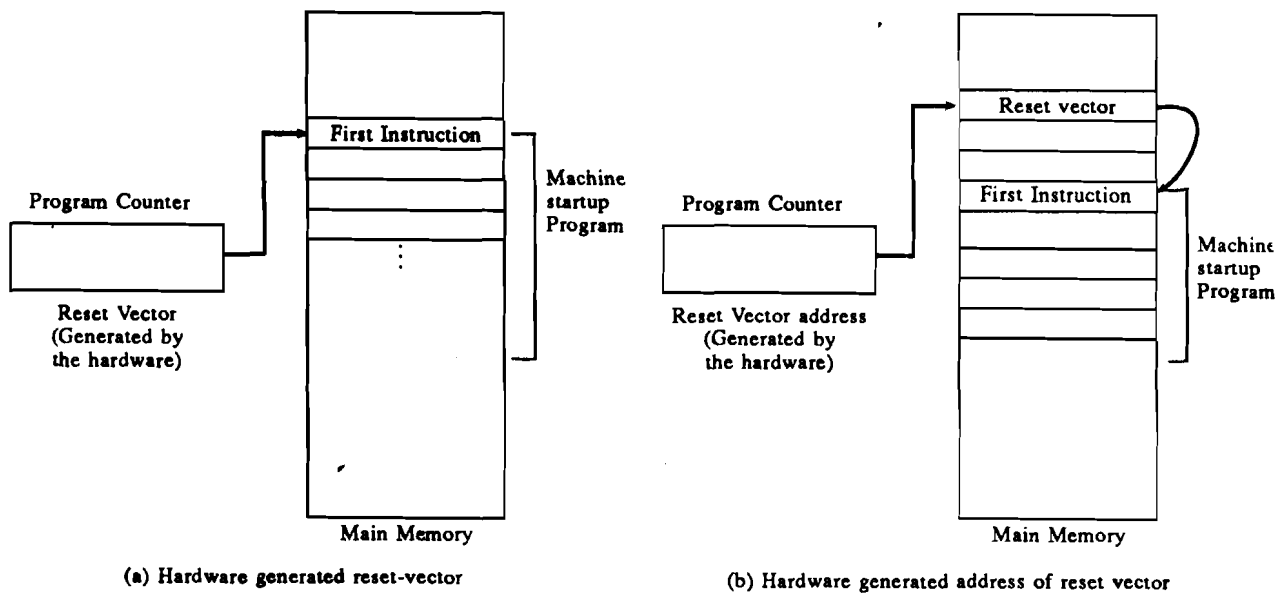


Figure 5 : Loading of machine startup program

### Check Your Progress 3

1. What are the possibilities for next instruction address?

.....

.....

.....

2. Compare two address field microinstruction with one address field microinstruction. Which of them is more commonly used?

.....

.....

.....

3. How many address fields are their in Wilkes Control Unit?

.....

.....

.....

.....

4. Compare and contrast unencoded and highly encoded microinstructions.

.....

.....

.....

.....

.....

.....

.....

## 4.9 SUMMARY

In this unit, we have discussed about the micro-programmed control unit. The key to such a unit is a microinstruction. A microinstruction has been defined in the unit. In addition we have also explained a basic structure of microprogrammed control unit. The two basic functions of microprogrammed control unit: microinstruction sequencing and microinstruction execution has also been discussed in the section. Detailed examples of microprogrammed control unit have not been included in this unit. You can refer to further readings for these details.

With this unit, we come to the end of block 2 on CPU organisation. Although we have tried to discuss the aspects related to the CPU in quite a detail, however, you must refer to the further readings for more details and examples.

## 4.10 MODEL ANSWERS

### Check Your Progress 1

1. Firmware is basically microprograms which are used in microprogrammed control unit. Firmware are difficult to write than software.
2. (a) False      (b) False      (c) True      (d) False      (e) False
3. 010,    011,    110,    111

### Check Your Progress 2

1. False      2. False      3. True      4. False      5. False      6. True      7. False

### Check Your Progress 3

1. The address of next microinstruction can be one of the following :
  - the address of next microinstruction in sequence
  - determined by op-code using mapping or any other method
  - branch address supplied on the internal address bus
2. 

<i>Two address field microinstructions</i>	<i>One address field microinstructions</i>
Microprogram counter is not needed.	Microprogram counter is needed.
Wastes lot of control memory space.	Smaller in comparison to two address.
Very simple to implement.	Slightly more complex to implement than two address.

The one-address microinstructions are most commonly used. Have you noticed that we have used one address microinstruction formats while explaining a simple structure?

3. Wilkes control typically have one address field, however, for a conditional branching microinstruction, it contains two addresses. The Wilkes control, in fact, is a hardware representation of microprogrammed control unit.
4. Refer to section 4.7 and do it yourself.
5. A reset vector is an address of the first instruction to be executed on machine start-up.