CONTROL UNIT DESIGN

User Inaccessible (Invisible) Registers:

I. **PC (Program Counter):**

It contains the address of the next instruction to be fetched.
After every instruction is fetched, value of PC is incremented by the processor.
The programmer cannot change PC directly.
PC gets a new value when a branch instruction is encountered.

II. **Instruction Register:**

It is used to temporarily hold the newly fetched instruction for decoding.

III. **MAR (Memory Address Register):**

It contains the address of the current memory transfer.

IV. **MBR (Memory Buffer Register):** Also called as memory Data R

It contains the data that is currently transferred.

MICRO-OPERATIONS

Every instruction is further divided into a set of very basic Micro-operations. A Micro-operation takes one clock pulse i.e. 1 T-state duration.

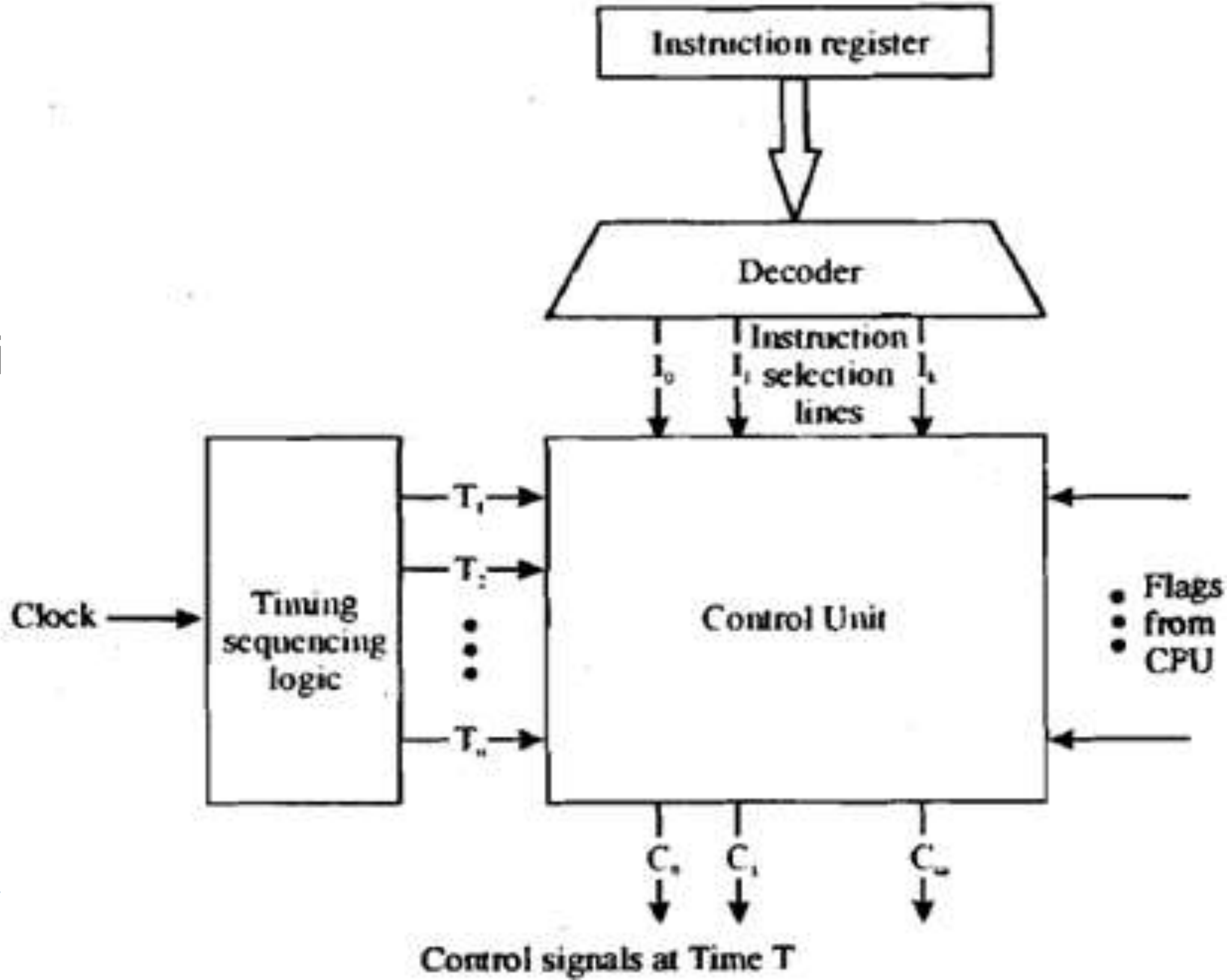


Fig. Control uni organization

- In the hardwired organization control unit is designed as a combinational circuit. The control unit is applied by gates, flip-flops, decoder and other digital circuits.

- Hardwired control units can be optimised for fast operations. Block diagram of control unit is displayed in Figure below. Major inputs to circuit are instruction register, clock, and flags.

- Control unit uses the opcode of instruction stored in IR register to perform various actions for various instructions.

- The Control unit logic has unique logic input for every opcode. It simplifies the control logic. This control line selection can be executed by a decoder.

- A decoder will have n binary inputs and 2^n binary outputs. Every one of these 2^n different input patterns will trigger a single unique output line.

- Clock part of the control unit issues a repetitive sequence of pulses of single state duration for each micro-operation(s).

- The required control signals are determined by

1) content of control step counter or time generation

2) content of instruction register

3)Content of condition code flags

Micro-operations for a simple Fetch Cycle:

```
T1:    MAR ← [PC]        ; load the address of the instruction to be fetched
T2:    MBR ← Memory      ; get the instruction from the memory
T3:    IR ← [MBR]        ; load the instruction into the instruction register
       PC ← PC + I       ; increment PC to point to the next instruction
                           Here I = Length of the current instruction
```

Notice that the last two operations took place in the same cycle. This is possible because **when two or more Micro-operations are independent of each other, they can be performed simultaneously, in one clock cycle.**

Now we will see Micro-operations for entire instructions:

1) Give the Micro-instructions for: **ADD A, B** {Dec 99}

Soln:

$$\left.\begin{array}{llll} \text{T1:} & \text{MAR} & \leftarrow & [\text{PC}] \\ \text{T2:} & \text{MBR} & \leftarrow & \text{Memory} \\ \text{T3:} & \text{IR} & \leftarrow & [\text{MBR}] \\ & \text{PC} & \leftarrow & \text{PC} + \text{I} \end{array}\right\} \textbf{Fetching}$$

$$\left.\begin{array}{llll} \textbf{T4:} & \textbf{A} & \leftarrow & \textbf{A + B} \end{array}\right\} \textbf{Execution}$$

Note that the two operands to be added were **registers already present in the processor.** Hence **no additional steps** were required to fetch them. If an operand is from the memory, then additional steps will be required to fetch it.

Such instructions are used in **Intel 8085.**

CONTROL UNIT

- All the **Micro-operations** are **performed** by the **Control Unit.**
- The **main task** of Control Unit is to **generate internal control signals** to all internal registers and also to enable activities on the internal bus of the processor.
- If these control signals are generated using a **combinational logic circuit** then it is called a **Hardwired** Control Unit.
- If these control signals are generated by **software** then it is called a **Microprogrammed** Control Unit.

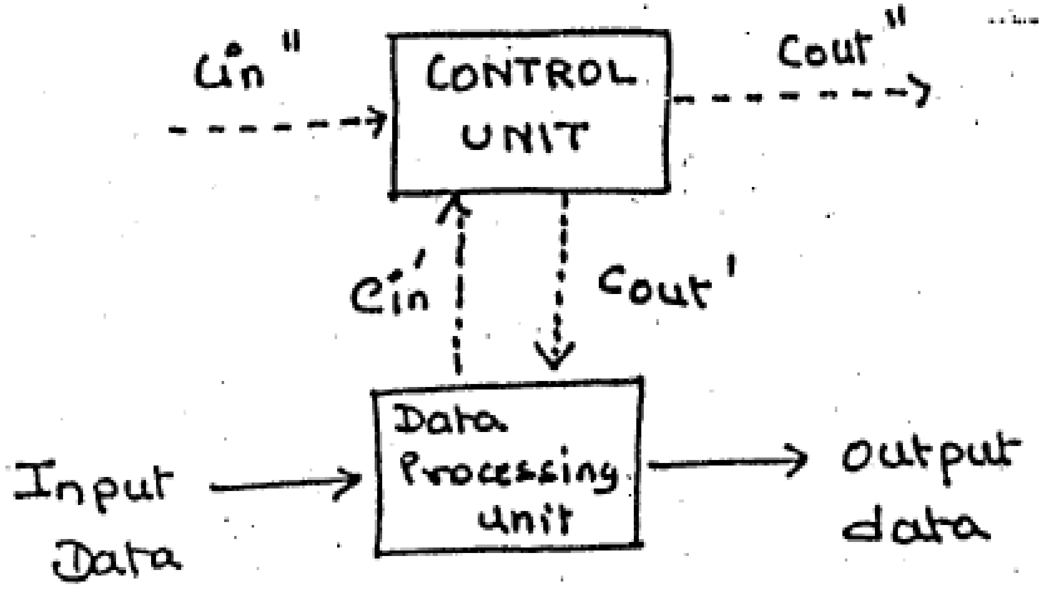
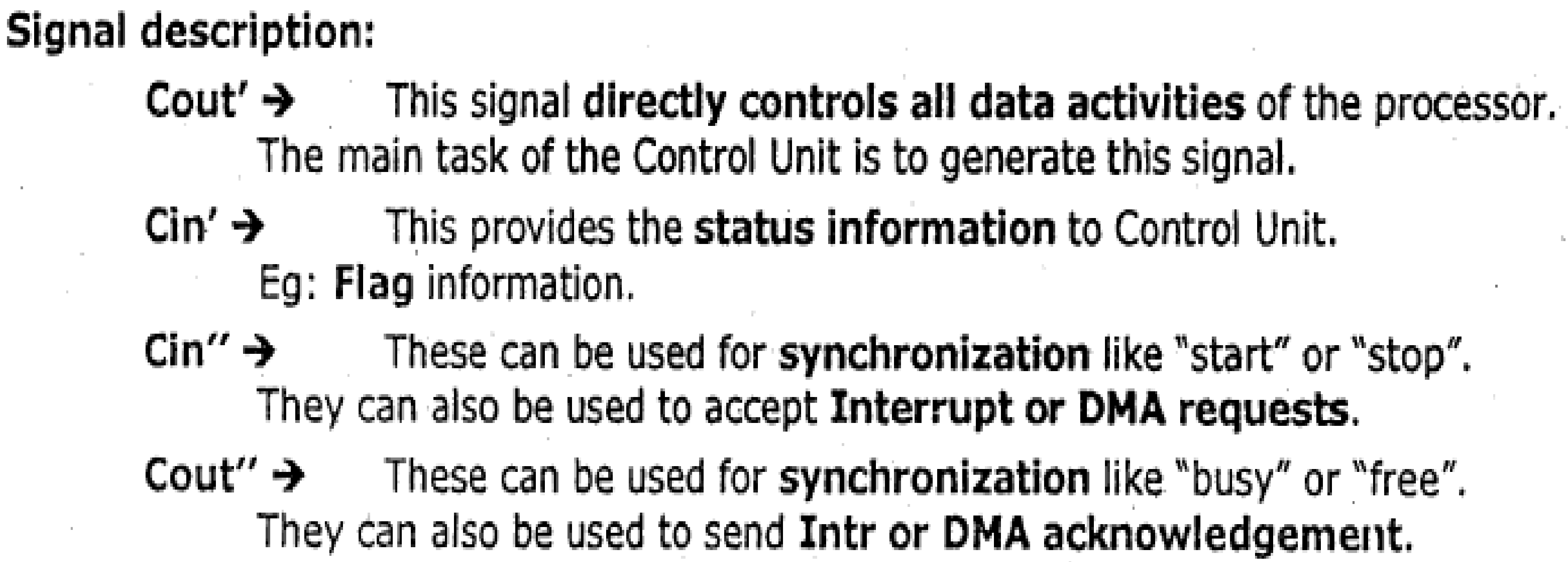


Fig. A control unit and its input-output lines

The four groups of control signals distinguished in fig. have following functions



Signal description:

Cout' → This signal **directly controls all data activities** of the processor. The main task of the Control Unit is to generate this signal.

Cin' → This provides the **status information** to Control Unit. Eg: **Flag** information.

Cin'' → These can be used for **synchronization** like "start" or "stop". They can also be used to accept **Interrupt or DMA requests**.

Cout'' → These can be used for **synchronization** like "busy" or "free". They can also be used to send **Intr or DMA acknowledgement.**

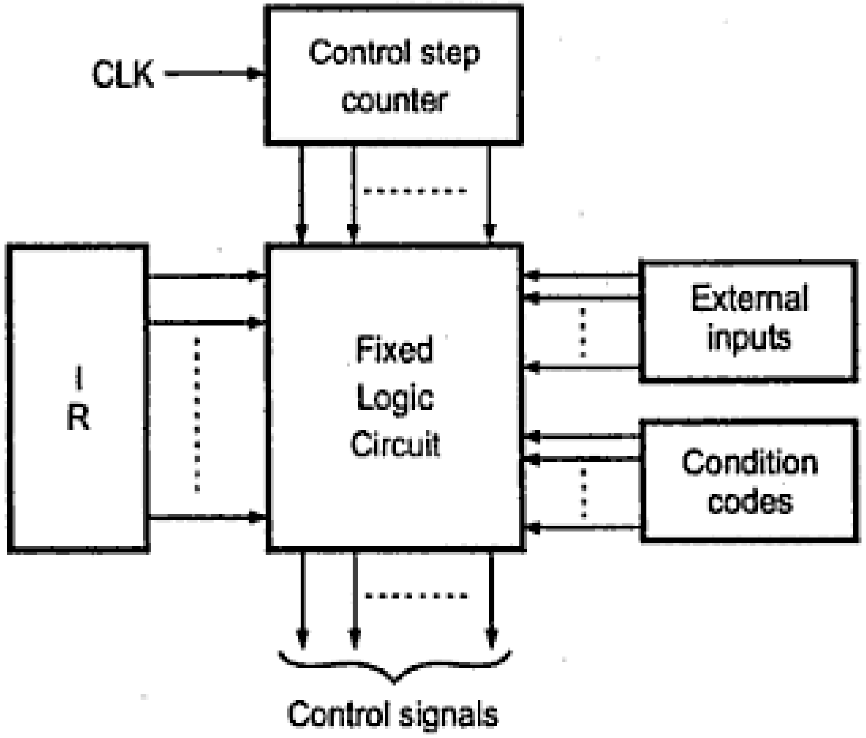
Cin' – to indicate occurrences of unusual conditions such as errors.

Cin" , Cout"– these signals received from other control units eg. Supervisory controller. Both are primarily used to synchronize the control unit with the operation of other units.

Hardwired Control Unit

In the hardwired control, the control units use fixed logic circuits to interpret instructions and generate control signals from them. Fig. 3.10 shows the typical hardwired control unit. Here, the fixed logic circuit block includes combinational

detail block diagram for hardwired control unit as shown in the Fig. 3.11.

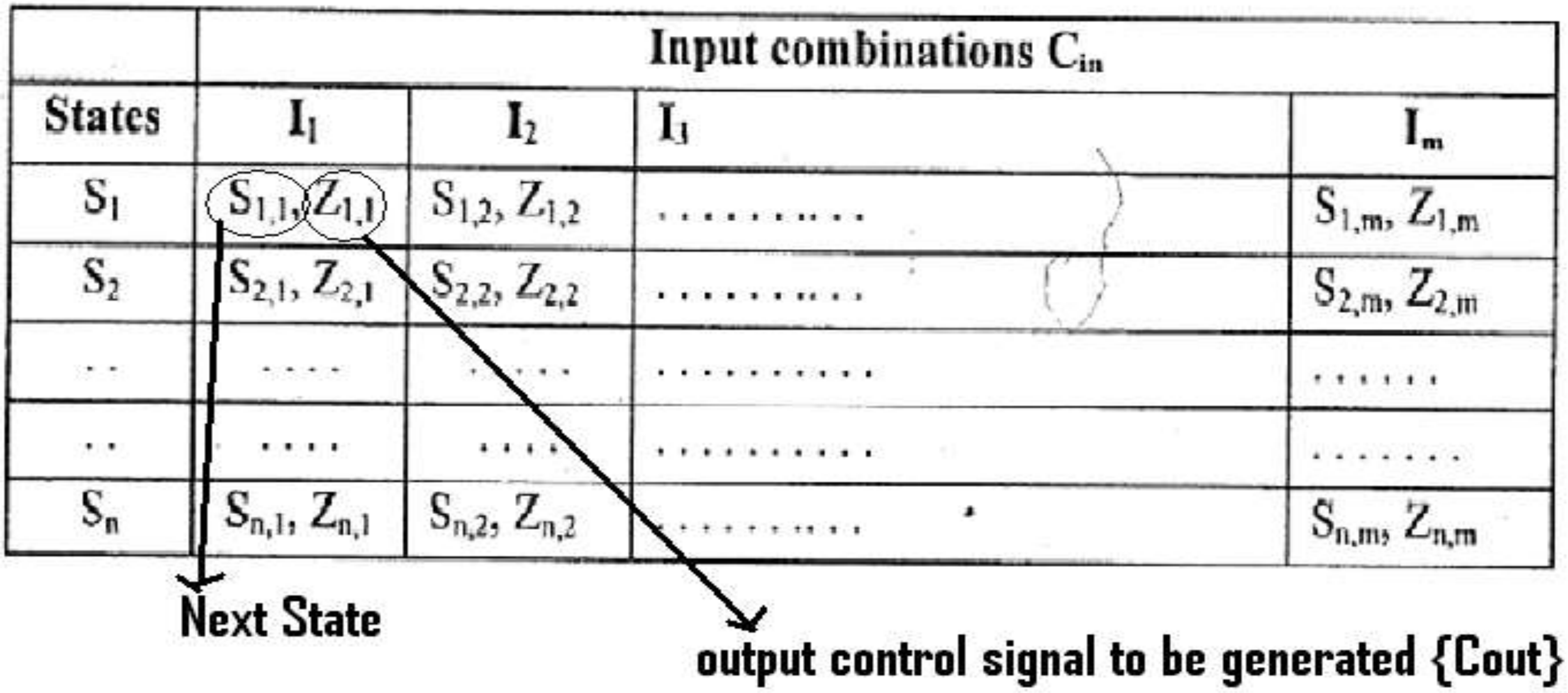


There are three types of Hardwired Control designs:

- State table method
- Delay element method
- ***Sequence counter method***

1) State Table Method

The Behavior of control unit is designed in the form of state table as shown in fig..



	Input combinations C_{in}			
States	I_1	I_2	I_3	I_m
S_1	$S_{1,1}, Z_{1,1}$	$S_{1,2}, Z_{1,2}$	…………	$S_{1,m}, Z_{1,m}$
S_2	$S_{2,1}, Z_{2,1}$	$S_{2,2}, Z_{2,2}$	…………	$S_{2,m}, Z_{2,m}$
..	….	……	…………	……
..	….	….	…………	……
S_n	$S_{n,1}, Z_{n,1}$	$S_{n,2}, Z_{n,2}$	…………	$S_{n,m}, Z_{n,m}$

Next State

output control signal to be generated {Cout}

- The table shows what corresponding output should be generated when different inputs are applied at various states.
- Let $\mathbf{C_{in}}$ and $\mathbf{C_{out}}$ denote the input and output variables of the Control Unit.
- **Each row** in the state table corresponds to an **internal state {S_i}** of the Control Unit. A **state** is determined by the **information stored** in the processor at that unit of **time.**
- Each **column** denotes the different set of external **input signals applied** to the control unit {C_{in}}.
- The **intersection** of the row S_i and column I_j means the following:
  - **When Input I_j is applied to state S_i, we get $S_{i,j}$, $Z_{i,j}$.**
  - $Z_{i,j}$ is the set of **output** signals to be activated.
  - $S_{i,j}$ should become the **next state** of the control unit
- A circuit is then constructed based on this table.

Advantage:

- This is the **simplest method** to implement a Hardwired Control Unit.

Disadvantage:

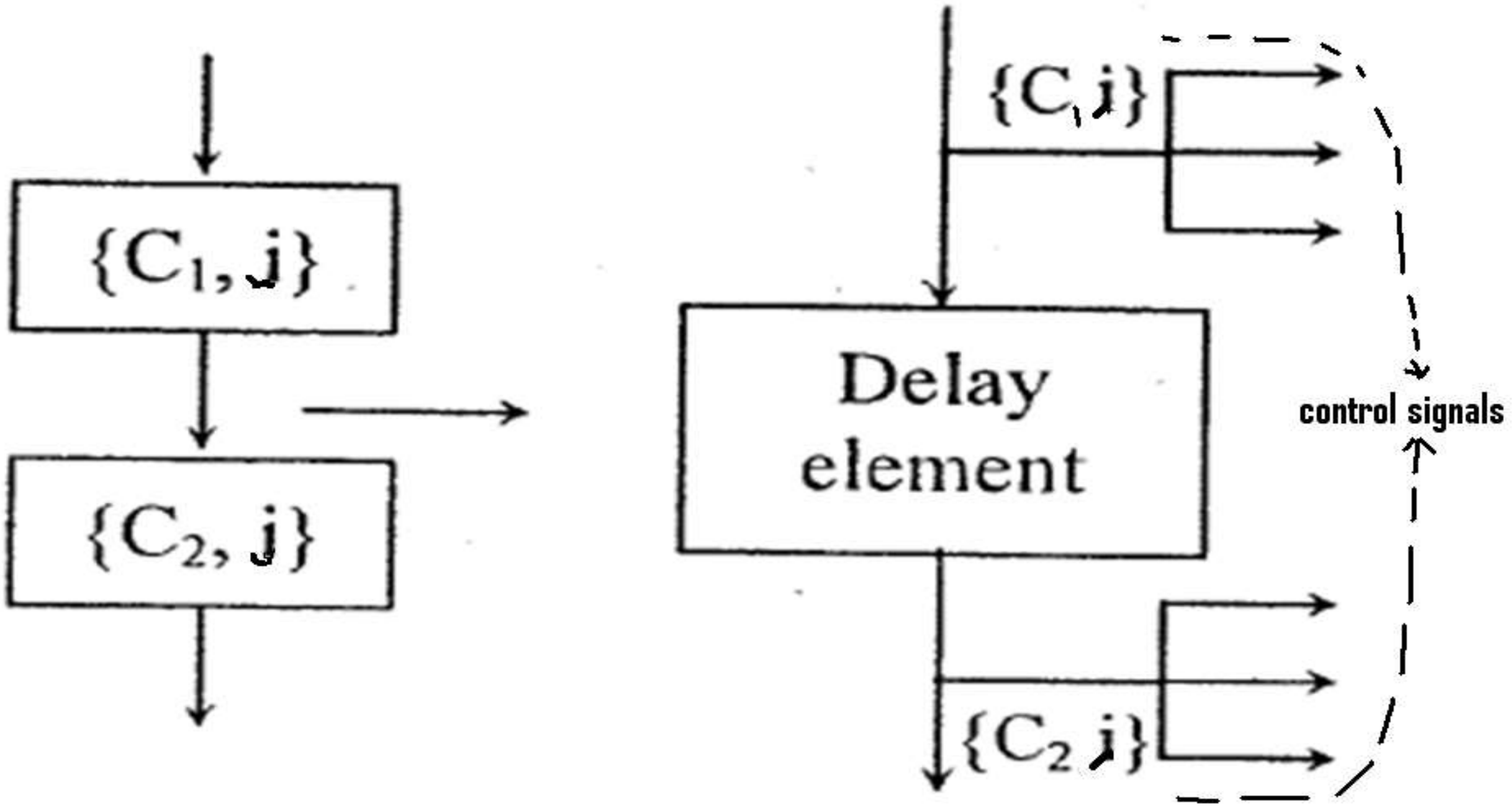
- **If** the processor has a **large number** of **states and input** combinations, then this method cannot be used as the **size of the table** will become **too large** and the **circuit** will become **very difficult** to implement.
- A state table tends to **hide useful information** like the existence of loops and repeated patterns.
  Even if two **different states** have the **same behavior**, we will **require separate hardware** for both as this **information is hidden.**
- **Circuits** designed using this method tends to have a **random structure** and are thus **difficult** to **debug** and **maintain.**

2) **Delay Element Method**

- Here the **behavior** of a Control Unit is represented in the form of a **flowchart.**
- **Every step** of the flowchart at time t_i: will activate **$\{C_{i,j}\}$,**
  where **C_i** is the **control signal at time ti**, for the execution of the instruction j.
  i.e: at t_1: Activate $\{C_{1,j}\}$
  at t_2: Activate $\{C_{2,j}\}$
  .
  .
  .
  at t_n: Activate $\{C_{n,j}\}$
- Once the flowchart is complete, then individual circuits for each $\{C_{i,j}\}$ are formed.
- It is obvious that the instruction j will be executed when all the steps of $\{C_{i,j}\}$ are performed from $C_{1,j}$, $C_{2,j}$, $C_{3,j}$, ... $C_{n,j}$.
- *But, all these steps should not be performed together; instead there should be a* ***finite time gap (delay), between every two steps.***
- The delay in the circuit is introduced by a **"D" flip-flop.**
  Delay time = t_2-t_1 = t_3-t_2 = **one clock pulse.**
- Thus, one after the other, all steps are performed, in the order of the flowchart.

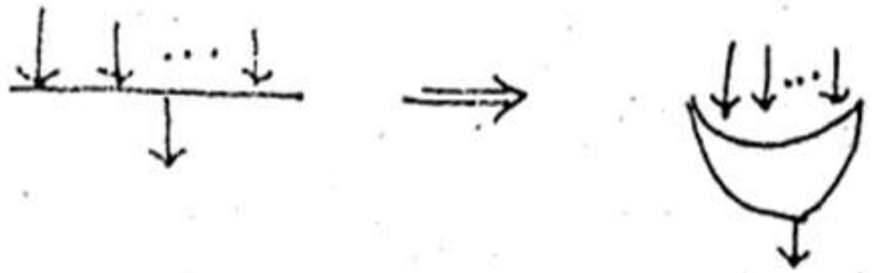
The different parts of a flowchart are handled as follows:

1) Between two successive steps simply a D Flip-Flop is inserted.



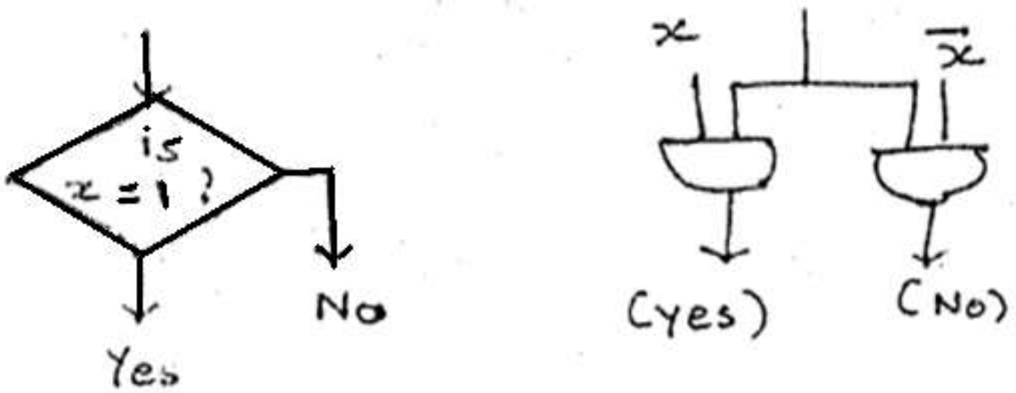
) **Multiple inputs are combined by an OR gate.**

This is because, if we arrive at the gate from any of the inputs, we should proceed ahead. An OR gate will produce a 1 when any of the inputs is 1.



)

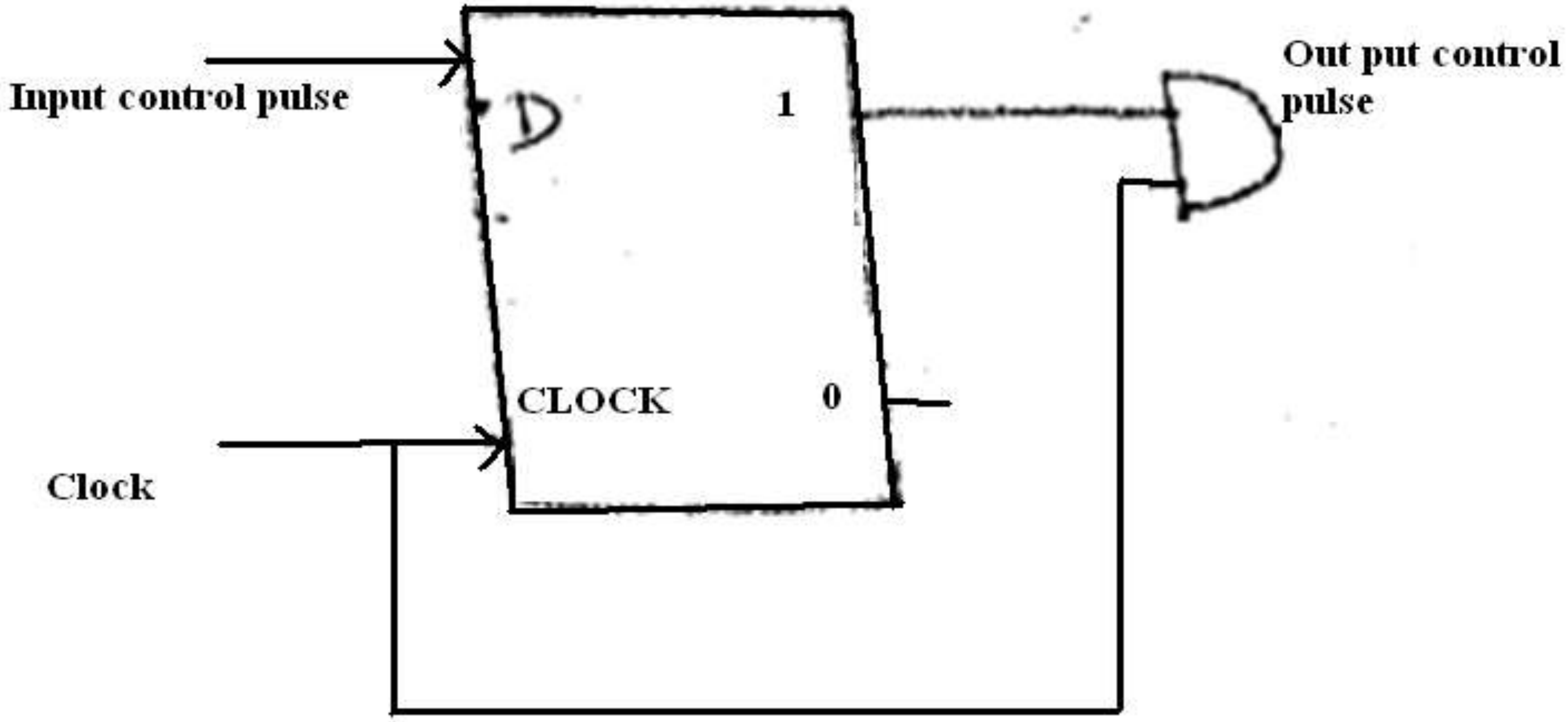
A Conditional branch (**decision box**) of the flowchart is implemented by a **pair of AND gates** as shown.



If **X = 1**: the gate on the **left** will be **active**, but
If **X = 0**: the gate on the **right** will be **active.**

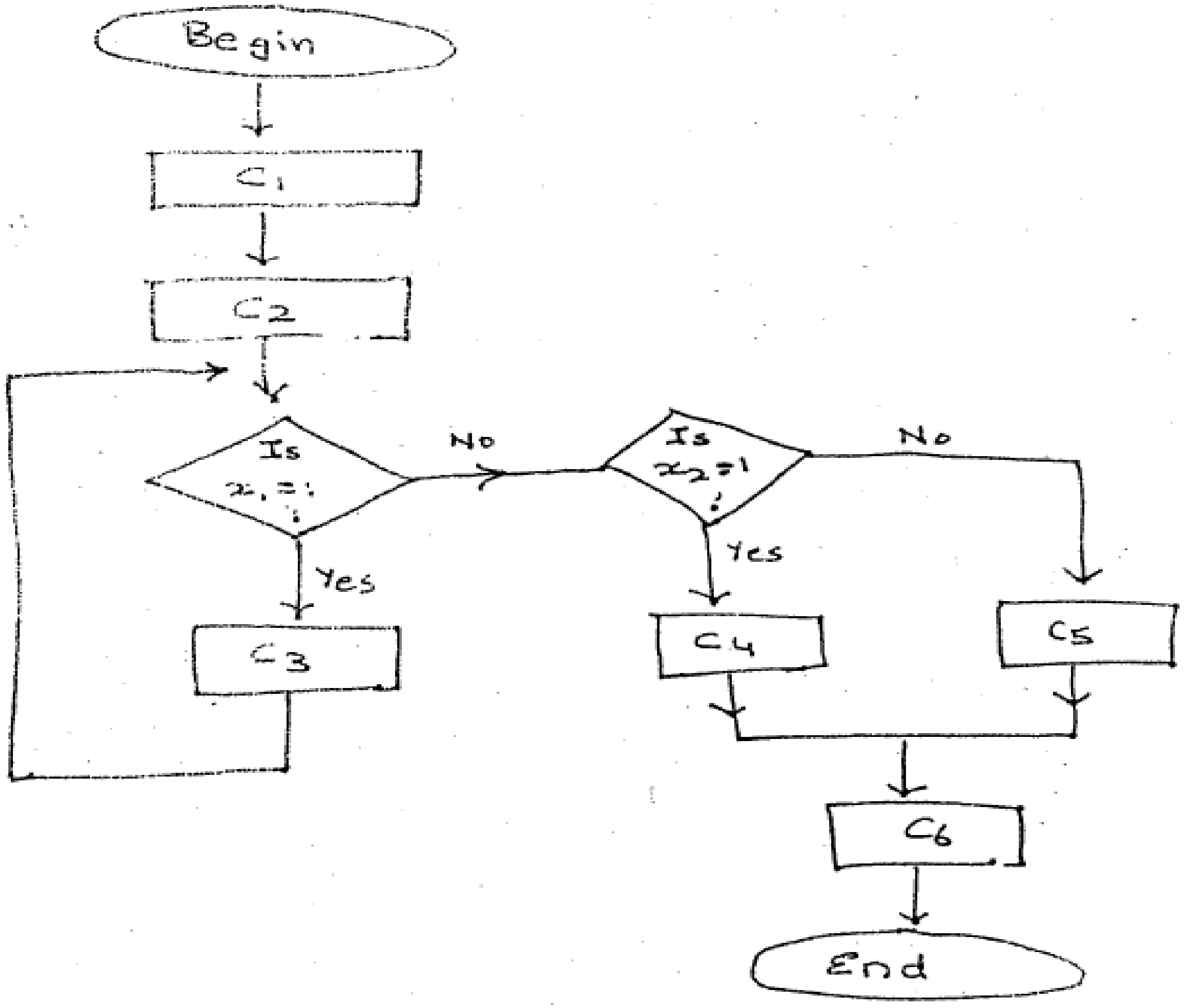
Delay element:

- A simple delay element for synchronous control circuits

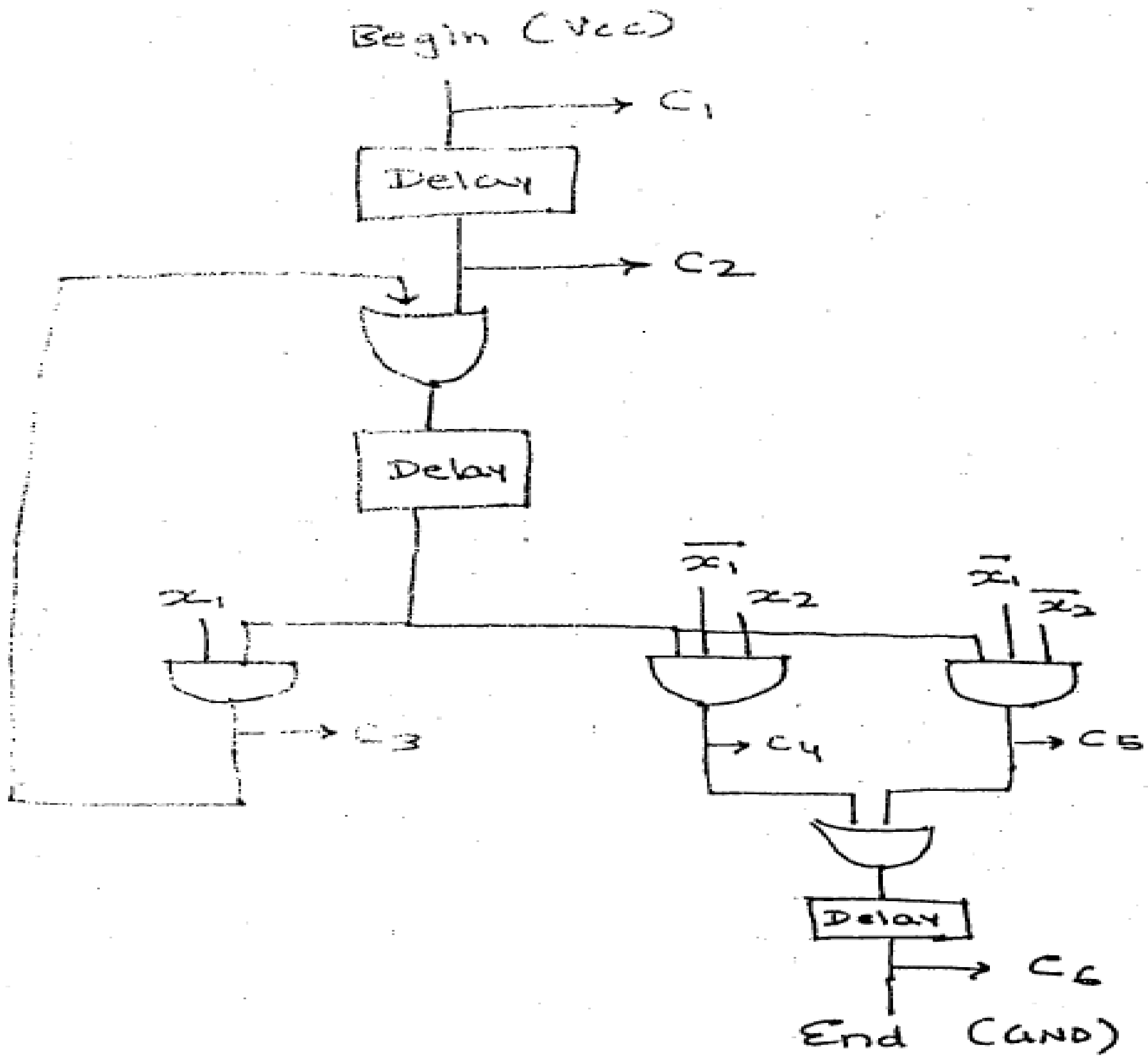


Example:

Consider the following flowchart…



The circuit based on the above flowchart using the Delay element method is as shown:



Advantage:

- In case of **looping programs**, the **same hardware can be used in a loop** as shown i the above example. This is a clear **advantage over state table** method where looping required duplicate hardware. Thus **hardware is reduced**.

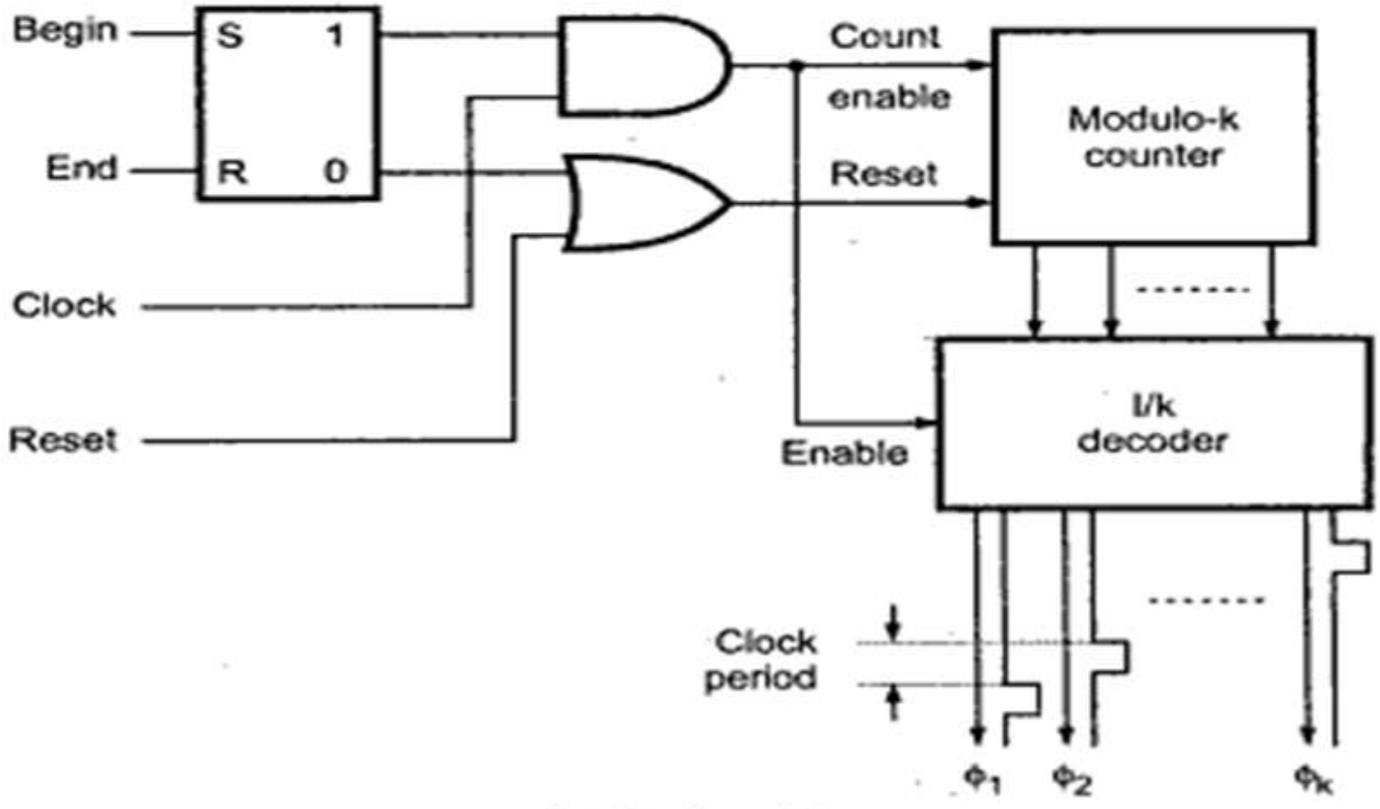
Disadvantage:

- The **number** of **delay elements** is approximately **equal** to the number of **states**. Thus **if** there are **lot of states**, the **circuit** can become **too large**.
- Moreover, if there are a **lot of D flip-flops** then the **synchronization** of delay can become **difficult**.

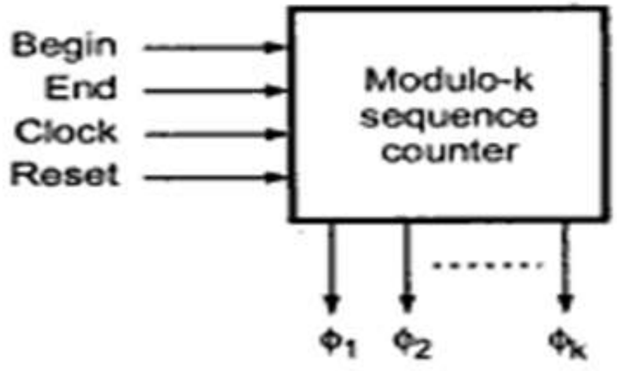
3) **Sequence Counter Method**

- This is the **best method** for Hardwired design.
- It is an improved version of the delay element method, where "k" delay elements are replaced by a single **modulo-K counter**.
- The **concept** is very simple:
  - Our objective is that the **"k" control signals** should be produced one after the other, with a **fixed time interval** of **one clock cycle** between each of them.
  - For this we had used "k" D flip-flops, each giving one clock pulse delay.
  - Now we use a single **modulo-k counter** and provide the same **clock input** to it.
  - The **output** of this counter is connected to a **1:k decoder.**
  - Thus we will get the **"k" different outputs** from the decoder, each after **one clock pulse,** as required.

Fig. A Modulo-K sequence counter : A) logic diagram B) Symbol



Logic diagram



Symbol

- As seen above, the time gap between each of the "k" control signals produced is of one clock pulse.
- Additionally the begin, End and reset signals are also provided, to control the sequence of counting.

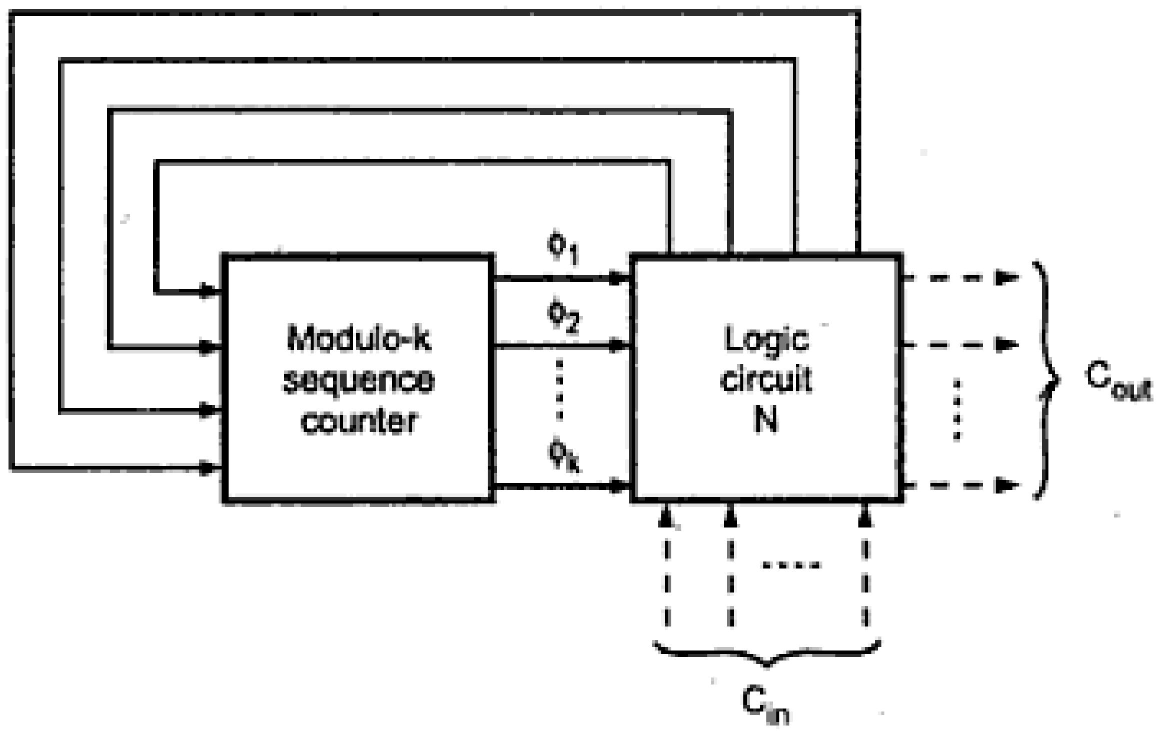
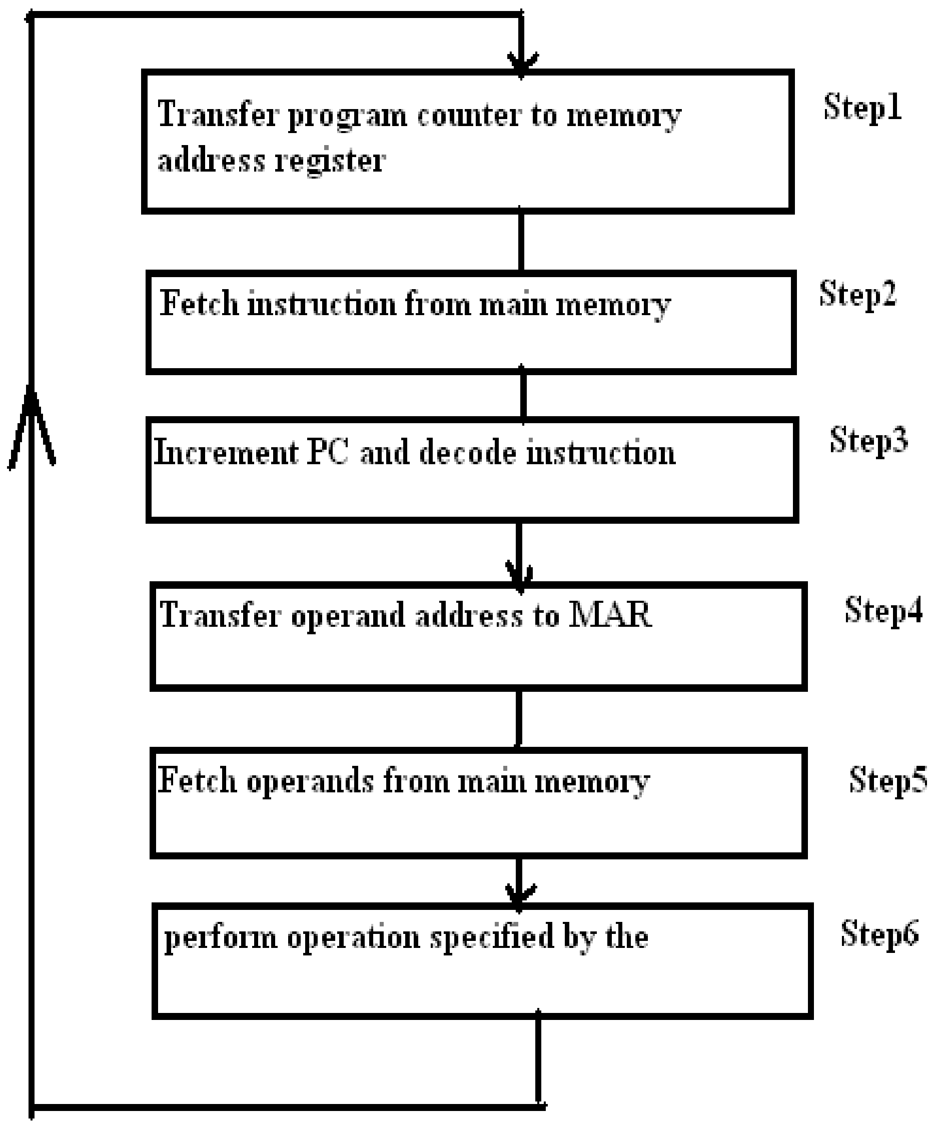


Fig. 3.20 (a) A control unit based on a sequence counter

Example:

Consider the closed loop behavior of a typical CPU


Transfer program counter to memory address register
Step1
Fetch instruction from main memory
Step2
Increment PC and decode instruction
Step3
Transfer operand address to MAR
Step4
Fetch operands from main memory
Step5
perform operation specified by the
Step6

- These six steps are performed repeatedly in a loop.
- This can be implemented using a modulo-6 counter.
- To enable repetition, the Q_6 output should be connected to the begin input in the circuit. This wi cause the counting to restart ach time, after reaching the last state.

Advantage:

> The use of hardware is minimal, and hence it is the best method for large, complex Control Units.

Disadvantage:

> For simple Control Units this method can be expensive due to the use of a sequence counter and a decoder.