

Module 4

Programmable Logic Control Systems

Lesson 21

Programming of PLCs: Sequential Function Charts

Instructional Objectives

After learning the lesson students should be able to

- A. Describe the major features of the IEC 1131-3 standard for PLC programming
- B. Describe the major syntax conventions of the SFC programming language
- C. Identify valid and invalid SFC segments
- D. Develop SFC programs for simple sequence control problems

Introduction

We have studied the RLL in a previous lesson. There are also other languages to program a PLC in, than the RLL. Most of the significant manufacturers support about 3 to 5 programming languages. Some of these languages, such as the RLL, have been in use for a long time. While most manufacturers used similar languages, these were not standardised in terms of syntactic features. Thus programs developed for one would not run in another without considerable modifications, often mostly syntactic. In the last few years there has been effort to standardise the PLC programming languages by the International Electrotechnical Commission (IEC). One of the languages, namely the Sequential Function Chart, which offers significant advantages towards development of complex structured PLC programs for concurrent industrial processes, is studied in detail. Know other languages are introduced in brief.

IEC 1131-3: The International Programmable Controller Language Standard

IEC 1131 is an international standard for PLCs formulated by the International Electrotechnical Commission (IEC). As regards PLC programming, it specifies the syntax, semantics and graphics symbols for the following PLC programming languages:

- Ladder diagram (LD)
- Sequential Function Charts (SFC)
- Function Block Diagram (FBD)
- Structured Text (ST)
- Instruction List (IL)

IEC 1131 was developed to address the industry demands for greater interoperability and standardisation among PLC hardware and software products and was completed in 1993. A component of the IEC 1131, the IEC 1131-3 define the standards for data types and programming. The goal for developing the standard was to propose a programming paradigm that would contain features to suit a large variety of control applications, which would eliminate proprietary barriers for the customer and their associated training costs. The language specification takes into account modern software engineering principles for developing clean, readable and modular code. One of the benefits of the standard is that it allows multiple languages to be used simultaneously, thus enabling the program developer to use the language best suited to each control task.

Major Features of IEC 1131-3

The following are some of the major features of the standard.

1. **Multiple Language Support:** One of the main features of the standard is that it allows multiple languages to be used simultaneously, thus enabling the program developer to use the language best suited to each control task.
2. **Code Reusability:** The control algorithm can include reusable entities referred to as "program organization units (POUs)" which include Functions, Function Blocks, and Programs. These POUs are reusable within a program and can be stored in user-declared libraries for import into other control programs.
3. **Library Support:** The IEC-1131 Standard includes a library of pre-programmed functions and function blocks. An IEC compliant controller supports these as a "firmware" library, that is, the library is pre-coded in executable form into a prom or flash ram on the device. Additionally, manufacturers can supply libraries of their own functions. Users can also develop their own libraries, which can include calls to the IEC standard library and any applicable manufacturers' libraries.
4. **Execution Models:** The general construct of a control algorithm includes the use of "tasks", each of which can have one or more Program POUs. A task is an independently schedulable software entity and can be assigned a cyclic rate of execution, can be event driven, or be triggered by specific system functions, such as startup.

IEC 1131-3 Programming Languages

IEC 1131-3 defines two graphical programming languages (Ladder Diagram and Function Block Diagram), two textual languages (Instruction List and Structured Text), and a fifth language (Sequential Function Chart) that is a tool to define the program architecture and execution semantics. The set of languages include assembly-like low-level language like the Instruction List, as well as Structured Text having features similar to those of a high level programming language. Using these, different computational tasks of a control algorithm can be programmed in different languages, then linked into a single executable file. Below we first describe FBD, IL and ST in brief. The SFC is discussed in greater detail later in this lesson.

Function Block Diagram (FBD)

The function block diagram is a key product of the standard IEC 1131-3. FBD is a graphical language that lets users easily describe complex procedures by simply wiring together function blocks, much like drawing a circuit diagram with the help of a graphical editor. Function blocks are basically algorithms that can retain their internal state and compute their outputs using the persistent internal state and the input arguments. Thus, while a static mathematical function will always return the same output given the same input (e.g. sine, cosine), a function block can return a different value given the same input, depending on its internal state (e.g. filters, PID control). This graphical language clearly indicates the information or data flow among the different computational blocks and how the over all computation is decomposed among smaller blocks, each computing a well defined operation. It also provides good program documentation. The IEC 1131 standard includes a wide range of standard function blocks for performing a

variety of operations, and both users and vendors can create their own. A typical simple function block is shown below in Fig. 21.1.

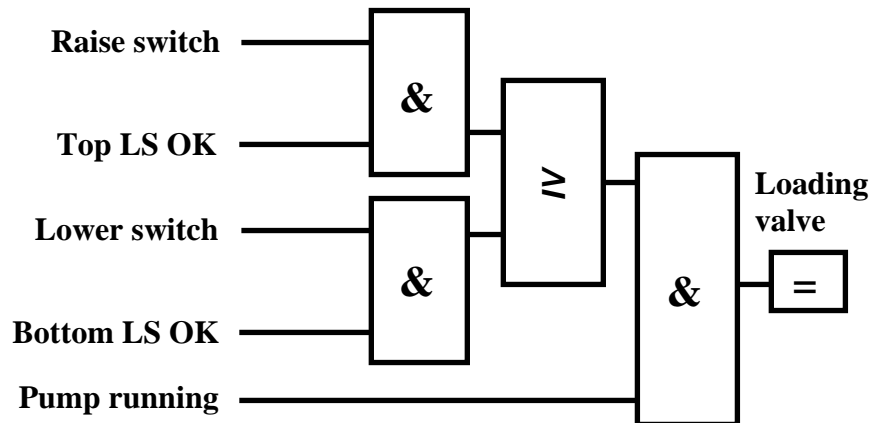


Fig. 21.1 Combinational logic programmed with function blocks

Structured Text (ST)

Structured Text (ST) is a high-level structured programming language designed for expressing algorithms with complex statements not suitable for description in a graphical format. ST supports a set of data types to accommodate analog and digital values, times, dates, and other data. It has operators to allow logical branching (IF), multiple branching (CASE), and looping (FOR, WHILE...DO and REPEAT...UNTIL). Typically, a programmer would create his own algorithms as Functions or Function Blocks in Structured Text and use them as callable procedures in any program. A typical simple program segment written in Structured Text is shown below in Fig. 21.2.

```

if (temp <= max_temp)
then
    cool_valve :=false;
    m_vlv := (vlv23 +dbh18) /2;
else
    alarm := true;
end_if;

```

Fig. 21.2 Simple program segment written in Structured Text.

Instruction List (IL)

A low-level assembly-like language, IL is useful for relatively simple applications, and works on simple digital data types such as boolean, integer. It is tedious and error prone to write large programs in such low level languages. However, because complete control of the implementation, including elementary arithmetic and logical operations, rests with the programmer, it is used for optimizing small parts of a program in terms of execution times and memory. A typical simple program segment written in Instruction List is shown below in Fig. 21.3.

start_cmd:	LD	ii 01
	ADD	10
mul_op:	MUL(i_gain
	SUB	offset 01
)	
	ST	op 01
	JMPNC	mul_op

Fig. 21.3 Simple program segment written in Instruction List

Point to Ponder: 1

- Name one programming task for which, IL would be your chosen language.*
- Name one programming task for which, ST would be your preferred language over FBD.*
- For whom are the features code reusability and library support important, and why?*

Sequential Function Chart (SFC)

SFC is a graphical method, which represents the functions of a sequential automated system as a sequence of steps and transitions.

SFC may also be viewed as an organizational language for structuring a program into well-defined steps, which are similar conceptually to states, and conditioned transitions between steps to form a sequential control algorithm. While an SFC defines the architecture of the software modules and how they are to be executed, the other four languages are used to code the action logic that exercises the outputs, within the modules to be executed within each step. Similar modules are used for computation of the logical enabling conditions for each transition.

Each step of SFC comprises actions that are executed, depending on whether the step is active or inactive. A step is active when the flow of control passes from one step to the next through a conditional transition that is enabled when the corresponding transition logic evaluates to true. If the transition condition is true, control passes from the current step, which becomes inactive, to the next step, which then becomes active. Each control function can, therefore, be represented by a group of steps and transitions in the form of a graph with steps labeling the nodes and transitions labeling the edges. This graph is called a Sequential Function Chart (SFC).

Steps

Each step is a control program module which may be programmed in RLL or any other language. Two types of steps may be used in a sequential function chart: initial and regular. They are represented graphically as shown below in Fig. 21.4.

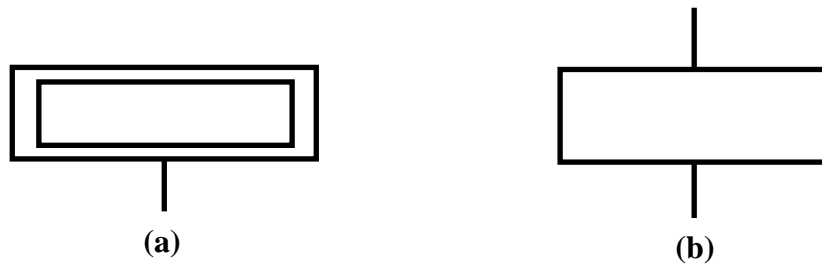


Fig. 21.4 An initial step (a) and a regular step (b) in an SFC

The *initial step* is executed the first time the SFC block is executed or as a result of a reset operation performed by a special function named SFC_RESET. There can be one and only one initial step in an SFC. The initial step cannot appear within a simultaneous branch construct, (which is described later in this section) but it may appear anywhere else.

A regular step is executed if the transitional logic preceding the step makes the step active. There can be one or many regular steps in an SFC network, one or more of which may be active at a time. Only the active steps are evaluated during a scan.

Each step may have action logic consisting, say, of zero or more rungs programmed in Relay Ladder Diagram (RLD) logic language. *Action Logic* is the logic associated with a step, i.e., the logic, programmed by RLL or any other logic, which is executed when the step is active. When a step becomes inactive, its state is initialised to its default state. A collection of steps may be labeled together as a macro-step.

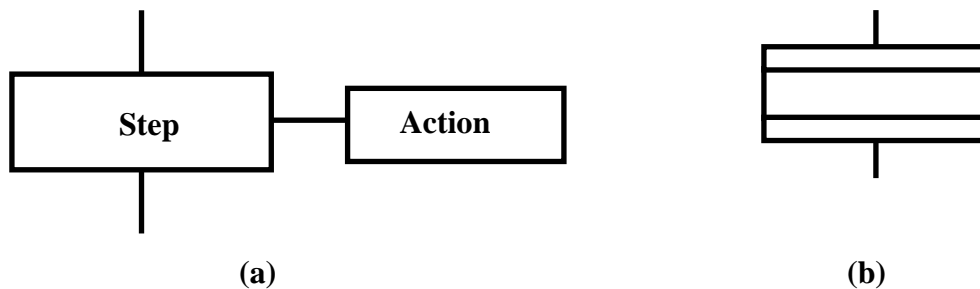


Fig. 21.5 A step with action logic (a) and a macro-step (b) in an SFC

Transitions

Each transition is a program module like a step that finally evaluates a transition variable. Once a transition variable evaluates to true the step(s) following it are activated and those preceding it are deactivated. Only transitions following active states are considered active and evaluated during a scan. Transitions can also be a simpler entity such as a variable value whose value may be set by simple digital input. Transition logic can be programmed in any language. If programmed in RLL, each transition must contain a rung that ends with an output coil to set its transition variable.

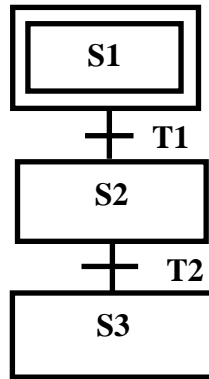


Fig. 21.6 Transitions connect steps in an SFC

The SFC in Fig. 21.6 shows how the transitions connect steps in an SFC. Initially, step S1 is active. Thus transition T1 is also active. When the transition variable T1 becomes true, immediately, S1 becomes inactive, S2 becomes active while T1 becomes inactive and T2 becomes active.

Point to Ponder: 2

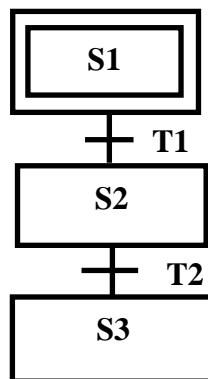
- A. *What is the difference between a step of an SFC and a state of an FSM?*
- B. *Why action logic is separately indicated from step logic, although both occur in the same step?*
- C. *How is the computation for step logic different from that of transition logic?*

Basic Control Structures

Six basic control structures used in a sequential function chart are discussed below. .

Simple Sequence

In a simple sequence, control passes from step S2 to step S3 **only if** step S2 is active and transition T2 evaluates true.



(a)

Scan	S1	T1	S2	T2	S3	T3
1	A	A	I	I	I	I
2	I	I	A	A	I	I
3	I	I	A	A	I	I
4	I	I	I	I	A	A

(b)

Fig. 21.7 A simple sequence in an SFC (a) and its execution over scans (b)

The table in Fig. 21.7 (b) indicates the status (A : active; I:inactive) of the steps and transitions over scan cycles.

Divergence of a Selective Sequence

Divergence of a Selective Sequence means that after a step there is a choice of two or more transitions which can evaluate to be true. However, at a time, only one of which can be true, and therefore, the sequence of steps following that transition only are activated. For example, in the divergent selective sequence shown in Fig. 21.8, control passes from step S1 to step S2 if step S1 is active and transition T1 evaluates true. Control passes from step S1 to step S3 if step S1 is active, *transition T1 is not true, and transition T2 is true* (left-to-right priority of transition). Thus, at a time, exactly one branch of the selective sequence is selected. A left-to-right priority is used to determine the action branch if more than one transition evaluates true at the same time. A selective divergence must be preceded by one step. The first element after a selective divergence must be a transition. In terms of familiar programming constructs, this is similar to an IF...THEN...ELSEIF...ELSEIF....ELSE...construct.

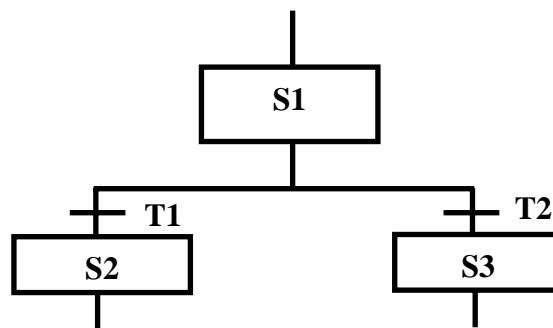


Fig. 21.8 Divergence of a selective sequence

Convergence of a Selective Sequence

A convergence of the divergent selective sequences must follow. Here transitions from each branch of the selective sequence converge eventually to one step. As shown in Fig. 21.9, in a convergent selective sequence, control passes from step S4 to step S6 if step S4 is active and transition T3 evaluates true. Similarly, control passes from step S5 to step S6 if step S5 is active and transition T4 is true.

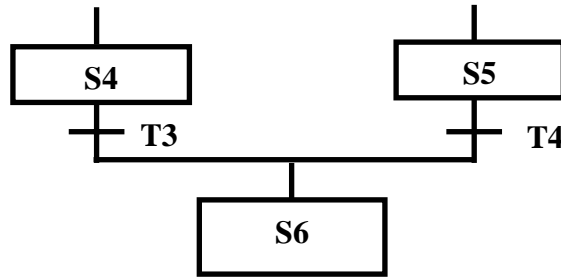


Fig. 21.9 Convergence of a selective sequence

Divergence of a Simultaneous Sequence

In contrast with a selective sequence, in a simultaneous divergent branch more than one step can become active. Thus, in this case, we have two or more branches to be active simultaneously. In the divergent simultaneous sequence shown in Fig. 21.10, control transfers from step S1 to step S2 and step S3 if step S1 is active *and* transition T1 evaluates true. Both steps S2 and S3 will become active. Note that the action logic of one step will be executed before the action logic of the other. The order of which step is executed first is undefined in the standard, and depends on a particular implementation. Note that the same thing happens for the state logic too, since computation is necessarily sequential in a single processor system. However the sequence becomes noticeable for action logic, since the output is observed in the field. A simultaneous divergent branch must be preceded by one transition. A step must be the first element after a simultaneous branch.

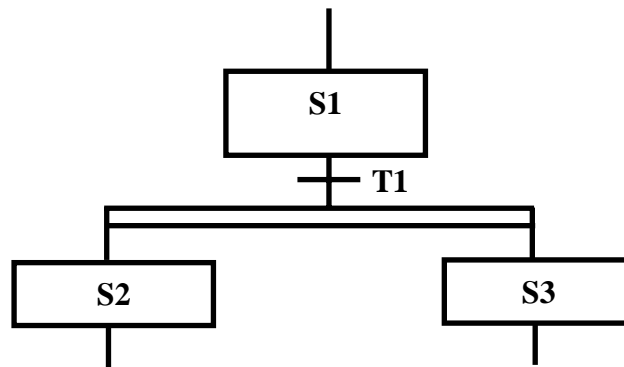


Fig. 21.10 Divergence of a simultaneous sequence

Convergence of a Simultaneous Sequence

A simultaneous convergent branch concludes a simultaneous sequence. It can only be preceded by step elements and not transitions as in the case of a selective sequence. It must be followed by one transition. In the convergent simultaneous sequence shown in Fig. 21.11, control passes from step S4 and step S5 to step S6 if steps S5 and S6 are both active and transition T2 evaluates true.

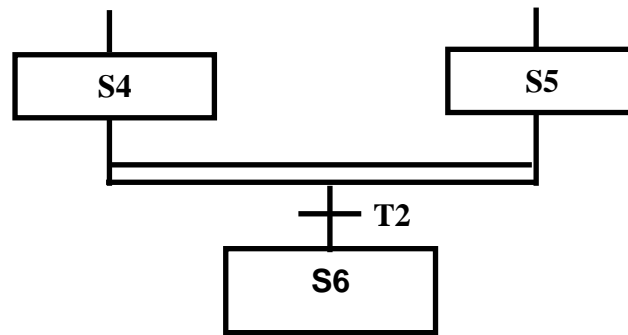


Fig. 21.11 Convergence of a simultaneous sequence

The transition logic for T2 is only executed when all of the steps at the end of the simultaneous sequence are active.

Source and Destination Connectors

Source and destination connectors are used to create forward and backward jumps in an SFC. The keywords *jump* and *cycle* denote connectors in the SFC shown below. Backward jumps are called *cycles*. In the forward *jump* sequence shown in Fig. 21.12, control passes from step S2 to step S4 if step S2 is active, transition T2 evaluates to be false, and transition T3 evaluates to be true. In the backward jump (or cycle) sequence, control passes from step S4 to step S1 if step S4 is active and transition T5 evaluates true. Source and destination connectors cannot occur before a transition. Source connectors must occur immediately after the transition and indicate that the transfer of control takes place once the transition fires. Destination connectors must occur immediately before a step indicating that the transfer of control after the transition before the corresponding source connector, this step becomes active.

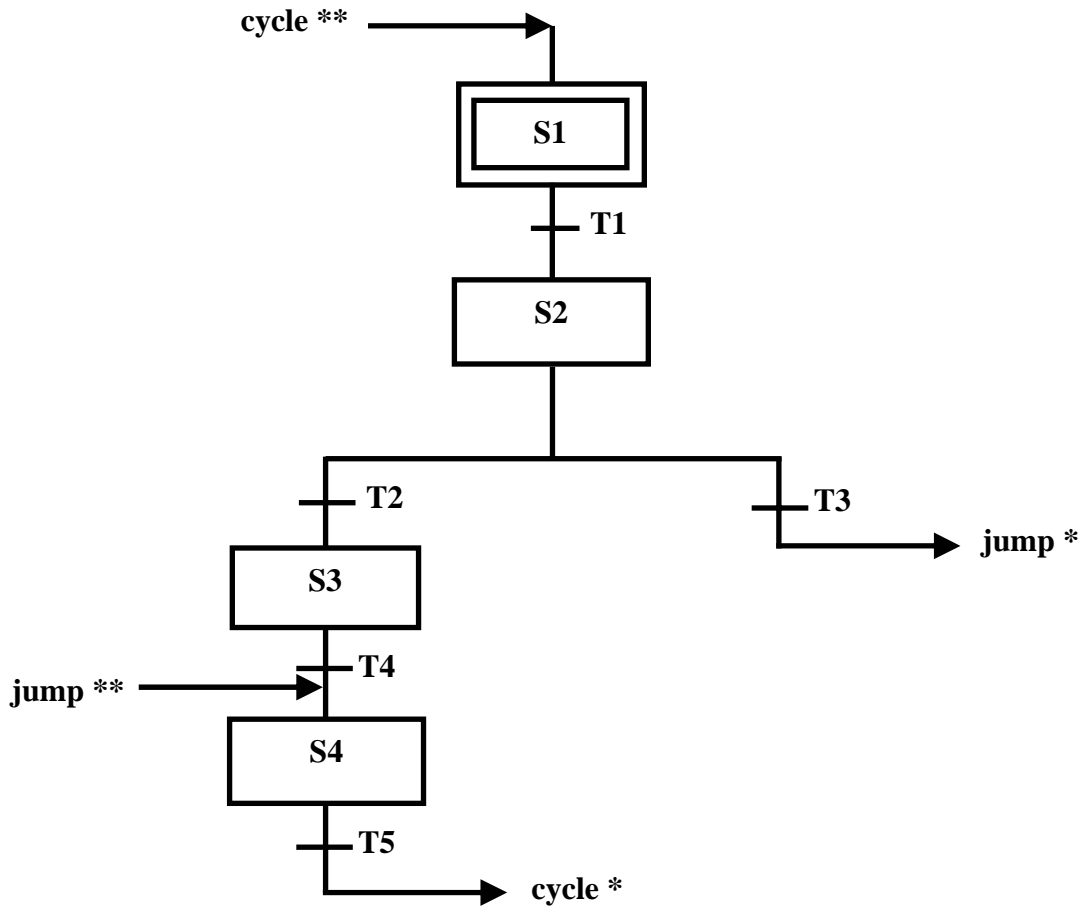


Fig. 21.12 Source and Destination Connectors

Many PLCs also allow SFCs to be entered as graphic diagrams. Small segments of ladder logic can then be entered for each transition and action. Each segment of ladder logic is kept in a separate program. The architecture of such programs is discussed next.

Point to Ponder: 3

- A. Identify whether the SFC segments indicated in Figs. 21.13-21.15 are valid. If not, justify your answer.

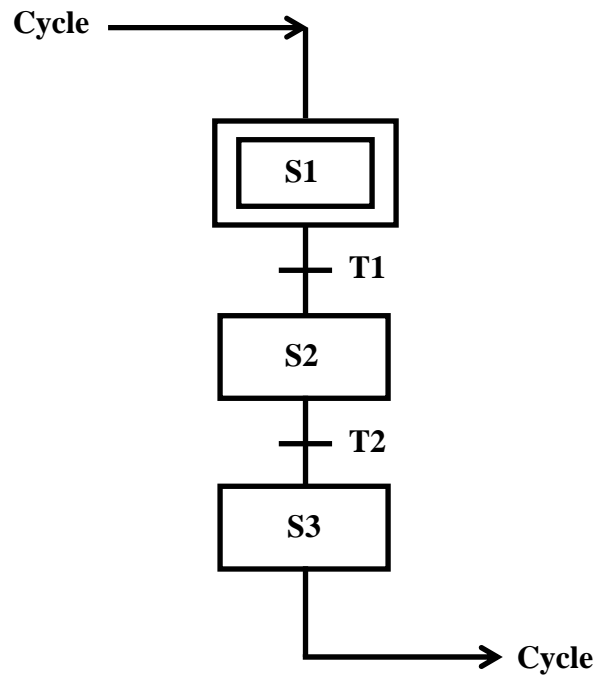


Fig. 21.13

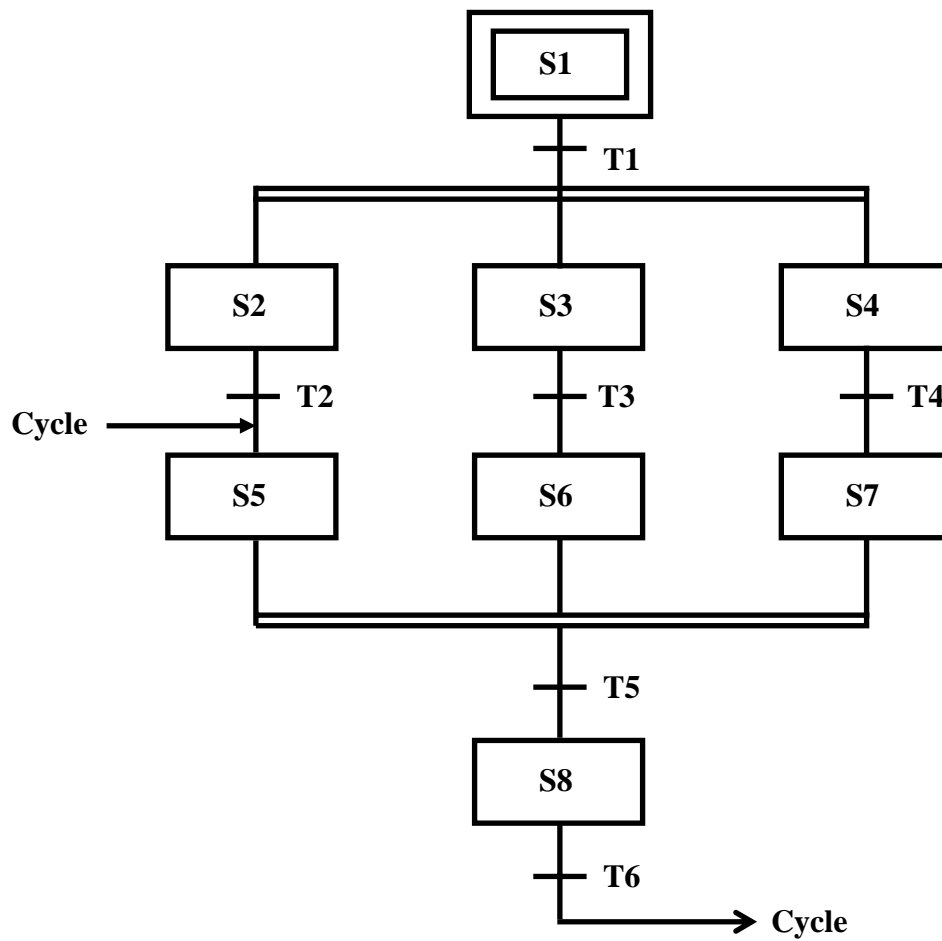


Fig. 21.14

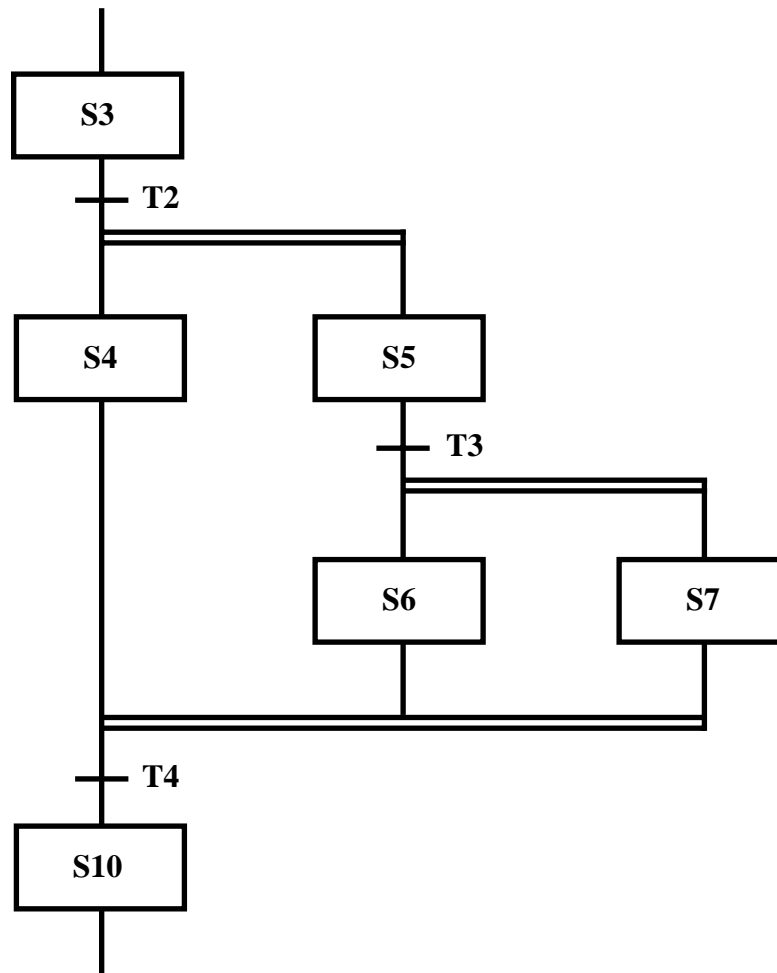


Fig. 21.15

Control Program Architecture with SFCs

A typical architecture of a control program with SFCs is shown in Fig. 21.16. Here the main program block is organised as an SFC. Each step and transition in the SFC of the main program block is coded as a module. These modules may be coded using any of the languages under the 1131-3 standard. These may be SFCs themselves. In general these modules may be organised in terms of a Preprocessing and a Post Processing block, in addition to the main sequential processing block.

Preprocessing

This section is processed at the start of every scan. Normally, RLD preprocessing logic is used to process, at the start of the scan cycle, events which may affect the sequential processing section of the program. These events may include:

- Initialization;
- Operator commands;
- Resetting the SFC to the initial state.

Sequential Processing

This portion of the PLC scan consists of evolving the SFC to its next state and processing the action logic of any steps that become active. Only the logic associated with active steps and transitions is scanned by the PLC, leading to a significant reduction of scan time.

Post processing

This section is processed every scan after the SFC is complete. It may contain Relay Ladder Diagram (RLD) logic to process safety interlocks, etc.

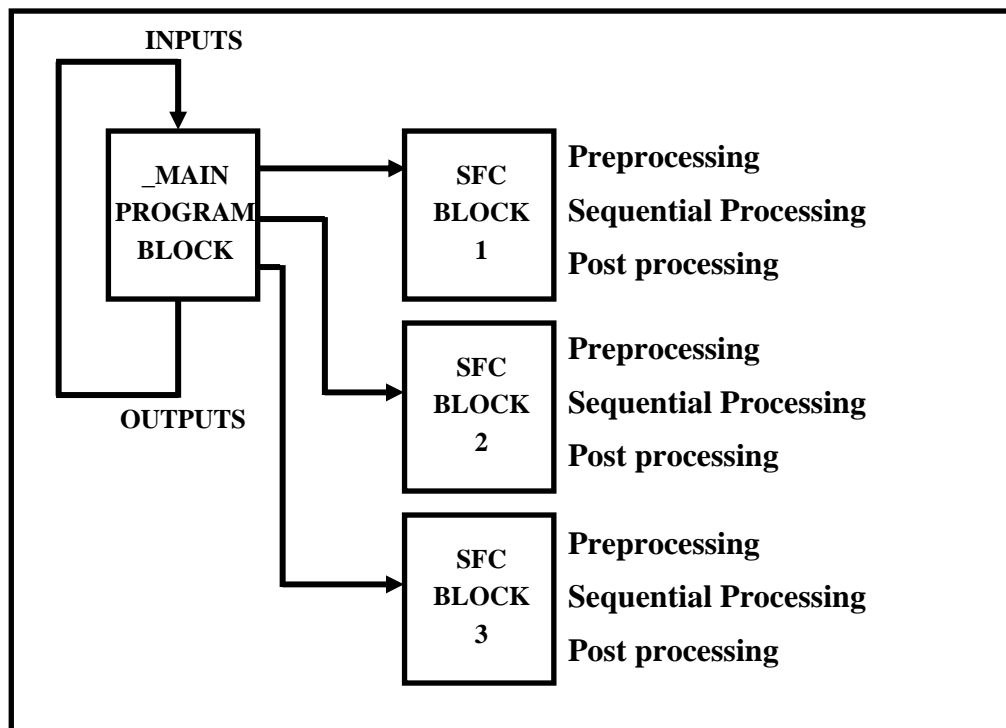


Fig. 21.16 Architecture of Control Software organized with SFCs

Although SFCs are really needed for large and complex control problem, within the constraints of this lesson, we present below an SFC-based solution for the industrial stamping process control problem below.

Industrial Logic Control Example Revisited

Consider the industrial logic control example of the stamping process presented in Lesson 18 and further discussed in Lesson 20. Let us recall the description of the process given in the lesson. For convenience of reference the description and a pictorial representation of the process is reproduced below.

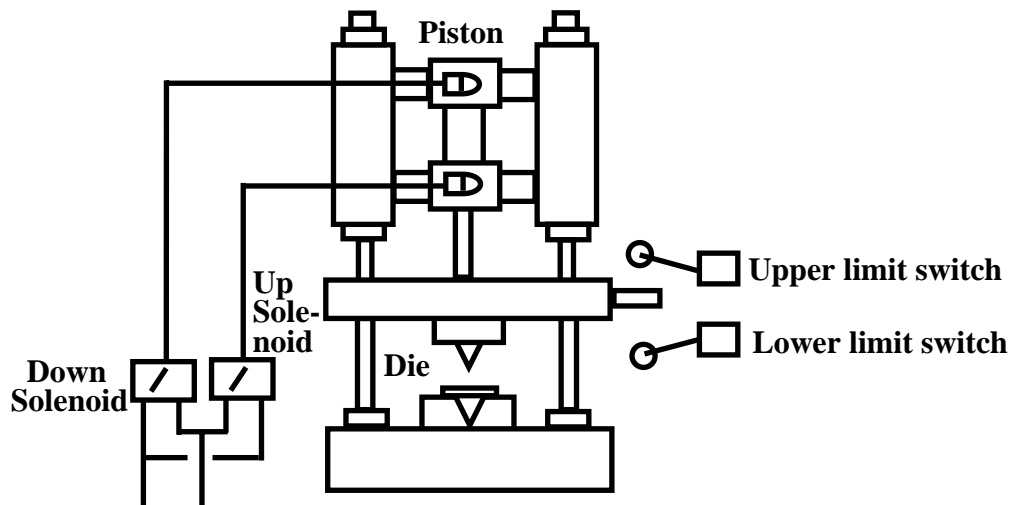


Fig. 21.17 An Industrial Logic Control Example

Summary of Requirements Analysis for the Stamping Process

A summary of the requirement analysis carried out in Lesson 20 is given below for ready reference.

Process Control Inputs

- *Part sensor*: A position switch that detects when a part has been placed.
- *Auto PB* : A push button that indicates the ‘automatic mode’
- *Stop PB*: A push button to stop the machine while the piston is moving down.
- *Reset PB*: After pressing Stop PB, this push button indicates that the machine is ready to return to stamping in the auto mode.
- *Bottom LS*: This sensor indicates the bottom position for the piston.
- *Top LS*: This sensor indicates top position for the piston.

Process Control Outputs

- *Up Solenoid*: Drives the piston up.
- *Down Solenoid*: Drives the piston down.
- *Auto Mode Indicator*: Indicates ‘Auto’ mode.
- *Part Hold*: Holds the part during stamping.

Sequence of Events and Actions

- A. The “Auto” PB turns the Auto Indicator Lamp on
- B. When a part is detected, the press ram advances down to the bottom limit switch
- C. The press then retracts up to top limit switch and stops

- D. A “Stop” PB, if pressed, stops the press only when it is going down
- E. If the “Stop” PB is pressed, the “Reset” PB must be pressed before the “Auto” PB can be pressed
- F. After retracting, the press waits till the part is removed and the next part is detected

State Transition Diagram

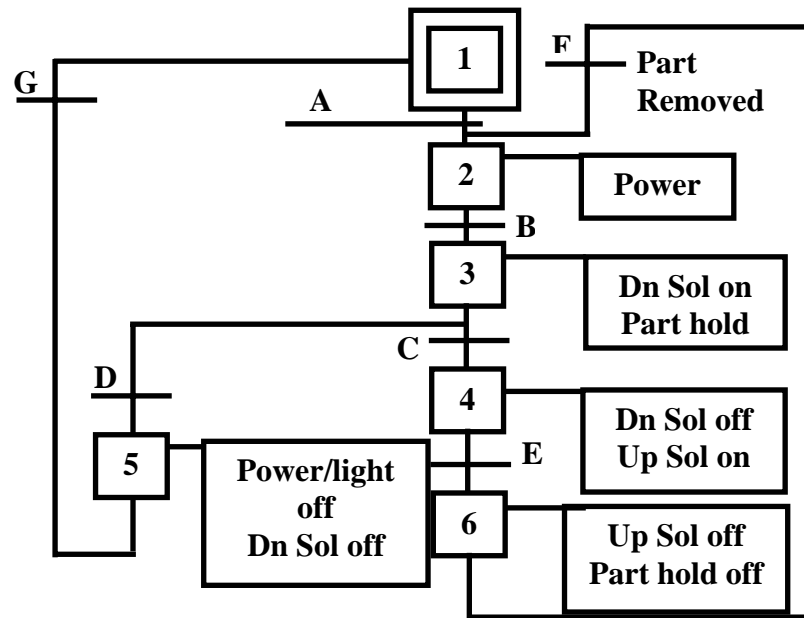


Fig. 21.18 The State Diagram for the industrial stamping press

State No.	1	2	3	4	5	6
O/P						
Auto Indicator	0	1	1	1	0	1
Part Hold	0	0	1	1	0	0
Up Sol	0	0	0	1	0	0
Down Sol	0	0	1	0	0	0

Fig. 21.19 The Output Table for the industrial stamping press

SFC-based Implementation of the Stamping Process Controller

The SFC for the industrial stamping process control problem is shown in Fig. 21.20. It is seen to be identical to the state diagram drawn for the process in Fig. 21.18, except possibly for graphical syntax used in the two diagrams. Thus for strictly sequential processes the SFC is nothing but the state diagram. However, it can also model concurrency, which cannot be captured in a simple FSM. There exist many DES modelling formalisms that capture concurrent FSM dynamics (such as Statecharts, widely used to model software dynamic modelling under Unified Modelling Language

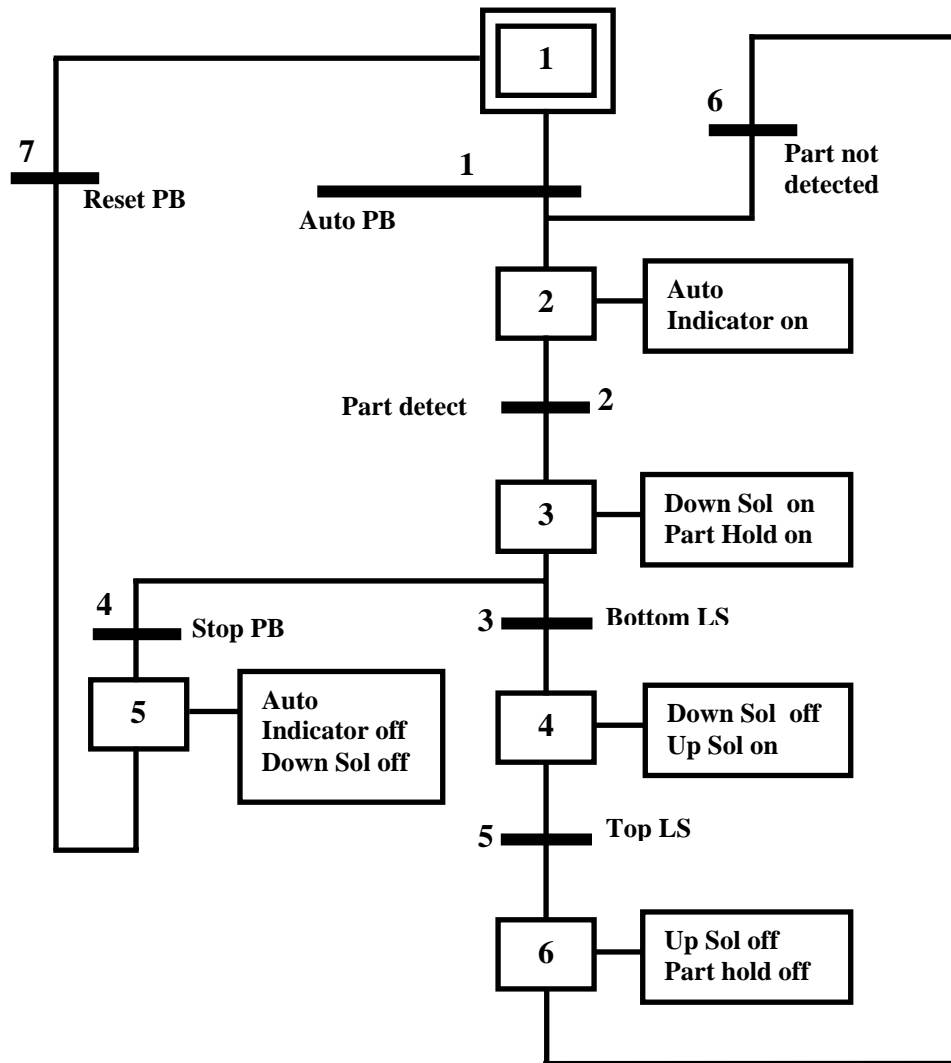


Figure 21.20 SFC for Controlling a Stamping Press

Formalism). One can now develop the logic for the steps transitions and actions. In this lesson RLL is used for this purpose. Some of the ladder logic for the SFC is shown in Figure 21.x.

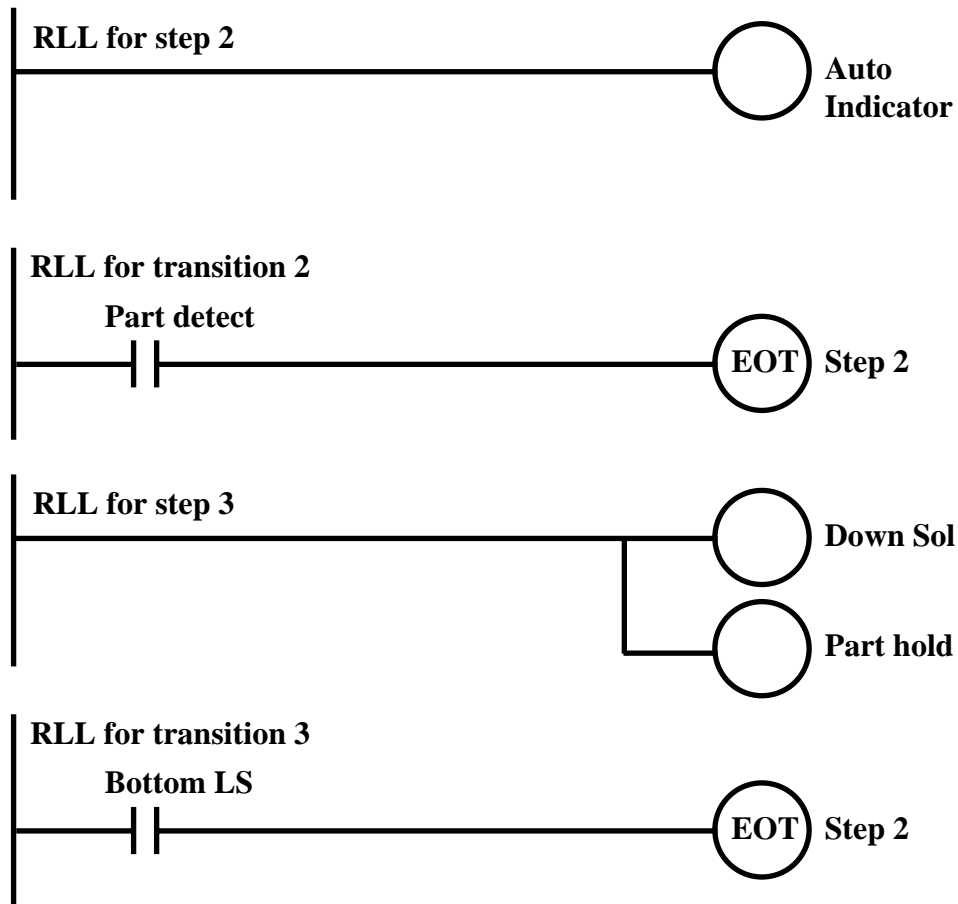


Fig. 21.21 Sample Ladder Logic for a Graphical SFC Program.

Note the following distinctions of the SFC based implementation with that of one the state-based implementation with pure RLL.

- A. The initialisation logic can now be organised within the initial step of the SFC, which is explicitly meant for this purpose. Note that in the RLL implementation this had to be included within the logic for State 1. The SFC-based implementation is cleaner in this respect.
- B. The order of execution of the state and transition logic is determined by SFC execution semantics. Thus there is no need to worry about the order in which these program segments are physically ordered within the overall program.
- C. In the RLL implementation each and every step, transition and action logic is evaluated in every scan cycle. In SFC only the active states, their action logic and the active transitions are evaluated. This results in significant saving in processor time, which may be utilised in the system for other purposes.
- D. The ladder logic includes a new instruction, EOT, which will tell the PLC when a transition has completed. When the rung of ladder logic with the EOT output becomes true the SFC will move to the next step or transition.

Point to Ponder: 4

- A. *Develop an SFC for a two person assembly station. The station has two presses that may be used at the same time. Each press has a cycle button that will start the advance of the press. A bottom limit switch will stop the advance, and the cylinder must then be retracted until a top limit switch is hit.*
- B. *Create an SFC for traffic light control. The lights should have cross walk buttons for both directions of traffic lights. A normal light sequence for both directions will be green 16 seconds and yellow 4 seconds. If the cross walk button has been pushed, a walk light will be on for 10 seconds, and the green light will be extended to 24 seconds.*
- C. *Draw an SFC for a stamping press that can advance and retract when a cycle button is pushed, and then stop until the button is pushed again.*
- D. *Design a garage door controller using an SFC. The behavior of the garage door controller is as follows,*
 - a. *there is a single button in the garage, and a single button remote control*
 - b. *when the button is pushed the door will move up or down.*
 - c. *if the button is pushed once while moving, the door will stop, a second push will start motion again in the opposite direction.*
 - d. *there are top/bottom limit switches to stop the motion of the door.*
 - e. *there is a light beam across the bottom of the door. If the beam is cut while the door is closing the door will stop and reverse.*
 - f. *there is a garage light that will be on for 5 minutes after the door opens or closes.*

Answers, Remarks and Hints to Points to Ponder

Point to Ponder: 1

A. *Name one programming task for which, IL would be your chosen language.*

Ans: Consider a triple redundancy voting logic for 3 digital sensor inputs, for tolerance against sensor failures. The logic aims to select the three desired bits corresponding to the sensors from an input word. Then it evaluates a Boolean function that implements the voting logic (the exact Boolean logic for voting is left to the learner as an exercise). Note that all the above involve bit-operations on binary data types and are therefore easily and efficiently implemented using assembly like low-level languages. Hence the preference for IL.

B. *Name one programming task for which, ST would be your preferred language over FBD.*

Ans: Consider implementing a custom fuzzy-logic based PID controller. Since the controller involves logic and arithmetic based on real-valued data, RLLs are clearly not suited. Neither is IL, since the computations are complex and algorithmic in nature and based on real-valued data. Thus ST is the best suited for implementation of this algorithm

C. *For whom are code reusability and library support important, and why?*

Ans: These are very important for developers of control algorithms. This is because a proven library of routines of routines not only lead to faster development of control programs, they also lead to a better quality program in terms of a cleaner and more readable and reliable code.

Point to Ponder: 2

A. *What is the difference between a step of an SFC and a state of an FSM?*

Ans: A step of an SFC denotes a computation module which gets executed cyclically, as long as the step is active. A state of an FSM is an instantiation of the values of its state variables. Note that a step in the SFC can represent a possibly cyclic subgraph of an FSM through which the FSM moves during the time the step is active. In the simplest case, a step of an SFC represents one state of an FSM.

B. *Why action logic is separately indicated from step logic, although both occur in the same step?*

Ans: The computation within a step can be of two types, namely, those that update internal state variables other than outputs and those that update the outputs which are exercised on physical outputs. The second type of computation is named action logic and explicitly indicated at the steps. Since the physical outputs are of final importance, these are separately indicated for each step.

C. *How is the computation for step logic different from that of transition logic?*

Ans: Systems are supposed to spend time in the states. Transitions are instantaneous and merely indicate conditions under which systems change from one state to another. In an SFC, at any point of time, a number of steps and transitions are active. However, the computations of the steps can update output variables in the action logic while the computations in the transitions cannot.

Point to Ponder: 3

A. *Identify whether the SFC segments indicated in Figs. 21.13-21.15 are valid. If not, justify your answer.*

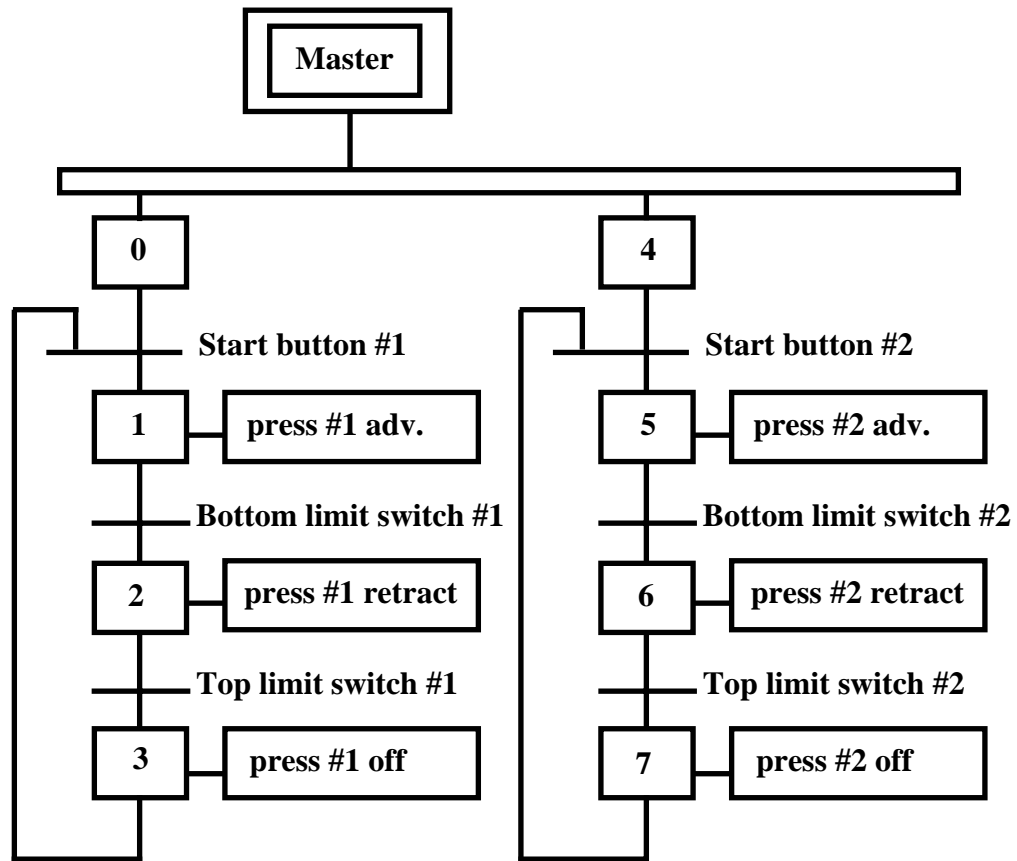
Ans:

- a. The SFC in Fig. 21.13 is invalid because a backward jump connects the two states S_3 and S_1 without an intervening transition. This is illegal. Any two steps in an SFC must contain an intermediate transition.
- b. The SFC in Fig. 21.14 is invalid because there is a jump into one of the branches of a simultaneous sequence. This is illegal, since the steps in the other branches of the simultaneous sequence are indeterminate.
- c. The SFC in Fig. 21.15 is valid.

Point to Ponder: 4

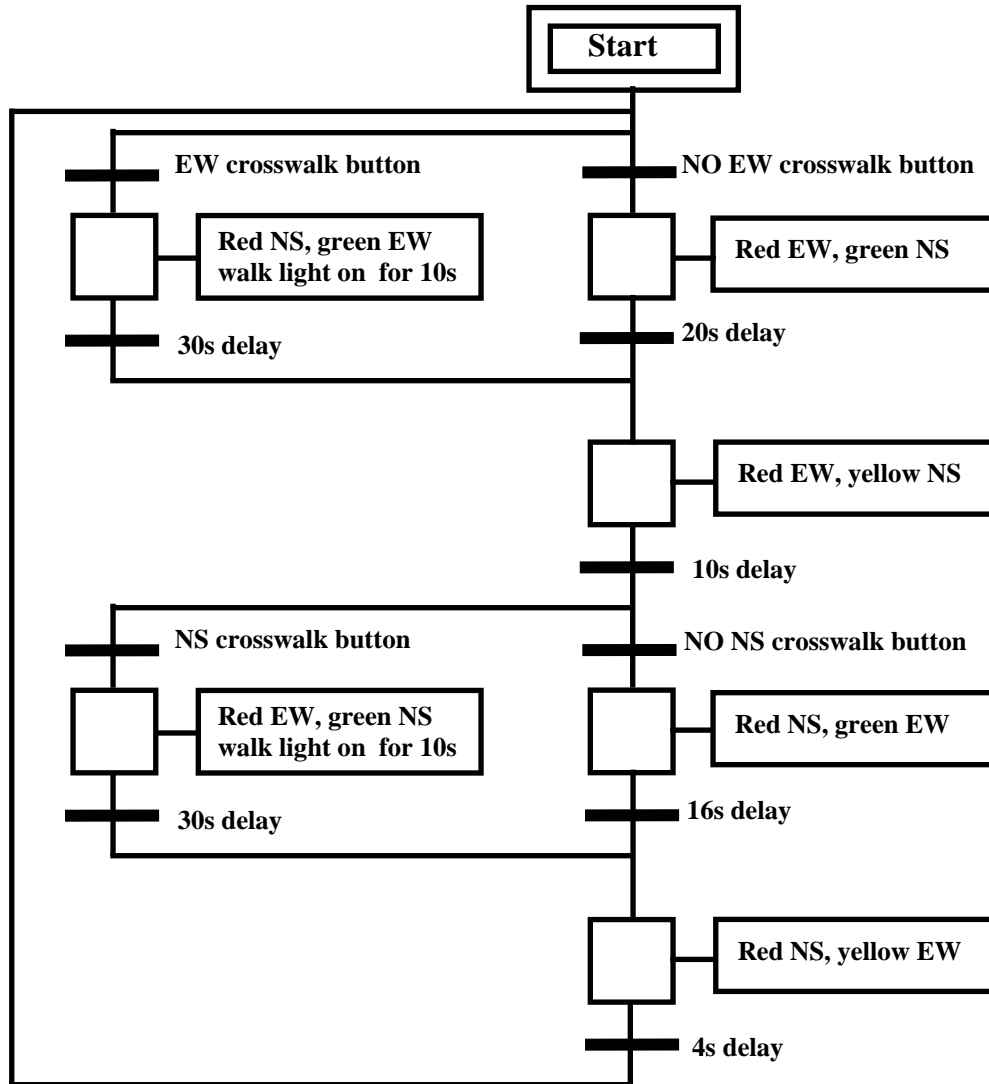
E. *Develop an SFC for a two person assembly station. The station has two presses that may be used at the same time. Each press has a start cycle button that will start the advance of the press. A bottom limit switch will stop the advance, and the cylinder must then be retracted until a top limit switch is hit.*

1.



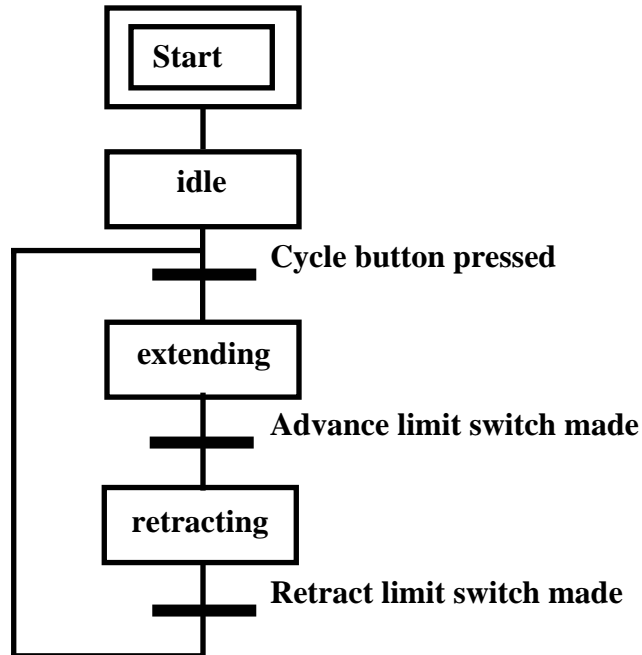
F. Create an SFC for traffic light control. The lights should have cross walk buttons for both directions of traffic lights. A normal light sequence for both directions will be green 20 seconds and yellow 10 seconds. If the cross walk button has been pushed, a walk light will be on for 10 seconds, and the green light will be extended to 30 seconds.

2.



G. Draw an SFC for a stamping press that can extend and retract when a cycle button is pushed, and then stop until the button is pushed again.

3.



H. Design a garage door controller using an SFC. The behavior of the garage door controller is as follow.

- a. There is one button inside the garage, and one button remote control.
- b. When either of the buttons are pushed the door will move up or down.
- c. If the button is pressed once while moving, the door will stop, a second press will start motion again in the opposite direction.
- d. There are top/bottom limit switches to stop the motion of the door when it reaches either of the two ends.
- e. There is an infrared beam across the bottom of the door. If the beam is interrupted while the door is closing the door will stop and reverse.
- f. There is a garage light that will be on for 5 minutes after the door opens or closes.

4.

