MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
*(A constituent unit of MAHE, Manipal)*

# Industrial Automation (ICE 3252)

## PLC Programming- Advanced Intermediate Functions

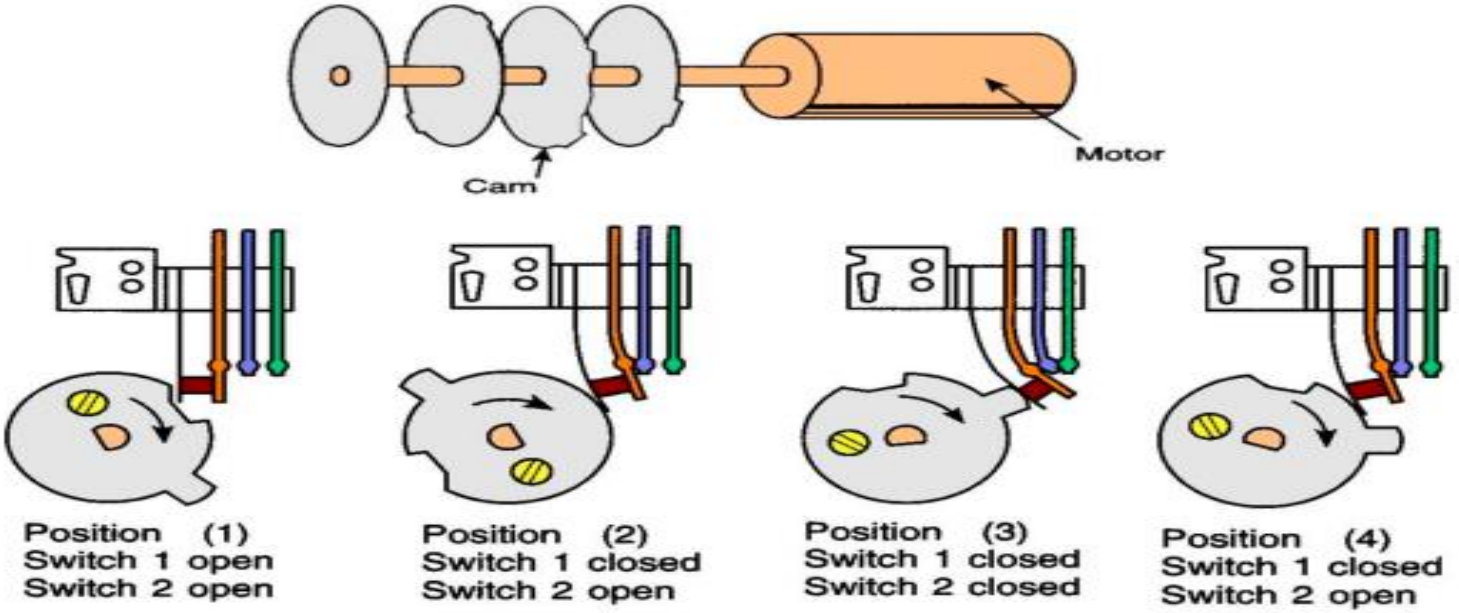
Bipin Krishna
Assistant Professor (Sr.)
ICE Department
Manipal Institute of Technology
MAHE, Karnataka, India

# Sequencer and Shift Register Instructions

- **Mechanical Sequencers**
- Sequencer instructions are designed to operate much like the mechanical rotating cam limit switch shown in Figure.
- These mechanical type sequencers are often referred to as drum switches, rotary switches, stepper switches, or cam switches.
- They are often used to control machinery that has a repetitive cycle of operation.

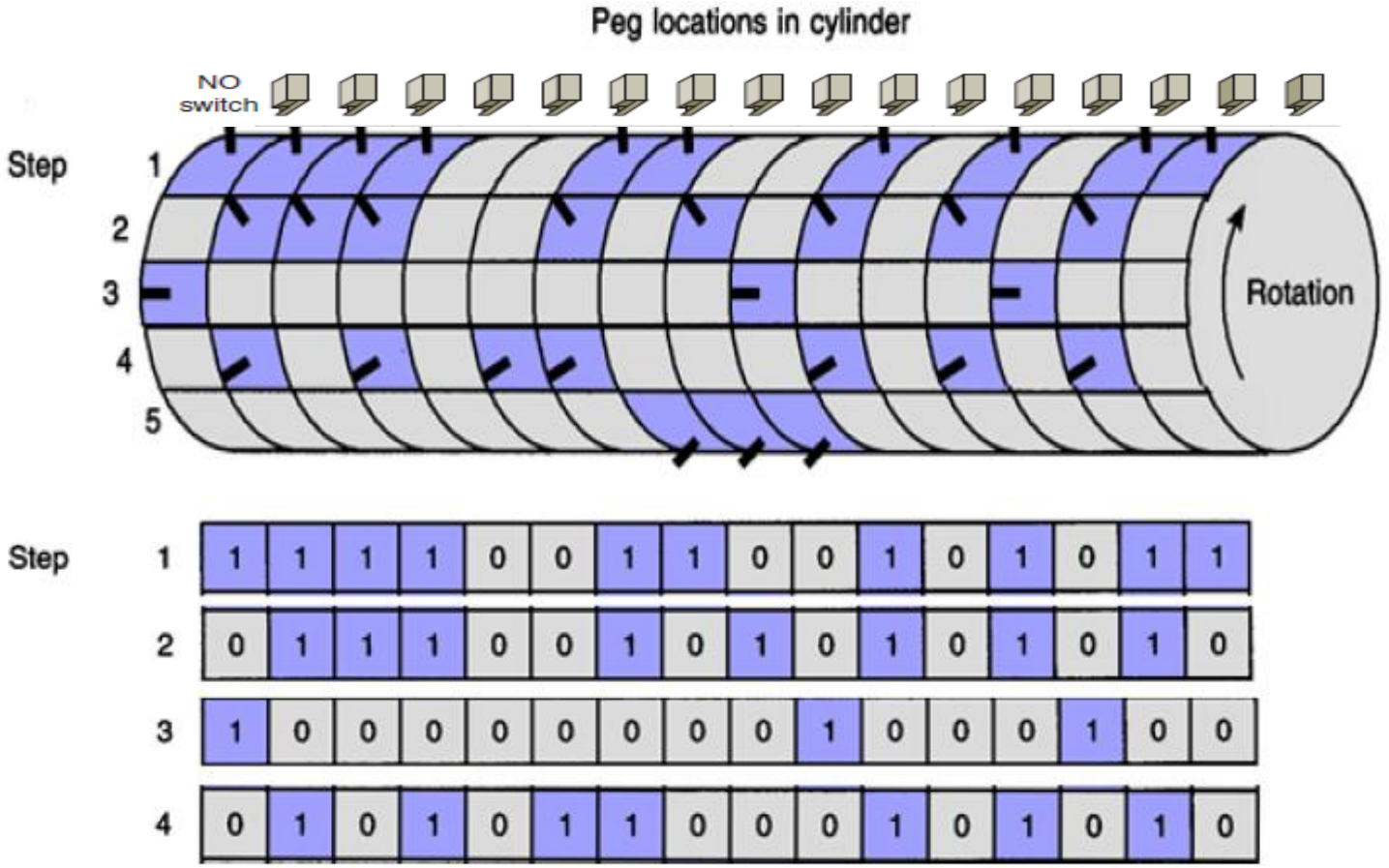
# Mechanical Sequencers

- An electric motor is used to drive the cams.
- A series of leaf-spring mounted contacts interacts with the cam so that in different degrees of rotation of the cam, various contacts are closed and opened to energize and de-energize various electrical devices.
- As the cams rotate, load devices connected to the contacts can change from an on to an off state, from an off to an on state, or remain at the same state.



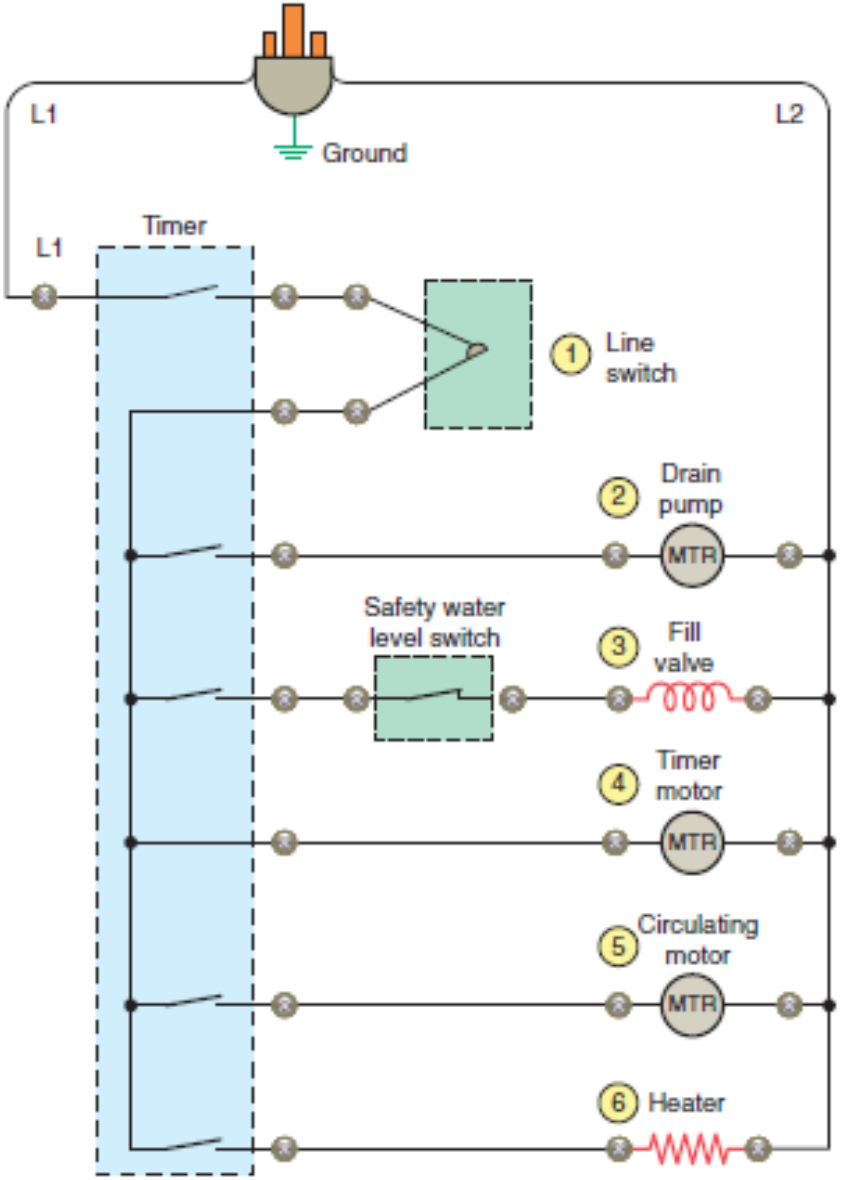
# Mechanical drum operated sequencer switch.

- The switch consists of a series of normally open contact blocks that are operated by pegs located on the motor-driven drum.



# Mechanical drum operated sequencer switch.

- Pegs are placed at specific locations around the circumference of the drum to operate the contact blocks.
- When the drum is rotated, contacts that align with the pegs will close, whereas the contacts where there are no pegs will remain open.
- The presence of a peg can be interpreted as logic 1, or on, and the absence of a peg as logic 0, or off.
- The equivalent sequencer data table illustrates the logic state for the first four steps of the drum cylinder.
- Each location where there was a peg is represented by a 1 (on), and the positions where there were no pegs are each represented by a 0 (off ).

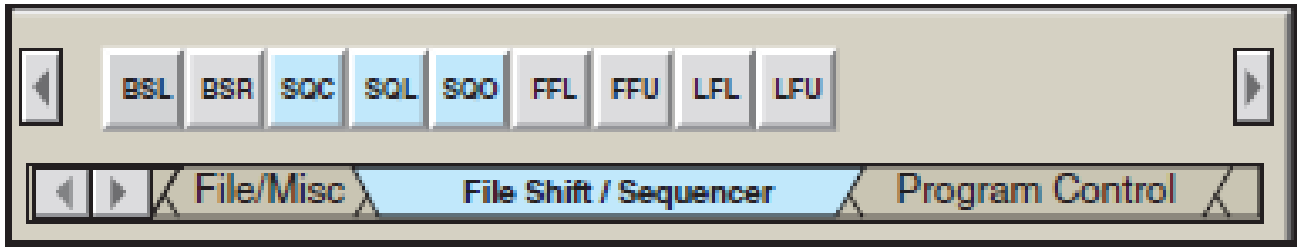



| Machine function | | Timer increment | Active devices |
|---|---|---|---|
| Off | | 0–1 | |
| First prerinse | Drain | 2 | 1 2 4 |
| | Fill | 3 | 1 3 4 5 |
| | Rinse | 4–5 | 1 4 5 6 |
| | Drain | 6 | 1 2 4 5 |
| Prewash | Fill | 7 | 1 3 4 5 |
| | Wash | 8–10 | 1 4 5 6 |
| | Drain | 11 | 1 2 4 5 |
| Second prerinse | Fill | 12 | 1 3 4 5 |
| | Rinse | 13–15 | 1 4 5 6 |
| | Drain | 16 | 1 2 4 |
| Wash | Fill | 17 | 1 3 4 |
| | Wash | 18–30 | 1 4 5 6 |
| | Drain | 31 | 1 2 4 5 |
| First rinse | Fill | 32 | 1 3 4 5 |
| | Rinse | 33–34 | 1 4 5 6 |
| | Drain | 35 | 1 2 4 5 |
| Second rinse | Fill | 36 | 1 3 4 5 |
| | Rinse | 37–41 | 1 4 5 6 |
| | Drain | 42 | 1 2 4 5 |
| Dry | Dry | 43–58 | 1 4 6 |
| | Drain | 59 | 1 2 4 6 |
| | Dry | 60 | 1 4 6 |

Dishwasher wiring diagram and timing chart.

# Sequencer Instructions

- PLC sequencer instructions replace the mechanical drum sequencer that is used to control machines that have a stepped sequence of repeatable operations.
- The advantage of PLC sequencer over mechanical sequencer are:
  - More flexibility
  - Complex operations can be performed with single or a pair of instruction
  - Using one ladder rung 16 discrete outputs can be controlled.

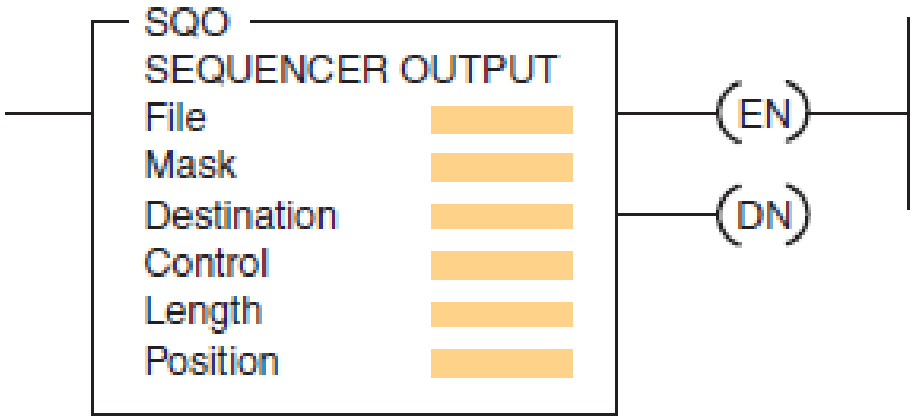


# Sequencer Instructions

- **SQO (Sequencer Output)** —Is an output instruction that uses a file to control various output devices.
- **SQI (Sequencer Input)** —Is an input instruction that compares bits from an input fi le to corresponding bits from a source address. The instruction is true if all pairs of bits are the same.
- **SQC (Sequencer Compare)** —Is an output instruction that compares bits from an input source file to corresponding bits from data words in a sequence file. If all pairs of bits are the same, then a bit in the control register is set to 1.
- **SQL (Sequencer Load)** —Is an output instruction used to capture reference conditions by manually stepping the machine through its operating sequences. It transfers data from the input source module to the sequencer file. The instruction functions much like a file-to-word transfer instruction.

# SQO (Sequencer Output) instruction

- The SQO instruction reads data file elements (words) one at a time,
- applies a mask word to enable or disable bits from the current data file element,
- and transfers the masked data file element to a designated output.



- **File** —Is the starting address for the registers in the sequencer file and you must use the indexed file indicator (#) for this address. The file contains the data that will be transferred to the destination address when the instruction undergoes a false-to-true transition. Each word in the fi le represents a position, starting with position 0 and continuing to the fi le length.
- **Mask** —Is the bit pattern through which the sequencer instruction moves source data to the destination address. In the mask bit pattern, a 1 passes values while a 0 blocks the data flow but the existing bit value remains the same. An ***h*** is placed behind the parameter to indicate that the mask is a hexadecimal number or a ***B*** to indicate binary notation. Decimal notation is entered without any indicator.
- **Source** —Is the address of the input word or file from which the SQC and SQL instruction obtains data for comparison or input to its sequencer fi le.

- **Destination** —Is the address of the output word or file to which the SQO moves the data from its sequencer file.
- **Control** —Is the address that contains the parameters with control information for the instruction. The control register stores the status byte of the instruction, the length of the sequencer file, and the instantaneous position in the file as follows:
  - The **enable bit (EN;** bit 15) is set by a false-to-true rung transition and indicates that the instruction is enabled. It follows the rung condition.
  - The **done bit (DN;** bit 13) is set after the last word in the sequencer file is transferred. On the next false-to-true transition of the rung with the done bit set, the position pointer is reset to 1.
  - The **error bit (ER;** bit 11) is set when the processor detects a negative position value, or a negative or zero length value.

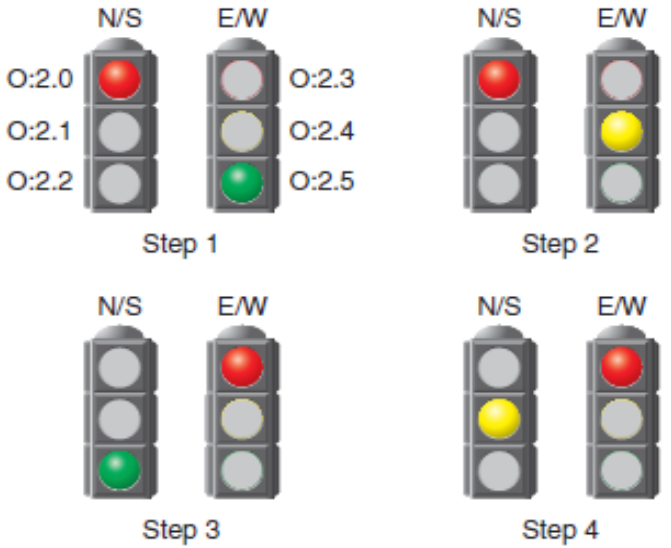
- **Length** —Is the number of steps of the sequencer file starting at position 1. Position 0 is the start-up position. The instruction resets (wraps) to position 1 at each cycle completion. The actual file length will be 1 plus the file length entered in the instruction.
- **Position** —Indicates the step that is desired to start the sequencer instruction. The position is the word location or step in the sequencer file from which the instruction moves data.

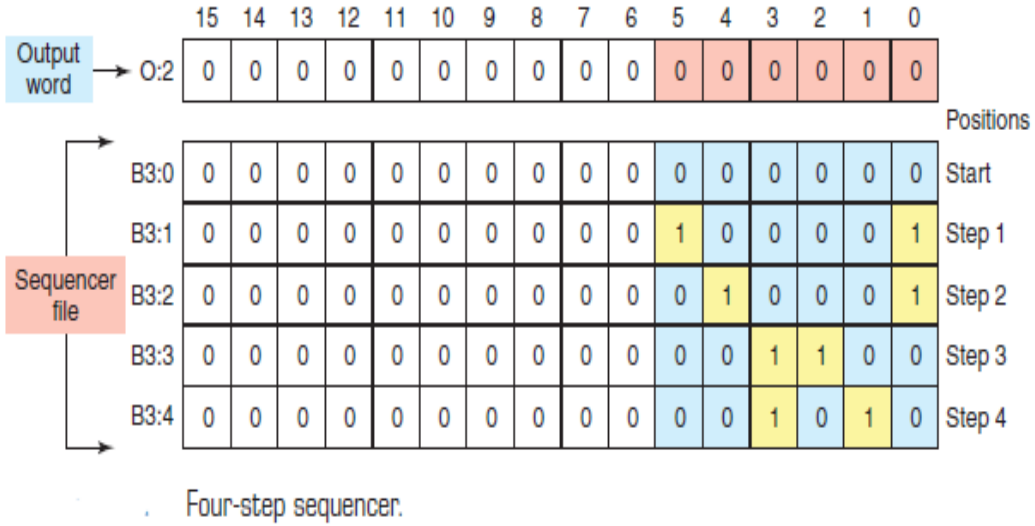
# Example 1: Control of traffic lights using sequencer

- This sequencer is to be used to control traffic in two directions.
- The operation of the process can be summarized as follows:
  - Six outputs are to be energized from one 16-point output module.
  - Each light is controlled by one bit address of output word O:2.
  - The first 6 bits are programmed to execute the following sequence of light outputs:

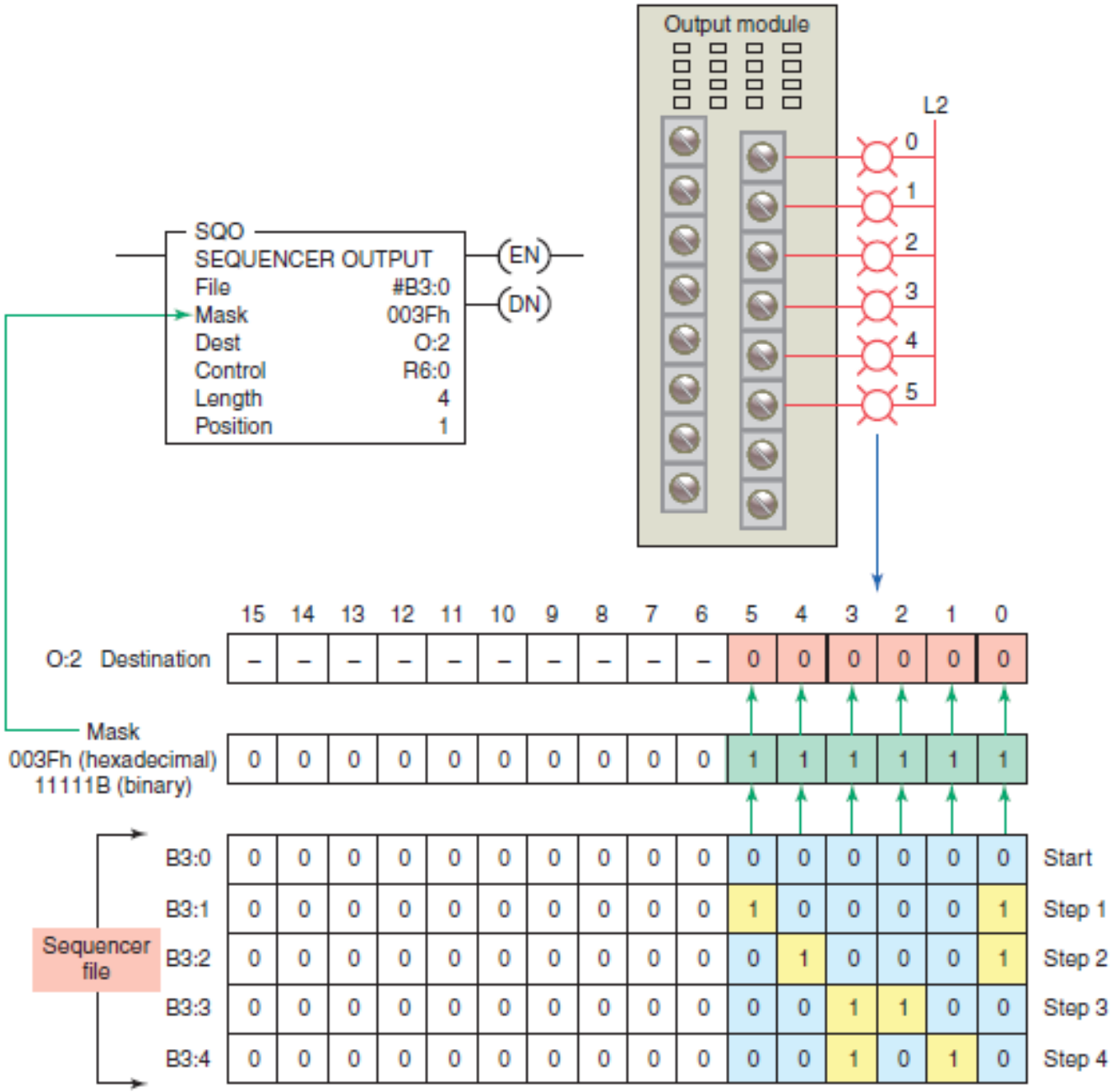
- - **Step 1:** Outputs O:2.0 (red) and O:2.5 (green) lights will be energized.
- - **Step 2:** Outputs O:2.0 (red) and O:2.4 (yellow) will be energized.
- - **Step 3:** Outputs O:2.2 (green) and O:2.3 (red) will be energized.
- - **Step 4:** Outputs O:2.1 (yellow) and O:2.3 (red) will be energized.

- Words B3:0, B3:1, B3:2, B3:3 and B3:4 make up the sequencer file.
- Binary information (1s and 0s) that reflects the desired on or off light status for each of the four steps is entered into each word of the sequencer file.
- Before starting the sequence, you need a starting point where the sequencer is in a neutral position. This is provided by the start position which is all zeros.



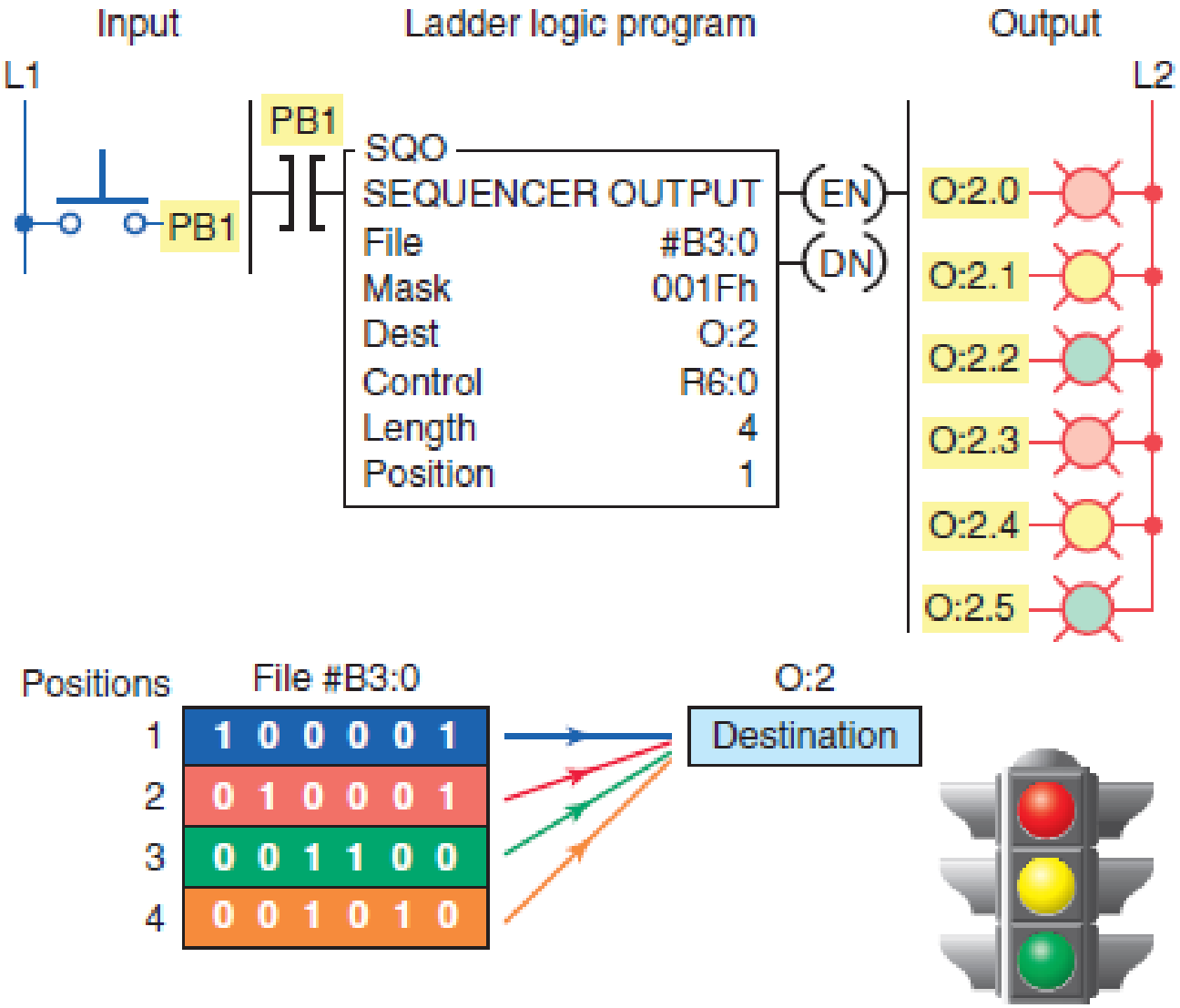




Four-step sequencer.


Output module
L2
0
1
2
3
4
5
SQO
SEQUENCER OUTPUT
File #B3:0
Mask 003Fh
Dest O:2
Control R6:0
Length 4
Position 1
(EN)
(DN)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
O:2 Destination
Mask
003Fh (hexadecimal)
11111B (binary)
Sequencer file
B3:0 Start
B3:1 Step 1
B3:2 Step 2
B3:3 Step 3
B3:4 Step 4

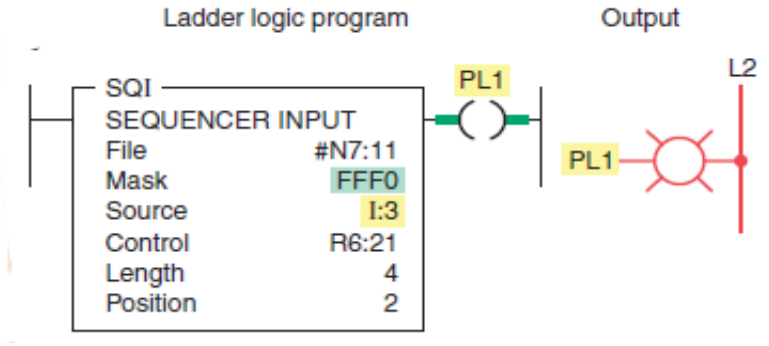


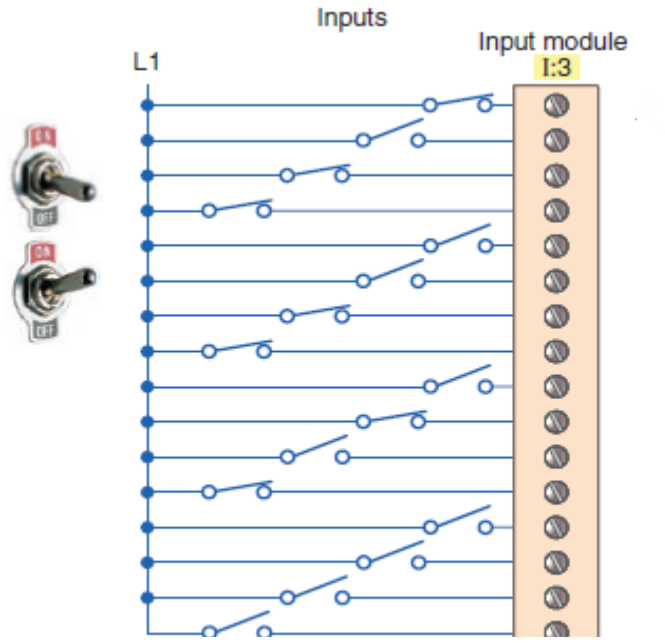
Indu

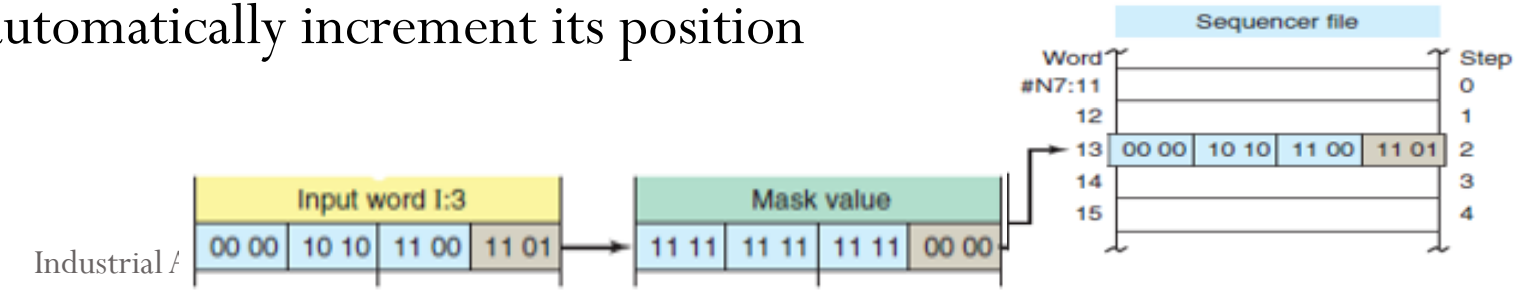

Input
Ladder logic program
Output
L1
L2
PB1
PB1
SQO
SEQUENCER OUTPUT
File #B3:0
Mask 001Fh
Dest O:2
Control R6:0
Length 4
Position 1
EN
DN
O:2.0
O:2.1
O:2.2
O:2.3
O:2.4
O:2.5
Positions
File #B3:0
O:2
Destination
1 1 0 0 0 0 1
2 0 1 0 0 0 1
3 0 0 1 1 0 0
4 0 0 1 0 1 0

# SQI (Sequencer Input)

- The sequencer *input* (SQI) instruction allows input data to be compared for equality against data stored in the sequencer file.
- For example, it can make comparisons between the states of input devices and their desired states: if the conditions match, the instruction is true.
- The SQI instruction uses a control register but does not have a done bit.
- The SQI instruction does not automatically increment its position

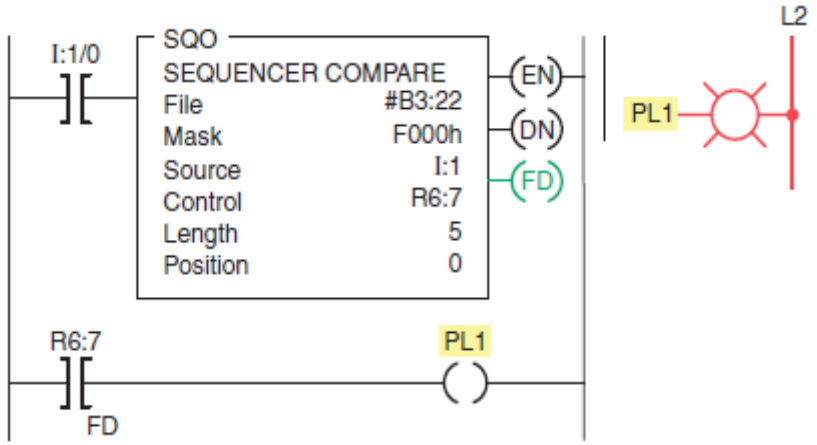


# SQC (Sequencer Compare)

- The sequencer compare (SQC) instruction is an output instruction used to compare bits from an input source file to corresponding bits from data words in a sequencer file.
- When the pairs of bits are the same, then the found(FD) bit in the control register is set to 1.
- This instruction can be used to compare the status of a machine's input devices with what is required for normal operation.
- When the status of the input devices on the machine (on or off)are identical to the data stored in the sequencer file, the control found bit is set to 1.

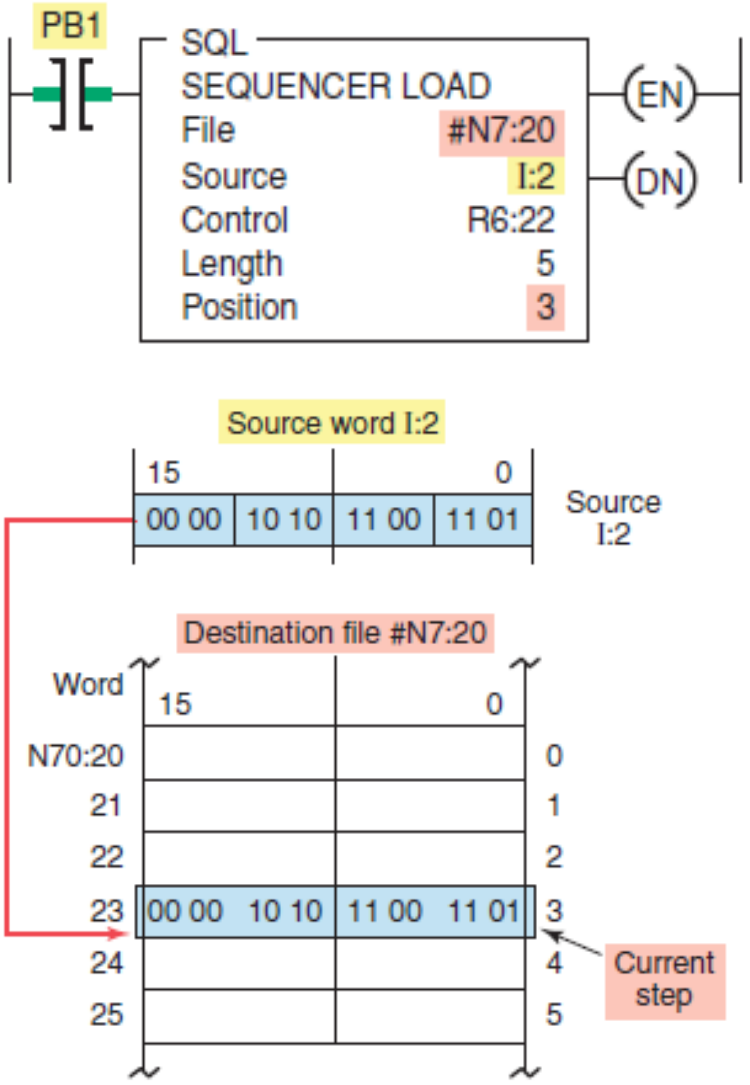


# SQI & SQC Instructions

- *Sequencer compare (SQC)* instructions are similar but not identical to the SQI instruction. Differences between the two include:
  - The SQC instruction is an output rather than an input instruction.
  - The SQC instruction increments the position parameter
  - The SQC instruction has an additional status bit — the *found bit (FD)*.
  - When the source pattern matches the sequencer file word the FD is set to 1. It is zero under all other conditions.

# SQL (Sequencer Load)

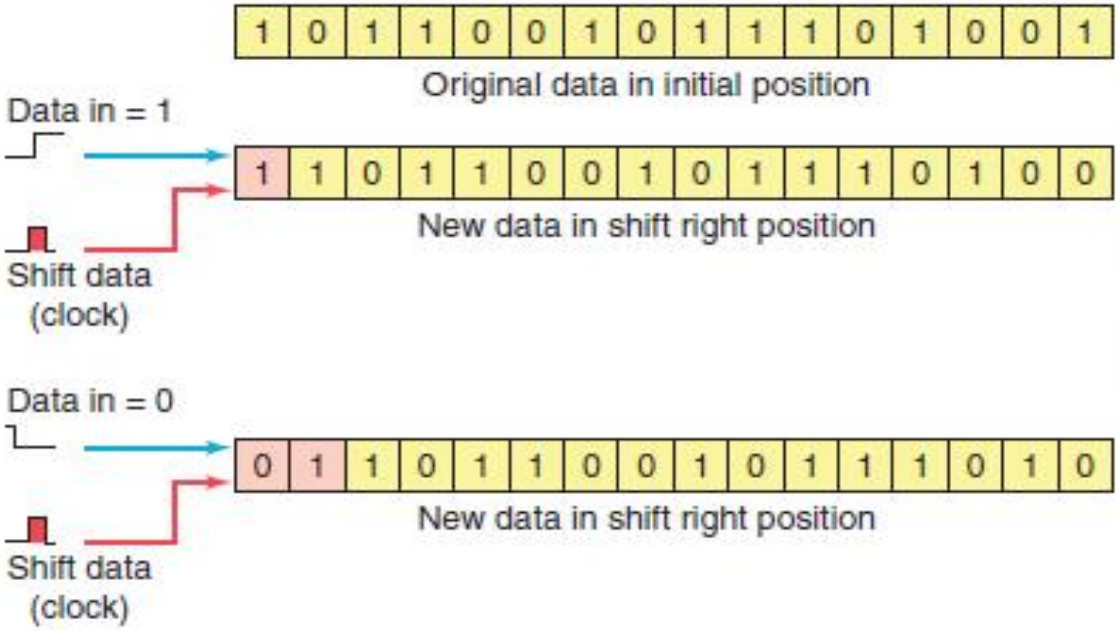
- The sequencer load (SQL) instruction is used to read the PLC input module and store the input data in the sequencer file.
- The sequencer load instruction is used to load the file and does not function during the machine's normal operation.
- It replaces the manual loading of data into the file with the programming terminal.
- The sequencer load instruction does not use a mask. It copies data directly from the source address to the sequencer file.
- When the instruction goes from false to true, the instruction indexes to the next position and copies the data.
- When the instruction has operated on the last position and has a true-to-false transition, it resets to position 1.
- It transfers data in position 0 only if it is at position0 and the instruction is true and the processor goes from the program to run mode.



# Bit Shift Registers

- The PLC not only uses a fixed pattern of register (word) bits, but also can easily manipulate and change individual bits.
- A ***bit shift register*** is a register that allows the shifting of bits through a single register or group of registers.
- The bit shift register shifts bits serially (from bit to bit) through an array in an orderly fashion.
- Common applications for shift registers include the following:
  - Tracking of parts through an assembly line
  - Controlling of machine or process operations
  - Inventory control
  - System diagnostics

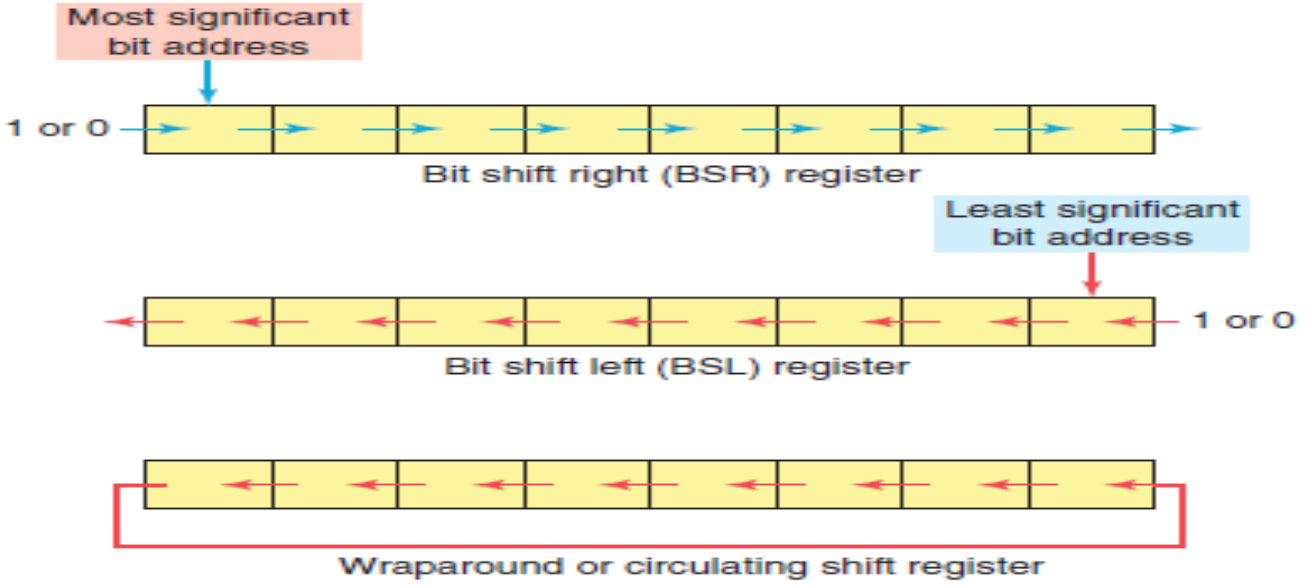
- Typically, data in the shift register could represent the following:
  - Part types, quality, and size
  - The presence or absence of parts
  - The order in which events occur
  - Identification numbers or locations
  - A fault condition that caused a shutdown




Basic concept of a shift register.

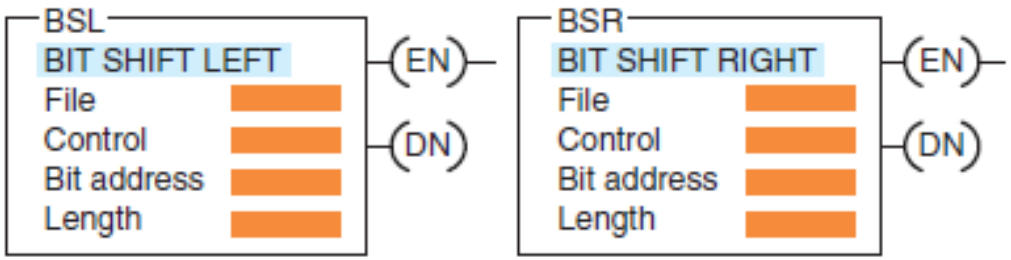
Tracking the absence of bottles

- There are two bit shift instructions:
- **BSL (Bit Shift Left)** —Loads a bit of data into a bit array, shifts the pattern of data through the array to the left, and unloads the last bit of data in the array.
- **BSR (Bit Shift Right)** —Loads a bit of data into a bit array, shifts the pattern of data through the array to the right, and unloads the last bit of data in the array.



Types of shift registers.

# BSL & BSL Instructions

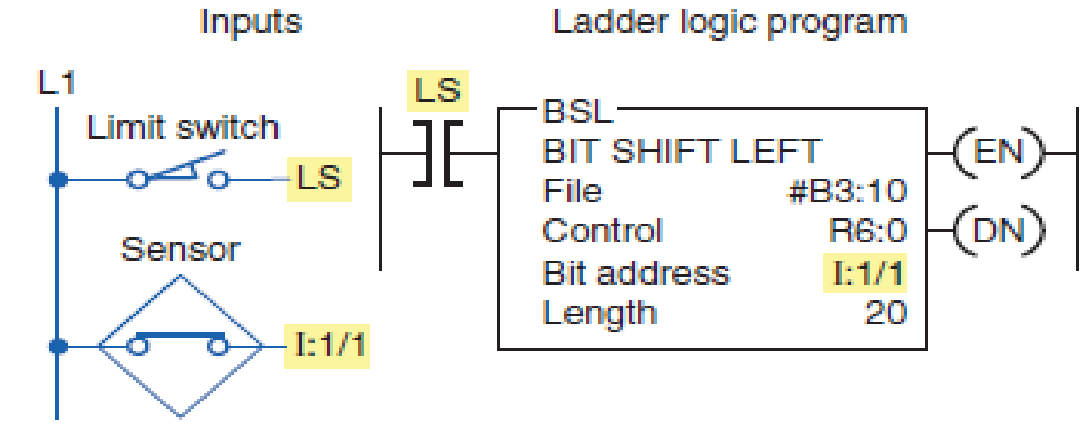


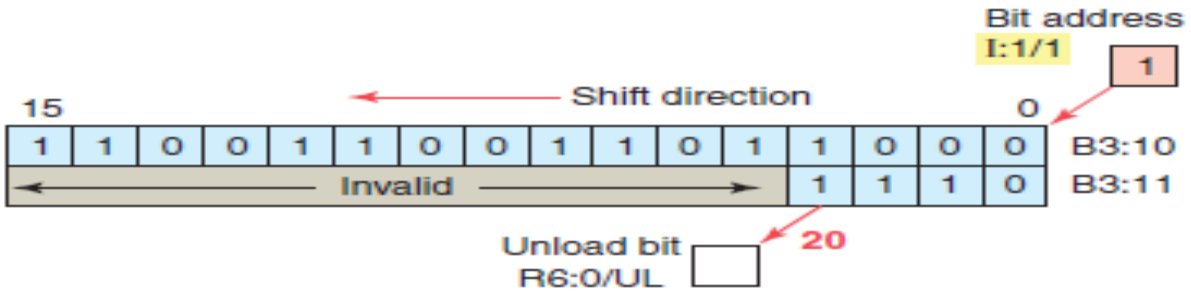

Bit shift left and bit shift right instructions.

- **File** —The address of the bit array you want to manipulate. The address must start with the # sign and at bit 0 of the first word or element.
- **Control** —R data-table type. The address is unique to the instruction and cannot be used to control any other instruction. It is a three-word element that consists of the status word, the length, and the position.

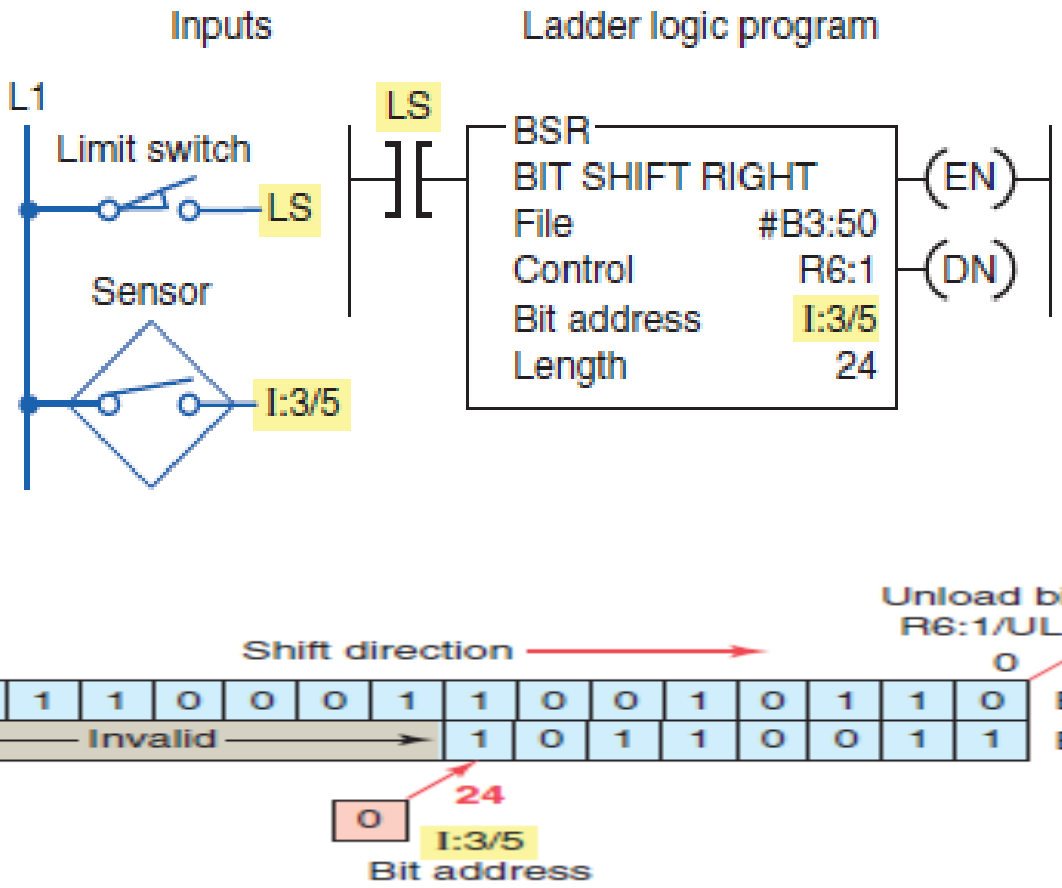
- **Bit address** —Is the address of the source bit. The instruction inserts the status of this bit in either the first (lowest) bit position (for the BSL instruction) or the last (highest) bit position (for the BSR instruction) in the array.
- **Length** —Indicates the number of bits to be shifted, or the file length, in bits. The status bits of the control word are the enable, done, error, and unload bits.
- Their functions can be summarized as follows:
- **Enable Bit (EN)** —The enable bit follows the instructions status and is set to 1 when the instruction is true.
- **D one Bit (DN)** —The done bit is set to 1 when the instruction has shifted all bits in the file one position. It resets to 0 when the instruction goes false.
- **Error Bit (ER)** —The error bit is set to 1 when the instruction has detected an error, which can happen when a negative number is entered in the length.
- **Unload Bit (UL)** —The unload bit's status is controlled by shifting of the last bit of the fi le into the unload bit when the instruction is executed.

# Bit Shift Left (BSL) Instruction



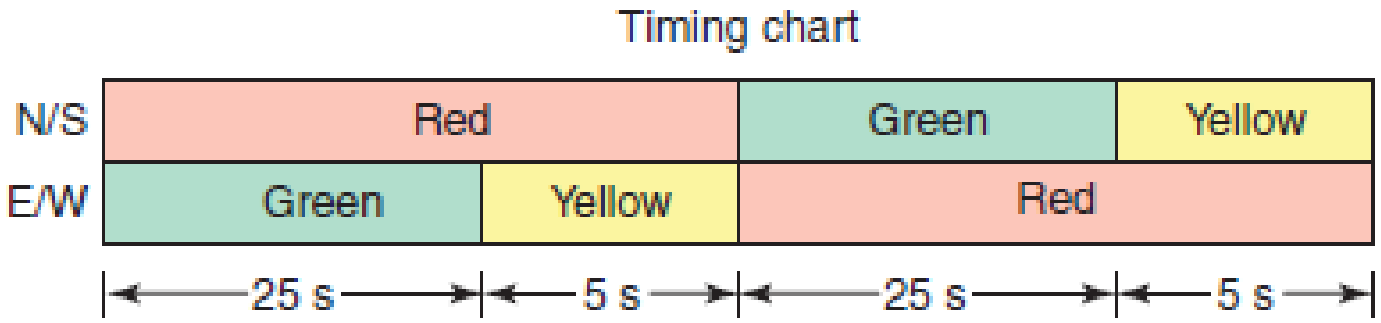



# Bit Shift Right (BSR) Instruction

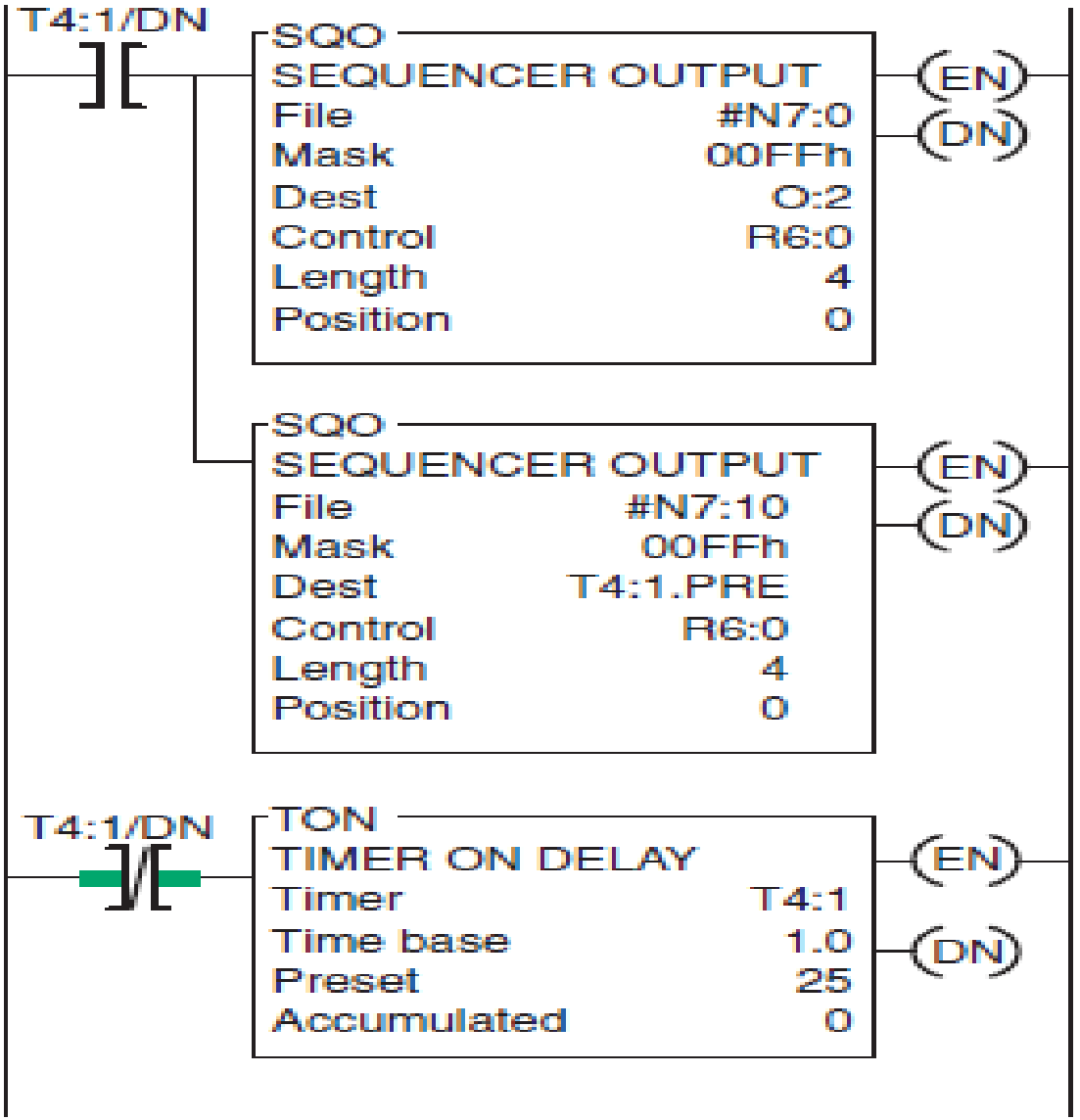


# Example 2: Time driven sequence

- Create a ladder logic to implement the traffic light control using sequencer. (modify the traffic control problem explained in Example 1). The timing chart for the lights are given as below.



Ladder logic program





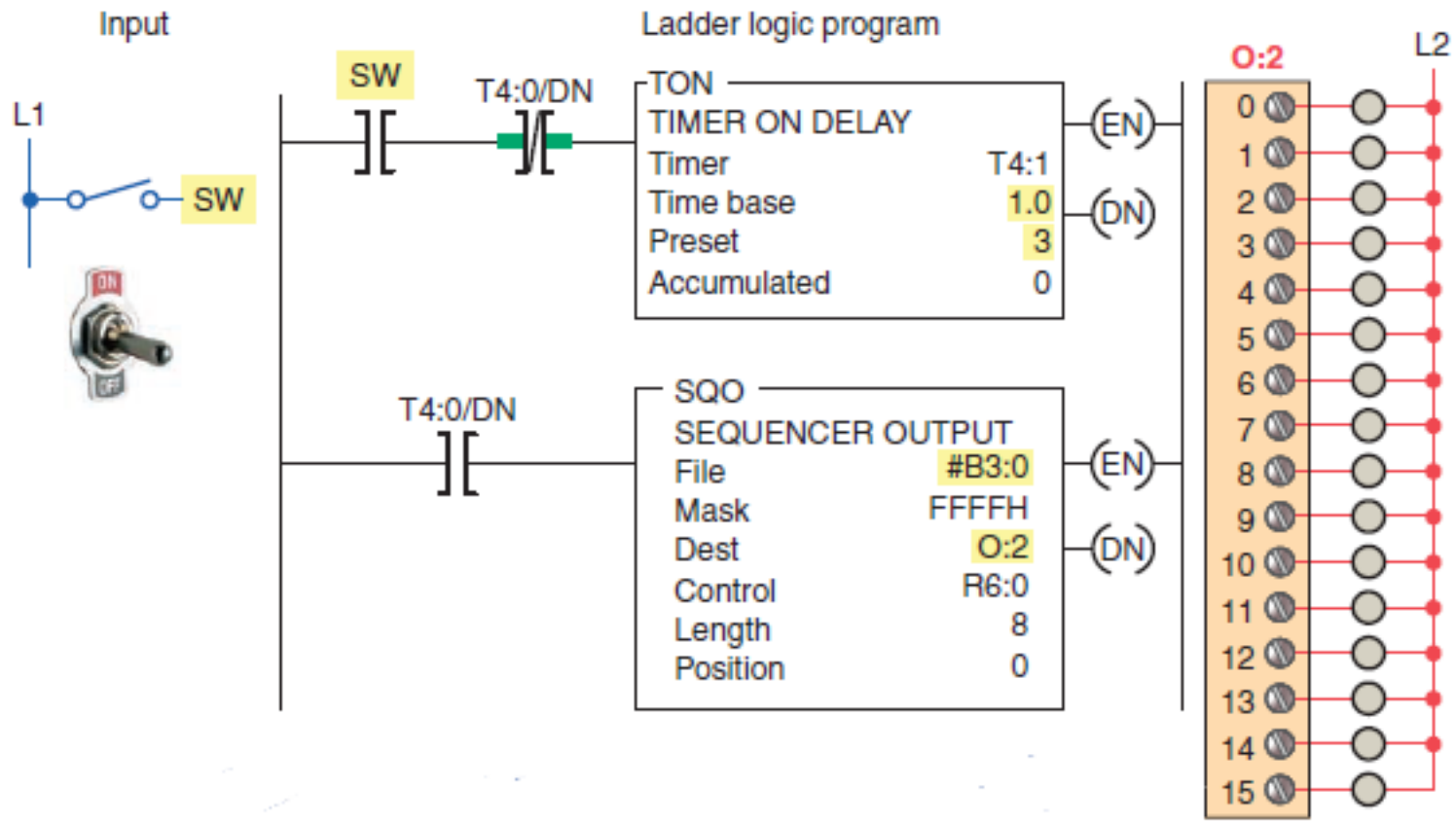
Sequencer file #N7:10 timer settings.

# Example 3: Time driven sequence

- Create a ladder logic to send the following sequence hen the switch SW1 in ON to the LED's connected to the output module O:2 with a time delay of 3s between each sequence.

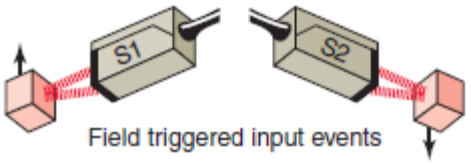
Binary Table — File #B3:0

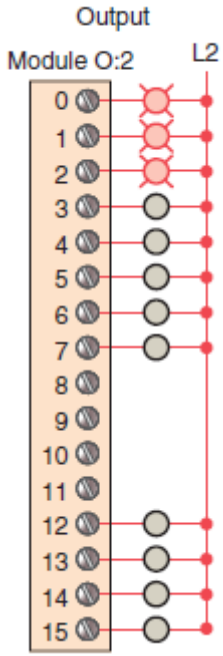
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B3:0/ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B3:/1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| B3:/2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| B3:/3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| B3:/4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| B3:/5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| B3:/6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| B3:/7 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| B3:/8 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |


Input
Ladder logic program
L1
SW
SW
T4:0/DN
TON
TIMER ON DELAY
Timer T4:1
Time base 1.0
Preset 3
Accumulated 0
(EN)
(DN)
T4:0/DN
SQO
SEQUENCER OUTPUT
File #B3:0
Mask FFFFH
Dest O:2
Control R6:0
Length 8
Position 0
(EN)
(DN)
O:2
L2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

# Example 4: Event driven sequence

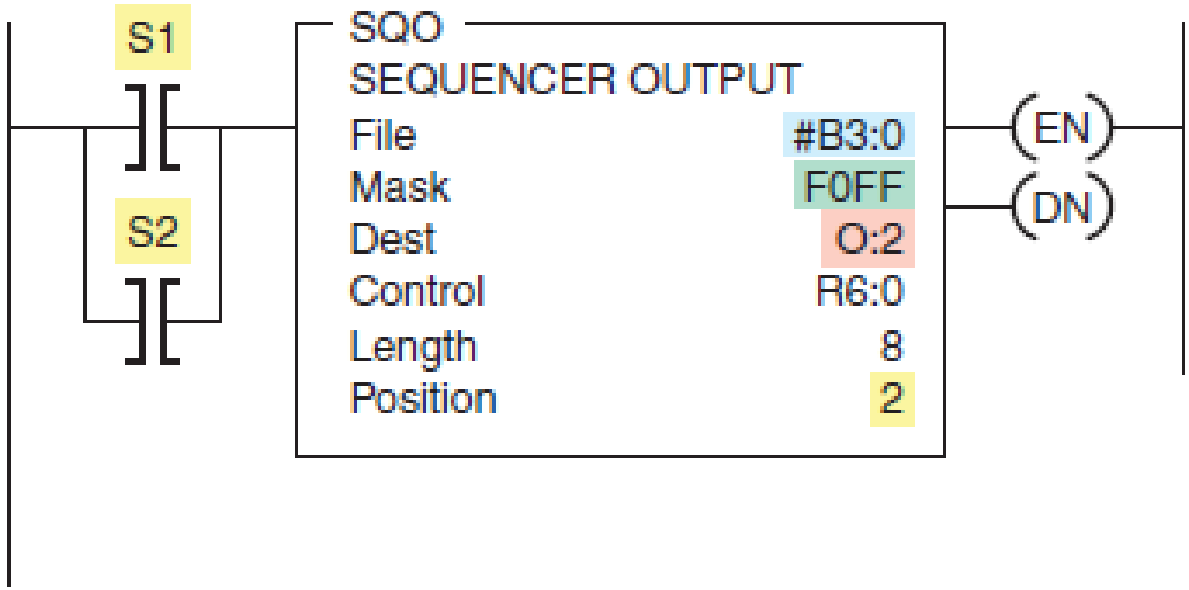
- Create a ladder logic to pass the sequence to the o/p module O:2 to which the LED's are connected as shown below. The sequence should pass only if any of the switches SW1 or SW2 are ON.



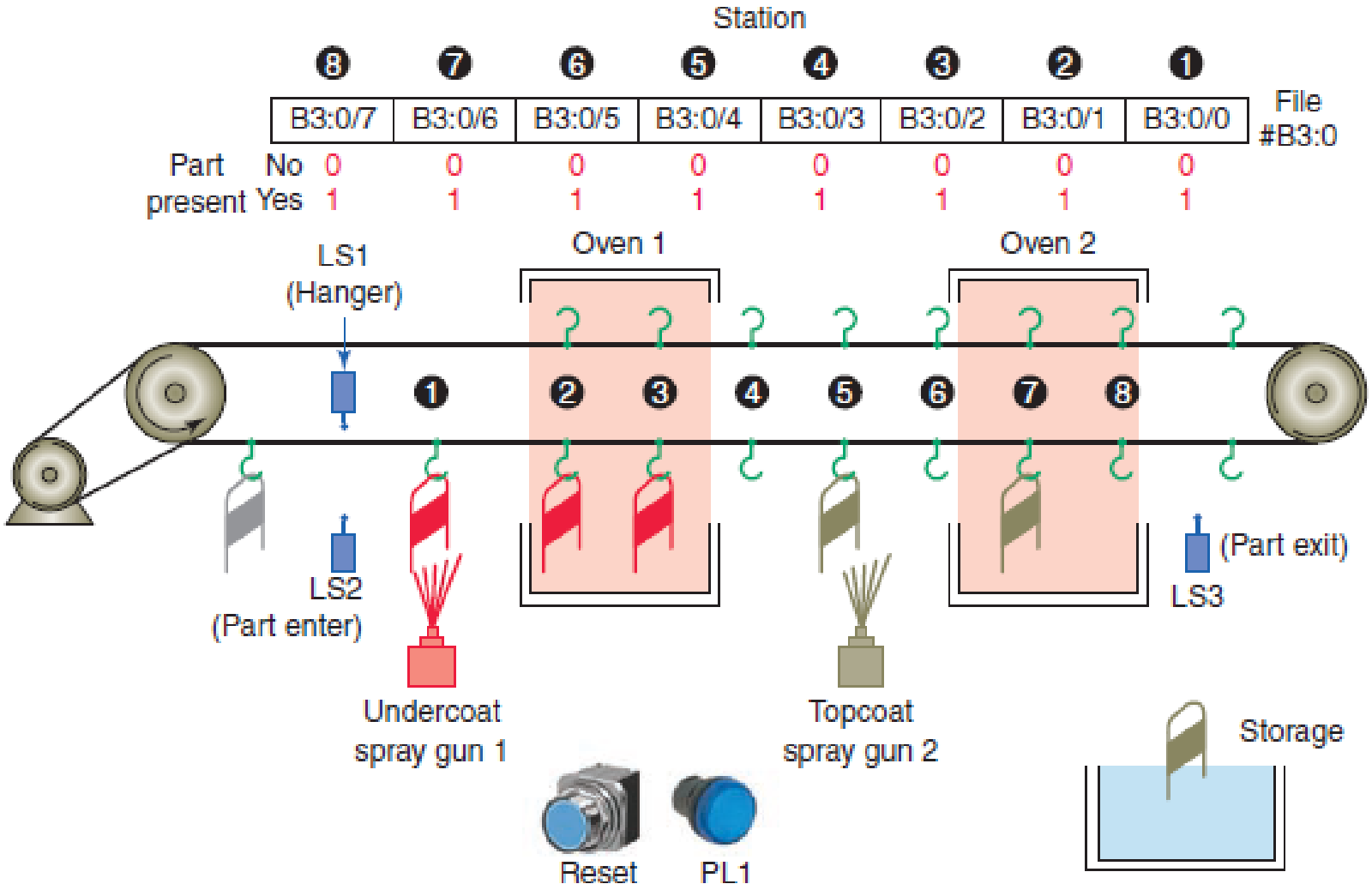

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 8 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

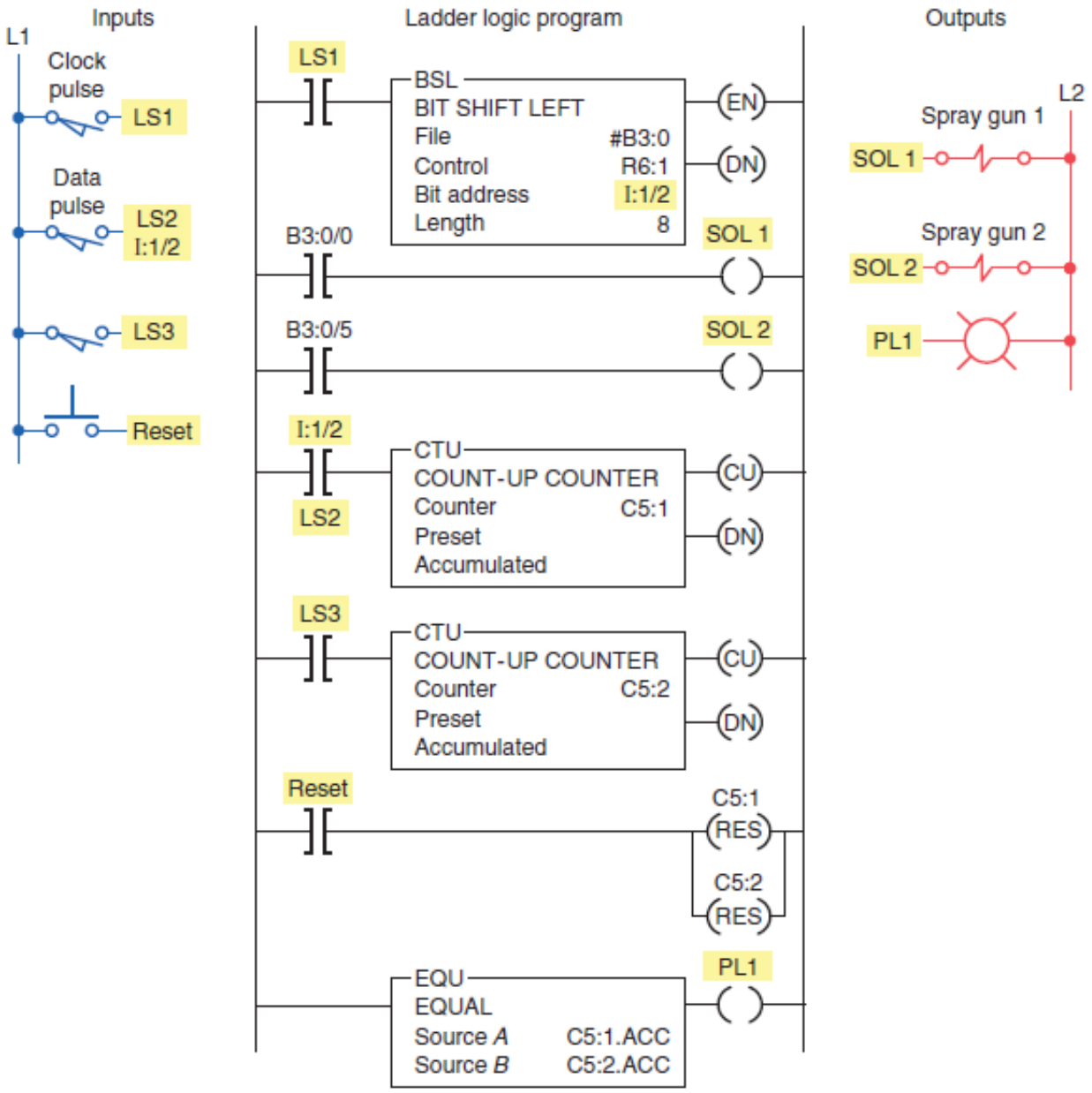
Ladder logic program



# Example 5: Application of shift register

- As the parts pass along the production line, the shift register bit patterns represent the items on the conveyor hangers to be painted.
- Each file bit location represents a station on the line, and the status of the bit indicates whether or not a part is present at that station.
- LS1 detects the presence of hanger, LS2 detects the part and LS3 detects the finished part.
- Whenever a part available at station 1, undercoat spray coating should happen.
- Whenever a part available at station 5, overcoat spray coating should happen.
- A pilot light PL1is turned ON when the parts commencing the spray painting run equal the parts that have completed it.
- Assume that the heaters are running continuously.
- The reset button resets the process.


Station
8
7
6
5
4
3
2
1
B3:0/7
B3:0/6
B3:0/5
B3:0/4
B3:0/3
B3:0/2
B3:0/1
B3:0/0
File #B3:0
Part present
No 0 0 0 0 0 0 0 0
Yes 1 1 1 1 1 1 1 1
LS1 (Hanger)
Oven 1
Oven 2
LS2 (Part enter)
LS3
(Part exit)
Undercoat spray gun 1
Topcoat spray gun 2
Storage
Reset
PL1


Inputs
L1
Clock pulse
LS1
Data pulse
LS2
I:1/2
LS3
Reset
Ladder logic program
LS1
BSL
BIT SHIFT LEFT
File #B3:0
Control R6:1
Bit address I:1/2
Length 8
EN
DN
B3:0/0
SOL 1
B3:0/5
SOL 2
I:1/2
LS2
CTU
COUNT-UP COUNTER
Counter C5:1
Preset
Accumulated
CU
DN
LS3
CTU
COUNT-UP COUNTER
Counter C5:2
Preset
Accumulated
CU
DN
Reset
C5:1
RES
C5:2
RES
EQU
EQUAL
Source A C5:1.ACC
Source B C5:2.ACC
PL1
Outputs
L2
Spray gun 1
SOL 1
Spray gun 2
SOL 2
PL1

# Reference


- Frank D. Petruzella, *Programmable Logic Controllers,* MGH, (2e), 1997.