

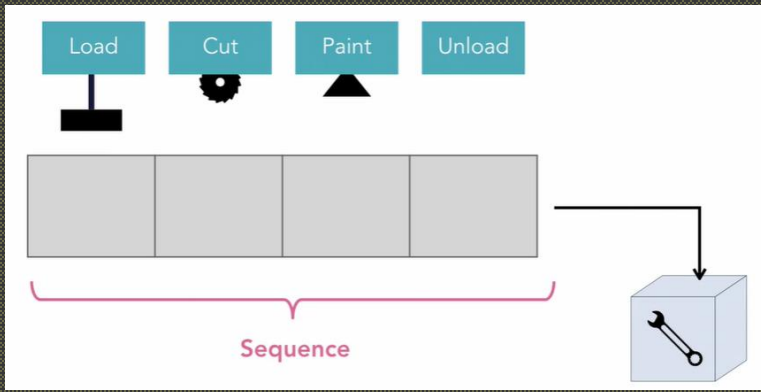
PROGRAMMABLE LOGIC CONTROLLERS MENG 3500



SEQUENTIAL CONTROL AND SEQUENCERS

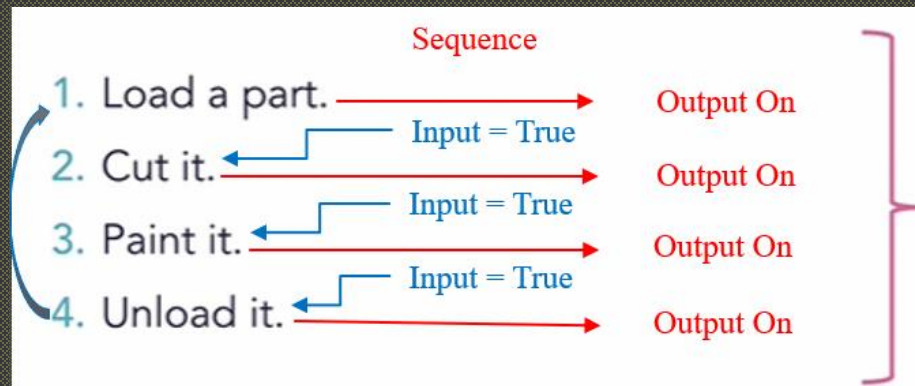
Cascading Sequence

The cascading sequence is a programming technique that allows the programmer to program and troubleshoot any sequential process with discrete output controlled by discrete inputs.



For example:

- a part is loaded in a conveyer,
- it is cut as per the design,
- it is painted, and
- unloaded into a box,
- these steps may repeat many times as per the number of parts required in the box,
- each step depends on the previous one.

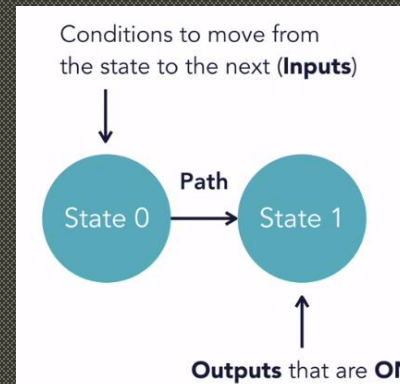
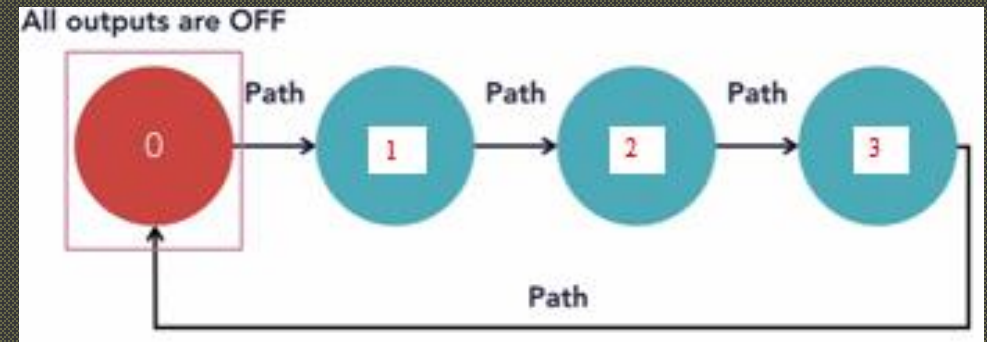
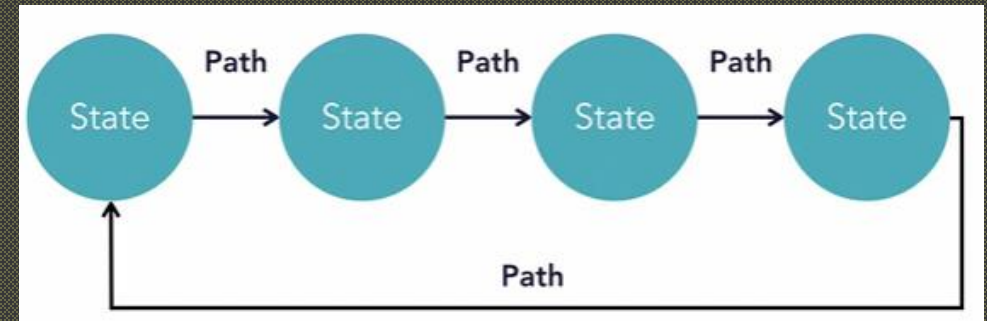
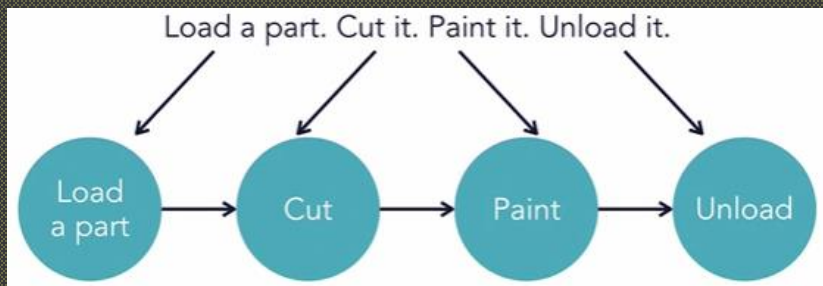
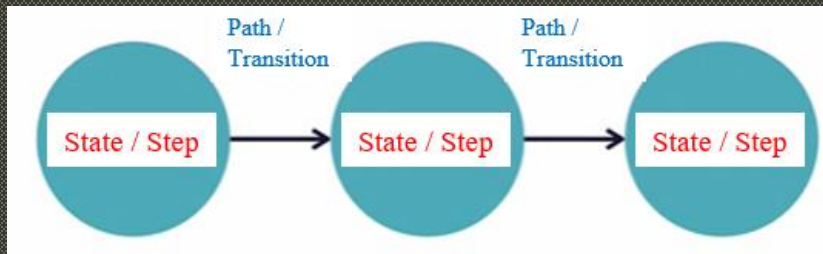


SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – State Diagram

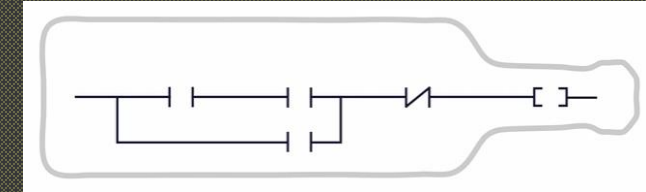
State Diagram is a method to visualize the sequence

- 1 Load a part. ← State / Step
- 2 Cut it. ← State / Step
- 3 Paint it. ← State / Step
- 4 Unload it. ← State / Step

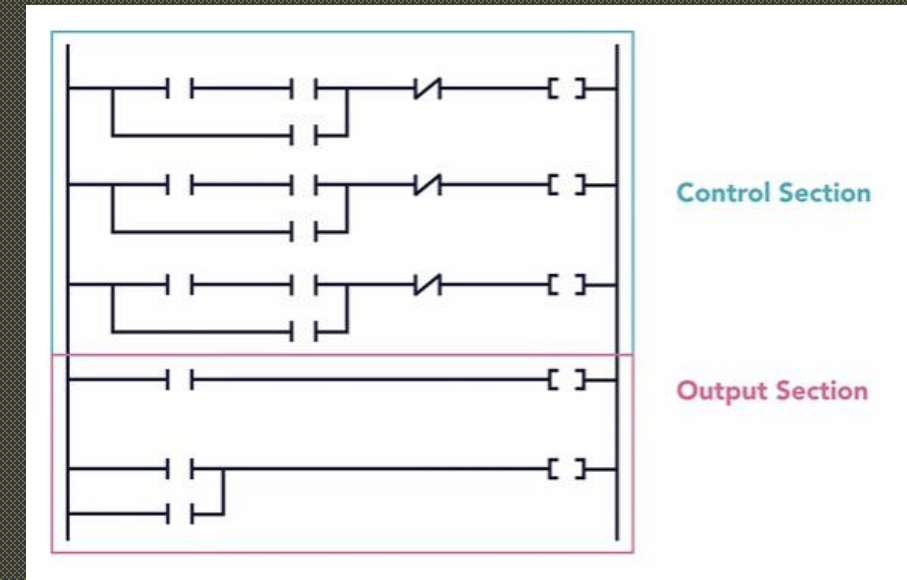
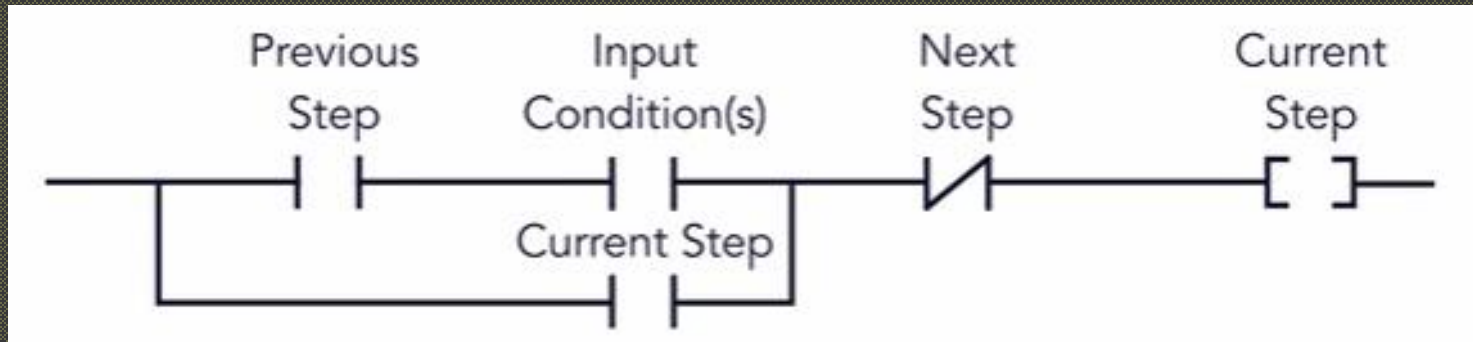


SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Bottle Logic



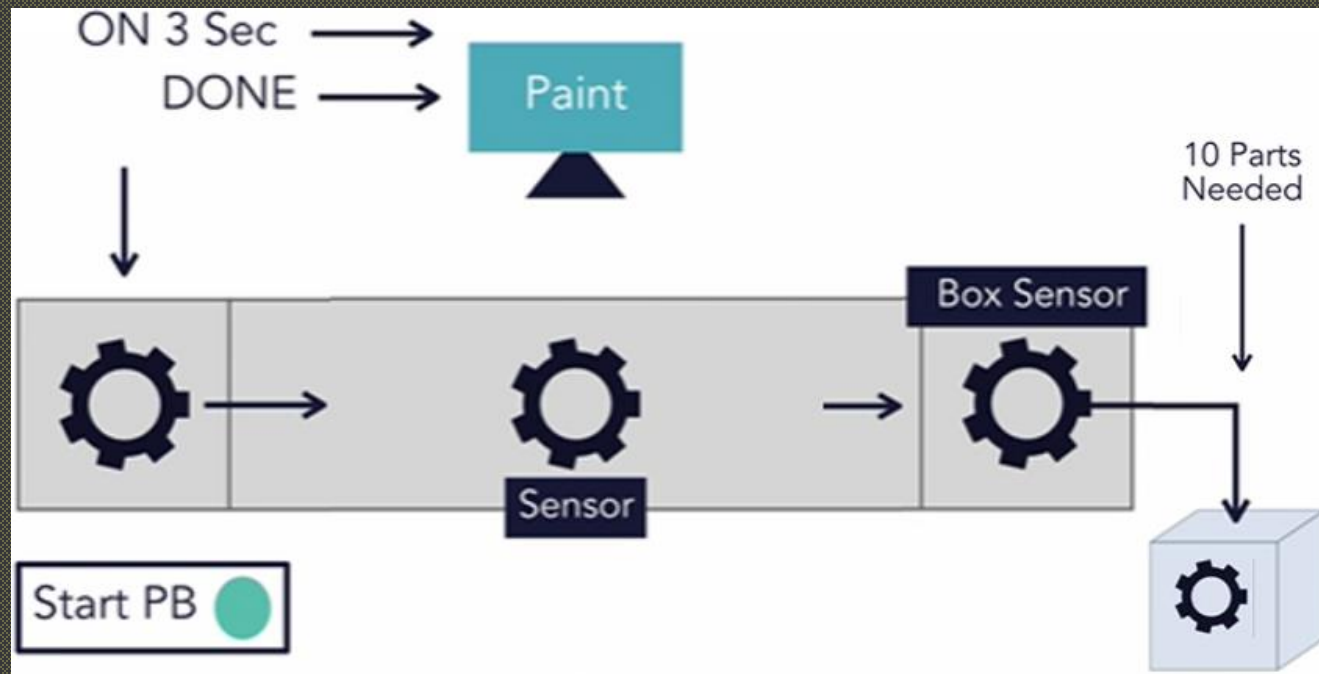
General Format:



SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1

1. Press the start pushbutton to start the conveyor.
2. The sensor detects a part, then the conveyor turns off.
3. The paint turns on for three seconds.
4. After three seconds are done, the conveyor turns ON again.
5. The box sensor detects the part and it falls into its designated box.
6. The sequence repeats until ten parts are painted.



SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1

Inputs

Start Pushbutton

Part Sensor

Box Sensor

1. Press the start pushbutton to start the conveyor.
2. The sensor detects a part, then the conveyor turns off.
3. The paint turns on for three seconds.
4. After three seconds are done, the conveyor turns ON again.
5. The box sensor detects the part and it falls into its designated box.
6. The sequence repeats until ten parts are painted.

Outputs

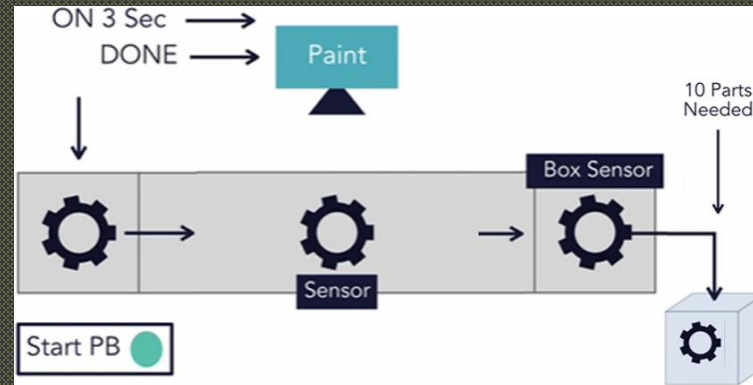
Conveyor ON

Paint ON

Timer ON

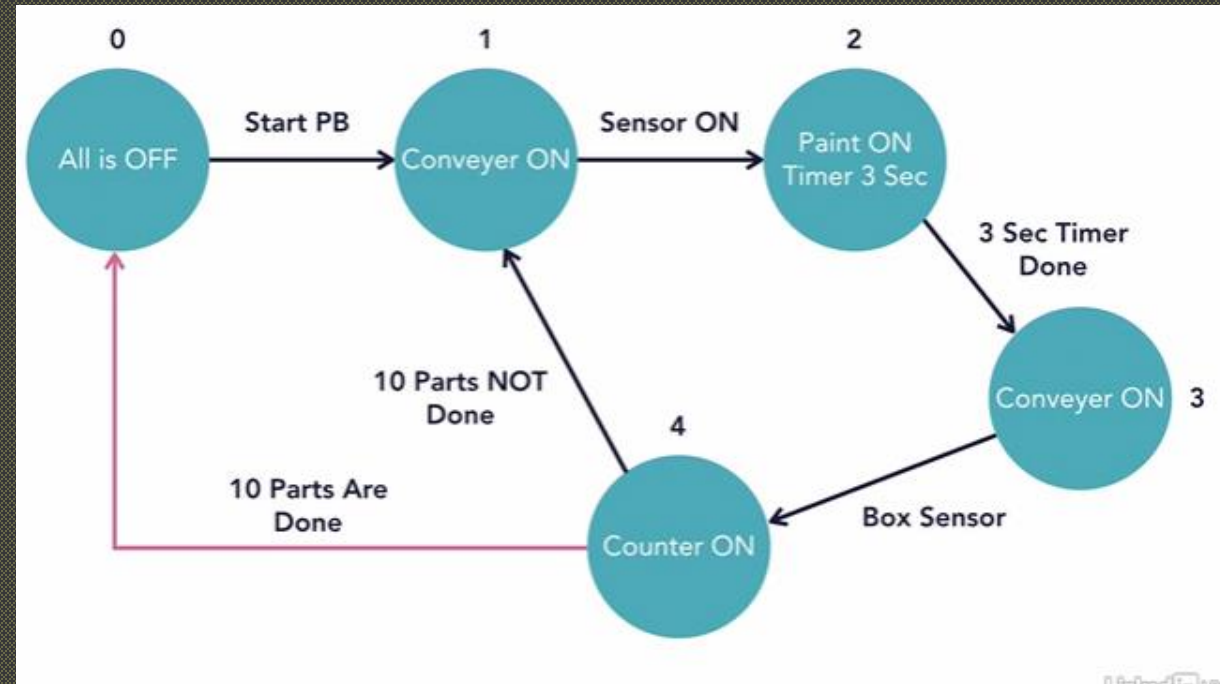
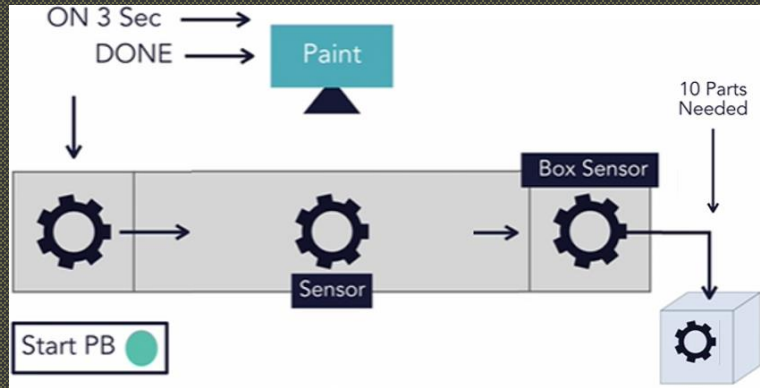
Counter ON

1. Press the start pushbutton to start the conveyor.
2. The sensor detects a part, then the conveyor turns off.
3. The paint turns on for three seconds.
4. After three seconds are done, the conveyor turns ON again.
5. The box sensor detects the part and it falls into its designated box.
6. The sequence repeats until ten parts are painted.



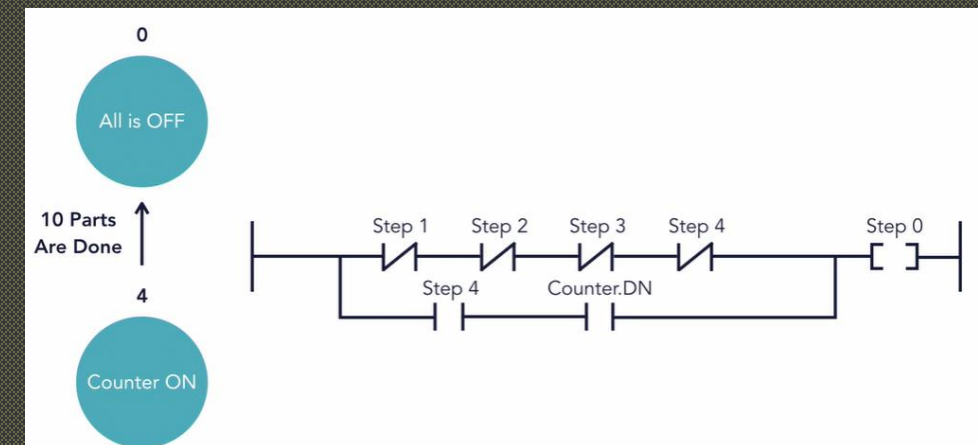
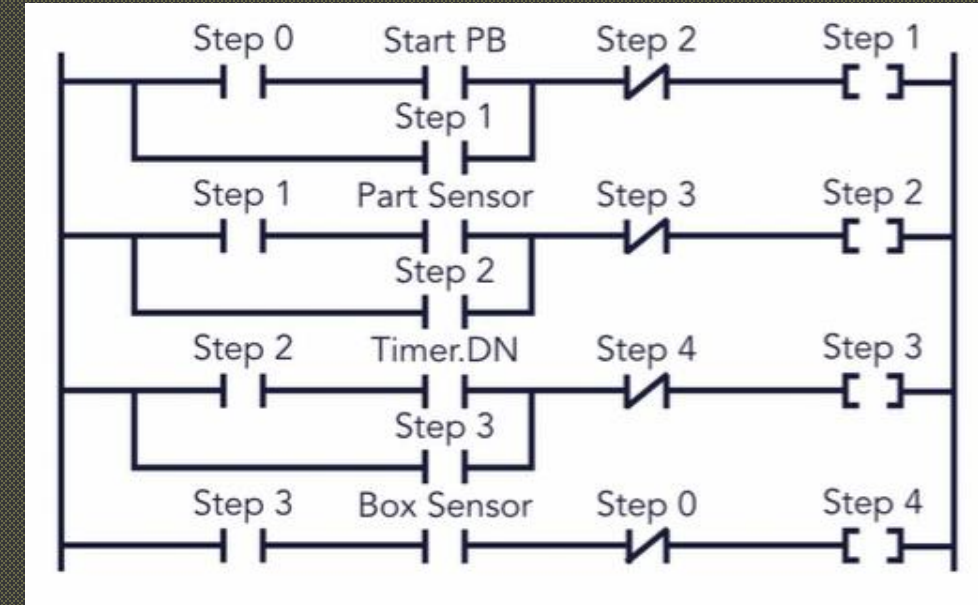
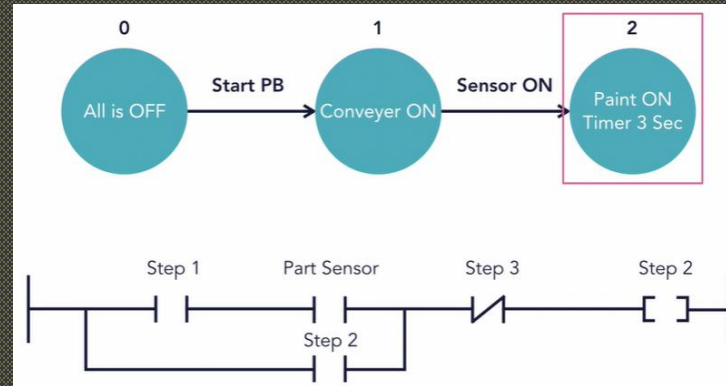
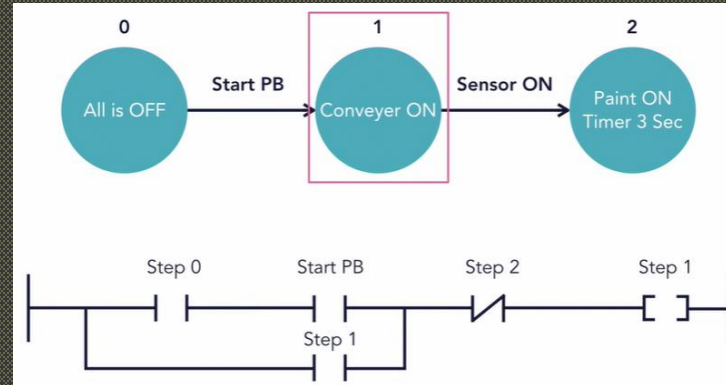
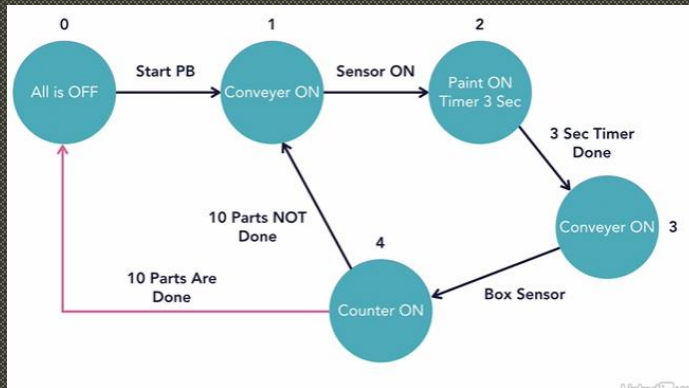
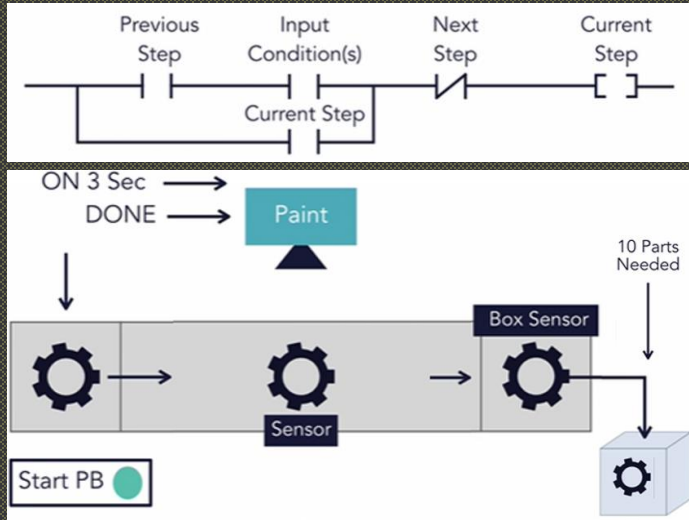
SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1



SEQUENTIAL CONTROL AND SEQUENCERS

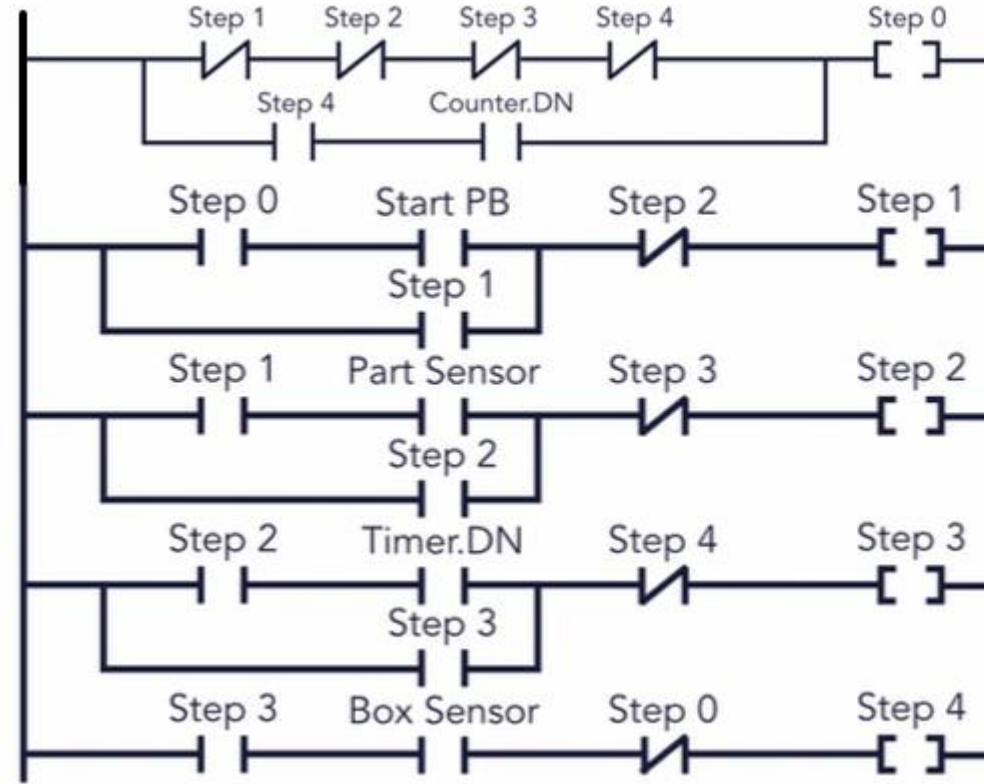
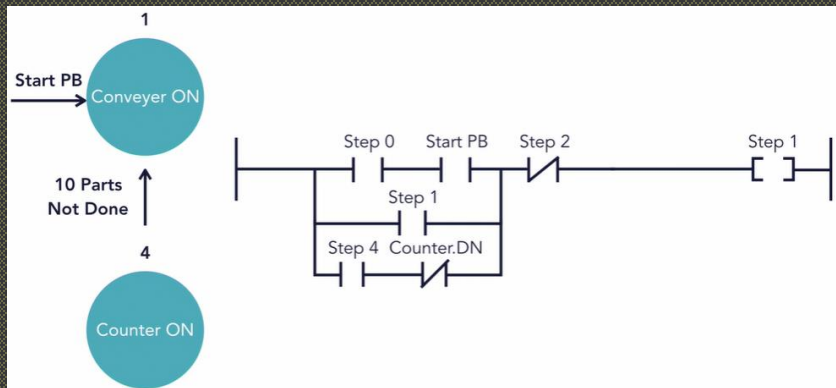
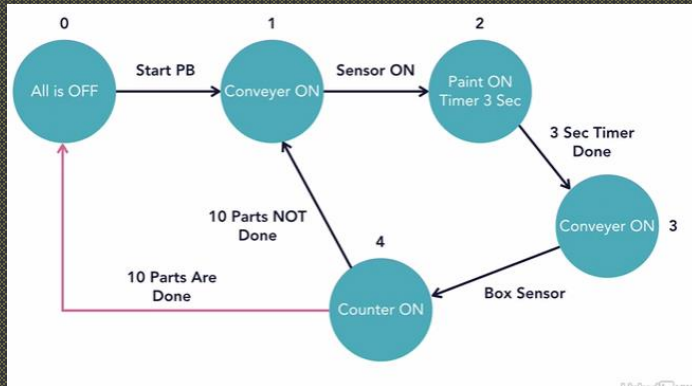
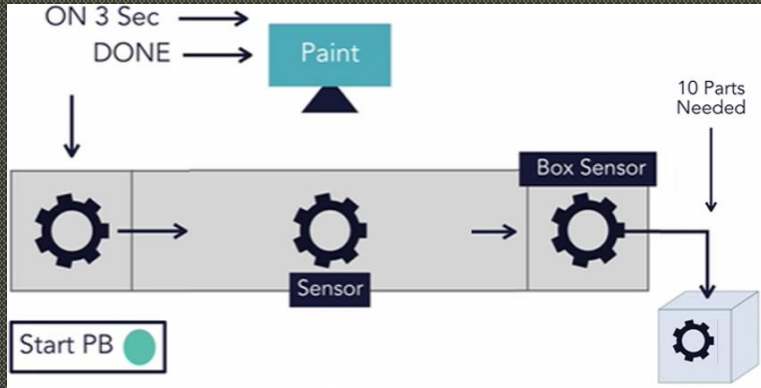
Cascading Sequence – Example 1



The sealing part for step 4 is not included because the counter is not done yet and we may need to jump to step 1, instead of step 0.

SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1



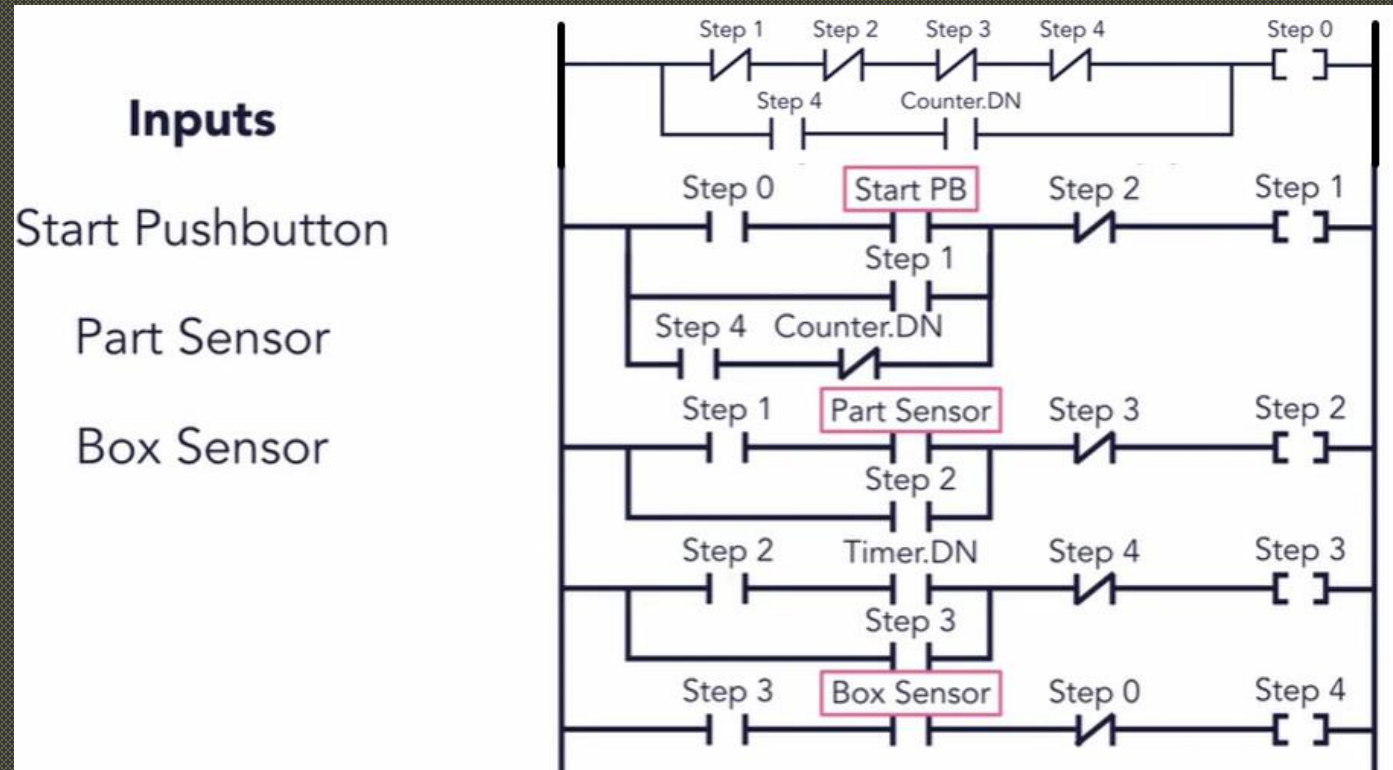
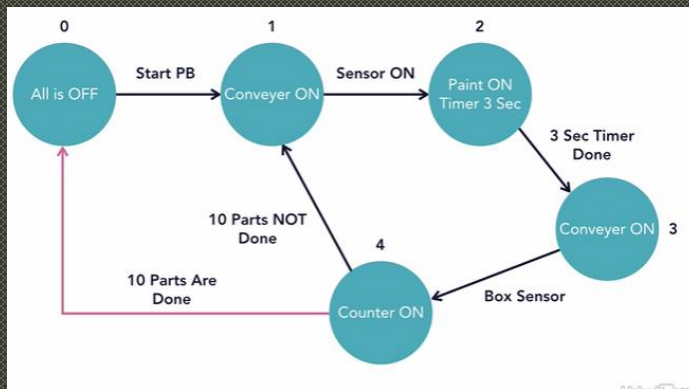
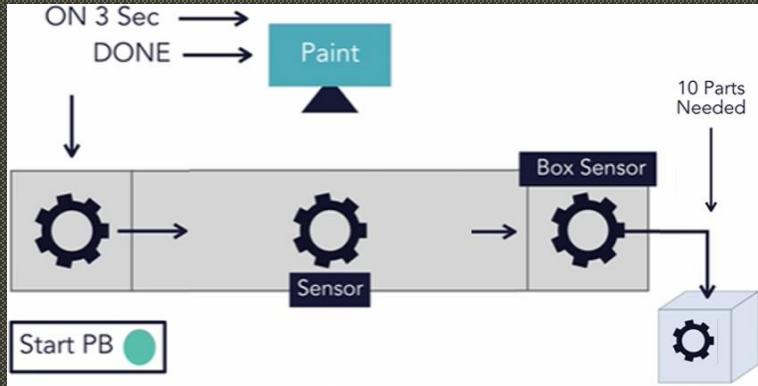
Control Section

Step 1 can be energized in two ways:

- By pressing the Start push button (PB).
- By Step 4 being energized while the counter is not yet complete.

SEQUENTIAL CONTROL AND SEQUENCERS

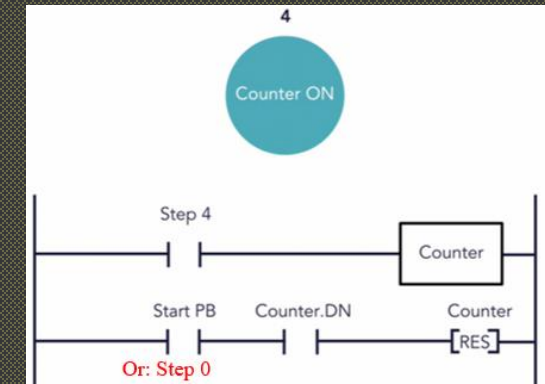
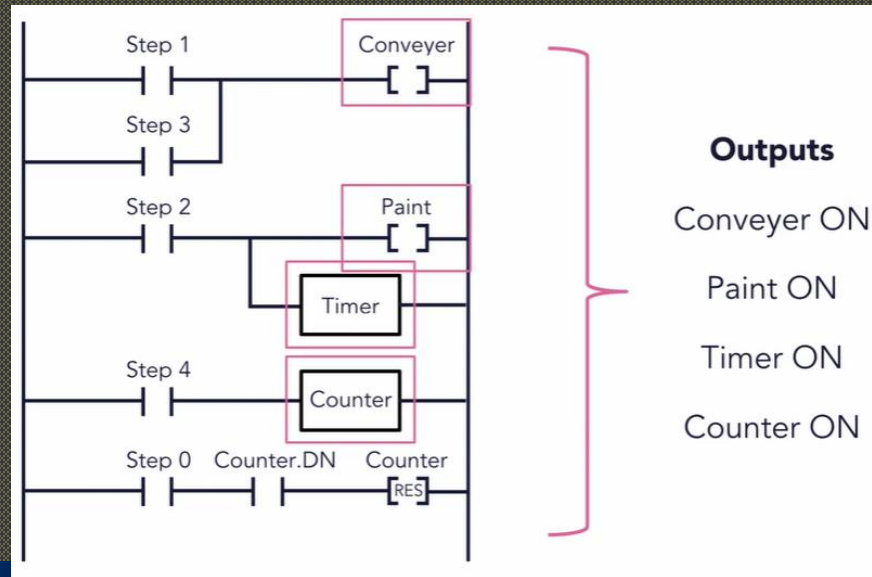
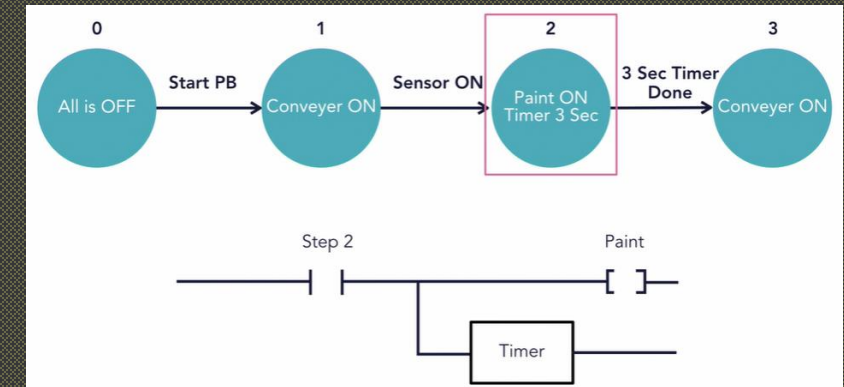
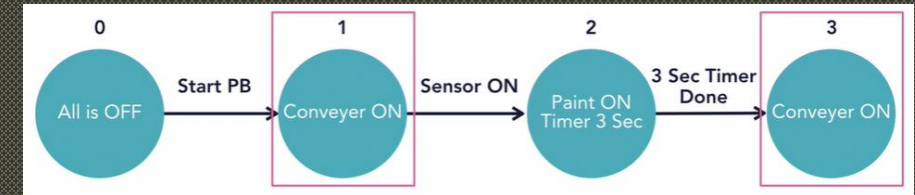
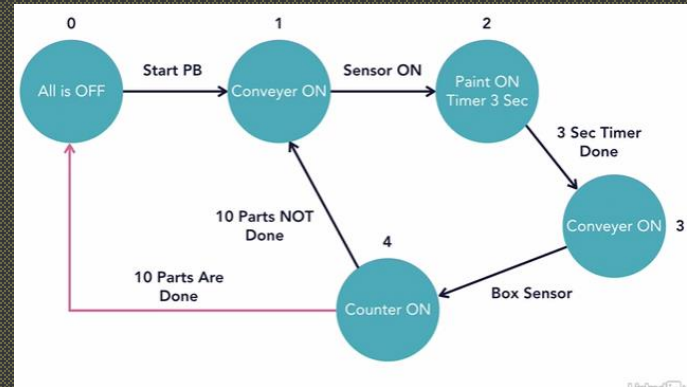
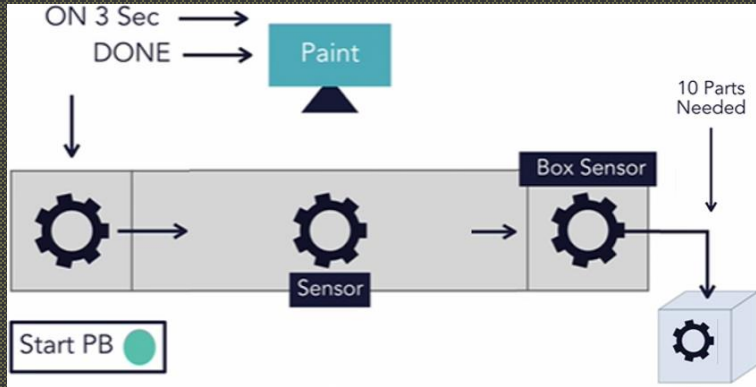
Cascading Sequence – Example 1



- Internal bits commonly used for the addresses of the steps (Boolean)

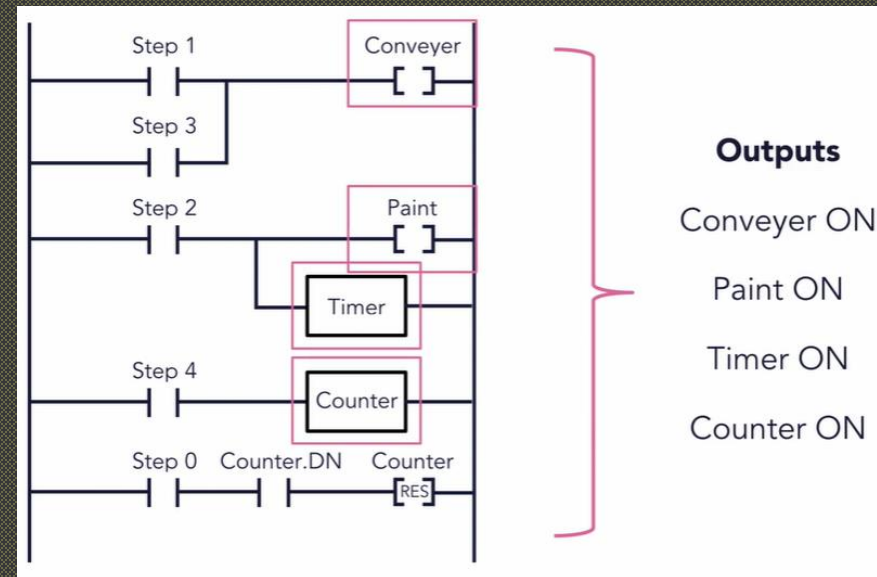
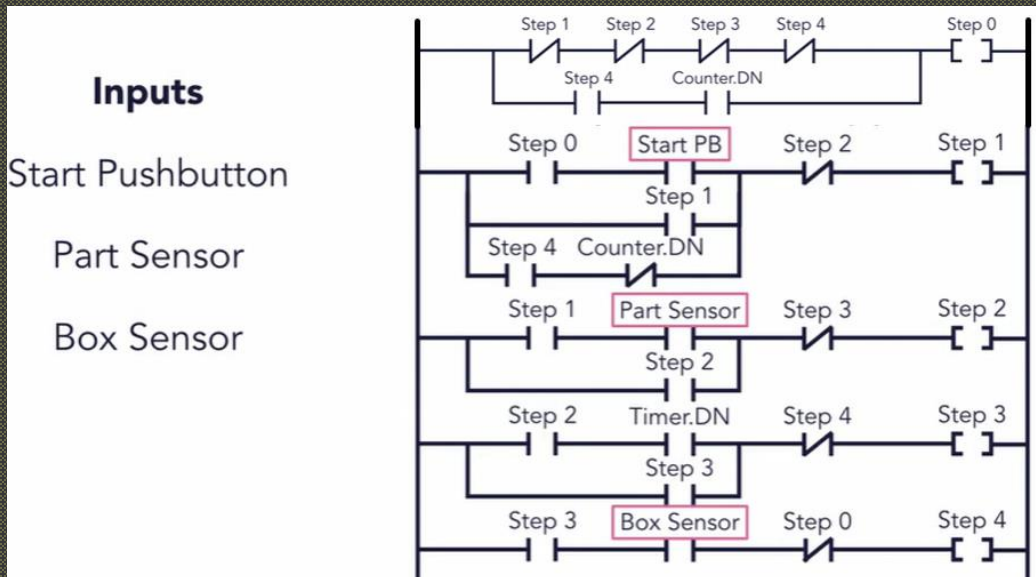
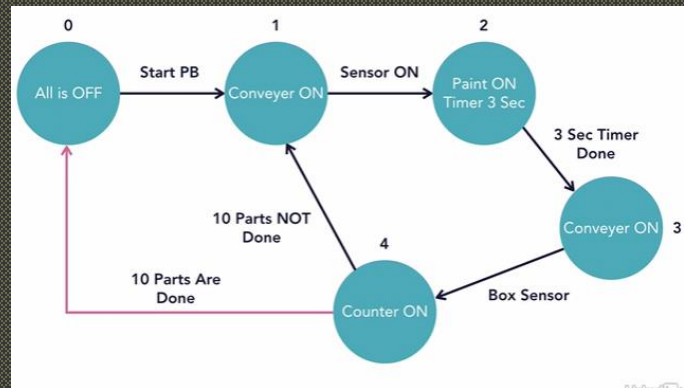
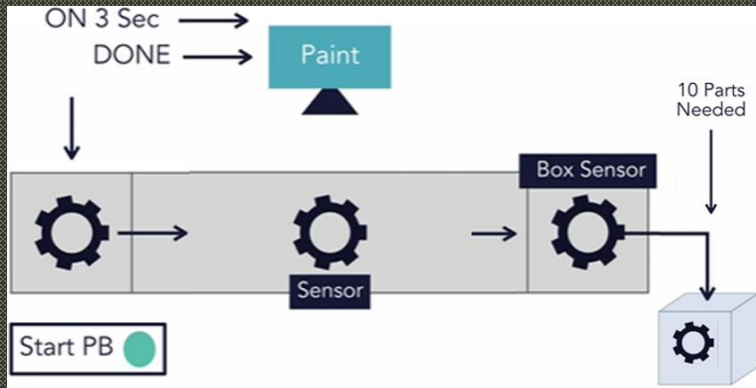
SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1



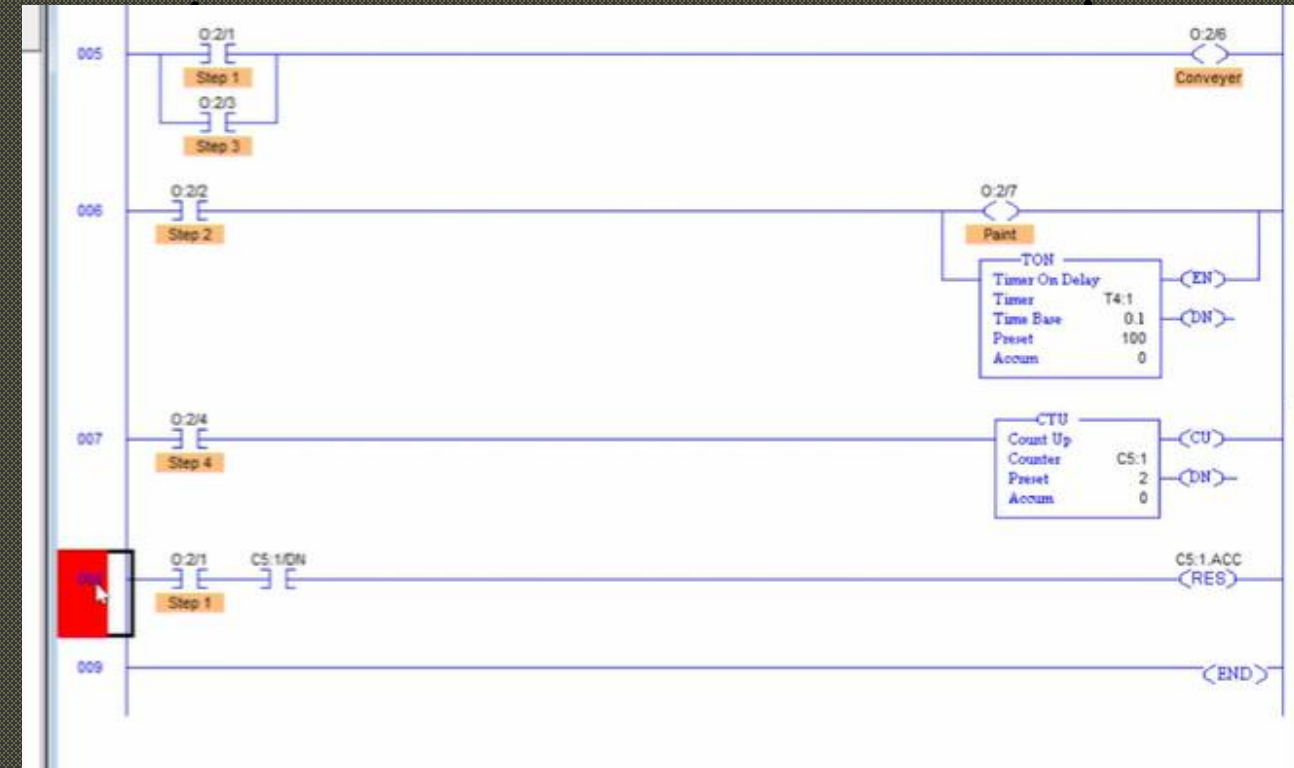
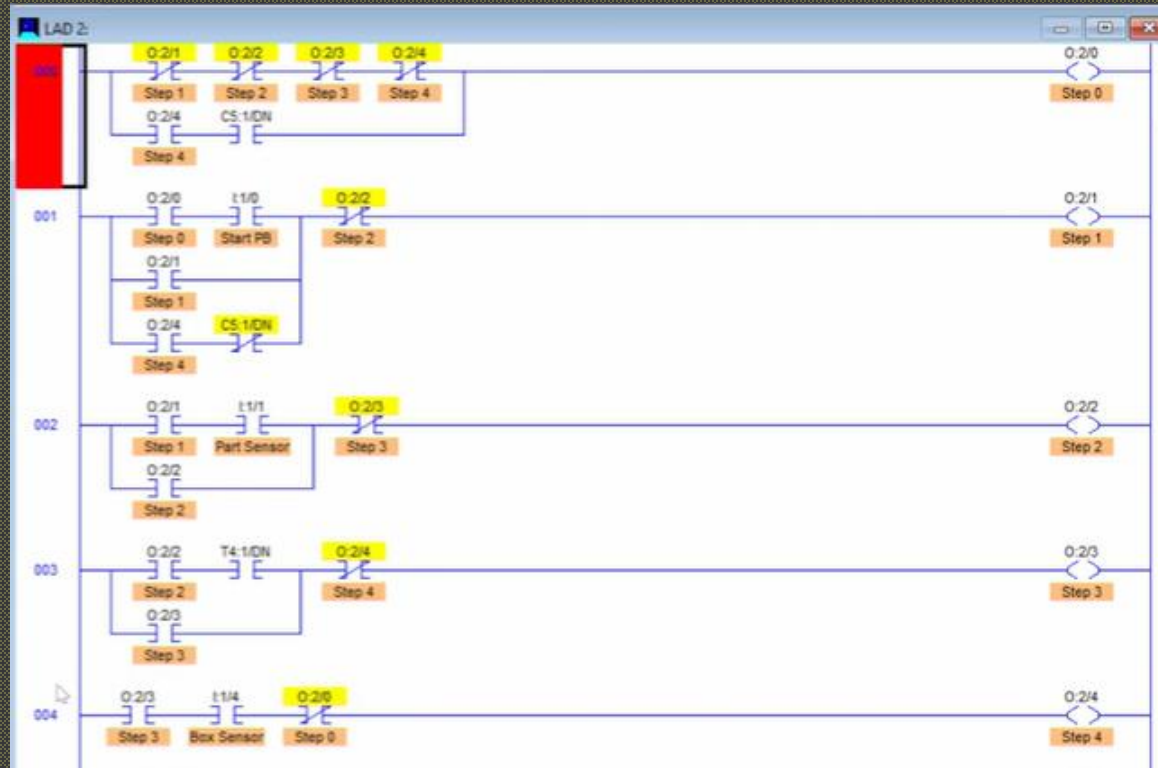
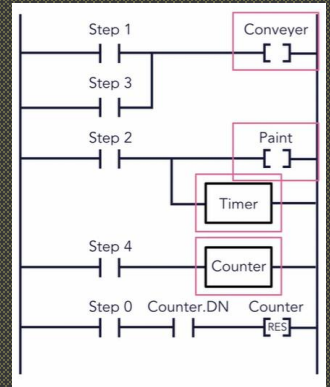
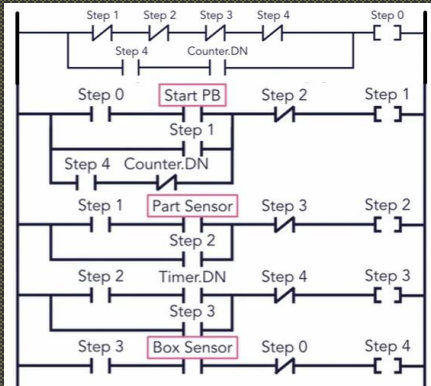
SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1



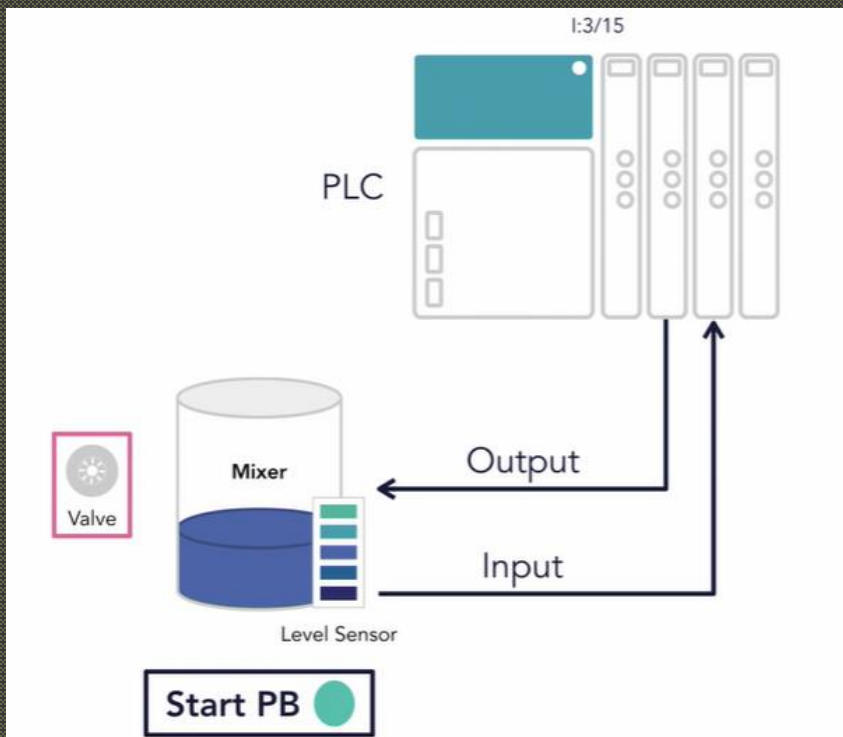
SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 1



SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 2



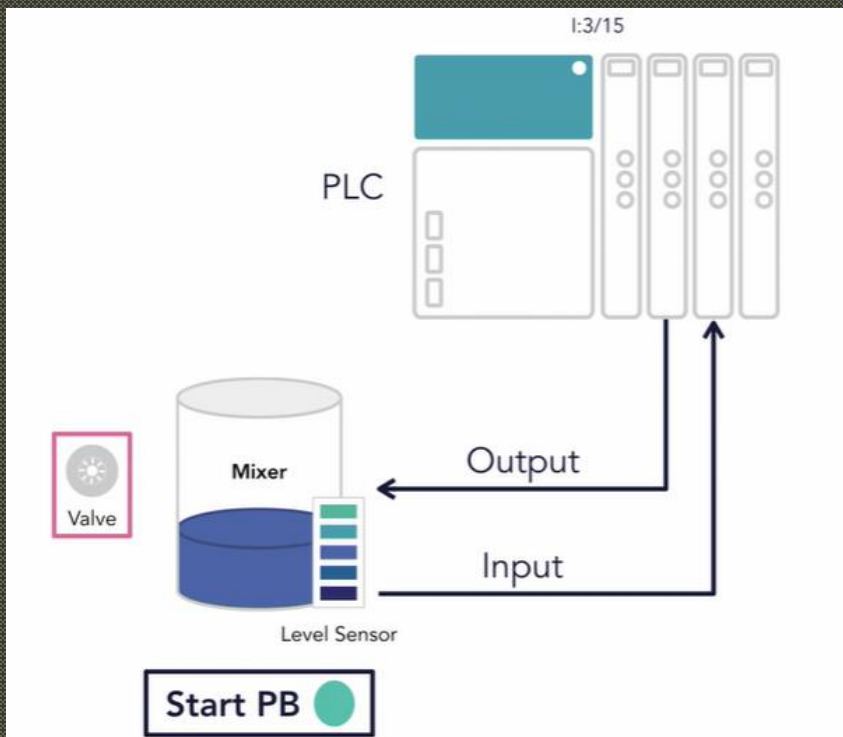
1. Activate a start pushbutton to open a valve.
2. Once the liquid reaches a certain level, the level sensor will indicate ON and the mixer will turn ON.
3. The mixer is ON for three seconds. Once the three seconds end, the process completes.

Assumptions:

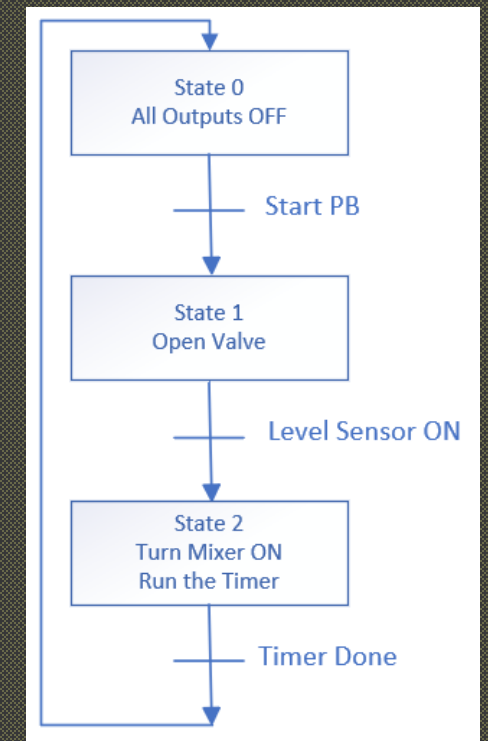
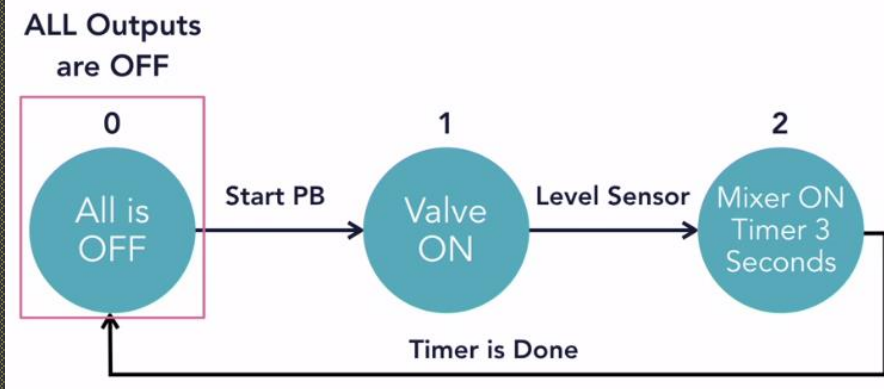
- Once the level reaches the required threshold, the fill-in valve will close.
- The tank level may fluctuate even though a drain valve is not depicted in the figure.

SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 2

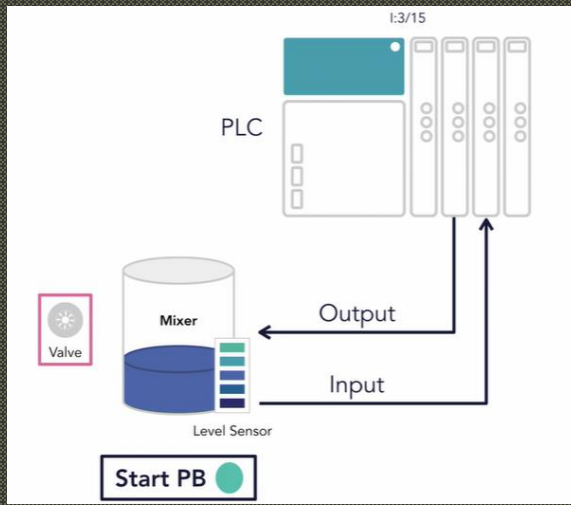


1. Activate a start pushbutton to open a valve.
2. Once the liquid reaches a certain level, the level sensor will indicate ON and the mixer will turn ON.
3. The mixer is ON for three seconds. Once the three seconds end, the process completes.

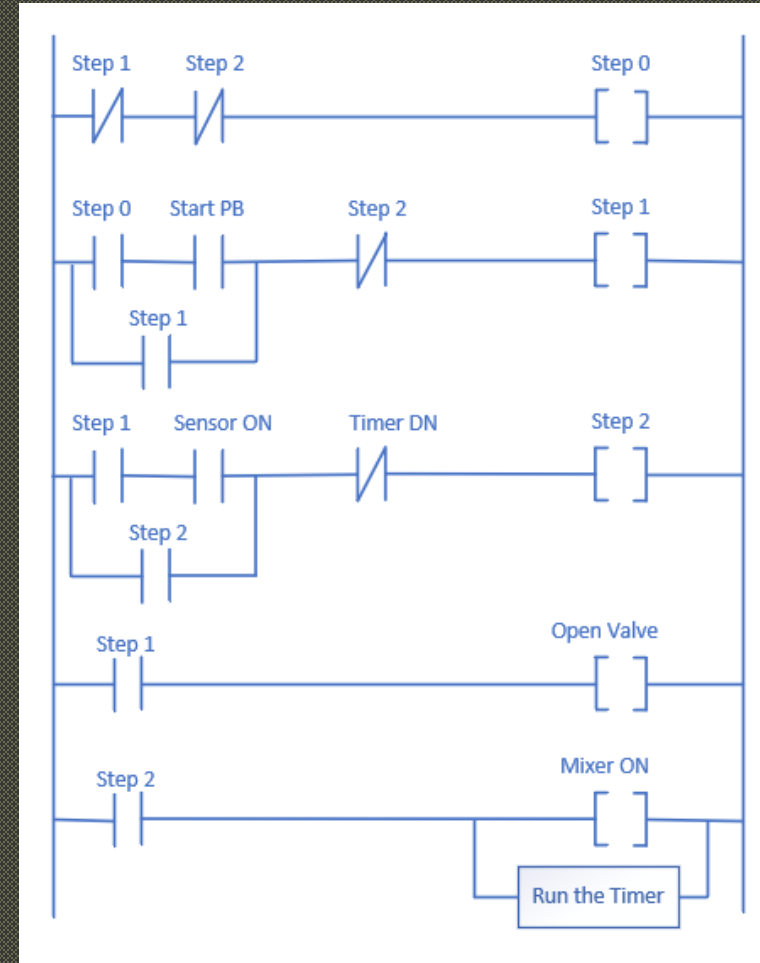
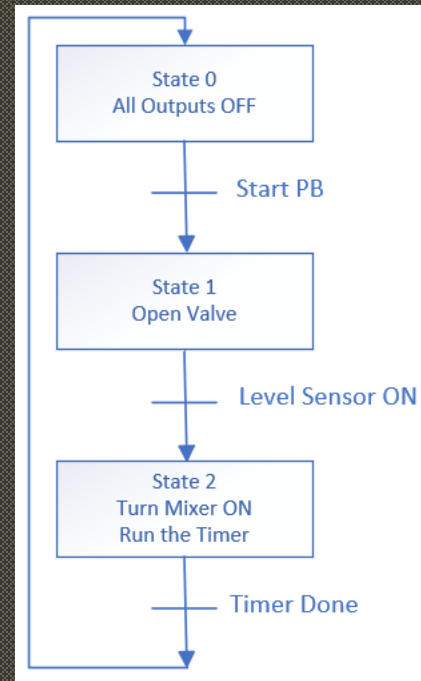
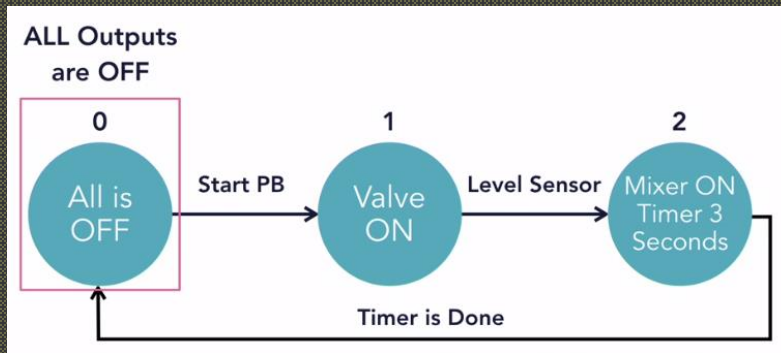


SEQUENTIAL CONTROL AND SEQUENCERS

Cascading Sequence – Example 2



1. Activate a start pushbutton to open a valve.
2. Once the liquid reaches a certain level, the level sensor will indicate ON and the mixer will turn ON.
3. The mixer is ON for three seconds. Once the three seconds end, the process completes.



SEQUENTIAL CONTROL AND SEQUENCERS

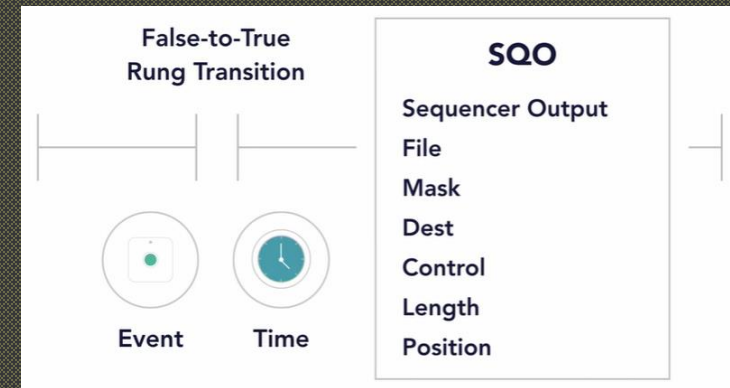
Sequencer Instructions

Sequencer instructions are used to control machines that have a stepped sequence of repeatable operations.

- **SQO (Sequencer Output)** - Is an output instruction that uses a file to control various output devices in a predetermined sequence.
- **SQI (Sequencer Input)** - Is an input instruction that compares bits from an input file to corresponding bits from a source address. The instruction is true if all pairs of bits are the same.
- **SQL (Sequencer Load)** - The SQL instruction loads the source operand value into the sequencer array.

SQO – Sequencer Output

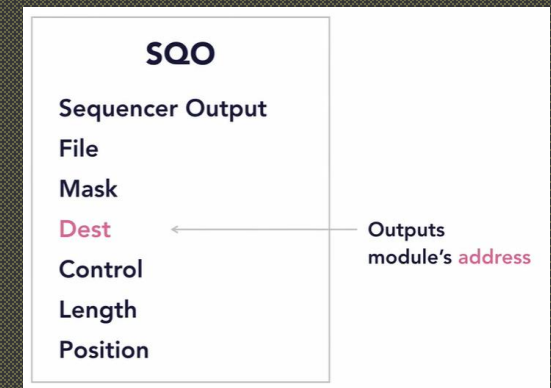
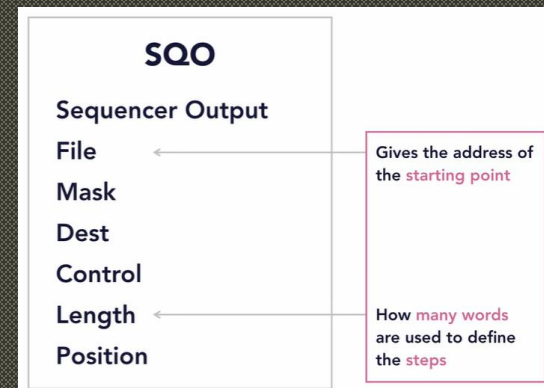
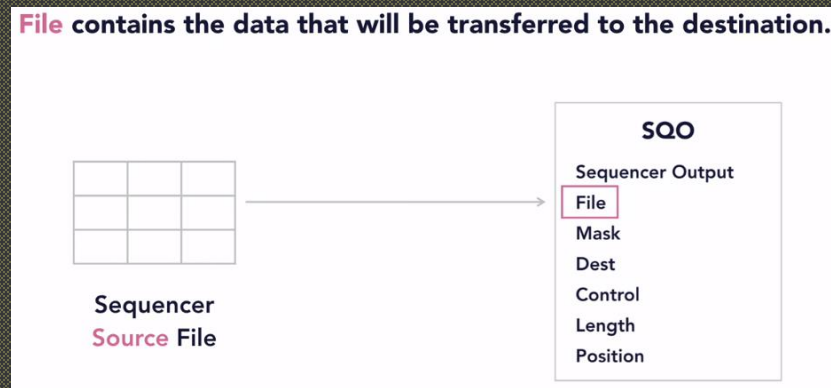
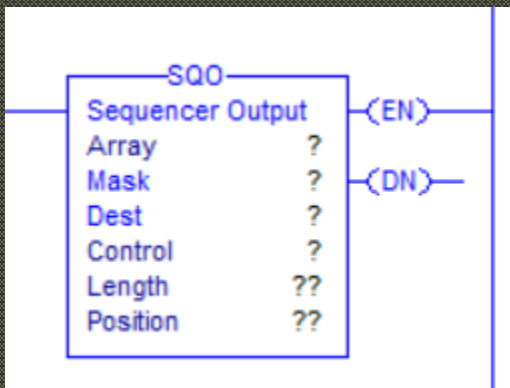
SQO (Sequencer Output) - Is an output instruction that uses a file to control various output devices



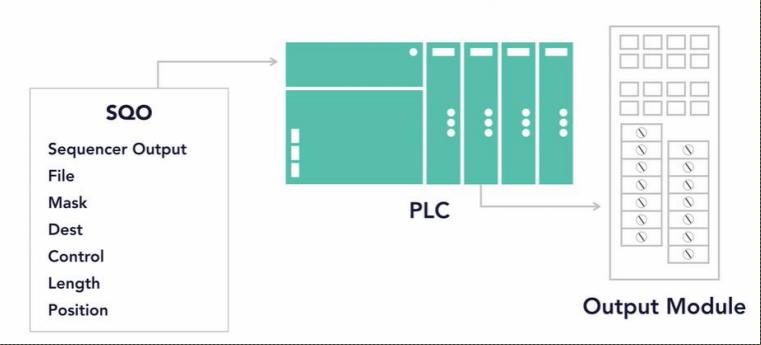
Array - Is the starting address for the registers in the sequencer file. The file contains the data that will be transferred to the Destination address when the instruction undergoes a false-to-true transition.

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.

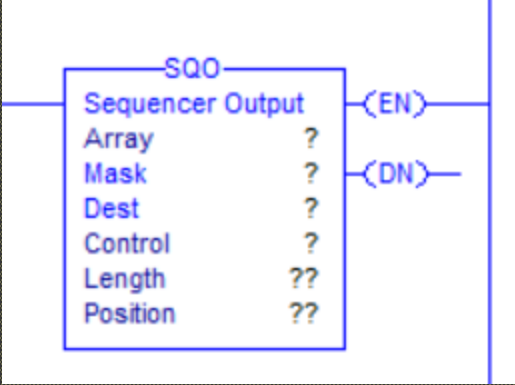
Destination - Is the address of the output word or file to which the SQO moves the data from its sequencer file.



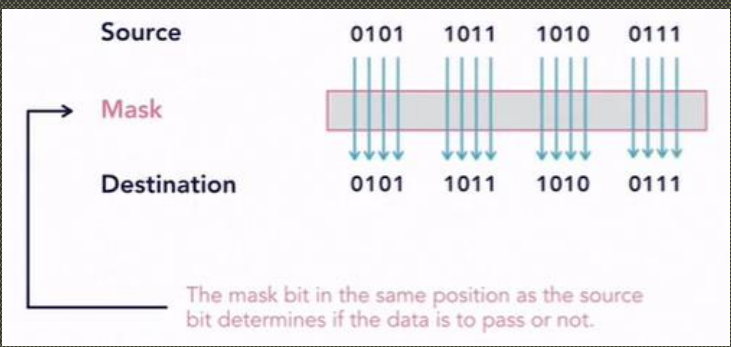
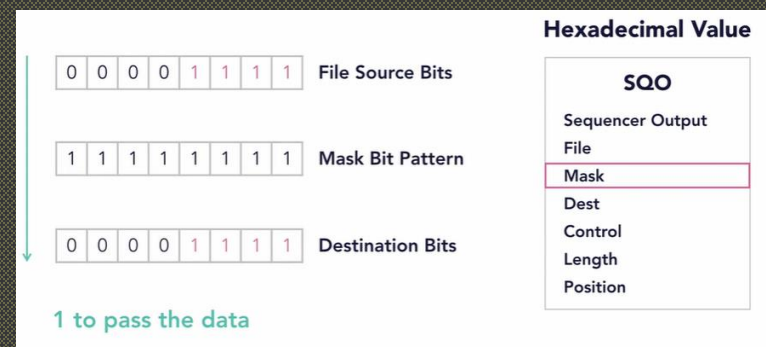
All output points must be on a single output module.



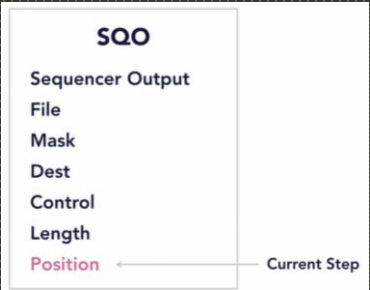
SQQ – Sequencer Output



Mask - Is the bit pattern through which the sequencer instruction moves source data to the Destination address.



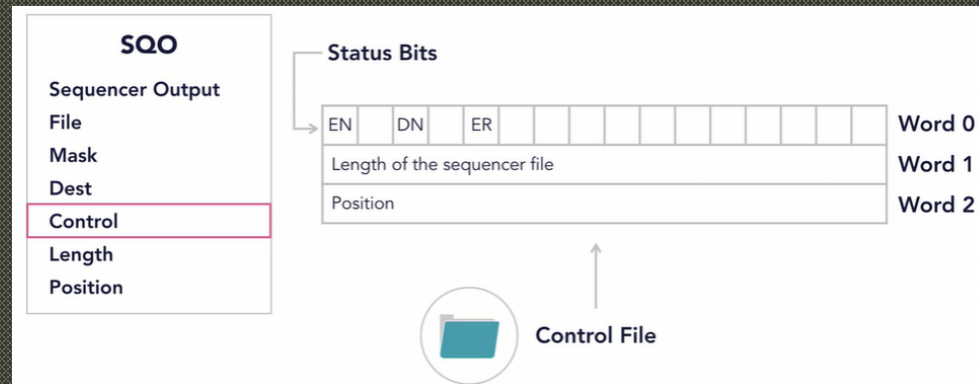
Position - Indicates the step that is desired to start the sequencer instruction. The position is the word location or step in the sequencer file from which the instruction moves data.



SQO – Sequencer Output

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.
- 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.



SQO – Sequencer Output Example

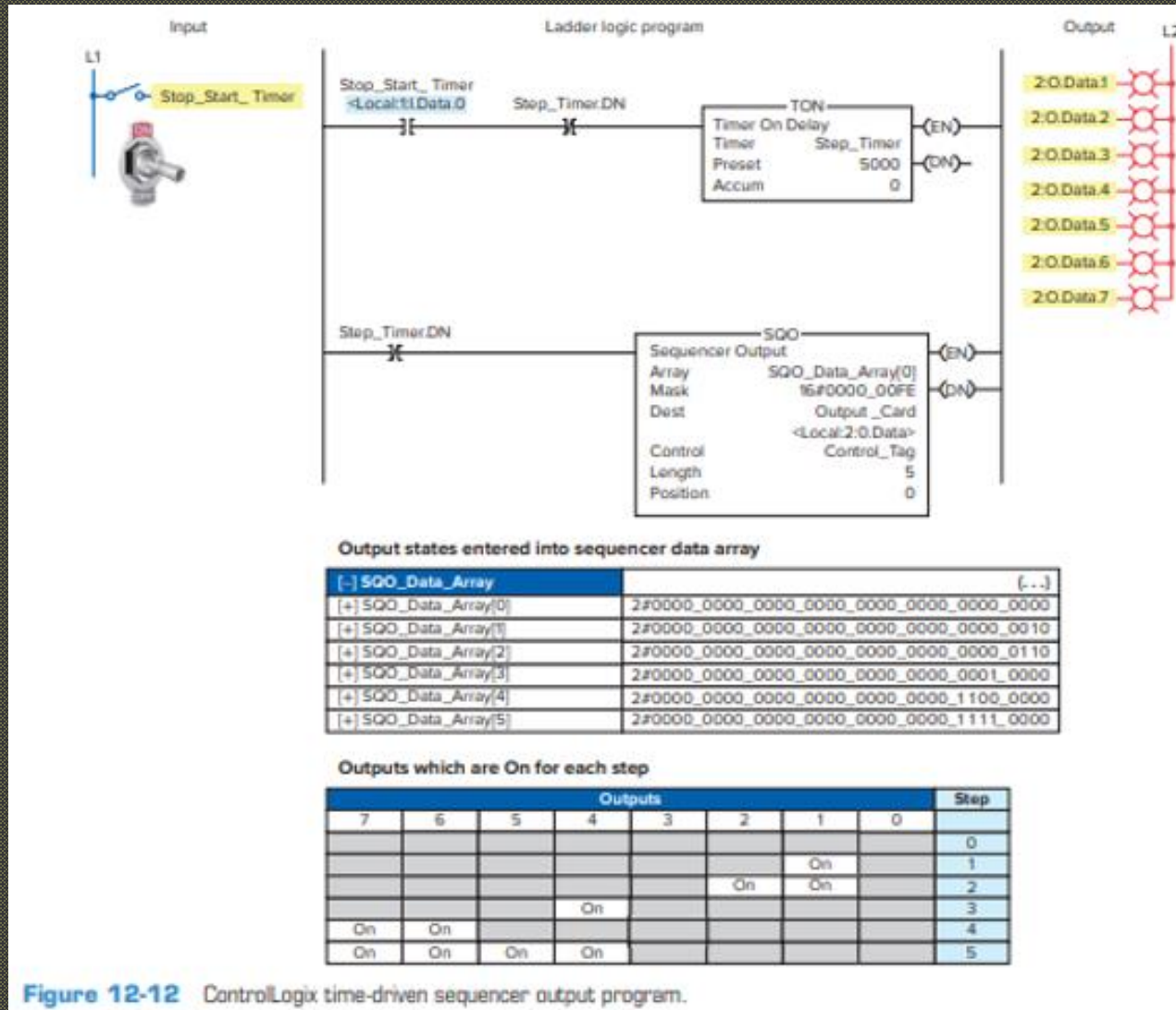


Figure 12-12 ControlLogix time-driven sequencer output program.

- The array parameter SQO_Data_Array contains the desired output states for each step.
- The output states for each step are entered at the starting location SQO_Data_Array[0].
- In this application none of the outputs are energized in step 0. When the SQO instruction executes, it will be in step 0 on initial start-up.
- When the SQO instruction advances to step 5, it will return to step 1 and continue from there.

SQO – Sequencer Output Example

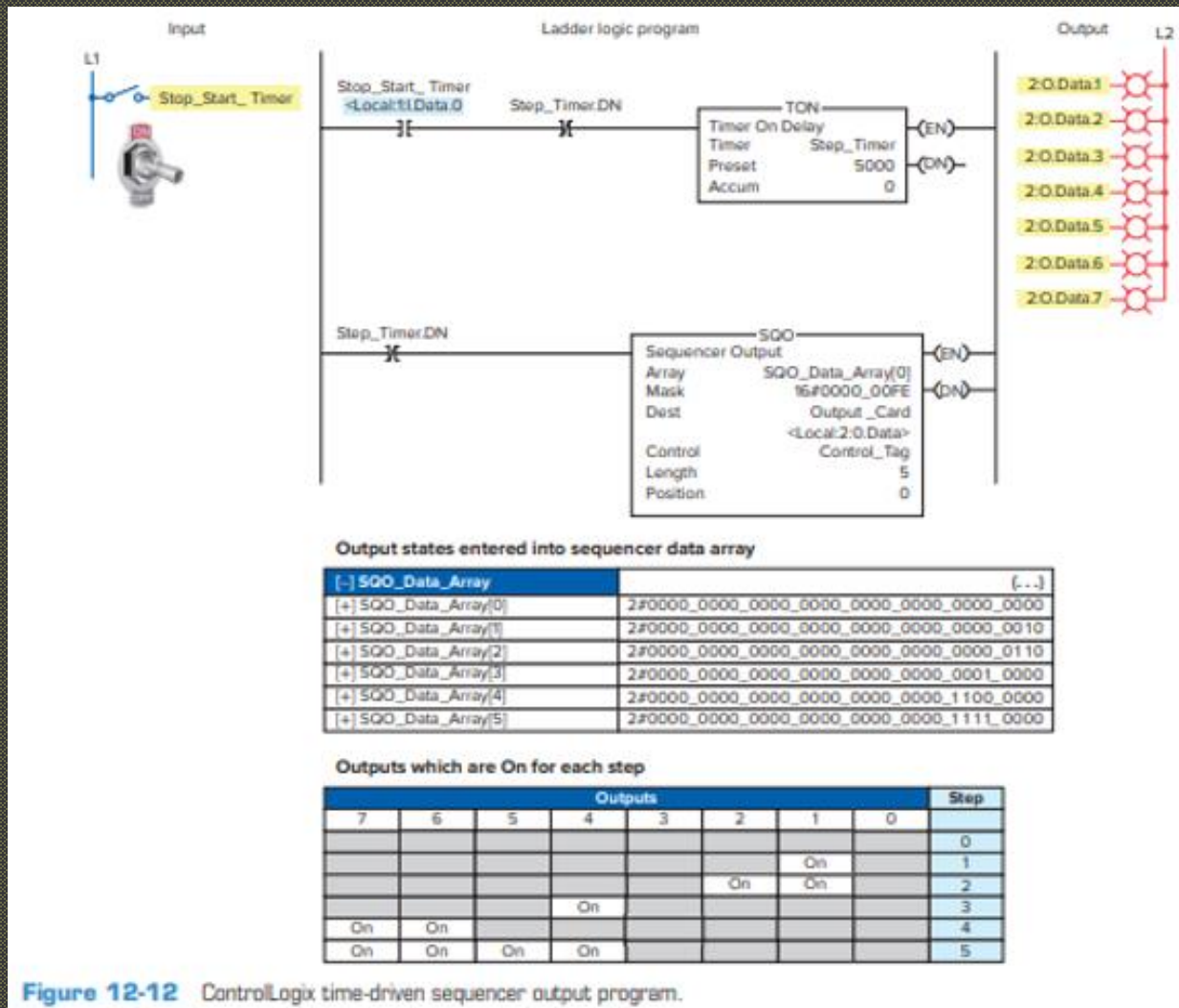
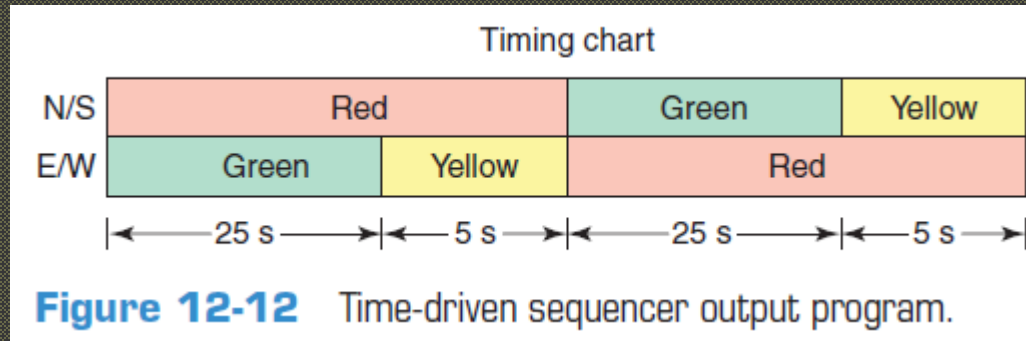


Figure 12-12 ControlLogix time-driven sequencer output program.

- The mask has a constant hexadecimal value of 0000_00FE which is the same as 0000 0000 0000 0000 0000 0000 1111 1110 binary bits. Each bit corresponds to one output in the SQO instruction.
- In this example, the hexadecimal F & E represent 7 binary 1s in memory. A value of 1 in the mask allows the output state from the step to be sent to the Dest.
- The DN (done) bit of the 5-second TON Step_Timer is used to trigger the SQO instruction.
- Every time the timer ACC value reaches 5 seconds, the timer DN bit changes state causing the SQO instruction to increment the Position number of the Control tag and move to the next step.
- Note that the timer DN bit also resets the ACC value of the timer to 0 and the timer starts timing to 5 seconds again.

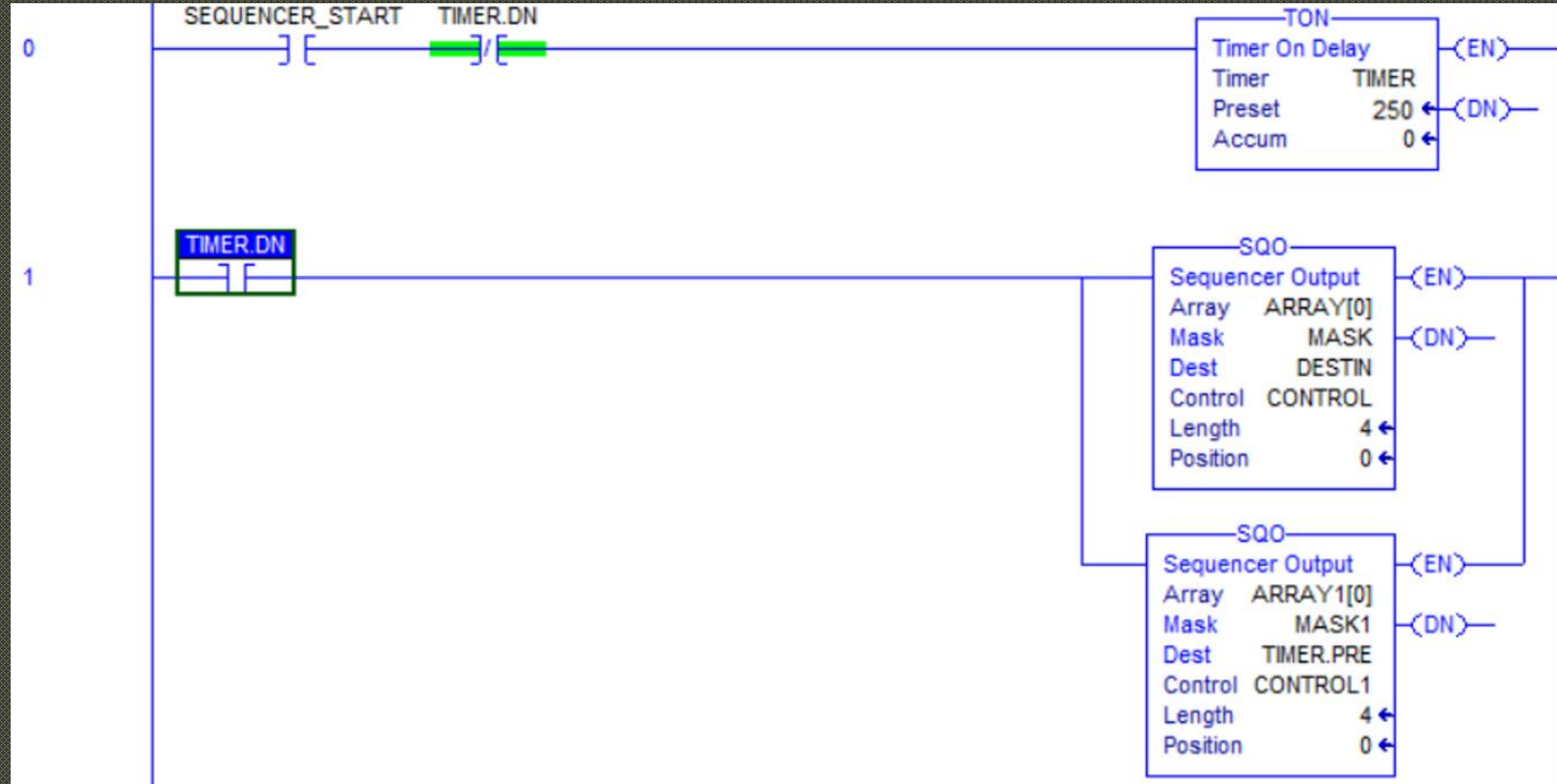
SQO – Sequencer Output

North-South East-West Traffic Light Control



SQO – Sequencer Output

North-South East-West Traffic Light Control



SQO – Sequencer Output

North-South East-West Traffic Light Control



SQO – Sequencer Output

North-South East-West Traffic Light Control

Tag Properties - TIMER

General

Name:

Description:

Type:

Alias For:

Data Type:

Scope:

External Access:

Style:

☐ Constant

Tag Properties - ARRAY

General

Name:

Description:

Type:

Alias For:

Data Type:

Scope:

External Access:

Style:

☐ Constant

Tag Properties - MASK

General

Name:

Description:

Type:

Alias For:

Data Type:

Scope:

External Access:

Style:

☐ Constant

SQO – Sequencer Output

North-South East-West Traffic Light Control

Tag Properties - DESTIN

General

Name: DESTIN

Description:

Type: Base Connection...

Alias For:

Data Type: DINT

Scope: MainProgram

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

Tag Properties - CONTROL

General

Name: CONTROL

Description:

Type: Base Connection...

Alias For:

Data Type: CONTROL

Scope: MainProgram

External Access: Read/Write

Style:

☐ Constant

OK Cancel Apply Help

SQO – Sequencer Output

North-South East-West Traffic Light Control

Tag Properties - ARRAY1

General

Name: ARRAY1

Description:

Type: Base Connection...

Alias For:

Data Type: DINT[5]

Scope: MainProgram

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

Tag Properties - MASK1

General

Name: MASK1

Description:

Type: Base Connection...

Alias For:

Data Type: DINT

Scope: MainProgram

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

Tag Properties - TIMER

General

Name: TIMER

Description:

Type: Base Connection...

Alias For:

Data Type: TIMER

Scope: MainProgram

External Access: Read/Write

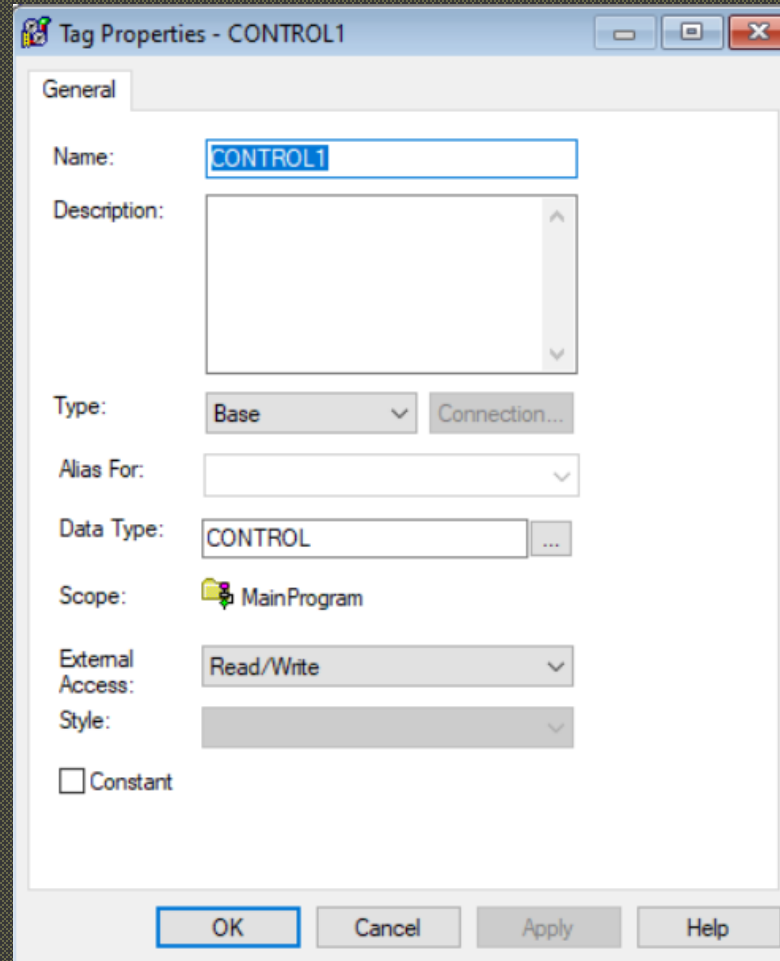
Style:

☐ Constant

OK Cancel Apply Help

SQO – Sequencer Output

North-South East-West Traffic Light Control



The image shows a 'Tag Properties' dialog box for a tag named 'CONTROL1'. The dialog has a 'General' tab and several fields for configuring the tag's properties. The 'Name' field is set to 'CONTROL1'. The 'Description' field is empty. The 'Type' is set to 'Base'. The 'Alias For' field is empty. The 'Data Type' is set to 'CONTROL'. The 'Scope' is set to 'MainProgram'. The 'External Access' is set to 'Read/Write'. The 'Style' is set to an empty dropdown. There is an unchecked 'Constant' checkbox at the bottom. The dialog has 'OK', 'Cancel', 'Apply', and 'Help' buttons at the bottom.

Tag Properties - CONTROL1

General

Name: CONTROL1

Description:

Type: Base

Alias For:

Data Type: CONTROL

Scope: MainProgram

External Access: Read/Write

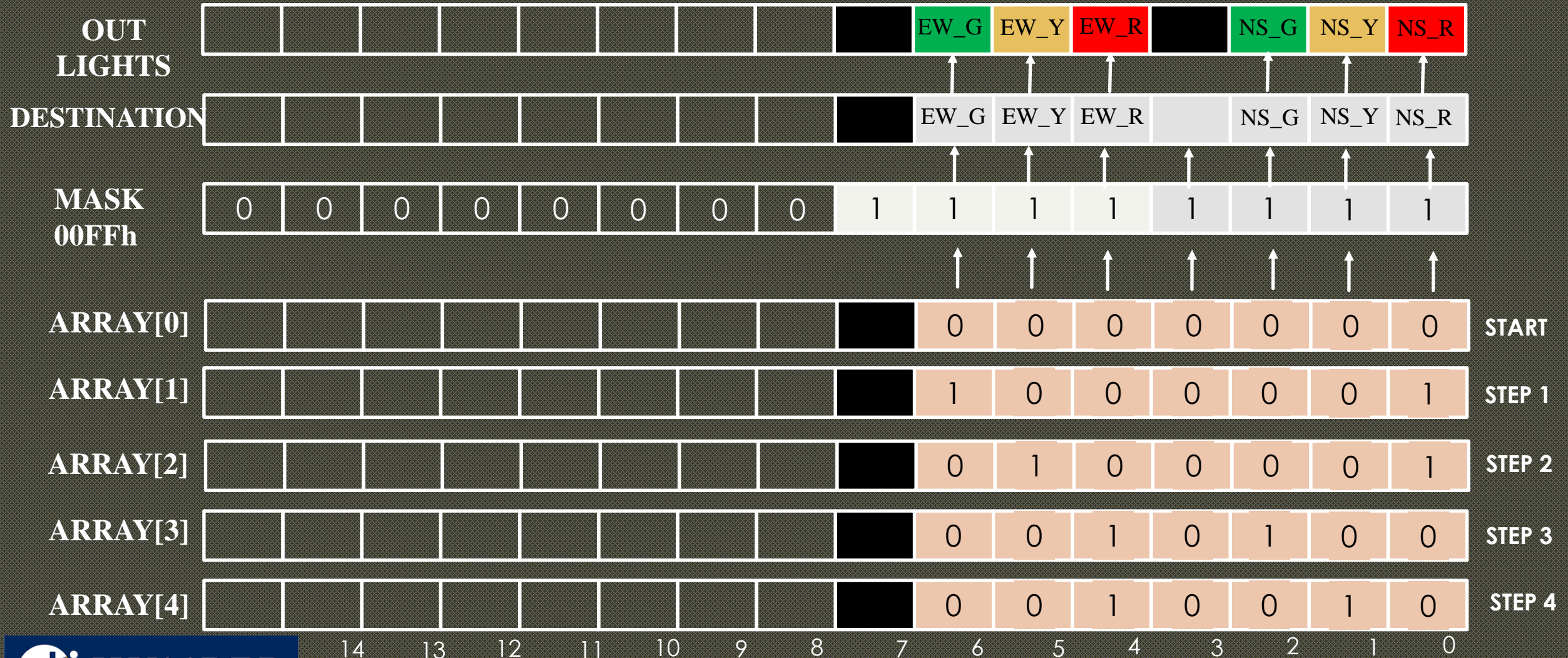
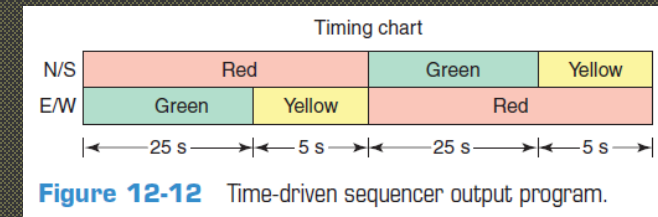
Style:

☐ Constant

OK Cancel Apply Help

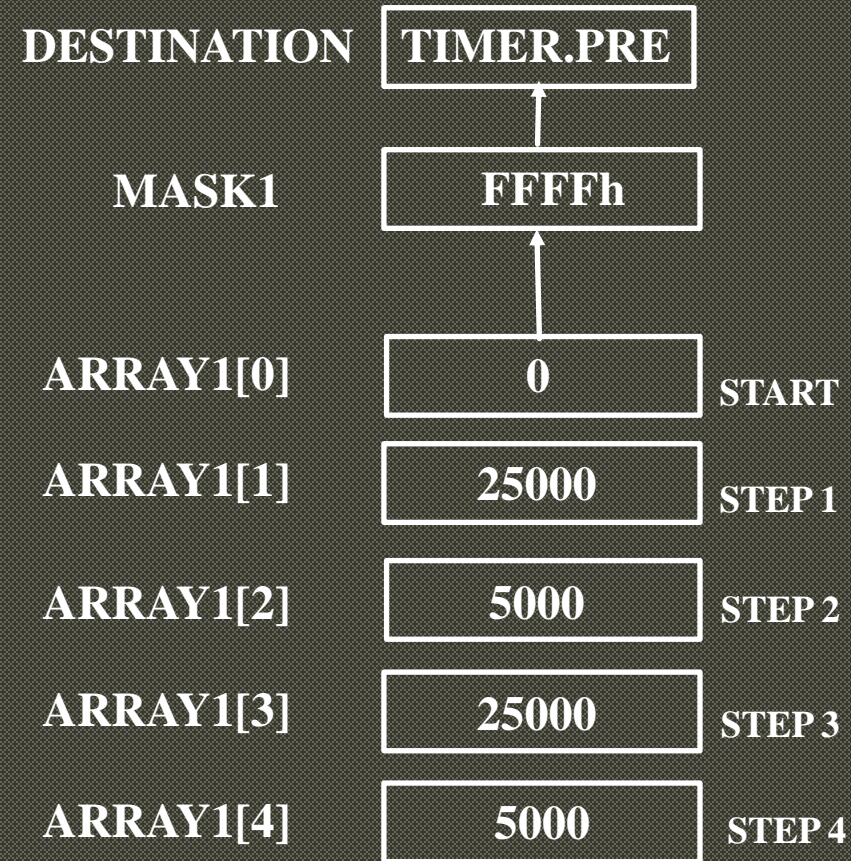
SQO – Sequencer Output

North-South East-West Traffic Light Control



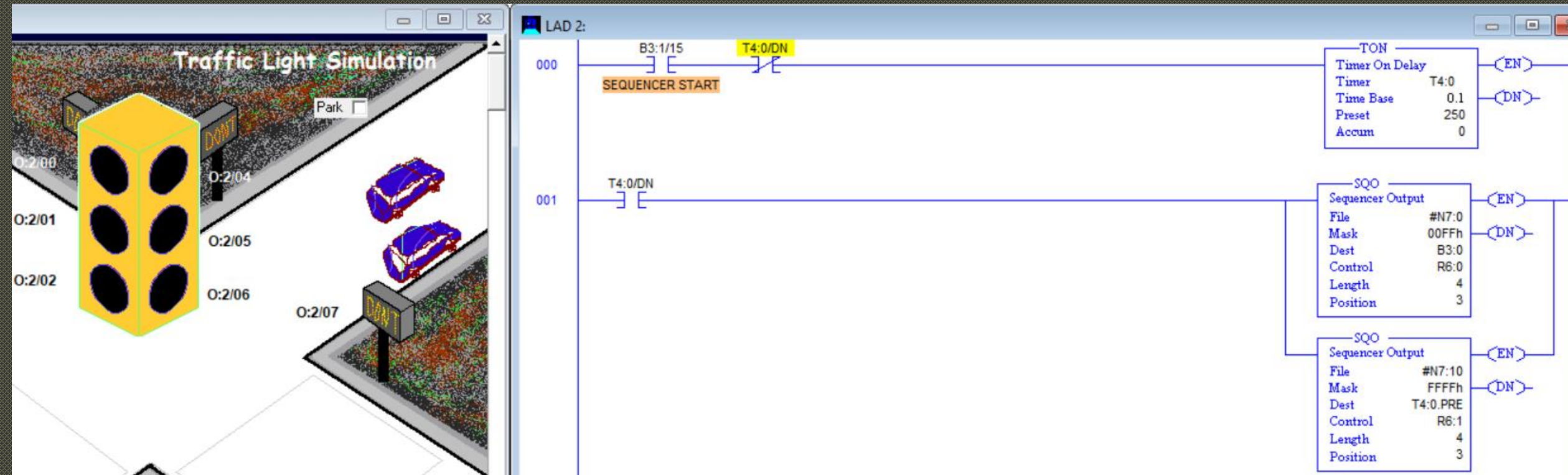
SQO – Sequencer Output

North-South East-West Traffic Light Control



SQO – Sequencer Output

North-South East-West Traffic Light Control (LogixPro Simulation)

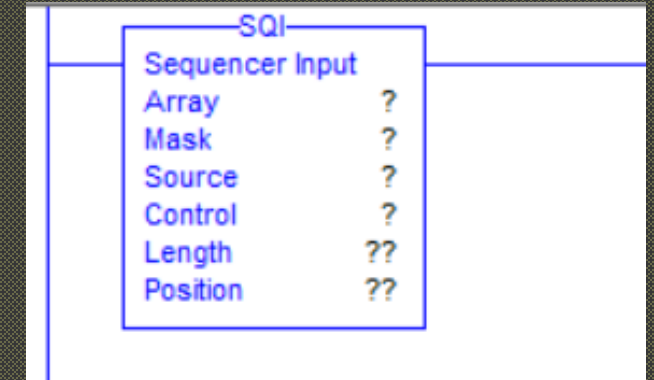


SQO – Sequencer Output

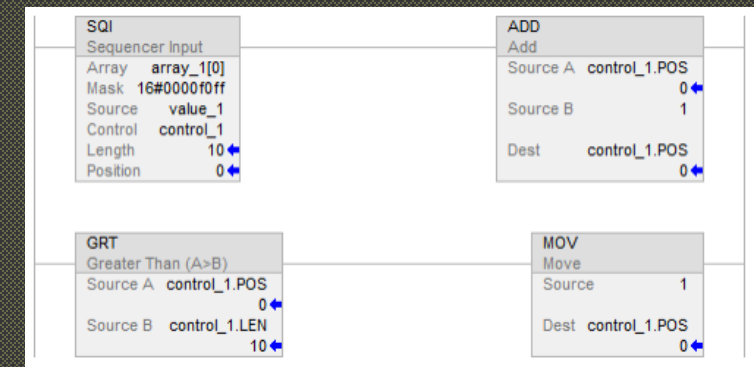
North-South East-West Traffic Light Control (LogixPro Simulation)



SQI – Sequencer Input



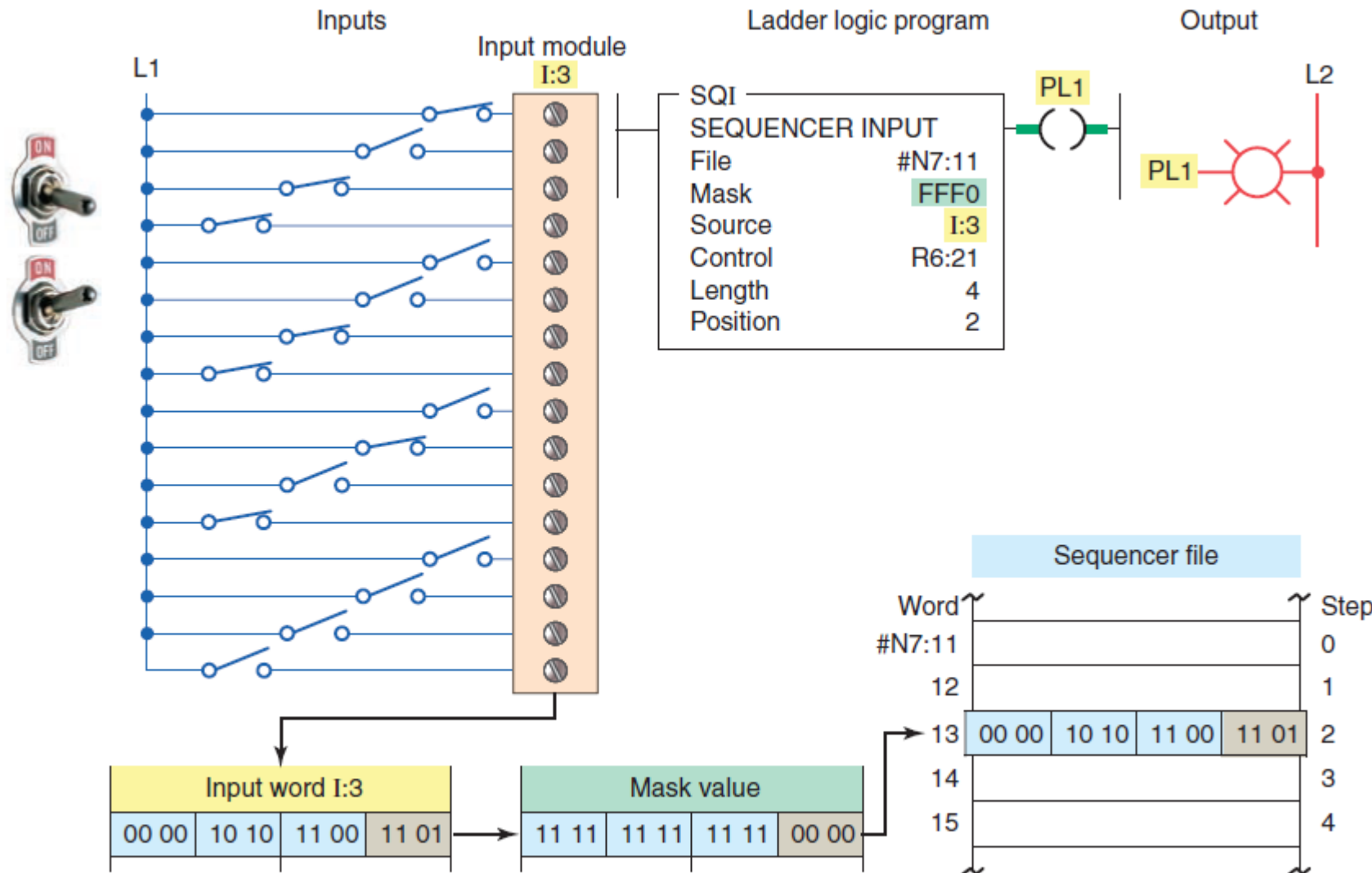
- **SQI (Sequencer Input)** —Is an input instruction that compares bits from an input file to corresponding bits from a source address. The instruction is true if all pairs of bits are the same.
- The SQI instruction uses a control register like the SQO instruction but does not have a done bit. In addition, the SQI instruction does not automatically increment its position each time its control logic makes a false-to-true transition at its input.
- If the SQI instruction is used alone, the position value must be changed by another instruction (such as the move instruction) to select a new input file value to compare against the value from the source address.



(Frank D. Petruzella Programmable Logic Controllers 4th edition, CH 12.2)

SQI – Sequencer Input

(Frank D. Petruzella Programmable Logic Controllers 4th edition, CH 12.2)

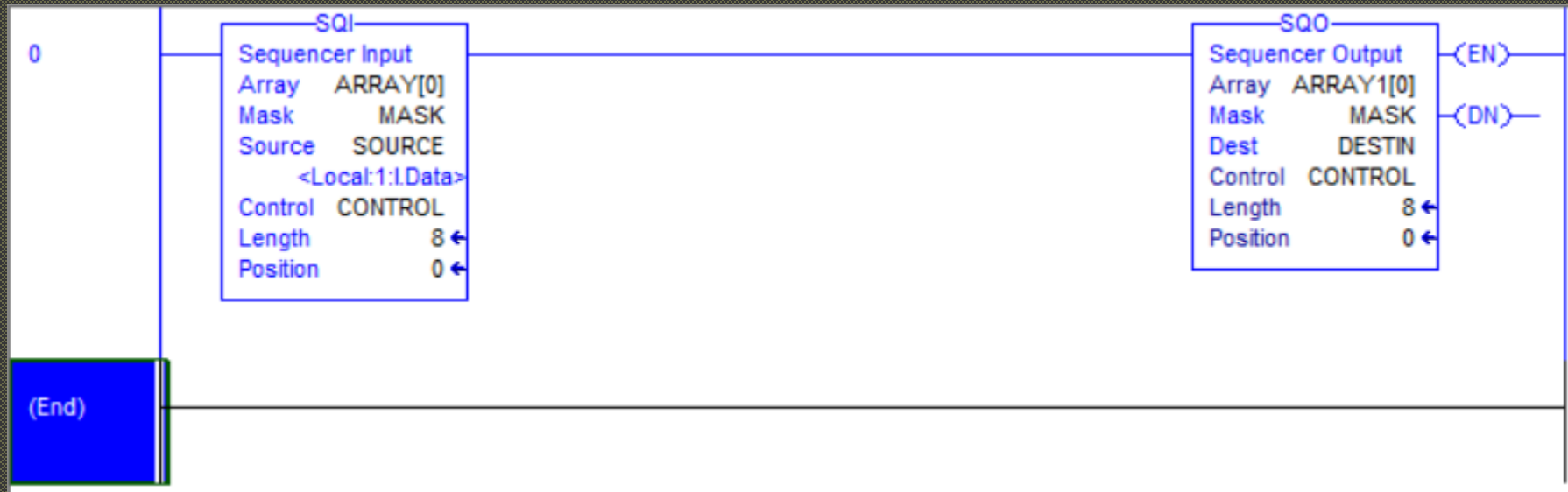


When the unmasked source bits match those of the corresponding sequencer file word, the instruction goes true; otherwise, the instruction is false.

Figure 12-17 Sequencer input (SQI) instruction.

SQI – Sequencer Input

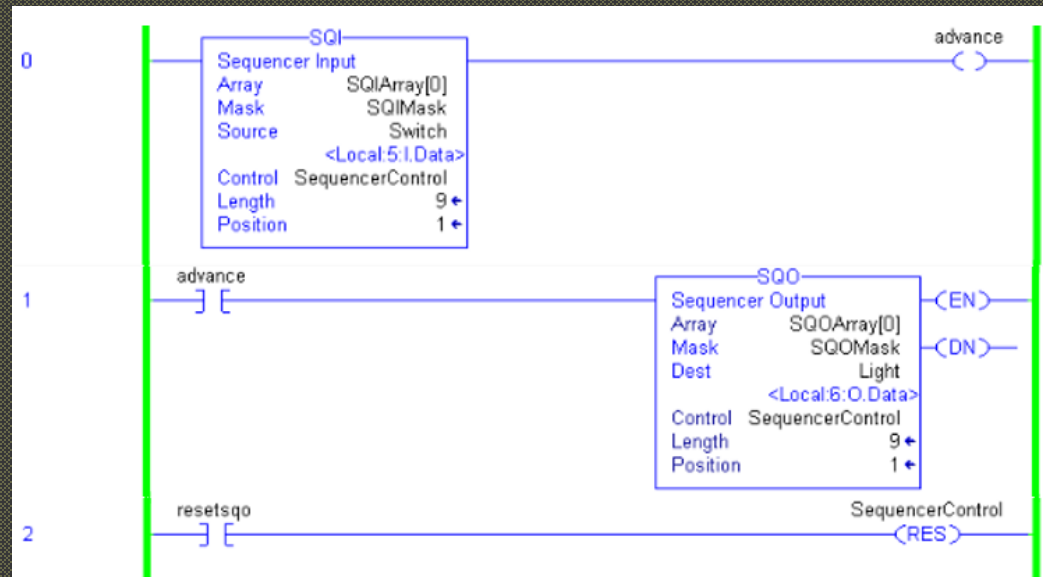
- When the SQI is paired with an SQO instruction with identical control addresses the position is incremented by the SQO instruction for both.



- The same control address, length value, and position value is used for each instruction.
- The sequencer input instruction is indexed by the sequencer output instruction because both control elements have the same address, "CONTROL".
- This type of programming technique allows input and output sequences to function in unison, causing a specific output sequence to occur when a specific input sequence takes place.

SQI – Sequencer Input - Example

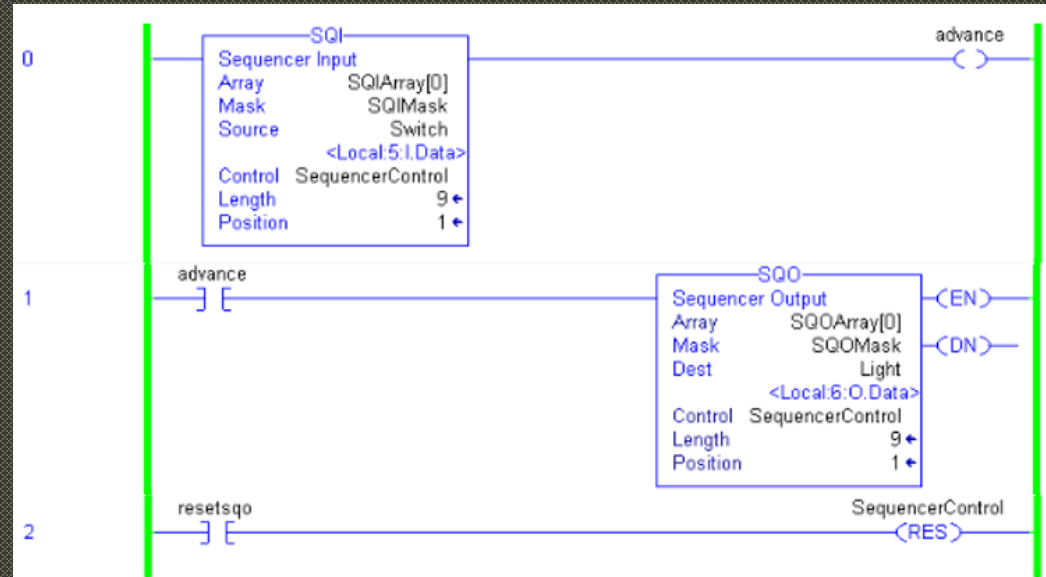
⊕ Switch	Local:5:I.Data	Local:5:I.Data	DINT
⊕ Light	Local:6:O.Data	Local:6:O.Data	DINT
⊕ SQIArray			DINT[10]
advance			BOOL
⊕ SQIMask			DINT
⊕ SQOArray			DINT[10]
⊕ SQOMask			DINT
⊕ SequencerControl			CONTROL
resetsqo			BOOL



- SQIArray	{...}
⊕ SQIArray[0]	0
⊕ SQIArray[1]	1
⊕ SQIArray[2]	2
⊕ SQIArray[3]	4
⊕ SQIArray[4]	8
⊕ SQIArray[5]	16
⊕ SQIArray[6]	32
⊕ SQIArray[7]	64
⊕ SQIArray[8]	128
⊕ SQIArray[9]	256

- SQIArray	{...}
⊕ SQIArray[0]	2#0000_0000_0000_0000_0000_0000_0000_0000
⊕ SQIArray[1]	2#0000_0000_0000_0000_0000_0000_0000_0001
⊕ SQIArray[2]	2#0000_0000_0000_0000_0000_0000_0000_0010
⊕ SQIArray[3]	2#0000_0000_0000_0000_0000_0000_0000_0100
⊕ SQIArray[4]	2#0000_0000_0000_0000_0000_0000_0000_1000
⊕ SQIArray[5]	2#0000_0000_0000_0000_0000_0000_0001_0000
⊕ SQIArray[6]	2#0000_0000_0000_0000_0000_0000_0010_0000
⊕ SQIArray[7]	2#0000_0000_0000_0000_0000_0000_0100_0000
⊕ SQIArray[8]	2#0000_0000_0000_0000_0000_0000_1000_0000
⊕ SQIArray[9]	2#0000_0000_0000_0000_0000_0001_0000_0000

SQI – Sequencer Input - Example



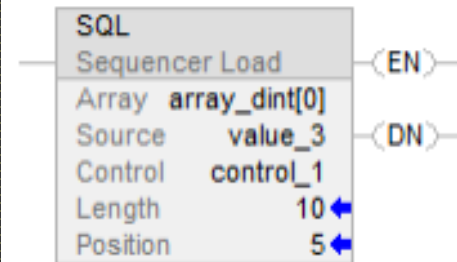
- SQOArray	{...}
+ SQOArray[0]	0
+ SQOArray[1]	1
+ SQOArray[2]	2
+ SQOArray[3]	4
+ SQOArray[4]	8
+ SQOArray[5]	16
+ SQOArray[6]	32
+ SQOArray[7]	64
+ SQOArray[8]	128
+ SQOArray[9]	256
+ SQOMask	16#ffff_ffff

- SQOArray	{...}
+ SQOArray[0]	2#0000_0000_0000_0000_0000_0000_0000_0000
+ SQOArray[1]	2#0000_0000_0000_0000_0000_0000_0000_0001
+ SQOArray[2]	2#0000_0000_0000_0000_0000_0000_0000_0010
+ SQOArray[3]	2#0000_0000_0000_0000_0000_0000_0000_0100
+ SQOArray[4]	2#0000_0000_0000_0000_0000_0000_0000_1000
+ SQOArray[5]	2#0000_0000_0000_0000_0000_0000_0001_0000
+ SQOArray[6]	2#0000_0000_0000_0000_0000_0000_0010_0000
+ SQOArray[7]	2#0000_0000_0000_0000_0000_0000_0100_0000
+ SQOArray[8]	2#0000_0000_0000_0000_0000_0000_1000_0000
+ SQOArray[9]	2#0000_0000_0000_0000_0000_0001_0000_0000

SQL – Sequencer Load

- When .EN transitions from false to true, the .POS is incremented.
 - The .POS is reset to 1 when the .POS becomes $>$ or $=$ to .LEN.
 - The SQL instruction loads the Source value into the Array at the new position.
- SQL uses the same CONTROL structure as the SQI and SQO instructions.

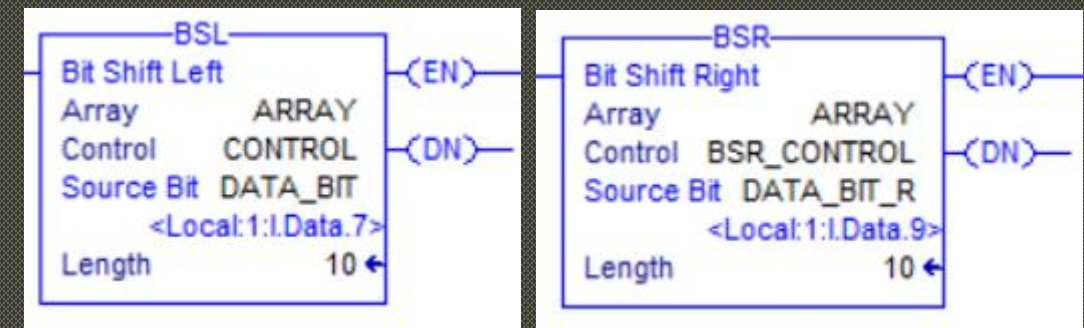
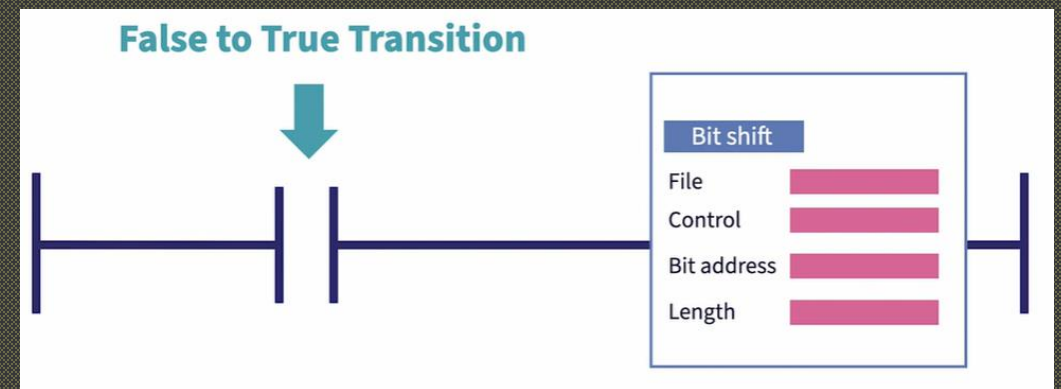
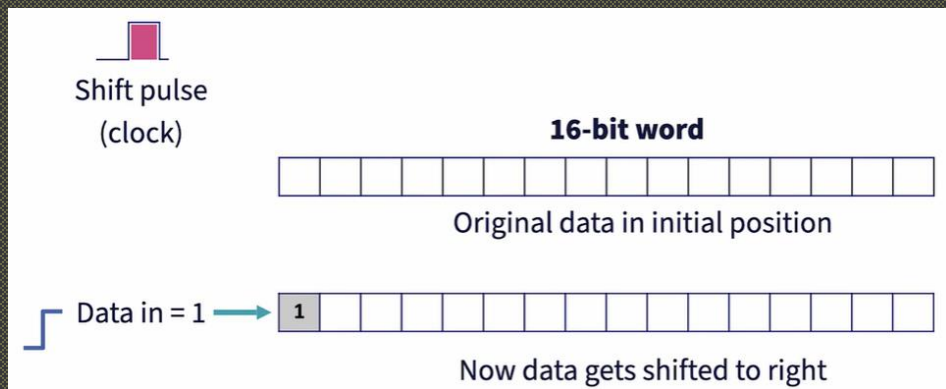
Ladder Diagram



When enabled, the SQL instruction loads `value_3` into the next position in the sequencer array, which is `array_dint[5]` in this example.

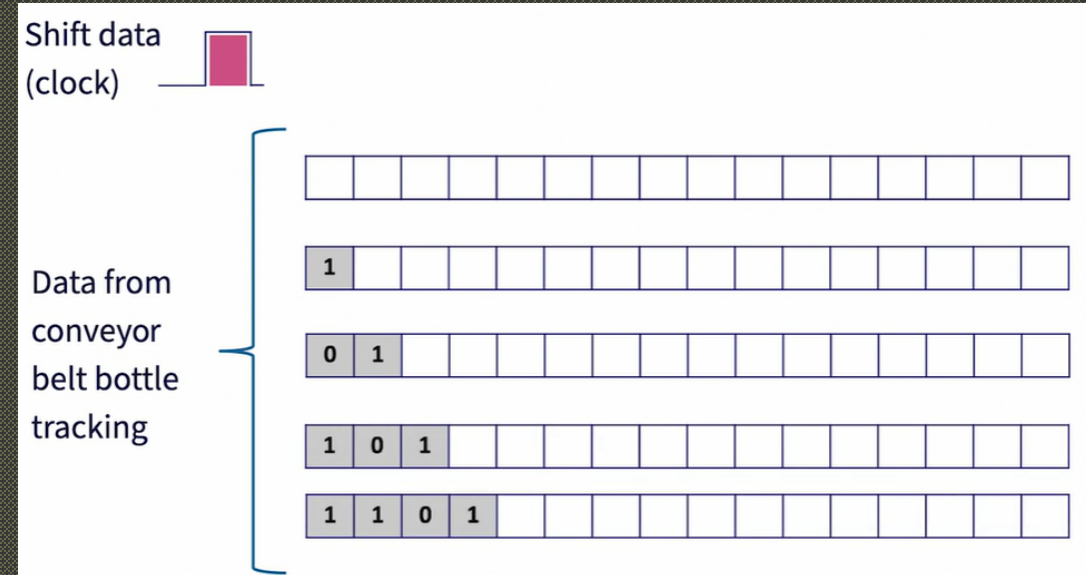
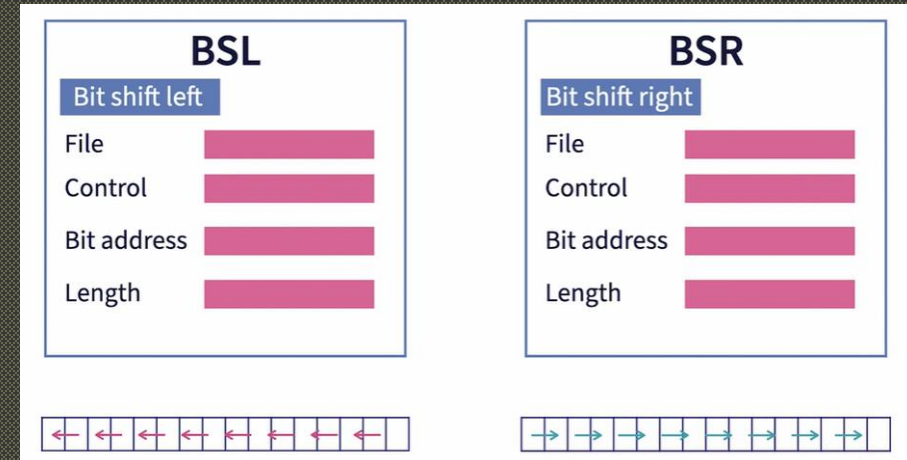
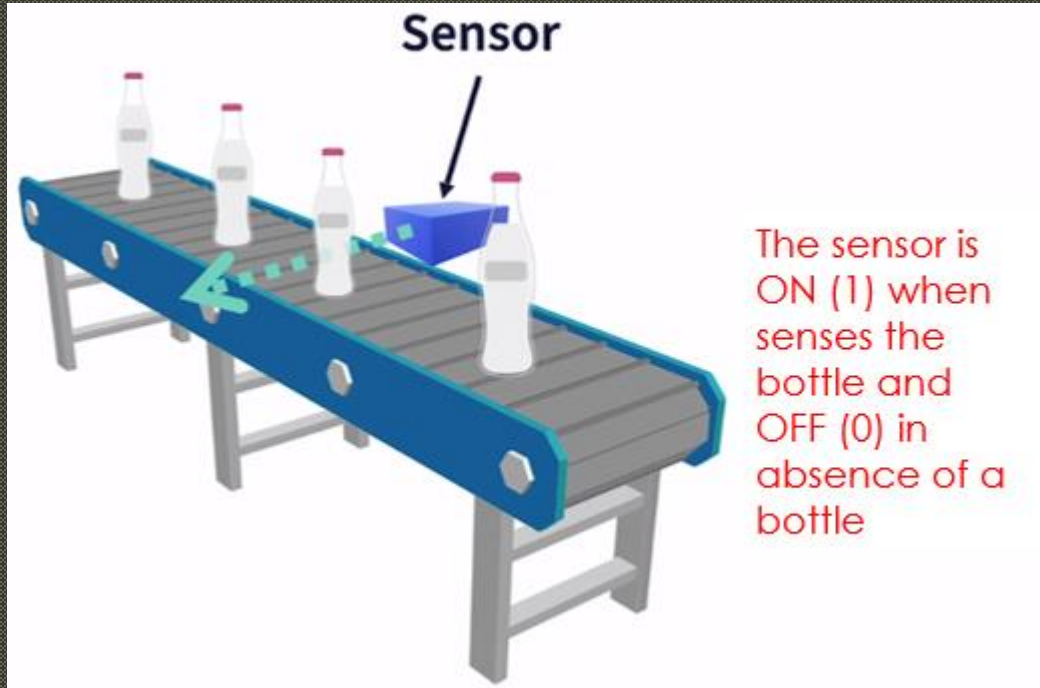
BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT

- A bit shift register instruction allows the shifting of bits through a single register or group of registers. The bit shift register shifts bits serially (from one register to the next one) through an array in an orderly fashion.
- 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.

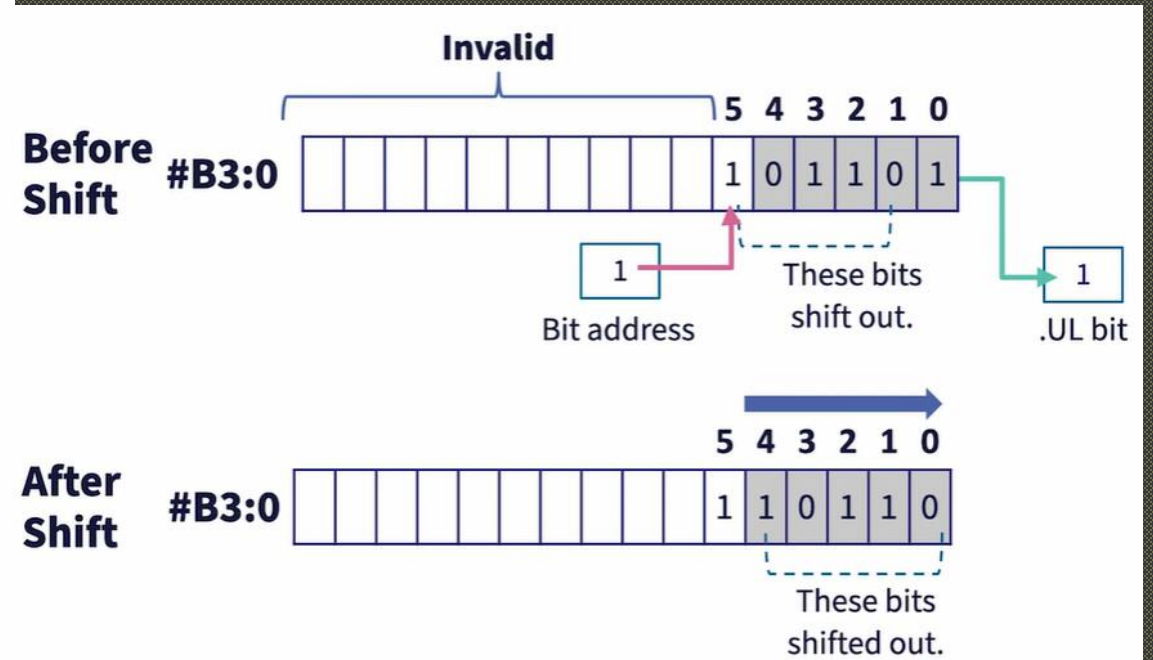
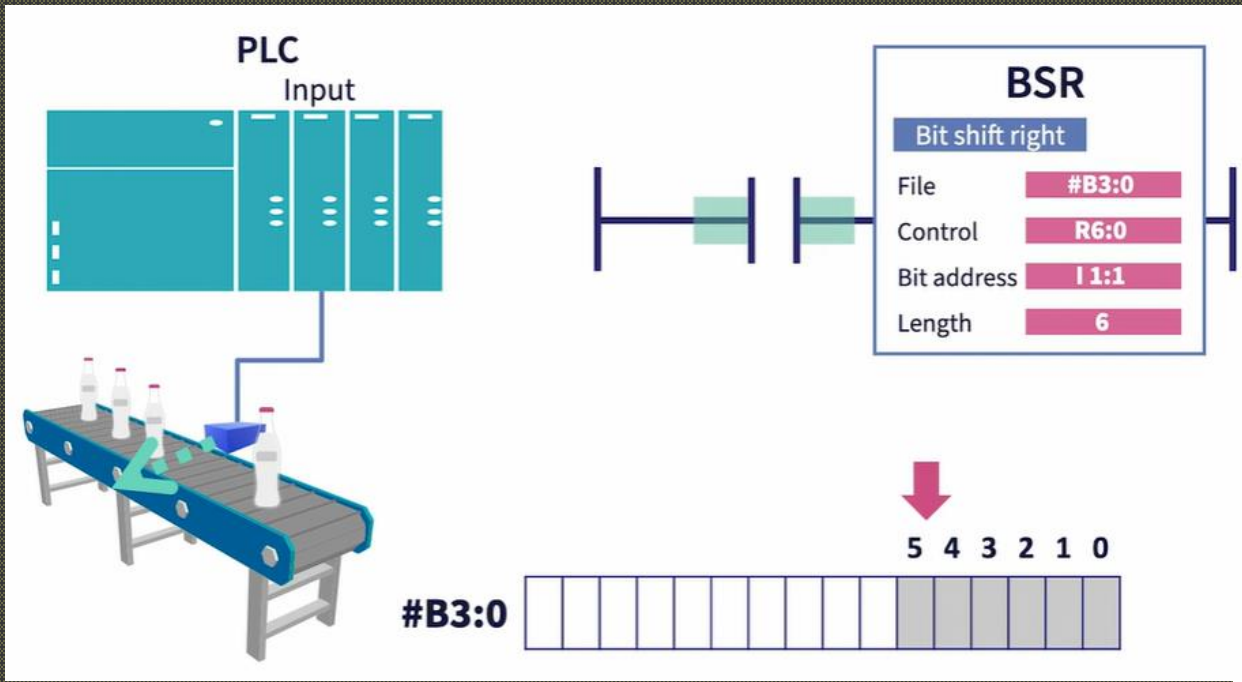


<https://www.linkedin.com/learning/plc-program-flow-and-control-instructions/>

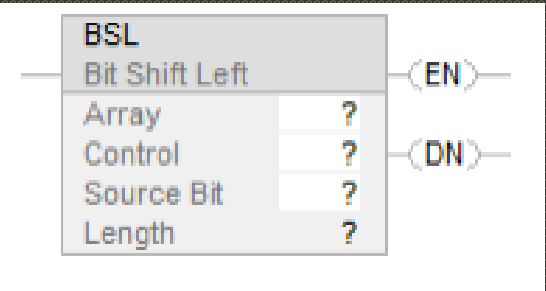
BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT



BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT



BSL – BIT SHIFT LEFT



When enabled, the instruction unloads the uppermost bit of the specified bits to the .UL bit, shifts the remaining bits one position left, and loads Bit address into bit 0 of Array.

CONTROL Structure		
Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates the BSL instruction is enabled.
.DN	BOOL	The done bit is set to indicate that bits shifted one position to the left.
.UL	BOOL	The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits.
.ER	BOOL	The error bit is set when .LEN < 0.
.LEN	DINT	The length specifies the number of array bits to shift.

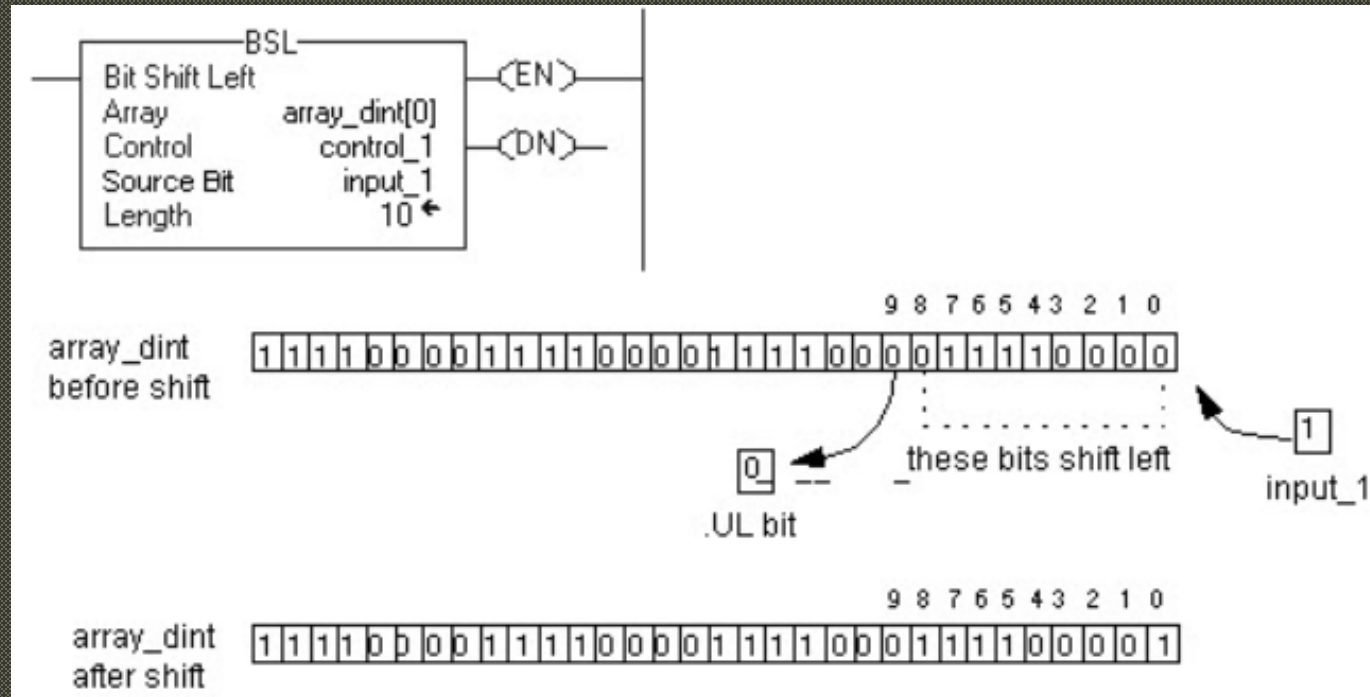
In this transitional instruction, the relay ladder toggles the rung-condition-in from false to true for the instruction to execute.

Operand	Type	Format	Description
Array	DINT ARRAY	tag	Array to modify specify the first element where to begin the shift
Control	CONTROL	tag	Control structure for the operation
Source Bit	BOOL	tag	Bit to shift into the vacated position.
Length	DINT	immediate	Number of bits in the array to shift

BSL – BIT SHIFT LEFT

Example 1

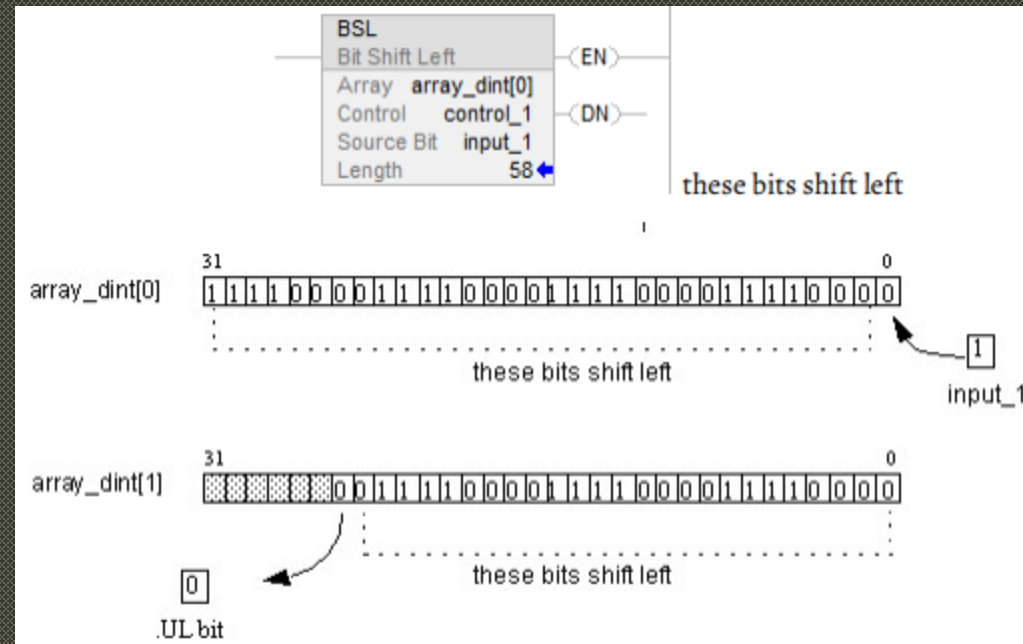
- When enabled, the BSL instruction starts at bit 0 in array_dint[0]
- The instruction unloads array_dint[0].9 into the control_1.UL bit, shifts the remaining bits left, and loads input_1 into array_dint[0].0
- The remaining bits (10-31) are invalid



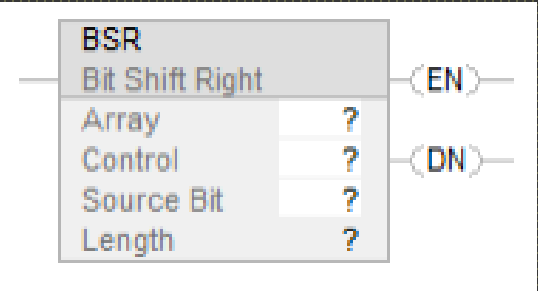
BSL – BIT SHIFT LEFT

Example 2

- When enabled, the BSL instruction starts at bit 0 in array_dint[0].
- The instruction unloads array_dint[1].25 into the control_1.UL bit, shifts the remaining bits, and loads input_1 into array_dint[0].0
- The remaining bits (31-26 in array_dint[1]) are invalid.



BSR – BIT SHIFT RIGHT



When enabled, the instruction unloads the value at bit 0 of an array to the .UL bit, shifts the remaining bits one position right, and loads the bit from the Bit address.

Mnemonic	Data Type	Description
.EN	BOOL	The enable bit indicates the BSR instruction is enabled.
.DN	BOOL	The done bit is set to indicate that bits shifted one position to the right.
.UL	BOOL	The unload bit is the instruction's output. The .UL bit stores the status of the bit that was shifted out of the range of bits.
.ER	BOOL	The error bit is set when .LEN < 0.
.LEN	DINT	The length specifies the number of array bits to shift.

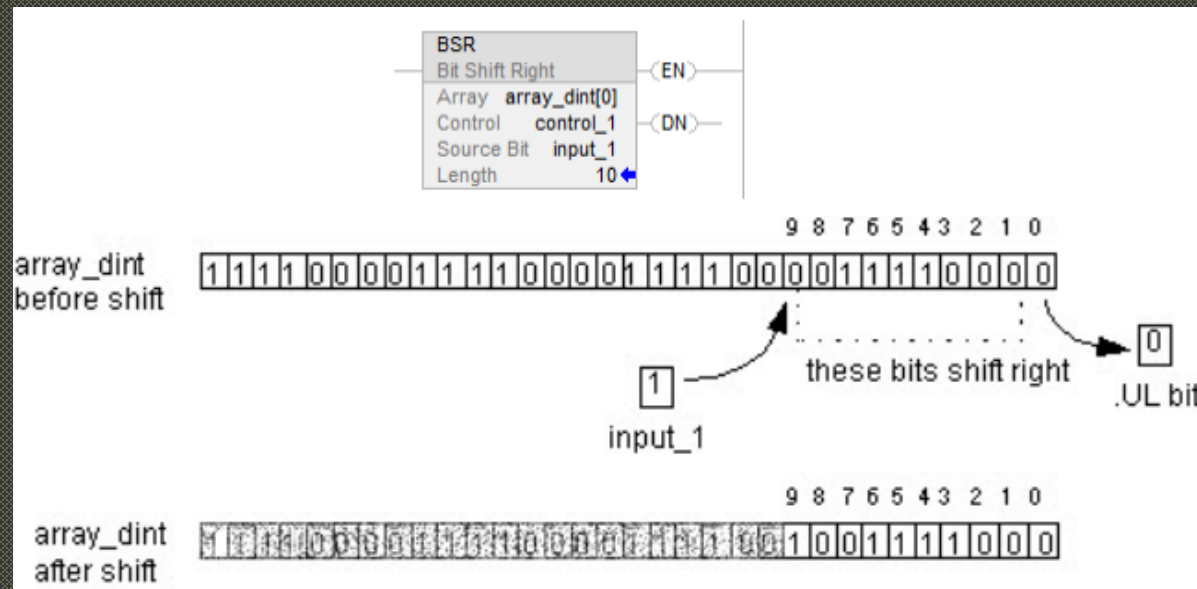
In this transitional instruction, the relay ladder toggles the rung-condition-in from false to true for the instruction to execute.

Operand	Data Type	Format	Description
Array	DINT ARRAY	tag	Array to modify specify the first element to be shifted.
Control	CONTROL	tag	Control structure for the operation
Source Bit	BOOL	tag	Bit to load into the vacated position.
Length	DINT	immediate	Number of bits in the array to shift

BSR – BIT SHIFT RIGHT

Example 1

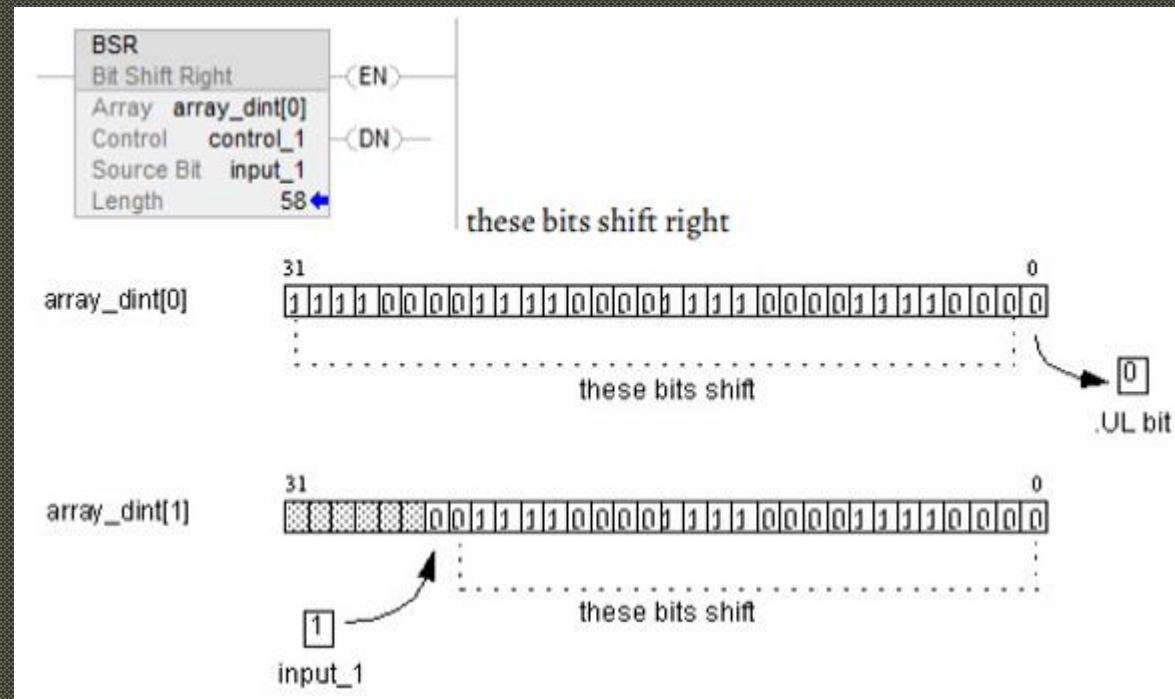
- When enabled, the BSR instruction copies array_dint[0].0 to the .UL bit
- Shifts 0-9 to the right and loads the input_1 into array_dint[0].9
- The remaining bits (10-31) are invalid, which indicates the bits may not be modified



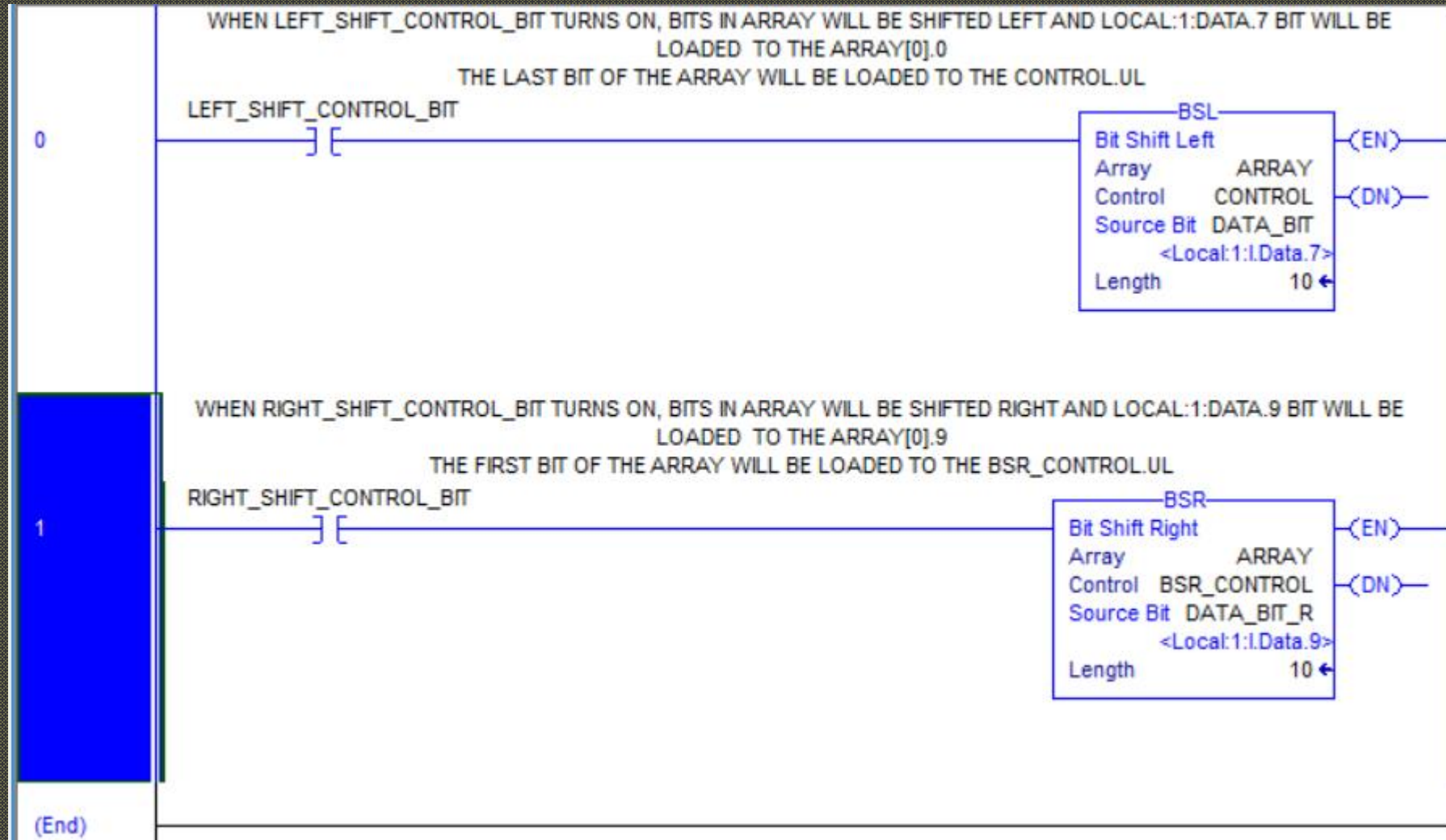
BSR – BIT SHIFT RIGHT

Example 2

- When enabled, the BSR instruction copies array_dint[0].0 to the .UL bit
- Shifts 0-57 to the right, and loads the input_1 into array_dint[1].25
- The remaining bits (31-26 in dint_array[1]) are invalid, which indicates that the bits may not be modified
- Note how array_dint[1].0 shifts across words into array_dint[0].31



BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT



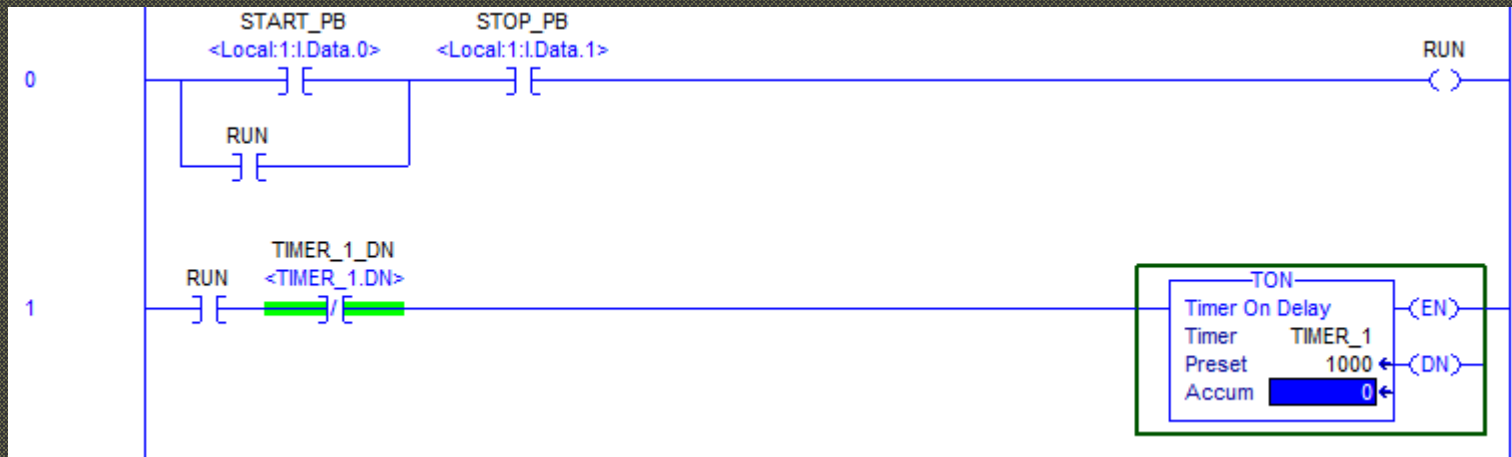
BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT

Four Windings Stepper Motor Control

Use the following link to understand the functionality of stepper motors:

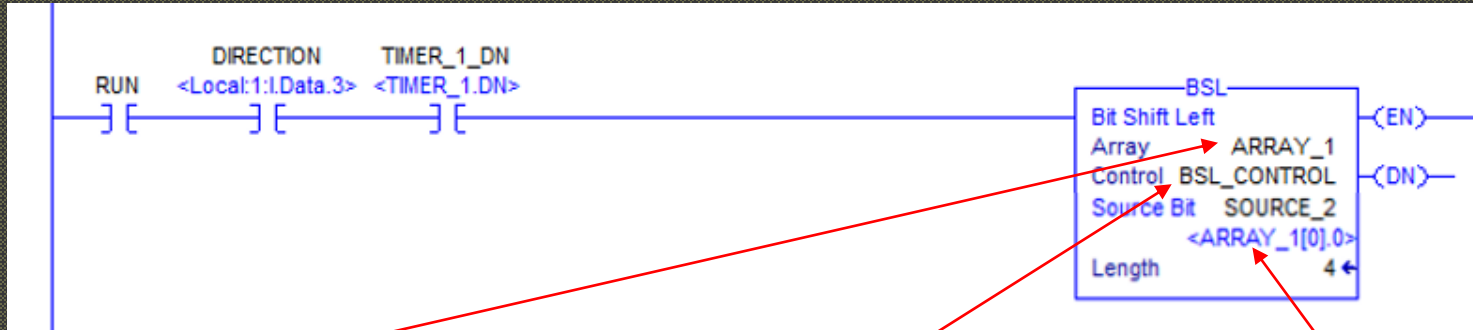
<https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/>

One second self resetting timer (frequency of rotation). Extremely slow rotation.



BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT

Four Windings Stepper Motor Control



Type digit “1” into BSL_CONTROL.UL to load digit “1” to ARRAY_1[0].0 when you press START_PB

Tag Properties - ARRAY_1

General

Name: ARRAY_1

Description:

Type: Base

Alias For:

Data Type: DINT[1]

Scope: MainProgram

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

Tag Properties - BSL_CONTROL

General

Name: BSL_CONTROL

Description:

Type: Base

Alias For:

Data Type: CONTROL

Scope: MainProgram

External Access: Read/Write

Style:

☐ Constant

OK Cancel Apply Help

Tag Properties - SOURCE_2

General

Name: SOURCE_2

Description:

Type: Alias

Alias For: ARRAY_1[0].0

Data Type: BOOL

Scope: MainProgram

External Access: Read/Write

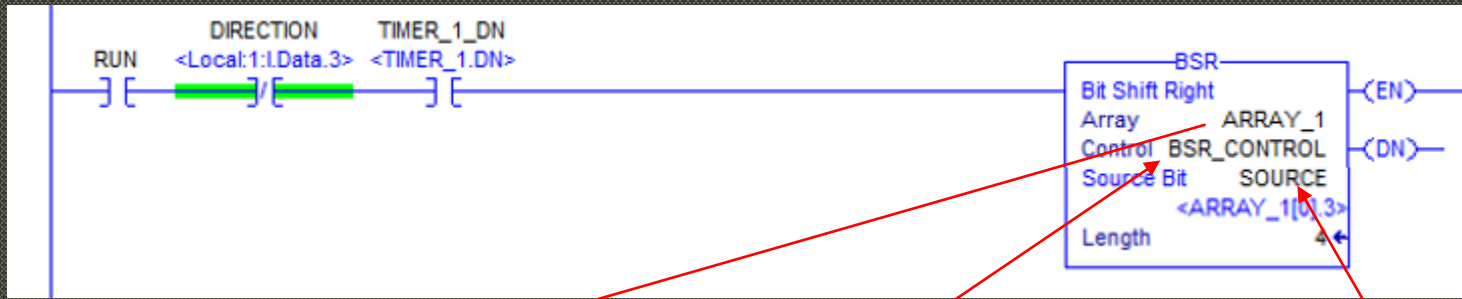
Style: Decimal

☐ Constant

OK Cancel Apply Help

BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT

Four Windings Stepper Motor Control



Same tag as for BSL

Type digit “1” into
BSR_CONTROL.UL to
load digit “1” to
ARRAY_1[0].3 when
you press START_PB

Tag Properties - BSR_CONTROL

General

Name: BSR_CONTROL

Description:

Type: Base

Alias For:

Data Type: CONTROL

Scope: Main Program

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

Tag Properties - SOURCE

General

Name: SOURCE

Description:

Type: Alias

Alias For: ARRAY_1[0..3]

Data Type: BOOL

Scope: Main Program

External Access: Read/Write

Style: Decimal

☐ Constant

OK Cancel Apply Help

BSL – BIT SHIFT LEFT AND BSR – BIT SHIFT RIGHT

Four Windings Stepper Motor Control

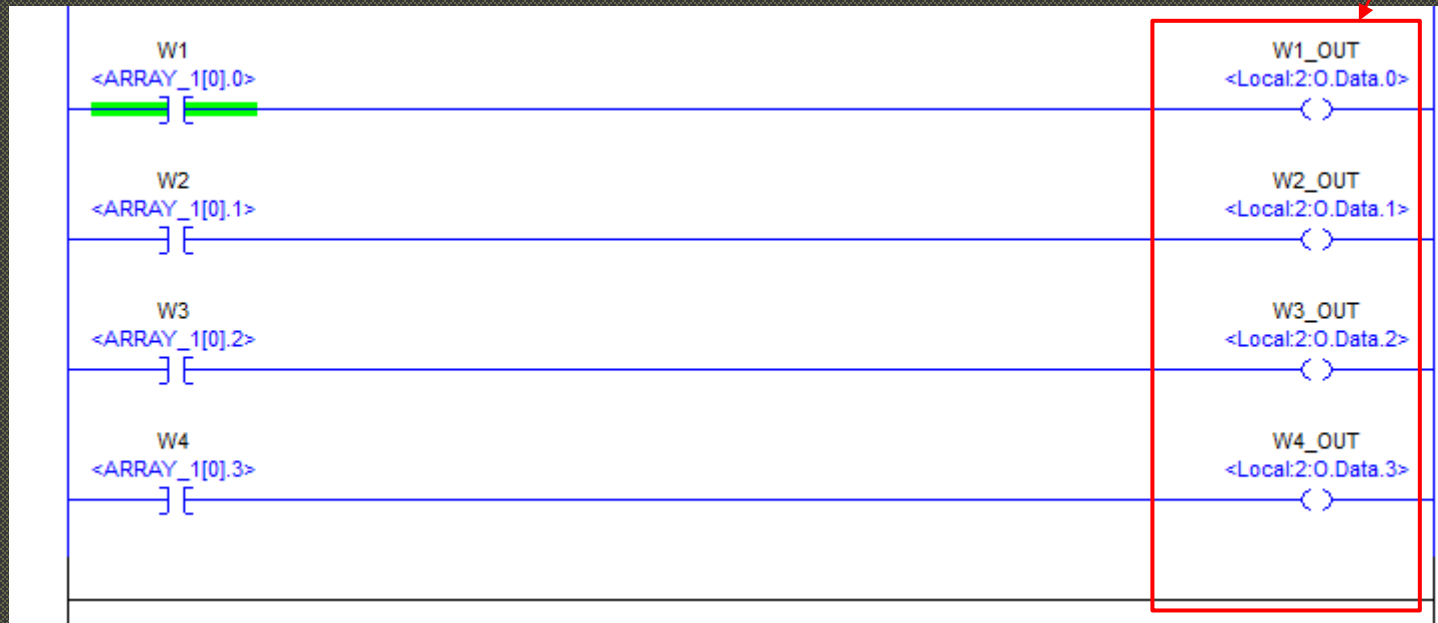
Type digit “1” into
BSL_CONTROL.UL to
load digit “1” to
ARRAY_1[0].0 when
you press START_PB



Type digit “1” into
BSR_CONTROL.UL to
load digit “1” to
ARRAY_1[0].3 when
you press START_PB



Activation of the field outputs



PROBLEMS USING SEQUENCER AND BIT SHIFT INSTRUCTIONS

Problem 1. CompactLogix PLC manages the control of an advertisement bench illumination with various combinations of 7 lights. The illumination functions in cycles each of them containing 5 steps described below. Each step has a duration of 0.5s

Step 1: Lights 1, 3, 5

Step 2: Lights 2, 4, 6

Step 3: Lights 1, 2, 3, 7

Step 4: Lights 3, 4, 6, 7

Step 5: Lights 1 and 7

- The cycle starts and stops by pressing N.O. Start and N.C. Stop pushbuttons. Both are momentary.
- Every time a program counter counts 3 cycles, the direction of the steps reverses from Step 1 – Step 7 to Step 7 – Step 1.
- Design the PLC program to control the process using SQO instruction.

Problem 2. Design the program for the Problem 1 using Sequential Control technique.

PROGRAMMABLE LOGIC CONTROLLERS

Sequential Control

Erickson, K. (2016) Programmable logic controllers: An emphasis on design and application (3rd edition). Rolla MO: Dogwood Valley Press.

Chapter 6

Petruzella, F. (2017) Programmable Logic Controllers (5th Edition). McGraw-Hill Education, NY.

Chapter 12

PROGRAMMABLE LOGIC CONTROLLERS

MENG 3500

Thank you!

Discussions?