Chapter 12 HANDLING DATA

Introduction

In this chapter are found a number of output commands that handle data. These are instructions that tend to be accepted with most PLC manufacturers as a core of instructions that accomplish a simple task and provide a useful function to the larger program.

These instructions include boolean instructions, instructions for handling simple math conversions, file manipulation instructions, queuing instructions, and instructions for bit shifting.

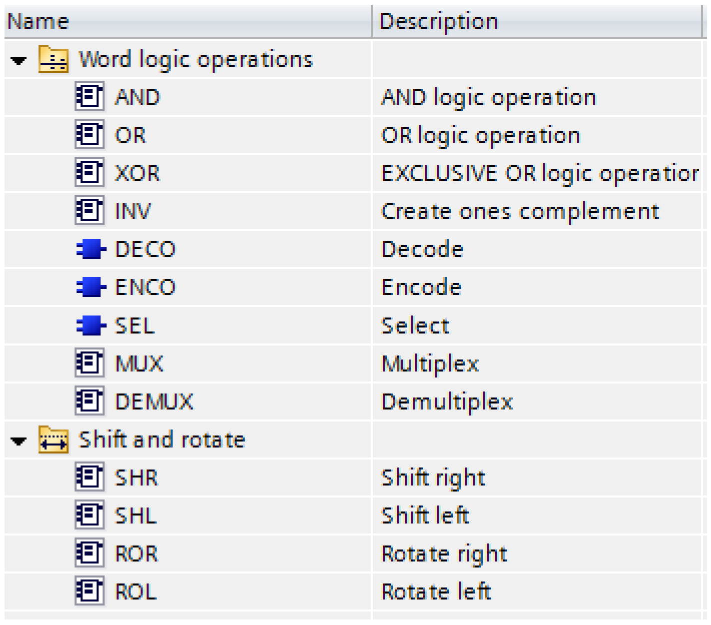
Many of these instructions were created after earlier PLCs reported some functions being performed again and again at the cost of a great amount of PLC code. For instance, the queuing operations of FIFO or LIFO could be created in PLC code using a number of other instructions. This became too confusing to the PLC programmer and the FIFO or LIFO instruction was provided as a result.

Some of the instructions came about as the result of the PLC emulating the microprocessor instruction set. Word-length boolean instructions such as AND and moving operations such as bit shifting were copied and installed in the PLC instruction set from similar microprocessor instructions.

The most important point of the chapter is that most instructions are given with an example from industry. Each of these examples has been programmed by many different programmers using techniques similar to the examples shown. Instructions such as these have been used to provide control of automation for a wide variety of industry and solve many complex problems.

Siemens Word Logic and Shift/Rotate Instructions

Instructions from Siemens are given first. They are word logic operations and shift/rotate instructions. The instructions are shown in the groups below and their operations are explained in the following pages.



Name	Description
Word logic operations	
AND	AND logic operation
OR	OR logic operation
XOR	EXCLUSIVE OR logic operatior
INV	Create ones complement
DECO	Decode
ENCO	Encode
SEL	Select
MUX	Multiplex
DEMUX	Demultiplex
Shift and rotate	
SHR	Shift right
SHL	Shift left
ROR	Rotate right
ROL	Rotate left

Fig. 12-1 Siemens Word Logic, Shift/Rotate Instructions

These instructions are covered in the following with the instruction definition given followed by, in many cases, an example. The instructions may be further described by using the 'help' menu while in the programming menu on the TIA portal.

AND

"You can use the AND logic operation instruction to combine the value at the IN1 input and the value at the IN2 input bit-by-bit by AND logic and query the result at the OUT output. When the instruction is executed, bit 0 of the value at the IN1 input and bit 0 of the value at the IN2 input are logically ANDed. The result is stored in bit 0 of the OUT output. The same logic operation is executed for all other bits of the specified values."

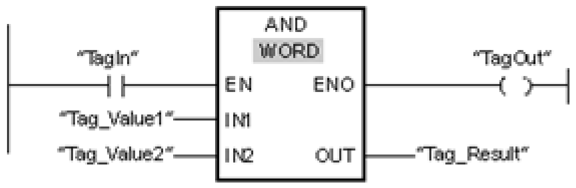


Fig. 12-2 Siemens AND Instruction

The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value
IN1	Tag_Value1	0101 0101 0101 0101
IN2	Tag_Value2	0000 0000 0000 1111
OUT	Tag_Result	0000 0000 0000 0101

OR

“You can use the OR logic operation instruction to combine the value at the IN1 input and the value at the IN2 input bit-by-bit by OR logic and query the result at the OUT output. When the instruction is executed, bit 0 of the value at the IN1 input and bit 0 of the value at the IN2 input are logically ORed. The result is stored in bit 0 of the OUT output. The same logic operation is executed for all bits of the specified tags.”

The following example shows how the instruction works:

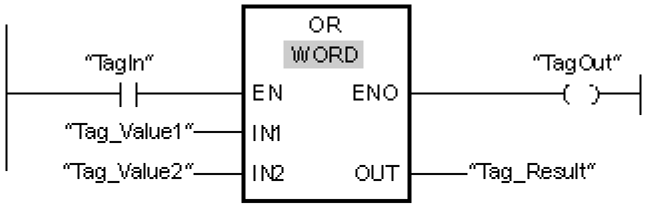


Fig. 12-3 Siemens OR Instruction

XOR

“You can use the EXCLUSIVE OR logic operation to combine the value at the IN1 input and the value at the IN2 input bit-by-bit by EXCLUSIVE OR logic and query the result at the OUT output.”

The following example shows how the instruction works:.

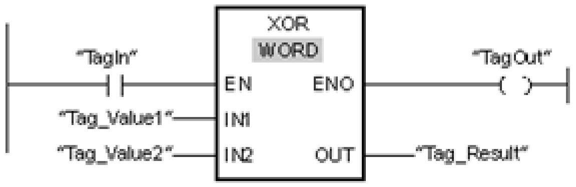


Fig. 12-4 Siemens XOR Instruction

INV

“You can use the Create ones complement instruction to invert the signal state of the bits at the IN input. When the instruction is processed, the value at the IN input and a hexadecimal template (W#16#FFFF for 16-bit numbers or DW#16#FFFF FFFF for 32-bit numbers) are logically EXCLUSIVELY ORed. As a result, the signal state of the individual bits is inverted and sent to the OUT output. The instruction is only executed if the signal state is 1 at the EN enable input. In this case, the ENO output also has the signal state 1.”

The following example shows how the instruction works:.

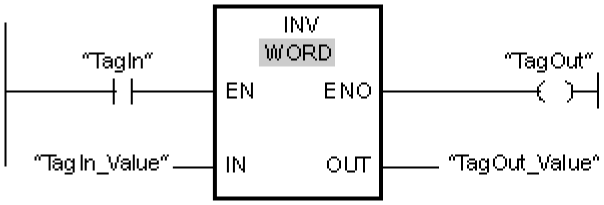


Fig. 12-5 Siemens INVERT Instruction

DECO

"You can use the Decode instruction to set a bit in the output value specified by the input value. The Decode instruction reads the value at the IN input and sets the bit in the output value whose bit position corresponds to the read value. The other bits in the output value will be overwritten with zeroes. When the value at the IN input is greater than 31, a modulo-32 instruction is executed."

The following example shows how the instruction works:

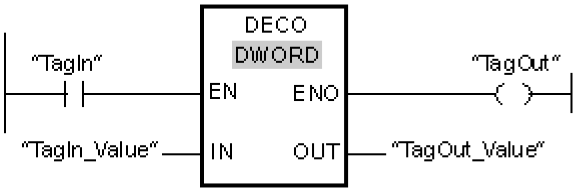
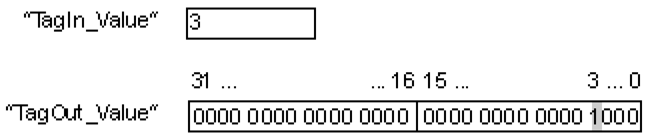


Fig. 12-6 Siemens DECODE Instruction

The following figure shows how the instruction works using specific operand values:



ENCO

"You can use the Encode instruction to read the bit number of the least significant bit in the input value and to send it to the OUT output. The Encode instruction selects the least significant bit of the value at the IN input and writes its bit number to the tag in the OUT output."

The following example shows how the instruction works:

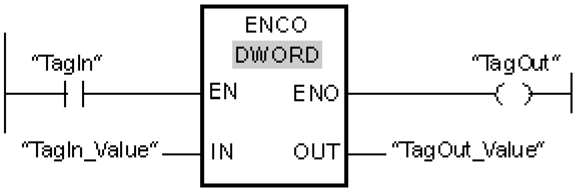
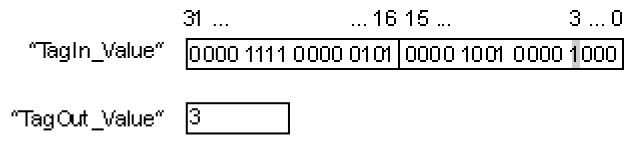


Fig. 12-7 Siemens ENCODE Instruction

The following figure shows how the instruction works using specific operand values:



SEL

"Depending on a switch (G input), the Select instruction selects one of the IN0 or IN1 inputs and copies its content to the OUT output. When the G input has the signal state 0, the value at the IN0 input is moved. When the G input has the signal state 1, the value at the IN1 input is copied to the OUT output."

The following example shows how the instruction works:

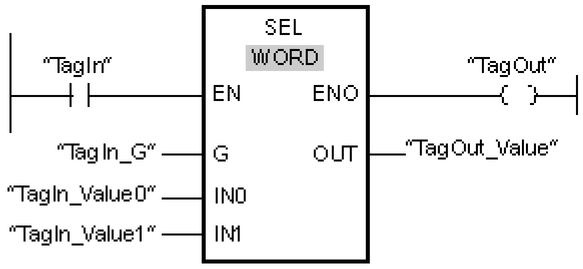


Fig. 12-8 Siemens SELECT Instruction

The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value	
G	TagIn_G	0	1
IN0	TagIn_Value0	W#16#0000	W#16#4C
IN1	TagIn_Value1	W#16#FFFF	W#16#5E
OUT	TagOut_Value	W#16#0000	W#16#5E

MUX

"You can use the Multiplex instruction to copy the content of a selected input to the OUT output. The number of selectable inputs of the instruction box can be expanded. The inputs are automatically numbered in the box. Numbering starts at IN0 and continues consecutively with each new input. You use the K parameter to define the input whose content is to be copied to the OUT output. If the value of the K parameter is greater than the number of available inputs, the content of the ELSE parameter is copied to the OUT output and the ENO enable output is assigned the signal state 0."

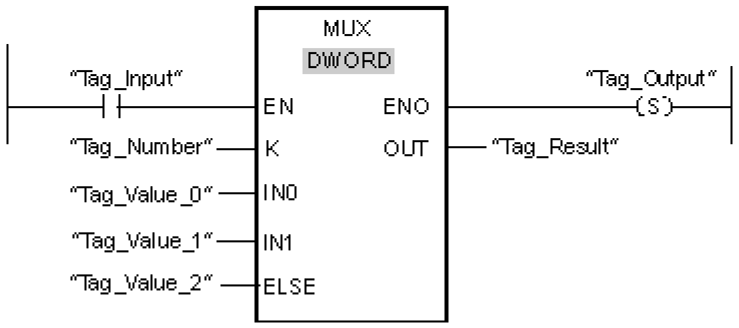


Fig. 12-9 Siemens MUX Instruction

The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value
K	Tag_Number	1
IN0	Tag_Value_0	DW#16#00000000
IN1	Tag_Value_1	DW#16#3E4A7D
ELSE	Tag_Value_2	DW#16#FFFF0000
OUT	Tag_Result	DW#16#3E4A7D

DEMUX

"You can use the Demultiplex instruction to copy the content of the IN input to a selected output. The number of selectable outputs can be expanded in the instruction box. The outputs are automatically numbered in the box. Numbering starts at OUT0 and continues consecutively with each new input. You use the K parameter to define the output to which the content of the IN input is to be copied. The other outputs are not changed. If the value of the K parameter is greater than the number of available outputs, then the content of the IN input will be copied to the ELSE parameter and the signal state 0 is assigned to the ENO enable output."

The following example shows how the instruction works.

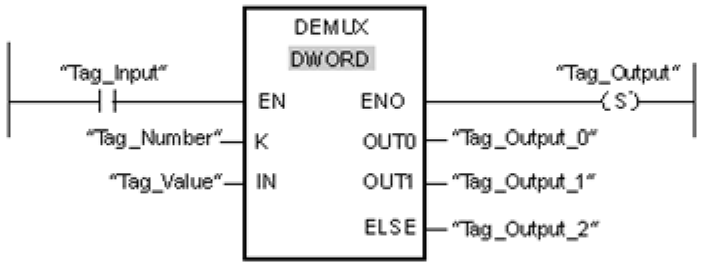


Fig. 12-10 Siemens DEMUX Instruction

The following table shows how the instruction works using specific operand values:

Input values of the Demultiplex instruction **before** network execution:

Parameters	Operand	Values	
K	Tag_Number	1	4
IN	Tag_Value	DW#16#FFFFFFFF	DW#16#3E4A7D

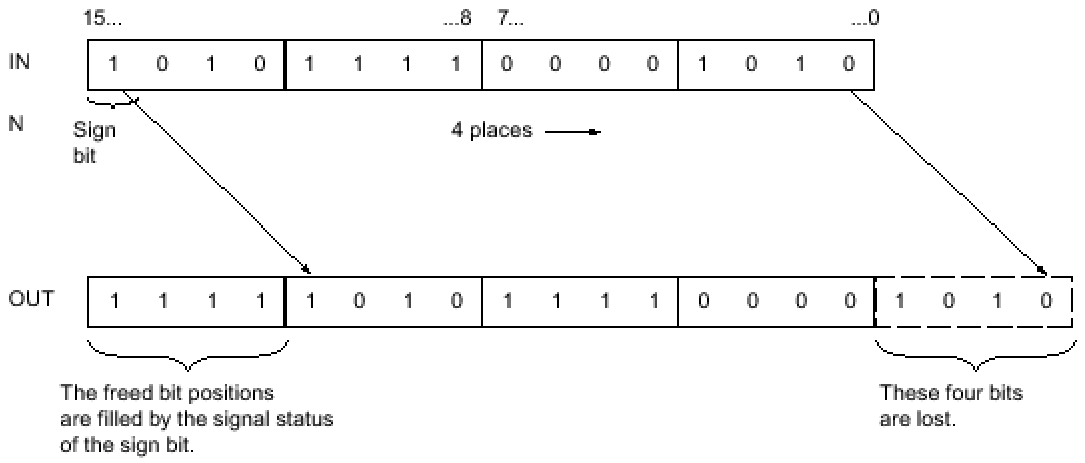
Output values of the Demultiplex instruction **after** network execution:

Parameters	Operand	Values	
OUT0	Tag_Output_0	Unchanged	Unchanged
OUT1	Tag_Output_1	DW#16#FFFFFFFF	Unchanged
ELSE	Tag_Output_2	Unchanged	DW#16#3E4A7D

SHR

"You can use the Shift right instruction to shift the content of the operand at the IN input bit-by-bit to the right and query the result at the OUT output. You use the N parameter to specify the number of bit positions by which the specified value is to be shifted."

The following figure show how the content of an operand of integer data type is shifted four bit positions to the right:



The following example shows how the instruction works:

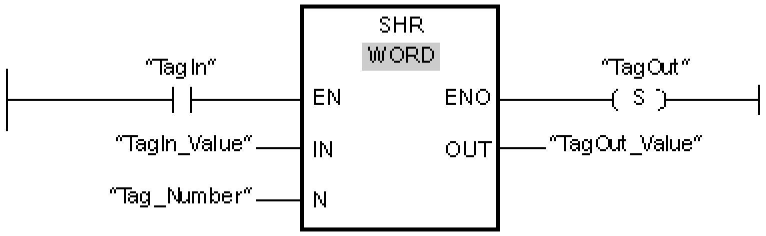


Fig. 12-11 Siemens SHIFT RT Instruction

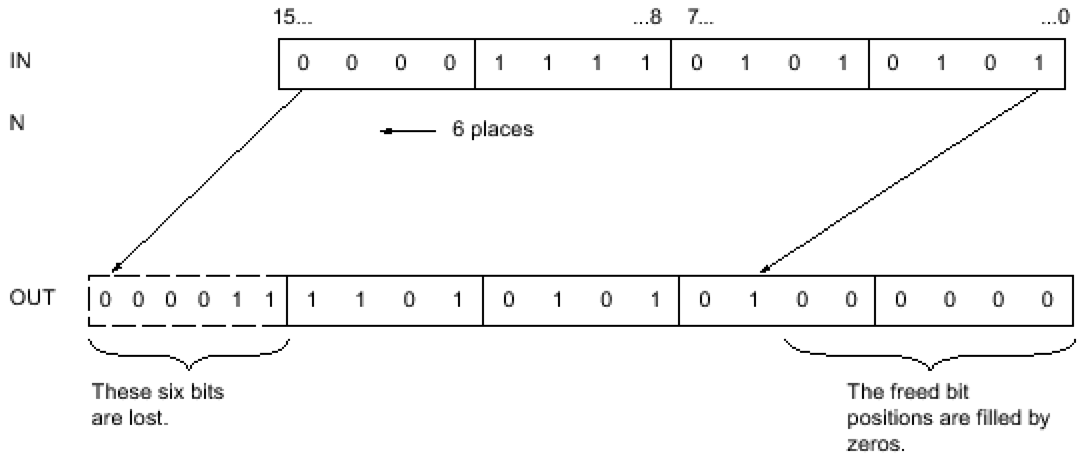
The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value
IN	TagIn_Value	0011 1111 1010 1111
N	Tag_Number	3
OUT	TagOut_Value	0000 0111 1111 0101

SHL

"You can use the Shift left instruction to shift the content of the operand at the IN input bit-by-bit to the left and query the result at the OUT output. You use the N parameter to specify the number of bit positions by which the specified value is to be shifted. When the value at the N parameter is 0, the value at the IN input is copied to the operand at the OUT output."

following figure shows how the content of an operand of WORD data type is shifted six bit positions to the left:



The following example shows how the instruction works:

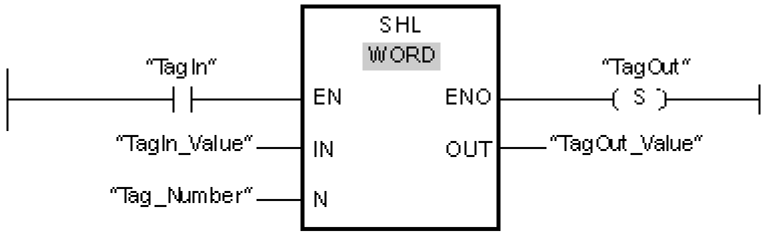


Fig. 12-12 Siemens SHIFT LT Instruction

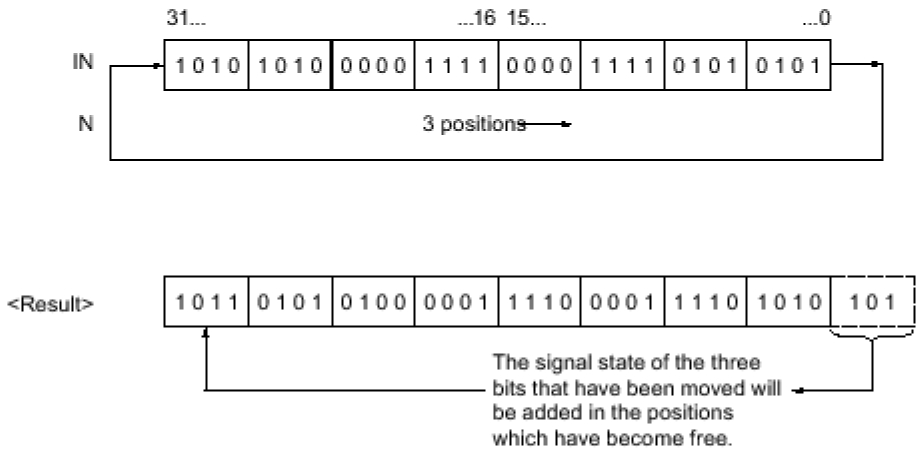
The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value
IN	TagIn_Value	0011 1111 1010 1111
N	Tag_Number	4
OUT	TagOut_Value	1111 1010 1111 0000

ROR

“The Rotate right instruction rotates the content of the operand at the IN input bit-by-bit to the right and queries the result at the OUT output. You use the N parameter to specify the number of bit positions by which the specified value is to be rotated. The bit positions freed by rotating are filled with the bit positions that are pushed out.”

The following figure shows how the content of an operand of DWORD data type is rotated three positions to the right:



The following example shows how the instruction works:

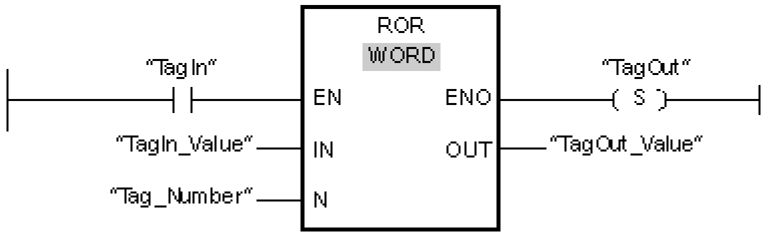


Fig. 12-13 Siemens ROTATE RT Instruction

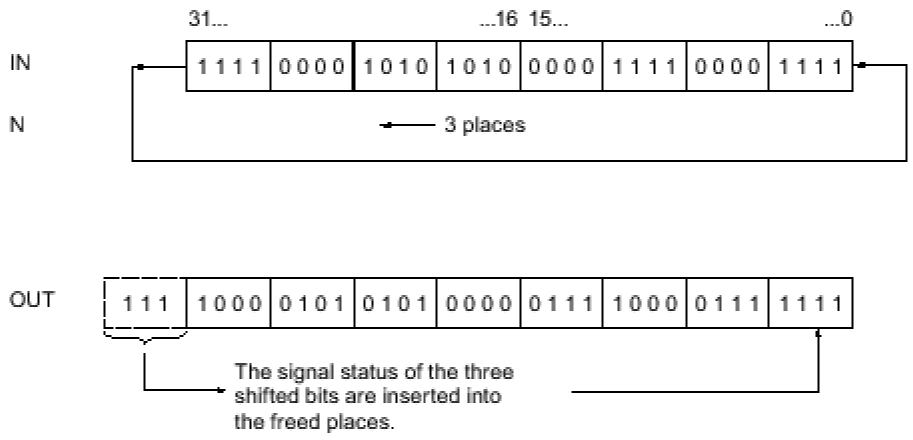
The following table shows how the instruction works using specific operand values:

Parameters	Operand	Value
IN	TagIn_Value	0011 1111 1001 0101
N	Tag_Number	5
OUT	TagOut_Value	1010 1000 0111 1100

ROL

“The Rotate left instruction rotates the content of the operand at the IN input bit-by-bit to the left and queries the result at the OUT output. You use the N parameter to specify the number of bit

positions by which the specified value is to be rotated. The bit positions freed by rotating are filled with the bit positions that are pushed out.”
The following figure shows how the content of an operand of DWORD data type is rotated three positions to the left:



The following example shows how the instruction works:

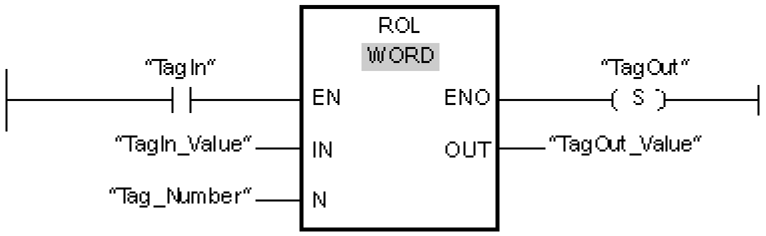
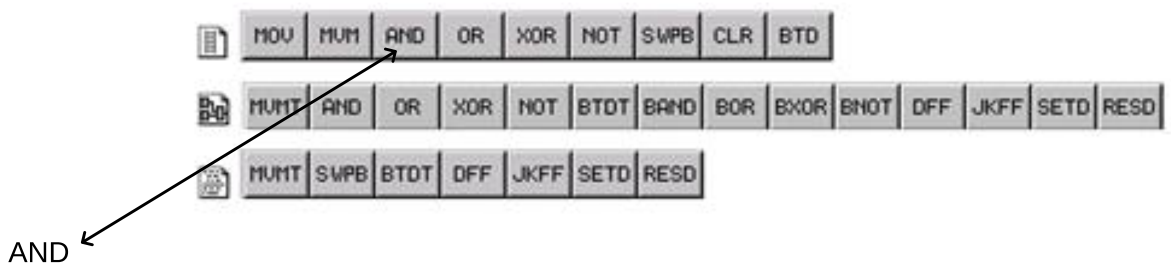


Fig. 12-14 Siemens ROTATE LT Instruction

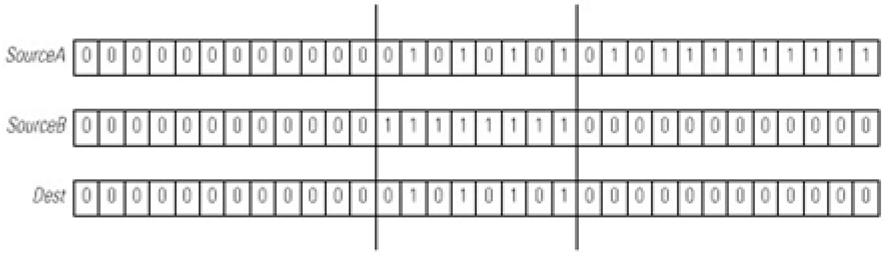
The following table shows how the instruction works using specific operand values:

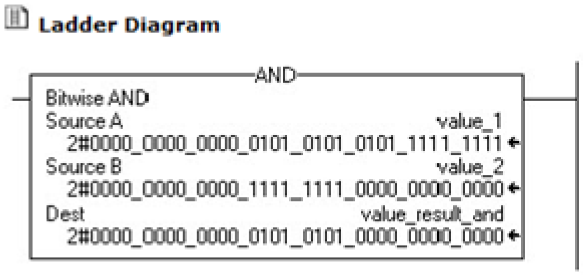
Parameters	Operand	Value
IN	TagIn_Value	1010 1000 1111 0110
N	Tag_Number	5
OUT	TagOut_Value	0001 1110 1101 0101

Allen-Bradley Logical Instructions



When enabled, the AND instruction performs a bitwise AND operation on SourceA and SourceB and places the result in Dest.



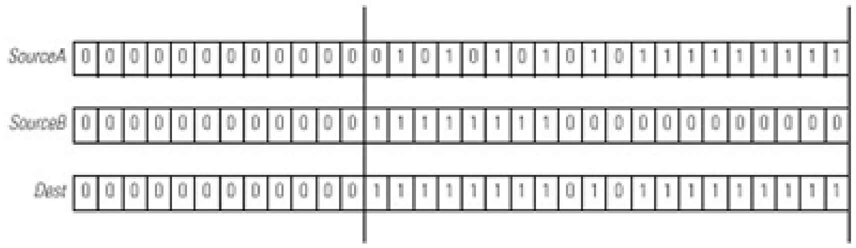


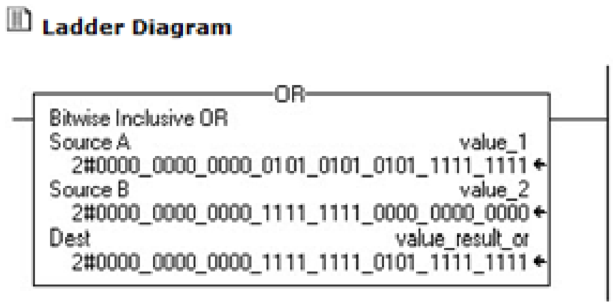
```
Ladder Diagram

AND
Bitwise AND
Source A                              value_1
   2#0000_0000_0000_0101_0101_0101_1111_1111
Source B                              value_2
   2#0000_0000_0000_1111_1111_0000_0000_0000
Dest                         value_result_and
   2#0000_0000_0000_0101_0101_0000_0000_0000
```

Fig. 12-15 A-B AND Instruction

OR

When enabled, the OR instruction performs a bitwise OR operation on SourceA and SourceB and places the result in dest.



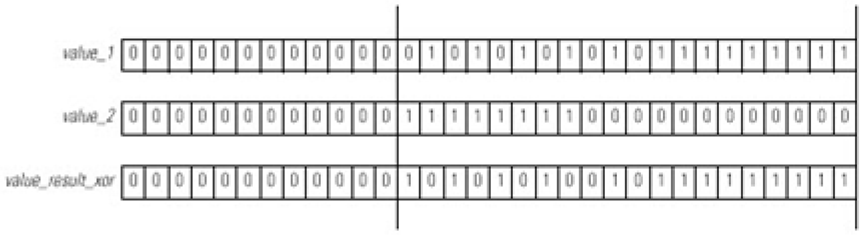


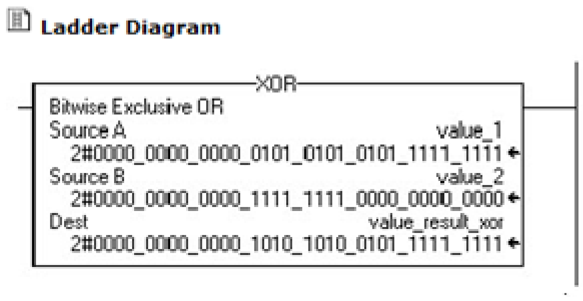
```
Ladder Diagram

OR
Bitwise Inclusive OR
Source A                              value_1
   2#0000_0000_0000_0101_0101_0101_1111_1111
Source B                              value_2
   2#0000_0000_0000_1111_1111_0000_0000_0000
Dest                          value_result_or
   2#0000_0000_0000_1111_1111_0101_1111_1111
```

Fig. 12-16 A-B OR Instruction

XOR

When enabled, the XOR instruction performs a bitwise XOR operation on SourceA and SourceB and places the result in the destination tag.


value_1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1
value_2 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
value_result_xor 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1

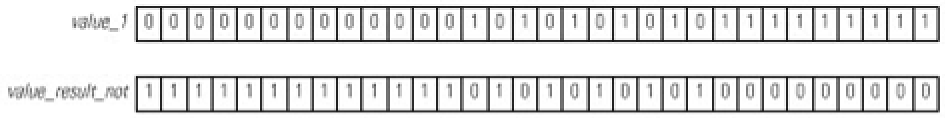


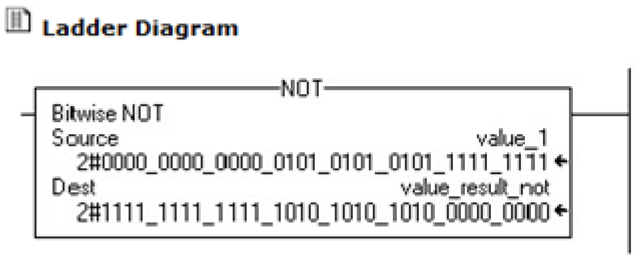
```
Ladder Diagram
XOR
Bitwise Exclusive OR
Source A                                  value_1
  2#0000_0000_0000_0101_0101_0101_1111_1111
Source B                                  value_2
  2#0000_0000_0000_1111_1111_0000_0000_0000
Dest                             value_result_xor
  2#0000_0000_0000_1010_1010_0101_1111_1111
```

Fig. 12-17 A-B XOR Instruction

NOT

When enabled, the NOT instruction performs a bitwise NOT operation on value_1 and places the result in value_result_not.


value_1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1
value_result_not 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0



```
Ladder Diagram
NOT
Bitwise NOT
Source                                    value_1
  2#0000_0000_0000_0101_0101_0101_1111_1111
Dest                             value_result_not
  2#1111_1111_1111_1010_1010_1010_0000_0000
```

Fig. 12-18 A-B NOT Instruction

Swap Byte – SWPB

The three SWPB instructions each reorder the bytes of DINT_1 according to a different order mode. The display style is ASCII, and each character represents one byte. Each instruction places the bytes, in the new order, in a different Destination.

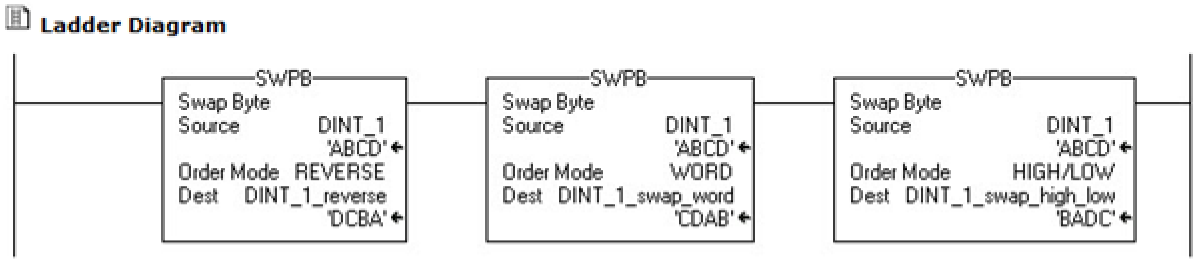


Fig. 12-19 A-B Swap Byte Instructions

Clear – CLR

Clear all the bits of value to 0.

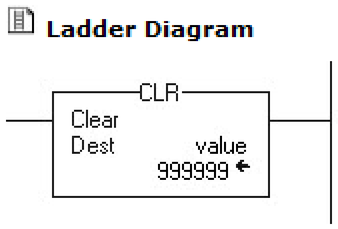


Fig. 12-21 A-B CLR Instruction

Bit Field Distribute (BTD)

"When enabled, the BTD instruction copies a group of bits from the Source to the Destination. The group of bits is identified by the Source bit (lowest bit number of the group) and the Length (number of bits to copy). The Destination bit identifies the lowest bit number bit to start with in the Destination. The Source remains unchanged."

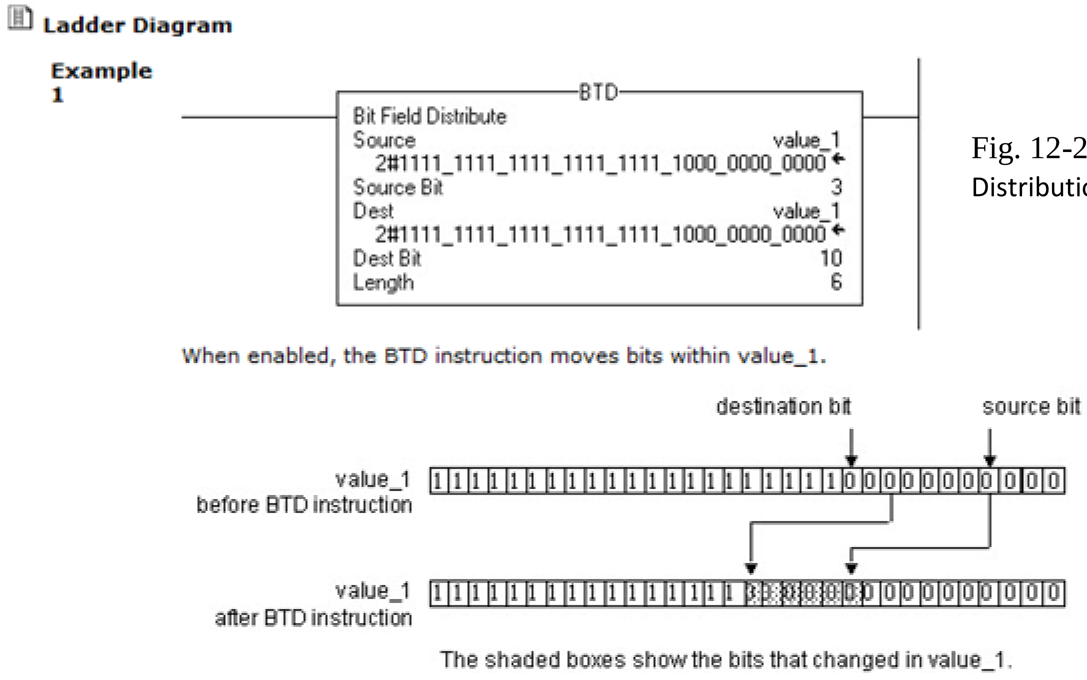
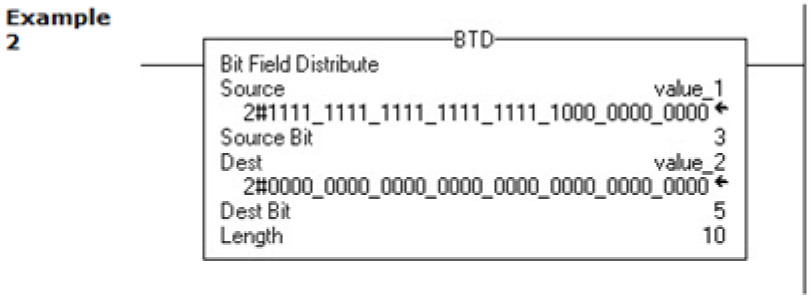


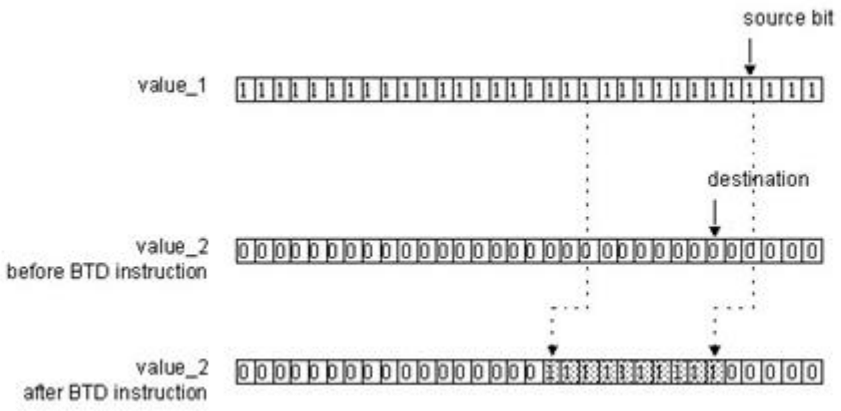
Fig. 12-22a A-B Bit Field Distribution (BTD) Instruction


Example 2

BTD
Bit Field Distribute
Source value_1
2#1111_1111_1111_1111_1111_1000_0000_0000
Source Bit 3
Dest value_2
2#0000_0000_0000_0000_0000_0000_0000_0000
Dest Bit 5
Length 10

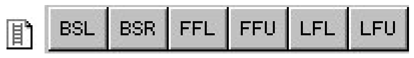
Fig. 12-22b A-B Bit Field Distribution (BTD) Instruction

When enabled, the BTD instruction moves 10 bits from value_1 to value_2.



The shaded boxes show the bits that changed in value_2.

Allen-Bradley Array/Shift Instructions



Bit Shift Left (BSL)

“The BSL instruction shifts the specified bits within the Array one position left.”

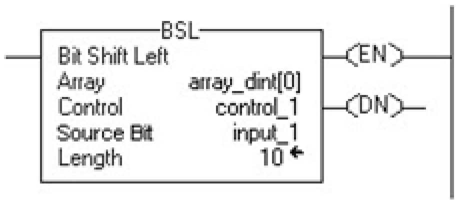
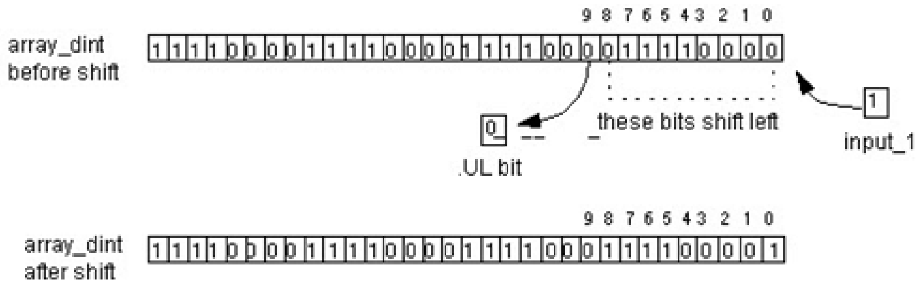

BSL
Bit Shift Left
Array array_dint[0]
Control control_1
Source Bit input_1
Length 10
EN
DN

Fig. 12-23 A-B Bit Shift Left (BSL) Instruction



Bit Shift Right (BSR)

“The BSR instruction shifts the specified bits within the Array one position right.”

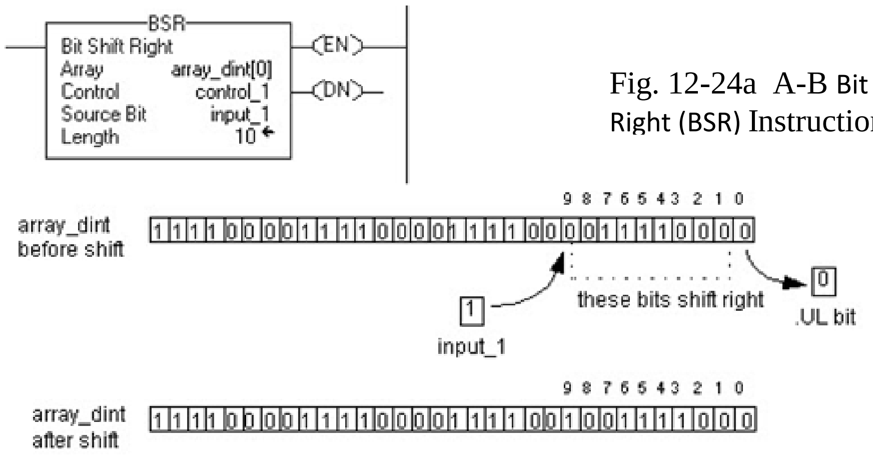


Fig. 12-24a A-B Bit Shift Right (BSR) Instruction

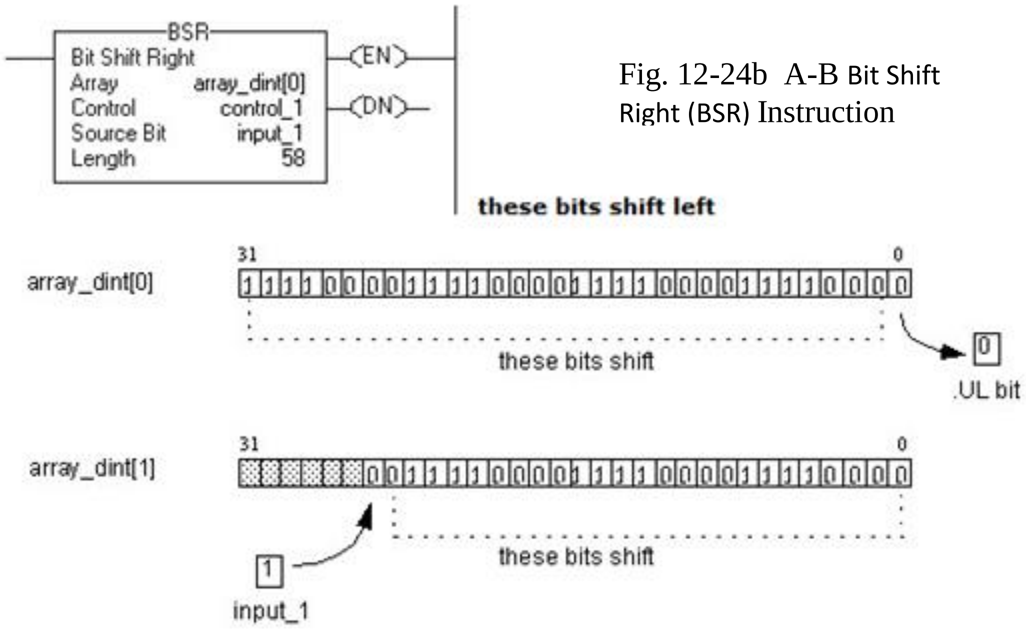


Fig. 12-24b A-B Bit Shift Right (BSR) Instruction

FIFO Load (FFL)

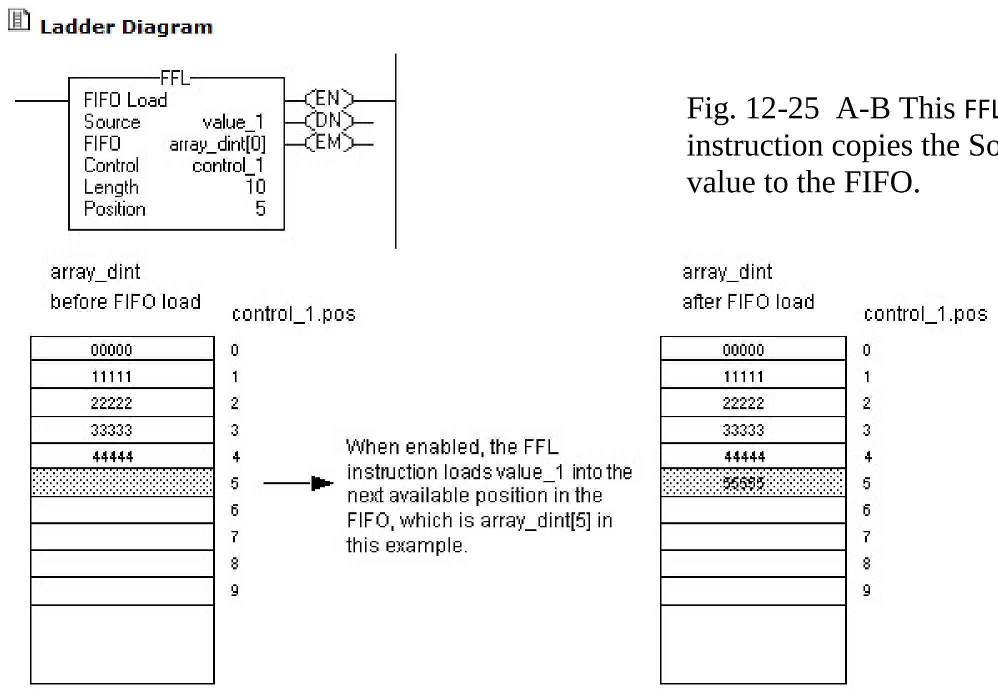


Fig. 12-25 A-B This FFL instruction copies the Source value to the FIFO.

FIFO Unload (FFU)

"The FFU instruction unloads the value from position 0 (first position) of the FIFO and stores that value in the Destination. The remaining data in the FIFO shifts down one position."

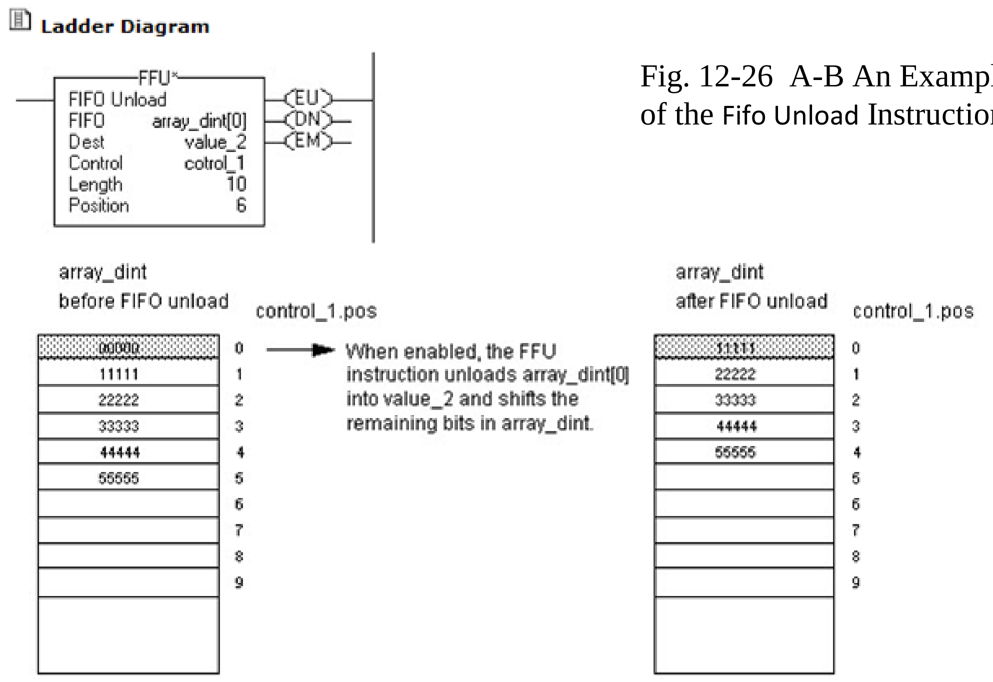


Fig. 12-26 A-B An Example of the Fifo Unload Instruction

LIFO Load (LFL)

"The LFL instruction copies the Source value to the LIFO."

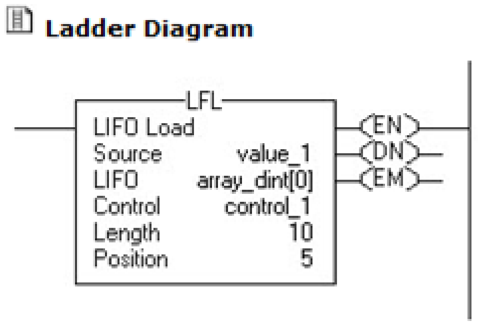
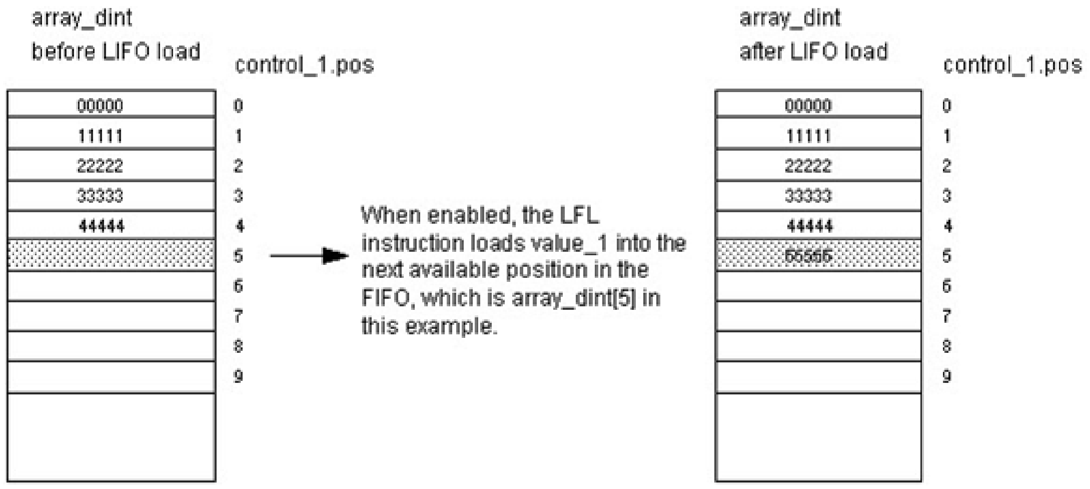


Fig. 12-27 A-B An Example of the Lifo Load Instruction



LIFO Unload (LFU)

"The LFL instruction unloads the value at .POS of the LIFO and stores 0 in that location."

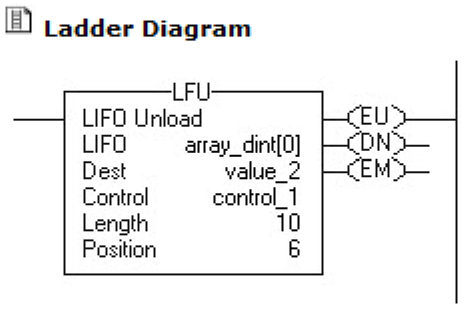
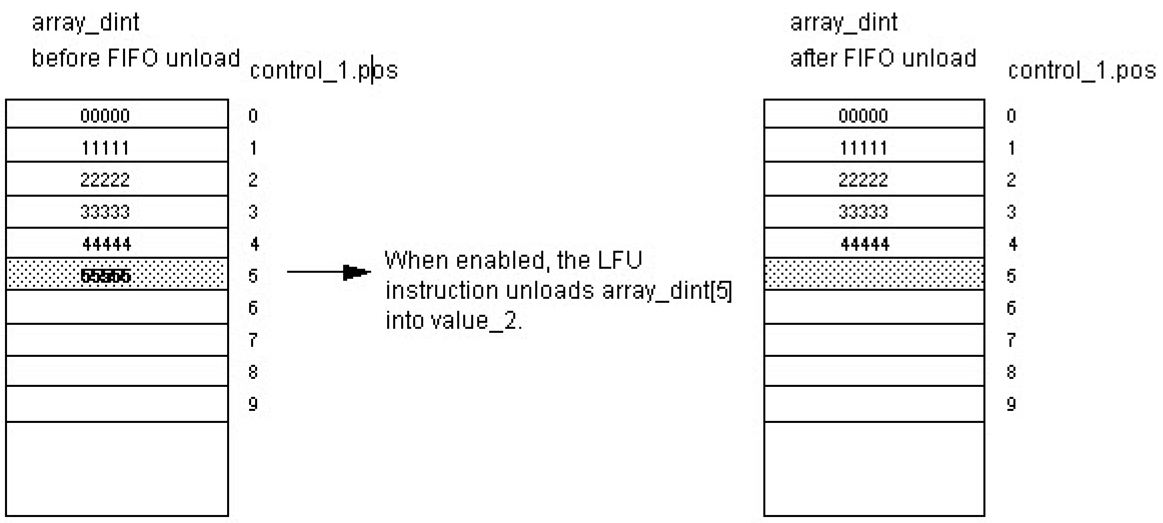


Fig. 12-28 A-B An Example of the Lifo Unload Instruction



Queueing Operations

If a queueing operation is needed in a process, consider using the FIFO or LIFO instructions. FIFO is short for First in, First out. LIFO is short for Last in, First out. Both refer to queueing of a part or entity for use in a stage of a machine. For instance, consider the following batching system using the FIFO instruction:

Start with the Mix System empty and all Process Tanks A-F full:

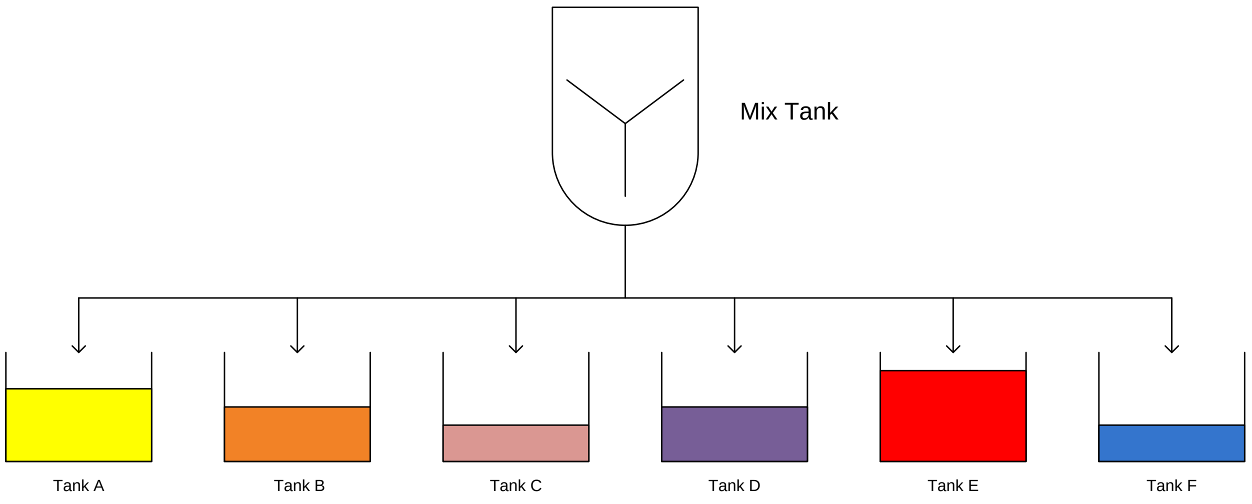


Fig. 12-29a Mix System Requiring Queueing

The system above consists of 6 processes that are filled separately with a liquid made in the Mix System tank. Tanks A-F call for a new batch of liquid when down to ¼ full. If the Mix System is in the process of making a batch for a particular tank, the tank that just fell less than ¼ full must wait for its batch to be made. Since batches are unique for each Tank A-F, the system must completely drain, go through a flush cycle, and then start mixing for the next tank. Multiple tanks may call for a batch but no tank may call more than once.

After some time passes, the tank levels lower due to the demand from the processes needing the liquids. The first one to fall to its low level switch calls for a new batch to be made. That tank's request is entered into the FIFO stack and immediately taken from the FIFO stack to the recipe program since the Mix System tank is not active.

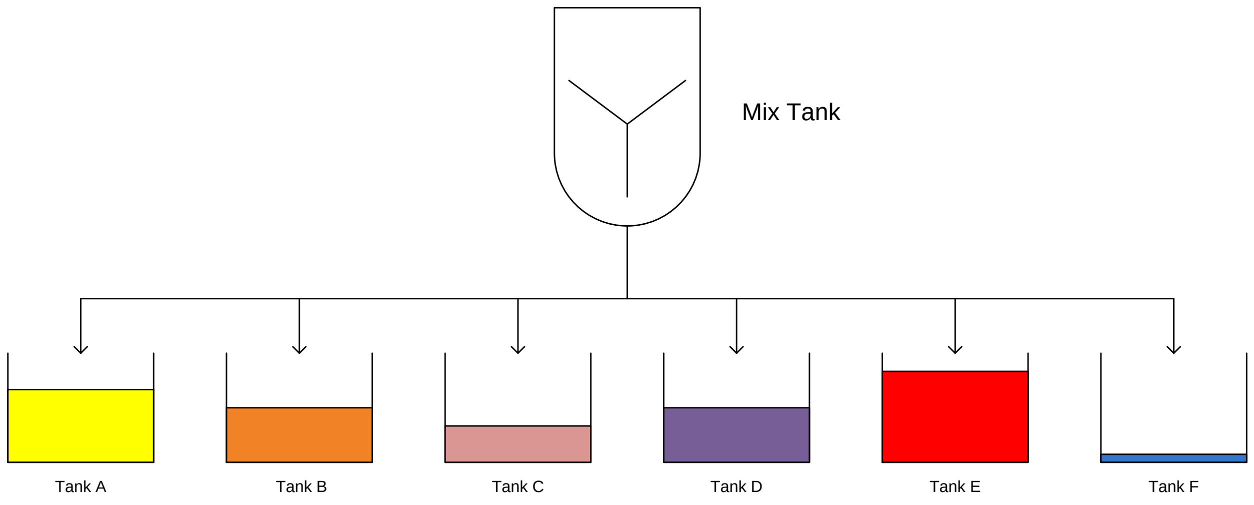
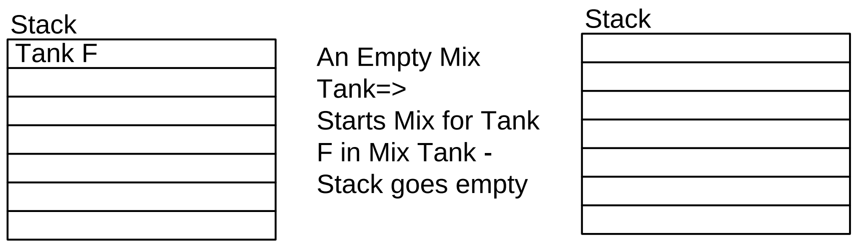


Fig. 12-29b Mix System – One Tank Low – Request to Stack


Stack
Tank F
An Empty Mix Tank=> Starts Mix for Tank F in Mix Tank - Stack goes empty
Stack

Tank F requests a batch be made. Tank F's request enters the FIFO stack. Since the batching program is not running, Tank F's request is immediately acted on by the batching program and a new batch starts in the Mix System for Tank F. The stack is emptied.

A mix is being made for Tank F. Other tanks continue to drop but no tanks have called for a new mix. The Stack continues to remain empty.

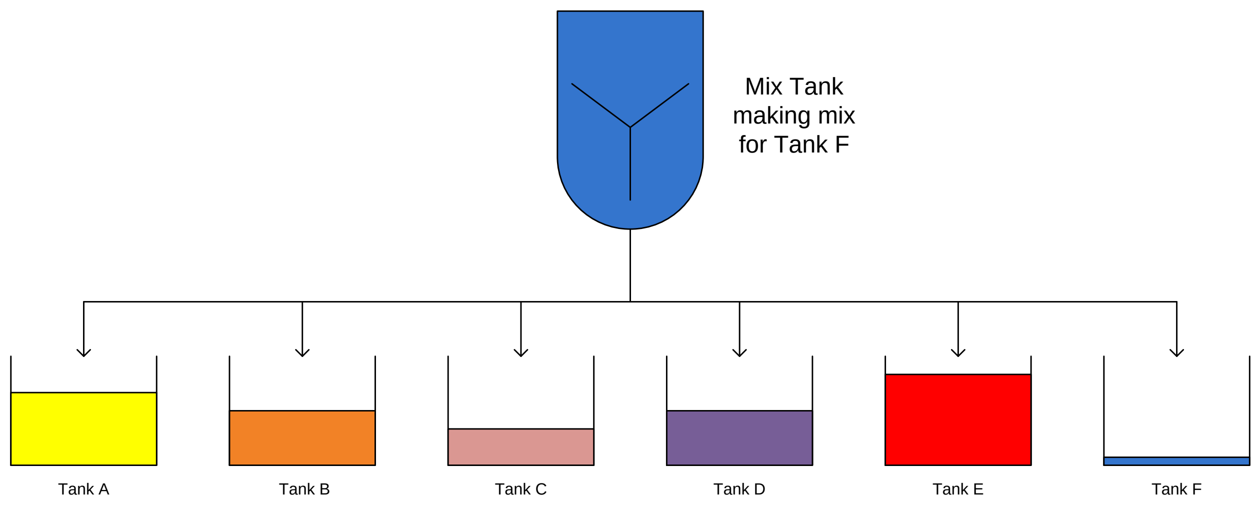
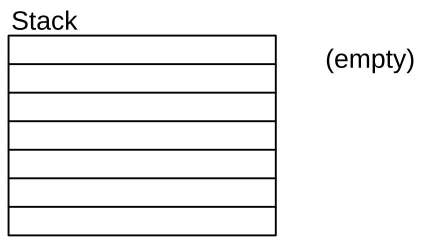


Fig. 12-29c Mix System – Mix for Tank F Starts



While the batch is being mixed for F, two other tanks reach their low tank limit and enter the stack. They enter in the order that they reached their low limit. The following shows the stack with Tank B and Tank D in the same order as the tanks went low.

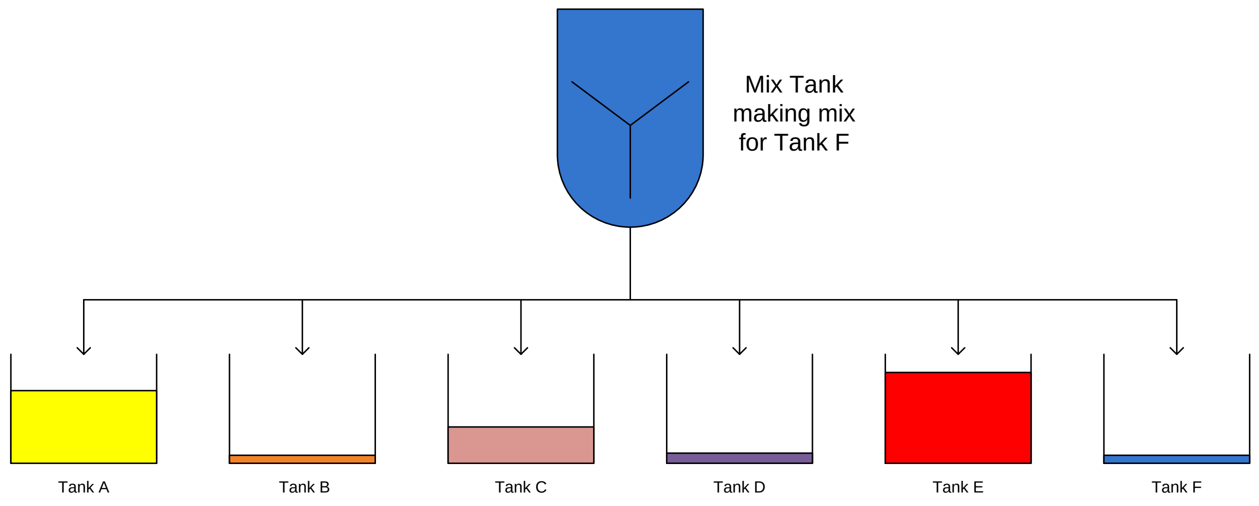


Fig. 12-29d Mix System – Tanks B and D to Stack

Stack

Tank B	(first)
Tank D	(second)

As soon as the batch for Tank F is made and emptied, the request for Tank B is taken using the FIFO unload and the mix system starts making a batch for Tank B. Tank D moves to the top of the stack.

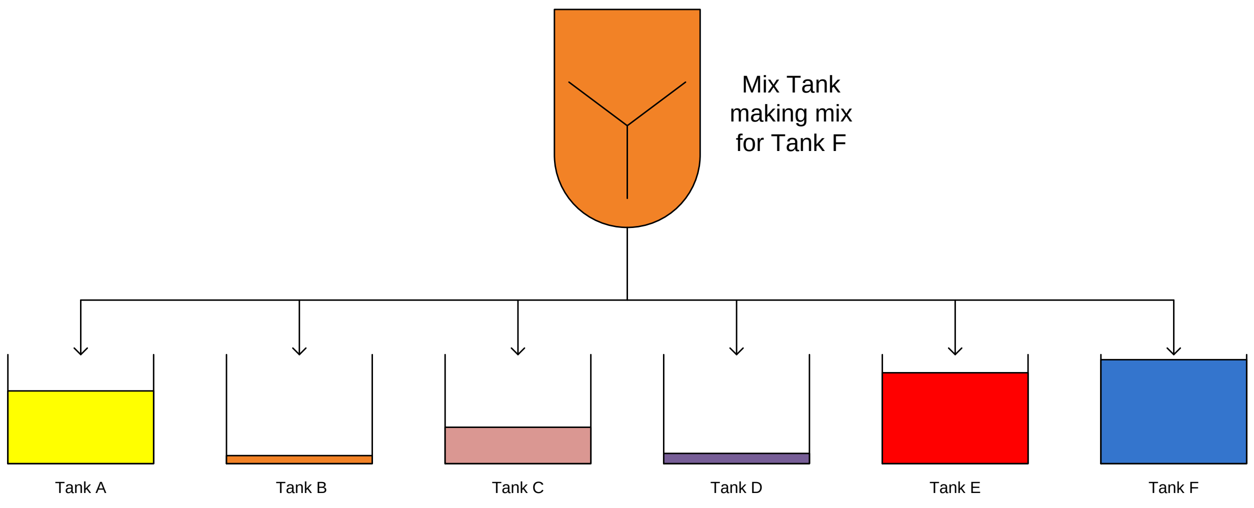


Fig. 12-29e Mix System – Tank F Completes, B Starts, D Moves Up Stack

Tank F Mix completes =>
Tank B Mix Starts
Tank D moves up stack

Stack

Tank D	(first)

While the process continues supplying mix to each tank as needed, the FIFO stack will continue to control the queueing operation. The FIFO controls the operation.

An Aside on Simulation

The various scenarios on queuing can be planned and implemented and the process not run at all as planned. It cannot always be predicted when a bottleneck will appear so care must be taken before the tanks are designed and the process is built to predict whether the various processes will work as advertised. It is not acceptable to build the process only to tear it out and add equipment in a few months based on observed bottlenecks. Computer simulation may be a necessary component of the design of a process. It may be required to size equipment and predict flows in order to maximize the manufacturer's investment.

With a good software simulation package, the user can adequately predict flow rates and equipment throughput efficiently enough to size the equipment for the process. Not all processes must be simulated but those with queues in them are more apt to need a good simulation than other types of processes.

The simulation becomes an experiment in which as much as is known about the process is introduced and the various parts are combined to run a model of the process. The model is constructed of elements that act the role of machine parts and the final product is assumed to contain all elements of the process involved. Statistics play a role in the successful simulation. If a rate is not known, a random variable may be substituted. If a rate is known, it is used in the simulation. Better than a predictable rate is a rate and a statistical distribution of the rate. With this information, a process may be seen to not fail under most circumstances but if a number of entities in the model have statistically long manufacturing times or high fail rates, the process may react differently than at the average. If this is the case, bottlenecks may spoil the overall throughput and cause the engineer to re-think the design.

The use of sophisticated simulation software with its statistical prediction algorithms will predict bottlenecks not necessarily predicted by a simpler analysis.

Using common sense is also a tool to be used. For example, if the loading rate of a part can be determined, the operator should not be allowed to make more of that product per unit of time than can be off-loaded from a manufacturing line. If rules such as this are not demanded in the control strategy, chaos is as predictable as snow in Toledo in January. So, the best plan of a designed queuing strategy may fall apart if the process is not adequately designed and perhaps a software simulation tried to predict failure and future bottlenecks. After-the-fact programming on the PLC cannot compensate for a design failure in this area. Do not be surprised if you are asked to try, however, when this happens.

Books on the theory of queuing may also be explored but, unless the process is very simple, the process is better served with a simulation.

Application Specific Instructions

The other instruction type is BSL and BSR. These instructions will be reviewed here. Although similar to other Data Handling Instructions, they are used for a specific type of application.

BSL and BSR instructions are used primarily as shift register functions, tracking parts down conveyors.

Definition of BSL and BSR (from the Reference Manual):

Instruction Mnemonic	Name	Purpose
BSL	Bit Shift Left	Loads a bit of data into a bit array, shifts the pattern of data through the array, and unloads the last bit of data in the array. BSL shifts data to the left.
BSR	Bit Shift Right	Loads a bit of data into a bit array, shifts the pattern of data through the array, and unloads the last bit of data in the array. BSR shifts the data to the right.

An Application using BSL or BSR:

If a conveyor is installed in a system, the BSL or BSR is used to track pieces on the conveyor (if the piece does not slide or scoot on the conveyor).

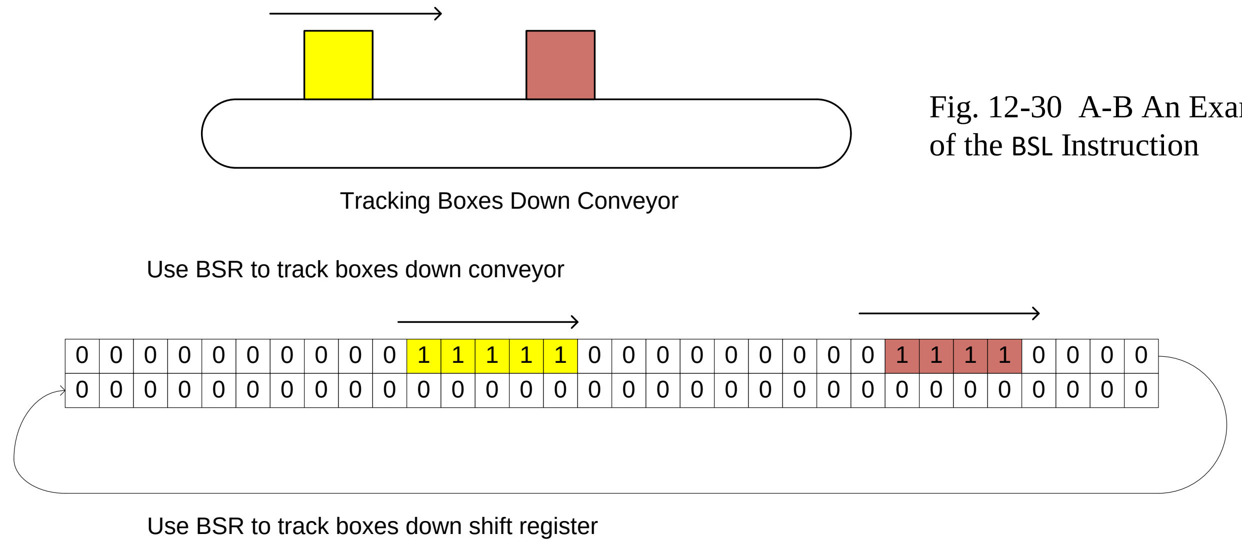


Fig. 12-30 A-B An Example of the BSL Instruction

To move the bits through the shift register requires an input pulse signal. Bits are shifted with each leading edge of the BSR or BSL. For example, if an input watched the sprocket of a conveyor as the conveyor turned, the input would pulse on and off as the conveyor's sprocket rotated. Proximity switches work well for this application. Bits in the shift register move each time the input from the proximity switch activates the BSR or BSL instruction.

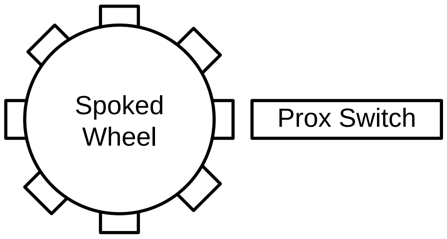


Fig. 12-31 Movement Sensor for Shift Register

With each new leading edge of the proximity switch signal, the bits shift one place in the shift register.

Each type of product needing to be tracked needs a separate shift register. In the example above, in order to track yellow boxes separately from pink boxes, the bit present in one shift register must represent yellow boxes. Same position bits in a second shift register must represent pink boxes.

Input into the shift register of 1's come from the Bit Address. Usually this address is an input address but it may represent the result of logic as well. For instance, a photo-eye is usually the device used to see a box prior to introduction onto a conveyor. If the box is bar-coded, entrance into a shift register requires the successful read of a bar code and the photo-eye seeing the product. Both must be present to turn on the input to the Bit Address.

The shift register works equally well with paint applications. Moving parts down a conveyor to paint poses the requirement that no sensor can be placed immediately in the paint area since paint can cover and make inoperable most types of presence sensors. Parts must be tracked to the paint area and painted with various colors based on the process requirement. For instance, pink parts are painted at station A and yellow parts at station B. If a part is to be painted pink, it is entered in the pink shift register. If the part is to be painted yellow, it is entered in the yellow shift register.

Pink Part Shift Register

Movement of bits $\longrightarrow$ Bit moves from bit 0 of word 0 to bit 15 of word 1.

```
0  0  0  0  0  0  1  1  1  1  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Yellow Part Shift Register

```
0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1
1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  1  1  1  1  0  0  0  0  0  0  0
```

Paint Application:

Parts on Chain:
Movement of parts ⟶

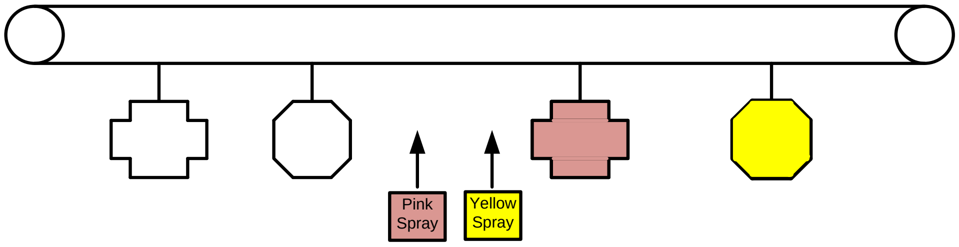


Fig. 12-32 Multiple Types of Parts Tracked through Spray Booth

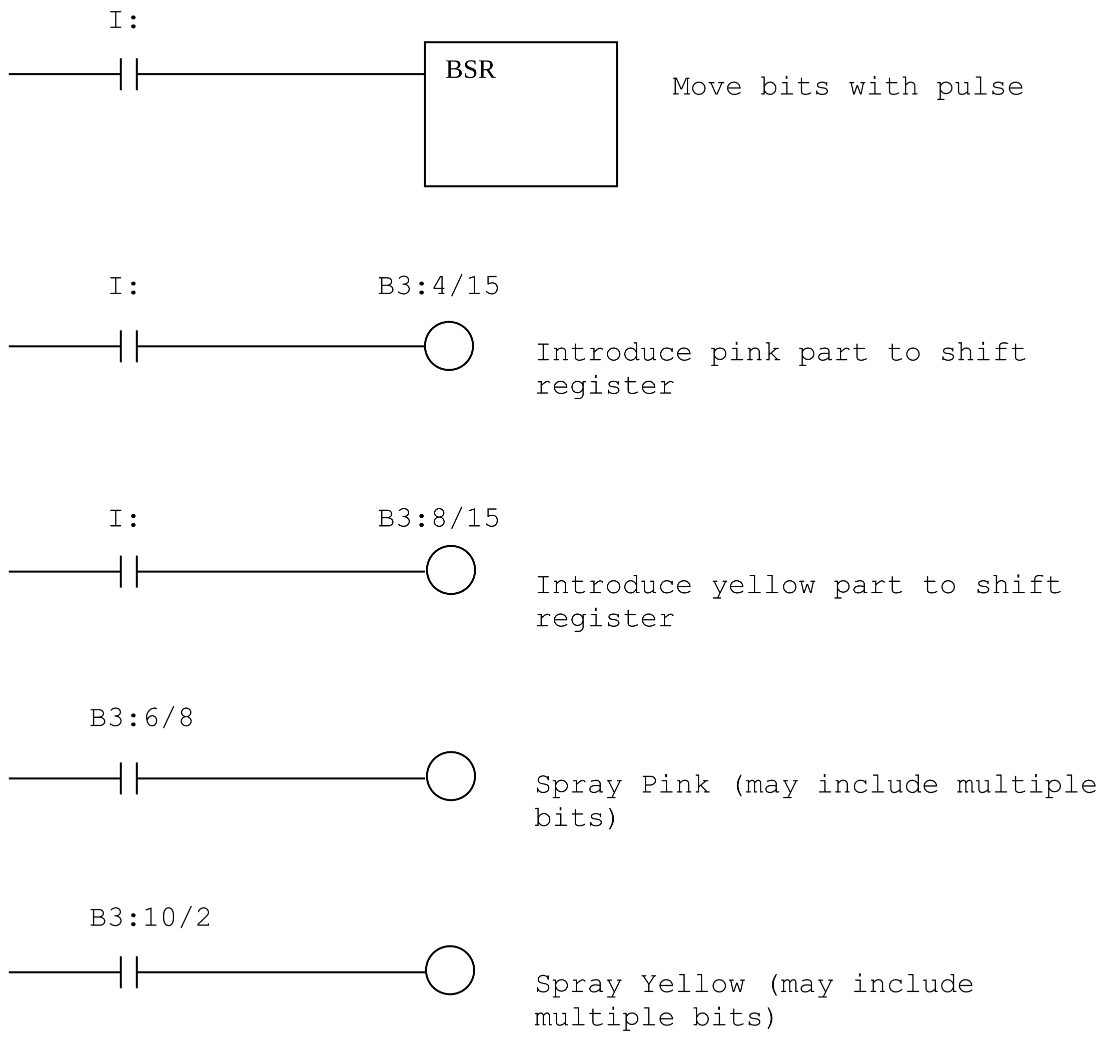
Pink Spray Shift Register

B3:4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
B3:5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B3:6	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
B3:7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Yellow Spray Shift Register

B3:8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B3:9	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
B3:10	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
B3:11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Logic for Shift Registers


I:
BSR
Move bits with pulse
I:
B3:4/15
Introduce pink part to shift register
I:
B3:8/15
Introduce yellow part to shift register
B3:6/8
Spray Pink (may include multiple bits)
B3:10/2
Spray Yellow (may include multiple bits)

Logic shown in the above rungs is used to control the output of the shift register and control down-stream operations in a tracking application.

While the shift right (or left) instruction is chosen for tracking applications, it should be considered for only the simplest applications. For more involved applications, an algorithmic approach including a FIFO table of leading edges with a counter to index through a zone should be used. The approach should include a pointer to a recipe of choices in which the recipe includes specific control bits to turn on or off an output when the parts' leading edge passes an action point. Since shift instructions take significant time to execute, the algorithm must be considered where multiple zones are involved and multiple parts are included in any specific zone. Pulse tachometers can be used that are significantly faster than the scan of the PLC with no loss of motion with the more advanced algorithm. The shift instruction depends on

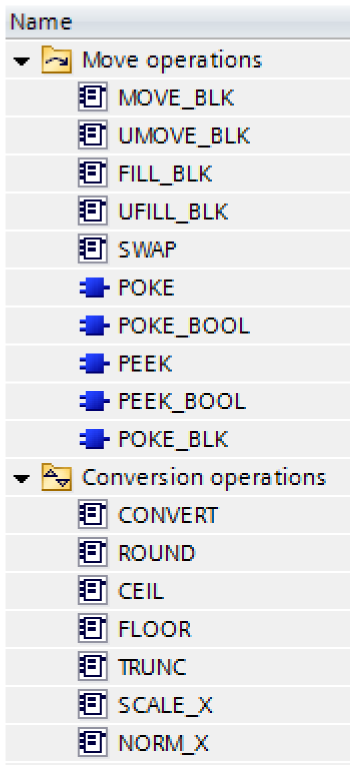
SCL

With a number of different applications being presented, it is appropriate to ask the question as to which language to use. The programming languages of A-B and Siemens commonly referred to as SCL or Sequential Logic is a procedural langage instead of object-oriented. The two languages discussed so far (LAD and FBD) are object-oriented. The language introduced below is a procedural language and looks very much like "C" or Visual Basic. The languages are useful in looping or iterative applications and may be useful as well in a number of other control schemes.

The examples below are from Siemens but the A-B equivalent language is very similar and supports many of the same procedural operations.

Siemens SCL Language

The instructions shown here are part of the SCL or procedural language from Siemens' S7-1200.


Name

- Move operations
  - MOVE_BLK
  - UMOVE_BLK
  - FILL_BLK
  - UFILL_BLK
  - SWAP
  - POKE
  - POKE_BOOL
  - PEEK
  - PEEK_BOOL
  - POKE_BLK
- Conversion operations
  - CONVERT
  - ROUND
  - CEIL
  - FLOOR
  - TRUNC
  - SCALE_X
  - NORM_X

Fig. 12-33 Siemens SCL Move and Conversion Operations

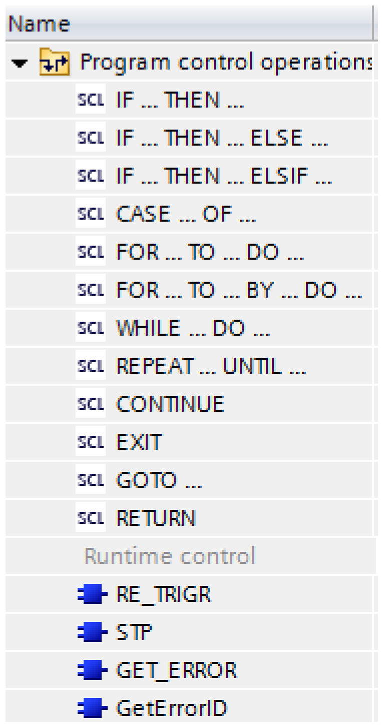

Name
Program control operations
SCL IF ... THEN ...
SCL IF ... THEN ... ELSE ...
SCL IF ... THEN ... ELSIF ...
SCL CASE ... OF ...
SCL FOR ... TO ... DO ...
SCL FOR ... TO ... BY ... DO ...
SCL WHILE ... DO ...
SCL REPEAT ... UNTIL ...
SCL CONTINUE
SCL EXIT
SCL GOTO ...
SCL RETURN
Runtime control
RE_TRIGR
STP
GET_ERROR
GetErrorID

Fig. 12-34 Siemens SCL Program Control Operations

Explanations of some of the more important SCL instructions with examples are given below. These are found in the instruction explanations in the SCL language in the TIA portal helps:

“IF: Run conditionally

Description

The instruction "Run conditionally" branches the program flow depending on a condition. The condition is an expression with Boolean value ((TRUE or FALSE). Logical expression or comparative expressions can be stated as conditions.

When the instruction is executed, the stated expressions are evaluated. If the value of an expression is TRUE, the condition is fulfilled; if the value is FALSE, it is not fulfilled.

Syntax

Depending on the type of branch, you can program the following forms of the instruction:

- Branch through IF:

```
IF <Condition> THEN <Instructions>
```

```
END_IF
```

If the condition is satisfied, the instructions programmed after the THEN are executed. If the condition is not satisfied, the execution of the program continues with the next instruction after the END_IF.

- Branch through IF and ELSE:

```
IF <Condition> THEN <Instructions1>

ELSE <Instructions0>;

END_IF
```

If the condition is satisfied, the instructions programmed after the THEN are executed. If the condition is not satisfied, the instructions programmed after the ELSE are executed. Then the execution of the program continues with the next instruction after the END_IF.

- Branch through IF, ELSIF and ELSE:

```
IF <Condition1> THEN <Instructions1>

ELSIF <Condition2> THEN <Instruction2>

ELSE <Instructions0>;

END_IF;
```

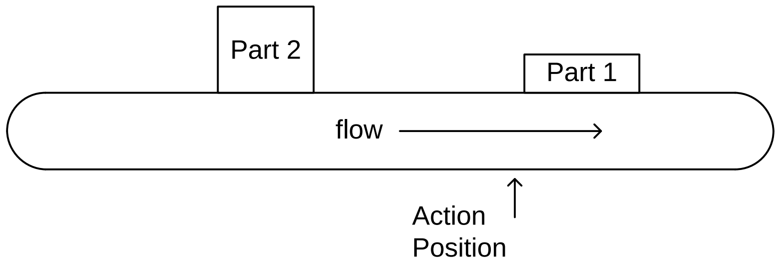
Example

The following example shows how the instruction works:

```
IF "Tag_1" = 1 THEN "Tag_Value" := 10;

ELSIF "Tag_2" = 1 THEN "Tag_Value" := 20;

ELSIF "Tag_3" = 1 THEN "Tag_Value" := 30;

ELSE "Tag_Value" := 0;

END_IF; "
```

Example of Procedural Tracking (Similar to Shift Register)

The use of a shift register to track parts on a conveyor is a very good visual way to track items. However, it may not be efficient and capable of more sophisticated tracking problems.
If there are a number of different parts being tracked and there are a number of parts in an area, then consider using a table similar to the one below and track the part and its position in a table. If the part has an action that is configured for a position in the table, the program can ask whether the part is to have the particular action take place or not. The sophistication of tracking multiple parts and keeping close track of each part lends itself to advanced tracking more sophisticated to either state diagram tracking or shift register tracking and with a small amount of computer code that can be very efficiently executed.

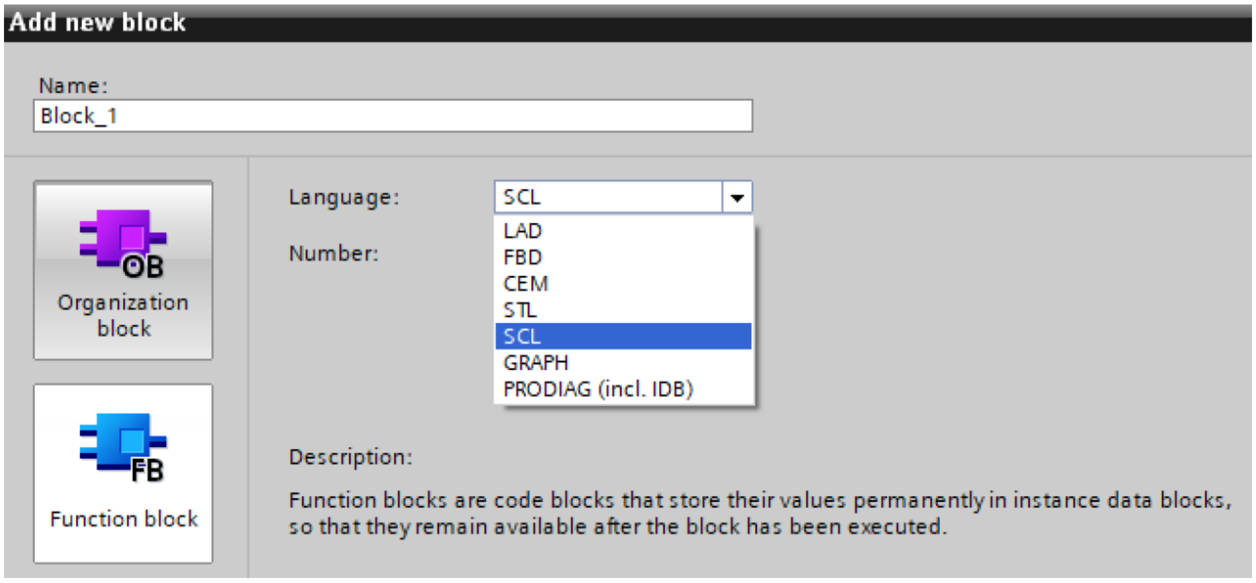


Part one's Pattern	Part one's Position
Part two's Pattern	Part two's Position
empty	empty
empty	empty
empty	empty

Fig. 12-35 Part Tracking with Procedural Language

The tracking program of the parts above is more sophisticated than the tracking using shift registers described earlier. Either may be chosen and the programmer must decide which is best in a specific case.

There are several choices and they improve with the S7-1500 processor. The following languages are available for use in the FB. SCL is a 'C' like language. With the choice SCL, the following C language is avaiable. SCL is also available in the Siemens 1500's. Also available in the 1500's is CEM, a cause-and-effect matrix language.

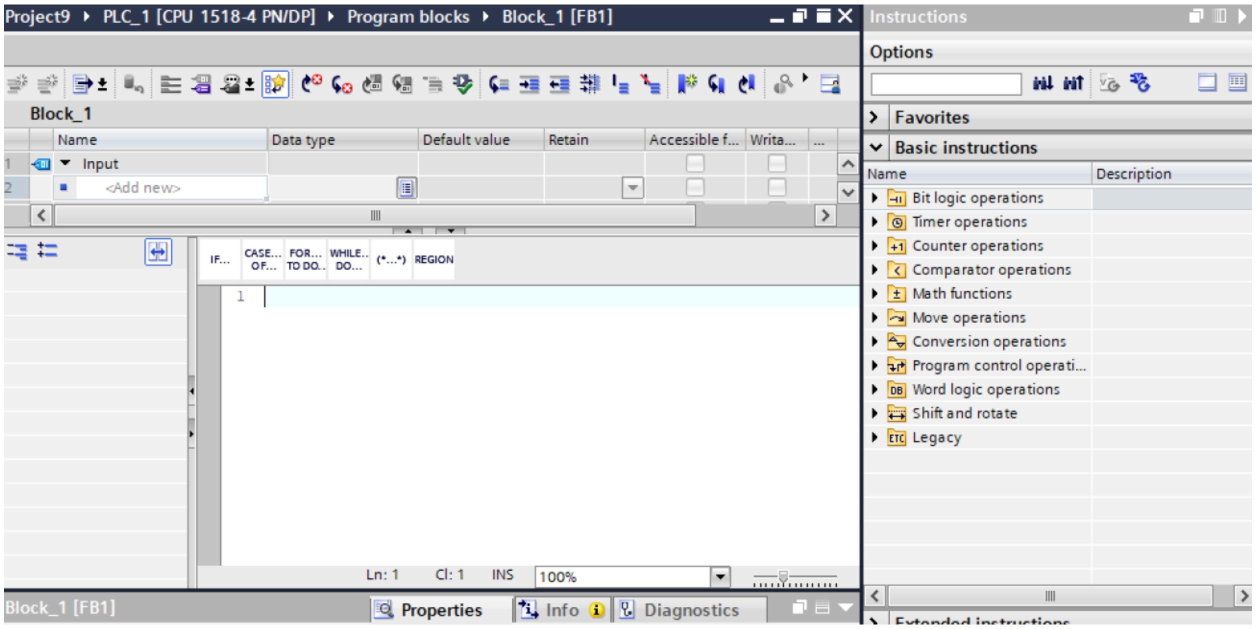

Add new block

Name: Block_1

Language: SCL

Description:
Function blocks are code blocks that store their values permanently in instance data blocks, so that they remain available after the block has been executed.

This sets the following language as the language of choice: C


Project9 ▸ PLC_1 [CPU 1518-4 PN/DP] ▸ Program blocks ▸ Block_1 [FB1]

Block_1

	Name	Data type	Default value	Retain	Accessible f...	Writa...
1	▼ Input					
2	<Add new>					

There are a number of websites and free books teaching programming in C. One of the best is from the originators of C – Kernighan and Ritchie:

C Programming by Kernighan and Ritchie – 2nd Ed. – Free PDF

https://www.engr.colostate.edu/ECE251/References/The%20C%20Programming%20Language.pdf

Considerations for Tracking

There are many considerations when determining whether to use tracking or not. If multiple conveyors are used to track parts, then the part must be tracked from conveyor to conveyor. When overlapping two conveyors, does one use the pulse from the first or the second. Also, the part may slip or not move with the pulse train causing an inaccuracy. Or the piece may break. But, with tracking, keeping track of parts is superior to other methods such as state diagrams. If a bar code or a rfid tag can be placed on the part, tracking is not needed. But with these devices, more programming must be included reading the tag. All these facts must be considered when determining the best method of controlling pieces on a line.

Summary

This chapter explains a number of data handling instructions and provides applications for their use in factory automation. Instructions are provided for a number of operations that would otherwise have required a significant programming effort to provide. Included are the queuing operations using FIFO instructions. Other interesting instructions in data handling include the bit-shift instructions. These instructions are used for shift register part tracking. Instructions used in data manipulation were shown with examples included for each type of instruction.

Examples of different tracking options are discussed and the methods of programming are examined.

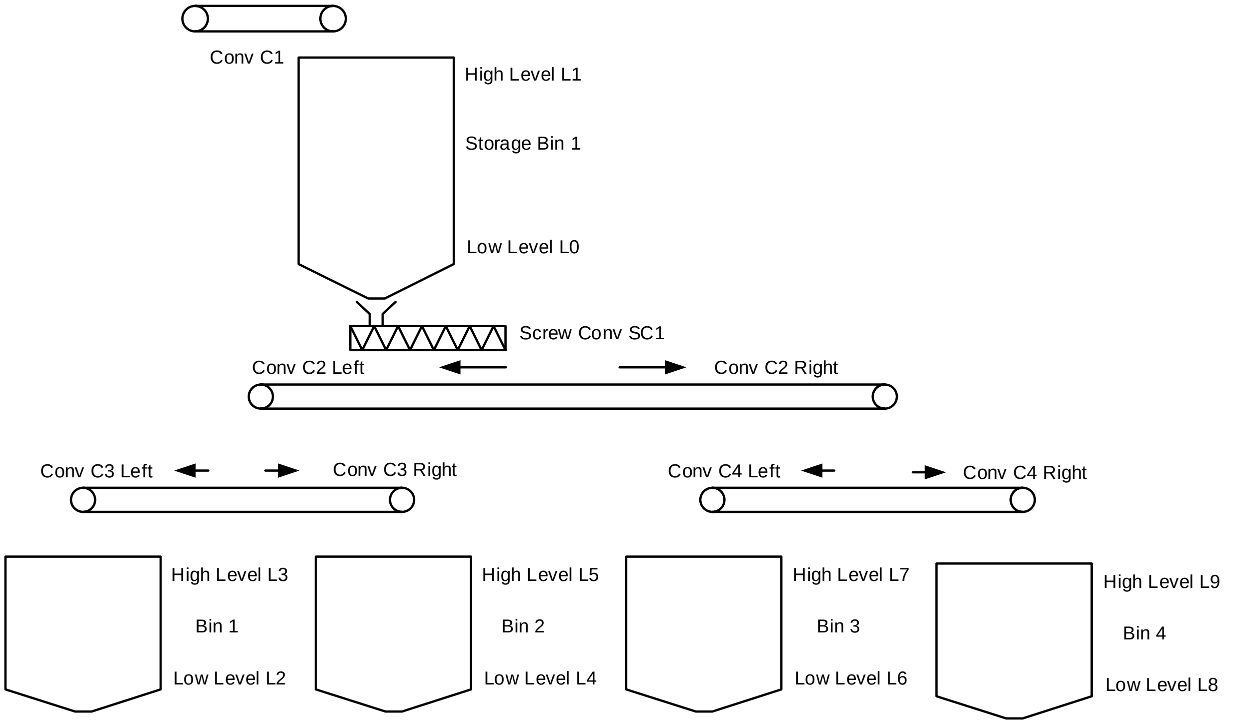
Exercises:

1. Discuss alternative methods of writing tracking programs other than the shift register. Which would you prefer? Under what conditions would you prefer the alternative? What advantages/disadvantages does the shift register approach have?

1. Why would an instruction called "SEL" or select be useful in a PLC? How else could the programmer execute this operation if the instruction were not present?

2. In Chapter 6, we first presented this problem. However, the problem would have a better outcome if a FIFO was installed to answer the question as to which tank to fill next.

   Write the logic in Ladder to satisfy the following control problem:



The process depends on a level switch in the four bins at the bottom (Bin 1-4). For any bin to fill, it must be at a low level. Then the conveyor C2 will turn on as well as either C3 or C4 to fill the appropriate bin. The direction of C2, C3 and C4 must be correct as well (forward or reverse). Also, Storage Bin 1 has a high and low level switch and will be filled from above by conveyor C1 as needed.

Fill in Definition of Inputs:

Sensor	Function/State	Signal Assignment

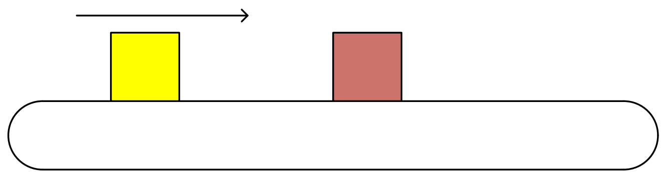
Fill in Definition of Outputs:

Actuator	Function/State	Signal Assignment

Next, write the ladder logic to fill any bin that falls below the low level in the order that a request is made.

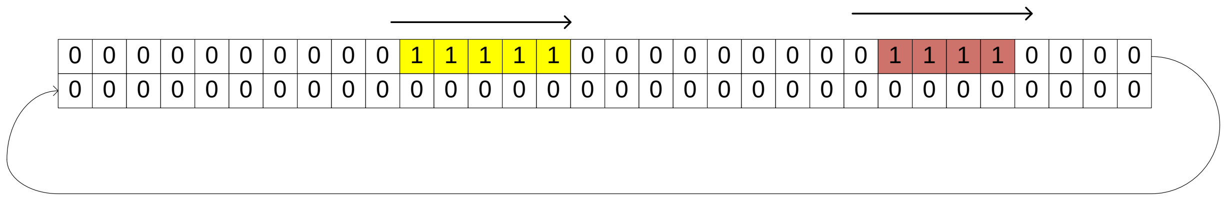
Lab 12.1

Set up the Shift Register for tracking two different parts similar to the example explained in the chapter. Use inputs for introducing the two different boxes into two different shift registers and the prox switch. You can also use a timer to introduce pulses for a simulated prox switch. Make a variable distance that can be inserted at a location to vary the point of extraction. Turn on an output to signal an extractor solenoid turning on.

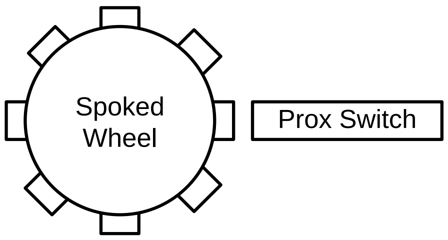


Tracking Boxes Down Conveyor

Use BSR to track boxes down conveyor

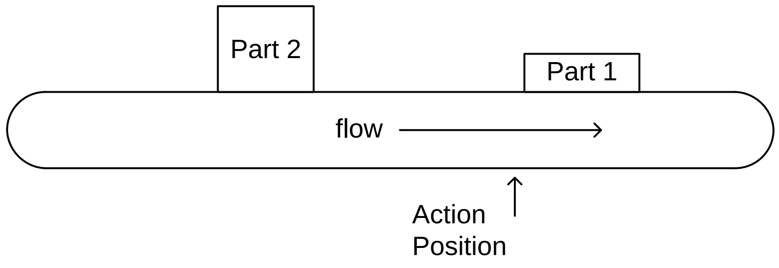


Use BSR to track boxes down shift register



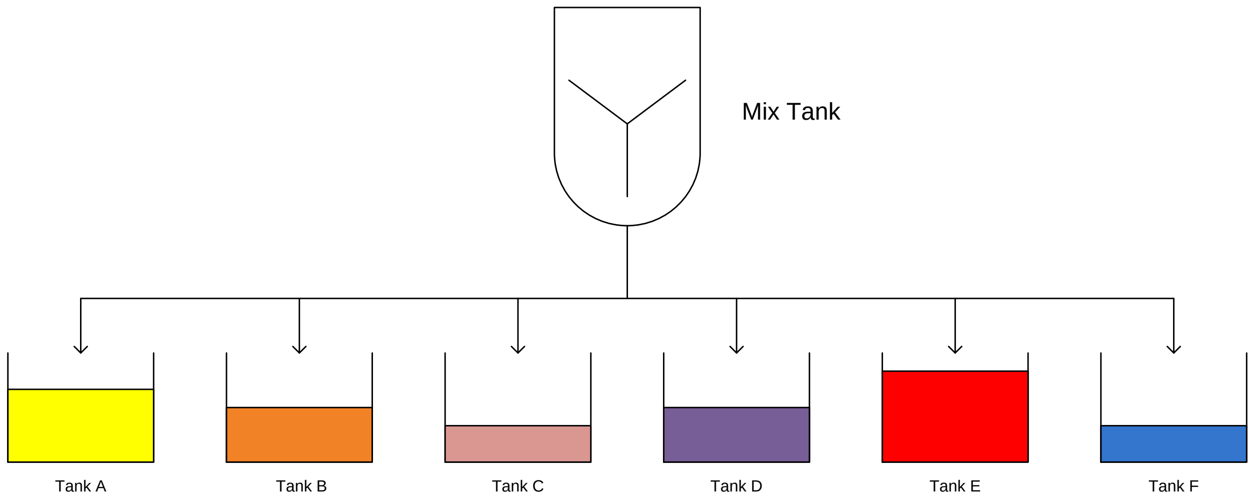
Lab 12.2

Set up the shift register for tracking multiple parts on a conveyor using SCL similar to the application described in Fig. 12-35.



Lab 12.3

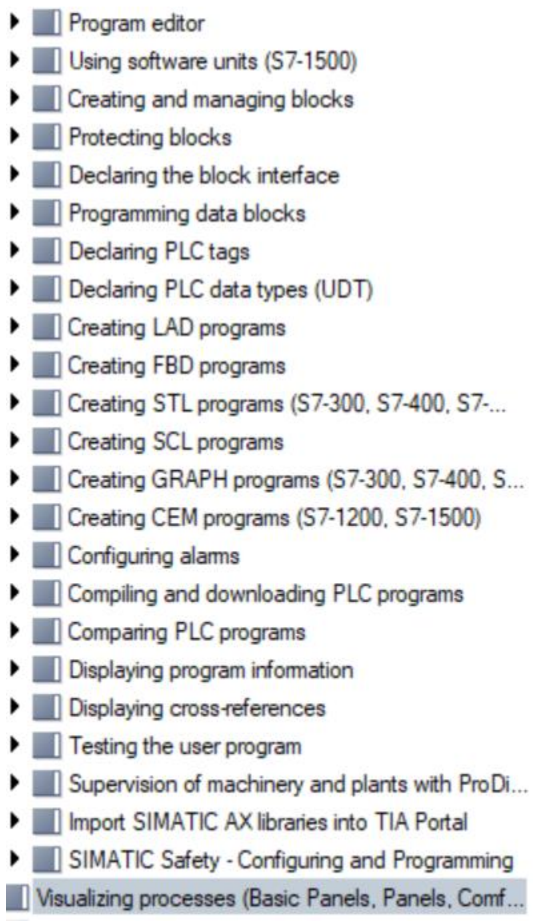
Set up the queuing operation similar to the mix tank example in the chapter. Demonstrate a tank level switch going low and the subsequent operation of the mix tank to make a mix, drain it to the appropriate tank and continue with the next queued operation.



Appendix A

Languages:

Siemens

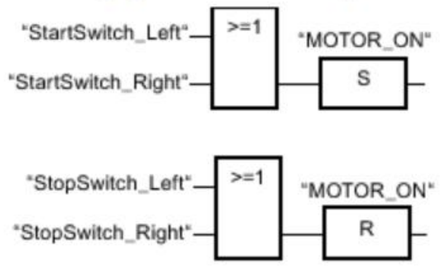

Program editor
Using software units (S7-1500)
Creating and managing blocks
Protecting blocks
Declaring the block interface
Programming data blocks
Declaring PLC tags
Declaring PLC data types (UDT)
Creating LAD programs
Creating FBD programs
Creating STL programs (S7-300, S7-400, S7-...
Creating SCL programs
Creating GRAPH programs (S7-300, S7-400, S...
Creating CEM programs (S7-1200, S7-1500)
Configuring alarms
Compiling and downloading PLC programs
Comparing PLC programs
Displaying program information
Displaying cross-references
Testing the user program
Supervision of machinery and plants with ProDi...
Import SIMATIC AX libraries into TIA Portal
SIMATIC Safety - Configuring and Programming
Visualizing processes (Basic Panels, Panels, Comf...

Basic information on logic paths in FBD

Use of logic paths

The user program will be mapped in one or more networks. The networks can contain one or more logic paths on which the binary signals are arranged in the form of boxes.

The following figure shows an example of the use of several logic paths within a network:



Rules

Remember the following rules when using logic paths:

- Connections are not permitted between logic paths.
- Only one jump instruction is permissible per network. The positioning rules for jump instructions remain valid.

Executing logic paths

Logic paths are executed from top to bottom and from left to right. This means that the first instruction in the first logic path of the first network is executed first. All instructions of this logic path are then executed. After this come all other logic paths of the first network. The next network is executed only after all logic paths have first been executed.

When jumps are used the regular execution of the logic paths is circumvented and the instruction is executed at the jump destination.

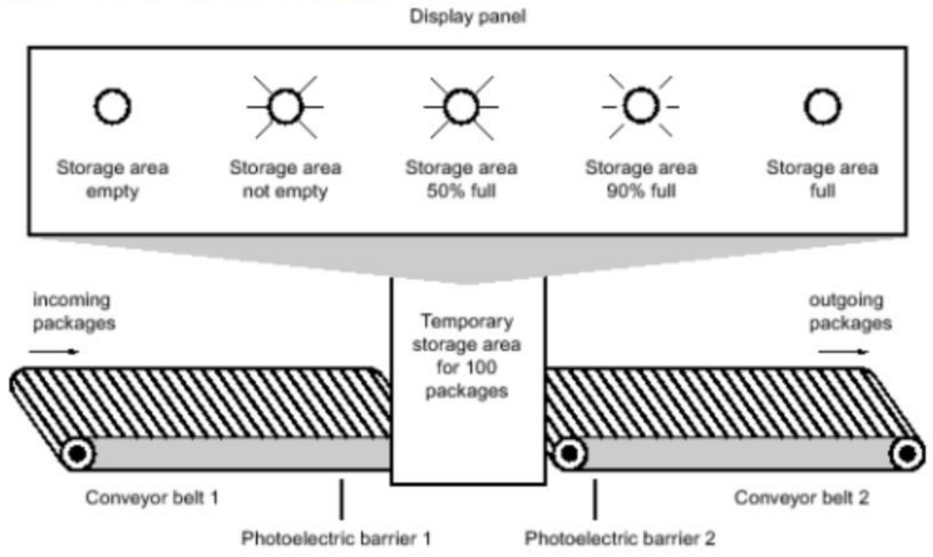
Differences between branches and logic paths

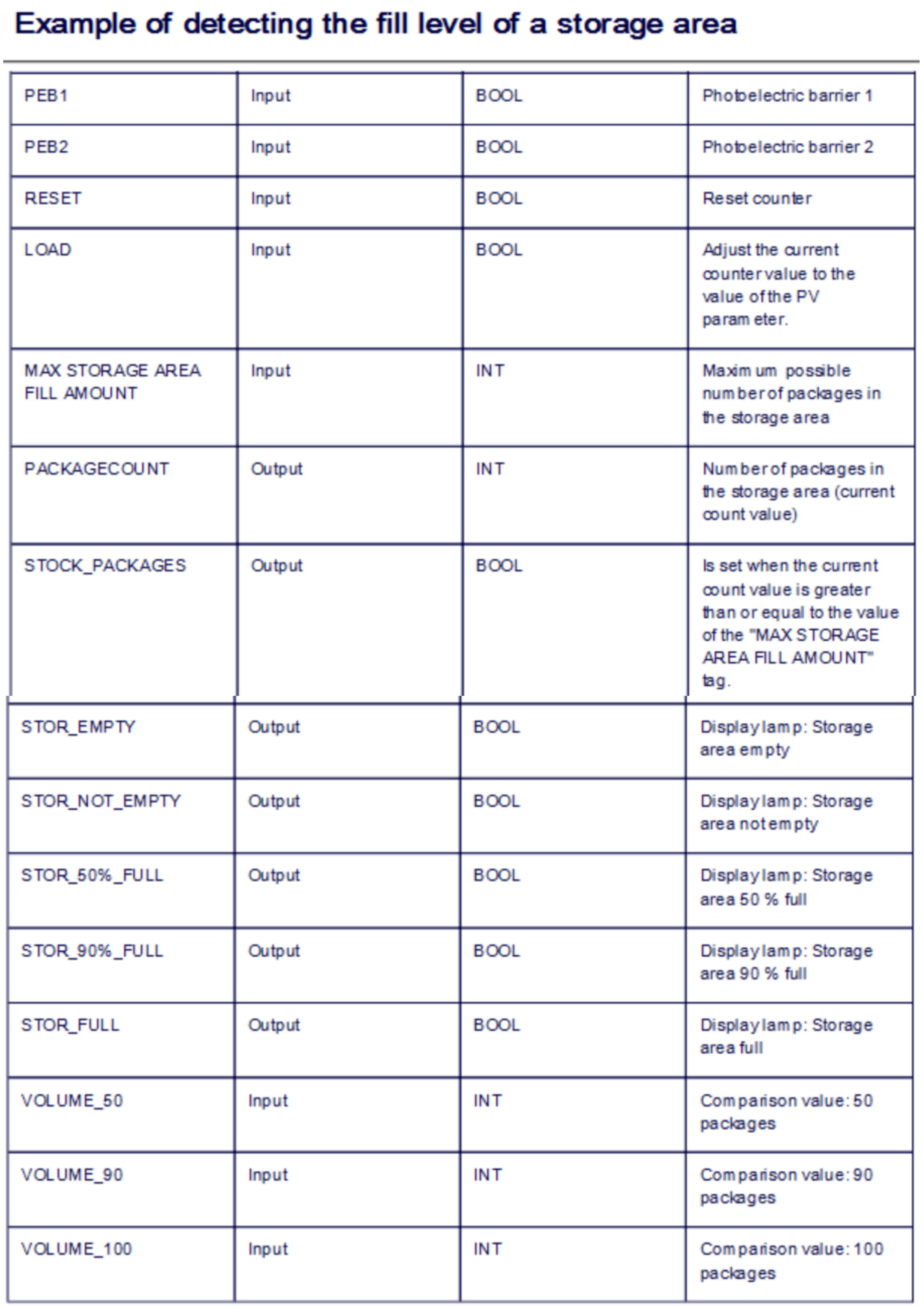
The difference between branches and logic paths is that the logic paths are independent branches that can also stand in a different network. Branches, on the other hand, permit the programming of a parallel connection and have a common preceding logic operation.

Example of detecting the fill level of a storage area

Detecting the fill level of a storage area

The following figure shows a system with two conveyor belts and a temporary storage area between them. Conveyor belt 1 delivers packages to the storage area. A photoelectric barrier at the end of conveyor belt 1 near the storage area detects how many packages are delivered to the storage area. Conveyor belt 2 transports packages from the temporary storage area to a loading dock where they are loaded onto trucks. A photoelectric barrier at the storage area exit detects how many packages leave the storage area to be transported to the loading dock. Five display lamps indicate the capacity of the temporary storage area.





Example of detecting the fill level of a storage area

PEB1	Input	BOOL	Photoelectric barrier 1
PEB2	Input	BOOL	Photoelectric barrier 2
RESET	Input	BOOL	Reset counter
LOAD	Input	BOOL	Adjust the current counter value to the value of the PV parameter.
MAX STORAGE AREA FILL AMOUNT	Input	INT	Maximum possible number of packages in the storage area
PACKAGECOUNT	Output	INT	Number of packages in the storage area (current count value)
STOCK_PACKAGES	Output	BOOL	Is set when the current count value is greater than or equal to the value of the "MAX STORAGE AREA FILL AMOUNT" tag.
STOR_EMPTY	Output	BOOL	Display lamp: Storage area empty
STOR_NOT_EMPTY	Output	BOOL	Display lamp: Storage area not empty
STOR_50%_FULL	Output	BOOL	Display lamp: Storage area 50 % full
STOR_90%_FULL	Output	BOOL	Display lamp: Storage area 90 % full
STOR_FULL	Output	BOOL	Display lamp: Storage area full
VOLUME_50	Input	INT	Comparison value: 50 packages
VOLUME_90	Input	INT	Comparison value: 90 packages
VOLUME_100	Input	INT	Comparison value: 100 packages

The following networks show the LAD program ming for activating the lam ps:

Network 1:

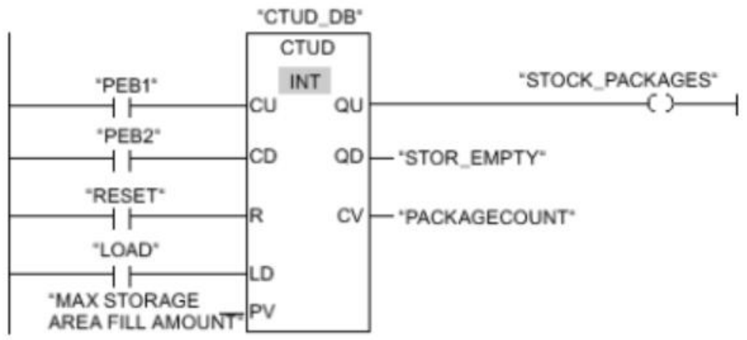
When a package is delivered to the storage area, the signal state at "PEB1" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB1", the "Up" counter is enabled, and the current count value of "PACKAGECOUNT" is increased by one.

When a package is delivered from the storage area to the loading dock, the signal state at "PEB2" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB2", the "Down" counter is enabled, and the current count value of "PACKAGECOUNT" is decreased by one.

If there are no packages in the storage area ("PACKAGECOUNT" = "0"), the "STOR_EMPTY" tag is set to signal state "1", and the "Storage area empty" lamp is switched on.

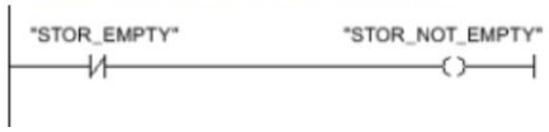
The current count value can be reset to "0" if the "RESET" tag is set to signal state "1".

If the "LOAD" tag is set to signal state "1", the current count value is set to the value of the "MAX STORAGE AREA FILL AMOUNT" tag. As long as the current count value is greater than or equal to the value of the "MAX STORAGE AREA FILL AMOUNT" tag, the "STOCK_PACKAGES" tag supplies the signal state "1".



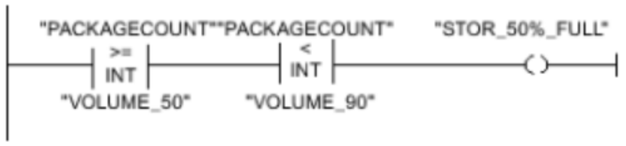
Network 2:

As long as there are packages in the storage area, the "STOR_NOT_EMPTY" tag is set to signal state "1", and the "Storage area not em pty" lam p is switched on.



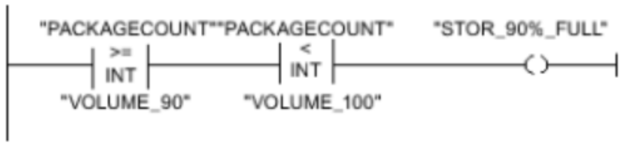
Network 3:

If the num ber of packages in the storage area is greater than or equal to 50, the "Storage area 50 % full" lamp switches on.



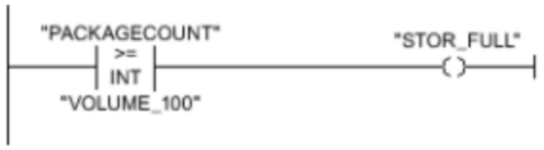
Network 4:

If the num ber of packages in the storage area is greater than or equal to 90, the "Storage area 90% full" lamp switches on.



Network 5:

If the number of packages in the storage area reaches 100, the lamp for the "Storage area full" message switches on.



The following networks show the FBD programming for activating the lamps:

Network 1:

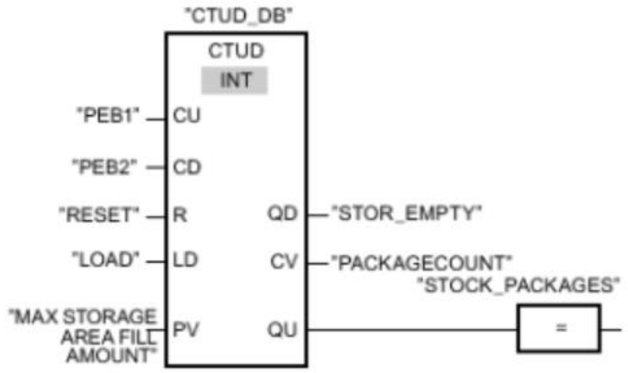
When a package is delivered to the storage area, the signal state at "PEB1" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB1", the "Up" counter is enabled, and the current count value of "PACKAGECOUNT" is increased by one.

When a package is delivered from the storage area to the loading dock, the signal state at "PEB2" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB2", the "Down" counter is enabled, and the current count value of "PACKAGECOUNT" is decreased by one.

If there are no packages in the storage area ("PACKAGECOUNT" = "0"), the "STOR_EMPTY" tag is set to signal state "1", and the "Storage area empty" lamp is switched on.

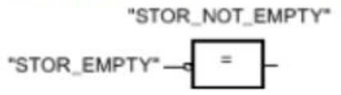
The current count value can be reset to "0" if the "RESET" tag is set to signal state "1".

If the "LOAD" tag is set to signal state "1", the current count value is set to the value of the "MAX STORAGE AREA FILL AMOUNT" tag. As long as the current count value is greater than or equal to the value of the "MAX STORAGE AREA FILL AMOUNT" tag, the "STOCK_PACKAGES" tag supplies the signal state "1".



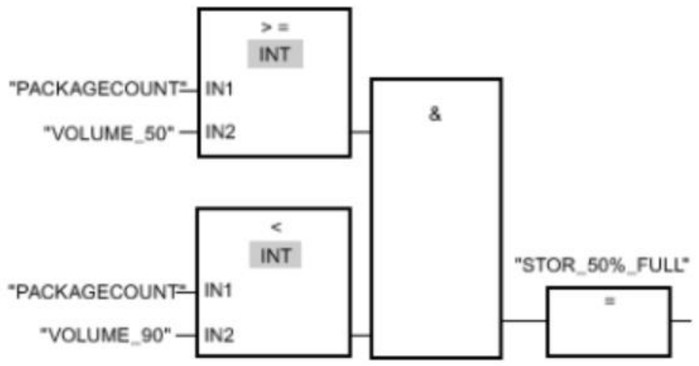
Network 2:

As long as there are packages in the storage area, the "STOR_NOT_EMPTY" tag is set to signal state "1", and the "Storage area not empty" lamp is switched on.



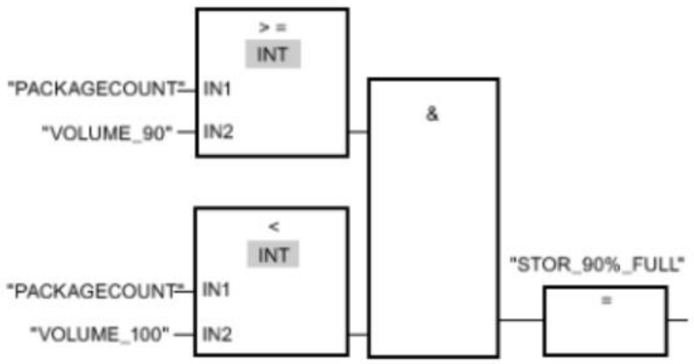
Network 3:

If the number of packages in the storage area is greater than or equal to 50, the "Storage area 50 % full" lamp switches on.



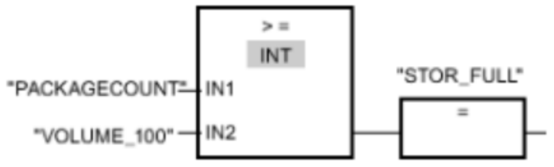
Network 4:

If the number of packages in the storage area is greater than or equal to 90, the "Storage area 90% full" lamp switches on.



Network 5:

If the number of packages in the storage area reaches 100, the lamp for the "Storage area full" message switches on



STL	Explanation
`A #LS1`	`// Scan photoelectric barrier "LS1" for "1".`
`CU "PACKAGECOUNT"`	`// In the event of a positive signal edge at photoelectric barrier "LS1", the count value of counter "PACKAGECOUNT" is increased by one.`
`A #LS2`	`// Scan photoelectric barrier "LS2" for "1".`
`CD "PACKAGECOUNT"`	`// In the event of a positive signal edge at photoelectric barrier "LS2", the count value of counter "PACKAGECOUNT" is decreased by one.`
`AN "PACKAGECOUNT"`	`// Scan count value for "0".`
`= #STOR_EMPTY`	`// With a count value of "0" the display lamp "storage area empty" is switched on.`
`A "PACKAGECOUNT"`	`// Scan count value for "1".`
`= #STOR_NOT_EMPTY`	`// With a count value greater than "0" the display lamp "Storage area not empty" is switched on.`
`L 50`	`// Load the comparison value "50" to accumulator 1.`
`L "PACKAGECOUNT"`	`// Move the comparison value to accumulator 2.` `// Load the current count value to accumulator 1.`
`<=I`	`// Compare values`
`= #"STOR_50%_FULL"`	`// With a count value greater than or equal to "50" the display lamp "Storage area 50% full" is switched on.`
`L 90`	`// Move the counter value to accumulator 2.` `// Load the comparison value "90" to accumulator 1.`
`>=I`	`// Compare values`
`= #"STOR_90%_FULL"`	`// With a count value greater than or equal to "90" the display lamp "Storage area 90% full" is switched on.`
`L "PACKAGECOUNT"`	`// Load the current count value to accumulator 1.`
`L 100`	`// Move the counter value to accumulator 2.` `// Load the comparison value "100" to accumulator 1.`
`>=I`	`// Compare values`
`= #STOR_FULL`	`// At a count value greater than "100" the display lamp "Storage area full" is switched on.`

The following SCL program shows how to implement this example:

When a package is delivered to the storage area, the signal state at "PEB1" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB1", the "Up" counter is enabled, and the current count value of "PACKAGECOUNT" is increased by one.

When a package is delivered from the storage area to the loading dock, the signal state at "PEB2" switches from "0" to "1" (positive signal edge). On a positive signal edge at "PEB2", the "Down" counter is enabled, and the current count value of "PACKAGECOUNT" is decreased by one.

If there are no packages in the storage area ("PACKAGECOUNT" = "0"), the "STOR_EMPTY" tag is set to signal state "1", and the "Storage area empty" lamp is switched on.

The current count value can be reset to "0" if the "RESET" tag is set to signal state "1".

If the "LOAD" tag is set to signal state "1", the current count value is set to the value of the "MAX STORAGE AREA FILL AMOUNT" tag. As long as the current count value is greater than or equal to the value of the "MAX STORAGE AREA FILL AMOUNT" tag, the "STOCK_PACKAGES" tag supplies the signal state "1".

SCL

```
"VOLUME_50" := 5; // Preassigning of the comparison value to 50 packages (for the test
only 5 packages)

"VOLUME_90" := 9; // Preassigning of the comparison value to 90 packages (for the test
only 9 packages)

"VOLUME_100" := 10; // Preassigning of the comparison value to 100 packages (for the test
only 10 packages)

"MAX STORAGE AREA FILL AMOUNT" := 10; // Preassigning of the maximum amount in storage
area to 100 packages (for the test only 10 packages)
```

```
"IEC_Counter_0_DB".CTUD(CU := "PEB1",

CD := "PEB2",

R := "RESET",

LD := "LOAD",

PV := "MAX STORAGE AREA FILL AMOUNT",

QU => "STOCK_PACKAGES",

QD => "STOR_EMPTY",

CV => "PACKAGECOUNT");
```

As long as the storage area contains packages, the "Storage area not em pty" lam p is switched on.

SCL

```
"STOR_NOT_EMPTY" := NOT "STOR_EMPTY"
```

If the num ber of packages in the storage area is lower than 50%, the lam ps for the alarm s "Storage area 50% full", "Storage area 90% full" and "Storage area full" switch off.

SCL

```
IF "PACKAGECOUNT" < "VOLUME_50" THEN

"STOR_50%_FULL" := 0;

"STOR_90%_FULL" := 0;

"STOR_FULL" := 0;

END_IF;
```

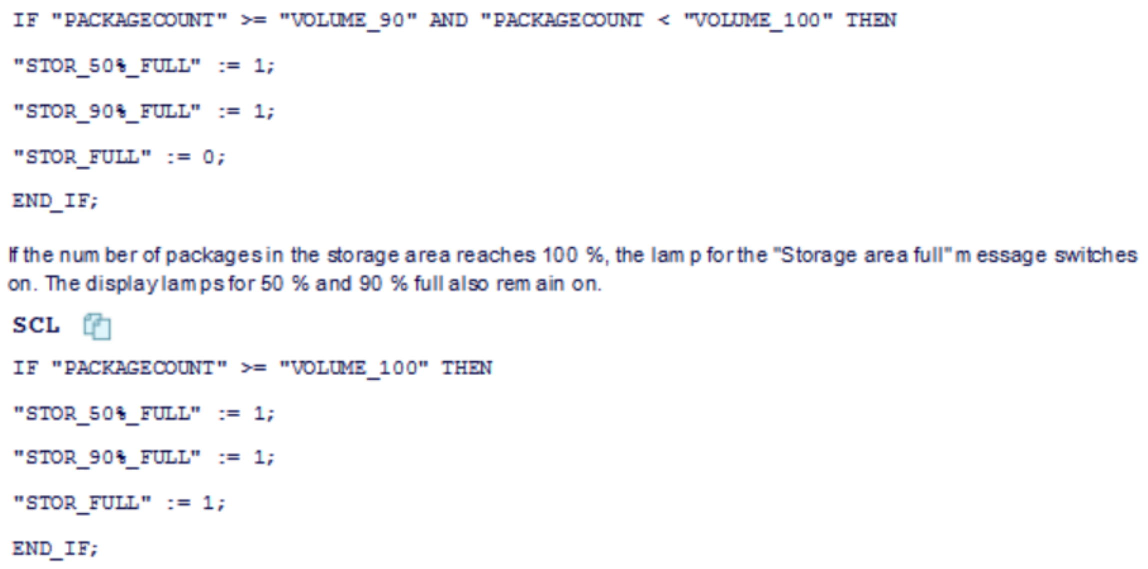
If the num ber of packages in the storage area is greater than or equal to 50 %, the "Storage area 50 % full" lam p switches on.

SCL

```
IF "PACKAGECOUNT" >= "VOLUME_50" AND "PACKAGECOUNT <= "VOLUME_90" THEN

"STOR_50%_FULL" := 1;

"STOR_90%_FULL" := 0;

"STOR_FULL" := 0;

END_IF;
```

If the num ber of packages in the storage area is greater than or equal to 90 %, the "Storage area 90 % full" lam p switches on. The display lam p for 50 % full also remains on.

SCL



```
IF "PACKAGECOUNT" >= "VOLUME_90" AND "PACKAGECOUNT < "VOLUME_100" THEN
"STOR_50%_FULL" := 1;
"STOR_90%_FULL" := 1;
"STOR_FULL" := 0;
END_IF;
```

If the num ber of packages in the storage area reaches 100 %, the lam p for the "Storage area full" m essage switches on. The display lam ps for 50 % and 90 % full also rem ain on.

SCL

```
IF "PACKAGECOUNT" >= "VOLUME_100" THEN
"STOR_50%_FULL" := 1;
"STOR_90%_FULL" := 1;
"STOR_FULL" := 1;
END_IF;
```

CEM
https://www.youtube.com/watch?v=FaR9rVXUkJo