

PLC Shift and Rotate Instructions

Shift designated array elements.

The shift designated array elements (SHIFT_A) shifts the sub array elements starting at the start shifting location(Start) and terminating at the end shifting location(End). Each time the instruction enable bit (EN) experiences a positive transition, the sub array elements are shifted N times. The sub array first N elements (from the Start position) are filled by the data stored in the IN input variable. The data leaving the end position (over flowing data) is shifted to the output variable OUT. The direction of shifting is determined by the comparison state of Start and End. For Start>End, the shift direction is from the upper positions to the lower ones and vice versa. The source array (SRC) data type can be any type except the string type. Figure 1 shows the ladder symbol of the instruction under investigation and also the shifting movement for one positive transition of IN .

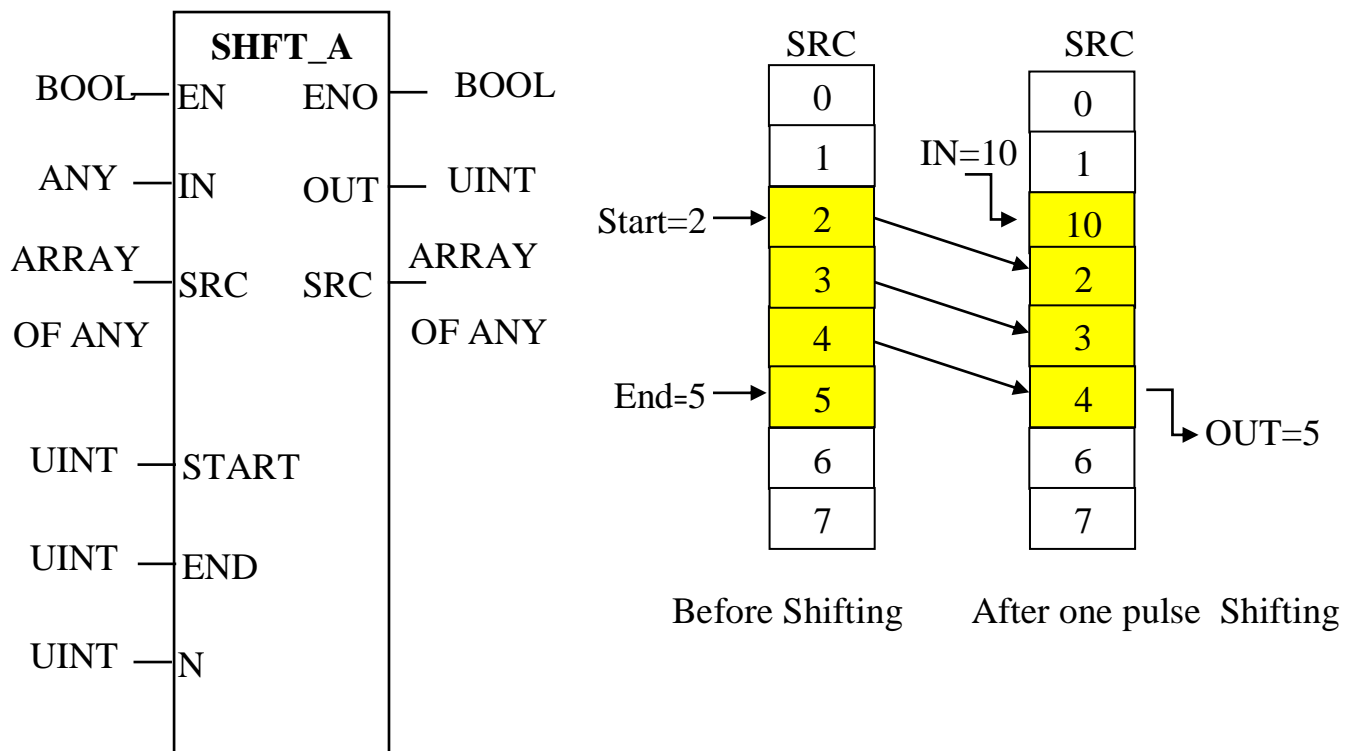


Figure 1: SHFT_A ladder symbol and behavior diagram

Example 1:

Using SHIFT_A instruction, write ladder program to generate the following repeated sequence [0,1,4,8,16,32,64,128].

Solution : The solution is as drawn in figure 2 with an initial value of A equals to {1,4,8,16,32,64,128}.

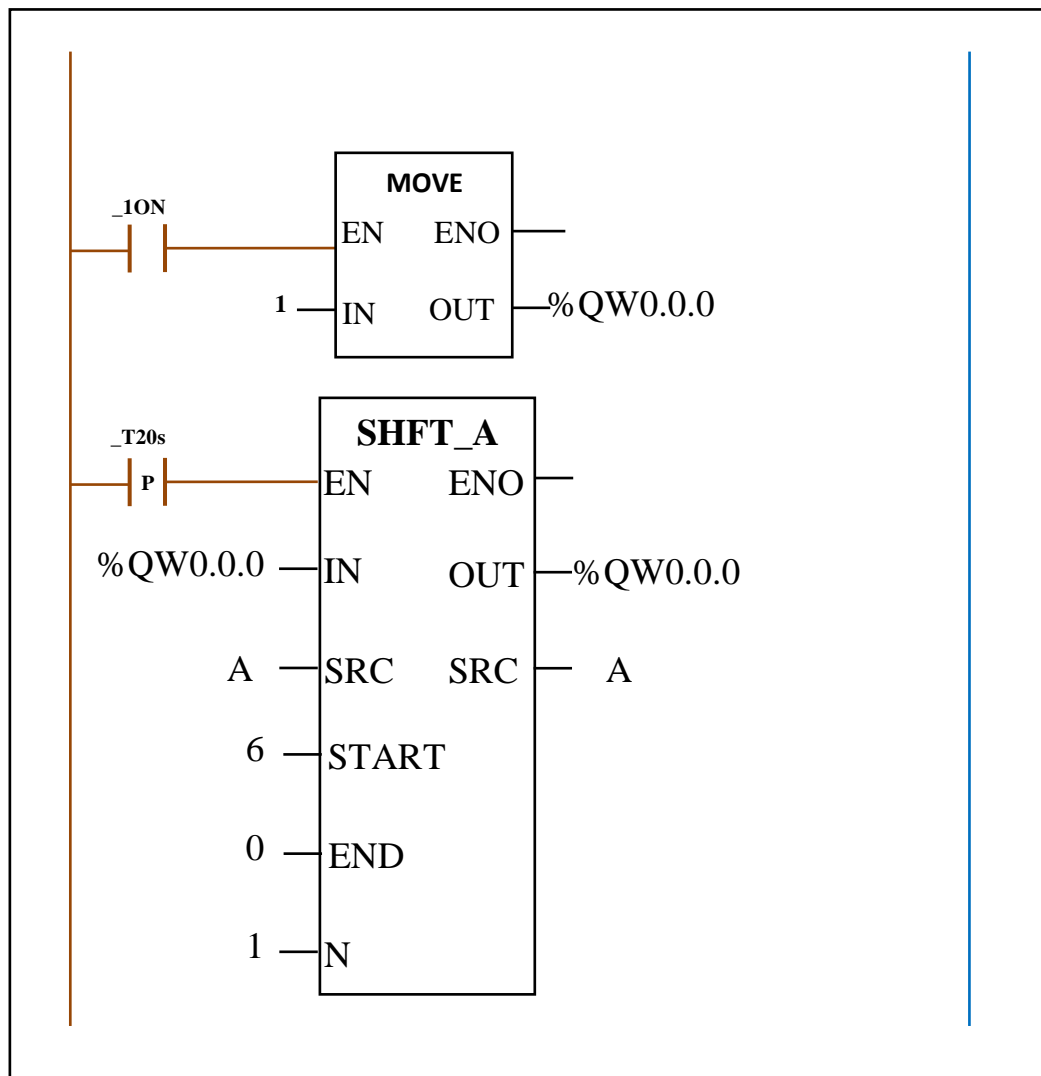


Figure 2: Example 1 ladder diagram

Shift With Carry.

The shift with carry instruction (SHIFT_C) behaves the same way the SHIFT_A instruction does. It can be considered as SHIFT_A with Boolean data type but SRC is not an array data type. Here SCR may be Byte, Word, Dword, or Lword. So to treat SHIFT_C as SHIFT_A, the byte type data should be treated as if it were an array of 8 Boolean type elements, word type data as an array of 16 Boolean type elements, dword as an array of 32 Boolean type elements, and lword as an array of 64 Boolean type elements. Figure 3 shows the ladder symbol of this instruction and also the shifting movement for one positive transition of IN.

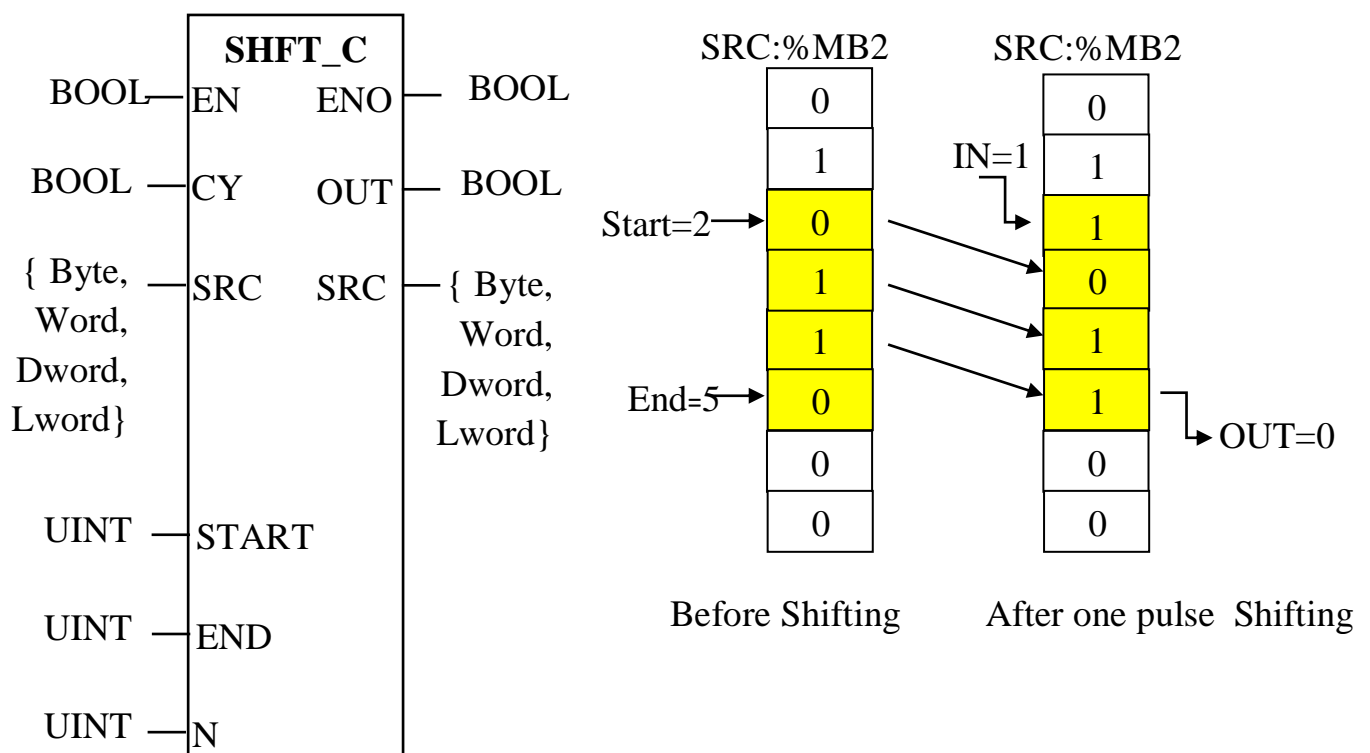


Figure 3: SHIFT_C ladder symbol and behavior diagram

Example 2:

Using SHIFT_C instruction, write ladder program to generate the following repeated sequence [0,1,2,4,8,16,32,64,128].

Solution : The solution is as drawn in figure 9.3 with an initial value of %mw2 equals 1.

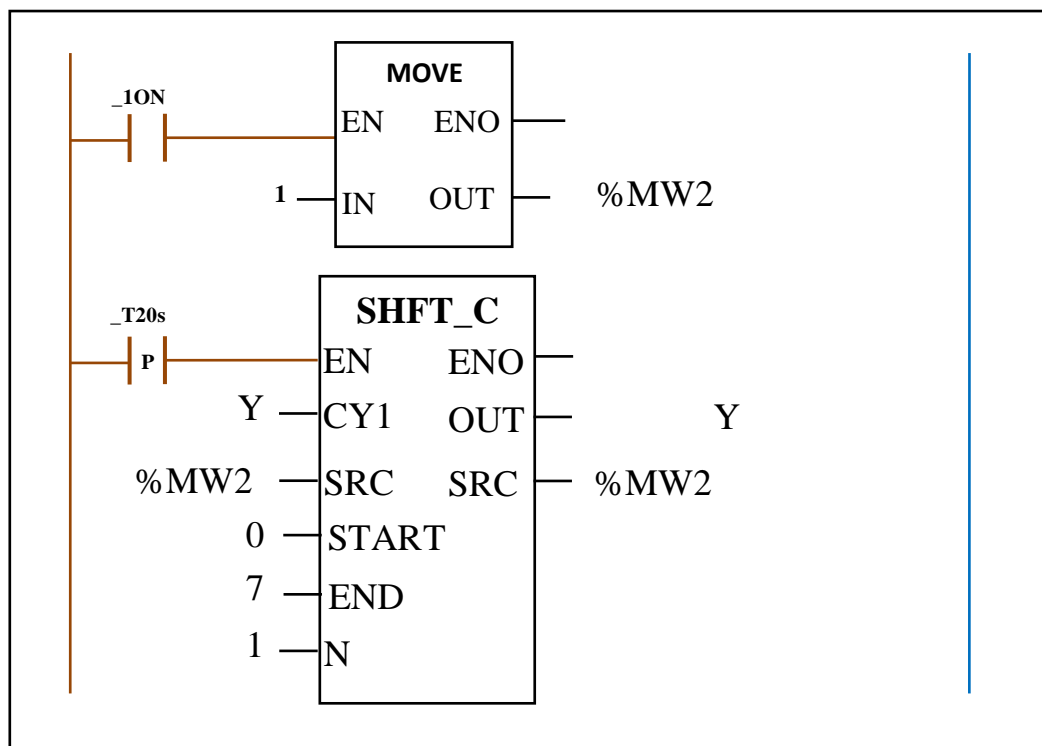


Figure 4: Example 2 ladder diagram

Shift Right.

The shift right instruction (SHR) shifts the image of the bit string (Byte, Word, Dword, Lword) assigned to IN (instruction data input) as N (instruction shifting times) bits number and stores the shifted result in the instruction output variable OUT. As compared to SHIFT_C, this one has one shift direction (from left to right), the shifting process is executed over the whole string image, the overflow data is lost, and there is no specific inlet to substitute the left end bit or bits. Figure 5 highlights the ladder symbol used and illustrates its operation for different N values.

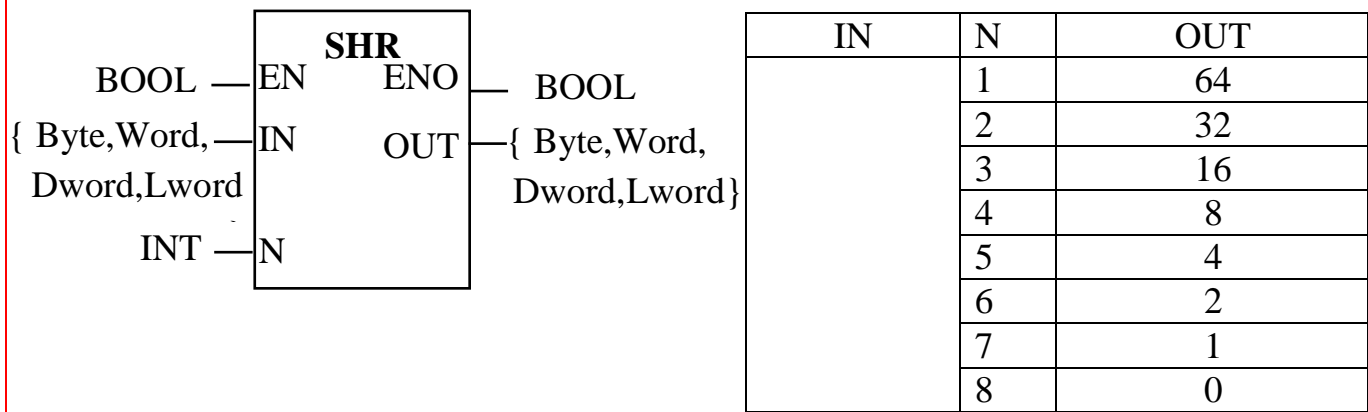


Figure 5: SHR ladder symbol and shifting Process as function of N

Example 3:

Draw ladder diagram to copy the upper byte of %MW4 to the lower byte of %QW0.0.0. without affecting the content of the upper part of %QW0.0.0.

Solution:

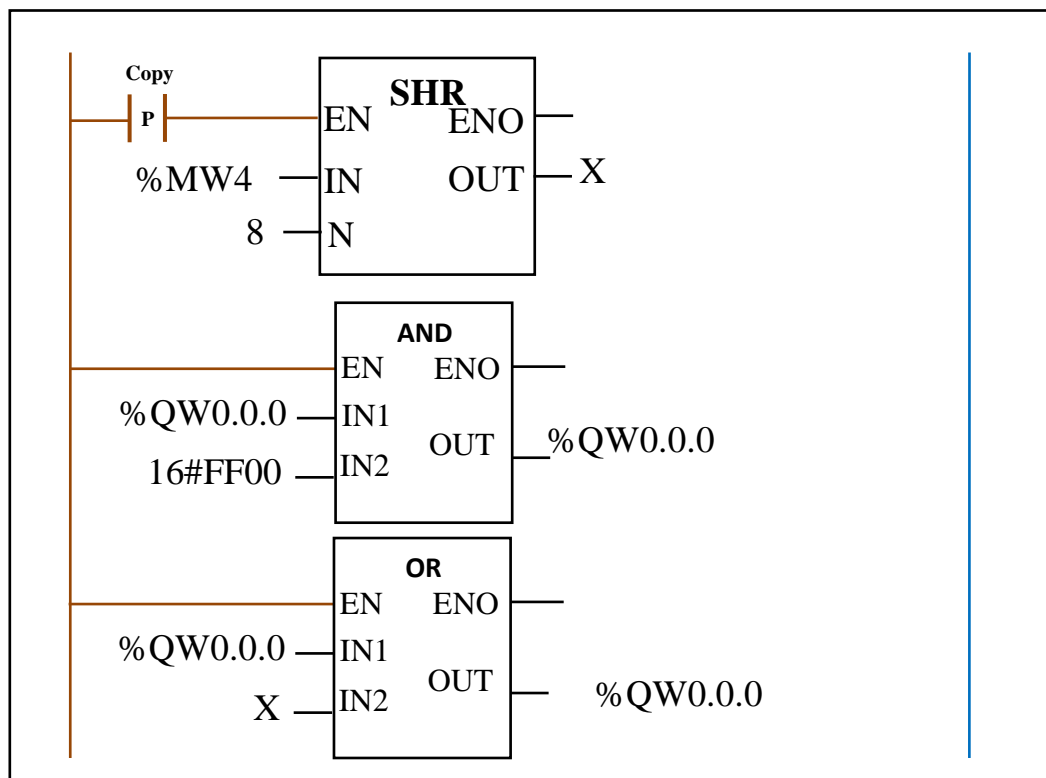


Figure 6: Example 3 ladder diagram

Shift Left.

The shift left instruction (SHL) does the reverse done by its predecessor SHR. SHR does the integer division by two, but SHL does the multiplication by 2. It shifts the image of the bit string IN as N times from the least significant bit towards the most significant one. Figure 7 displays the ladder symbol used and illustrates the instruction operation for different N values.

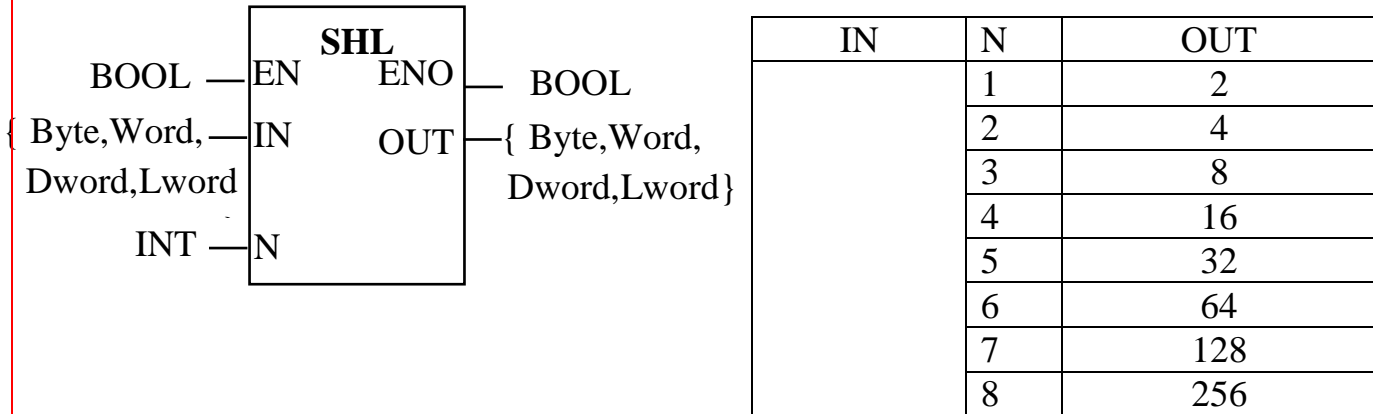


Figure 7: SHL ladder symbol and shifting Process as function of N

Example 4:

Draw ladder diagram to copy the lower byte of %MW4 to the upper byte of %QW0.0.0. without affecting the content of the lower part of %QW0.0.0..

Solution:

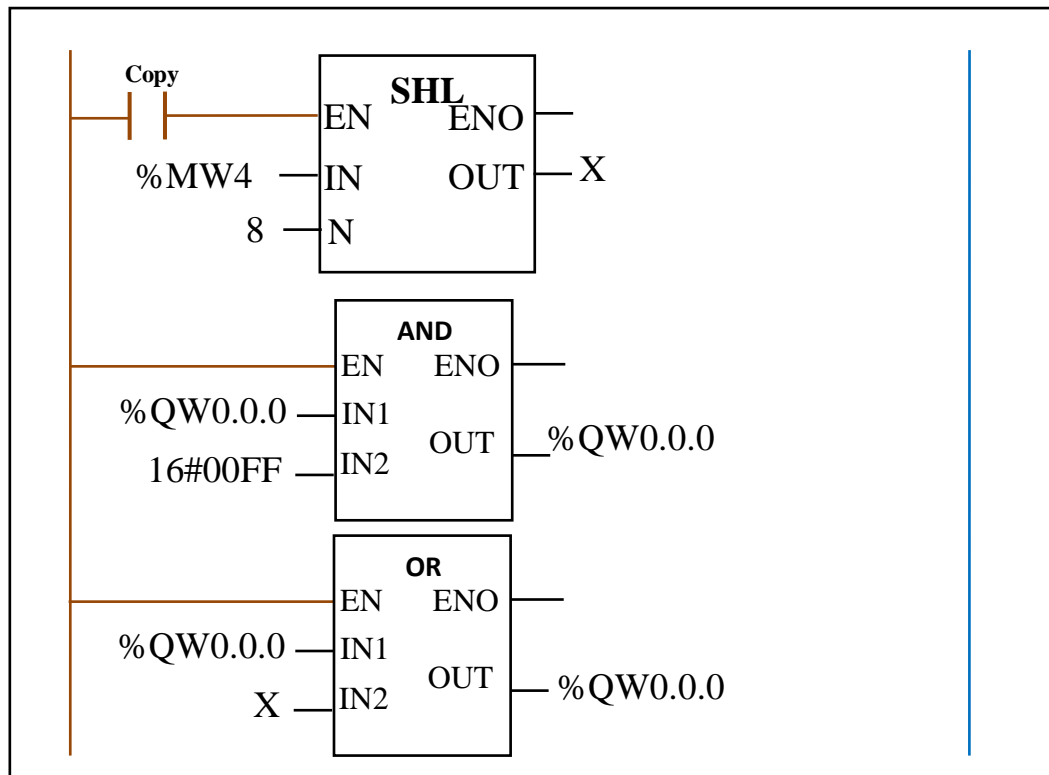
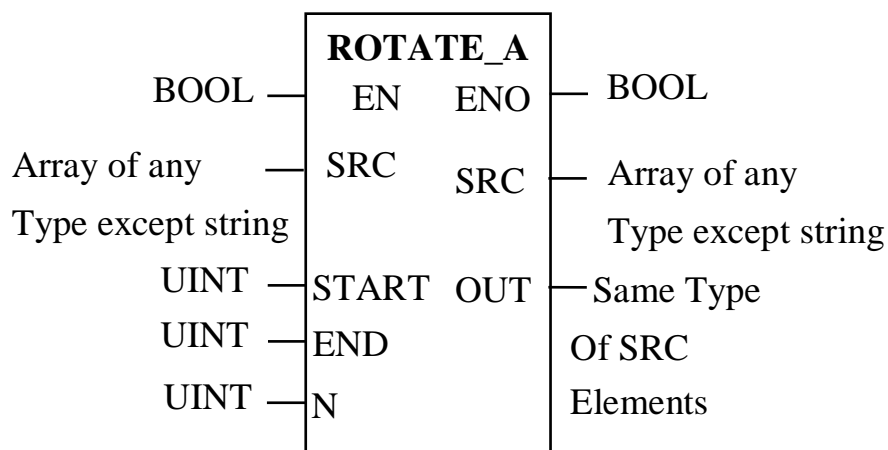


Figure 8: Example 4 ladder diagram

Rotates Designated Array Elements.

The Rotate designated array elements (ROTATE_A) shifts the sub array elements starting from the start shifting location (Start) and terminating at the end shifting location (End). Each time the instruction enable bit (EN) experiences a positive transition, the sub array elements are shifted N times. The sub array first N elements (from the Start position) are filled with the data coming from the end location. The data leaving the end position (over flowing data) is stored in the output variable OUT. The direction of shifting is determined by the comparison state of Start and End. For start>End, the shift direction is from the upper positions to the lower ones and vice versa. The source array (SRC) data type can be any type except the string type. Figure 9 shows the ladder symbol of the instruction under investigation and also the shifting movement for three different configurations.



SRC Initial vale { 1,2,3,4,5,6,7 } where 1 represents the content of SRC[0].			
Pulse number	1 st configuration OUT Start=0, End=6, N=1	2 nd configuration OUT Start=0, End=6, N=2	3 rd configuration OUT Start=1, End=5, N=1
1	7	6	6
2	6	4	5
3	5	2	4
4	4	7	3
5	3	5	2
6	2	3	6
7	1	1	5
8	7	6	4
9	6	4	3

Figure 9: ROTATE_A ladder symbol and behavior for Three different Configuration.

Example 5:

Propose ladder diagram to control the ON/OFF switching of eight lamps according to the data listed in figure 10. Assume the step rhythm to be 20 sec.

Step	Output State							
	8	7	6	5	4	3	2	1
1	0	0	0	1	1	0	0	0
2	0	0	1	0	0	1	0	0
3	0	1	0	0	0	0	1	0
4	1	0	0	0	0	0	0	1
5	0	1	0	0	0	0	1	0
6	0	0	1	0	0	1	0	0
7	0	0	0	1	1	0	0	0
8	0	0	0	0	0	0	0	0

Step	Output State							
	8	7	6	5	4	3	2	1
9	1	1	1	1	1	1	1	1
10	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0
13	1	1	1	1	1	1	1	1

Figure 10: Data table of example 5.

Figure 11 details the solution in which the array Y is initialized with the hexa decimal values { 18,24,42,81,42,24,18,00,FF,00,FF,00,FF}.

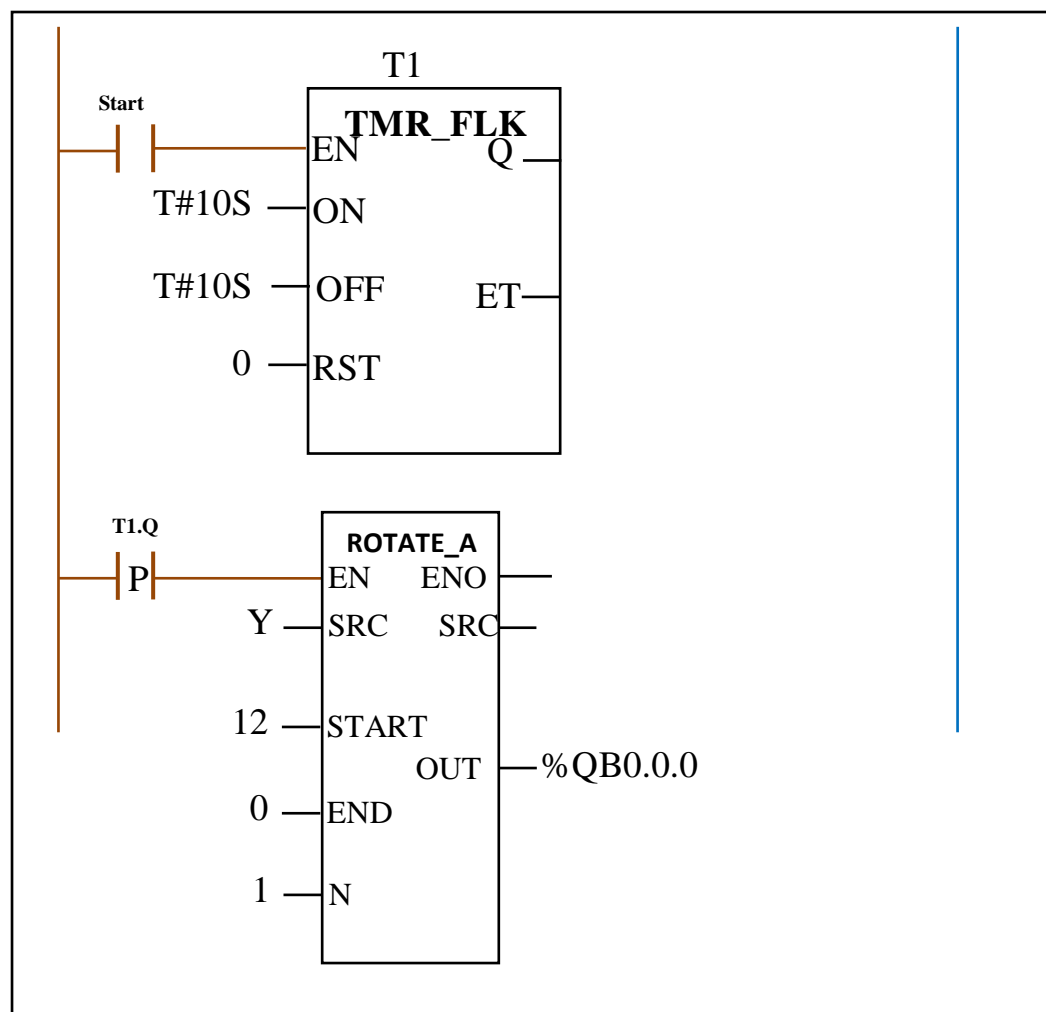
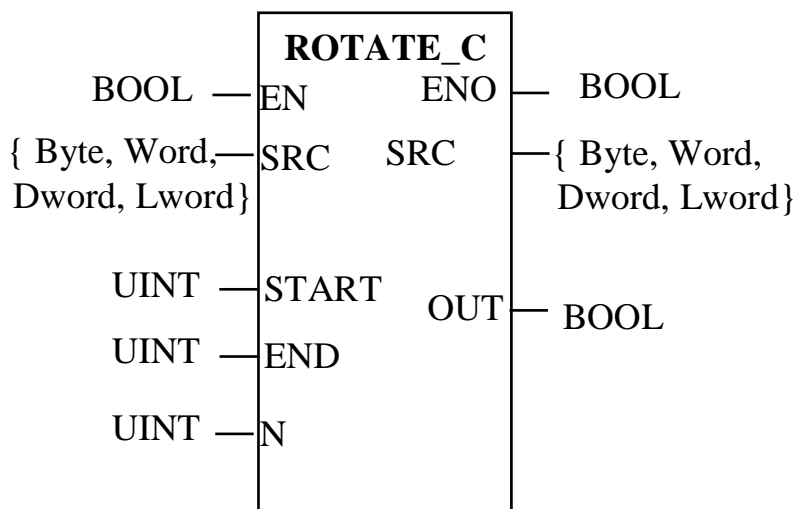


Figure 11: Example 5 ladder diagram

Rotate With Carry.

The rotate with carry instruction (ROTATE_C) shifts the sub bit string SRC (SRC may be Byte, Word, Sword, Lword) bits starting from the start shifting location (start) and terminating at the end shifting location(END). Each time the instruction enable bit (EN) experiences a positive transition, the sub string bits are shifted N times. The sub string first N bits (from the Start position) are filled with the data coming from the end location. The data leaving the end position (over flowing data) is stored in the Boolean output variable OUT. The direction of shifting is determined by the comparison state of Start and End. For start > End, the shift direction is from the upper positions to the lower ones and vice versa. Figure 12 shows the ladder symbol of this shifting instruction and its behavior under different configurations..



SRC Initial vale { 16#12 }				
Pulse No.	1 st configuration SRC Start=0, End=6, N=1	2 nd configuration SRC Start=0, End=6, N=2	3 rd configuration SRC Start=6, End=0, N=1	4th configuration OUT Start=1, End=5, N=1
1	24	48	09	24
2	48	22	44	0A
3	11	9	22	14
4	22	24	11	28
5	44	11	48	12
6	09	44	24	24
7	12	12	12	0A
8	24	48	09	14
9	48	22	44	28

Figure 12: ROTATE_C ladder symbol and behavior for four different Configuration.

Example 6:

The water circulation for cooling tower is to be 24 hours. Propose a three water pump system in which each pump runs one hour and rests two.

Solution:

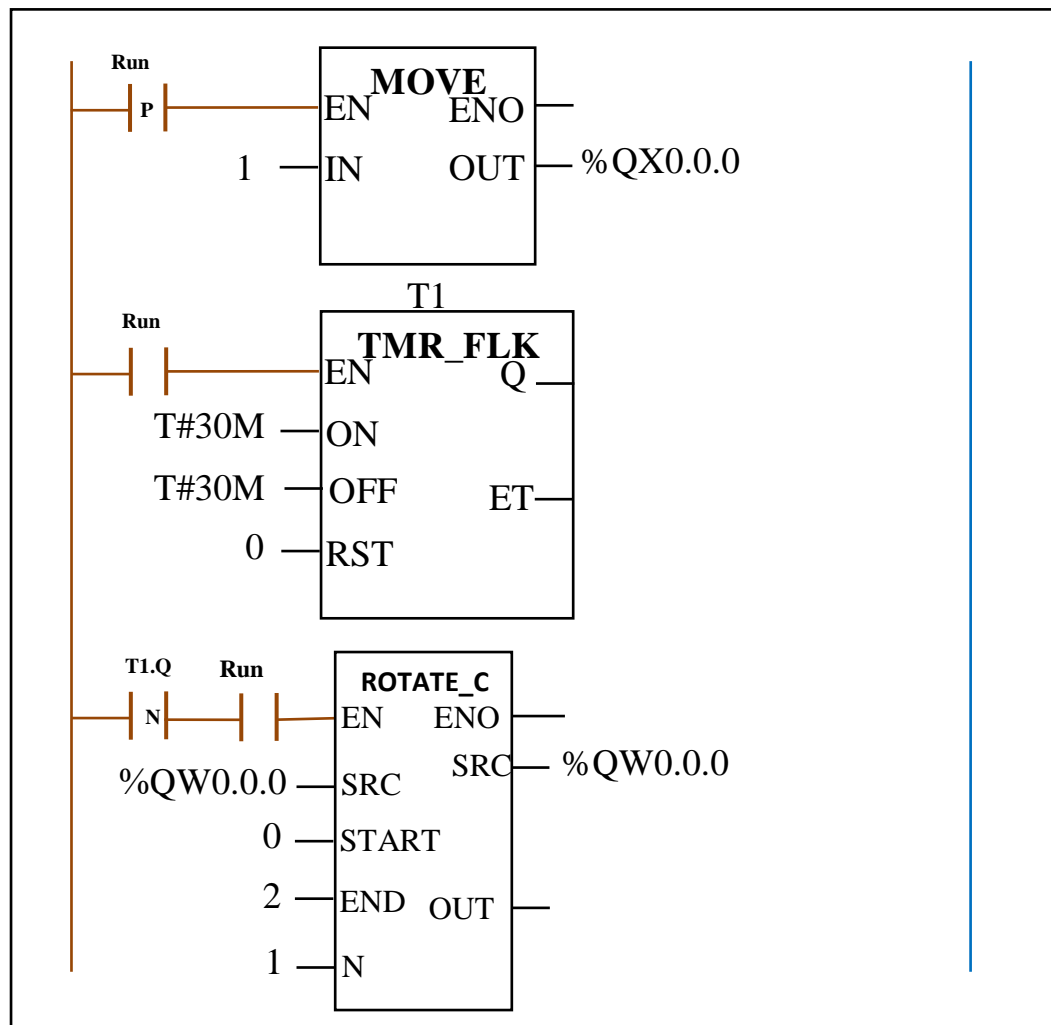
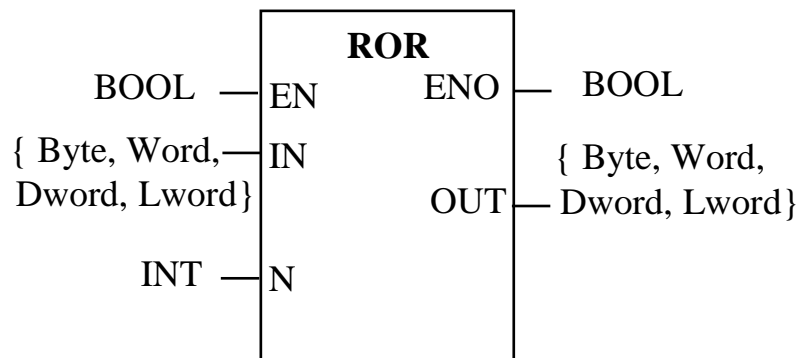


Figure 13: The proposed three pump controller

Rotate Right.

The rotate right instruction (ROR) shifts the image of the input bit string IN (Byte, Word, Dword, Lword) as N (shifting times) bits number, fills the leftmost locations with the values shifted out of the right most locations such that for M bits string, the least significant bit B_0 of IN is rounded to image $B_{(M-N)}$ bit. The modified (shifted and rounded) version of the input bit string is stored in the output variable OUT. Figure 14 shows the ladder symbol and enhances the aforementioned explanation.



IN vale {2#10101010}				
Bit	IN	OUT (N=1)	OUT (N=2)	OUT (N=3)
7	1	OUTB7=INB0=0	OUTB7=INB1=1	OUTB7=INB2=0
6	0	OUTB6=INB7=1	OUTB6=INB0=0	OUTB6=INB1=1
5	1	OUTB5=INB6=0	OUTB5=INB7=1	OUTB5=INB0=0
4	0	OUTB4=INB5=1	OUTB4=INB6=0	OUTB4=INB7=1
3	1	OUTB3=INB4=0	OUTB3=INB5=1	OUTB3=INB6=0
2	0	OUTB2=INB3=1	OUTB2=INB4=0	OUTB2=INB5=1
1	1	OUTB1=INB2=0	OUTB1=INB3=1	OUTB1=INB4=0
0	0	OUTB0=INB1=1	OUTB0=INB2=0	OUTB0=INB3=1

Figure 14: ROR ladder symbol and its behavior under different values of N

Example 7: Using ROR instruction, write ladder program to generate the following repeated sequence [16,8,4,2,1].

Solution :

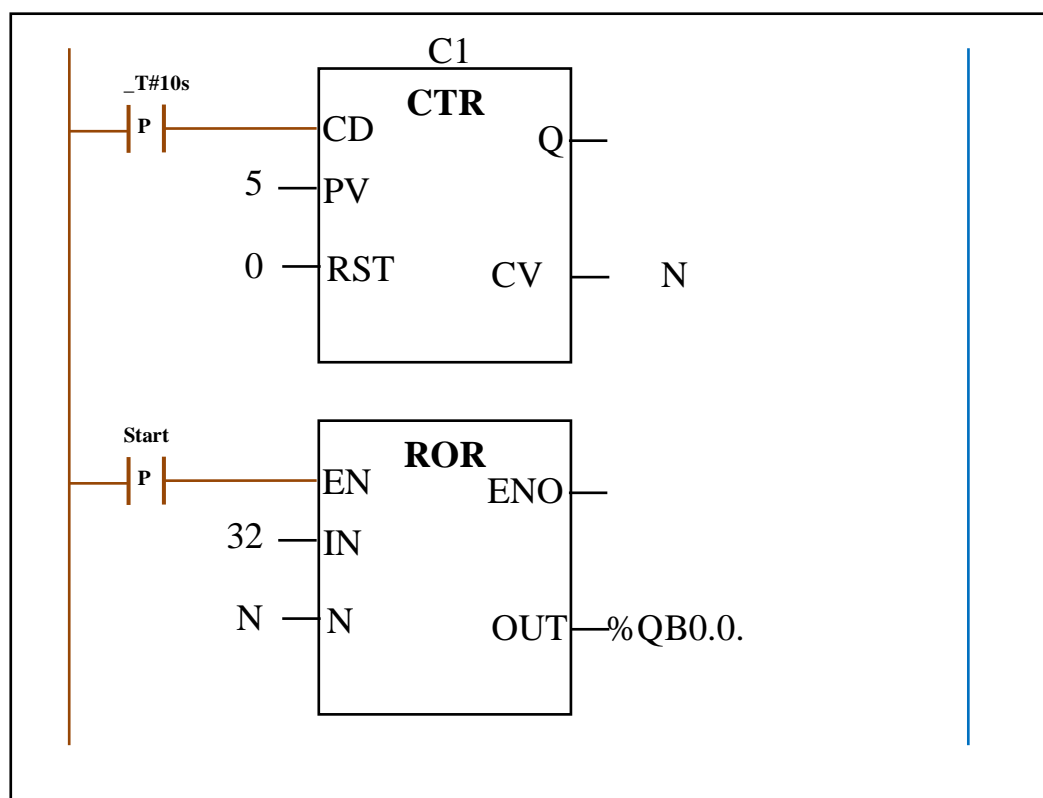


Figure 15: Ladder diagram of example 7

Rotate Left.

The rotate left instruction (ROL) shifts the image of the input bit string IN (Byte, Word, Dword, Lword) as N bits number, fills the rightmost locations with the values shifted out of the left most locations such that for M bits string, the most significant bit B_{M-1} of IN is rounded to image B_{N-1} bit. The modified version of the input bit string is stored in the output variable OUT. Figure 16 shows the ladder symbol and tabulates the left rotation of %MB3 for different values of N.

IN vale=%MB3= {2#10101010}				
Bit	IN	OUT (N=1)	OUT (N=2)	OUT (N=3)
7	1	OUTB7=INB6=0	OUTB7=INB5=1	OUTB7=INB4=0
6	0	OUTB6=INB5=1	OUTB6=INB4=0	OUTB6=INB3=1
5	1	OUTB5=INB4=0	OUTB5=INB3=1	OUTB5=INB2=0
4	0	OUTB4=INB3=1	OUTB4=INB2=0	OUTB4=INB1=1
3	1	OUTB3=INB2=0	OUTB3=INB1=1	OUTB3=INB0=0
2	0	OUTB2=INB1=1	OUTB2=INB0=0	OUTB2=INB7=1
1	1	OUTB1=INB0=0	OUTB1=INB7=1	OUTB1=INB6=0
0	0	OUTB0=INB7=1	OUTB0=INB6=0	OUTB0=INB5=1

Figure 16: ROL ladder symbol and its OUT value as function of N

Example 8: Using ROL instruction, write ladder program to generate the following repeated sequence [1,2,4,8,16].

Solution

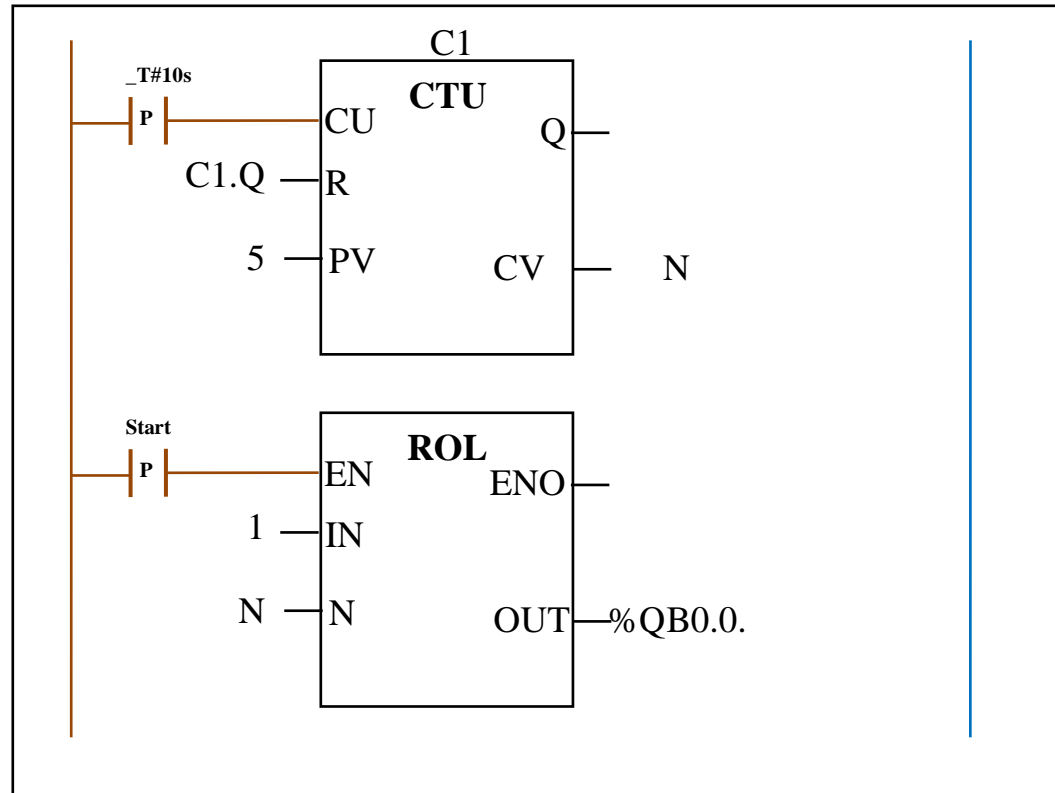


Figure 17: Ladder diagram of example 9.8 solution.

First IN First OUT.

The first-in-first-out (FIFO_XXXXX) is two gates controlled queue. Its inlet gate is controlled by the input marked LOAD, its exit gate is controlled by the UNLD input. The queue itself is an array of any type of data except the string one ($XXXXX \in \{\text{BOOL, BYTE, WORD, DWORD, LWORD, SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, TIME, DATE, TOD, DT}\}$). It has two flags and one pointer. The flags are the empty flag (EMPTY) which rises 1 when the queue is free from any item and the full flag (FULL) which declares there is no empty position in the queue (the queue is full). The pointer (PNT) points to the top of the queue, it ranges from 0 to the queue capacity. The in and out traffic of items are controlled by the queue request input (REQ). New item entering depends upon the states of the REQ, LOAD, the FULL, a new item is allowed to enter only if $\text{REQ}=1, \text{LOAD}=1, \text{and Full}=0$. Pushing items out of the queue depends upon

UNLD and REQ states, UNLD should be "1" and also REQ should be "1". The function block of this instruction is shown in figure 18.

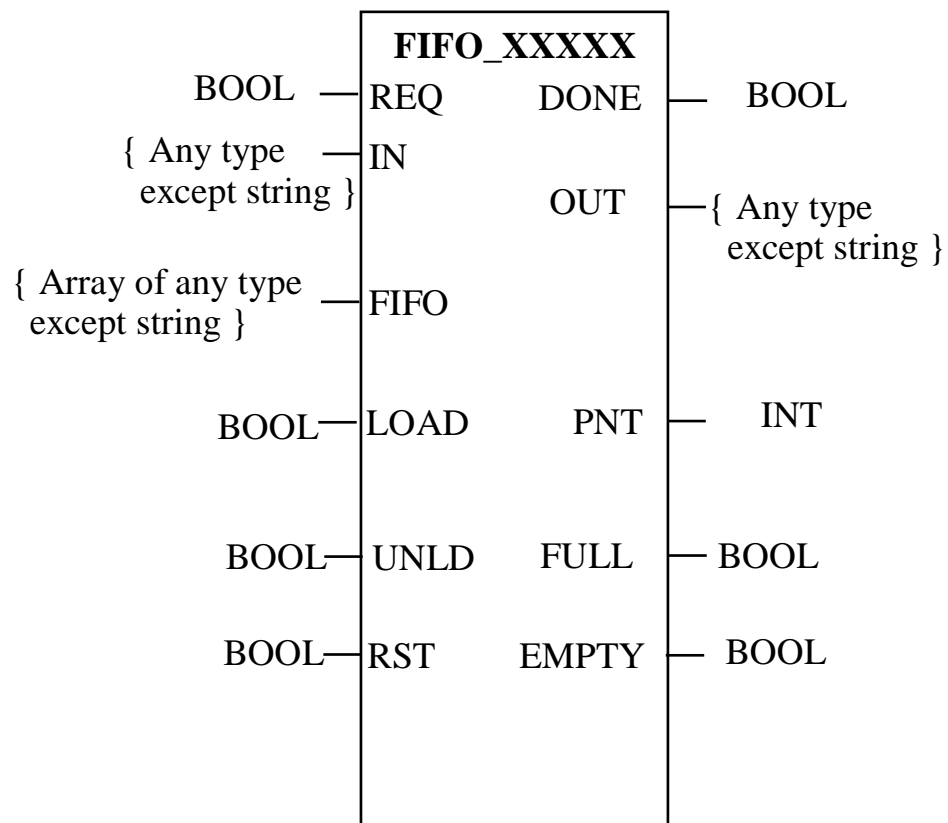


Figure 18: FIFO function block.

Example 9: Write ladder program to keep track of the last 12 values of a certain integer type variable .

Solution.

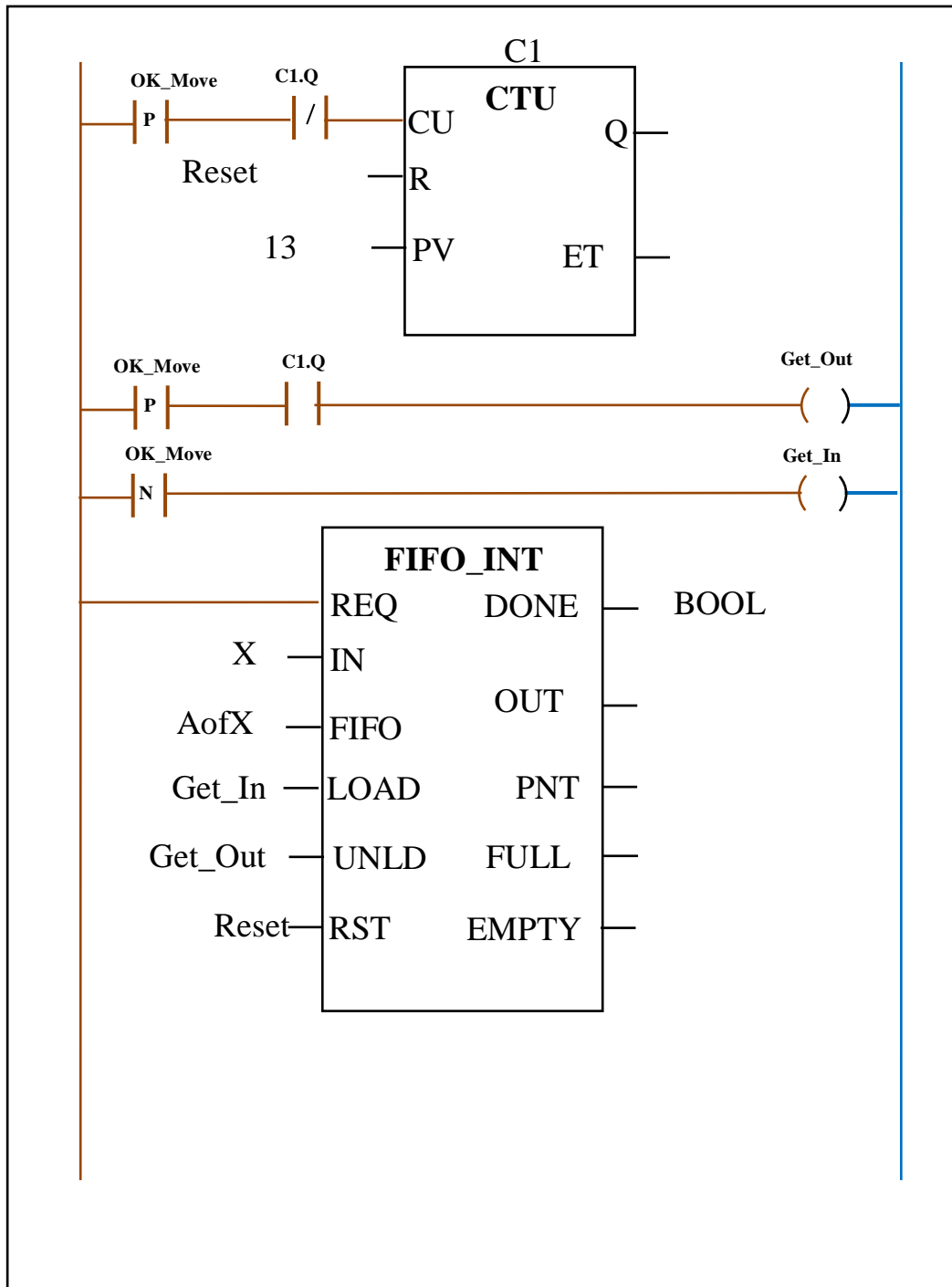


Figure 19:

Last IN First OUT.

The last-in-first-out (LIFO_XXXXX) is a stack in which the inlet and exit share the same terminal (opening) . The stack its self is an array of any type of data except the string one (XXXXX \in {BOOL, BYTE,WORD DWORD, LWORD, SINT,INT, DINT, LINT, USINT,UINT,UDINT,ULINT, REAL,LREAL,TIME,DATE,TOD, DT} . Loading the stack (entering new item) is done when the request is one (REQ=1),the stack is not full (FULL=0), and the load enable is active (LOAD=1). Unloading the stack(releasing the stack top item) is ok when the request is one and the unload input is active (UNLD=1. The stack instruction also includes an integer type pointer (PNT) to keep track of the top of the stack (current position). The function block of this instruction is well illustrated in figure 20.

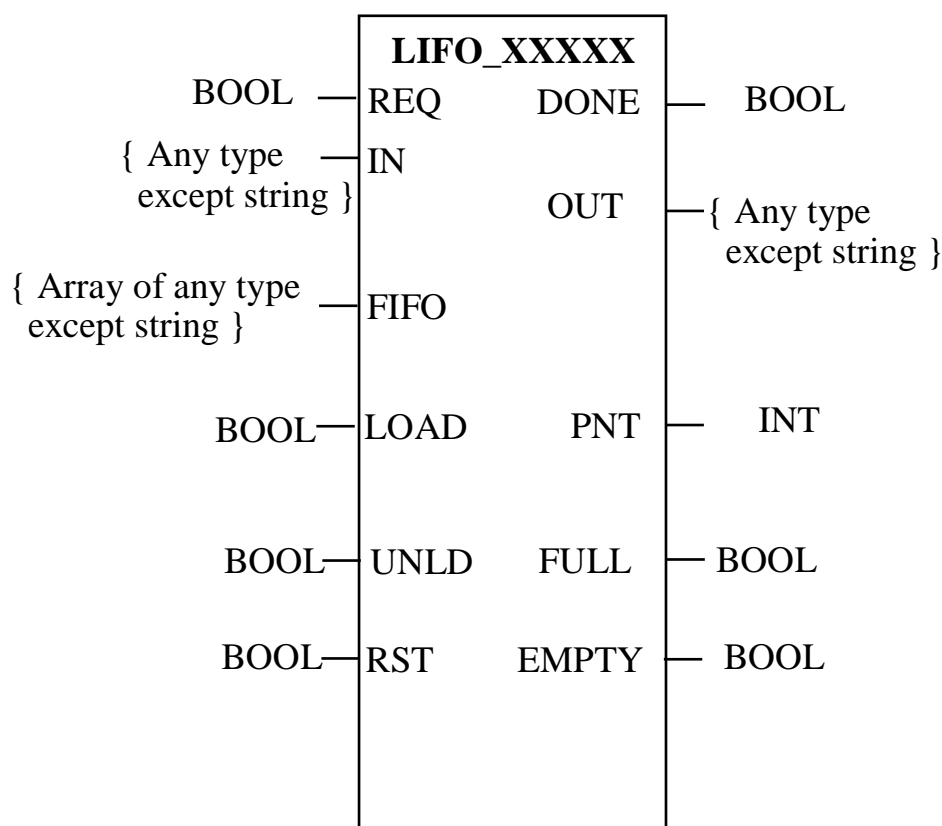


Figure 20: LIFO function block.

Example 10: Write ladder program to capture the first N values of a certain word type variable .

Solution:

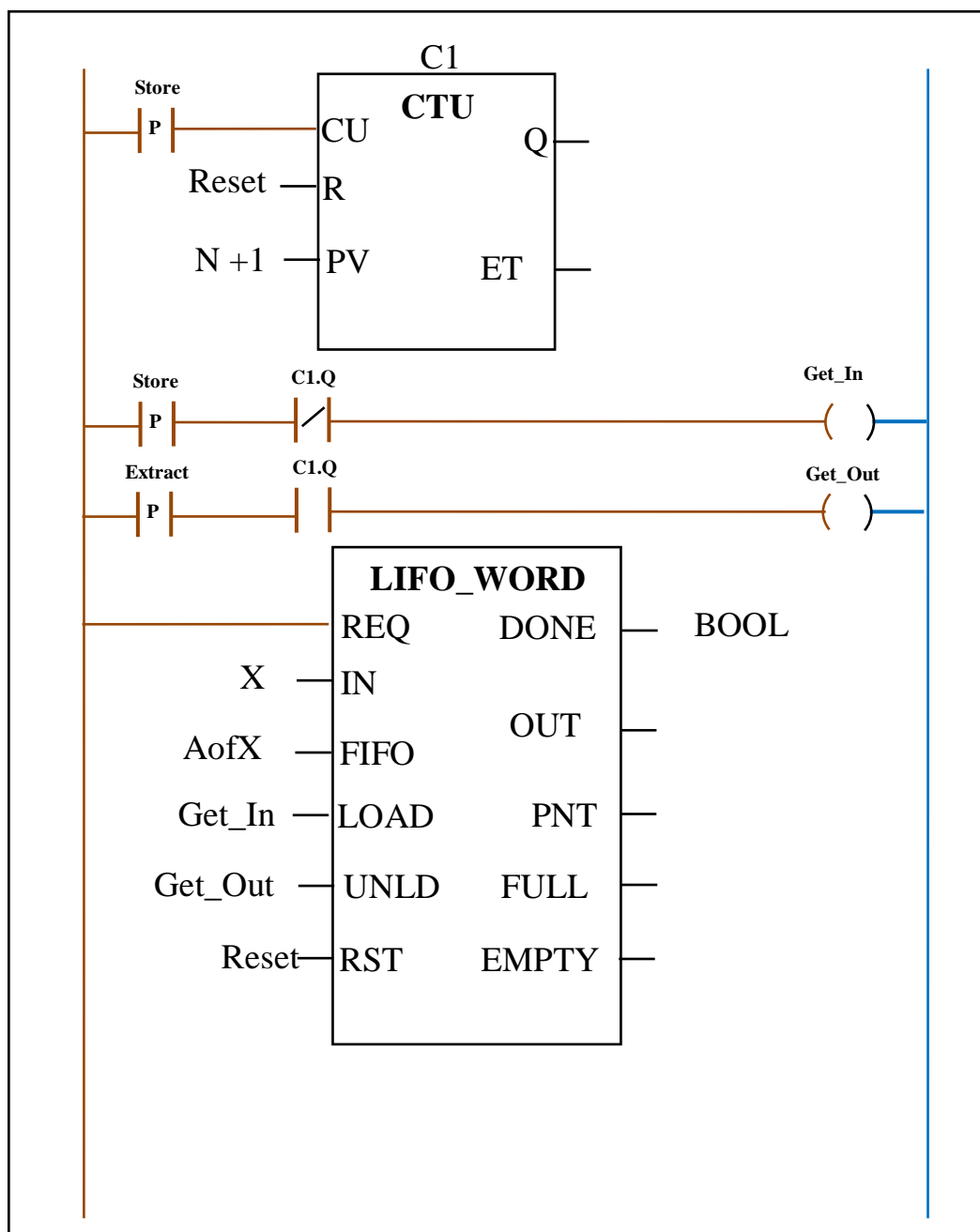


Figure 21: Storing the first N values