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

Industrial Automation (ICE 3252)

PLC Programming – IL, ST, FBD

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

INSTRUCTION LIST

- Uses very simple instructions similar to the original mnemonic
- Similar to assembly language programming
- All other programming languages can be converted to IL programs.
- Allen- Bradley version IL programming is detailed here.

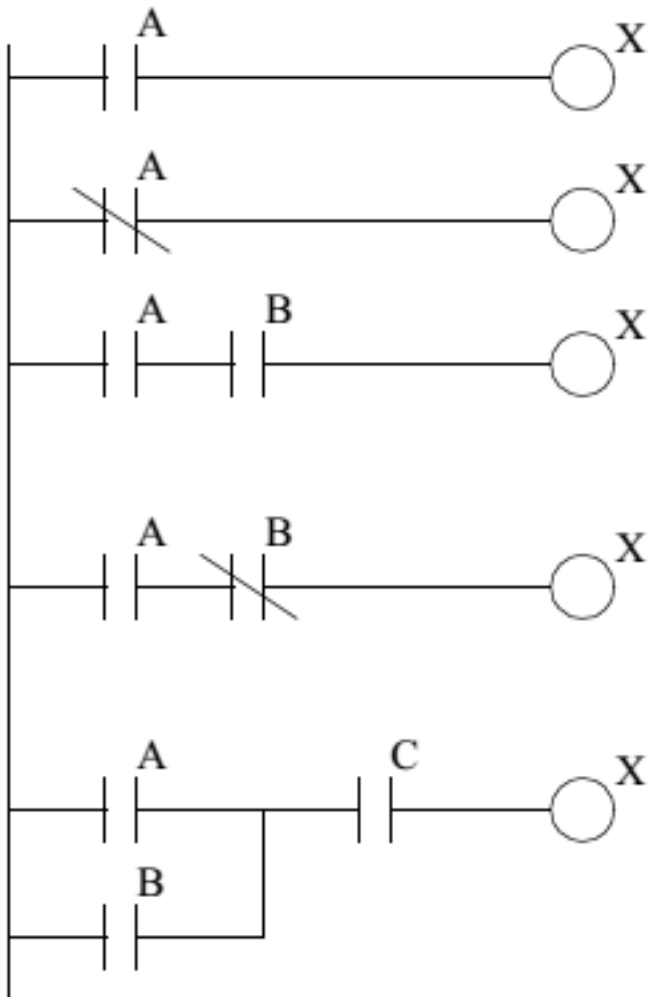
IL Operations

Operator	Modifiers	Data Types	Description
LD	N	many	set current result to value
ST	N	many	store current result to location
S, R		BOOL	set or reset a value (latches or flip-flops)
AND, &	N, (	BOOL	boolean and
OR	N, (	BOOL	boolean or
XOR	N, (	BOOL	boolean exclusive or
ADD	(	many	mathematical add
SUB	(	many	mathematical subtraction
MUL	(	many	mathematical multiplication
DIV	(	many	mathematical division
GT	(	many	comparison greater than >
GE	(	many	comparison greater than or equal >=
EQ	(	many	comparison equals =
NE	(	many	comparison not equal <>
LE	(	many	comparison less than or equals <=
LT	(	many	comparison less than <
JMP	C, N	LABEL	jump to LABEL
CAL	C, N	NAME	call subroutine NAME
RET	C, N		return from subroutine call
)			get value from stack

ANB and *ORB instructions*

- When a *LD* or *LDN* instruction is encountered it will put a value on the top of the stack.
- The ***ANB and ORB*** instructions will remove the top two values from the stack, and replace them with a single value that is the result of a Boolean operation.

IL equivalent



LD A
ST X

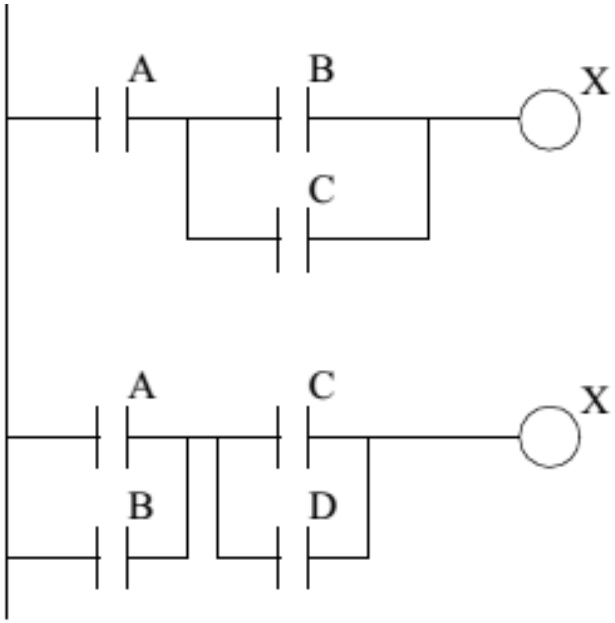
LDN A
ST X

LD A	LD A
LD B	AND B
ANB	ST X
ST X	

LD A	LD A
LDN B	ANDN B
ANB	ST X
ST X	

LD A	LD A
LD B	OR B
ORB	AND C
LD C	ST X
ANB	
ST X	

IL equivalent




```
LD A        LD A
LD B        LD B
LD C        OR C
ORB         ANB
ANB         ST X
ST X
```

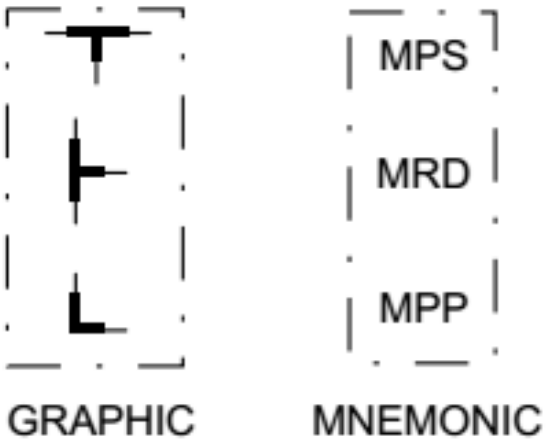
```
LD A        LD A
LD B        OR B
ORB         LD C
LD C        OR D
LD D        ANB
ORB         ST X
ANB
ST X
```

Multiple outputs: MPS/ MRP/MPP

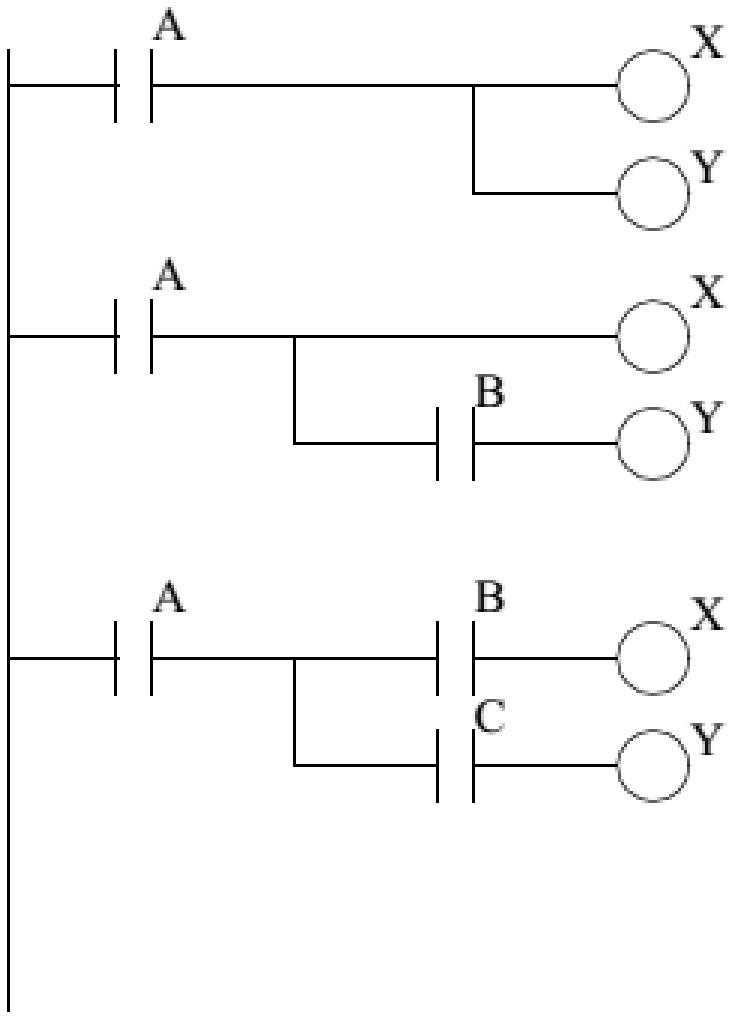
MPS .....Multiple Point Start

MRD .....Multiple Read Down

MPP .....Multiple Point Period



IL programming(continued)




```
LD A
ST X
ST Y
```

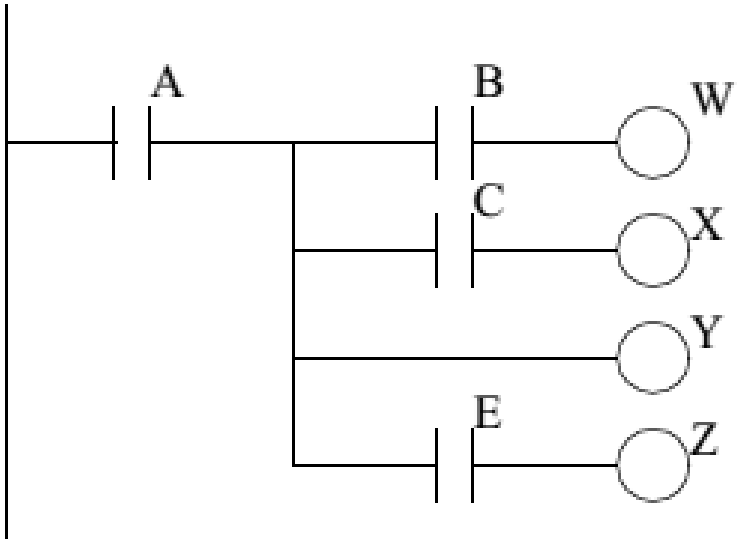
```
LD A
ST X
LD B
ANB
ST Y
```

```
LD A
ST X
AND B
ST Y
```

```
LD A
MPS
LD B
ANB
ST X
MPP
LD C
ANB
ST Y
```

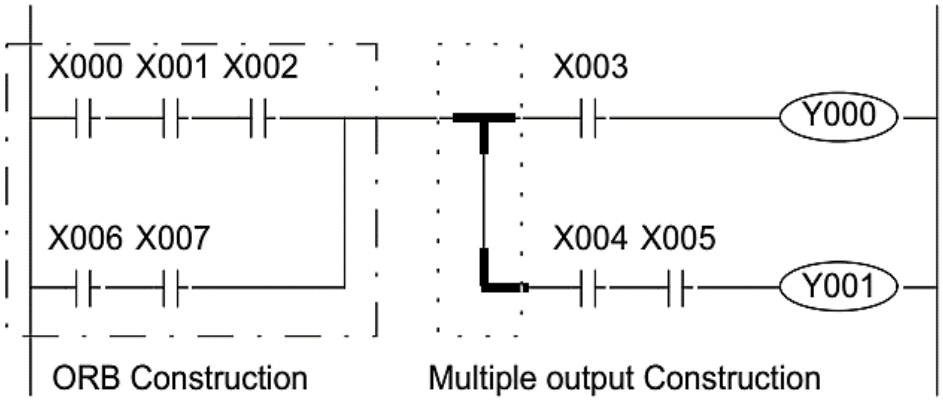
```
LD A
MPS
AND B
ST X
MPP
AND C
ST Y
```

IL programming(continued)



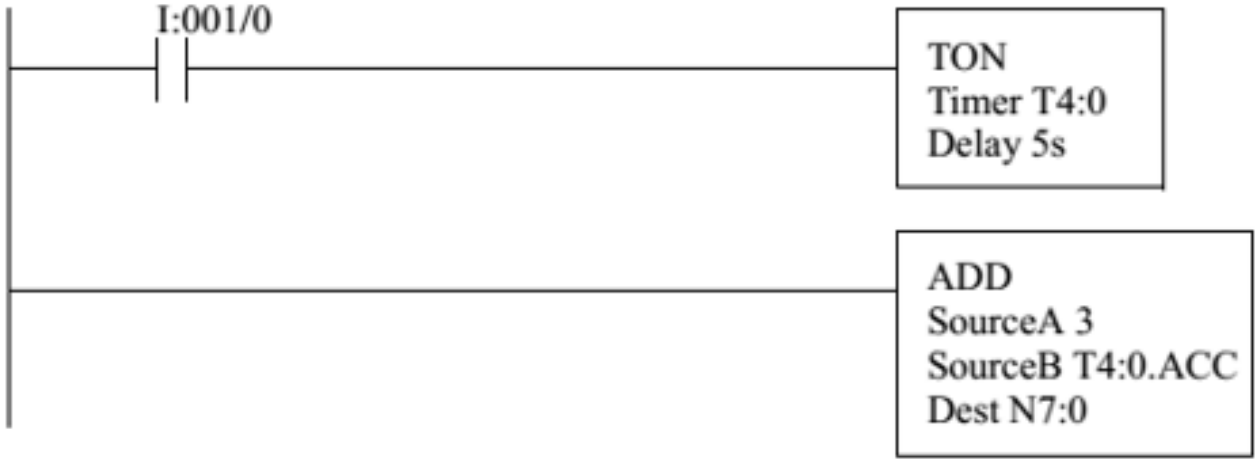

```
LD A        LD A
MPS         MPS
LD B        AND B
ANB         ST W
ST W        MRD
MRD         AND C
LD C        ST X
ANB         MRD
ST X        ST Y
MRD         MPP
STY         AND E
MPP         ST Z
LD E
ANB
ST Z
```

IL programming(continued)



PROGRAM STEP No.	INSTRUCTION PROGRAM		
0	LD	X	000
1	AND	X	001
2	AND	X	002
3	LD	X	006
4	AND	X	007
5	ORB		
6	MPS		
7	AND	X	003
8	OUT	Y	000
9	MPP		
10	AND	X	004
11	AND	X	005
12	OUT	Y	001
13	END		

IL programming(continued)




```
START:LD I:001/0
      TON(T4:0, 1.0, 5, 0)
      LD 1
      ADD (3, T4:0.ACC, N7:0)
      END
```

```
TON
TIMER ON DELAY
TIMER T4:0
TIME BASE 1.0
PRESET 5
ACCUM 0
```

STRUCTURED TEXT (ST) PROGRAMMING

ST PROGRAMMING

- The syntax of a high-level programming language with loops, variables, conditions and operators.
- A text-based PLC programming language.
- Your program will take up much smaller space
- And the flow/logic will be easier to read and understand.

Example:

```
PROGRAM stexample
  VAR
    x : BOOL;
  END_VAR
  x := TRUE;
  REPEAT
    x := FALSE;
  UNTIL x := FALSE;
  END_REPEAT;
END_PROGRAM;
```

Keywords

ST PROGRAMMING(continued)

- Variable X is defined in between two other keywords – **VAR** and **END_VAR**.

- Both the PROGRAM/END_PROGRAM and VAR/END_VAR are **constructs**.

- The PROGRAM construct is where all your PLC program is, and the VAR construct is where you define variables

ST PROGRAMMING(continued)

Declaration	Description
VAR	the general variable declaration
VAR_INPUT	defines a variable list for a function
VAR_OUTPUT	defines output variables from a function
VAR_IN_OUT	defines variable that are both inputs and outputs from a function
VAR_EXTERNAL	
VAR_GLOBAL	a global variable
VAR_ACCESS	
RETAIN	a value will be retained when the power is cycled
CONSTANT	a value that cannot be changed
AT	can tie a variable to a specific location in memory (without this variable locations are chosen by the compiler
END_VAR	marks the end of a variable declaration

Literal Number Examples

number type	examples
integers	-100, 0, 100, 10_000
real numbers	-100.0, 0.0, 100.0, 10_000.0
real with exponents	-1.0E-2, -1.0e-2, 0.0e0, 1.0E2
binary numbers	2#111111111, 2#1111_1111, 2#1111_1101_0110_0101
octal numbers	8#123, 8#777, 8#14
hexadecimal numbers	16#FF, 16#ff, 16#9a, 16#01
boolean	0, FALSE, 1, TRUE

Operators precedence

Operator	Description	Precedence
(...)	Parenthesized (bracketed) expression	Highest
Function(...)	List of parameters of a function	
**	Raising to a power	
-, NOT	Negation, Boolean NOT	
*, /, MOD	Multiplication, division, modulus operation	
+, -	Addition, subtraction	
<, >, <=, >=	Less than, greater than, less than or equal to, greater than or equal to	
=, <>	Equality, inequality	
AND, &	Boolean AND	
XOR	Boolean XOR	
OR	Boolean OR	Lowest

Data range

Integers:

IEC Data Type	Format	Range
SINT	Short Integer	-128 ... 127
INT	Integer	-32768 ... 32767
DINT	Double Integer	-2^31 ... 2^31-1
LINT	Long Integer	-2^63 ... 2^63-1
USINT	Unsigned Short Integer	0 ... 255
UINT	Unsigned Integer	0 ... 2^16-1
LDINT	Long Double Integer	0 ... 2^32-1
ULINT	Unsigned Long Integer	0 ... 2^64-1

Data range

Strings:

IEC Data Type	Format	Range
STRING	Character String	'My string'

Bit strings:

IEC Data Type	Format	Range
BOOL	Boolean	1 bit
BYTE	Byte	8 bits
WORD	Word	16 bits
DWORD	Double Word	32 bits
LWORD	Long Word	64 bits

Math Functions

:=	assigns a value to a variable
+	addition
-	subtraction
/	division
*	multiplication
MOD(A,B)	modulo - this provides the remainder for an integer divide A/B
SQR(A)	square root of A
FRD(A)	from BCD to decimal
TOD(A)	to BCD from decimal
NEG(A)	reverse sign +/-
LN(A)	natural logarithm
LOG(A)	base 10 logarithm
DEG(A)	from radians to degrees
RAD(A)	to radians from degrees
SIN(A)	sine
COS(A)	cosine
TAN(A)	tangent
ASN(A)	arcsine, inverse sine
ACS(A)	arccosine - inverse cosine
ATN(A)	arctan - inverse tangent
XPY(A,B)	A to the power of B
A**B	A to the power of B

All the operators in the table are sorted after **precedence (priority)**

Operation	Symbol	Precedence
Parenthesization	(expression)	Highest
Function Evaluation	MAX(A,B)	
Negation Complement	– NOT	
Exponentiation	**	
Multiply Divide Modulo	* / MOD	
Add Subtract	+ –	
Comparison	<, >, <=, >=	
Equality Inequality	= <>	
Boolean AND Boolean AND	& AND	
Boolean Exclusive OR	XOR	
Boolean OR	OR	Lowest

```
A + B * MAX(C, D)
```

Comparisons

>	greater than
>=	greater than or equal
=	equal
<=	less than or equal
<	less than
<>	not equal

Boolean Functions

AND(A,B)	logical and
OR(A,B)	logical or
XOR(A,B)	exclusive or
NOT(A)	logical not
!	logical not

Flow Control Functions

IF-THEN-ELSIF-ELSE-END_IF;	normal if-then structure
CASE-value:-ELSE-END_CASE;	a case switching function
FOR-TO-BY-DO-END_FOR;	for-next loop
WHILE-DO-END_WHILE;	

Points:

- **All statements are divided by semicolons**
  Structured Text consists of statements and semicolons to separate them.
- **The language is not case-sensitive**

- **Spaces have no function**
  But they should be used for readability.

ST functions

Function	Description
ABS(A);	absolute value of A
ACOS(A);	the inverse cosine of A
ADD(A,B,...);	add A+B+...
AND(A,B,...);	logical and of inputs A,B,...
ASIN(A);	the inverse sine of A
ATAN(A);	the inverse tangent of A
BCD_TO_INT(A);	converts a BCD to an integer
CONCAT(A,B,...);	will return strings A,B,... joined together
COS(A);	finds the cosine of A
CTD(CD:=A,LD:=B,PV:=C);	down counter active <=0, A decreases, B loads preset
CTU(CU:=A,R:=B,PV:=C);	up counter active >=C, A decreases, B resets
CTUD(CU:=A,CD:=B,R:=C,LD:=D,PV:=E);	up/down counter combined functions of the up and down counters

ST functions

DELETE(IN:=A,L:=B,P:=C);	will delete B characters at position C in string A
DIV(A,B);	A/B
EQ(A,B,C,...);	will compare A=B=C=...
EXP(A);	finds e**A where e is the natural number
EXPT(A,B);	A**B
FIND(IN1:=A,IN2:=B);	will find the start of string B in string A
F_TRIG(A);	a falling edge trigger
GE(A,B,C,...);	will compare A>=B, B>=C, C>=...
GT(A,B,C,...);	will compare A>B, B>C, C>...
INSERT(IN1:=A,IN2:=B,P:=C);	will insert string B into A at position C
INT_TO_BCD(A);	converts an integer to BCD
INT_TO_REAL(A);	converts A from integer to real
LE(A,B,C,...);	will compare A<=B, B<=C, C<=...
LEFT(IN:=A,L:=B);	will return the left B characters of string A
LEN(A);	will return the length of string A

ST functions

LOG(A);	base 10 log of A
LT(A,B,C,...);	will compare A<B, B<C, C<...
MAX(A,B,...);	outputs the maximum of A,B,...
MID(IN:=A,L:=B,P:=C);	will return B characters starting at C of string A
MIN(A,B,...);	outputs the minimum of A,B,...
MOD(A,B);	the remainder or fractional part of A/B
MOVE(A);	outputs the input, the same as :=
MUL(A,B,...);	multiply values A*B*....
MUX(A,B,C,...);	the value of A will select output B,C,...
NE(A,B);	will compare A <> B
NOT(A);	logical not of A
OR(A,B,...);	logical or of inputs A,B,...

ST functions

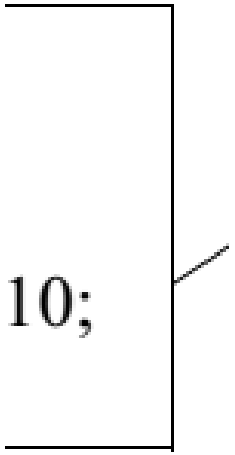
REAL_TO_INT(A);	converts A from real to integer
REPLACE(IN1:=A,IN2:=B,L:=C,P:=D);	will replace C characters at position D in string A wi string B
RIGHT(IN:=A,L:=B);	will return the right A characters of string B
ROL(IN:=A,N:=B);	rolls left value A of length B bits
ROR(IN:=A,N:=B);	rolls right value A of length B bits
RS(A,B);	RS flip flop with input A and B
RTC(IN:=A,PDT:=B);	will set and/or return current system time
R_TRIG(A);	a rising edge trigger
SEL(A,B,C);	if a=0 output B if A=1 output C
SHL(IN:=A,N:=B);	shift left value A of length B bits

ST functions

SHR(IN:=A,N:=B);	shift right value A of length B bits
SIN(A);	finds the sine of A
SQRT(A);	square root of A
SR(S1:=A,R:=B);	SR flipflop with inputs A and B
SUB(A,B);	A-B
TAN(A);	finds the tangent of A
TOF(IN:=A,PT:=B);	off delay timer
TON(IN:=A,PT:=B);	on delay timer
TP(IN:=A,PT:=B);	pulse timer - a rising edge fires a fixed period pulse
TRUNC(A);	converts a real to an integer, no rounding
XOR(A,B,...);	logical exclusive or of inputs A,B,...

Now analyze this program

```
PROGRAM main
VAR
        i : INT;
END_VAR
i := 0;
REPEAT
        i := i + 1;
        UNTIL i >= 10;
END_REPEAT;
END_PROGRAM
```



And this

```
FUNCTION sample
        INPUT_VAR
                                start : BOOL;  (* a NO start input *)
                                stop : BOOL;  (* a NC stop input *)
        END_VAR
        OUTPUT_VAR
                                motor : BOOL;(* a motor control relay
)
        END_VAR
        motor := (motor + start) * stop;(* get the motor output *)
END_FUNCTION
```

ST statements

Keyword	Description	Example	Explanation
:=	Assignment	d := 10;	Assignment of a calculated value on the right to the identifier on the left.
	Call of an FB [a]	FBName(Par1:=10, Par2:=20, Par3:=>Res);	Call of another POU of type FB including its parameters. ":=" for input parameters, "=>" for output parameters
RETURN	Return	RETURN;	Leave the current POU and return to the calling POU
IF	Selection	IF d < e THEN f:=1; ELSIF d=e THEN f:=2; ELSE f:= 3; END_IF;	Selection of alternatives by means of Boolean expressions.
CASE	Multi-selection	CASE f OF 1: g:=11; 2: g:=12; ELSE g:=FunName(); END_CASE;	Selection of a statement block, depending on the value of the expression "f".

ST statements

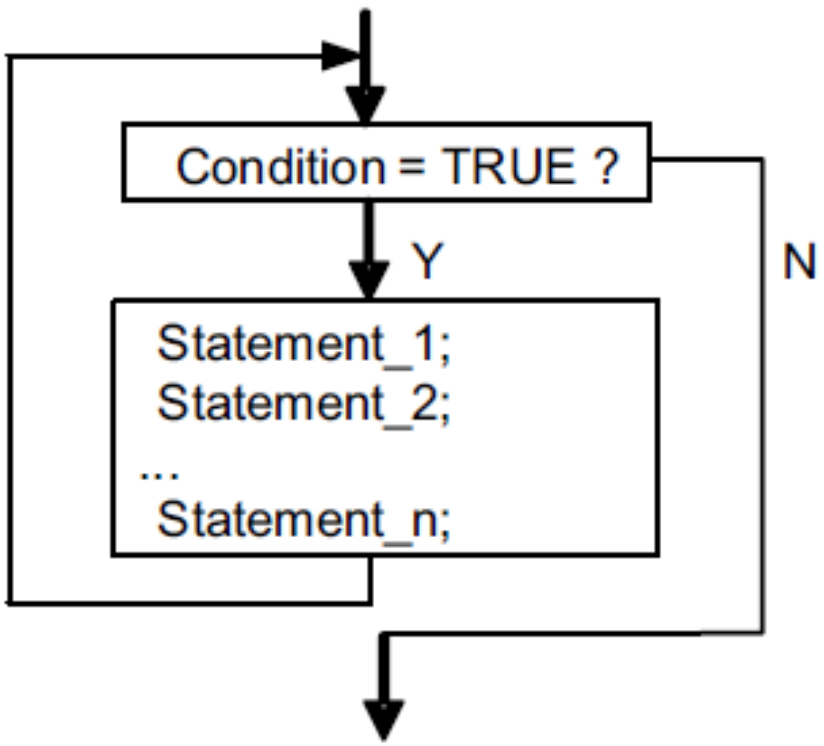
FOR	Iteration (1)	**FOR** h:=1 **TO** 10 **BY** 2 **DO**
		f[h/2] := h;
		END_FOR;
			Multiple loop of a statement block with a start and end condition and an increment value.

WHILE	Iteration (2)	**WHILE** m > 1 **DO**
		n := n / 2;
		END_WHILE;
			Multiple loop of a statement block with end condition at the beginning.

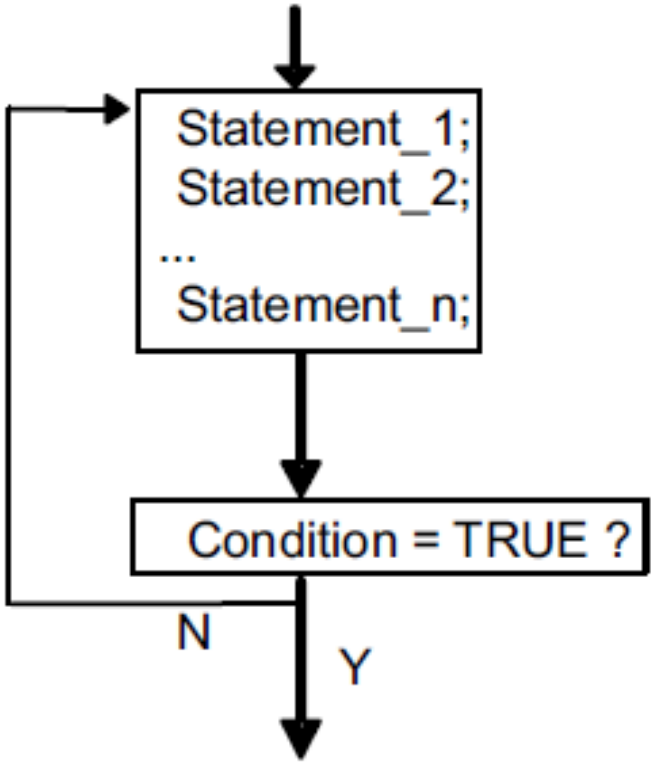
REPEAT	Iteration (3)	**REPEAT**
		i := i*j;
		UNTIL i < 10000
		END_REPEAT;
			Multiple loop of a statement block with end condition at the end.

EXIT	End of loop	**EXIT**;	Premature termination of an iteration statement
;	Dummy statement	; ;	

Comparison of while and repeat



a) WHILE



b) REPEAT

Example:

- Write a program to find the average of first five numbers floating point memory..

```
F8:10 := 0;
FOR (N7:0 := 0 TO 4) DO
        F8:10 := F8:10 + F8:[N7:0];
END_FOR;
```

```
F8:10 := 0;
WHILE (N7:0 < 5) DO
        F8:10 := F8:10 + F8:[N7:0];
        N7:0 := N7:0 + 1;
END_WHILE;
```

Example:

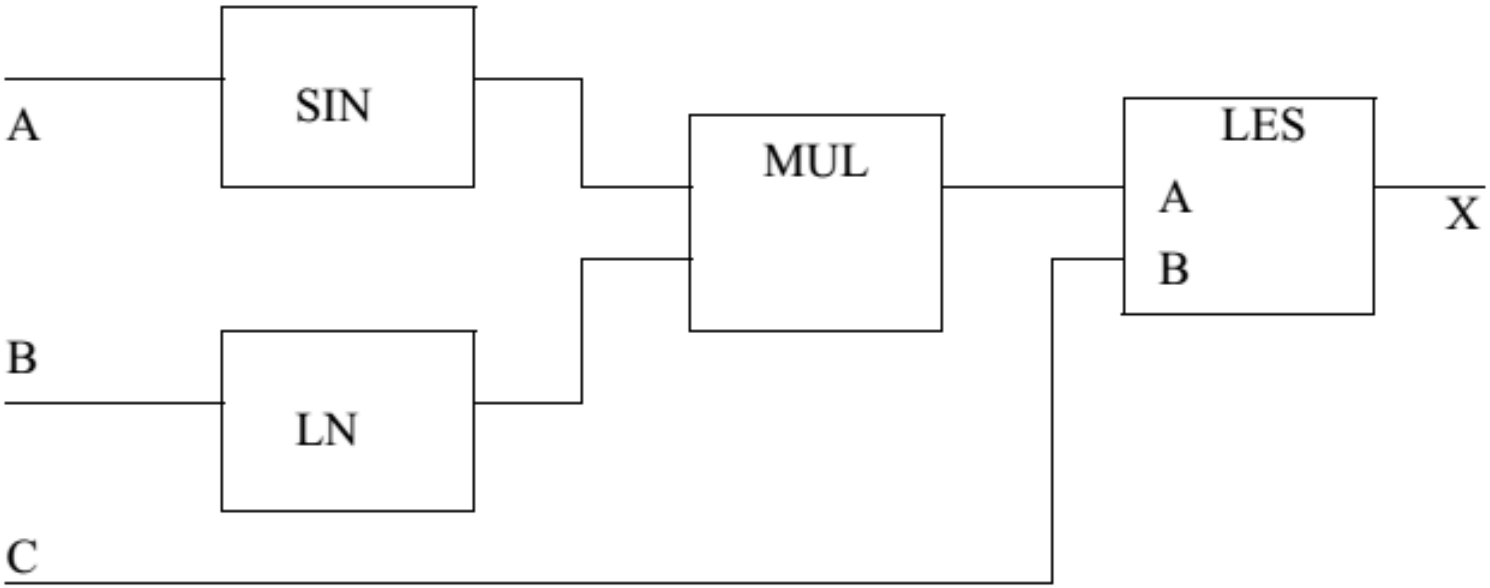
- Analyze the program…

```
IF (I:000/00 = 1) THEN
        O:001/00 := 1;
ELSIF (I:000/01 = 1 AND T4:0/DN = 1) THEN
        O:001/00 := 1;
        IF (I:000/02 = 0) THEN
        O:001/01 := 1;
        END_IF;
ELSE
        O:001/01 := 1;
END_IF;
```

FUNCTION BLOCK PROGRAMMING

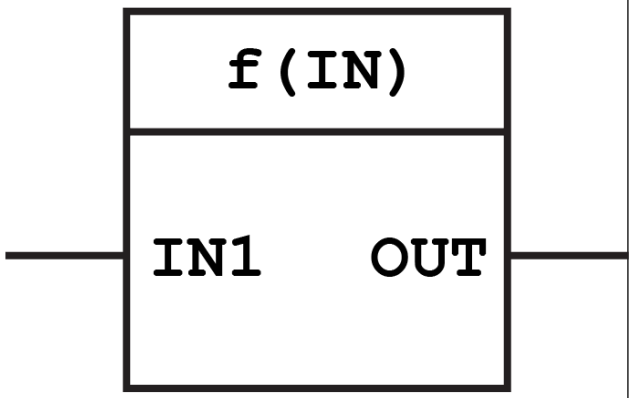
FBD

- FBD programming with respect to IEC 61131-3 standards.



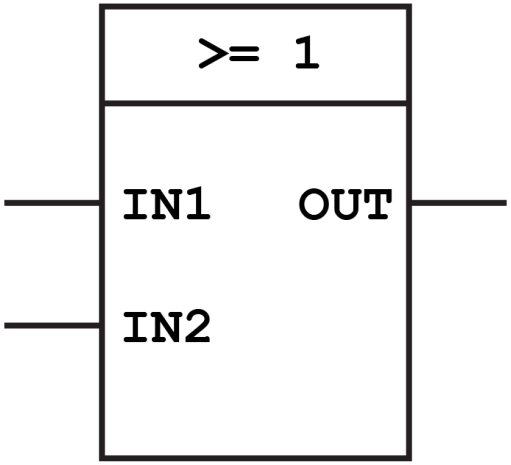
FBD

- The function block is illustrated with a box. In the middle of the box is often a symbol or a text. This symbol represents the actual functionality of the function block.
- Depending on the function there can be any number of inputs and outputs on the function block. You can connect the output of one function block to the input of another. Thereby creating a **Function Block Diagram.**



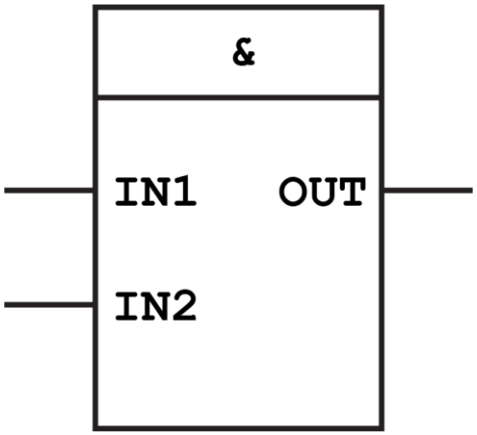
FBD

- **Bit Logic Function Blocks**
  - OR Logic Operation:the symbol for an OR operation is **>=1**. It is basically the condition for the output. If the sum of the two inputs are greater than or equal to 1, the output becomes true.

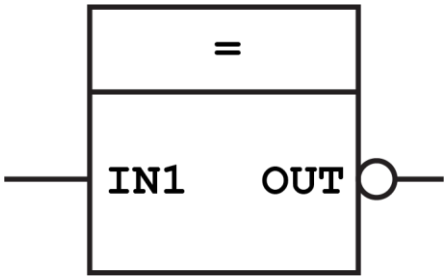


FBD

- AND Logic Operation:

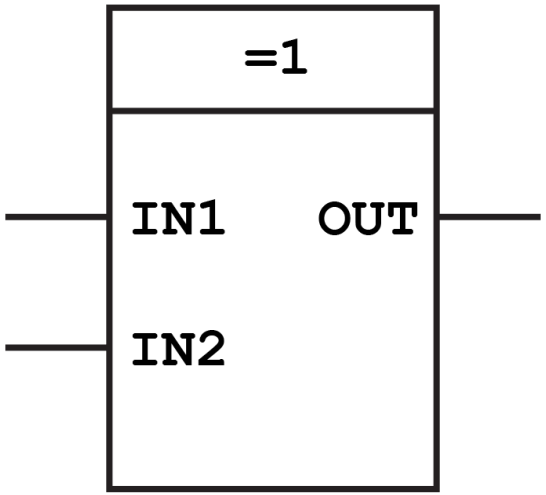


- Negation Operation



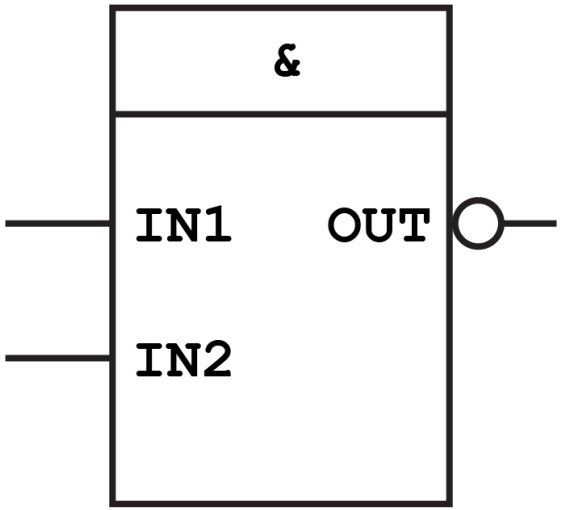
FBD

- Exclusive OR Operation: This block is a special case of the OR block. The input values on the OR block has to be greater than or equal to 1. But as you can see below, the Exclusive OR or just XOR block requires the two inputs to be equal to 1.

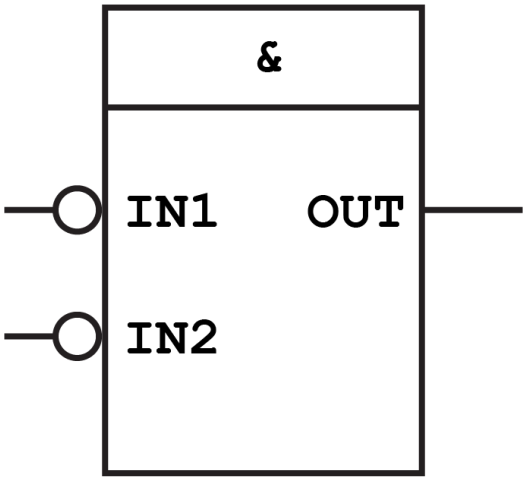


FBD

- NAND logic:

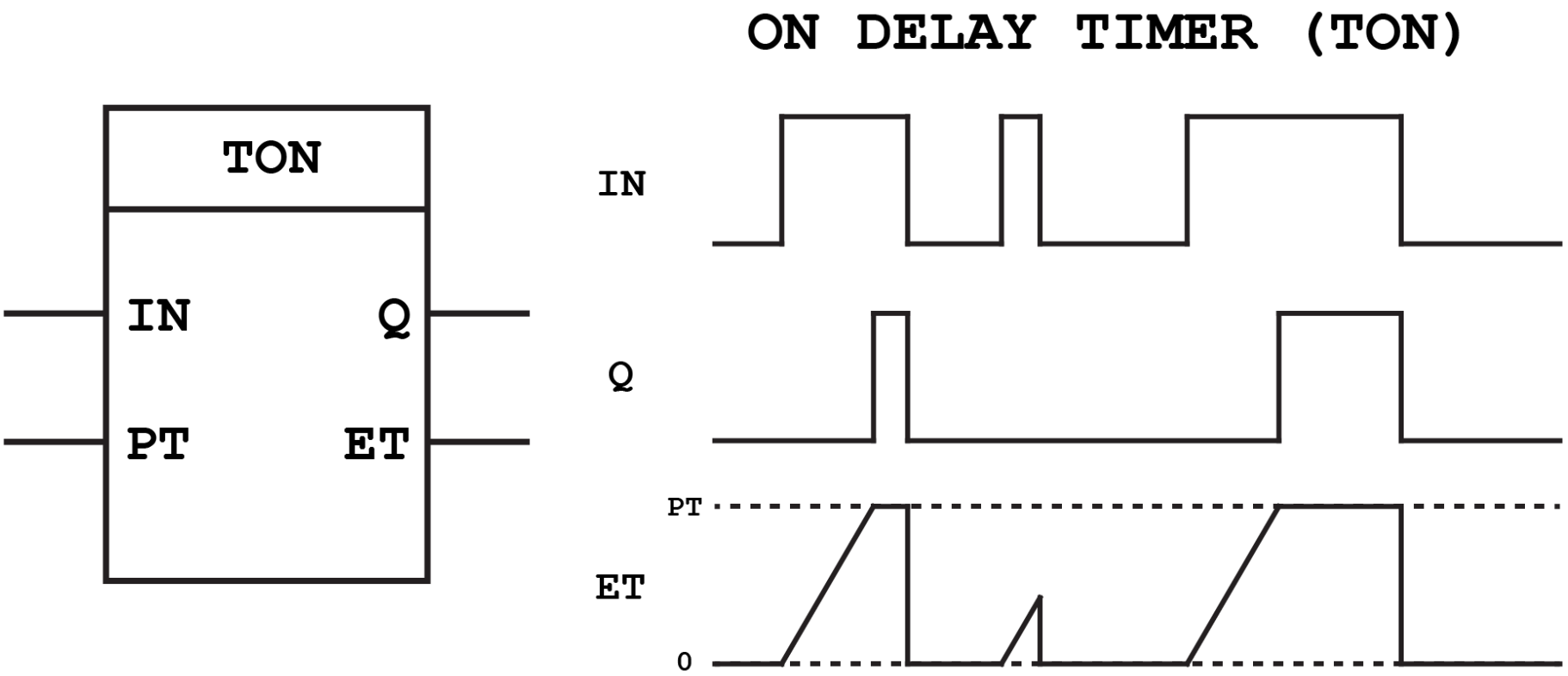


- NOR logic:



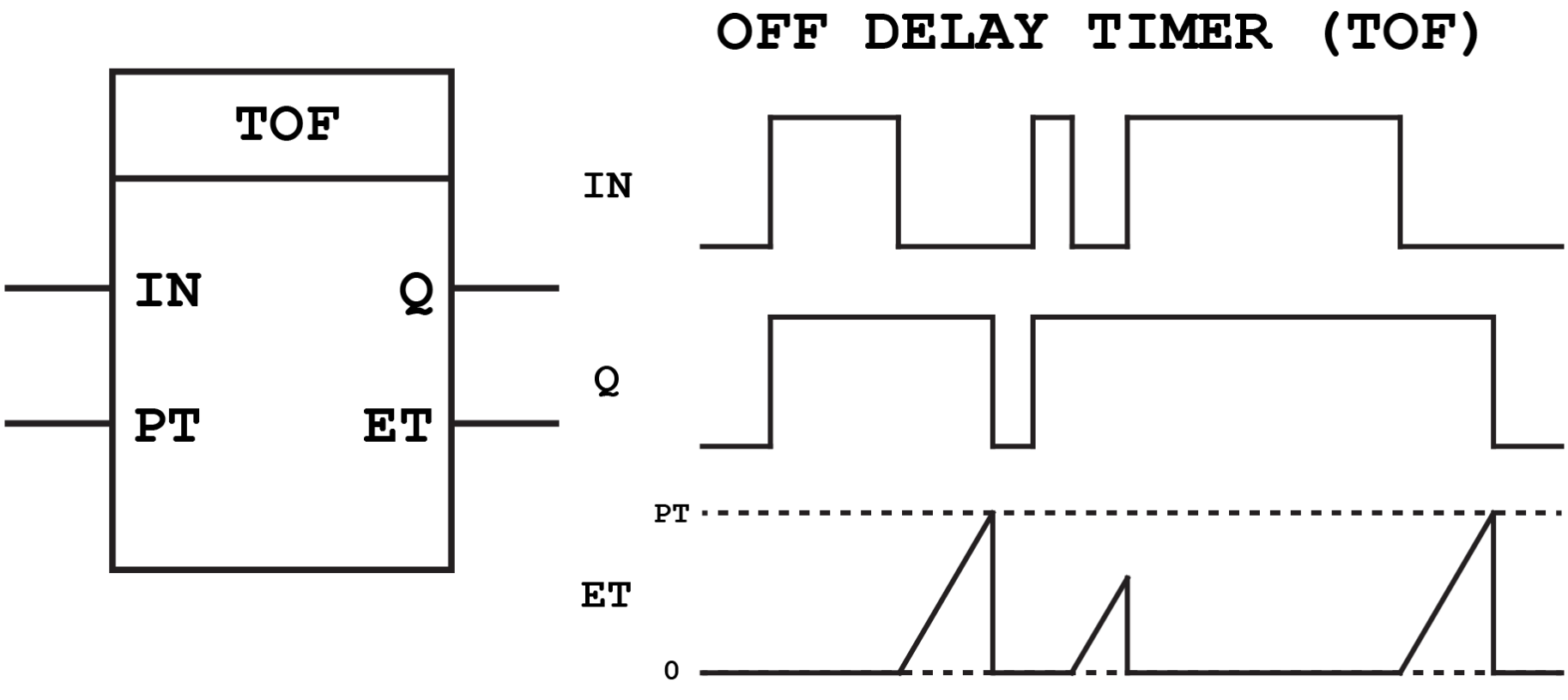
FBD

- On Delay Timer (TON):



FBD

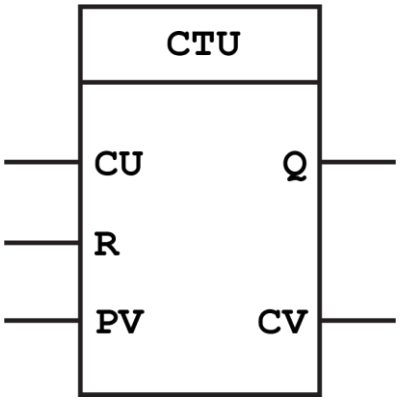
- Off Delay Timer (TOF):



FBD

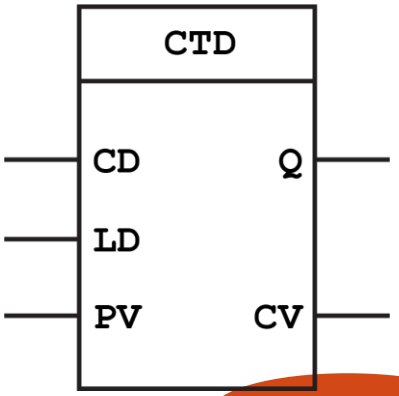
- Up Counter (CTU):

  Each pulse on CU will count CV up by 1. When CV >= PV then Q is set.



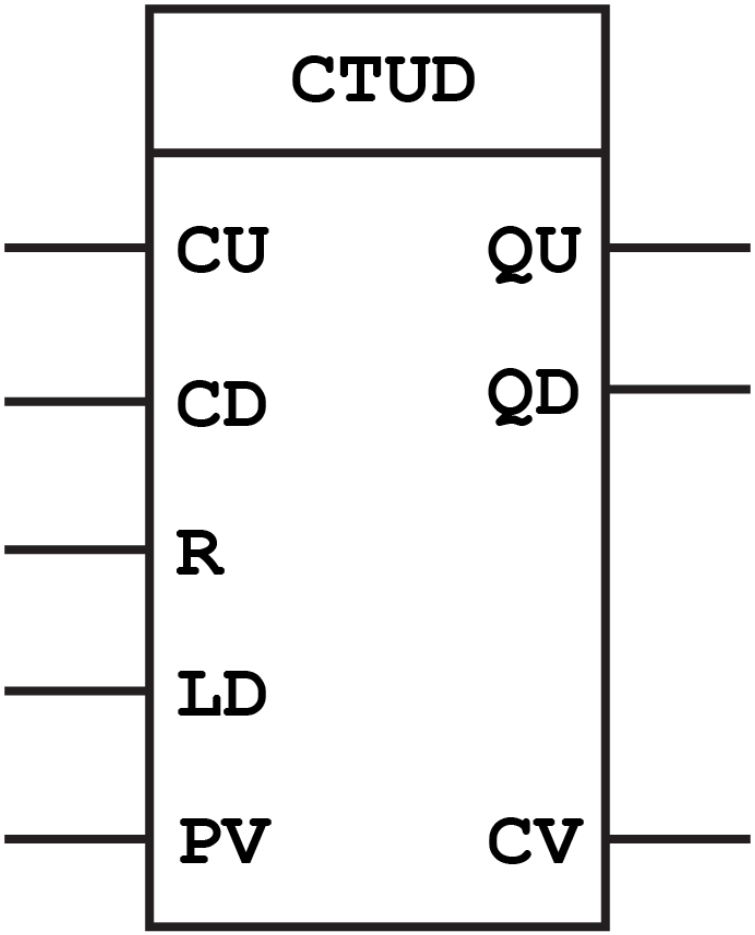
- Down Counter (CTD):

  Each pulse on CD will count CV down by 1. When CV <= 0 then Q is set.

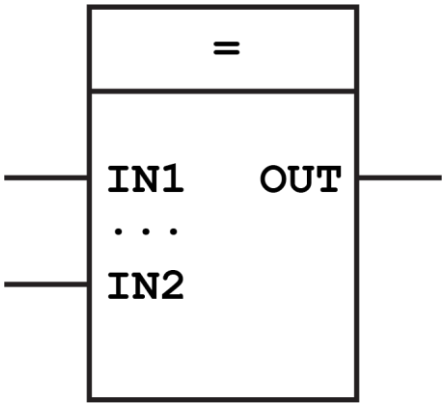


FBD

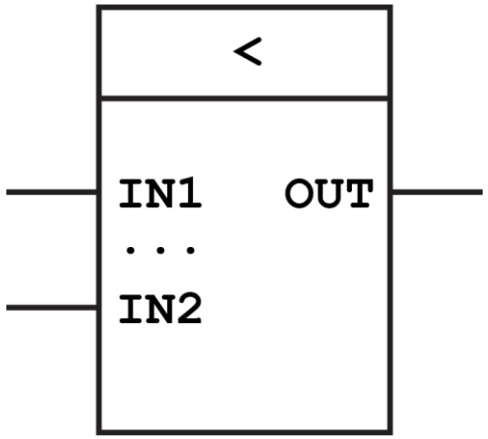
- Up Down Counters (CTUD):



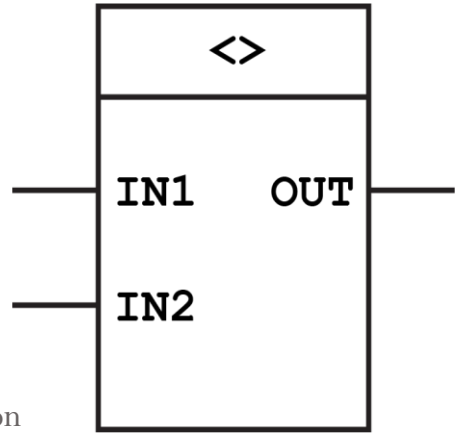
- Comparison functions
  - Equality



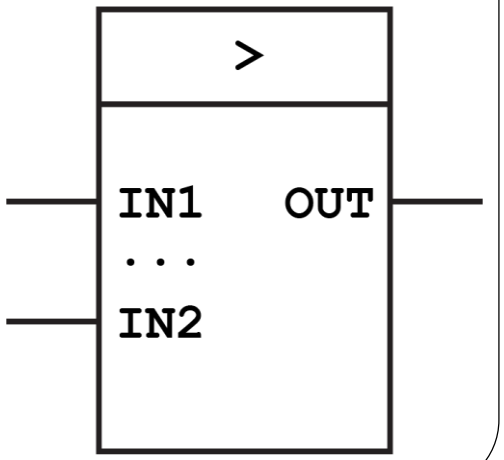
  - Less than



  - Inequality



  - Greater than

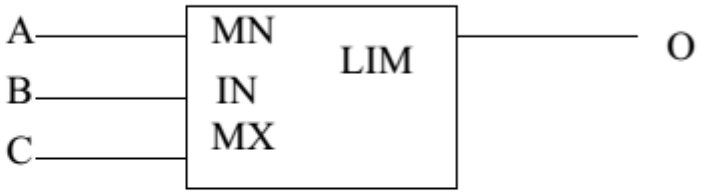
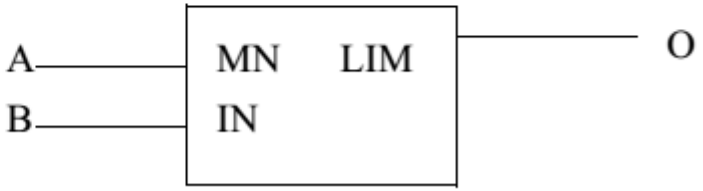


FBD

Structural Text Function	Function Block Equivalent
O := ADD(A, B);	

FBD

- OUT := MIN (MAX (IN, MN), MX)

Structural Text Function	Function Block Equivalent
O := LIM(MN := A, IN := B, MX := C);	
O := LIM(MN := A, IN := B);	

Reference

- Hugh Jack, *Automating Manufacturing Systems with PLCs*, (Version 4.7, April 14, 2005).