ionware Soft Logic Manual

ionware BETA
OUTSIDE THE BOX THINKING

Table of Contents

Licensing:

Instructions

The instructions are documented in detail elsewhere. The following is just a brief summary.

Instruction type	# of Instructions
Boolean input	8
Edge contact	6
Boolean output	8
Comparison	18
Program control	8
Counter and timer	6
Copy	5
Search	12
Shift register	1
Math operators and functions	26

Addressing Data Types

Type	Size (bytes)	Min Value	Max Value
Signed integer	2	-32768	32767
Signed double integer	4	-2147483648	2147483647
Floating point	Undefined	-1.5e+308[1]	1.5e+308[1]
Unsigned integer (hex)	2	0	65535
Single character	1	N/A	N/A
Text string	Undefined	N/A	N/A

[1] The maximum constant that can be entered is +/- 1.9e307

Data Table Addresses

All data is stored in data table addresses. These addresses are either "bit" (boolean) or "register" (word) addresses. Each data table begins at "1" (not "0"), and extends to the maximum address listed below.

Prefix	Description	Max	Type	Example
X	Inputs	2000	Boolean	X9
Y	Outputs	2000	Boolean	Y1023
C	Control relays	2000	Boolean	C58
T	Timer status	500	Boolean	T421
CT	Counter status	250	Boolean	CT6
SC	System control	1000	Boolean	SC21
DS	Data register	10000	Integer	DS6432
DD	Data register	2000	Double integer	DD4
DH	Data register	2000	Unsigned integer	DH100
DF	Data register	2000	Floating point	DF57
XD	Input register	125	Unsigned integer	XD1
YD	Output register	125	Unsigned integer	YD99
XS	Input register	125	Signed integer	XS1
YS	Output register	125	Signed integer	YS99
TD	Timer current value	500	Integer [1]	TD45
CTD	Counter current value	250	Double integer [1]	CTD3
SD	System data	1000	Integer	SD9
TXT	Text data	10000	ASCII	TXT7420

[1] Only positive numbers (0 to max) are allowed for these addresses.

Pointers or Indirect Addressing

Representation

Data table addresses must be represented exactly as defined. Case is significant, so while X1 is a valid address, x1 is not. Leading zeros are not permitted. X1 is a valid address, X01 is not.

Constants

Many instructions which accept registers addresses as inputs will also accept constants. The type of constant must be compatible with the parameter expected by the instruction.

	Type	Example
KInt	Signed integer	125
KDInt	Signed double integer	-50000
KF	Floating point	123.456 or 1.23456E+2
KHex	Unsigned integer (hex)	f73h
KTxtChar	Single character	"A"
KTxtStr	Text string	"abc123"

- Hexadecimal constants must have an "h" suffix to be valid. The "h" must be lower case.
- Text constants must be enclosed in double quotes (").
- Integers and double integers are distinguished by the magnitude of the value. An integer constant is assumed to be a normal (single) integer unless it exceeds the maximum permitted value, in which case it is automatically promoted to a double integer. Any instruction which expects a double integer will also accept a regular integer.
- Floating point numbers may be represented in decimal or exponential notation.
- Leading zeros are permitted, and are not significant.

System Control Relays

System control relays are used to provide convenience functions, or to signal error conditions.

- Error relays are set immediately by the instruction which encountered the error, and remain in effect until they are reset by another instruction which uses that relay, or until the system is restarted.
- Free running clock relays are updated at the start of each scan.
- The "always on" relay is set at the start of scan.

Address	Description
SC1	Always ON
SC2	ON for the first program scan
SC3	Turns on and off on alternate scans
SC4	Free running clock with a 10ms period
SC5	Free running clock with a 100ms period
SC6	Free running clock with a 500ms period
SC7	Free running clock with a 1sec period
SC8	Free running clock with a 1m period
SC9	Free running clock with a 1 hour period
SC19	ON if an error occurs
SC24	There is no PLC program, or the program is invalid.
SC25	The run time system is not compatible with the program version.
SC26	The watchdog timer timed out.[1]
SC27	The data table contents were lost.
SC40	An attempt was made to divide by zero.
SC43	Data overflow, underflow, or data conversion error.
SC44	An invalid address was used
SC46	A math error occurred
SC50	Set run mode to stop.[1]
SC51	Resets watchdog timer to zero.[1]

Notes: 1 = Not implemented at this time. The SC relays are not writable by a user instruction.

System Control Registers

System control registers are used to provide convenience functions, or to signal error conditions.

Address	Description
SD1	Current PLC error code[1]
SD5	Low word of runtime version[1]
SD6	High word of runtime version[1]
SD9	Counts number of scans (rolls over at 32,767)
SD10	The current scan time
SD11	The minimum scan time since starting
SD12	The maximum scan time since starting
SD20	Real time clock year
SD21	Real time clock month
SD22	Real time clock day
SD23	Real time clock day of week (where Mon = 0) [2]
SD24	Real time clock hour
SD25	Real time clock minute
SD26	Real time clock second

1 = Not implemented at this time.

2 = This is different than the Click hardware.

Subroutines

Subroutines are referenced by names which may be from 1 to 24 characters long and which may contain the characters a to z, A to Z and 1 to 9. No other characters are permitted, and no spaces are permitted within the name. Examples, "SubroutineName1", or "123ValidName". The following are invalid names "ThisNameIsMuchTooLongToBeAccepted", and "Bad&Characters*"

There is no limit to the number of subroutines permitted, but each subroutine name must be unique.

The Logic Stack

A logic stack is one of the fundamental features of all conventional PLCs, although its presence is often not explicitly mentioned. The logic stack is the location where the results of logic operations are stored, and where many instructions implicitly draw one or more of their Boolean parameters.

The logic stack is implemented as a stack. This stack has an undefined maximum size, but the limit may be assumed to be very large, and for all practical purposes unlimited. The minimum size is zero.

Instructions which use the logic stack may push a value onto the stack, modify the top of the stack, read the top of the stack, or read any value below the top of the stack. In the instruction documentation, the top of the logic stack is referred to as "the top of the logic stack", "top of stack", or just "logic stack". Values in locations below the top of the stack are referred to as the logic stack value (top - 1) (the value just below the top of stack), or logic stack value (top - 2)(the value two positions below the top of stack). No instruction reads more than two positions below the top of stack.

Whenever a new network (rung) is started, the logic stack is cleared and the top of the stack is initialised to "false". If an instruction attempts to read a logic stack location which does not exist a value of "false" is returned to the instruction. This means that attempting to read an invalid logic stack location does not result in a run time error, but it does mean the instruction will never become true.

When a subroutine is called, the logic stack is re-initialised. When the subroutine returns, the state of the logic stack is undefined. This means that a subroutine must not rely on the logic stack being in a particular state when called, and the calling location must not rely on the logic stack being either set by the subroutine or being in the same state as before the subroutine call.

```
// Re-initialise the logic stack.
// Push a value onto the stack. The top of stack is now true.
NETWORK 1
STR SC1

// Re-initialise the logic stack again.
// The previous value is now gone.
NETWORK 2

// Re-initialise the logic stack again.
// Push four values onto the stack. The top is true,
// (top - 1) is false, (top - 2) and (top - 3) are true.
NETWORK 3
STR SC1
STR SC1
STRN SC1
STR SC1

// Re-initialise the logic stack again.
// The previous values are now gone.
NETWORK 4
```

Program Formatting Instructions

- **//** - Comments.
- **Network** - Starting rungs.

Program formatting instructions do not directly affect the program flow or results, but they do act to help document and organise a program.

Instruction	Description	# Params	Parameters
//	Comment line	Undefined	Arbitrary text
NETWORK	Network	1	Integer

Anything following a comment token ("//") is considered to be a comment and is ignored. The comment token must be separated from the following comment text by one or more spaces.

When displayed in ladder format, all comments in a rung are gathered together into a single block and displayed just below the "NETWORK" instruction. This affects the display of comments only, and does not affect the original program file.

Example:

```
// This is a comment.
```

A network is also often known as a "rung". The "NETWORK" token is an instruction that resets the logic stack and begins a new block of code. The NETWORK instruction takes an one integer as a parameter. Network (rung) numbers do not have to be unique or consecutive.

Network numbers are used to report syntax errors when the program is compiled, and are also used to report run time errors.

Example:

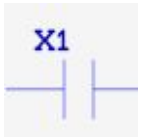
```
NETWORK 1
```

Boolean Input Instructions

- **AND** - AND bit with top of logic stack.
- **ANDN** - AND NOT bit with top of logic stack.
- **ANDSTR** - AND top two values on logic stack.
- **OR** - bit with top of logic stack.
- **ORN** - OR NOT bit with top of logic stack.
- **ORSTR** - OR top two values on logic stack.
- **STR** - Store bit onto logic stack.
- **STRN** - Store NOT bit onto logic stack.

Boolean input instructions read a Boolean value from the data table or the logic stack, and output the result to the logic stack.

Instruction	Description	# Params	X	Y	C	T	CT	SC	Symbol
AND	AND bit with top of logic stack	1	X	X	X	X	X	X	
ANDN	AND NOT bit with top of logic stack	1	X	X	X	X	X	X	
ANDSTR	AND top two values on logic stack	0							
OR	OR bit with top of logic stack	1	X	X	X	X	X	X	
ORN	OR NOT bit with top of logic stack	1	X	X	X	X	X	X	
ORSTR	OR top two values on logic stack	0							

STR	Store bit onto logic stack	1	X	X	X	X	X	X	
STRN	Store NOT bit onto logic stack	1	X	X	X	X	X	X	

Boolean input instructions have two types of input parameters, explicit and implicit. A Boolean input instruction will accept no more than one explicit parameter, and zero, one, or two implicit parameters.

- An explicit parameter is a Boolean (bit) address such as X, Y, C, etc.
- An implicit parameter is the value on the top of the logic stack.
- Some instructions use the top two logic stack values as implicit parameters.

All Boolean input instructions output their result by modifying the logic stack.

Boolean input instructions fall into three categories: AND, OR, and STORE.

- AND instructions perform a logical AND on two inputs and store the result by replacing the original value on the top of the logic stack with the new value.
- OR instructions perform a logical OR on two inputs and store the result by replacing the original value on the top of the logic stack with the new value.
- STORE instructions simply store a value on the top of the logic stack, pushing the original value further down on the stack without modifying it.

All Boolean input instructions have both normal and "NOT" versions.

Example:

```
NETWORK 1
STR X1
AND X2
OR X3
OUT Y1

NETWORK 2
STRN C10
ANDN C11
ORN C12
OUT C21

NETWORK 3
STR Y1
STR Y2
ANDSTR
STR Y3
ORSTR
OUT Y10
```

Edge Contact Instructions

- **ANDND** - AND negative differential.
- **ANDPD** - AND positive differential.
- **ORND** - OR negative differential.
- **ORPD** - OR positive differential.
- **STRND** - STORE negative differential.
- **STRPD** - STORE positive differential.

Edge contact instructions are a type of Boolean input instruction, and operate in a manner similar to that of the normal Boolean input instructions. The difference is that they are one-shot instructions and output a true for one scan only.

Instruction	Description	# Params	X	Y	C	T	CT	SC	Symbol
ANDND	AND negative differential	1	X	X	X	X	X	X	
ANDPD	AND positive differential	1	X	X	X	X	X	X	
ORND	OR negative differential	1	X	X	X	X	X	X	
ORPD	OR positive differential	1	X	X	X	X	X	X	
STRND	STORE negative differential	1	X	X	X	X	X	X	

STRPD	STORE positive differential	1	X	X	X	X	X	X	

Edge contact instructions can be categorised as "positive differentiate" and "negative differentiate" instructions. "Positive differentiate" instructions are true for one scan when the logic stack makes a false to true transition. "Negative differentiate" instructions are true for one scan when the logic stack makes a true to false transition.

The address used as a parameter is used for an AND, OR, or STORE the operation. It is not necessary to specify an address to store the one shot state, as the instruction stores this information automatically internally.

Example:

```
NETWORK 1
STR X1
ANDPD C1
OUT Y1

NETWORK 2
STR X2
ANDND C2
OUT Y2

NETWORK 3
STR X3
ORPD C3
OUT Y3

NETWORK 4
STR X4
ORND C4
OUT Y4

NETWORK 5
STRPD X5
OUT Y5

NETWORK 6
STRND X6
OUT Y6
```

Boolean Output Instructions

- **OUT** - Output logic stack to bit.
- **OUT** - Output logic stack to multiple bits.
- **PD** - Output logic stack one shot.
- **PD** - Output logic stack one shot to multiple bits.
- **RST** - Reset bit if logic stack true.
- **RST** - Reset multiple bits if logic stack true.
- **SET** - Set bit if logic stack true.
- **SET** - Set multiple bits if logic stack true.

Boolean output instructions output the value on the top of the logic stack to an address given as an explicit parameter. Output instructions do not modify the logic stack. This means that multiple output instructions can be used in series without the result of one instruction affecting the next.

Instruction	Description	# Params	X	Y	C	T	CT	SC	Bit Range	Symbol
OUT	Output logic stack to bit	1		X	X					
OUT	Output logic stack to multiple bits	2		X	X				X	
PD	Output logic stack one shot	1		X	X					
PD	Output logic stack one shot to multiple bits	2		X	X				X	
RST	Reset bit if logic stack true	1		X	X					

RST	Reset multiple bits if logic stack true	2		X	X				X	
SET	Set bit if logic stack true	1		X	X					
SET	Set multiple bits if logic stack true	2		X	X				X	

Boolean output instructions come in two forms: single output and bit range output. The system distinguishes between the two forms by the number of parameters given.

The PD instruction is a one-shot instruction. The output will be turned on for one scan if the logic stack input is true.

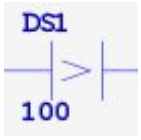
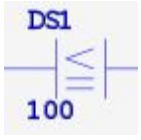
Example:

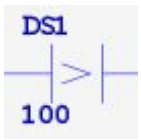
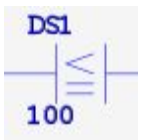
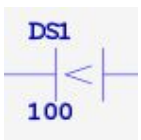
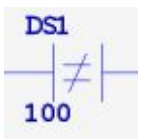
```
// Output to a single bit.
NETWORK 1
STR X1
OUT Y1
RST Y2
SET Y3

// Output to a range of bits.
NETWORK 2
STR X2
OUT C1 C9
RST C10 C19
SET C20 C21

// Output using one-shot.
NETWORK 3
STR X3
PD C30
PD C40 C60
```

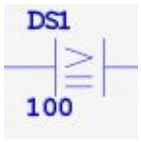
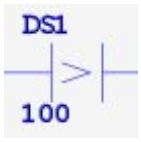
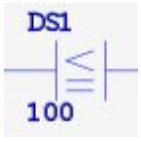
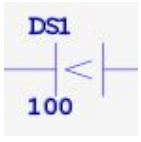
Comparison Instructions

- **ANDE** - AND if parm1 equals param2.
- **ANDGE** - AND if parm1 >= parm2.
- **ANDGT** - AND if parm1 > parm2.
- **ANDLE** - AND if parm1 <= parm2.
- **ANDLT** - AND if parm1 < parm2.
- **ANDNE** - AND if parm1 is not equal to parm2.
- **ORE** - OR if parm1 equals parm2.
- **ORGE** - OR if parm1 >= parm2.
- **ORGT** - OR if parm1 > parm2.
- **ORLE** - OR if parm1 <= parm2.
- **ORLT** - OR if parm1 < parm2.
- **ORNE** - OR if parm1 is not equal to parm2.
- **STRE** - STR if parm1 equals parm2.
- **STRGE** - STR if parm1 >= parm2.
- **STRGT** - STR if parm1 > parm2.
- **STRLE** - STR if parm1 <= parm2.
- **STRLT** - STR if parm1 < parm2.
- **STRNE** - STR if parm1 is not equal to parm2.

Comparison instructions compare two word parameters and output the Boolean result to the logic stack.

Instruction	Description	# Params	Symbol
ANDE	AND if parm1 equals param2	2	
ANDGE	AND if parm1 >= parm2	2	
ANDGT	AND if parm1 > parm2	2	
ANDLE	AND if parm1 <= parm2	2	

ANDLT	AND if parm1 < parm2	2	
ANDNE	AND if parm1 is not equal to parm2	2	
ORE	OR if parm1 equals parm2	2	
ORGE	OR if parm1 >= parm2	2	
ORGT	OR if parm1 > parm2	2	
ORLE	OR if parm1 <= parm2	2	
ORLT	OR if parm1 < parm2	2	
ORNE	OR if parm1 is not equal to parm2	2	
STRE	STR if parm1 equals parm2	2	

STRGE	STR if parm1 >= parm2	2	
STRGT	STR if parm1 > parm2	2	
STRLE	STR if parm1 <= parm2	2	
STRLT	STR if parm1 < parm2	2	
STRNE	STR if parm1 is not equal to parm2	2	

All comparison instructions take two explicit parameters and output their result to the logic stack. Each parameter is a word register or constant.

There are three categories of comparison instruction: AND, OR, and STORE. Each of these operates in a manner similar to the corresponding Boolean input instructions, with the exception that they apply the Boolean result of the comparison operation to to top of the logic stack.

For any comparison operation, both parameters must be compatible with each other. Parameters are compatible if they are both within the same compatibility group. The compatibility groups are defined as follows:

- Signed numeric - DS, DD, DF, TD, CTD, SD, KInt, KDInt, KF, XS, YS
- Unsigned numeric (hex) - DH, XD, YD. KHex
- Text - TXT, KTxtChar, KTxtStr

The type and address abbreviations are defined in the section on addresses and constants.

If one parameter is a string constant (e.g. "abc123"), and the other is a text register, the string constant will be compared to the series of text registers beginning at the register specified. The comparison will be

on a character by character basis, such that the comparison will fail at the first character that does not meet the comparison criteria.

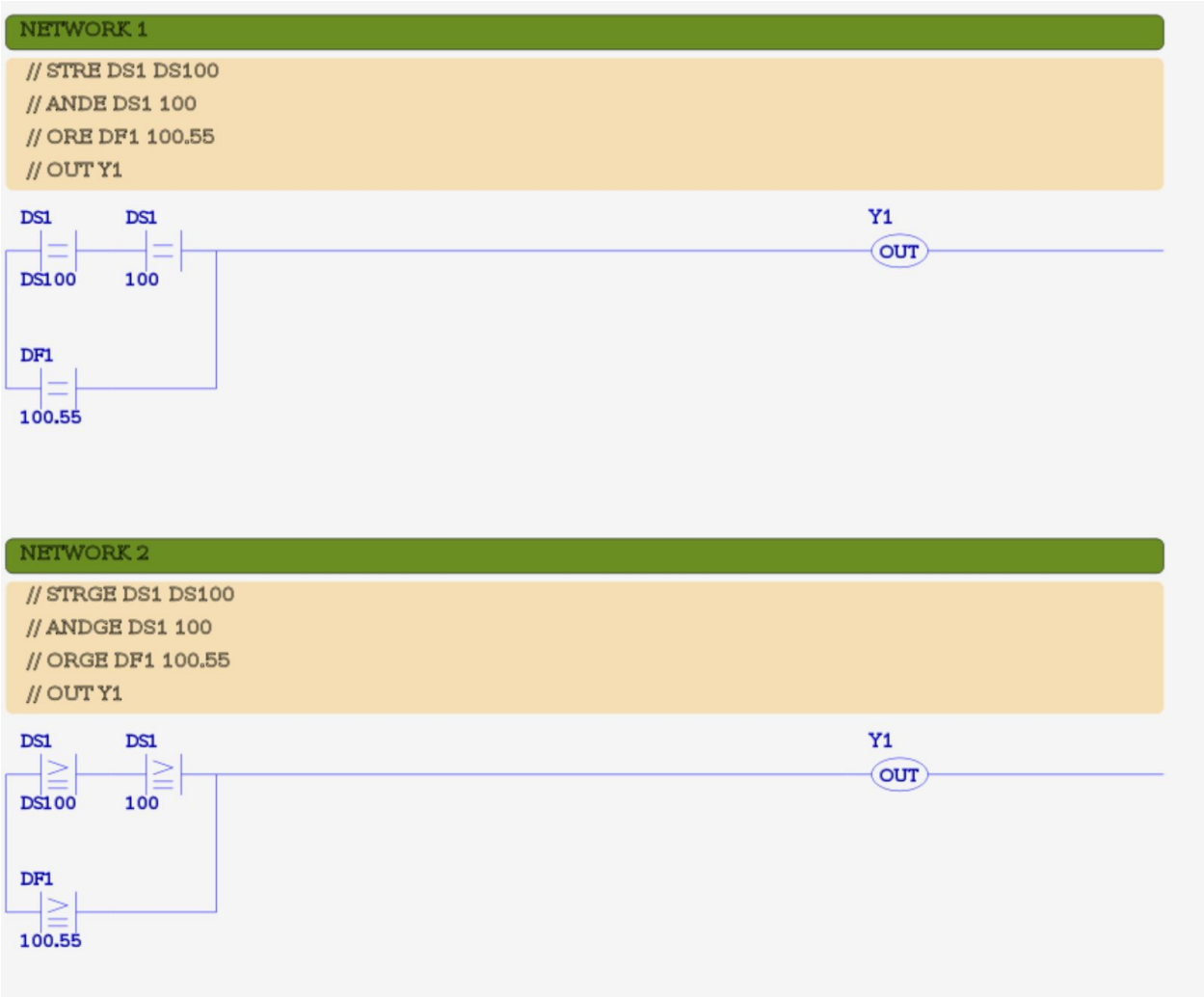
For example, if the string "hijklanop" is compared to to see if it is greater that a series of registers containing "abcdefghi", the comparison will fail on the sixth character because "a" is not greater than "f".

Example:

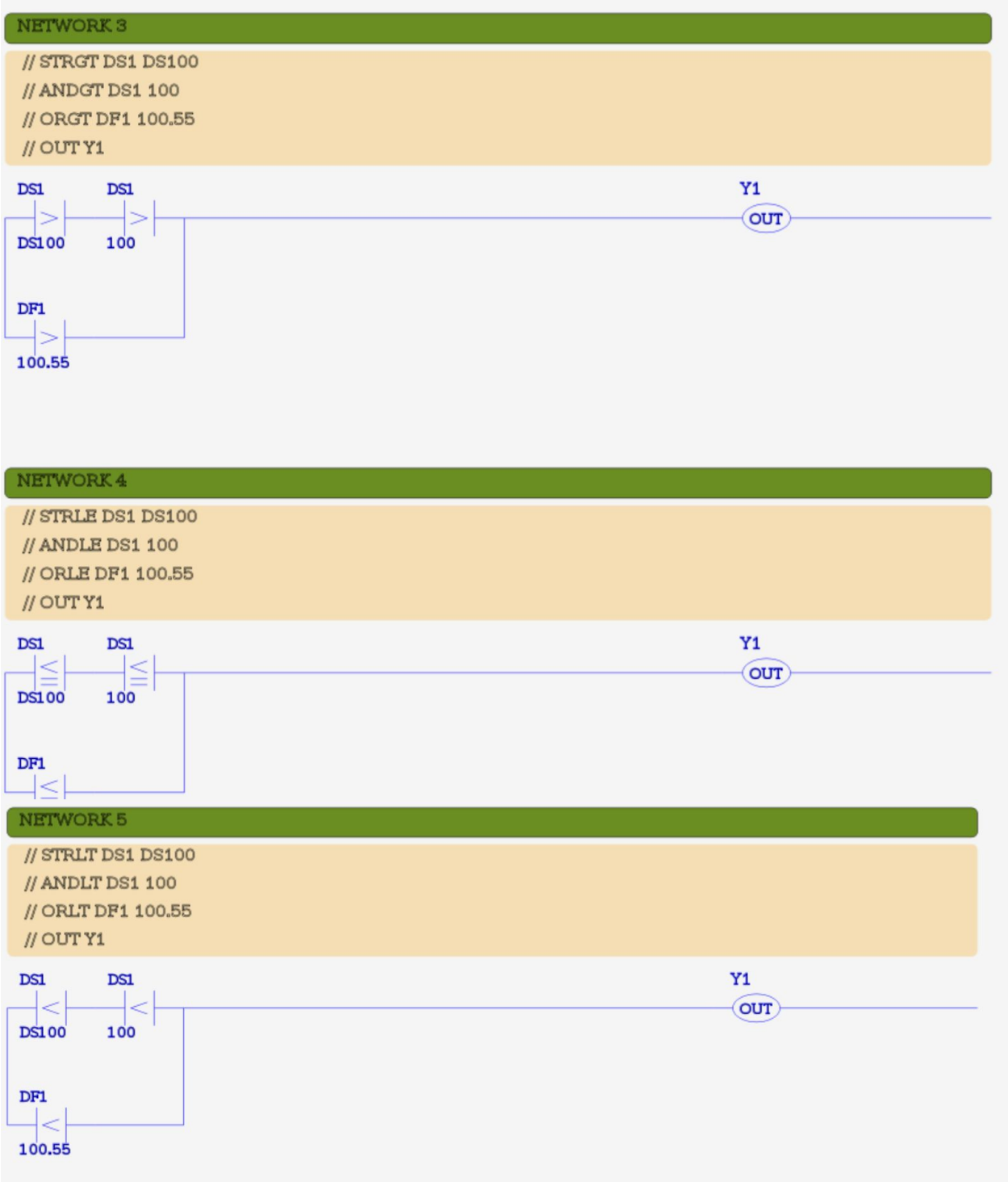
```
// Compare signed numbers.
NETWORK 1
STRE DS1 DS2
ANDGT 123 DD10
ORLE DF20 123.876
ORGE DF5 DS3
ANDLT 93 CTD3
OUT Y1

// Compare unsigned (hex) numbers.
NETWORK 2
STNE 123fh DH52
ANDGE YD4 DH2
SET C20

// Compare text.
NETWORK 3
STR X3
ANDE TXT5 TXT10
ORGE "A" TXT12
ANDE "pass" TXT50
OUT C35
```

Ladder Examples

The following shows examples in ladder format. Each example shows the IL code as comments, followed by the ladder equivalent.



cont..


NETWORK 3
// STRGT DS1 DS100
// ANDGT DS1 100
// ORGT DF1 100.55
// OUT Y1
DS1
DS100
DS1
100
Y1
OUT
DF1
100.55
NETWORK 4
// STRLE DS1 DS100
// ANDLE DS1 100
// ORLE DF1 100.55
// OUT Y1
DS1
DS100
DS1
100
Y1
OUT
DF1
NETWORK 5
// STRLT DS1 DS100
// ANDLT DS1 100
// ORLT DF1 100.55
// OUT Y1
DS1
DS100
DS1
100
Y1
OUT
DF1
100.55

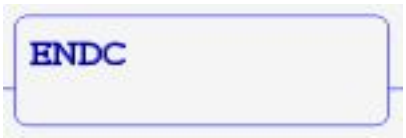
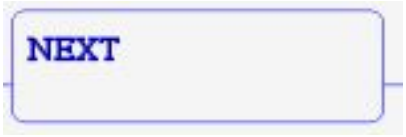
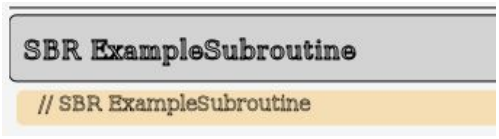
cont ..


NETWORK 6
// STRNE DS1 DS100
// ANDNE DS1 100
// ORNE DF1 100.55
// OUT Y1
DS1
DS100
DS1
100
DF1
100.55
Y1
OUT

Program Control Instructions

- **END** - Program end .
- **ENDC** - Program end conditional.
- **FOR** - For/next loop.
- **NEXT** - Next in For/next loop.
- **SBR** - Define a subroutine.
- **CALL** - Call subroutine.
- **RT** - Return from subroutine.
- **RTC** - Return from subroutine conditional.

Program control instructions control the flow of the PLC program.

Instr.	Descr.	# Param	Subr	DS	KInt	One Shot	Symbol
END	Program end	0					
ENDC	Program end conditional	0					
FOR	For/next loop	1 or 2		X	X	X	
NEXT	Next in For/next loop	0					
SBR	Define a subroutine	1	X				
CALL	Call subroutine	1	X				

RT	Return from subroutine	0					
RTC	Return from subroutine conditional	0					

Program END

END and ENDC will terminate a program scan. END terminates the scan unconditionally, while ENDC will terminate the scan if the logic stack is true.

Example:

```
// END.
        NETWORK 100
        STR SC1
        AND X5
        ENDC
        STRN SC1
        END
```

Subroutines

SBR, CALL, RT, and RTC are subroutine instructions. SBR is used to define a subroutine. CALL will call a subroutine. SBR and CALL expect a valid subroutine name as a parameter. RT will return from a subroutine unconditionally. RTC will return from a subroutine if the logic stack is true.

Example:

```
// Subroutines.
NETWORK 1
STR SC1
OUT Y1
CALL SubTest

NETWORK 2
STR C47
OUT Y19

END

// Define subroutine.
SBR SubTest
NETWORK 1
STR SC1
RST Y10
AND X21
RTC
SET C47
RT
```

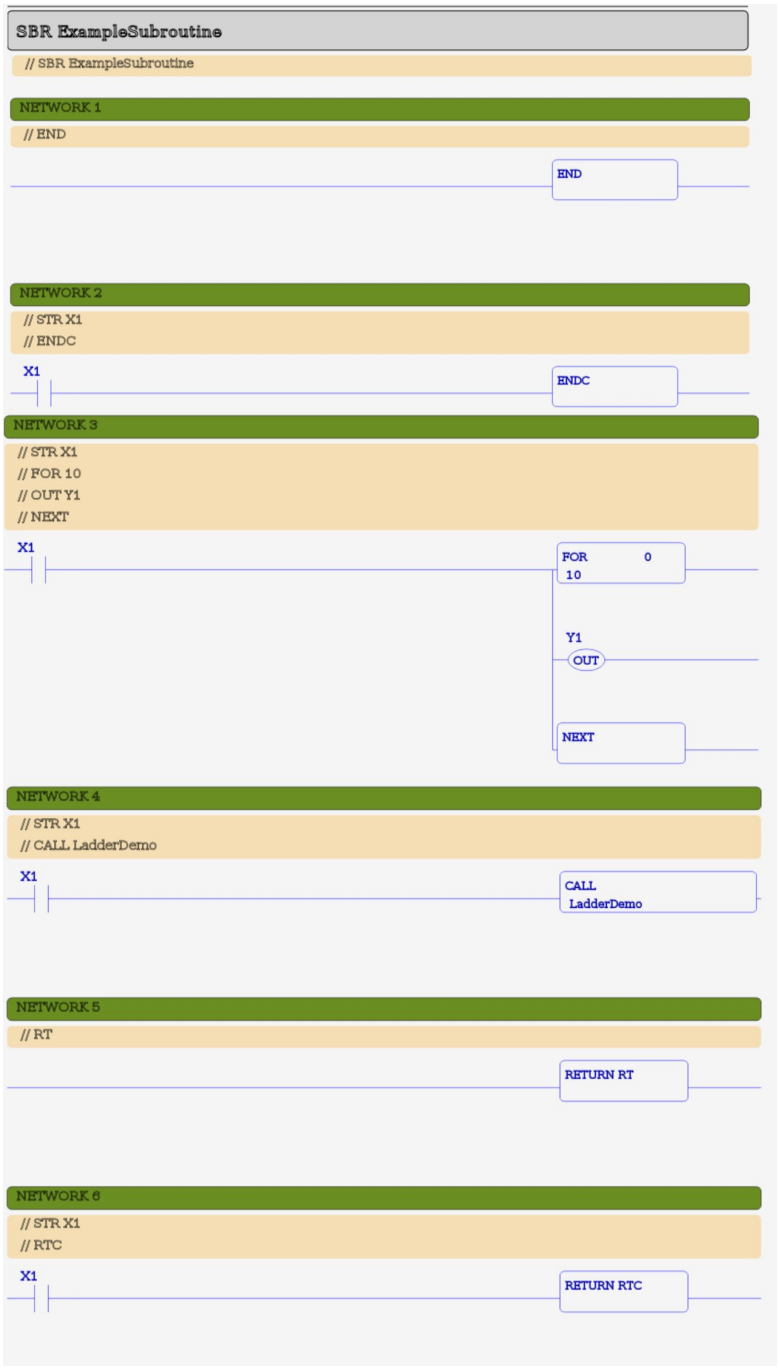
.¥¥z¥¦

Using RT or RTC in the main program will cause a run-time error. Using NEXT without FOR may cause a run-time error. Using FOR without NEXT may cause unpredictable runtime operation.

Subroutines may call other subroutines, and they may even call themselves (recursion is permitted). However, if the nesting level reaches an excessive level, a run time error will occur and the program will exit. The maximum nesting level is undefined, but is typically about 1000.

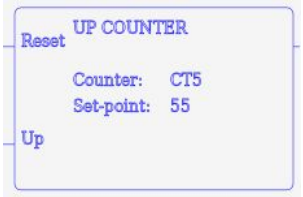
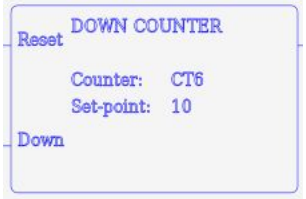
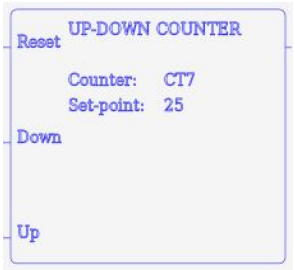
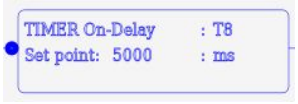
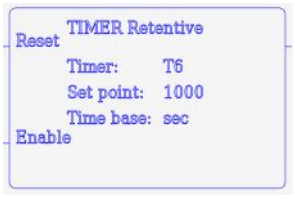
=Yddf¥.°Yw¢vf¦

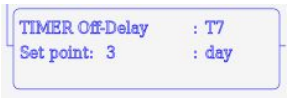
The following shows examples in ladder format. Each example shows the IL code as comments, followed by the ladder equivalent.



Counter and Timer Instructions

- **CNTU** - Count up.
- **CNTD** - Count down.
- **UDC** - Up/down counter.
- **TMR** - On delay timer.
- **TMRA** - On delay accumulating timer.
- **TMROFF** - Off delay timer.

Instr.	Descr.	# Param	T	CT	DS	DD	KInt	KDInt	Time Base	Symbol
CNTU	Count up	2		X	X	X	X	X		
CNTD	Count down	2		X	X	X	X	X		
UDC	Up / down counter	2		X	X	X	X	X		
TMR	On delay timer	3	X		X		X		X	
TMRA	On delay accum. timer	3	X		X		X		X	

TMROFF	Off delay timer	3	X		X		X		X	

Counters

*z©x¨f¥ nx¦¨¥©b¨nzx¦ bzwf nx ¨m¥f f ¨±¢f¦Í

- CNTU - Up counter. This counts up by one each time the logic stack transitions from off to on. If the logic stack value (top - 1) turns on, the counter is reset.
- CNTD - Down counter. This counts down by one each time the logic stack transitions from off to on. If the logic stack value (top - 1) turns on, the counter is reset.
- UDC - Up/down counter. This counts up by one each time the top of the logic stack transitions from off to on, and down by one each time the logic stack value (top - 1) transitions from off to on. When the logic stack value (top - 2) turns on, the counter is reset

All counter instructions take two parameters. The first is a counter number, and the second is the counter preset. The counter preset may be either a constant or a register address. When the present value of the counter is equal to the preset, the counter status bit turns on. The address of the counter status bit is the same as the counter number. The present value of the counter is stored in the counter data register (CTD) of the same number as the counter number. E.g. for counter CT5, the counter status bit is also CT5, and the present count is in CTD5.

Example:

```
// Up counter.

        NETWORK 1
        STR C1
        STR C2
        CNTU CT5 155793
        STR CT5
        OUT Y1

        // Down counter.
        NETWORK 2
        STR C11
        STR C12
        CNTD CT15 DD11
        STR CT15
        OUT Y11

        // Up/down counter.
        NETWORK 3
        STR C21
        STR C22
        STR C23
        UDC CT25 DS5
        STR CT25
        OUT Y21

        // Examine counter present values.
        NETWORK 7
        STRE CTD15 157
        OUT Y112
```

Timers

Timers come in three types:

- TMR - On delay timer. The timer begins timing up when the top of the logic stack is true, and resets when the top of the logic stack is false. When the present value reaches the preset value, the timer status bit turns on.
- TMRA - On delay accumulating timer. The timer begins timing up when the top of the logic stack is true, and stops when the top of the logic stack is false. When the top of the logic stack becomes true, it continues timing from the point at which it stopped. When the logic stack value (top - 1) turns on, the timer is reset. When the present value reaches the preset value, and the top of the logic stack is true, the timer status bit turns on.
- TMROFF - Off delay timer. The timer begins timing when the top of the logic stack transitions from on to off. When the timer begins timing, the status bit turns on immediately. The status bit turns off when the timer reaches the preset.

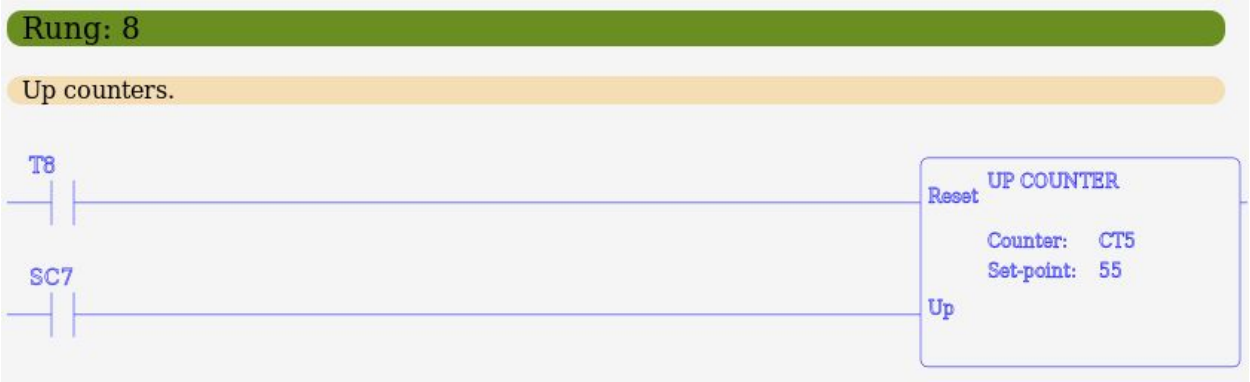
All timer instructions take three parameters. The first is the timer address. The second is the timer preset. The timer preset may be either a constant or a register address. The third parameter is the time base. This may be either "ms" (milliseconds), "sec" (seconds), "min" (minutes), "hour" (hours), or "day" (days).

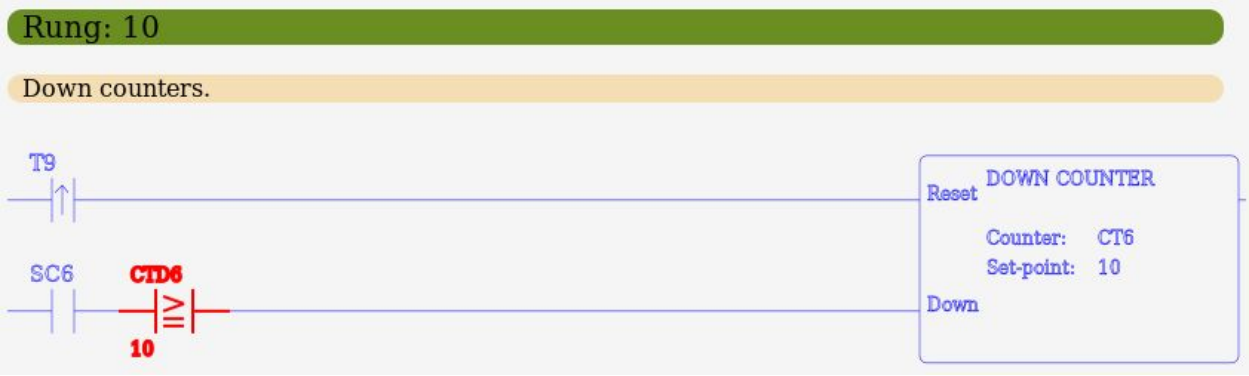
Example:

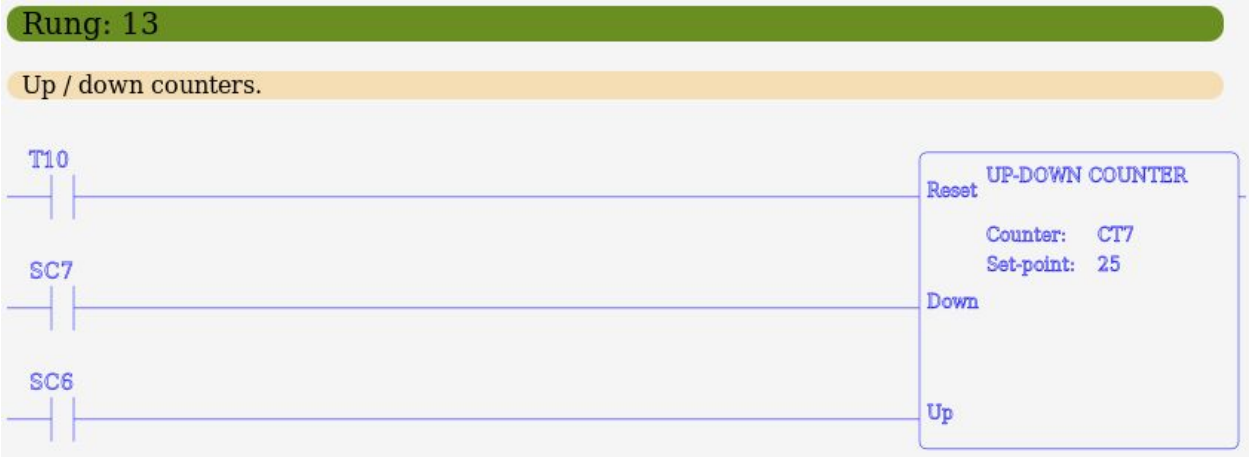
```
// On delay timer.
        NETWORK 4
        STR X1
        TMR T5 25000 ms
        STR T5
        OUT Y51

        // Accumulating on delay timer.
        NETWORK 5
        STR X11
        STR X12
        TMRA T6 DS9 sec
        STR T6
        OUT Y52

        // Off delay timer.
        NETWORK 6
        STR X62
        TMROFF T192 DS412 min
        STR T192
        OUT Y57

        // Examine timer present values.
        NETWORK 7
        STRE TD6 97
        OUT Y120
```

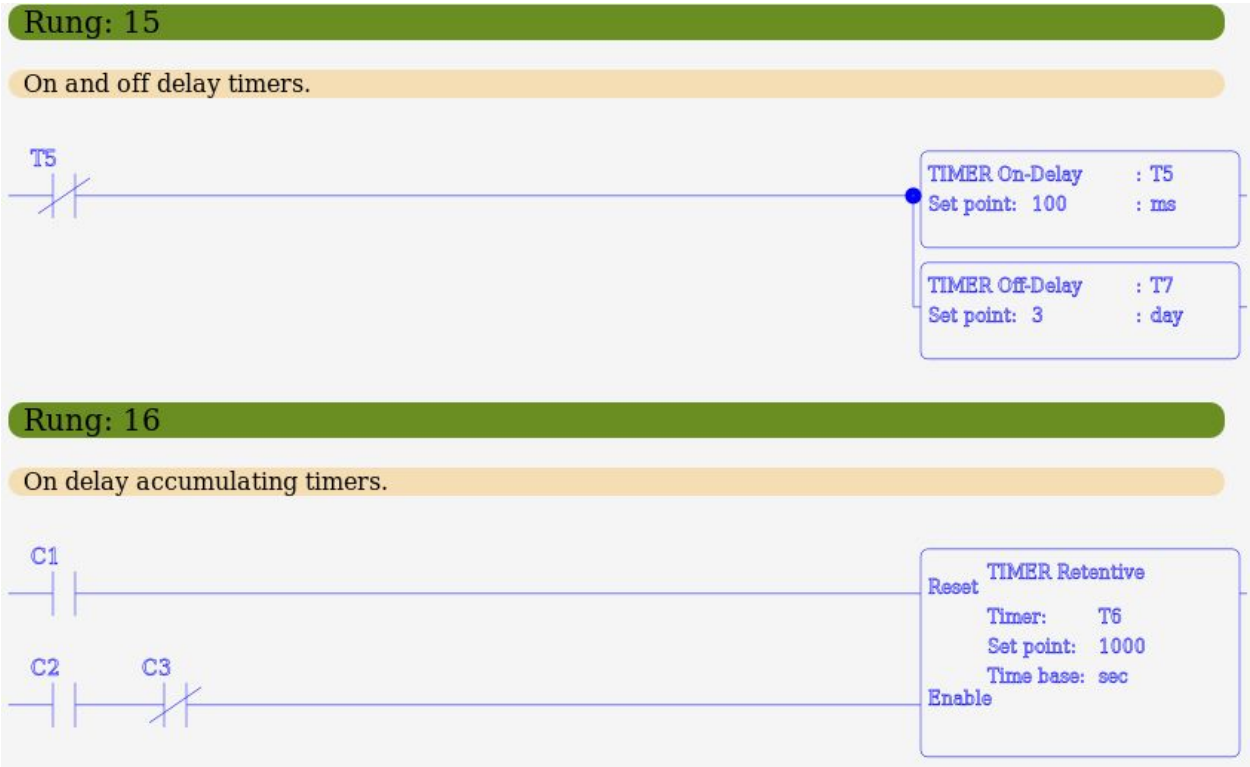
Ladder Examples

The following shows examples in ladder format.



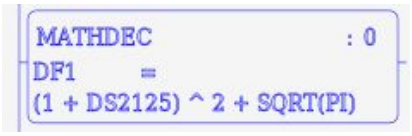
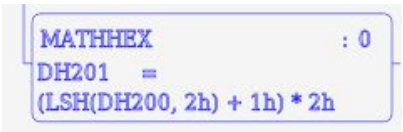





Rung: 15
On and off delay timers.
T5
TIMER On-Delay : T5
Set point: 100 : ms
TIMER Off-Delay : T7
Set point: 3 : day
Rung: 16
On delay accumulating timers.
C1
C2
C3
TIMER Retentive
Reset
Timer: T6
Set point: 1000
Time base: sec
Enable

Math Instructions

Math instructions perform mathematical operations on registers and constants, and output the results to registers.

Instr.	Desc.	# Param	One Shot	SC40	SC43	SC46	Symbol
MATHDEC	Decimal math	3	X	X	X	X	
MATHHEX	Hexa-decimal math	3	X	X	X	X	
SUM	Sum a range of registers	3 or 4	X		X		

There are only three math instructions. However, they accept complete equations as input parameters allowing them to perform many different operations.

The decimal and hexadecimal math instructions are similar except for the functions and operators they offer, and for the type of registers they operate on. The general operation of both is described here with the details of the functions and operators listed separately below.

The MATHDEC and MATHHEX instruction expects the following parameters:

- Destination - The destination register. Register types may be DS, DD, DF, DH. The destination register must be compatible with the source registers. DH registers may not be mixed with the other register types.
- A mandatory one shot parameter of 0 or 1. If the parameter is set to "1", the one-shot option is enabled and the instruction executes only when the logic stack transitions from false to true. If the parameter is set to "0", the one-shot option is disabled and the instruction executes whenever the logic stack is true. The one shot parameter must always be specified for these instructions as the variable number of equation elements would otherwise make it impossible to distinguish from the equations.
- Math equation - All elements following are treated as part of the math equation and must form a legal mathematical equation.

MATHDEC ***destination one-shot equationparameters***

MATHHEX ***destination one-shot equationparameters***

The error flags are set under the following conditions:

- SC40 - A division by zero was attempted.
- SC43 - The data could not be converted to the correct type, or the data value is out of range for the destination register.
- SC46 - An unspecified math error has occurred. This includes all math errors not covered by the other math error flags.

Example:

```
// Decimal math. Will resolve to 10.77245

NETWORK 1

STR X1

COPY 2 DS1

MATHDEC DF1 0 (1 + DS1) ^ 2 + SQRT(PI)


// Hexadecimal math. Will resolve to 22h (in hexadecimal)

NETWORK 2

STR X2

COPY 4 DH2

MATHHEX DH1 0 (LSH(DH2, 2h) + 1h) * 2h
```

Decimal Math Operations

Decimal math operations are conducted by functions and operators. The input values may be DS, DD, or DF registers, or decimal constants. The destination may be a DS, DD, or DF register.

Angles for transcendental functions (SIN, COS, etc.) are in radians.

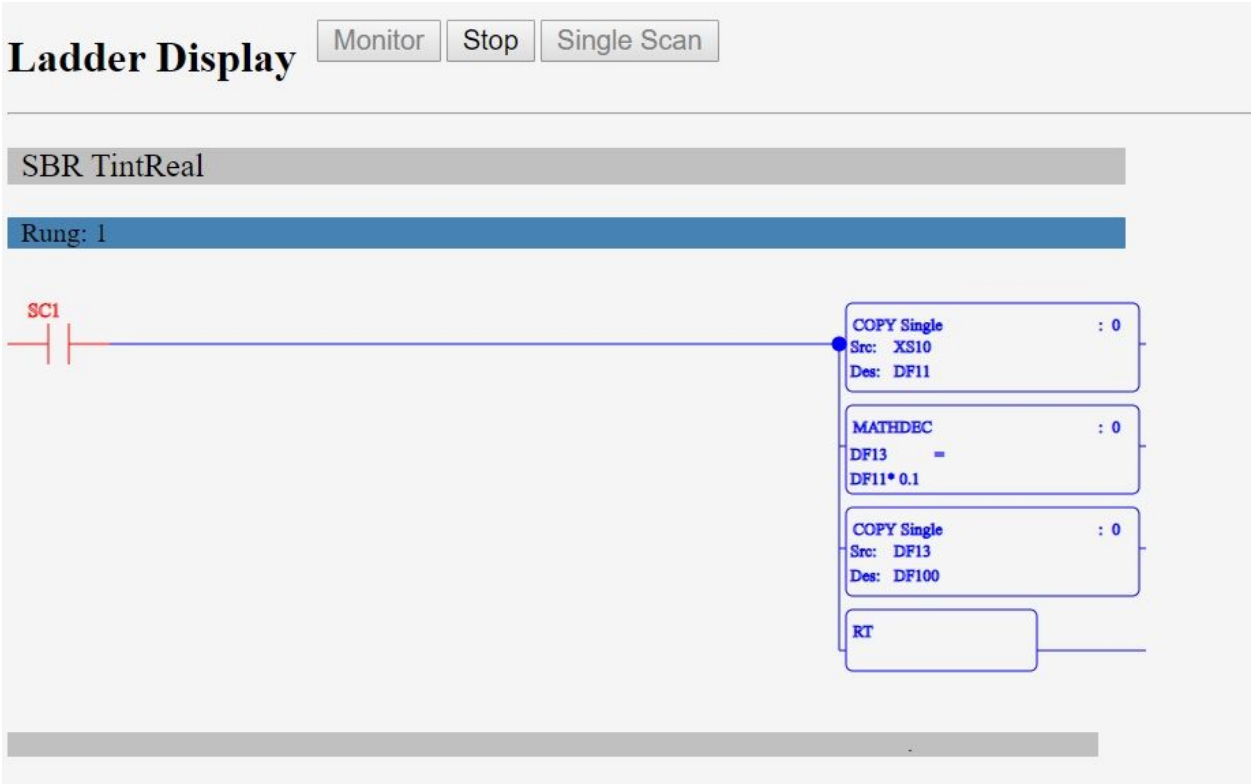
Using a function that operates in floating point (transcendental, logarithmic, square root), or using any floating point number (including the PI constant) will cause the entire equation to be conducted in floating point. The result will then be converted to a type that is compatible with the destination register. If a floating point number is copied to an integer register, the decimal part of the result is truncated, not rounded.

Operation	Description	Example
SQRT	Square root	SQRT(DD3)
+	Add	DS1 + 7
-	Subtract	DS3 - DS4
*	Multiply	DS5 * 10
/	Divide	DF7 / DS1
MOD	Modulus (remainder)	DS22 MOD DS21
^	Exponentiate	DF52 ^ 3
PI	The constant PI	PI * DF102
()	Parentheses (group operations)	(DS2 + 71) * DF9

Example - TintReal Subroutine - Convert raw Tint value to Real Temperature Value (Deg. C)

```
NETWORK 1

STR SC1

COPY XS10 DF11

MATHDEC DF13 0 DF11* 0.1

COPY DF13 DF100

RT
```

Ladder Example



Sum

The SUM function is provided as a separate instruction instead of being called from within the other math instructions.

The Sum instruction expects the following parameters:

- Source Start - The start of the register address range. Register types may be DS, DD, DF, DH.
- Source End - The end of the register address range. This must be a higher address of the same type as the source start.
- Destination - The destination register. Register types may be DS, DD, DF, DH. The destination register must be compatible with the source registers.
- An optional one shot parameter of 0 or 1. If the parameter is set to "1", the one-shot option is enabled and the instruction executes only when the logic stack transitions from false to true. If the parameter is set to "0", the one-shot option is disabled and the instruction executes whenever the logic stack is true. If the parameter is missing, it has the same effect as setting it to "0".

SUM ***sourcestart sourceend destination (one-shot)***

The error flags are set under the following conditions:

- SC43 - The data could not be converted to the correct type, or the data value is out of range for the destination register.

Example:

```
// Sum a series of registers.

NETWORK 1

STR X1

SUM DS10 DS19 DF20


// Sum with a one shot.

NETWORK 2

STR X2

SUM DH10 DH19 DH20 1


end
```