EcoStruxure™

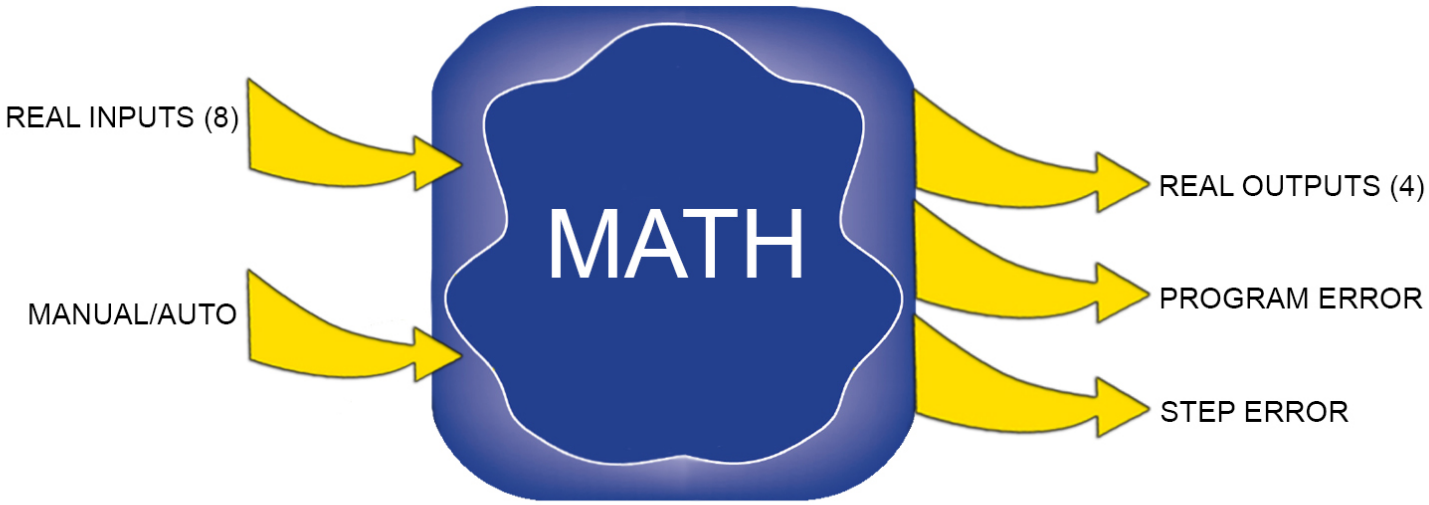
Foxboro™ DCS

Mathematic (MATH) Block

PSS 41S-3MATH

Product Specification

May 2019



Schneider Electric™

www.schneider-electric.com

Legal Information

Overview

The MATH block is a multiple input, 20-step, floating point, programmable calculator. It provides real-time computational capability for the modeling of specialized algorithms, signal characterization, and alteration of control waveforms to augment the operation of standard blocks.

The MATH block provides arithmetic computational capability to implement specialized control functions that cannot be implemented with the standard control blocks in time-critical applications.

All input connections, constant data values, and programming steps are entered via the block configuration process.

Every program step contains an *opcode*, which identifies the operation to be performed, and up to two command line arguments. The command line arguments consist of the actual operands for the step, the location of the operands, a specification of details that further refine the opcode, or some combination of these factors.

Standard Features

- 8 real inputs and 4 real outputs
- Auto/Manual control of the real outputs, which can be initiated by a host process or another block
- 5 floating point memory data storage registers that are preserved between execution cycles
- Stack of 24 floating point values for storage of intermediate computational results, which provides chaining ability for up to 24 calculations
- Up to 20 programming steps of up to 16 characters each
- Initialization of all memory registers
- Dual operand capability for several mathematical instructions
- Conditional execution of mathematical calculations, depending on mathematical conditions detected under program control
- Algorithm ability to read the status bits (for example, Bad, Out-of-Service, Error) of input/output parameters and directly control the status bits of output parameters
- Forward branching of program control
- Propagation of the cascade acknowledgment from an upstream block to a downstream block
- Syntax check of all programming steps following block installation and reconfiguration
- Input and output parameter error detection and control
- Detection of program runtime errors

Instructions

Arithmetic	
ABS	Absolute value
ACOS	Arc cosine
ADD	Add
ALN	Natural antilogarithm
ALOG	Common antilogarithm
ASIN	Arc sine
ATAN	Arc tangent
AVE	Average
CHS	Change operand sign
COS	Cosine
DEC	Decrement operand
DIV	Divide
EXP	Exponent
INC	Increment operand
LN	Natural logarithm
LOG	Common logarithm
MAX	Select maximum
MIN	Select minimum
MEDN	Select median
MUL	Multiply
SIN	Sine
SQR	Square
SQRT	Square root
SUB	Subtract
TAN	Tangent
Input/Output Reference	
CBD	Clear output bad status bit
IN	Input value
INR	Input indexed real input value
INS	Input status
OUT	Write accumulator value to output
RBD	Read bad and out-of-service status bits
RCL	Read and clear operand
REL	Release output
RQE	Read quality status and error bit

RQL	Read quality status
SBD	Set output bad status bit
SEC	Secure output
Cascade	
PRO	Propagate downstream
Memory and Stack Reference	
CLA	Clear all memory registers
CLM	Clear designated memory register
CST	Clear stack
DUP	Duplicate operands
LACI	Load accumulator indirect
POP	Pop the last value off the stack
STM	Store accumulator value in memory register
STMI	Store memory indirect
SWP	Swap operands
Program Control	
BII	Branch if block is initializing
BIN	Branch if accumulator is negative
BIP	Branch if accumulator is positive
BIZ	Branch if accumulator is zero
END	End of program
EXIT	Terminate program execution
GTI	Go to step number in accumulator or operand
GTO	Go to step number in operand
NOP	No operation; branch to next step
Clear/Set	
CLR	Clear Boolean
SET	Set Boolean
SSI	Set Boolean and skip if block is initializing
SSN	Set Boolean and skip if accumulator is negative
SSP	Set Boolean and skip if accumulator is positive
SSZ	Set Boolean and skip if accumulator is zero

Program Examples

Table 1 shows a program example that includes a typical instruction (ADD) which uses two inputs (dyadic).

Figure 1 shows the stack operation for each program instruction in Table 1.

Table 2 shows a program example that includes a typical instruction (AVE) which uses more than two inputs (polyadic).

Figure 2 shows the stack operation for each program instruction in Table 2.

Table 1. Program Example with Typical Dyadic Instructions

STEP01	ADD RI01 RI02	Adds RI01 to RI02 and pushes the result (Sum1) onto stack
STEP02	ADD RI03 RI04	Adds RI03 to RI04 and pushes the result (Sum2) onto stack
STEP03	ADD	Pops Sum2 and Sum1 from stack, performs addition, and pushes the result (Sum3) onto stack
STEP04	IN 4	Pushes constant "4" onto stack
STEP05	DIV	Pops "4" and Sum3 from stack, divides them, and pushes Quotient onto stack

Figure 1. Examples of Stack Operation for Dyadic Instructions

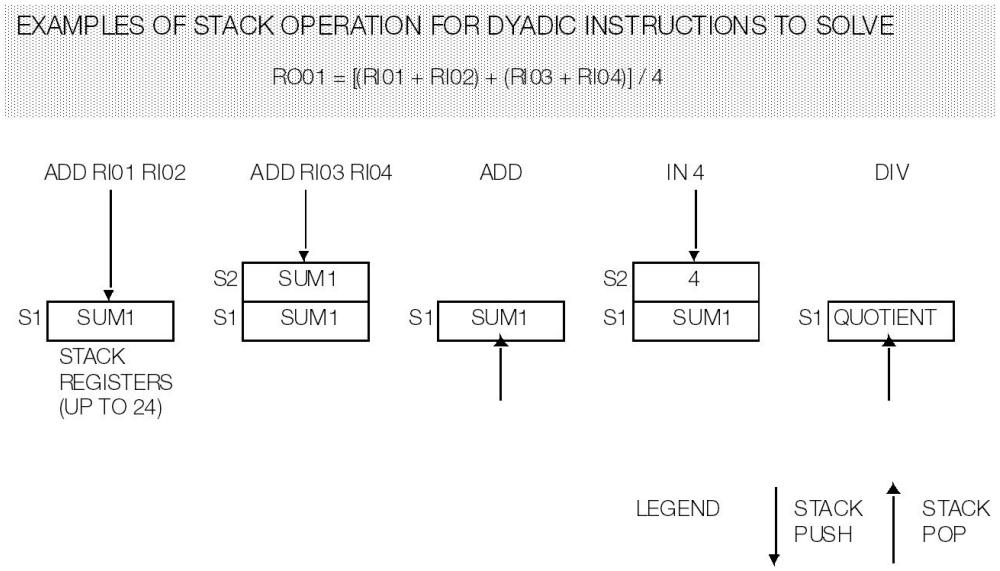


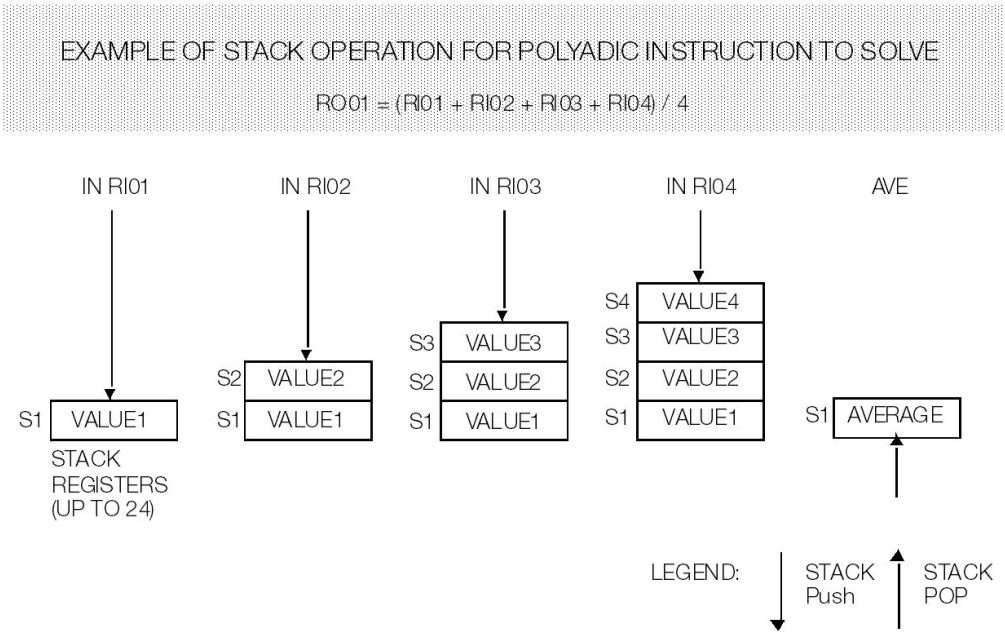
Table 2. Program Example with Typical Polyadic Instruction (AVE)

STEP01	CST	Clears stack
STEP02	IN RI01	Pushes RI01 value onto stack
STEP03	IN RI02	Pushes RI02 value onto stack
STEP04	IN RI03	Pushes RI03 value onto stack

Table 2. Program Example with Typical Polyadic Instruction (AVE) (Continued)

STEP05	IN RI04	Pushes RI04 value onto stack
STEP06	AVE	Pops Value4 to Value1 from stack, averages them, and pushes Average onto stack

Figure 2. Examples of Stack Operation for Polyadic Instruction



WARNING: This product can expose you to chemicals including lead and lead compounds, which are known to the State of California to cause cancer and birth defects or other reproductive harm. For more information, go to www.p65warnings.ca.gov/.

Schneider Electric Systems USA, Inc.
38 Neponset Avenue
Foxborough, Massachusetts 02035–2037
United States of America

Global Customer Support: https://pasupport.schneider-electric.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.

PSS 41S-3MATH, Rev A