Proficy* Logic Developer - PLC

Ladder Diagram (LD) Language

Proficy Machine Edition 8.0
2013

Notice

GE Intelligent Platforms, Inc. reserves the right to make improvements to the products described in this publication at any time and without notice.

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address: doc@ge.com

Disclaimer of Warranties and Liability

The information contained in this manual is believed to be accurate and reliable. However, GE Intelligent Platforms, Inc. assumes no responsibilities for any errors, omissions or inaccuracies whatsoever. Without limiting the foregoing, GE Intelligent Platforms, Inc. disclaims any and all warranties, expressed or implied, including the warranty of merchantability and fitness for a particular purpose, with respect to the information contained in this manual and the equipment or software described herein. The entire risk as to the quality and performance of such information, equipment and software, is upon the buyer or user. GE Intelligent Platforms, Inc. shall not be liable for any damages, including special or consequential damages, arising out of the use of such information, equipment and software, even if GE Intelligent Platforms, Inc. has been advised in advance of the possibility of such damages. The use of the information contained in the manual and the software described herein is subject to GE Intelligent Platforms, Inc. standard license agreement, which must be executed by the buyer or user before the use of such information, equipment or software.

Table Of Contents

LD Instructions

LD logic instructions are what LD blocks are composed of. To create executable units of logic, you insert the instructions and their operands into an LD block. The editor automatically creates new rungs as required. Each instruction performs an operation on variables defined for the target the LD logic is associated with.

Note: All available instructions are contained in the **LD Instructions** drawer of the Toolchest. You can drag these instructions to any rung in your LD logic.

LD logic instructions are grouped according to the type of operation performed. The instruction groups are:

- Advanced Math
- Bit Operations
- Coils
- Communication
- Contacts
- Control
- Conversion
- Counters
- Data Move
- Data Table
- Math
- PACMotion
- Program Flow
- Relational
- Timers
- VersaMax Micro Motion

PACMotion instructions are described at length in both GFK-2448 and online help.

Supported CPUs for Each LD Instruction

A	CPU Families, Models, Minimum Firmware Version (MFV)			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
ABS_DINT ABS_INT	None	All	MFV: 3.00	None
ABS_LREAL	None	MFV: 5.50	None	None
ABS_REAL	None	All	MFV: 3.00 of floating-point CPUs	None
ACOS	All	All	MFV: 3.00 of floating-point CPUs	Floating-point CPUs
ACOS_LREAL	None	MFV: 5.50	None	None
ACOS_REAL	None	All	None	None
ADD_DINT	All	All	All	All. In CPU341 and earlier, DINT constants are limited to values between -32,768 and +32,767.
ADD_INT	All	All	All	All
ADD_LREAL	None	MFV: 5.50	None	None
ADD_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
ADD_UINT	None	All	All	None
AND_DWORD	None	All	All	None
AND_WORD	All	All	All	All
ARRAY_MOVE_BOOL ARRAY_MOVE_BYTE	All	All	MFV: 4.00	All
ARRAY_MOVE_DINT	All	All	MFV: 4.00	All
ARRAY_MOVE_DWORD	None	All	MFV: 4.00	None
ARRAY_MOVE_INT	All	All	MFV: 4.00	All
ARRAY_MOVE_UINT	None	All	MFV: 4.00	None
ARRAY_MOVE_WORD	All	All	MFV: 4.00	All

ARRAY_RANGE_DINT ARRAY_RANGE_DWORD ARRAY_RANGE_INT ARRAY_RANGE_UINT ARRAY_RANGE_WORD	None	All	MFV: 5.00	None
ARRAY_SIZE	None	All	None	None
ARRAY_SIZE_DIM1	None	All	None	None
ARRAY_SIZE_DIM2	None	All	None	None
ASIN	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
ASIN_LREAL	None	MFV: 5.50	None	None
ASIN_REAL	None	All	None	None
ATAN	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
ATAN_LREAL	None	MFV: 5.50	None	None
ATAN_REAL	None	All	None	None

B	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
BCD4_TO_INT	All	All	MFV: 3.00	All
BCD4_TO_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
BCD4_TO_UINT	None	All	MFV: 3.00	None
BCD8_TO_DINT	None	All	MFV: 3.00	None
BCD8_TO_REAL	None	All	Floating-point CPUs, MFV: 3.00	None
BIT_CLEAR_DWORD	None	All	All	None
BIT_CLEAR_WORD	All	All	All	All
BIT_POS_DWORD	None	All	All	None
BIT_POS_WORD	All	All	All	All
BIT_SEQ	All	All	All	All
BIT_SET_DWORD	None	All	All	None
BIT_SET_WORD	All	All	All	All
BIT_TEST_DWORD	None	All	All	None

BIT_TEST_WORD	All	All	All	All
BLK_CLR_WORD	All	All	All	All
BLKMOV_DINT BLKMOV_DWORD	None	All	All	None
BLKMOV_INT	All	All	All	All
BLKMOV_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
BLKMOV_UINT	None	All	All	None
BLKMOV_WORD	All	All	All	All
BUS_RD_BYTE BUS_RD_DWORD BUS_RD_WORD	None	All	None	None
BUS_RMW_BYTE BUS_RMW_DWORD BUS_RMW_WORD	None	All	None	None
BUS_TS_BYTE BUS_TS_WORD	None	All	None	None
BUS_WRT_BYTE BUS_WRT_DWORD BUS_WRT_WORD	None	All	None	None

C-D	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
CALL	All	All	All	All, except Micro Controllers
CMP_DINT CMP_INT	None	All	All	None
CMP_LREAL	None	MFV: 5.50	None	None
CMP_REAL	None	All	Floating-point CPUs, MFV: 3.00	None
CMP_UINT	None	All	All	None
COIL	All	All	All	All
COMM_REQ	All	All	All	All
COMMENT	All	All	All	All
CONTCOIL	All	All	MFV: 4.00	All

CONTCON	All	All	MFV: 4.00	All
COS	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
COS_LREAL	None	MFV: 5.50	None	None
COS_REAL	None	All	None	None
DATA_INIT_ASCII DATA_INIT_COMM DATA_INIT_DINT	None	All	MFV: 4.00	None
DATA_INIT_DLAN	None	All, but the IC697BEM763 module is supported for MFV 1.50	MFV: 4.00	None
DATA_INIT_DWORD DATA_INIT_INT	None	All	MFV: 4.00	None
DATA_INIT_LREAL	None	MFV: 5.50	None	None
DATA_INIT_REAL DATA_INIT_UINT DATA_INIT_WORD	None	All	MFV: 4.00	None
DEG_TO_RAD	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
DEG_TO_RAD_LREAL	None	MFV: 5.50	None	None
DEG_TO_RAD_REAL	None	All	None	None
DINT_TO_BCD8 DINT_TO_INT	None	All	MFV: 3.00	None
DINT_TO_REAL	All	All	MFV: 3.00 floating-point CPUs	Floating-point CPUs
DINT_TO_UINT	None	All	MFV: 3.00	None
DIV_DINT	All	All	All	All. In CPU341 and earlier, DINT constants are limited to values between -32,768 and +32,767.
DIV_INT	All	All	All	All

DIV_LREAL	None	MFV: 5.50	None	None
DIV_MIXED	None	All	MFV: 3.00	None
DIV_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
DIV_UINT	None	All	All	None
DNCTR	All	All	All	All
DO_IO	All	All	All, but is supported for I/O modules only	All, but CPU 330 and earlier had limited functionality
DRUM	All	MFV: 2.00	None	MFV: 10.00 of models 350, 352, 360, 363, and 364

E-F	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
END	All	None	None	CPU350 and later. Before MFV 7.00, no fault message was reported when incorrectly using END.
END_FOR	None	All	MFV: 4.00	None
ENDMCR	None	None	None	CPU341 and earlier
ENDMCRN	All	All	All	All
EQ_DATA	None	MFV: 5.60	None	None
EQ_DINT EQ_INT	All	All	All	All
EQ_LREAL	None	MFV: 5.50	None	None
EQ_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
EQ_UINT	None	All	All	None
EXIT_FOR	None	All	MFV: 4.00	None
EXP	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs

EXP_LREAL	None	MFV: 5.50	None	None
EXP_REAL	None	All	None	None
EXPT	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
EXPT_LREAL	None	MFV: 5.50	None	None
EXPT_REAL	None	All	None	None
F_TRIG	None	All	MFV: 5.00	MFV: 5.00
FAULT	None	All	All	None
FIFO_RD_DINT FIFO_RD_DWORD FIFO_RD_INT FIFO_RD_UINT FIFO_RD_WORD	None	All	All	None
FIFO_WRT_DINT FIFO_WRT_DWORD FIFO_WRT_INT FIFO_WRT_UINT FIFO_WRT_WORD	None	All	All	None
FOR_LOOP	None	All	MFV: 4.00	None

G, H, I	CPU Families, Models, Minimum Firmware Version (MFV)			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
GE_DINT GE_INT	All	All	All	All
GE_LREAL	None	MFV: 5.50	None	None
GE_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
GE_UINT	None	All	All	None
GT_DINT GT_INT	All	All	All	All
GT_LREAL	None	MFV: 5.50	None	None
GT_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
GT_UINT	None	All	All	None
H_WIRE	All	All	All	All
HIALR	None	All	All	None
INT_TO_BCD4	All	All	MFV: 3.00	All
INT_TO_DINT	None	All	MFV: 3.00	None
INT_TO_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
INT_TO_UINT	None	All	MFV: 3.00	None

J, K, L	CPU Families, Models, Minimum Firmware Version (MFV)			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
JUMP	None	None	None	CPU341 and earlier
JUMPN	All	All	All	All
LABEL	None	None	None	CPU341 and earlier
LABELN	All	All	All	All
LE_DINT LE_INT	All	All	All	All
LE_LREAL	None	MFV: 5.50	None	None
LE_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
LE_UINT	None	All	All	None

LIFO_RD_DINT LIFO_RD_DWORD LIFO_RD_INT LIFO_RD_UINT LIFO_RD_WORD	None	All	All	None
LIFO_WRT_DINT LIFO_WRT_DWORD LIFO_WRT_INT LIFO_WRT_UINT LIFO_WRT_WORD	None	All	All	None
LN	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
LN_LREAL	None	MFV: 5.50	None	None
LN_REAL	None	All	None	None
LOALR	None	All	All	None
LOG	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
LOG_LREAL	None	MFV: 5.50	None	None
LOG_REAL	None	All	None	None
LREAL_TO_REAL	None	MFV: 5.50	None	None
LT_DINT LT_INT	All	All	All	All
LT_LREAL	None	MFV: 5.50	None	None
LT_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
LT_UINT	None	All	All	None

M	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
MASK_COMP_DWORD MASK_COMP_WORD	All	All	All	MFV: 4.40
MASK_IO_INTR	None	MFV: 3.50	None	None
MCR	None	None	None	CPU341 and earlier
MCRN	All	All	All	All
MOD_DINT	All	All	All	All. In CPU341 and earlier, DINT constants are limited

				to values between -32,768 and +32,767.
MOD_INT	All	All	All	All
MOD_UINT	None	All	All	None
MOVE_BOOL	All	All	MFV: 3.00	MFV: 3.00. Overlapping move instruction in 350 only.
MOVE_DATA	None	MFV: 5.60	None	None
MOVE_DATA_EX	None	MFV: 6.00	None	None
MOVE_DINT MOVE_DWORD	None	All	All	None
MOVE_FROM_FLAT	None	MFV: 6.00	None	None
MOVE_INT	All	All	All	All. Overlapping move instruction in 350 only.
MOVE_LREAL	None	MFV: 5.50	None	None
MOVE_REAL	All	All	Floating-point CPUs	Floating-point CPUs
MOVE_TO_FLAT	None	MFV: 6.00	None	None
MOVE_UINT	None	All	All	None
MOVE_WORD	All	All	All	All. Overlapping move instruction in 350 only.
MUL_DINT	All	All	All	All. In CPU341 and earlier, DINT constants are limited to values between -32,768 and +32,767
MUL_INT	All	All	All	All
MUL_LREAL	None	MFV: 5.50	None	None
MUL_MIXED	None	All	All	None
MUL_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
MUL_UINT	None	All	All	None

N	**CPU Families, Models, Minimum Firmware Version (MFV)**

Mnemonic	VersaMax	PACSystems	Series 90-70	Series 90-30
NCCOIL	All	All	All	All
NCCON	All	All	All	All
NE_DINT NE_INT	All	All	All	All
NE_LREAL	None	MFV: 5.50	None	None
NE_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
NE_UINT	None	All	All	None
NEGCOIL	All	All	All	All
NEGCON	None	All	All	None
NOCON	All	All	All	All
NOFLT	None	All	All	None
NOT_DWORD	None	All	All	None
NOT_WORD	All	All	All	All
NTCOIL	None	All	None	None
NTCON	None	All	None	None

O, P, Q	CPU Families, Models, Minimum Firmware Version (MFV)			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
OFDT_HUNDS	All	All	All	MFV: 4.40
OFDT_SEC	None	All	All	None
OFDT_TENTHS	All	All	All	MFV: 4.40
OFDT_THOUS	All	MFV: 2.00	None	MFV: 4.40
ONDTR_HUNDS	All	All	All	All
ONDTR_SEC	None	All	All	None
ONDTR_TENTHS	All	All	All	All
ONDTR_THOUS	All	MFV: 2.00	None	All
OR_DWORD	None	All	All	None
OR_WORD	All	All	All	All
PID_IND	All	All	MFV: 2.02	All
PID_ISA	All	All	MFV: 2.02	All. but antireset windup action bit available only in MFV 6.50.
POSCOIL	All	All	All	All

POSCON	None	All	All	None
PTCOIL	None	All	None	None
PTCON	None	All	None	None

R	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
R_TRIG	None	All	MFV: 5.00	MFV: 5.00
RAD_TO_DEG	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
RAD_TO_DEG_LREAL	None	MFV: 5.50	None	None
RAD_TO_DEG_REAL	None	All	None	None
RANGE_DINT	All	All	MFV: 5.00	MFV: 4.40
RANGE_DWORD	None	All	MFV: 5.00	None
RANGE_INT	All	All	MFV: 5.00	MFV: 4.40
RANGE_UINT	None	All	MFV: 5.00	None
RANGE_WORD	All	All	MFV: 5.00	MFV: 4.40
REAL_TO_DINT	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
REAL_TO_INT	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
REAL_to_LREAL	None	MFV: 5.50	None	None
REAL_TO_UINT	None	All	Floating-point CPUs, MFV: 3.00	None
REAL_TO_WORD	All	None	None	Floating-point CPUs
RESETCOIL	All	All	All	All
ROL_DWORD	None	All	All	None
ROL_WORD	All	All	All	All
ROR_DWORD	None	All	All	None
ROR_WORD	All	All	All	All

S	CPU Families, Models, Minimum Firmware Version (MFV)			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
SCALE_DINT	None	MFV: 2.00	None	None
SCALE_INT	All	MFV: 2.00	None	None
SCALE_UINT	None	MFV: 2.00	None	None
SCALE_WORD	All	None	None	None
SCAN_SET_IO	None	All	MFV: 5.00	MFV: 5.00
SEARCH_EQ_BYTE SEARCH_EQ_DINT	All	All	All	All
SEARCH_EQ_DWORD	None	All	All	None
SEARCH_EQ_INT	All	All	All	All
SEARCH_EQ_UINT	None	All	All	None
SEARCH_EQ_WORD	All	All	All	All
SEARCH_GE_BYTE SEARCH_GE_DINT	All	All	All	All
SEARCH_GE_DWORD	None	All	All	None
SEARCH_GE_INT	All	All	All	All
SEARCH_GE_UINT	None	All	All	None
SEARCH_GE_WORD	All	All	All	All
SEARCH_GT_BYTE SEARCH_GT_DINT	All	All	All	All
SEARCH_GT_DWORD	None	All	All	None
SEARCH_GT_INT	All	All	All	All
SEARCH_GT_UINT	None	All	All	None
SEARCH_GT_WORD	All	All	All	All
SEARCH_LE_BYTE SEARCH_LE_DINT	All	All	All	All
SEARCH_LE_DWORD	None	All	All	None
SEARCH_LE_INT	All	All	All	All
SEARCH_LE_UINT	None	All	All	None
SEARCH_LE_WORD	All	All	All	All
SEARCH_LT_BYTE SEARCH_LT_DINT	All	All	All	All
SEARCH_LT_DWORD	None	All	All	None
SEARCH_LT_INT	All	All	All	All

SEARCH_LT_UINT	None	All	All	None
SEARCH_LT_WORD	All	All	All	All
SEARCH_NE_BYTE SEARCH_NE_DINT	All	All	All	All
SEARCH_NE_DWORD	None	All	All	None
SEARCH_NE_INT	All	All	All	All
SEARCH_NE_UINT	None	All	All	None
SEARCH_NE_WORD	All	All	All	All
SER	None	None	None	MFV: 9.00 of CPU350 and later
SETCOIL	All	All	All	All
SHFR_BIT	All	All	MFV: 3.00	All
SHFR_DWORD	None	All	All	None
SHFR_WORD	All	All	All	All
SHIFTL_DWORD	None	All	All	None
SHIFTL_WORD	All	All	All	All
SHIFTR_DWORD	None	All	All	None
SHIFTR_WORD	All	All	All	All
SIN	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
SIN_LREAL	None	MFV: 5.50	None	None
SIN_REAL	None	All	None	None
SIZE_OF	None	All	None	None
SORT_INT SORT_UINT SORT_WORD	None	All	All	None
SQRT_DINT	All	All	MFV: 2.00	All. In CPU341 and earlier, DINT constants are limited to values between -32,768 and +32,767.
SQRT_INT	All	All	MFV: 2.00	All
SQRT_LREAL	None	MFV: 5.50	None	None
SQRT_REAL	All	All	Floating-point	Floating-point

			CPUs, MFV: 3.00	CPUs
SUB_DINT	All	All	All	All. In CPU341 and earlier, DINT constants are limited to values between -32,768 and +32,767.
SUB_INT	All	All	All	All
SUB_LREAL	None	MFV: 5.50	None	None
SUB_REAL	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
SUB_UINT	None	All	All	None
SUS_IO	None	All	All	None
SUSP_IO_INTR	None	MFV: 3.50	None	None
SVC_REQ	The instruction is supported for all GE IP Controllers; however, support for individual Controller services varies from service to service. See below for details on individual services. For example, SVC_REQ 1 is service 1 of SVC_REQ.			
SVC_REQ 1	All	All	MFV: 2.02; however, there is a difference in the way newer CPUs work with SVC_REQ 1: MFV 6.00 provides the return values 0 for Normal Sweep, 1 for Constant Sweep, or 2 for Microcycle for request to read the timer state and value	MFV: 8.00
SVC_REQ 2 SVC_REQ 3 SVC_REQ 4	All	All	MFV: 2.02	MFV: 8.00
SVC_REQ 5	None	All	MFV: 2.02	None

SVC_REQ 6	All	All	MFV: 2.02	All except Micros
SVC_REQ 7	All	All	MFV: 2.02, but the POSIX format is supported only for MFV 7.92.	331+; 28-point Micros (IC693UDR005, UAA007, and UDR010); 23-point Micro IC693UAL006.
SVC_REQ 8 SVC_REQ 9 SVC_REQ 10 SVC_REQ 11	All	All	MFV: 2.02	MFV: 8.00
SVC_REQ 12	None	All	MFV: 2.02	MFV: 8.00
SVC_REQ 13 SVC_REQ 14 SVC_REQ 15 SVC_REQ 16	All	All	MFV: 2.02	All
SVC_REQ 17	None	All	MFV: 2.02	None
SVC_REQ 18	All	All	MFV: 2.02	Models 331+
SVC_REQ 19 SVC_REQ 20 SVC_REQ 21	None	All	MFV: 2.02	None
SVC_REQ 22	None	All	MFV: 4.02	None
SVC_REQ 23	All	All	MFV: 4.12	MFV: 4.40
SVC_REQ 24	None	All	None	All
SVC_REQ 25	None	All	MFV: 4.02	None
SVC_REQ 26 Role Switch	None	IC698CRE020	Used with CPU Redundancy, which is available on IC697CPU780 (MFV: 4.56), IC697CGR772, and IC697CGR935	None
SVC_REQ 26/30 Interrogate I/O	All	None	None	MFV: 4.40; not supported by Micro Controllers
SVC_REQ 27	None	IC698CRE020	Used with CPU	None

SVC_REQ 28			Redundancy, which is available on IC697CPU780 (MFV: 4.56), IC697CGR772, and IC697CGR935	
SVC_REQ 29	All	MFV: 2.00	None	331+
SVC_REQ 30	All	None	None	MFV: 4.40; not supported by Micro Controllers
SVC_REQ 32	None	All	MFV 5.03 of the 731, 732, 771, and 772 CPUs; MFV 5.50 of the 781 and later CPUs.	None
SVC_REQ 36	None	None (use %W mappings to access Bulk Memory Area)	IC697CPX772, IC697CPX782, IC697CPX928 and IC697CPX935 (MFV: 7.80)	None
SVC_REQ 39	None	None	CPX and CGR Controllers.	None
SVC_REQ 43	None	IC698CRE020	IC697CGR772, IC697CGR935, and IC697CPU780	None
SVC_REQ 44	None	None	MFV 7.91 CPX Series 90-70 CPUs. The adapter module IC697CMM742, MFV 2.70, supports logic-driven EGD.	None
SVC_REQ 45	None	RX3i; for use with an DSM324i or Motion Mate	None	MFV 10.00 of 350, 352, 360, 363, and 364 CPUs; for use

		DSM314 module		with a DSM324i or Motion Mate DSM314
SVC_REQ 46 SVC_REQ 48	None	None	None	MFV 10.00 of 350, 352, 360, 363, and 364 CPUs; for use with a DSM324i or Motion Mate DSM314
SVC_REQ 50 SVC_REQ 51	None	All	None	None
SWAP_DWORD SWAP_WORD	None	All	All	None
SWITCH_POS	None	MFV: 2.00	None	None

T	**CPU Families, Models, Minimum Firmware Version (MFV)**			
Mnemonic	**VersaMax**	**PACSystems**	**Series 90-70**	**Series 90-30**
TAN	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs
TAN_LREAL	None	MFV: 5.50	None	None
TAN_REAL	None	All	None	None
TBL_RD_DINT TBL_RD_DWORD TBL_RD_INT TBL_RD_UINT TBL_RD_WORD	None	All	All	None
TBL_WRT_DINT TBL_WRT_DWORD TBL_WRT_INT TBL_WRT_UINT TBL_WRT_WORD	None	All	All	None
TMR_HUNDS	All	All	All	All
TMR_SEC	None	All	All	None
TMR_TENTHS	All	All	All	All
TMR_THOUS	All	MFV: 2.00	None	All
TRUNC_DINT TRUNC_INT	All	All	Floating-point CPUs, MFV: 3.00	Floating-point CPUs

U - Z	**CPU Families, Models, Minimum Firmware Version (MFV)**

Mnemonic	VersaMax	PACSystems	Series 90-70	Series 90-30
UINT_TO_BCD4 UINT_TO_DINT UINT_TO_INT	None	All	MFV: 3.00	None
UINT_TO_REAL	None	All	Floating-point CPUs, MFV: 3.00	None
UPCTR	All	All	All	All
V_WIRE	All	All	All	All
VME_CFG_READ VME_CFG_WRITE	None	None	MFV: 4.00	None
VME_RD_BYTE VME_RD_WORD VME_RMW_BYTE VME_RMW_WORD VME_TS_BYTE VME_TS_WORD VME_WRT_BYTE VME_WRT_WORD	None	None	All	None
WORD_TO_REAL	All	None	None	Floating-point CPUs
XOR_DWORD	None	All	All	None
XOR_WORD	All	All	All	All

Notes

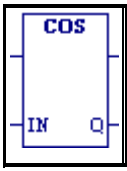
- The Series 90-**70** CPUs that have floating-point capabilities are CPU732, CPU772, CPU780, CPU782, CPM790, CPM914, CPM924, CPM915, and CPM925.
- The Series 90-**30** CPUs that have floating-point capabilities are all firmware versions of CPU352 and firmware version 9.00 and later of CPU350 and later.

Advanced Math Instructions

The LD Advanced Math instructions perform logarithmic, exponential, square root, trigonometric, and inverse trigonometric operations.

Instruction	*Mnemonics*	*Description*
Cosine	COS COS_LREAL COS_REAL	Calculates the cosine of the operand 'IN' (in radians).
Exponential	EXP EXP_LREAL EXP_REAL	Calculates the inverse natural logarithm of the 'IN' operand ($\mathbf{e}^{IN}$).
	EXPT EXPT_LREAL EXPT_REAL	Calculates 'IN1' to the 'IN2' power ($IN1^{IN2}$).
Inverse Cosine	ACOS ACOS_LREAL ACOS_REAL	Calculates the inverse cosine of the 'IN' operand and expresses the result in radians.
Inverse Sine	ASIN ASIN_LREAL ASIN_REAL	Calculates the inverse sine of the 'IN' operand and expresses the result in radians.
Inverse Tangent	ATAN ATAN_LREAL ATAN_REAL	Calculates the inverse tangent of the 'IN' operand and expresses the result in radians.
Logarithmic	LN LN_LREAL LN_REAL	Calculates the natural logarithm of the operand 'IN'.
	LOG LOG_LREAL LOG_REAL	Calculates the base 10 logarithm of the operand 'IN'.
Sine	SIN SIN_LREAL SIN_REAL	Calculates the sine of the operand 'IN' (in radians).
Square Root	SQRT_DINT SQRT_INT SQRT_LREAL SQRT_REAL	Calculates the square root of the operand 'IN'.
Tangent	TAN TAN_LREAL TAN_REAL	Calculates the tangent of the operand 'IN' (in radians).

Cosine



Mnemonics:
COS
COS_LREAL
COS_REAL

Operation

The Cosine instruction is used to find the trigonometric cosine of its input. When it receives power flow, it computes the cosine of IN and stores the result in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

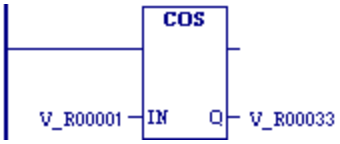
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The COS mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Number of radians. $-2^{63} < IN < +2^{63}$. ($2^{63} \approx 9.22 \times 10^{18}$.)
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Cosine value of IN

Example

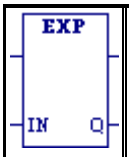
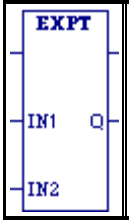
The cosine of the value in %R00001 is placed in %R00033.



CPU Support

COS is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
COS_LREAL is supported for PACSystems firmware version 5.50 or later.
COS_REAL is supported for all PACSystems. It has replaced COS for PACSystems, which supports COS for backward compatibility.

Exponential

	**Mnemonics:** EXP EXP_LREAL EXP_REAL
	**Mnemonics:** EXPT EXPT_LREAL EXPT_REAL

Operands: EXP, EXP_LREAL, and EXP_REAL | EXPT, EXPT_LREAL, and EXPT_REAL

Operation

When an exponential instruction receives power flow, it performs the appropriate exponential operation on the input value(s) and places the result in output Q.

- For the inverse natural log (EXP, EXP_LREAL, or EXP_REAL) instruction, ***e*** is raised to the power specified by IN and the result is placed in Q. IN and Q must be of the same data type.
- For the Power of X (EXPT, EXPT_LREAL, or EXPT_REAL) instruction, the value of input IN1 is raised to the power specified by the value IN2 and the result is placed in output Q. IN1, IN2, and Q must be of the same data type.

The power flow output is energized unless at least one of the following situations is encountered:

- There is overflow.
- IN, IN1, or IN2 is NaN (Not a Number).
- For EXPT, IN1 is negative.
- (PACSystems CPUs or Series 90-30 CPU352.) For EXP, IN is negative infinity.

CPU Support

EXP and EXPT are supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and for Series 90-30 floating-point CPUs.

EXP_LREAL and EXPT_LREAL are supported for PACSystems firmware version 5.50 or later.

EXP_REAL and EXPT_REAL are supported for all PACSystems. They have respectively replaced EXP and EXPT for PACSystems, which supports EXP and EXPT for backward compatibility.

Operands for EXP, EXP_LREAL, and EXP_REAL

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.
- IN and Q must be of the same data type.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The EXP mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The power to raise ***e*** to; the number to calculate the inverse natural log of. **Note:** When IN = -∞, EXP returns a value of 0, as expected, but does *not* pass power for PACSystems CPUs or Series 90-30 CPU352. All other Series 90-30 CPUs do pass power in this case, even though the output is 0.
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	e^{IN}

Operands for EXPT, EXPT_LREAL, and EXPT_REAL

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.
- IN1, IN2, and Q must be of the same data type.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	REAL or LREAL variable or constant. Must be the same data type as for IN2 and Q. **Note:** The EXPT mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G.	The base value
IN2	REAL or LREAL variable or	data flow, R, P, L, AI, AQ,	The exponent

	constant	W. PACSystems also supports I, Q, M, T, G.	
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G.	IN1[IN2]

Inverse Cosine

Mnemonics:
ACOS
ACOS_LREAL
ACOS_REAL

Operation

When an Inverse Cosine (ACOS) instruction receives power flow, it computes the inverse cosine of IN and stores the result in radians in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

Notes

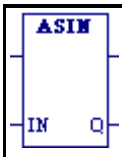
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The ACOS mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	-1 ≤ IN ≤ 1. The value to process.
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Inverse cosine of IN. Expressed in radians. 0 ≤ Q ≤ π.

CPU Support

ACOS is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
ACOS_LREAL is supported for PACSystems firmware version 5.50 or later.
ACOS_REAL is supported for all PACSystems. It has replaced ACOS for PACSystems, which supports ACOS for backward compatibility.

Inverse Sine



Mnemonics:
ASIN
ASIN_LREAL
ASIN_REAL

Operation

When an Inverse Sine (ASIN) instruction receives power flow, it computes the inverse sine of IN and stores the result in radians in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

Notes

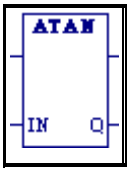
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The ASIN mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	-1 ≤ IN ≤ 1. The value to process.
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Inverse sine of IN. Expressed in radians. (-π/2) ≤ Q ≤ π/2).

CPU Support

ASIN is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
ASIN_LREAL is supported for PACSystems firmware version 5.50 or later.
ASIN_REAL is supported for all PACSystems. It has replaced ASIN for PACSystems, which supports ASIN for backward compatibility.

Inverse Tangent



Mnemonics:
ATAN
ATAN_LREAL
ATAN_REAL

Operation

When an Inverse Tangent (ATAN) instruction receives power flow, it computes the inverse tangent of IN and stores the result in radians in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

Notes

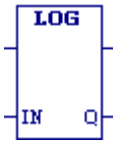
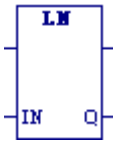
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The ATAN mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	$-\infty \leq IN \leq \infty$. The value to process.
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Inverse tangent of IN. Expressed in radians. $(-\pi/2) \leq Q \leq \pi/2)$.

CPU Support

ATAN is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
ATAN_LREAL is supported for PACSystems firmware version 5.50 or later.
ATAN_REAL is supported for all PACSystems. It has replaced ATAN for PACSystems, which supports ATAN for backward compatibility.

Logarithmic

	**Mnemonics:** LOG LOG_LREAL LOG_REAL
	**Mnemonics:** LN LN_LREAL LN_REAL

Operation

When a logarithmic instruction receives power flow, it performs the appropriate logarithmic operation on the value in input IN and places the result in output Q. IN and Q must be of the same data type.

- For the Base 10 Logarithm (LOG, LOG_LREAL, or LOG_REAL) instruction, the base 10 logarithm of IN is placed in Q.
- For the Natural Logarithm (LN, LN_LREAL, or LN_REAL) instruction, the natural logarithm of IN is placed in Q.

The power flow output is energized unless at least one of the following invalid situations is encountered:

- There is overflow.
- IN is negative.
- IN is NaN (Not a Number).

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The LOG and LN mnemonics support only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to calculate the base 10 or natural logarithm of
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W.	The base 10 or natural logarithm of IN

		PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	

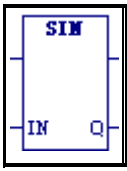
CPU Support

LOG and LN are supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

LOG_LREAL and LN_LREAL are supported for PACSystems firmware version 5.50 or later.

LOG_REAL and LN_REAL are supported for all PACSystems. LOG and LN have respectively replaced LOG and LN for PACSystems, which respectively supports LOG and LN for backward compatibility.

Sine



Mnemonics:
SIN
SIN_LREAL
SIN_REAL

Operation

The Sine instruction is used to find the trigonometric sine of its input. When it receives power flow, it computes the sine of IN and stores the result in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

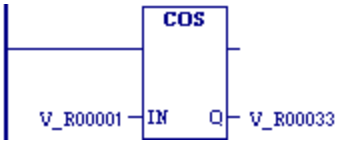
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The SIN mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Number of radians. $-2^{63} < IN < +2^{63}$. ($2^{63} \approx 9.22 \times 10^{18}$.)
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Sine value of IN

Example

The sine of the value in %R00001 is placed in %R00033.



CPU Support

SIN is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
SIN_LREAL is supported for PACSystems firmware version 5.50 or later.
SIN_REAL is supported for all PACSystems. It has replaced SIN for PACSystems, which supports SIN for backward compatibility.

Square Root

Mnemonics:
SQRT_DINT
SQRT_INT
SQRT_LREAL
SQRT_REAL

Operands and CPU support: SQRT_DINT | SQRT_INT | SQRT_LREAL | SQRT_REAL

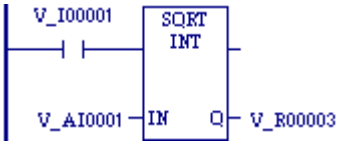
Operation

When the Square Root instruction receives power flow, it finds the square root of IN and stores the result in Q. IN and Q must be of the same data type.
The power flow output is energized unless at least one of these invalid situations is encountered:

- There is overflow.
- IN is negative.
- IN is a NaN (Not a Number).

Example

The square root of the integer number located at %AI0001 is placed into %R00003 whenever %I00001 is ON.



SQRT_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to calculate the square root of. If IN < 0, the instruction does not pass power flow. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The calculated square root.

CPU Support

SQRT_DINT is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Revision 2.0 or later CPUs, and all Series 90-30 CPUs.

In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

SQRT_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The single-precision integer to calculate the square root of. If IN < 0, the instruction does not pass power flow.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The calculated square root.

CPU Support

SQRT_INT is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Revision 2.00 or later CPUs, and all Series 90-30 CPUs.

SQRT_LREAL

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 64 or more instead of an LREAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*

IN	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable.	The value to calculate the square root of. If IN < 0, the instruction does not pass power flow.
Q	LREAL variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable.	The calculated square root

CPU Support

SQRT_LREAL is supported for PACSystems firmware version 5.50 or later.

SQRT_REAL

Operands

Notes

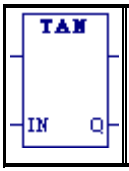
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a REAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to calculate the square root of. If IN < 0, the instruction does not pass power flow.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The calculated square root

CPU Support

SQRT_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

Tangent



Mnemonics:
TAN
TAN_LREAL
TAN_REAL

Operation

The Tangent instruction is used to find the trigonometric tangent of its input. When it receives power flow, it computes the tangent of IN and stores the result in output Q. IN and Q must be of the same data type.
The output power flow is energized when the instruction is performed without overflow, unless an invalid operation occurs and/or IN is not a number.

Operands

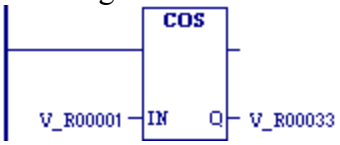
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The TAN mnemonic supports only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Number of radians. $-2^{63} < IN < +2^{63}$. ($2^{63} \approx 9.22 \times 10^{18}$.)
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	Tangent value of IN

Example

The tangent of the value in %R00001 is placed in %R00033.



CPU Support

TAN is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
TAN_LREAL is supported for PACSystems firmware version 5.50 or later.
TAN_REAL is supported for all PACSystems. It has replaced TAN for PACSystems, which supports TAN for backward compatibility.

Bit Operations Instructions

Data Lengths

The LD Bit Operation instructions perform comparison, logical, and move operations on bit strings.

Instruction	*Mnemonics*	*Description*
Bit Position	BIT_POS_DWORD BIT_POS_WORD	Bit Position. Locates a bit set to 1 in a bit string.
Bit Sequencer	BIT_SEQ	Bit Sequencer. Sequences a string of bit values, starting at ST. Performs a bit sequence shift through an array of bits. The maximum length allowed is 256 words.
Bit Set, Clear	BIT_SET_DWORD BIT_SET_WORD	Bit Set. Sets a bit in a bit string to 1.
	BIT_CLR_DWORD BIT_CLR_WORD	Bit Clear. Clear a bit within a string by setting that bit to 0.
Bit Test	BIT_TEST_DWORD BIT_TEST_WORD	Bit Test. Tests a bit within a bit string to determine whether that bit is currently 1 or 0.
Logical AND	AND_DWORD AND_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 1, places a 1 in the corresponding location in output string Q; otherwise, places a 0 in the corresponding location in Q.
Logical NOT	NOT_DWORD NOT_WORD	Logical invert. Sets the state of each bit in output bit string Q to the opposite state of the corresponding bit in bit string IN1.
Logical OR	OR_DWORD OR_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 0, places a 0 in the corresponding location in output string Q; otherwise, places a 1 in the corresponding location in Q.
Logical XOR	XOR_DWORD XOR_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are different, places a 1 in the corresponding location in the output bit string Q; when a pair of corresponding bits are the same, places a 0 in Q.
Masked Compare	MASK_COMP_DWORD MASK_COMP_WORD	Masked Compare. Compares the contents of two separate bit strings with the ability to mask selected bits.
Rotate Bits	ROL_DWORD ROL_WORD	Rotate Left. Rotates all the bits in a string a specified number of places to the left.

	ROR_DWORD ROR_WORD	Rotate Right. Rotates all the bits in a string a specified number of places to the right.
Shift Bits	SHIFTL_DWORD SHIFTL_WORD	Shift Left. Shifts all the bits in a word or string of words to the left by a specified number of places.
	SHIFTR_DWORD SHIFTR_WORD	Shift Right. Shifts all the bits in a word or string of words to the right by a specified number of places.

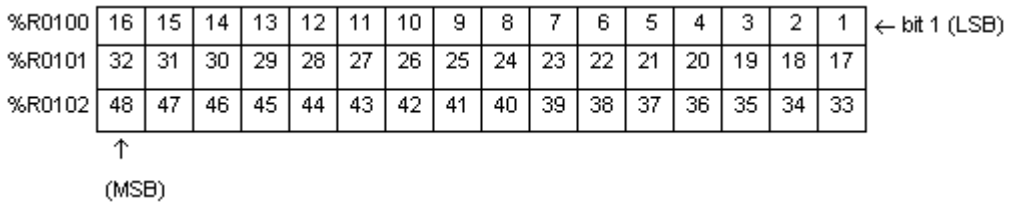
Note: For all bit operations, the bit group of instructions not explicitly bit-typed will affect the transition coils and transition contacts for all bits in the written BYTE, WORD, or DWORD.

Data Lengths for the Bit Operation instructions

On VersaMax CPUs and Series 90-30 CPUs, the Bit Operation instructions operate on a single WORD of data or up to 256 WORDs that occupy adjacent memory locations, except the Logical AND, OR, XOR, and NOT (Invert) instructions, which operate on a single word of data.

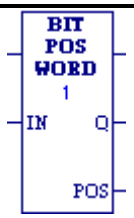
On PACSystems CPUs and Series 90-70 CPUs, all the Bit Operation instructions operate on a single WORD or DWORD of data or up to 256 WORDs or DWORDs that occupy adjacent memory locations.

All Bit Operation instructions treat the WORD or DWORD data as a continuous string of bits, with bit 1 of the first WORD or DWORD being the Least Significant Bit (LSB). The last bit of the last WORD or DWORD is the Most Significant Bit (MSB). For example, if you specify three WORDs of data beginning at reference %R0100, they are treated as 48 contiguous bits.



Warning: Overlapping input and output reference address ranges in multiword instructions is not recommended, as it can produce unexpected results.

Bit Position

		
PACSystems and Series 90-70		Other CPUs

Operation

The Bit Position instruction locates a bit set to 1 in a bit string.
Each scan that power is received, the instruction scans the bit string starting at IN. When the instruction stops scanning, either a bit equal to 1 has been found or the entire length of the string has been scanned.
POS is set to the position within the bit string of the first non-zero bit; POS is set to zero if no non-zero bit is found.
A string length of 1 to 256 WORDs can be selected. On PACSystems and Series 90-70, you can select DWORDs. The instruction passes power flow to the right whenever it receives power.

Note: When using the Bit Test, Bit Set, Bit Clear or Bit Position instruction, the bits are numbered 1 through 16, NOT 0 through 15.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

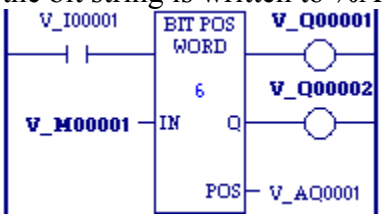
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs or DWORDs in the bit string. 1 ≤ Length ≤ 256. Default: 1.
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The first WORD or DWORD of the data to operate on
	DWORD variable or constant	data flow, R, P, L, W, AI, AQ PACSystems also supports I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable	
Q	Power flow		(PACSystems and Series 90-70

			only.) Energized if a bit set to 1 is found
POS	INT variable	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The position of the first non-zero bit found, or zero if no non-zero bit is found

Examples

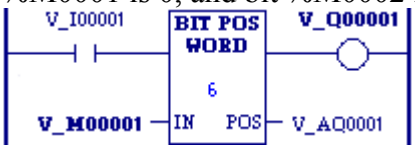
Example 1: for PACSystems CPUs and Series 90-70 CPUs

The only difference with example 1 is that if a bit equal to 1 is found, its location within the bit string is written to %AQ0001 *and %Q00002 is turned on*.



Example 2: for other CPUs

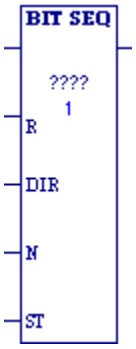
When %I0001 is set, the bit string starting at %M0001 is searched until a bit equal to 1 is found, or 6 words have been searched. Coil %Q0001 is turned on. If a bit equal to 1 is found, its location within the bit string is written to %AQ001. If %I0001 is set, bit %M0001 is 0, and bit %M0002 is 1, then the value written to %AQ001 is 2.



CPU Support

BIT_POS_WORD is supported for all GE IP CPUs.
BIT_POS_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Bit Sequencer



Operation

A Bit Sequencer (BIT_SEQ) built-in function block instance performs a bit sequence shift through a series of contiguous bits.
The operation of BIT_SEQ depends on the value of the reset input (R) and both the current value and previous value of the enabling power flow input (EN):

R Current Execution	*EN Previous Execution*	*EN Current Execution*	*Bit Sequencer Execution*
ON	ON/OFF	ON/OFF	Bit sequencer resets
OFF	OFF	ON	Bit sequencer increments/decrements by 1
		OFF	Bit sequencer does not execute
	ON	ON/OFF	Bit sequencer does not execute

The reset input (R) overrides the enabling power flow (EN) and always resets the sequencer. When R is active, the current step number is set to the value of the optional N operand. If the user did not specify N, the step number is set to 1. All of the bits in the bit sequencer, ST, are set to 0, except for the bit pointed to by the current step, which is set to 1.
When EN is active and Reset is not active and the previous EN was OFF, the bit pointed to by the current step number is cleared. The current step number is incremented or decremented, based on the direction (DIR) operand. Then the bit pointed to by the new step number is set to 1.

- When the step number is being incremented and it goes outside the range of (1 ≤ step number ≤ Length operand), it is set back to 1.
- When the step number is being decremented and it goes outside the range of (1 ≤ step number ≤ Length operand), it is set to Length.

The parameter ST is optional. If it is not used, BIT_SEQ operates as described above, except that no bits are set or cleared. Basically, BIT_SEQ just cycles the current step number through its legal range.
BIT_SEQ passes power to the right whenever it receives power.

Note: Coil checking, for a BIT_SEQ function block instance, checks for 16 bits from the ST parameter, even when the Length operand is less than 16.

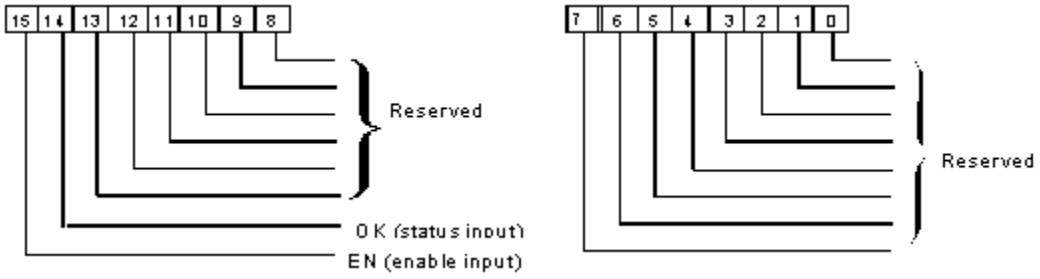
Memory Required for a Bit Sequencer

Each bit sequencer uses a one-dimensional, three-word array of %R, %L, or %P memory to store the information:

- Word 1: current step number
- Word 2: length of sequence (in bits)
- Word 3: control word

Note: Do not write to these registers by any means.

Word 3 (the control word) stores the state of the boolean inputs and outputs of its function block instance, in the following format:



Notes

- Bits 0 through 13 are not used.
- In the N operand, bits need to be entered as 1 through 16, *not* 0 through 15.

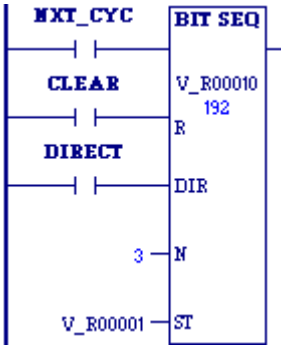
Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	???? is the beginning address of a three-word array: Word 1: current step number Word 2: length of sequence in bits Word 3: control word, which tracks the status of the last enabling power flow and the status of the power flow to the right. **Warning:** Do not use write to these reference addresses by any means. Overlapping references results in erratic operation of BIT_SEQ.
Length	Constant		The length. The number of bits in the bit sequencer, ST, that BIT_SEQ will step through. 1 ≤ Length ≤ 256. Default: 1.
R	Power flow		When R is energized, the step number of BIT_SEQ is set to the value in N (default = 1), and the bit sequencer, ST, is filled with

			zeros, except for the current step number bit.
DIR	Power flow		When DIR and EN are energized, and R is OFF and the previous EN was OFF, the step number of BIT_SEQ is incremented prior to the shift. Otherwise, it is decremented.
N	INT variable or constant	I, Q, M, T, G, R, P, L, W, AI, AQ. PACSystems also supports data flow, symbolic, I/O variable.	Optional. The value that the step number is set to when R is energized. Default value is 1. 1 ≤ N ≤ Length. If N < 1, the step number will be reset to 1 when R is energized. If N > Length, the step number will be reset to Length. **Note:** Bits need to be entered as 1 through 16, *not* 0 through 15.
ST	BYTE, WORD, or DWORD variable	I, Q, M, T, SA, SB, SC, G, R, P, L, W, AQ, symbolic, I/O variable	Optional. The bit sequencer, which occupies (Length / 8) contiguous bytes in memory to accommodate the Length number of bits to be sequenced. If ST is not used, the Bit Sequencer built-in function block instance operates as described above, except that no bits are set or cleared. The function block instance just cycles the current step number (in word 2 of the control block) through its legal range. **Note:** (Series 90 Micro firmware bug.) A Series 90 Micro treats the ST input of both BIT_SEQ and SHFR_WORD as an output when it updates its coil usage map. To prevent equality problems with a Series 90 Micro target, Machine Edition also adds the ST input of those instructions to the coil usage map. However, if you map the ST input of multiple BIT_SEQ and/or SHFR_WORD instructions to the same address, you will get a Multiple Coil Use Warning upon validation, unless you edit the Multiple Coil Use Warning option and set it to No Warning.

Example

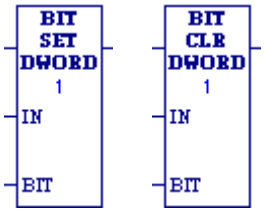
The Bit Sequencer operates on register memory %R0001. Its instance data is stored in registers %R0010, %R0011, and %R0012. When CLEAR is active, the sequencer is reset and the current step is set to step number 3, as specified in N. The third bit of %R0001 is set to one and the other seven bits are set to zero.
When NXT_CYC is active and CLEAR is not active, the bit for step number 3 is cleared and the bit for step number 2 or 4 (depending on whether DIR is energized) is set.



CPU Support

BIT_SEQ is supported for all GE IP CPUs.

Bit Set, Bit Clear



Mnemonics
BIT_SET_DWORD
BIT_SET_WORD
BIT_CLR_DWORD
BIT_CLR_WORD

Operation

The Bit Set (BIT_SET_DWORD and BIT_SET_WORD) instruction sets a bit in a bit string to 1. The Bit Clear (BIT_CLR_DWORD and BIT_CLR_WORD) instruction clears a bit in a string by setting the bit to 0.
Each scan that power is received, the instruction sets or clears the specified bit. If a variable rather than a constant is used to specify the bit number, the same instruction can set or clear different bits on successive scans.
You can select a string length of 1 to 256 WORDs. On PACSystems and Series 90-70, you can select DWORDs. The instruction passes power flow to the right, unless the value for BIT is outside the range (1 ≤ BIT ≤ (16 * length) for a WORD and 1 ≤ BIT ≤ (32 * length) for a DWORD), in which case there is no power flow.

Note: When using the Bit Set or Bit Clear instruction, the bits are numbered 1 through 16 for a WORD, *not* 0 through 15. They are numbered 1 through 32 for a DWORD.

Operands

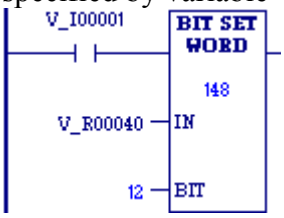
Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs or DWORDs in the bit string. 1 ≤ Length ≤ 256. Default: 1.
IN	WORD variable	I, Q, M, T, SA, SB, SC, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The first WORD or DWORD of the data to process
	DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
BIT	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The number of the bit to set or clear in IN. 1 ≤ BIT ≤ (16 * length).

Examples

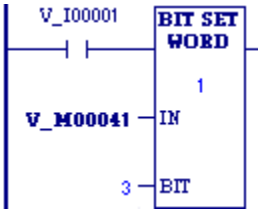
Example 1: any CPU

Whenever input V_I0001 is set, bit 12 of the string beginning at reference %R0040 (as specified by variable V_R0040) is set to 1.



Example 2: PACSystems CPUs and Series 90-70 CPUs only

%M41–%M48 will be solved as written to be a Transition Status. These bits may not perform as expected when used as a transition contact or coil. If you wish to use Bit Operation instructions in conjunction with transition instructions, your Bit Operation instruction should be type BOOL.



Note: On a Series 90-70, for all bit operations, the bit group of instructions not explicitly bit-typed will affect the transitions (coils and contacts) for all bits in the written BYTE, WORD, or DWORD.

CPU Support

BIT_SET_WORD and BIT_CLR_WORD are supported for all GE IP CPUs. BIT_SET_DWORD and BIT_CLR_DWORD are supported for PACSystems CPUs and Series 90-70 CPUs.

Bit Test

PACSystems and Series 90-70 only

All CPUs

Operation

When the Bit Test instruction receives power flow, it tests a bit within a bit string to determine whether that bit is currently 1 or 0. The result of the test is placed in output Q. Each scan that power is received, the Bit Test instruction sets its output Q to the same state as the specified bit. If a register rather than a constant is used to specify the bit number, the same instruction can test different bits on successive sweeps. If the value of BIT is outside the range (1 ≤ BIT ≤ (16 * length) for a WORD and 1 ≤ BIT ≤ (32 * length) for a DWORD), then Q is set OFF.
You can specify a string length of 1 to 256 WORDs. On a PACSystems or Series 90-70 CPU, you can also select DWORDs.

Note: When using the Bit Test instruction, the bits are numbered 1 through 16 for a WORD, *not* 0 through 15. They are numbered 1 through 32 for a DWORD.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

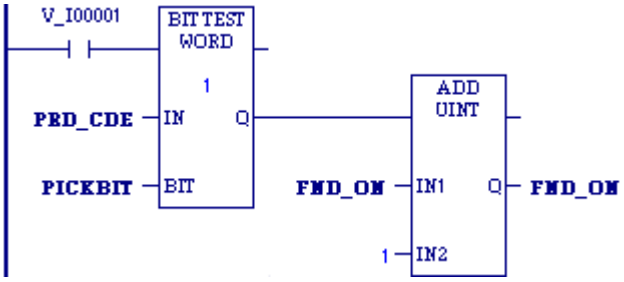
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs or DWORDs in the data string to test. 1 ≤ Length ≤ 256. Default: 1. **Note:** BIT_TEST_DWORD is available only on Series 90-70 CPUs.
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The first WORD or DWORD in the data to test

	(PACSystems and series 90-70 only.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
BIT	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The number of the bit to test in IN. 1 ≤ BIT ≤ (16*Length).
Q	Power flow		The state of the specific bit tested; Q is energized if the bit tested is a 1.

Examples

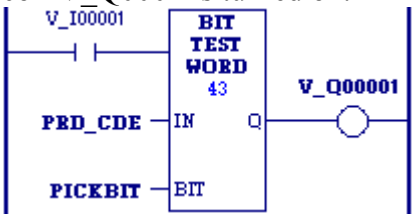
Example for PACSystems CPUs and Series 90-70 CPUs

Whenever input V_I0001 is set, the bit at the location contained in reference PICKBIT is tested. The bit is part of string PRD_CDE. If it is 1, output Q passes power flow to the ADD instruction, causing 1 to be added to the current value of the ADD instruction input IN1.



Example for any CPU

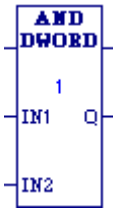
Whenever input V_I0001 is set, the bit at the location contained in reference PICKBIT is tested. The bit is part of string PRD_CDE. If it is 1, output Q passes power flow and the coil V_Q0001 is turned on.



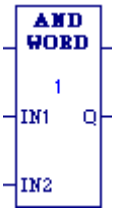
CPU Support

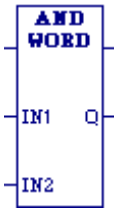
BIT_TEST_WORD is supported for all GE IP CPUs.
BIT_TEST_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Logical AND



PACSystems and Series 90-70





Other CPUs

Operation

Each scan that power is received, the Logical AND instruction examines each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each. You can specify a string length of 1 to 256 WORDs. On a PACSystems CPU or a Series 90-70 CPU, you can also select DWORDs.

- On a PACSystems or a Series 90-70, AND requires the Length operand.
- On other CPUs, AND operates on a single WORD.

If both bits examined by the Logical AND instruction are 1, AND places a 1 in the corresponding location in output string Q. If either bit is 0 or both bits are 0, AND places a 0 in string Q in that location.
AND passes power flow to the right whenever it receives power.

Tips

- You can use the Logical AND instruction to build masks or screens, where only certain bits are passed (the bits opposite a 1 in the mask), and all other bits are set to 0.
- You can also use the Logical AND instruction to clear an area of WORD or DWORD memory by ANDing the bits with another bit string known to contain all 0s. The IN1 and IN2 bit strings specified may overlap.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

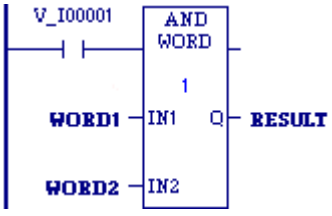
Operand	*Data Type*	*Memory Area*	*Description*
Length (PACSystems and Series 90-70.)	Constant		The number of words in the bit string to AND. 1 ≤ Length ≤ 256. Default: 1.
IN1	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to AND, or the reference for the first WORD or DWORD of the first string to AND

	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
IN2 (Must be the same data type as IN1.)	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to AND, or the reference for the first WORD or DWORD of the second string to AND
	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
Q (Must be the same data type as IN1.)	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The reference for the first WORD or DWORD of the AND's result
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	

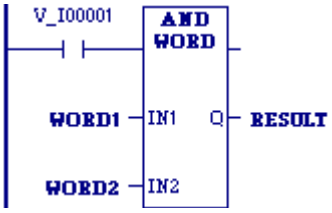
Example

When input v_I0001 is set, the 16-bit strings represented by variables WORD1 and WORD2 are examined. The logical AND places the results in output string RESULT.

Logic for a PACSystems or a Series 90-70 CPU



Logic for other CPUs



WORD1	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
WORD2	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	1

RESULT	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0

CPU Support

AND_WORD is supported for all GE IP CPUs.
AND_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Logical NOT

PACSystems and Series 90-70

Other CPUs

Operation

When the Logical Not or Logical Invert (NOT) instruction receives power flow, it sets the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string IN1.

All bits are altered on each scan that power is received, making output string Q the logical complement of IN1. Logical NOT passes power flow to the right whenever it receives power. You can specify a string length of 1 to 256 WORDs. On a PACSystems CPU or Series 90-70 CPU, you can also select DWORDs.

- On a PACSystems or a Series 90-70, NOT requires the Length operand.
- On other CPUs, NOT operates on a single WORD.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

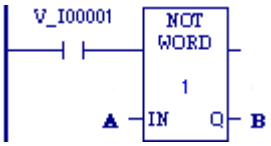
Operand	*Data Type*	*Memory Area*	*Description*
Length (PACSystems and Series 90-70.)	Constant		(PACSystems and 90-70.) The number of WORDs or DWORDs in the bit string to NOT. 1 ≤ Length ≤ 256. Default: 1.
IN1	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to NOT, or the reference for the first WORD or DWORD of the string to NOT
	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	

Q (Must be the same data type as IN1.)	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The reference for the first WORD or DWORD of the NOT's result
	(PACSystems and Series 90-70.) DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	

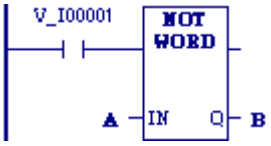
Example

Whenever input V_I0001 is set, the bit string represented by the variable A is negated. Logical NOT stores the resulting inverse bit string in variable B.

Logic for a PACSystems or a Series 90-70 CPU



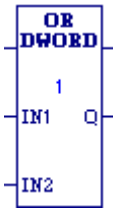
Logic for other CPUs



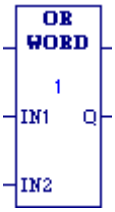
CPU Support

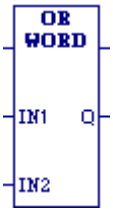
NOT_WORD is supported for all GE IP CPUs.
NOT_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Logical OR



PACSystems and Series 90-70





Other CPUs

Operation

Each scan that power is received, the Logical OR instruction examines each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each. You can specify a string length of 1 to 256 WORDs. On a PACSystems CPU or a Series 90-70 CPU, you can also select DWORDs.

- On a PACSystems or a Series 90-70, OR requires the Length operand.
- On other CPUs, OR operates on a single WORD.

If the Logical OR instruction finds at least one bit that is 1, it places a 1 in the corresponding location in output string Q. If both bits are 0, Logical OR places a 0 in string Q in that location. The instruction passes power flow to the right whenever it receives power.

Tips

- You can use the Logical OR instruction to combine strings or to control many outputs with one simple logical structure. The Logical OR instruction is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string.
- You can use the Logical OR instruction to drive indicator lamps directly from input states, or to superimpose blinking conditions on status lights.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

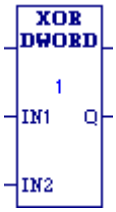
Operand	*Data Type*	*Memory Area*	*Description*
Length (PACSystems and Series 90-70 only.)	Constant		The number of WORDs or DWORDs in the bit string to OR. 1 ≤ Length ≤ 256. Default: 1.
IN1	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to OR, or the reference for the first WORD or DWORD of the first

	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	DWORD of the first string to OR
IN2 (Must be the same data type as IN1.)	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to OR, or the reference for the first WORD or DWORD of the second string to OR
	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
Q (Must be the same data type as IN1.)	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The reference for the first WORD or DWORD of the OR's result
	(PACSystems and Series 90-70.) DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	

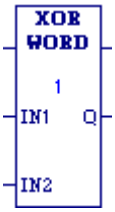
CPU Support

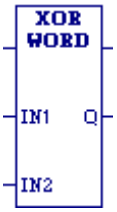
OR_WORD is supported for all GE IP CPUs.
OR_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Logical XOR



PACSystems and Series 90-70





Other CPUs

Operation

When the Exclusive OR (XOR) instruction receives power flow, it compares each bit in bit string IN1 with the corresponding bit in string IN2. If the bits are different, a 1 is placed in the corresponding position in the output bit string.

Each scan that power is received, XOR examines each bit in string IN1 and the corresponding bit in string IN2, beginning with the least significant bit in each. You can specify a string length of 1 to 256 WORDs. On a PACSystems or a Series 90-70 CPU, you can also select DWORDs.

- On a PACSystems or Series 90-70, XOR requires the Length operand.
- On other CPUs, XOR operates on a single WORD.

For each pair of bits examined, if only one bit is 1, then XOR places a 1 in the corresponding location in bit string Q. XOR passes power flow to the right whenever it receives power.

Tips

- If string IN2 and output string Q begin at the same reference, a 1 placed in string IN1 will cause the corresponding bit in string IN2 to alternate between 0 and 1, changing state with each scan as long as power is received.
- You can program longer cycles by pulsing the power flow to the instruction at twice the desired rate of flashing. The power flow pulse should be one scan long (oneshot type coil or selfresetting timer).
- You can use XOR to quickly compare two bit strings, or to blink a group of bits at the rate of one ON state per two scans.
- XOR is useful for transparency masks.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

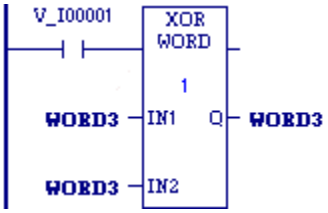
Operand	*Data Type*	*Memory Area*	*Description*
Length (PACSystems	Constant		The number of WORDs or DWORDs in the bit

and Series 90-70.)			string to XOR. 1 ≤ Length ≤ 256. Default: 1.
IN1	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to XOR, or the reference for the first WORD or DWORD of the first string to XOR
	(PACSystems and Series 90-70.) DWORD variable or constant	data flow,R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
IN2 (Must be the same data type as IN1.)	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The constant to XOR, or the reference for the first WORD or DWORD of the second string to XOR
	(PACSystems and Series 90-70.) DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
Q (Must be the same data type as IN1.)	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The reference for the first WORD or DWORD of the XOR's result
	(PACSystems and Series 90-70.) DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	

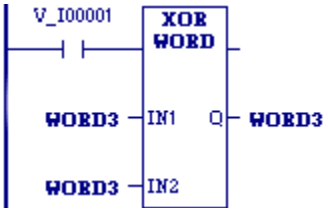
Example

Whenever V_I0001 is set, the bit string represented by the variable WORD3 is cleared (set to all zeros).

Logic for a PACSystems or a Series 90-70 CPU



Logic for other CPUs



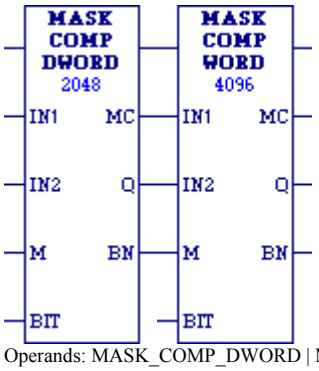
I1 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
I2 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0

Q (WORD3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPU Support

XOR_WORD is supported for all GE IP CPUs.
XOR_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.

Masked Compare



Operands: MASK_COMP_DWORD | MASK_COMP_WORD

Operation

The Masked Compare (MASK_COMP_DWORD and MASK_COMP_WORD) instruction compares the contents of two bit strings. It provides the ability to mask selected bits.

Tip: Input string 1 might contain the states of outputs such as solenoids or motor starters. Input string 2 might contain their input state feedback, such as limit switches or contacts.

When the instruction receives power flow, it begins comparing the bits in the first string with the corresponding bits in the second string. Comparison continues until a miscompare is found or until the end of the string is reached.

The BIT input stores the bit number where the next comparison should start. Ordinarily, this is the same as the number where the last miscompare occurred. Because the bit number of the last miscompare is stored in output BN, the same reference can be used for both BIT and BN. The comparison actually begins 1 bit following BIT; therefore, the initial value of BIT should be 1 less first bit to be compared (for example, zero (0) to begin comparison at %I00001). Using the same reference for BIT and BN causes the compare to start at the next bit position after a miscompare; or, if all bits compared successfully upon the next invocation of the instruction, the compare starts at the beginning.

Tip: If you want to start the next comparison at some other location in the string, you can enter different references for BIT and BN. If the value of BIT is a location that is beyond the end of the string, BIT is reset to 0 before starting the next comparison.

The instruction passes power flow whenever it receives power. The other outputs of the instruction depend on the state of the corresponding mask bit.

If all corresponding bits in strings IN1 and IN2 match, the instruction sets the miscompare output MC to 0 and BN to the highest bit number in the input strings. The comparison then stops. On the next invocation of a Masked Compare, it is reset to 0.

If a Miscompare is found, that is when the two bits being compared are not the same, the instruction checks the correspondingly numbered bit in string M (the mask).

If the mask bit is a 1, the comparison continues until it reaches another miscompare or the end of the input strings.

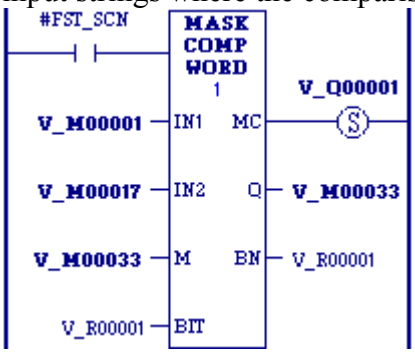
If a miscompare is detected and the corresponding mask bit is a 0, the instruction does the following:

1. Sets the corresponding mask bit in M to 1.
2. Sets the miscompare (MC) output to 1.
3. Updates the output bit string Q to match the new content of mask string M.
4. Sets the bit number output (BN) to the number of the miscompared bit.
5. Stops the comparison.

Examples

Example 1

After the first scan, the Masked Compare Word instruction executes. %M0001 through %M0016 is compared with %M0017 through %M0032. %M0033 through %M0048 contains the mask value. The value in %R0001 determines the bit position in the two input strings where the comparison starts.



Note: MASK_COMP_DWORD would not accept %M references for any of its input operands.

Before the instruction is executed, the contents of the above references are:

(I1) – %M0001 = 6C6Ch =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 = 606Fh =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033 = 000Fh =

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

(BIT/BN) – %R0001 = 0
(MC) – %Q0001 = OFF

The contents of these references after the function block is executed are as follows:

(I1) – %M0001 =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033 =

0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1

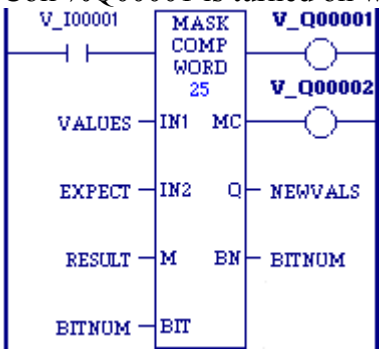
(BIT/BN) – %R0001 = 8

(MC) – %Q0001 = ON

The #FST_SCN contact forces one and only one execution; otherwise, the instruction would repeat with possibly unexpected results.

Example 2

Whenever %I00001 is set, MASK_COMP_WORD compares the bits represented by the reference VALUES against the bits represented by the reference EXPECT. Comparison begins at BITNUM+1. If an unmasked miscompare is detected, the comparison stops. The corresponding bit is set in the mask RESULT. In addition, the output string NEWVALS is updated with the new value of RESULT, and coil %Q00002 is turned on. Coil %Q00001 is turned on whenever MASK_COMP_WORD receives power flow.



CPU Support

MASK_COMP_DWORD and MASK_COMP_WORD are supported for PACSystems CPUs, VersaMax CPUs, Series 90-30 Version 4.40 or later CPUs, and Series 90-70 CPUs.

MASK_COMP_DWORD Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

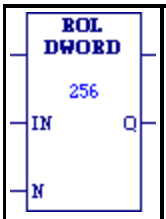
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of DWORDs in either compared string. 1 ≤ Length ≤ 2,048.
IN1	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The first DWORD in the first string of DWORDs to compare
IN2	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The first DWORD in the second string of DWORDs to compare
M	DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The bit string mask containing the ongoing status of the compare
BIT	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable **Note:** %S is not supported here on a Series 90-70.	BIT+1=the bit number where the next comparison starts
Q	DWORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The output copy of the compare mask bit string
BN	WORD variable	I, Q, M, T, S, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable. **Note:** %S is not supported here on a Series 90-70.	The number of the bit where the latest miscompare occurred
MC	Power flow		Can be used to determine if a miscompare has occurred.

MASK_COMP_WORD Operands

Note: ((PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs in either compared string. 1 ≤ Length ≤ 4,096.
IN1	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD in the first string of WORDs to compare
IN2	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD in the second string of WORDs to compare
M	WORD variable	I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The bit string mask containing the ongoing status of the compare
BIT	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable **Note:** %S is not supported here on a Series 90-70.	BIT+1=the bit number where the next comparison starts
Q	WORD variable	I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	The output copy of the compare mask bit string
BN	WORD variable	I, Q, M, T, S, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable. **Note:** %S is not supported here on a Series 90-70.	The number of the bit where the latest miscompare occurred
MC	Power flow		Can be used to determine if a miscompare has occurred.

Rotate Bits



Mnemonics:
ROL_DWORD
ROL_WORD

ROR_DWORD
ROR_WORD

Operation

When receiving power flow, the Rotate Bits Right (ROR_DWORD and ROR_WORD) and Rotate Bits Left (ROL_DWORD and ROL_WORD) instructions rotate all the bits in a string of WORDs or DWORDs N positions respectively to the right or to the left. When rotation occurs, the specified number of bits is rotated out of the input string respectively to the right or to the left and back into the string on the other side. The result is placed in output string Q. If you want the input string to be rotated, the output parameter Q must use the same memory location as the input parameter IN. The entire rotated string is written on each scan that power is received.
A string length of 1 to 256 words or double words can be selected for either instruction. The conditions for the instructions to pass power to the right depend on the Controller family type.

PACSystems and Series 90-70

The Rotate Bits instructions pass power flow to the right when the number of bits to rotate, N, is greater than or equal to zero, and is less than or equal to the total length of the string.
If the number of bits to rotate is less than zero, or is greater than the total length of the string, the input is copied as is to the output and power is not passed to the right.

Series 90-30 and VersaMax

The Rotate Bits instructions pass power flow to the right when the number of bits to rotate is greater than zero, and is less than or equal to the total length of the string.
If the number of bits to rotate is less than zero, or is greater than the total length of the string, the input is copied as is to the output and power is passed to the right.
If the number of bits to rotate is zero, the input is copied as is to the output and power is not passed to the right.

Operands

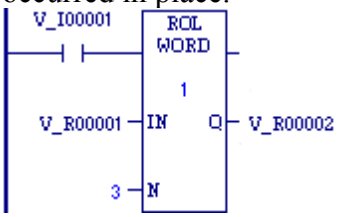
Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

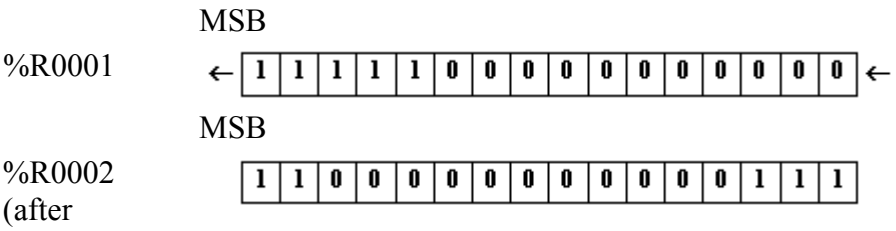
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs or

			DWORDs in the string to rotate. 1 ≤ Length ≤ 256.
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD in the string to rotate
	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
N	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of positions to rotate. 0 ≤ N ≤ (number of bits in the string).
Q	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The first WORD or DWORD of the rotated string
	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

Whenever input V_I0001 is set, the input bit string in location %R0001 is rotated left 3 bits and the result is placed in %R00002. The actual input bit string %R0001 is left unchanged. If the same reference had been used for IN and Q, a rotation would have occurred in place.



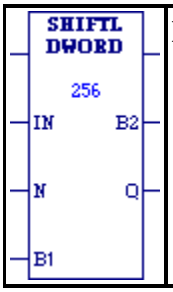


CPU Support

ROR_WORD and ROL_WORD are supported for all GE IP CPUs.

ROR_DWORD and ROL_DWORD are supported for PACSystems CPUs and Series 90-70 CPUs.

Shift Bits

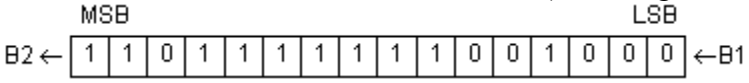


Mnemonics:
SHIFTL_DWORD
SHIFTL_WORD
SHIFTR_DWORD
SHIFTR_WORD

Operation

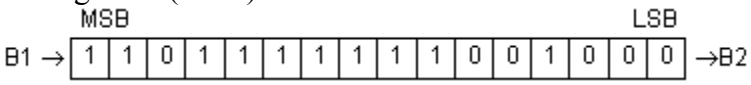
Shift Left

When the Shift Left (SHIFTL_WORD) instruction receives power flow, it shifts all the bits in a word or group of words to the left by a specified number of places. When the shift occurs, the specified number of bits is shifted out of the output string to the left. As bits are shifted out of the high end of the string (Most Significant Bit (MSB)), the same number of bits is shifted in at the low end (Least Significant Bit (LSB)).



Shift Right

When the Shift Right (SHIFTR_WORD) instruction receives power flow, it shifts all the bits in a word or group of words a specified number of places to the right. When the shift occurs, the specified number of bits is shifted out of the output string to the right. As bits are shifted out of the low end of the string (LSB), the same number of bits is shifted in at the high end (MSB).



Shift Left and Shift Right

A string length of 1 to 256 words can be selected for either instruction.
The number of places specified for the shift (N) must be more than zero and less than the number of bits in the string.
If N is out of range, CPUs behave differently:

- On a **PACSystems** or a **Series 90-70** CPU, no shift occurs and no power flow is generated.
- On **other** CPUs, if the number of bits to be shifted (N) is greater than the number of words in the array * 16, the array (Q) is filled with copies of the input bit (B1), and the input bit is copied to the output power flow (B2). If the number of bits to be shifted is zero, then no shifting is performed, the input array is copied into the output array, and input bit (B1) is copied into the power flow.

The bits being shifted into the beginning of the string are specified by means of input parameter B1. If a length greater than 1 has been specified as the number of bits to shift, each bit is filled with the same value (0 or 1). This can be:

- The boolean output of another instruction.
- All 1s. To do this, use the #ALW_ON (always on) system bit (in memory location %S7), as a permissive to input B1.
- All 0s. To do this, use the #ALW_OFF (always off) system bit (in memory location %S8), as a permissive to input B1.

The Shift Bits instruction passes power flow to the right, unless the number of bits specified to shift is zero or is greater than the array size.
Output Q is the shifted copy of the input string. If you want the input string to be shifted, the output parameter Q must use the same memory location as the input parameter IN. The entire shifted string is written on each scan that power is received. Output B2 is the last bit shifted out. For example, if four bits were shifted, B2 would be the fourth bit shifted out.

Operands

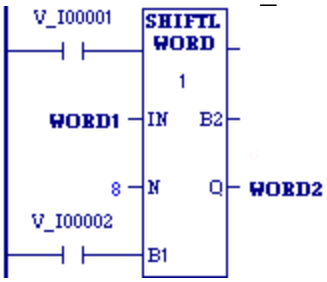
Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of WORDs or DWORDs in the string. 1 ≤ Length ≤ 256.
IN	WORD variable. **Note:** On a PACSystems or Series 90-70, can be a WORD variable or constant.	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD in a string of WORDs or DWORDs to shift
	DWORD variable. **Note:** On a PACSystems or Series 90-70, can be a DWORD variable or constant.	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
N	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of places (bits) to shift the array. 0 < N < [number of bits in the string=(16 * Length)]. If N is out of range, the result differs depending on the CPU type.
B1	Power flow		The bit value to shift into the array
B2	Power flow		(Optional.) The bit value of the last bit shifted out of the array.

Q (Must be the same data type as IN.)	WORD variable	I, Q, M, T, G, SA, SB, SC, R, AI, AQ, W, symbolic, I/O variable	The first WORD or DWORD of the shifted array
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	

Example

Whenever input V_I00001 is set, the bits in the input string that begins at WORD1 are copied to the output bit string that starts at WORD2. WORD2 is left-shifted by 8 bits, as specified by the input N. The resulting open bits at the beginning of the output string are set to the value of V_I0002.



CPU Support

SHIFTL_WORD and SHIFTR_WORD are supported for all GE IP CPUs. SHIFTL_DWORD and SHIFTR_DWORD are supported for PACSystems CPUs and Series 90-70 CPUs.

Coils

Coils are used to control the discrete (BOOL) variables assigned to them. Conditional logic must be used to control the flow of power to a coil. Coils cause action directly. They do not pass power flow to the right. If additional logic should be executed as a result of the coil condition, you can use an internal reference for the coil or a continuation coil/contact combination.

Coil checking

The level of coil checking is set to "Show as error" by default. If you want a coil conflict to result in a warning instead of this error, or if you want no warning at all, edit the following Controller option: **Multiple Coil Use Warning**.
The "Show as warning" option enables you to use any coil reference with multiple Coils, Set Coils, and Reset Coils, but you will be warned at validation time every time you do so. With both the "Show as warning" and the "no warning" options, a variable can be set ON by either a Set Coil or a normal Coil and can be set OFF by a Reset Coil or by a normal Coil.

Graphical representation

The graphical representation of coil instructions in logic depends on the retentive state of the BOOL variables assigned to them.

- **Retentive** BOOL variables. The assigned BOOL variable's value is saved when power is cycled and restored when the Controller goes from Stop to Run mode.
- **Non-retentive** BOOL variables. The assigned BOOL variable's value is set to zero when power is cycled or the Controller goes from Stop to Run mode. %T is always non-retentive. Only %M and %Q can be non-retentive or retentive.

Instruction	*Mnemonic*	*Representation of coil in logic when it has a **non-retentive** variable*	*Representation of coil in logic when it has a **retentive** variable*
Coil (normally open)	COIL		
Continuation Coil	CONTCOIL	Cannot be assigned a variable. Is non-retentive by nature. 	Cannot be assigned a variable
Negated Coil	NCCOIL		
Set, Reset Coil	SETCOIL		
	RESETCOIL		
Transitional Coils	NEGCOIL	Is represented the same,  , whether the variable is retentive or not	
	NTCOIL	Is represented the same,  , whether the variable is retentive or not	

	POSCOIL	Is represented the same,  , whether the variable is retentive or not
	PTCOIL	Is represented the same,  , whether the variable is retentive or not

Notes

- A continuation coil does not use an internal variable. It must be followed by a continuation contact at the beginning of any rung following the continuation coil.
- Coils are always located at the rightmost position of a line of logic. In a Series 90-70 CPU, this must be the 10th column.
- (All CPUs except Series 90-70 CPUs.) You can force coils to display no further left than the "coil justification column."
- (Series 90-70 CPUs only.) A rung may contain a maximum of eight coils.

Coil

A retentive variable is assigned to the coil

A non-retentive variable is assigned to the coil

Operation

When a COIL receives power flow, it sets its associated BOOL variable to ON (1). When it receives no power flow, it sets the associated BOOL variable to OFF (0). COIL can be assigned a retentive variable or a non-retentive variable.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, SA, SB, SC, G, symbolic, I/O variable	The variable associated with COIL

CPU Support

COIL is supported for all GE IP CPUs.

Continuation Coil

Operation

A continuation coil instructs the Controller to continue the present rung's LD logic at the continuation contact on a following rung if the following conditions are met:

- The continuation coil is the last instruction in its rung. On a Series 90-70, it must be on the tenth column.
- The continuation contact is the first instruction of its rung.

The flow state of the continuation coil is passed to the continuation contact. A continuation coil has no associated variable.

Notes

- In a PACSystems, you can use a continuation contact even if there is no preceding continuation coil.
- In a Series 90-70, Series 90-30, or VersaMax, if the flow of logic does not execute a continuation coil before it executes a continuation contact, the state of the continuation contact is no flow.
- The continuation coil and the continuation contact do not use parameters and do not have associated variables.
- There can be only one continuation coil per rung.
- The continuation coil and continuation contact are not re-entrant and must not be used in different interrupt blocks.
- You can have multiple rungs with continuation contacts after a single continuation coil.
- You can have multiple rungs with continuation coils before one rung with a continuation contact.

CPU Support

CONTCOIL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-30 CPU, and Series 90-70 Version 4.00 or later CPUs.

Negated Coil

A retentive variable is assigned to the negated coil

A non-retentive variable is assigned to the negated coil

Operation

When it does *not* receive power flow, a negated coil (NCCOIL) sets a discrete reference ON. NCCOIL can be assigned a retentive variable or a non-retentive variable.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, SA, SB, SC, G, symbolic, I/O variable	The variable associated with NCCOIL

CPU Support

NCCOIL is supported for all GE IP CPUs.

Set Coil and Reset Coil

A retentive variable is assigned to the Set Coil and Reset Coil

A non-retentive variable is assigned to the Set Coil and Reset Coil

Operation

You can use SETCOIL and RESETCOIL to keep ("latch") the state of a variable either ON or OFF. You can assign the same BOOL variable to a SETCOIL and a RESETCOIL without any restrictions.
When a SETCOIL receives power flow, it sets its associated BOOL variable to ON. When it receives no power flow, it leaves the BOOL variable alone. SETCOIL is the only kind of coil that cannot set its associated BOOL variable to OFF. To set the associated BOOL variable to OFF, it is recommended to use a RESETCOIL.
When a RESETCOIL receives power flow, it sets its associated BOOL variable to OFF. When it receives no power flow, it leaves the BOOL variable alone. RESETCOIL is the only kind of coil that cannot set its associated BOOL variable to ON. To set the associated BOOL variable to ON, it is recommended to use a SETCOIL.
There are other ways to write to the BOOL variable assigned to a SETCOIL or RESETCOIL, for example:

- Using another coil
- Using a MOVE_BOOL instruction to move a value of 0 or 1 to the BOOL variable
- Assigning the output power flow of a relational instruction to the BOOL variable

When both a SETCOIL and a RESETCOIL are associated with the same variable, the last-solved coil of the pair takes precedence.
SETCOIL and RESETCOIL can be assigned a retentive variable or a non-retentive variable.

Notes

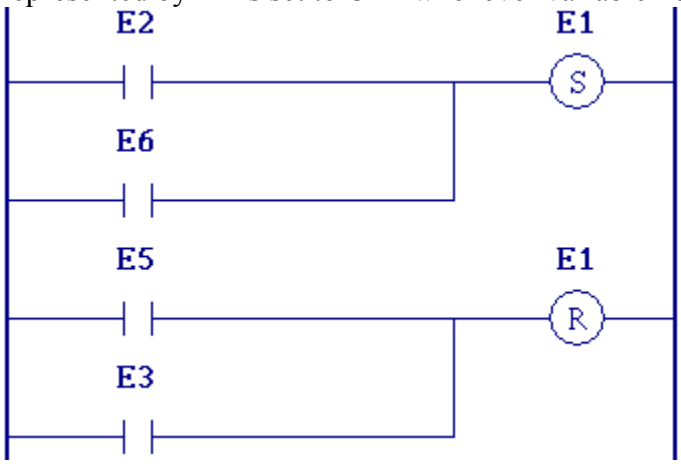
- Do not associate transition contacts with variables used with SETCOIL or RESETCOIL.
- (Series 90-70 only.) SETCOIL and RESETCOIL write an undefined result to the transition bit for their associated variable.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, SA, SB, SC, G, symbolic, I/O variable	The variable associated with SETCOIL or RESETCOIL

Example

The coil represented by E1 is turned ON whenever variable E2 or E6 is ON. The coil represented by E1 is set to OFF whenever variable E5 or E3 is ON.



CPU Support

SETCOIL and RESETCOIL are supported for all GE IP CPUs.

Transition Coils - POSCOIL and NEGCOIL

Operation

 *Positive Transition Coil* (POSCOIL)	 *Negative Transition Coil* (NEGCOIL)
Warning: Do not override a transitional coil by putting force on its reference bit. If a transitional coil is overridden, either transition coil has no effect on the bit, and if the override is then removed, the coil may be set to ON for one sweep. This can cause unexpected consequences in the Controller LD logic and in field devices attached to the Controller.	
If there is *no* force on the bit and if: ▪ the current value of the *transition* bit is **OFF**, ▪ the current value of the power flow input is **ON**, then the Positive Transition Coil sets its *reference* bit to ON until the coil is executed again. When the coil is executed again, the reference bit is set to OFF. **Note:** When the Positive Transition Coil sets its *reference* bit to ON, it also sets its *transition* bit to **ON**. The next time the Positive Transition coil executes, it finds the transition bit set to **ON** and it then sets its reference bit to OFF.	If there is *no* force on the bit and if: ▪ the current value of the *transition* bit is **ON**, ▪ the current value of the power flow input is **OFF**, then the Negative Transition Coil sets its *reference* bit to ON until the coil is executed again. When the coil is executed again, the reference bit is set to OFF. **Note:** When the Negative Transition Coil sets its *reference* bit to ON, it also sets its *transition* bit to **OFF**. The next time the Negative Transition coil executes, it finds the transition bit set to **OFF** and it then sets its reference bit to OFF.

Notes

- You can use either type of transitional coil with references from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC).
- Do not use a transition contact on a transition coil because the coil uses the transition bit to store the power flow value into the coil.
- A rung that ends with a transition coil cannot have another branch that ends with a coil, not even another transition coil.

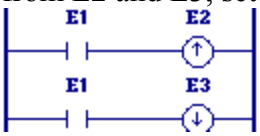
Warning: Do not write from external devices (for example, PCM, programmer, or ADS) to the reference bit of a transitional coil, since that would destroy the coil's one-shot nature and the coil may not behave as described.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable	I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable	The variable associated with the transition coil
	Bit reference in BOOL variable	I, Q, M, T, G, S, SA, SB, SC	

Example

When reference E1 is set from OFF to ON, coils E2 and E3 receive power flow, setting E2 to ON for one logic scan. When E1 is set from ON to OFF, power flow is removed from E2 and E3, setting coil E3 to ON for one scan.



CPU Support

POSCOIL and NEGCOIL are supported for all GE IP CPUs.

Transition Coils - PTCOIL and NTCOIL

Operation

 *Positive Transition Coil* (PTCOIL)	 *Negative Transition Coil* (NTCOIL)
Warning: Do not override a transitional coil by putting force on its reference bit. If a transitional coil is overridden, a transition coil has no effect on the bit, and if the override is then removed, the coil may be set to on for one sweep. This can cause unexpected consequences in the Controller LD logic and in field devices attached to the Controller.	
If there is *no* force on the reference bit, PTCOIL is set to ON when the following conditions are simultaneously met: ▪ The input power flow to PTCOIL is set to ON. ▪ The instance data is set to OFF.	If there is *no* force on the reference bit, NTCOIL is set to ON when the following conditions are simultaneously met: ▪ The input power flow to NTCOIL is set to OFF. ▪ The instance data is set to ON.
The **instance data** is the value of the input power flow the last time the instance of the PTCOIL or NTCOIL was executed.	

Notes

- As soon as a PTCOIL or NTCOIL is set to ON or OFF, it updates its instance data.
- Multiple instances of PTCOIL and/or NTCOIL can be associated with the same BOOL variable, but the instance data of each instance of the PTCOIL or NTCOIL associated with the BOOL variable is unique, that is, it is tracked independently.
- Instance data is non-retentive, that is, it is set to OFF when the CPU transitions from stop to run.
- You can use either type of transitional coil with reference addresses from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC).
- A rung that ends with a transition coil cannot have another branch that ends with a coil, not even another transition coil.
- Instance data uses symbolic discrete memory.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOL_V	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable	The variable associated with PTCOIL or NTCOIL

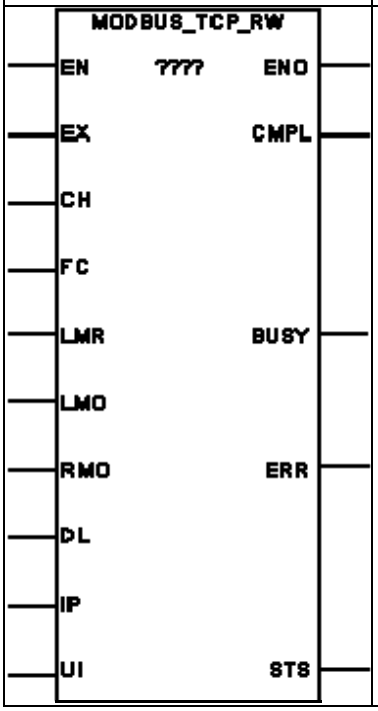
CPU Support

PTCOIL and NTCOIL are supported for PACSystems CPUs.

LD Communication

Instruction	*Description*
MODBUS_TCP_RW	Facilitates communications between a Modbus/TCP server device and a 20-pt, 40-pt, or 64-pt VersaMax Micro CPU with firmware version 4.00 or later

MODBUS_TCP_RW

LD	IL
	MODBUS_TCP_RW(INST, EX, CH, FC, LMR, LMO, RMO, DL, IP, UI, CMPL, BUSY, ERR, STS)

Operation

MODBUS_TCP_RW facilitates communications between a Modbus/TCP server device and a 20-point, 40-point, or 64-point VersaMax Micro CPU with firmware version 3.83 or later.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
???? (LD), INST (IL)	DWORD variable	R	Control block. The reference address of the DWORD containing the function block instance data. **Cautions** ▪ Do not write to the control block by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
EN (LD only)	Power flow		Enable. When set to On, MODBUS_TCP_RW executes. When set to Off, MODBUS_TCP_RW does not execute.
EX	Power flow (LD), BOOL variable (IL)	I, Q, M, G, T	Execute. A low to high transition starts command processing.
CH	WORD constant		The channel number on which the current request needs to be processed. Valid entry: 1. Other values are caught as validation errors.
FC	WORD variable or constant	AI, AQ, R	Function code. Input word. Valid range: 1 through 7; 15 or 16.
LMR	WORD variable or constant	AI, AQ, R	Local PLC Reference memory Type. Valid entries: 8, 18, 72.
LMO	WORD variable or constant	AI, AQ, R	Local PLC Reference Memory Address. For a read operation, this memory is the destination. For a write operation, this memory is the source (Modbus/TCP Client device's location).
RMO	WORD variable or constant	AI, AQ, R	Remote PLC Reference Memory Address. For a read operation, this memory is the source. For a write operation, this memory is the destination (Modbus/TCP Server device's location).
DL	WORD variable or constant	AI, AQ, R	Data Length. Length of data to be read.
IP	WORD variable	AI, AQ, R	IP start address. IP Address of the Server device. Starting from this address, four WORDs contain the four octets of the IP.
UI	WORD variable or constant	AI, AQ, R	Unit Identifier. The Modbus/TCP Unit identifier, a special control code used in a Modbus/TCP message. This value is 1 for most Modbus/TCP devices except if an Ethernet to Serial Bridge is used to multidrop to Modbus RTU devices. Valid range: 0 through 255.

Output Operands

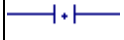
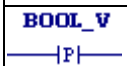
Operand	*Data Type*	*Memory Area*	*Description*
ENO (LD only; optional)	Power flow		Enable Output. Set to On when MODBUS_TCP_RW has executed successfully.
CMPL	BOOL variable	I, Q, M, G, T	Complete. When set to 1, the request has been processed.
BUSY	BOOL variable	I, Q, M, G, T	When set to 1, the request is in progress.
ERR	BOOL variable	I, Q, M, G, T	Error. Used for function block errors.
STS	WORD variable	AI, AQ, R	Status or Error ID. Status from the Ethernet option board, after processing a command. After execute has been triggered, the firmware checks continuously for a match between the processed sequence number and the response sequence number.

CPU Support

MODBUS_TCP_RW is supported for VersaMax Micro CPUs with firmware version 4.00 or later.

Contacts

A contact is used to monitor the state of a reference address. Whether the contact passes power flow depends on the state or status of the reference address being monitored and on the contact type. A reference address is ON if its state is 1; it is OFF if its state is 0.

Contact	*Display*	*Mnemonic*	*Contact Passes Power to Right...*
Continuation Contact		CONTCON	if the preceding continuation coil is set ON
Fault Contact (PACSystems and Series 90-70 only)		FAULT	if its associated BOOL or WORD variable has a point fault
High Alarm Contact (PACSystems and Series 90-70 only)		HIALR	if the high alarm bit associated with the analog (WORD) reference is ON
Low Alarm Contact (PACSystems and Series 90-70 only)		LOALR	if the low alarm bit associated with the analog (WORD) reference is ON
No Fault Contact (PACSystems and Series 90-70 only)		NOFLT	if its associated BOOL or WORD variable does not have a point fault
Normally Closed Contact		NCCON	if associated BOOL variable is OFF
Normally Open Contact		NOCON	if associated BOOL variable is ON
Transition Contacts (PACSystems and Series 90-70 only)		NEGCON	(negative transition contact) if BOOL reference transitions from ON to OFF
		NTCON (PACSystems only)	(negative transition contact) if BOOL reference transitions from ON to OFF
		POSCON	(positive transition contact) if BOOL reference transitions from OFF to ON
		PTCON (PACSystems only)	(positive transition contact) if BOOL reference transitions from OFF to ON

Continuation Contact

CPU Support

Operation

A continuation contact continues the LD logic from any previous rung if the following conditions are met:

- The continuation coil is the last instruction in its rung.
- The continuation contact is the first instruction of the current rung.

The flow state of the continuation contact is the same as the preceding continuation coil. A continuation contact has no associated variable.

Notes

- In a PACSystems, you can use a continuation contact even if there is no preceding continuation coil.
- In a Series 90-70, Series 90-30, or VersaMax, if the flow of logic does not execute a continuation coil before it executes a continuation contact, the state of the continuation contact is no flow.
- The state of the continuation contact is cleared when the Controller transitions from Stop to Run, and there will be no flow unless the transition coil has been set since going to Run mode.
- The continuation coil and the continuation contact do not use parameters and do not have associated variables.
- There can be only one continuation contact per rung.
- The continuation coil and continuation contact are not re-entrant and must not be used in different interrupt blocks.
- You can have multiple rungs with continuation contacts after a single continuation coil.
- You can have multiple rungs with continuation coils before one rung with a continuation contact.

CPU Support

CONTCON is supported for PACSystems CPUs, VersaMax CPUs, Series 90-30 CPUs, and Series 90-70 Version 4.00 or later CPUs.

Fault Contact

Operation

A Fault contact (FAULT) detects faults in discrete or analog reference addresses, or locates faults (rack, slot, bus, module).

- To guarantee correct indication of module status, use the reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts.
- To locate a fault, use the rack, slot, bus, module fault locating system variable with a FAULT/NOFLT contact.

Note: The fault indication of a given module is cleared when the associated fault is cleared from the fault table.

- For I/O point fault reporting, you must configure your Hardware Configuration (HWC) to enable the Controller point faults.

FAULT passes power flow if its associated variable or location has a point fault.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BWVAR	BOOL or WORD variable	I, Q, AI, AQ, I/O variable	The variable associated with the FAULT contact
	BOOL element of UDT variable (not of UDT parameter in UDFB or parameterized block)	I/O variable	
	Fault locating system variable		

CPU Support

FAULT is supported for PACSystems CPUs and Series 90-70 CPUs.

High Alarm Contact

Operation

The high alarm contact (HIALR) is used to detect a high alarm associated with an analog reference. Use of this contact and the low alarm contact must be enabled during CPU configuration.
A high alarm contact passes power flow if the high alarm bit associated with the analog reference is ON.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
WORDV	WORD variable	AI, AQ, I/O variable	The variable associated with the HIALR contact

CPU Support

HIALR is supported for PACSystems CPUs and Series 90-70 CPUs.

Low Alarm Contact

Operation

The low alarm contact (LOALR) detects a low alarm associated with an analog reference. Use of this contact must be enabled during CPU configuration.
A low alarm contact passes power flow if the low alarm bit associated with the analog reference is ON.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
WORDV	WORD variable	AI, AQ, I/O variable	The variable associated with the LOALR contact

CPU Support

LOALR is supported for PACSystems CPUs and Series 90-70 CPUs.

No Fault Contact

Operation

A No Fault (NOFLT) contact detects faults in discrete or analog reference addresses, or locates faults (rack, slot, bus, module).

- To guarantee correct indication of module status, use the reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts.
- To locate a fault, use the rack, slot, bus, module fault locating system variable with a FAULT/NOFLT contact.

  Note: The fault indication of a given module is cleared when the associated fault is cleared from the fault table.
- For I/O point fault reporting, you must configure your Hardware Configuration to enable the Controller point faults.

NOFLT passes power flow if its associated variable or location does not have a point fault.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BWVAR	BOOL or WORD variable	I, Q, AI, AQ, I/O variable	The variable associated with the NOFLT contact
	Fault locating system variable		

CPU Support

NOFLT is supported for PACSystems CPUs and Series 90-70 CPUs.

Normally Closed Contact

Operation

A normally closed contact (NCCON) acts as a switch that passes power flow if the BOOLV operand is OFF (false, 0).

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, S, SA, SB, SC, G, symbolic, I/O variable	The variable associated with the NCCON contact. If BOOLV is ON, the normally closed contact does not pass power flow. If BOOLV is OFF, the contact passes power flow.

Note: If you use a WORD in discrete memory on a NCCON, the state of the contact is determined by the starting bit of the WORD value.

CPU Support

NCCON is supported for all GE IP CPUs.

Normally Open Contact

Operation

A normally open contact (NOCON) acts as a switch that passes power flow to the right if the BOOLV operand is ON (true, 1).

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, S, SA, SB, SC, G, symbolic, I/O variable	The variable associated with the NOCON contact. If BOOLV is ON, the normally open contact passes power flow to the right. If BOOLV is OFF, the contact does not pass power flow.

Note: If you use a WORD in discrete memory on a NOCON, the state of the contact is determined by the starting bit of the WORD value.

CPU Support

NOCON is supported for all GE IP CPUs.

Transition Contacts - POSCON and NEGCON

Operation

 *Positive Transition Contact* POSCON	 *Negative Transition Contact* NEGCON
A POSCON or NEGCON transition contact starts passing power flow when input power flow to the contact is set to ON and its transition bit is set to ON.	
The POSCON's transition bit is set to ON when the variable associated with the POSCON transitions from **OFF to ON**.	The NEGCON's transition bit is set to ON when the variable associated with the NEGCON transitions from **ON to OFF**.
The POSCON or NEGCON transition contact continues to pass power as long as input power flow to the contact is set ON and its transition bit remains is set ON.	
The transition bit is set to OFF when the associated variable is written to, no matter whether the value written to it is ON or OFF.	
As soon as the transition bit is set to OFF, the POSCON or NEGCON transition contact stops passing power. As long as the transition bit remains set to OFF, the POSCON or NEGCON does not pass power.	
Depending on the logic flow, the writes to the POSCON or NEGCON's associated variable may occur at different intervals with regards to the Controller scan: ▪ The writes may occur multiple times during a Controller scan, resulting in the transition bit being set to ON for only a portion of the scan. ▪ The writes may occur several Controller scans apart, resulting in the transition bit being set to ON for more than one scan. ▪ The writes may occur once per scan, for example if the POSCON or NEGCON's associated variable is a %I input bit.	

Warning: Do not use POSCON or NEGCON transition contacts for variables used with transition coils (also called one-shots) or SET and RESET coils.

The transition bit for a reference point is affected every time that point is written to.

- It is set for a POSCON when the point transitions from OFF to ON.
- It is set for a NEGCON when the point transitions from ON to OFF.
- It is cleared for either POSCON or NEGCON when the point transitions in the opposite direction or when the state after the write is the same as the state before the write, that is, set from ON to ON or is set from OFF to OFF.

The source of the write is immaterial; it can be an output coil, an instruction output, the input scan, an input interrupt, a PCM SYSWRITE, a data change from the logic, or external communications. When the point is written, the transition bit is immediately affected. Transition bits are not changed by the scan itself; only by a write to the reference point. A write must be made to a reference in order to clear the transition bit, or it will appear to be stuck. Nothing is done automatically per scan to clear transition bits.

Overrides do not protect transition bits. If a write occurs to an overridden point, the transition bit is cleared. For example, the transition bit of an overridden input point is cleared when the input is scanned.

Operational differences between Series 90-70 and PACSystems RX7i

The following table describes operational differences between Series 90-70 and PACSystems RX7i with regards to POSCON and NEGCON.

	Series 90-70	*PACSystems RX7i*
Effect of SETCOIL and RESETCOIL on transition bits	**Warning:** Do not use these contacts with variables that are also used with transition coils or with SETCOIL or RESETCOIL. If a SETCOIL or RESETCOIL receives positive power flow, it writes an **undefined** result to the transition bit of the associated variable.	**Warning:** Do not use these contacts with variables that are also used with transition coils or with SETCOIL or RESETCOIL. If a SETCOIL or RESETCOIL receives positive power flow and its associated variable is not overridden, the SETCOIL or RESETCOIL writes the expected result to the transition bit for the associated variable (that is, the transition bit is set if the variable's value is set from ON to OFF or is set from OFF to ON, and is cleared when its value remains the same). However, if the SETCOIL or RESETCOIL receives positive power flow and if its associated variable is overridden, the SETCOIL or RESETCOIL causes the transition bit to be cleared.
Effect of a transition to run mode on a transition bit	▪ For retentive memory areas (%I, %S - %SC, %G), values and transition bits are not changed. ▪ For non-retentive memory areas (%T), all values and transition bits are cleared to 0. ▪ For %Q and %M memory areas, which are selectively retentive, variables that are not retentive and not overridden will have their value set to 0 and their transition bit set or cleared based on whether the 0 value is or is not a change from their previous value. Variables that are retentive or are overridden will retain their values and will have their transition bit set to 0.	Same, with the following addition: ▪ For discrete symbolic or I/O variable memory, which is selectively retentive, variables that are non-retentive and not overridden will have values and transitions cleared to 0. Variables that are non-retentive and overridden will retain their values and transition bits. Variables that are retentive will retain their values and transition bits.

Operands

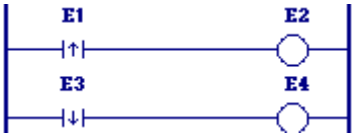
Operand	*Data Type*	*Memory Area*	*Description*
BOOLV	BOOL variable	I, Q, M, T, G, S, SA, SB, SC, symbolic, I/O variable	The variable associated with the transition contact
	Bit reference in BOOL variable	I, Q, M, T, G, S, SA, SB, SC	

Note: POSCON and NEGCON do not support BOOL elements of UDT parameters in parameterized blocks or UDFBs.

Examples

Example 1

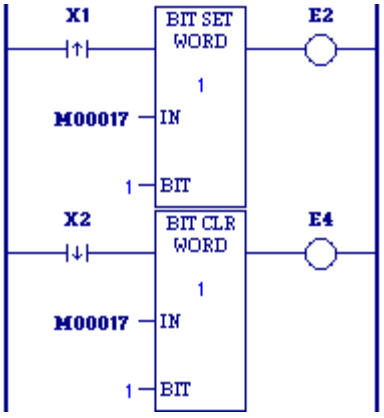
Coil E2 is set to ON for one logic sweep when element E1 transitions from OFF to ON. Coil E4 is set to ON for one sweep when element E3 transitions from ON to OFF.



Example 2

Bit %M00017 is set by a BIT_SET instruction and then cleared by a BIT_CLR instruction. The positive transition contact X1 activates the BIT_SET, and the negative transition X2 activates the BIT_CLR.

The positive transition associated with bit %M00017 will be on until %M00017 is reset by the BIT_CLR instruction. This occurs because the bit is only written when contact X1 is set from OFF to ON. Similarly, the negative transition associated with bit %M00017 is set to ON until %M00017 is set by the BIT_SET instruction.



CPU Support

POSCON and NEGCON are supported for PACSystems CPUs and for Series 90-70 CPUs.

Transition Contacts - PTCON and NTCON

Operation

 *Positive Transition Contact* PTCON	 *Negative Transition Contact* NTCON
PTCON passes power flow to the right only when the following conditions are simultaneously met: ▪ The input power flow to PTCON is set to ON. ▪ The BOOL variable associated with PTCON is set to ON (1) when this instance of PTCON is executed. ▪ The instance data is set to OFF (0).	NTCON passes power flow to the right only when both of the following conditions are met: ▪ The input power flow to NTCON is set to ON. ▪ The BOOL variable associated with NTCON is set to OFF (0) when this instance of NTCON is executed. ▪ The instance data is set to ON (1).
The **instance data** is the value of the BOOL variable associated with this instance of PTCON or NTCON the last time this instance of PTCON or NTCON was executed.	

Notes

- As soon as a PTCON or NTCON is set to ON or OFF, it updates its instance data.
- Multiple instances of PTCON and/or NTCON can be associated with the same BOOL variable, but the instance data of each instance of the PTCON or NTCON associated with the BOOL variable is unique, that is, it is tracked independently.
- Instance data is non-retentive, that is, it is cleared to OFF when the CPU transitions from stop to run. As a result, the first time a PTCON executes with its input power flow set to ON and its associated BOOL variable also set to ON, it passes power flow to the right.
- Instance data uses symbolic discrete memory.

Caution: The instance data of a given PTCON or NTCON is changed only once per CPU scan. Therefore, using a PTCON or NTCON in a block that can be called multiple times per scan may have adverse effects on all calls after the first one because the PTCON or NTCON cannot detect the transition on the second and subsequent calls. This is particularly true when using a PTCON or NTCON in a parameterized block or user-defined function block with a parameter or member. In these cases, we recommend using R_TRIG or F_TRIG instead.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
BOOL_V	BOOL variable, bit reference in non-BOOL variable	I, Q, M, T, G, S, SA, SB, SC, symbolic, I/O variable	The variable associated with the PTCON or NTCON contact

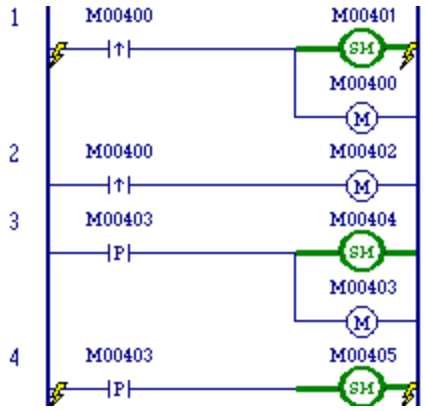
CPU Support

PTCON and NTCON are supported for PACSystems CPUs.

POSCON vs. PTCON — NEGCON vs. NTCON

NEGCON vs. NTCON

Both POSCON and PTCON are positive transition contacts. The following screenshot taken while online illustrates the differences between POSCON and PTCON:



Note: In the above example, each variable name indicates precisely where the variable is mapped to. For example, M00400 is mapped to %M400.

If the BOOL variable at %M400 has just been set to ON, its associated transition bit is set to ON. The POSCON on rung 1 of this LD block receives power from the left and looks up the transition bit, which is ON, and the POSCON turns ON. This sets the set coil at %M401 and turns the other coil ON, which writes the value ON to %M400. This write to %M400 causes the transition bit associated with %M400 to turn OFF. As a result, in rung 2, when the POSCON receives power from the left and looks up the transition bit that is OFF, it remains OFF. The next time the POSCON on rung 1 is executed (it could be within the same sweep, one sweep later, or multiple sweeps later), it looks up the transition bit, which is OFF, and it turns OFF, which turns %M400 OFF. This leaves the transition bit associated with %M400 OFF. The POSCON on rung 2 will never see the transition bit ON and will never pass power to the right.

If the BOOL variable at %M403 has just turned ON, there is no transition bit involved as far as PTCON is concerned. The PTCON on rung 3 receives power from the left and looks up its own instance data, which holds the value that %M403 had the last time this PTCON was executed, which was OFF. The PTCON detects the transition from OFF to ON, and it turns ON, setting the set coil at %M404 and turning the other coil ON, which writes ON to %M403. Writing to the coil at %M403 has no effect on the instance data of the PTCON in rung 4. When the PTCON in rung 4 receives power from the left, it looks up its own instance data, which holds the value that %M403 had the last time this PTCON executed, which was OFF, and it looks up %M403, which is currently ON. To the PTCON, this is a transition from OFF to ON, and, as a result, the PTCON turns ON and the set coil at %M405 also turns ON.

The main difference between POSCON and PTCON is how the transition is tracked. For POSCON, the transition is tracked at the level of the associated BOOL variable. Any write to the associated variable affects its transition bit and this affects the behavior of all the POSCONs associated with this variable.

For PTCON, the transition is tracked by comparing the current value of the associated BOOL variable with the value that this variable had the last time this instance of the PTCON was executed. Writing to the associated BOOL variable does not necessarily affect the behavior of all the PTCONs associated with this BOOL variable.

NEGCON vs. NTCON

The difference between NEGCON and NTCON is the same as that between POSCON and PTCON. The negative transition for a NEGCON is tracked at the level of the associated BOOL variable, while the negative transition for an NTCON is tracked by comparing the current value of the associated BOOL variable with the value this variable had the last time this instance of the NTCON was executed.

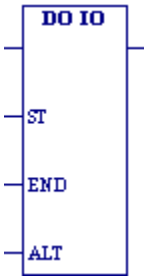
Control instructions

The control instructions limit logic execution and change the way the CPU executes the application program.

instruction	*Mnemonic*	*Description*
Do I/O	DO_IO	For one scan, immediately services a specified range of inputs or outputs. (All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O instruction. Partial I/O module updates are not performed.) Optionally, a copy of the scanned I/O can be placed in internal memory, rather than at the real input points.
DRUM	DRUM	Drum sequencer. Provides predefined ON/OFF patterns to a set of 16 discrete outputs in the manner of a mechanical drum sequencer.
Edge Detector	R_TRIG F_TRIG	The *Rising Edge Trigger* R_TRIG and *Falling Edge Trigger* F_TRIG detect the changing state of a Boolean signal. The outputs of both function blocks produce a single pulse when an edge is detected. Execution of the function block defines the pulse.
For Loop (PACSystems and Series 90-70 only)	FOR_LOOP EXIT_FOR END_FOR	For loop. Repeats the logic between the FOR_LOOP instruction and END_FOR instruction a specified number of times or until EXIT_FOR is encountered.
Mask IO Interrupt	MASK_IO_INTR	Requests special Controller service 17. Use MASK_IO_INTR with I/O variables to mask or unmask an interrupt from an input board.
Proportional Integral Derivative Control	PID_ISA PID_IND	Provides two PID (Proportional/Integral/Derivative) closed-loop control algorithms: ▪ Standard ISA PID algorithm (PID_ISA). ▪ Independent term algorithm (PID_IND).
Scan Set IO	SCAN_SET_IO	Scans the I/O of a specified scan set.
Sequential Event Recorder	SER	Collects a series of samples. A control block for each instance of SER contains user-supplied configuration of function block execution, sample configuration, and operation parameters.
Service Request	SVC_REQ	Requests a special Controller service.
Suspend IO	SUS_IO	Suspends for one sweep all normal I/O updates.

(PACSystems and 90-70 only)		except those specified by DO I/O instructions.
Suspend IO Interrupt	SUSP_IO_INTR	Requests special Controller service 32. Use SUSP_IO_INTR with I/O variables to suspend a set of I/O interrupts and to cause occurrences of these interrupts to be queued until these interrupts are resumed.
Switch Position (PACSystems firmware version 2.00 and later)	SWITCH_POS	Allows logic to read current switch position. Two values are returned: the switch position and the mode (configured use) of the switch.

Do I/O



Operation

When the DO I/O (DO_IO) instruction receives power flow, it updates inputs or outputs for one scan while the instruction is running. You can also use DO_IO to update selected I/O during the instruction in addition to the normal I/O scan.

Note: (PACSystems or Series 90-70 CPUs.) You can use DO_IO in conjunction with a SUS_IO instruction, which stops the normal I/O scan.

If input references are specified, DO_IO allows the most recent values of inputs to be obtained for instruction logic. If output references are specified, DO I/O updates outputs based on the most current values stored in I/O memory. I/O is serviced in increments of entire I/O modules; the Controller adjusts the references, if necessary, while DO_IO executes. DO_IO does not scan I/O modules that are not configured.

DO_IO continues to execute until all inputs in the selected range have reported or all outputs have been serviced on the I/O modules. Program execution then returns to the instruction that follows the DO_IO.

If the range of references includes an option module (HSC, APM, and so on), all the input data (%I and %AI) or all the output data (%Q and %AQ) for that module are scanned. The ALT parameter is ignored while scanning option modules.

Note: An Enhanced Genius Communication Module (GCM) can only be used in the reference range for all VersaMax CPUs and for the Series 90-30 firmware version 4.30 and later for IC693CPU331 and later.

DO_IO passes power to the right whenever it receives power unless:

- Not all references of the type specified are present within the selected range.
- The CPU is not able to properly handle the temporary list of I/O created by the instruction.
- The range specified includes I/O modules that are associated with a "Loss of I/O" fault.

Warning: If DO_IO is used with timed or I/O interrupts, transitional contacts associated with scanned inputs may not operate as expected.

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) A DO_IO instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

Do I/O for Inputs

When DO_IO receives power flow and input references are specified, the Controller scans input points from the starting reference (ST) to the ending reference (END). If a reference is specified for ALT, a copy of the new input values is placed in memory beginning at that reference, and the real input values are not updated.
If no reference is specified for ALT, the real input values are updated. This allows inputs to be scanned one or more times during the instruction execution portion of the CPU scan.

Do I/O for Outputs

When DO_IO receives power flow and output references are specified, the Controller writes to the output points. If no value is specified in ALT, the range of outputs written to the output modules is specified by the starting reference (ST) and the ending reference (END). If outputs should be written to the output points from internal memory other than %Q or %AQ, the beginning reference is specified for ALT and the end reference is automatically calculated from the length of the END-ST range.

Do I/O to One Module (Enhanced Do I/O)

You can use DO_IO on a single discrete input or discrete output module located in the main Controller.

Note: This is available only on VersaMax CPUs or Series 90-30 firmware version 4.30 and later on IC693CPU331 and later, and only to be used on a single discrete input or discrete output 8-point, 16-point, or 32-point module located in the main rack.

DO_IO executes much faster when just one module is read or written to, as documented in the following table:

Module		*Normal DO_IO Execution Time*	*Enhanced DO_IO Execution Time*
8-Pt Discrete	Input	224 microseconds	67 microseconds
	Output	208 microseconds	48 microseconds
16-Pt Discrete	Input	224 microseconds	68 microseconds
	Output	211 microseconds	47 microseconds
32-Pt Discrete	Input	247 microseconds	91 microseconds
	Output	226 microseconds	50 microseconds

The module to be read/written is specified in the ALT parameter. For example, a constant value of 2 in this parameter indicates to the CPU that it is to execute the DO_IO instruction for the module in location 2. The ALT parameter must be between 2 and 5 for

a 5-slot rack, and between 2 and 10 for a 10-slot rack. The ST and END references specify the first and last reference the module is configured for.

Note: The only checking done by the enhanced DO_IO instruction is checking the state of the module in the slot specified to see if the module is okay.

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a WORD variable. Restrictions apply.

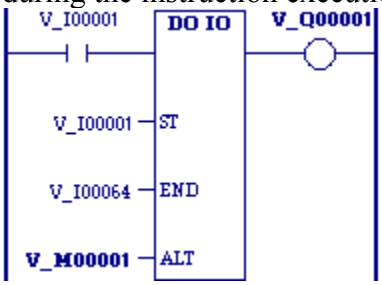
Operand	*Data Type*	*Memory Area*	*Description*
ST	WORD or BOOL variable	I, Q, AI, AQ, R, M, W, I/O variable	The starting address of the set of input or output points or words to be serviced. ST and END must be in the same memory area. **Notes** ▪ If ST and END are mapped to discrete memory, ST must be byte-aligned, that is, its reference address must start at (8n+1), for example, %I01, %Q09, %Q49. ▪ (Series 90-70 and PACSystems.) If ST and END are mapped to analog memory, they can have the same reference address. ▪ If an I/O variable is assigned to the ST parameter, then the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.
END	WORD or BOOL variable	I, Q, AI, AQ, R, M, W, I/O variable	The address of the end bit of input or output points or words to be serviced. Must be in the same memory area. **Notes** ▪ If ST and END are mapped to discrete memory, END's reference address must be 8n, for example, %I08, %Q16. ▪ (Series 90-70 and PACSystems.) If ST and END are mapped to analog memory, they can have the same reference address. ▪ If an I/O variable is assigned to the END parameter, then the same I/O variable must also be assigned to the ST parameter, and the entire module is scanned.
ALT	WORD variable	I, Q, M, T, G, R, AI,	Optional. For an input scan, ALT specifies the address to store scanned input point/word values. For

		AQ, W, symbolic	an output scan, ALT specifies the address to get output point/word values from, to send to the I/O modules. **Note:** For VersaMax CPUs and Series 90-30 IC693CPU331 and later, you can use the ALT operand to enter the slot of a single module in the main rack. When that is done, the DO_IO instruction executes in 80 microseconds instead of the 236 microseconds required when the instruction is programmed without the ALT parameter. No error checking is performed to prevent overlapping reference addresses or module type mismatches.

Examples

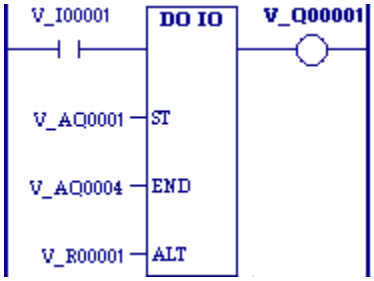
Do I/O for Inputs

When DO_IO receives power flow, the Controller scans references %I0001-64 and %Q0001 is set to ON. A copy of the scanned inputs is placed in internal memory from %M0001-64. Because a reference is specified for ALT, the real inputs are not updated. This allows the current values of inputs to be compared with their values at the beginning of the scan. This form of DO_IO allows input points to be scanned one or more times during the instruction execution portion of the CPU scan.

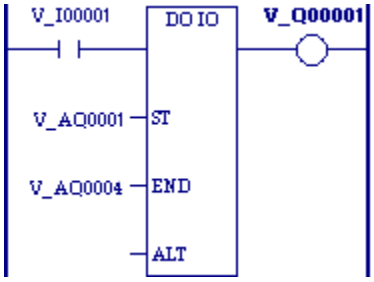


Do I/O For Outputs

Because a reference is entered for ALT, the values at %AQ001-004 are *not* written to output modules. When DO_IO receives power flow, the Controller writes the values from references %R0001-0004 to the analog output modules and %Q0001 is turned on.

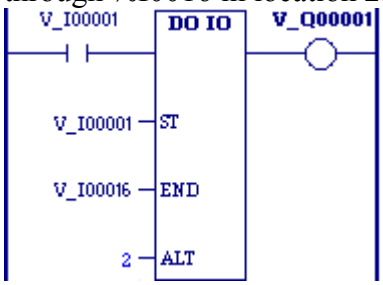


In the following example, because no reference is specified for ALT, the Controller writes values at references %AQ001-004 to analog output channels and turns %Q0001 ON.



Do I/O for One Module

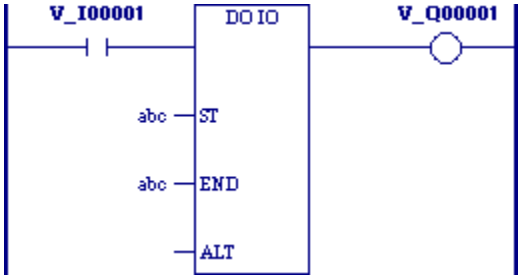
DO_IO is executed only to a 16-point input module which is configured at %I0001 through %I0016 in location 2.



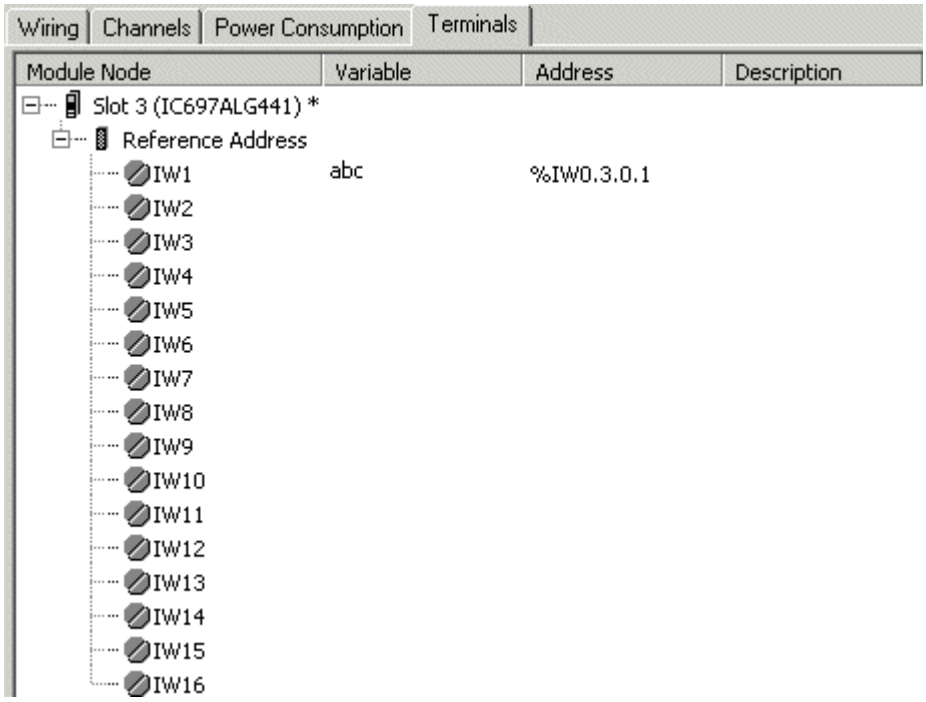
Do I/O and I/O variables

The following example shows the same I/O variable named abc assigned to the ST and END parameters.

Note: If an I/O variable is assigned to the ST parameter, as in this example, then the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.



The I/O variable named abc is also mapped to the IW1 (Input WORD) terminal on the Terminals tab of an IC697ALG441 module.



CPU Support

DO_IO is supported for all GE IP CPUs, but Series 90-30 CPUs versions 330 and earlier had limited functionality.

- An Enhanced Genius Communication Module (GCM) can be used only in the reference range for VersaMax CPUs or Series 90-30 firmware version 4.30 and later for IC693CPU331 and later.
- For VersaMax CPUs and Series 90-30 firmware version 4.30 and later for IC693CPU331 and later, you can use the ALT operand to enter the slot of a single module in the main rack. When that is done, the DO_IO instruction executes in 80 microseconds instead of the 236 microseconds required when the instruction is programmed without the ALT parameter. No error checking is performed to prevent overlapping reference addresses or module type mismatches.

DRUM

Operation

A Drum Sequencer built-in function block instance operates like a mechanical drum sequencer. The Drum Sequencer steps through a set of potential output bit patterns and selects one based on inputs to the function block instance. The selected value is copied to a group of 16 discrete output references.
When the DRUM function block instance receives power flow, it copies the contents of a selected reference to the Q reference.
Power flow to the R (Reset) input or to the S (Step) input selects the reference to be copied.
The ???? (Control Block) input is the beginning reference for the Drum Sequencer's parameter block, which includes information used by the function block instance.
The function block instance passes power to the right only if it receives power from the left and no error condition is detected.
The DTO (Dwell Timeout Output) bit is cleared the first time the drum is in a new step. This is true:

- Whether the drum is introduced to a new step by changing the Active Step or by using the S (Step) Input.
- Regardless of the DT (Dwell Time array) value associated with the step (even if it is 0).
- During the first sweep the Active Step is initialized.

The Active Step and Preset Step of the Drum Sequencer's control block must be initialized for the Drum Sequencer to work or to pass power flow. Even if the Active Step is in the correct range (between 1 and length of the Pattern array) and the Preset Step is not used, the drum will not work if the Preset Step is not in the proper range.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-	R, P, L, W,	???? is the beginning address of a five-word

	dimensional WORD array of 5 words	symbolic	array: the Drum Sequencer's parameter block. The Control Block contains the following values, required to operate the Drum Sequencer: ▪ **Active Step** (at address ????): The active step value that specifies the element in the PTN (Pattern) array to copy to the Q output memory location. This is used as the array index into the PTN, DT (Dwell Time), FTT (Fault Timeout), and FF (First Follower) arrays. ▪ **Preset Step** (at address ????+1): A word input that is copied to the Active Step output when R (Reset) is On. ▪ **Step Control** (at address ????+2): A word that is used to detect OFF to ON transitions on both the S (Step) input and the Enable power flow input. The Step Control word is reserved for use by the built-in function block instance, and **must not be written to**. ▪ **Timer Control** (at address ????+3): Two words of data that hold values needed to run the timer. These values are reserved for use by the function block instance and **must not be written to**.
Length	Constant		The number of steps. 1 ≤ Length ≤ 128.
S	Power flow		Step input. Used to go one step forward in the sequence. When the function block instance receives power flow and S makes an OFF to ON transition, the Drum Sequencer moves one step. When R (Reset) is active, the function block instance ignores S.
R	Power flow		Reset input. Used to select a specific step in the sequence. When the DRUM function block instance and Reset both receive power flow, DRUM copies the Preset Step value in the Control Block to the Active Step reference in the Control Block. Then the function block instance copies the value in the Preset Step reference to the Q reference bits. When R is active, the function block instance ignores S.
PTN	WORD	I, Q, M, T, G,	Pattern. The PTN operand is the starting

	variable	R, P, L, AI, AQ, W, symbolic, I/O variable	address of Length words of memory, where Length is the number of steps. Each Pattern word represents one step of the Drum Sequencer. The value of each word represents the desired combination of outputs for a particular value of Active Step. The first element corresponds to an Active Step value of one; the last element corresponds to an Active Step value of Length. Logic Developer - PLC does not create an array for you. You must ensure you have enough memory for PTN.
DT	WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Dwell Time. Optional, but if you use the DT operand, you must also use the DTO operand, and vice-versa. The DT operand is the starting address of Length words of memory, where Length is the number of steps. Each DT word corresponds to one word of PTN. The value of each word represents the dwell time for the corresponding step of the Drum Sequencer in 0.1 second units. When the dwell time expires for a given step, the Dwell Timeout (DTO) bit is set. If a Dwell Time is specified, the drum cannot sequence into its next step until the Dwell Time has expired. Logic Developer - PLC does not create an array for you. You must ensure you have enough memory for DT
FTT	WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Fault Timeout. Optional, but if you use the FTT operand, you must also use the TFT operand, and vice-versa. The FTT operand is the starting address of Length words of memory, where Length is the number of steps. Each FTT word corresponds to one word of PTN. The value of each word represents the fault timeout for the corresponding step of the Drum Sequencer in 0.1 second units. When the fault timeout has expired, the Timeout Fault (TFT) bit is set. Logic Developer - PLC does not create an array for you. You must ensure you allocate enough memory for FTT.
Q	WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The same contents as the element of the Pattern Array that corresponds to the current Active Step.
DRC	BOOL	R, P, L, AI,	Drum Coil. Optional. Bit reference that is set

	variable	AQ, W, I, Q, M, T, G, symbolic, I/O variable	whenever the function block instance is enabled and the Active Step is not equal to the Preset Step.
DTO	BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	Dwell Timeout Output. Optional, but if you use the DTO operand, you must also use the DT operand, and vice-versa. Bit reference that is set if the Dwell Time for the current step has expired.
TFT	BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	Timeout Fault. Optional, but if you use the TFT operand, you must also use the FTT operand, and vice-versa. Bit reference that is set if the drum has been in a particular step longer than the step's specified Fault Timeout.
FF	BYTE variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	First Follower. Optional. The FF operand is the starting address of (Length/8) or ((Length/8)+1) bytes of memory, where Length is the number of steps. If MOD (Length/8) > 0, FF has ((Length/8)+1) bytes. Each bit in the bytes of FF corresponds to one word of PTN. No more than one bit in the FF bytes is ON at any time and that bit corresponds to the value of the Active Step. The first bit corresponds to an Active Step value of one; the last used bit corresponds to an Active Step value of Length.

CPU Support

DRUM is supported for VersaMax Nano/Micro Controllers firmware version 2.00, firmware version 10 or later of Series 90-30 CPUs 350 and 374, and PACSystems CPUs with firmware version 2.00 or later.

Edge Detectors - R_TRIG and F_TRIG

Operation

The IEC 61131-3 standard defines two edge detector function blocks. The *Rising Edge Trigger* R_TRIG and *Falling Edge Trigger* F_TRIG detect the changing state of a Boolean signal. The outputs of both function blocks produce a single pulse when an edge is detected.

 **Rising Edge Trigger R_TRIG**	 **Falling Edge Trigger F_TRIG**
You can declare instance variables of type R_TRIG and F_TRIG.	
The structure elements CLK and STATE are not accessible in logic. If they are used in logic, the compiler reports an error.	
All structure elements can be viewed in the Data Watch window and Data Monitor.	
Elements of an edge detector structure variable cannot be published. Publishing the elements would allow other applications to write to the instance data.	
In an LD block, each edge detector has power flow. The input power flow controls whether or not the function block instance is executed. The ENO parameter is the standard output parameter that indicates the function block executed correctly. There is no power flow output when there is no power flow input or one of the other parameters is invalid. When power flow is false, all Boolean output flow arguments are set to FALSE (this is the standard behavior for a function or function block that is not Enabled). This means that if power flow or Q argument is Boolean flow, it is set to False; if it is a variable, no change is made, that is, the variable will retain its last set value.	
For R_TRIG, when the CLK input transitions from False to True, the output Q is set to True (Q = CLK and NOT STATE; STATE = CLK) for one function block instance execution. The output Q then remains False until a new rising edge is detected.	For F_TRIG, when the CLK input transitions from True to False, the output Q is set to True (Q = NOT CLK and NOT STATE; STATE = NOT CLK) for one function block instance execution. The output Q then remains False until a new falling edge is detected.
For R_TRIG, when the Controller transitions from stop to run mode and the CLK input is set to True, the output Q is set to True after the first execution of the function block instance. After the second	For F_TRIG, when the Controller transitions from stop to run mode and the CLK input is set to False, the output Q is set to True after the first execution of the function block instance. After the next

execution, the output is set to False.	execution, the output is set to False.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
CLK	BOOL	I, Q, M, T, SA, SB, SC, G, symbolic	Edge detecting input
Q	Power flow		Edge detecting output
STATE	BOOL		Internal Value

Notes

- C blocks cannot use edge detector function blocks.
- The Structure element Q is accessible in user logic. It is read-only.
- Instance variables reside in symbolic discrete memory, which is non-retentive.

CPU Support

R_TRIG and F_TRIG are supported for PACSystems CPUs with firmware version 5.00 or later.

For Loop

Operation

A For loop repeats rung logic a specified number of times while varying the value of the INDEX variable in the loop. A For loop begins with a FOR_LOOP instruction and ends with an END_FOR instruction. The rung logic to be repeated must be placed between the FOR and END_FOR instructions. The optional EXIT_FOR instruction enables you to exit from the loop if a condition is met before the For loop ends normally.

When FOR_LOOP receives power flow, it saves the START, END, and INC (Increment) operands and uses them to evaluate the number of times the rungs between the FOR_LOOP and its END_FOR instructions are executed. Changing the START and END operands while the For loop is executing does not affect its operation.

When an END_FOR receives power flow, the For loop is terminated and power flow jumps directly to the instruction following the END_FOR instruction.

A FOR_LOOP instruction must be the last instruction in a rung. If the rung has multiple branches, other branches cannot end with a coil.

An EXIT_FOR instruction can be placed only between a FOR instruction and an END_FOR instruction. An EXIT_FOR instruction must be the last instruction in a rung. If the rung has multiple branches, other branches cannot end with a coil.

The END_FOR instruction occupies an entire rung.

A FOR_LOOP can assign decreasing values to its index variable by setting the increment to a negative number. For example, if the START value is 21, the END value is 1, and the increment value is –5, the statements of the FOR loop are executed five times, and the index variable is decremented by 5 in each pass. The values of the index variable will be 21, 16, 11, 6, and 1.

When the START and END values are set equal, the instructions of the FOR loop are executed only once.

When START cannot be incremented to reach the END or when START cannot be decremented to reach the END, the instructions within the FOR loop are not executed. For example, if the value of START is 10, the value of END is 5, and the INCREMENT is 1, power flow jumps directly from the FOR instruction to the instruction after the END_FOR instruction.

Note: If the power flow input for the FOR_LOOP instruction has power flow when it is first tested, the rungs between the FOR and its corresponding END_FOR instruction are executed the number of times initially specified by START, END, and INCREMENT. This repeated execution occurs on a single sweep of the Controller and may cause the watchdog timer to expire if the loop is long.

Nesting of FOR loops is allowed, but it is restricted to five FOR/END_FOR pairs. Each FOR instruction must have a matching END_FOR instruction following it.
Nesting with JUMPs and MCRs is allowed, provided that they are properly nested. MCRs and ENDMCRs must be completely within or completely outside the scope of a FOR/END_FOR pair. JUMPs and LABEL instructions must also be completely within or completely outside the scope of a FOR/END_FOR pair. Jumping into or out of the scope of a FOR/END_FOR is not allowed.

Operands

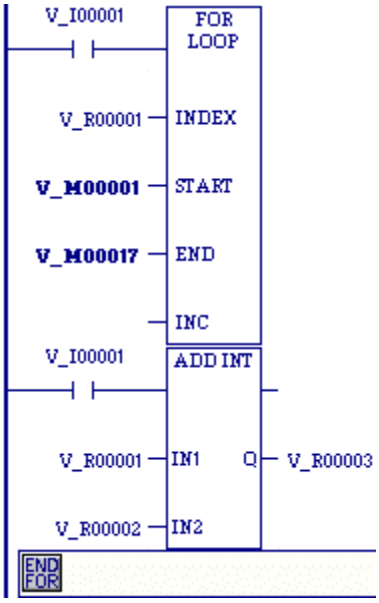
Only the FOR_LOOP instruction requires operands.

Operand	*Data Type*	*Memory Area*	*Description*
INDEX	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The index variable. When the loop has completed, this value is undefined. **Note:** Changing the value of the index variable within the scope of the FOR loop is not recommended.
START	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The index start value.
END	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The index end value.
INC	constant		(Optional.) The increment value. Default: 1.

Examples

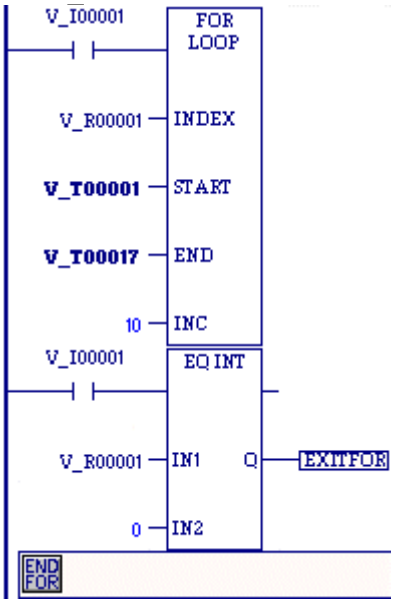
Example 1

The value for %M00001 (START) is 1 and the value for %M00017 (END) is 10. The INDEX (%R00001) increments by the value of the INC operand (which is assumed to be 1 when omitted) starting at 1 until it reaches the ending value 10. The ADD instruction of the loop is executed 10 times, adding the current value of I1 (%R00001) 1 ... 10 to I2 (%R00002).



Example 2

The value for %T00001 (START) is -100 and the value for %T00017 (END) is 100. The INDEX (%R00001) increments by tens, starting at -100 until it reaches it end value of +100. The EQ instruction of the loop tries to execute 21 times, with the INDEX (%R00001) being equal to –100, –90, –80, –70, –60, –50, –40, –30, –20, –10, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. However, when the INDEX (%R00001) is 0, the EXIT statement is enabled and power flow jumps directly to the statement after the END_FOR statement.



CPU Support

The FOR loop is supported for PACSystems CPUs and Series 90-70 Version 4.00 or later CPUs.

MASK_IO_INTR

CPU Support

MASK_IO_INTR is supported for:

PACSystems CPUs firmware version 3.50 and later only

Operation

Use MASK_IO_INTR to mask or unmask an interrupt from an input board when using I/O variables.

Tip: When not using I/O variables, you can use SVC_REQ 17.

When an interrupt is masked, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.

Successful execution occurs unless:

- The I/O board is not a supported input module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

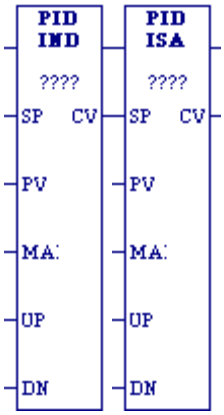
Operand	*Data Type*	*Memory Area*	*Description*
MASK	BOOL variable or bit references in nondiscrete memory	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	If state of MASK is set to ON, this indicates mask. If state of MASK is set to OFF, this indicates unmask.
IN1	BOOL or WORD variable	I, Q M, T, G, R, P, L, AI, AQ, W, I/O variable	The value of IN1 indicates the interrupt trigger to be masked or unmasked.

Masking / Unmasking Module Interrupts

During module configuration, interrupts from a module can be enabled or disabled. If a module's interrupt is disabled, it cannot be used to trigger logic execution in the application logic and it cannot be unmasked. However, if an interrupt is enabled in the

configuration, it can be dynamically masked or unmasked by the application logic during system operation.
For the application logic to mask and unmask interrupts that are enabled, use MASK_IO_INTR when using I/O variables.
When the interrupt is not masked, the CPU processes the interrupt and schedules the associated logic for execution. When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated logic for execution.
When the CPU transitions from Stop to Run, the interrupt is unmasked.
For additional information on configuring and using VME module interrupts in a PACSystems control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules* (GFK-2235).

PID



Overview

The Proportional Integral Derivative (PID) built-in function block is a general purpose algorithm used for closed-loop process control. When it receives power flow through a contact, the PID built-in function block compares a Process Variable (PV) feedback with a desired process Set Point (SP) and updates a Control Variable (CV) output based on the error.

The PID built-in function block uses PID loop gains and other parameters stored in the Reference Array, a 40-WORD array, to solve the PID algorithm at the desired time interval. All parameters are 16-bit WORDs for compatibility with 16-bit analog process variables. This allows %AI memory to be used for input Process Variables and %AQ to be used for output Control Variables.

As scaled 16-bit integer numbers, many parameters must be defined in either PV counts or units, or CV counts or units. For example, the SP input must be scaled over the same range as PV because the PID built-in function block calculates the error from the difference of these two inputs. The PV and CV counts can range from -32768 or 0 to +32767, matching analog scaling, or from 0 to 10000, to display variables as 0.00% to 100.00%. The PV and CV Counts do not have to have the same scaling, in which case there will be scale factors included in the PID gains.

The power flow output is energized when the built-in function block is performed without calculation error. If at least one calculation error exists, there is no power flow output.

Note: PID does not execute more often than once every 10 milliseconds. This could change your results if you set it up to execute every scan and the scan is less than 10 milliseconds. In such a case, PID does not run until enough scans have occurred to accumulate an elapsed time of 10 milliseconds. For example, if the scan time is 9 milliseconds, PID executes every other scan with an elapsed time of 18 milliseconds for every time it executes.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 40 words	R, P, L, W, symbolic	This is a 40-word array: the Reference Array. This is the PID built-in function block control information, which contains both user and internal parameters. The 40 consecutive words must not be shared.
SP	INT variable or	data flow, I, Q,	The control loop or process Set Point.

	constant; BOOL array of length 16 or more (restrictions apply)	M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Comparing the PV Counts to SP, PID adjusts the output CV so that PV matches SP (zero error).
PV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The control loop Process Variable input. Must be scaled over the same range as SP. Often a %AI input.
MAN	Power flow		Manual mode indicator. When energized to 1 (through a contact), the PID built-in function block is in Manual mode. If MAN is not energized (0), the PID built-in function block is in Automatic mode.
UP	Power flow		If energized along with MAN, UP adjusts the CV up by 1 CV per solution, that is, 1 CV per access to the PID built-in function block.
DN	Power flow		If energized along with MAN, DN adjusts the CV down by 1 CV per solution, that is, 1 CV per access to the PID built-in function block.
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The Control Variable output to the process. Often a %AQ analog output.

Reference Array Parameters

Besides the two input words and the three Manual control contacts, the PID built-in function block requires 13 user-defined parameters in the Reference Array. These parameters must be set before calling the built-in function block. The other 27 parameters are used by the Controller and are non-configurable. The %Ref shown in the table below is the beginning address of the Reference Array (the ???? operand). The number after the plus sign is the offset in the array. For example, if the Reference Array starts at %R100, %R113 contains the Manual Command (%Ref+0013) used to set the Control Variable and the integrator in Manual mode.

Note: The Reference Array must be %R, %P, or %L registers. Every PID built-in function block in the logic must use a different 40-word array even if all 13 user parameters are the same, because other words in the array are used for internal PID data storage. Ensure that there are at least 40 %R, %P, or %L registers between the starting reference address and the highest configurable %R, %P, or %L register.

Register and Parameter	*Low Bit Units and Range of Values*	*Description*
%REF+0000	UINT. 0 to 255	Optional. Loop number; number of the PID

Loop Number		built-in function block. For user display only. Provides a common identification in the Controller with the loop number defined by an operator interface device. **Note:** The loop number is displayed under the built-in function block address when logic is monitored from the LD editor.
%REF+0001 Algorithm	UINT. Non-configurable. Set and maintained by the Controller.	1 = ISA algorithm (PID_ISA) 2 = Independent algorithm (PID_IND)
%REF+0002 Sample Period	Low bit set at 1 represents 10ms. UINT. Range: 0 (every scan) to 65,535 (10.9 Min).	The shortest time in 10ms increments, between solutions of the PID algorithm. For example, use a 10 for a 100ms sample period. If Sample Period is 0, the algorithm is solved every time PID is called, unless the scan is under 10ms. See Sample Period and PID Scheduling. The PID algorithm is solved only if the current Controller elapsed time clock is at or later than the last PID solution time plus this Sample Period. PID compensates for the actual time elapsed since the last execution, within 100 microseconds.
%REF+0003 Dead Band +	PV Counts. 0 to 32767 (never negative)	Integer (INT) values defining the upper (+) and lower (-) Dead Band limits in PV Counts. If no Dead Band is required, these values must be 0. If the PID Error (SP – PV) or (PV- SP) is above the (-) value and below the (+) value, the PID calculations are solved with an Error of 0. If non-zero, the (+) value must be greater than 0 and the (-) value less than 0 or the PID built-in function block will not work. **Tip:** Leave these at 0 until the PID loop gains are set up or tuned. After that, a Dead Band might be added to avoid small CV output changes due to variations in error, perhaps to reduce mechanical wear. **Note:** The Deadband Action bit determines how PID uses the deadband limits.
%REF+0004 Dead Band —	PV Counts. -32768 to 0 (never positive)	
%REF+0005 ▪ In the	Low bit set at 1 represents 1% of	The change in the CV in CV Counts for a 100 PV Count change in the Error Term. It

PID_IND algorithm, this word stores the Proportional Gain, Kp ▪ In the PID_ISA algorithm, this word stores the Controller gain, Kc	(CV%)/(PV%). Range of represented values: 0 to 327.67%	is displayed as 0.00 %/% with an implied decimal point of 2. A Kp or Kc entered as 450 is displayed as 4.50 and results in a (Kp*Error/100) or (Kc*Error/100) = (450*Error/100) contribution to the PID Output. **Tip:** When the PID_IND algorithm is used, Kp is generally the first gain set to adjust a PID loop, a PD loop (Proportional – Derivative) or a PI loop (Proportional – Integral, which is rarely used). Using Kp by itself may be sufficient to control certain types of processes.
%REF+0006 Derivative Gain–Kd	Low bit set at 1 represents 0.01 seconds. Integer. Range of represented values: 0 to 327.67 sec.	The change in the CV in CV Counts if the Error or PV changes 1 PV Count every 10ms. Entered as a time with the low bit indicating 10ms, it is displayed as 0.00 seconds with an implied decimal point of 2. For example, a Kd entered as 120 is displayed as 1.20 Sec and results in a [Kd * (delta Error)/(delta time)] = (120*4/3) contribution to the PID Output if Error was changing by 4 PV Counts every 30ms. **Tip:** When the PID_IND algorithm is used, Kd can be used to speed up a slow loop response, but is very sensitive to PV input noise. (On PACSystems, you can alleviate noise sensitivity by using the derivative filter, which is enabled by setting bit 5 of the Config Word.) In some processes, you can omit Kd and control the process by using Kp by itself or Kp in combination with Ki.
%REF+0007 Integral Rate–Ki	Low bit set at 1 represents 1 repeat/1000 seconds. Integer values, 0 to 32,767, representing the values 0 to 32.767 repeats/sec	The change in the CV in CV Counts if the Error is a constant 1 PV Count. Displayed as 0.000 Repeats/Sec with an implied decimal point of 3. For example, a Ki entered as 1400 is displayed as 1.400 Repeats/Sec and results in a (Ki * Error * dt) = (1400*20*50/1000) contribution to PID Output for an Error of 20 PV Counts and a 50ms Controller scan time (Sample Period of 0). **Tip:** When the PID_IND algorithm is used, Ki is usually the second gain set in a PID loop, Kp being the first gain set. Ki can be omitted entirely in a PD (Proportional – Derivative) loop or P (Proportional only) loop. Ki provides inertia to the control system.

		that is, it acts as an automatic bias.
%REF+0008 CV Bias/Output Offset	CV Counts. Integer, -32768 to +32767 (add to integrator output)	Number of CV Counts added to the PID Output before the rate and amplitude clamps. Can be used to set non-zero CV values if only Kp Proportional gains are used, or for feed forward control of this PID loop output from another control loop. For example, if you want to let the built-in function block regulate error around the output midpoint, set the Bias to the midpoint of the range. For example, for a full range 0 through +32,767, +16,383 is the midpoint. **Note:** In the PID_IND algorithm, Ki acts as an automatic bias.
%REF+0009 Upper Clamp	CV Counts. Integer, –32768 to +32767. (Must be greater than %Ref+10.) Output limit.	Required values. Number of CV Counts that define the highest and lowest value for CV. The Upper Clamp must have a more positive value than the Lower Clamp; otherwise, the PID built-in function block will not work. These are usually used to define limits based on physical limits for a CV output. They are also used to scale the Bar Graph display for CV (visible when tuning the PID in the **PID Project Values** dialog box). The built-in function block has antireset windup to modify the integrator value when a CV clamp is reached.
%REF+0010 Lower Clamp	CV Counts. Integer, –32768 to +32767. (Must be less than %Ref+09.) Output limit.	
%Ref+0011 Minimum Slew Time	UINT. Number of seconds/Full Travel. Range: 0 (none) to +32767 sec to move +32767 CV	Minimum number of seconds for the CV output to move from 0 to full travel of 100% or +32767 CV Counts. It is an inverse rate limit on how fast the CV output can be changed. If positive, CV cannot change more than (+32767 CV Counts) * (Delta Time (in seconds)) / (Minimum Slew Time). For example, if the Sample Period is 2.5 seconds and the Minimum Slew Time is 500 seconds, CV cannot change more than +32767*2.5/500 = 163 CV Counts per PID solution. An antiwindup feature adjusts the integrator value if the CV rate limit is exceeded. If Minimum Slew Time is 0, there is no CV rate limit. **Note:** Set Minimum Slew Time to 0 while tuning or adjusting PID loop gains.

%Ref+0012 Config Word	Low 6 bits used. BOOLEAN.	The low 6 bits of this word are used to modify five standard PID settings. The other bits should be set to 0. **Bit 0:** Error Term. When this bit is 0, the error term is the normal (SP-PV). When this bit is 1, the error term is (PV-SP), which reverses the sign of the feedback term: this is for Reverse Acting controls where the CV must go down when the PV goes up. **Bit 1:** Output Polarity. When this bit is 0, the CV output represents the output of the PID calculation. When it is set to 1, the CV output represents the negative of output of the PID calculation rather than the normal positive value. **Bit 2:** Derivative action on PV. When this bit is 0, the derivative action is applied to the Error Term. When it is set to 1, the derivative action is applied to PV. **Bit 3:** Deadband action. When this bit is 0: ▪ If the error is within the deadband limits, then the error is forced to be zero. ▪ Otherwise, the error is not affected by the deadband limits. When the Deadband action bit is 1: ▪ If the error is within the deadband limits, then the error is forced to be zero. ▪ If the error is outside the deadband limits, then the error is reduced by the deadband limit (error error – deadband limit). **Bit 4:** Antireset windup action. When this bit is 0, the antireset windup action uses a reset back calculation. When the output is clamped, this replaces the accumulated Y remainder value with whatever value is necessary to produce the clamped output exactly. When the bit is 1, this replaces accumulated Y term with the value of the Y term at the start of the calculation. In this way, the preclamp Y value is held as long as the output is clamped. **Bit 5:** (Used in PACSystems only.) Enable

		derivative filtering. When this bit is set to 0, no filtering is applied to the derivative term. When set to 1, a first order filter is applied. This limits the effects of higher frequency process disturbances on the derivative term. In Controllers other than PACSystems, this bit should be set to 0. **Bits 6 and 7:** Unused. Should be set to 0. **Notes** ▪ The antireset windup action bit is available in PACSystems; VersaMax; release 6.50 or later Series 90-30; and release 6.01 or later Series 90-70. ▪ The bits are set in powers of 2. For example, to set Config Word to 0 for default PID configuration, you would add 1 to change the Error Term from (SP-PV) to (PV-SP), or add 2 to change the Output Polarity from (CV = PID Output) to (CV = - PID Output), or add 4 to change Derivative Action from Error rate of change to PV rate of change.
%Ref+0013 Manual Command	CV Counts. Integer. Tracks CV in Auto or Sets CV in Manual	Value set to the current CV output while the PID built-in function block is in Automatic mode. When the built-in function block is switched to Manual mode, this value is used to set the CV output and the internal value of the integrator within the Upper and Lower Clamp and Slew Time limits.
%Ref+0014 Control Word	Low 5 bits used. BOOLEAN. Maintained by the Controller, ***unless*** Bit 0 is set.	A discrete data structure with the first five bit positions in the following format: **Bit 0.** Word value: 1. Override. Internal parameter normally left at 0. If this low bit is set to 1, this word and the internal SP, PV and CV parameters must be used for remote operation of this PID built-in function block. This allows remote operator interface devices, such as a computer, to take control away from the Controller logic. **Note:** If you do not want to enable remote operation of the PID built-in function block, ensure the Control Word is set to 0. If the low bit is 0, the next 4 bits can be read to track the status of the PID input contacts as long as the PID Enable contact has power. **Bit 1.** Word value: 2. Manual/Auto. If 1, PID built-in function block is in Manual mode; if 0, it is in Automatic mode.

		Bit 2. Word value: 4. Enable. Should normally be 1; otherwise, built-in function block is never called. **Bit 3.** Word value: 8. UP/Raise. If 1 and Manual (Bit 1) is 1, CV is incremented every solution **Bit 4.** Word value: 16. DN/Lower. If 1 and Manual (Bit 1) is 1, CV is decremented every solution.
%Ref+0015 Internal SP	N/A. Set and maintained by the Controller. Non-configurable.	Tracks SP in. Must be set externally if Override bit = 1.
%Ref+0016 Internal CV	N/A. Set and maintained by the Controller. Non-configurable.	Tracks CV out.
%Ref+0017 Internal PV	N/A. Set and maintained by the Controller. Non-configurable.	Tracks PV in. Must be set externally if Override bit = 1.
%Ref+0018 Output	N/A. Set and maintained by the Controller. Non-configurable.	Signed word value representing the output of the built-in function block before the application of the optional inversion. If no output inversion is configured and the output polarity bit in the Config Word is set to 0, this value equals the CV output. If inversion is selected and the output polarity bit is set to 1, this value equals the negative of the CV output.
%Ref+0019 Diff Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to this location.*
%Ref+0020 and %Ref+0021 Int Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to these locations.*
%Ref+0022 Slew Term Storage	N/A. Set and maintained by the Controller. Non-configurable.	Used internally for storage of intermediate values. *Do not write to this location.*
%Ref+0023,	N/A. Set and	Internal elapsed time storage (time last PID

0024, 0025 Clock	maintained by the Controller. Non-configurable.	executed). *Normally, do not write to these locations. Occasionally, circumstances may justify writing to these locations.*
%Ref+0026 Y Remainder Storage	N/A. Set and maintained by the Controller. Non-configurable.	Holds remainder for integrator division scaling for 0 steady state error. Also used for antireset windup action.
%Ref+0027 Lower Range for SP, PV	PV Counts. Integer, -32768 to +32767 (Must be less than %Ref+28) for display	Optional integer values in PV Counts that define the highest and lowest display value for the SP and PV.
%Ref+0028 Upper Range for SP, PV	PV Counts. Integer, -32768 to +32767 (Must be greater than %Ref+27) for display	
%Ref+0029 to %Ref+0034 Reserved for internal use	N/A. Non-configurable.	Do not use these references. On PACSystems targets, %Ref+0030 and %Ref+0031 are used when calculating the derivative filter.
%Ref+0035 to Ref+0039 Reserved for external use	N/A. Non-configurable.	

Operation

Automatic operation

The PID built-in function block can be called every scan by providing power flow to Enable and no power flow to the Manual input contacts. The PID built-in function block compares the current Controller elapsed time clock with the last PID solution time stored in the internal Reference Array. If the time difference is greater than the sample period defined in the third word (%Ref+2) of the Reference Array, the PID algorithm is solved using the time difference and both the last solution time and Control Variable output are updated. In Automatic mode, the output Control Variable is placed in the Manual Command parameter (%Ref+13).

Manual operation

If power flow is provided to both Enable and Manual input contacts, the PID built-in function block is placed in Manual mode and the output Control Variable is set from the Manual Command parameter %Ref+13. If either the UP input or DN input has power flow, the Manual Command word is respectively incremented or decremented by one CV count every PID built-in function block solution. For faster manual changes of the output Control Variable, you can also add or subtract any CV count value directly to/from the Manual Command word.

The PID built-in function block uses the CV Upper and CV Lower Clamp parameters to limit the CV output. If a positive Minimum Slew Time is defined, it is used to limit the rate of change of the CV output. If either the CV amplitude or rate limit is exceeded, the value stored in the integrator is adjusted so that CV is at the limit. This antireset windup feature means that even if the error tried to drive CV above (or below)

the clamps for a long period of time, the CV output will move off the clamp as soon as the error term changes sign.
This operation, with the Manual Command tracking CV in Automatic mode and setting CV in Manual mode, provides a bumpless transfer between Automatic and Manual modes. The CV Upper and Lower Clamps and the Minimum Slew Time still apply to the CV output in Manual mode and the internal value stored in the integrator is updated. This means that if you step the Manual Command in Manual mode, the CV output does not change any faster that the Minimum Slew Time (Inverse) rate limit and does not go above or below the CV Upper or CV Lower Clamp limits.

Note: A specific PID built-in function block should not be called more than once per scan.

Internal Parameters in Reference Array

The PID built-in function block reads 13 user parameters and uses the rest of the 40-word Reference Array for internal PID storage. Normally you would not change any of these values. If you call the PID built-in function block in Automatic mode after a long delay, you might want to use SVC_REQ 16 or SVC_REQ 51 to load the current Controller elapsed time clock into %Ref+23 to update the last PID solution time to avoid a step change on the integrator. If you have set the Override low bit of the Control Word (%Ref+14) to 1, the next four bits of the Control Word must be set to control the PID built-in function block input contacts, and the Internal SP and PV must be set because you have taken control of the PID built-in function block away from the LD logic.

PID Algorithm Selection (PID_IND or PID_ISA) and Gains

The PID built-in function block can be programmed selecting either the Independent (PID_IND) term or standard ISA (PID_ISA) versions of the PID algorithm. The only differences in the algorithms are:

- How the Integral and Derivative gains are defined.
- What word 6 of the Reference Array is used for.

Algorithm	*Value*	*Definition*	*Condition*
Both	*Error*=	(SP – PV) Reverse Acting	if the low bit of Config Word is set to 0
		(PV - SP) Direct Acting	if the low bit of Config Word is set to 1

Note: Direct Acting is sometimes referred to as Forward Acting.

Both PID built-in function block types calculate the Error term as SP – PV, Reverse Acting, which can be changed to the Direct Acting mode, PV – SP, by setting the Error Term to 1. The Error Term is the low bit (0) in the Config Word (%Ref+12).

- In a **direct** acting proportional (P) loop, an increase in the process variable (PV) causes an increase in the output (CV).
- In a **reverse** acting P loop, an increase in PV causes a decrease in CV.

Introducing the integral term (I) changes the behavior:

- In a **direct** acting PI loop, the output (CV) increases when the process variable (PV) is greater than the setpoint (SP).
- In a **reverse** acting PI loop, the output (CV) decreases when the process variable (PV) is greater than the setpoint (SP).

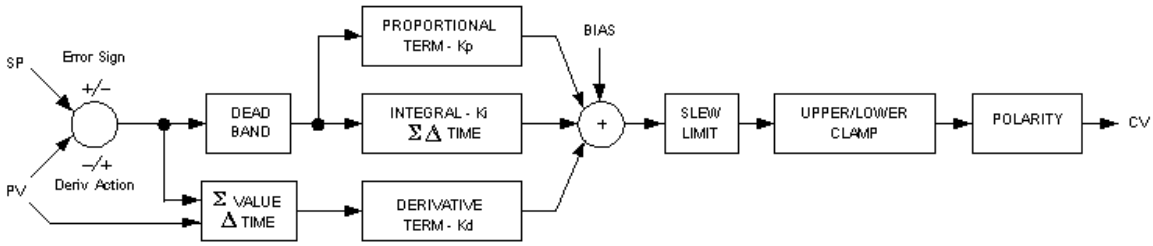
Algorithm	*Value*	*Definition*	*Condition*
Both	*Derivative*=	(Error – previous Error)/dt	if the 3rd bit of Config Word is set to 0

		(PV – previous PV)/dt	if the 3rd bit of Config Word is set to 1
Both	*dt*=	(Current Controller Elapsed Time clock) – (Controller Elapsed Time Clock at Last PID built-in function block solution)	

The Derivative is normally based on the change of the Error term since the last PID built-in function block solution, which may cause a large change in the output if the SP value is changed. If this is not desired, the third bit of the Config Word can be set to 1 to calculate the Derivative based on the change of the PV. The dt (or Delta Time) is determined by subtracting the last PID solution clock time for this built-in function block from the current Controller elapsed time clock.

Algorithm	*Value*	*Definition*
PID_IND	PID output	Kp * Error + Ki * Error * dt + Kd * Derivative + CV Bias
PID_ISA	PID output	Kc * (Error + Error * dt/Ti + Td * Derivative) + CV Bias

where Kc is the Controller gain, and Ti is the Integral time and Td is the Derivative time. The advantage of PID_ISA is that adjusting the Kc changes the contribution for the integral and derivative terms as well as the proportional one, which may make loop tuning easier. If you have PID gains in terms or Ti and Td, use Kp = Kc, Ki = Kc/Ti, and Kd = Kc*Td
to convert them to use as PID User Parameter inputs.
The CV Bias term is an additive term separate from the PID built-in function block components. It may be required if you are using only Proportional Kp gain and you want the CV to be a non-zero value when the PV equals the SP and the Error is 0. In this case, set the CV Bias to the desired CV when the PV is at the SP. CV Bias can also be used for feed forward control where another PID loop or control algorithm is used to adjust the CV output of this PID loop.
If an Integral Ki gain is used, the CV Bias would normally be 0 as the integrator acts as an automatic bias. Just start up in Manual mode and use the Manual Command word (%Ref+13) to set the integrator to the desired CV, then switch to Automatic mode. This also works if Ki is 0, except the integrator will not be adjusted based on the Error after going into Automatic mode.
PID_IND. The following diagram shows how the PID_IND algorithm works:



PID_ISA. The ISA Algorithm (PID_ISA) is similar except that the Kc gain is factored out of Ti and Td so that the integral gain is (Kc/Ti) and the derivative gain is (Kc * Td). The Error sign, DerivAction and Polarity are set by bits in the Config Word user parameter.

CV Amplitude and Rate Limits

The PID built-in function block does not send the calculated PID Output directly to CV. Both PID built-in function block algorithms can impose amplitude and rate of change limits on the output Control Variable. The maximum rate of change is determined by dividing the maximum 100% CV value (32767) by the Minimum Slew Time, if specified as greater than 0. For example, if the Minimum Slew Time is 100 seconds, the rate limit will be 327 CV counts per second. If the dt solution time was 50 milliseconds, the new CV output cannot change more than 327*50/1000 or 16 CV counts from the previous CV output.

The CV output is then compared to the CV Upper and CV Lower Clamp values. If either limit is exceeded, the CV output is set to the clamped value. If either the rate or the amplitude limit is exceeded modifying CV, the internal integrator value is adjusted to match the limited value to avoid reset windup.
Finally, the built-in function block checks the Output Polarity (2nd bit of the Config Word (%Ref+12)) and changes the sign of the output if the bit is 1.

Value	*Definition*	*Condition*
CV=	Clamped PID output	if the Output Polarity bit is set to 0
	(-Clamped PID output)	if the Output Polarity bit is set to 1

If the PID built-in function block is in Automatic mode, the final CV is placed in the Manual Command (%Ref+13). If the PID built-in function block is in Manual mode, the PID equation is skipped as CV is set by the Manual Command, but all the rate and amplitude limits are still checked. That means the Manual Command cannot change the output above the CV Upper Clamp or below the CV Lower Clamp and the output cannot change faster than the Minimum Slew Time allowed.

Sample Period and PID Scheduling

The PID built-in function block is a digital implementation of an analog control instruction, so the dt sample time in the PID Output equation is not the infinitesimally small sample time available with analog controls. The majority of processes being controlled can be approximated as a gain with a first or second order lag, possibly with a pure time delay. The PID built-in function block sets a CV output to the process and uses the process feedback PV to determine an Error to adjust the next CV output. A key process parameter is the total time constant, which is how fast the PV responds when the CV is changed. The total time constant, Tp+Tc, for a first order system is the time required for PV to reach 63% of its final value when CV is stepped. (For further information, see PV Unit Reaction Curve Input from Process.) The PID built-in function block cannot control a process unless its Sample Period is well under half the total time constant. Larger Sample Periods will make it unstable.
The Sample Period should be no bigger than the total time constant divided by 10 (or down to 5 worst case). For example, if PV seems to reach about 2/3 of its final value in 2 seconds, the Sample Period should be less than 0.2 seconds, or 0.4 seconds worst case. On the other hand, the Sample Period should not be too small, such as less than the total time constant divided by 1000, or the (Ki * Error * dt) term for the PID integrator will round down to 0. For example, a very slow process that takes 10 hours or 36000 seconds to reach the 63% level should have a Sample Period of 40 seconds or longer.
Unless the process is very fast, it is not usually necessary to use a Sample Period of 0 to solve the PID algorithm every PID scan. If many PID loops are used with a Sample Period greater than the scan time, there may be wide variations in Controller scan time if many loops end up solving the algorithm at the same time. The simple solution is to sequence one or more 1 bits through an array of bits set to 0 that is being used to enable power flow to individual PID built-in function blocks.

Setting User Parameters

- set user parameters and tune loop gains for simple processes
- set user parameters and tune loop gains for not-so-simple processes
- set user parameters in the Reference Array

Setting User Parameters and Tuning Loop Gains for Simple Processes

As all PID built-in function block parameters are totally dependent on the process being controlled, there are no predetermined values that will work; however, it is usually a simple, iterative procedure to find acceptable loop gain for simple processes.

Note: The following is only a ***possible*** sequence of steps.

To set user parameters and tune loop gains for *simple* processes:

1. Set all the the user-defined Reference Array parameters to 0, then set the CV Upper and CV Lower Clamps to the highest and lowest CV expected. Set the

Sample Period to a value in the range [(estimated process time constant)/10] to [(estimated process time constant)/100].

2. Put the PID built-in function block instance in Manual mode and set Manual Command (%Ref+13) at different values to check if CV can be moved to Upper and Lower Clamp. Record PV value at some CV point and load it into SP.
3. Set a small gain, such as (100 * Maximum CV/Maximum PV), into Kp and turn off Manual mode. Step SP by 2 to 10% of the Maximum PV range and observe PV response. Increase Kp if PV step response is too slow or reduce Kp if PV overshoots and oscillates without reaching a steady value.
4. Once a Kp is found, start increasing Ki to get overshooting that dampens out to a steady value in 2 to 3 oscillations. This may require reducing Kp. Also try different step sizes and CV operating points.
5. After suitable Kp and Ki gains are found, try adding Kd to get quicker responses to input changes providing it does not cause oscillations.
   Kd is often not needed and does not work with noisy PV.
6. Check gains over different SP operating points and add Dead Band and Minimum Slew Time if needed. Some Reverse Acting processes may need setting Config Word Error Sign or Polarity bits.

Setting User Parameters and Tuning Loop Gains for Not-So-Simple Processes

To set user parameters and tune loop gains for *not-so-simple* processes:

1. Determine the K, Tp, and Tc process characteristics.
2. Use the K, Tp, and Tc parameters to estimate initial Kp, Ki, and Kd parameters.
3. Convert the estimated Kp, Ki, and Kd parameters into the units required by the PID built-in function block instance.
4. Set the converted Kp, Ki, and Kd parameters, and other user parameters in the Reference Array.
5. Fine tune until you get the desired result.

Determining the K, Tp, and Tc Process Characteristics

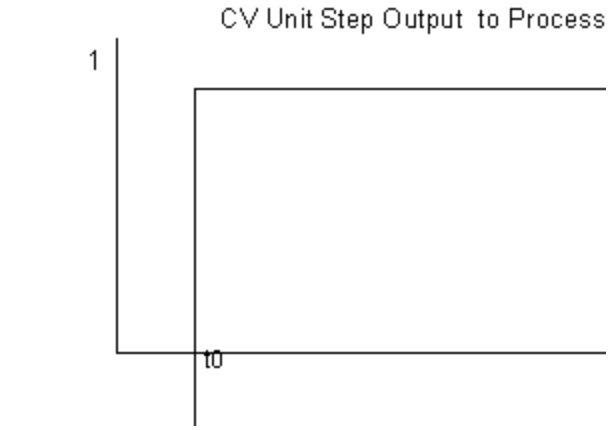
The loop gains of a PID built-in function block instance, Kp, Ki, and Kd, are determined by the characteristics of the process being controlled. Two key questions when setting up a PID loop are:

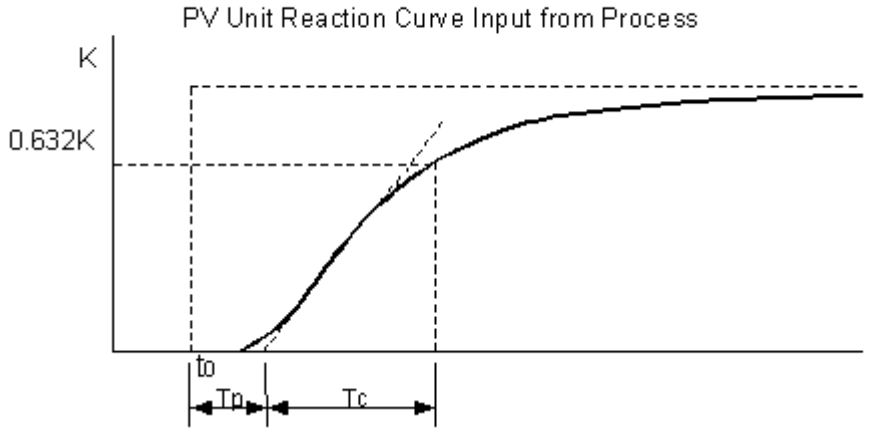
- How big is the change in PV when CV changes by a fixed amount, or what is the open loop gain?
- How fast does the system respond, or how quickly does PV change after the CV output is stepped?

Many processes can be approximated by a process gain, first or second order lag and a pure time delay. In the frequency domain, the transfer calculation for a first order lag system with a pure time delay is:

PV(s)/CV(s) = G(s) = K * e **(–Tp s)/(1 + Tc s)

Plotting a step response at time t0 in the time domain provides an open loop unit reaction curve:





The tangent at the maximum slope of the PV unit reaction curve enables you to determine the following process model parameters:

Parameter	*Description*
K	Process open loop gain = final change in PV/change in CV at time t0 **Note:** No subscript on K.
Tp	Process or pipeline time delay or dead time after t0 before the process output PV starts moving
Tc	First order Process time constant, time required after Tp for PV to reach 63.2% of the final PV

The following procedure is usually the quickest way to measure the K, Tp, and Tc parameters:

To determine the K, Tp, and Tc process characteristics

1. Put the PID built-in function block instance in Manual mode.
2. Make a small step in CV output, by changing the Manual Command (%Ref+13), and plot the PV response.
3. Repeat step 2 over time.
   For slow processes, this can be done manually, but for faster processes a chart recorder or computer graphic data logging package will help.

The CV step size should be large enough to cause an observable change in PV, but not so large that it disrupts the process being measured. A good size may be from 2 to 10% of the difference between the CV Upper and CV Lower Clamp values.

Setting Loop Gains for Not-So-Simple Processes

Once you have determined the three process model parameters, K, Tp and Tc, you can use them to estimate initial loop gains for a PID built-in function block instance. Two approaches are suggested:

- The approach developed by Ziegler and Nichols in the 1940s provides good response to system disturbances with gains producing an amplitude ratio of 1/4. The amplitude ratio is the ratio of the second peak over the first peak in the closed loop response.
- The "Ideal Tuning" procedure provides the best response to SP changes, delayed only by the Tp process delay or dead time.

To determine initial loop gains using the Ziegler and Nichols approach:

1. Calculate the Reaction rate:
   R = K/Tc
2. For Proportional control only, calculate Kp as
   Kp = 1/(R * Tp) = Tc/(K * Tp)
   For Proportional and Integral control, use
   Kp = 0.9/(R * Tp) = 0.9 * Tc/(K * Tp)
   Ki = 0.3 * Kp/Tp
   For Proportional, Integral and Derivative control, use
   Kp = G/(R * Tp), where G is from 1.2 to 2.0
   Ki = 0.5 * Kp/Tp
   Kd = 0.5 * Kp * Tp
3. Check that the Sample Period is in the range (Tp + Tc)/10 to (Tp + Tc)/1000

To determine initial loop gains using the "Ideal Tuning" procedure:

1. Calculate Kp = 2 * Tc/(3 * K * Tp)
2. Calculate Ki = Tc
3. If Derivative term is used, calculate Kd = Ki/4

To convert the estimated Kp, Ki, and Kd parameters into the units required by the PID built-in function block instance:

1. Once initial gains are determined, convert them to integer User Parameters. To avoid scaling problems, the Process gain, K, should be calculated as a change in input PV Counts divided by the output step change in CV Counts and not in process PV or CV engineering units. Specify all times in seconds.
2. Once Kp, Ki and Kd are determined, Kp and Kd can be multiplied by 100 and entered as integer while Ki can be multiplied by 1000 and entered into the User Parameter Reference Array.

To set user-defined Reference Array parameters:

1. In the LD editor, right-click a PID built-in function block instance and choose **Tuning**.

   Note: Alternatively, from the Data menu, choose Tune PID.

   The PID Project Values dialog box appears.
2. Enter values and select options as required.
3. To save your entries, click **Update Project** and when the confirmation popup appears, click Yes.
   This updates the initial values of the PID built-in function block in the project on your computer, but it does not update the Controller.

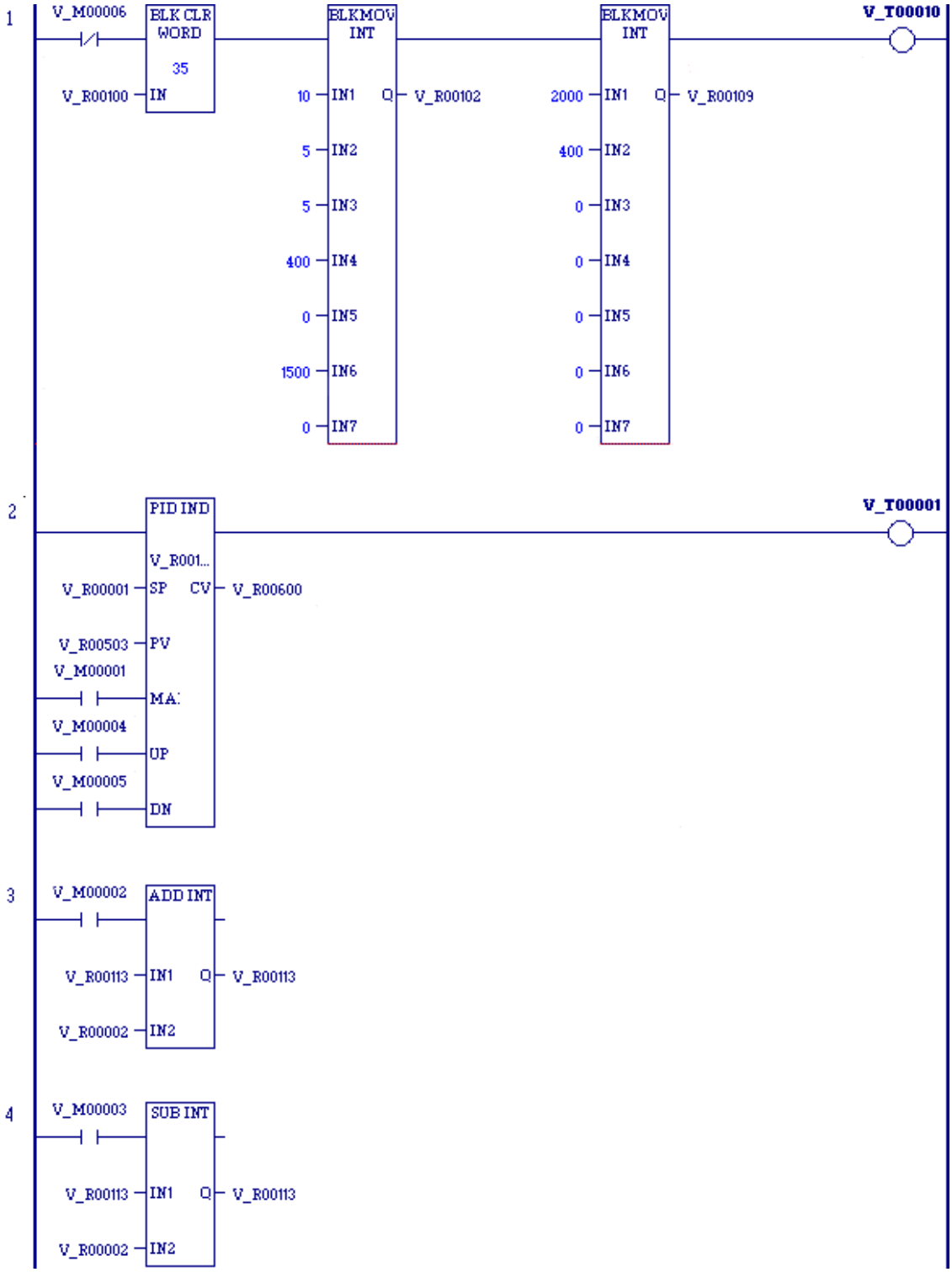
Tip: On the PID - Project Values dialog box, click the Help button to view reference information about the various tuning parameters. The Help topics automatically scroll into view as you hover over or select controls on the dialog box.

Notes

- You can use 0 for most default values, except the CV Upper Clamp, which must be greater than the CV Lower Clamp for the PID built-in function block instance to operate.
- A PID built-in function block instance does **not** pass power if there is an error in User Parameters, so monitor with a temporary coil while modifying data.
- Once suitable PID built-in function block instance values have been chosen, they should be defined as constants in a BLKMOV instruction so that they can be used to reload default PID user parameters if needed.

Example

A PID_IND built-in function block instance has a Sample Period of 100 milliseconds (%R102), a Kp gain of 4.00 (%R105) and a Ki gain of 1.500 (%R107). The Set Point is stored in %R1 with the Control Variable output in %R600 and the Process Variable returned in %R503. CV Upper and CV Lower Clamps must be set, in this case to 2000 and 400 (%R109 and %R110), and an optional small Dead Band of +5 and –5 has been included (%R103 and %R104). The 40-word Reference Array starts in %R100. Normally User Parameters are set in the Reference Array, but %M6 can be set to initialize the 14 words starting at %R102 (%Ref+2) from constants stored in logic.
The PID_IND function block instance can be switched to Manual mode with %M1 so that the Manual Command, %R113, can be adjusted. Bits %M4 or %M5 can be used to increase or decrease %R113 and the PID CV and integrator by 1 every 100 millisecond solution. For faster manual operation, bits %M2 and %M3 can be used to add or subtract the value in %R2 to/from %R113 every Controller scan. The %T1 output is on when the PID function block instance is OK.

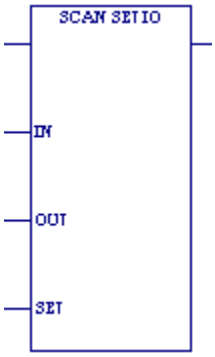


CPU Support

The PID_IND and PID_ISA built-in function blocks are supported for PACSystems CPUs, Series 90-70 Version 2.02 or later CPUs, Series 90-30 CPUs, and VersaMax CPUs.

Note: The antireset windup action bit is available only in PACSystems CPUs, VersaMax CPUs, in Series 90-30 CPUs version 6.50 or later, and in Series 90-70 CPUs release 6.01 or later.

SCAN_SET_IO



Operation

A ScanSet I/O (SCAN_SET_IO) instruction scans the I/O of a specified scan set number. You can specify whether the inputs and outputs of the associated scan are scanned.

To assign less frequent scans:

1. In the Project tab of the Navigator, expand the Hardware Configuration node.
2. Expand the main rack.
3. Double-click the slot that contains the CPU.
   The Parameter editor displays the CPUs parameters.
4. In the Scan Sets tab, set the Number of Sweeps to a value greater than 1.

Notes

- Modules can be assigned to scan sets in hardware configuration.
- Execution of the SCAN_SET_IO instruction will not impact the normal scanning process of the corresponding scan set. For example, if the corresponding scan set is configured for non-default Number of Sweeps and/or Output delay settings, these are still in effect regardless of how many executions of the SCAN_SET_IO instruction occur in a sweep.
- The SCAN_SET_IO instruction will not scan any modules in the specified scan set if the modules do not support DO_IO scanning.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

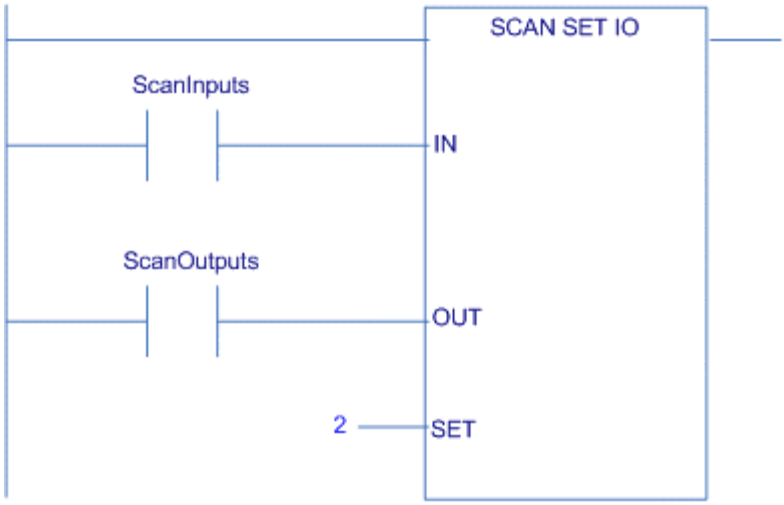
Operand	*Data Type*	*Memory Area*	*Description*
ENO	BOOL variable, bit reference in non-BOOL variable	Power Flow	Optional. Activated when all arguments to the instruction are valid and there are no errors in scanning.
IN			If set to True, the inputs are scanned.
OUT			If set to True, the outputs are scanned.

SET	UINT variable	All except %S memory types	Number of the scan set to be scanned. Scan Sets are specified in the CPU hardware configuration and assigned to modules in the module hardware configuration. **Note:** Valid range for PACSystems is 1 through 32.

Example

By using the SCAN_SET_IO instruction in an interrupt block, you can create a custom I/O scan. For example, two SCAN_SET_IO instructions can be used in an interrupt block to scan the inputs of a Scan Set at the beginning of the block and the outputs of the same Scan Set at the end of the block.
See the following example.

- When ScanInputs is set to ON, input data for all I/O modules assigned to Scan Set 2 is updated.
- When ScanOutputs is set to ON, output data for all I/O modules assigned to Scan Set 2 is updated.



CPU Support

SCAN_SET_IO is supported for PACSystems CPUs with a Firmware version of 5.0 or later.

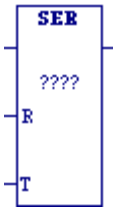
Unsupported Modules

The following PACSystems modules do not support DO_IO scanning:

IC693BEM331 90-30 Genius Bus Controller
IC694BEM331 RX3i Genius Bus Controller
IC693BEM341 90-30 2.5 GHz FIP Bus Controller
IC693DNM200 90-30 Device Net Master

IC695PBM300 RX3i ProfiBus Master
IC695PBS301 RX3i ProfiBus Slave
IC687BEM731 90-70 Genius Bus Controller
IC697BEM731 90-70 Standard Width Genius Bus Controller

Sequential Event Recorder



Operation

A SER (Sequential Event Recorder) built-in function block instance collects a series of samples. Whenever SER receives power flow when scanned and the reset input is off, SER collects up to 32 contiguous or non-contiguous bits per sample, collecting one sample from each configured channel. Each SER can capture up to 1024 samples. SER reads the configured sample points and puts them in a circular list.
After the configured number of samples is taken and when the trigger conditions are satisfied (specified by the trigger mode parameter), the power flow output is turned on. The output continues to receive power flow regardless of the state of the enable input until R (reset input) receives power flow.

Tip: The transition of the output can be used to record the time that the last sample is taken or to initiate additional sampling.

SER operation is configured by a control block, which you can create using a series of Block Move (BLKMOV) instructions. On a Series 90-70 CPU, you can use a DATA_INIT_WORD instruction.
The SER can be configured for pre-, mid-, or post-trigger modes.
SER must be reset (by enabling the reset input power flow) before sampling is started. Resetting initializes the data block area.

Warning: If SER's status is not reset, SER executes with the current values in the data block, causing the current sample offset to be incorrect and causing invalid data in the data block.

SER's control block is scanned every time SER is executed in the reset, active, or triggered state. If you change a configuration parameter in the control block during logic execution, the change takes effect the next time the SER associated with that control block is scanned. If an error is encountered, operation stops and SER goes to the appropriate error state. To begin sampling again, you must correct the error and then reset SER (by enabling the reset input power flow).
If you select an input module to be scanned, the Controller will *not* verify that the module is a discrete input module, or that channel descriptions associated with the module have valid lengths and offsets based upon the module size. You must correctly set up the sampling of an input module. Although multiple channel descriptions can target an input module, the module is still only scanned once per SER execution.
SER can be placed in the normal logic or within a periodic subroutine. If placed in the logic, the resolution of the interval between scans is the resolution of the scan time, which can vary depending on the number and types of instructions active on any particular scan. If SER is embedded in a periodic subroutine, sampling rate is determined

by the periodic subroutine execution rate. If placed in an interrupt subroutine, the interval can be set to as little as 1ms, and the resolution will be highly repeatable at 1ms with little jitter.

Warning: Depending on the mix of the samples being collected, SER could take more than 1msec to execute.

Notes

- Execution time of one SER with a 1ms periodic subroutine can consume up to 50% of the CPU's resources. You should not plan on executing more than two SER instances within a 1ms periodic subroutine.
- Only the trigger sample is time-stamped. The trigger sample can be time-stamped in BCD (maximum resolution is 1s) or POSIX format (maximum resolution is 10ms). The time stamp is only placed once at the trigger point. The SER does not support more than one time stamp per recording.
- Controller-to-Controller synchronization is not supported.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 78 words	R, W	The beginning address of a 78-word array: the SER control block. The control block defines SER execution, sample configuration, and operation parameters.
R	Power flow		Reset input. When R is ON, SER is reset regardless of the state of the enable input. Sample buffer, trigger sample offset, trigger time, and current sample offset are all cleared to zero. SER remains in the reset state until R is turned OFF. The power flow output is turned off while in the reset state. When R is turned OFF, sampling resumes.
T	Power flow		Trigger input. If the trigger input mode is selected and SER is enabled, when T goes ON, SER transitions to the triggered state. The trigger time, trigger sample offset and a sample are recorded. The trigger sample is recorded regardless of the number of samples taken. Once triggered, the event recorder continues sampling until the number of samples After trigger is satisfied, at which time it stops collecting samples until R (reset input) is ON. If trigger mode is set to Full buffer, the trigger signal is ignored. For information on configuring trigger mode, see Trigger Mode.

Control Block

The control block of a built-in function block instance is a one-dimensional, 78-word array that defines information about the data capture and trigger mechanism for SER. In logic, each instance of SER must be associated with its own control block and data block.

To configure parameters for SER:

1. Add SER to your LD logic and define the starting address for its control block. Logic Developer - PLC automatically creates a 78-word array beginning at the starting address you define.
2. Initialize the array's values as defined in the table below.
   To initialize the array elements, you can:
   - Use block moves
   - (Series 90-70 only) Use DATA_INIT_WORD
   - Edit the Value property of each array element

Note: If you require x channels where x is not equal to 8, 16, 24, but is less than 32, you must select a number of channels which is greater than x and a multiple of 8, and fill in a null channel description for the unused channels. A null channel description has a segment selector of 0xFFh, a length parameter equal to the number of unused channels, and a 0 offset.

Word	*Parameter*	*Description*
0 (starting reference)	Status	(Read-only variable.) The current state of SER. Additional information is provided in status extra data (Word 1). **Note:** If an error is detected in the control block, the status is set to 6, the power flow output is cleared and no action occurs. Settings for status include: ▪ 0 = Reset ▪ 1 = Inactive ▪ 2 = Active ▪ 3 = Triggered ▪ 4 = Complete ▪ 5 = Overrun error ▪ 6 = Parameter error ▪ 7=Status error
1	Status extra data	(Read-only variable.) Additional state information about the SER function block instance. For information on the settings for this parameter, see Status extra data states.
2	Trigger mode	Defines conditions for SER to transition to the triggered state. Valid settings are:

		▪ 0 = Trigger input mode ▪ 1 = Full buffer mode In trigger input mode, if SER is enabled, a time stamp is generated when the trigger signal is activated. Sampling continues until the number of samples After trigger has been satisfied. When this occurs, the power flow output is activated. In Full buffer mode, the trigger signal is ignored. When SER is enabled, sampling continues until the sample buffer is filled. When this happens, the power flow output is activated. The number of samples parameter sets buffer size.
3	Trigger time format	Determines how the trigger time will be displayed. For BCD display, set this parameter to 0. For POSIX display, set this parameter to 1. See SER Trigger Timestamp Formats Example.)
4—7	Reserved	Words 4 through 7 are reserved and should be set to zero.
8	Number of channels (bits per sample)	The number of bits of data that are sampled and returned to the sample buffer for each execution of this instance of SER. Valid choices are 8, 16, 24 or 32 bits. The increment is in byte size (8 bits) and any unused channels must be configured with a null channel description. (See Words 14—77.)
9	Number of samples	The sample buffer size. Valid choices are 1 to 1024 samples. (Actual buffer size in bits is number of samples times number of channels.)
10	Number of samples after trigger	The number of samples that are collected after the trigger condition becomes true. This parameter can be set to a value between 0 and the number of samples. This parameter is valid only when the trigger mode is set to trigger input (0).
11	Input module slot	The location of the input module for data sampling (slot in the main rack). If the value is 0, scanning of the input module is disabled. When an input module is scanned, its values are stored locally and the values of the reference addresses configured for the module are not affected. To store values from the scanned input

		module into the data block sample buffer, a channel description must be provided. If the module is not present, or faulted, at the time of the scan the data returned is zero. A fault is not logged in the fault table if this occurs; fault indication is left to the IO scanner.
12	Data block segment selector	The data type allocated for the data block. For example, if you want to begin at %R0100, enter 08 for this parameter. Valid settings for this parameter include: %R (08h), %AI (0Ah), %AQ (0Ch). **Note:** "h" in the parameters (08h) stands for hexadecimal. Do not type the "h" when entering the parameter values.
13	Data block offset	The starting reference for the data block. This parameter is zero based. For example, if you want to begin at %R0100, enter 99 for this parameter. Be sure to allow enough memory for the entire data block.
14—77	Channel descriptions	There can be from 1 to 32 channel descriptions, depending on the number of channels being sampled and data length. Each channel description uses two words: 14-15, 16-17, 18-19, etc., until and including 76-77. Every channel description specifies the reference location associated with a particular channel. The first word of every channel description specifies the segment selector and Length, and the second word specifies the offset. Data is returned in the order defined in this section of the control block.
14, 16, 18, and every even-numbered word until and including 76	Channel segment selector/length for each channel description	Hexadecimal value that defines the segment selector and data length (in bits). The word contains two bytes. The most significant byte is the segment selector. The least significant byte is the data Length. The data length is useful for samples that are contiguous. The **segment selector** can be set to any discrete data type: %I (46h), %Q (48h), %M (4Ch), %T (4Ah), %G (56h), %S (54h), %SA (4Eh), %SB (50h), %SC (52h), Null selector (FFh), and input module selector (00h). The **Length** parameter can range from 1 to 32,

		but the sum of all of the lengths must not be greater than the number of channels parameter. A length greater than 1 allows multiple contiguous channels to be configured with a single channel description.
15, 17, 19, and every odd-numbered word until and including 77	Channel offset associated with the channel segment selector/length in the previous word	Hexadecimal value that defines the BIT offset for the data type or input module specified in the segment selector. This offset is zero-based. The range for this parameter varies, depending on the segment selector (data type and length). The offset indicates the location within the data table or input module at which to sample.

Status extra data states

The status extra data (Word 1 in the control block) provides additional state information on the status (word 0) of the SER built-in function block instance.

Value of Word 0	*State of Word 0*	*Description of Word 1 for each corresponding value of Word 0*
0	Reset state	The reset input is receiving power flow. Sample buffer, trigger sample offset, trigger time, and current sample offset are all cleared to zero. The output is held to no power flow. Transition to the *inactive state* occurs when the reset power flow is removed. Status extra data has no significance and will be cleared to zero.
1	Inactive	State between the reset state and the active state. No actions are performed in this state. The SER output is held to no power flow. Transition to the active state occurs when SER receives enable power flow.
2	Active	The Enable input has received power flow, but SER is not reset, in error, or triggered. One sample is recorded for each execution when SER is enabled. The output is held to no power flow. The trigger condition (specified by the trigger mode parameter) is monitored and will cause transition to the triggered state if conditions are true. If more than the number of samples has been taken, status extra data is set to 0x01; otherwise, it is set to 0x00.
3	Triggered	State if the trigger condition defined by trigger mode is true. Additional samples are taken depending upon the trigger mode and parameter settings. The output is held to no power flow. Transition to the Complete state will occur when all sampling is complete. If more than the number of samples has been taken, status extra data is set to 0x01; otherwise, it is set to 0x00.
4	Complete	All sampling is complete. The output receives power flow. Only transition to the reset state is allowed. If more than the number of

		samples has been taken, then status extra data is set to 0x01; otherwise, it is set to 0x00.
5	Overrun error	The control/data block has exceeded the end of its memory type. The output is held to no power flow. Only transition to the reset state is allowed. status extra data has no significance and will be cleared to zero.
6	Parameter error	There is an error in the control block or other operation parameters. The output is held to no power flow. Only transition to the reset state is allowed. The status extra data word contains the offset into the control block at which the parameter error occurred.
7	Status error	The status parameter is invalid. The output is held to no power flow. Only transition to the reset state is allowed. The invalid status value is stored in the status extra data location in the control block.

Data Block

The SER data block contains the sample buffer, sample offsets, and trigger information. The CPU supplies this information and you should only read from this data area. It is your responsibility to allocate enough register space for the data block.
The data block format is described in the following table. Column 1 is the offset from the starting reference defined by the data block segment selector (Word 12) and the data block offset (Word 13) in the control block of the SER instance.

Offset	*Parameter Description*
0	**Current sample offset number.** References the location where the most recent sample was placed. The parameter is zero-based. Valid ranges are –1 to 1023. Register location of sample = (num bytes per sample) * (offset parameter)/2 + (sample buffer starting register). **Note:** This value is not valid until a sample is taken. This value is set to –1 when the SER instance is reset through the reset input.
1	**Trigger sample offset number.** References the storage location of the sample obtained when the trigger condition transitioned to the True state. The parameter is zero-based. Valid ranges are 0 to 1023. Register location of sample = (num bytes per sample) * (offset parameter)/2 + (sample buffer starting register). **Note:** This value is not valid until the trigger condition is met. This value is set to 0 when the SER instance is reset (through the reset input).
2 through 5	**Trigger time.** Indicates the time, according to the time of day clock within the Controller, that the trigger condition transitioned to the true state within SER. The time value can be displayed in BCD format (default) or POSIX format. The format is determined by the trigger time format parameter in the control block. This value is initialized to zero upon activation of the reset input.

6 to the end of the data block	**Sample buffer.** The area of memory that holds the data samples. This area is set to zero when the reset parameter is energized. The sample buffer size varies, depending on the number of channels and sample size. The sample buffer is a circular buffer: when the last location is written, the next sample overwrites the sample in the first register. End of sample buffer = 5 + ({[(# of samples to be taken) * (# of channels to be sampled / 8)] +1} / 2)

Sampling Modes

The SER sampling mode is determined by the trigger mode (Word 2 in the built-in function control block) and number of samples after trigger (Word 10) parameters. You will need to interpret the contents of the sample buffer based on how you configured these parameters.

Trigger-controlled sampling

To configure pre-, mid-, and post-trigger sampling modes, select trigger mode (Word 2) = 0. The sampling mode is controlled by the number of samples after trigger (Word 10). In all cases, sampling starts when the enable signal goes high. When the trigger signal goes high, sampling continues until the number of samples after trigger is collected. SER's output signal goes high when sampling is completed.
If more than the configured number of samples (Word 9) is collected before the number of samples after trigger condition is satisfied, the buffer "wraps around," which means that SER returns to the beginning of the buffer and overwrites the initial samples.
When the trigger first transitions from OFF to ON, the trigger time is placed in a configured location.

Pre Trigger

When the trigger mode (Word 2) = 0, SER collects samples continuously until the trigger is detected.
To configure this mode, set Word 10 to a value of 0, so that when the trigger signal is activated, sampling stops and a time stamp is generated in separated data block offset (2-5). (All samples collected before the trigger.)

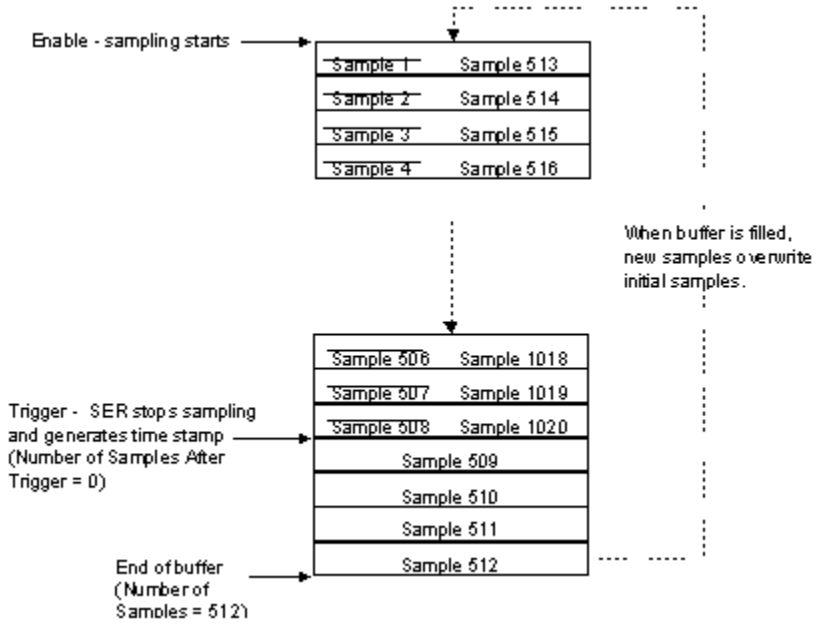
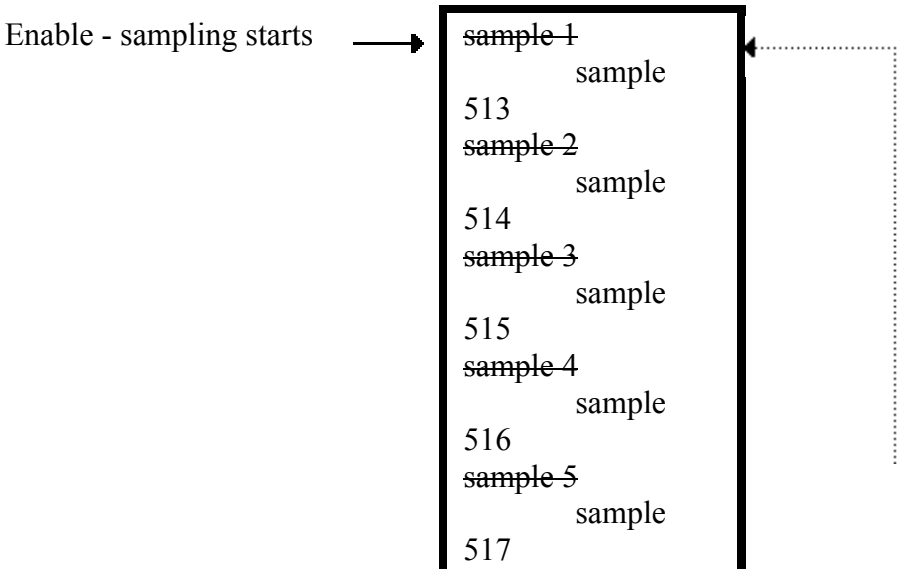


Figure 1. Example of pre-trigger SER sampling

Mid Trigger

Collects samples continuously until number of samples after trigger (Word 10) has been collected.
To configure this mode, set Word 10 to a value between 1 and (number of samples - 1). When the trigger signal is activated, sampling continues until the configured number has been collected. In the following example, number of samples after trigger is 12. When sampling is complete, the buffer will contain 500 pre-trigger samples and 12 post-trigger samples.



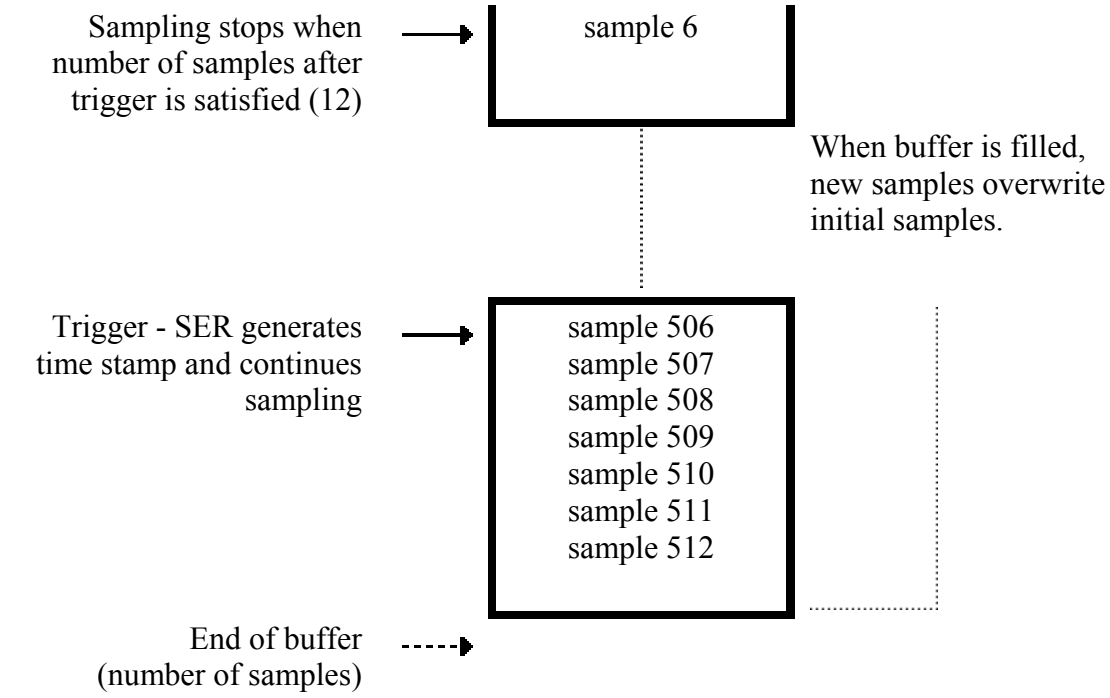
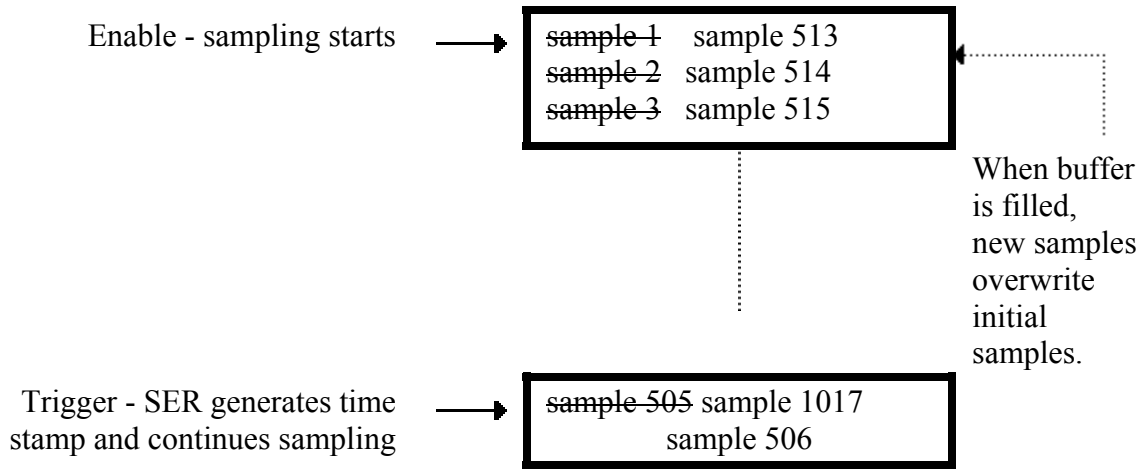


Figure 2. Example of mid-trigger SER sampling

Post Trigger

Collects sample continuously until number of samples is reached.
To configure this mode, set Word 10 to a value equal to the number of samples (Word 9). When the trigger signal is activated, sampling continues until the configured number has been collected. (All collected after the trigger.)



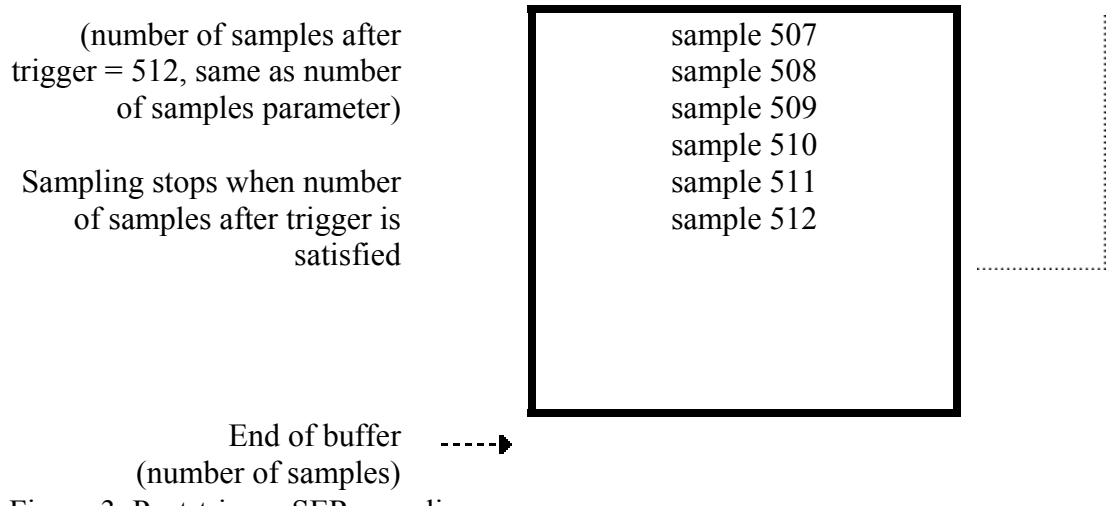


Figure 3. Post-trigger SER sampling

Full Buffer (trigger does not control sampling)

If the trigger mode is set to 1, the number of samples after trigger parameter (Word 10) is ignored and the trigger input signal has no effect on SER operation. When SER is enabled, sampling continues until the number of samples (Word 9) is collected, filling the sample buffer. When the buffer is full, sampling stops, a trigger time stamp is generated, and SER's power flow output goes high.

Example

The system has a 16-point discrete input module in rack 0 slot 4, has been executing long enough for 572 samples (512 + 60) to be taken.

- 512 is the sample buffer size (see Word 9 in the table below).
- 60 is the number of samples taken after the sample buffer was full. These new 60 samples overwrote the first 60 samples. The wrapping around is noted in Word 1.

The enable input is receiving power flow but the reset and trigger inputs are not. The control block has been set up as described in the following table:

Control Block for SER Example

Word	*Register (%R)*	*Parameter*	*Value (dec)*	*Value (hex)*	*Description*
0	100	Status	2	0002	SER is in the active state. This means SER is executing normally and taking a sample each time this instance of SER is encountered in logic.
1	101	Status extra data	1	0001	The extra status data indicates that more than 512 samples have been taken and, as a result, the sample buffer has already wrapped at least once.
2	102	Trigger mode	0	0000	The event recorder is configured to trigger based on the trigger input.

3	103	Trigger time format	0	0000	BCD
4	104	Reserved	0	0000	The reserved parameters are always set to 0.
5	105	Reserved	0	0000	
6	106	Reserved	0	0000	
7	107	Reserved	0	0000	
8	108	Number of channels	24	0018	Sample configuration consists of 24 bits of data.
9	109	Number of samples to be taken	512	0200	Sample buffer size is 512 samples. Note that the sample buffer is not 512 bytes. It is 512 x (24/8) = 1536 bytes or 768 words. (Each sample is 3 bytes long.)
10	110	Number of samples after trigger	12	000C	The number of samples to be collected after the trigger is 12.
11	111	Input module slot	4	0004	The input module in rack 0 slot 4 will be scanned so its current values are available for sampling.
12	112	Data block segment selector	8	0008	The data segment is 0x08 (%R).
13	113	Data block offset	200	00C8	The offset is 200, which places the start of the data block at %R0201. The offset is a zero-based value, but the register tables begin at %R0001. Therefore, the data block starting point is %R0001 + 200 = %R0201.

Note: The remaining words contain the channel descriptions. In this example, six channel descriptions have been defined.

14	114	Segment selector : length	17921	4601	Channel description 1: The first channel description selects the %I segment with a length of 1, and an offset of 0. This chooses %I0001 for channel 1.
15	115	Offset	0	0000	
16	116	Segment selector : length	-253	FF03	Channel description 2: The second channel description selects the NULL selector with length of 3, and offset of 0. The NULL selector causes channels 2 - 4 to be ignored or "skipped". These channels will always contain a sample value of zero.
17	117	Offset	0	0000	
18	118	Segment selector : length	3	0003	Channel description 3: The third channel description selects the input module selector with a length of 3, and offset of 12. The input module selector causes samples

19	119	Offset	12	0012	offset of 12. The input module selector causes samples to be taken from the input module. This channel description chooses the values in points 13, 14, and 15 of the input module for channels 5 - 7.
20	120	Segment selector : length	18434	4802	Channel description 4: The fourth channel description selects the %Q segment with a length of 2 and offset of 8. This chooses %Q0009 and %Q0010 for channels 8 and 9.
21	121	Offset	8	0008	
22	122	Segment selector : length	8	0008	Channel description 5: The fifth channel description is another input module selector. It has a length of 8, and offset of 0. This causes the values for points 1 to 8 of the input module to be placed in channels 10 - 17.
23	123	Offset	0	0000	
24	124	Segment selector : length	-249	FF07	Channel description 6: The sixth channel description is another NULL selector. It has a length of 7, and offset of 0. This NULL channel description causes channels 18 - 24 to be filled with zeros. This last channel description is required to pad the sample buffer out to the 24 bits specified in the number of channels parameter. Since all 24 channels are configured, no more channel descriptions are needed.
25	125	Offset	0	0000	

Sample Contents **for SER Example**

The following table summarizes the values contained in a single sample based upon the channel descriptions in the sample control block.

Channel number	*Channel contents*
1	%I0001
2 - 4	Zeros
5	Input module point 13
6	Input module point 14
7	Input module point 15
8	%Q0009
9	%Q0010
10 - 17	Input module points 1 - 8
18 - 24	Zeros

Data Block for SER Control Block Example

The following table lists the format of the data block resulting from the example control block. Note that it begins at register 201 as described by the segment offset parameters (Words 12 and 13) in the control block example.

Offset	*Register*	*Parameter Description*	*Value (dec)*	*Value (hex)*
0	%R0201	Current sample offset number	59	003B

1	202	Trigger sample offset number	0	0000
2 - 5	203 – 206	Trigger time (BCD)	0 0 0 0	0000 0000 0000 0000
6 - 768	207 – 975	Sample buffer	sample data	sample data

Current sample offset is 59, which means that the 59th sample is the last sample placed in the sample buffer (not 59 registers). With 3 bytes per sample, the current offset is actually at 59 * 3 = 177 bytes or the high byte of the 89th register. Since the trigger conditions have not been met, the trigger sample and trigger time are 0 and the output is not set. The sample buffer contains 512 samples where 59 is the newest sample and 60 is the oldest sample.

SER Trigger Timestamp Formats Example

Example trigger time of November 3, 1998 at 8:34:05:16 a.m.

BCD format:

```
struct time_of_day_clk_rec {
     unsigned char      seconds;
     unsigned char      minutes;
     unsigned char      hours;
     unsigned char      day_of_month;
     unsigned char      month;
     unsigned char      year;
};
```

Register	*Parameter*	*Value (dec)*	*Value (hex)*
%R0203	Minutes/seconds	13317	3405
%R204	Day of month/hours	776	0308
%R205	Year/month	-26607	9811
%R206	Unused	0	0

POSIX format:

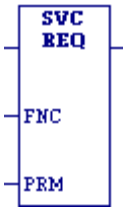
```
struct timespec {
    long    tv_sec;
/* Number of seconds since January 1, 1970 */
    long    tv_nsec;
/* Number of nanoseconds into next seconds */
};
```

Register	*Parameter*	*Value (dec)*	*Value (hex)*
%R0203	Seconds low Word	-7,811	e17d
%R204	Seconds high Word	13,845	3615
%R205	Nano-seconds low Word	26,624	6800
%R206	Nano-seconds high Word	2,441	0989

CPU Support

SER is supported for Series 90-30 version 9.00 or later of CPU350 or later.

Service Request



Operation

When SVC_REQ receives power flow, it requests the Controller to perform the special Controller service identified by the FNC operand.
Parameters for SVC_REQ are located in the parameter block, which begins at the reference identified by the PRM operand. The number of 16-bit references required depends on the type of special Controller service being requested. The parameter block is used to store both the instruction's inputs and outputs.
SVC_REQ passes power flow **unless** an incorrect instruction number, incorrect parameters, or out-of-range references are specified. Various specific SVC_REQ instructions have additional causes for failure.

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) Most of the SVC_REQ instructions, when used inside an interrupt block being executed, may cause the block to be preempted when a new, incoming interrupt has the same priority.

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
FNC	INT variable or constant	data flow, I, Q M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Instruction number; Service Request number. The constant or reference that identifies the requested service.
PRM	INT variable	I, Q M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first WORD in the parameter block for the requested service. Successive 16-bit locations store additional parameters. The number of WORDs required for SVC_REQ depends on the value of the FNC operand. **Note:** If you use a symbolic or I/O variable, ensure that its Array Dimension 1 property is set to a value large enough to contain the entire parameter block.

Available SVC_REQ instructions

SVC_REQ 1 Change/Read Constant Sweep Timer
SVC_REQ 2 Read Window Modes and Times Values
SVC_REQ 3
- For PACSystems and Series 90-70 and 90-30: Change Programmer Communications Window Mode and Timer Value.
- For VersaMax: Change Programmer Communications Window Mode

SVC_REQ 4
- For PACSystems and Series 90-70 and 90-30: Change System Communications Window Mode and Timer Value.
- For VersaMax: Change System Communications Window Mode

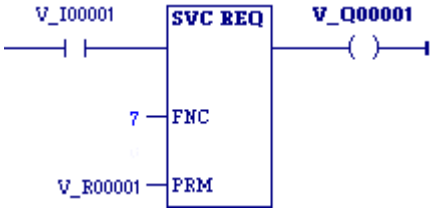
SVC_REQ 5 (PACSystems and Series 90-70 only): Change Background Task Window Mode and Timer Value
SVC_REQ 6 Change/Read Number of Words to Checksum
SVC_REQ 7 Read or Change the Time-of-Day Clock
SVC_REQ 8 Reset Watchdog Timer
SVC_REQ 9 Read Sweep Time from Beginning of Sweep
SVC_REQ 10: Read Folder Name
SVC_REQ 11 Read Controller ID
SVC_REQ 12 (PACSystems, Series 90-70, and Series 90-30 only): Read Controller Run State
SVC_REQ 13 Shut Down (Stop) Controller
SVC_REQ 14 Clear Fault Tables
SVC_REQ 15 Read Last-Logged Fault Table Entry
SVC_REQ 16 Read Elapsed Time Clock
SVC_REQ 17 (PACSystems and Series 90-70 only): Mask/Unmask I/O Interrupt
SVC_REQ 18 Read I/O Override Status
SVC_REQ 19 (PACSystems and Series 90-70 only): Set Run Enable/Disable
SVC_REQ 20 (PACSystems and Series 90-70 only): Read Fault Tables
SVC_REQ 21 (PACSystems and Series 90-70 only): User-Defined Fault Logging
SVC_REQ 22 (PACSystems and Series 90-70 only): Mask/Unmask Timed Interrupts
SVC_REQ 23 Read Master Checksum
SVC REQ 24 (PACSystems and Series 90-30 only): Reset Smart Module or Daughterboard
SVC_REQ 25 (PACSystems and Series 90-70 only): Disable/Enable EXE Block and Standalone C Program Checksums
SVC_REQ 26
- SVC_REQ 26 or 30 (Series 90-30 and VersaMax only): Interrogate I/O
- SVC_REQ 26 (PACSystems RX7i and Series 90-70 only): Role Switch

SVC_REQ 27 (PACSystems RX7i and Series 90-70 only): Write to Reverse Transfer Area
SVC_REQ 28 (PACSystems RX7i and Series 90-70 only): Read from Reverse Transfer Area
SVC_REQ 29 (PACSystems, Series 90-30, and VersaMax only): Read Elapsed Power Down Time
SVC_REQ 32 (PACSystems and Series 90-70 only): Suspend/Resume I/O Interrupt

SVC_REQ 36 (Series 90-70 only): Read from/Write to Bulk Memory Area
SVC_REQ 39 (Series 90-70 only): ESCM Port Status
SVC_REQ 43 (PACSystems RX7i and Series 90-70 only): Disable Data Transfer Copy in Backup
SVC_REQ 44 (Series 90-70 only): Logic Driven Dynamic Ethernet Global Data
SVC_REQ 45 (PACSystems RX3i and Series 90-30 only): Skip Next Output and Input Scan (Suspend I/O)
SVC_REQ 46 (Series 90-30 only): Access Fast Backplane Status
SVC_REQ 48 (Series 90-30): Auto Reset
SVC_REQ 50 (PACSystems): Read Elapsed Time Clock (Two DWORDs)
SVC_REQ 51 (PACSystems): Read Sweep Time from Beginning of Sweep (DWORD)
SVC_REQ 52 (VersaMax Micro CPUs release 3.00 or later): Read from Flash
SVC_REQ 53 (VersaMax Micro CPUs release 3.00 or later): Write to Flash
SVC_REQ 55 (PACSystems): Set Application Redundancy Mode
SVC_REQ 56 (PACSystems): Logic Driven Read of Nonvolatile Storage
SVC_REQ 57 (PACSystems): Logic Driven Write to Nonvolatile Storage

Example

When the enabling input %I0001 is ON, SVC_REQ instruction number 7 is called, with the parameter block starting at %R0001. If the operation succeeds, output coil %Q0001 is set ON.



CPU Support

SVC_REQ is supported for:

- All PACSystems CPUs
- Series 90-70 CPUs
- Series 90-30 CPUs
- All VersaMax CPUs

However, not all Service Request values (FNC operand) are supported for each and every one of those CPUs. Documentation for each specific SVC_REQ instruction specifies the CPU support for that SVC_REQ instruction.

SVC_REQ 1: Change/Read Constant Sweep Timer

CPU Support

SVC_REQ 1 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Note: Series 90-70 Release 6.0 or later CPUs return 0 for Normal Sweep, 1 for Constant Sweep, and 2 for Microcycle when timer state and values are requested.

Operation

Use SVC_REQ instruction 1 to:

- Disable Constant Sweep mode
- Enable Constant Sweep mode and use the old Constant Sweep timer value
- Enable Constant Sweep mode and use a new Constant Sweep timer value
- Set a new Constant Sweep timer value only
- Read Constant Sweep mode state and timer value.

The parameter block has a length of two words used for both input and output.

SVC_REQ executes successfully unless:

- A number other than 0, 1, 2, or 3 is entered as the requested operation:
- The scan time value is greater than 2550 ms (2.55 seconds) for a PACSystems, Series 90-30, and a Series 90-70, and greater than 500ms (0.5 seconds) for a VersaMax.
- Constant sweep time is enabled with no timer value programmed or with an old value of 0 for the timer.

To disable Constant Sweep mode:

Enter SVC_REQ 1 with this parameter block:

Address	0
Address + 1	Ignored

To enable Constant Sweep mode and use the old timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	0

If the timer value does not already exist, entering 0 causes the instruction to set the OK output to OFF.

To enable Constant Sweep mode and use a new timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	New timer value **Note:** If the timer value does not already exist, entering 0 causes the instruction to set the OK output to OFF.

To change the timer value without changing the selection for sweep mode state:

Enter SVC_REQ 1 with this parameter block:

Address	2
Address + 1	New timer value

To read the current timer state and value without changing either:

Enter SVC_REQ 1 with this parameter block:

Address	3
Address + 1	ignored

Note: Series 90-30 Release 8 and higher CPUs provide the return values 0 for Normal Sweep, 1 for Constant Sweep.

Output

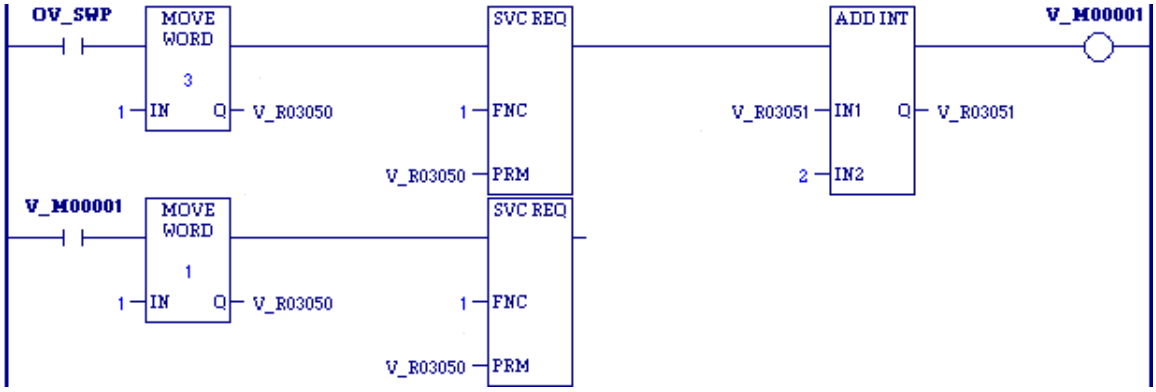
SVC_REQ 1 returns the timer state and value in the same parameter block references:

Address	0 = disabled 1 = enabled **Note:** For PACSystems and Release 6 and higher of Series 90-70, ▪ 0 = Normal Sweep ▪ 1 = Constant Sweep ▪ 2 = Microcycle (Series 90-70 only.)
Address + 1	Current timer value

If the word address + 1 contains the hexadecimal value FFFF, no timer value has ever been programmed.

Example

If contact OV_SWP is set, the Constant SweepTimer is read, the timer is increased by two milliseconds, and the new timer value is sent back to the Controller. The parameter block is at location %R3050. The example logic uses discrete internal coil %M0001 as a temporary location to hold the successful result of the first rung line. On any sweep in which OV_SWP is not set, %M00001 is turned off.


OV_SWP
MOVE WORD
3
1
IN
Q
V_R03050
SVC REQ
1
FNC
V_R03050
PRM
ADD INT
V_R03051
IN1
Q
V_R03051
2
IN2
V_M00001
V_M00001
MOVE WORD
1
1
IN
Q
V_R03050
SVC REQ
1
FNC
V_R03050
PRM

SVC_REQ 2: Read Window Modes and Time Values

CPU Support

SVC_REQ 2 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Note: There are several Output differences between Series 90-30 and VersaMax support for SVC_REQ 2.

Operation

Use SVC_REQ 2 to obtain the current window mode and time values for the programmer communications window and the system communications (and, for all Series 90-70 CPUs, the background task window).
The parameter block has a length of three words. All parameters are output parameters. It is not necessary to enter values in the parameter block for this instruction.

Output

Address	*Window*	*High Byte*	*Low Byte*
address	Programmer Window	Mode	Value in ms
address+1	System Communications Window	Mode	Value in ms
address+2	(Series 90-30 and VersaMax) Reserved	All zeroes	All zeroes
	(PACSystems and Series 90-70) Background Window	Mode	Value in ms

Note: A window is disabled when the time value is zero.

PACSystems, Series 90-30 and Series 90-70 Mode Values

Mode Name	*Value*	*Description*
Limited Mode	0	The execution time of the window is limited to its respective default value or to a value defined using SVC_REQ 3 for the programmer communications window or SVC_REQ 4 for the systems communications window. The window will terminate when it has no more tasks to complete.
Constant Mode	1	Each window will operate in a Run to Completion mode, and the CPU will alternate among the three windows for a time equal to the sum of each window's respective time value. If one window is

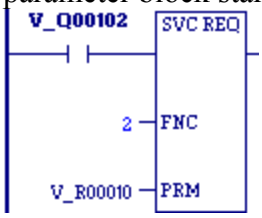
		placed in Constant mode, the remaining two windows are automatically placed in Constant mode. If the CPU is operating in Constant Window mode and a particular window's execution time is not defined using the associated SVC_REQ instruction, the default time for that window is used in the constant window time calculation.
Run to Completion Mode	2	Regardless of the window time associated with a particular window, whether default or defined using a service request instruction, the window will run until all tasks within that window are completed.

VersaMax Mode Values

Mode Name	*Value*	*Description*
Limited Mode	0	The execution time of the window is limited to 6ms. The window terminates when it has no more tasks to complete or after 6ms elapses.
Run to Completion Mode	2	Regardless of the time assigned to a window, it runs until all tasks within that window are completed (up to 400ms).

Example

When %Q00102 is set, the CPU places the current time values of the windows in the parameter block starting at location %R0010.



SVC_REQ 3

CPU Support

SVC_REQ 3 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Note: There are several differences between PACSystems/Series 90-70/Series 90-30 support and VersaMax support for SVC_REQ 3.

PACSystems, Series 90-70, and Series 90-30 Operation

SVC_REQ 3: Change Controller communications window mode and timer value

Use SVC_REQ 3 to change the Controller communications window mode and timer value. The change takes place during the next CPU sweep after the instruction is called. SVC_REQ 3 passes power flow to the right unless one of the following occurs:

- A mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.
- (Series 90-70 CPU only): The Controller is in Microcycle Sweep Mode and a mode other than 1 (Constant) is selected

The parameter block has a length of one word.

To disable the programmer communications window:

Enter SVC_REQ 3 with this parameter block:

Address	High Byte	*Low Byte*
Address	0	0

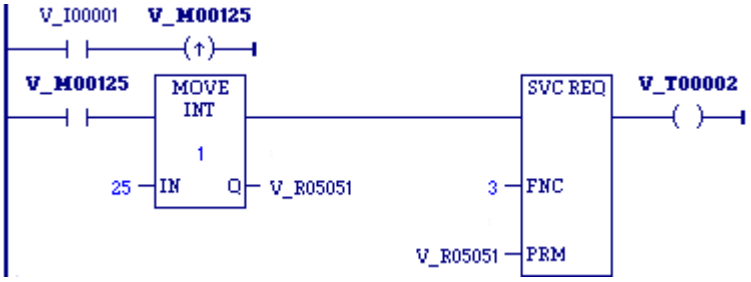
To reenable the programmer communications window mode or change it:

Enter SVC_REQ 3 with this parameter block:

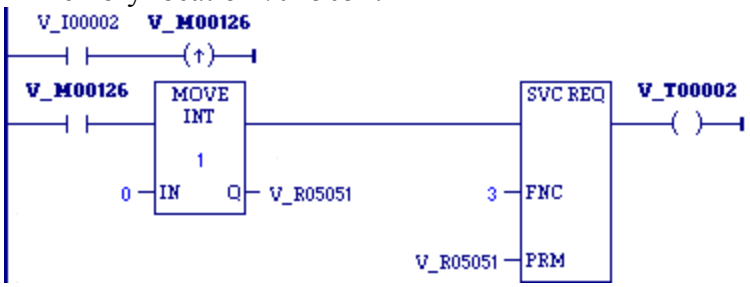
Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

Series 90-30 Examples

When %M0125 transitions on, the programmer communications window is enabled in mode 0 (Limited) and assigned a value of 25 ms. The parameter block is in memory location %R5051.

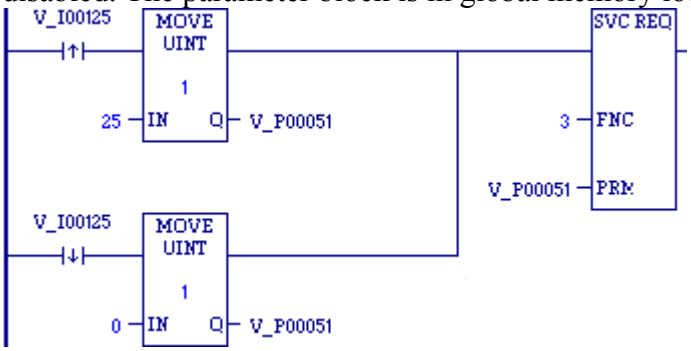


To disable the programmer communications window, use SVC_REQ 3 to assign a value of zero (0). In this example, when %M0126 transitions on, the programmer communications window is enabled and assigned a value of 0 ms. The parameter block is in memory location %R5051.



Series 90-70 Example

When enabling input %I00125 transitions on, the programmer communications window is enabled and assigned a value of 25 ms. When the contact transitions off, the window is disabled. The parameter block is in global memory location %P00051.



VersaMax Operation

SVC_REQ 3: Change programmer communications window mode

Use SVC_REQ 3 to change the programmer communications window mode. The change takes place during the next CPU sweep after the instruction is called. The time of the window cannot be changed; it is always 6ms.

SVC_REQ 3 passes power flow to the right unless a mode other than 0 (Limited), or 2 (Run-to-Completion) is selected.

The parameter block has a length of one word.

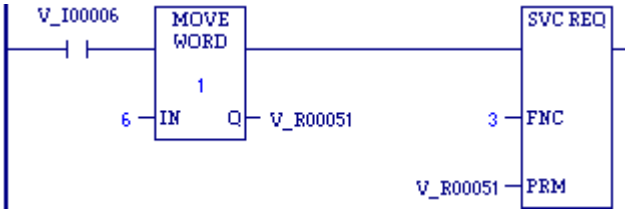
To change the programmer communications window mode:

Enter SVC_REQ 3 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	6

VersaMax Example

When enabling input %I006 goes ON, the programmer communications window is enabled and assigned a value of 6ms. The parameter block is in reference memory location %R0051.



SVC_REQ 4

CPU Support

SVC_REQ 4 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Note: There are several differences between PACSystems/Series 90-70/Series 90-30 support and VersaMax support for SVC_REQ 4.

PACSystems, Series 90-70, and Series 90-30 Operation

SVC_REQ 4: Change system communications window mode and timer value

Use SVC_REQ 4 to change the system communications window mode and timer value. The change takes place during the next CPU sweep after the instruction is called.
SVC_REQ 4 passes power flow to the right unless one of the following occurs:

- A mode other than 0 (limited), 1 (constant), or 2 (run-to-completion) is selected.
- (Series 90-70 only) The Controller is in microcycle sweep mode and a mode other than 1 (constant) is selected.

The parameter block has a length of one word.

To disable the system communications window:

Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

To enable the system communications window mode:

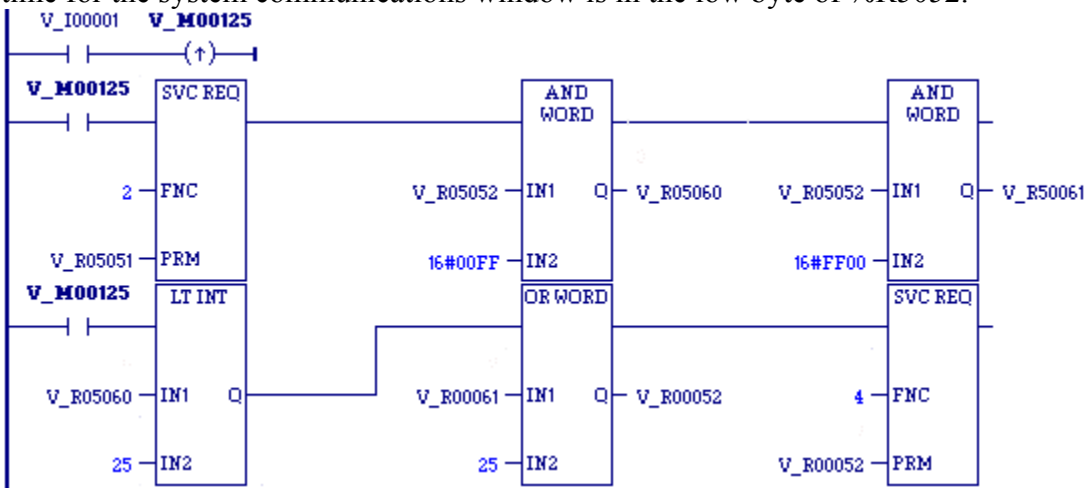
Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

PACSystems/Series 90-70/Series 90-30 Example

When enabling output %M0125 transitions on, the mode and timer value of the system communications window is read. If the timer value is greater than or equal to 25 ms, the value is not changed. If it is less than 25 ms, the value is changed to 25 ms. In either case, when the rung completes execution the window is enabled. The parameter block for all three windows is at location %R5051. Since the mode and timer for the system

communications window is the second value in the parameter block returned from the Read Window Values instruction (SVC_REQ 2), the location of the existing window time for the system communications window is in the low byte of %R5052.



VersaMax Operation

SVC_REQ 4: Change system communications window mode

Use SVC_REQ 4 to change the system communications window mode. The change takes place during the next CPU sweep after the instruction is called. The time of the window cannot be changed; it is always 6ms.

SVC_REQ 4 passes power flow to the right unless a mode other than 0 (limited), or 2 (run-to-completion) is selected.

The parameter block has a length of one word.

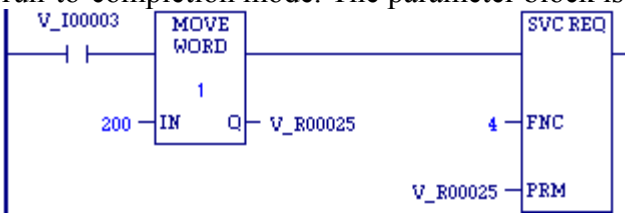
To change the system communications window mode:

Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	6

VersaMax Example

When enabling input %I003 is ON, the system communications window is changed to run-to-completion mode. The parameter block is in reference memory location %R0025.



SVC_REQ 5: Change Background Task Window Mode and Timer Value

CPU Support

SVC_REQ 5 is supported for:

- PACSystems CPUs.
- Series 90-70 Version 2.02 or later CPUs.

Operation

Use SVC_REQ 5 to change the background task window mode and timer value. The change takes place during the next CPU sweep after the instruction is called. SVC_REQ 5 always passes power flow to the right. The parameter block has a length of one word.

To disable the background task window:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

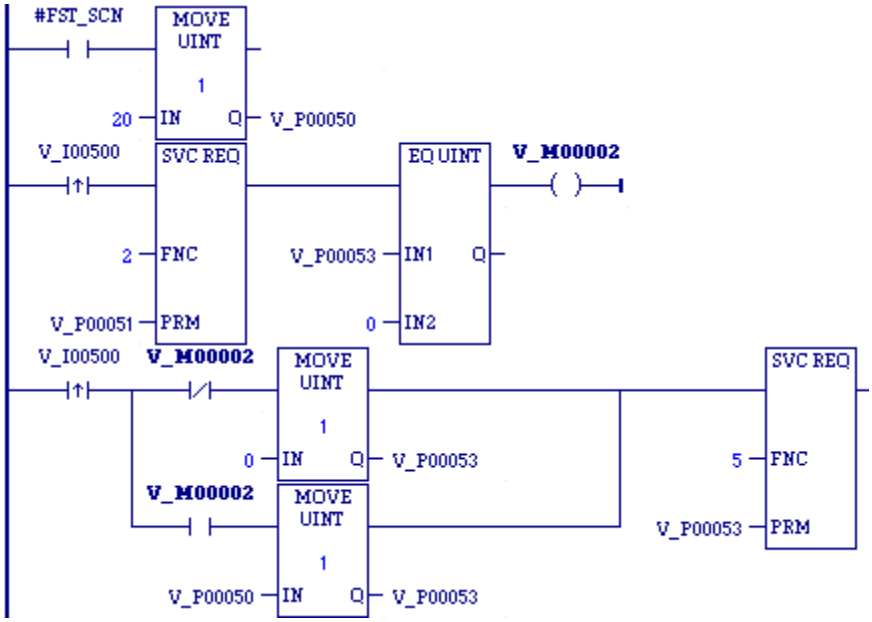
To enable the background task window mode:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode	1ms ≤ value ≤ 255ms

Example

When enabling contact #FST_SCN is set in the first scan, the MOVE instruction establishes a default value of 20 ms for the background task window, using a parameter block beginning at %P00050. Later in the logic, when input %I00500 transitions on, the state of the background task window toggles on and off. The parameter block for all three windows is at location %P00051. The time for the background task window is the third value in the parameter block returned from the read window values instruction (instruction #2); therefore, the location of the existing window time for the system communications window is %P00053.


#FST_SCN
MOVE
UINT
1
20
IN
Q
V_P00050
V_I00500
SVC REQ
EQ UINT
V_M00002
2
FNC
V_P00053
IN1
Q
V_P00051
PRM
0
IN2
V_I00500
V_M00002
MOVE
UINT
1
0
IN
Q
V_P00053
SVC REQ
5
FNC
V_M00002
MOVE
UINT
V_P00053
PRM
1
V_P00050
IN
Q
V_P00053

SVC_REQ 6: Change/Read Number of Words to Checksum

CPU Support

SVC_REQ 6 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 CPUs, but not Series 90 Micro CPUs.

Operation

Use SVC_REQ 6 to read the current word count in the logic to be checksummed or set a new word count. (In a PACSystems or Series 90-70, by default, 16 words are checked.) The instruction is successful unless some number other than 0 or 1 is entered as the requested operation.
The parameter block has a length of 2 words.

To read the word count:

Enter a zero in the first word of the parameter block.

Address	0
Address + 1	Ignored

The instruction returns the current checksum (word count) in the second word of the parameter block. No range is specified for the read instruction; the value returned is the number of words currently being checksummed.

Address	0
Address + 1	Current word count

To set a new word count:

Enter a one in the first word of the parameter block and the new word count in the second word.

Address	1	
Address + 1	New word count	Series 90-20 CPU211: 0 through 4
		Series 90-30 CPUs: 0 through 32 (default: 8). 0 disables the checksum. A value greater than 32 creates an error condition, which results in no power flow from the Service Request's OK output.
		PACSystems and Series 90-70 CPUs: 0 disables the checksum. Any other value is rounded up to the next multiple of 8.

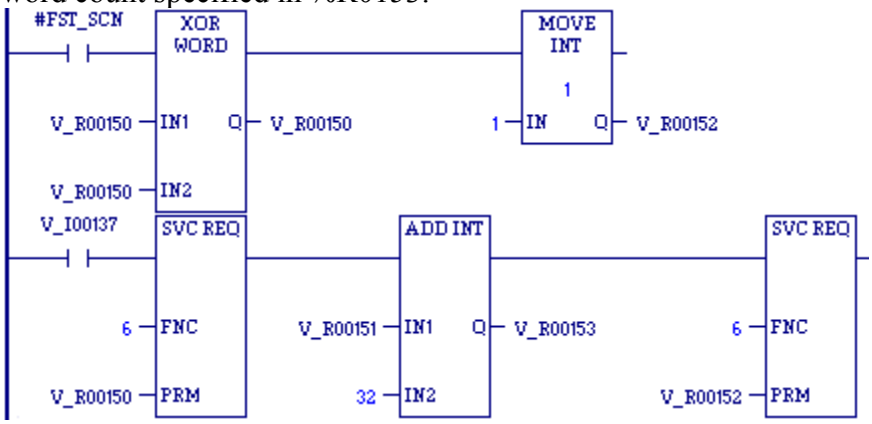
		VersaMax: 0 or 32

The Controller changes the number of words to be checksummed to the value given in the second word of the parameter block.

Examples

Example for VersaMax and Series 90-30 CPUs

When enabling contact #FST_SCN is set, the parameter blocks for the checksum instruction are built. Later in the logic, if input %I0137 turns on, the SVC_REQ reads the number of words being checksummed. The parameter block for the Read instruction is located at %R0150-151. The ADD instruction adds 32 to the current word count in %R0151 and places the result in %R0153. The parameter block for the Change instruction is located at %R00152-153. The second SVC_REQ then changes to the new word count specified in %R0153.

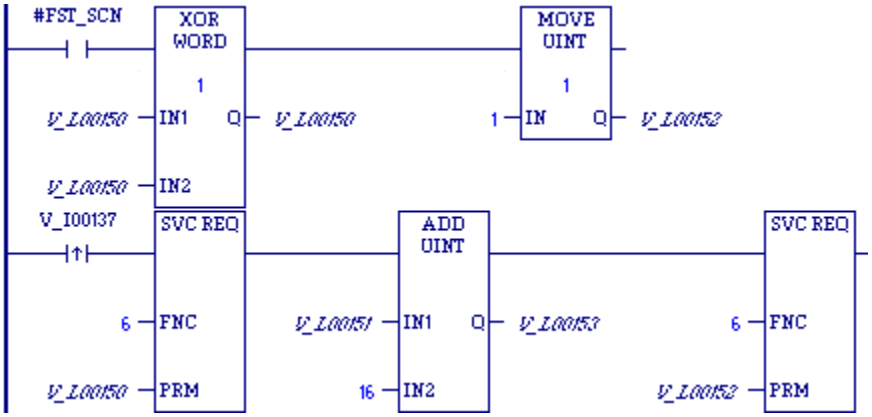


The example's parameter blocks are located at address %R0150-153. They have the following contents:

%R0150	0 = read current count
%R0151	ignored
%R0152	1 = set current count
%R0153	32 = new count to be set

Example for Series 90-70 CPUs

When enabling contact #FST_SCN is set, the parameter blocks for the checksum task instruction are built. Later in the logic, when input %I00137 transitions on, the number of words being checksummed is read from the Controller operating system. This number is increased by 16, with the results of the ADD_UINT instruction being placed in the "hold new count for set" parameter. The second service request instruction requests the Controller to set the new word count.



The example parameter blocks are located at address %L00150. They have the following contents:

%L00150	0 = read current count
%L00151	hold current count
%L00152	1 = set current count
%L00153	hold new count for set

SVC_REQ 7: Read or Change the Time-of-Day Clock

CPU Support

SVC_REQ 7 is supported for:

- PACSystems CPUs.
- Series 90-30 CPU331 and later.
- Series 90-70 CPUs, Version 2.02 or later, but the POSIX format is supported only for version 7.92 and later CPUs.
- VersaMax CPUs.
- 28-point Series 90 Micro Controller CPUs (that is, IC693UAA007, IC693UDR005 and IC693UDR010) and the 23-point Series 90 Micro Controller CPUs (IC693UAL006).

The CPU families support various parameter block formats.

Operation

Use SVC_REQ 7 to read or change the time of day clock in the CPU. The instruction is successful unless:

- An invalid number is entered for the requested operation.
- An invalid data format is specified.
- Data is provided in an unexpected format.

Parameter Block Formats

In the first two words of the parameter block, you specify whether to read or set the time and date, and which format to use.

Note: On PACSystems and VersaMax , you also specify whether you use a 2-digit year format or a 4-digit year.

Address	*2-Digit Year Format*	*4-Digit Year Format* **Note:** Supported only for PACSystems and VersaMax.
Address (word 1)	0 = read time and date	0 = read time and date
	1 = set time and date	1 = set time and date
Address+1 (word 2)	0 = (PACSystems and Series 90-70 only.) numeric data format	80h = (PACSystems only.) numeric data format
	1 = BCD format	81h = BCD format
	2 = (PACSystems and Series 90-70 only.) unpacked BCD format	82h = (PACSystems only.) unpacked BCD format
	3 = packed ASCII format (with embedded spaces and colons)	83h = packed ASCII format

	4 = (PACSystems and Release 7.92 and later 90-70 CPUs only.) POSIX format	n/a
Address+2 to the end (word 3)	Data	Data

Words 3 to the end of the parameter block contain output data returned by a read instruction, or new data being supplied by a change instruction. In both cases, format of these data words is the same. When reading the date and time, words (address + 2) to the end of the parameter block are ignored on input.

The format and length of the parameter block depends on the data format and number of digits required for the year:

Data Format and N-digit Year	*Length of parameter block (number of words)*
BCD, 2-digit year	6
(PACSystems and VersaMax only.) BCD, 4-digit year	6
(PACSystems and Series 90-70 only.) POSIX format	6
(PACSystems and Series 90-70 only.) Unpacked BCD2	9
(PACSystems only.) Unpacked BCD4	10
(PACSystems and Series 90-70 only.) Numeric	9
Packed ASCII, 2-digit year	12
(PACSystems and VersaMax only) Packed ASCII, 4-digit year	13

In any format,

- Hours are stored in 24-hour format.
- Day of the week is a numeric value ranging from 1 (Sunday) to 7 (Saturday).

Value	*Day of the Week*
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Note: Series 90-70 and Series 90-30 CPUs supports only 2-digit years. PACSystems and VersaMax supports 2-digit years and 4-digit years in both the BCD and packed ASCII formats.

BCD, 2-Digit Year

In BCD format, each time and date item occupies one byte, so the parameter block has six words. The last byte of the sixth word is not used. When setting the date and time, this byte is ignored; when reading date and time, the instruction returns a null character (00).

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	0 (read)
1 (BCD format)	address+1	1 (BCD format)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
month	year	address+2	07 (July)	05 (year)
hours	day of month	address+3	14 (hours)	03 (day)
seconds	minutes	address+4	30 (seconds)	45 (minutes)
(null)	day of week	address+5	00	01 (Sunday)

(PACSystems and VersaMax only) BCD, 4-Digit Year

In BCD format, each time and date item occupies one byte, so the parameter block has six words. All bytes are used.

Note: This is supported for PACSystems and VersaMax only.

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	00 (read)
81h (BCD format, 4-digit)	address+1	81h (BCD, 4-digit)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
year	year	address+2	20 (year)	05 (year)
day of month	month	address+3	03 (day)	07 (July)
minutes	hours	address+4	45 (minutes)	14 (hours)
day of week	seconds	address+5	01 (Sunday)	30 (seconds)

(PACSystems and Release 7.92 and later Series 90-70 only) POSIX

The POSIX format of the Time-of-Day clock uses two signed 32-bit integers (two DINTs) to represent the number of seconds and nanoseconds since midnight January 1, 1970. Reading the clock in POSIX format might make it easier for your application to

calculate time differences. This format can also be useful if your application communicates to other devices using the POSIX time format. To read and/or change the date and time using POSIX format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example* December 1st, 2000 at 12 noon
1 = change or 0 = read	address	0
4 (POSIX format)	address+1	4
Seconds (LSW)	address+2	975,672,000
(MSW)	address+3	
Nanoseconds (LSW)	address+4	0
(MSW)	address+5	

The resolution of the CPU's POSIX clock is 100 microseconds. The overall accuracy of the POSIX clock is +/- 0.01%. The accuracy of an individual sample of the POSIX clock is approximately 105 microseconds; however, the SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. As a result, the clock sample returned by SVC_REQ 7 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic. Interrupts can also delay when a new clock time is installed using SVC_REQ 7.
The CPU's maximum POSIX clock value is 7FFFFFFF (hexadecimal) seconds and 999,999,999 (decimal) nanoseconds, which corresponds to January 19th, 2038 at 3:14 am. This is the maximum POSIX value that SVC_REQ 7 will accept for changing the clock. This is also the maximum POSIX value SVC_REQ 7 will return once the Time-Of-Day clock passes this date.
If SVC_REQ 7 receives an invalid POSIX time to write to the clock, it does not change the Time-Of-Day clock and disables its power-flow output.

Note: When reading the CPUclock in POSIX format, the data returned is not easily interpreted by a human viewer. If desired, it is up to the application logic to convert the POSIX time into year, month, day of month, hour, and seconds.

(PACSystems and Series 90-70 only) Unpacked BCD (2-Digit Year)

In Unpacked BCD format, each digit of the time and date items occupies the low-order four bits of a byte. The upper four bits of each byte are always zero. This format requires nine words.

Parameter Block Format	*Address*	*Example (Thurs., Dec. 8, 2002, at 9:34:57 a.m.*
1 = change or 0 = read	address	0h
2 (Unpacked BCD format)	address+1	2h

High Byte	*Low Byte*		*High Byte*	*Low Byte*
	year	address+2	00h	02h
	month	address+3	01h	02h
	day of month	address+4	02h	08h
	hours	address+5	00h	09h

	minutes	address+6	03h	04h
	seconds	address+7	05h	07h
	day of week	address+8	00h	05h

(PACSystems and Series 90-70 only) Numeric, 2-digit year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example Wed., June 15, 2005, at 12:15:30 a.m.*
1 = change or 0 = read	address	0
0 (Numeric format, 2- digit year)	address+1	2

High Byte	*Low Byte*		*Value*
	year	address+2	05
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15
	seconds	address+7	30
	day of week	address+8	04

(PACSystems only) Numeric, 4-digit year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	*Address*	*Example Wed., June 15, 2005, at 12:15:30 a.m.*
1 = change or 0 = read	address	0
80h (Numeric format, 4- digit year)	address+1	80h

High Byte	*Low Byte*		*Value*
	year	address+2	05
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15

	seconds	address+7	30
	day of week	address+8	04

Packed ASCII, 2-Digit Year

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. Spaces and colons are embedded into the data to format it for printing or display. ASCII format for a 2-digit year requires 12 words in the parameter block.

Parameter Block Format	*Address*	*Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)*
1 = change or 0 = read	address	0h (read)
3 (ASCII format)	address+1	3h (ASCII format)

High Byte	*Low Byte*		*High Byte*	*Low Byte*
year	year	address+2	35h (5)	30h (0)
month	(space)	address+3	31h (1)	20h (space)
(space)	month	address+4	20h (space)	30h (0)
day of month	day of month	address+5	35h (5)	30h (leading 0)
hours	(space)	address+6	32h (2)	20h (space)
: (colon)	hours	address+7	3Ah (:)	33h (3)
minutes	minutes	address+8	33h (3)	31h (1)
seconds	: (colon)	address+9	32h (2)	3Ah (:)
(space)	seconds	address+10	20h (space)	35h (5)
day of week	day of week	address+11	32h (2 = Mon.)	30h (leading 0)

(PACSystems and VersaMax only) Packed ASCII, 4-Digit Year

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. Spaces and colons are embedded into the data to format it for printing or display. ASCII format for a 4-digit year requires 13 words in the parameter block.

Note: This is supported for PACSystems and VersaMax only.

Parameter Block Format	*Address*	*Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)*
1 = change or 0 = read	address	0 (read)
83h (ASCII 4 digit)	address+1	83h (ASCII 4 digit)

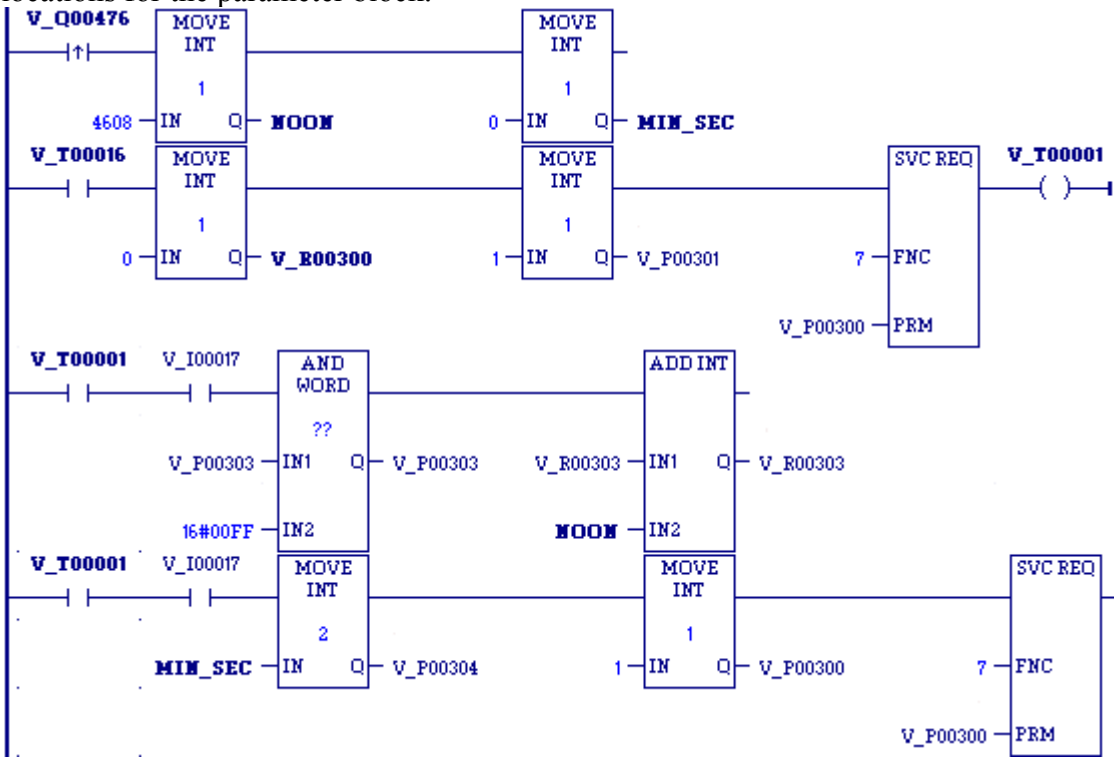
High Byte	*Low Byte*		*High Byte*	*Low Byte*
year (hundreds)	year (thousands)	address+2	30 (0)	32 (2)
year (ones)	year (tens)	address+3	35 (5)	30 (0)

month (tens)	(space)	address+4	31 (1)	20 (space)
(space)	month (ones)	address+5	20 (space)	30 (0)
day of month (ones)	day of month (tens)	address+6	35 (5)	30 (leading 0)
hours (tens)	(space)	address+7	32 (2)	20 (space)
: (colon)	hours (ones)	address+8	3A (:)	33 (3)
minutes (ones)	minutes (tens)	address+9	33 (3)	31 (1)
seconds (tens)	: (colon)	address+10	32 (2)	3A (:)
(space)	seconds (ones)	address+11	20 (space)	35 (5)
day of week (ones)	day of week (tens)	address+12	32 (2 = Mon.)	30 (leading 0)

Examples

Example 1: Series 90-70 CPUs

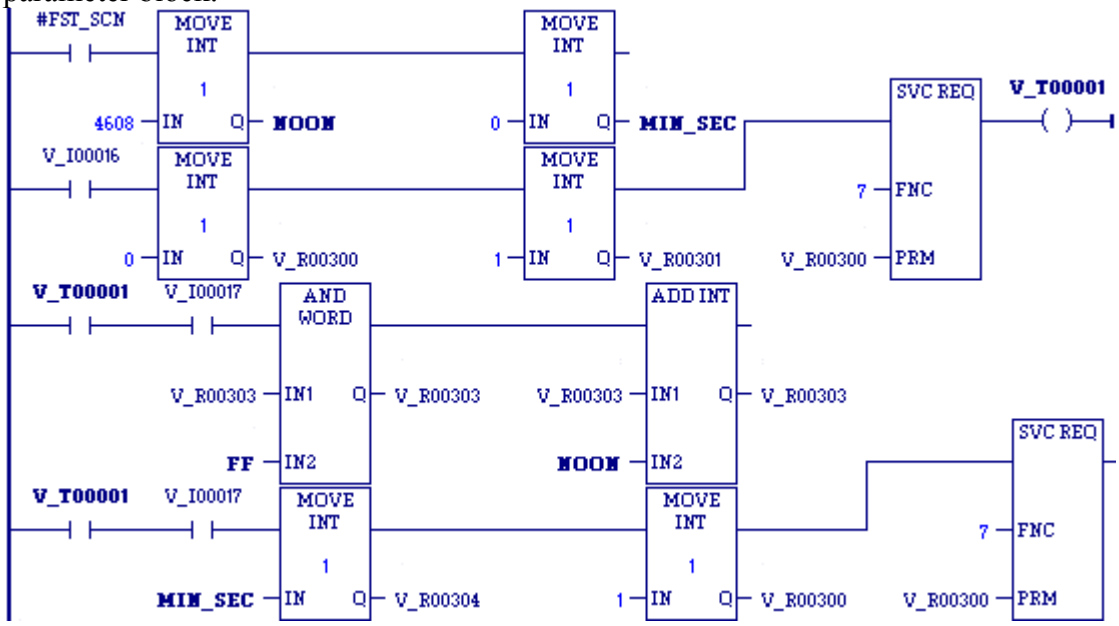
When output %Q00476 is on, a parameter block for the time-of-day clock is built to first request the current date and time, and then set the clock to 12 noon using the BCD format. The parameter block is located at global data location %P00300. Array NOON has been set up elsewhere in the logic to contain the values 12, 0, and 0. (Array NOON must also contain the data at %R0300.) The BCD format requires six contiguous memory locations for the parameter block.



Example 2: Other CPUs

When called for by previous logic, a parameter block for the time-of-day clock is built. It requests the current date and time, then sets the clock to 12 noon using BCD format. The

parameter block is located at global data location %R0300. Array NOON has been set up elsewhere in the logic to contain the values 12, 0, and 0. (Array NOON must also contain the data at %R0300.) BCD format requires six contiguous memory locations for the parameter block.



SVC_REQ 8: Reset Watchdog Timer

CPU Support

SVC_REQ 8 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Operation

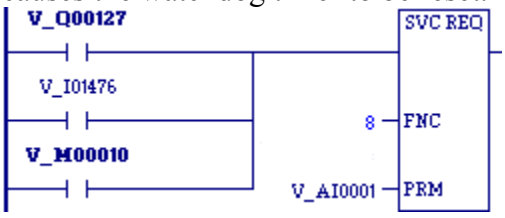
Use SVC_REQ 8 to reset the watchdog timer during the scan.
Ordinarily, when the watchdog timer expires, the CPU shuts down without warning. SVC_REQ 8 allows the timer to keep going during a time-consuming task (for example, while waiting for a response from a communications line).

Warning: Ensure that resetting the watchdog timer does not adversely affect the controlled process.

SVC_REQ 8 has no associated parameter block; however, you must still specify a dummy parameter, which SVC_REQ 8 will not use.

Example

Power flow through enabling output %Q0127 or input %I1476 or internal coil %M00010 causes the watchdog timer to be reset.



SVC_REQ 9: Read Sweep Time from Beginning of Sweep

CPU Support

SVC_REQ 9 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 version 2.02 or later CPUs.
- Series 90-30 version 8.0 or later CPUs.

Operation

Use SVC_REQ 9 to read the time in milliseconds since the start of the scan. The data is unsigned 16-bit integer.

Notes

- (PACSystems only.) For a higher resolution, in nanoseconds, use SVC_REQ 51.
- (Series 90-70 only.) SVC_REQ 9 has a different meaning with Microcycle mode. It reflects the time from the beginning of the sweep in which the logic was scheduled to begin execution.

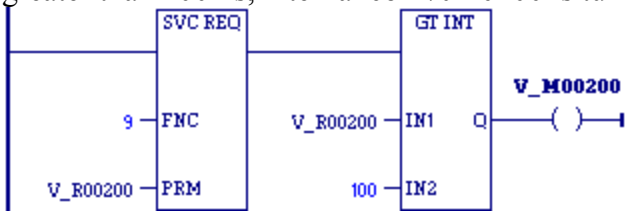
Output

The parameter block is an output parameter block only; it has a length of one word.

address	time since start of scan

Example

The elapsed time from the start of the scan is always read into location %R0200. If it is greater than 100ms, internal coil %M0200 is turned on.



SVC_REQ 10: Read Folder Name

CPU Support

SVC_REQ 10 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Operation

Use SVC_REQ 10 to read the name of the currently executing project.

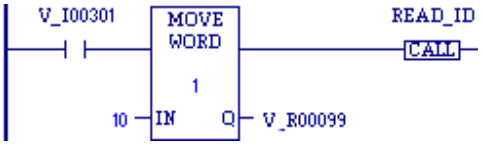
Output

The output parameter block has a length of four words. It returns eight ASCII characters: the logic name (from one to seven characters) followed by null characters (00h). The last character is always a null character. If the logic name has fewer than seven characters, null characters are appended to the end.

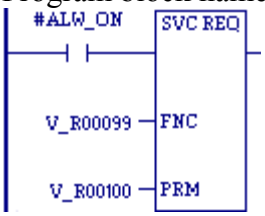
Address	*Low Byte*	*High Byte*
address	character 1	character 2
address+1	character 3	character 4
address+2	character 5	character 6
address+3	character 7	00

Example

When enabling input %I0301 goes ON, register location %R0099 is loaded with the value 10, which is the code for the Read Target Name. The program block named READ_ID is then called to retrieve the target name. The parameter block is located at address %R0100.



Program block named READ_ID:



SVC_REQ 11: Read Controller ID

CPU Support

SVC_REQ 11 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Operation

Use SVC_REQ 11 to read the name of the Controller executing the logic.

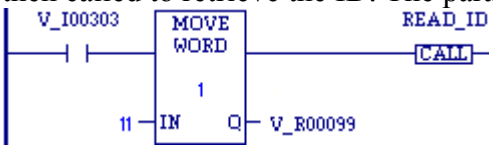
Output

The output parameter block has a length of four words. It returns eight ASCII characters: the Controller ID (from one to seven characters) followed by null characters (00h). The last character is always a null character. If the Controller ID has fewer than seven characters, null characters are appended to the end.

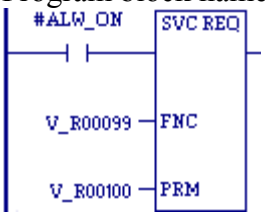
Address	*Low Byte*	*High Byte*
address	character 1	character 2
address+1	character 3	character 4
address+2	character 5	character 6
address+3	character 7	00

Example

When enabling input %I0303 is ON, register location %R0099 is loaded with the value 11, which is the code for the Read Controller ID. The program block named READ_ID is then called to retrieve the ID. The parameter block is located at address %R0100.



Program block named READ_ID:



SVC_REQ 12: Read Controller Run State

CPU Support

SVC_REQ 12 is supported for:

- PACSystems CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 Version 8.0 or later CPUs.

Operation

Use SVC_REQ 12 to read the current RUN state of the CPU.

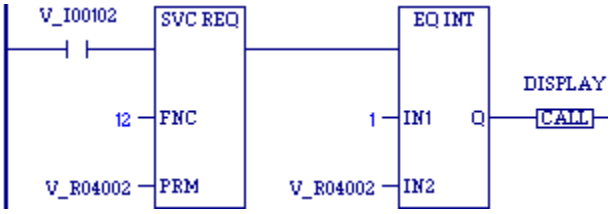
Output

The parameter block is an output parameter block only; it has a length of one word.

address	1 = run/disabled
	2 = run/enabled

Example

When contact V_I00102 is ON, the Controller run state is read into location %R4002. If the state is Run/Disabled, the CALL instruction calls the program block named DISPLAY.



SVC_REQ 13: Shut Down (Stop) CPU

CPU Support

SVC_REQ 13 is supported for:

- Series 90-30 CPUs
- VersaMax CPUs
- Series 90-70 firmware version 2.02 or later CPUs
- PACSystems CPUs

Operation

Use SVC_REQ 13 to stop the CPU at the start of scan n+1, where n ranges from 0 through 5, depending on the CPU type (see below). All outputs go to their designated default states at the start of scan n+1. An informational "Shut Down Controller" fault is placed in the Controller Fault table. The I/O scan continues as configured.

For Series 90-30 and VersaMax

n=1. The CPU executes one additional sweep after the sweep in which the SVC_REQ 13 is executed in order to ensure that the %S0002 LST_SCN contact operates correctly. The CPU stops at the start of the second scan that follows the scan in which the SVC_REQ 13 is executed.

For Series 90-70 firmware version 2.02 and later, and PACSystems firmware version prior to 2.00

n=0. The CPU stops at the start of the scan that follows the scan in which the SVC_REQ 13 is executed.

For PACSystems firmware version 2.00 and later

You specify the value of n. See below.

Input

For PACSystems firmware version 2.00 and later

The parameter block has a length of 1 WORD.

Address	0 = the PACSystems CPU transitions to Stop mode at the start of next scan. 1 through 5 = the PACSystems finishes executing this scan, then executes this number of scans, and then transitions to Stop mode at the start of the following scan. -1 = the PACSystems CPU finishes executing this scan, then executes the number of scans configured in the Number of Last Scans parameter in the CPU's Scan tab, and then transition to Stop mode at the start of the following scan.

The CPU may execute fewer scans than specified in this parameter, for example, if a fatal error or another SVC_REQ 13 with a smaller value is encountered.
To mimic a Series 90-70 firmware version 2.02 and later, do one of the following:

- On the CPU's Scan tab, set the Number of Last Scans parameter to 0 and then set the input of SVC_REQ to -1.

  - or -

- Set the input of SVC_REQ to 0.

To mimic a Series 90-30, do one of the following:

- On the CPU's Scan tab, set the Number of Last Scans parameter to 1 and then set the input of SVC_REQ to -1.

  - or -

- Set the input of SVC_REQ to 1.

For PACSystems firmware version prior to 2.00

The parameter block has a length of 1 and the value must be set to 0; otherwise, the PACSystems does not stop.

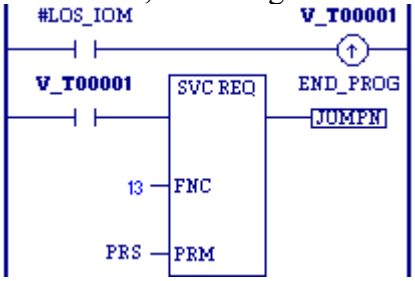
Address	0

For Series 90-70 firmware version 2.02 and later, Series 90-30, and VersaMax

SVC_REQ 13 has no parameter block; however, you must still specify a dummy parameter, which SVC_REQ 13 does not use.

Example

When a "Loss of I/O Module" fault occurs, the #LOS_IOM contact is set to ON and SVC_REQ 13 executes. The PRM parameter, PRS, is set to 0. If the Shut Down CPU instruction executes successfully, the JUMPN to the END_PROG LABELN at the end of the _MAIN block prevents the logic that follows the JUMPN from executing in the current sweep.
A Series 90-70 CPU firmware version 2.02 and later ignores the PRM parameter and transitions to Stop mode at the start of next scan.
A PACSystems CPU firmware version prior to 2.00 transitions to Stop mode at the start of next scan. Had the PRM parameter not been set to 0, the CPU would not stop.
A PACSystems CPU firmware version 2.00 and later looks up the PRM parameter, and because it is set to 0, it transitions to Stop mode at the start of next scan.
A Series 90-30 or VersaMax CPU ignores the PRM parameter and transitions to Stop mode only at the start of the second next sweep.
In all cases, all coils go to their default values.



The _MAIN block's last instruction is a LABELN:


END_PROG

SVC_REQ 14: Clear Controller or I/O Fault Table

CPU Support

SVC_REQ 14 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 CPUs.

Operation

Use SVC_REQ 14 to clear either the Controller Fault table or the I/O fault table. The SVC_REQ output is set ON unless some number other than 0 or 1 is entered as the requested operation.

The parameter block has a length of 1 word. It is an input parameter block only. There is no output parameter block.

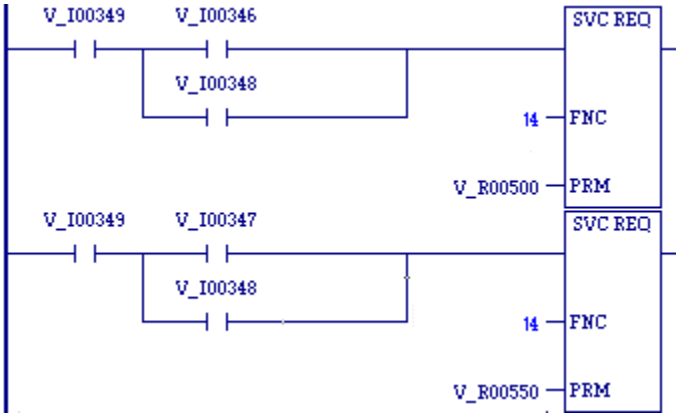
Address	0 = clear Controller Fault table
	1 = clear I/O fault table

Example

When inputs %I0346 and %I0349 are on, the Controller Fault table is cleared. When inputs %I0347 and %I0349 are on, the I/O fault table is cleared. When input %I0348 is on and input %I0349 is on, both are cleared.

The parameter block for the Controller Fault table is located at %R0500; for the I/O fault table the parameter block is located at %R0550.

Note: Both parameter blocks are set up elsewhere in the logic.



SVC_REQ 15: Read Last-Logged Fault Table Entry

CPU Support

SVC_REQ 15 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-70 Version 2.02 or later CPUs.
- Series 90-30 CPUs.

Operation

Use SVC_REQ 15 to read the last entry logged in either the Controller fault table or the I/O fault table. The SVC_REQ power flow output is set ON unless some invalid number is entered as the requested operation or the fault table is empty.
The parameter block has a length of 22 words. The first word is used only for input, and the other words only for output.
The input parameter block is as follows:

Address	*2-Digit Year Format*	*(VersaMax only) 4-Digit Year Format*
Address	0 = Read Controller Fault table	8 = read Controller Fault table
	1 = ▪ VersaMax and Series 90-30: Read I/O fault table ▪ PACSystems and Series 90-70: Read extended I/O fault table	9 = read I/O fault table
	(PACSystems and Series 90-70 only.) 80h = Read extended Controller Fault table	N/a
	(PACSystems and Series 90-70 only.) 81h = Read extended I/O fault table	N/a

Output

The format of the output parameter block depends on whether SVC_REQ 15 reads the Controller Fault table or the I/O fault table, or (PACSystems and Series 90-70 only) the extended Controller Fault table or the extended I/O fault table.
The first word of the parameter block is used for input. The following words are used for output.

Controller Fault table Output Format		*Address*	*I/O Fault Table Output Format*	
High Byte	*Low Byte*		*High Byte*	*Low Byte*

spare	long/short	address+1	memory type	long/short
spare	spare	address+2		offset
slot	rack	address+3	slot	rack
	task	address+4	block	bus
fault action	fault group	address+5		point
	error code	address+6	fault action	fault category
	fault specific data	address+7	fault type	fault category
		address+8 to address+18	fault specific data	fault description
minutes	seconds	address+19	minutes	seconds
day of month	hour	address+20	day of month	hour
year	month	**2-digit year format** address+21	year	month
spare	month	**4-digit year format (VersaMax only)** address+21	spare	month
year		address+22	year	

Long/Short Value

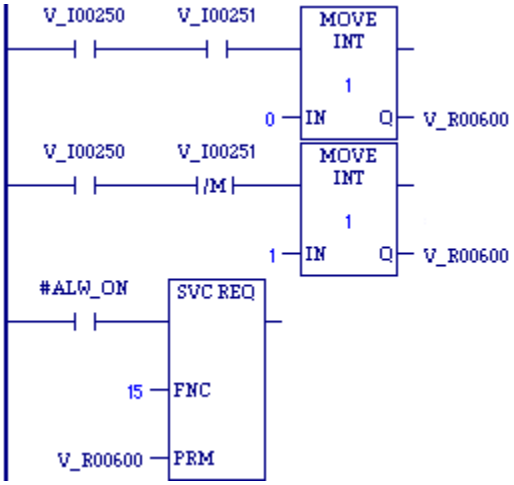
The first byte (low byte) of word *address +1* contains a number that indicates the length of the fault-specific data in the fault entry. These possible values are:

Controller Fault table	00 = 8 bytes (short)
Extended Controller Fault table	01 = 24 bytes (long)
I/O fault table	(VersaMax and Series 90-30 only.) 02 = 5 bytes (short)
Extended I/O fault table	03 = 21 bytes (long)

Note: (PACSystems and Series 90-70 only.) The short I/O fault table is not available. The value returned for the I/O fault table is always 03 = 21 bytes (long).

Example 1

When inputs %I0250 and %I0251 are both on, the first Move instruction places a zero (read Controller Fault table) into the parameter block for SVC_REQ 15. When input %I0250 is on and input %I0251 is off, the Move instruction instead places a one (read I/O fault table) in the SVC_REQ parameter block. The parameter block is located at location %R0600.



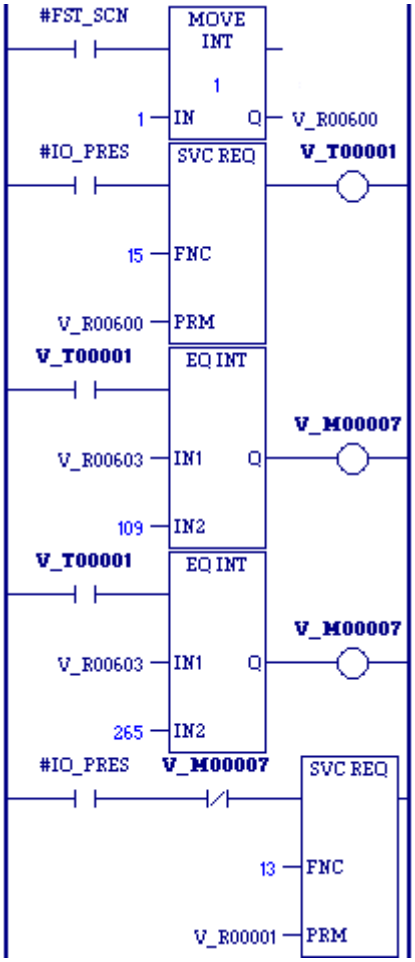
Example 2

The Controller is shut down when any fault occurs on an I/O module except when the fault occurs on modules in rack 0, slot 9 and in rack 1, slot 9. If faults occur on these two modules, the system remains running. The parameter for "table type" is set up on the first scan. The contact IO_PRES, when set, indicates that the I/O fault table contains an entry. The Controller CPU sets the normally open contact in the scan after the fault logic places a fault in the table. If faults are placed in the table in two consecutive scans, the normally open contact is set for two consecutive scans.

The example uses a parameter block located at %R0600. After the SVC_REQ instruction executes, the fourth, fifth, and sixth words of the parameter block contain the address of the I/O module that faulted:

	High Byte	*Low Byte*
%R0600		1
%R0601		long/short
%R0602	reference address	
%R0603	slot number	rack number
%R0604	block (bus address)	I/O bus no.
%R0605		point address
%R0606	fault data	

In the logic, the EQ_INT instructions compare the rack/slot address in the table to hexadecimal constants. The internal coil %M0007 is turned on when the rack/slot where the fault occurred meets the criteria specified above. If %M0007 is on, its normally closed contact is off, preventing the shutdown. Conversely, if %M0007 is off because the fault occurred on a different module, the normally closed contact is on and the shutdown occurs.


#FST_SCN
MOVE
INT
1
1
IN
Q
V_R00600
#IO_PRES
SVC REQ
V_T00001
15
FNC
V_R00600
PRM
V_T00001
EQ INT
V_M00007
V_R00603
IN1
Q
109
IN2
V_T00001
EQ INT
V_M00007
V_R00603
IN1
Q
265
IN2
#IO_PRES
V_M00007
SVC REQ
13
FNC
V_R00001
PRM

SVC_REQ 16: Read Elapsed Time Clock

CPU Support

SVC_REQ 16 is supported for:

- PACSystems CPUs
- VersaMax CPUs
- Series 90-70 Version 2.02 or later CPUs
- Series 90-30 CPUs

Operation

Use SVC_REQ 16 to read the system's elapsed time clock. The elapsed time clock measures the time in seconds since the CPU was powered on.
The parameter block has a length of three words used for output only.
The resolution of the Controller's elapsed time clock is 100 microseconds. The overall accuracy of the elapsed time clock is +/- 0.01%. The accuracy of an individual sample of the elapsed time clock is approximately 105 microseconds.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 16 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Tip: (PACSystems only.) For higher resolution, in nanoseconds, use SVC_REQ 50.

Output

address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	100 microsecond ticks

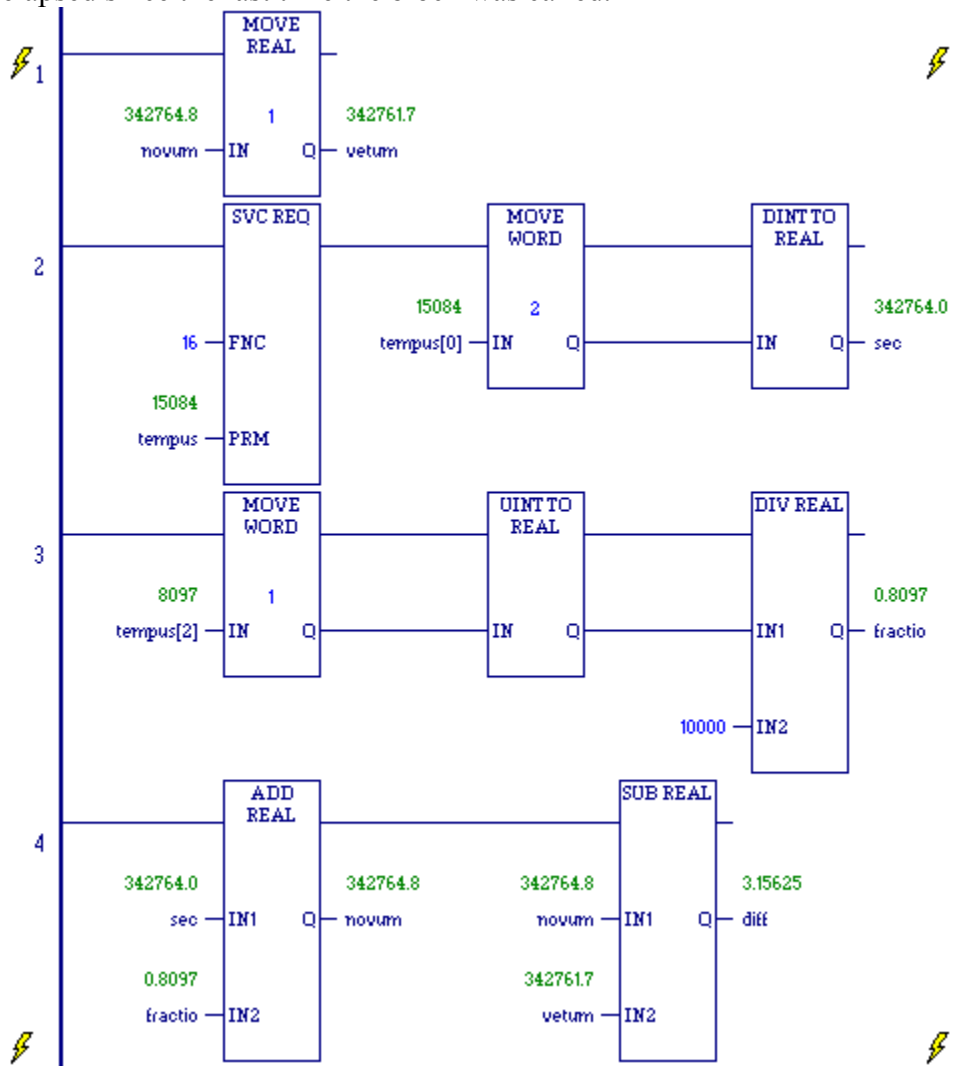
The first two words are the elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 16 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Example

The following logic is used in a block that is called once in a while. The screen shot was taken between calls to the block. The logic displayed calculates the number of seconds that have elapsed since the last time the block was called. It performs the final operation on rung 4 by subtracting the time obtained by SVC_REQ 16 the last time the block was called (vetum) from the time currently obtained by SVC_REQ 16 (novum) and storing the calculated value in the variable named diff.
On rung 2, SVC_REQ 16 returns three WORDs, stored in the 3-WORD array tempus. The first two WORDs (16-bit values) are moved to a DINT (a 32-bit value). This move amounts to a rough data type conversion that ignores the fact that the DINT type is

actually a signed value. Despite that, the subsequent calculations are correct until the time since power-on reaches approximately 50 years. The DINT is converted to REAL to yield the number of whole seconds elapsed since power-on, stored in variable sec. On rung 3, the third word returned by SVC_REQ 16, tempus[2], is converted to REAL. This is the number of 100 microsecond ticks. To obtain a fraction of a second, stored in the variable fractio, the value is divided by 10,000. On rung 4, sec and fractio are added to express the exact number of seconds elapsed since power-on, and this value is stored in the variable novum. On rung 1, the previous value of novum was saved as vetum, the exact number of seconds elapsed since power-on the last time the block was called. The last instruction on the fourth rung subtracts vetum from novum to yield the number of seconds that have elapsed since the last time the block was called.



Alternative

You could output the elapsed times as seconds and 100 microsecond ticks. See the example for the similar service request SVC_REQ 50, Read Elapsed Time Clock (Two DWORDs).

SVC_REQ 17: Mask/Unmask I/O Interrupt

CPU Support

SVC_REQ 17 is supported for:

- PACSystems CPUs.
- Series 90-70 CPUs, Version 2.02 or later.

Operation

Tip: When using I/O variables, you must use MASK_IO_INTR.

Use SVC_REQ 17 to mask or unmask an interrupt from an input board. When an interrupt is masked, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.

The parameter block is an input parameter block only; it has a length of three words.

address	0 = unmask input 1 = mask input
address+1	memory type
address+2	reference (offset)

"Memory type" is a decimal number that resides in the low byte of word *address + 1*. It corresponds to the memory type of the input:

70	%I memory in bit mode
10	%AI memory

Successful execution occurs unless:

- Some number other than 0 or 1 is entered as the requested operation.
- The memory type of the input to be masked or unmasked is not %I or %AI memory.
- The I/O board is not a supported input module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Masking / Unmasking Module Interrupts

(PACSystems only.) During module configuration, interrupts from a module can be enabled or disabled. If a module's interrupt is disabled, it cannot be used to trigger logic execution and it cannot be unmasked. However, if an interrupt is enabled in the configuration, it can be dynamically masked or unmasked by the logic during system operation.

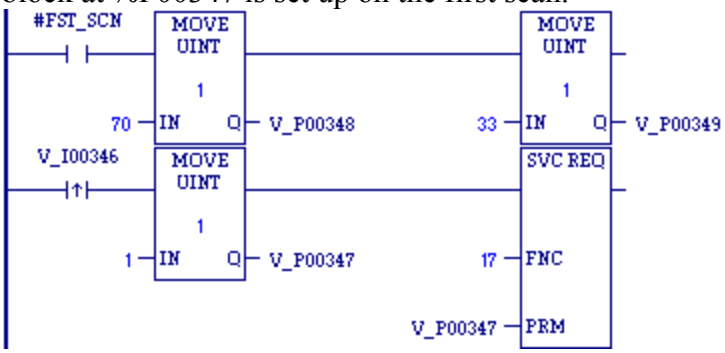
For the logic to mask and unmask interrupts that are enabled, use SVC_REQ 17. To mask or unmask an interrupt from an open VME module, the application logic should pass VME_INT_ID (17 decimal, 11H) as the memory type and the VME interrupt id as the offset to SVC_REQ 17.

When the interrupt is not masked, the CPU processes the interrupt and schedules the associated logic for execution. When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated logic for execution.
When the CPU transitions from Stop to Run, the interrupt is unmasked.
For additional information on configuring and using VME module interrupts in a PACSystems RX7i control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules* (GFK-2235).

Examples

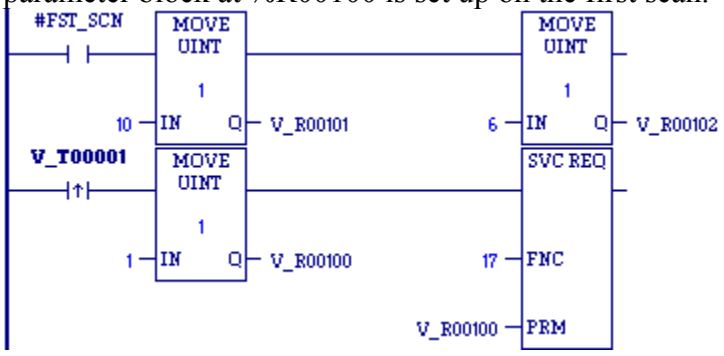
Example 1

When %I00346 transitions on, interrupts from input %I00033 are masked. The parameter block at %P00347 is set up on the first scan.



Example 2

When %T00001 transitions on, alarm interrupts from input %AI0006 are masked. The parameter block at %R00100 is set up on the first scan.



SVC_REQ 18: Read I/O Forced Status

CPU Support

SVC_REQ 18 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Series 90-30 CPUs 331 and later.
- Series 90-70 Version 2.02 or later CPUs.

Operation

Use SVC_REQ 18 to read the current status of forced values in the CPU's %I and %Q memory areas.

Note: SVC_REQ 18 does not detect forced values in %G or %M memory types. Use %S0011 (#OVR_PRE) to detect forced values in %I, %Q, %G, and/or %M memory types.

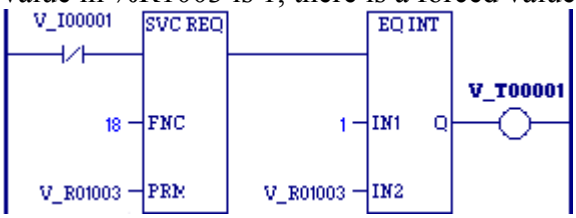
The parameter block has a length of one word used for output only.

Output

address	0 = No forced values are set
	1 = Forced values are set

Example

SVC_REQ reads the status of I/O forced values into location %R1003. If the returned value in %R1003 is 1, there is a forced value, and EQ INT turns the %T0001 coil ON.



SVC_REQ 19: Set Run Enable/Disable

CPU Support

SVC_REQ 19 is supported for:

- PACSystems CPUs.
- Series 90-70 Version 2.02 or later CPUs.

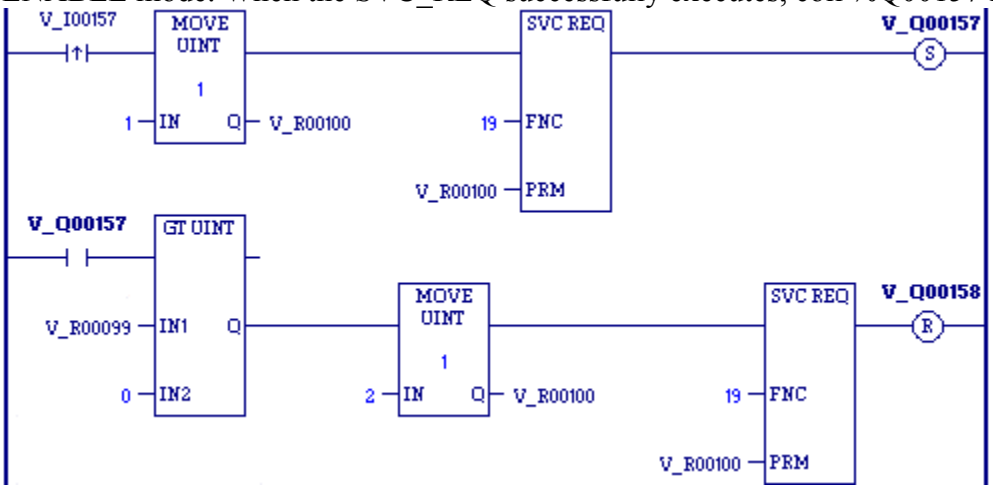
Operation

Use SVC_REQ 19 to enable LD logic to control the RUN mode of the CPU.
The parameter passed indicates which instruction to perform. The OK output is set to ON if the instruction executes successfully. It is set to OFF if the requested operation is not SET RUN DISABLE mode (1) or SET RUN ENABLE mode (2).
The parameter block is an input parameter block only with this format:

address	1 = SET RUN DISABLE mode 2 = SET RUN ENABLE mode

Example

When input %I00157 transitions to on, the RUN DISABLE mode is set. When the SVC_REQ instruction successfully executes, coil %Q00157 is turned on. When %Q00157 is on and register %R00099 is greater than zero, the mode is changed to RUN ENABLE mode. When the SVC_REQ successfully executes, coil %Q00157 is turned off.



SVC_REQ 20: Read Fault Tables

CPU Support

SVC_REQ 20 is supported for:

- PACSystems
- Series 90-70 firmware version 2.02 or later

Operation

Use SVC_REQ 20 to retrieve the entire Controller or I/O fault table and return it to designated registers, from where they are accessible to LD logic.

The first input parameter designates which table is to be read. The second input parameter (always zero for the standard Read Fault Tables) is used by the extended format to read a designated fault entry or to read a range of fault entries. The fault table data is placed in the parameter block following the input parameters.

The OK output is turned on if the instruction executes successfully. It is off if the requested operation is not Read Controller Fault table (00h), Read I/O Fault Table (01h), Read Extended Controller Fault table (80h), or Read Extended I/O Fault Table (81h), or if there is not enough of the specified memory reference to hold the fault data. If the specified fault table is empty, the instruction sets the OK output on, but returns only the fault table header information.

The parameter block is an input and output parameter block. The parameter block comes in two formats:

- Non-Extended: Read Controller Fault Table (00h), Read I/O Fault Table (01h)
- Extended: Read Extended Controller Fault Table (80h), Read Extended I/O Fault Table (81h)

Non-Extended Formats

For non-extended formats (Read Controller Fault table (00h), Read I/O Fault Table (01h)), SVC_REQ 20 requires 693 registers available.

Format of the input parameter block

Address	Value
address	00h = Read Controller Fault table 01h = Read I/O fault table
address+1	Always zero (0)

Format of the output parameter block

Address	High Byte	Low Byte
address	Unused	0 = Controller Fault table 1 = I/O fault table
address+1	Unused	Always zero (0)

address+2 through address+14	Unused	Unused
address+15 through address+17	Time since last clear, in BCD format:	
address+15	Minutes	Seconds
address+16	Day of Month	Hour
address+17	Year	Month
address+18	Number of faults since last clear	
address+19	Number of faults in queue	
address+20	Number of faults read	
address+21	Start of fault data	

For the non-extended formats, the returned data for each fault consists of 21 words (42 bytes). This request returns 16 Controller Fault table entries or 32 I/O fault table entries, or the actual number of faults if it is fewer. If the fault table read is empty, no data is returned.
The following table shows the return format of both a Controller Fault table entry and an I/O fault table entry.

Format of Returned Data for Fault Table Entries

Address	**Controller Fault table**		**I/O Fault Table**	
	High Byte	**Low Byte**	**High Byte**	**Low Byte**
address+21	Unused	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 00: 8 bytes ▪ 01: 24 bytes	Memory type	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 02: 5 bytes ▪ 03: 21 bytes
address+22	Unused	Spare	Offset	
address+23	Slot	Rack	Slot	Rack
address+24	Task		Bus address	I/O Bus Number (block)
address+25	Fault action	Fault group	Point	
address+26	Error code		Fault action	Fault group
address+27	Fault extra data		Fault type	Fault category
address+28			Fault extra data	Fault description

address+29 through address+38			Fault extra data	
address+39 through address+41	Time stamp in BCD format:		Time stamp in BCD format:	
address+39	Minutes	Seconds	Minutes	Seconds
address+40	Day of month	Hour	Day of month	Hour
address+41	Year	Month	Year	Month
address+42	Start of next fault output parameter block			

Extended Formats

Each extended format request can read a maximum of 64 faults, or the size of the fault table if it contains less than 64 faults.
For extended formats (Read Extended Controller Fault table (80h), or Read Extended I/O Fault Table (81h)), the Controller calculates the number of entries being read. You must ensure that enough registers are available to receive the amount of fault entries requested. If the amount of data requested exceeds the registers available, the CPU returns a fault indicating that reference memory is out of range.
The total size of the fault table for the extended fault format is
Header Size + ((# fault entries) * (size of fault entry))

Format of the input parameter block

Address	**Value**
address	80h = Read extended Controller Fault table 81h = Read extended I/O fault table
address+1	Starting index of faults to be read
address+2	Number of faults to be read

Format of the output parameter block

Address	**High Byte**	**Low Byte**
address	Unused	80h = Extended Controller Fault table 81h = Extended I/O fault table
address+1	Starting index of faults to be read	
address+2	Number of faults to be read	
address+3 through address+14	Unused	
address+15 through address+17	Time since last clear, in BCD format	

address+15	Minutes	Seconds
address+16	Day of month	Hour
address+17	Year	Month
address+18	Number of faults since last clear	
address+19	Number of faults in queue	
address+20	Number of faults read	
address+21 through address+36	Unused	
address+37	Start of fault data	

For Read Extended Controller Fault table (80h) and Read Extended I/O Fault Table (81h), each extended fault table entry is 23 words long (46 bytes).

Format of Returned Data for Fault Table Entries

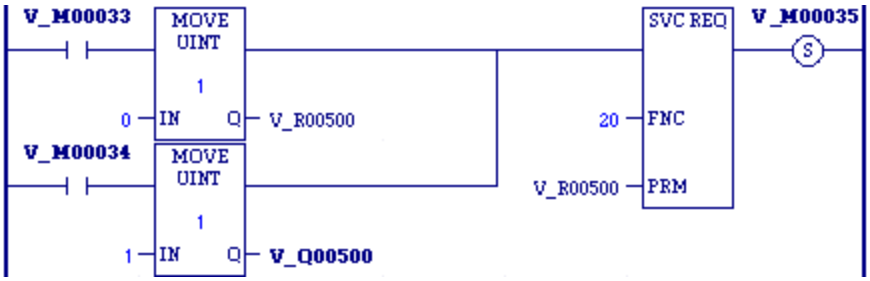
Address	**Controller Fault table**		**I/O Fault Table**	
	High Byte	**Low Byte**	**High Byte**	**Low Byte**
address+37	Unused	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 00: 8 bytes ▪ 01: 24 bytes	Reference address memory type	Long/Short: number of bytes of fault extra data in the fault entry: ▪ 02: 5 bytes ▪ 03: 21 bytes
address+38	Unused		Reference address offset	
address+39	Slot	Rack	Slot	Rack
address+40	Task		Bus address	I/O Bus Number (block)
address+41	Fault action	Fault group	Point	
address+42	Error code		Fault action	Fault group
address+43	Fault extra data		Fault type	Fault category
address+44			Fault extra data	Fault description
address+45 through address+54			Fault extra data	
address+55 through address+58	Time stamp in BCD format:		Time stamp in BCD format:	
address+55	Minutes	Seconds	Minutes	Seconds

address+56	Day of month	Hour	Day of month	Hour
address+57	Year	Month	Year	Month
address+58	Milliseconds		Milliseconds	
address+59	Unused		Unused	
address+60	Start of next fault output parameter block			

Examples

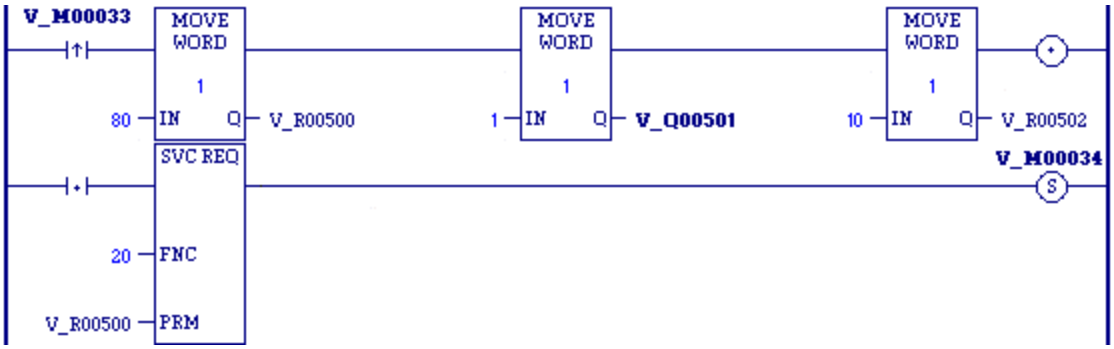
Example 1: Non-Extended Format

When the enabling input %M00033 is On, the Controller Fault table is read. When the enabling input %M00034 is On, the I/O fault table is read. The parameter block begins at the reference address %R00500. When the SVC_REQ instruction successfully executes, coil %M00035 is set to On.



Example 2: Extended Format

When the input %M00033 is On, the Extended Controller Fault table is read. The parameter block begins at the reference address %R00500. %R00500 contains the fault table type (Controller Extended); %R00501 contains the starting fault to read, and %R005002 contains the number of faults to read starting with the fault number in %R00501. When the SVC_REQ instruction successfully executes, coil %M00034 is set to On.



SVC_REQ 21: User-Defined Fault Logging

CPU Support

SVC_REQ 21 is supported for:

- PACSystems CPUs.
- Series 90-70 Version 2.02 or later CPUs.

Operation

Use SVC_REQ 21 to define a fault that can be displayed in the Controller fault table. The fault contains binary information or an ASCII message. The user-defined fault codes start at 0 hex.

The error code information for the fault must be within the range 0 to 2047 for an "Application Msg:" to be displayed. If the error code is in the range 81 to 112 decimal, the CPU sets a fault bit of the same number in %SA system memory. This allows up to 32 bits to be individually set.

Error Code	*Status Bit*
Errors 0–80	No bit set
Errors 81–112	Sets %SA
Errors 113–2047	No bit set
Errors 2048–32,767	Reserved

When EN is active, the fault data array referenced by IN is logged as a fault to the Controller Fault table. If EN is not enabled, the ok bit is cleared. If the error code is out of range, the ok bit is cleared and the fault will not be logged as requested.

The parameter block is an input parameter block only with this format:

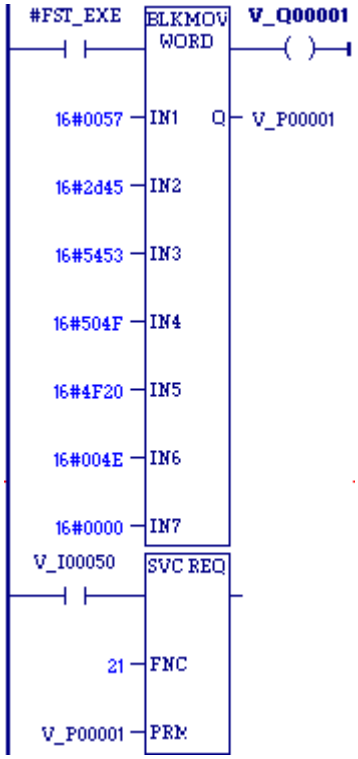
Parameter	*MSB*	*LSB*
address	Error code	
address+1	Text2	Text1
address+2	Text4	Text3
address+3	Text6	Text5
address+4	Text8	Text7
address+5	Text10	Text9
address+6	Text12	Text11
address+7	Text14	Text13
address+8	Text16	Text15
address+9	Text18	Text17
address+10	Text20	Text19

address+11	Text22	Text21
address+12	Text24	Text23

In the input parameter data, you can select an error code in the range 0 to 2047 and text information that will be placed in the fault extra data portion of a long Controller fault. The Controller fault address, fault group, and fault action are filled in by SVC_REQ. The fault text bytes 1 through 24 can be used to pass binary or ASCII data with the fault. If the first byte of the fault text data is non-zero, the data will be an ASCII message string. This message will then be displayed in the fault description area of the fault table. If the message is less than 24 characters, the ASCII string must be NULL byte-terminated. The programmer displays "Application Msg:" and the ASCII data is displayed as a message immediately following "Application Msg:". If the error code is between 1 and 2047, the error code number is displayed immediately after "Msg" in the "Application Msg:" string. If the error code is greater than 2047, it will be converted to error code 0.
If the first byte of text is zero, then only "Application Msg:" is displayed in the fault description. The next 1-23 bytes will be considered binary data for user data logging. This data can be displayed.
For more information, refer to chapter 5, "PLC Control and Status," in the *Programming Software User's Manual*, GFK-0263.

Example

The value passed to IN1 is the fault error code. The value passed in, 16x0057, represents an error code of 87 decimal and will appear as part of the fault message. The values of the next inputs give the ASCII codes for the text of the error message. For IN2, the input is 2D45. The low byte, 45, decodes to the letter **E** and the high byte, 2D, decodes to -. Continuing in this manner, the string continues with **S T O P O** and **N**. The final character, **00**, is the null character that terminates the string. In summary, the decoding yields the string message **E_STOP ON**.


#FST_EXE
BLKMOV
WORD
V_Q00001
16#0057
IN1
Q
V_P00001
16#2d45
IN2
16#5453
IN3
16#504F
IN4
16#4F20
IN5
16#004E
IN6
16#0000
IN7
V_I00050
SVC REQ
21
FNC
V_P00001
PRM

SVC_REQ 22: Mask/Unmask Timed Interrupts

CPU Support

SVC_REQ 22 is supported for

- PACSystems CPUs.
- Series 90-70 Version 4.02 or later CPUs.

Operation

Use SVC_REQ 22 to mask or unmask timed interrupts and to read the current mask. When the interrupts are masked, the CPU does not execute any timed interrupt block timed logic that is associated with a timed interrupt. Timed interrupts are masked/unmasked as a group. They cannot be individually masked or unmasked. Successful execution occurs unless some number other than 0 or 1 is entered as the requested operation or mask value.

The parameter block is an input and output parameter block.

To determine the current mask, use this format:

address	0 = Read interrupt mask

The Controller returns this format:

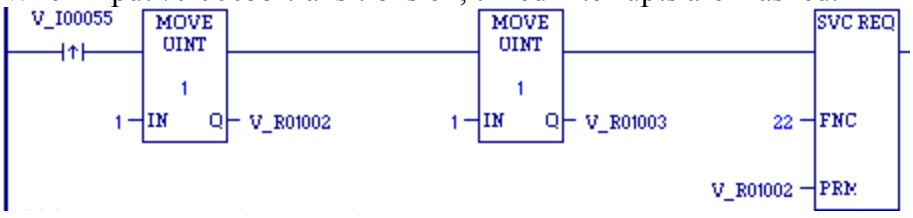
address	0 = Read interrupt mask
address+1	0 = Timed interrupts are unmasked 1 = Timed interrupts are masked

To change the current mask, use this format:

address	1 = Mask/unmask interrupts
address+1	0 = Unmask timed interrupts 1 = Mask timed interrupts

Example

When input %I00055 transitions on, timed interrupts are masked.



SVC_REQ 23: Read Master Checksum

CPU Support

SVC_REQ 23 is supported for:

- PACSystems CPUs.
- VersaMax CPUs.
- Version 4.40 or later of Series 90-30 CPUs
- Version 4.12 or later of Series 90-70 CPUs.

Note: The operation and values returned are different for the Series 90-70 CPUs.

Series 90-30 and VersaMax

Operation

Use SVC_REQ 23 to read the master checksums of the application logic and the configuration. When SVC_REQ 23 is enabled, the output is always ON.
The parameter block has a length of twelve words used for output only.

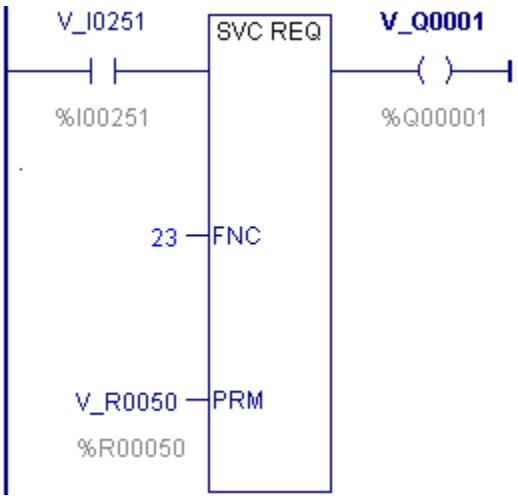
Output

When a Run Mode Store is active, the logic checksums may not be valid until the store (download) is complete. To know whether the logic and configuration checksums are valid, look up the two flags at the beginning of the output parameter block.

Address	*Description*
Address	Master Program Checksum Valid (0 = not valid, 1 = valid)
Address + 1	Master Configuration Checksum Valid (0 = not valid, 1 = valid)
Address + 2	Number of Program Blocks (including _MAIN)
Address + 3	Size of User Program in Bytes (DWORD data type)
Address + 5	Program Additive Checksum
Address + 6	Program CRC Checksum (DWORD data type)
Address + 8	Size of Configuration Data in Bytes
Address + 9	Configuration Additive Checksum
Address + 10	Configuration CRC Checksum (DWORD data type)

Example

When input %I0251 is ON, the master checksum information is placed into the parameter block at %R0050 and the output coil (%Q0001) is turned on.



PACSystems and Series 90-70

Operation

Use SVC_REQ 23 to read master checksums for the set of user program(s) and the configuration, and to read the checksum for the block from which the service request is made.

There is no input parameter block for this service request. The output parameter block requires 15 WORDs of memory.

Output

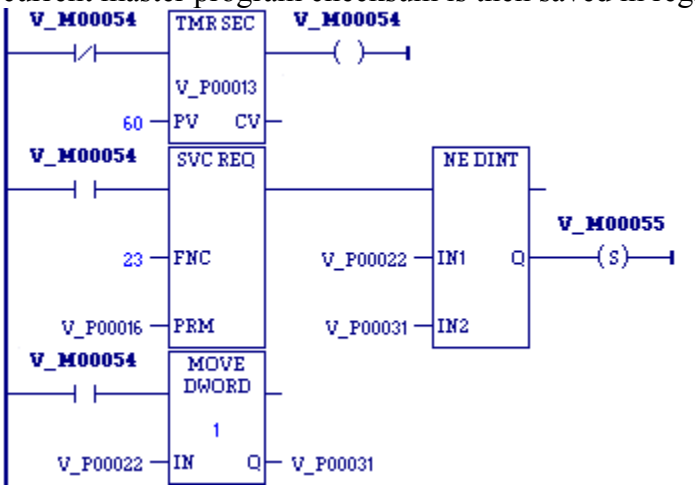
When a Run Mode Store is active, the program checksums may not be valid until the store (download) is complete. To determine when checksums are valid, three flags (one each for Program Block Checksum, Master Program Checksum, and Master Configuration Checksum) are provided at the beginning of the output parameter block.

Address	*Description*
Address	Program Checksum Valid (0 = not valid, 1 = valid)
Address + 1	Master Program Checksum Valid (0 = not valid, 1 = valid)
Address + 2	Master Configuration Checksum Valid (0 = not valid, 1 = valid)
Address + 3	Number of LD/SFC Blocks (including _MAIN)
Address + 4	Size of User Program in Bytes (DWORD data type)
Address + 6	Program Set Additive Checksum
Address + 7	Program CRC Checksum (DWORD data type)
Address + 9	Size of Configuration Data in Bytes
Address + 10	Configuration Additive Checksum
Address + 11	Configuration CRC Checksum (DWORD data type)
Address + 13	high byte: always zero low byte: Currently Executing Block's Additive Checksum

Address + 14	Currently Executing Block's CRC Checksum

Example

When the timer using registers %P00013 through %P00015 expires, the checksum read is performed. The checksum data returns in registers %P00016 through %P00030. The master program checksum in registers %P00022 and %P00023 (the program checksum is a DWORD data type and occupies two adjacent registers) is compared with the last saved master program checksum. If these are different, coil %M00055 is latched on. The current master program checksum is then saved in registers %P00031 and %P00032.



SVC_REQ 24: Reset Smart Module

CPU Support

SVC_REQ 24 is supported for:

- PACSystems CPUs
- Series 90-30 CPUs

Note: PACSystems RX3i redundancy CPUs support SVC_REQ 24 to reset an IC695RMX128 module only if the Redundancy Link parameter on the Settings tab of the RMX128 is set to Disabled.

Operation

Use SVC_REQ 24 to reset a daughterboard or smart module. The SVC_REQ output is set to ON unless one of the following conditions exists:

- An invalid number for rack and/or slot is entered.
- There is no module at the specified location.
- The module at the specified location does not support a runtime reset.
- The CPU was unable to reset the module at the specified location.

For this instruction, the parameter block has a length of 1 WORD. It is an input parameter block only.

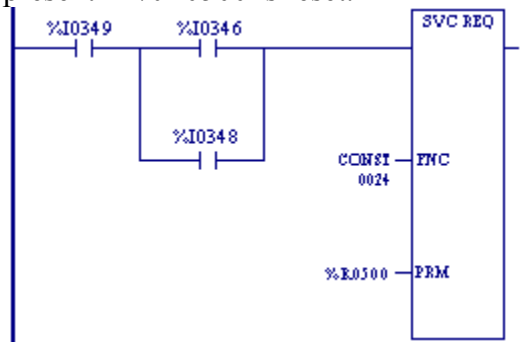
address	Module Slot (low byte)
	Module Rack (high byte)

Notes

- Rack 0, Slot 1 indicates that a reset is to be sent to the daughterboard.
- It is important to invoke SVC_REQ 24 for a given module for only one sweep at a time. Each time SVC_REQ 24 executes, the target module is reset again, regardless of whether it has finished starting up from a previous reset.

Example

In the following example, the parameter block (set up elsewhere in the logic) containing the module's rack and slot for this service request is located at %R0500.
When input %I0346 is on and input %I0349 is on, the module indicated by the Rack/Slot present in %R0500 is reset.



SVC_REQ 25: Disable/Enable EXE Block and Standalone C Program Checksums

CPU Support

SVC_REQ 25 is supported for:

- PACSystems CPUs.
- Series 90-70 Version 4.02 or later CPUs.

Operation

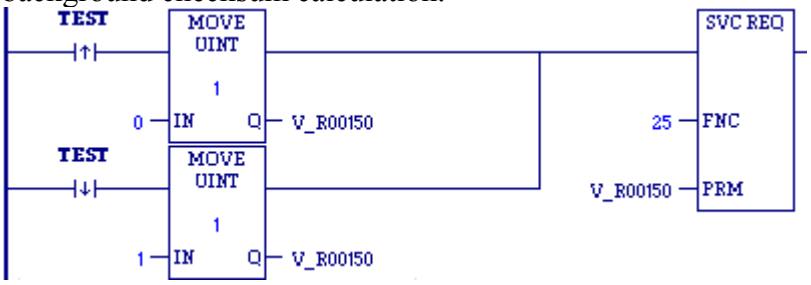
Use SVC_REQ 25 to enable or disable the inclusion of C blocks and C programs in the background checksum calculation. The default is to include the checksums.
This service request uses only an input parameter block.

address	0 = Disable C applications inclusion in checksum calculation
	1 = Enable C application inclusion in checksum calculation

The parameter block is unchanged after execution of the service request.

Example

When the coil TEST transitions from OFF to ON, SVC_REQ 25 executes to disable the inclusion of EXE blocks in the background checksum calculation. When coil TEST transitions from ON to OFF, the SVC_REQ executes to again include EXE blocks in the background checksum calculation.



SVC_REQ 26: Role Switch (PACSystems and Series 90-70)

CPU Support

SVC_REQ 26 is intended for use with:

- Enhanced Hot Standby CPU Redundancy, which is available only on PACSystems RX7i CRE models, PACSystems RX3i CRU models, and Series 90-70 CPU models IC697CGR772 and IC697CGR935.
- Hot Standby CPU Redundancy, which is available only on a Series 90 IC697CPU780, beginning with version 4.56.

Note: On Series 90-30 CPUs and VersaMax CPUs, SVC_REQ 26 requests an entirely different service.

Operation

Use SVC_REQ 26 to cause the CPUs to switch roles on the next scan (active to backup and backup to active) if the CPUs are synchronized and the timing requirements of the role switch request are met. A manual role switch cannot occur within 10 seconds of a previous manual role switch. Role switches due to failures or resynchronization are always supported (the 10-second limitation does not apply).

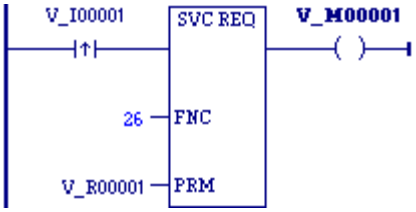
Note: Power flow from SVC_REQ 26 indicates that a role switch will be attempted on the next scan. It does not indicate that a role switch has occurred or that a role switch will occur on the next scan.

SVC_REQ 26 has no associated parameter block; however, the programming software requires that an entry be made for the PRM input parameter. Enter any appropriate reference here; it will not be used.

For more information about Hot Standby CPU Redundancy, refer to the *Series 90-70 Hot Standby CPU Redundancy User's Guide*, GFK-0827. For more information about Enhanced Hot Standby CPU Redundancy, refer to the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide*, GFK-1527.

Example

A switch on a control console is wired to %I00001, the input to the SVC_REQ 26 instruction. When closed, the switch will activate the SVC_REQ 26, causing a role switch between CPUs.



The 10-second limitation allows this SVC_REQ to be in both CPUs so that only a single switch occurs if the input is seen by both CPUs.

SVC_REQ 26/30: Interrogate I/O

CPU Support

SVC_REQ 26 and SVC_REQ 30 are supported for:

- Version 4.40 and later of Series 90-30 CPUs.
- All VersaMax CPUs.

SVC_REQ 26 and 30 are identical. You can use either number to accomplish the same purpose.

They are *not* available on Micro Controllers.

On Series 90-70 CPUs, SVC_REQ 26 requests an entirely different service and SVC_REQ 30 does not exist.

Operation

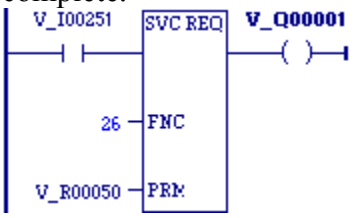
Use SVC_REQ 26 or 30 to check whether the installed modules match the software configuration. If not, SVC_REQ 26 or 30 place appropriate addition, loss, and mismatch faults in the Controller and/or I/O fault tables.

The more configuration faults there are, the longer it takes these SVC_REQs to execute.

SVC_REQ 26 and 30 have no parameter block. They always output power flow.

Example

When input %I0251 is ON, SVC_REQ 26 checks the installed modules and compares them to the software configuration. Output %Q0001 is turned on after SVC_REQ 26 is complete.



SVC_REQ 27: Write to Reverse Transfer Area

CPU Support

SVC_REQ 27 is intended for use with:

- Enhanced CPU Redundancy, supported on PACSystems CRE and CRU models and on Series 90-70 IC697CGR772 and IC697CGR935.
- CPU Redundancy, which is available only on a Series 90-70 IC697CPU780, beginning with version 4.56.

For all information about using this service request, refer to "Programming a Data Transfer from Backup Unit to Active Unit" in the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide* (GFK-1527).

SVC_REQ 28: Read from Reverse Transfer Area

CPU Support

SVC_REQ 28 is intended for use with:

- Enhanced CPU Redundancy, supported on PACSystems CRE and CRU models and on Series 90-70 IC697CGR772 and IC697CGR935.
- CPU Redundancy, which is available only on a Series 90-70 IC697CPU780, beginning with version 4.56.

For all information about using this service request, refer to "Programming a Data Transfer from Backup Unit to Active Unit" in the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide* (GFK-1527).

SVC_REQ 29: Read Elapsed Power Down Time

CPU Support

SVC_REQ 29 is supported for:

- PACSystems CPUs firmware version 2.00 and later.
- Series 90-30 CPU331 and later CPUs.
- VersaMax CPUs.

Operation

Use SVC_REQ 29 to read the amount of time elapsed between the last power-down and the most recent powerup.
SVC_REQ 29 always sets the power flow output to ON.
The parameter block has a length of three words used for output only.

Output

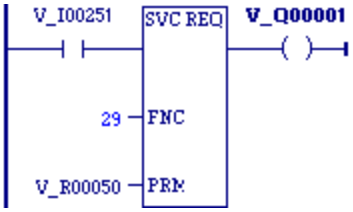
Address	*Description*
Address	Power-Down Elapsed Seconds (low order)
Address+1	Power-Down Elapsed Seconds (high order)
Address+2	Zero

The first two words are the power-down elapsed time in seconds. Whenever the Controller cannot properly calculate the power down elapsed time, the time will be set to 0. This happens if the watchdog timer times out before power-down.

Note: A Series 90-30 CPU 331 cannot properly calculate the power-down elapsed time when it is powered up with CLR M/T pressed on the HHP.

Example

When input %I0251 is ON, the Elapsed Power-Down Time is placed into the parameter block that starts at %R0050. The output coil (%Q0001) is turned on.



SVC_REQ 26/30: Interrogate I/O

CPU Support

SVC_REQ 26 and SVC_REQ 30 are supported for:

- Version 4.40 and later of Series 90-30 CPUs.
- All VersaMax CPUs.

SVC_REQ 26 and 30 are identical. You can use either number to accomplish the same purpose.

They are *not* available on Micro Controllers.

On Series 90-70 CPUs, SVC_REQ 26 requests an entirely different service and SVC_REQ 30 does not exist.

Operation

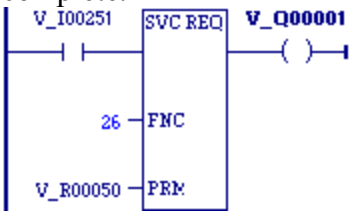
Use SVC_REQ 26 or 30 to check whether the installed modules match the software configuration. If not, SVC_REQ 26 or 30 place appropriate addition, loss, and mismatch faults in the Controller and/or I/O fault tables.

The more configuration faults there are, the longer it takes these SVC_REQs to execute.

SVC_REQ 26 and 30 have no parameter block. They always output power flow.

Example

When input %I0251 is ON, SVC_REQ 26 checks the installed modules and compares them to the software configuration. Output %Q0001 is turned on after SVC_REQ 26 is complete.



SVC_REQ 32: Suspend/Resume I/O Interrupt

CPU Support

SVC_REQ 32 is supported for:

- PACSystems CPUs.
- The following Series 90-70 CPUs: Version 5.03 or later of CPU731, CPU732, CPU771 and CPU772; and Version 5.50 or later of CPU781 and later CPUs.

Operation

Tip: When using I/O variables, you must use SUSP_IO_INTR.

Use SVC_REQ 32 to suspend a set of I/O interrupts and cause occurrences of these interrupts to be queued until these interrupts are resumed. The set of I/O interrupts are those that can be generated from the Series 90-70 High Speed Counter. The number of I/O interrupts that can be queued depends on the I/O module's capabilities. The CPU informs the I/O module that its interrupts are to be suspended or resumed. The I/O module's default is resumed. The Suspend applies to all I/O interrupts associated with the I/O module. Interrupts are suspended and resumed within a single scan.
SVC_REQ 32 uses only an input parameter block. Its length is three WORDs.

Address	0 = resume interrupt 1 = suspend interrupt
Address + 1	Memory type. Must be set to 70 (%I memory).
Address + 2	Reference address

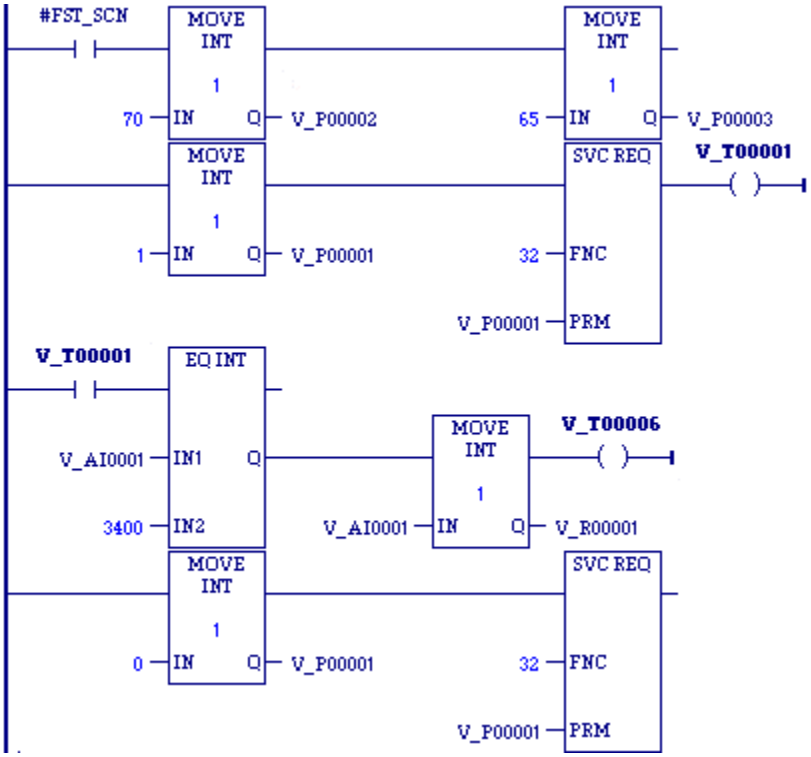
Successful execution occurs unless:

- Some number other than 0 or 1 is passed in as the first parameter.
- The memory type parameter is not 70 (%I memory).
- The I/O module associated with the specified address is not an appropriate module for this operation. (On a Series 90-70, the module must be a Series 90-70 High Speed Counter.)
- For a Series 90-70 High Speed Counter, the reference address specified is not the first %I reference set for the High Speed Counter.
- Communication between the CPU and this I/O module has failed. (The board is not present, or it has experienced a fatal fault.)

Example

Interrupts from the high speed counter module whose starting point reference address is %I00065 will be suspended while the CPU solves the logic of the second rung. Without the Suspend, an interrupt from the HSC could occur during execution of the third rung and %T00006 could be set while %R000001 has a value other than 3,400. (%AI00001 is the first non-discrete input reference for the High Speed Counter.)

Note: I/O interrupts, unless suspended or masked, can interrupt the execution of an instruction. The most often used application of this Service Request is to prevent the effects of the interrupts for diagnostic or other purposes.



SVC_REQ 36: Read from/Write to Bulk Memory Area

CPU Support

SVC_REQ 36 is supported for Series 90-70 CPU models that have a Bulk Memory Area (BMA): IC697CPX772, IC697CPX782, IC697CPX928 and IC697CPX935 (version 7.80 and later).

Notes

- On PACSystems CPEs, the Bulk Memory Area is %W memory. To access %W memory, you do not need SVC_REQ 36. Simply map variables to %W memory.
- The BMA is a contiguous block of memory (up to 4MB) that can be used by logic and external devices. Before BMA can be used, its size must be set to a value greater than zero in the Settings tab for the CPX or CPE processor configuration. On a CPX, the acceptable range is 0 through 4 Megabytes, in 4 Kilobyte increments. On a CPE, the acceptable range is 0 through 4 Megabytes. Bulk memory area size cannot be greater than the daughterboard memory.

Operation

Use SVC_REQ 36 to read from the BMA to a local buffer area in the Controller, write from the local buffer area to the BMA, and retrieve the address of the BMA in the Controller. The only check done by the Controller is verifying that the specified local buffer does not fall outside the minimum and maximum available addresses for the application. The application must provide a valid local buffer area. Otherwise, unexpected results can occur.

SVC_REQ 36 uses an input parameter block that consists of 11 words. The parameter block format determines how the local buffer is specified and is selected by the MSB (most significant bit) in the Operation word. The format can be Segment/Offset (MSB cleared) or Address (MSB set).

If a requested operation in Service Request 36 fails, the instruction does not pass power flow. If the BMA is not configured or if the parameter block contains invalid parameters, a read or write operation will fail. Only invalid input parameters cause the Get BMA Size operation to fail, in which case the instruction does not pass power flow. If a read or write operation is requested and the BMA is unconfigured (that is, has a size of 0 in the Controller), the Controller logs an informational fault.

Segment/Offset Format

The Segment/Offset format specifies the local buffer as a segment selector and offset pair. These two parameters provide the location in a Controller reference table where the local buffer begins. This format is intended for use primarily by real-mode LD and C applications.

Address	Operation (MSB is cleared) 0001 = read from the BMA 0002 = write to the BMA

<table>
<tr><td></td><td>0004 = get BMA size</td></tr>
<tr><td>Address + 1</td><td>Segment selector (local buffer memory type)
<table>
<tr><th>Reference Memory</th><th>Segment Selector</th></tr>
<tr><td>%I</td><td>16</td></tr>
<tr><td>%T</td><td>20</td></tr>
<tr><td>%G</td><td>56</td></tr>
<tr><td>%SA</td><td>24</td></tr>
<tr><td>%SB</td><td>26</td></tr>
<tr><td>%SC</td><td>28</td></tr>
<tr><td>%Q</td><td>18</td></tr>
<tr><td>%M</td><td>22</td></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%W</td><td></td></tr>
<tr><td>(LD blocks only)</td><td></td></tr>
<tr><td>%L</td><td>0</td></tr>
<tr><td>%P</td><td>2</td></tr>
<tr><td>%SD</td><td>30</td></tr>
<tr><td>%I diagnostics</td><td>110</td></tr>
<tr><td>%Q diagnostics</td><td>112</td></tr>
<tr><td>%AI diagnostics</td><td>192</td></tr>
<tr><td>%AQ diagnostics</td><td>194</td></tr>
<tr><td>Rack/slot Report References</td><td>188</td></tr>
</table></td></tr>
<tr><td>Address + 2</td><td>Local buffer offset (low word)</td></tr>
<tr><td>Address + 3</td><td>Local buffer offset (high word)</td></tr>
<tr><td>Address + 4</td><td>Not used</td></tr>
<tr><td>Address + 5</td><td>Controller memory type (always 196 for BMA)</td></tr>
<tr><td>Address + 6 (low word)</td><td rowspan="2">Controller memory offset (low word and high word). Specifies the zero-based offset in bytes into the BMA where the read or write operation will begin. The maximum value for this field is the configured size of the BMA. If Operation is 0004 (Get BMA Size), these parameters are not used and should be set to 0.</td></tr>
<tr><td>Address + 7 (high word)</td></tr>
<tr><td>Address +</td><td>Not used</td></tr>
</table>

8	
Address + 9 (low word)	Length (low word and high word). If Operation is 0004 (Get BMA Size), these parameters are not used and should be set to 0.
Address + 10 (high word)	

Address Format

The Address format specifies the local buffer and address. This format is intended for use by C applications.

Address	Operation (MSB is cleared) 0001 = read from the BMA 0002 = write to the BMA 0004 = get BMA size
Address + 1	Local buffer pointer (low word)
Address + 2	Local buffer pointer (high word)
Address + 3	Not used
Address + 4	Not used
Address + 5	Controller memory type (always 196 for BMA)
Address + 6 (low word)	Controller offset (low word and high word). Specifies the zero-based offset in bytes into the BMA where the read or write operation will begin. The maximum value for this field is the configured size of the BMA. If Operation is 0004 (Get BMA Size), these parameters are not used and should be set to 0.
Address + 7 (high word)	
Address + 8	Not used
Address + 9 (low word)	Length (low word and high word). If Operation is 0004 (Get BMA Size), these parameters are not used and should be set to 0.
Address + 10 (high word)	

Example

Four bytes (%R26 and %R27) are written from a local buffer in Controller memory into the BMA. All offsets are zero-based, byte-offset numbers.
Word 1 (address + 0) specifies a Write (2=Write) operation using the Segment/Offset format. Most Significant Bit=0.
Word 2 (address + 1) specifies %R (8=%R) as the memory type for the local buffer that will be written from.
Words 3 and 4 form a DWORD, zero-based byte offset into the reference table being accessed. Word 3 is the low-ordered word.

- Word 3 (address + 2) specifies that the Controller's local buffer starts at %R26. To determine the byte offset, subtract 1 from the reference address, then multiply by 2 (in the example, byte offset=2 (26-1)=50). For BIT-oriented tables, subtract 1 from the address, then divide by 8. For example, for %I9, offset=(9-1)/8=1.
- Word 4 (address + 3) specifies the upper word address of the local buffer.

Word 5 is spare. Set to 0.

Word 6 (address + 5) must always have a value of 196, which selects the BMA.

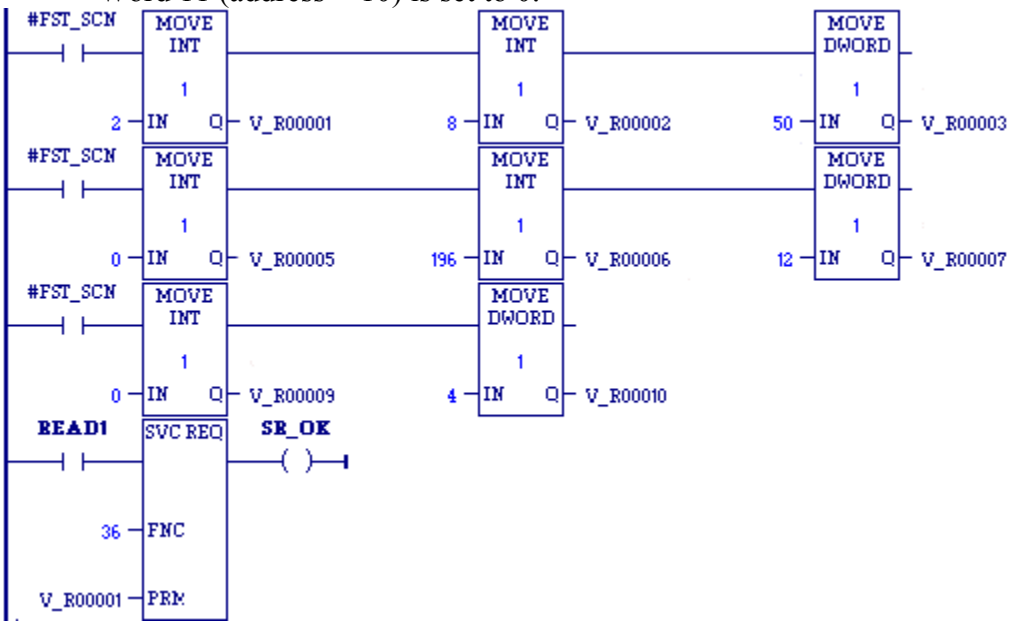
Words 7 and 8 form a DWORD, zero-based byte offset into the BMA. Word 7 is the low-ordered word.

- Word 7 (address + 6) specifies that the write operation starts at an offset of 12 bytes in the BMA.
- Word 8 is set to 0.

Word 9 is spare. Set to 0.

Words 10 and 11 form a DWORD specifying the number of bytes to be transferred. Word 10 is the low-ordered word.

- Word 10 (address + 9) specifies a length of four bytes to write.
- Word 11 (address + 10) is set to 0.



SVC_REQ 39: ESCM Port Status

CPU Support

SVC_REQ 39 is supported for CPX and CGR Series 90-70 Controllers.

Operation

Use SVC_REQ 39 ESCM (Embedded Serial Comm Module) Port Status to provide status of Ports 1 and 2 to the Controller CPU. The address specified in the parameter should be set to a value of 1 or 2 (indicating Port 1 or 2). The port status is returned in the following value:

Address	1 = Port 1 2 = Port 2
Address + 1	The location in which the status is returned

Successful execution occurs unless:

- The first parameter is not 1 or 2.
- The CPU does not support use of Ports 1 and 2.

Note: Only CPX and CGR models of CPUs support Ports 1 and 2.

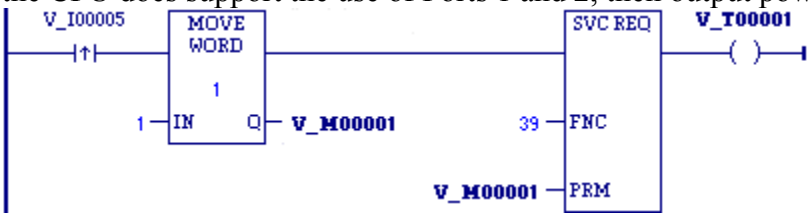
Output

The ESCM status information is delivered in the form of a word of bit-encoded data. Each bit of the first four bits (beginning with the Least Significant Bit) indicates something about the status of the ESCM as described in this table shown below:

State	*Bit Position*	*Description*
PORTN_OK	0	Requested port is ready. If value is 1, the port is ready. If value is 0, the port is not usable.
PORTN_ACTIVE	1	There is activity on this port. If value is 1, the port is active. If value is 0, the port is inactive.
PORTN_DISABLED	2	Requested port is disabled. If value is 1, the port is disabled. If value is 0, the port is enabled.
PORTN_FUSE_BLOWN	3	Requested port's fuse is blown (for Port 2) or supply voltage is not within range (for Port 1). If value is 1, the fuse is blown (or voltage not within range). If value is 0, the fuse (or supply voltage) is okay.
RESERVED	4-16	Set to zero (reserved).

Example

When SVC_REQ 39 executes, the CPU determines the port number from the command word %M00001 and validates the port number. It then retrieves the current corresponding ESCM status word and places the retrieved status word into %M00017.
If the port number is invalid, or if input power flow is off, or if the current CPU does not support the use of Ports 1 and 2, then output power will be OFF. If the port is valid and the CPU does support the use of Ports 1 and 2, then output power will be ON.



SVC_REQ 43: Disabling Data Transfer Copy in Backup Unit

CPU Support

SVC_REQ 43 is supported for:

- PACSystems redundancy CPUs (CRE and CRU models).
- Series 90-70 CPUs IC697CGR772, IC697CGR935, and IC697CPU780.

Operation

Use SVC_REQ 43 on the backup unit of a CPU redundancy system or CPU over Genius redundancy system to allow the backup unit to bypass the copy of the shared I/O data from the active unit. This operation can help determine if the active and backup CPUs are arriving at the same results.

SVC_REQ 43 is ignored if issued when the units are not synchronized, or if issued in the active unit.

SVC_REQ 43 disables the copy of data for one sweep beginning with the output data transfer and ending with the input data transfer of the next sweep. The copy can be disabled for multiple sweeps by invoking SVC_REQ 43 once each sweep for the appropriate number of sweeps.

The special resynchronization data transfer always occurs, even if SVC_REQ 43 is invoked in the first sweep after synchronization (this data transfer includes all shared inputs, all shared outputs, and the internal data that must be exchanged) since the resynchronization data transfer occurs before the start of logic execution.

SVC_REQ 43 can be set up to disable the copies for all transfers or just the output transfers. If just the output copy is disabled, the two units can still use the same set of inputs on each unit. This enables you to test the ability of the two units to derive the same results from the same inputs.

In all cases, the configured data transfers are still transferred over the link every sweep and the rendezvous points are still met. The effect of SVC_REQ 43 is to disable the copy of the data from the transfer to the actual reference memory areas configured.

Warning: When SVC_REQ 43 is in effect, the backup unit still takes control of the redundancy system in event of a failure or role switch. Switches to the backup unit may cause a momentary interruption of data of the outputs since the two units may not be generating the exact same results.

Consider disabling outputs on the backup unit while SVC_REQ 43 is in effect. Disabling outputs on the backup unit eliminates the risk of an unsynchronized switch of control (which can cause a momentary interruption of data in the outputs) if the active unit fails or loses power while the input/output copies are disabled. However, if the active unit does fail or lose power while outputs are disabled on the backup unit, the system's outputs will go to their default settings. A secondary effect of disabling outputs on the backup unit is that the unsynchronized fault action table is used by the active unit to determine which faults are fatal.

Note: If the CPU is already in RUN/ENABLED mode, a command to disable its outputs will not take effect until one sweep after the command is received. Therefore, disable the outputs at least one sweep before you enable SVC_REQ 43.

SVC_REQ 43 can be used with any control strategy, that is HSB on a PACSystems and GHS and GDB on a Series 90-70. However, with the HSB and GDB control strategies, SVC_REQ 43 cannot be used to disable output data transfer on the Primary unit when outputs are enabled on the Primary Unit. If that is attempted, SVC_REQ 43 is rejected. A fault is logged the first time SVC_REQ 43 is used as a warning that the Controllers are not completely synchronized.

The reverse data transfer, if any, is unaffected by SVC_REQ 43.

Enabling logic should be used with SVC_REQ 43. A contact with a non-transferred reference should be part of this enabling logic. That will allow SVC_REQ 43 to be activated or deactivated directly without being overwritten with the value from the active unit.

If SVC_REQ 43 is invoked multiple times in a single sweep, the last call is the one that determines the action taken.

SVC_REQ 43 has an input parameter block and no output parameter block.

Successful execution occurs unless:

- The values in the input parameter block are out of range.
- SVC_REQ 43 was invoked when the redundant units in a CPU redundancy system or CPU over Genius redundancy system were not synchronized.
- SVC_REQ 43 was issued to the active unit.
- The CPU does not support SVC_REQ 43.

If SVC_REQ 43 is unsuccessful, it does not pass power flow to the right.

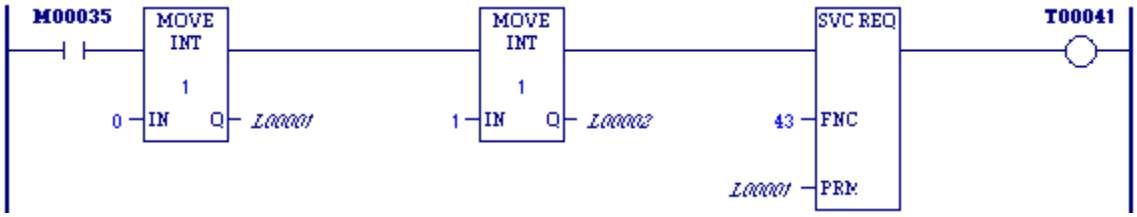
Input

The parameter block has a length of 2 WORDs.

Address	0
Address+1	1 = disable input and output copies 2 = disable output copy only

Example

In the following example, when %M00035 is on, the input and output copies are disabled.



Validating the backup unit

You can use SVC_REQ 43 to validate the input scan, that is, to determine if the backup Controller unit is collecting inputs properly. You can also use it to validate the logic solution, that is, to determine whether the backup Controller unit is calculating outputs and internal variables properly.

To validate the backup unit's input scan:

1. Activate SVC_REQ 43 on the backup unit, passing the values 0 and 1 to disable the input and output data transfer copies.
2. Observe the backup unit's %I and %AI memory areas. The values in these memory areas correspond to the inputs that the backup is currently collecting.
3. Visually compare the backup unit's %I and %AI memory areas with the active unit's memory areas. Pay special attention to the %I and %AI reference addresses that are configured to be shared between the two units.
4. When you are satisfied that the backup unit is collecting inputs properly, disable the rung that calls SVC_REQ 43.

To validate the backup unit's logic solution:

1. Activate SVC_REQ 43 on the backup unit, passing the values 0 and 2 to disable the output data transfer copy.
2. Observe the backup unit's %Q, %AQ, %M, and %R memory areas. On a PACSystems also observe %W. The values in these memory areas correspond to the outputs that the backup is currently calculating.
3. Visually compare the backup unit's %Q, %AQ, %M, %R, and %W memory areas with the active unit's memory areas. Pay special attention to the %Q, %AQ, %M, %R, and %W reference addresses that are configured to be shared between the two units.
4. When you are satisfied that the backup unit is calculating outputs and internal variables properly, disable the rung that calls SVC_REQ 43.

SVC_REQ 44: Logic Driven Dynamic Ethernet Global Data

CPU Support

SVC_REQ 44 is supported for Version 7.91 and later CPX Series 90-70 CPUs. The adapter module IC697CMM742, version 2.70 or later, supports Logic Driven EGD.

Operation

Use SVC_REQ 44 to establish, terminate, and monitor Ethernet Global Data exchanges from within the logic. Hardware configuration of Ethernet Global Data is not required; however, the adapter(s) must be configured by the Configuration software.
The adapter module IC697CMM742, version 2.70 or later supports Logic Driven EGD.
SVC_REQ 44 must not be issued (that is, from an interrupt block) while another SVC_REQ 44 is in process in the same CPU.

Note: On the scan that an exchange is established, the CPU will require more time during the start of sweep processing. Furthermore, if you are starting multiple LD EGD connections during a single scan, the watchdog timeout may need to be adjusted.

Service Request Instruction

Address	Status. Indicates success or failure of request. Consult the set of possible responses for each command. The user should set this field to an initial value of 0.
Address + 1	Command. Indicates which command is to be executed: 1 Set local producer ID 2 Retrieve local producer ID 3 Establish a produced exchange 4 Establish a consumed exchange 5 Terminate produced exchange 6 Terminate consumed exchange 7 Refresh production data every sweep
Address+2 ... Address+n	Command Specific Information Block. The format of the remaining fields depends on the command selected.

For details, see *Series 90-70 PLC CPU Instruction Set*, GFK-0265J, pp. 12-75 through 12-83.

Output

The following general errors can be returned in the status. Other values are possible for each command.

Value	*Description*	*Power Flow*

-18	Invalid command. Not in range of 1 – 7.	No
-19	Parameter Block length exceeds memory range.	No
-23	SVC_REQ 44 power flow interrupted. That is, a new command was issued before the last one completed.	No

For details, see *Series 90-70 PLC CPU Instruction Set*, GFK-0265J, pp. 12-76 through 12-84.

SVC_REQ 45: Skip Next Output and Input Scan (Suspend I/O)

CPU Support

SVC_REQ 45 is supported for the following GE IP Controllers:

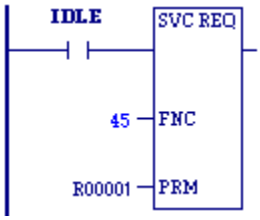
- Series 90-30 release 10 or later of the 350, 352, 360, 363, and 364 CPUs; only for use with the DSM324i and Motion Mate DSM314 modules.
- PACSystems RX3i; only for use with the DSM324i and Motion Mate DSM314. This service request is supported by PACSystems RX3i for easy conversion of Series 90-30 applications. The Suspend I/O (SUS_IO) instruction, which is supported by all PACSystems firmware versions, should be used in new applications.

Operation

Use SVC_REQ 45 to skip the next output and input scans. Any changes to the output reference tables during the scan in which the SVC_REQ 45 was executed is not reflected on the physical outputs of the corresponding modules. Any change to the physical input data on the modules is not reflected in the corresponding input references during the scan after the one in which the SVC_REQ 45 was executed.
SVC_REQ 45 has no parameter block.
The DO_IO instruction is not affected by the use of SVC_REQ 45. It still updates the I/O when used in the same logic as SVC_REQ 45.

Example

When the "Idle" contact passes power flow, the next output and input scans are skipped.



SVC_REQ 46: Fast Backplane Status Access

CPU Support

SVC_REQ 46 is supported for Series 90-30 release 10 or later of the 350, 352, 360, 363, and 364 CPUs; only for use with the DSM324i and Motion Mate DSM314 modules).

Operation

Use SVC_REQ 46 to perform one of the following fast backplane status access instructions:

- Read a word of extra status data from one of more specified smart modules.
- Write a word of extra status data from one of more specified smart modules.
- Read/Write: Read a word of extra status data from one or more specified modules and write the data value between 0 and 15 to the same module, all in one operation.

Notes

- A COMM_REQ or DO_IO instruction should not be performed with the specified module(s) during the same logic scan during which either data write instruction is performed, since COMM_REQ and DO_IO can cause the write data to be lost.
- Two instructions that write to a module (Write or Read/Write) should not be performed with the same module during the same logic scan because they can cause the first write data to be lost.

The first word of the parameter block determines which instruction to use. The remaining words are used for output, with a different output format for each instruction. The first word has the following format:

Address	1 = Read extra data
	2 = Write extra data
	3 = Read/write extra data

Read Extra Status Data

The Read Extra Data instruction reads a word of extra status data from each of the modules specified by a list in the parameter block and places the status data values into the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data is written.

Location	*Field*	*Description*
Address	Instruction	1 = read extra status data
Address + 1	Error Code	An error code is placed here if the instruction fails because any of the modules is not present, inappropriate, or not working. For details, see Error Codes.
Address +	Error rack	The rack and slot number at which the error occurred

2	& slot	
Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module from which the data is read
Address + 4	Read data from first module	The data read from the first module
Address + 5	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module from which the data is read
Address + 6	Read data from second module	The data read from the second module
Address + (I * 2) + 1	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module from which the data will be read
Address + (I * 2) + 2	Read data from Ith module	The data read from the Ith module
Address + (N * 2) + 1	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module from which the data is read
Address + (N * 2) + 2	Read data from last module	The data read from the last module
Address + (N * 2) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

Write Data

The write data instruction writes a data value between 0 and 15 from the parameter block to one or more modules specified by a list in the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data is written.

Location	*Field*	*Description*
Address	Instruction	2 = write data
Address + 1	Error Code	An error code is placed here if the instruction fails because any of the modules is not present, inappropriate, or not working. No error code is set if the instruction executes but any of the modules does not receive the write data properly. For details, see Error Codes.

Address + 2	Error rack & slot	The rack and slot number at which the error occurred
Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module to which the data is sent
Address + 4	Write data for first module	The data value written to the first module
Address + 5	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module to which the data is sent
Address + 6	Write data for second module	The data value written to the second module
Address + (I * 2) + 1	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module to which the data is sent
Address + (I * 2) + 2	Write data for Ith module	The data value written to the Ith module
Address + (N * 2) + 1	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module to which the data is sent
Address + (N * 2) + 2	Write data for last module	The data value written to the last module
Address + (N * 2) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

Read/Write Data

The read/write instruction reads a word of extra status data from a module specified in the parameter block, then writes a data value between 0 and 15 from the parameter block to that module. This read write process is repeated for each module in a list in the parameter block. The parameter block requires (N * 3) + 3 words of reference memory, where N is the number of modules with which data is exchanged.

Location	*Field*	*Description*
Address	Instruction	3 = read/write
Address + 1	Error Code	An error code is placed here if the instruction fails because any of the modules is not present, inappropriate, or not working. No error code is set if the instruction executes but any of the modules does not receive the write data properly. For details, see Error Codes.
Address + 2	Error rack & slot	The rack and slot number at which the error occurred

Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module with which data is exchanged
Address + 4	Read data from first module	The data read from the first module
Address + 5	Write data for first module	The data value written to the first module
Address + 6	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module with which data is exchanged
Address + 7	Read data from second module	The data read from the second module
Address + 8	Write data for second module	The data value written to the second module
Address + ((I-1) * 3) + 3	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module with which data is exchanged
Address + ((I-1) * 3) + 4	Read data from Ith module	The data read from the Ith module
Address + ((I-1) * 3) + 5	Write data for Ith module	The data value written to the Ith module
Address + ((N-1) * 3) + 3	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module with which data is exchanged
Address + ((N-1) * 3) + 4	Read data from last module	The data read from the last module
Address + ((N-1) * 3) + 5	Write data for last module	The data value written to the last module
Address + (N * 3) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

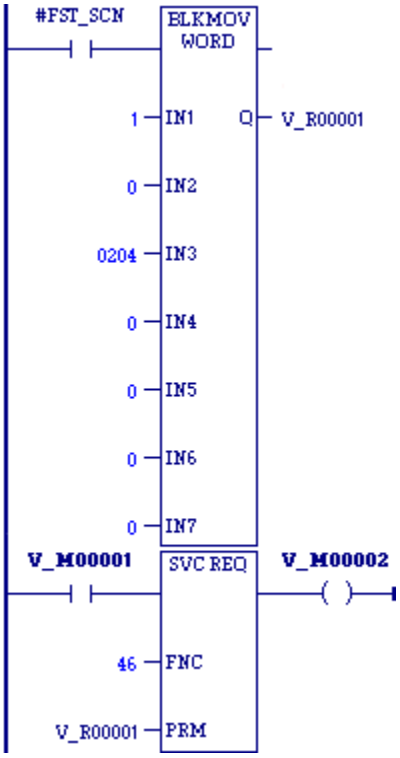
Error Codes

Value	*Description*
1	Success: the instruction has executed normally.

-1	Module not present in the specified slot.
-2	Module inappropriate: module in the specified slot is not a smart module or does not support this functionality.
-3	Module not working: module in the specified slot is not communicating with the CPU properly.
-4	Read data parity error: parity error occurred during a read operation from an expansion or remote rack.
-5	Invalid instruction specified in the command block.

Example

The logic reads a single module at Rack 2, Slot 4. IN4 and IN5 must be set to zero (0). IN6 and IN7 are not important in this example. If the instruction completes successfully, the data will be in %R0004.



SVC_REQ 48: Auto Reset

CPU Support

SVC_REQ 48 is supported for Series 90-30 release 10 or later of the 350, 352, 360, 363, and 364 CPUs; only for use with the DSM324i and Motion Mate DSM314 modules).

Operation

Reboot after Fatal Fault enables the Series 90-30 Controller system to automatically resume normal operation after a fatal fault has occurred. Following the fatal fault, the Controller will automatically reset, and resume execution. The faults will not be cleared, but will be treated as non-fatal. If fatal faults are present following the power up, the Controller will still be allowed to transition to run mode. This feature is enabled by the Ignore Fatal Faults (or Fatal Fault Override) parameter in the CPU's hardware configuration.

Use SVC_REQ 48 to set the maximum number of retries and the time period during which these retries can occur.

If the number of retries allowed within the time period is exceeded, the CPU mode is set to STOP/FAULT. If the period is 0, the CPU mode is set to STOP/FAULT when the number of retries allowed is exceeded.

If the operator cycles power, fatal faults are ignored. The current fault count and time period are initialized. The total number of fatal faults is unchanged, but the total number of retries is incremented.

System bit %S21 is set to 1 whenever retry is successful and remains set until all fatal faults are cleared, or the mode is set to STOP/FAULT.

The parameter block has a length of four words. Before using SVC_REQ 48, set up the parameter block as follows:

Input Parameter Block

Address	*Description*
Address	Service request status. Logic must initialize this word to zero. SVC_REQ 48 uses this word for its output.
Address+1	Unlimited retries: ▪ 0 = Disable (Number of retries is set by Word 3.) ▪ 1 = Enable (Words 3 & 4 ignored.)
Address+2	Number of retries allowed. Range: 0 to 128 ▪ 0 = Automatic reboot is disabled. ▪ 1 to 128 = Maximum number of retries that are allowed to occur within the period set in address+3.
Address+3	Retry period (minutes). Range: 0 to 5,940 minutes (99 hours) ▪ 0 = No time limit on maximum number of retries set in Word 3. Auto Reboot will be allowed for the number of retries.

	▪ 1 to 5,940 = Auto Reboot is disabled if the number of retries specified is exceeded within the period specified.

Output

SVC_REQ 48 uses the first word of the parameter block for the Return Status.

Status	*Description*	*Notes*	*Power Flow*
-5	Invalid retry period	Valid range: 0 to 5,940	No
-4	Invalid no. of retries	Valid range: 0 to 128	No
-3	Invalid unlimited retries	Must be 0 or 1	No
-2	Configuration disabled	Ignore Fatal Faults (Fatal Fault Override) option must be enabled in hardware configuration.	No
0	No action	Command requires no change.	Yes
1	Auto reset enabled	Valid command enables Reboot after Fatal Fault.	Yes
2	Auto reset disabled	Valid command disables Reboot after Fatal Fault. Ignore Fatal Faults remains enabled.	Yes

SVC_REQ 50: Read Elapsed Time Clock (Two DWORDs)

CPU Support

SVC_REQ 50 is supported for PACSystems CPUs.

Operation

Use SVC_REQ 50 to read the system's elapsed time clock. The elapsed time clock measures the time since the CPU was powered on and SVC_REQ 50 expresses it in seconds and nanoseconds.
The parameter block has a length of two DWORDs (four words) used for output only. The resolution of the Controller's elapsed time clock may vary depending on the hardware platform. The overall accuracy of the elapsed time clock is +/- 0.01%. The accuracy of an individual sample of the elapsed time clock may be approximately 105 microseconds.

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 50 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Output

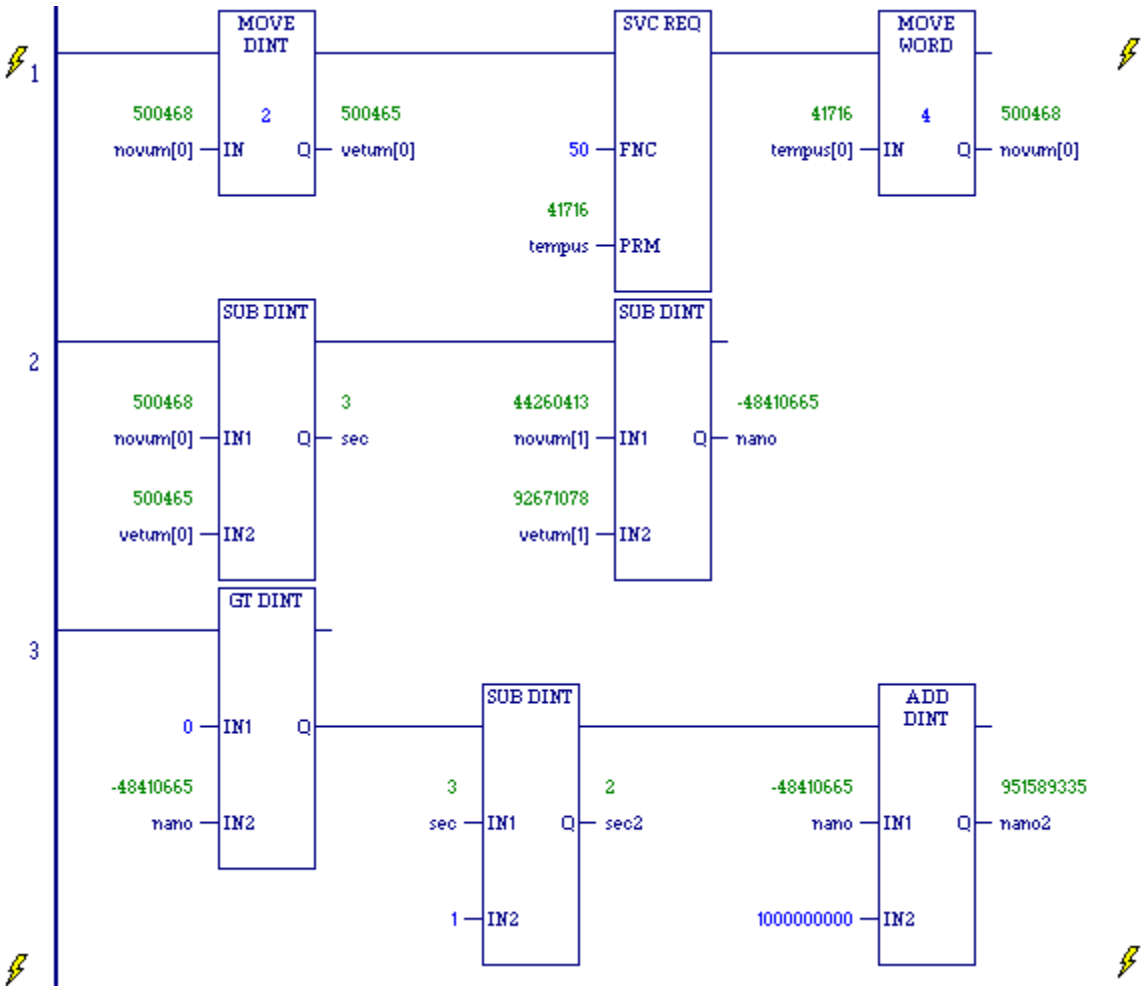
address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	Approximate nanosecond ticks (low order)
address+3	Approximate nanosecond ticks (high order)

The first two words contain the elapsed time in seconds. The last two words contain the approximate number of nanoseconds in the current second. (The hardware does not count nanoseconds. It may count 100 microsecond ticks that are expressed as nanoseconds.)

Warning: The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 50 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic.

Example

The following logic is used in a block that is called once in a while. The screen shot was taken between calls to the block. The second rung of logic calculates the number of seconds that have elapsed since the last time the block was called and the number of nanoseconds to be added to, or subtracted from, the number of seconds. If a subtraction is required, the third rung performs the subtraction for you to display the number of seconds and the positive number of nanoseconds that have elapsed since the last time the block was called. The first rung saves the previous value of novum[0] and novum[1] into vetum[0] and vetum[1] before SVC_REQ 50 and the MOVE_WORD instruction place the current time values in novum[0] and novum[1].



More specifically, in the first rung, when the block is called, SVC_REQ 50 reads the system's elapsed time clock and places the time values in tempus, a 4-WORD array; each WORD is 16 bits long. The MOVE_WORD instruction moves the four WORDs into novum, a 2-DINT array; each DINT is 32 bits long. This move amounts to a rough data type conversion that ignores the fact that the DINT type is actually a signed value. Despite that, the subsequent calculations are correct until the time since power-on reaches approximately 50 years. The value for tempus[0], 41716, is a 16-bit value. After the move, novum[0] is set to 500468, a 32-bit value that reads the values of tempus[0] and tempus[1] and expresses them as a DINT number.

On the second rung, the first SUB_DINT instruction subtracts the previous value of the seconds from power-on, stored in vetum[0], from the current value of the seconds from power-on, stored in novum[0]. The result, sec, is the number of seconds that have elapsed since the last time the block was called. Likewise, the second SUB_DINT instruction calculates the difference in the numbers of nanoseconds.

If the difference in nanoseconds is negative, a subtraction is required. Look up the values in the third rung, which performs the subtraction for you to display the number of seconds and the positive number of nanoseconds that have elapsed since the last time the block was called.

Alternative

You could convert the elapsed times into REAL values to output a final value in seconds. See the example for the similar service request SVC_REQ 16, Read Elapsed Time Clock.

SVC_REQ 51: Read Sweep Time from Beginning of Sweep (DWORD)

CPU Support

SVC_REQ 51 is supported for PACSystems CPUs.

Operation

Use SVC_REQ 51 to read the time (expressed in nanoseconds) since the start of the scan. The data is DWORD (two words, that is, four bytes).

Output

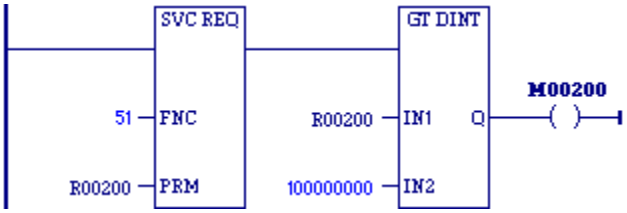
The parameter block is an output parameter block only; it has a length of two words (one DWORD). The time is expressed in nanoseconds, but the actual resolution is closer to 100 microsecond ticks and may vary depending on the hardware platform.

address	time since start of scan (low order)
address + 1	time since start of scan (high order)

If the time does not fit into a DWORD, the value is clamped at 0xFFFFFFFF and the OK output is disabled.

Example

The elapsed time from the start of the scan is always read into locations %R0200 and %R0201. If the time is greater than 100ms (100,000,000 nanoseconds), internal coil %M0200 is turned on.



SVC_REQ 52: Read from Flash

CPU Support

SVC_REQ 52 is supported for VersaMax Micro CPUs release 3.00 or later.

Operation

Use SVC_REQ 52 to specify which part of memory to read from flash and the memory location in CPU memory in which to place the read data.

Input

The parameter block is a parameter block used for input. It has a length of seven words (14 bytes).

<table>
<tr><td>address</td><td>Source memory, that is, the memory area to read from flash memory.
<table>
<tr><th>For this memory type</th><th>Enter this decimal value</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
</table></td></tr>
<tr><td>address + 1</td><td>The zero-based offset N to read from flash memory.
For %R, %AI, and %AQ memory, N = Ra - 1, where Ra = one-based reference address. For example, to read from the one-based reference address %R200, enter the zero-based reference offset 199.
For %I, %Q, %M, %T, and %G memory, N = (Ra - 1) / 8. For example, to read from the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.</td></tr>
<tr><td>address + 2</td><td>Length, that is, the number of words (16-bit registers) to read from %R, %AI, or %AQ flash memory, or the number of bytes to read from %I, %Q, %M, %T, or %G flash memory, beginning at the reference address calculated from the offset defined at [address + 1].
Valid range: 1 through 10. This value must reside in the low byte. The high byte must be set to zero.</td></tr>
<tr><td>address + 3</td><td>Destination memory, that is, the CPU memory area to write the read data to. This does not need to be the same memory area as specified at [address].
<table>
<tr><th>For this memory type</th><th>Enter this decimal value</th></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
</table>
</td></tr>
<tr><td>address + 4</td><td>The zero-based offset N in CPU memory to start writing the read data to.
For %R, %AI, and %AQ memory, N = Ra - 1, where Ra = one-based reference address. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199.
For %I, %Q, %M, %T, and %G memory, N = (Ra - 1) / 8. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.</td></tr>
<tr><td>address + 5</td><td>The CPU memory area used to write the status to.
<table>
<tr><td>For this memory type</td><td>Enter this decimal value</td></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
</table>
</td></tr>
<tr><td>address + 6</td><td>The zero-based offset N in CPU memory to start writing the status to.
For %R, %AI, and %AQ memory, N = Ra - 1, where Ra = one-based reference address. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199.
For %I, %Q, %M, %T, and %G memory, N = (Ra - 1) / 8. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.</td></tr>
</table>

Status

SVC_REQ 52 outputs a status of its read from flash operation. The status is written to the CPU memory location specified in [address+5] and [address+6] of the parameter block. The status is two words long, that is, two 16-bit registers long.

First Word

The first word of the status of the read from flash operation is broken down as follows:

- The low byte contains the major error code.
- The high byte contains the minor error code.

For example, 0001 means minor error code 00, major error code 01.

Minor	*Major*	*Meaning*
00	01	SVC_REQ 52 is successful
01	02	Memory location is invalid, does not exist in Controller
02	02	Length invalid
03	02	Insufficient destination memory, that is, an attempt was made to write the data read from flash memory past the configured limit for the Controller memory.

Second Word

The second word of the status of the read from flash operation is broken down as follows:

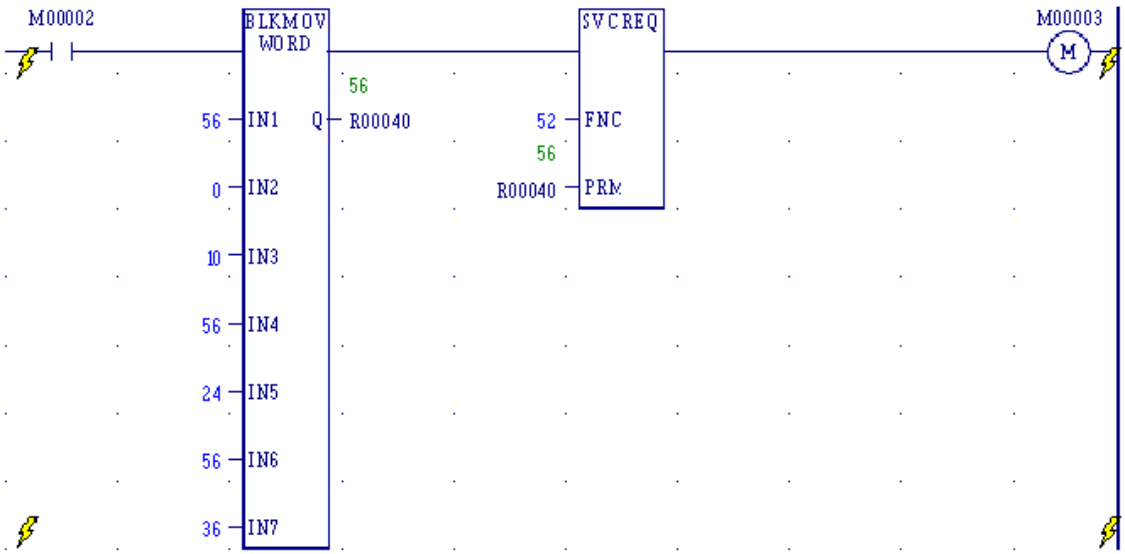
- The upper six bits are unused.
- Each of the other ten bits indicates the success or failure of the read for each reference address. The format is as follows: UUUU UU[RA10][RA9] [RA8][RA7][RA6][RA5] [RA4][RA3][RA2][RA1], where U represents unused bits and [RA10] through [RA1] represent the success or failure of reference address 10 through reference address 1. A 1 indicates success, a 0 indicates failure.

Example: Reading %M1 through %M3 is successful and reading %M4 and %M5 is unsuccessful. The status reads UUUU UUUU UUU0 0111, where U are unused bits.

Examples

Example 1

To read ten continuous bytes of %G memory ranging from %G1 through %G80 from flash memory, you can use the following LD logic. The segment selector for the %G memory area, 56, is for byte access and so are the offsets and lengths.



Example 2

Overwriting data in the same memory area. See SVC_REQ 53, Example 2.

SVC_REQ 53: Write to Flash

CPU Support

SVC_REQ 53 is supported for VersaMax Micro CPUs release 3.00 or later.

Operation

Use SVC_REQ 53 to specify which part of memory to read from the CPU and write to flash memory.

Input

The parameter block is a parameter block used for input. It has a length of five words (ten bytes).

<table>
<tr><td>address</td><td>The number representing the memory area that contains the data to write to flash memory.
<table>
<tr><th>For this memory type</th><th>Enter this decimal value</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
</table></td></tr>
<tr><td>address + 1</td><td>The zero-based offset N in the memory area that contains the data to write to flash memory.
For %R, %AI, and %AQ memory, N = Ra - 1, where Ra = one-based reference address. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199.
For %I, %Q, %M, %T, and %G memory, N = (Ra - 1) / 8. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.</td></tr>
<tr><td>address + 2</td><td>Length, that is, the number of words (16-bit registers) to write to %R, %AI, or %AQ flash memory, or the number of bytes to write to %I, %Q, %M, %T, or %G flash memory, beginning at the reference address calculated from the offset defined at [address + 1].
Valid range: 1 through 10. This value must reside in the low byte. The high byte must be set to zero.</td></tr>
<tr><td>address + 3</td><td>The number that represents the CPU memory area used to write the status to.</td></tr>
</table>

	For this memory type / Enter this decimal value %R: 8 %AI: 10 %AQ: 12 %I (byte mode): 16 %Q (byte mode): 18 %M (byte mode): 22 %T (byte mode): 20 %G (byte mode): 56
address + 4	The zero-based offset N in the CPU memory area to start writing the status to. ▪ For %R, %AI, and %AQ memory, N = Ra - 1, where Ra = one-based reference address. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199. ▪ For %I, %Q, %M, %T, and %G memory, N = (Ra - 1) / 8. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.

Status

SVC_REQ 53 outputs a status of its write to flash operation. The status is written to the CPU memory location specified in [address+3] and [address+4] of the parameter block. The status is three words long, that is, three 16-bit registers long.

First Word

The first word of the status of the write to flash operation is broken down as follows:

- The low byte contains the major error code.
- The high byte contains the minor error code.

For example, 0001 means minor error code 00, major error code 01.

Minor	*Major*	*Meaning*
00	01	SVC_REQ 53 is successful
01	02	Memory location is invalid, does not exist in Controller
02	02	Length invalid
03	02	Insufficient flash memory

Second Word

The number of bytes written.

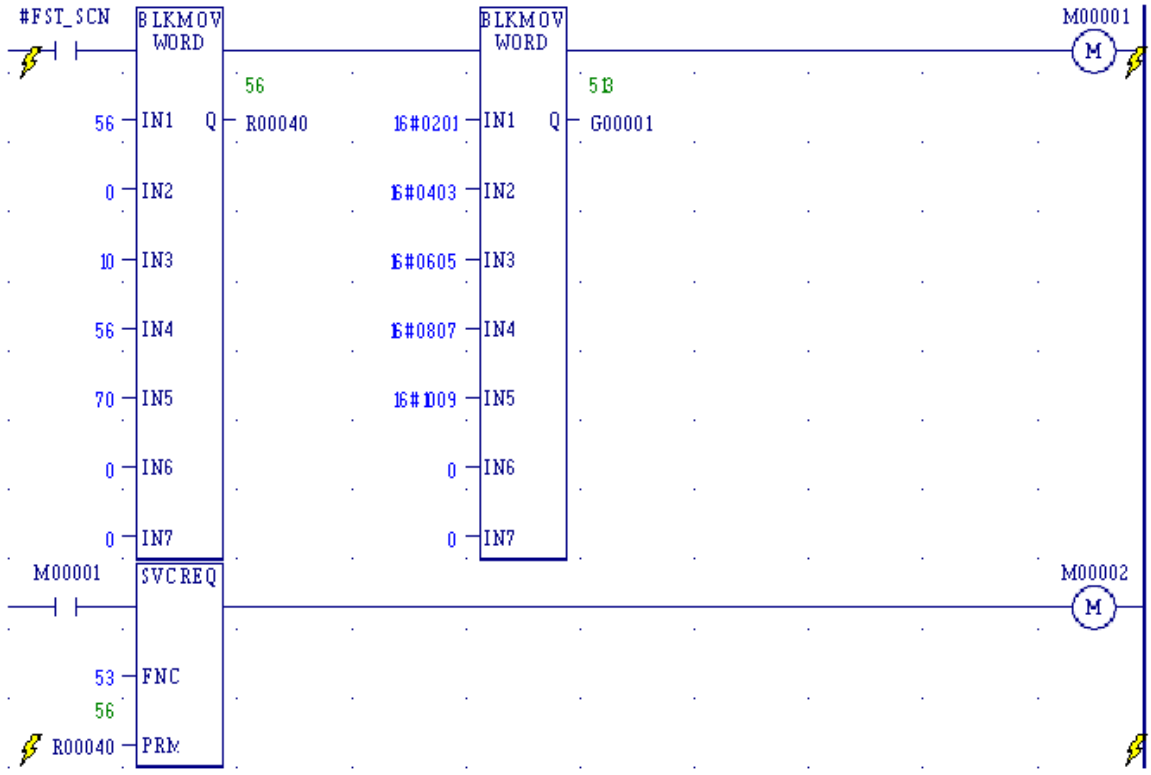
Third Word

The number of bytes still available in flash memory.

Examples

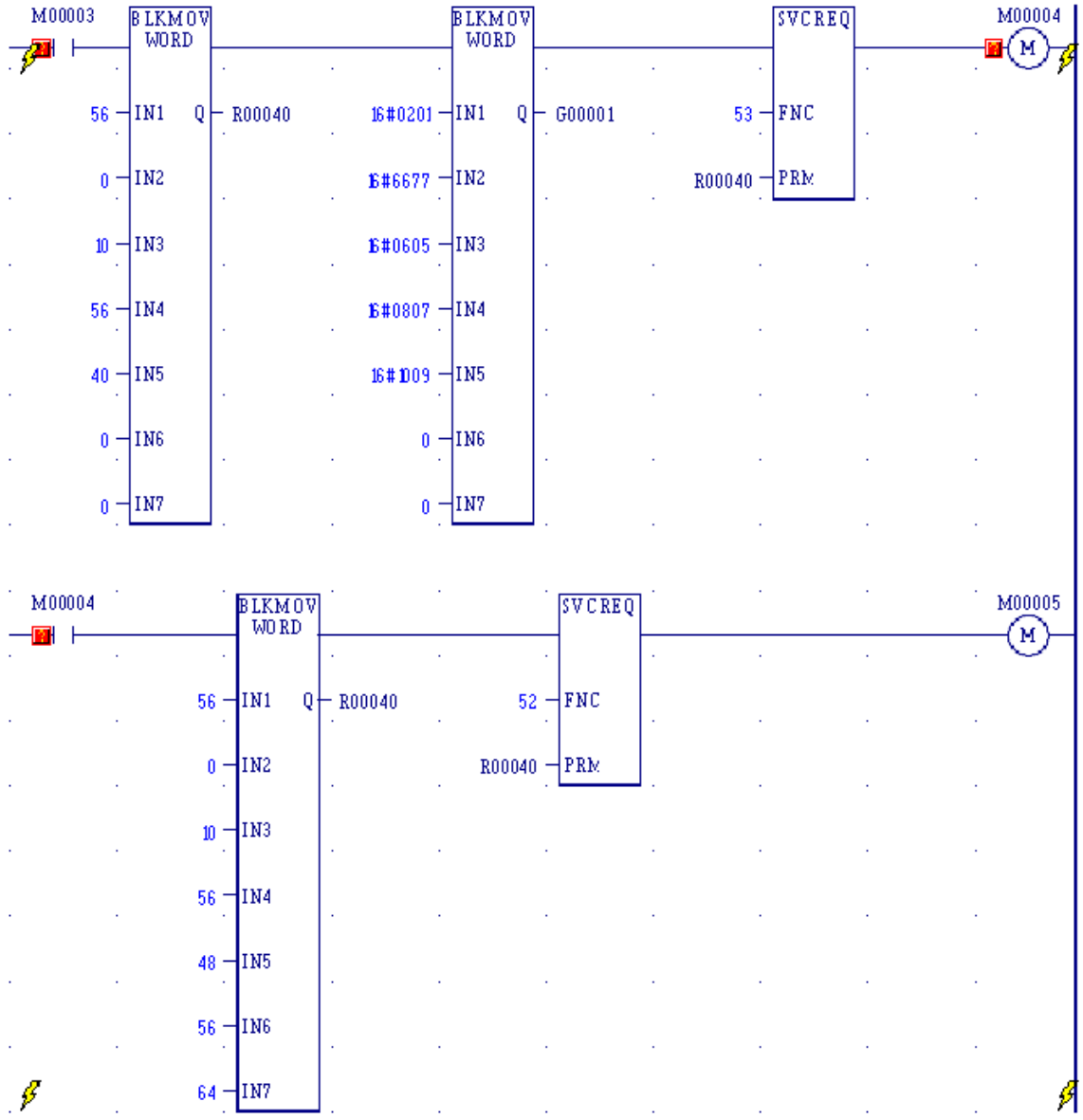
Example 1

To write ten continuous bytes of %G memory ranging from %G1 through %G80 into flash memory, you can use the following sample LD logic. The segment selector for the %G memory area, 56, is for byte access and so are the offsets and lengths.



Example 2: Overwriting data in the same memory area

The following example illustrates the use of SVC_REQ 52 and SVC_REQ 53.
To overwrite some of the data in the same memory area, we can use the same LD logic with different values. Look at the status words returned for the below SVC_REQ execution starting from %G321.


M00003
BLKMOV WORD
56 IN1
0 IN2
10 IN3
56 IN4
40 IN5
0 IN6
0 IN7
Q R00040
BLKMOV WORD
16#0201 IN1
16#6677 IN2
16#0605 IN3
16#0807 IN4
16#1009 IN5
0 IN6
0 IN7
Q G00001
SVCREQ
53 FNC
R00040 PRM
M00004
M
M00004
BLKMOV WORD
56 IN1
0 IN2
10 IN3
56 IN4
48 IN5
56 IN6
64 IN7
Q R00040
SVCREQ
52 FNC
R00040 PRM
M00005
M

SVC_REQ 55: Set Application Redundancy Mode

CPU Support

SVC_REQ 55 is supported for PACSystems non-HSB CPUs with firmware version 5.00 or later.

Operation

Use SVC_REQ 55 with a non-HSB CPU to request the CPU to send a redundancy role switch command to all Ethernet interfaces in the Controller that are configured for redundant IP operation.
Your application must monitor the LAN Interface Status (LIS) word for each Ethernet interface to determine whether the Redundant IP address is active at that interface.
SVC_REQ 55 is recognized in non-HSB CPUs only. SVC_REQ 55 has no effect on Ethernet interfaces that are not configured for redundant IP operation.

Input

The input parameter block has a length of one word (two bytes).

address	0 = Backup redundancy role 1 = Active redundancy role

SVC_REQ 56: Read from Nonvolatile Storage

CPU Support

SVC_REQ 56 is supported for PACSystems with firmware version 6.00 or later.

Operation

PACSystems Controllers support a 64KB nonvolatile flash memory area, which can be accessed by the logic-driven read/write service requests. Values are stored in the nonvolatile storage area by means of SVC_REQ 57. These values are applied to the Controller user memory on powerup.

If you want only to write to nonvolatile storage and have the values restored on a power cycle, you may not need to use SVC_REQ 56. However, a logic driven read from nonvolatile storage can be commanded as needed. For example, you can use #FST_SCN with SVC_REQ 56 calls to force a reload on each Stop to Run transition.

SVC_REQ 56 specifies a read operation from nonvolatile storage when the PACSystems is running. You can specify which reference address range to read and optionally a different destination memory location in CPU memory in which to place the read data. Using different memory locations enables you to set up a comparison between existing values in CPU memory with values in nonvolatile storage.

SVC_REQ 56 execution time varies depending on the number of values stored in nonvolatile storage, because it finds the most recent value for the requested reference address range.

Note: You can also read flash memory on a stopped Controller.

Data length

You can read up to 32 words (64 bytes) inclusively per invocation of SVC_REQ 56.

Discrete memory

Discrete memory can be read as individual bits or as bytes. For more information, see address.

If a discrete memory destination is forced, the forced value remains intact in CPU memory even though the count in word 10 (address + 10) indicates that all the data was read and transferred.

Transitions are affected. If a memory location has an associated transition bit and SVC_REQ 56 causes a transition on that value, the transition bit is set.

Maximum of one active instruction

When SVC_REQ 56 is active, it does not support an interrupt that attempts to activate SVC_REQ 57 or a second instance of SVC_REQ 56. If an attempt fails, an error indicating that another instance is active will be returned.

Storage disabled conditions

By default, the following write operations disable SVC_REQ 56 until logic is written to nonvolatile storage:

- Run Mode Store (RMS), even if a second RMS reverts everything to the original state
- Test-Edit session, even when you cancel your edits
- Word-for-word change
- Downloading to RAM only of a stopped PACSystems CPU, even if the downloaded contents are equal to the contents already on the nonvolatile storage

Setting input word 8 (address + 7) to a value of 1 enables SVC_REQ 56 despite the above conditions.

ENO and power flow to the right

If the status is Success or Partial Read (see address+9), then on the SVC_REQ instruction, ENO is set to True in FBD and ST, and power flow passes to the right in LD.

Input and Status

The parameter block has a length of 11 words (22 bytes). The first nine words contain the input you supply for the read operation. The last two words contain the status of the read operation.

<table>
<tr><td>address</td><td>Source memory, that is, the memory area to read from nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
<tr><td>%M (bit mode)</td><td>76</td></tr>
<tr><td>%G (bit mode)</td><td>86</td></tr>
<tr><td>%W</td><td>196</td></tr>
</table></td></tr>
<tr><td>address + 1 and address + 2</td><td>The zero-based offset N to read from nonvolatile storage. Address + 1, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 2 for offsets greater than 65,535.
▪ For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to read</td></tr>
</table>

<table>
<tr><td></td><td>from the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.
▪ For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to read from the one-based reference address %R200, enter the zero-based reference offset 199; to read from %I73 in bit mode, enter offset 72. For memory in bit mode, the offset must be set on a byte boundary, that is, a number exactly divisible by 8: 0, 8, 16, 24, and so on.</td></tr>
<tr><td>address + 3</td><td>Length. The number of items to read from nonvolatile storage beginning at the reference address calculated from the offset defined at [address + 1 and address + 2]. The length can be one of the following:
<table>
<tr><th>Description</th><th>Valid range</th></tr>
<tr><td>The number of words (16-bit registers) to read from %W, %R, %AI, or %AQ nonvolatile storage</td><td>1 through 32 words</td></tr>
<tr><td>The number of bytes to read from %I, %Q, %M, %T, or %G in byte mode nonvolatile storage</td><td>1 through 64 bytes</td></tr>
<tr><td>The number of bits to read from %I, %Q, %M, %T, or %G in bit mode nonvolatile storage</td><td>1 through 512 bits in increments of 8 bits</td></tr>
</table>
The value must reside in the low byte of address + 3. The high byte must be set to zero.</td></tr>
<tr><td>address + 4</td><td>Destination memory. The CPU memory area to write the read data to. This does not need to be the same memory area as specified at [address]. Writing to a different memory area enables you to compare the values that were already in the CPU with the values read from nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
</table></td></tr>
</table>

	%T (bit mode) 74 %M (bit mode) 76 %G (bit mode) 86 %W 196
address + 5 and address + 6	The zero-based offset N in CPU memory to start writing the read data to. Address + 5, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 6 for offsets greater than 65,535. ▪ For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to write to the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4. ▪ For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to write to the one-based reference address %R200, enter the zero-based reference offset 199; to write to %I73 in bit mode, enter offset 72.
address + 7	▪ When bit 0 is set to 1, storage disabled conditions are ignored. A read is allowed even if the logic in RAM has changed since nonvolatile storage was read or written. ▪ Bits 1 through 15 must be set to zero; otherwise, the read fails.
address + 8	Reserved. Must be set to zero; otherwise, the read fails.
address + 9	Response status. The status read from nonvolatile storage. ▪ The low byte contains the major error code. ▪ The high byte contains the minor error code.

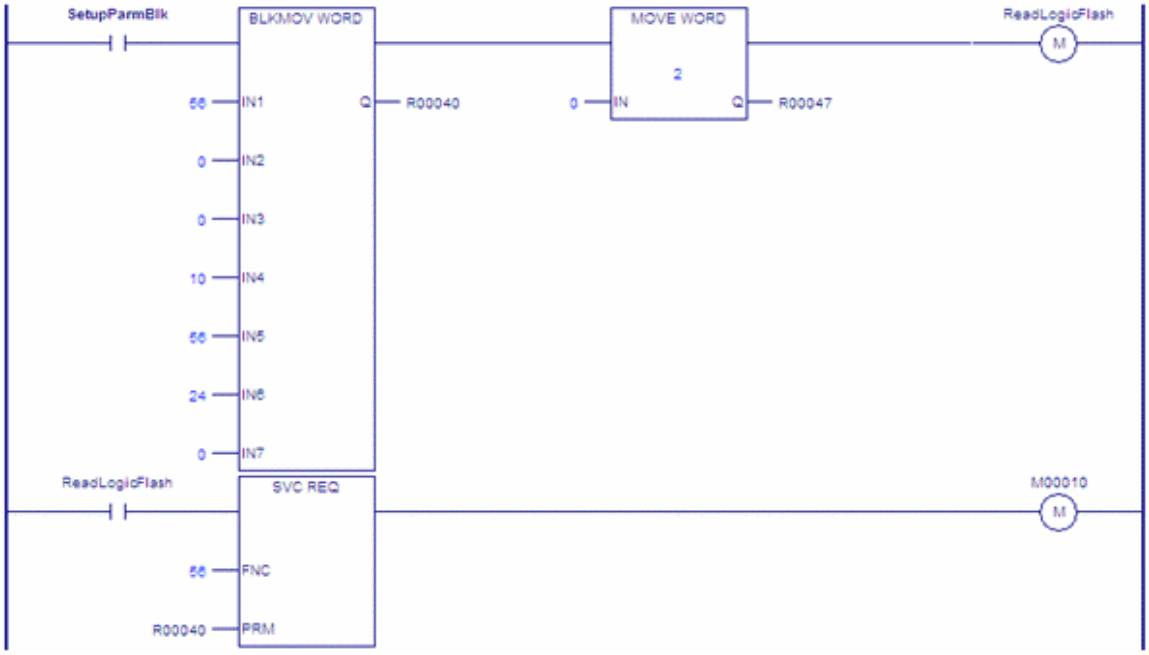
Minor	*Major*	*Description*
00	**01**	Success. All values requested were found and copied.
01	**01**	Partial Read. All values found were copied, but some or all values were not in storage.
01	**02**	Insufficient Destination Memory. The Destination memory location is not large enough to store the requested values.
02	**02**	Invalid Length. The length requested is larger than 64 bytes or less than 1 byte or the number of bits is not an exact multiple of 8.

	01	**03**	Storage Busy. A SVC_REQ 57 or another SVC_REQ 56 instruction is active. For example, an interrupt block is attempting to execute SVC_REQ 56 when the block it interrupted was executing SVC_REQ 56.
	01	**04**	Storage Disabled. The logic in RAM differs from the logic in nonvolatile storage. See Storage disabled conditions.
	02	**04**	Storage Closed. Either the storage has not been created or a previous corruption error or unexpected read/write failure closed the storage.
	01	**05**	Unexpected Read Failure. A command to the storage hardware failed unexpectedly.
	02	**05**	Corrupted storage. A corrupted checksum or storage header caused a read to fail.
address + 10	Response Count. The number of words, bytes, or bits copied.		

Examples

Example 1

The following LD logic reads ten continuous bytes written to nonvolatile storage, ranging from %G1 through %G80. The read values are written to the range %G193 thrugh %G273. The value applied to IN1, 56, determines byte mode.
The parameter block starts at %R00040. The response words are returned to %R00049 and %R00050.



Parameter block for Example 1:

Address + offset	Address	Input Value	Definition
address + 0	%R00040	56	Data type = %G (byte mode)
address + 1	%R00041	0	Address written from, to be read, low word
address + 2	%R00042	0	Address written from, to be read, high word
address + 3	%R00043	10	Length = 10 bytes
address + 4	%R00044	56	Data type to write to = %G (byte mode)
address + 5	%R00045	24	Address to write to, low word
address + 6	%R00046	0	Address to write to, high word
address + 7	%R00047	0	Storage disabled conditions are enforced.
address + 8	%R00048	0	Reserved. Must be set to 0.
address + 9	%R00049	n/a	Response status
address + 10	%R00050	n/a	Response count

Example 2

Overwriting data in the same memory area. See SVC_REQ 57, Example 2.

SVC_REQ 57: Write to Nonvolatile Storage

CPU Support

SVC_REQ 57 is supported for PACSystems with firmware version 6.00 or later.

Operation

PACSystems Controllers support a 64KB nonvolatile flash memory area, which can be accessed by the logic-driven read/write service requests. Values are stored in the nonvolatile storage area by using SVC_REQ 57. These values are applied to the Controller user memory on power up.
SVC_REQ 57 specifies a range of reference addresses to read from a running PACSystems CPU and write to nonvolatile storage. This functionality is intended to retain a limited set of values, such as set points or tuning parameters, that need to change when the PACSystems is running. To write data that is static when the PACSystems is stopped, see write flash memory on a stopped Controller.

Note: Nonvolatile storage is intended for storing values that do not change frequently. Once the nonvolatile storage area fills up, a power cycle or stop mode store is required to store more values. The logic -driven write is not a replacement for battery backed RAM for values that change frequently or during every sweep.

SVC_REQ 57 scans the nonvolatile storage to find the most recent values stored for the specified range. If it finds no values for the range or the most recent stored values are different, then the new values are written to nonvolatile storage.

Data length

You can write up to 32 words (64 bytes) inclusively per invocation of SVC_REQ 57. Each invocation requires 4 words of command data (8 bytes). A 1-byte write requires 9 bytes whereas a 64-byte write requires 72 bytes. You can generally make the most efficient use of nonvolatile storage by transferring data in 64-byte increments. See, however, Fragmentation.

Erase Cycles

The flash component on the PACSystems CPU is rated for 100K erase cycles. Erase cycles occur under the following conditions:

- Write to flash is commanded from the programmer
- Clear flash operation
- Flash compaction after a power cycle when flash memory allotted for SVC_REQ 57 has become full

Discrete memory

Discrete memory can be written to as individual bits or as bytes. For more information, see address.
Forced and transition information is not written to nonvolatile storage.

Retentiveness

Writing values to nonvolatile storage for non-retentive memory such as %T does not make the memory retentive. For example, all values stored to %T memory are set to zero on power-up or a stop to run transition. You can, however, read such values from storage after power-up or stop to run transition by using SVC_REQ 56.

Maximum of one active instruction

When SVC_REQ 57 is active, it does not support an interrupt that attempts to activate SVC_REQ 56 or a second instance of SVC_REQ 57.

Storage disabled conditions

By default, the following write operations disable SVC_REQ 57 until logic is written to nonvolatile storage:

- Run Mode Store (RMS), even if a second RMS reverts everything to the original state
- Test-Edit session, even when you cancel your edits
- Word-for-word change
- Downloading to RAM only of a stopped PACSystems CPU, even if the downloaded contents are equal to the contents already on the nonvolatile storage

Setting bit 0 of input word 4 (address + 4) to a value of 1 enables SVC_REQ 57 despite the above conditions.

Error checking

When writing to nonvolatile storage, error checking is provided to ensure that logic and the Hardware Configuration (HWC) in nonvolatile memory match the logic and HWC in PACSystems RAM.

Fragmentation

Due to the nature of the media in PACSystems CPUs, writes may produce fragmentation, which causes the loss of more space on a write than is actually required for the write. For instance, if you write 64 bytes, but there are fewer than 64 bytes remaining in the current memory sector, the data is written into a new sector. This occurs because data records are not allowed to span sectors, which means that there may be unused bytes at the end of full sectors. The response data (address + 8 and address + 9) to the write request will show that the amount of available memory is reduced by the amount of data lost in the old sector plus the 64 bytes of data plus the 8 bytes of command data.

When nonvolatile storage is full

When logic driven user nonvolatile storage is full, a fault is logged. Before you can use SVC_REQ 57 to write again, use one of the following solutions:

To retain the most up-to-date data and continue writing with SVC_REQ 57 to nonvolatile storage:

1. Stop the PACSystems.
2. Power cycle the PACSystems.

A power cycle *when nonvolatile storage is full* triggers a compaction of existing data. During compaction, multiple writes of the same reference memory address are removed, which leaves only the most recent data, and contiguous reference memory addresses are combined into the fewest number of records necessary.
If compaction cannot take place, a second fault is logged and you need to use one of the following two solutions.

To retain specific data from nonvolatile storage, clear nonvolatile storage, and then return the data to nonvolatile storage:

1. While the PACSystems is still running, use SVC_REQ 56 to read the desired values into PACSystems memory.
2. Upload the current values from PACSystems memory as initial values to your project.
3. Stop the PACSystems.
4. Do one of the following:
   - Clear the flash memory
     - or -
   - Write to flash. The flash is erased prior to writing, which frees up some space.
5. Download the initial values to the PACSystems.
6. Start the PACSystems.
7. Use SVC_REQ 57 to write the desired values from PACSystems memory to nonvolatile storage.

To write to flash to erase everything:

1. Stop the PACSystems.
2. Write to flash.
   The flash is erased prior to writing, which frees up some space.

Equality

Because data in nonvolatile storage is not considered part of the project, writing to nonvolatile storage does not impact equality between the CPU and Logic Developer - PLC.

Redundancy

Redundancy systems can benefit from the use of logic driven user nonvolatile storage as long as all of the references saved to nonvolatile storage are included in the transfer lists. Each redundancy CPU maintains its own separate logic driven user nonvolatile storage by means of SVC_REQ 57 during its logic scan. If the values of reference addresses to be stored to user nonvolatile storage are synchronized, the logic driven user nonvolatile storage data in each CPU is identical. If the values to be stored are not synchronized, then each CPU's user nonvolatile storage may be different.

ENO and power flow to the right

If the status is Success or Partial Read (see address + 6), then on the SVC_REQ instruction, ENO is set to True in FBD and ST, and power flow passes to the right in LD.

Input and Output

The parameter block has a length of 12 words (24 bytes). The first six words contain the input you supply for the write operation. The last six words contain the status of the write operation and storage numbers.

<table>
<tr><td>address</td><td>The number representing the memory area that contains the data to write to nonvolatile storage.
<table>
<tr><th>For this memory type:</th><th>Enter this decimal value:</th></tr>
<tr><td>%R</td><td>8</td></tr>
<tr><td>%AI</td><td>10</td></tr>
<tr><td>%AQ</td><td>12</td></tr>
<tr><td>%I (byte mode)</td><td>16</td></tr>
<tr><td>%Q (byte mode)</td><td>18</td></tr>
<tr><td>%T (byte mode)</td><td>20</td></tr>
<tr><td>%M (byte mode)</td><td>22</td></tr>
<tr><td>%G (byte mode)</td><td>56</td></tr>
<tr><td>%I (bit mode)</td><td>70</td></tr>
<tr><td>%Q (bit mode)</td><td>72</td></tr>
<tr><td>%T (bit mode)</td><td>74</td></tr>
<tr><td>%M (bit mode)</td><td>76</td></tr>
<tr><td>%G (bit mode)</td><td>86</td></tr>
<tr><td>%W</td><td>196</td></tr>
</table></td></tr>
<tr><td>address + 1 and address + 2</td><td>The zero-based offset N in the CPU memory area that contains the data to write to nonvolatile storage memory. Address + 1, the least significant word, contains the complete offset for any memory area except %W, which also requires the use of address + 2 for offsets greater than 65,535.
For %I, %Q, %M, %T, and %G memory in byte mode, N = (Ra - 1) / 8, where Ra = one-based reference address. For example, to write the data starting at the one-based bit reference address %T33, enter the byte offset 4: (33 - 1) / 8 = 4.
For %W, %R, %AI, and %AQ memory, and for %I, %Q, %M, %T, and %G memory in bit mode, N = Ra - 1. For example, to write to nonvolatile storage the data starting at the one-based reference address %R200 in CPU memory, enter the zero-based reference offset 199; to write to nonvolatile storage the data starting at CPU memory address %I73 in bit mode, enter offset 72. For memory in bit mode, the offset must be set on a byte boundary, that is, a number exactly divisible by 8: 0, 8, 16, 24, and so on.</td></tr>
</table>

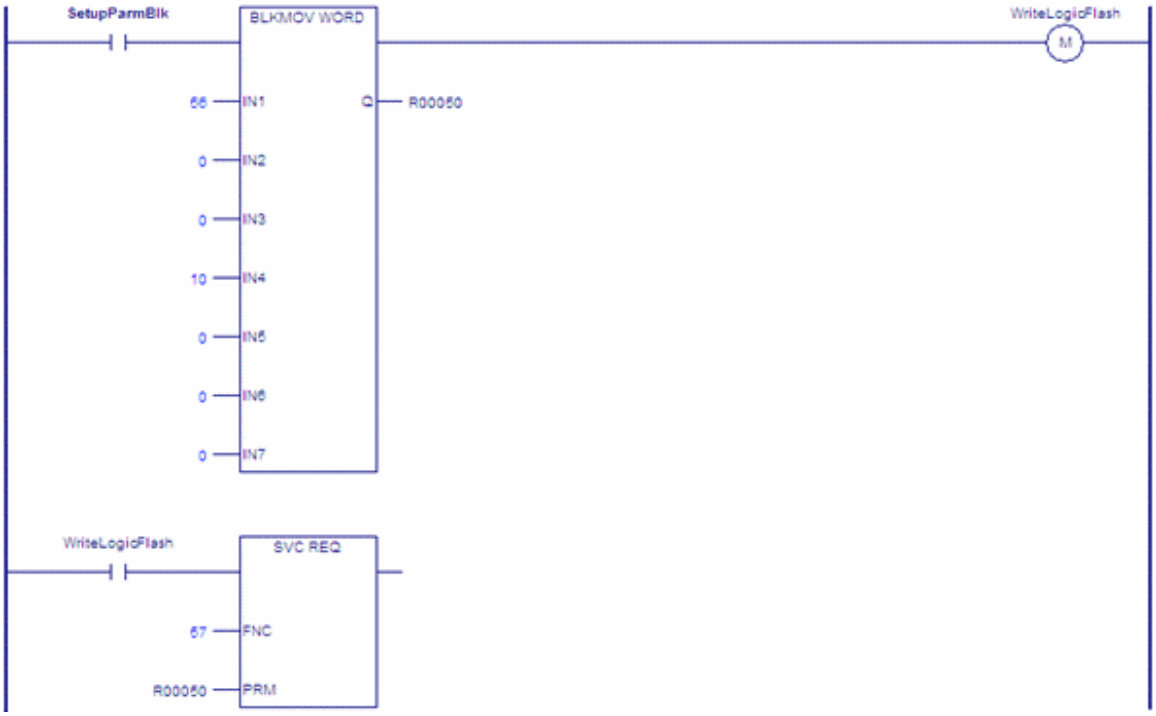
<table>
<tr><td>address + 3</td><td>Length. The number of items to write to nonvolatile storage beginning at the reference address calculated from the offset defined at [address + 1 and address + 2]. The length can be one of the following:
<table>
<tr><th>Description</th><th>Valid range</th></tr>
<tr><td>The number of words (16-bit registers) to write to %W, %R, %AI, or %AQ nonvolatile storage</td><td>1 through 32 words</td></tr>
<tr><td>The number of bytes to write to %I, %Q, %M, %T, or %G in byte mode nonvolatile storage</td><td>1 through 64 bytes</td></tr>
<tr><td>The number of bits to write to %I, %Q, %M, %T, or %G in bit mode nonvolatile storage</td><td>1 through 512 bits in increments of 8 bits</td></tr>
</table>
The value must reside in the low byte of address + 3. The high byte must be set to zero.</td></tr>
<tr><td>address + 4</td><td>▪ When bit 0 is set to 1, storage disabled conditions are ignored. A write is allowed even if the logic in RAM has changed since nonvolatile storage was read or written.
▪ Bits 1 through 15 must be set to zero; otherwise, the write fails.</td></tr>
<tr><td>address + 5</td><td>Reserved. Value must be set to zero.</td></tr>
<tr><td>address + 6</td><td>Response status.
▪ The low byte contains the major error code.
▪ The high byte contains the minor error code.
For example, 0001 means minor error code 00, major error code 01.
<table>
<tr><th>Minor</th><th>Major</th><th>Description</th></tr>
<tr><td>00</td><td>01</td><td>Success. All values requested were written.</td></tr>
<tr><td>01</td><td>01</td><td>Existing values found. All values requested are in storage, but one or more values were already stored.</td></tr>
<tr><td>01</td><td>02</td><td>Insufficient source memory. Counting from the offset, not enough reference addresses are left in the specified memory area.</td></tr>
<tr><td>02</td><td>02</td><td>Invalid length. The length requested was larger than 64 bytes or less than 1 byte or the number of bits is not divisible by 8.</td></tr>
<tr><td>03</td><td>02</td><td>Invalid source reference address. The memory area specified is not supported, the starting or ending offset is out of range, or the offset is not byte-aligned for discrete memory areas.</td></tr>
<tr><td>04</td><td>02</td><td>Invalid request. Spare bits or spare words in the parameter block are not set to zero.</td></tr>
</table></td></tr>
</table>

	01	**03**	Storage busy. A SVC_REQ 56 or another SVC_REQ 57 instruction is active. For example, an interrupt block is attempting to execute SVC_REQ 57 when the block it interrupted was executing SVC_REQ 57.
	01	**04**	Storage disabled. The logic in RAM differs from the logic stored in nonvolatile storage. See Storage disabled conditions.
	02	**04**	Storage closed. Either the storage has not been created or a previous corruption error or unexpected read/write failure closed the storage.
	01	**05**	Unexpected write failure. The command to the storage hardware failed unexpectedly.
	02	**05**	Corrupted storage. The write failed due to a bad checksum or corrupted storage header information.
	01	**06**	Write failed. Storage is full.
address + 7	Count of items written. Words, bytes, or bits.		
address + 8 and address + 9	The number of bytes available in nonvolatile storage.		
address + 10 and address + 11	Reserved.		

Examples

Example 1

The following LD logic writes ten continuous bytes to nonvolatile storage, ranging from %G1 through %G80. The value applied to IN1, 56, determines byte mode.



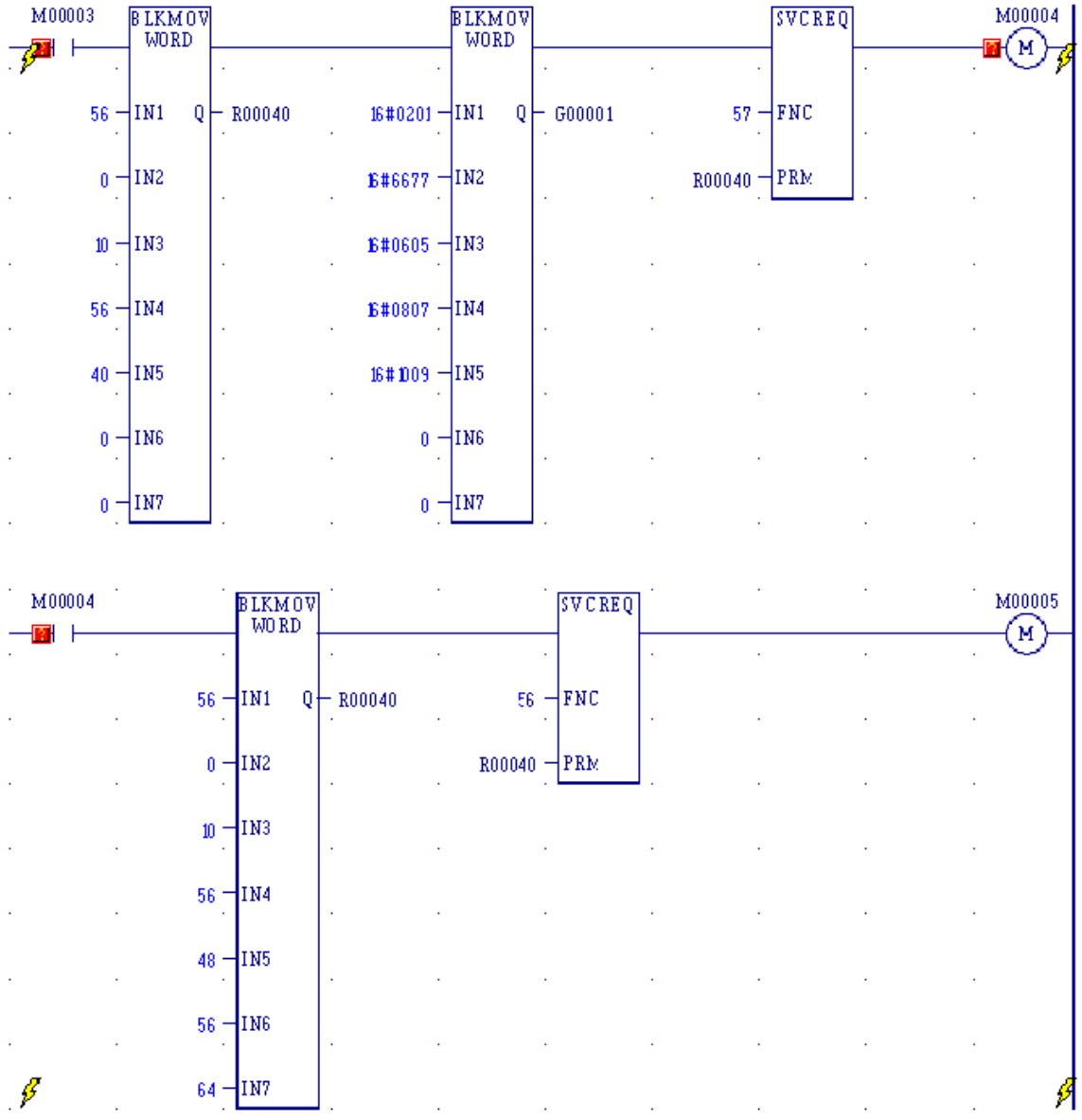
Parameter block for Example 1:

Address + offset	Address	Input Value	Definition
address + 0	%R00050	56	Data type = %G (byte mode)
address + 1	%R00051	0	Beginning address of data to write, low word
address + 2	%R00052	0	Beginning address of data to write, high word
address + 3	%R00053	10	Length = 10 bytes
address + 4	%R00054	0	Storage disabled conditions are enforced.
address + 5	%R00055	0	Reserved. Must be set to 0.
address + 6	%R00056	n/a	Response status. The low byte contains the major error code; the high byte contains the minor error code.
address + 7	%R00057	n/a	Count of bytes written.
address + 8	%R00058	n/a	The number of bytes available in nonvolatile storage.
address + 9	%R00059	n/a	
address + 10	%R00060	n/a	Reserved

address + 11	%R00061	n/a	Reserved

Example 2: Overwriting data in the same memory area

The following example illustrates the use of SVC_REQ 56 and SVC_REQ 57.
To overwrite some of the data in the same memory area, we can use the same LD logic with different values. Look at the status words returned for the below SVC_REQ execution starting from %G321.



Suspend I/O

Operation

The Suspend I/O (SUS_IO) instruction stops normal I/O scans from occurring for one CPU sweep. During the next output scan, all outputs are held at their current states. During the next input scan, the input references are not updated with data from inputs. However, during the input scan portion of the sweep, the CPU verifies that Genius bus Controllers have completed their previous output updates.

Note: The SUS_IO instruction suspends all I/O, both analog and discrete, whether integrated I/O, Genius I/O, or Ethernet Global Data. For details, refer to Chapter 4 in the *TCP/IP Ethernet Communications for the Series 90 PLC User's Manual*, GFK-1541.

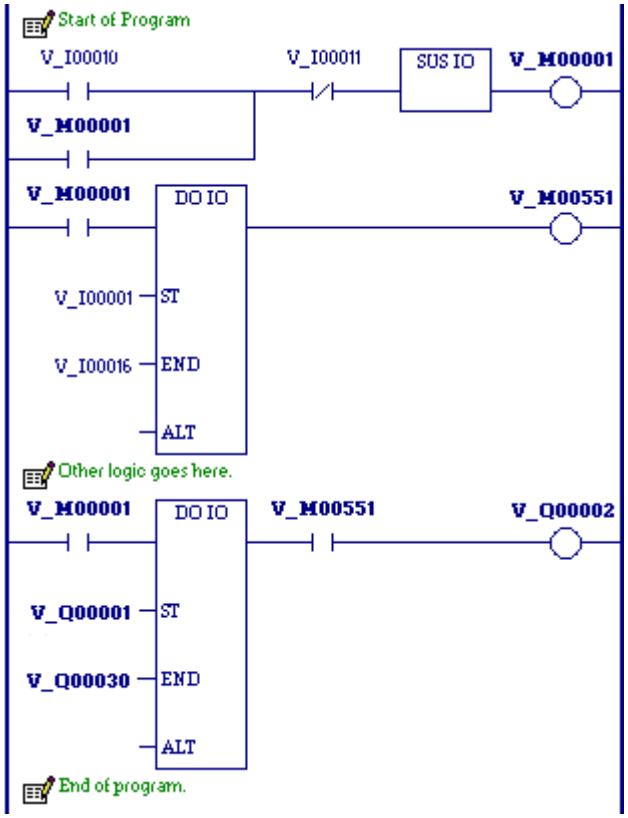
When SUS_IO receives power flow, all I/O servicing stops except that provided by DO_IO instructions.

Warning: If SUS_IO were placed at the left rail of the ladder, without enabling logic to regulate its execution, no regular I/O scan would ever be performed.

SUS_IO passes power flow to the right whenever it receives power.

Example

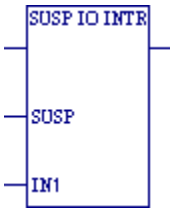
This example shows a SUS_IO instruction and a DO_IO instruction used to stop I/O scans, then cause certain I/O to be scanned from the logic.
Inputs %I00010 and %I00011 form a latch circuit with the contact from %M00001. This keeps the SUS_IO instruction active on each sweep until %I00011 goes on. If this input were not scanned by DO_IO after SUS_IO went active, SUS_IO could only be disabled by powering down the Controller.
Output %Q00002 is set when both DO_IO instructions execute successfully. The rung is constructed so that both DO_IO instructions execute even if one does not set its OK output. With normal I/O suspended, output %Q00002 is not updated until a DO_IO instruction with %Q00002 in its range executes. This does not occur until the sweep after the setting of %Q00002. Outputs that are set after a DO_IO instruction executes are not updated until another DO_IO instruction executes, typically in the next sweep. Because of this delay, most logic that uses SUS_IO and DO_IO places the SUS_IO instruction in the first rung of the logic, the DO_IO instruction that processes inputs in the next rung, and the DO_IO instruction that processes outputs in the last rung.
The range of the DO_IO instruction doing outputs is %Q00001 through %Q00030. If the module in this range were a 32-point module, the DO_IO instruction would actually perform a scan of the entire module. A DO_IO instruction will not break the scan in the middle of an I/O module.



CPU Support

SUS_IO is supported for PACSystems CPUs and Series 90-70 CPUs.

SUSP_IO_INTR



CPU Support

SUSP_IO_INTR is supported for:

PACSystems CPUs firmware version 3.50 and later only

Operation

Use SUSP_IO_INTR with I/O variables to suspend a set of I/O interrupts and cause occurrences of these interrupts to be queued until these interrupts are resumed.

Tip: When not using I/O variables, you can use SVC_REQ 32.

The set of I/O interrupts are those that can be generated from the PACSystems High Speed Counter. The number of I/O interrupts that can be queued depends on the I/O module's capabilities. The CPU informs the I/O module that its interrupts are to be suspended or resumed. The I/O module's default is resumed. The Suspend applies to all I/O interrupts associated with the I/O module. Interrupts are suspended and resumed within a single scan.

Successful execution occurs unless:

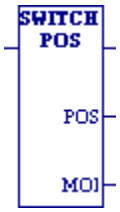
- The I/O module associated with the specified address is not an appropriate module for this operation.
- Communication between the CPU and this I/O module has failed. (The board is not present, or it has experienced a fatal fault.)

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
SUSP	BOOL variable or bit references in nondiscrete memory	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	If state of SUSP is set to ON, this indicates suspend. If state of SUSP is set to OFF, this indicates resume.
IN1	BOOL or WORD variable	I, Q M, T, G, R, P, L, AI, AQ, W, I/O variable	The value of IN1 indicates the interrupt trigger to be suspended or resumed.

Switch Position



Operation

Switch Position allows the logic to read the current position of the PACSystems switch, as well as the mode for which the switch is configured.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
POS	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Memory location at which to write current switch position value. 1 - Run I/O Enabled 2 - Run I/O Disabled 3 - Stop Mode
MOD	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Memory location to which switch configuration value is written. 0 - Switch configuration not supported 1 - Switch controls run/stop mode 2 - Switch not used, or is used by the user application 3 - Switch controls both memory protection and run/stop mode 4 - Switch controls memory protection

CPU Support

SWITCH_POS is supported for PACSystems CPUs with firmware version 2.00 or later.

Conversion Instructions

The Data Type Conversion instructions change a data item from one number format (data type) to another.

Note: Many instructions, such as math instructions, must be used with data of one type. As a result, a data conversion is often required before using those instructions.

Instruction	*Mnemonics*	*Description*
Convert Angles	DEG_TO_RAD DEG_TO_RAD_LREAL DEG_TO_RAD_REAL	Converts degrees to radians.
	RAD_TO_DEG RAD_TO_DEG_LREAL RAD_TO_DEG_REAL	Converts radians to degrees.
Convert BCD4 to INT	BCD4_TO_INT	Converts 4-digit Binary-Coded-Decimal (BCD4) to INT (16-bit signed integer).
Convert BCD4 to REAL	BCD4_TO_REAL	Converts BCD4 to REAL (32-bit signed floating-point).
Convert BCD4 to UINT	BCD4_TO_UINT	Converts BCD4 to UINT (16-bit unsigned integer).
Convert BCD8 to DINT	BCD8_TO_DINT	Converts 8-digit Binary-Coded-Decimal (BCD8) to DINT (32-bit signed integer).
Convert BCD8 to REAL	BCD8_TO_REAL	Converts 8-digit Binary-Coded-Decimal (BCD8) to REAL (32-bit signed floating-point).
Convert DINT to BCD8	DINT_TO_BCD8	Converts DINT (32-bit signed integer) to 8-digit Binary-Coded-Decimal (BCD8).
Convert DINT to INT	DINT_TO_INT	Converts DINT to INT (16-bit signed integer).
Convert DINT to LREAL	DINT_TO_LREAL	Converts DINT to LREAL (64-bit signed floating-point).
Convert DINT to REAL	DINT_TO_REAL	Converts DINT to REAL (32-bit signed floating-point).
Convert DINT to UINT	DINT_TO_UINT	Converts DINT to UINT (16-bit unsigned integer).

Convert INT to BCD4	INT_TO_BCD4	Converts INT (16-bit signed integer) to 4-digit Binary-Coded-Decimal (BCD4).
Convert INT to DINT	INT_TO_DINT	Converts INT to DINT (32-bit signed integer).
Convert INT to REAL	INT_TO_REAL	Converts INT to REAL (32-bit signed floating-point).
Convert INT to UINT	INT_TO_UINT	Converts INT to UINT (16-bit unsigned integer).
Convert LREAL to DINT	LREAL_TO_DINT	Converts LREAL (64-bit signed floating-point) to DINT (32-bit signed integer).
Convert LREAL to REAL	LREAL_TO_REAL	Converts LREAL to REAL (32-bit signed floating-point).
Convert REAL to DINT	REAL_TO_DINT	Converts REAL (32-bit signed floating-point) to DINT (32-bit signed integer).
Convert REAL to INT	REAL_TO_INT	Converts REAL to INT (16-bit signed integer).
Convert REAL to LREAL	REAL_TO_LREAL	Converts REAL to LREAL (64-bit signed floating-point)
Convert REAL to UINT	REAL_TO_UINT	Converts REAL to UINT (16-bit unsigned integer).
Convert REAL to WORD	REAL_TO_WORD	Converts REAL to WORD (16-bit bit string).
Convert UINT to BCD4	UINT_TO_BCD4	Converts UINT (16-bit unsigned integer) to 4-digit Binary-Coded-Decimal (BCD4).
Convert UINT to DINT	UINT_TO_DINT	Converts UINT to DINT (32-bit signed integer).
Convert UINT to INT	UINT_TO_INT	Converts UINT to INT (16-bit signed integer).
Convert UINT to REAL	UINT_TO_REAL	Converts UINT to REAL (32-bit signed floating-point).
Convert WORD to REAL	WORD_TO_REAL	Converts WORD (16-bit bit string) to REAL (32-bit signed floating-point).

Truncate	TRUNC_DINT	Truncate. Rounds a REAL (32-bit signed floating-point) number down to a DINT (32-bit signed integer) number.
	TRUNC_INT	Truncate. Rounds a REAL (32-bit signed floating-point) number down to an INT (16-bit signed integer) number.

Convert Angles

	**Mnemonics:** DEG_TO_RAD DEG_TO_RAD_LREAL DEG_TO_RAD_REAL
	**Mnemonics:** RAD_TO_DEG RAD_TO_DEG_LREAL RAD_TO_DEG_REAL

Operation

When the Degrees to Radians (DEG_TO_RAD or sibling mnemonics) or the Radians to Degrees (RAD_TO_DEG or sibling mnemonics) instruction receives power flow, it performs the appropriate angle conversion on the value in input IN and places the result in output Q. IN and Q must be of the same data type.
The angle conversions pass power flow to the right when they perform without overflow, unless IN is NaN (Not a Number).

Operands

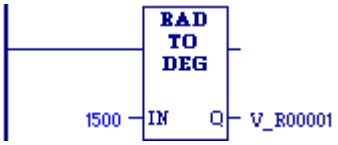
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL or LREAL variable or constant. Must be the same data type as for Q. **Note:** The DEG_TO_RAD and RAD_TO_DEG mnemonics support only REAL.	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to convert.
Q	REAL or LREAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The converted value.

Example

+1500 is converted to DEG. The result is placed in %R0001.



CPU Support

DEG_TO_RAD and RAD_TO_DEG are supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
DEG_TO_RAD_LREAL and RAD_TO_DEG_LREAL are supported for PACSystems firmware version 5.50 or later.
DEG_TO_RAD_REAL and RAD_TO_DEG_REAL are supported for all PACSystems. They have respectively replaced DEG_TO_RAD and RAD_TO_DEG for PACSystems, which respectively supports DEG_TO_RAD and RAD_TO_DEG for backward compatibility.

Convert BCD4 to INT

Operation

When BCD4_TO_INT receives power flow, it converts the input 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent single-precision signed integer (INT) value, which it outputs to Q. BCD4_TO_INT does not change the original BCD4 data.

Note: (PACSystems and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the data is out of range.

Operands

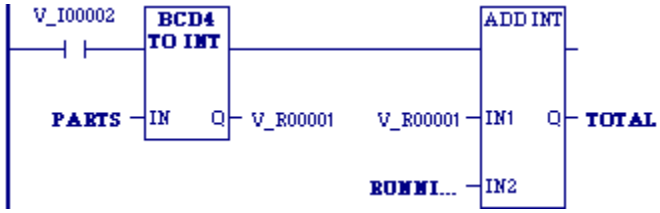
Note

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to INT.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT equivalent value of the original BCD4 value in IN.

Example

Whenever input %I0002 is set, the BCD-4 value in PARTS is converted to a signed integer (INT) and passed to the ADD_INT instruction, where it is added to the INT value represented by the reference RUNNING. The sum is output by ADD_INT to the reference TOTAL.



CPU Support

BCD4_TO_INT is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later CPUs, and Series 90-30 CPUs.

Convert BCD4 to REAL

Operation

When BCD4_TO_REAL receives power flow, it converts the 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent floating-point (REAL) value, which it outputs to Q. BCD4_TO_REAL does not change the original BCD4 data.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to REAL.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL equivalent value of the original BCD4 value in IN.

CPU Support

BCD4_TO_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

Convert BCD4 to UINT

Operation

When BCD4_TO_UINT receives power flow, it converts the input 4-digit Binary-Coded-Decimal (BCD4) data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q. BCD4_TO_UINT does not change the original BCD4 data.

Note: The output data can be used directly as input for another instruction, as in the example.

The instruction passes power flow when power is received, unless the resulting data is outside the range 0 to +65,535.

Tip: One use of BCD4_TO_UINT is to convert BCD data from the I/O structure into integer data and store it in memory. This can provide an interface to BCD thumbwheels or external BCD electronics, such as high-speed counters and position encoders.

Operands

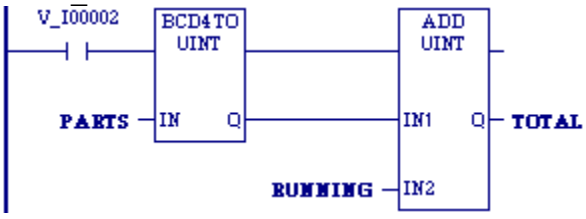
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 value to convert to UINT.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT equivalent value of the original BCD4 value in IN.

Example

Whenever input %I0002 is set, the BCD-4 value in PARTS is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT instruction, where it is added to the UINT value represented by the reference RUNNING. The sum is output by ADD_UINT to the reference TOTAL.



CPU Support

BCD4_TO_UINT is supported for PACSystems CPUs and for Series 90-70 Version 3.00 or later CPUs.

Convert BCD8 to DINT

Operation

When BCD8_TO_DINT receives power flow, it converts the input 8-digit Binary-Coded-Decimal (BCD8) data into the equivalent double-precision signed integer (DINT) value, which it outputs to Q. BCD8_TO_DINT does not change the original BCD8 data.

Note: The output data can be used directly as input for another instruction, as in the example.

The instruction passes power flow when power is received, unless the data is out of range.

Operands

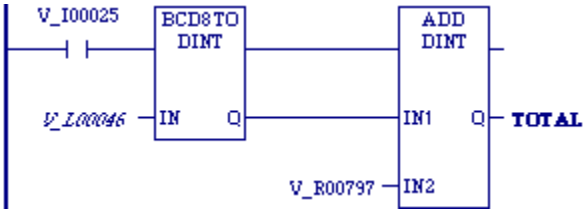
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The BCD8 value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The DINT equivalent value of the original BCD8 value in IN.

Example

Whenever input %I00025 is set, the BCD-8 value in %L00046 is converted to a signed double-precision integer (DINT) and passed to the ADD_DINT instruction, where it is added to the DINT value in %R00797. The sum is output by ADD_DINT to the reference TOTAL.



CPU Support

BCD8_TO_DINT is supported for PACSystems CPUs and for Series 90-70 Version 3.00 or later CPUs.

Convert BCD8 to REAL

Operation

When BCD8_TO_REAL receives power flow, it converts the input 8-digit Binary-Coded-Decimal (BCD8) data into the equivalent floating-point (REAL) value, which it outputs to Q. BCD8_TO_REAL does not change the original BCD8 data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Warning: Converting from BCD8 to REAL may result in the loss of significant digits. This is because a BCD8 is stored in a DWORD, which uses 32 bits to store a value, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The BCD8 value to convert to REAL.
Q	REAL variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The REAL equivalent value of the original BCD8 value in IN.

CPU Support

BCD8_TO_REAL is supported for PACSystems CPUs and for Series 90-70 Version 3.00 or later floating-point CPUs.

Convert DINT to BCD8

Operation

When DINT_TO_BCD8 receives power flow, it converts the input signed double-precision integer (DINT) data into the equivalent 8-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. DINT_TO_BCD8 does not change the original DINT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is outside the range 0 to 99,999,999.

Operands

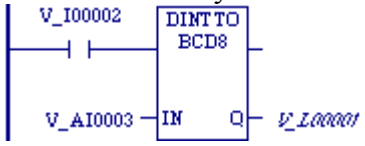
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT value to convert to BCD8.
Q	DWORD variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable.	The BCD8 equivalent value of the original DINT value in IN.

Example

Whenever input %I00002 is set and no errors exist, the double-precision signed integer (DINT) at input location %AI0003 is converted to eight BCD digits and the result is stored in memory locations %L00001 through %L00002.



CPU Support

DINT_TO_BCD8 is supported for PACSystems CPUs and Series 90-70 Version 3.00 or later CPUs.

Convert DINT to INT

Operation

When DINT_TO_INT receives power flow, if the input signed double-precision integer (DINT) data is within the range of the single-precision signed integer (INT) data type, DINT_TO_INT converts the input value to the equivalent INT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the INT range, respectively the maximum or minimum value of the INT range is output to Q and DINT_TO_INT does not pass power flow.
If DINT_TO_INT receives no power flow, it passes no power flow.
DINT_TO_INT does not change the DINT data in its original memory location.
The output data can be used directly as input for another instruction, as in the example.

Operands

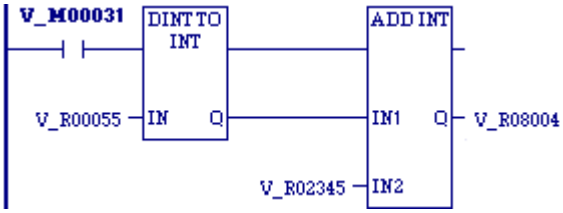
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT value to convert to INT
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT equivalent value of the original DINT value in IN

Example

Whenever input %M00031 is set, the DINT value in %R00055 is converted to a signed integer (INT) and passed to the ADD instruction, where it is added to the INT at %R02345. The sum is output by the ADD instruction to %R08004.



CPU Support

DINT_TO_INT is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later CPUs.

Convert DINT to LREAL

Operation

When DINT_TO_LREAL receives power flow, it converts the input signed double-precision integer (DINT) data into the equivalent double-precision floating-point (LREAL) value, which it outputs to Q. DINT_TO_LREAL does not change the original DINT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT value to convert to LREAL
Q	LREAL variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The LREAL equivalent value of the original DINT value in IN

CPU Support

DINT_TO_LREAL is supported for PACSystems firmware version 5.50 or later.

Convert DINT to REAL

Operation

When DINT_TO_REAL receives power flow, it converts the input signed double-precision integer (DINT) data into the equivalent floating-point (REAL) value, which it outputs to Q. DINT_TO_REAL does not change the original DINT data.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Warning: Converting from DINT to REAL may result in the loss of significant digits for numbers with more than 7 significant base-10 digits. This is because a DINT uses 32 bits to store a value, which is the equivalent of up to 10 significant base-10 digits, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa, which is the equivalent of 7 or 8 significant base-10 digits. When the REAL result is displayed as a base-10 number, it may have up to 10 digits, but these are converted from the rounded 24-bit mantissa, so that the last 2 or 3 digits may be inaccurate.

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT value to convert to REAL.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL equivalent value of the original DINT value in IN.

CPU Support

DINT_TO_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

Convert DINT to UINT

Operation

When DINT_TO_UINT receives power flow, if the input signed double-precision integer (DINT) data is within the range of the single-precision unsigned integer (UINT) data type, DINT_TO_UINT converts the input value to the equivalent UINT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the UINT range, respectively the maximum or minimum value of the UINT range is output to Q and DINT_TO_UINT does not pass power flow.
If DINT_TO_UINT receives no power flow, it passes no power flow.
DINT_TO_UINT does not change the DINT data in its original memory location.
The output data can be used directly as input for another instruction, as in the example.

Operands

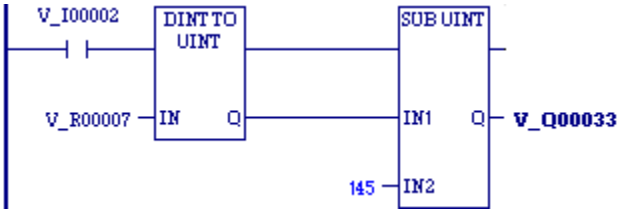
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT value to convert to UINT.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT equivalent value of the original DINT value in IN.

Example

Whenever input %I00002 is set and no errors exist, the double precision signed integer (DINT) at input location %R00007 is converted to an unsigned integer (UINT) and passed to the SUB instruction, where the constant value 145 is subtracted from it. The result of the subtraction is stored in the output reference location %Q00033.



CPU Support

DINT_TO_UINT is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later CPUs.

Convert INT to BCD4

Operation

When INT_TO_BCD4 receives power flow, it converts the input signed single-precision integer (INT) data into the equivalent 4-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. INT_TO_BCD4 does not change the original INT data.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is outside the range 0 through 9,999.

Tip: Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

Operands

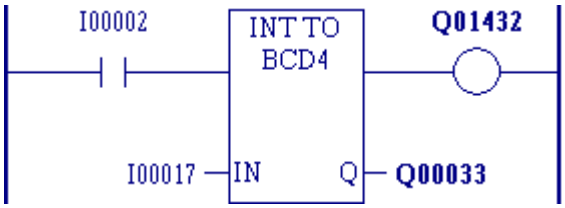
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to BCD4.
Q	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 equivalent value of the original INT value in IN.

Example

Whenever input %I0002 is set and no errors exist, the INT values at input locations %I0017 through %I0032 are converted to four BCD digits, and the result is stored in memory locations %Q0033 through %Q0048. Coil %Q1432 is used to check for successful conversion.



CPU Support

INT_TO_BCD4 is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 version 3.00 or later CPUs, and Series 90-30 CPUs.

Convert INT to DINT

Operation

When INT_TO_DINT receives power flow, it converts the input single-precision signed integer (INT) data into the equivalent double-precision signed integer (DINT) value, which it outputs to Q. INT_TO_DINT does not change the original INT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the data is out of range.

Operands

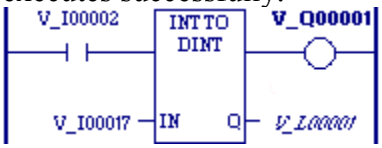
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT equivalent value of the original INT value in IN.

Example

Whenever input %I00002 is set, the signed single-precision integer (INT) value at input location %I00017 is converted to a double-precision signed integer (DINT) and the result is placed in location %L00001. The output %Q01001 is set whenever the instruction executes successfully.



CPU Support

INT_TO_DINT is supported for PACSystems CPUs and Series 90-70 Version 3.00 or later CPUs.

Convert INT to REAL

Operation

When INT_TO_REAL receives power flow, it converts the input single-precision integer (INT) data into the equivalent floating-point (REAL) value, which it outputs to Q. INT_TO_REAL does not change the original INT data.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Operands

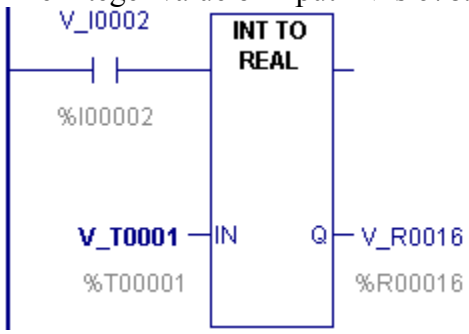
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to REAL.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL equivalent value of the original INT value in IN.

Example

The integer value of input IN is 678. The result value placed in %T0016 is 678.000.



CPU Support

INT_TO_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

Convert INT to UINT

Operation

When INT_TO_UINT receives power flow, if the input signed single-precision integer (INT) data is within the range of the single-precision unsigned integer (UINT) data type, INT_TO_UINT converts the input value to the equivalent UINT value, outputs it to Q, and passes power flow to the right.
If the input value underflows the UINT range, the minimum value of the UINT range is output to Q and INT_TO_UINT does not pass power flow.
If INT_TO_UINT receives no power flow, it passes no power flow.
INT_TO_UINT does not change the INT data in its original memory location.
The output data can be used directly as input for another instruction, as in the example.

Operands

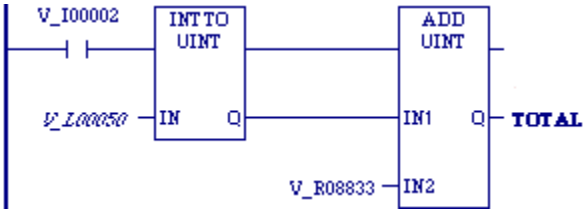
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT value to convert to UINT
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT equivalent value of the original DINT value in IN

Example

Whenever input %I0002 is set, the INT value in %L00050 is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT instruction, where it is added to the UINT value in %R08833. The sum is output by ADD_UINT to the reference TOTAL.



CPU Support

INT_TO_UINT is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later CPUs.

Convert LREAL to DINT

Operation

When LREAL_TO_DINT receives power flow, if the input signed double-precision floating-point (LREAL) data is within the range of the double-precision signed integer (DINT) data type, LREAL_TO_DINT rounds the input LREAL value to the nearest DINT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the DINT range, respectively the maximum or minimum value of the DINT range is output to Q and LREAL_TO_DINT does not pass power flow.
If LREAL_TO_DINT receives no power flow, it passes no power flow.
LREAL_TO_DINT does not change the LREAL data in its original memory location. The output data can be used directly as input for another instruction.

Operands

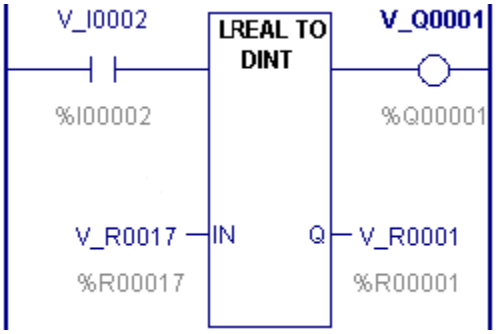
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The LREAL value to convert to DINT
Q	DINT variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The DINT equivalent value of the original LREAL value in IN

Example

Whenever input %I0002 is set, the LREAL value at input location %R0017 is converted to a double precision signed integer (DINT) and the result is placed in location %R0001. The output %Q1001 is set whenever the instruction executes successfully.



CPU Support

LREAL_TO_DINT is supported for PACSystems firmware version 5.50 or later.

Convert LREAL to REAL

Operation

When LREAL_TO_REAL receives power flow, if the input signed double-precision floating-point (LREAL) data is within the range of the single-precision signed floating-point (REAL) data type, LREAL_TO_REAL converts the input LREAL value to the equivalent REAL value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the REAL range, respectively the maximum or minimum value of the REAL range is output to Q and LREAL_TO_REAL does not pass power flow.
If LREAL_TO_REAL receives no power flow, it passes no power flow.
LREAL_TO_REAL does not change the LREAL data in its original memory location. The output data can be used directly as input for another instruction.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
IN	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The LREAL value to convert to REAL
Q	REAL variable		The REAL equivalent value of the original LREAL value in IN

CPU Support

LREAL_to_REAL is supported for PACSystems firmware version 5.50 or later.

Convert REAL to DINT

Operation

When REAL_TO_DINT receives power flow, if the input signed single-precision floating-point (REAL) data is within the range of the double-precision signed integer (DINT) data type, REAL_TO_DINT rounds the input REAL value to the nearest DINT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the DINT range, respectively the maximum or minimum value of the DINT range is output to Q and REAL_TO_DINT does not pass power flow.
If REAL_TO_DINT receives no power flow, it passes no power flow.
REAL_TO_DINT does not change the original REAL data in its memory location.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

Tip: To truncate a REAL value and express the result as a DINT, that is, to remove the fractional part of the REAL number and express the remaining integer value as a DINT, use TRUNC_DINT.

Operands

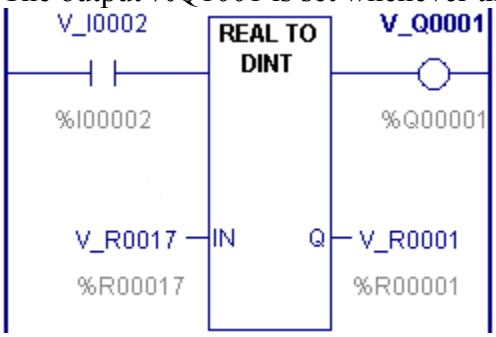
Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use BOOL arrays instead of other data types.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT equivalent value of the original REAL value in IN.

Example

Whenever input %I0002 is set, the REAL value at input location %R0017 is converted to a double precision signed integer (DINT) and the result is placed in location %R0001. The output %Q1001 is set whenever the instruction executes successfully.



CPU Support

REAL_TO_DINT is supported for PACSystems CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, Series 90-30 floating-point CPUs, and VersaMax CPUs.

Convert REAL to INT

Operation

When REAL_TO_INT receives power flow, if the input signed single-precision floating-point (REAL) data is within the range of the single-precision signed integer (INT) data type, REAL_TO_INT rounds the input value to the nearest INT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the INT range, respectively the maximum or minimum value of the INT range is output to Q and REAL_TO_INT does not pass power flow.
If REAL_TO_INT receives no power flow or the REAL data is NaN (Not a Number), it passes no power flow.
REAL_TO_INT does not change the REAL data in its original memory location.

Note: (PACSystems CPUs and Series 90-70 CPUs.) The output data can be used directly as input for another instruction.

Tip: To truncate a REAL value and express the result as an INT, that is, to remove the fractional part of the REAL number and express the remaining integer value as an INT, use TRUNC_INT.

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL value to convert to INT.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT equivalent value of the original REAL value in IN.

CPU Support

REAL_TO_INT is supported for PACSystems CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, Series 90-30 floating-point CPUs, and VersaMax CPUs.

Convert REAL to LREAL

Operation

When REAL_to_LREAL receives power flow, it converts the input single-precision floating-point (REAL) data into the equivalent double-precision floating-point (LREAL) value, which it outputs to Q. REAL_to_LREAL does not change the original REAL data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received.

Operands

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The REAL value to convert to LREAL
Q	LREAL variable		The LREAL equivalent value of the original REAL value in IN

CPU Support

REAL_to_LREAL is supported for PACSystems firmware version 5.50 or later.

Convert REAL to UINT

Operation

When REAL_TO_UINT receives power flow, if the input signed single-precision floating-point (REAL) data is within the range of the single-precision unsigned integer (UINT) data type, REAL_TO_UINT rounds the input value to the nearest UINT value, outputs it to Q, and passes power flow to the right.
If the input value overflows or underflows the UINT range, respectively the maximum or minimum value of the UINT range is output to Q and REAL_TO_UINT does not pass power flow.
If REAL_TO_UINT receives no power flow, it passes no power flow.
REAL_TO_UINT does not change the REAL data in its original memory location.
The output data can be used directly as input for another instruction, as in the example.

Operands

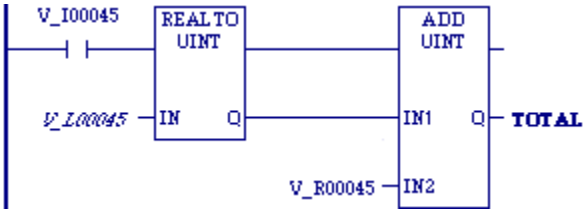
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL value to convert to UINT
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT equivalent value of the original REAL value in IN

Example

Whenever input %I00045 is set, the REAL value in %L00045 is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT instruction, where it is added to the UINT value in %R00045. The sum is output by ADD_UINT to the reference TOTAL.



CPU Support

REAL_TO_UINT is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later floating-point CPUs.

Convert REAL to WORD

Operation

When REAL_TO_WORD receives power flow, it rounds a positive input REAL data up or down to the nearest unsigned single-precision integer value, which it outputs to the WORD variable in Q. Any value larger than 65,535 is output as 65,535. If the input REAL data is negative, REAL_TO_WORD outputs the value 0 to the WORD variable in Q. REAL_TO_WORD does not change the original REAL data.
The instruction passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to FFFFh.

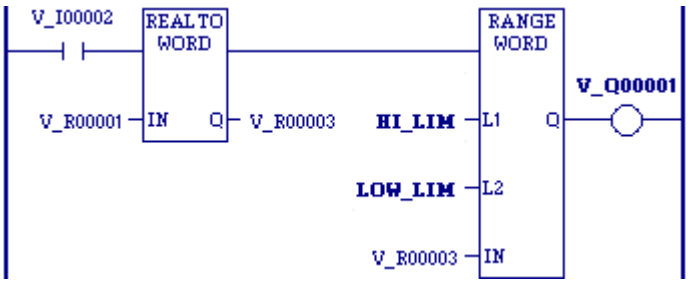
Warning: Converting from REAL to WORD may result in overflow. For example, REAL 6.8E18, which equals $6.8 * 10^{18}$, converts to WORD OVERFLOW.

Operands

Note: You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	R, AI, AQ, W	The REAL value to convert to WORD.
Q	WORD variable	I, Q, M, T, G, R, AI, AQ, W	The WORD equivalent value of the original REAL value in IN. $0 \leq Q \leq 65,535$.

Example



CPU Support

REAL_TO_WORD is supported for VersaMax CPUs and Series 90-30 floating-point CPUs.

Convert UINT to BCD4

Operation

When UINT_TO_BCD4 receives power flow, it converts the input unsigned single-precision integer (UINT) data into the equivalent 4-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. UINT_TO_BCD4 does not change the original UINT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is outside the range 0 to 9,999.

Tip: Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

Operands

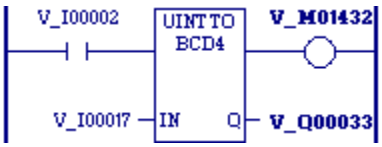
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to BCD4
Q	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BCD4 equivalent value of the original UINT value in IN

Example

Whenever input %I00002 is set and no errors exist, the UINT at input location %I00017 through %I00032 is converted to four BCD digits and the result is stored in memory locations %Q00033 through %Q00048. Coil %M01432 is used to check for successful conversion.



CPU Support

UINT_TO_BCD4 is supported for PACSystems CPUs and for Series 90-70 version 3.00 or later CPUs.

Convert UINT to DINT

Operation

When UINT_TO_DINT receives power flow, it converts the input unsigned single-precision integer (UINT) data into the equivalent signed double-precision integer (DINT) value, which it outputs to Q. UINT_TO_DINT does not change the original UINT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the data is out of range.

Operands

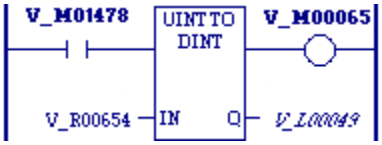
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to DINT.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The DINT equivalent value of the original UINT value in IN.

Example

Whenever input %M01478 is set, the unsigned single-precision integer (UINT) value at input location %R00654 is converted to a double-precision signed integer (DINT) and the result is placed in location %L00049. The output %M00065 is set whenever the instruction executes successfully.



CPU Support

UINT_TO_DINT is supported for PACSystems CPUs and Series 90-70 Version 3.00 or later CPUs.

Convert UINT to INT

Operation

When UINT_TO_INT receives power flow, if the input unsigned single-precision integer (UINT) data is within the range of the single-precision signed integer (INT) data type, UINT_TO_INT converts the input UINT value to the equivalent INT value, outputs it to Q, and passes power flow to the right.
If the input value overflows the INT range, the maximum value of the INT range is output to Q and UINT_TO_INT does not pass power flow.
If UINT_TO_INT receives no power flow, it passes no power flow.
UINT_TO_INT does not change the UINT data in its original memory location.
The output data can be used directly as input for another instruction, as in the example.

Operands

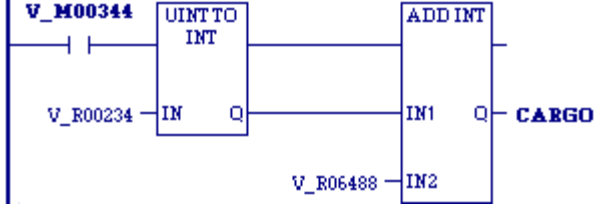
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to INT
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The INT equivalent value of the original UINT value in IN

Example

Whenever input %M00344 is set, the UINT value in %R00234 is converted to a signed integer (INT) and passed to the ADD instruction, where it is added to the INT value in %R06488. The sum is output by the ADD instruction to the reference CARGO.



CPU Support

UINT_TO_INT is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later CPUs.

Convert UINT to REAL

Operation

When UINT_TO_REAL receives power flow, it converts the input unsigned single-precision integer (UINT) data into the equivalent floating-point (REAL) value, which it outputs to Q. UINT_TO_REAL does not change the original UINT data.

Note: The output data can be used directly as input for another instruction.

The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Operands

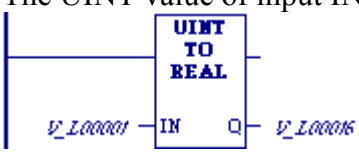
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The UINT value to convert to REAL
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL equivalent value of the original UINT value in IN

Example

The UINT value of input IN is 825. The result value placed in %L00016 is 825.000.



CPU Support

UINT_TO_REAL is supported for PACSystems CPUs and Series 90-70 Version 3.00 or later floating-point CPUs.

Convert WORD to REAL

Operation

When WORD_TO_REAL receives power flow, it converts the input WORD data into the equivalent REAL value, which it outputs to Q. WORD_TO_REAL does not change the original WORD data.
The instruction passes power flow when power is received, unless the conversion would result in a value that is out of range.

Operands

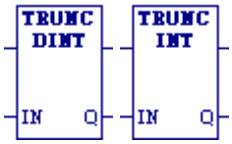
Note: You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	WORD variable or constant	I, Q, M, T, G, R, AI, AQ, W	The WORD value to convert to REAL.
Q	REAL variable	R, AI, AQ, W	The REAL equivalent value of the original WORD value in IN.

CPU Support

WORD_TO_REAL is supported for Series 90-30 floating-point CPUs and VersaMax CPUs.

Truncate



Operation

When power is received, the Truncate instructions TRUNC_DINT and TRUNC_INT round a floating-point (REAL) value down respectively to the nearest signed double-precision signed integer (DINT) or signed single-precision integer (INT) value. TRUNC_DINT and TRUNC_INT output the converted value to Q. The original data is not changed.

Note: (PACSystems CPUs and Series 90-70 CPUs) The output data can be used directly as input for another instruction.

TRUNC_DINT and TRUNC_INT pass power flow when power is received, unless the specified conversion would result in a value that is out of range or unless IN is NaN (Not a Number).

Operands

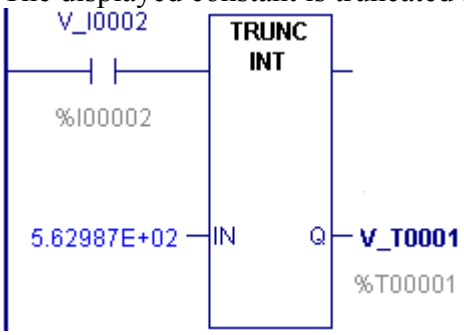
Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The REAL value whose copy is to be converted and truncated. The original is left intact.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The truncated value of the original REAL value in IN.
	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

The displayed constant is truncated and the integer result 562 is placed in %T0001.



CPU Support

TRUNC_DINT and TRUNC_INT are supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

Counter Built-in Function Blocks

Built-in function block data required for counters

The data associated with the counter built-in function blocks is retentive through power cycles.

Built-in Function Block	*Mnemonic*	*Description*
Down Counter	DNCTR	Counts down from a preset value. The output is ON whenever the Current Value is ≤ 0.
Up Counter	UPCTR	Counts up to a designated value. The output is ON whenever the Current Value is ≥ the Preset Value.

Built-in Function Block Instance Data Required for Counters

Each counter built-in function block instance uses a one-dimensional, three-word array of %R, %P, %L, or %W memory to store the following information:

Current Value (CV) Word 1

Preset Value (PV) Word 2

Control word Word 3

When you enter a counter, you must enter a beginning address for the three-word array (block of registers), as the ???? operand.

Warning: Do not use two consecutive registers as the starting addresses of two counters. Logic Developer - PLC does not check or warn you if register blocks overlap. Counters will not work if you place the current value of a second counter on top of the preset value for the previous counter.

Word 1: Current value (CV)

Warning: Be very careful if you write to word 1 (CV), as there is a risk that the built-in function block instance may not work properly.

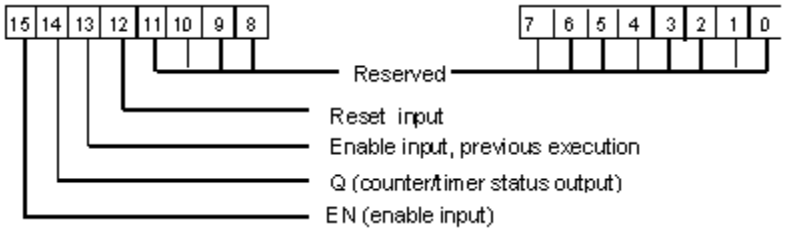
Word 2: Preset value (PV)

When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 of the counter's instance data.

- If you use a different address and you change word 2 directly, your change will have no effect, as PV will overwrite word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the counter is running and the change will be effective.

Word 3: Control word

The control word stores the state of the boolean inputs and outputs of its associated counter, as shown in the following diagram:

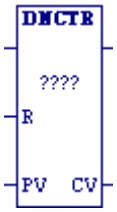


Notes

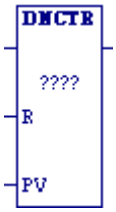
- Bits 0 through 11 are not used for counters.
- (Series 90-70) Bits 12 and 13 are not used for counters.
- When using the Bit Test, Bit Set, Bit Clear, or Bit Position instruction, the bits are numbered 1 through 16, *not* 0 through 15 as shown above.

Warning: The third word (Control) can be read but should not be written to; otherwise, the built-in function block instance will not work.

Down Counter



PACSystems and Series 90-70 only



Other CPUs

Operation

A Down Counter (DNCTR) built-in function block instance counts down from a preset value. The minimum Preset Value (PV) is zero; the maximum PV is +32,767 counts. When the Current Value (CV) reaches the minimum value, -32,768, it stays there until reset. When DNCTR is reset, CV is set to PV. When the power flow input transitions from OFF to ON, CV is decremented by one. The output is ON whenever CV ≤ 0.

Notes

- You must perform an initial reset to enable a DNCTR instance to operate properly. If the DNCTR instance is not initially reset, CV decrements from zero, and the output of the DNCTR instance is immediately set to ON.
- For any CPU, the value of CV is stored in Word 1 of the ???? operand. In PACSystems CPUs and Series 90-70 CPUs, the value of CV is also written to the CV operand.

The output state (Q) of DNCTR is retentive on power failure; no automatic initialization occurs at power-up.

Operands

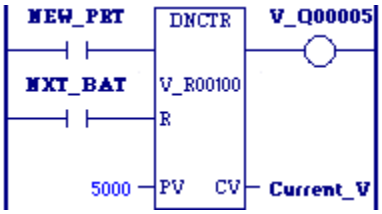
Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	▪ Word 1: Current Value (CV) ▪ Word 2: Preset Value (PV) ▪ Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic counter operation.
R	Power flow		When R receives power flow, it resets the counter's CV to PV.

PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) Preset Value to copy into word 2 of the counter's address when the counter is enabled or reset. 0 ≤PV ≤32,767. If PV is out of range, word 2 cannot be reset. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE_INT instruction or an operator interface.
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(PACSystems and Series 90-70 only; optional.) The current value of the counter; the same value as Word 1 in the ???? operand.

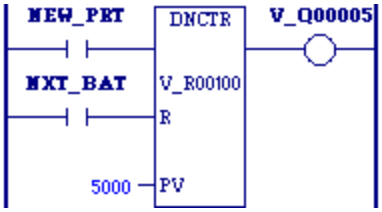
Example 1

The DNCTR built-in function block instance counts 5000 new parts before energizing output %Q00005.

Logic for PACSystems CPUs and Series 90-70:



Logic for other CPUs:



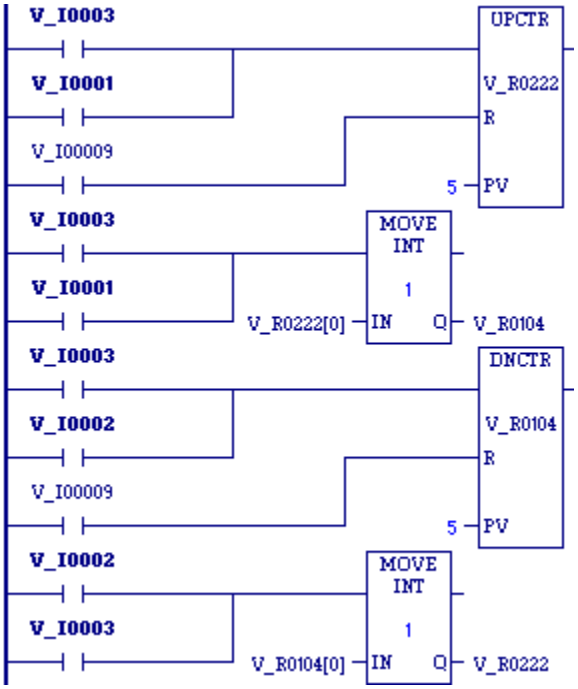
Example 2: CPUs other than PACSystems and Series 90-70

To keep track of the number of parts in a temporary storage area, you can use a down counter and up counter with a shared register for the accumulated or current value. When a part leaves the storage area, the DNCTR built-in function block instance decreases the current value of the parts in storage by 1. When a part enters the storage area, the up counter (UPCTR) increments by 1, increasing the inventory storage value by 1. To avoid conflict with the shared register, the counters use different register addresses:

Counter	*Three Word Array*	*Current Value Array Element*
DNCTR	V_R0104	V_R0104[0]

UPCTR	V_R0222	V_R0222[0]

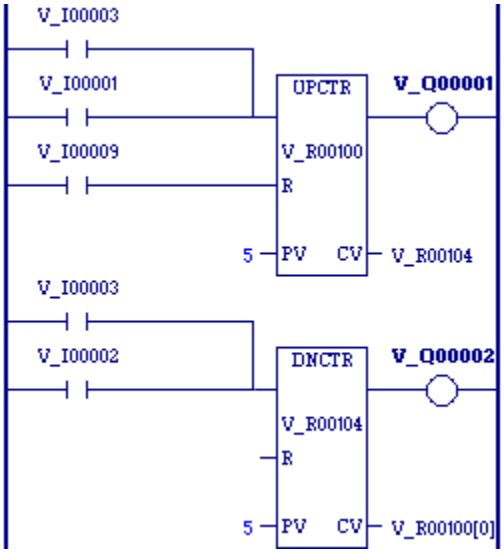
When a register counts, its current value must be moved to the current value register of the other counter.



Example 3: PACSystems CPUs and Series 90-70 CPUs

This example is similar to example 2, but takes advantage of the CV output operand, which is unavailable when running counters on other CPUs.

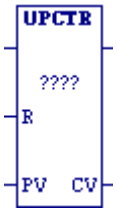
This example uses an up/down counter pair with a shared register for the accumulated or current value. When the parts enter the storage area, the up counter increments by 1, increasing the current value of the parts in storage by a value of 1. When a part leaves the storage area, the down counter decrements by 1, decreasing the inventory storage value by 1. To avoid conflict with the shared register, both counters use different register addresses but each has a current value (CV) address that is the same as the accumulated value for the other register.



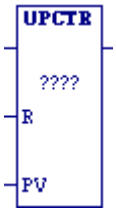
CPU Support

The DNCTR built-in function block instance is supported for all GE IP CPUs.

Up Counter



PACSystems and Series 90-70 only



Other CPUs

Operation

An Up Counter (UPCTR) built-in function block instance counts up to the Preset Value (PV). The range is 0 through +32,767 counts. When the Current Value (CV) of the counter reaches 32,767, it remains there until reset. When the UPCTR built-in function block reset is ON, CV resets to 0. Each time the power flow input transitions from OFF to ON, CV increments by 1. CV can be incremented past the Preset Value (PV). The output is ON whenever CV ≥ PV. The output (Q) stays ON until the R input receives power flow to reset CV to zero.

Note: For any CPU, the value of CV is stored in Word 1 of the ???? operand. In PACSystems CPUs and Series 90-70 CPUs, the value of CV is also written to the CV operand.

The state of an UPCTR instance is retentive on power failure; no automatic initialization occurs at powerup.

Operands

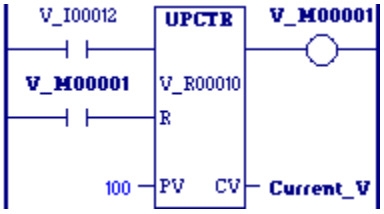
Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	▪ Word 1: Current Value (CV) ▪ Word 2: Preset Value (PV) ▪ Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic counter operation.
R	Power flow		When R is ON, it resets the counter's CV to 0.
PV	INT variable or constant; BOOL array of length 16 or	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W,	(Optional.) Preset Value to copy into word 2 of the counter's address when the counter is enabled or reset. 0 ≤ PV

	more (restrictions apply)	symbolic, I/O variable	≤ 32,767. If PV is out of range, it does not affect word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE_INT instruction or an operator interface.
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(PACSystems and Series 90-70 only; optional.) The current value of the counter; the same value as Word 1 in the ???? operand.

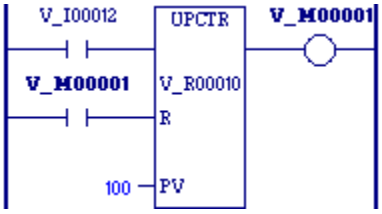
Example

Every time input %I0012 transitions from OFF to ON, the Up Counter counts up by 1; internal coil %M0001 is energized whenever 100 parts have been counted. Whenever %M0001 is ON, the accumulated count is reset to zero.

Logic for PACSystems CPUs or Series 90-70 CPUs:



Logic for other CPUs:



For an example illustrating the use of the UPCTR built-in function block instance in conjunction with DNCTR, see Down Counter.

CPU Support

The UPCTR built-in function block is supported for all GE IP CPUs.

Data Move Instructions

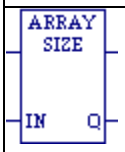
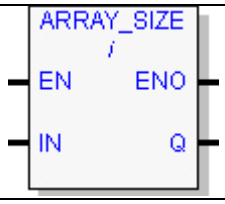
Move instructions provide move/copy capabilities.

Instruction	*Mnemonics*	*Description*
Array Size	ARRAY_SIZE	Counts the number of elements in an array
Array Size, Dimension 1	ARRAY_SIZE_DIM1	Returns the value of the Array Dimension 1 property of an array
Array Size, Dimension 2	ARRAY_SIZE_DIM2	Returns the value of the Array Dimension 2 property of a two-dimensional array
Block Clear	BLK_CLR_WORD	Replaces all the contents of a block of data with zeros. Can be used to clear an area of discrete (%I, %Q, %M, %G, %T, or symbolic) or non-discrete (%R, %P, %L, %AI, %AQ, %W, or symbolic) memory.
Block Move	BLKMOV_DINT BLKMOV_DWORD BLKMOV_INT BLKMOV_REAL BLKMOV_UINT BLKMOV_WORD	Copies a block of seven constants to a specified memory location. The constants are input as part of the instruction.
BUS Read	BUS_RD_BYTE BUS_RD_DWORD BUS_RD_WORD	Reads data from the backplane of a bus.
BUS Read Modify Write	BUS_RMW_BYTE BUS_RMW_DWORD BUS_RMW_WORD	Updates a data element using the read/modify/write cycle on a bus.
BUS Test and Set	BUS_TS_BYTE BUS_TS_WORD	Handles semaphores on a bus.
BUS Write	BUS_WRT_BYTE BUS_WRT_DWORD BUS_WRT_WORD	Writes data to the backplane of a bus.
Communication Request	COMM_REQ	Allows the logic to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.
Data Initialization	DATA_INIT_DINT DATA_INIT_DWORD DATA_INIT_INT	Copies a block of constant data to a reference range. The mnemonic specifies the data type.

	DATA_INIT_LREAL DATA_INIT_REAL DATA_INIT_UINT DATA_INIT_WORD	
Data Initialize ASCII	DATA_INIT_ASCII	Copies a block of constant ASCII text to a reference range.
Data Initialize Communications Request	DATA_INIT_COMM	Initializes a COMMREQ instruction with a block of constant data. The length should equal the size of the COMMREQ instruction's entire command logic.
Data initialize DLAN	DATA_INIT_DLAN	Configures a block of memory for use by a DLAN module.
Move	MOVE_BOOL MOVE_DATA MOVE_DINT MOVE_DWORD MOVE_INT MOVE_LREAL MOVE_REAL MOVE_UINT MOVE_WORD	Copies data as individual bits, so the new location does not have to be of the same data type. Data can be moved into a different data type without prior conversion.
MOVE_DATA_EX	MOVE_DATA_EX	Copies data with the same data types without prior conversion and with optional data coherency
MOVE_FROM_FLAT	MOVE_FROM_FLAT	Copies flat non-symbolic memory data into symbolic User-defined Data Types (UDTs) across mismatching data types with optional data coherency
MOVE_TO_FLAT	MOVE_TO_FLAT	Copies elements of symbolic User-defined Data Types (UDTs) to flat non-symbolic memory across mismatching data types with optional data coherency
Shift Register	SHFR_BIT SHFR_DWORD SHFR_WORD	Shifts one or more data bits, data WORDs or data DWORDs from a reference location into a specified area of memory. Data already in the area is shifted out.
Size of	SIZE_OF	Counts the number of bits used by a variable of any data type except a BYTE array in non-discrete memory

		or a double-segment structure variable
Swap	SWAP_DWORD SWAP_WORD	Swaps two BYTEs of data within a WORD or two WORDs within a DWORD.
VME Configuration Read	VME_CFG_READ	Reads the configuration for a VME module.
VME Configuration Write	VME_CFG_WRITE	Writes the configuration to a VME module.
VME Read	VME_RD_BYTE VME_RD_WORD	Reads data from the VME backplane.
VME Read Modify Write	VME_RMW_BYTE VME_RMW_WORD	Updates a data element using the read/modify/write cycle on the VME bus.
VME Test and Set	VME_TS_BYTE VME_TS_WORD	Handles semaphores on the VME bus.
VME Write	VME_WRT_BYTE VME_WRT_WORD	Writes data to the VME backplane.

Array Size

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE(In := [input]);

Operation

ARRAY_SIZE counts the number of elements in the array assigned to input IN.

Output

- In LD and FBD, ARRAY_SIZE writes the number to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 1.
- In an array of n structure variables, the value n is written to Q: the elements in each structure variable are not counted.

Tip: If the array assigned to input IN of ARRAY_SIZE is passed to a parameterized C block for processing, also pass the value of output Q to the block. In the C block logic, use the value of output Q to ensure you process all the array elements without exceeding the end of the array. For a two-dimensional array, this method works only if all elements are treated identically, for example, all are initialized to the same value.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE executes. When set to Off, ARRAY_SIZE does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE solves. When set to Off, ARRAY_SIZE does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic,	Array whose elements are counted

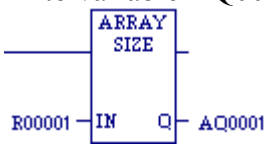
		I/O variable	

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Number of elements in the array assigned to input IN. In ST, the value of Q is assigned to the variable on the left side of the assignment operator.

Example

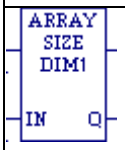
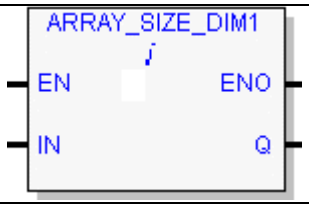
The two-dimensional array R00001 has its Array Dimension 1 property set to 4 and its Array Dimension 2 property set to 3. ARRAY_SIZE calculates 4 * 3 and writes the value 12 to variable AQ0001.



CPU Support

ARRAY_SIZE is supported for all PACSystems CPUs.

Array Size Dimension 1

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE_DIM1(In := [input]);

Operation

ARRAY_SIZE_DIM1 returns the value of the Array Dimension 1 property of an array.
Output

- In LD and FBD, ARRAY_SIZE_DIM1 writes the value to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 0.

You can use the value to ensure that a loop using a variable index to access array elements does not exceed the array's first dimension.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE_DIM1 executes. When set to Off, ARRAY_SIZE_DIM1 does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE_DIM1 solves. When set to Off, ARRAY_SIZE_DIM1 does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O	The array whose Array Dimension 1 property value is returned in the Q output (LD, FBD) or assigned to the variable on the left side of the assignment statement

		variable	(ST)

Output Operands

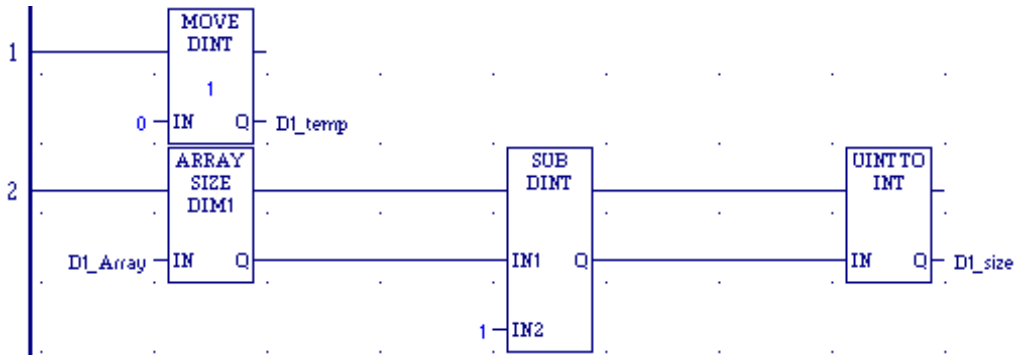
Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE_DIM1 has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value of the Array Dimension 1 property of the array assigned to input IN. The value is set to 0 if a non-array is assigned to IN. In ST, the value is assigned to the variable on the left side of the assignment operator. **Note:** Because the index of the first element of an array is zero, the index of the last element is one less than the value assigned to Q.

Examples

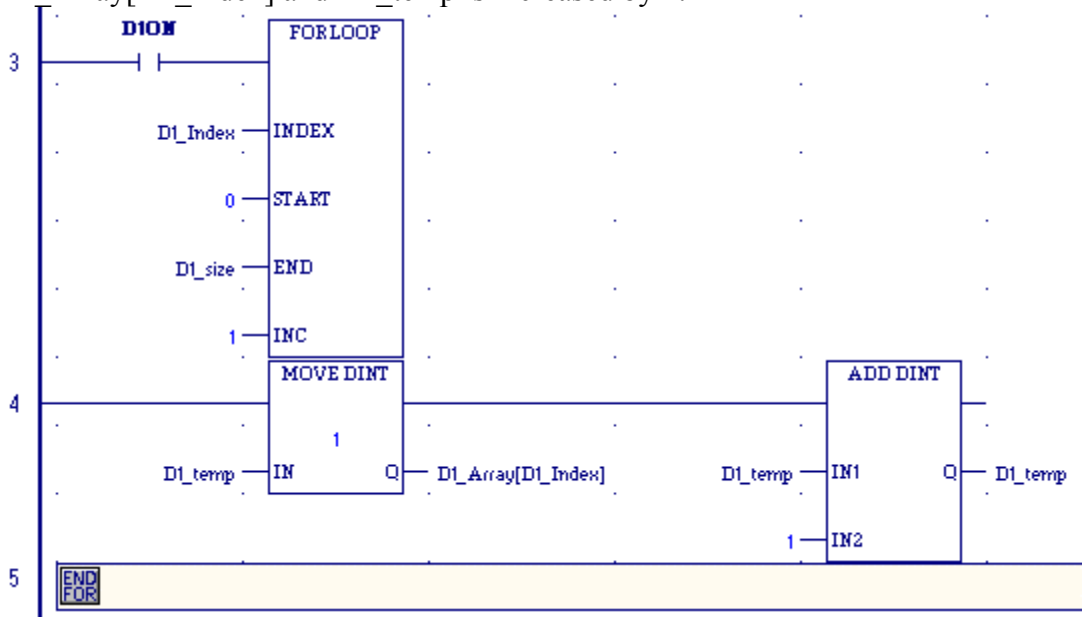
FOR_LOOP in LD logic

If you set up a FOR_LOOP that accesses array elements by means of a variable index, you must ensure that the FOR_LOOP does not iterate beyond the last element of the array.

In the following logic, MOVE_DINT initializes the variable D1_temp to 0. ARRAY_SIZE_DIM1 counts the number of elements of a one-dimensional array named D1_Array and outputs the result to output Q. Because the index of the first element of an array is zero, the loop must iterate (Q - 1) times. SUB_DINT performs the subtraction and the result is converted to an INT value and assigned to variable D1_size.



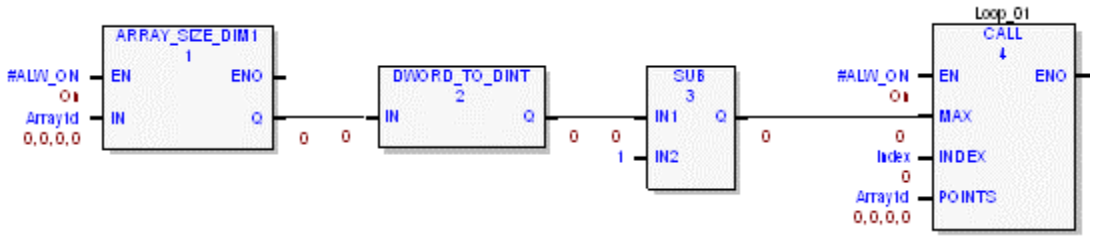
In the diagram below, the FOR_LOOP executes when D1ON is set to On. The variable index (D1_Index, see INDEX input) increments by 1 (see INC input) from 0 (see START input) through D1_size, the value calculated by ARRAY_SIZE_DIM1 and SUB_DINT (see END input). In each loop, the value of D1_temp is assigned to the element D1_Array[D1_Index] and D1_temp is increased by 1.



Loop initialization in FBD logic

FBD is not designed for loops that are completed in one scan. You can, however, use FBD to set up the parameters required for a loop and pass them to an LD or ST block that performs the loop.

In the following example, ARRAY_SIZE_DIM1 counts the number of elements of a one-dimensional array named Array1d and the value is converted from DWORD to DINT. Because an array element index is zero-based, the loop must iterate 1 less time than the number of elements. SUB performs the subtraction and passes the result to the MAX input of an LD or ST parameterized block named Loop_01. A value of 0 (zero) is assigned to the INDEX input and the array named Array1d is passed to the POINTS parameter.



In the non-displayed logic of Loop_01, a loop iterates by means of a variable index through all the elements of the POINTS formal parameter. The latter is set up as an 8-DWORD array, whereas the Array1d array passed to POINTS is a 4-DWORD array. Because the value assigned to MAX is 3, the loop iterates only four times, from 0 through 3, thus not exceeding the number of elements of the Array1d array used to populate the POINTS parameter. The Loop_01 block can thus be used to process the points of a 4-point or 8-point analog module.

Loop in ST logic

The following ST logic uses a For loop to initialize all elements of a one-dimensional REAL array, Array1d, to 0.0.
Dim1Size and i are INT variables. Array_Size_Dim1 is used to count the number of elements of Array1d. Because the lowest possible index of an array element is zero, the for loop iterates from 0 through a value 1 less than the number of elements.

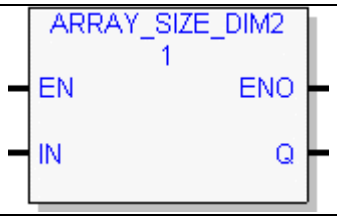
```
Dim1Size := Array_Size_Dim1(Array1d);
for i := 0 to (Dim1Size - 1) do
     Array1d[i] := 0.0
end_for;
```

If the above logic is used in an ST parameterized block or user-defined function block, any one-dimensional REAL array can be passed to it and have its elements initialized to 0.0.

CPU Support

ARRAY_SIZE_DIM1 is supported for all PACSystems CPUs.

Array Size Dimension 2

LD	FBD	ST
		Formal convention: Q := ARRAY_SIZE_DIM2(In := [input]);

Operation

ARRAY_SIZE_DIM2 returns the value of the Array Dimension 2 property of a two-dimensional array.

Output

- In LD and FBD, ARRAY_SIZE_DIM2 writes the value to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.
- If a non-array variable is assigned to input IN, the value of Q is 0.

Using output Q as the upper limit of variable index loops in LD and ST

In an LD or ST block that is not a parameterized block or a User Defined Function Block (UDFB), you can use the output Q value to ensure that a loop using a variable index to access array elements does not exceed the array's second dimension.
Examples:

- LD logic: FOR_LOOP that iterates through an array's second dimension
- LD logic: FOR_LOOP that iterates through both dimensions of an array
- ST logic: FOR_LOOP that iterates through both dimensions of an array

Workaround to process two-dimensional arrays in parameterized blocks and UDFBs

Parameterized blocks and UDFBs do not support two-dimensional array parameters. All two-dimensional arrays passed into a parameterized block or UDFB are converted to flat, one-dimensional arrays. The total number of elements in the two-dimensional array input should equal the number of elements in the one-dimensional array parameter. If a two-dimensional array is passed into a parameterized block or UDFB, the ARRAY_SIZE_DIM1 and ARRAY_SIZE_DIM2 values should also be passed to determine which array element is being modified. The following example shows how a two-dimensional array is represented in one-dimensional format:

Two-dimensional array [3,2]	One-dimensional array [6]
Array[0,0]	Array[0]
Array[0,1]	Array[1]
Array[1,0]	Array[2]

Array[1,1]	Array[3]
Array[2,0]	Array[4]
Array[2,1]	Array[5]

The code to determine the one-dimensional parameter array index from the array indexes and array dimension 2 value of the two-dimensional array is

```
Flat_Index = (Dim1 * Array_Dim2) + Dim2
```

where

- Flat_Index is the index of the one-dimensional array element
- Dim1 is the one-dimensional index of the two-dimensional array element
- Array_Dim2 is the value of the Array Dimension 2 property of the two-dimensional array
- Dim2 is the two-dimensional index of the two-dimensional array element

Example: Passing a two-dimensional array from LD logic to an ST UDFB

No one-scan loop support in FBD

FBD is not designed for loops that are completed in one scan. You can use FBD to set up the loop parameters and pass them to an LD or ST block that performs the loop.

First dimension

To ensure that a variable index does not exceed the first dimension, ARRAY_SIZE_DIM1 is required.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, ARRAY_SIZE_DIM2 executes. When set to Off, ARRAY_SIZE_DIM2 does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, ARRAY_SIZE_DIM2 solves. When set to Off, ARRAY_SIZE_DIM2 does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Array of any data type	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ,	The array whose Array Dimension 2 property value is returned in the Q output (LD, FBD) or assigned to the variable on

		W, symbolic, I/O variable	the left side of the assignment statement (ST)

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when ARRAY_SIZE_DIM2 has executed successfully.
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value of the Array Dimension 2 property of the array assigned to the IN input. The value is set to 0 if a non-array is assigned to IN. In ST, the value is assigned to the variable on the left side of the assignment operator. **Note:** Because the index of the first element of an array is zero, the index of the last element is one less than the value assigned to Q.

Examples

- LD logic: FOR_LOOP that iterates through an array's second dimension
- LD logic: FOR_LOOP that iterates through both dimensions of an array
- ST logic: FOR_LOOP that iterates through both dimensions of an array
- Passing a two-dimensional array from LD logic to an ST UDFB

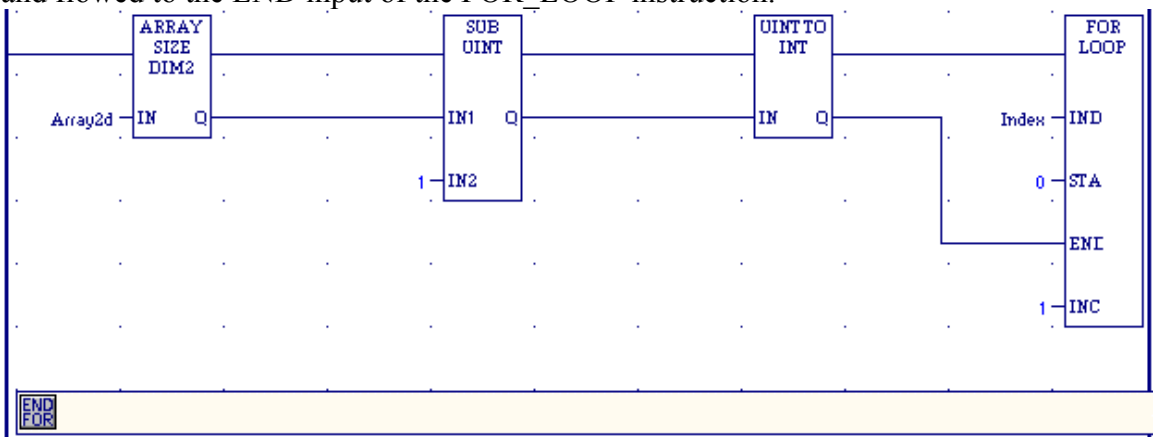
LD logic: FOR_LOOP that iterates through an array's second dimension

In an LD block that is not a parameterized block or a User Defined Function Block (UDFB), if you set up a FOR_LOOP that accesses elements in the second dimension of an array by means of a variable index, you must ensure that the FOR_LOOP does not exceed the array's second dimension.

Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

In the following logic, ARRAY_SIZE_DIM2 counts the number of elements of a two-dimensional array named Array2d and outputs the result to output Q. Because the index

of the first element of any dimension of an array is zero, the loop must iterate (Q - 1) times. SUB_UINT performs the subtraction and the result is converted to an INT value and flowed to the END input of the FOR_LOOP instruction.



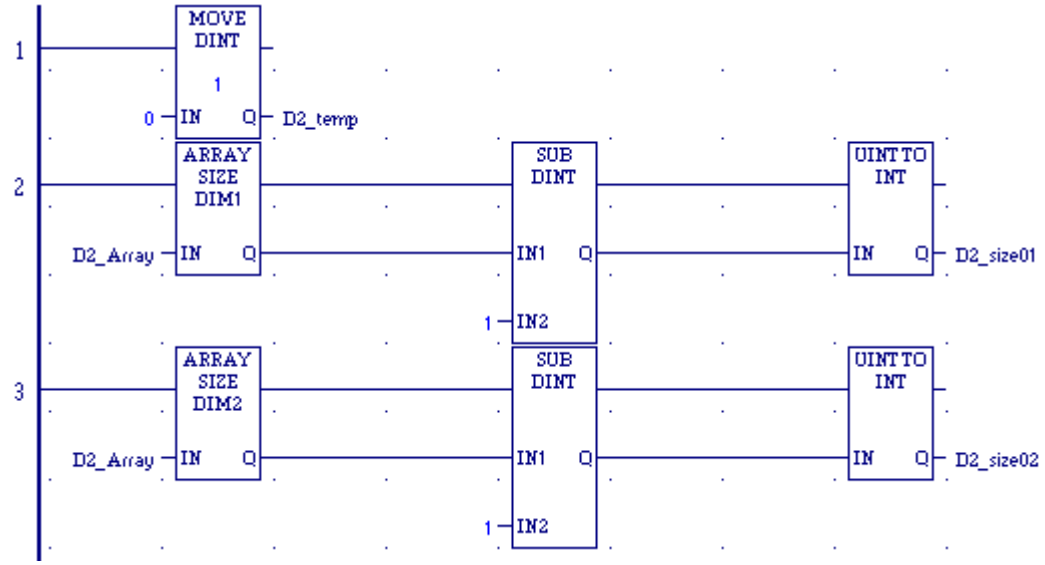
In the logic appearing between the FOR_LOOP and END_FOR instructions, various operations can take place with, for example, Array2d(1,Index), where Index increments by 1 (see INC input) from 0 (see START input) through the value calculated by ARRAY_SIZE_DIM2 and SUB_DINT (see END input).

LD logic: FOR_LOOP that iterates through both dimensions of an array

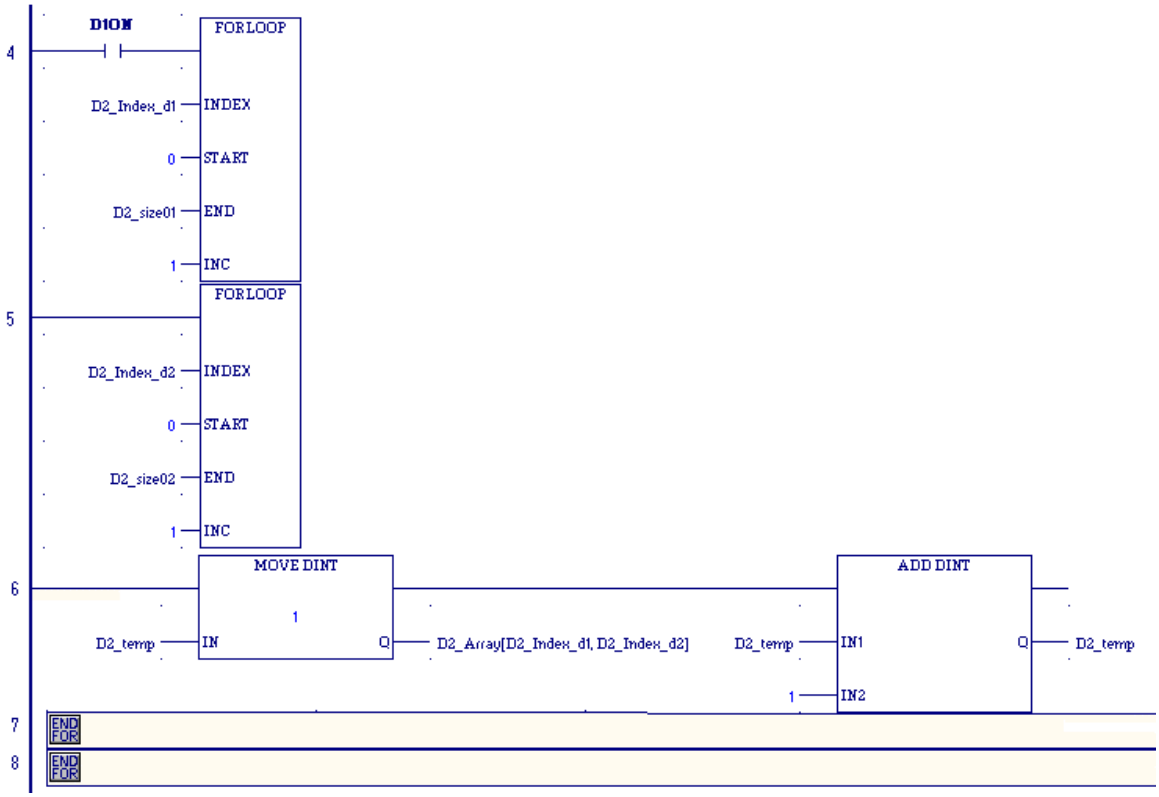
In an LD or ST block that is not a parameterized block or a User Defined Function Block (UDFB), you can use ARRAY_SIZE_DIM1, ARRAY_SIZE_DIM2, and nested FOR_LOOPs to ensure that operations on elements using two variable indexes each do not exceed either array dimension.

Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

In the following logic, MOVE_DINT initializes the variable D2_temp to 0.
ARRAY_SIZE_DIM1 counts the number of elements in the first dimension of a two-dimensional array named D2_Array. The value is placed in output Q. Because the index of the first element of an array is zero, the loop must iterate (Q - 1) times. SUB_DINT performs the subtraction and the result is converted to an INT value and assigned to variable D2_size01.
Likewise, ARRAY_SIZE_DIM2 counts the number of elements in the second dimension. After the subtraction and conversion, the value is assigned to variable D2_size02.



In the diagram below, the first FOR_LOOP executes when D1ON is set to ON. The variable index (D2_Index_d1, see INDEX input) increments by 1 (see INC input) from 0 (see START input) through D2_size01, the value calculated by ARRAY_SIZE_DIM1 and SUB_DINT (see END input). Within the first loop, the second FOR_LOOP executes. The variable index is D2_Index_d2, which increments by 1 from 0 through D2_size02, the value calculated by ARRAY_SIZE_DIM2 and SUB_DINT. Each time the second FOR_LOOP executes, the value of D2_temp is assigned to the element D2_Array[D2_Index_d1,D2_Index_d2] and D2_temp is incremented by 1.



ST logic: FOR_LOOP that iterates through both dimensions of an array

In an ST block that is not a parameterized block or a User Defined Function Block (UDFB), the following ST logic uses nested for loops to initialize all the elements of a two-dimensional REAL array, Array2d, to 0.0.

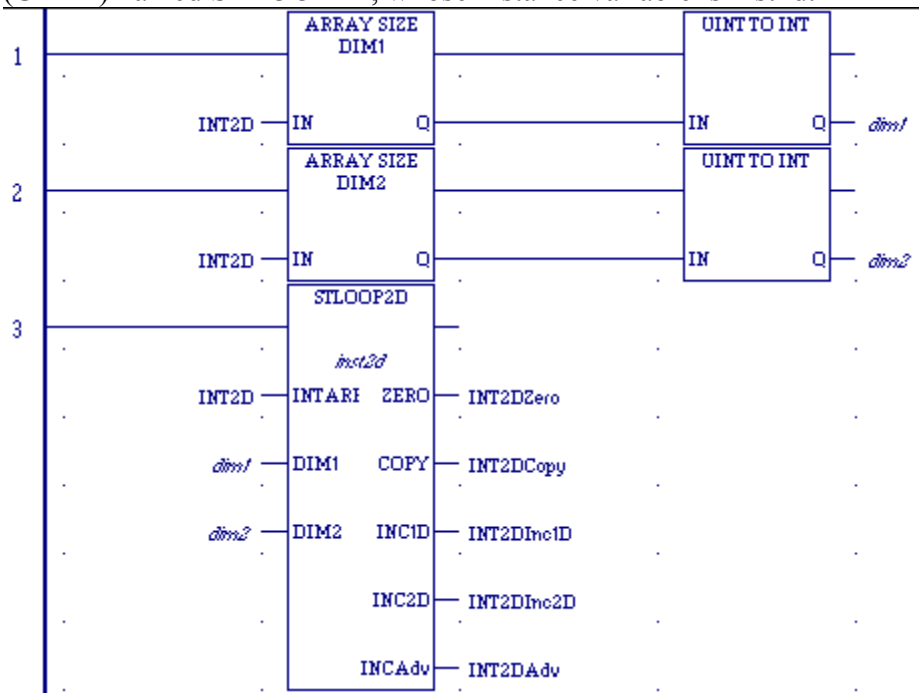
Note: Parameterized blocks and User Defined Function Blocks (UDFBs) do not support two-dimensional array parameters. See Workaround to process two-dimensional arrays in parameterized blocks and UDFBs.

Dim1Size, Dim2Size, i, and j are INT variables. Array_Size_Dim1 and Array_Size_Dim2 count the number of elements respectively in the first and second dimensions of Array2d. Because the lowest possible index of an array element is zero, the for loops iterate from 0 through a value 1 less than the number of elements in the respective dimension.

```
Dim1Size := Array_Size_Dim1(Array2d);
Dim2Size := Array_Size_Dim2(Array2d);
for i := 0 to (Dim1Size - 1) do
     for j := 0 to (Dim2Size - 1) do
          Array2d[i,j] := 0.0;
     end_for;
end_for;
```

Passing a two-dimensional array from LD logic to an ST UDFB

In the following LD logic, the two-dimensional array INT2D and its Array Dimension 1 and Array Dimension 2 values are passed to the ST User Defined Function Block (UDFB) named STLOOP2D, whose instance variable is inst2d.



INT2D is passed to the parameter INTARR, which is a one-dimensional array. One of the output parameters of STLOOP2D, INC1D, is a one-dimensional array; the other four are two-dimensional arrays. All five outputs are assigned to two-dimensional arrays for the calling LD logic to use.

In the following ST logic of UDFB STLOOP2D, five different operations on one-dimensional arrays are performed in such a way that the one-dimensional elements are

correctly mapped to two-dimensional elements of the two-dimensional arrays assigned to the LD Call to STLOOP2D.

```
'Zero a 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        ZERO[loopcount] := 0;
    end_for;
end_for;

'Copy the 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        COPY[loopcount] := INTARR[loopcount];
    end_for;
end_for;

'Incrementing a 1D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INC1D[loopcount] := loopcount;
    end_for;
end_for;

'Incrementing a 2D array
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INC2D[loopcount] := j;
    end_for;
end_for;

'2D array value = DIM1 * 1000 + DIM2
'For example, 0, 1, 2, ... j, 1000, 1001, 1002, ...
for i := 0 to (DIM1 - 1) do
    for j := 0 to (DIM2 - 1) do
        loopcount := (i*(DIM2) + j);
        INCAdv[loopcount] := i*1000 + j;
    end_for;
end_for;
```

CPU Support

ARRAY_SIZE_DIM2 is supported for all PACSystems CPUs.

Block Clear

Operation

When the Block Clear (BLKCLR_WORD) instruction receives power flow, it fills the specified block of data with zeros, beginning at the reference specified by IN. When the data to be cleared is from BOOL (discrete) memory (%I, %Q, %M, %G, or %T), the transition information associated with the references is also cleared. BLKCLR_WORD passes power to the right whenever it receives power.

Note: The input parameter IN is not included in coil checking.

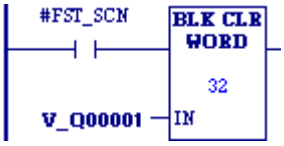
Operands

Note: (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of words to clear, starting at the IN location. 1 ≤ Length ≤ 256 words. Default: 1.
IN	WORD variable	I, Q, M, T, G, SA, SB, SC, R, P, L, W, AI, AQ, symbolic, I/O variable **Note:** %S cannot be used.	The first WORD of the memory block to clear to 0.

Example

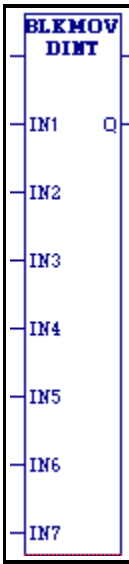
At power-up, 32 words of %Q memory (512 points) beginning at %Q0001 are filled with zeros. The transition information associated with these references will also be cleared.



CPU Support

BLKCLR_WORD is supported for all GE IP CPUs.

Block Move



Mnemonics:
BLKMOV_DINT
BLKMOV_DWORD
BLKMOV_INT
BLKMOV_REAL
BLKMOV_UINT
BLKMOV_WORD

Operation

When the Block Move (BLKMOV) instruction receives power flow, it copies a block of seven constants into consecutive locations beginning at the destination specified in output Q. BLKMOV passes power to the right whenever it receives power.

Note: (Series 90-70.) Output Q cannot be the input of another instruction.

Operands

Notes

- For each mnemonic, use the corresponding data type for the Q operand. For example, BLKMOV_DINT requires Q to be a DINT variable.
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

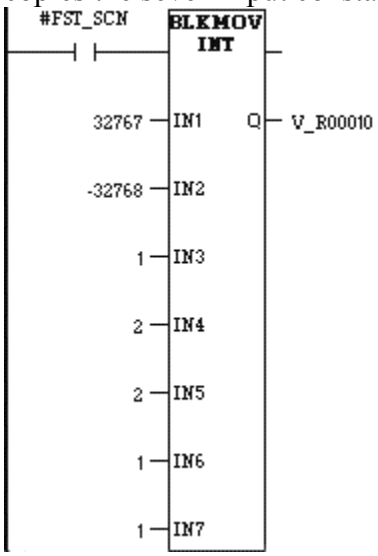
Operand	*Data Type*	*Memory Area*	*Description*
IN1 to IN7	DINT or DWORD constants		The seven constant values to move.
	INT constants		The seven constant values to move. Displayed as signed decimals (signed values in base 10).

	REAL constants		The seven constant values to move.
	UINT constants		The seven constant values to move.
	WORD constants		The seven constant values to move. **Note:** Logicmaster displays these values in hexadecimal format, but when you import a Logicmaster project to Logic Developer - PLC, these values are converted to the decimal format. To display them as hexadecimal constants in Logic Developer - PLC, see Inserting Constants in Logic.
Q	DINT variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, data flow, symbolic, I/O variable.	The first memory location of the destination for the moved values. IN1 is moved to Q.
	DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, data flow, symbolic, I/O variable.	
	INT variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ. PACSystems also supports data flow, symbolic, I/O variable.	
	REAL variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, data flow, symbolic, I/O variable.	
	UINT variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ. PACSystems also supports data flow, symbolic, I/O variable.	

	WORD variable	I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ. PACSystems also supports data flow, symbolic, I/O variable.	

Example

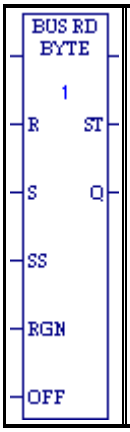
When the enabling input represented by the name #FST_SCN is ON, BLKMOV_INT copies the seven input constants into memory locations %R0010 through %R0016.



CPU Support

BLKMOV_INT and BLKMOV_WORD are supported for all GE IP CPUs.
BLKMOV_DINT and BLKMOV_DWORD are supported for PACSystems CPUs, Series 90-70 CPUs, and 20-pt, 40-pt, and 64-pt VersaMax Micro CPUs with firmware version 3.60 or later, except for IC200UDR064 and IC200UDR164.
BLKMOV_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.
BLKMOV_REAL is supported for PACSystems CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, Series 90-30 floating-point CPUs, and VersaMax CPUs.

Bus Read



Mnemonics:
BUS_RD_BYTE
BUS_RD_DWORD
BUS_RD_WORD

Operation

The Bus Read (BUS_RD) instruction reads data from the bus.

Note: Using a Bus instruction (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the board. This information may be obtained from one of two sources:

- The board vendor may issue application notes on the correct use of the board.
- For VME boards, you can also refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_RD receives power flow, it accesses the module on the bus. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from. BUS_RD then copies data with the length LEN to Controller locations beginning at output Q. BUS_RD passes power to the right when its operation is successful.

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) A BUS_RD instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

Operands

Notes

- For each mnemonic, use the corresponding data type for the Q operand. For example, BUS_RD_BYTE requires Q to be a BYTE variable.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

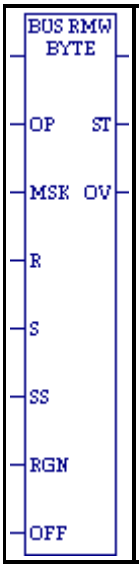
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length. The number of BYTEs, DWORDs, or WORDs. 1 ≤ Length ≤ 32,767. Default: 1.

R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is found by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
ST	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation
Q	BYTE, DWORD, or WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The data read from the module

CPU Support

BUS_RD_BYTE, BUS_RD_DWORD, and BUS_RD_WORD are supported for PACSystems CPUs.

Bus Read Modify Write



Mnemonics:
BUS_RMW_BYTE
BUS_RMW_DWORD
BUS_RMW_WORD

Operation

The Bus Read/Modify/Write (BUS_RMW) instruction updates a data element on the bus.

Notes

- BUS_RMW_WORD works only for even absolute addresses (absolute address = base address + offset).
- BUS_RMW_DWORD works only for absolute addresses that are a multiple of 4.
- Using a Bus instruction (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the board. This information may be obtained from one of two sources:
  - The board vendor may issue application notes on the correct use of the board.
  - For VME boards, you can also refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_RMW receives power flow, it accesses the module on the bus. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from.

- **Read phase:** BUS_RMW reads the byte, word, or dword of data from the bus at the address.
- **Modify phase:** This byte, word, or dword of data is combined (AND/OR) with the data mask MSK. Selection of AND or OR is made using the OP input. MSK is a word value. If byte data is operated on, only the lower eight bits of MSK are used. If word data is operated on, only the lower 16 bits of MSK are used.
- **Write phase:** The result is written back to the same bus address from which it was read.

BUS_RMW passes power flow to the right whenever it receives power, unless an error occurs.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

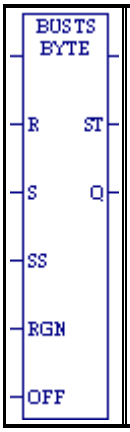
Operand	*Data Type*	*Memory Area*	*Description*
OP	Constant		▪ A value of 0 ANDs the data with the MSK data. ▪ A value of 1 ORs the data with the MSK data. ▪ No other value is valid.
MSK	DWORD variable or constant	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	The data mask. **Note:** For BUS_RMW_BYTE, only the lower 8 bits are used; for BUS_RMW_WORD, only the lower 16 bits are used.
R	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI,	The offset in the memory region. The exact memory location is found by adding the offset to the memory

		AQ, W, symbolic, I/O variable	region's base address, which is defined on the module's Memory tab.
ST	WORD variable	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation. **Note:** If BUS_RMW_WORD does not contain an even absolute address, or, if BUS_RMW_DWORD does not contain an absolute address that is a multiple of 4, the instruction returns a status code of seven, and will not pass power.
OV	BYTE, DWORD, or WORD variable, depending on the mnemonic you are using	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The original value that was read from the module.

CPU Support

BUS_RMW_BYTE, BUS_RMW_DWORD, and BUS_RMW_WORD are supported for PACSystems CPUs.

Bus Test and Set



Mnemonics:
BUS_TS_BYTE
BUS_TS_WORD

Operation

The Bus Test and Set (BUS_TS) instruction handles semaphores on the bus. BUS_TS exchanges a Boolean ON (1) for the value currently at the semaphore location. If that value was already ON, then BUS_TS does not obtain the semaphore. If the existing value was OFF, then the semaphore is reset and BUS_TS instruction has the semaphore and the use of the memory area it controls. The semaphore is cleared using the BUS_WRT instruction to write a 0 to the semaphore location.

Notes

- BUS_TS_WORD works only for even absolute addresses (absolute address = base address + offset).
- Using a Bus instruction (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the board. This information may be obtained from one of two sources:
  - The board vendor may issue application notes on the correct use of the board.
  - For VME boards, you can also refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_TS receives power flow, it accesses the module on the bus. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from. BUS_TS then exchanges a Boolean ON with the data at the address. BUS_TS sets the Q output to ON if the semaphore was available (OFF) and was acquired. BUS_TS passes power flow to the right whenever it receives power and no error occurs during execution.

Operands

Notes

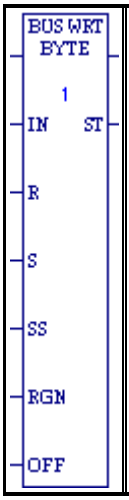
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
R	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.
OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is found by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
ST	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation. **Note:** If BUS_TS_WORD does not contain an even absolute address, the instruction will return a status code of seven, and will not pass power.
Q		data flow	Set ON if the semaphore was available (OFF). Otherwise, Q is set OFF.

CPU Support

BUS_TS_BYTE and BUS_TS_WORD are supported for PACSystems CPUs.

Bus Write



Mnemonics:
BUS_WRT_BYTE
BUS_WRT_DWORD
BUS_WRT_WORD

Operation

The Bus Write (BUS_WRT) instruction writes data to the bus.

Note: Using a Bus instruction (BUS_RD, BUS_WRT, BUS_RMW, or BUS_TS) requires additional information on the correct way to address the bus board. This information may be obtained from one of two sources:

- The board vendor may issue application notes on the correct use of the board.
- For VME boards, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When BUS_WRT receives power flow, it accesses the bus module. It uses the R, S, SS, RGN, and OFF operands to determine the address of the memory to read from. BUS_WRT then copies the data from the input parameter IN to the bus module at the address. BUS_WRT passes power to the right to indicate a successful transfer of data.

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) A BUS_WRT instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

If EN is set to ON, the BUS_WRT instruction stores a status value in the ST output, indicating whether or not the operation was successful and, if not, why it was unsuccessful.

ST Value	***Description***
0	Operation successful
1	Bus error
2	Module does not exist at rack/slot location
3	Module at rack/slot location is an invalid type
4	Start address outside the configured range
5	End address outside the configured address range

6	Absolute address is even but the interface is configured as odd byte only
7	*Not used*
8	Region not enabled
9	*Not used*
10	Function parameter invalid

Operands

Notes

- For each mnemonic, use the corresponding data type for the IN operand. For example, BUS_WRT_BYTE requires IN to be a BYTE variable.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

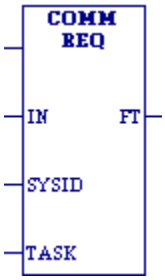
Operand	***Data Type***	***Memory Area***	***Description***
Length	Constant		Length. The number of BYTEs, DWORDs, or WORDs to write to the bus module. 1 ≤ Length ≤ 32,767. Default: 1.
IN	BYTE or WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The data to write to the module
	DWORD variable or constant	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	
R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the rack that contains the module
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of the slot that contains the module
SS	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The number of the subslot that contains the module. Default: 0.
RGN	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The memory region number, as defined on the module's Memory tab. Default: 1.

OFF	DWORD variable or constant	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The offset in the memory region. The exact memory location is found by adding the offset to the memory region's base address, which is defined on the module's Memory tab.
ST	WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The status of the operation.

CPU Support

BUS_WRT_BYTE, BUS_WRT_DWORD, and BUS_WRT_WORD are supported for PACSystems CPUs.

Communication Request



Operation

The Communication Request (COMM_REQ) instruction communicates with a GE intelligent module, such as a Genius Communications Module, a Programmable Coprocessor Module, or a LAN Interface Module.

Notes

- The information presented in this help topic shows only the basic format of the COMM_REQ instruction. Many types of COMM_REQs have been defined. You will need additional information to program the COMM_REQ for each type of device. Programming requirements for each module that uses the COMM_REQ instruction are described in the specialty module's user documentation.

- If you are using Serial Communications, refer to the *Series 90 PLC Serial Communications User's Manual* (GFK-0582). If you are using MMS-Ethernet Communications, refer to the *MMS-Ethernet Communications for the Series 90-70 PLC User's Manual* (GFK-0686).

When COMM_REQ receives power flow, it sends the command block of data specified by the IN operand to the communications TASK in the intelligent or specialty module, at the rack/slot location specified by the SYSID operand. After sending the command block, COMM_REQ can either suspend execution and wait for a reply for a maximum waiting period specified in the command, or resume immediately.

- If the command block specifies that the program will not wait for a reply, the command block contents are sent to the receiving device and the program execution resumes immediately. (The timeout value is ignored.) This is referred to as NOWAIT mode.
- If the command block specifies that the program will wait for a reply, the command block contents are sent to the receiving device and the CPU waits for a reply. The maximum length of time the Controller will wait for the device to respond is specified in the command block. If the device does not respond within that time, program execution resumes. This is referred to as WAIT mode.

COMM_REQ passes power flow, unless the timeout period is exceeded, or if a 0 timeout period has been specified. The instruction Faulted (FT) output may be set ON if:

- The specified target module is not present or is faulted.
- The specified task is not valid for the device.
- The data length is 0.

The instruction Faulted output may have these states:

Enable	*Error?*	*Instruction Faulted Output*
active	no	OFF
active	yes	ON
not active	no execution	OFF

Note: (PACSystems firmware version 2.00 and later, preemptive block scheduling.) A COMM_REQ instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

Command Block

The command block provides information to the intelligent module on the command to be performed. The command block starts at the reference specified by the operand IN. This address may be in any word-oriented area of memory (%R, %P, %L, %W, %AI, or %AQ). The data block length of the command block depends on the amount of data sent to the device.

The Command Block contains the data to be communicated to the other device, plus information related to the execution of the COMM_REQ. The command block has the following structure:

Address	*Data Block Length (in words)*
Address + 1	Wait/No Wait Flag
Address + 2	Status Pointer Memory
Address + 3	Status Pointer Offset
Address + 4	Idle Timeout Value
Address + 5	Maximum Communication Time
Address + 6 to Address + 133	Data Block

Information required for the command block can be placed in the designated memory area using an appropriate programming instruction, such as MOVE, BLKMOV, or DATA_INIT_COMM. (See To configure a DATA_INIT_ instruction.)

When entering information for the command block, refer to these definitions:

Data Block Length

The number of data words starting with the data at address+6 to the end of the command block, inclusive. The data block length ranges from 1 to 128 words. Each COMM_REQ command has its own data block length. When entering the data block length, you must ensure that the command block fits within the register limits

Wait/No Wait Flag

This selects whether or not the program should wait for CCM communications to be completed.

For	*Enter*

No wait	0
Wait for reply	1

The flag bit is stored in the least significant bit (LSB) at address + 1. The rest of the word should be filled with zeros.

Note: Wait mode COMM_REQs cannot be directed to serial ports 1 and 2 on release 7.0 or later CPX CPUs.

Status Pointer Memory Type

The two status pointer words specify a Controller memory location where the status word returned by the device will be written when the COMM_REQ completes.

Status Pointer Memory Type	address + 2
Status Pointer Offset	address + 3

Status pointer memory type contains a numeric code that specifies the user reference memory type for the status word. The table below shows the code for each reference type:

For this memory type		*Enter this decimal value*
%I	Discrete input table (BIT mode)	70
%Q	Discrete output table (BIT mode)	72
%I	Discrete input table (BYTE mode)	16
%Q	Discrete output table (BYTE mode)	18
%R	Register memory	8
%AI	Analog input table	10
%AQ	Analog output table	12

Notes

- The value entered determines the mode. For example, if you enter the %I bit mode is 70, then the offset will be viewed as that bit. On the other hand, if the %I value is 16, then the offset will be viewed as that byte.
- The high byte at address + 2 should contain zero.

Status Pointer Offset

The word at address + 3 contains the offset for the status word within the selected memory type.

Note: The status pointer offset is a zero-based value. For example, %R00001is at offset zero in the register table.

Idle Timeout Value

The idle timeout value is the maximum time the Controller CPU waits for the device to acknowledge receipt of the COMM_REQ. This value is ignored in NOWAIT mode. If WAIT mode is selected, address + 4 specifies the idle timeout period in 100-microsecond increments.

Maximum Communication Time

The value at address +5 specifies the maximum time the Controller CPU waits for the device to complete the COMM_REQ. This time is also specified in 100-microsecond increments and is ignored in NOWAIT mode.

Data Block

The data block contains the command's parameters. The data block begins with a command number in address + 6, which identifies the type of communications instruction to be performed. Refer to the specific device manual (that is, PCM, GBC, Communications) for specific COMM_REQ command formats.

Operands

Note: (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

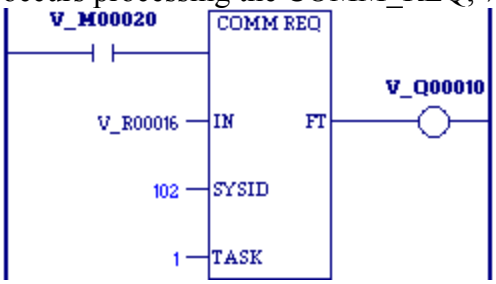
Operand	*Data Type*	*Memory Area*	*Description*
IN	WORD variable	R, P, L, AI, AQ, W, symbolic, I/O variable	The reference of the first WORD of the command block.
SYSID	WORD variable or constant. PACSystems also supports BOOL array of length 16 or more.	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The rack number (most significant byte) and slot number (least significant byte) of the target device (intelligent module). **Note:** For systems that do not have expansion racks, SYSID must be zero for the main rack.
TASK	DWORD variable or constant. PACSystems also supports BOOL array of length 32 or more.	R, P, L, AI, AQ, W, symbolic, I/O variable PACSystems also supports I, Q, M, T, G, SA, SB, SC	The task ID of the process on the target device
FT	Power flow		Instruction Faulted output. FT is energized if an error is detected processing the COMM_REQ: ▪ The specified target address

			(SYSID operand) is not present. ▪ The specified task (TASK operand) is not valid for the device. ▪ The data length is 0. ▪ The device's status pointer address (part of the command block) does not exist. This may be due to an incorrect memory type selection, or an address within that memory type that is out of range.

Examples

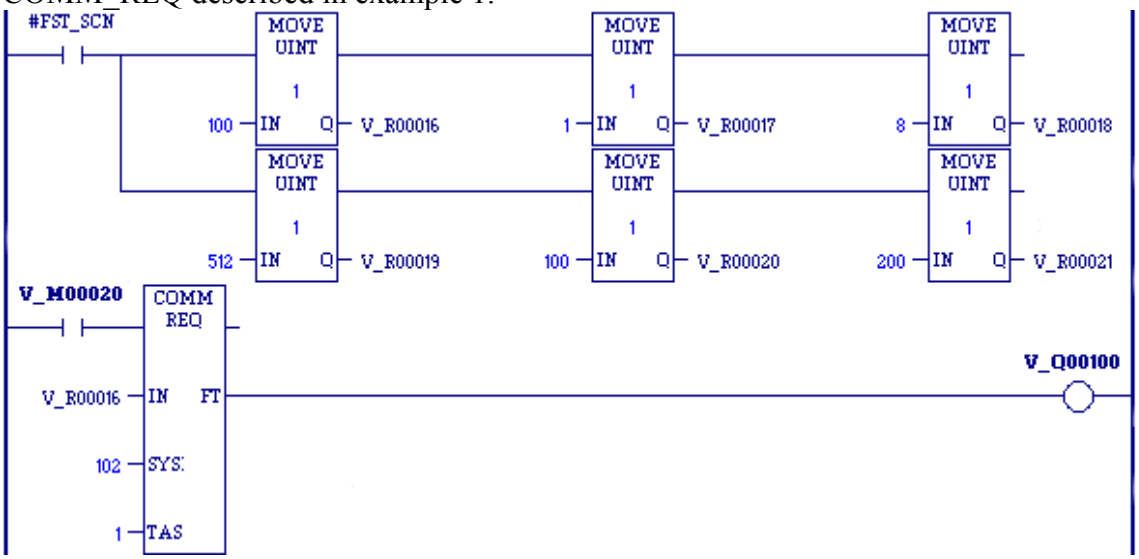
Example 1

When enabling input %M0020 is ON, a command block starting at %R0016 is sent to communications task 1 in the device located at rack 1, slot 2 of the Controller. If an error occurs processing the COMM_REQ, %Q0100 is set.



Example 2

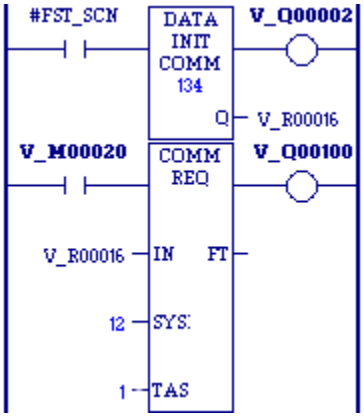
The MOVE instruction can be used to enter the command block contents for the COMM_REQ described in example 1.



Input IN of the COMM_REQ specifies %R00016 as the beginning reference for the command block. Successive references contain the following:

%R00016	Data Block Length
%R00017	Wait/No Wait Flag
%R00018	Status Pointer Memory Type
%R00019	Status Pointer Offset
%R00020	Idle Timeout Value
%R00021	Maximum Communication Time
%R00022 to end of data	Data Block

MOVE instructions supply the following command block data for the COMM_REQ.
The first MOVE instruction places the length of the data being communicated in %R00016.
The second MOVE instruction places the constant 1 in %R00017. This specifies WAIT mode.
The third MOVE instruction places the constant 8 in %R00018. This specifies the register table as the location for the status pointer.
The fourth MOVE instruction places the constant 512 in reference %R00019. Therefore, the status pointer is located at %R00513.
Additional MOVE instructions place the constant 100 in %R00020 and 200 in %R00021. 100 is an idle timeout value of 10 milliseconds for the COMM_REQ; 200 represents the maximum communications time of 20 milliseconds.
The programming logic displayed in example 2 can be simplified by replacing the six MOVE instructions with one DATA_INIT_COMM instruction.



CPU Support

COMM_REQ is supported for all GE IP CPUs.

Data Initialization

	**Mnemonics:** DATA_INIT_DINT DATA_INIT_DWORD DATA_INIT_INT DATA_INIT_LREAL DATA_INIT_UINT DATA_INIT_REAL DATA_INIT_WORD

Note: The mnemonics DATA_INIT_ASCII, DATA_INIT_COMM, and DATA_INIT_DLAN operate differently.

Operation

The Data Initialization (DATA_INIT) instruction copies a block of constant data to a reference range.
When the DATA_INIT instruction is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT instruction in the LD editor. More.
When DATA_INIT receives power flow, it copies the constant data to output Q. DATA_INIT's constant data length (LEN) specifies how much constant data of the instruction type is copied to consecutive reference addresses starting at output Q.
DATA_INIT passes power to the right whenever it receives power.

Notes

- The output parameter is not included in coil checking.
- If you replace one of the DATA_INIT instructions (except DATA_INIT_ASCII, DATA_INIT_COMM, or DATA_INIT_DLAN) with another (except DATA_INIT_ASCII, DATA_INIT_COMM, or DATA_INIT_DLAN), Logic Developer - PLC attempts to keep the same data. For example, configuring a DATA_INIT_INT with 8 rows and then replacing the instruction with a DATA_INIT_DINT would keep the data for the 8 rows. Some precision may be lost when replacing a DATA_INIT_ instruction, and a warning message will be displayed when this case is detected.

Operands

Notes

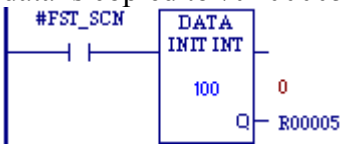
- For each mnemonic, use the corresponding data type for the Q operand. For example, DATA_INIT_DINT requires Q to be a DINT variable.
- (PACSystems only; all DATA_INIT mnemonics except DATA_INIT_REAL.) Indirect referencing is available for register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
"1"	Constant		The quantity (default 1) of constant

			data copied to consecutive reference addresses starting at output Q.
Q	DINT variable	R, P, L, AI, AQ, W. PACSystems also supports data flow, I, Q, M, T, G, symbolic, I/O variable.	The beginning address of where the data is copied to.
	DWORD variable	R, P, L, AI, AQ, W. PACSystems also supports data flow, I, Q, M, T, G, SA, SB, SC, symbolic, I/O variable.	
	INT or UINT variable	I, Q, M, T, G, R, P, L, AI, AQ, W. PACSystems also supports data flow, symbolic, I/O variable.	
	LREAL variable	R, P, L, AI, AQ, W, data flow, I, Q, M, T, G , symbolic, I/O variable	
	REAL variable	R, P, L, AI, AQ, W. PACSystems also supports data flow, I, Q, M, T, G , symbolic, I/O variable.	
	WORD variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W. PACSystems also supports data flow, symbolic, I/O variable.	

Example

On the first scan (as restricted by the #FST_SCN system variable), 100 words of initial data is copied to %R00005 through %R00104.



CPU Support

DATA_INIT_INT, DATA_INIT_UINT, DATA_INIT_DINT, DATA_INIT_WORD, DATA_INIT_DWORD and DATA_INIT_REAL are supported for PACSystems CPUs and Series 90-70 Version 4.00 or later CPUs.
DATA_INIT_LREAL is supported for PACSystems firmware version 5.50 and later.

Data Initialize ASCII

Operation

The Data Initialize ASCII (DATA_INIT_ASCII) instruction copies a block of constant ASCII text to a reference range.
When DATA_INIT_ASCII is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT_ASCII instruction in the LD editor. More.
When DATA_INIT_ASCII receives power flow, it copies the constant data to output Q. DATA_INIT_ASCII's constant data length (LEN) specifies how many bytes of constant text are copied to consecutive reference addresses starting at output Q. LEN must be an even number. DATA_INIT_ASCII passes power to the right whenever it receives power.

Note: The output parameter is not included in coil checking.

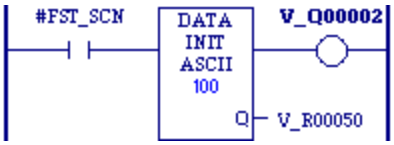
Operands

Note: (PACSystems only.) Indirect referencing is available for register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
"1"	Constant		The number (default 1) of bytes of constant text copied to consecutive reference addresses starting at output Q. LEN must be an even number.
Q	BYTE variable	I, Q, M, T, G, R, P, L, AI, AQ, W. PACSystems also supports data flow, symbolic, I/O variable.	The beginning address of where the data is copied to.

Example

On the first scan (as restricted by the #FST_SCN system variable) the decimal equivalent of 100 bytes of ASCII text is copied to %R00050 through %R00149. %Q00002 receives power.



CPU Support

DATA_INIT_ASCII is supported for PACSystems CPUs and Series 90-70 Version 4.00 or later CPUs.

Data Initialize Communications Request

Operation

The Data Initialize Communications Request (DATA_INIT_COMM) instruction initializes a COMM_REQ instruction with a block of constant data. The IN parameter of the COMM_REQ must correspond with output Q of this DATA_INIT_COMM instruction.

When DATA_INIT_COMM is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT_COMM instruction in the LD editor. More.

When DATA_INIT_COMM receives power flow, it copies the constant data to output Q. DATA_INIT_COMM's constant data length operand specifies how many words of constant data to copy to consecutive reference addresses starting at output Q. The length should be equal to the size of the COMM_REQ instruction's entire command block. DATA_INIT_COMM passes power to the right whenever it receives power.

Note: The output parameter is not included in coil checking.

Operands

Note: (PACSystems only.) Indirect referencing is available for register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
"7"	Constant		The number of WORDs (default 7) of constant data copied to consecutive reference addresses starting at output Q. Must equal the size of the COMM_REQ instruction's entire command block, including the header (words 0-5).
Q	WORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	The beginning address of where the data is copied to.

Notes

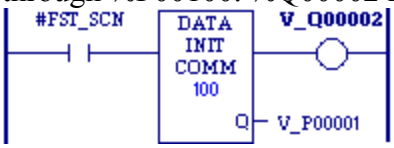
- %W is not a valid status address for the DATA_INIT_COMM instruction in Series 90-70 CPUs. If copying a configured DATA_INIT_COMM instruction from a PACSystems target to a Series 90-70

target, %W defined for the status address will result in an error message (on validation) stating that an invalid reference type is assigned for the operand.

- If mapping the Q operand to %W, an offset must be 16 bits because COMM_REQ does not support more than one word for offset.

Example

On the first scan (as restricted by the #FST_SCN system variable), a command block consisting of 100 words of data, including the 6 header words, is copied to %P00001 through %P00100. %Q00002 receives power.



CPU Support

DATA_INIT_COMM is supported for PACSystems CPUs and Series 90-70 Version 4.00 or later CPUs.

Data initialize DLAN

Operation

The DATA Initialize DLAN (DATA_INIT_DLAN) instruction is for use with a DLAN system which is a limited availability, specialty system. It can only be used in a subroutine block, and can only use %L memory.
When DATA_INIT_DLAN is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT_DLAN instruction in the LD editor. More.
If you have a DLAN system, refer to the Series *90-70 DLAN/DLAN+ Interface Module User's Manual* (GFK-0729B) for details.

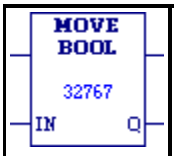
Note: This module is not available as a general purchase item.

CPU Support

DATA_INIT_DLAN is supported for PACSystems version 1.00 or later CPUs, and for Series 90-70 Version 4.00 or later CPUs.

Note: The IC697BEM763 module is supported for PACSystems version 1.50 or later CPUs, and for all Series 90-70 CPUs.

Move



Mnemonics:
MOVE_BOOL
MOVE_DATA
MOVE_DINT
MOVE_DWORD
MOVE_INT
MOVE_LREAL
MOVE_REAL
MOVE_UINT
MOVE_WORD

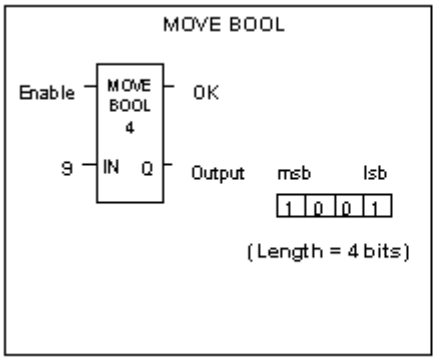
Operands and CPU support: MOVE_BOOL | MOVE_DATA | MOVE_DINT | MOVE_DWORD | MOVE_INT | MOVE_LREAL | MOVE_REAL | MOVE_UINT | MOVE_WORD

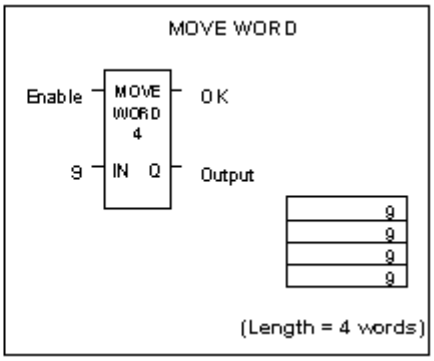
Operation

When a MOVE instruction receives power flow, it copies data as individual bits from one location in Controller memory to another. Because the data is copied in bit format, the new location does not need to be the same data type as the original.
When a MOVE instruction receives power flow, it copies data from input operand IN to output operand Q as bits. If data is moved from one location in discrete memory to another, for example, from %I memory to %T memory, the transition information associated with the discrete memory elements is updated to indicate whether or not the MOVE operation caused any discrete memory elements to change state. Data at the input operand does not change unless there is an overlap in the source and destination.

Note: If an array of BOOL-type data specified in the Q operand does not include all the bits in a byte, the transition bits associated with that byte (which are not in the array) are cleared when the Move instruction receives power flow. The input IN can be either a variable providing a reference for the data to be moved or a constant. If a constant is specified, then the constant value is placed in the location specified by the output reference. For example, if a constant value of 4 is specified for IN, then 4 is placed in the memory location specified by Q. If the length is greater than 1 and a constant is specified, then the constant is placed in the memory location specified by Q and the locations following, up to the length specified. Do not allow overlapping of IN and Q operands.

The result of the MOVE depends on the data type selected for the instruction, as shown below. For example, if the constant value 9 is specified for IN and the length is 4, then 9 is placed in the bit memory location specified by Q and the three locations following:



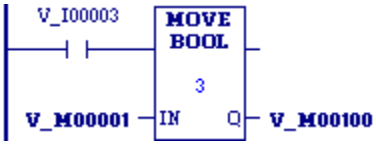


A MOVE instruction passes power to the right whenever it receives power.

Examples

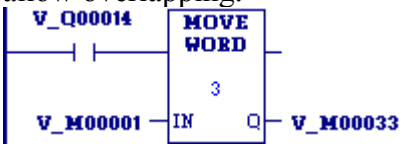
Example 1

Whenever %I00003 is set, the three bits %M00001, %M00002, and %M00003 are moved to %M00100, %M00101, and %M00102, respectively. Coil %Q00001 is turned on.



Example 2

V_M00001 and V_M00033 are both WORD arrays of length 3, for a total of 48 bits in each array. Since Controllers do not recognize arrays, the Length operand has to be set to 3, for the total number of WORDs to be moved. When enabling input V_Q0014 is ON, MOVE_WORD moves 48 bits from the memory location %M00001 to memory location %M00033. Even though the destination overlaps the source for 16 bits, the move is done correctly, except on Series 90-30 CPU351, CPU352, and CPU36x CPUs, which do not allow overlapping.



MOVE_BOOL

Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The length of IN; the number of bits to move. Valid range: ▪ If IN is a constant: 1 ≤ Length ≤ 16. ▪ All other cases: PACSystems and Series 90-70: 1 ≤ Length ≤ 32,767; other Controller families: 1 ≤ Length ≤ 256. Default: 1.
IN	BOOL or WORD variable or constant, bit reference in	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first data item to move. An %I, %Q, %M, or %T reference address need not be byte-aligned, but 16 bits beginning with the reference address specified are displayed online.

	non-BOOL variable		If IN is a constant, it is treated as an array of bits. The value of the least significant bit is copied into the memory location specified by Q. If the Length is greater than one, the bits are copied in order from the least significant to the most significant into successive memory locations, up to the length specified. (PACSystems.) When you select a bit reference in a non-BOOL symbolic variable or I/O variable, ensure that the value IN+Length-1 does not exceed the total number of bits available in the symbolic variable or I/O variable. For example, if MySymbolic is a 16-bit variable, then the combination of values IN = MySymbolic.X[10] and Length=8 is invalid because 10+8-1=17, but if MySymbolic is a symbolic array of length 2, its total number of bits is 32 and the combination of values IN = MySymbolic.X[10] and Length=8 is valid.
Q	BOOL or WORD variable, bit reference in non-BOOL variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first destination data item. An %I, %Q, %M, or %T reference address does not need to be byte-aligned, but 16 bits beginning with the reference address specified are displayed online.

CPU Support

MOVE_BOOL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later CPUs, and Series 90-30 firmware version 3.00 or later CPUs.

In VersaMax CPUs and in Series 90-30 CPU350, you can overlap MOVE_BOOL instructions.

MOVE_DATA

Operands

Notes

- All operands are required.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant	N/A	The length of IN; the number of ENUMs to copy.

			Valid range: 1 through 32,767. Default: 1.
IN	ENUM variable, array of ENUM variables, structure variable, or array of structure variables. (LD only.) Constant 0. (Must be of the same data type as Q, except when the constant 0 (LD only) is assigned to IN)	data flow, R, P, L, AI, AQ, I, Q, M, T, G, S, SA, SB, SC, W, symbolic.	The data to copy to the variable assigned to the output Q. **Note:** (LD only.) If the constant 0 is assigned to IN, then the variable or structure assigned to output Q is set to its default (original) value.
Q	ENUM variable, array of ENUM variables, structure variable, or array of structure variables. (Must be of the same data type as IN, except when the constant 0 is assigned to IN)		The variable assigned to Q contains the data of the variable assigned to IN, as determined by the value of Length. **Note:** If the constant 0 is assigned to IN, then the output variable assigned to Q is set to its default (original) value.

CPU Support

MOVE_DATA is supported for PACSystems only, firmware version 5.60 or later CPUs.

MOVE_DINT

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of DINTs to move. 1 ≤ Length ≤ 32,767.
IN	DINT variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The location of the first DINT to move
Q	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The location of the first destination DINT

CPU Support

MOVE_DINT is supported for PACSystems CPUs, Series 90-70 CPUs, and 20-pt, 40-pt, and 64-pt VersaMax Micro CPUs with firmware version 3.60 or later, except for IC200UDR064 and IC200UDR164.

MOVE_DWORD

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of DWORDs to move. 1 ≤ Length ≤ 32,767.
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The location of the first DWORD to move
Q	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The location of the first destination DWORD

CPU Support

MOVE_DWORD is supported for PACSystems CPUs, Series 90-70 CPUs, and 20-pt, 40-pt, and 64-pt VersaMax Micro CPUs with firmware version 3.60 or later, except for IC200UDR064 and IC200UDR164.

MOVE_INT

Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of INTs to move. ▪ PACSystems and Series 90"-70: 1 ≤ Length ≤ 32,767. ▪ Other Controller family types: 1 ≤ Length ≤ 256.
IN	INT variable	data flow, I, Q, M, T, G, R,	The location of the first INT to

	or constant	P, L, AI, AQ, W, symbolic, I/O variable	move
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first destination INT

CPU Support

MOVE_INT is supported for all GE IP CPUs.
On Series 90-30 CPU351, CPU352, CPU36x, and CPU37x CPUs, MOVE_INT does not support overlapping of IN and Q parameters, where the IN reference is less than the Q reference.
For example, with the following values: IN=%R0001, Q=%R0004, LEN=5 (words), the %R0007 and %R0008 contents will be indeterminate; however, using the following values: Q=%R0001, IN=%R0004, LEN=5 (words) will yield valid contents.

MOVE_LREAL

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of LREAL data items to move. 1 ≤ Length ≤ 32,767.
IN	LREAL variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The location of the first LREAL to move
Q	LREAL variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, W, symbolic, I/O variable	The location of the first destination LREAL

CPU Support

MOVE_LREAL is supported for PACSystems firmware version 5.50 or later.

MOVE_REAL

Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of REAL data items to move. ▪ PACSystems and Series

			90-70: 1 ≤ Length ≤ 32,767. ▪ Other Controller family types: 1 ≤ Length ≤ 256.
IN	REAL variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The location of the first REAL to move
Q	REAL variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The location of the first destination REAL

CPU Support

MOVE_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 floating-point CPUs, and Series 90-30 floating-point CPUs.

MOVE_UINT

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of UINTs to move. 1 ≤ Length ≤ 32,767.
IN	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first UINT to move
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first destination UINT

CPU Support

MOVE_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

MOVE_WORD

Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

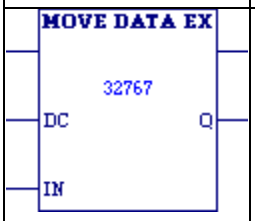
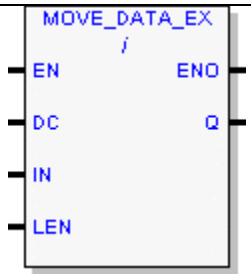
Operand	*Data Type*	*Memory Area*	*Description*

Length	Constant		Length; the number of WORDs to move. ▪ PACSystems and Series 90-70: 1 ≤ Length ≤ 32,767. ▪ Other Controller family types: 1 ≤ Length ≤ 256.
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first WORD to move
Q	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The location of the first destination WORD

CPU Support

MOVE_WORD is supported for all GE IP CPUs.
On Series 90-30 CPU351, CPU352, and CPU36x CPUs, MOVE_WORD does not support overlapping of IN and Q parameters, where the IN reference is less than the Q reference.
For example, with the following values: IN=%R0001, Q=%R0004, LEN=5 (words), the %R0007 and %R0008 contents will be indeterminate; however, using the following values: Q=%R0001, IN=%R0004, LEN=5 (words) will yield valid contents.

Move Data Explicit

LD	FBD	ST
		Formal convention: MOVE_DATA_EX(DC := , IN := , Length := , Q =>); - or - MOVE_DATA_EX(IN := , Length := , Q =>); **Tip:** Drag instruction from the Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. The instruction with its input and output operand names are inserted in ST logic. Now assign a variable to the inputs and outputs, and a variable to the left of the assignment operator.

Operation

MOVE_DATA_EX provides optional data coherency by locking symbolic memory being written to during the copy operation. This enables a large number of bytes to be copied at one time.

Notes

- The input DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block.
- If DC is True, an interrupt block cannot preempt the copy operation.
- If DC is False or not present, then interrupts can preempt the copy.
- Using DC can impact interrupt latency if the amount of data copied is large.

Operands

Notes

- All operands are required except for input DC (Data Coherency) and output Q.
- The constant value 0 is valid for DC.
- Input LEN (Length) must be a constant value.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) Solve order for the instruction.		

Power flow (LD)			When On, MOVE_DATA_EX executes. When Off, MOVE_DATA_EX does not execute.
EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, MOVE_DATA_EX solves. When Off, MOVE_DATA_EX does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
DC (Optional.)	▪ LD: data flow ▪ FBD: BOOL variable or BOOL system variable ▪ ST: BOOL variable, BOOL system variable, or BOOL constant	data flow, I, Q, M, T, G, symbolic, I/O variable	Data Coherency. ▪ If True, symbolic memory being written to is locked, enabling a coherent copying of symbolic data from one Controller memory area to another. ▪ If False (default), symbolic data is copied normally from one Controller memory area to another *without* data coherency. **Notes** ▪ DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block. ▪ If DC is True, an interrupt block cannot preempt the copy operation. ▪ If DC is False or not present, then interrupts

			can preempt the copy.
IN	▪ Enumerated variable, array of enumerated variables, structure variable, or array of structure variables. ▪ (LD only.) Constant value 0.	I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic. data flow only if IN is connected to output Q of another MOVE_DATA_EX, MOVE_FROM_FLAT, or MOVE_TO_FLAT instruction.	Symbolic data to copy to the variable assigned to output Q as determined by the constant value assigned to LEN (Length). **Notes** ▪ The variable assigned to IN must be of the same data type as the variable assigned to output Q, except when the constant 0 (LD only) is assigned to IN. ▪ (LD only.) If the constant 0 is assigned to IN, then LEN (Length) is assigned the constant 1 and the variable or structure assigned to Q is set to its default (original) value.
▪ LEN (FBD) ▪ Length (ST)	INT constant in all languages	N/A	Length of IN; number of IN elements to copy. Valid range: 1 through 32,767. Default: 1. **Tip:** In the LD editor, double-click MOVE_DATA_EX to edit the LEN (Length) constant value.

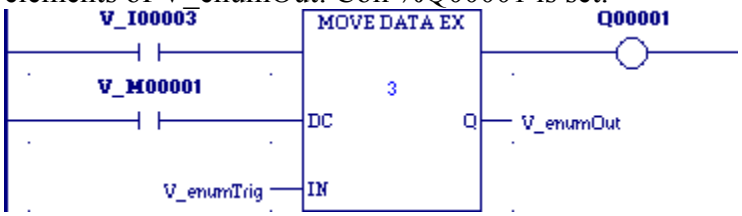
Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			On when MOVE_DATA_EX executed successfully
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
Q	Enumerated variable, array of enumerated variables, structure variable, or array of structure variables	I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic data flow only if Q is connected to input IN of another MOVE_DATA_EX, MOVE_FROM_FLAT, or MOVE_TO_FLAT instruction	Variable or array to which IN is copied. **Note:** The variable assigned to Q must be of the same data type as the variable assigned to input IN, except when the constant 0 (LD only) is assigned to IN.

Examples

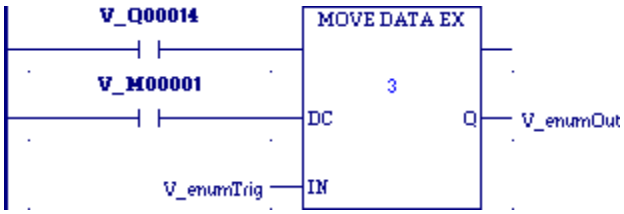
Example 1

When %I00003 is set, the first three elements of V_enumTrig are copied to the first three elements of V_enumOut. Coil %Q00001 is set.



Example 2

V_enumTrig and V_enumOut are both enumerated arrays of three elements in each array. Because Controllers do not recognize arrays, input Length should be 3, for the total number of enumerated elements to be copied. When the enabling input V_Q0014 is On, MOVE_DATA_EX copies three elements from memory location V_enumTrig to memory location V_enumOut.



CPU Support

MOVE_DATA_EX is supported for PACSystems with firmware version 6.00 or later.

MOVE_FROM_FLAT

LD	FBD	ST
	This instruction is not currently available in FBD and ST. Use LD for the MOVE_FROM_FLAT operation. If you use MOVE_FROM_FLAT in FBD or ST, you will get validation error 12586.	

Operation

MOVE_FROM_FLAT copies reference memory data to a User-defined Data Type (UDT) variable or UDT array. MOVE_FROM_FLAT provides optional data coherency by locking the symbolic or I/O variable memory area being written to during the copy operation. This enables a large number of bytes to be replaced at one time.

Notes

- The input DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block.
- If DC is True, an interrupt block cannot preempt the copy operation.
- If DC is False or not present, then interrupts can preempt the copy.
- Using DC can impact interrupt latency if the amount of data copied is large.

Copying arrays and array elements

The constant value assigned to input LEN (Length) determines the number of UDT array elements to be filled by copying data from reference memory to output Q.
Example: If constant value 6 is assigned to input LEN (Length), there should be a UDT array of at least six elements assigned to output Q. During logic execution, *n* bytes of data are copied from reference memory to the first six UDT array elements, where *n* is the length of the UDT array element (in bytes) times six.

Copying to specified array elements

For output Q, a single element of a UDT array can be specified, for example, myUDT_array[4] (5th element of myUDT_array). In this case, the input LEN (Length) operand applies to the array elements starting from and including myUDT_array[4].
Example: myUDT_array is a UDT array of ten elements, of which each element is a UDT variable, and myUDT_array[4] is assigned to output Q. This restricts the value of input LEN (Length) to be six or less because there are six remaining UDT array elements that can be filled in myUDT_array. If LEN (Length) is greater than six, then during validation, the following error appears in the Feedback Zone:
Error 9672: Managed variable too small for parameter Block [block name]: Rung [rung number]

Bounds check

If the number of bytes of the variable in reference memory assigned to input IN is greater than the Controller memory limit, an error appears in  Feedback Zone and prevents downloading to the Controller.

Note: The constant value assigned to input LEN (Length) is not considered in this check.

Example: If a variable assigned to input IN is mapped to %R1 and the length (in bytes) of the UDT variable assigned to output Q is 6, then the highest used reference address is %R3. If the Controller %R memory limit is less than 3, downloading to Controller is stopped and an error appears in the Feedback Zone:
Error 8038: %R memory usage exceeds limits in [block name].
When MOVE_FROM_FLAT receives power flow, the following occur:

- Reference memory data is copied to a UDT variable in Controller memory.
- The number inside MOVE_FROM_FLAT is the number of output elements to be written to. Valid range: 1 through 32,767.
- (Input Data Coherency optional.) If input DC is True, the symbolic or I/O variable memory area assigned to output Q is locked, enabling coherent copying of data from reference memory of input IN.
- If input DC is False (default), data is copied normally from the memory referenced by input IN to a UDT variable in Controller memory without data coherency; that is, without locking the symbolic or I/O variable memory area being copied to.

Operands

Input Operands

In the LD editor, double-click MOVE_FROM_FLAT to edit the LEN (Length) constant value.

Operand	*Data Type*	*Memory Area*	*Description*
Power flow			When On, MOVE_FROM_FLAT executes. When Off, MOVE_FROM_FLAT does not execute.

DC (Optional.)	data flow	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	Data Coherency. ▪ If True, the UDT variable being written to in Controller memory is locked, enabling a coherent copying of reference data to the UDT variable in symbolic or I/O variable memory. ▪ If False (default), reference data is copied normally to a UDT variable without locking the symbolic or I/O variable memory area being copied to; that is, *without* data coherency. **Notes** ▪ DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block. ▪ If DC is True, an interrupt block cannot preempt the copy operation. ▪ If DC is False or not present, then interrupts can preempt the copy.
IN	BYTE, WORD, or array of BYTEs or WORDs	All memory areas except [%S, discrete symbolic, discrete I/O variable]	Reference memory data being copied to UDT variable elements in output Q as determined by the constant value assigned to LEN (Length). **Notes** ▪ Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ). ▪ BYTE arrays must be packed; that is, they must be in discrete memory.

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (optional.)			On when MOVE_FROM_FLAT executed successfully

Q	User-defined Data Type (UDT) variable or UDT array	Discrete or non-discrete symbolic, discrete or non-discrete I/O variable	On execution, reference memory data from input IN is copied to Q. **Notes** ▪ Input LEN (Length) determines how many UDT variable elements to overwrite in Q. ▪ If an array head is assigned to input IN, the constant value assigned to LEN (Length) determines how many UDT array elements assigned to Q are filled by copying data from reference memory.

Example

Consider the following:

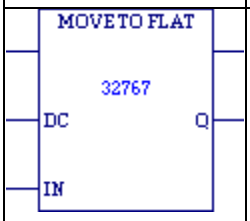
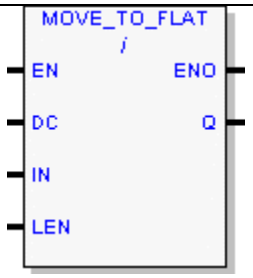
- A WORD variable mapped to %R1 is assigned to input IN.
- The constant value 1 is assigned to input LEN (Length).
- A UDT variable or UDT array is assigned to output Q.

When MOVE_FROM_FLAT executes, *n* bytes of data are copied starting at %R1 to a UDT variable or UDT array, where *n* is the UDT array element length (in bytes). If a UDT array is assigned to output Q, *n* bytes of data are copied to the first UDT array element.

CPU Support

MOVE_FROM_FLAT is supported in LD for PACSystems with firmware version 6.00 or later.

MOVE_TO_FLAT

LD	FBD	ST
		Formal convention: MOVE_TO_FLAT(DC := , IN := , Length := , Q =>); - or - MOVE_TO_FLAT(IN := , Length := , Q =>); **Tip:** Drag instruction from Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. The instruction with its input and output operand names are inserted in ST logic. Now assign a variable to the inputs and outputs, and a variable to the left of the assignment operator.

Operation

MOVE_TO_FLAT instruction copies data from symbolic or I/O variable memory to reference memory. MOVE_TO_FLAT copies across mismatching data types for an operation such as a Modbus transfer. MOVE_TO_FLAT provides optional data coherency by locking the reference memory being written to during the copy operation. This enables a large number of bytes to be copied at one time.

Notes

- The input DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block.
- If DC is True, an interrupt block cannot preempt the copy operation.
- If DC is False or not present, then interrupts can preempt the copy.
- Using DC can impact interrupt latency if the amount of data copied is large.

Copying arrays and array elements

(LD, FBD.) The constant value assigned to input LEN (Length) determines the number of UDT array elements to be copied to the reference memory of the variable assigned to output Q.

Example: If the constant value 6 is assigned to LEN (length), then there should be a UDT array of at least six elements assigned to input IN. When logic executes, *n* bytes of data are copied from the UDT array elements to the reference memory of the variable assigned to output Q, where *n* is the length of the UDT array element (in bytes) times six.

Bounds check

If the number of bytes in the reference memory of the variable assigned to output Q is greater than the memory limit configured in Controller memory, an error appears in the  Feedback Zone and prevents downloading to the Controller.

Note: The constant value assigned to input LEN (Length) is not considered in this check.

Example: If the reference memory of the variable assigned to output Q is %R1 and the length (in bytes) of the UDT assigned to input IN is 6, then the highest used reference address is %R3. If the %R memory limit in the Controller is less than 3, then downloading to the Controller is prevented and an error appears in the Feedback Zone: Error 8038: %R memory usage exceeds limits in <block name>.
When MOVE_TO_FLAT receives power flow in LD, or executes in FBD or ST, the following occur:

- Data is copied from one location in Controller memory to another as determined by the constant value assigned to input LEN (Length).
- (LD.) The number inside MOVE_TO_FLAT is the number of UDT elements to be copied from. This number is the same as the input LEN value in FBD or ST. Valid range: 1 through 32,767.
- (Input Data Coherency optional.) If input DC is True, the reference memory of the variable assigned to output Q is locked, enabling a coherent copy of data from the UDT of input IN.
- If input DC is False (default), data is copied normally from the UDT to the reference memory of the variable assigned to Q without data coherency; that is, without locking the reference memory being copied to.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) Solve order for the instruction.		
Power flow (LD)			When On, MOVE_TO_FLAT executes. When Off, MOVE_TO_FLAT does not execute.
EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, MOVE_TO_FLAT solves. When Off, MOVE_TO_FLAT does not solve.

	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
DC (Optional.)	▪ LD: data flow ▪ FBD: BOOL variable or BOOL system variable ▪ ST: BOOL variable, BOOL system variable, or BOOL constant	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	Data Coherency. ▪ If True, the reference memory being copied to is locked. This enables a coherent copy of a UDT to reference memory. ▪ If False (default), the UDT is copied normally to reference memory without locking the reference memory area being copied to; that is, *without* data coherency. **Notes** ▪ DC should be used only when using interrupt blocks and is required only when the same memory is used in more than one interrupt block, or in the main program and an interrupt block. ▪ If DC is True, an interrupt block cannot preempt the copy operation. ▪ If DC is False or not present, then interrupts can preempt the copy.
IN	User-defined Data Type (UDT) variable or UDT array	Discrete or non-discrete symbolic, discrete or non-discrete I/O variable	The data copied to the reference memory that the variable assigned to Q is mapped to. **Notes** ▪ Input LEN (Length) determines how many UDT variable elements to copy to Q. ▪ If an array head is assigned to input IN, the constant value assigned to LEN (Length) determines how many UDT array elements assigned to IN are copied to reference memory.

▪ LEN (FBD) ▪ Length (ST)	INT constant in all languages	N/A	Length of IN; number of UDT elements to copy to reference memory in output Q. Valid range: 1 through 32,767. Default: 1. **Tip:** In the LD editor, double-click MOVE_TO_FLAT to edit the LEN (Length) constant value.

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			On when MOVE_TO_FLAT executed successfully
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
Q	BYTE, WORD, or array of BYTEs or WORDs	All memory areas except [%S, discrete symbolic, discrete I/O variable]	The variable, mapped to reference memory, that IN is copied to. The amount of data copied is determined by the constant value assigned to input LEN (Length). **Notes** ▪ Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ). ▪ BYTE arrays must be packed; that is, they must be in discrete memory.

Example

Consider the following:

- A UDT variable or UDT array is assigned to input IN.
- The constant value 8 is assigned to input LEN (Length).
- A DWORD variable mapped to %R1 is assigned to output Q.

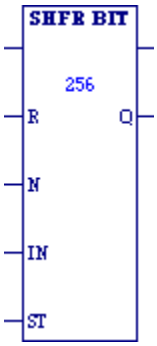
If the constant value 8 assigned is assigned to LEN (length), there should be a UDT array of at least eight elements assigned to IN. When MOVE_TO_FLAT executes, *n* bytes of

data are copied from the UDT variable or array to %R memory, starting at %R1 in the example, where *n* is the length of a UDT array element (in bytes) times eight.

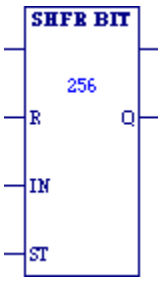
CPU Support

MOVE_TO_FLAT is supported for PACSystems with firmware version 6.00 or later.

Shift Register



PACSystems and Series 90-70 only



Other CPUs

Mnemonics:
SHFR_BIT
SHFR_DWORD
SHFR_WORD

Operands: SHFR_BIT | SHFR_DWORD | SHFR_WORD

Operation

When the Shift Register (SHFR_BIT, SHFR_DWORD, or SHFR_WORD) instruction receives power and the R operand does not, SHFR shifts one or more data BITs, data DWORDs, or data WORDs from a reference location into a specified area of memory. A contiguous section of memory serves as a shift register. For example, one word might be shifted into an area of memory with a specified length of five words. As a result of this shift, another word of data would be shifted out of the end of the memory area.

Warning: The use of overlapping input and output reference address ranges in multiword instructions is not recommended, as it may produce unexpected results.

The reset input (R) takes precedence over the instruction enable input. When the reset is active, all references beginning at the shift register (ST) up to the length specified for the Length operand, are filled with zeros.
If the instruction receives power flow and R is not active, each BIT, DWORD, or WORD of the shift register is moved to the next highest reference. The last element in the shift register is shifted into Q. The highest reference of the shift register element of IN is shifted into the vacated element starting at ST.

Note: The contents of the shift register are accessible throughout the logic because they are overlaid on absolute locations in logic addressable memory.

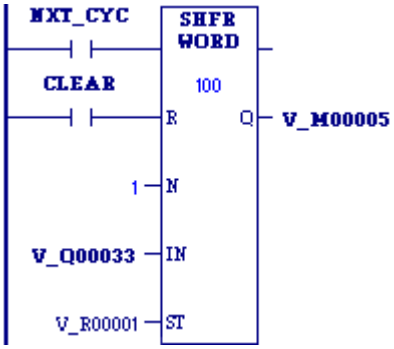
The instruction passes power to the right whenever it receives power flow and the R operand does not.

Examples

Example for PACSystems or Series 90-70

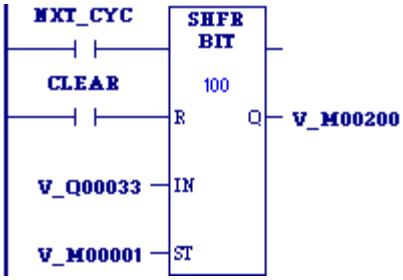
SHFR_WORD operates on register memory locations %R0001 through %R0100. When the reset reference CLEAR is active, the Shift Register words are set to zero.

When the NXT_CYC reference is active and CLEAR is not, the word from output status table location %Q0033 is shifted into the Shift Register at %R0001. The word shifted out of the Shift Register from %R0100 is stored in output %M0005. Note that, for this example, the length specified for LEN and the amount of data to be shifted (N) are not the same.



Example for other CPUs

SHFR_BIT operates on discrete memory locations %M0001 through %M0100. When the reset reference CLEAR is active, SHFR_BIT fills %M0001 through %M0100 with zeros. When NXT_CYC is active and CLEAR is not, SHFR_BIT shifts the data in %M0001 to %M0100 down by one bit. The bit in %Q0033 is shifted into %M0001 while the bit shifted out of %M0100 is written to %M0200.



CPU Support

SHFR_BIT is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 firmware version 3.00 or later CPUs, and Series 90-30 CPUs.
SHFR_DWORD is supported for PACSystems CPUs and Series 90-70 CPUs.
SHFR_WORD is supported for all GE IP CPUs.

SHFR_BIT

Operands

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of bits in the shift register. 1 ≤ Length ≤ 256. Default: 1.
R	Power flow		Reset. When R is ON, the shift register

			located at ST is filled with zeroes.
N	Constant		(PACSystems and Series 90-70.) The number of bits to shift into the shift register. 1 ≤ N ≤ Length.
IN	BOOL or WORD variable or constant, bit reference in non-BOOL variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to shift into the first bit or word of the shift register. When you select a bit reference in a non-BOOL symbolic variable or I/O variable, Machine Edition ensures that the value [index of IN]+N-1 does not exceed the total number of bits available in the symbolic variable or I/O variable. For example, if MySymbolic is a 16-bit variable and IN = MySymbolic.X[10], then the combination of values [index of IN = 10] and N=8 is invalid because 10+8-1=17, but if MySymbolic is a symbolic array of length 2, its total number of bits is 32 and the combination of values [index of IN = 10] and N=8 is valid. (PACSystems and Series 90-70.) For %I, %Q, %M and %T memory, a discrete reference address does not need to be byte-aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
ST	BOOL or WORD variable, bit reference in non-BOOL variable	I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The shift register. **Note:** (PACSystems and Series 90-70.) For %I, %Q, %M and %T memory, a discrete reference address does not need to be byte-aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
Q	BOOL or WORD variable, bit reference in non-BOOL variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The BIT(s) or WORD(s) shifted out of the shift register. In PACSystems and Series 90-70, Q must be able to hold a number of bits greater than or equal to N. **Note:** (PACSystems and Series 90-70.) For %I, %Q, %M and %T memory, a discrete reference address does not need to be byte-aligned. However, 1 bit, beginning with the reference address specified, is displayed online.

SHFR_DWORD

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of DWORDs in the shift register. 1 ≤ Length ≤ 256.
R	Power flow		Reset. When R is ON, the shift register located at ST is filled with zeroes.
N	Constant		The number of DWORDs to shift into the shift register. 1 ≤ N ≤ Length.
IN	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The value to shift into the first DWORD of the shift register
ST	DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The shift register.
Q	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The DWORD(s) shifted out of the shift register. In PACSystems and Series 90-70, Q must be able to hold a number of DWORDs greater than or equal to N.

Notes

- **The memory limit for a mapped reference address must be greater than or equal to the following:**

  ST + Length * 2 -1

  IN + N * 2 - 1

  Q + N * 2 - 1

- **For a symbolic or I/O variable, the following apply:**

  The array size of ST is greater than or equal to Length.

  The array size of IN is greater than or equal to N.

  The array size of Q is greater than or equal to N.

SHFR_WORD

Note: (PACSystems and Series 90-70.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operands

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of WORDs in the shift register. 1 ≤ Length ≤ 256.
R	Power flow		Reset. When R is ON, the shift register located at ST is filled with zeroes.
N	Constant		(PACSystems and Series 90-70.) The number of WORDs to shift into the shift register. 1 ≤ N ≤ Length.
IN	WORD variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to shift into the first WORD of the shift register
ST	WORD variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The shift register. **Note:** (Series 90 Micro firmware bug.) A Series 90 Micro treats the ST input of both SHFR_WORD and BIT_SEQ as an output when it updates its coil usage map. To prevent equality problems with a Series 90 Micro target, Machine Edition also adds the ST input of those instructions to the coil usage map. However, if you map the ST input of multiple BIT_SEQ and/or SHFR_WORD instructions to the same address, you will get a Multiple Coil Use error or warning upon validation, unless you edit the Multiple Coil Use Warning option and set it to No

			Warning.
Q	WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	The WORD(s) shifted out of the shift register. In PACSystems and Series 90-70, Q must be able to hold a number of WORDs greater than or equal to N.

Notes

- **The memory limit for a mapped reference address must be greater than or equal to the following:**

  ST + Length -1

  IN + N - 1

  Q + N - 1

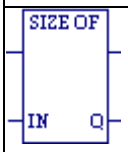
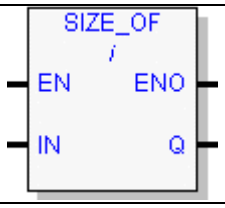
- **For a symbolic or I/O variable, the following apply:**

  The array size of ST is greater than or equal to Length.

  The array size of IN is greater than or equal to N.

  The array size of Q is greater than or equal to N.

Size Of

LD	FBD	ST
		Formal convention: Q := SIZE_OF(In := [input]);

Operation

SIZE_OF counts the number of bits used by the variable assigned to input IN.

Output

- In LD and FBD, SIZE_OF writes the number of bits to output Q.
- In ST, the value is assigned to the variable to the left of the assignment operator, represented by variable Q in the ST code above.

A validation error occurs if you assign one of the following to input IN:

- A BYTE array in non-discrete memory
- A double-segment structure variable

Tip: If the variable assigned to input IN of SIZE_OF is passed to a parameterized C block for processing, also pass the value of output Q to the block. In C block logic, use the value of output Q to ensure that you process all the bits used by the variable without exceeding the end of the variable.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) The solve order for the instruction.		
power flow (LD only)			When set to On, SIZE_OF executes. When set to Off, SIZE_OF does not execute.
EN (FBD only)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	Enable input. When set to On, SIZE_OF solves. When set to Off, SIZE_OF does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
IN	Variable of any data type except BYTE arrays in non-	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L,	The variable whose size in bits is

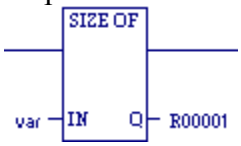
	discrete memory and double-segment structures	AI, AQ, W, symbolic, I/O variable	calculated

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
power flow (LD only; optional)			Set to On when SIZE_OF has executed successfully
ENO (FBD only; optional)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, I/O variable	ENO is set to On if EN is set to On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, I/O variable	
Q	DINT or DWORD variable. ST also supports INT and WORD variables.	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The number of bits used by the variable assigned to input IN. In ST, the value is assigned to the variable on the left side of the assignment operator.

Example

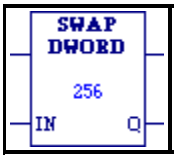
The single-segment structure named var assigned to input IN contains eight BOOL elements (8 * 1 = 8 bits) and twelve WORD elements (12 * 16 = 192 bits). SIZE_OF outputs the value 8 + 192 = 200 to the variable R00001 assigned to output Q.



CPU Support

SIZE_OF is supported for all PACSystems CPUs.

Swap



Mnemonics:
SWAP_DWORD
SWAP_WORD

Operation

SWAP_DWORD swaps the two WORDs within a DWORD and SWAP_WORD swaps the two BYTEs within a WORD. To operate over a range of memory, specify a length greater than 1: each word or double word of data within the specified length will then be appropriately swapped.
When SWAP receives power flow, it performs the swap operation on each word or double word of data within the specified area and stores the results of the swap to output Q. SWAP passes power to the right whenever it receives power.

Operands

Notes

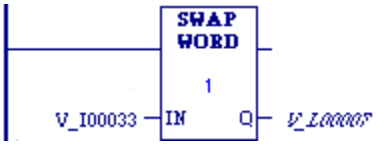
- For each mnemonic, use the corresponding data type for the IN and Q operands. For example, SWAP_DWORD requires IN and Q to be DWORD variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of WORDs or DWORDs to operate on. 1 ≤ Length ≤ 256.
IN (Must be the same data type as Q.)	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	The data to swap.
	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q (Must be the same data type as IN.)	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	The swapped version of the original data IN.

	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

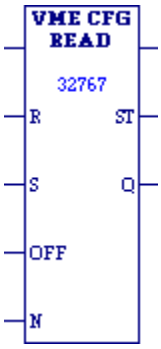
Two bytes located in bits %I00033 through %I00048 are swapped. The result is stored in %L00007.



CPU Support

SWAP_WORD and SWAP_DWORD are supported for PACSystems CPUs and Series 90-70 CPUs.

VME Configuration Read



Operation

When VME_CFG_READ receives power, it reads the data elements (N) from the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF). The data read is placed in output Q. The status of the operation is placed in the status word output (ST). VME_CFG_READ has a length specification (LEN) of the maximum size of the output array.

Note: The module at the specified rack and slot must be configured as a third-party VME module in BUS INTERFACE mode for this instruction to execute successfully. Additional parameters indicating the AM code, the location and size of the module's dual port, and the bus interface type must be specified in the module's configuration for correct operation. See chapter 11 of the *Logicmaster 90-70 Programming Software User's Manual* (GFK-0263) for more information on configuration of third-party VME modules.

VME_CFG_READ passes power to the right only when it completes successfully. VME_CFG_READ fails if any of the following occurs:

- The number of data elements (N) is greater than the length (LEN) specified.
- The rack/slot value (R and S) is out of range or is not a valid VME location.
- The most significant byte of the dual port offset (OFF) is not zero.
- The most significant byte of the dual port address plus the dual port offset is not zero.
- Read beyond the end of dual port memory.
- Specified rack/slot not configured for a Third-Party VME module in BUS INTERFACE mode.
- If the dual port offset is an even number, configure for the odd byte only. If the dual port offset is an odd number, configure for word or single word.

Operands

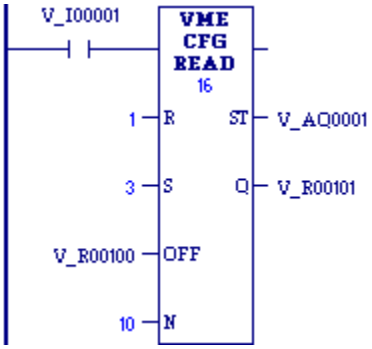
Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length of the output array in bytes.

R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Rack number of VME module
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Slot number of VME module
OFF	DWORD variable or constant	data flow, S, R, P, L, AI, AQ	Dual port offset
N	INT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Amount of data (data elements) to read from the VME bus.
ST	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ	Status word, which contains the status of the operation
Q	BYTE variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ	Address of where the data read is written to

Example

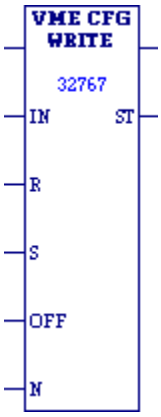
when enable is ON, VME data at rack 1, slot 3 and dual port offset defined by %R00100 is read into %R00101 through %R00110 of the memory block %R00101 through %R00116. If an error is encountered, the status word %AQ0001 contains an error code.



CPU Support

VME_CFG_READ is supported for Series 90-70 Version 4.00 or later CPUs.

VME Configuration Write



Operation

When VME_CFG_WRITE receives power, it writes the data elements (N) from the data array (IN) to the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF). The status of the operation is placed in the status word output (ST). The instruction has a length specification (LEN) of the maximum size of the output memory block.

Note: The module at the specified rack and slot must be configured as a third-party VME module in BUS INTERFACE mode for this instruction to execute successfully. Additional parameters indicating the AM code, the location and size of the module's dual port, and the bus interface type must be specified in the module's configuration for correct operation. See chapter 11 of the *Logicmaster 90-70 Programming Software User's Manual* (GFK-0263) for more information on configuration of third-party VME modules.

VME_CFG_WRITE passes power to the right only when it completes successfully. VME_CFG_WRITE fails if any of the following occurs:

- The number of data elements (N) is greater than the length (LEN) specified.
- The rack/slot value (R and S) is out of range or is not a valid VME location.
- The most significant byte of the dual port offset (OFF) is not zero.
- The most significant byte of the dual port address plus the dual port offset is not zero.
- Read beyond the end of dual port memory.
- Specified rack/slot not configured for a Third-Party VME module in BUS INTERFACE mode.
- If the dual port offset is an even number, configure for the odd byte only. If the dual port offset is an odd number, configure for word or single word.

Operands

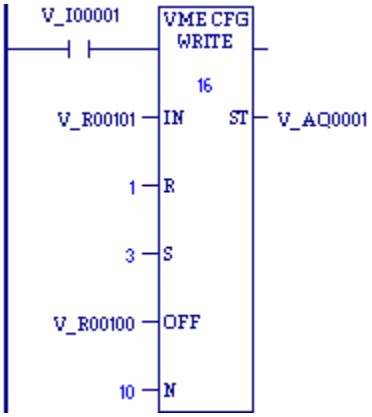
Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length of the input array in bytes.

			1 ≤ Length ≤ 32,767.
IN	BYTE variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	The data to write to the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF)
R	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Rack number of VME module
S	UINT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Slot number of VME module
OFF	DWORD variable or constant	data flow, S, R, P, L, AI, AQ	Dual port offset
N	INT variable or constant	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ	Number of registers of data to write to the VME bus. Should be ≤ Length.
ST	WORD variable	data flow, I, Q, M, T, SA, SB, SC, G, R, P, L, AI, AQ	Status word, which contains the status of the operation

Example

When enable is ON, data from %R00101 through %R00110 of the memory block %R00101 through %R00116 is written to the VME bus at rack 1, slot 3 and dual port offset defined by %R00100. If an error is encountered, the status word %AQ0001 contains an error code.



CPU Support

VME_CFG_WRITE is supported for Series 90-70 Version 4.00 or later CPUs.

VME Read

Mnemonics:
VME_RD_BYTE
VME_RD_WORD

Operation

The VME Read (VME_RD) instruction reads data from the VME bus.

Note: Using a VME instruction (VME_RD, VME_WRT, VME_RMW, or VME_TS) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources:

- For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board.
- Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When VME_RD receives power flow, it accesses the VME module at the address specified by ADR and the address modifier AM. VME_RD then copies data with the length LEN to Controller locations beginning at output Q. VME_RD passes power to the right when its operation is successful.

Operands

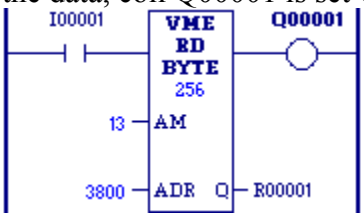
Notes

- For each mnemonic, use the corresponding data type for the Q operand. For example, VME_RD_BYTE requires Q to be a BYTE variable.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; 1 ≤ Length ≤ 32,767
AM	WORD variable or constant	data flow, R, P, L, AI, AQ	Address modifier of VME operation
ADR	DWORD constant or variable	data flow, R, P, L, AI, AQ	Address of the data to read
Q	BYTE variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ	The data read from the address specified by ADR and AM.
	WORD variable	R, P, L, AI, AQ	

Example

When enabling input I00001 goes ON, 256 bytes of short supervisory space are read from address 3800 on the VME bus into registers %R00001 through %R00128. (In a multiple rack system with a BTM, this would be on rack 4.) Unless an error occurs while reading the data, coil Q00001 is set ON.



CPU Support

VME_RD_BYTE and VME_RD_WORD are supported for Series 90-70 CPUs.

VME Read Modify Write

Mnemonics:
VME_RMW_BYTE
VME_RMW_WORD

Operation

The VME Read/Modify/Write (VME_RMW) instruction updates a data element on the VME bus.

Note: Using a VME instruction (VME_RD, VME_WRT, VME_RMW, or VME_TS) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources:

- For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board.
- Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When VME_RMW receives power flow, it accesses the VME module at the address specified by ADR and the address modifier AM.

- **Read phase:** VME_RMW reads the byte or word of data from the VME at the address.
- **Modify phase:** This byte or word of data is combined (AND/OR) with the data mask MSK. Selection of AND or OR is made using the OP input. MSK is a word value. If byte data is operated on, only the lower 8 bits of MSK are used.
- **Write phase:** The result is written back to the same VME address from which it was read.

VME_RMW passes power flow to the right whenever it receives power, unless an error occurs.

Operands

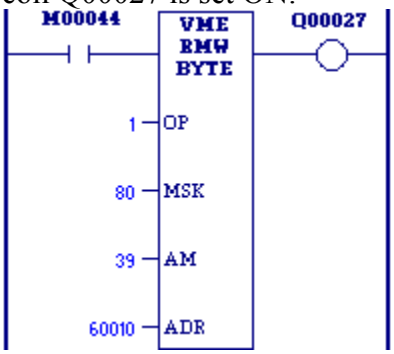
Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
OP	Constant		Only 0 and 1 are valid: ▪ A value of 0 ANDs the data with the MSK data ▪ A value of 1 ORs the data

			with the MSK data
MSK	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ	The data mask. **Note:** For VME_RMW_BYTE, only the lower 8 bits are used.
AM	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ	Address modifier of VME operation
ADR	DWORD variable	data flow, R, P, L, AI, AQ	Address where to read and write the data

Example

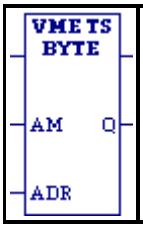
When enabling input M00044 is ON, the hexadecimal value 80H is ORed with the byte of data read from address 060010H on the VME bus in rack 0 (the main rack) using Standard Non-Privileged Data Access. Unless an error occurs while accessing the data, coil Q00027 is set ON.



CPU Support

VME_RMW_BYTE and VME_RMW_WORD are supported for Series 90-70 CPUs.

VME Test and Set



Mnemonics:
VME_TS_BYTE
VME_TS_WORD

Operation

The VME Test and Set (VME_TS) instruction handles semaphores on the VME bus. VME_TS exchanges a Boolean ON (1) for the value currently at the semaphore location. If that value was already ON, then VME_TS does not obtain the semaphore. If the existing value was OFF, then the semaphore is reset and VME_TS instruction has the semaphore and the use of the memory area it controls. The semaphore is cleared using the VME_WRT instruction to write a 0 to the semaphore location.

Note: Using a VME instruction (VME_RD, VME_WRT, VME_RMW, or VME_TS) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources:

- For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board.
- Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When VME_TS receives power flow, it accesses the VME module at the address specified by ADR and the address modifier AM. VME_TS then exchanges a Boolean ON with the data at the address. VME_TS sets the Q output to ON if the semaphore was available (OFF) and was acquired. VME_TS passes power flow to the right whenever it receives power and no error occurs during execution.

Operands

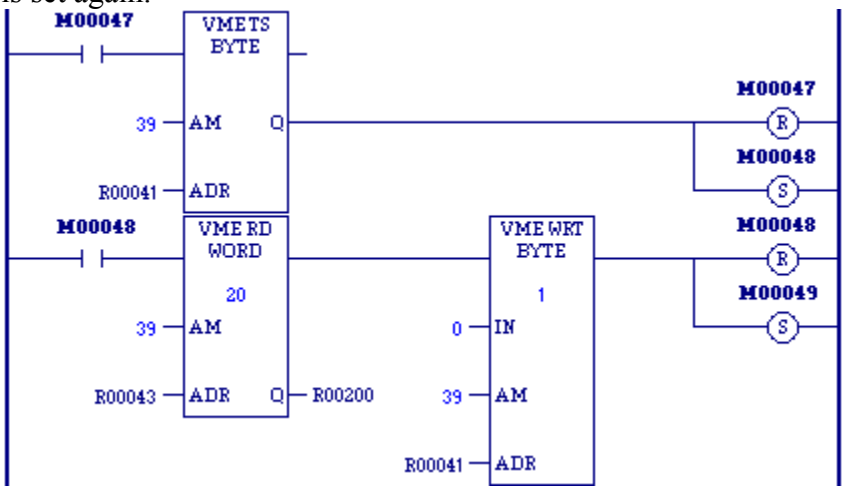
Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
AM	WORD variable or constant	data flow, R, P, L, AI, AQ	Address modifier of semaphore
ADR	DWORD constant or variable	data flow, R, P, L, AI, AQ	Address of the semaphore
Q	Power flow		Set ON if the semaphore was available (OFF). Otherwise, Q is set OFF.

Example

VME_RD, VME_WRT, and VME_TS are used to read data protected by a semaphore. When enabling input M00047 is ON, VME_TS is executed to obtain the semaphore. The semaphore VME address is stored in %R00041 and %R00042 using Standard Non-Privileged Data Access. When this is successful, coil M00047 is reset and coil M00048 is set. When %M00048 is set, VME_RD reads the data (20 words of data whose VME address is stored in %R00043 and %R00044, data read into %R00200 through %R00219). When the read is successful (if it is not, something is broken or not programmed correctly), VME_WRT relinquishes the semaphore. Coil M00048 is reset when VME_WRT is successful. M00049 is set to indicate that fresh data is now available.

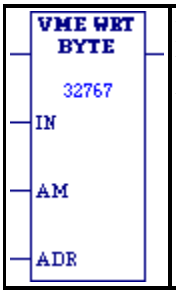
If the semaphore is not available, VME_RD and VME_WRT are not executed. The effect is that setting %M00047 causes the Controller to check the semaphore each sweep until the semaphore is available. When it becomes available, the semaphore is acquired, the data is read, and the semaphore is relinquished. No further action is taken until %M00047 is set again.



CPU Support

VME_TS_BYTE and VME_TS_WORD are supported for Series 90-70 CPUs.

VME Write



Mnemonics:
VME_WRT_BYTE
VME_WRT_WORD

Operation

The VME Write (VME_WRT) instruction writes data to the VME bus.

Note: Using a VME instruction (VME_RD, VME_WRT, VME_RMW, or VME_TS) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources:

- For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board.
- Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

When VME_WRT receives power flow, it accesses the VME module at the address specified by ADR and the address modifier AM. VME_WRT then copies the data from the input parameter IN to the VME module at the address. VME_WRT passes power to the right to indicate a successful transfer of data.

Operands

Notes

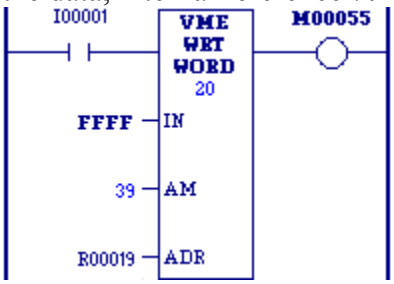
- For each mnemonic, use the corresponding data type for the IN operand. For example, VME_WRT_BYTE requires IN to be a BYTE variable.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		Length; the number of elements to write. 1 ≤ Length ≤ 32,767
IN	BYTE variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ	The data to write to the address specified by ADR and AM.
	WORD variable	data flow, R, P, L, AI, AQ	
AM	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ	Address modifier of VME operation

ADR	DWORD constant or variable	data flow, R, P, L, AI, AQ	Address where the data is to be written

Example

When enabling input I00001 is ON, the hexadecimal value FFFF is written to each of 20 words on the VME bus, the first (lowest address) being specified by the content of %R00019 (low word) and %R00020 (high word). Unless an error occurs while writing the data, internal reference %M00055 is set ON.



CPU Support

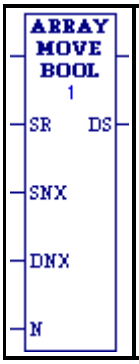
VME_WRT_BYTE and VME_WRT_WORD are supported for Series 90-70 CPUs.

Data Table Instructions

Instruction	*Mnemonic*	*Description*
Array Move	ARRAY_MOVE_BOOL ARRAY_MOVE_BYTE ARRAY_MOVE_DINT ARRAY_MOVE_DWORD ARRAY_MOVE_INT ARRAY_MOVE_UINT ARRAY_MOVE_WORD	Copies a specified number of data elements from a source memory block to a destination memory block. **Note:** The memory blocks do *not* need to be defined as arrays. What you must supply is a starting address and the number of contiguous registers to use for the move.
Array Range (PACSystems and Series 90-70 only)	ARRAY_RANGE_DINT ARRAY_RANGE_DWORD ARRAY_RANGE_INT ARRAY_RANGE_UINT ARRAY_RANGE_WORD	Determines if a value is between the range specified in two tables
FIFO Read (PACSystems and Series 90-70 only)	FIFO_RD_DINT FIFO_RD_DWORD FIFO_RD_INT FIFO_RD_UINT FIFO_RD_WORD	Removes the entry at the bottom of the First In First Out (FIFO) table, and decrements the pointer by one
FIFO Write (PACSystems and Series 90-70 only)	FIFO_WRT_DINT FIFO_WRT_DWORD FIFO_WRT_INT FIFO_WRT_UINT FIFO_WRT_WORD	Increments the table pointer and writes data to the bottom of the FIFO table
LIFO Read (PACSystems and Series 90-70 only)	LIFO_RD_DINT LIFO_RD_DWORD LIFO_RD_INT LIFO_RD_UINT LIFO_RD_WORD	Removes the entry at the pointer location in the LIFO (Last In First Out) table, and decrements the pointer by one
LIFO Write (PACSystems and Series 90-70 only)	LIFO_WRT_DINT LIFO_WRT_DWORD LIFO_WRT_INT LIFO_WRT_UINT LIFO_WRT_WORD	Increments the LIFO table's pointer and writes data to the table
Search	SEARCH_EQ_BYTE SEARCH_EQ_DINT SEARCH_EQ_DWORD SEARCH_EQ_INT SEARCH_EQ_UINT SEARCH_EQ_WORD	Searches for all array values equal to a specified value

	SEARCH_GE_BYTE SEARCH_GE_DINT SEARCH_GE_DWORD SEARCH_GE_INT SEARCH_GE_UINT SEARCH_GE_WORD	Searches for all array values greater than or equal to a specified value
	SEARCH_GT_BYTE SEARCH_GT_DINT SEARCH_GT_DWORD SEARCH_GT_INT SEARCH_GT_UINT SEARCH_GT_WORD	Searches for all array values greater than a specified value
	SEARCH_LE_BYTE SEARCH_LE_DINT SEARCH_LE_DWORD SEARCH_LE_INT SEARCH_LE_UINT SEARCH_LE_WORD	Searches for all array values less than or equal to a specified value
	SEARCH_LT_BYTE SEARCH_LT_DINT SEARCH_LT_DWORD SEARCH_LT_INT SEARCH_LT_UINT SEARCH_LT_WORD	Searches for all array values less than a specified value
	SEARCH_NE_BYTE SEARCH_NE_DINT SEARCH_NE_DWORD SEARCH_NE_INT SEARCH_NE_UINT SEARCH_NE_WORD	Searches for all array values not equal to a specified value
Sort (PACSystems and Series 90-70 only)	SORT_INT SORT_UINT SORT_WORD	Sorts a memory block in ascending order
Table Read (PACSystems and Series 90-70 only)	TBL_RD_DINT TBL_RD_DWORD TBL_RD_INT TBL_RD_UINT TBL_RD_WORD	Copies a value from a specified table location to an output reference
Table Write (PACSystems and Series 90-70 only)	TBL_WRT_DINT TBL_WRT_DWORD TBL_WRT_INT TBL_WRT_UINT TBL_WRT_WORD	Copies a value from an input reference to a specified table location

Array Move



Mnemonics:
ARRAY_MOVE_BOOL
ARRAY_MOVE_BYTE
ARRAY_MOVE_DINT
ARRAY_MOVE_DWORD
ARRAY_MOVE_INT
ARRAY_MOVE_UINT
ARRAY_MOVE_WORD

Operation

When the Array Move instruction receives power flow, it copies a specified number of elements from a source memory block to a destination memory block. Starting at the indexed location (SR+SNX-1) of the input memory block, it copies N elements to the output memory block, starting at the indexed location (DS+DNX-1) of the output memory block.

Note: For ARRAY_MOVE_BOOL, when 16-bit registers are selected for the operands of the source memory block and/or destination memory block starting address, the least significant bit of the specified 16-bit register is the first bit of the memory block. The value displayed contains 16 bits, regardless of the length of the memory block.

The indices in an Array Move instruction are 1-based. In using an Array Move, no element outside either the source or destination memory blocks (as specified by their starting address and length) can be referenced.

The instruction passes power flow unless one of the following conditions occurs:

- It receives no power flow.
- (N + SNX - 1) is greater than the Length operand.
- (N + DNX - 1) is greater than the Length operand.

Operands

Notes

- For each mnemonic, use the corresponding data type for the SR and DS operands. For example, ARRAY_MOVE_DINT expects SR and DS to be DINT variables.
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		The length of each memory block (source and destination); the number of elements in each memory block. 1 ≤ Length ≤ 32,767. Default:

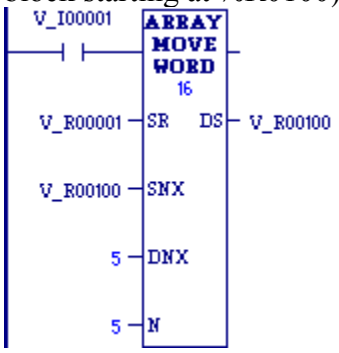
			1.
SR (Must be the same data type as DS.)	BOOL or WORD variable, bit reference in non-BOOL variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	The starting reference address of the source memory block for ARRAY_MOVE_BOOL. ▪ A BOOL variable or a bit reference in a non-BOOL variable does not need to be byte-aligned. ▪ When you select a WORD variable, the LD editor displays the value of the sixteen bits that begin at the starting reference address, regardless of the number of bits that you want to move (operand N) or the index of the first bit that you want to move (operand SNX). ▪ (PACSystems.) When you select a bit reference in a non-BOOL symbolic variable or I/O variable, ensure that the value SR+SNX-1+N does not exceed the total number of bits available in the symbolic variable or I/O variable. For example, if MySymbolic is a 16-bit variable, then the combination of values SR = MySymbolic.X[10], SNX=1, and N=7 is invalid because 10+1-1+7=17, but if MySymbolic is a symbolic array of length 2, its total number of bits is 32 and the combination of values SR = MySymbolic[0].X[10], SNX=1, and N=7 is valid.
	BYTE or WORD variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	The starting reference address of the source memory block for the mnemonics other than ARRAY_MOVE_BOOL
	DINT variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	

	DWORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	
SNX	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ	The index of the source memory block
DNX	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The index of the destination memory block
N	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The number of elements copied from the source memory block to the destination memory block
DS (Must be the same data type as SR.)	BOOL or WORD variable, bit reference in non-BOOL variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ PACSystems also supports W, symbolic, I/O variable.	The starting reference address of the destination memory block for ARRAY_MOVE_BOOL. ▪ A BOOL variable or a bit reference in a non-BOOL variable does not need to be byte-aligned. ▪ When you select a WORD variable, the LD editor displays the value of the sixteen bits that begin at the starting reference address, regardless of the number of bits that are moved (operand N) or the destination index of the first bit that is moved (operand DNX).

	BYTE or WORD variable	data flow, I, Q, M, T, G, SA, SB, SC, R, P, L, W, AI, AQ, symbolic, I/O variable	The starting address of the destination memory block for the mnemonics other than ARRAY_MOVE_BOOL
	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	

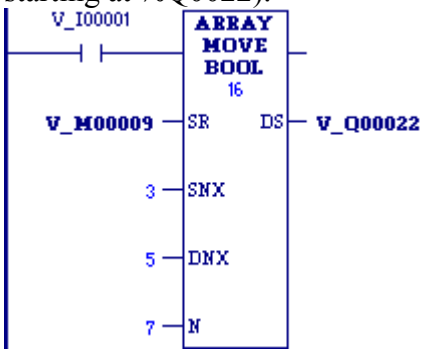
Example 1

To define the input memory block %R0001 - %R0016 and the output memory block %R0100 - %R0115, SR is set as %R0001, DS is set as %R0100, and Length is set to 16. To copy the five registers %R0003 - %R0007 to the registers %R0104 - %R0108, N is set to 5, SNX=%R0100 is set to 3 (to designate the third register, %R0003, of the block starting at %R0001), and DNX is set to 5 (to designate the fifth register, %R0104, of the block starting at %R0100).



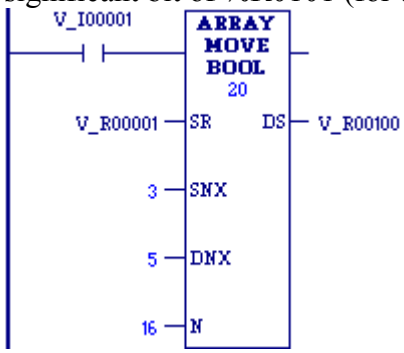
Example 2

Using bit memory blocks, the input block starts at SR=%M0009, the output block starts at %Q0022, and the length of both blocks is 16 one-bit registers (Length=16).
To copy the seven registers %M0011 - %M0017 to %Q0026 - %Q0032, N is set to 7, SNX is set to 3 (to designate the third register, %M0011, of the block starting at %M0009), and DNX is set to 5 (to designate the fifth register, %Q0026, of the block starting at %Q0022).



Example 3

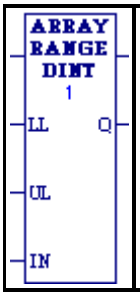
Sixteen (=N) bits that are not byte-aligned are moved from the two 16-bit registers that start at %R00001 (SR) to the two 16-bit registers that begin at %R00100 (DS). For the purposes of this boolean move, the length is set to 20, because the other 12 bits in either memory block are not considered.
By setting SNX to 3, N to 16, and DNX to 5, the third (SNX) least significant bit of %R0001 through the second least significant bit of %R0002 (for a total of 16 bits=N) are written into the fifth (DNX) least significant bit of %R0100 through the fourth least significant bit of %R0101 (for the same total of 16 bits).



CPU Support

ARRAY_MOVE_BOOL, ARRAY_MOVE_BYTE, ARRAY_MOVE_DINT, ARRAY_MOVE_INT, and ARRAY_MOVE_WORD are supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 4.00 or later CPUs, and Series 90-30 CPUs.
ARRAY_MOVE_DWORD and ARRAY_MOVE_UINT are supported for Series 90-70 Version 4.00 or later CPUs.

Array Range



Mnemonics:
ARRAY_RANGE_DINT
ARRAY_RANGE_DWORD
ARRAY_RANGE_INT
ARRAY_RANGE_UINT
ARRAY_RANGE_WORD

Operation

The ARRAY_RANGE instruction compares a single input value against two arrays of delimiters that specify an upper and lower bound to determine if the input value falls within the range specified by the delimiters. The output is an array of bits that is set ON (1) when the input value is greater than or equal to the lower limit and less than or equal to the upper limit. The output is set OFF (0) when the input is outside this range or when the range is invalid, as when the lower limit exceeds the upper limit.
When ARRAY_RANGE receives power, it compares the value in input parameter IN against each range specified by the array element values of LL and UL. Output Q sets a bit ON (1) for each corresponding array element where the value of IN is greater than or equal to the value of LL and is less than or equal to the value of UL. Output Q sets a bit OFF (0) for each corresponding array element where the value of IN is not within this range or when the range is invalid, as when the value of LL exceeds the value of UL. If the operation is successful, ARRAY_RANGE passes power flow to the right.

Operands

Notes

- For each mnemonic, use the corresponding data type for the LL, UL, and Q operands. For example, ARRAY_RANGE_DINT requires LL, UL, and Q to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of elements in each array. Default: 1.
LL (Must be the same data type as UL	DINT variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The lower limit of the range

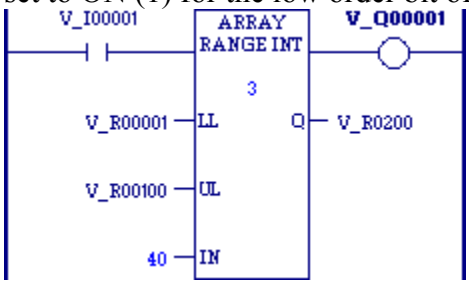
and IN.)	DWORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	
UL (Must be the same data type as LL and IN.)	DINT variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The upper limit of the range
	DWORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	
IN (Must be the same data type as LL and UL.)	DINT variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The value to compare against each range specified by LL and UL
	DWORD variable	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	

Q	BOOL, DINT, DWORD, INT, REAL, UINT, or WORD variable	PACSystems also supports data flow.	Set to ON (1) when the value in IN is within the range specified by LL and UL, inclusive. **Note:** Q is not aligned. It is displayed in bit format. It displays either a 1 (ON) or a 0 (OFF) for the first array element. For BOOL references, it represents the reference displayed. For other references, it represents the low order bit of the reference displayed.

Examples

Example 1

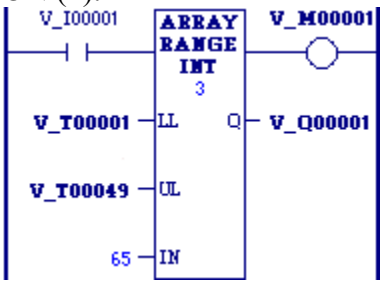
The lower limit (LL) values of %R00001 through %R00008 are 1, 20, 30, 100, 25, 50, 10, and 200. The upper limit (UL) values of %R00100 through %R00108 are 40, 50, 150, 2, 45, 90, 250, and 47. The resulting Q values are placed in the first 8 bits of %R00200. The bit values low order to high are: 1, 1, 1, 0, 1, 0, 1, and 0. The bit value displayed is set to ON (1) for the low order bit of %R00200. The output is set to ON (1).



Example 2

The lower limit (LL) array contains %T00001 through %T00016, %T00017 through %T00032, and %T00033 through %T00048. The lower limit values are 100, 65, and 1. The upper limit (UL) array contains %T00049 through %T00064, %T00065 through %T00080, and %T00081 through %T00096. The upper limit (UL) values are 29, 165, and 2.

The resulting Q values of 0, 1, and 0 are placed in %Q00001 through %Q00003. The first bit value is set to 0 (OFF), representing the value of %Q00001. The power output is set to ON (1).



CPU Support

All mnemonics of ARRAY_RANGE are supported for PACSystems CPUs and Series 90-70 Version 5.00 or later CPUs.

FIFO Read

Mnemonics:
FIFO_RD_DINT
FIFO_RD_DWORD
FIFO_RD_INT
FIFO_RD_UINT
FIFO_RD_WORD

Operation

The First-In-First-Out (FIFO) Read (FIFO_RD) instruction moves data out of tables. Values are always moved out of the bottom of the table. If the pointer reaches the last location and the table becomes full, FIFO_RD must be used to remove the entry at the pointer location and decrement the pointer by one. FIFO_RD is used in conjunction with the FIFO_WRT instruction, which increments the pointer and writes entries into the table.

1. FIFO_RD copies the top location (entry 0) of the table to output parameter Q. Additional program logic must then be used to place the data in the input reference.
2. The remaining items in the table are copied to a lower numbered position in the table.
3. FIFO_RD decrements the pointer by one.
4. Steps 1, 2, and 3 are repeated each time FIFO_RD is executed, until the table is empty (PTR = 0).

The pointer does not wrap around when the table is full.
When FIFO_RD receives power flow, the data at the first location of the table is copied to output Q. Next, each item in the table is moved down to the next lower location. This begins with item 2 in the table, which is moved into position 1. Finally, the pointer is decremented. If this causes the pointer location to become 0, the output EM is set ON, that is, EM indicates whether or not the table is empty.
FIFO_RD passes power to the right if the pointer is greater than zero and less than the value specified for LEN.

Note: A FIFO table is a queue. A LIFO table is a stack.

Operands

Notes

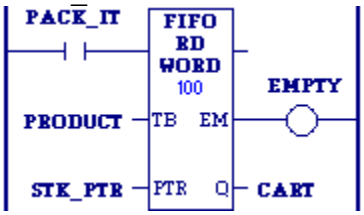
- For each mnemonic, use the corresponding data type for the TB and Q operands. For example, FIFO_RD_DINT requires TB and Q to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*

Length	constant		Length. 1 ≤ Length ≤ 32,767. Default: 1.
TB (Must be the same data type as Q.)	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the FIFO table
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the last element of the FIFO table.
EM	power flow		Energized when the last element of the table is read
Q (Must be the same data type as TB.)	DINT or DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element read from the FIFO table
	INT, UINT, or WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

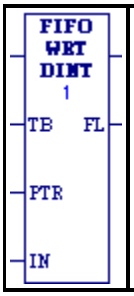
PRODUCT is a FIFO table with 100 word-sized elements. When the enabling input PACK_IT is ON, the PRODUCT data item in the table location pointed to by STK_PTR is copied to the reference location specified in CART. This table location pointed to would be the bottom, or oldest data item in the table. The number in STK_PTR is then decremented. A copy of the oldest data item in the PRODUCT table is left behind in each table location as the current data is copied out during successive PACK_IT triggers. Output node EM passes power when the PTR = 0, firing the coil EMPTY. No further data from the PRODUCT table can be read without first copying data in using the FIFO_WRT instruction.



CPU Support

All mnemonics of FIFO_RD are supported for PACSystems CPUs and Series 90-70 CPUs.

FIFO Write



Mnemonics:
FIFO_WRT_DINT
FIFO_WRT_DWORD
FIFO_WRT_INT
FIFO_WRT_UINT
FIFO_WRT_WORD

Operation

The First-In-First-Out (FIFO) Write (FIFO_WRT) instruction increments the table pointer by one and adds an entry at the new pointer location in a FIFO table. Values are always moved in at the bottom of the table. If the pointer reaches the last location and the table becomes full, FIFO_WRT can add no further values. The FIFO_RD instruction must then be used to remove the entry at the pointer location and decrement the pointer by one.

1. FIFO_WRT increments the pointer by one.
2. FIFO_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional program logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time FIFO_WRT is executed, until the table is full (PTR = 0).

The pointer does not wrap around when the table is full.
When FIFO_WRT receives power flow, the pointer is incremented by 1. Then, input data is written into the table at the pointer location. If the pointer was already at the last location in the table, no data is written and FIFO_WRT does not pass power to the right. The pointer always indicates the last item entered into the table. If the table becomes full, it is not possible to add more entries to it.
FIFO_WRT passes power to the right after a successful execution (PTR < LEN).

Note: A FIFO table is a queue. A LIFO table is a stack.

Operands

Notes

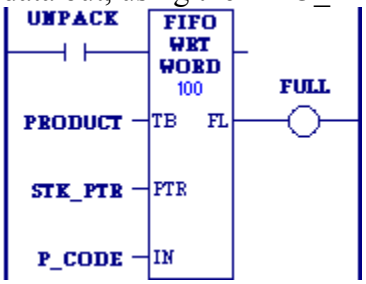
- For each mnemonic, use the corresponding data type for the TB and IN operands. For example, FIFO_WRT_DINT requires TB and IN to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*

Length	constant		Length. 1 ≤ Length ≤ 32,767. Default: 1.
TB (Must be of the same data type as IN.)	DINT or DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the FIFO table
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the last element of the FIFO table.
IN (Must be of the same data type as TB.)	DINT variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element to write to the FIFO table
	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
FL	power flow		Energized when IN is written to the last element of the table

Example

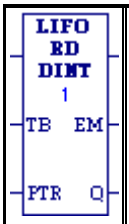
PRODUCT is a FIFO table with 100 word-sized elements. When the enabling input UNPACK is ON, a data item from P_CODE is copied to the table location pointed to by the value in STK_PTR. Output node FL passes power when PTR = LEN, firing the FULL coil. No further data from P_CODE can be added to the table without first copying data out, using the FIFO_RD instruction.



CPU Support

All mnemonics of FIFO_WRT are supported for PACSystems CPUs and Series 90-70 CPUs.

LIFO Read



Mnemonics:
LIFO_RD_DINT
LIFO_RD_DWORD
LIFO_RD_INT
LIFO_RD_UINT
LIFO_RD_WORD

Operation

The Last-In-First-Out (LIFO) Read (LIFO_RD) instruction moves data out of tables. Values are always moved out of the top of the table. If the pointer reaches the last location and the table becomes full, LIFO_RD must be used to remove the entry at the pointer location and decrement the pointer by one. LIFO_RD is used in conjunction with the LIFO_WRT instruction, which increments the pointer and writes entries into the table.

1. LIFO_RD copies data indicated by the pointer to output parameter Q. Additional logic must then be used to place the data in the input reference.
2. LIFO_RD decrements the pointer by one.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the table is empty (PTR = LEN).

The pointer does not wrap around when the table is full.
When LIFO_RD receives power flow, the data at the pointer location is copied to output Q, then the pointer is decremented. If this causes the pointer location to become 0, the output EM is set ON, that is, EM indicates whether or not the table is empty. If the table is empty when LIFO_RD receives power flow, no read occurs. The pointer always indicates the last item entered into the table.
LIFO_RD passes power to the right if the pointer was in range for an element to be read.

Note: A LIFO table is a stack. A FIFO table is a queue.

Operands

Notes

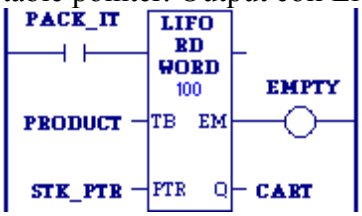
- For each mnemonic, use the corresponding data type for the TB and Q operands. For example, LIFO_RD_DINT requires TB and Q to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		1 ≤ Length ≤ 32,767. Default: 1.

TB (Must be of the same data type as Q.)	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the table
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the next element to read.
EM	power flow		Energized when the last element of the table is read
Q (Must be of the same data type as TB.)	DINT or DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element read from the table
	INT, UINT, or WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

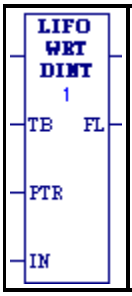
PRODUCT is a LIFO table with 100 word-sized elements. When the enabling input PACK_IT is ON, the data item at the top of the table is copied into the reference indicated by the nickname CART. The reference identified by STK_PTR contains the table pointer. Output coil EMPTY indicates when the table is empty.



CPU Support

All mnemonics of LIFO_RD are supported for PACSystems CPUs and Series 90-70 CPUs.

LIFO Write



Mnemonics:
LIFO_WRT_DINT
LIFO_WRT_DWORD
LIFO_WRT_INT
LIFO_WRT_UINT
LIFO_WRT_WORD

Operation

The Last-In-First-Out (LIFO) Write (LIFO_WRT) instruction increments the table pointer by one and then adds an entry at the new pointer location in a table. Values are always moved in at the top of the table. If the pointer reaches the last location and the table becomes full, LIFO_WRT cannot add further values. LIFO_RD must then be used to remove the entry at the pointer location and decrement the pointer by one.

1. LIFO_WRT increments the table pointer by one.
2. LIFO_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time LIFO_WRT is executed, until the table is full (PTR = LEN).

The pointer does not wrap around when the table is full.
When LIFO_WRT receives power flow, the pointer increments by 1; then the new data is written at the pointer location. If the pointer was already at the last location in the table, no data is written and LIFO_WRT does not pass power to the right. The pointer always indicates the last item entered into the table. If the table is full, it is not possible to add more entries to it.
LIFO_WRT passes power to the right after a successful execution (PTR < LEN).

Note: A LIFO table is a stack. A FIFO table is a queue.

Operands

Notes

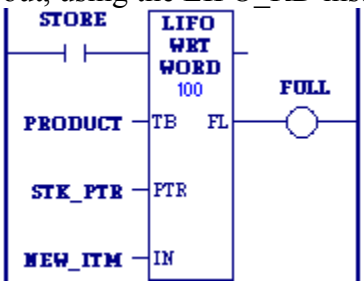
- For each mnemonic, use the corresponding data type for the TB and IN operands. For example, LIFO_WRT_DINT requires TB and Q to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		1 ≤ Length ≤ 32,767. Default: 1.

TB (Must be the same data type as IN.)	DINT or DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the table
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the next element to write.
IN (Must be of the same data type as TB.)	DINT variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element to write to the table
	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
FL	power flow		Energized when IN is written to the last element of the table

Example

PRODUCT is a LIFO table with 100 word-sized elements. When the enabling input STORE is ON, a data item from NEW_ITEM is copied to the table location pointed to by the value in STK_PTR. Output FL passes power when PTR = LEN, firing the FULL coil. No further data from NEW_ITEM can be added to the table without first copying data out, using the LIFO_RD instruction.



CPU Support

All mnemonics of LIFO_WRT are supported for PACSystems CPUs and Series 90-70 CPUs.

Search

Mnemonics

The Search instruction supports up to thirty-six variations depending on the CPU you have, not only the mnemonic depicted above for illustrative purposes.

Operation

When the Search instruction receives power, it searches the specified memory block for a value that satisfies the search criteria. For example, SEARCH_GE_DWORD searches for a DWORD that is greater than or equal to the specified value (the IN operand).
Search can evaluate six different relationships for up to six data types, for a total of up to thirty-six mnemonics.

Relationships:

- SEARCH_EQ_... searches for a value of data type ... equal to the IN operand.
- SEARCH_GE_... searches for a value of data type ... greater than or equal to IN.
- SEARCH_GT_... searches for a value of data type ... greater than IN.
- SEARCH_LE_... searches for a value of data type ... less than or equal to IN.
- SEARCH_LT_... searches for a value of data type ... less than IN.
- SEARCH_NE_... searches for a value of data type ... that is not equal to IN.

Data types:

- BYTE
- DINT
- DWORD
- INT
- UINT
- WORD

Searching begins at AR+INX, where AR is the starting address and INX is the index value into the memory block. The search continues either until a register that satisfies the search criteria is found or until the end of the memory block is reached.

- If a register is found, the Found Indication (FD) is set ON and the Output Index (ONX) is set to the relative position of this register within the block.
- If no register is found before the end of the block is reached, the Found Indication (FD) is set OFF and the Output Index (ONX) is set to zero.

The input index (INX) is zero-based, that is, 0 the means first reference, whereas the output index (ONX) is one-based, that is, 1 means the first reference.
The valid values for INX are 0 through (the value represented by the operand Length - 1).
The valid values for ONX are 1 through Length.
INX should be set to zero to begin searching at the memory block's first register. This value increments by one at the time of execution. If the value of input INX is out of -range, (< 0 or > [Length - 1]), INX is set to the default value of zero.
SEARCH passes power flow to the right when it performs without error. If INX is out of range, SEARCH does not pass power flow to the right.

Operands

Notes

- For each mnemonic, use the corresponding data type for the AR and IN operands. For example, SEARCH_EQ_BYTE requires AR and IN to be BYTE variables.
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

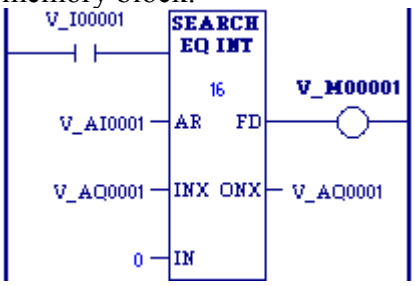
Operand	*Data Type*	*Memory Area*	*Description*
Length	Constant		The number of registers starting at AR that make up the memory block to search. 1 ≤ Length ≤ 32,767 8-bit or 16-bit registers.
AR (Must be the same data type as IN.)	BYTE or WORD	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	The starting address of the memory block to search; the address of the first register in the memory block.
	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

INX	WORD variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The zero-based index into the memory block at which to begin the search. Zero points to the first reference. Valid range: 0 ≤ INX ≤ (Length - 1). If INX is out of range, it is set to the default value of 0.
IN (Must be the same data type as AR.)	BYTE or WORD variable or constant	data flow, I, Q, M, T, G, S, R, P, L, AI, AQ, W, symbolic, I/O variable	The value that the search is based on. For example: ▪ SEARCH_GT_DINT searches for a DINT value that is greater than IN. ▪ SEARCH_NE_UINT searches for a UINT value that is not equal to IN. ▪ SEARCH_GE_WORD searches for a WORD value that is greater than or equal to IN.
	DINT variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	
	DWORD variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
ONX	WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The one-based position within the memory block of the search target. A value of 1 points to the first reference. Valid range: 1 ≤ ONX ≤ Length
FD	Power flow		Found indicator. This power flow indicator is energized when a register that satisfies the search criteria is found and the instruction was successful.

Example

To search the memory block %AI001 - %AI016, AR is set as %AI001 and Length is set as 16. The values of the 16 registers are 100, 20, 0, 5, 90, 200, 0, 79, 102, 80, 24, 34, 987, 8, 0, and 500. Initially, the search index into AR, %AQ0001, is 5. When power flow input is ON, each scan searches the memory block looking for a match to the IN value of 0. The first scan starts searching at %AI0006 and finds a match at %AI0007, so FD turns

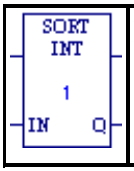
ON and %AQ0001 becomes 7. The second scan starts searching at %AI0008 and finds a match at %AI0015, so FD remains ON and %AQ0001 becomes 15. The next scan starts at %AI0016. Since the end of the memory block is reached without a match, FD is set OFF and %AQ0001 is set to zero. The next scan starts searching at the beginning of the memory block.



CPU Support

All the SEARCH_??_DWORD and SEARCH_??_UINT mnemonics are supported for PACSystems CPUs and Series 90-70 CPUs.
All other mnemonics of the Search instruction are supported for all GE IP CPUs.

Sort



Mnemonics:
SORT_INT
SORT_UINT
SORT_WORD

Operation

When it receives power flow, the SORT instruction sorts the elements of the memory block 'IN' in ascending order. The output memory block Q contains integers that gives the index that the sorted elements had in the original memory block or list. Q is exactly the same size as IN. It also has a specification (LEN) of the number of elements to be sorted. SORT operates on memory blocks of no more than 64 elements. When EN is ON, all of the elements of IN are sorted into ascending order, based on their data type. The array Q is also created, giving the original position that each sorted element held in the unsorted array. OK is always set ON.

Note: Do not use the SORT instruction in a timed or triggered input block.

Operands

Notes

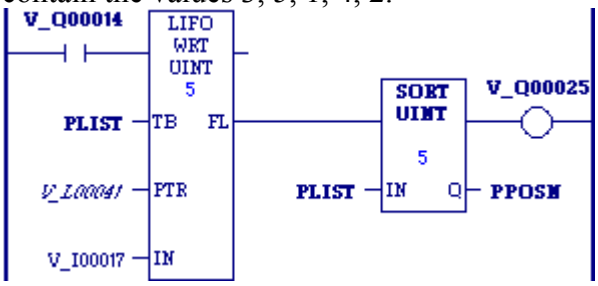
- For each mnemonic, use the corresponding data type for the IN and Q operands. For example, SORT_INT requires IN and Q to be INT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		The number (1 - 64) of elements that make up the memory block to sort
IN	INT, UINT, or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The memory block that contains the elements to sort. After the sort, IN contains the elements in the sorted order.
Q (Must be the same data type as IN)	INT, UINT, or WORD variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	An array of indexes that gives the position of the sorted elements in the original memory block

Example

New part numbers (%I00017 - %I00032) are pushed onto a parts array PLIST every time %Q00014 is ON. When the array is filled, it is sorted and the output %Q00025 is turned

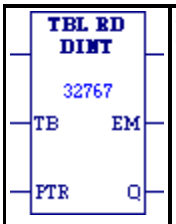
on. The array PPOSN then contains the original position that the now-sorted elements held before the sort was done on PLIST.
If PLIST was an array of five elements and contained the values 25, 67, 12, 35, 14 before the sort, then after the sort it would contain the values 12, 14, 25, 35, 67. PPOSN would contain the values 3, 5, 1, 4, 2.



CPU Support

SORT_INT, SORT_UINT, and SORT_WORD are supported for PACSystems CPUs and Series 90-70 CPUs.

Table Read



Mnemonics:
TBL_RD_DINT
TBL_RD_DWORD
TBL_RD_INT
TBL_RD_UINT
TBL_RD_WORD

Operation

The Table Read (TBL_RD) instruction sequentially reads values in a table that never becomes full. When the pointer reaches the end of the table, it automatically wraps around to the beginning of the table.

1. TBL_RD increments the pointer by one.
2. TBL_RD copies data indicated by the pointer to output parameter Q. Additional logic must then be used to capture the data from the output reference.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the table is empty (PTR = LEN).

Note: The TBL_RD and TBL_WRT instructions can operate on the same or different tables. By specifying a different reference for the pointer, these instructions can access the same data table at different locations or at different rates.

When TBL_RD receives power flow, the pointer (PTR) increments by one. If this new pointer location is the last item in the table, the output EM is set to ON. The next time TBL_RD executes, PTR is automatically set back to 1. After PTR is incremented, the content at the new pointer location is copied to output Q.
TBL_RD always passes power to the right when it receives power.

Note: TBL_RD is like FIFO_RD with a wrap-around.

Operands

Notes

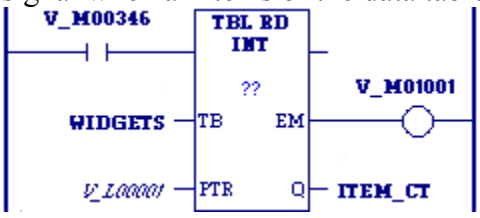
- For each mnemonic, use the corresponding data type for the TB and Q operands. For example, TBL_RD_DINT requires TB and Q to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		Length. 1 ≤ Length ≤ 32,767.
TB (Must be of the same data type as Q.)	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the table

Q.)	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the next element.
EM	power flow		Energized when the last element of the table is read.
Q (Must be of the same data type as TB.)	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element read from the table
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	

Example

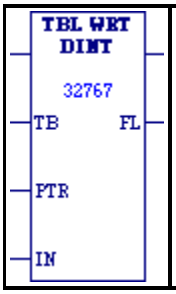
WIDGETS is a table with 20 integer elements. When the enabling input %M00346 is ON, the pointer increments and the contents of the next element of the table is copied into ITEM_CT. %L00001 instructions as the pointer into the data table. %M01001 is used to signal when all items of the data table have been accessed.



CPU Support

All the mnemonics of TBL_RD are supported for PACSystems CPUs and Series 90-70 CPUs.

Table Write



Mnemonics:
TBL_WRT_DINT
TBL_WRT_DWORD
TBL_WRT_INT
TBL_WRT_UINT
TBL_WRT_WORD

Operation

The Table Write (TBL_WRT) instruction sequentially updates values in a table that never becomes full. When the pointer (PTR) reaches the end of the table, it automatically returns to the beginning of the table.

1. TBL_WRT increments the pointer by one.
2. TBL_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the table is full (PTR = LEN).

When the table is full, the pointer wraps around to the beginning of the table.

Note: The TBL_WRT and TBL_RD instructions can operate on the same or different tables. By specifying a different reference for the pointer, these instructions can access the same data table at different locations or at different rates.

When TBL_WRT receives power flow, the pointer (PTR) increments by 1. If this new pointer location is the last item in the table, the output FL is set to ON. The next time TBL_WRT executes, PTR is automatically set back to 1. After incrementing PTR, TBL_WRT writes the content of the input reference to the current pointer location, overwriting data already stored there.
TBL_WRT always passes power to the right when it receives power.

Note: TBL_WRT is like FIFO_WRT with a wrap-around.

Operands

Notes

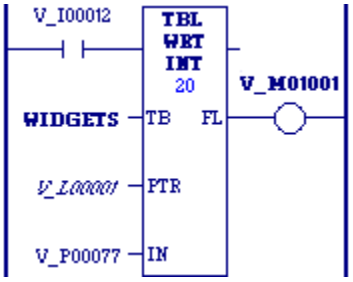
- For each mnemonic, use the corresponding data type for the TB and IN operands. For example, TBL_WRT_DINT requires TB and IN to be DINT variables.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
Length	constant		Length. 1 ≤ Length ≤

			32,767.
TB (Must be the same data type as IN.)	DINT variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The elements in the table
	DWORD variable	R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
	WORD variable	I, Q, M, T, G, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	
PTR	INT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	Pointer. Index of the next element.
IN (Must be the same data type as TB.)	DINT variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	The element to write to the table
	DWORD variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT or UINT variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
	WORD variable	data flow, I, Q, M, T, S, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
FL	power flow		Energized when IN is written to the last element of the table

Example

WIDGETS is a table with 20 integer elements. When the enabling input %I00012 is ON, the pointer increments and the contents of %P00077 are written into the table at the pointer location. %L00001 instructions as the pointer into the data table.



CPU Support

All the mnemonics of TBL_WRT are supported for PACSystems CPUs and Series 90-70 CPUs.

Math Instructions

Your logic may need to include logic to convert data to a different type before using a Math or Numerical instruction. The description of each instruction includes information about appropriate data types. The section Data Type Conversion instructions explains how to convert data to a different type.

Instruction	*Mnemonic*	*Description*
Absolute Value (PACSystems and 90-70 only)	ABS_DINT ABS_INT ABS_REAL	Finds the absolute value of a double- precision integer (DINT), signed single-precision integer (INT), or REAL (floating-point) value. The mnemonic specifies the value's data type.
Add	ADD_DINT ADD_INT ADD_REAL ADD_UINT	Addition. Adds two numbers.
Divide	DIV_DINT DIV_INT DIV_MIXED DIV_REAL DIV_UINT	Division. Divides one number by another and outputs the quotient. **Note:** Take care to avoid overflow conditions when performing divisions.
Modulus	MOD_DINT MOD_INT MOD_UINT	Modulo Division. Divides one number by another and outputs the remainder.
Multiply	MUL_DINT MUL_INT MUL_MIXED MUL_REAL MUL_UINT	Multiplication. Multiplies two numbers. **Note:** Take care to avoid overflow conditions when performing multiplications.
Scale (VersaMax and PACSystems only)	SCALE_DINT SCALE_INT SCALE_UINT SCALE_WORD	Scaling. Scales an input parameter and places the result in an output location.
Subtract	SUB_DINT SUB_INT SUB_REAL SUB_UINT	Subtraction. Subtracts one number from another.

Avoiding Overflows

Be careful to avoid overflows when using Multiplication and Division instructions.
If you have to convert INT to DINT values, remember that the CPU uses standard 2's complement with the sign extended to the highest bit. You must check the sign of the low 16 bits and extend it into the second 16 bits. If the most significant bit in an INT is 0

(positive), move a 0 to all 16 high bits. If the most significant bit in an INT is 1 (negative), move a -1 or hex 0FFFFh to the 16 high bits.
Converting from DINT to INT data is easier, because the low 16 bits (first register) is the integer portion of a DINT (32-bit). The upper 16 bits should be either a 0 (positive) or -1 (negative) value or the DINT number will be too large to convert to 16 bits.

Absolute Value

Mnemonics:
ABS_DINT
ABS_INT
ABS_LREAL
ABS_REAL

Operation

When the instruction receives power flow, it places the absolute value of input IN in output Q. IN and Q must be of the same data type.
The instruction outputs power flow, unless one of the following conditions occurs:

- For INT type, IN is MININT.
- For DINT type, IN is MINDINT.
- For LREAL or REAL type, IN is NaN (Not a Number).

Operands

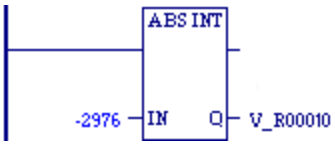
Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN (Must be of the same data type as Q.)	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to process
	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	
Q (Must be of the same data type as IN.)	INT variable	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The absolute value of IN
	DINT, LREAL, or REAL variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, W, symbolic, I/O variable.	

Example

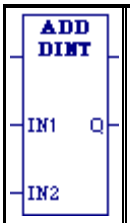
Placing the absolute value of –2,976, which is 2,976, in %R00010:



CPU Support

ABS_DINT and ABS_INT are supported for PACSystems CPUs and Series 90-70 Version 3.00 or later CPUs.
ABS_LREAL is supported for PACSystems firmware version 5.50 or later.
ABS_REAL is supported for PACSystems CPUs and Series 90-70 Version 3.00 or later floating-point CPUs.

Add



Mnemonics:
ADD_DINT
ADD_INT
ADD_LREAL
ADD_REAL
ADD_UINT

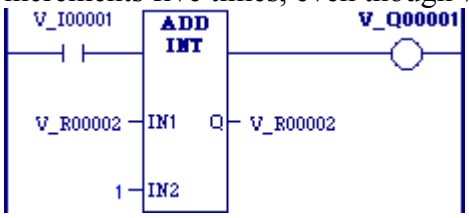
Operands and CPU support: ADD_DINT | ADD_INT | ADD_LREAL | ADD_REAL | ADD_UINT

Operation

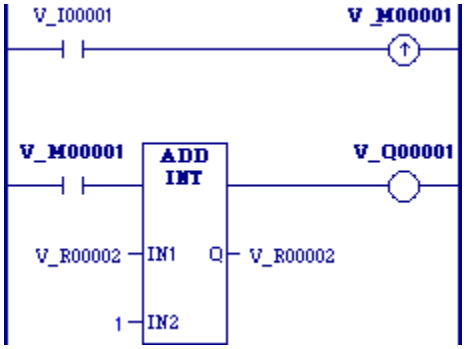
When the ADD instruction receives power flow, it adds the two operands IN1 and IN2 of the same data type and stores the sum in the output variable assigned to Q, also of the same data type.
The power flow output is energized when ADD is performed without overflow, unless an invalid operation occurs. If an overflow occurs, the result is the largest possible value with the proper sign and no power flow.

Examples

The first example is a failed attempt to create a counter circuit that would count the number of times switch %I0001 closes. The running total is stored in register %R0002. The intent of this design is that when %I0001 closes, the ADD instruction should add one to the value in %R0002 and place the new value right back into %R0002. The problem with this design is that the ADD instruction executes once every Controller scan while %I0001 is closed. So, for example, if %I0001 stays closed for five scans, the output increments five times, even though %I0001 only closed once during that period.



To correct the above problem, the enable input to the ADD instruction should come from a transition ("one-shot") coil, as shown below. In the improved circuit, the %I0001 input switch controls a transition ("one-shot") coil, %M0001, whose contact turns on the enable input of the ADD instruction for only one scan each time contact %I0001 closes. In order for the %M0001 contact to close again, contact %I0001 has to open and close again.



ADD_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to the left of the plus sign (+) in the equation IN1+IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT constants are limited to values between -32,768 and +32,767.
IN2	DINT variable or constant	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to the right of the plus sign (+) in the equation IN1+IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT constants are limited to values between -32,768 and +32,767.
Q	DINT variable	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1+IN2. If an overflow occurs, the result is the largest possible value with the proper sign and no power flow.

CPU Support

ADD_DINT is supported for all GE IP CPUs.

Note: In Series 90-30 CPU341 and lower, DINT constants are limited to values between -32,768 and +32,767.

ADD_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to the left of the plus sign (+) in the equation IN1+IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to the right of the plus sign (+) in the equation IN1+IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The result of IN1+IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

ADD_INT is supported for all GE IP CPUs.

ADD_LREAL

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 64 or more instead of an LREAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	LREAL variable or constant	data flow, R, P, L, W, AI, AQ, I, Q, M, T, G, symbolic, I/O variable	The value to the left of the plus sign (+) in the equation IN1+IN2=Q.
IN2	LREAL variable or constant	data flow, R, P, L, W, AI, AQ, I, Q, M, T, G, symbolic, I/O variable	The value to the right of the plus sign (+) in the equation IN1+IN2=Q.
Q	LREAL variable	data flow, R, P, L, W, AI, AQ, I, Q, M, T, G,	The result of IN1+IN2. If an overflow occurs, the result is the largest value

		symbolic, I/O variable	with the proper sign and no power flow.

Note: If IN1 and/or IN2 is NaN (Not a Number), ADD_LREAL passes no power flow.

CPU Support

ADD_LREAL is supported for PACSystems firmware version 5.50 or later.

ADD_REAL

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a REAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	REAL variable or constant	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to the left of the plus sign (+) in the equation IN1+IN2=Q.
IN2	REAL variable or constant	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to the right of the plus sign (+) in the equation IN1+IN2=Q.
Q	REAL variable	data flow, R, P, L, W, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1+IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

Note: If IN1 and/or IN2 is NaN (Not a Number), ADD_REAL passes no power flow.

CPU Support

ADD_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

ADD_UINT

Operands

Notes

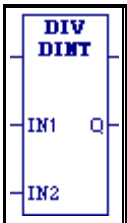
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a UINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to the left of the plus sign (+) in the equation IN1+IN2=Q.
IN2	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The value to the right of the plus sign (+) in the equation IN1+IN2=Q.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, W, AI, AQ, symbolic, I/O variable	The result of IN1+IN2. If an overflow occurs, the result is the largest possible value and no power flow.

CPU Support

ADD_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Divide



Mnemonics:
DIV_DINT
DIV_INT
DIV_LREAL
DIV_MIXED
DIV_REAL
DIV_UINT

Operands and CPU support: DIV_DINT | DIV_INT | DIV_LREAL | DIV_MIXED | DIV_REAL | DIV_UINT

Operation

When the DIV instruction receives power flow, it divides the operand IN1 by the operand IN2 of the same data type as IN1 and stores the quotient in the output variable assigned to Q, also of the same data type as IN1 and IN2.

Notes

- DIV rounds down; it does not round to the closest integer. For example, 24 DIV 5 = 4.
- (Series 90-70 CPUs.) DIV_MIXED uses mixed data types.

Caution: Avoid overflows.

The power flow output is energized when DIV is performed without overflow, unless an invalid operation occurs. If an overflow occurs, the result is the largest possible value with the proper sign and no power flow.

Example

DIV_DINT can be used in conjunction with a MUL_DINT instruction to scale a ±10 volt input to ±25,000 engineering units. More.

DIV_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, AI, AQ, W.	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.

		PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	**Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
IN2	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

DIV_DINT is supported for all GE IP CPUs.

Note: In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

DIV_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

DIV_INT is supported for all GE IP CPUs.

DIV_LREAL

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 64 or more instead of an LREAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	LREAL variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, symbolic, I/O variable	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.
IN2	LREAL variable or constant	data flow, R, P, L, AI, AQ, I, Q, M, T, G, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.
Q	LREAL variable	data flow, R, P, L, AI, AQ, I, Q, M, T, G, symbolic, I/O variable	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

DIV_LREAL is supported for PACSystems firmware version 5.50 or later.

DIV_MIXED

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ,	The quotient of IN1/IN2. If an overflow occurs, the result is the

		symbolic, I/O variable	largest value with the proper sign and no power flow.

CPU Support

DIV_MIXED is supported for PACSystems CPUs and for firmware version 3.00 or later of Series 90-70 CPUs.

DIV_REAL

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a REAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	REAL variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.
IN2	REAL variable or constant	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.
Q	REAL variable	data flow, R, P, L, AI, AQ. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

DIV_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

DIV_UINT

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a UINT variable. Restrictions apply.

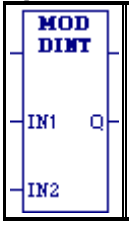
Operand	*Data Type*	*Memory Area*	*Description*
IN1	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.
IN2	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, symbolic, I/O variable	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

DIV_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Modulus

Operation



Mnemonics:
MOD_DINT
MOD_INT
MOD_UINT

Operation

When the Modulo Division (MOD) instruction receives power flow, it divides input IN1 by input IN2 and outputs the remainder of the division to Q.
All three operands must be of the same data type. The sign of the result is always the same as the sign of input parameter IN1. Output Q is calculated using the formula:

```
Q = IN1-((IN1 DIV IN2) * IN2)
```

where DIV produces an integer number.
The power flow output is always ON when the instruction receives power flow, unless there is an attempt to divide by zero. In that case, the power flow output is set to OFF.

MOD_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to be divided to obtain the remainder; the value to the left of "MOD" in the equation IN1 MOD IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
IN2	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to divide IN1 with; the value to the right of "MOD" in the equation IN1 MOD IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The remainder of IN1/IN2.

CPU Support

MOD_DINT is supported for all GE IP CPUs.

Note: In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

MOD_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to be divided to obtain the remainder; the value to the left of "MOD" in the equation IN1 MOD IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "MOD" in the equation IN1 MOD IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The quotient of IN1/IN2.

CPU Support

MOD_INT is supported for all GE IP CPUs.

MOD_UINT

Operands

Notes

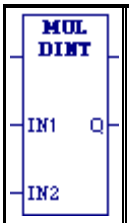
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a UINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to be divided to obtain the remainder; the value to the left of "MOD" in the equation IN1 MOD IN2=Q.
IN2	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to divide IN1 with; the value to the right of "MOD" in the equation IN1 MOD IN2=Q.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The quotient of IN1/IN2.

CPU Support

MOD_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Multiply



Mnemonics:
MUL_DINT
MUL_INT
MUL_LREAL
MUL_MIXED
MUL_REAL
MUL_UINT

Operands and CPU support: MUL_DINT | MUL_INT | MUL_LREAL | MUL_MIXED | MUL_REAL | MUL_UINT

Operation

When the MUL instruction receives power flow, it multiplies the two operands IN1 and IN2 of the same data type and stores the result in the output variable assigned to Q, also of the same data type.

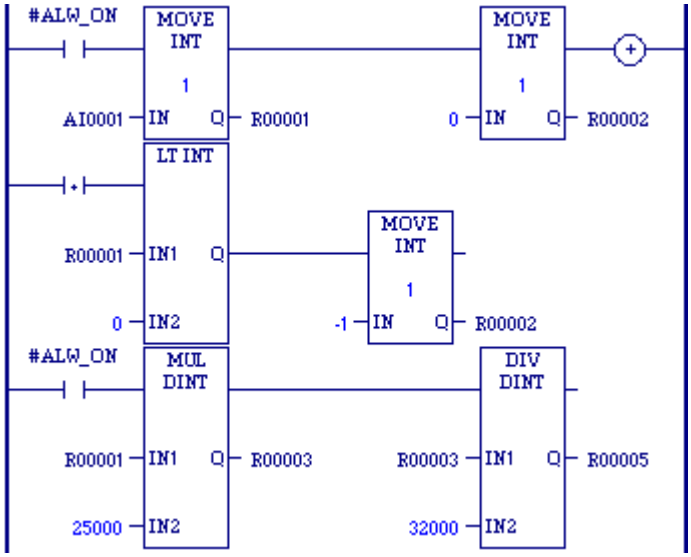
Note: (Series 90-70 CPUs.) MUL_MIXED uses mixed data types.

Caution: Avoid overflows.

The power flow output is energized when the instruction is performed without overflow, unless an invalid operation occurs. If an overflow occurs, the result is the largest possible value with the proper sign and no power flow.

Example

A common application is to scale analog input values with a MUL operation followed by a DIV and possibly an ADD operation. A 0 to ±10 volt analog input will place values of 0 to ±32,000 in its corresponding %AI input register. Multiplying this input register using an MUL_INT instruction will result in an overflow since an INT type instruction has an input and output range of 32,767 to –32,768. Using the %AI value as in input to a MUL_DINT also does not work as the 32-bit IN1 will combine 2 analog inputs at the same time. To solve this problem, you can move the analog input to the low word of a double register, then test the sign and set the second register to 0 if the sign tests positive or –1 if negative. Then use the double register just created with a MUL_DINT which gives a 32-bit result, and which can be used with a following DIV_DINT instruction. For example, the following logic could be used to scale a ±10 volt input %AI1 to ±25000 engineering units in %R5.



An alternate, but less accurate, way of programming this circuit using INT values involves placing the DIV_DINT instruction first, followed by the MUL_DINT instruction. The value of IN2 for the DIV instruction would be 32, and the value of IN2 for the MUL would be 25. This maintains the scaling proportion of the above circuit and keeps the values within the working range of the INT type instructions. However, the DIV instruction inherently discards any remainder value, so when the DIV output is multiplied by the MUL instruction, the error introduced by a discarded remainder is multiplied. The percent of error is non-linear over the full range of input values and is greater at lower input values.

By contrast, in the example above, the results are more accurate because the DIV operation is performed last, so the discarded remainder is not multiplied. If even greater precision is required, substitute REAL or LREAL math instructions in this example so that the remainder is not discarded.

MUL_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

IN2	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_DINT is supported for all GE IP CPUs.

Note: In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.

MUL_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_INT is supported for all GE IP CPUs.

MUL_LREAL

Operands

Note

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 64 or more instead of an LREAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q
IN2	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.
Q	LREAL variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_LREAL is supported for PACSystems firmware version 5.50 or later.

MUL_MIXED

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.
Q	DINT variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_MIXED is supported for PACSystems CPUs and Series 90-70 CPUs.

MUL_REAL

Operands

Note

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a REAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q
IN2	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 Version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

MUL_UINT

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a UINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*

IN1	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q.
IN2	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of IN1*IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

MUL_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Scale

Mnemonics:
SCALE_DINT
SCALE_INT
SCALE_UINT
SCALE_WORD

Operands: SCALE DINT | SCALE_INT | SCALE_UINT | SCALE_WORD

Operation

When the SCALE instruction receives power flow, it scales the input operand IN and places the result in the output variable assigned to output operand OUT. The power flow output is energized when SCALE is performed without overflow.

SCALE_DINT Operands

The operands for SCALE_DINT are the same as those for the other mnemonics, except for the data type.
All operands can map to the following memory areas: data flow, R, P, L, W, AI, AQ, I, Q, M, T, G, symbolic, and I/O variables.

Operand	*Data Type*		*Description*
IHI	DINT variable or constant		Input's HIgh; maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI and OLO to calculate the scaling factor applied to the input value IN.
ILO	DINT variable or constant		Input's LOw; minimum input value (module-related). The lower limit of the unscaled data. Must be the same data type as IHI.
OHI	DINT variable or constant		Output's HIgh; maximum output value. The upper limit of the scaled data. Must be the same data type as IHI. When the IN input is at the IHI value, the OUT value is the same as the OHI value.
OLO	DINT variable or constant		Output's LOw; minimum output value. The lower limit of the scaled data. Must be the same data type as IHI. When the IN input is at the ILO value, the OUT value is the same

			as the OLO value.
IN	DINT variable or constant		INput value. The value to be scaled. Must be the same data type as IHI.
OUT	DINT variable		OUTput value. The scaled equivalent of the input value. Must be the same data type as IHI.

SCALE_INT Operands

The operands for SCALE_INT are the same as those for the other mnemonics, except for the data type.

All operands can map to the following memory areas: R, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, P, L, W, symbolic, and I/O variables.

Operand	*Data Type*		*Description*
IHI	INT variable or constant		Input's HIgh; maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI and OLO to calculate the scaling factor applied to the input value IN.
ILO	INT variable or constant		Input's LOw; minimum input value (module-related). The lower limit of the unscaled data. Must be the same data type as IHI.
OHI	INT variable or constant		Output's HIgh; maximum output value. The upper limit of the scaled data. Must be the same data type as IHI. When the IN input is at the IHI value, the OUT value is the same as the OHI value.
OLO	INT variable or constant		Output's LOw; minimum output value. The lower limit of the scaled data. Must be the same data type as IHI. When the IN input is at the ILO value, the OUT value is the same as the OLO value.
IN	INT variable or constant		INput value. The value to be scaled. Must be the same data type as IHI.
OUT	INT variable		OUTput value. The scaled equivalent of the input value. Must be the same data type as IHI.

SCALE_UINT Operands

The operands for SCALE_UINT are the same as those for the other mnemonics, except for the data type.

All operands can map to the following memory areas: data flow, R, P, L, W, AI, AQ, I, Q, M, T, G, symbolic, and I/O variables.

Operand	*Data Type*		*Description*
IHI	UINT variable or		Input's HIgh; maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI

	constant		and OLO to calculate the scaling factor applied to the input value IN.
ILO	UINT variable or constant		Input's LOw; minimum input value (module-related). The lower limit of the unscaled data. Must be the same data type as IHI.
OHI	UINT variable or constant		Output's HIgh; maximum output value. The upper limit of the scaled data. Must be the same data type as IHI. When the IN input is at the IHI value, the OUT value is the same as the OHI value.
OLO	UINT variable or constant		Output's LOw; minimum output value. The lower limit of the scaled data. Must be the same data type as IHI. When the IN input is at the ILO value, the OUT value is the same as the OLO value.
IN	UINT variable or constant		INput value. The value to be scaled. Must be the same data type as IHI.
OUT	UINT variable		OUTput value. The scaled equivalent of the input value. Must be the same data type as IHI.

SCALE_WORD Operands

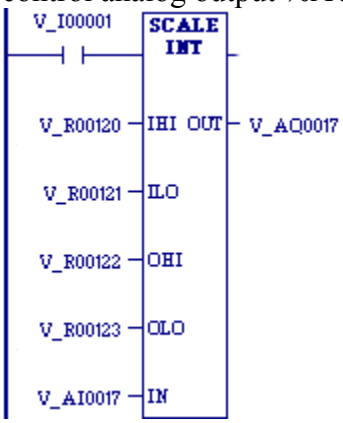
The operands for SCALE_WORD are the same as those for the other mnemonics, except for the data type.
All operands can map to the following memory areas: R, AI, AQ.

Operand	*Data Type*		*Description*
IHI	WORD variable or constant		Input's HIgh; maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI and OLO to calculate the scaling factor applied to the input value IN.
ILO	WORD variable or constant		Input's LOw; minimum input value (module-related). The lower limit of the unscaled data. Must be the same data type as IHI.
OHI	WORD variable or constant		Output's HIgh; maximum output value. The upper limit of the scaled data. Must be the same data type as IHI. When the IN input is at the IHI value, the OUT value is the same as the OHI value.
OLO	WORD variable or constant		Output's LOw; minimum output value. The lower limit of the scaled data. Must be the same data type as IHI. When the IN input is at the ILO value, the OUT value is the same as the OLO value.
IN	WORD		INput value. The value to be scaled. Must be the same data

	variable or constant		type as IHI.
OUT	WORD variable		OUTput value. The scaled equivalent of the input value. Must be the same data type as IHI.

Example

The registers %R0120 through %R0123 are used to store the high and low scaling values. The input value to be scaled is analog input %AI0017. The scaled output data is used to control analog output %AQ0017. The scaling is performed whenever %I0001 is ON.



CPU Support

SCALE_DINT and SCALE_UINT are supported only for PACSystems CPUs with firmware version 2.00 or later.
SCALE_INT is supported for VersaMax CPUs, and for PACSystems with version 2.0 or later CPUs.
SCALE_WORD is supported only for VersaMax CPUs.

Subtract

Operation

	**Mnemonics:** SUB_DINT SUB_INT SUB_LREAL SUB_REAL SUB_UINT

Operation

When the SUB instruction receives power flow, it subtracts the operand IN2 from the operand IN1 of the same data type as IN2 and stores the result in the output variable assigned to Q, also of the same data type.
The power flow output is energized when SUB is performed without overflow, unless an invalid operation occurs. If an overflow occurs, the result is the largest possible value with the proper sign and no power flow.

SUB_DINT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a DINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
IN2	DINT variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q. **Note:** In Series 90-30 CPU341 and lower, DINT **constants** are limited to values between -32,768 and +32,767.
Q	DINT	data flow, R, P, L, AI,	The result of IN1-IN2. If an overflow

	variable	AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

SUB_DINT is supported for all GE IP CPUs.

Note: In Series 90-30 CPU341 and earlier, DINT **constants** are limited to values between -32,768 and +32,767.

SUB_INT

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of an INT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q.
IN2	INT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q.
Q	INT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of IN1-IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

SUB_INT is supported for all GE IP CPUs.

SUB_LREAL

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 64 or more instead of an LREAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*

IN1	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q.
IN2	LREAL variable or constant	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q.
Q	LREAL variable	data flow, R, P, L, AI, AQ, W, I, Q, M, T, G, symbolic, I/O variable	The result of IN1-IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

SUB_LREAL is supported for PACSystems firmware version 5.50 or later.

SUB_REAL

Operands

Notes

- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- (PACSystems only.) You can use a BOOL array of length 32 or more instead of a REAL variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q.
IN2	REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q.
Q	REAL variable	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The result of IN1-IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

SUB_REAL is supported for PACSystems CPUs, VersaMax CPUs, Series 90-70 version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.

SUB_UINT

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use a BOOL array of length 16 or more instead of a UINT variable. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q.
IN2	UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q.
Q	UINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	The result of IN1-IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

CPU Support

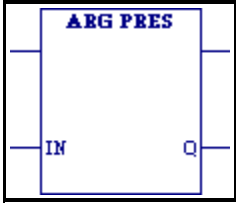
SUB_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Program Flow Instructions

The program flow instructions limit logic execution or change the way the CPU executes the application logic.

Instruction	*Mnemonic*	*Description*
Argument Present	ARG_PRES	Determines if an input or output parameter value existed when the function block instance of the parameter was invoked.
Call	CALL	Causes logic execution to go to a specified subroutine block.
Comment	COMMENT	Places a text explanation in the logic.
End Master Control Relay	ENDMCR	Non-nested End Master Control Relay. Indicates that the subsequent logic is to be executed with normal power flow.
	ENDMCRN	Nested End Master Control Relay. Indicates that the subsequent logic is to be executed with normal power flow.
End of Logic	END	Provides an unconditional end of logic. The logic executes from the first rung to the last rung or the END instruction, whichever is encountered first.
Jump	JUMP	Non-nested jump. Causes logic execution to jump to a specified location indicated by a LABEL.
	JUMPN	Nested jump. Causes logic execution to jump to a specified location indicated by a LABELN. JUMPN/LABELN pairs can be nested within one another. Multiple JUMPNs can share the same LABELN.
Label	LABEL	Non-nested label. Specifies the target location of a JUMP instruction.
	LABELN	Nested label. Specifies the target location of a JUMPN instruction.
Master Control Relay	MCR	Non-nested Master Control Relay. Causes all rungs between the MCR and its subsequent ENDMCR to be executed without power flow.
	MCRN	Nested Master Control Relay. Causes all rungs between the MCR and its subsequent ENDMCRN to be executed without power flow. Up to eight MCRN/ENDMCRN pairs can be nested within one another.
Wires	H_WIRE	Horizontally connects elements of a line of LD logic, to complete the power flow.
	V_WIRE	Vertically connects elements of a line of LD logic, to complete the power flow.

Argument Present

	**Mnemonic:** ARG_PRES

Operation

When the instruction receives power flow, it determines if an input or output parameter value existed when the function block instance of the parameter was invoked. For example, a parameter may be optional (pass by value).

Note: This instruction must be called from a function block instance or from a parameterized block.

Operands

Notes

- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	▪ BOOL, DINT, DWORD, INT, REAL, UINT, WORD variable ▪ Variable array head name or variable array head name element [000]	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable BOOL, WORD, and DWORD variables can also use S, SA, SB, SC	Input parameter value. **Notes** ▪ IN cannot be a constant. ▪ For an array variable named *myArray*, specify *myArray* or *myArray [000]*.
Q	BOOL variable	data flow, I, Q, M, T, S, G, symbolic, I/O variable	▪ True: Input or output parameter value exists. ▪ False: (Default.) Input or output parameter value does not exist. **Note:** Q is required.

Examples

Example 1

ARG_PRES(my_input_REAL, my_REAL_exists); ' Check if parameter my_input_REAL passed

Example 2

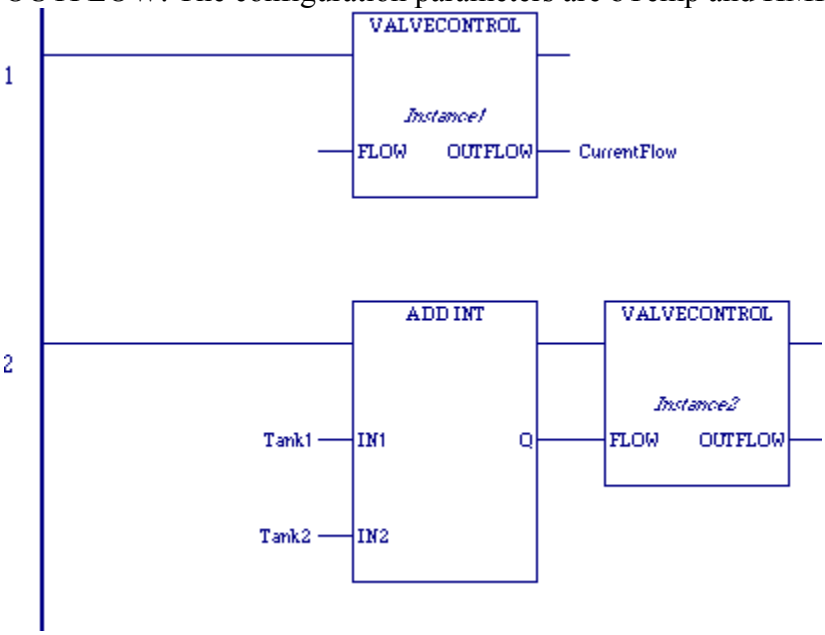
ARG_PRES(my_input_BOOL, my_BOOL_exists); ' Check if parameter my_input_BOOL passed

Example 3

```
If not my_REAL_exists or not my_BOOL_exists then
        Use_HMI := 1; ' If not connected, then use the HMI settings instead of the input
        settings
Else
        Use_HMI := 0;
End_If;
```

Example 4

VALVECONTROL contains an input parameter FLOW and an output parameter OUTFLOW. The configuration parameters are bTemp and HMIFlow.



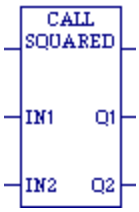

```
'In the ST logic below, if the input FLOW is attached, then use the
'provided FLOW rate; otherwise, use a flow rate provided by an HMI.
ARG_PRES (FLOW, bTemp);
if bTemp = 1 then
        OUTFLOW := OUTFLOW + FLOW;
else
        OUTFLOW := OUTFLOW + HMIFLOW;
end_if;
```

```
if OUTFLOW > 10000 then
        OUTFLOW := 0;
end_if;
```

CPU Support

PACSystems with firmware version 5.00 or later.

Call

	
Non-parameterized CALL	(PACSystems CPUs and Series 90-70 CPUs only.) Parameterized CALL to a parameterized C block or a parameterized block in another language.

Operation

When the CALL instruction receives power flow, it causes the scan execution to go immediately to the designated subroutine LD block, FBD block, ST block, C block, or IL block (whether it is a parameterized block or not) and execute it. After the subroutine block execution is complete, control returns to the point in the logic immediately following the CALL instruction.
You can automatically create a CALL instruction by dragging an LD Diagnostic Logic Block (LD DLB), LD, ST, FBD, C, IL, Local Logic, or Motion block fxPart to the LD, FBD, or ST editor.

Notes

Operands

Note: (PACSystems only.) You can use symbolic variables or I/O variables for any operand and data type.

Operand	*Description*
????	Block name; the name of the block to transfer to. ▪ You cannot CALL the _MAIN block. ▪ You cannot CALL a block before actually creating the block. ▪ An LD block can call itself. ▪ An IL block can call itself. If the block name is invalid, an error message displays and the name is ignored.
(PACSystems and Series 90-70 only, parameterized calls only.) The parameter names are user-defined.	**Notes for parameterized C blocks** ▪ The valid data type, value range, and memory area for each parameter are stated in the C block's written documentation. ▪ You must set the C block's parameters. ▪ The names you have assigned to the parameters appear on the

	CALL instruction. ▪ There are several differences between Series 90-70 and PACSystems RX7i with regards to parameterized C blocks. **Notes for parameterized LD blocks** ▪ You must set the parameterized block's parameters. ▪ A parameterized block is not required to have the same number of inputs and outputs. ▪ The names you have assigned to the parameters appear on the CALL instruction. ▪ Valid operands on the CALL instruction include variables and indirect references. Input operands can also be constants. ▪ If a formal parameter is an array of BIT type and has a length evenly divisible by 16, then a variable or array residing in %R memory can be passed to the parameterized block as an operand. For example, if a parameterized block has a formal parameter Y1 of data type BIT and length 48, you can pass a WORD array of length 3 to Y1. ▪ You do not need to define the BOOL parameter Y0 to use it in the parameterized block's logic. When the parameterized block stops executing and Y0 is ON, the CALL passes power flow to the right. If Y0 is OFF, the CALL passes no power flow. ▪ If a row in the parameter grid is blank, and there are more parameters below it, a gap will be left in the CALL instruction. ▪ A gap is also left if a parameter is of type NONE. ▪ There are several differences between Series 90-70 and PACSystems RX7i with regards to parameterized blocks.

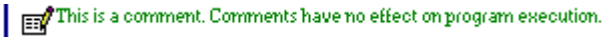
CPU Support

CALL is supported for all GE IP Controllers, *except* Series 90 Micro Controllers.

Comment

Operation

The Comment instruction is used to enter a text explanation in LD logic. It occupies an entire rung. When you insert a Comment instruction into the LD logic, it displays ????. After you key in a comment, the first few words are displayed.



You can set the Comment mode option to Brief or Full.

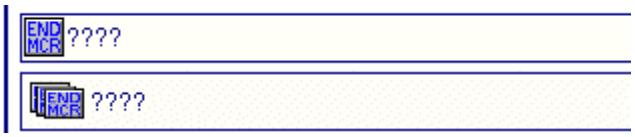
Notes

- A COMMENT must be placed in an empty row.
- The maximum length of a comment is 30,000 characters.
- A comment has no effect on program execution.
- Editing a comment makes you lose equality.

CPU Support

COMMENT is supported for all GE IP CPUs.

End Master Control Relay



Operation

The End Master Control Relay instruction comes in two forms:

Mnemonic	*Description*	*Always associated with...*
ENDMCR	Non-nested form	an MCR instruction
ENDMCRN	Nested form	an MCRN instruction

The End Master Control Relay instruction marks the end of a section of logic begun with a Master Control Relay instruction.
When the MCR(N) associated with the ENDMCR(N) is active, the ENDMCR(N) causes program execution to resume with normal power flow. When the MCR(N) associated with the ENDMCR(N) is not active, the ENDMCR(N) has no effect.
ENDMCR and ENDMCRN have a negated Boolean input EN. ENDMCR(N) must be tied to the power rail; there can be no logic before it in the rung; execution cannot be conditional.
The ENDMCR instruction also has a name, which identifies the ENDMCR and associates it with the corresponding MCR. Likewise, ENDMCRN has a name that identifies it and associates it with the corresponding MCRN(s). Up to eight MCRNs can be associated with the same ENDMCRN, except in Series 90-35x and 36x, where only one MCRN may be associated with any given ENDMCRN.
The ENDMCR(N) instruction has no outputs; there can be nothing after an ENDMCR instruction in a rung.

Operands

Operand	*Description*
????	Name; the name associated with the MCR(N) that starts the section of logic.

CPU Support

ENDMCR is supported for Series 90- 30 CPU311 through CPU341.
ENDMCRN is supported for all GE IP CPUs.
When importing an LD block written with ENDMCRs for use with a target that requires ENDMCRNs, you can easily convert the ENDMCRs into ENDMCRNs.

End of Logic

CPU Support

Operation

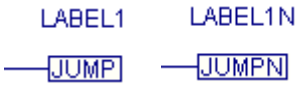
The End of Logic (END) instruction provides an unconditional end of logic. The program executes from the first rung to the last rung or END, whichever is encountered first.
END unconditionally terminates program execution. There can be nothing after END in the rung. No logic beyond END is executed, and control is transferred to the beginning of the program, that is, the first rung of the _MAIN block, for the next scan.
END is useful for debugging purposes because it prevents any logic that follows from being executed.

Note: The programming software provides an [END OF PROGRAM LOGIC] marker to indicate the end of program execution. This marker is used if no END is programmed in the logic.

CPU Support

END is supported for VersaMax CPUs and for Series 90-30 350 and higher CPUs; however, until Version 7 of the Series 90-30, no fault message was reported when incorrectly using END.

Jump



Specific operational notes: JUMP | JUMPN

Operation (Features Common to JUMP and JUMPN)

The Jump instruction comes in two forms:

Mnemonic	*Description*	*Always associated with...*
JUMP	Non-nested form	a LABEL instruction
JUMPN	Nested form	a LABELN instruction

A JUMP(N) instruction causes a portion of the logic to be bypassed. Program execution continues at the LABEL(N) specified in the same LD block. Power flow jumps directly from the JUMP(N) to the rung with the named LABEL(N).
When the Jump is active, any instructions between the jump and the label are not executed. All coils between JUMP(N) and its associated LABEL(N) are left at their previous states. This includes coils associated with timers, counters, latches, and relays.
Any JUMP(N) can be either a forward or a backward jump, that is, its LABEL(N) can be either in a further or previous rung. The LABEL(N) must be in the same LD block.

Warning: To avoid creating an endless loop with forward and backward JUMP(N) instructions, a backward JUMP(N) must contain a way to make it conditional.

A JUMP(N) instruction cannot be connected to the right with an H_WIRE.
For all Controllers except PACSystems, a JUMP(N) instruction must be the last instruction in a rung and if the rung has multiple branches, the other branches cannot end with a coil.
On a PACSystems, the JUMP(N) can be at the end of one branch, while another branch can end with a coil. The placement of the JUMP(N) and the coil relative to one another affects the way logic executes.
For a Series 90-70, a JUMPN instruction must be in column 10.
For other CPUs, the LD editor does not let you place a JUMP(N) instruction to the left of the coil justification column; it automatically pushes it right to that column.

Operands

Operand	*Description*
????	Label name; the name assigned to the destination LABEL(N).

CPU Support

JUMP is supported for Series 90-30 CPU311 through CPU341.
JUMPN is supported for all GE IP CPUs.
When importing an LD block written with JUMPs for use with a target that requires JUMPNs, you can easily convert the JUMPs into JUMPNs.

JUMP: Specific Operational Notes

There can be only one JUMP instruction for each LABEL instruction. A JUMP and its associated LABEL can be placed anywhere in the logic, provided that the range between the JUMP and the LABEL does not overlap or contain or fall within the range of any other JUMP / LABEL pair or any MCR / ENDMCR pair of instructions.

JUMPN: Specific Operational Notes

A JUMPN and its associated LABELN can be placed anywhere in the logic, as long as the JUMPN / LABELN range:

- Is completely nested within another JUMPN / LABELN range, up to a maximum eight levels of nesting.
- Does not overlap at all with the range of a JUMP / LABEL pair.
- Does not overlap at all with the range of a MCR / ENDMCR pair.

Label

Operation

The Label instruction comes in two forms:

Mnemonic	*Description*	*Associated with...*
LABEL	Non-nested form	a JUMP instruction
LABELN	Nested form	a JUMPN instruction

A LABEL instruction can be used as an empty marker to organize your logic, just like a comment. A LABEL instruction's specific role, however, is to mark the destination point of a JUMP instruction in the same LD block when JUMP and LABEL bear the same name. A LABELN instruction marks the destination point of a JUMPN instruction in the same LD block, when LABELN and JUMPN bear the same name.
Each LABEL or LABELN must have a unique name in any given target.
Normal program execution resumes upon encountering a LABEL or LABELN instruction.
Programs without a matched JUMP(N) / LABEL(N) pair can be created and downloaded to the Controller. A warning will be issued.
LABEL(N) has no inputs and no outputs; there can be nothing either before or after a LABEL(N) in a rung.

Operands

Operand	*Description*
????	Label name; a unique name for the LABEL(N)

CPU Support

LABEL is supported for Series 90-30 CPU311 through CPU341.
LABELN is supported for all GE IP CPUs.
When importing an LD block written with LABELs for use with a target that requires LABELNs, you can easily convert the LABELs into LABELNs.

Master Control Relay

Specific operational notes and examples: MCR | MCRN

Operation (Features Common to MCR and MCRN)

The Master Control Relay instruction comes in two forms:

Mnemonic	*Description*	*Always associated with...*
MCR	Non-nested form	an ENDMCR instruction
MCRN	Nested form	an ENDMCRN instruction

An MCR(N) instruction marks the beginning of a section of logic that will be executed with no power flow. The end of an MCR section must be marked with an ENDMCR having the same name as the MCR and the end of an MCRN section must be marked with an ENDMCRN having the same name as the MCRN. ENDMCRs and ENDMCRNs must follow their corresponding MCRs and MCRNs in the logic.

All rungs between an active MCR(N) and its corresponding ENDMCR(N) are executed without power flow to coils. The ENDMCR(N) instruction associated with the MCR(N) causes normal logic execution to resume.

With a Master Control Relay, instructions within the scope of the Master Control Relay are executed *without power flow*, and *coils are turned off*.

Note: With a JUMP instruction, any instructions between the JUMP and the LABEL are *not executed*, and *coils are not affected*.

Block calls within the scope of an active Master Control Relay will not execute. However, any timers in the logic will continue to accumulate time.

An MCR(N) instruction cannot be connected to the right with an H_WIRE.

For all Controllers except PACSystems, an MCR(N) instruction must be the last instruction in a rung and if the rung has multiple branches, the other branches cannot end with a coil.

On a PACSystems, the MCR(N) can be at the end of one branch, while another branch can end with a coil. The placement of the MCR(N) and the coil relative to one another affects the way logic executes.

Unlike JUMP instructions, MCR(N)s can only move forward. An ENDMCR(N) instruction must appear after its corresponding MCR(N) instruction in the logic.

The following controls are imposed by an MCR(N):

- Timers do not increment or decrement. TMR types are reset. For an ONDTR built-in function block instance, the accumulator holds its value.
- Normal outputs are off; negated outputs are on.

Note: When an MCR(N) is energized, the logic it controls is scanned and contact status is displayed, but no outputs are energized. If you are not aware that an MCR(N) is controlling the logic being observed, this might appear to be a faulty condition.

Operands

Both forms of the Master Control Relay instruction have the same operand: a name that identifies the MCR(N). This name is used again with an ENDMCR(N) instruction. Both MCR and MCRN have no output.

Operand	*Description*
????	Name; the name associated with the MCR that starts the section of logic.

CPU Support

MCR is supported for Series 90-30 CPU311 through CPU341.
MCRN is supported for all GE IP CPUs.
When importing an LD block written with MCRs for use with a target that requires MCRNs, you can easily convert the MCRs into MCRNs.

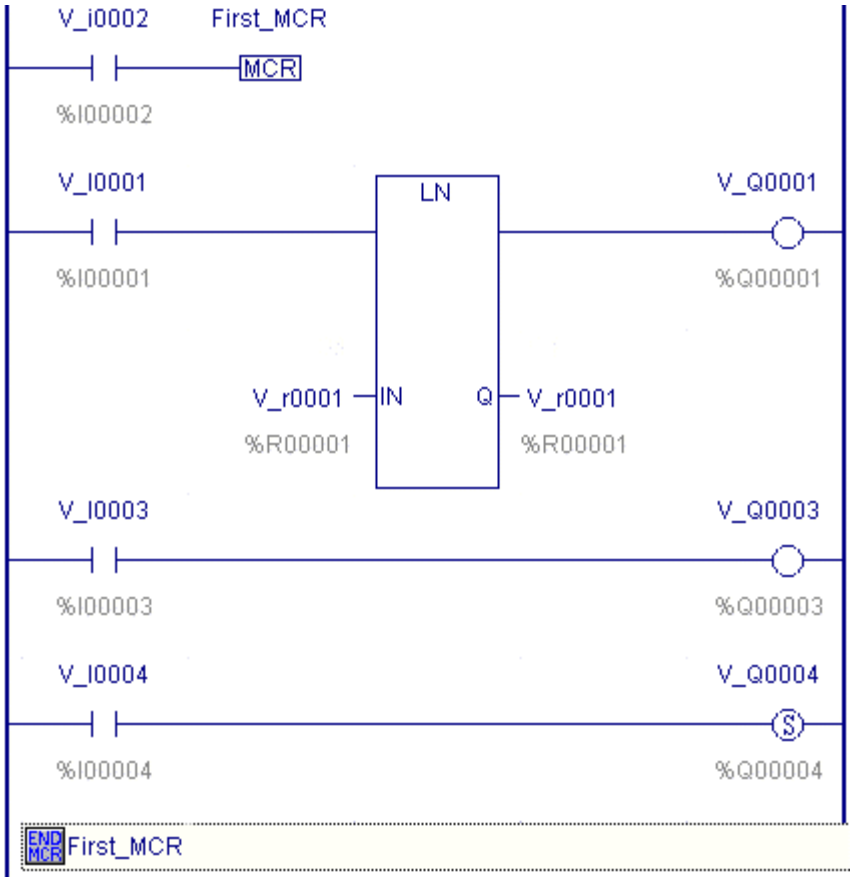
MCR

Specific Operational Notes

There can be only one MCR instruction for each ENDMCR instruction. The range for non-nested MCRs and ENDMCRs cannot overlap or contain or fall within the range of any other MCR/ENDMCR pair or any JUMP/LABEL pair of instructions.

Example

When the V_i0002 contact is ON, the Master Control Relay First_MCR is enabled. When First_MCR is enabled, even if the V_I0001 contact is ON, the LN instruction is executed without power flow (that is, it does not calculate the natural log of V_r0001), and the V_Q0001 coil is turned OFF.
If the V_I0003 and V_I0004 contacts are ON, the V_Q0003 coil is turned OFF and the V_Q0004 coil remains ON.



MCRN

Operation

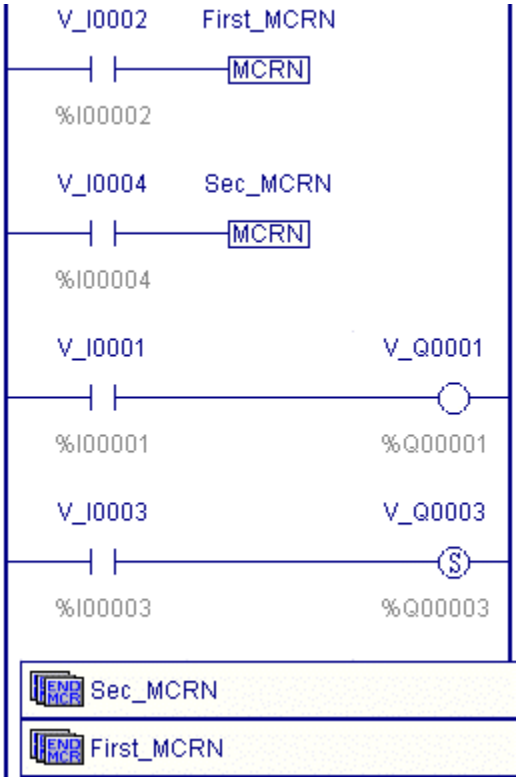
An MCRN and its associated ENDMCRN can be placed anywhere in the logic, as long as the MCRN / ENDMCRN range:

- Is completely nested within another MCRN / ENDMCRN range, up to a maximum eight levels of nesting.
- Does not overlap at all with the range of a MCR / ENDMCR pair
- Does not overlap at all with the range of a JUMP / LABEL pair.

Multiple MCRNs can correspond to a single ENDMCRN, except for the 35x and 36x series CPUs, where only one MCRN may be associated with any given ENDMCRN. This is analogous to the nested JUMP, where you can have multiple JUMPs to the same LABEL.

Example

The following example shows an MCRN named "Sec_MCRN" nested inside the MCRN named "First_MCRN." Whenever the V_I0002 contact allows power flow into the MCRN instruction, logic execution will continue without power flow to the coils until the associated ENDMCRN is reached. If the V_I0001 and V_I0003 contacts are ON, the V_Q0001 coil is turned OFF and the coil V_Q0003 remains ON.


V_I0002
First_MCRN
MCRN
%I00002
V_I0004
Sec_MCRN
MCRN
%I00004
V_I0001
V_Q0001
%I00001
%Q00001
V_I0003
V_Q0003
S
%I00003
%Q00003
END MCR
Sec_MCRN
END MCR
First_MCRN

Wires

Operation

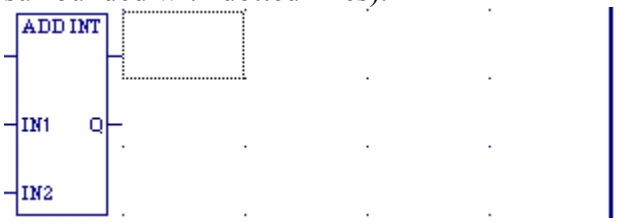
Horizontal and vertical wires (H_WIRE and V_WIRE) are used to connect elements of a line of LD logic between instructions. Their purpose is to complete the flow of logic ("power") from left to right in a line of logic. There are several ways to insert wires. In some cases, wires are automatically inserted.

Notes

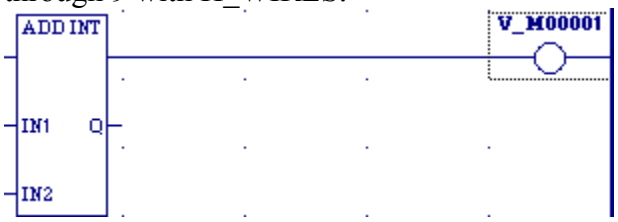
- On PACSystems and Series 90-70 Controllers, you can use wires for data flow, but you cannot route data flow leftwards.
- On Series 90 Micro Controllers, using a H_WIRE to link an instruction directly to the power rail prevents obtaining equality after a download or an upload. Instead of an H_WIRE, use a Normally Open Contact (NOCON) and assign the #ALW_ON system variable to it.

Automatic insertion

H_WIREs are automatically inserted when you insert a coil or a jump to the left of the coil justification column. The LD editor forces the coil or jump right to the coil justification column and links the coil or jump back to the left with H_WIREs.
For example, you have an ADD_INT instruction in column 6, the coil justification is column 10, and you want to insert a coil in column 7, in the active cell (the cell surrounded with dotted lines).



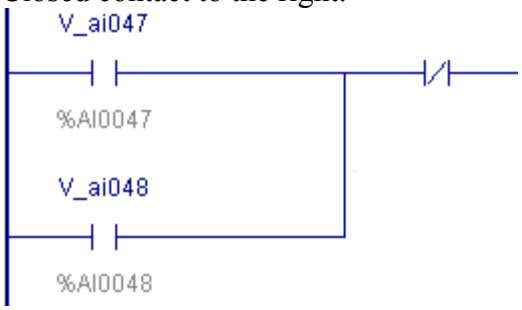
Once you have inserted the coil, the LD editor pushes it to column 10 and fills columns 7 through 9 with H_WIRES.



Had you tried to insert the coil in column 8 or 9, the result would have been the same. The LD editor would have added H_WIRES as far left as required to link up with the ADD_INT instruction.

Example

A horizontal wire connects the contact V_ai047 with the Normally Closed contact to the right. A horizontal wire and a vertical wire connect contact V_ai048 with the Normally Closed contact to the right.



CPU Support

H_WIRE and V_WIRE are supported for all GE IP CPUs.

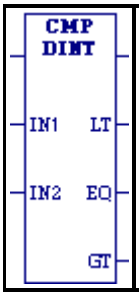
Relational Instructions

Relational instructions compare two values of the same data type or determine whether a number lies within a specified range. The original values are unaffected.

Instruction	*Mnemonic*	*Description*
Compare (PACSystems and 90-70 only)	CMP_DINT CMP_INT CMP_LREAL CMP_REAL CMP_UINT	Compares two numbers, IN1 and IN2, of the data type specified by the mnemonic. ▪ If IN1 < IN2, the LT output is turned ON. ▪ If IN1 = IN2, the EQ output is turned ON. ▪ If IN1 > IN2, the GT output is turned ON.
Equal	EQ_DATA	Tests two values or two sets of values for equality
	EQ_DINT EQ_INT EQ_LREAL EQ_REAL EQ_UINT	Tests two numbers for equality
Greater or Equal	GE_DINT GE_INT GE_LREAL GE_REAL GE_UINT	Tests whether one number is greater than or equal to another
Greater Than	GT_DINT GT_INT GE_LREAL GT_REAL GT_UINT	Tests whether one number is greater than another
Less or Equal	LE_DINT LE_INT LE_LREAL LE_REAL LE_UINT	Tests whether one number is less than or equal to another
Less Than	LT_DINT LT_INT LT_LREAL LT_REAL LT_UINT	Tests whether one number is less than another
Not Equal	NE_DINT NE_INT	Tests two numbers for inequality

	NE_LREAL NE_REAL NE_UINT	
Range	RANGE_DINT RANGE_DWORD RANGE_INT RANGE_UINT RANGE_WORD	Tests whether one number is within the range defined by two other supplied numbers

Compare



Mnemonics:
CMP_DINT
CMP_INT
CMP_LREAL
CMP_REAL
CMP_UINT

Operation

When the Compare (CMP) instruction receives power flow, it compares the value IN1 to the value IN2.

- If IN1 < IN2, CMP energizes the LT (Less Than) output.
- If IN1 = IN2, CMP energizes the EQ (Equal) output.
- If IN1 > IN2, CMP energizes the GT (Greater Than) output.

IN1 and IN2 must be of the same data type.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

When it receives power flow, CMP always passes power flow to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Operands

Notes

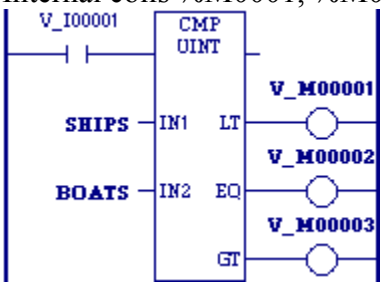
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The two values to compare.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
LT	power flow		Output LT is energized when I1 < I2.
EQ	power flow		Output EQ is energized when I1 =

			I2.
GT	power flow		Output GT is energized when I1 > I2.

Example

When %I00001 is ON, the integer variable SHIPS is compared with the variable BOATS. Internal coils %M0001, %M0002, and %M0003 are set to the results of the compare.



CPU Support

CMP_DINT, CMP_INT, and CMP_UINT are supported for PACSystems CPUs and Series 90-70 CPUs.
CMP_LREAL is supported for PACSystems firmware version 5.50 or later.
CMP_REAL is supported for PACSystems CPUs and Series 90-70 firmware version 3.00 or later floating-point CPUs.

Equal

	**Mnemonics:** EQ_DATA EQ_DINT EQ_INT EQ_LREAL EQ_REAL EQ_UINT

Operation

When an EQ instruction receives power flow, it compares input IN1 to input IN2. These operands must be of the same data type. If inputs IN1 and IN2 are equal, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when EQ_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for EQ_DINT or EQ_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If EQ_DINT or EQ_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

EQ_DATA

Operands

Notes

- All operands are required.
- Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	Enumerated variables, arrays of enumerated variables, structure variables, arrays of structure variables. IN1 and IN2 must be of the same data type.	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic	Values to compare

Q	BOOL variable		Result of the comparison. The variable assigned to the output 'Q' is set to True if 'IN1' and 'IN2' are equal (power flow). Otherwise, 'Q' is set to False (no power flow).

CPU Support

EQ_DATA is supported for PACSystems, firmware version 5.60 or later CPUs.

EQ_DINT, EQ_INT, EQ_LREAL, EQ_REAL, EQ_UINT

Operands

Notes

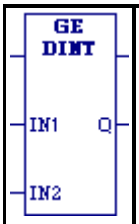
- (PACSystems CPUs and Series 90-70 CPUs) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 = IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q			The power flow. If IN1 = IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

EQ_DINT and EQ_INT are supported for all GE IP CPUs.
EQ_LREAL is supported for PACSystems firmware version 5.50 or later.
EQ_REAL is supported for all PACSystems CPUs, all VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
EQ_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Greater or Equal



Mnemonics:
GE_DINT
GE_INT
GE_LREAL
GE_REAL
GE_UINT

Operation

When the instruction receives power flow, it compares input IN1 to input IN2. These operands must be the same data type. If IN1 ≥ IN2, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when GE_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for GE_DINT or GE_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If GE_DINT or GE_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

Operands

Notes

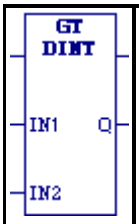
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 ≥ IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	Power flow		If IN1 ≥ IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

GE_DINT and GE_INT are supported for all GE IP CPUs.
GE_LREAL is supported for all PACSystems firmware version 5.50 or later.
GE_REAL is supported for all PACSystems CPUS, all VersaMax CPUs, Series 90-70 firmware version 3.00 or later CPUs, and Series 90-30 floating-point CPUs.
GE_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Greater Than



Mnemonics:
GT_DINT
GT_INT
GE_LREAL
GT_REAL
GT_UINT

Operation

When the instruction receives power flow, it compares input IN1 to input IN2. These operands must be of the same data type. If IN1 > IN2, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when GT_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for GT_DINT or GT_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If GT_DINT or GT_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

Operands

Notes

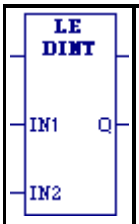
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 > IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	Power flow		If IN1 > IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

GT_DINT and GT_INT are supported for all GE IP CPUs.
GT_LREAL is supported for PACSystems firmware version 5.50 or later.
GT_REAL is supported for all PACSystems CPUs, all VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
GT_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Less or Equal

	**Mnemonics:** LE_DINT LE_INT LE_LREAL LE_REAL LE_UINT

Operation

When the instruction receives power flow, it compares input IN1 to input IN2. These operands must be of the same data type. If IN1 ≤ IN2, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when LE_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for LE_DINT or LE_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If LE_DINT or LE_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

Operands

Notes

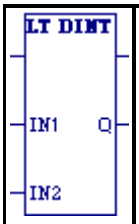
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 ≤ IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	Power flow		If IN1 ≤ IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

LE_DINT and LE_INT are supported for all GE IP CPUs.
LE_LREAL is supported for PACSystems firmware version 5.50 or later.
LE_REAL is supported for all PACSystems CPUs, all VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
LE_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Less Than



Mnemonics:
LT_DINT
LT_INT
LT_LREAL
LT_REAL
LT_UINT

Operation

When the instruction receives power flow, it compares input IN1 to input IN2. These operands must be of the same data type. If IN1 < IN2, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when LT_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for LT_DINT or LT_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If LT_DINT or LT_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.
;

Operands

Notes

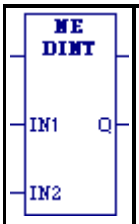
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 < IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	Power flow		If IN1 < IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

LT_DINT and LT_INT are supported for all GE IP CPUs.
LT_LREAL is supported for PACSystems firmware version 5.50 or later.
LT_REAL is supported for all PACSystems CPUs, all VersaMax CPUs, Series 90-70 firmware version 3.00 or later floating-point CPUs, and Series 90-30 floating-point CPUs.
LT_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Not Equal



Mnemonics:
NE_DINT
NE_INT
NE_LREAL
NE_REAL
NE_UINT

Operation

When the instruction receives power flow, it compares input IN1 to input IN2. These operands must be of the same data type. If IN1 ≠ IN2, the instruction passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Note: (VersaMax, VersaMax Nano/Micro, and Series 90-30 floating-point CPUs only.) The %S0020 bit is set to ON when NE_REAL executes successfully. It is cleared when either input is NaN (Not a Number). This is because NaN has a reserved representation in the REAL number format, which makes it detectable by any instruction. No such functionality exists for NE_DINT or NE_INT. If an overflow occurred on a previous DINT or INT operation, the result was the largest possible value with the proper sign and no power flow. If NE_DINT or NE_INT is fed the largest possible value with any sign, it cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked.

Tip: To compare values of different data types, first use conversion instructions to make the types the same.

Operands

Notes

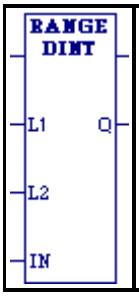
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN1, IN2	DINT, LREAL, or REAL variable or constant	data flow, R, P, L, AI, AQ, W. PACSystems also supports I, Q, M, T, G, symbolic, I/O variable.	The values to be compared in the relational equation IN1 ≠ IN2. IN1 and IN2 must be of the same data type.
	INT or UINT variable or constant	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
Q	Power flow		If IN1 ≠ IN2, then Q is energized, unless IN1 or IN2 is NaN.

CPU Support

NE_DINT and NE_INT are supported for all GE IP CPUs.
NE_LREAL is supported for PACSystems CPUs firmware version 5.50 or later.
NE_REAL is supported for all PACSystems CPUs, all VersaMax CPUs, Series 90-70 firmware version 3.00 and later floating-point CPUs, and Series 90-30 floating-point CPUs.
NE_UINT is supported for PACSystems CPUs and Series 90-70 CPUs.

Range



Mnemonics:
RANGE_DINT
RANGE_DWORD
RANGE_INT
RANGE_UINT
RANGE_WORD

Operation

When the Range instruction is enabled, it compares the value of input IN against the range delimited by operands L1 and L2. Either L1 or L2 can be the high or low limit. When L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, output parameter Q is set ON (1). Otherwise, Q is set OFF (0).
If the operation is successful, it passes power flow to the right.

Operands

Notes

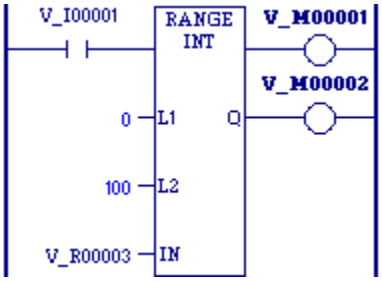
- (PACSystems CPUs and Series 90-70 CPUs.) Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).
- You can use BOOL arrays instead of other data types. Restrictions apply.

Operand	*Data Type*	*Memory Area*	*Description*
IN	DINT variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The value to compare against the range delimited by L1 and L2. Must be the same data type as L1 and L2.
	DWORD variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable or constant	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	

L1	DINT variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The start point of the range. May be the upper limit or the lower limit. Must be the same data type as IN and L2.
	DWORD variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable or constant	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	
L2	DINT variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, W, symbolic, I/O variable.	The end point of the range. May be the lower or upper limit. Must be the same data type as IN and L1.
	DWORD variable or constant	R, P, L, AI, AQ. PACSystems also supports data flow, I, Q, M, T, G, S, SA, SB, SC, W, symbolic, I/O variable.	
	INT, UINT, or WORD variable or constant	I, Q, M, T, G, R, P, L, AI, AQ. PACSystems also supports data flow, W, symbolic, I/O variable.	
Q	Power flow		If L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, then Q is energized; otherwise, Q is off.

Example

When RANGE_INT receives power flow from the normally open contact %I0001, it determines whether the value in %R00003 is within the range 0 to 100 inclusively. Output coil %M00002 is ON only if 0 ≤ %AI0050 ≤ 100.



CPU Support

RANGE_DINT, RANGE_INT, and RANGE_WORD are supported for PACSystems CPUs, VersaMax CPUs, for Series 90-30 Version 4.40 or later CPUs, and Series 90-70 Version 5.00 or later CPUs.
RANGE_DWORD and RANGE_UINT are supported for PACSystems CPUs and Series 90-70 Release 5.00 or later CPUs.

Timers

(Not available for LD Diagnostic Logic Blocks (DLBs), whether active or inactive.)

Function block	*Mnemonic*	*Description*
Off Delay Timer	OFDT_HUNDS OFDT_SEC OFDT_TENTHS OFDT_THOUS	The timer's Current Value (CV) resets to zero when power flow input is on. CV increments while power flow is off. When CV=PV (Preset Value), power flow is no longer passed to the right until power flow input is on again.
On Delay Stopwatch Timer	ONDTR_HUNDS ONDTR_SEC ONDTR_TENTHS ONDTR_THOUS	Retentive on delay timer. Increments while it receives power flow and holds its value when power flow stops.
On Delay Timer	TMR_HUNDS TMR_SEC TMR_TENTHS TMR_THOUS	Simple on delay timer. Increments while it receives power flow and resets to zero when power flow stops.
Timer Off Delay	TOF	The LD timer off delay (TOF) standard function block delays setting the output, Q, to OFF, for the input preset time, PT, when you set the input, IN, to OFF.
Timer On Delay	TON	The LD timer on delay (TON) standard function block delays setting the output, Q, to ON, when you set the input, IN, to ON, for the input preset time, PT.
Timer Pulse	TP	The LD timer pulse (TP) standard function block outputs a pulse of a duration determined by the input preset time, PT, when you set the input, IN, to ON.

Notes

- All GE IP Controllers support time-tick contacts that provide regular pulses of power flow to other instructions.
- The data associated with the timer built-in function blocks is retentive through power cycles.
- On Series 90 Micro Controller targets, using a timer built-in function block sometimes prevents obtaining equality after a download or an upload. Simply use a Normally Open Contact (NOCON) somewhere in the logic that has the timer and assign the #ALW_ON system variable to the NOCON.

Instance Data Required for the OFDT, ONDTR, and TMR Timers

Each OFDT, ONDTR, and TMR timer uses a one-dimensional, three-word array of %R, %P, %L, or %W memory to store the following information:

Current value (CV) Word 1

Preset value (PV) Word 2

Control word Word 3

When you enter a timer, you must enter a beginning address for the three-word array (three-word block of registers).

Warning: Do not use two consecutive words (registers) as the starting addresses of two timers. Logic Developer - PLC does not check or warn you if register blocks overlap. Timers will not work if you place the current value of a second timer on top of the preset value for the previous timer.

Word 1: Current value (CV)

Warning: The first word (CV) can be read but should not be written to, or the built-in function block instance may not work properly.

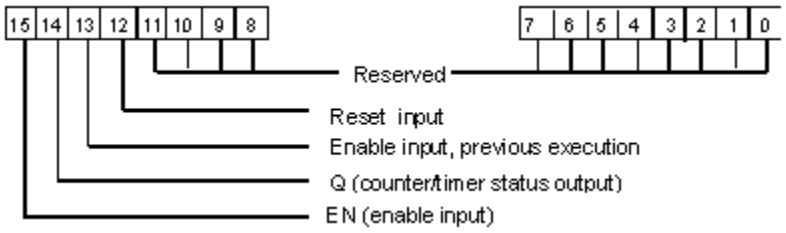
Word 2: Preset value (PV)

When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 in the timer's built-in function block instance.

- If you use a different address and you change word 2 directly, your change will have no effect, as PV will overwrite word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the timer is running and the change will be effective.

Word 3: Control word

The control word stores the state of the boolean inputs and outputs of its associated timer, as shown in the following diagram:



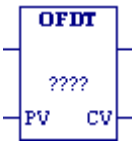
Notes

- (VersaMax and Series 90-30) Bits 0 through 11 are used for timer accuracy; not for counters.
- (Series 90-70) Bits 12 and 13 are not used for counters. Bits 0 through 13 are used for timer accuracy.
- When using the Bit Test, Bit Set, Bit Clear, or Bit Position instruction, the bits are numbered 1 through 16, *not* 0 through 15 as shown above.

Warning: The third word (Control) can be read but should not be written to; otherwise, the built-in function block instance will not work.

Off Delay Timer

(Not available for LD Diagnostic Logic Blocks (DLBs), whether active or inactive.)



Mnemonics:
OFDT_SEC
OFDT_TENTHS
OFDT_HUNDS
OFDT_THOUS

The CV operand is available only for PACSystems CPUs.

Operation

An Off-Delay Timer (OFDT) built-in function block instance increments while power flow is off, and the timer's Current Value (CV) resets to zero when power flow is on. OFDT passes power until the specified interval PV (Preset Value) has elapsed. Because no automatic initialization to the outgoing power flow state occurs at power-up, the Q state is retentive across power failure.
Time can be counted in various increments, depending on the CPU family:

- **Seconds** (PACSystems and Series 90-70)
- **Tenths** (0.1) of a second (all CPUs)
- **Hundredths** (0.01) of a second (all CPUs)
- **Thousandths** (0.001) of a second (VersaMax, PACSystems CPUs with firmware version 2.00 or later, and Series 90-30).

The range for PV is 0 to +32,767 time units. If PV is out of range, it has no effect on the timer's word 2.
When OFDT receives power flow, CV is set to zero and the timer passes power to the right, even if PV=0.

Note: (PACSystems and Series 90-70.) OFDT does not pass power flow if the preset value is zero or negative.

The output remains on as long as OFDT receives power flow.
Each time OFDT is invoked with the power flow input set to OFF, CV is updated to reflect the elapsed time since the timer was reset. OFDT continues passing power to the right until CV equals or exceeds PV. When this happens, OFDT stops passing power to the right and OFDT stops accumulating time. If PV equals 0 or is negative, the timer stops passing power to the right the first time that it is invoked with its power flow input set to OFF.
When the function block instance receives power flow again, CV resets to zero.

Notes

- On Series 90 Micro Controller targets, a timer function block instance sometimes prevents obtaining equality after a download or an upload. Use a Normally Open Contact (NOCON) somewhere in the block that has the timer and assign the #ALW_ON system variable to the NOCON.

- (Series 90-30.) Normal use of OFDT is to invoke the built-in function block with a particular reference address exactly one time each scan. Do not invoke OFDT with the same reference

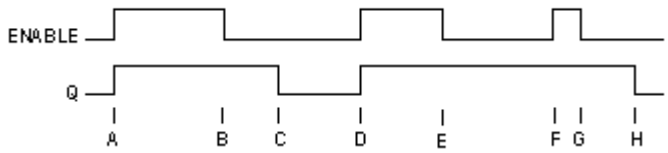
address more than once per scan (inappropriate accumulation of time would result). If OFDT is used within a subroutine block, then ensure that the subroutine block is not called more than once per scan. Also, you should not program a JUMP around an OFDT function block instance.

- (PACSystems and Series 90-70.) The best way to use an OFDT instance is to invoke it with a particular reference address exactly one time each scan. Do not invoke an OFDT with the same reference address more than once per scan (inappropriate accumulation of time would result). When an OFDT instance appears in an LD block, it accumulates time only once per scan. Subsequent calls to that same LD block within the same scan will have no effect on its OFDT instances. Do not program an OFDT instance with the same reference address in two different blocks. You should not program a JUMP around an OFDT instance. Also, if you use recursion, that is, having a block call itself either directly or indirectly, program the LD block so that it invokes the timer before it makes any recursive calls to itself. See Using OFDT, ONDTR, and TMR Timers in PACSystems and Series 90-70 Parameterized LD Blocks and OFDT, ONDTR, and TMR Timer Limitations in UDFBs.

- An OFDT expires, that is, turns OFF power flow to the right, the first scan that it does not receive power flow if the previous scan time was greater than PV.

- When an OFDT instance is used in a logic block that is not called every scan, the timer accumulates time between calls to the logic block unless it is reset. This means that OFDT instances are like a timer operating in logic with a much slower scan than the timer in the main logic block. For logic blocks that are inactive for a long time, an OFDT instance should be programmed to allow for this catchup feature. For example, if an instance in a logic block is reset and the logic block is not called (is inactive) for four minutes, when the logic block is called, four minutes of time will already have accumulated. If the enable input is OFF, these four minutes are applied to the timer, that is, CV is set to four minutes.

Timing diagram



A. ENABLE and Q both go high; timer is reset (CV = 0).
B. ENABLE goes low; timer starts accumulating time.
C. CV reaches PV; Q goes low and timer stops accumulating time.
D. ENABLE goes high; timer is reset (CV = 0).
E. ENABLE goes low; timer starts accumulating time.
F. ENABLE goes high; timer is reset (CV = 0) before CV had a chance to reach PV. (The diagram is not to scale.)
G. ENABLE goes low; timer begins accumulating time.
H. CV reaches PV; Q goes low and timer stops accumulating time.

Operands

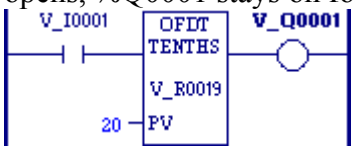
Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any

			means. ▪ Overlapping reference addresses may cause erratic timer operation.
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE_INT instruction or an operator interface.
CV	INT variable;BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(PACSystems and Series 90-70 only; optional.) The current value of the timer; the same value as Word 1 in the ???? operand.

Examples

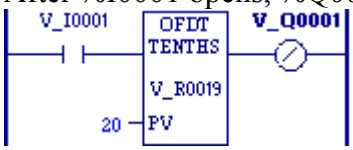
Example 1: Series 90-30 CPUs, version 4.40 and later, VersaMax CPUs

OFDT turns on output coil %Q00001 whenever contact %I00001 is closed. After %I0001 opens, %Q0001 stays on for 2 seconds then turns off.



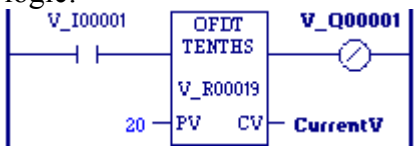
Example 2: Series 90-30 CPUs, version 4.40 and later, VersaMax CPUs

The output action is reversed by the use of a negated output coil. In this circuit, the OFDT timer turns off negated output coil %Q0001 whenever contact %I0001 is closed. After %I0001 opens, %Q0001 stays off for 2 seconds then turns on.



Example 3: Series 90-70 CPUs

The same situation as example 2 translates as follows into PACSystems and Series 90-70 logic:

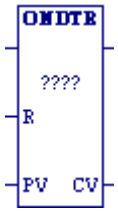


CPU Support

OFDT_SEC is supported for PACSystems CPUs and Series 90-70 CPUs.
OFDT_TENTHS and OFDT_HUNDS are supported for all GE IP CPUs, except pre-version 4.40 Series 90-30 CPUs.
OFDT_THOUS is supported for Series 90-30 CPUs, Version 4.40 and later, PACSystems CPUs with firmware version 2.00 or later, and VersaMax CPUs.

On Delay Stopwatch Timer

(Not available for LD Diagnostic Logic Blocks (DLBs), whether active or inactive.)



Mnemonics:
ONDTR_SEC
ONDTR_TENTHS
ONDTR_HUNDS
ONDTR_THOUS

The CV operand is available only for PACSystems CPUs.

Operation

A retentive On-Delay Stopwatch Timer (ONDTR) built-in function block instance increments while it receives power flow and holds its value when power flow stops. Time may be counted in various increments, depending on the CPU family:

- **Seconds** (PACSystems and Series 90-70)
- **Tenths** (0.1) of a second (all CPUs)
- **Hundredths** (0.01) of a second (all CPUs)
- **Thousandths** (0.001) of a second (VersaMax, PACSystems CPUs with firmware version 2.00 or later, and Series 90-30).

The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When ONDTR first receives power flow, it starts accumulating time (Current Value (CV)). When this timer is encountered in the LD logic, its CV is updated. When the CV equals or exceeds Preset Value (PV), output Q is energized, regardless of the state of the power flow input.

As long as the timer continues to receive power flow, it continues accumulating until CV equals the maximum value (+32,767 time units). Once the maximum value is reached, it is retained and Q remains energized regardless of the state of the enable input.

When power flow to the timer stops, CV stops incrementing and is retained. Output Q, if energized, will remain energized. When ONDTR receives power flow again, CV again increments, beginning at the retained value.

When reset (R) receives power flow and PV is not equal to zero, CV is set back to zero and output Q is de-energized.

When R receives power flow and PV=0, the behavior of ONDTR depends on the CPU type:

- On the Series 90-30 CPU35x and CPU36x CPUs, if the power flow input to the ONDTR is low, PV = 0 and reset R receives power flow, then CV is set back to zero and the output will be low.
- On the Series 90-30 CPU311 and CPU341, under these same conditions, the output will be high.

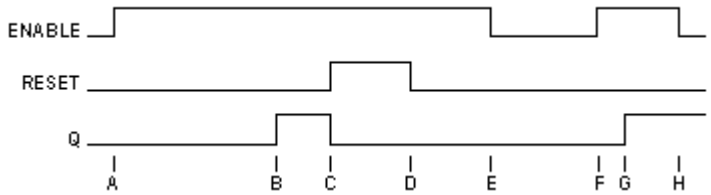
Note: (PACSystems and Series 90-70 only.) If PV equals zero and the timer is enabled, the output of the timer activates. Subsequent removal of enable or activation of reset will have no effect on the timer output; it will remain enabled.

ONDTR passes power flow to the right when CV is greater than or equal to PV. Since no automatic initialization to the outgoing power flow state occurs at power-up, the power flow state is retentive across power failure.

Notes

- On Series 90 Micro Controller targets, a timer built-in function block instance sometimes prevents obtaining equality after a download or an upload. Use a Normally Open Contact (NOCON) somewhere in the block that has the timer and assign the #ALW_ON system variable to the NOCON.
- (Series 90-30.) Normal use of ONDTR is to invoke the built-in function block with a particular reference address exactly one time each sweep. Do not invoke ONDTR with the same reference address more than once per scan (inappropriate accumulation of time would result). If ONDTR is used within a subroutine block, then ensure that the subroutine block is not called more than once per scan. Also, you should not program a JUMP around an ONDTR instance.
- (PACSystems and Series 90-70.) The best way to use an ONDTR built-in function block is to invoke it with a particular reference address exactly one time each scan. Do not invoke an ONDTR with the same reference address more than once per scan (inappropriate accumulation of time would result). When an ONDTR instance appears in an LD block, it accumulates time only once per scan. Subsequent calls to that same LD block within the same scan will have no effect on its ONDTR instances. Do not program an ONDTR instance with the same reference address in two different blocks. You should not program a JUMP around an ONDTR instance. Also, if you use recursion (that is, having a block call itself either directly or indirectly), program the LD block so that it invokes the timer before it makes any recursive calls to itself. See Using OFDT, ONDTR, and TMR Timers in PACSystems and Series 90-70 Parameterized LD Blocks and OFDT, ONDTR, and TMR Timer Limitations in UDFBs.
- When ONDTR is used in a logic block that is not called every scan, it accumulates time between calls to the logic block unless it is reset. This means that ONDTR instances are like a timer operating in logic with a much slower scan than the timer in the main logic block. For logic blocks that are inactive for a long time, an ONDTR instance should be programmed to allow for this catch-up feature. For example, if a timer instance in a logic block is reset and the logic block is not called (is inactive) for four minutes, when the logic block is called, four minutes of time will already have accumulated. This time is applied to the timer when enabled, unless the timer is first reset.

Timing diagram



A. ENABLE goes high; timer starts accumulating.
B. Current value (CV) reaches preset value (PV); Q goes high. Timer continues to accumulate time until ENABLE goes low, RESET goes high or current value becomes equal to the maximum time.
C. RESET goes high; Q goes low, accumulated time is reset (CV=0).
D. RESET goes low; timer then starts accumulating again, as ENABLE is high.
E. ENABLE goes low; timer stops accumulating. Accumulated time stays the same.

F. ENABLE goes high again; timer continues accumulating time.
G. CV becomes equal to PV; Q goes high. Timer continues to accumulate time until ENABLE goes low, RESET goes high or CV becomes equal to the maximum time.
H. ENABLE goes low; timer stops accumulating time.

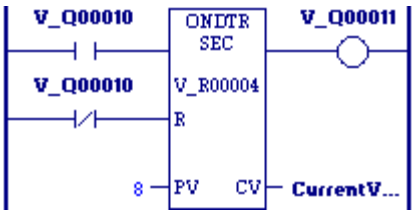
Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word **Cautions** ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic timer operation.
R	Power flow		When R is ON, it resets the Current Value (Word 1) to zero.
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE_INT instruction or an operator interface.
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(PACSystems and Series 90-70 only; optional.) Current Value of the timer; the same value as Word 1 in the ???? operand.

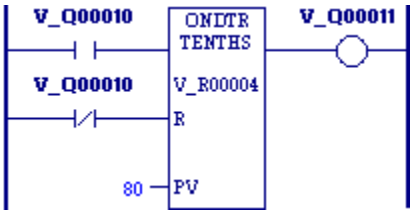
Example

An on-delay timer is used to create a signal (%Q0011) that turns on 8.0 seconds after %Q0010 turns on, and turns off when %Q0010 turns off.

PACSystems and Series 90-70 logic:



VersaMax and Series 90-30 logic:

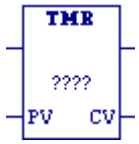


CPU Support

ONDTR_SEC is supported for PACSystems CPUs and Series 90-70 CPUs.
ONDTR_TENTHS and ONDTR_HUNDS are supported for all GE IP CPUs
ONDTR_THOUS is supported for VersaMax CPUs, PACSystems CPUs with firmware version 2.00 or later, and Series 90-30 CPUs.

On Delay Timer

(Not available for LD Diagnostic Logic Blocks (DLBs), whether active or inactive.)



Mnemonics:
TMR_SEC
TMR_TENTHS
TMR_HUNDS
TMR_THOUS

The CV operand is available only for PACSystems CPUs.

Operation

An On-Delay Timer (TMR) built-in function block instance increments while it receives power flow and resets to zero when power flow stops. The timer passes power after the specified interval PV (Preset Value) has elapsed, as long as power is received.
Time may be counted in various increments, depending on the CPU family:

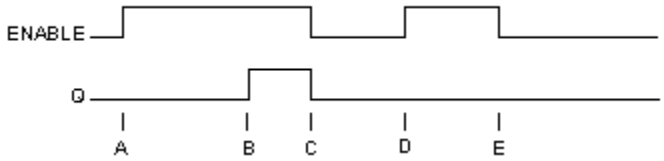
- **Seconds** (PACSystems and Series 90-70 CPUs)
- **Tenths** (0.1) of a second (all CPUs)
- **Hundredths** (0.01) of a second (all CPUs)
- **Thousandths** (0.001) of a second (VersaMax, PACSystems with firmware version 2.00 or later, and Series 90-30 CPUs).

The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.
When TMR receives power flow, it starts accumulating time (Current Value (CV)). CV is updated when TMR is encountered in the logic to reflect the total elapsed time since TMR was last reset.
This update occurs as long as the power flow input remains ON. When CV equals or exceeds the Preset Value (PV), TMR expires and begins passing power flow to the right. The timer continues accumulating time until the maximum value (32,767 time units) is reached.
When the power flow input transitions from ON to OFF, TMR stops accumulating time, CV is reset to zero, and Q is turned off.
Output Q is energized when TMR is enabled and PV ≤ CV.

Notes

- On Series 90 Micro Controller targets, a TMR instance sometimes prevents obtaining equality after a download or an upload. Use a Normally Open Contact (NOCON) somewhere in the block that has the timer and assign the #ALW_ON system variable to the NOCON.
- (Series 90-30.) Normal use of TMR is to invoke the built-in function block with a particular reference address exactly one time each scan. Do not invoke TMR with the same reference address more than once per scan (inappropriate accumulation of time would result). If TMR is used within a subroutine block, then ensure that the subroutine block is not called more than once per scan. Also, you should not program a JUMP around a TMR instance.
- (PACSystems and Series 90-70.) The best way to use an TMR built-in function block is to invoke it with a particular reference address exactly one time each scan. Do not invoke an TMR with the same reference address more than once per scan (inappropriate accumulation of time would result). When an TMR instance appears in an LD block, it accumulates time only once per scan. Subsequent calls to that same LD block within the same scan will have no effect on its TMR instances. Do not program a TMR instance with the same reference address in two different blocks. You should not program a JUMP around a TMR instance. Also, if you use recursion (that is,. having a block call itself either directly or indirectly), program the LD block so that it invokes the timer before it makes any recursive calls to itself. See Using OFDT, ONDTR, and TMR Timers in PACSystems and Series 90-70 Parameterized LD Blocks and OFDT, ONDTR, and TMR Timer Limitations in UDFBs.
- A TMR instance expires (passes power flow to the right) the first scan that it is enabled if the previous scan time was greater than PV.
- When a TMR instance is used in a logic block that is not called every scan, the instance accumulates time between calls to the logic block unless it is reset. This means that it operates like a timer operating in logic with a much slower sweep than the timer in the main logic block. For logic blocks that are inactive for a long time, a TMR instance should be programmed to allow for this catch-up feature. For example, if a TMR instance in a logic block is reset and the logic block is not called (is inactive) for 4 minutes, when the logic block is called, four minutes of time will already have accumulated. This time is applied to the timer when enabled, unless the timer is first reset.

Timing Diagram



A. ENABLE goes high; timer begins accumulating time.
B. CV reaches PV; Q goes high and timer continues accumulating time.
C. ENABLE goes low; Q goes low; timer stops accumulating time and CV is cleared.
D. ENABLE goes high; timer starts accumulating time.
E. ENABLE goes low before current value reaches PV; Q remains low; timer stops accumulating time and is cleared to zero (CV=0).

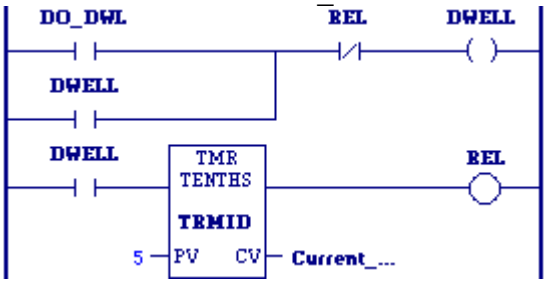
Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	one-dimensional WORD array of 3 words	R, P, L, W, symbolic	Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word

			Cautions ▪ Do not write to word 3 by any means. ▪ Overlapping reference addresses may cause erratic timer operation.
PV	INT variable or constant; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2. **Note:** Instead of using the PV parameter, you can write directly to word 2 by using a MOVE_INT instruction or an operator interface.
CV	INT variable; BOOL array of length 16 or more (restrictions apply)	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(PACSystems and Series 90-70 only; optional.) The current value of the timer; the same value as Word 1 in the ???? operand.

Example

An on-delay timer with address TMRID is used to control the length of time that a coil is on. This coil has been assigned the variable DWELL. When the normally open (momentary) contact DO_DWL is ON, coil DWELL is energized.
The contact of coil DWELL keeps coil DWELL energized (when contact DO_DWL is released) and also starts the timer TMRID. When TMRID reaches its preset value of five tenths of a second, coil REL energizes, interrupting the latched-on condition of coil DWELL. The contact DWELL interrupts power flow to TMRID, resetting its current value and de-energizing coil REL. The circuit is then ready for another momentary activation of contact DO_DWL.



CPU Support

TMR_SEC is supported for PACSystems CPUs and Series 90-70 CPUs.
TMR_TENTHS and TMR_HUNDS are supported for all GE IP CPUs.
TMR_THOUS is supported for VersaMax CPUs, Series 90-30 CPUs, and PACSystems CPUs with firmware version 2.00 or later.

Using OFDT, ONDTR, and TMR Timers in PACSystems and Series 90-70 Parameterized LD Blocks

Finding the Source Block

(Not available for LD Diagnostic Logic Blocks (DLBs), whether  active or  inactive.)

A parameterized LD block is a particular type of LD block that supports parameters. Special care must taken when programming OFDT, ONDTR, and TMR timers in PACSystems and Series 90-70 parameterized LD blocks. OFDT, ONDTR, and TMR in parameterized LD blocks can be programmed to track true real time as long as the guidelines and rules below are followed. If the guidelines and rules described here are not followed, then the operation of the OFDT, ONDTR, and TMR in parameterized LD blocks is undefined.

Note: These rules are not enforced by Machine Edition. It is your responsibility to ensure these rules are followed.

The best use of an OFDT, ONDTR, or TMR timer is to invoke it with a particular reference address exactly one time each scan. With parameterized LD blocks, it is important to use the appropriate reference address with the OFDT, ONDTR, or TMR and to call the parameterized LD block an appropriate number of times.

Note: The OFDT, ONDTR, and TMR timers also have limitations in user-defined function blocks (UDFBs), that is, blocks with a type of Function Block. See OFDT, ONDTR, and TMR Timer Limitations in PACSystems UDFBs.

Finding the Source Block

The source block is either the _MAIN block or the lowest logic block of type Block that appears above the parameterized LD block in the call tree. To determine the source block for a given parameterized LD block, determine which block invoked that parameterized LD block. If the calling block is _MAIN or of type Block, it is the source block. If the calling block is any other type (Parameterized block or UDFB), apply the same test to the calling block. Continue back up the call tree until the _MAIN block or a block of type Block is found. This is the source block for the parameterized LD block.

Programming OFDT, ONDTR, and TMR Timers in Parameterized LD Blocks

Different guidelines and rules apply depending on whether you want to use the parameterized LD block in more than one place in your logic.

Parameterized LD Block called from only one block

If your parameterized LD block that contains an OFDT, ONDTR, or TMR timer will be called from only one logic block, follow these rules:

1. Call the parameterized LD block exactly one time per execution of its source block.

2. Choose, for the OFDT, ONDTR, or TMR timer, a reference address that is not manipulated elsewhere. The reference address can be %R, %P, %L, %W, or a parameterized LD block formal parameter (word array passed by reference).

   Note: %L memory is the same %L memory available to a source block of type Block. %L memory corresponds to %P memory when the source block is _MAIN.

Parameterized LD Block called from multiple blocks

When calling the parameterized LD block from multiple blocks, it is imperative to separate the OFDT, ONDTR, or TMR timer reference address used by each call to the parameterized LD block. Even so, the timer may increment incorrectly, but to minimize the risk, follow these rules:

1. Call the parameterized LD block exactly one time per execution of each source block that it appears in.
2. Choose a %L reference address or parameterized LD block formal parameter for the OFDT, ONDTR, or TMR timer reference address. Do not use a %R, %P, or %W reference address.

Notes

- We strongly recommended a %L location, which is inherited from the parameterized LD block's source block. Each source block has its own reserved %L memory area, except the _MAIN block, which has a %P memory area instead. When the _MAIN block calls another block, the %P mappings from the _MAIN block are accessed by the called block as %L mappings.

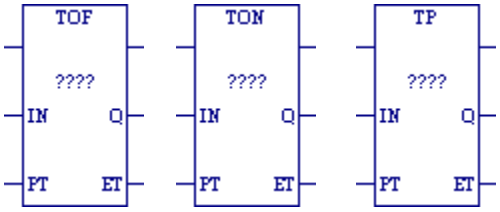
- If you use a parameterized LD block formal parameter (word array passed by reference), then the actual parameter that corresponds to this formal parameter must be a %L, %R, %P, or %W reference. If the actual parameter is a %R, %P, or %W reference, then a unique reference address must be used by each source block.

Recursion

If you use recursion, that is, if you have a block call itself either directly or indirectly, and your parameterized LD block contains an OFDT, ONDTR, or TMR timer built-in function block instance, then you must follow two additional rules:

1. Program the source block so that it invokes the parameterized LD block before making any recursive calls to itself.
2. Do not program the parameterized LD block to call itself directly.

TOF, TON, TP Timer Standard Function Blocks



Operation

An LD timer off delay (TOF) standard function block instance delays setting the output, Q, to OFF, for the preset time, PT, when you set the input, IN, to OFF.
An LD timer on delay (TON) standard function block instance delays setting the output, Q, to ON, when you set the input, IN, to ON, for the preset time, PT.
An LD timer pulse (TP) standard function block instance outputs a pulse of the preset time, PT, when you set the input, IN, to ON.

Notes

- Forcing or writing values to the IN, PT, Q, ET, ENO, or TI timer standard function block parameter may cause indeterminate results.
- TOF requires instance data of data type TOF; TON requires instance data of data type TON; TP requires instance data of data type TP.
- Instance data can be an input or member parameter of the current user-defined function block (UDFB) or parameterized block, or it can be a variable.
- When you assign instance data to a timer, its name replaces ???? inside the timer standard function block instance. For details, see Creating an instance of a function block.

Instance data required for the TOF, TON, and TP standard function block timers

Element	*Type*	*Public*	*Description*
IN	BOOL		Value of the input bit
PT	DINT		Input duration of time the timer measures time for
Q	BOOL	√	Value of the output bit
ET	DINT	√	Elapsed time the timer has been measuring time for
ENO	BOOL	√	Set to ON if the last call to the timer was successful
TI	BOOL	√	Set to ON while timer is measuring time

Notes

- You cannot write to an instance data element.
- You can read only a public element.
- See Function block instance variables in the References tab of the Feedback Zone.

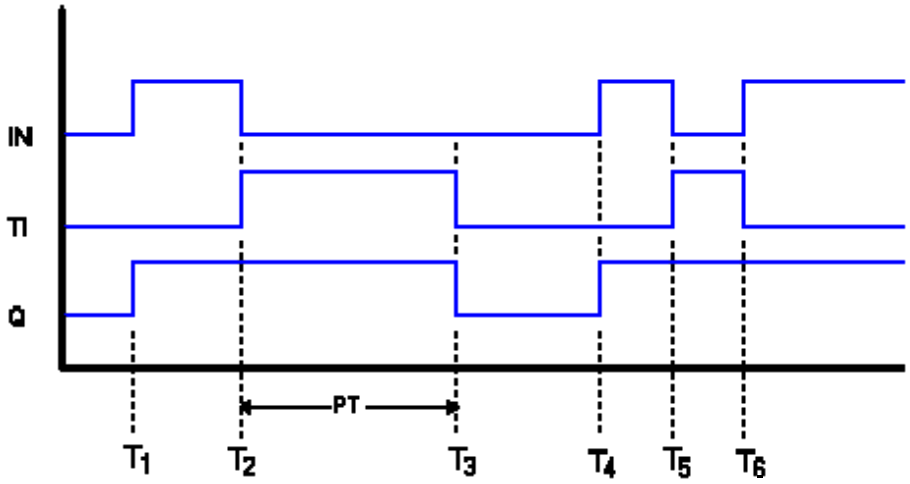
Operands

Operand	*Data Type*	*Memory Area*	*Description*
????	Instance data consisting of a structure variable or a parameter		(Required.) TOF requires instance data of data type TOF; TON requires instance data of data type TON; TP requires instance data of data type TP.
IN	LD: Power flow	Power flow	(Optional.) Input bit. If a transition occurs from OFF to ON, TON and TP begin to measure time. If a transition occurs from ON to OFF, TOF begins to measure time.
	FBD and ST: BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
PT	DINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) Input preset time in milliseconds. Indicates the amount of time the timer measures time before the timer sets Q to ON or OFF, depending on the timer type. When you set PT to zero, this resets the timer.
Q	LD: Power flow	Power flow	(Optional.) Output bit. TP: Set to ON while the timer is measuring time TON: Set to ON when the timer has completed TOF: Set to OFF when the timer has completed
	FBD and ST: BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	
ET	DINT variable	data flow, I, Q, M, T, G, R, P, L, AI, AQ, W, symbolic, I/O variable	(Optional.) Output elapsed time in milliseconds. Indicates the amount of time that the standard function block instance has been accumulating time for.

Note: If parameters are not assigned to the optional input operands (IN, PT), then the current values of the associated elements in the instance data are used.

TOF Timing Diagram

The following timing diagram illustrates the operation of the TOF standard function block.



T_1: When you set the timer input, IN, to ON, the timing bit, TI, remains set to OFF, the output enable bit, Q, is set to ON, and the elapsed time, ET, is reset to 0.

T_2: When you set the timer input, IN, to OFF, the timer starts to measure time (TI is set to ON), the output enable bit, Q, remains set to ON.

T_3: After the elapsed time, ET, equals the preset time, PT, the output enable bit, Q, is set to OFF, the timer stops measuring time (TI is set to OFF), and the elapsed time stays fixed at preset time (ET=PT).

T_4: When you set the timer input, IN, to ON, the timing bit, TI, remains set to OFF, the output enable bit, Q, is set to ON, and the elapsed time, ET, is set to zero.

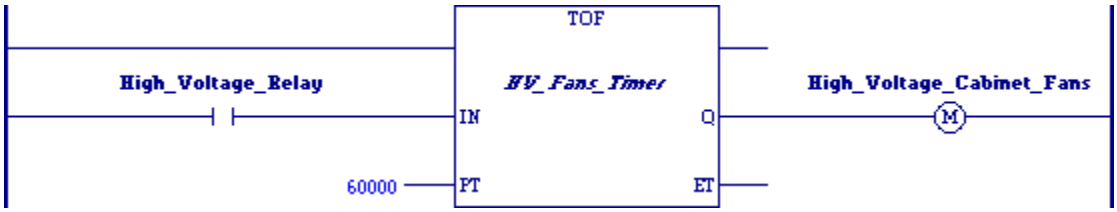
T_5: When you set the timer input, IN, to OFF, the timer starts to measure time (TI is set to ON), and the output enable bit, Q, remains set to ON.

T_6: When you set the timer input, IN, to ON before the elapsed time, ET, equals the preset time, PT, the timer stops to measure time (TI is set to OFF), the output enable bit, Q, remains set to ON, and the elapsed time, ET, is set to zero.

TOF Example

The following rung diagram illustrates how a timer off delay (TOF) standard function block instance can be used to keep the fans in a high voltage cabinet running for 1 minute (60,000 ms) after the high voltage is set to OFF, as well as keep them set to ON while high voltage is set to ON.

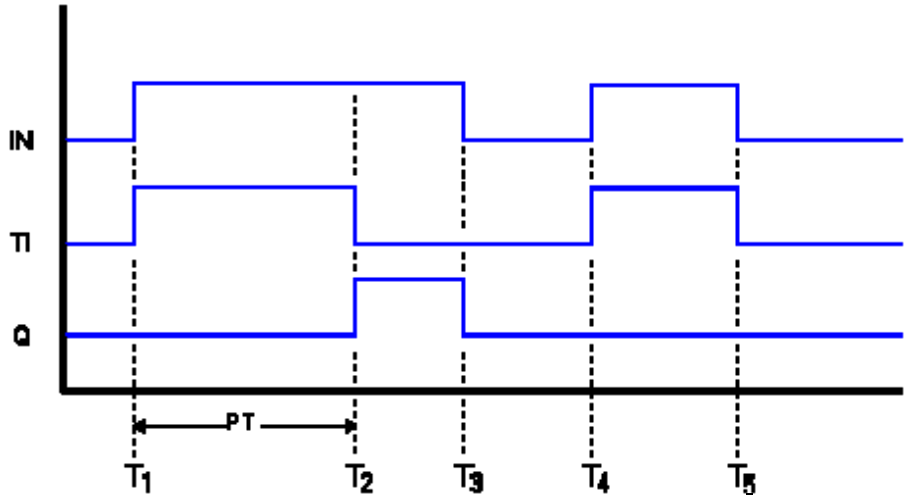
As long as 'High_Voltage_Relay' is set to ON, TOF sets the output, Q, and *High_Voltage_Cabinet_Fans*, to ON. When *High_Voltage_Relay* is set to OFF, TOF begins to measure time. One minute later, TOF and *High_Voltage_Cabinet_Fans* are set to OFF.



Note: The instance variable named *HV_Fans_Timer* is assigned to this TOF timer.

TON Timing Diagram

The following timing diagram illustrates the operation of the timer on delay standard function block.



T_1: When you set the timer input, IN, to ON, the timing bit, TI, is set to ON, and the timer begins to measure time (ET begins to increment). Q remains set to OFF.

T_2: After the elapsed time, ET, equals preset time, PT, the output enable bit, Q, is set to ON, and the elapsed time stays fixed at the preset time (ET=PT).

T_3: When you set the timer input, IN, to OFF, the output enable bit, Q, is set to OFF, and the elapsed time, ET, is set to zero.

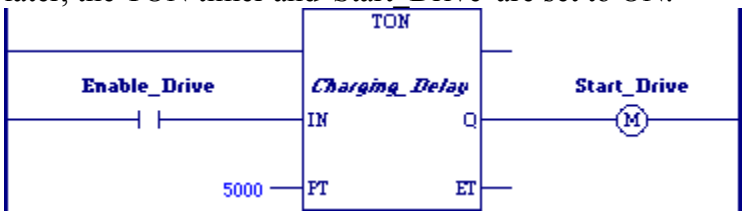
T_4: When you set the timer input, IN, to ON, the timing bit, TI, is set to ON, and the timer begins to measure time (ET increments).

T_5: When you set the timer input, IN, to OFF before the elapsed time, ET, equals the preset time, PT, the output enable bit, Q, remains set to OFF, and the elapsed time, ET, is set to zero.

TON Example

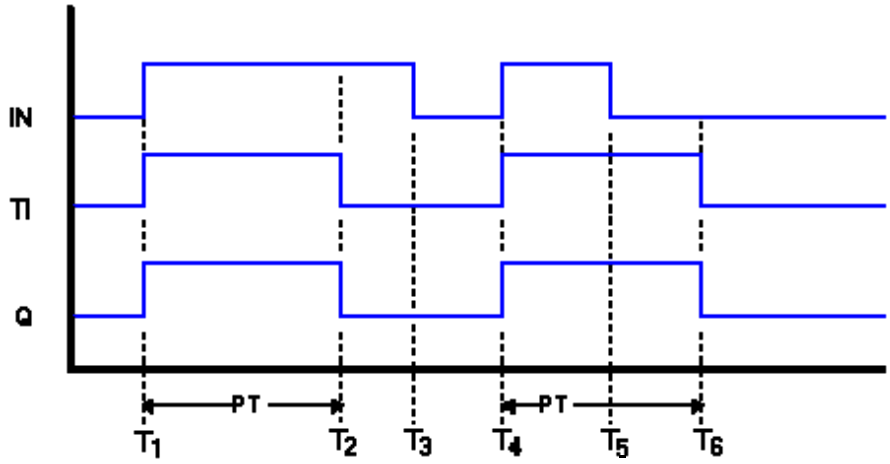
The following rung diagram illustrates how a timer on delay standard function block instance can be used to delay a start signal. A drive needs 5 seconds after it is enabled (powered) to charge up capacitors before it can be started.

When 'Enable_Drive' is set to ON, the TON timer starts to measure time. Five seconds later, the TON timer and 'Start_Drive' are set to ON.



TP Timing Diagram

The following timing diagram illustrates the operation of the timer pulse standard function block.



T_1: When you set the timer input, IN, to ON, the timing bit, TI, and the output enable bit, Q, are set to ON, and the timer starts timing.

T_2: After the elapsed time, ET, equals the preset time, PT, the output enable bit, Q, is set to OFF, the timer stops to measure time (TI is set to OFF), and the elapsed time stays fixed at preset time (ET=PT).

T_3: When you set the timer input, IN, to OFF, the elapsed time, ET, is set to zero.

T_4: When you set the timer input, IN, to ON, the timing bit, TI, and the output enable bit, Q, are set to ON.

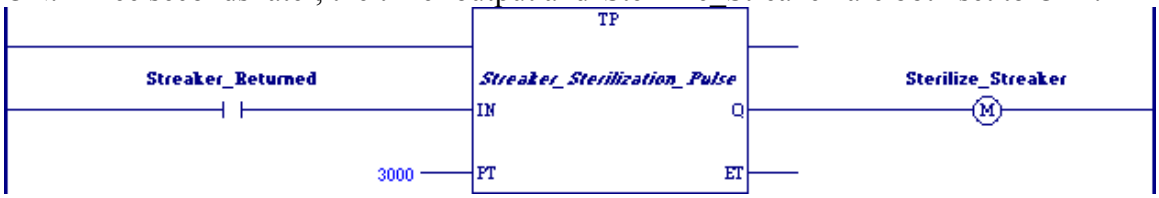
T_5: When you set the timer input, IN, to OFF, the timer continues to measure time (TI stays set to ON), and the output enable bit, Q, remains set to ON.

T_6: After the elapsed time, ET, equals the preset time, PT, the output enable bit, Q, is set to OFF, the timer stops to measure time (TI is set to OFF), and the elapsed time, ET, is set to zero because you have set the timer input, IN, to OFF.

TP Example

The following example demonstrates how a timer pulse standard function block instance can be used to set an output ON for a 3 second pulse. The Streaker is sterilized for 3 seconds after it is returned to its home position.

When 'Streaker_Returned' is set to ON, the TP timer and 'Sterilize_Streaker' are set to ON. Three seconds later, the timer output and 'Sterilize_Streaker' are both set to OFF.



Other Languages

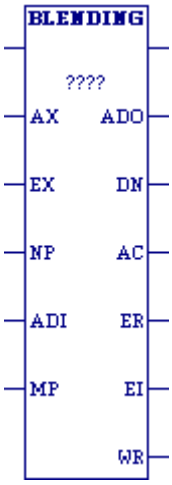
For details on the timer standard function block syntax in other languages, see FBD and ST.

VersaMax Micro Motion

20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later support the following motion instructions:

Instruction	*Description*
BLENDING	When power flow input is ON and rising edge is detected on EX, performs blending on axis AX in direction ADI for profile NP. The acceleration, deceleration, velocity, and number of pulses for profiles are specified in MP.
FIND_HOME	When power flow input is ON and rising edge is detected on EX, performs homing on axis AX in direction ADI, as per the acceleration, deceleration, and velocity specified in MP, the final home velocity FHV, and the home offset HO.
GO_HOME	When power flow input is ON and rising edge is detected on EX, performs homing on axis AX as per ramping acceleration ACC, ramping deceleration DEC, and go home velocity GHV.
JOGGING	When power flow input is ON, jogs axis AX forward if EF is set to ON or backward if EB is set to ON, as per ramping acceleration ACC, ramping deceleration DEC, and ramping velocity VEL.
STOP_MOTION	When power flow input is ON and rising edge is detected on EX, axis AX is stopped as per ramping deceleration DEC.

BLENDING



When power flow input is ON and a rising edge is detected on EX, a BLENDING built-in function block instance performs blending on axis AX in direction ADI for profile NP. The acceleration, deceleration, velocity, and distance for profiles are specified in MP. Each instance in logic should generally have its ???? operand set to a unique reference address that is not written to by any other process.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
????	One-dimensional WORD array of 2 words	R	The reference address of Word 1 of the function block instance data. ▪ Word 1: Control word ▪ Word 2: Error ID **Cautions** ▪ Do not write to words 1 or 2 by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
EN	Power		▪ 0: Does not start motion even if a rising edge is

<table>
<tr><td></td><td>flow</td><td></td><td>detected on EX. The control word is updated but not reflected to the outputs.
▪ 1: Execution starts when a rising edge is detected on EX. The Execute status is updated to the control word only.</td></tr>
<tr><td>AX</td><td>Constant</td><td></td><td>Axis. Channel number on which blending action is to be done.
Valid range: 1 through 4.</td></tr>
<tr><td>EX</td><td>Power flow</td><td></td><td>Execute. Rising edge is detected to start execution of the instruction. Parameters are also latched in at rising edge. If motion is going on and execute is re-triggered, it does not affect the current running motion. The Warning output parameter is set.</td></tr>
<tr><td>NP</td><td>BYTE variable or constant</td><td>AI, AQ, R</td><td>Number of profiles for which data is present.
Valid range: 1 through 4.</td></tr>
<tr><td>ADI</td><td>BOOL variable or constant</td><td>I, Q, M, G, T</td><td>Axis Direction Input.
▪ 0: Clockwise.
▪ 1: Counter-Clockwise.

Note: The actual direction depends on the field connections and drive settings.</td></tr>
<tr><td>MP</td><td>WORD Array</td><td>AI, AQ, R</td><td>Motion Parameters.
<table>
<tr><td>WORDs 1 and 2</td><td>DWORD value</td><td>Acceleration for profile 1.
Valid range:
▪ Axes 1, 2, 3: 10</td></tr>
<tr><td>WORDs 3 and 4</td><td>DWORD value</td><td>Deceleration for profile 1.
Valid range:
▪ Axes 1, 2, 3: 10</td></tr>
<tr><td>WORDs 5 and 6</td><td>DWORD value</td><td>Velocity for profile 1.
Valid range: 15 through 65,000.</td></tr>
<tr><td>WORDs 7 and 8</td><td>REAL value</td><td>Distance in user units for profile 1.</td></tr>
<tr><td>WORDs 9 and 10</td><td>DWORD value</td><td>Acceleration for profile 2.
Valid range:
▪ Axes 1, 2, 3: 10</td></tr>
</table></td></tr>
</table>

					through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs 11 and 12	DWORD value	Deceleration for profile 2. Valid range: ▪ Axes 1, 2, 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs 13 and 14	DWORD value	Velocity for profile 2. Valid range: 15 through 65,000.
			WORDs 15 and 16	REAL value	Distance in user units for profile 2.
			WORDs 17 and 18	DWORD value	Acceleration for profile 3. Valid range: ▪ Axes 1, 2, 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs 19 and 20	DWORD value	Deceleration for profile 3. Valid range: ▪ Axes 1, 2, 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs 21 and 22	DWORD value	Velocity for profile 3. Valid range: 15 through 65,000.
			WORDs 23 and 24	REAL value	Distance in user units for profile 3.
			WORDs 25 and 26	DWORD value	Acceleration for profile 4. Valid range: ▪ Axes 1, 2, 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs 27 and 28	DWORD value	Deceleration for profile 4. Valid range: ▪ Axes 1, 2, 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
			WORDs	DWORD	Velocity for profile 4.

			29 and 30	value	Valid range: 15 through 65,000.
			WORDs 31 and 32	REAL value	Distance in user units for profile 4.

Output Operands

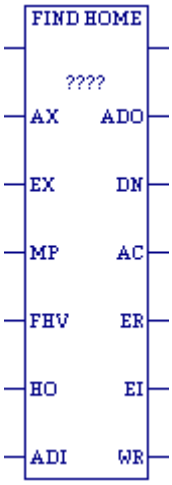
Operand	*Data Type*	*Memory Area*	*Description*
ENO	Power flow		Power flow output.
ADO	BOOL variable	%Q5 through %Q24	Axis Direction Output. Where output direction is to be outputted. ▪ 0: Clockwise. ▪ 1: Counter-Clockwise. **Note:** The actual direction depends on the field connections and drive settings.
DN	BOOL variable	I, Q, M, G, T	Done. Set to 0 when any of the following conditions is met: ▪ When the EX operand is a pulse, DN is set to 0 after 1 scan after completion of motion. ▪ The EX operand is set to 0. Set to 1 when any of the following conditions is met: ▪ Motion is complete on the channel. ▪ When the EX operand is a pulse, DN is set to 1 for one scan after completion of motion. ▪ When the EX operand is continuously set to 1, DN is set to 1 after completion of motion until EX is set to 0.
AC	BOOL variable	I, Q, M, G, T	Active. Set to 0 when any of the following conditions is met: ▪ Motion has stopped on the channel. ▪ The ER bit is set to 1. ▪ The HSC channel enable bit is set to 0. Set to 1 when blending is in progress.
ER	BOOL variable	I, Q, M, G, T	Error. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when an error has occurred in the instruction.
EI	WORD variable	AI, AQ, R	▪ When ER is set to 1, EI is the Error ID. ▪ When WR is set to 1, EI is the Warning ID.
WR	BOOL	I, Q, M, G,	(Optional.) Warning.

	variable	T	▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when a warning has occurred in the instruction.

CPU Support

BLENDING is supported for 20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later.

FIND_HOME



When power flow input is ON and a rising edge is detected on EX, a FIND_HOME built-in function block instance performs homing on axis AX in direction ADI, as per the acceleration, deceleration, and velocity specified in MP, the final home velocity FHV, and the home offset HO.
Each instance in logic should generally have its ???? operand set to a unique reference address that is not written to by any other process.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
????	One-dimensional WORD array of 2 words	R	The reference address of Word 1 of the function block instance data. ▪ Word 1: Control word ▪ Word 2: Error ID **Cautions** ▪ Do not write to words 1 or 2 by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*

<table>
<tr><td>EN</td><td>Power flow</td><td></td><td>

0: Does not start motion even if a rising edge is detected on EX. The control word is updated but not reflected to the outputs.
1: Execution starts when a rising edge is detected on EX. The Execute status is updated to the control word only.

</td></tr>
<tr><td>AX</td><td>Constant</td><td></td><td>Axis. Channel number on which homing action is to be done.
Valid range: 1 through 4.</td></tr>
<tr><td>EX</td><td>Power flow</td><td></td><td>Execute. Rising edge is detected to start execution of the instruction. Parameters are also latched in at rising edge. If motion is going on and execute is re-triggered, it does not affect the current running motion. The Warning output parameter is set.</td></tr>
<tr><td>MP</td><td>WORD array of 6 WORDs</td><td>AI, AQ, R</td><td>Motion Parameters.
<table>
<tr><td>WORDs 1 and 2</td><td>DWORD value</td><td>Homing acceleration.
Valid range:

Axes 1, 2, 3: 10 through 1,000,000.
Axis 4: 90 through 1,000,000.

</td></tr>
<tr><td>WORDs 3 and 4</td><td>DWORD value</td><td>Homing deceleration.
Valid range:

Axes 1, 2, 3: 10 through 1,000,000.
Axis 4: 90 through 1,000,000.

</td></tr>
<tr><td>WORDs 5 and 6</td><td>DWORD value</td><td>Find Home velocity.
Valid range: 15 through 65,000. The sum of velocities on all four channels must not exceed 65,000.</td></tr>
</table>
</td></tr>
<tr><td>FHV</td><td>DWORD variable or constant</td><td>AI, AQ, R</td><td>Final Home Velocity. Velocity at which the next cycles after the Home switch is activated are carried out.
Valid range: 1 through 65,000.</td></tr>
<tr><td>HO</td><td>REAL variable or constant</td><td>AI, AQ, R</td><td>Home Offset. Offset at which motion has to be stopped.
Maximum value: the maximum 32-bit value.</td></tr>
<tr><td>ADI</td><td>BOOL variable or constant</td><td>I, Q, M, G, T</td><td>Axis Direction Input.

0: Clockwise.
1: Counter-Clockwise.

Note: The actual direction depends on the field connections and</td></tr>
</table>

			drive settings.

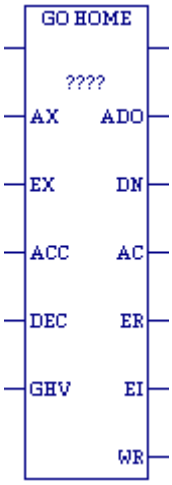
Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
ENO	Power flow		Power flow output. Set to the same value as EN.
ADO	BOOL variable	%Q5 through %Q24	Axis Direction Output. Where output direction is to be set. ▪ 0: Clockwise. ▪ 1: Counter-Clockwise. **Note:** The actual direction depends on the field connections and drive settings.
DN	BOOL variable	I, Q, M, G, T	Done. Set to 0 when any of the following conditions is met: ▪ When the EX operand is a pulse, DN is set to 0 after 1 scan after completion of motion. ▪ The EX operand is set to 0. Set to 1 when any of the following conditions is met: ▪ Motion is complete on the channel. ▪ When the EX operand is a pulse, DN is set to 1 for one scan after completion of motion. ▪ When the EX operand is continuously set to 1, DN is set to 1 after completion of motion until EX is set to 0.
AC	BOOL variable	I, Q, M, G, T	Active. Set to 0 when any of the following conditions is met: ▪ Motion has stopped on the channel. ▪ The ER bit is set to 1. ▪ The HSC channel enable bit is set to 0. Set to 1 when homing is in progress.
ER	BOOL variable	I, Q, M, G, T	Error. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when an error has occurred in the instruction.
EI	WORD variable	AI, AQ, R	▪ When ER is set to 1, EI is the Error ID. ▪ When WR is set to 1, EI is the Warning ID.
WR	BOOL variable	I, Q, M, G, T	(Optional.) Warning. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when a warning has occurred in the instruction.

CPU Support

FIND_HOME is supported for 20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later.

GO_HOME



When power flow input is ON and a rising edge is detected on EX, a GO_HOME built-in function block instance performs homing on axis AX as per ramping acceleration ACC, ramping deceleration DEC, and go home velocity GHV.
Each instance in logic should generally have its ???? operand set to a unique reference address that is not written to by any other process.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
????	One-dimensional WORD array of 2 words	R	The reference address of Word 1 of the function block instance data. ▪ Word 1: Control word ▪ Word 2: Error ID **Cautions** ▪ Do not write to words 1 or 2 by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
EN	Power flow		▪ 0: Does not start motion even if a rising edge

			is detected on EX. The control word is updated but not reflected to the outputs. ▪ 1: Execution starts when a rising edge is detected on EX. The Execute status is updated to the control word only.
AX	Constant		Axis. Channel number on which homing action is to be done. Valid range: 1 through 4.
EX	Power flow		Execute. Rising edge is detected to start execution of the instruction. Parameters are also latched in at rising edge. If EX is triggered when motion is in progress, then this trigger is ignored and the Warning output parameter is set.
ACC	DWORD variable or constant	AI, AQ, R	Acceleration for ramping. Valid range: ▪ Axes 1, 2, and 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
DEC	DWORD variable or constant	AI, AQ, R	Deceleration for ramping. Valid range: ▪ Axes 1, 2, and 3: 10 through 1,000,000. ▪ Axis 4: 90 through 1,000,000.
GHV	DWORD variable or constant	AI, AQ, R	Go Home Velocity. Velocity at which ramping is to be carried out. Valid range: 15 through 65,000. The sum of velocities on all four channels must not exceed 65,000.

Output Operands

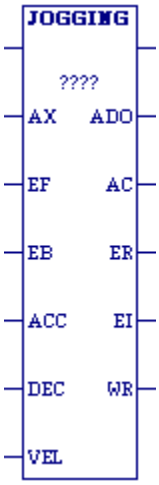
Operand	*Data Type*	*Memory Area*	*Description*
ENO	Power flow		Power flow output.
ADO	BOOL variable	%Q5 through %Q24	Axis Direction Output. Where output direction is to be set. Flow type not supported. ▪ 0: Clockwise. ▪ 1: Counter-clockwise. **Note:** The actual direction depends on the field connections and drive settings.
DN	BOOL variable	I, Q, M, G, T	Done. Set to 0 when any of the following conditions is met: ▪ When the EX operand is a pulse, DN is set to 0 after 1 scan after completion of motion. ▪ The EX operand is set to 0.

			Set to 1 when any of the following conditions is met: ▪ Motion is complete on the channel. ▪ When the EX operand is a pulse, DN is set to 1 for one scan after completion of motion. ▪ When the EX operand is continuously set to 1, DN is set to 1 after completion of motion until EX is set to 0.
AC	BOOL variable	I, Q, M, G, T	Active. Set to 0 when any of the following conditions is met: ▪ Motion has stopped on the channel. ▪ The ER bit is set to 1. ▪ The HSC channel enable bit is set to 0. Set to 1 when homing is in progress.
ER	BOOL variable	I, Q, M, G, T	Error. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when an error has occurred in the instruction.
EI	WORD variable	AI, AQ, R	▪ When ER is set to 1, EI is the Error ID. ▪ When WR is set to 1, EI is the Warning ID.
WR	BOOL variable	I, Q, M, G, T	(Optional.) Warning. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when a warning has occurred in the instruction.

CPU Support

GO_HOME is supported for 20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later.

JOGGING



When power flow input is ON, a JOGGING built-in function block instance jogs axis AX forward if EF is set to ON or backward if EB is set to ON, as per ramping acceleration ACC, ramping deceleration DEC, and ramping velocity VEL. IF both EF and EB are set to ON, then jogging doesn't start and generates an error. If jogging started because EF was set to ON, then setting EB to ON while the motion is running forces the jog to stop by logging an error.
Each instance in logic should generally have its ???? operand set to a unique reference address that is not written to by any other process.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
????	One-dimensional WORD array of 2 words	R	The reference address of Word 1 of the function block instance data. ▪ Word 1: Control word ▪ Word 2: Error ID **Cautions** ▪ Do not write to words 1 or 2 by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
EN	Power flow		▪ 0: Does not start motion if EF or EB is set to ON. Instance data is updated but not reflected to outputs. ▪ 1: Output depends on EF and EB.
AX	Constant		Axis. Channel number on which jogging is to be done. Valid range: 1 through 4.
EF	Power flow		Enable Forward. ▪ 0: Stop Jog motion if jog is in progress. ▪ 1: Start Jog motion if EN is set to ON.
EB	Power flow		Enable Backward. ▪ 0: Stop Jog motion if jog is in progress. ▪ 1: Start Jog motion if EN is set to ON.
ACC	DWORD variable or constant	AI, AQ, R	Acceleration for ramping. Valid range: ▪ For channels 1, 2, and 3: 10 through 1,000,000. ▪ For channel 4: 90 through 1,000,000.
DEC	DWORD variable or constant	AI, AQ, R	Deceleration for ramping. Valid range: ▪ For channels 1, 2, and 3: 10 through 1,000,000. ▪ For channel 4: 90 through 1,000,000.
VEL	DWORD variable or constant	AI, AQ, R	Velocity at which ramping is to be carried out. Valid range: 15 through 65,000. The sum of the velocities on all four channels must not exceed 65,000.

Output Operands

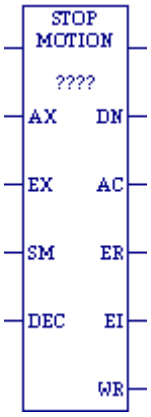
Operand	*Data Type*	*Memory Area*	*Description*
ENO	Power flow		Power flow output. Set to the same value as EN.
ADO	BOOL variable	%Q5 through %Q24	Axis Direction Output. Where output direction is to be outputted. Flow type not supported. ▪ 0: Clockwise. ▪ 1: Counter-clockwise. **Note:** The actual direction depends on the field connections and drive settings.
AC	BOOL variable	I, Q, M, G, T	Active. Set to 0 when any of the following conditions is met:

			▪ Motion has stopped on the channel. ▪ The ER bit is set to 1. ▪ The EF and EB bits are set to 0. ▪ The HSC channel enable bit is set to 0. Set to 1 when jogging is in progress.
ER	BOOL variable	I, Q, M, G, T	Error. ▪ Set to 0 when the EF or EB bit transitions from 0 to 1. ▪ Set to 1 when an error has occurred in the instruction.
EI	WORD variable	AI, AQ, R	▪ When ER is set to 1, EI is the Error ID. ▪ When WR is set to 1, EI is the Warning ID.
WR	BOOL variable	I, Q, M, G, T	(Optional.) Warning. ▪ Set to 0 when the EF or EB bit transitions from 0 to 1. ▪ Set to 1 when a warning has occurred in the instruction.

CPU Support

JOGGING is supported for 20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later.

STOP_MOTION



When power flow input is ON and a rising edge is detected on EX, a STOP_MOTION built-in function block instance stops axis AX. Optionally, you can specify a ramping deceleration (DEC).
Each instance in logic should generally have its ???? operand set to a unique reference address that is not written to by any other process.

Operands

Instance Data

Operand	*Data Type*	*Memory Area*	*Description*
????	One-dimensional WORD array of 2 words	R	The reference address of Word 1 of the function block instance data. ▪ Word 1: Control word ▪ Word 2: Error ID **Cautions** ▪ Do not write to words 1 or 2 by any means. ▪ Overlapping reference addresses may cause erratic instance operation.

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
EN	Power flow		▪ 0: The function block instance is not executed even if a rising edge is detected on EX. The control word is updated but not

			reflected to the outputs. ▪ 1: Execution starts when a rising edge is detected on EX. The Execute status is updated to the control word only.
AX	Constant		Axis. Channel number on which motion is to be stopped. Valid range: 1 through 4.
EX	BOOL variable or constant		Execute. If EN is set to 1 and a rising edge is detected on EX, the instance begins executing. Parameters are also latched in at rising edge.
SM	BOOL variable or constant	I, Q, M, G, T	Stop Mode. ▪ 0: Decelerate and stop (normal stop mode). ▪ 1: Stop immediately.
DEC	DWORD variable or constant	AI, AQ, R	(Optional.) Deceleration for ramping. Valid range: ▪ For channels 1, 2, and 3: 10 through 1,000,000. ▪ For channel 4: 90 through 1,000,000.

Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
ENO	Power flow		Power flow output
DN	BOOL variable	I, Q, M, G, T	Done. Set to 0 when any of the following conditions is met: ▪ When the EX operand is a pulse, DN is set to 0 after 1 scan after motion has stopped. ▪ The EX operand is set to 0. Set to 1 when any of the following conditions is met: ▪ Motion has stopped on the channel. ▪ When the EX operand is a pulse, DN is set to 1 for one scan after motion has stopped. ▪ When the EX operand is continuously set to 1, DN is set to 1 after motion has stopped until EX is set to 0.
AC	BOOL variable	I, Q, M, G, T	Active. Set to 0 when any of the following conditions is met: ▪ Motion has stopped. ▪ The ER bit is set to 1. ▪ The HSC channel enable bit is set to 0. Set to 1 when motion is being stopped.
ER	BOOL	I, Q, M,	Error.

	variable	G, T	▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when an error has occurred in the instruction.
EI	WORD variable	AI, AQ, R	▪ When ER is set to 1, EI is the Error ID. ▪ When WR is set to 1, EI is the Warning ID.
WR	BOOL variable	I, Q, M, G, T	(Optional.) Warning. ▪ Set to 0 when the EX operand transitions from 0 to 1. ▪ Set to 1 when a warning has occurred in the instruction.

CPU Support

STOP_MOTION is supported for 20-pt, 40-pt, and 64-pt VersaMax Micro IC200UDD CPUs with firmware version 3.60 or later.

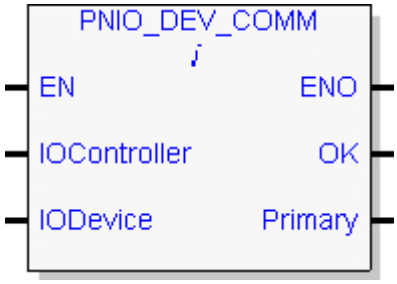
Index

Addendum

PNIO_DEV_COMM

LD

FBD



ST

Formal convention:

```
PNIO_DEV_COMM(IOController := [input], IODevice := [input], OK => [output], Primary => [output]);
```

Tip: To spare yourself some typing, drag the instruction from the Toolchest's LD Instructions or FBD Instructions drawer to your ST logic. All the input and output names are inserted into your ST logic. You need only to supply the variable names or constants and remove the optional inputs or outputs you don't want to use.

Operation

PNIO_DEV_COMM verifies whether the Profinet Controller (PNC) identified by input IOController and the Profinet IO-device identified by input IODevice are communicating with one another. If so, output OK is set to On; otherwise, it is set to Off.

When PNIO_DEV_COMM is used in a Hot Standby (HSB) redundancy system, if OK is set to On, PNIO_DEV_COMM verifies whether the PNC is the device's Primary IO-Controller. If so, the Primary output is set to On; otherwise, the output is set to Off.

When PNIO_DEV_COMM is not used in an HSB redundancy system, Primary is set to the same value as OK.

Operands

Input Operands

Operand	*Data Type*	*Memory Area*	*Description*
i	(FBD only.) Solve order for the instruction.		
Power flow (LD)			When On, PNIO_DEV_COMM executes. When Off, PNIO_DEV_COMM does not execute.

EN (FBD)	BOOL variable or BOOL system variable	I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	Enable input. When On, PNIO_DEV_COMM solves. When Off, PNIO_DEV_COMM does not solve.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
IOController	PNIO_CONTROLLER_REF variable	symbolic, I/O variable. LD and FBD also support data flow.	Variables used to identify respectively the Profinet Controller and Profinet device whose intercommunication status is to be reported
IODevice	PNIO_DEVICE_REF variable	symbolic, I/O variable. LD and FBD also support data flow.	

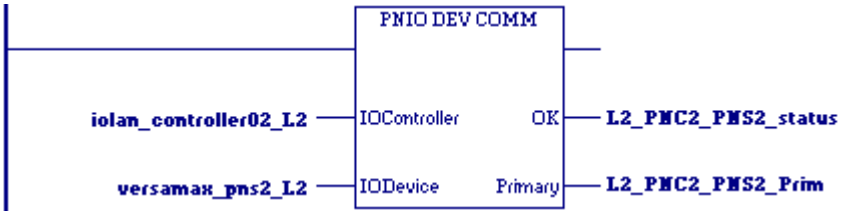
Output Operands

Operand	*Data Type*	*Memory Area*	*Description*
Power flow (LD; optional.)			There is power flow when PNIO_DEV_COMM executes successfully.
ENO (FBD; optional.)	BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, discrete symbolic, discrete I/O variable	ENO is set to On if EN is On.
	Bit reference in non-BOOL variable	R, P, L, AI, AQ, W, I, Q, M, T, G, non-discrete symbolic, non-discrete I/O variable	
OK	▪ LD: Power flow ▪ FBD and ST: BOOL variable		(Optional.) Set to On if the specified IO Controller and IO device are currently communicating.

Primary	▪ LD: Power flow ▪ FBD and ST: BOOL variable		(Optional.) Set to On if the PNC is the Primary IO Controller of the device in an HSB redundancy system. Requires communications with the device (OK is set to On). In a simplex system, Primary is set to On if OK is set to On.

Example

PNIO_DEV_COMM is used in an HSB redundancy system. It verifies whether the Profinet Controller (PNC) identified by the variable iolan_controller02_L2 and the Profinet device identified by the variable versamax_pns2_L2 are currently communicating with one another. If so, the BOOL variable L2_PNC2_PNS2_Prim is set to On. PNIO_DEV_COMM also verifies whether the identified PNC is the primary IO-Controller of the device; if so, the BOOL variable L2_PNC2_PNS2_status is set to On.



CPU Support

PNIO_DEV_COMM is supported for PACSystems CPUs with firmware version 7.00 or later.