Technical Information

Functions of VJ, M, and WXT Free-Program Computing Units

JUXTA

TI 231-01E

YOKOGAWA

Yokogawa Electric Corporation

TI 231-01E
© Copyright Feb.2000 (MCC)
4th Edition Dec. 2005 (KP)

Functions of VJ, M, and WXT Free-Program Computing Units

TI 231-01E 4th Edition

CONTENTS

Blank Page

1. Introduction

The VJ, M Series, and WXT free-program computing units for the JUXTA signal conditioners receive signals such as voltage, current, and contact, apply various computing functions to them, and then convert them into isolated DC voltage, DC current, or contact signals.

The computing units are available in six models that differ depending on their series and input/output specifications. Each one of them has an interface circuit for communication with a Handy Terminal and can perform the following:

- Setting computation programs
- Entering computation parameters
- Setting zero and span of the input range
- Monitoring input and output values

Table 1.1 Models of Free-Program Computing Units

Series	Model	Description
M Series (Plug-in type)	MXD-A	One analog input, one contact input, one isolated analog output, and one isolated contact output
	MXS-A	One analog input and two isolated analog outputs
	MXT-A	Three analog inputs and one isolated analog output
VJ Series (Compact plug-in type)	VJXS-A	One analog input and two isolated analog outputs
	VJX7-A	One analog input and one isolated analog output Option (Can select from second isolated analog output, communication output, or two contact outputs)
WXT (Front-panel terminal type)	WXT-A	Three analog inputs and one isolated analog output

In addition, it is possible to collectively set and load setting data and programs from a PC using the VJ77 Parameter Setting Tool.

2. Block Diagram and Terminal Arrangement

2.1 MXD Free-Program Computing Unit

2.1.1 Block Diagram

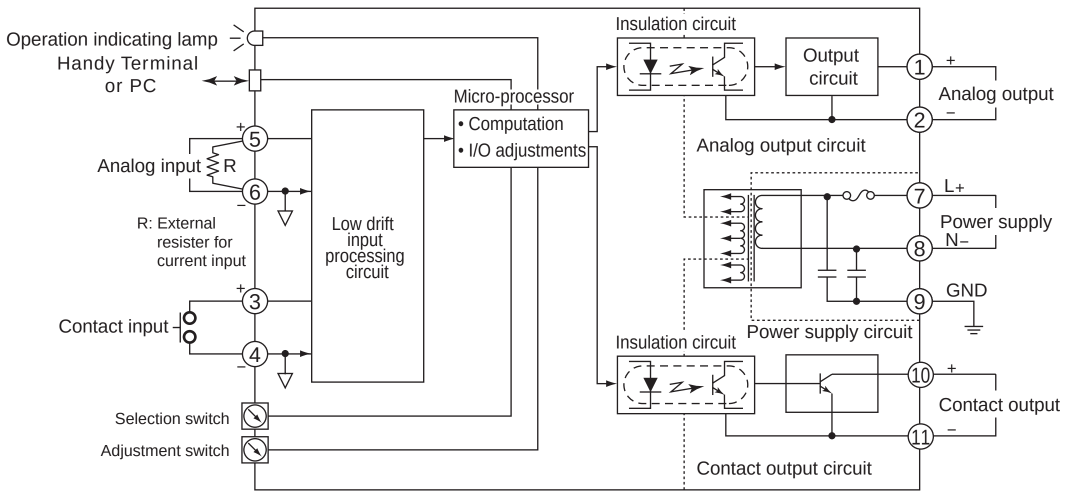
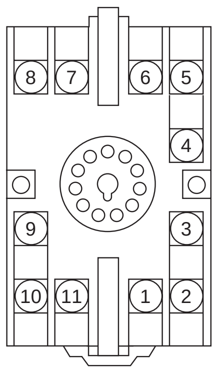


Figure 2.1.1 MXD Free-Program Computing Unit Block Diagram

2.1.2 Terminal Assignments



1	Analog Output	(+)
2	Analog Output	(–)
3	Contact Input	(+)
4	Contact Input	(–)
5	Analog Input	(+)
6	Analog Input	(–)
7	Power Supply	(L+)
8	Power Supply	(N–)
9	GND	(G)
10	Contact Output	(+)
11	Contact Output	(–)

Figure 2.1.2 MXD Free-Program Computing Unit Terminal Arrangement

2.2 MXS Free-Program Computing Unit

2.2.1 Block Diagram

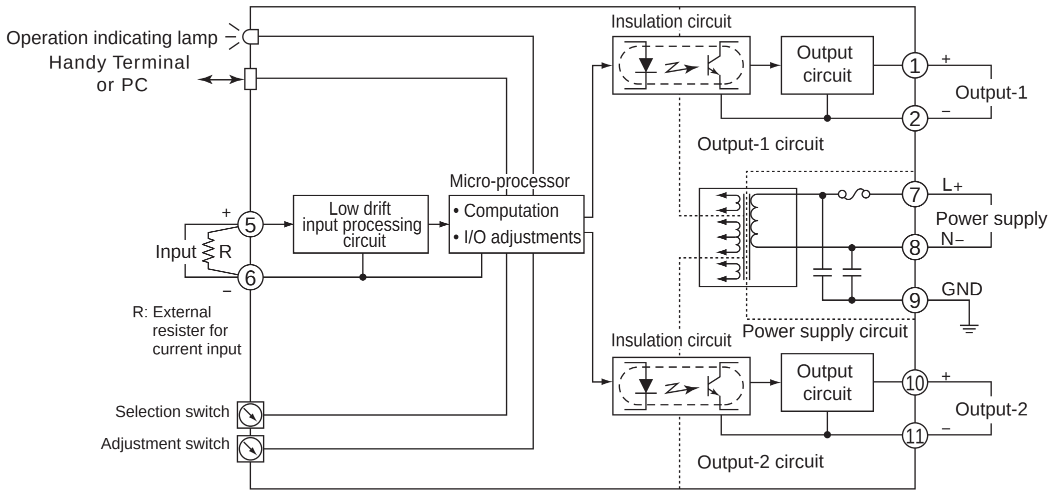
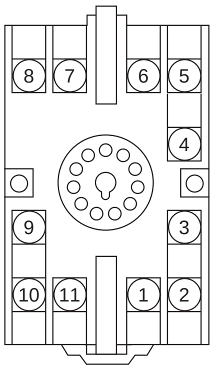


Figure 2.2.1 MXS Free-Program Computing Unit Block Diagram

2.2.2 Terminal Assignments

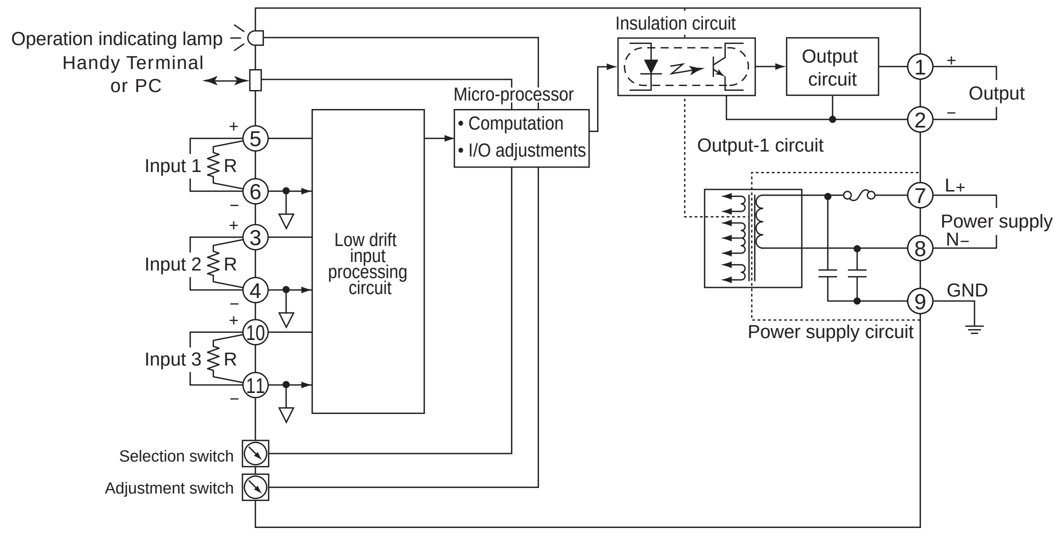


1	Output 1	(+)
2	Output 1	(–)
3	N.C.	
4	N.C.	
5	Input	(+)
6	Input	(–)
7	Power Supply	(L+)
8	Power Supply	(N–)
9	GND	(G)
10	Output 2	(+)
11	Output 2	(–)

Figure 2.2.2 MXS Free-Program Computing Unit Terminal Arrangement

2.3 MXT Free-Program Computing Unit

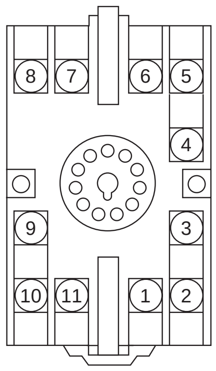
2.3.1 Block Diagram



R: External resister for current input

Figure 2.3.1 MXT Free-Program Computing Unit Block Diagram

2.3.2 Terminal Assignments



1	Output	(+)
2	Output	(–)
3	Input 2	(+)
4	Input 2	(–)
5	Input 1	(+)
6	Input 1	(–)
7	Power Supply	(L+)
8	Power Supply	(N–)
9	GND	(G)
10	Input 3	(+)
11	Input 3	(–)

Figure 2.3.2 MXT Free-Program Computing Unit Terminal Arrangement

2.4 VJXS Free-Program Computing Unit

2.4.1 Block Diagram

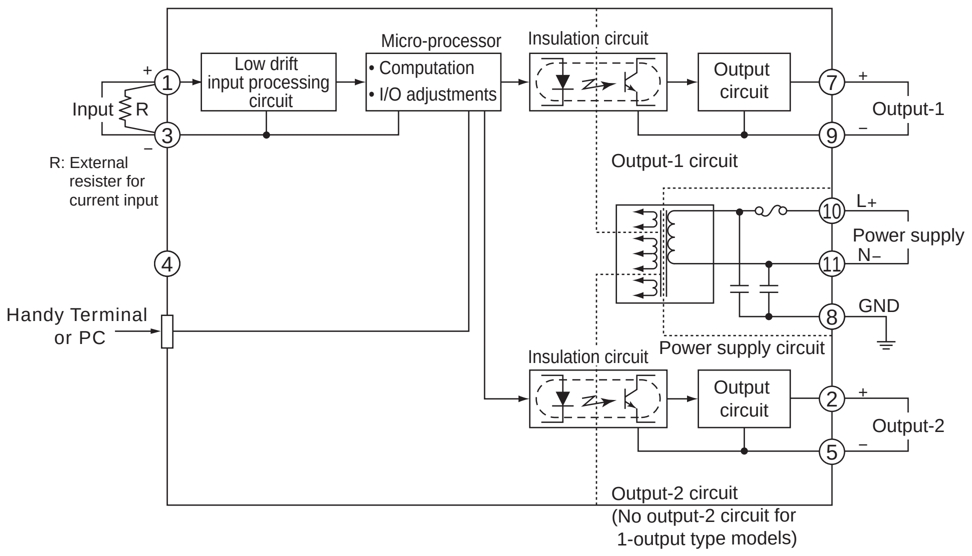
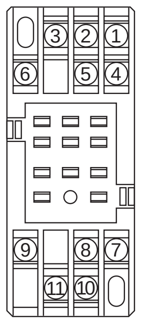


Figure 2.4.1 VJXS Free-Program Computing Unit Block Diagram

2.4.2 Terminal Assignments



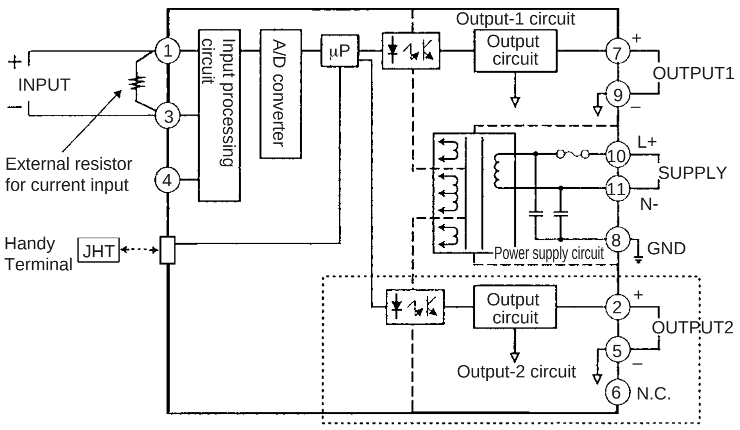
1	Input	(+)
2	Output 2	(+)
3	Input	(–)
4	N.C.	
5	Output 2	(–)
6	N.C.	
7	Output1	(+)
8	GND	(G)
9	Output1	(–)
10	Power Supply	(L+)
11	Power Supply	(N–)

Note: For 1-output type models, the output 2 terminals are not connected.

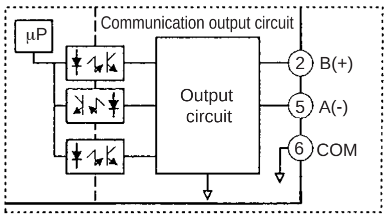
Figure 2.4.2 VJXS Free-Program Computing Unit Terminal Arrangement

2.5 VJX7 Free-Program Computing Unit

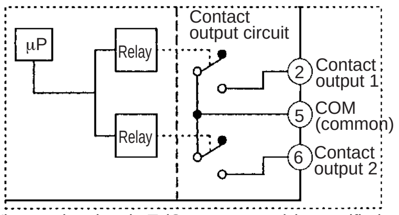
2.5.1 Block Diagram



When optional code A or 6 (Analog output) is specified.



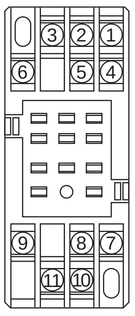
When optional code P (RS-485 output) is specified.



When optional code T (Contact output) is specified.

Figure 2.5.1 VJX7 Free-Program Computing Unit Block Diagram

2.5.2 Terminal Assignments

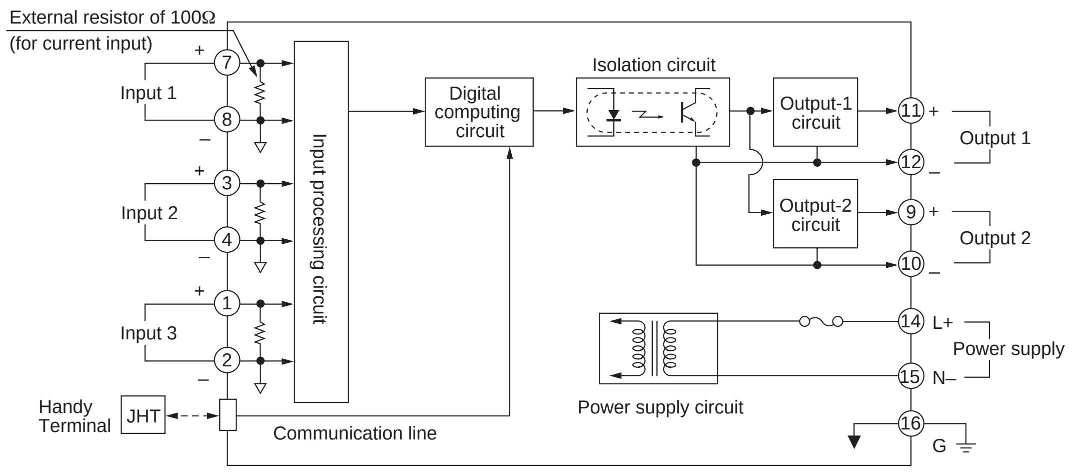


	No Optional Code	Optional Code		
		A, 6: Analog output	P:Communication output	T: Contact output
1	Input (+)	Input (+)	Input (+)	Input (+)
2	N.C.	Output 2 (+)	RS-485 B (+)	Contact Output 1
3	Input (–)	Input (–)	Input (–)	Input (–)
4	N.C.	N.C.	N.C.	N.C.
5	N.C.	Output 2 (–)	RS-485 A (–)	Contact Output COM
6	N.C.	N.C.	RS-485 COM	Contact Output 2
7	Output 1 (+)	Output 1 (+)	Output 1 (+)	Output 1 (+)
8	GND	GND	GND	GND
9	Output 1 (–)	Output 1 (–)	Output 1 (–)	Output 1 (–)
10	Power Supply (L+)	Power Supply (L+)	Power Supply (L+)	Power Supply (L+)
11	Power Supply (N–)	Power Supply (N–)	Power Supply (N–)	Power Supply (N–)

Figure 2.5.2 VJX7 Free-Program Computing Unit Terminal Arrangement

2.6 WXT Free-Program Computing Unit

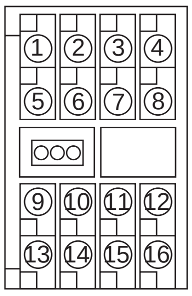
2.6.1 Block Diagram



Note: The optional output 2 should be specified by code /D0 and has the same signal as the output 1.

Figure 2.6.1 WXT Free-Program Computing Unit Block Diagram

2.6.2 Terminal Assignments



1	Input 3 (+)	9	Output 2 (+)
2	Input 3 (–)	10	Output 2 (–)
3	Input 2 (+)	11	Output 1 (+)
4	Input 2 (–)	12	Output 1 (–)
5	N.C.	13	N.C.
6	N.C.	14	Power Supply (L+)
7	Input 1 (+)	15	Power Supply (N–)
8	Input 1 (–)	16	GND (G)

Figure 2.6.2 WXT Free-Program Computing Unit Terminal Arrangement

3. Operation of Computation Program

The signal conditioner has the computing feature built-in as a set of library commands. These commands can be combined to create a specific computation function. The program structure of a computing unit is similar to the one used with calculators with a programming function. The computation program is expressed in the reverse Polish logic notation method without parentheses, which makes writing a program very easy.

3.1 Basic Flow

Figure 3.1.1 shows signal and processing flows in the computing unit.

(1) Input Conversion

The input signals are automatically converted into internal data by the input converter and stored in the corresponding input data registers.

(2) Executing User Program

After the input conversion, the computing unit calls up a user program, loads various data, and executes the objective computation. The results are stored in the corresponding data registers.

(3) Output Conversion

The computing unit stores the data in output data registers, automatically converts them into analog or contact signals, and then outputs them.

The steps (1) to (3) described above are periodically executed at computation intervals (selected from 50ms, 100 ms or 200 ms).

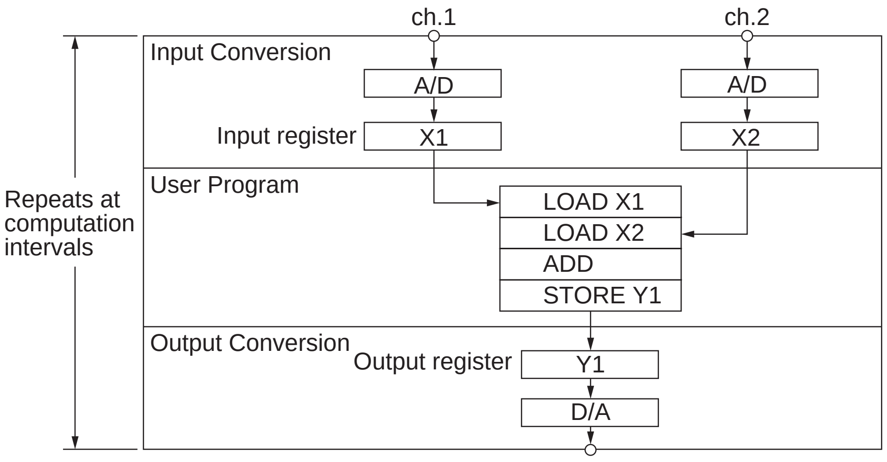


Figure 3.1.1 Computation Basic Flow

3.2 Principle of Computation

This section refers to an example of the addition of two inputs to explain the principle of computation in the user program and the behavior of registers (see figure 3.2.1).

The computation is carried out in a stack of four registers, called S registers (S1, S2, S3, and S4 registers).

Inputting an S register is performed by LOAD instruction (described as LD) and the input data is loaded into the S1 register. Consequently the old data previously stored in the S1, S2, and S3 registers are pushed in a sequential order. Note that the old data in the S4 register is lost.

The computation of data is performed by FUNCTION instruction, which has a variety of computation commands and each command is described by a unique symbol (e.g., ADD [+] or HSL [High Selector]). After performing computation, only the results remain in the S registers and the input data for the computation are lost. The emptied S registers pop up the data in turn. The S4 register holds the same data as stored before the FUNCTION instruction was performed. To store the results in the output registers, use the STORE command (described as ST). Performing the ST command does not affect the contents of S registers.

Assume that the data in S1 to S4 registers, before loading input 1, are A, B, C, and D respectively.

1. Loading Input 1 (LDX1)

   Data in the input register (X1) is loaded into the S1 register. Accordingly each data in the S registers will be pushed and the old data D in the S4 register will be extruded.

2. Loading Input 2 (LDX2)

   In the same way as input 1, data in the input register (X2) is loaded into the S1 register. Each data in the S registers will be pushed and the old data C in the S4 register will be extruded.

3. Addition (ADD)

   The addition of S1 and S2 registers is stored in the S1 register. In consequence, the data in each register will be popped up and the S4 register holds the same old data.

4. Output (STY1)

   The data in the S1 register is stored in the output register (Y1). This does not change the contents of the S registers.

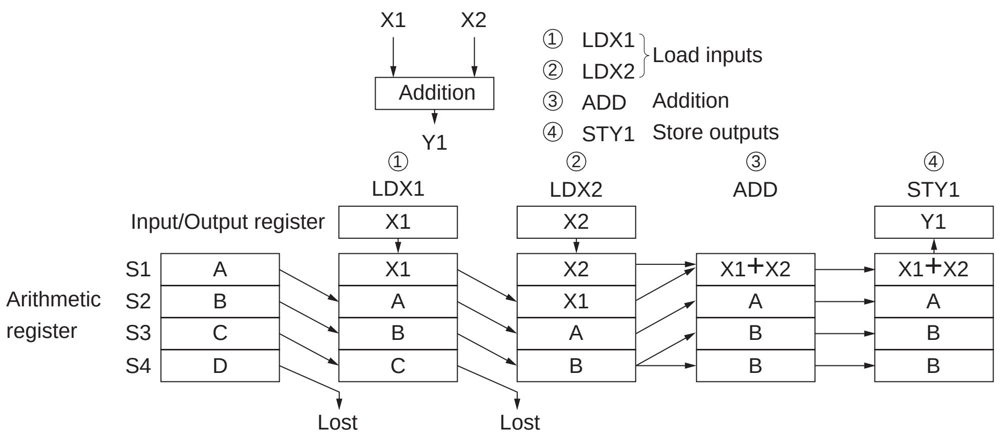


Figure 3.2.1 Two-Input Addition and Behavior of Arithmetic Register

3.3 Program Structure, Capacity, and Interval

3.3.1 Program Capacity

The number of program steps that the computing unit can execute is 59 (or 40 for WXT).

3.3.2 Program Load Factor

When the computing unit is connected to a Handy Terminal or VJ77 Parameter Setting Tool, the load factor is displayed as a percentage, with LOAD A17 (or A12 for WXT). This can be displayed but not modified.

$$\text{Load Factor} = \frac{\text{Program Running Time}}{\text{Computation Interval}} \times 100\%$$

Design your program so that the load factor is less than 100%. The computing unit does not operate correctly for load factors greater than 100%. Change the computation interval or modify the program.

3.3.3 Computation Interval

The computation interval of the computing unit can be set and indicated using the CYCLE TIME D47 (or CYCLE TIME B09 for WXT) setting parameter. It can be set to 50 ms, 100 ms or 200 ms.

3.3.4 Program Structure

The program begins with step G01 in the computing unit program area, carries out each step in succession, and ends when the END command or step G59 is executed.

For the WXT, the steps covered by the program area are B20 to B59. Therefore, read “G01 to G59” noted above as “B20 to B59,” for the program area of WXT. Also read “nn = 01 to 59” as “nn = 20 to 59,” for the step numbers of jump destination by commands (GOnn as unconditional jump and GIFnn as conditional jump).

3.3.5 Input/Output Signals and Registers

The input/output signals are stored in the input/output registers. The user program can store the data in an input register to the arithmetic registers or output the data in the arithmetic registers to the output register, by designating the input/output registers.

Table 3.1 below shows the relationship between the input/output signals and respective registers.

Table 3.1 Input/Output Signals and Registers

Input Register	Signal	Output Register	Signal
X1	Analog Input 1	Y1	Analog Output 1
X2	Analog Input 2	Y2	Analog Output 2
X3	Analog Input 3	DO1	Digital Output 1
DI1	Digital Input 1	DO2	Digital Output 2

Note: For the WXT, the analog output specified by the optional code /D0 is the same signal as Y1 (analog output 1).

3.3.6 Input/Output Signals and Internal Data

The computing unit deals with many input/output signals. This section refers to the example of inputs of 1-5 V and 0-100 mV, and outputs of 1-5 V and 4-20 mA, and their respective input/output ranges are given below in table 3.2, figure 3.3.1, and figure 3.3.2.

Table 3.2 Relationship between Input/Output Signals and Internal Data

	Signal	Percentage Notation	Corresponding Internal Data
Input	1 to 5 V	0% to 100%	0.000 to 1.000
	0 to 100 mV	0% to 100%	0.000 to 1.000
Output	1 to 5 V	0% to 100%	0.000 to 1.000
	4 to 20 mA	0% to 100%	0.000 to 1.000

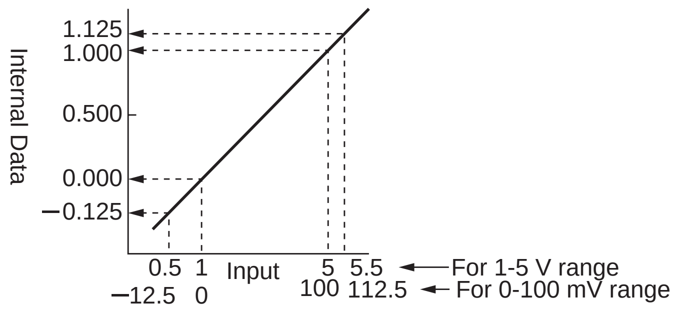


Figure 3.3.1 Input Signals and Internal Data

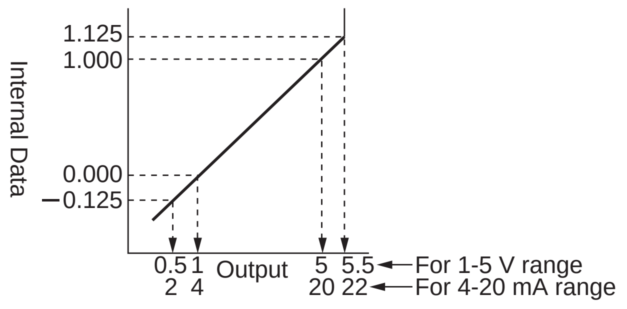


Figure 3.3.2 Internal Data and Output Signals

3.3.7 Internal Computation of Signals

The computing unit expresses its internal digital data in floating decimal format and can perform internal computations under the following conditions:

Range of numbers that can be processed: -9E + 18 to 9E + 18 (for VJX7 and WXT)

-3.4E + 38 to 3.4E + 38 (for VJXS, MXS, MXD and MXT)

Number of significant digits: 4 (for VJX7 and WXT)

7 (for VJXS, MXS, MXD and MXT)

However, it adopts the fixed decimal format for computations about time measurement.

3.3.8 Program End

The computing unit offers a programming area ranging from G01 to G59. Basically the program begins the computation with G01 and ends with G59. If it finds an END command during this process, then it ends with the command.

For the WXT, the program area is B20 to B59. Therefore, read “G01 to G59” program steps noted above as “B20 to B59”.

4. Computational Operations and Applications

The computing unit is capable of performing various computational functions, including input/output commands and arithmetical operations. This chapter describes their operation, setting procedure, and gives program examples.

● Limit to Number of Command Usage

The computational commands are categorized as follows:

• Basic command: Can be used limitlessly in the program.

• Dynamic command: Can be used only once every computational interval.

The dynamic command consists of time-related functions or functions that have an internal buffer for computation. The computing unit performs them only once every computational interval so that it can measure time and detect the occurrence of a status change after the previous cycle.

● Buffer Registers (Tn)

The computing unit is provided with buffer registers (Tn) to store the intermediate data of computations.

The data saved in the Tn register can be used in other steps of the program and holds the same value until another data is stored in the same register. In other words, the Tn register resets its data to zero not every computation period but when the power turns off.

The Tn register consists of T1 to T4 and their values can be displayed using the Handy Terminal or VJ77 Parameter Setting Tool. The on-screen data is shown as a percentage.

● Fixed Constants (Cn)

The fixed constants (Cn) are loaded into the arithmetic registers for use by the LD command during computations. The Cn register consists of 59 registers, C01 to C59, and can be set to a value (percentage) using the Handy Terminal (JHT200) or VJ77 Parameter Setting Tool. Take care as some commands use specific fixed constant area.

'H' can also be used instead of 'C' in C01 to C59 to load and display the fixed constants. They can also be used together in a program. In that case, a total of 59 numbers from 01 to 59 can be allocated to *nn* in C*nn* (or H*nn*) and C01 and H01, for example, refer to the same fixed constant.

For the WXT, 44 fixed constants, C20 to C63, are available. Therefore, read "C01 to C59" noted above as "C20 to C63."

For setting C01 to C59, set H01 to H59 of the parameter H: CONST.
For the WXT, set C20 to C63 of the parameter C: ADJUST.

4.1 Program Commands and Corresponding Registers

The program commands including input/output and arithmetic operations are shown in table 4.1 below.

Table 4.1 Commands and Symbols

Category	Command	Command Symbol	Section Number
Input/output	Load	LDXn, LDYn *1	
		LDCnn, LDTn *2	
		LDDIn *3	
	Store	LDDOn *1	3.2
		STXn, STYn *1	
		STTn, STDOn *1	
Basic operation	+,−, ×, ÷	ADD, SUB, MLT, DIV	4.2
	Square root extraction	SQR	4.3
	Absolute value	ABS	4.7
	Selector	HSL, LSL	4.8
	Limiter	HLM, LLM	4.9
	Line segment function	FX1, FX2, FX3, FX4 *4	4.10
	Comparison	CMP	4.11
	Signal switching	SW	4.12
Dynamic operation	First-order lag	LAGn (n = 1 to 3) *5	4.13
	First-order lead (Differential calculus)	LEDn (n = 1 to 3) *5	4.14
	Dead time	DED *6	4.15
	Velocity	VEL *6	4.16
	Velocity limiter	VLMn (n = 1 and 2) *5	4.17
	Moving average	MAV *6	4.18
	Timer	TIM	4.19
	Status change detection	CCD *7	4.20
	Pulse input counter	PIC *7	4.21
	Square root extraction with variable low-cut point	SQT, SQAn, SQBn (n = 1 to 3) *5	4.4, 4.5 and 4.6
	Integrated pulse output counter	CPO	4.22
	Alarm	HALn, LALn (n = 1 and 2) *5, *8	4.23
Logical operation	Logical multiplication	AND	
	Logical addition	OR	4.24
	Logical negation	NOT	
	Exclusive logical addition	EOR	
Function operation	Trigonometric function	SIN, COS, TAN, ASIN, ACOS, ATN *9	4.25
	Natural logarithm	LN	
	Common logarithm	LOG	4.26
	Exponential function	EXP	
	Exponentiation	PWR	
Others	Unconditional jump	GOnn (nn = 01 to 59) *10	4.27
	Conditional jump	GIFnn (nn = 01 to 59) *10	4.28
	S register exchange	CHG	4.29
	S register rotation	ROT	4.30
	No operation	NOP	4.31
	End of computation	END	4.34

*1: The analog input/output registers (Xn/Yn) and digital output registers (DOn) that do not correspond to actual hardware I/O ports can be used as buffers and flags, respectively.
*2: The “nn” in LDCnn is 01 to 59 for computing units other than the WXT and 20 to 63 for the WXT.
*3: The “n” in LDDIn is only 1 for the MXD.
*4: The line segment functions FXn use a specific area of the fixed constants Cnn as parameters for the I/O table. If you want to use a Cnn with any instruction command other than the FXn, select it from a different area.
*5: The same command, among the LAGn, LEDn, VLMn, SQAn, SQBn, HALn and LALn commands, can be used only once every computational interval. Therefore, more than one command (*n* commands) is available for each of these types of commands.
*6: Only one of the DED, VEL, MAV commands can be used in the same program step; they cannot be used only once every computational interval.
*7: Only available for the MXD.
*8: Not available for the VJX7 and WXT.
*9: Only available for the VJX7 and WXT.
*10: For computing units other than the WXT, the “nn”corresponds to the program step numbers G01 to G59. For the WXT, the “nn“ corresponds to the program step numbers B20 to B59.

The following table summarizes the registers, fixed constants, and program area related to the input/output commands.

Table 4.2 Input/Output Registers and Program Areas

Item	Register, Constant Area and Program Area Symbols	
	MXD, MXS, MXT, VJXS and VJX7	WXT
Arithmetic register (Sn)	S1 S2 S3 S4	S1 S2 S3 S4
Buffer register (Tn)	T1 T2 T3 T4	T1 T2 T3 T4
Analog input register (Xn)	X1 X2 *1, *5 X3 *1, *5	X1 X2 X3
Digital input register (DIn)	DI1 *2 DI2 *6 DI3 *6	-------------------------
Analog output register (Yn)	Y1 Y2 *3	Y1
User flag (DOn)	DO1 *4, *7 DO2 *7 DO3 DO4	DO1 DO2 DO3 DO4
Fixed constant area (Cnn)	C01 *8 • • • C59	C20 • • • C63
Program area (Bnn)	G01 • • • G59	B20 • • • B59

*1: Can be used for the MXT only.
*2: Can be used if the contact input is custom-ordered for the MXD or MXT.
*3: Can be used as an analog output if the optional analog output is selected for the MXS, VJXS or VJX7. This register can also be used as a buffer for any other computing unit.
*4: For the MXD, DO1 cannot be used as a buffer but as a digital output.
*5: For computing units other than the MXT, X2 and X3 can be used as buffers.
*6: Only if the contact input is custom-ordered for the MXT. This register can not be used for any other computing unit.
*7: If the optional contact input is selected for the VJX7, DO1 and DO2 can be used as digital outputs but not as user flags.
*8: If used as symbols for fixed constants, “C01 to C59” can also be expressed as “H01 to H59.”

4.2 Arithmetical Operation

[Mnemonic Instruction Code]

+	:	ADD	Addition	S2 + S1 → S1
–	:	SUB	Subtraction	S2 – S1 → S1
×	:	MLT	Multiplication	S2 × S1 → S1
÷	:	DIV	Division	S2 ÷ S1 → S1

[Operation]

The arithmetical operations are performed to data in the S1 and S2 registers and the result is stored in the S1 register.

[Function Block]

The function block can be expressed as an original equation as shown in figure 4.2.1.

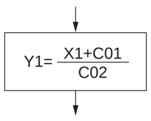


Figure 4.2.1 Function Block of Arithmetical Operation

[Program Example]

Figure 4.2.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load fixed constant C01
G03	ADD	X1+C01			Add
G04	LDC02	C02	X1+C01		Load fixed constant C02
G05	DIV	(X1+C01)/C02			Divide
G06	STY1	(X1+C01)/C02			Store the result in Y1
G07	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.3 Square Root Extraction without Variable Low-cut Point

[Mnemonic Instruction Code]

$\sqrt{\quad}$: SQR Extraction of square root

[Operation]

This command is performed to data in the S1 register and the result is stored in the S1 register.

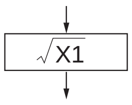


Figure 4.3.1 Function of Block Square Root Extraction

[Program Example]

Figure 4.3.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	SQR	$\sqrt{X1}$		Extract square root
G03	STY1	$\sqrt{X1}$		Store the result in Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.4 Square Root Extraction with Variable Low-cut Point (1)

[Mnemonic Instruction Code]

$\sqrt{}$: SQT Extraction of square root with variable low-cut point (for inputs not higher than the low-cut point, the output is the same as input)

[Operation]

This command is performed to data in the S2 register with the S1 register as the low-cut point and the result is stored in the S1 register. The input/output characteristics are shown in figure 4.4.1.

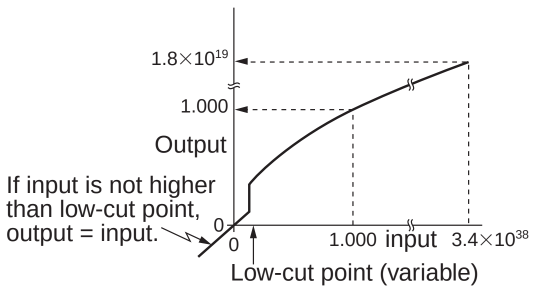


Figure 4.4.1 Input/output Characteristics of Square Root Extraction with Variable Low-cut Point

If the input is not higher than the low-cut point, the output is the same as input. The low-cut point can be set to a value within 0 to 3.4E + 38.

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

This command can form one function block in combination with arithmetical operations. The function block can be expressed as shown in figure 4.4.2, which extracts the input X1 not lower than the set low-cut point and stores the result.

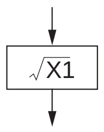


Figure 4.4.2 Function Block of Square Root Extraction with Variable Low-cut Point

[Program Example]

Figure 4.4.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load low-cut point
G03	SQT	$\sqrt{}$X1			Extract square root with low-cut
G04	STY1	$\sqrt{}$X1			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.5 Square Root Extraction with Variable Low-cut Point (2)

[Mnemonic Instruction Code]

$\sqrt{}$:	SQAn (n = 1 to 3)	Extraction of square root with variable low-cut point (for inputs not higher than the low-cut point, the output is the same as input)

[Operation]

This command is performed to data in the S2 register with the S1 register as the low-cut point and the result is stored in the S1 register. The input/output characteristics are shown in figure 4.5.1.

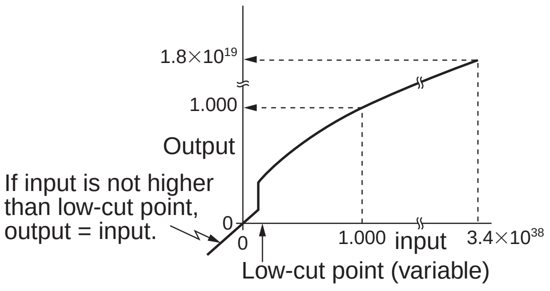


Figure 4.5.1 Input/output Characteristics of Square Root Extraction with Variable Low-cut Point

If the input is not higher than the low-cut point, the output is the same as input. The low-cut point can be set to a value within 0 to 3.4E + 38.

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

This command can form one function block in combination with arithmetical operations. The function block can be expressed as shown in figure 4.5.2, which extracts the input X1 not lower than the set low-cut point and stores the result.

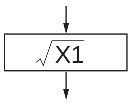


Figure 4.5.2 Function Block of Square Root Extraction with Variable Low-cut Point

[Program Example]

Figure 4.5.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load low-cut point
G03	SQA1	√X1			Extract square root with low-cut
G04	STY1	√X1			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.6 Square Root Extraction with Variable Low-cut Point (3)

[Mnemonic Instruction Code]

$\sqrt{\quad}$: SQBn (n = 1 to 3) Extraction of square root with variable low-cut point (for inputs not higher than the low-cut point, the output is 0%)

[Operation]

This command is performed to data in the S2 register with the S1 register as the low-cut point and the result is stored in the S1 register. The input/output characteristics are shown in figure 4.6.1.

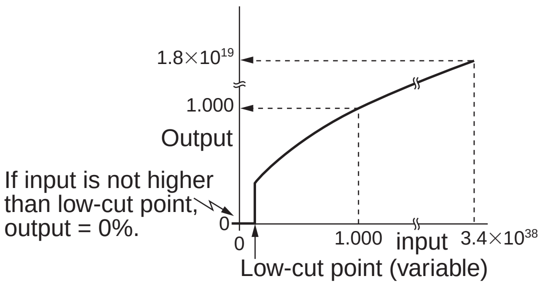


Figure 4.6.1 Input/output Characteristics of Square Root Extraction with Variable Low-cut Point

If the input is not higher than the low-cut point, the output is 0%. The low-cut point can be set to a value within 0 to 3.4E + 38.

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

This command can form one function block in combination with arithmetical operations. The function block can be expressed as shown in figure 4.6.2, which extracts the input X1 not lower than the set low-cut point and stores the result.

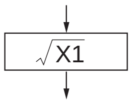


Figure 4.6.2 Function Block of Square Root Extraction with Variable Low-cut Point

[Program Example]

Figure 4.6.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load low-cut point
G03	SQB1	√X1			Extract square root with low-cut
G04	STY1	√X1			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.7 Absolute Value

[Mnemonic Instruction Code]

ABS	Absolute value

[Operation]

This command is performed to data in the S1 register and the result is stored in the S1 register.

[Function Block]

The function block can be expressed as shown in figure 4.7.1, which calculates an absolute value from the difference of two inputs and stores the result (for the MXT-A and WXT-A only).

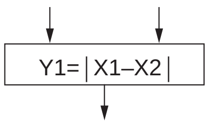


Figure 4.7.1 Function Block of Absolute Value

[Program Example]

Figure 4.7.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDX2	X2	X1		Load input 2
G03	SUB	X1 – X2			Subtract X2 from X1
G04	ABS	|X1–X2|			Find the absolute value
G05	STY1	|X1–X2|			Store the result in Y1
G06	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.8 Selector

[Mnemonic Instruction Code]

HSL	High selector
LSL	Low selector

[Operation]

High selector: This command compares the data in S1 and S2 registers, and then stores the higher one in the S1 register.

Low selector: This command compares the data in S1 and S2 registers, and then stores the lower one in the S1 register.

Note the data that has not been selected will be lost.

[Function Block]

The function block of the high selector can be expressed as shown in figure 4.8.1, which compares three inputs.

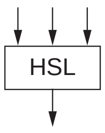


Figure 4.8.1 Function Block of High Selector for Three Inputs

[Program Example]

Figure 4.8.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDX2	X2	X1		Load input 2
G03	HSL	X1			Select higher data (X1 > X2 in this case)
G04	LDX3	X3	X1		Load input 3
G05	HSL	X1			Select higher data (X1 > X3 in this case)
G06	STY1	X1			Store the result in Y1
G07	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.9 Limiter

[Mnemonic Instruction Code]

HLM	High limiter
LLM	Low limiter

[Operation]

This command loads the preset ascending limit or descending limit in S1 and input in S2 register, and stores the result of limiting in the S1 register.

[Function Block]

The function block of the high selector can be expressed as shown in figure 4.9.1, which performs high limiting and low limiting in succession. The fixed constants enclosed with parentheses are high/low limits.

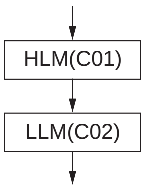


Figure 4.9.1 Function Block of High/Low Limiter

[Program Example]

Figure 4.9.1 can be programmed as shown in the table below.

Notice how the values of arithmetic registers change when X1 < C02 < C01.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load ascending limit
G03	HLM	X1			Upper limit (X1 < C20 in this case)
G04	LDC02	C02	X1		Load descending limit
G05	LLM	C02			Lower limit (X1 < C21 in this case)
G06	STY1	C02			Store the result in Y1
G07	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.10 Line Segment Function

[Mnemonic Instruction Code]

FX1	Ten-segment linearizer
FX2	Arbitrary line segment linearizer (10-segment)
FX3	Arbitrary line segment linearizer (20-segment)
FX4	Arbitrary line segment linearizer (any number of segments)

[Operation]

Table 4.10 gives the fixed constant area for the input/output data tables and figures 4.10.1, 4.10.3, 4.10.5 and 4.10.7 show the relationship between inputs and outputs. You can create two 10-segment functions by using the FX1 and FX2 linearizers.

Note: It is not possible to create two functions using the FX3 and FX4 linearizers.

Table 4.10 Fixed Constants as Input/Output Data of Line Segment Linearizer

MXD,MXS,MXT,VJXS,VJX7					
Fixed constant	Setting address	FX1	FX2	FX3	FX4[*1]
C01	H01	Y0		X0	X0
C02	H02	Y1		X1	X1
C03	H03	Y2		X2	X2
C04	H04	Y3		X3	X3
C05	H05	Y4		X4	X4
C06	H06	Y5		X5	X5
C07	H07	Y6		X6	X6
C08	H08	Y7		X7	X7
C09	H09	Y8		X8	X8
C10	H10	Y9		X9	X9
C11	H11	Y10		X10	X10
C12	H12		X0	X11	X11
C13	H13		X1	X12	X12
C14	H14		X2	X13	X13
C15	H15		X3	X14	X14
C16	H16		X4	X15	X15
C17	H17		X5	X16	X16
C18	H18		X6	X17	X17
C19	H19		X7	X18	X18
C20	H20		X8	X19	X19
C21	H21		X9	X20	X20
C22	H22		X10	Y0	Y0
C23	H23		Y0	Y1	Y1
C24	H24		Y1	Y2	Y2
C25	H25		Y2	Y3	Y3
C26	H26		Y3	Y4	Y4
C27	H27		Y4	Y5	Y5
C28	H28		Y5	Y6	Y6
C29	H29		Y6	Y7	Y7
C30	H30		Y7	Y8	Y8
C31	H31		Y8	Y9	Y9
C32	H32		Y9	Y10	Y10
C33	H33		Y10	Y11	Y11
C34	H34			Y12	Y12
C35	H35			Y13	Y13
C36	H36			Y14	Y14
C37	H37			Y15	Y15
C38	H38			Y16	Y16
C39	H39			Y17	Y17
C40	H40			Y18	Y18
C41	H41			Y19	Y19
C42	H42			Y20	Y20
C43	H43				Number of segments

WXT			
Fixed constant (Setting address)	FX1	FX2	FX3
C20	Y0		X0
C21	Y1		X1
C22	Y2		X2
C23	Y3		X3
C24	Y4		X4
C25	Y5		X5
C26	Y6		X6
C27	Y7		X7
C28	Y8		X8
C29	Y9		X9
C30	Y10		X10
C31		X0	X11
C32		X1	X12
C33		X2	X13
C34		X3	X14
C35		X4	X15
C36		X5	X16
C37		X6	X17
C38		X7	X18
C39		X8	X19
C40		X9	X20
C41		X10	Y0
C42		Y0	Y1
C43		Y1	Y2
C44		Y2	Y3
C45		Y3	Y4
C46		Y4	Y5
C47		Y5	Y6
C48		Y6	Y7
C49		Y7	Y8
C50		Y8	Y9
C51		Y9	Y10
C52		Y10	Y11
C53			Y12
C54			Y13
C55			Y14
C56			Y15
C57			Y16
C58			Y17
C59			Y18
C60			Y19
C61			Y20
C62			

Xn: Input register Yn: Output register

*1: For the VJX7, this line-segment function cannot be used.

FX1

The input data (X0 to X10) for the FX1 linearizer are 0 to 100% in 10% increments.

Output setting conditions: –6% ≤ C01 to C11 ≤ 106%

· When the input is below 0%, the computing unit does not limit output, which is calculated using the following statement:

C01 + (C02 – C01) /10 × (Input [%])

· When the input is above 100%, the computing unit calculates the output using the following statement:

C11 + (C11 – C10) /10 × (Input [%] – 100 [%])

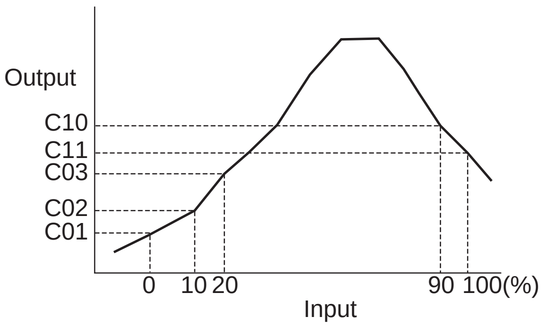


Figure 4.10.1 Relationship between Input and Output of FX1 Ten-Segment Linearizer

[Function Block]

The function block of the FX1 linearizer can be expressed as shown in figure 4.10.2.

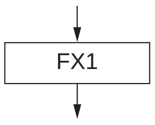


Figure 4.10.2 Function Block of FX1 Linearizer

[Program Example]

Figure 4.10.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	FX1	Output Breakpoints		Ten-segment linearizer
G03	STY1	Output Breakpoints		Store the result in Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

FX2

Setting conditions:

For input : –6% ≤ C12 to C22 ≤ 106%

C12 < C13 < C14 < C15 < C16< C17 < C18 < C19 < C20 < C21 < C22

For output : –6% ≤ C23 to C33 ≤ 106%

· When the input ≤ C12, the computing unit outputs the C23, and when the input ≥ C22, it outputs the C33.

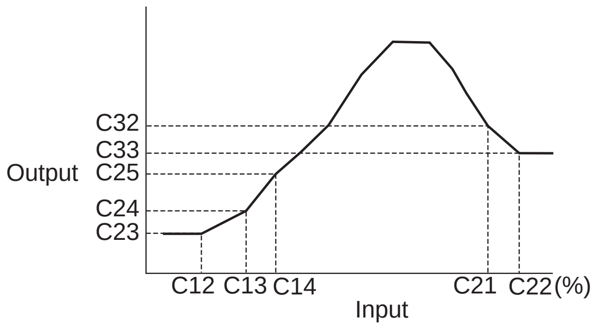


Figure 4.10.3 Relationship between Input and Output of FX2 Arbitrary Ten-Segment Linearizer

[Function Block]

The function block of the FX2 linearizer can be expressed as shown in figure 4.10.4.

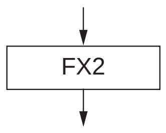


Figure 4.10.4 Function Block of FX2 Linearizer

[Program Example]

Figure 4.10.4 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	FX2	Output Breakpoints		Arbitrary ten-segment linearizer
G03	STY1	Output Breakpoints		Store the result in Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

FX3

Setting conditions:

For input : –6% ≤ C01 to C21 ≤ 106%

C01 < C02 < C03 < C04 < C05 < C06 < C07 < C08 < C09 < C10 < C11

< C12 < C13 < C14 < C15 < C16 < C17 < C18 < C19 < C20 < C21

For output : –6% ≤ C22 to C42 ≤ 106%

· When the input ≤ C01, the computing unit outputs the C22, and when the input ≥ C21, it outputs the C42.

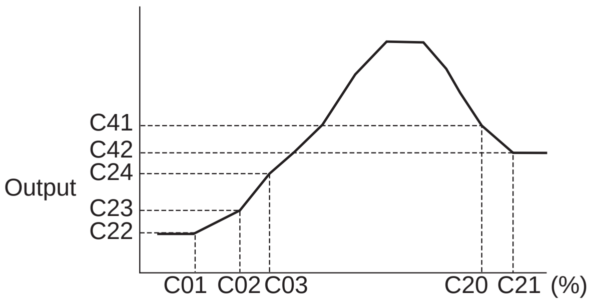


Figure 4.10.5 Relationship between Input and Output of FX3 Arbitrary Twenty-Segment Linearizer

[Function Block]

The function block of the FX3 linearizer can be expressed as shown in figure 4.10.6.

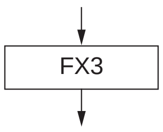


Figure 4.10.6 Function Block of FX3 Linearizer

[Program Example]

Figure 4.10.6 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	FX3	Output Breakpoints		Arbitrary twenty-segment linearizer
G03	STY1	Output Breakpoints		Store the result in Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

FX4

Setting conditions:

For number of line segments: Set the number of line segments in C43.
(Set it at % value. The number of line segments 1 to 20 corresponds to 100 to 2000%.)

For input : –6% ≤ C01 to C21 ≤ 106%

C01 < C02 < C03 < C04 < C05 < C06 < C07 < C08 < C09 < C10 < C11

< C12 < C13 < C14 < C15 < C16 < C17 < C18 < C19 < C20 < C21

For output : –6% ≤ C22 to C42 ≤ 106%

Since the FX4 linearizer operates based on the desired number of line segments set in C43, there is no need to set from the C01 to C42 fixed constants as with FX3. For the input data table, set as many fixed constants as necessary, beginning with C01 as in C01, C02, C03, . . . Likewise, set as many fixed constants as necessary for the output data table, beginning with C22 as in C22, C23, C24, . . . Setting more data items than necessary results in the extra data items being nullified.

For example, if you specify the number of line segments as 5, set the C01 to C06 and C22 to C27 fixed constants in the data table. Data values set in C07 to C21 and C28 to C42 are ignored.

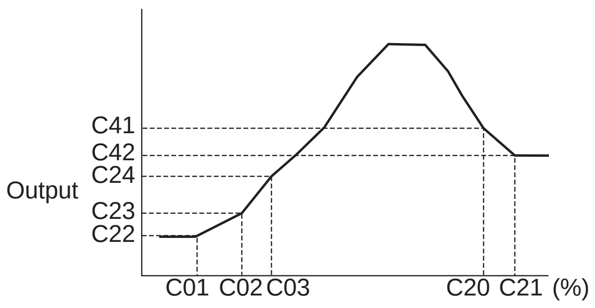


Figure 4.10.7 Relationship between Input and Output of FX4 Arbitrary Twenty (Max.)-Segment Linearizer

[Function Block]

The function block of the FX4 linearizer can be expressed as shown in figure 4.10.8.

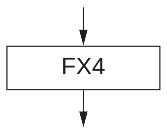


Figure 4.10.8 Function Block of FX4 Linearizer

[Program Example]

Figure 4.10.8 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	FX4	Output Breakpoints		Arbitrary twenty-segment linearizer
G03	STY1	Output Breakpoints		Store the result in Y1
G04	Next computation			

Note: For the VJX7 and WXT, this line-segment function cannot be used.

4.11 Comparison

[Mnemonic Instruction Code]

CMP Comparison

[Operation]

This command compares data in S1 and S2 registers, and stores the value in the S1 register in the following manner:

· If S1 ≤ S2, the value is 1.000.

· If S1 > S2, the value is 0.000.

The old value in the S1 register will be lost but the data in the S2 register remains.

[Function Block]

The function block can be expressed as shown in figure 4.11.1, which performs a comparison between two inputs and then outputs 1.000 (when X1 ≥ X2) or 0.000 (when X1 < X2).

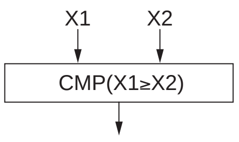


Figure 4.11.1 Function Block of Comparison

[Program Example]

Figure 4.11.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			
G02	LDX2	X2	X1		
G03	CMP	0 or 1	X1		Compare X1 and X2
G04	STY1	0 or 1	X1		Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.12 Signal Switching

[Mnemonic Instruction Code]

SW Signal Switching

[Operation]

This command loads data in the S2 and S3 registers, and the switching signal in the S1 register:

· If S1 < 0.5, the data in S3 register is stored in the S1 register.

· If S1 ≥ 0.5, the data in S2 register is stored in the S1 register.

[Function Block]

The function block can be expressed as shown in figure 4.12.1, which switches two inputs using the C01 as a switching signal.

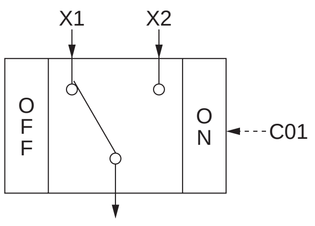


Figure 4.12.1 Function Block of Signal Switching

[Program Example]

Figure 4.12.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			
G02	LDX2	X2	X1		
G03	LDC01	C01	X2	X1	
G04	SW	X1 or X2			X1 if C01 < 0.5 or X2 if C01 ≥ 0.5
G05	STY1	X1 or X2			
G06	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.13 First-order Lag Computation

[Mnemonic Instruction Code]

LAGn First-order Lag Computation (n = 1 to 3)

$$Y = \frac{1}{1+T_S} X$$

[Operation]

This command loads the input and time constant in the S2 and S1 registers respectively, performs the computation, and stores the result in the S1 register.

· Setting range of time constant (minimum unit is 0.1 second)

The time constant for a first-order lag computation can be set up to 799.9 seconds where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-100 seconds. (The internal value setting for the maximum time constant is 7.999.)

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block can be expressed as shown in figure 4.13.1.

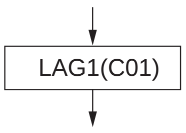


Figure 4.13.1 Function Block of First-order Lag Computation

[Program Example]

Figure 4.13.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load time constant
G03	LAG1	$(1-e^{-t/C01})\times X1$			First-order lag computation
G04	STY1	$(1-e^{-t/C01})\times X1$			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.14 First-order Lead (Differential Calculus)

[Mnemonic Instruction Code]

LEDn First-order Lead (Differential Calculus) (n = 1 to 3)

$$Y=\frac{T_{DS}}{1+T_{DS}}X$$

[Operation]

This command loads the input and time constant in the S2 and S1 registers respectively, performs the computation, and stores the result in the S1 register.

- Setting range of time constant (minimum unit is 0.1 second)

  As with a first-order lag computation, the time constant for a differential computation can be set up to 799.9 seconds where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-100 seconds. (The internal value setting for the maximum time constant is 7.999.)

- Derivative gain

  The derivative gain is 1.0 and you can multiply the result of first-order lead (differential) computation by a constant as necessary.

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block of the first-order lead (differential) computation with derivative gain can be expressed as shown in figure 4.14.1.

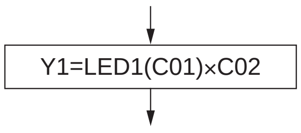


Figure 4.14.1 Function Block of Differential Calculus

[Program Example]

Figure 4.14.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load time constant
G03	LED1	$e^{-t/C01}\times\Delta X1$			First-order Lead (Differential calculus)
G04	LDC02	C02	$e^{-t/C01}\times\Delta X1$		Load derivative gain
G05	MLT	$C02\times e^{-t/C01}\times\Delta X1$			Multiply by gain
G06	STY1	$C02\times e^{-t/C01}\times\Delta X1$			Store the result in Y1
G07	Next computation				

$\Delta X1$: A variation of input 1

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.15 Dead Time Computation

[Mnemonic Instruction Code]

DED Dead Time Computation

$Y = e^{-LS} \times X$ where L is dead time

This command cannot be used together with the VEL and MAV commands in the same program step.

[Operation]

This command loads the input and dead time in the S2 and S1 registers respectively, performs the computation, and stores the result in the S1 register.

· Setting range of dead time

The dead time can be set up to 2,047,000 seconds where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-1000 seconds. (The internal value setting for the maximum dead time is 2047.0.)

· Principle

Figure 4.15.1 explains the principle of operation. The dead time computation stores the input in its dedicated buffer registers every sampling. The buffer register consists of 40 registers and the input moves to the right in the registers.

When the computing units turn on, the last input (A) is stored in all of the 40 buffer registers as an initial value. After dead time setting/40 seconds, the computing unit loads the next input (B). At the same time, all the data in buffer registers move to the right and data in the register 40 is output. Namely, reading an input, data shift, and outputting the 40th data are performed every dead time setting/40 seconds. Furthermore, the computing unit interpolates the outputs to achieve their gradual changes. However, the minimum sampling time is the same as the computation interval, which means if you set the dead time too short not all of the buffer registers may be used for computation. For instance, if the dead time is one second and the computation interval is 100 ms, only 10 buffer registers will be used.

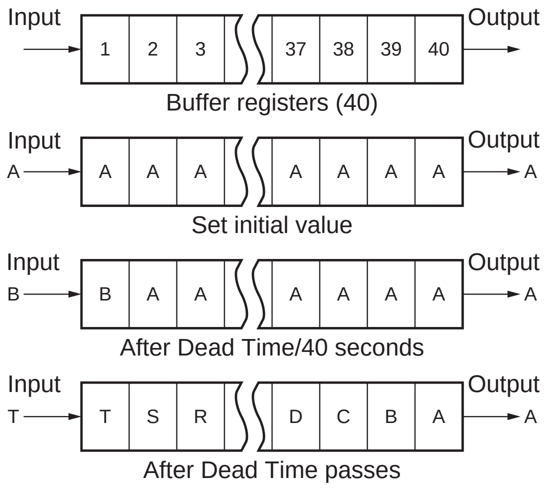


Figure 4.15.1 Operational Principle of Dead Time Computation

Note: As this command is a dynamic operation, it can be used only once every computational interval.

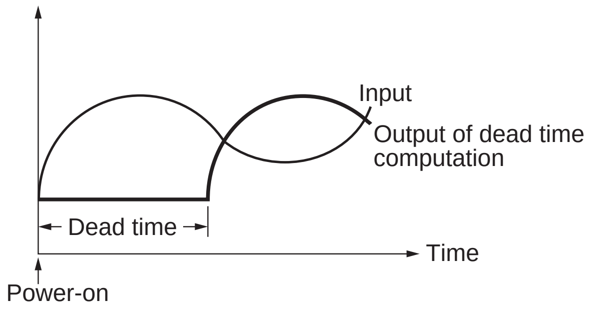


Figure 4.15.2 Input/output Characteristics of Dead Time Computation

[Function Block]

The function block of the differential computation with derivative gain can be expressed as shown in figure 4.15.3.

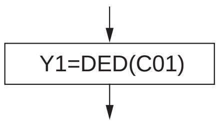


Figure 4.15.3 Function Block of Dead Time Computation

[Program Example]

Figure 4.15.3 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load dead time
G03	DED	$X1_{t-C01}$			Calculates value of X1 before C01 seconds
G04	STY1	$X1_{t-C01}$			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.16 Velocity Computation

[Mnemonic Instruction Code]

VEL Velocity Computation

This command cannot be used together with the DED and MAV commands in the same program step.

[Operation]

This command uses the dead time computation and subtracts the input value sampled at a point of time a specified period before the current time from the sampled current input value. This period is set as the 'dead time.'

The velocity computation is performed according to the following equation:

$Y1_t = X1_t - X1_{t-C01}$

where $Y1_t$: Output of velocity computation;

$X1_t$: Current input value

$X1_{t-C01}$: Input value C01 seconds before

· Setting range of dead time

As with the dead time computation, the dead time can be set up to 2,047,000 seconds where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-1000 seconds. (The internal value setting for the maximum dead time is 2047.)

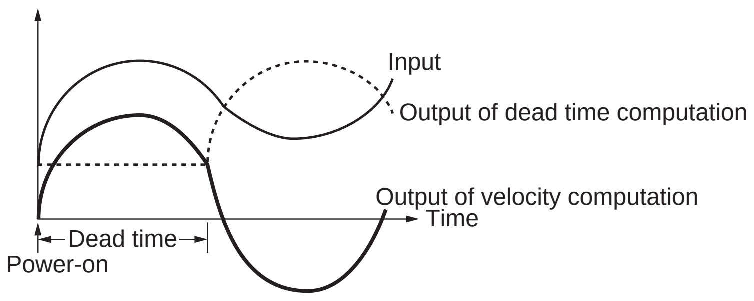


Figure 4.16.1 Input/output Characteristics of Velocity Computation

As in figure 4.16.1, the computation results can be a negative value. So you need to add certain bias or perform the absolute value computation to the output of velocity computation.

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block of the differential computation can be expressed as shown in figure 4.16.2.

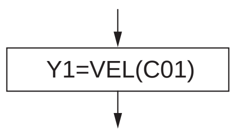


Figure 4.16.2 Function Block of Velocity Computation

[Program Example]

Figure 4.16.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input
G02	LDC01	C01	X1		Load velocity computation time
G03	VEL	$X1_t–X1_{t–C01}$			Calculates velocity
G04	STY1	$X1_t–X1_{t–C01}$			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.17 Velocity Limiter

[Mnemonic Instruction Code]

VLMn Velocity Limiter (n = 1 or 2)

[Operation]

This command loads the input, ascending velocity limit, and descending velocity limit in the S3, S2, and S1 registers respectively, performs the computation, and stores the result in the S1 register.

· Setting range of velocity limit:

0.1% to 699.9% per minute where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-100% per minute (the internal value setting for the maximum and minimum velocity limits are 6.999 and 0.001, respectively) .

· Principle

Figure 4.17.1 explains the principle of operation. Setting the limit at 700.0%/minute (the internal value 7.000) or above does not limit the input as is (i.e., works as an open limit function).

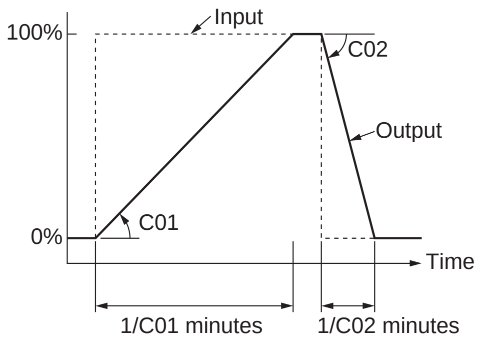


Figure 4.17.1 Input/output Characteristics of Velocity Limiter

[Function Block]

The function block can be expressed as shown in figure 4.17.2.

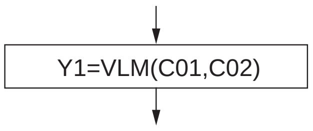


Figure 4.17.2 Function Block of Velocity Limiter

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Program Example]

Figure 4.17.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load ascending velocity limit
G03	LDC02	C02	C01	X1	Load descending velocity limit
G04	VLM1	X1 after limit			Velocity limit computation
G05	STY1	X1 after limit			Store the result in Y1
G06	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.18 Moving Average Computation

[Mnemonic Instruction Code]

MAV Moving Average Computation

This command cannot be used together with the DED and VEL commands in the same program step.

[Operation]

This command is application of the dead time computation, totals the 40 inputs that have been sampled since going back to the preset computation time from the last sampling, and calculates the average.

· Setting range of computation time (minimum unit is 1 second)

As with the dead time computation, the computation time can be set up to 2,047,000 seconds where 0.000-1.000 (0.0-100.0%) of internal data corresponds to 0-1000 seconds. (The internal value setting for the maximum computation time is 2047.)

However, the minimum sampling time is the same as the computation interval, which means the minimum computation time that can use the 40 sampling buffer registers to their full effectiveness is four seconds if the computation interval is 100 ms. Therefore, if you set the computation time too short, not all of buffer registers may be used for computation. For instance, if the computation time is one second and the computation interval is 100 ms, only 10 buffer registers will be used.

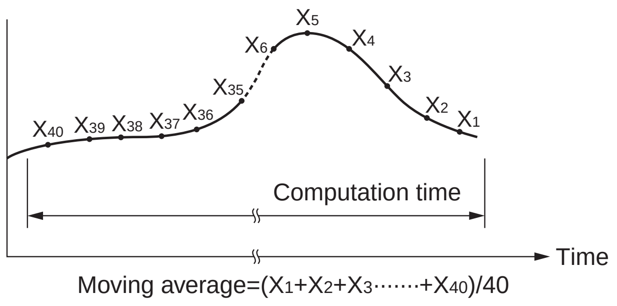


Figure 4.18.1 Input/output Characteristics of Moving Average Computation

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block of the moving average computation can be expressed as shown in figure 4.18.2.

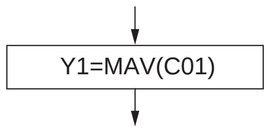


Figure 4.18.2 Function Block of Moving Average Computation

[Program Example]

Figure 4.18.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input
G02	LDC01	C01	X1		Load computation time
G03	MAV	Moving Average			Calculates moving average
G04	STY1	Moving Average			Store the result in Y1
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.19 Timer

[Mnemonic Instruction Code]

TIM Timer

[Operation]

This command loads value for timer start/stop before computation and the elapsed time after computation in the S1 register. The following table summarizes the behavior of the S1 register:

Before Computation	After Computation	Description
S1 < 0.5	0.000	Timer reset
S1 ≥ 0.5	Elapsed time	At start of timer or while running

Figure 4.19.1 shows the timer operation. When the timer start signal is off (less than 0.5), the result of TIM computation is 0.000. When the signal is turned on (0.5 or more), the timer starts counting time and the TIM result increases as time elapses. If the count reaches 4,095,999 seconds, it is reset to zero (the internal value 1.000 corresponds to 1000 seconds so that the timer can count up to 4,095,999 seconds).

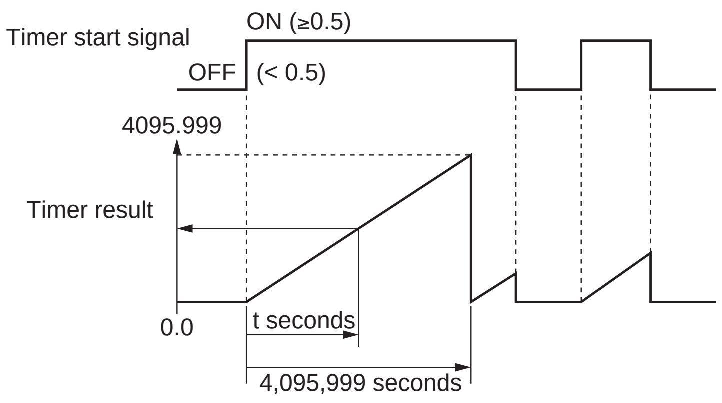


Figure 4.19.1 Timer Operation

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block can be expressed as shown in figure 4.19.2, which uses the X1 register as the timer start signal.

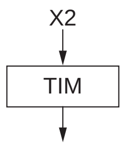


Figure 4.19.2 Function Block of Timer

[Program Example]

Figure 4.19.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX2	X2		Load timer start signal
G02	TIM	Time		Elapsed time
G03	STY1	Time		
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.20 Status Change Detection (for MXD-A Only)

[Mnemonic Instruction Code]

CCD Status Change Detection

[Operation]

This command returns 1 as output if the input of S1 register changes 0 to 1 at the last computation interval. If no change takes place (e.g., the input holds 1 or 0) or the input changes 1 to 0, it returns 0. Figure 4.20.1 explains the input/output operation when checking the input status change. The status output is 0 at cold start.

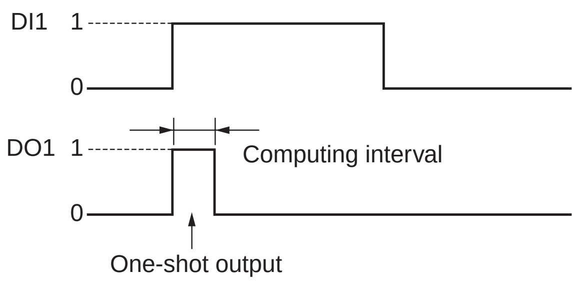


Figure 4.20.1 Input/output Operation of Status Change Detection

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block can be expressed as shown in figure 4.20.2.

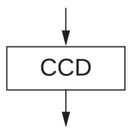


Figure 4.20.2 Function Block of Status Change Detection

[Program Example]

Figure 4.20.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDDI1	DI1			Load contact input
G02	CCD	0 or 1			Check Status Change
G03	STDO1	0 or 1			Output the result to DO1
G04	Next computation				

4.21 Pulse Input Counter (for MXD-A Only)

[Mnemonic Instruction Code]

PIC Pulse Input Counter

[Operation]

This command increases the counter as a pulse when the data in S2 register changes 0 to 1 while the content of S1 register is ON. For both the ON and OFF time, the computing unit only counts a pulse twice the control interval or longer. The counter can be increased up to 30,000 and if it exceeds 30,000 it limits the count and holds 30,000. The PIC command outputs analog signal of 0 to 30,000.

The internal value 1.000 corresponds to 1000 pulses. Furthermore, in the event of cold start, the computing unit resets the counter to zero and restarts counting from the next interval.

• Setting input conditions

For S2 register: Input

For S1 register: Start/Stop signal; starts or continues counting while the S1 register is ON (≥ 0.5), resets the counter when it is OFF (< 0.5).

• Output

S1 register: Counter output (internal value 0 to 1.00 corresponding to 0 to 1000 pulses)

Note: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block can be expressed as shown in figure 4.21.1.

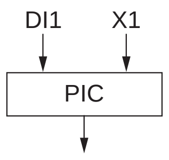


Figure 4.21.1 Function Block of Pulse Input Counter

[Program Example]

Figure 4.21.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDDI1	DI1			Load pulse input
G02	LDX1	X1	DI1		Check start/reset status
G03	PIC	Computation result			Count pulse
G04	STY1	Computation result			Store the result in Y1
G05	Next computation				

4.22 Integrated Pulse Output Counter (for MXD and VJX7 Free Program Only)

[Mnemonic Instruction Code]

CPO Integrated Pulse Output

Note that this command uses the DO1 register, therefor the contact output is unavailable if the command is executed. This is because the DO1 cannot be used as a contact output.

For the VJX7, DO2 cannot be used.

[Operation]

This command loads the input and integrating ratio in the S2 and S1 registers, respectively. The S2 register has been made capable of storing data within the range of 0.000 to 10.000, because the data may exceed 1.0 as a result of temperature and pressure compensation. The S1 register can store the integrating ratio of 0.01 to 18.000. Note that the maximum output pulse is 5 pulses/second (for both computation intervals of 0.1 and 0.2 second).

The relationship between the integrating ratio (S1) and input (S2), and the pulse output is as follows:

For the input of 100% (S2):

The integrating ratio of 0 to 1.0 corresponds to 0 to 1000 pulses/hour, namely:

Integrated pulse output = integrating ratio (S1) $\times$ input (S2) $\times$ 1000 [pulse/hour]

For example, if the integrating ratio (S1) = 0.500 and the input value (S2) = 0.750, then the integrated pulse = 0.5 $\times$ 0.75 $\times$ 1000 = 375 pulses/hour.

In addition, pulse outputs are ON-pulse of 100 ±1 ms long.

Note 1: A potential problem exists if the integrating ratio is too small, because the integrated value of the ratio and input also becomes too small to calculate or the results include integration errors. In other words, the integrating ratio of 0.01 cannot accommodate inputs no more than 2.4%.

Note 2: As this command is a dynamic operation, it can be used only once every computational interval.

[Function Block]

The function block can be expressed as shown in figure 4.22.1, where C01 is the integrating ratio.

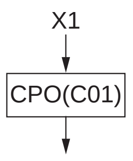


Figure 4.22.1 Function Block of Integrated Pulse Output Counter

[Program Example]

Figure 4.22.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input
G02	LDC01	C01	X1		Load integrating ratio
G03	CPO	X1			Output the result to DO1
G04	Next computation				

4.23 Alarm

[Mnemonic Instruction Code]

HALn	High alarm (n = 1 or 2)
LALn	Low alarm (n = 1 or 2)

[Operation]

Stores the input value in the S3 register, the alarm setpoint in the S2 register, and the hysteresis width in the S1 register. A positive value greater than 0 can be stored in the S1 register.

After computation, the computing unit stores "1" in the S1 register if the input value is equal to or greater than the setpoint (abnormal), or stores "0" if the input value is smaller than the setpoint.

The contact outputs of the VJX7 and MXD are not interlocked with the alarm discussed in this section.

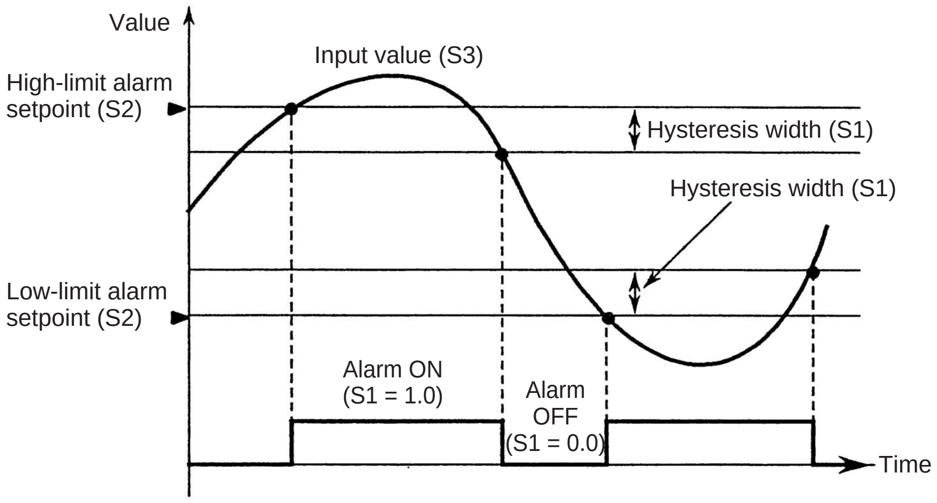


Figure 4.23.1 Input/Output Characteristics of Alarm Operation

[Function Block]

The function block can be expressed as shown in figure 4.23.2, where C01 is the alarm setpoint and C02 is the hysteresis width.

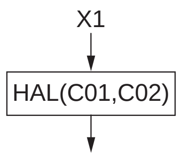


Figure 4.23.2 Function Block of Alarm

[Program Example]

Figure 4.23.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDC01	C01	X1		Load alarm setpoint
G03	LDC02	C02	C01	X1	Load hysteresis width
G04	HAL1	0/1	X1		High alarm
G05	STDO1	0/1	X1		Store the result in DO1
G06	Next computation				

Note: For the VJX7 and WXT, this alarm function cannot be used.

4.24 Logical Operation

[Mnemonic Instruction Code]

$\cap$: AND	S2 $\cap$ S1→S1
$\cup$: OR	S2 $\cup$ S1→S1
$\overline{\ }$: NOT	$\overline{\text{S1}}$ →S1
$\uplus$: EOR	S2 $\uplus$ S1→S1

[Operation]

This command deals with the S1 and S2 registers. The result is stored in the S1 register as 0 or 1. The EOR operation returns 0 when the S1 and S2 contain the same value. The data in registers is considered to be 1 if it is 0.5 or more, and 0 if it is not.

[AND]

S1	S2	S1
0	0	0
0	1	0
1	0	0
1	1	1

[OR]

S1	S2	S1
0	0	0
0	1	1
1	0	1
1	1	1

[NOT]

S1	S1
0	1
1	0

[EOR]

S1	S2	S1
0	0	0
0	1	1
1	0	1
1	1	0

Register Values before and after Operations

4.25 Trigonometric Function (for VJX7 and WXT Only)

[Mnemonic Instruction Code]

SIN	Sine
COS	Cosine
TAN	Tangent
ASIN	Arcsine (SIN^{-1})
ACOS	Arccosine (COS^{-1})
ATAN	Arctangent (TAN^{-1})

[Operation]

The trigonometric function commands deal with the S1 register. The result is stored back in the S1 register . As with angle of trigonometric function,0-1.000 of internal data corresponds to 0-360 degree (0 to 2π radian).

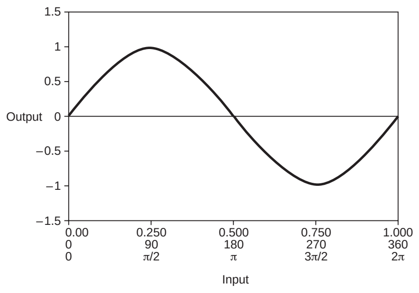


Figure 4.25.1 Input/output Characteristics of Sine Calculation

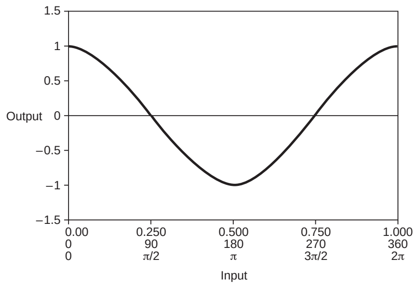


Figure 4.25.2 Input/output Characteristics of Cosine Calculation

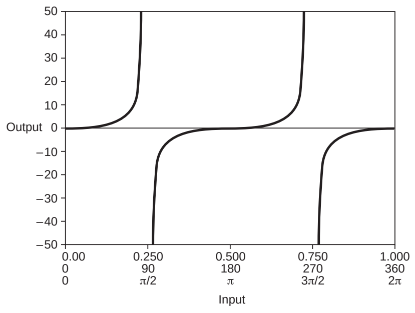


Figure 4.25.3 Input/output Characteristics of Tangent Calculation

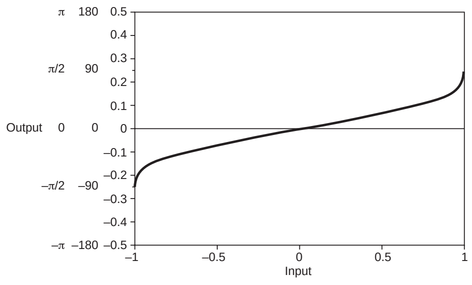


Figure 4.25.4 Input/output Characteristics of Arcsine Calculation

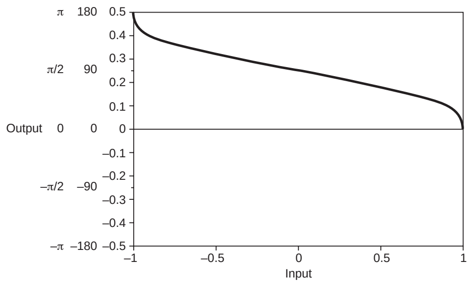


Figure 4.25.5 Input/output Characteristics of Arccosine Calculation

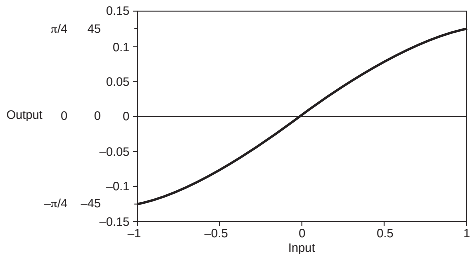


Figure 4.25.6 Input/output Characteristics of Arctangent Calculation

[Function Block]

A function block example of sine function can be expressed as shown in figure 4.24.7.

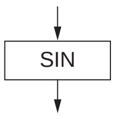


Figure 4.25.7 Function Block of Sine Calculation

[Program Example]

Figure 4.24.7 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	SIN	SIN (X1)		Sine calculation
G03	STY1	SIN (X1)		Store the result to Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.26 Other Functions

[Mnemonic Instruction Code]

LN	Natural Logarithm
LOG	Common Logarithm
EXP	Exponential Function
PWR	Power Function

[Operation]

All functions except PWR function are performed to the S1 register and the result is stored in the S1 register. The PWR function calculates “(S2)$^{(S1)}$” and stores the result in the S1 register. The behavior of registers when the PWR function is executed is shown in figure 4.26.1 below.

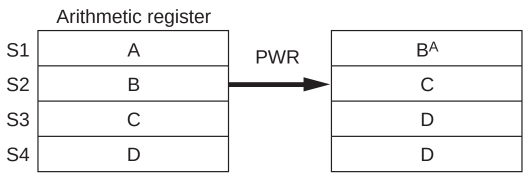


Figure 4.26.1 Behavior of S Registers When PWR Function Is Executed

[Program Example a]

The PWR function can be programmed as follows.

Step	Program Statement	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	LDC01	C01	X1	C01 (exponent)
G03	PWR	$X1^{C01}$		Exponential calculation
G04	STY1	$X1^{C01}$		Store the result to Y1
G05	Next computation			

[Function Block]

A function block example of the LOG function can be expressed as shown in figure 4.26.2.

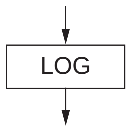


Figure 4.26.2 Function Block of LOG Function

[Program Example b]

Figure 4.26.2 can be programmed as shown in the table below.

Step	Program Statemen	S1	S2	Description
G01	LDX1	X1		Load input 1
G02	LOG	LOG (X1)		LOG calculation
G03	STY1	LOG (X1)		Store the result to Y1
G04	Next computation			

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: “G01 to G59” as “B20 to B59”
Fixed constants: “C01 to C59 (or H01 to H59)” as “C20 to C63”

4.27 Unconditional Jump

[Mnemonic Instruction Code]

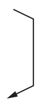
GOw w Unconditional Jump; where w w: 01 to 59 (20 to 59 for the WXT)

Unconditional jump is used to branch a program flow.

[Operation]

With this command, the program execution jumps to the step Gnn. The command does not change the arithmetic registers.

[Program Example]

Step	Program Statement	S1	S2	S3	Description
G01	GO04	A	B	C	
G02					
G03					
G04	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.28 Conditional Jump

[Mnemonic Instruction Code]

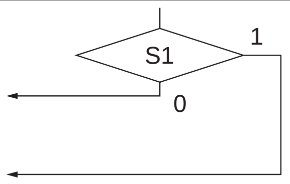
GIFnn Conditional Jump; where nn: 01 to 59 (20 to 59 for the WXT)

[Operation]

The program execution jumps to the step Gnn if the content of S1 register is 1. If the S1 register contains 0, then the program execution continues to the step next to the GIF command. After this command, the last data in S1 register will be lost and the data in S2 to S4 registers are pushed up to the S1 to S3 registers, respectively. The S4 register holds the old value.

The data in S1 register is considered to be 1 if it is 0.5 or more, and 0 if it is not.

[Program Example]

Step	Program Statement	S1	S2	S3	Description
G01	LDDI1	DI1	A	B	
G02	GIF06	A	B	C	
G03	Next computation				
G04					
G05					
G06	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.29 S Register Exchange

[Mnemonic Instruction Code]

CHG S Register Exchange

[Operation]

This command exchanges the data between S1 and S2 registers. The other registers, S3 and S4, still hold previous data. Figure 4.29.1 given below shows the behavior of registers when the CHG command is executed.

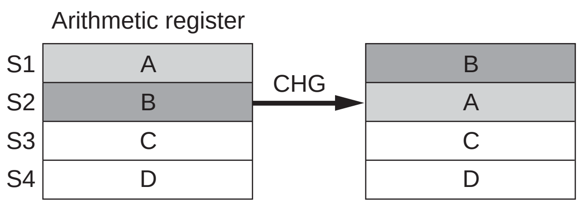


Figure 4.29.1 Behavior of S Registers When CHG Command Is Executed

[Function Block]

The function block can be expressed as shown in figure 4.28.2.

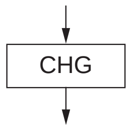


Figure 4.29.2 Function Block of S Register Exchange

[Program Example]

Exchanging the data between S1 and S2 registers can be programmed as follows.

Step	Program Statement	S1	S2	S3	Description
G01	LDX1	X1			Load input 1
G02	LDX2	X2	X1		Load input 2
G03	LDX3	X3	X2	X1	Load input 3
G04	CHG	X2	X3	X1	Exchange registers
G05	Next computation				

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.30 S Register Rotation

[Mnemonic Instruction Code]

ROT S Register Rotation

[Operation]

This command rotates data in all registers. Namely, the data in S2, S3, and S4 registers are pushed into the S1, S2, and S3 registers respectively and the data in the S1 register moves to the S4 register. The behavior of the registers when the ROT command is executed is shown below in figure 4.30.1.

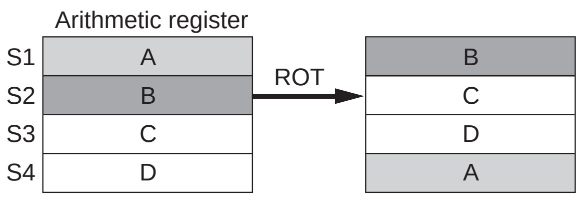


Figure 4.30.1 Behavior of the S Registers When ROT Command Is Executed

[Function Block]

The function block can be expressed as shown in figure 4.30.2.

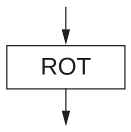


Figure 4.30.2 Function Block of S Register Rotation

[Program Example]

Loading and rotating the data in S registers can be programmed as follows.

Step	Program Statement	S1	S2	S3	S4	Description
G01	LDX1	X1				Load input 1
G02	LDX2	X2	X1			Load input 2
G03	LDX3	X3	X2	X1		Load input 3
G04	LDC01	C01	X3	X2	X1	Load fixed constant
G05	ROT	X3	X2	X1	C01	Rotate registers
G06	Next computation					

Note: For the WXT, read the program steps and fixed constants in this section as shown below:
Program steps: "G01 to G59" as "B20 to B59"
Fixed constants: "C01 to C59 (or H01 to H59)" as "C20 to C63"

4.31 No Operation

[Mnemonic Instruction Code]

NOP

[Operation]

This command has no effect on the program operation.

4.32 Contact Input and Output (for MXD-A and VJX7 with Optional Contact Output Only)

[Mnemonic Instruction Code]

Symbol	MXD	VJX7 with Optional Contact Output
LDDI1	Loads contact input	————
LDDO1	Loads contact output status	Loads contact output status
LDDO2	————	Loads contact output status
STDO1	Outputs contact	Outputs contact
STDO2	————	Outputs contact

Note: Those commands are available only with MXD computing unit which has the standard contact I/O function and VJX7 which has an optional contact output function. For other models, DO1 and DO2 registers can be used as user flags.

[Operation]

With these commands, you can input and output contacts in the same way as analog input and output.

The data '0' means contact ON and '1' means OFF.

LD: This command reads contact input and output statuses to the S1 register. The register's internal data is 0.000 for 0, and 1.000 for 1.

ST: This command outputs the data of the S1 register as a status. The status data is 0 (contact ON) if the register data is less than 0.5, and 1 (contact OFF) if it is not.

• MXD-A Computing Unit

[Function Block]

The function block can be expressed as shown in figure 4.32.1, which turns ON or OFF the contact output depending on the contact input status. The dotted line indicates the flow of the contact signal.

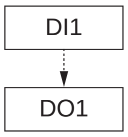


Figure 4.32.1 Function Block of Contact Input and Output 1

[Program Example]

Figure 4.32.1 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDDI1	0 or 1		Load contact input 1
G02	STDO1	0 or 1		Output contact output 1
G03	Next computation			

• VJX7 Computing Unit with Optional Contact Output

[Function Block]

The function block can be expressed as shown in figure 4.32.2, which exchanges the data between contact output 1 and 2.

DO1 ⇔ DO2

Figure 4.32.2 Function Block of Contact Output 1 and Output 2

[Program Example]

Figure 4.32.2 can be programmed as shown in the table below.

Step	Program Statement	S1	S2	Description
G01	LDDO1	0 or 1		Load contact output status 1
G02	LDDO2	0 or 1	0 or 1	Load contact output status 2
G03	STDO1	0 or 1	0 or 1	Output data in S1 register (old DO2 status) to contact output 1
G04	CHG	0 or 1	0 or 1	Exchange S1 and S2 registers
G05	STDO2	0 or 1	0 or 1	Output data in S1 register (old DO1 status) to contact output 2
G06	Next computation			

4.33 User Flag

[Mnemonic Instruction Code]

LDDOnLoading Status Signal

(n = 2 to 4 for the MXD; n = 1 to 4 for the MXS, MXT, VJX7*)

STDOnStoring Status Signal

(n = 2 to 4 for the MXD; n = 1 to 4 for the MXS, MXT, VJX7*)

*: For the VJX7 with an optional contact output, n = 3 or 4.

[Operation]

These commands can be used as temporary storage for the status signal (0 or 1).

4.34 End of Computation

[Mnemonic Instruction Code]

END

[Operation]

This command finishes the computation. If the program encounters the END command, it skips the rest of the computation and quits itself.

5. Programming

You can create programs by combining the examples mentioned in previous chapters. Refer to the following items when creating a program.

- Programs should be as simple as possible

  Auxiliary functions can be added at the later stage if required, but at the primary stage, programs should be simple, legible and consideration of the main objective function should be made.

- Programs should be understandable by anyone

  It is not necessary to create super-efficient programs that miss out steps. Such programs are difficult to understand. Bear in mind that programs should be understood by anyone.

- Make full use of exercises and precedents

  Making good use of examples and precedents can save labor and is a short cut to making accurate programs.Make use of the exercises given in this document.

5.1 Programming Procedures

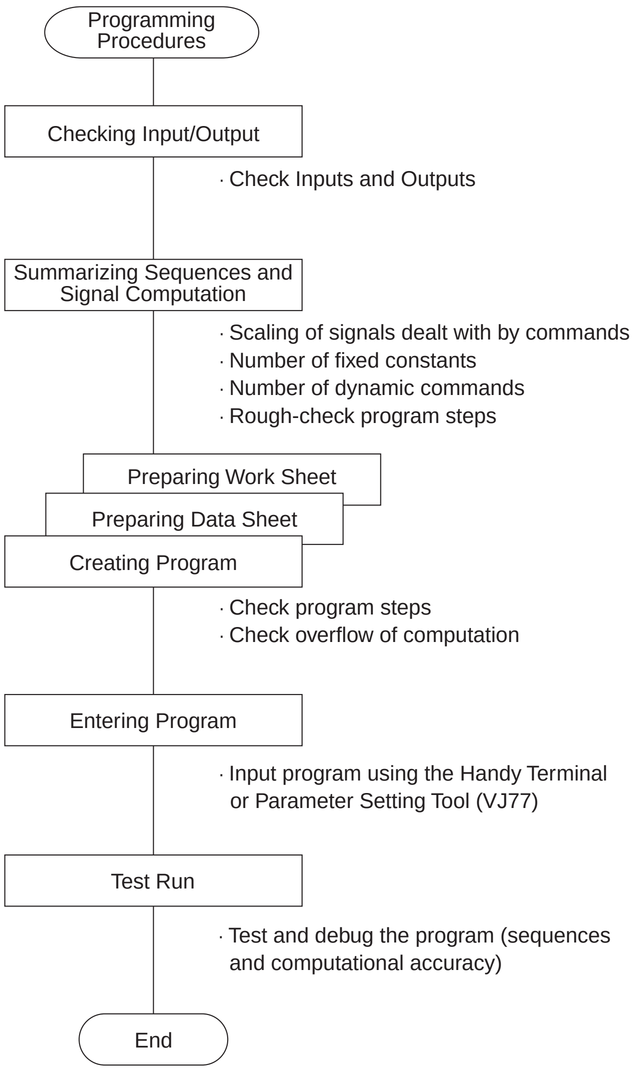

Programming Procedures
Checking Input/Output
· Check Inputs and Outputs
Summarizing Sequences and Signal Computation
· Scaling of signals dealt with by commands
· Number of fixed constants
· Number of dynamic commands
· Rough-check program steps
Preparing Work Sheet
Preparing Data Sheet
Creating Program
· Check program steps
· Check overflow of computation
Entering Program
· Input program using the Handy Terminal or Parameter Setting Tool (VJ77)
Test Run
· Test and debug the program (sequences and computational accuracy)
End

Figure 5.1.1 Programming Procedure Flow

5.2 Programming Exercise (Temperature-pressure compensation for ideal gas flow control)

This example of computation process shows temperature-pressure compensation of gas flow (Programmable on the MXT-A and WXT-A units).

The example describes scaling of the arithmetic computation and the method of determining a fixed constant.

(1) Determining computation method

As shown in figure 5.2.1, to compensate for flow signals from a differential pressure transmitter, pressure and temperature are adjusted.

Input signals from each detector are entered into the computing unit via a distributor or converter.

The unit performs computation of the three inputs; differential pressure, pressure, and temperature, and then outputs the compensated values.

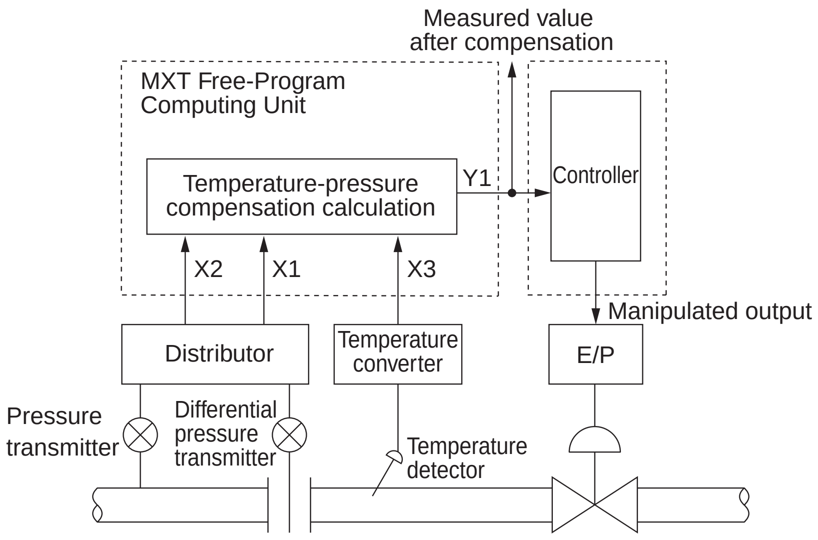

Measured value after compensation
MXT Free-Program Computing Unit
Temperature-pressure compensation calculation
Y1
Controller
X2
X1
X3
Manipulated output
Distributor
Temperature converter
E/P
Pressure transmitter
Differential pressure transmitter
Temperature detector

Figure 5.2.1 Temperature-Pressure Compensation for Gas Flow Control

(2) Assembling of the arithmetic computation

Temperature-pressure compensation for ideal gas flow is normally expressed as follows:

$$Q_x = \sqrt{\frac{P_f \cdot T_n}{P_n \cdot T_f} \cdot \Delta P}$$

ΔP	:	Differential pressure
Q_x	:	Gas flow converted to reference conditions
P_f	:	Working gas pressure expressed as an absolute pressure
P_n	:	Orifice design reference pressure expressed as an absolute pressure
T_f	:	Working gas temperature expressed as an absolute temperature
T_n	:	Orifice design reference temperature expressed as an absolute temperature

(3) Normalization of Computation

Physical quantity of compensation expression is converted to normalized signal (0.000-1.000) of computing unit.

$P_f = P_{span} \cdot X2 + P_{min}$

$T_f = T_{span} \cdot X3 + T_{min}$

$Q_x = Q_{span} \cdot Y$

$\Delta P = \Delta P_{span} \cdot X1$

where X1	:	Differential pressure signal (0.000-1.000)
X2	:	Pressure signal (0.000-1.000)
X3	:	Temperature signal (0.000-1.000)
Y	:	Compensated flow signal (0.000-1.000)

By substituting normalized signal values, the following expression is obtained:

$$Q_{span} \cdot Y = \sqrt{\frac{\frac{P_{span}}{P_n} \cdot X2 + \frac{P_{min}}{P_n}}{\frac{T_{span}}{T_n} \cdot X3 + \frac{T_{min}}{T_n}} \cdot \Delta P_{span} \cdot X1}$$

Orifice Qx and ΔP are normally designed so as to be $Y=\sqrt{X}$ in design reference condition. Therefore, scaling by $\sqrt{\Delta P_{span}}/Q_{span}$ is not needed and as shown below, temperature pressure compensation expression is

$$Y = \sqrt{\frac{K2 \cdot X2 + A2}{K3 \cdot X3 + A3} \cdot X1}$$

where $K2 = P_{span}/P_n$

$A2 = P_{min}/P_n$

$K3 = T_{span}/T_n$

$A3 = T_{min}/T_n$.

Since the MXT computing unit has three inputs, temperature and pressure compensations can be made in one unit.

Setting conditions for computation of temperature-pressure compensation

1. Orifice design reference pressure: P_n = 600 kPa
2. Orifice design reference temperature: T_n = 300°C
3. Pressure transmitter range: 0-1000 kPa
4. Temperature converter range: 0-500°C

By substituting the above data, the following expressions can be used to obtain K2, A2, K3, and A3:

K2 = Pressure transmitter span/Orifice design reference pressure (absolute pressure) = 1000/(600 + 101.3) = 1.426

A2 = Pressure transmitter minimum scale (absolute pressure)/Orifice design reference pressure (absolute pressure) = (0 + 101.3)/(600 + 101.3) = 0.1445

K3 = Temperature converter span/Orifice design reference temperature (absolute temperature) = 500/(300 + 273.15) = 0.8724

A3 = Temperature transmitter minimum scale (absolute temperature)/Orifice design reference temperature (absolute temperature) = (0 + 273.15)/(300 + 273.15) = 0.4766

By substituting the above:

$$Y = \sqrt{\frac{1.426 \cdot X2 + 0.1445}{0.8724 \cdot X3 + 0.4766}} \cdot X1$$

(4) Fixed constants setting

Allocation of the fixed constants is as shown below:

Pressure compensation computation

K2 = 1.426	C02 = 142.6%
A2 = 0.1445	C04 = 14.45%

Temperature compensation computation

K3 = 0.8724	C07 = 87.24%
A3 = 0.4766	C08 = 47.66%

(5) Assembling of the function block

This example can be assembled in the function block by one computation expression:

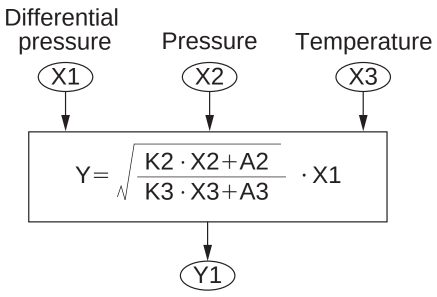


Figure 5.3.1 Flow Sheet of Function Block

(6) Data entry

After completion of the allotment of input/output and constants, the data is entered in the data sheet.

Table 5.1 shows an example of data entry of temperature-pressure compensation computation.

Table 5.1 Example of Data Sheet Entry (Temperature-Pressure Compensation Computation)

Data		Description	0%	100%
Analog input	X1	Differential pressure (mmH_2O)	0	3200
	X2	Pressure (kPa)	0	1000
	X3	Temperature (˚C)	0	500.0
Analog output	Y	Temperature-pressure compensation output (%)	0	100.0

Fixed constant	Value	Description
C02	142.6%	K2 = 1.426
C03		
C04	87.24%	K3 = 0.8724
C05		
C06		
C07	14.45%	A2 = 0.1445
C08	47.66%	A3 = 0.4766
C09	0.6%	

K2 = C02
K3 = C04
A2 = C07
A3 = C08

(7) Program sheet

Table 5.2 shows an example of a program for temperature-pressure compensation computation. Enter the data in the program field of the table. Explanations are entered into “Description” column of the program sheet so that execution can be easily and quickly understood at a later date. The arithmetic register columns of the program sheet indicate the contents of registers, other than S4 register, following the execution of each program step. Data does not have to be entered in the arithmetic register columns.

Table 5.2 Program Example (Temperature-Pressure Compensation Computation)

Step	Program Statement	S1	S2	S3	Description
G01	LDX2	X2			Load pressure signal
G02	LDC02	C02	X2		Load fixed constant C02 = 142.6% (K2 = 1.426)
G03	MLT	C02 × X2			K2 × X2
G04	LDC07	C07	C02 × X2		Load fixed constant C07 = 14.45% (A2 = 0.1445)
G05	ADD	a			Pressure compensating term (a = C02 × X2 + C07)
G06	LDX3	X3	a		Load temperature signal
G07	LDC04	C04	X3	a	Load fixed constant C04 = 87.24% (K3 = 0.8724)
G08	MLT	C04 × X3	a		K3 × X3
G09	LDC08	C08	C04 × X3	a	Load fixed constant C08 = 47.66% (A3 = 0.4766)
G10	ADD	b	a		Temperature compensating term (b = C04 × X3 + C08)
G11	DIV	a/b			Pressure and temperature compensating term computation
G12	LDX1	X1	a/b		Load differential pressure signal
G13	MLT	a/b × X1			
G14	LDC09	C09	a/b × X1		Load low-cut point C09 = 0.6%
G15	SQT	$\sqrt{a/b \cdot X1}$			Temperature-pressure signal compensation computation
G16	STY1	$\sqrt{a/b \cdot X1}$			Output compensated signal
G17	END				

Note: The values in the S1, S2, and S3 registers show data related to compensating computation only.

6. Program Entry and Setting Fixed Constants

There are two ways to enter program code or set fixed constants.

One way is to use the Handy Terminal (JHT200) that deals with data one item at a time. The other is to use the Parameter Setting Tool (VJ77) that can collectively process data sent from a PC. Both of them can set and read program code and data, and monitor the data.

Refer to the respective document for details.

Appendix 1. List of Program Functions

Category	Command symbol	Command	Arithmetic Register						Description
			Before Command Execution			After Command Execution			
			S1	S2	S3	S1	S2	S3	
Load	LDXn	Load Xn	A	B	C	Xn	A	B	For MXS, VJXS, VJX7, and MXD: n = 1; For MXT and WXT:n=1 to 3(X_n:Input)
	LDYn	Load Yn	A	B	C	Yn	A	B	For MXD, MXT, VJXS, VJX7 without optional analog output, and WXT: n = 1; For MXS, VJXS, and VJX7 with optional analog output: n = 1 to 2 (Yn: Output)
	LDCnn	Load Cnn	A	B	C	Cnn	A	B	nn = 01 to 59, Cnn: Fixed constant *1
	LDTn	Load Tn	A	B	C	Tn	A	B	n = 1 to 4, Tn: Buffer register
	LDDIn	Load DIn	A	B	C	DIn	A	B	n = 1, DIn: Digital input *2
	LDDOn	Load DOn	A	B	C	DOn	A	B	n = 1 to 4, DOn: Digital output *3
Store	STXn	Store in Xn	A	B	C	A	B	C	Stores S1 in Xn.
	STYn	Store in Yn	A	B	C	A	B	C	Stores S1 in Yn.
	STTn	Store in Tn	A	B	C	A	B	C	Stores S1 in Tn.
	STDOn	Store in DOn	A	B	C	A	B	C	Stores S1 in DOn.
Basic Operation	ADD	Addition	A	B	C	B+A	C	D	S1←S2+S1
	SUB	Subtraction	A	B	C	B−A	C	D	S1←S2−S1
	MLT	Multiplication	A	B	C	B×A	C	D	S1←S2×S1
	DIV	Division	A	B	C	B÷A	C	D	S1←S2÷S1
	SQR	Square root extraction	A	B	C	$\sqrt{A}$	B	C	S1←$\sqrt{S1}$
	ABS	Absolute value	A	B	C	|A|	B	C	S1←|S1|
	HSL	High selector	A	B	C	The higher of A and B	C	D	Compares S1 and S2 registers and stores the higher in S1
	LSL	Low selector	A	B	C	The lower of A and B	C	D	Compares S1 and S2 registers and stores the lower in S1
	HLM	High limiter	Upper limit	Input	A	Limited input	A	B	If input is less than the upper limit it is stored in S1, if not the upper limit value is stored in S1.
	LLM	Low limiter	Lower limit	Input	A	Limited input	A	B	If input is more than lower limit it is stored in S1, if not the lower limit value is stored in S1.
	FX1,2	Ten-segment linearizer	Input	A	B	Function output	A	B	Ten-segment linearizer with equally divided ten break points FX1: C01 to C11 Arbitrary ten-segment linearizer FX2: C12 to C33 *4
	FX3,4	Arbitrary line segment linearizer	Input	A	B	Function output	A	B	Arbitrary twenty-segment linearizer X-axis: C01 to C21, Y-axis: C22 to C42 *5*6
	CMP	Comparison	A	B	C	0/1	B	C	If S1 > S2, 0.000 → S1; If not, 1.000→S1
	SW	Signal switching	0/1	A	B	AorB	C	C	If S1 < 0.500, S3 → S1; If not, S2→ S1
Function Operation	SIN	Sine	A	B	C	SINA	B	C	S1←SIN(S1) *7
	COS	Cosine	A	B	C	COSA	B	C	S1←COS(S1) *7
	TAN	Tangent	A	B	C	TANA	B	C	S1←TAN(S1) *7
	ASIN	Arcsine	A	B	C	ASINA	B	C	S1←ASIN(S1) *7
	ACOS	Arccosine	A	B	C	ACOSA	B	C	S1←ACOS(S1) *7
	ATAN	Arctangent	A	B	C	ATANA	B	C	S1←ATAN(S1) *7
	LN	Natural logarithm	A	B	C	LN (A)	B	C	S1←LN(S1)
	LOG	Common logarithm	A	B	C	LOG(A)	B	C	S1←LOG(S1)
	EXP	Exponential Function	A	B	C	EXP(A)	B	C	S1←EXP(S1)
	PWR	Power Function	A	B	C	B^A	C	D	S1←$S2^{S1}$

A, B, C and D represent data prestored in arithmetic registers.

Although the computing unit has the four built-in registers S1 to S4, this table only shows S1 to S3.

*1: nn = 20 to 63 for the WXT.
*2: This function can be used for the MXD only.
*3: For the MXD, DO1 is used as a digital output and DO2 to DO4 are used as buffers.
For a VJX7 with optional contact output, DO1, DO2, DO3 and DO4 are used as buffers.
*4: For the WXT, FX1: C20 to C30 and FX2: C31 to C52.
*5: For the WXT, FX3: C20 to C61.
*6: For the FX4 linearizer, set as many data items as the number of line segments specified using C43. For the VJX7 and WXT, this function cannot be used.
*7: This function can be used for the VJX7 and WXT only.

Category	Command symbol	Command	Arithmetic Register						Description
			Before Command Execution			After Command Execution			
			S1	S2	S3	S1	S2	S3	
Dynamic Operation	LAGn	First-order lag	Time constant	Input	A	First-order lag computation output	A	B	Performs first-order lag computation of input, and stores the value in S1 (n = 1 to 3).
	LEDn	Differential calculus	Time constant	Input	A	Differential calculus output	A	B	Performs differential calculus computation of input, and stores the value in S1 (n = 1 to 3).
	DED	Dead time	Dead time	Input	A	Dead time computation output	A	B	Stores in S1, the input value sampled at a point of time a specified period before the current time *1
	VEL	Velocity	Computation time	Input	A	Velocity computation output	A	B	Subtracts the input value sampled at a point of time a specified period before the current time from the sampled current input value and stores it in S1 *1
	VLMn	Velocity limiter	Descending limit	Ascending limit	Input	Limited velocity input	A	A	Limits input velocity within preset limits and stores the value in S1 (n = 1 to 2).
	MAV	Moving average computation	Computation time	Input	A	Moving average	A	B	Calculates average from a time preset in the past to now and stores the result in S1 *1
	TIM	Timer	0/1	A	B	Elapsed time	A	B	If S1 < 0.500, resets timer; If not, starts or continues timer.
	CCD	Status change detection	0/1	A	B	0/1	A	B	When S1 changes 0 to 1, S1 = 1 *2
	PIC	Pulse input counter	Reset counter	Input	A	Counter output	A	B	When S1 changes 0 to 1, S1 = 1 *2
	SQT	Square root extraction with variable low-cut point	Low-cut point	A	B	Low-cut $\sqrt{A}$	B	C	S1 ← Low-cut $\sqrt{S2}$
	SQAn	Square root extraction with variable low-cut point	Low-cut point	A	B	Low-cut $\sqrt{A}$	B	C	S1 ← Low-cut $\sqrt{S2}$(n = 1 to 3)
	SQBn	Square root extraction with variable low-cut point	Low-cut point	A	B	Low-cut $\sqrt{A}$	B	C	S1 ← Low-cut $\sqrt{S2}$(n = 1 to 3)
	CPO	Integrated pulse output	Integrating ratio	Input	A	Input	A	B	Converts input S2 to pulses using S1 as integrating ratio and outputs the result as digital signals *2
	HALn	High alarm	Hysteresis	Alarm point	Input value	0/1	A	A	Stores in S1, the result of comparing the input value of the S3 register with the alarm point of the S2 register and the hysteresis of the S1 register (n = 1 and 2). If the alarm is on, S1 = 1. *4
	LALn	Low alarm							
Logical Operation	AND	Logical AND	A	B	C	A∩B	C	D	S1←S2∩S1
	OR	Logical OR	A	B	C	A∪B	C	D	S1←S2∪S1
	NOT	Logical NOT	A	B	C	$\overline{A}$	B	C	S1←$\overline{S1}$
	EOR	Logical EOR	A	B	C	A⩂B	C	D	S1←S2⩂S1
Other	GOnn	Unconditional jump	A	B	C	A	B	C	Jumps to step Bnn (nn = 01 to 59) *3
	GIFnn	Conditional jump	A	B	C	B	C	D	If S1 = 0 jumps to the next step, if S1 = 1 jumps to step Bnn (nn = 01 to 59) *3
	CHG	S register exchange	A	B	C	B	A	C	Exchanges S1 and S2 registers.
	ROT	S register rotation	A	B	C	B	C	D	S2→S1, S3→S2, S4→S3, S1→S4
	NOP	No operation	A	B	C	A	B	C	No operation
	END	End of computation	A	B	C	A	B	C	

A, B, C and D represent data prestored in arithmetic registers.

Although the computing unit has the four built-in registers S1 to S4, this table only shows S1 to S3.

*1: Any two commands among dead time, velocity, and moving average cannot be used together in a program step.
*2: For MXD only.
*3: For the WXT, nn = 20 to 59.
*4: For the VJX7 and WXT, this function cannot be used.

Appendix 2. Work Sheet

WORKSHEET	Model and Suffix Codes
	TAG No.
Plant	

△	No.	Revised by	CH.	CH.	K	CUSTOMER		REP.		ENGINEER		K
△						DR.	CH.	DR.	CH.	DR.	CH.	
△												
△												

Appendix 3. Data Sheet (for WXT)

DATA SHEET	Program No. :
	Model :
Tag No. :	Serial No. :
Plant	

Data		Description	0%	100%	Buffer Register		Digital Input (DI1), Output (DOn)	
Analog Input	X1				T1		DO1	
	X2				T2		DO2	
	X3				T3		DO3	
Analog Output	Y1				T4		DO4	
	Y2				Note: Use DO2 to DO4 as user flags.		DI1	

Fixed Constant	Value	Description	Fixed Constant	Value	Description
C20			C42		
C21			C43		
C22			C44		
C23			C45		
C24			C46		
C25			C47		
C26			C48		
C27			C49		
C28			C50		
C29			C51		
C30			C52		
C31			C53		
C32			C54		
C33			C55		
C34			C56		
C35			C57		
C36			C58		
C37			C59		
C38			C60		
C39			C61		
C40			C62		
C41			C63		

Remarks:

△	No.	Revised	Dr.	Ch.	Customer	App.	Ch.	Dr.

Appendix 4. Data Sheet (for VJX7, VJXS, MXS, MXD and MXT)

DATA SHEET	Program No. :
	Model : VJX7
Tag No. :	Serial No. :
Plant	

Data		Description	0%	100%	Buffer Register		Digital Output (DOn)	
Analog Input	X1				T1		DO1	
	X2				T2		DO2	
	X3				T3		DO3	
Analog Output	Y1				T4		DO4	
	Y2				Note: The analog input (Xn) and output (Yn) registers that do not corresspond to actual hardware I/O ports can be used as buffers or user flags.			

Fixed Constant	Value	Description	Fixed Constant	Value	Description
H01			H31		
H02			H32		
H03			H33		
H04			H34		
H05			H35		
H06			H36		
H07			H37		
H08			H38		
H09			H39		
H10			H40		
H11			H41		
H12			H42		
H13			H43		
H14			H44		
H15			H45		
H16			H46		
H17			H47		
H18			H48		
H19			H49		
H20			H50		
H21			H51		
H22			H52		
H23			H53		
H24			H54		
H25			H55		
H26			H56		
H27			H57		
H28			H58		
H29			H59		
H30					

Remarks:

△	No.	Revised	Dr.	Ch.	Customer	App.	Ch.	Dr.

Appendix 5. Program Sheet

Page

PROGRAM SHEET	Program No. :
	Model :
Tag No. :	Serial No. :

Plant

Step	Command	S1	S2	S3	Description

Remarks:

△	No.	Revised	Dr.	Ch.	Customer	App.	Ch.	Dr.

Blank Page

Revision Information

● Title	:	Function of VJ, M, and WXT Free-Program Computing Units
● Manual No.	:	TI 231-01E

Feb. 2000/1st Edition
Newly published

May 2004/2nd Edition

Jul. 2005/3rd Edition

Dec. 2005/4th Edition

Blank Page