

Ladder Diagram

for Festo Controllers

Programming Ladder Diagram with Festo FST

Order no.:	18348
Description:	MANUAL KOP
Designation:	E.HB-KOP-GB
Status:	02/97
Author:	B. Plagemann
Editor:	S. Baerwald, YC-ECI

2nd edition

© Copyright by Festo KG, 73734 Esslingen,
Germany, 1991

All rights, including translation rights, reserved.
No part of this document may be reproduced
in any form (printing, photocopy, microfilm or
any other process) without the prior written
permission of Festo KG.

Programming Ladder Diagram with Festo FST

Preface

This book is intended as an aid to anyone who has to program programmable logic controllers from Festo.

It is not intended as a textbook but as a reference work and an aid to solving problems associated with control technology. All operations and operands that can be used with a Festo PLC are explained and - in most cases - illustrated with examples.

You know the sort of question that sometimes occurs when working on a project. "How do you carry out an exclusive OR operation in Ladder diagram?"

This is where this book provides help: simply refer to the section "Exclusive Or" and you'll find a program example.

It's as simple as that.

Have you found any errors? Have you found any particular question unanswered?

Certainly there will be room for improvement. Help us to improve this book by sending a postcard with a description of the missing solution, the error or with any other comments to:

Festo KG
Ruiter Str. 82
D-73734 Esslingen
Germany

Many thanks!

Programming Ladder Diagram with FESTO FST

Contents

Chapter 1	Fundamental characteristics of FST software	7
1.1	Notes	9
1.2	Structure of FST Software.....	10
1.3	Programming languages	11
Chapter 2	Entering a program.....	13
Chapter 3	Operands	19
3.1	Notes	21
3.2	Bit and word operands	23
3.3	Bit operands	24
3.4	Word operands	26
3.5	Retentive operands.....	29
Chapter 4	Structure of a Ladder diagram program .	31
4.1	Notes	33
4.2	What is a rung comprised of?	34
4.3	Conditional and executive part	37
4.4	How is a rung processed?.....	38
4.5	How fast is a controller?	46
Chapter 5	Operations - an overview	47
Chapter 6	The operations in detail	
	Examples for reference	59
Chapter 7	Addressing operands	143
7.1	Absolute and symbolic addresses.....	145
7.2	Information content of absolute addresses.....	153

Programming Ladder Diagram with FESTO FST

Chapter 8	Programming timers	159
	8.1 General	161
	8.2 Timer types	165
	8.3 Examples	169
	8.4 Evaluation of the individual timer elements	178
Chapter 9	Programming counters	181
	9.1 What is an incremental counter?	184
	9.2 Elements of the counter	186
	9.3 Programming counters.....	191
Chapter 10	Using registers	197
Chapter 11	Using flags	201
	11.1 Flag bit.....	203
	11.2 Flag word	207
Appendix A	Example programs	209
	A.1 Sequence programs.....	211
	A.2 Measuring the cycle time	220
	A.3 Structured programming in Ladder diagram	233
Appendix B	Differences in program execution.....	239
	B.1 Process image	241
	B.2 Multitasking	244
	B.3 Multiprocessing	246
	B.4 Interrupt capability	247
	B.5 Internal inputs and flags	248
	B.6 Serial data port	251
	B.7 Behaviour after battery failure.....	253
	B.8 Input signal delay	255

Programming Ladder Diagram with FESTO FST

Appendix C	Subroutines, Multitasking and Multiprocessing	257
	C.1 Modules	265
	C.2 Multitasking	267
	C.3 Multiprocessing	272
Appendix D	The binary and hexadecimal number system	277
	D.1 The structure of number systems	279
	D.2 The binary number system	282
	D.3 Differentiation in Ladder diagram	289
Index	291

Programming Ladder Diagram with FESTO FST

Chapter 1

Fundamental characteristics of FST Software

Programming Ladder Diagram with FESTO FST

Contents

1.1	NOTES	9
1.2	STRUCTURE OF FST SOFTWARE.	10
1.3	PROGRAMMING LANGUAGES.....	11

Programming Ladder Diagram with FESTO FST

1.1 NOTES

In FST software every self-contained PLC system is called a **project**. Each project can contain one or more **programs**, **modules** and program and module **versions**.

Each project has an **allocation list** that can contain several (internal and external) parts.

Each program or module version has its own **error list**, which records all errors found by the system.

Within a project individual programs and modules can be processed together (cross-reference list, load, print).

Programs or modules can be imported from other projects.

Documentation on a project can be created with a comment header and operating instruction.

Programming Ladder Diagram with FESTO FST

1.2 STRUCTURE OF THE FST SOFTWARE

You want to program an automatic assembly machine for pneumatic valves. A rotary indexing table offers 8 automatic work stations and a manual work station. The valve is fully assembled on this rotary indexing table. It is then placed on a conveyor belt which transports the valve to an automatic testing machine and to a packing machine.

A programmable logic controller is used for the automatic assembly machine. The entire program for this controller is a **project**.

Each automatic work station and the rotary indexing table are individual **programs** or **modules within this project**. The arrangement will depend on the programming method used by the programmer.

Sensors and actuators are mounted on the automatic assembly machine. These are connected to the inputs and outputs of the PLC - the **operands** of the PLC. In addition there are internal operands such as flags, timers, counters, registers, programs and modules.

A project may not only contain several programs or modules but also - when using the FPC 405 - several **processor modules**. All of these processor modules access the inputs and outputs. Each processor module can again execute several programs or modules - but only **within one project**.

Programming Ladder Diagram with FESTO FST

Documentation is prepared for the entire project. The documentation comprises

- Cover sheet with the name of the project
- Text to describe in the project and the individual processor modules, programs, modules, etc., possibly with notes on operation, spare parts, etc.
- Allocation list with all operands
- Printout of all programs and modules with comment lines
- Cross-reference list showing which operand is used at which point in which program or module.

1.3 PROGRAMMING LANGUAGES

The FST software comprises a range of programming languages for programming the various PLCs. These include

- Ladder diagram
- Statement list
- BASIC
- Assembler
- Matrix programming

This book deals only with ladder diagram programming, which is available with all Festo PLCs (FPC 100, FPC 405, IPC, FEC).

Notes

[illegible]

2. Entering a program

Chapter 2
Entering a program

2. Entering a program

Contents

Entering a program	15
--------------------------	----

2. Entering a program

When creating a Ladder diagram program with FST software, we recommend the following procedure:

- **Plan the program** as a function chart, sequence diagram, text, structure chart or similar.
- **Call up FST software**
- **Create project:**
The project name must meet the requirements of the MS-DOS operating system, as a subdirectory is created on your mass storage medium (normally the hard disk) for the project. This means that the project name can be up to 8 characters, and must not contain spaces or periods. For MS-DOS version 3.0 or later, a subdirectory may also contain special characters such as the German Umlaut (ä, ü, etc.). This does not apply to DOS versions prior to 3.0.
- Enter text for **Title page** and page header:
This makes your printout easier to read.
- Create an **allocation list** for inputs and outputs:
It makes sense to create the allocation list for inputs and outputs before entering the program and then to print it out immediately. Then it is not necessary to write down inputs and outputs.

2. Entering a program

- Entering the **Ladder diagram program**:
When creating the program you are prompted for some or all of the following entries, depending on the controller type:

Central control unit

Allows the entry of the processor module number for projects involving several processors (FPC 405).

Important: If only one CCU is used, no. 0 must be used.

Default value: No. 0

Type program/module

Allows you to indicate whether you want to create a program or a module.

Important: At least one program must be entered.

Default value: Program

Prog./module no.

Allows entry of the number for the program or module.

Important: A program number 0 must always exist, otherwise your processor module cannot start execution.

Default value: 0

2. Entering a program

Version number

Allows you to create up to 9 different versions of a program or module.

Important: You can only load one version of a program or module into a processor module.

Default value: 1

Task

Here you can enter a commentary which will help you to relocate this particular program, version or module. This commentary does not appear on any printout and is only intended as a programming aid.

You want to start a completely new program? Simple. Leave everything as it is and press the <F1> key for "Create". The default values are such that you can load a controller with a single program and execute it.

2. Entering a program

- **Enter program:**
The Ladder diagram editor for FST software is controlled by means of the function keys and/or mouse (version 3.1 or later). You only use the keyboard of your PC for entering the names of operands. All operations (with the exception of entries in the boxes) are called with the aid of function keys or the mouse. Function key <F9> is the help key. It opens a help window where ever you are in the FST software. Function key <F10> toggles the function key allocation.
- **Save program:**
Using function key "File menu" (normally <F8>).
- Test the **syntax** after program. If necessary, view or print the error list and correct the errors.
- **Print out the program.**
- **Load the program into the controller.**
- **Test the program** with the aid of the on-line function and with the aid of the status display (version 3.1 or later).
- Create **Documentation**.

3. Operands

Chapter 3

Operands

3. Operands

Contents

3.1	NOTES	21
3.2	BIT AND WORD OPERANDS	23
3.3	BIT OPERANDS.....	25
3.4	WORD OPERANDS.....	27
3.5	RETENTIVE OPERANDS	30

3. Operands

3.1 NOTES

The FST software allows the use of bit operands and word operands (no byte and no double/longword operands).

The FST software allows the use of absolute and symbolic operand addresses. Only absolute addresses are used in the following. For details on symbolic addresses see Chapter 7, "Addressing the operands".

The number of operands of an individual type that can be used depends on the type of Festo PLC.

3. Operands

The absolute operand address has the following structure:

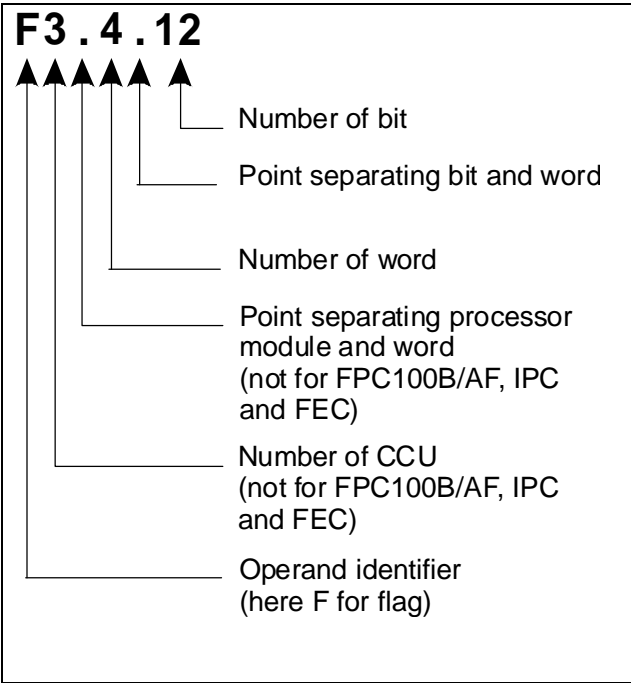


Fig. 3/1: Construction of an absolute operand address

The operand address must never contain space characters!

3. Operands

3.2 BIT AND WORD OPERANDS

All Festo PLCs have a word-oriented internal organization. This means that all operands are organized and can be used as words (with the exception of the initialization flag FI, the one-bit operands T for Timer and C for Counter and the operands P1 for Program and CMP/CFM for modules).

Every word is 16 bits long and can be used for an unsigned decimal number (0 to 65,535), as a signed decimal number (-32,767 to +32,767) or as a hexadecimal number (\$0 to \$FFFF). For inputs and outputs not all 16 bits are always available externally, so that in many cases only 8 bits of an input or output word can be used.

In the case of input, output and flag words each bit can be addressed individually.

In the case of registers, timer preselects, counter words and error word, the words can only be addressed as words.

The counter commands increment and decrement can be used with all words, not only counter words.

Arithmetic and logic operations can be used for all words.

3. Operands

3.3 BIT OPERANDS

Operand	Designator	Syntax	Application	Example
Output	O	On.n (On)	Condition	
Output	O	On.n (On)	Execution	
Module	CMP	CMP n	Execution	
Input	I	In.n (In)	Condition	
Error	E	E	Condition	
Function module	CFM	CFM n	Execution	
Flag	F	Fn.n.n (Fn.n)	Condition	
Flag	F	Fn.n.n (Fn.n)	Execution	

3. Operands

Operand	Designator	Syntax	Application	Example
Initializa- tion flag	FI	FI	Condition	
Program ¹	P	Pn	Condition	
Program ¹	P	Pn	Execution	
Timer	T	Tn	Condition	
Timer	T	Tn	Execution	
Counter	C	Cn	Condition	
Counter	C	Cn	Execution	
Counter	C	Cn	Execution	
Counter	C	Cn	Execution	

1. The FPC 100 does not have the operand Program.

Important: There is no „Program“ operand for the FPC 100B/AF. The syntax may vary according to the controller (as shown in brackets).

3. Operands

3.4 WORD OPERANDS

Operand	Designator	Syntax	Application	Example
Output word	OW	OWn (OW)	Condition	
Output word	OW	OWn (OW)	Execution	
Input word	IW	IWn (IW)	Condition	
Error word	EW	EW	Condition	
Error word	EW	EW	Execution	
Function unit	FU	FUnn	Condition	
Function unit	FU	FUnn	Execution	
Constant	V	Vnnnnn	Condition	

1. Any multibit operand can be entered here.
2. Any multibit operand can be entered here with the exception of an input word.

3. Operands

Operand	Designator	Syntax	Application	Example
Timer preselect	TP	TPn	Execution	
Timer word	TW	TWn	Condition	
Timer word	TW	TWn	Execution	
Counter preselect	CP	CPn	Condition	
Counter preselect	CP	CPn	Execution	
Counter word	CW	CWn	Condition	
Counter word	CW	CWn	Execution	

1. Any multibit operand can be entered here.
2. Any multibit operand can be entered here with the exception of an input word.

Important: The syntax of the flag word varies with a greater or lesser number of operands depending on the type of controller used. *nn* **always** starts at 0 and goes to the type-dependent maximum value.

3. Operands

3.5 RETENTIVE OPERANDS

Retentiveness is the property of an operand to retain its status even during a power failure. As operands are always stored in RAM, this property is only possible if a buffer battery is installed, or in a flash memory.

In the case of Festo PLCs, all operands are retentive where it makes sense, i.e. all flags, registers, counters, counter preselects, counter words and timer words.

All outputs are generally volatile. For safety reasons all outputs must not remain ON in the event of a power failure.

In the event of power failure **and** battery failure all operands (except outputs) must be initialized by the application program, setting them to the desired initial value (possibly with the aid of an appropriate module to zero all operands).

Notes

[illegible]

4. The structure of a ladder diagram program

Chapter 4

Structure of a Ladder diagram program

4. The structure of a ladder diagram program

Contents

4.1	NOTES	33
4.2	WHAT IS A RUNG COMPRISED OF?	34
4.3	CONDITIONAL AND EXECUTIVE PART	37
4.4	HOW IS A RUNG PROCESSED?	38
	Advantages of the process image method	44
	Advantages of the direct access method.....	45
4.5	HOW FAST IS A CONTROLLER?....	46

4. The structure of a ladder diagram program

4.1 NOTES

A Ladder diagram program consists of **Rungs**. Each rung has a minimum of one contact set, and a maximum of 12 contacts in series. Each rung also has at least one, and generally at least 6 rung sections.

Additionally up to 9 parallel branches can be inserted, resulting in a maximum of 10 contacts in parallel.

The parallel branches can be positioned anywhere, except that they must not overlap.

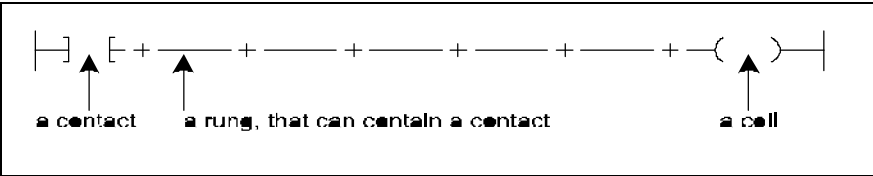
Coils can also be configured in parallel, but they **must all have one common input**. A rung can thus only send a single logical signal to all coil contacts.

4. The structure of a ladder diagram program

4.2 WHAT IS A RUNG COMPRISED OF?

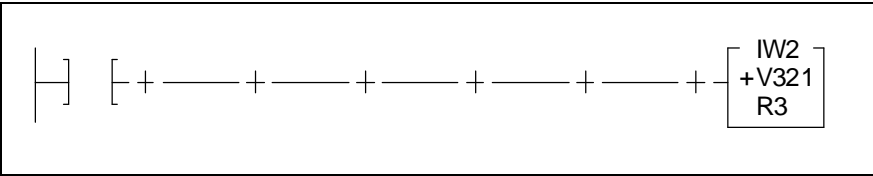
In its simplest form the Ladder diagram is an image of the American circuit diagram. It is similar to the German circuit diagram, but is always read **from left to right**. First let's look at some terms:
The Ladder diagram consists of rungs, parallel paths, rung sections, contacts, boxes, coils and parallel coils.

Fig. 4/1: The rung



A **Box** can be used to replace a contact or a coil. Boxes are used for functions that are difficult or impossible to represent in the Ladder diagram or circuit diagram, for example an arithmetic operation, like the addition of two numbers.

Fig. 4/2: Adding two numbers



Important: A maximum of 5 boxes can be used in one rung.

4. The structure of a ladder diagram program

Parallel branches can be used at any point in the rung, as long as their connections do not cross. Parallel branches are called parallel coils.

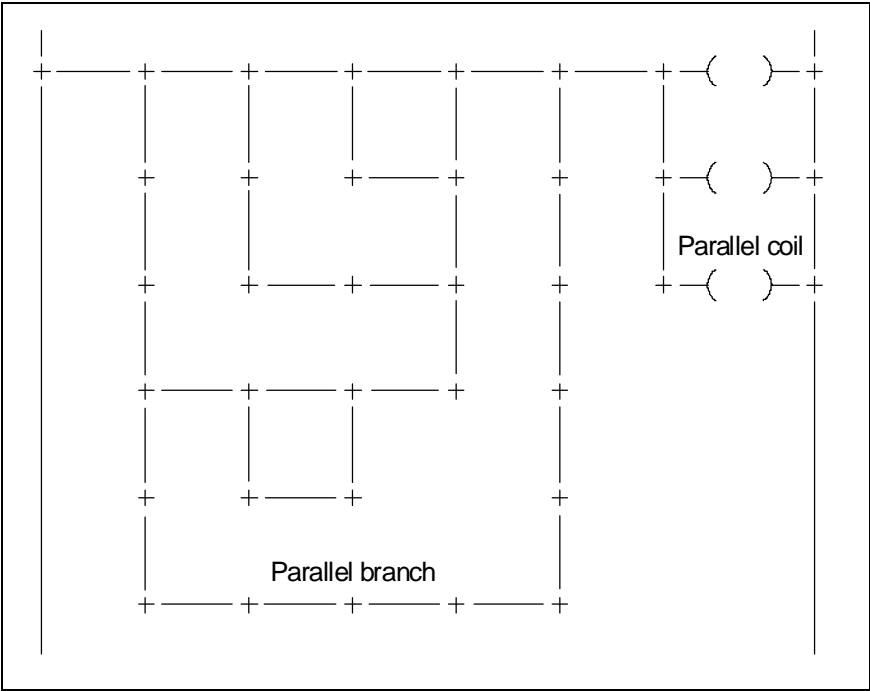


Fig. 4/3: Parallel branches and parallel coils

4. The structure of a ladder diagram program

The rungs displayed so far only contain **operations**, and therefore represent nothing more than a logical linkage of binary elements. Each operation, that is each contact, coil and box (but not each rung section) requires an operand. If the operand is missing, the FST software will reject the program as containing errors and will not load it into the controller.

The operand is written **over** the contact/coil. This is done with the aid of the function key (or for FST software version 3.1 or later with the mouse) activating the function "Enter operand".

In the case of boxes it is quite common for two, three or more operands to be entered. More on that later.

4. The structure of a ladder diagram program

4.3 CONDITIONAL AND EXECUTIVE PART

Like all Boolean or logic operations, the rung can be divided into a **conditional part** and an **executive part**. The conditional part (in rung section) contains the condition for a given action; the executive part (coils and parallel coils) contains the action to be carried out if the condition is met.

This is the same in an electrical circuit diagram. An electrical "load" represents the executive part of the current path. The contacts in series or parallel, with normally-open and normally-closed contacts, represent the conditional part.

4. The structure of a ladder diagram program

4.4 HOW IS A RUNG PROCESSED?

The rungs are numbered. The numbering is carried out automatically. For this reason there is a clear **sequence** of rungs. Before loading the ladder diagram program into the controller the program is translated into the machine code used by the controller. The rungs are translated consecutively in their number sequence. Even within a rung the machine code uses the sequence programmed by you. It follows that a second coil in a rung will also appear second in the machine code.

In the case of the FPC 405, the second coil is also switched second, with a delay measured in microseconds. In the case of the FPC 100, IPC and FEC, the second coil is written as second in the process image, and transmission to the physical outputs takes place independent of the programmed sequence.

The entire program is then executed strictly **in sequence** (as in every computer with only one processor). When powering up or starting with the run/stop switch each controller begins at the first program line. So when you are programming you always know which operation will be executed first.

4. The structure of a ladder diagram program

This is utilized in particular when applying the so-called **initialization flag**. Initialization flag FI is a special one-bit operand which has the value logic 1 for the first cycle of a program. With the aid of FI it is possible to put any operand into the desired initial status for the program (initializing operands). This method is used to reset retentive operands after powering up, but also to set particular outputs to an initial value (displays, for example, showing the operating mode). The initialization flag FI has the value logic 1 from the beginning of the program for one cycle.

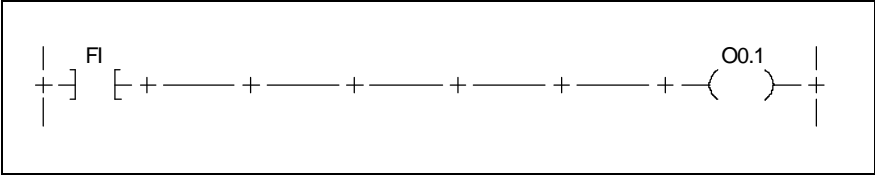


Fig. 4/4: Interrogation of FI

The negated interrogation of FI is an existing 1, which has the value logic 1 as long as the program is executed after the first cycle.

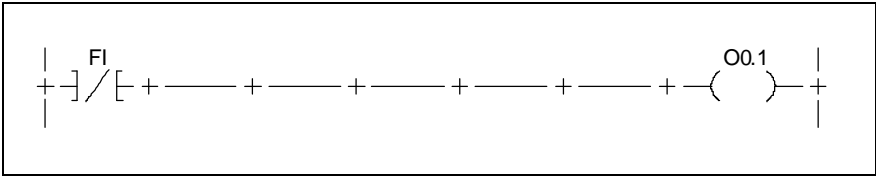


Fig. 4/5: Negated interrogation of FI

This treatment of the initialization flag FI is the same for all Festo controllers programmed in Ladder diagram.

4. The structure of a ladder diagram program

However, the internal processing of certain Festo controllers differ slightly:

The FPC 100, IPC and FEC controllers (as typical single processor system) use a **process image** for program execution. The process image represents the status of inputs and outputs. Before processing the first rung this process image is updated. During execution of the program the status of the inputs is never read in, and the status of externally visible outputs is not changed. (The process image is also updated when jumping to an earlier part of the program. It is also updated before processing the first rung of each Ladder diagram program. So, if you use a project with several programs, the process image is updated before commencing the execution of each program.)

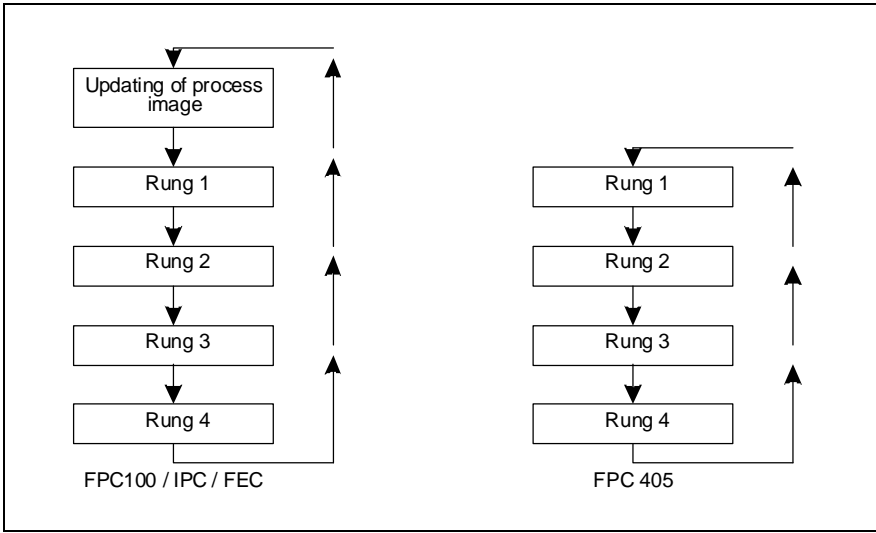


Fig. 4/6: Program execution with/without process image

4. The structure of a ladder diagram program

The FPC 405 controller (as a typical multi-processor system) does not use a process image. The status of an input is interrogated immediately following the input signal delay, when the input is interrogated in the program. An output signal is written to an output at exactly the time the appropriate command is executed in the program.

What does it mean in practice?

The following program could be written for any Festo PLC:

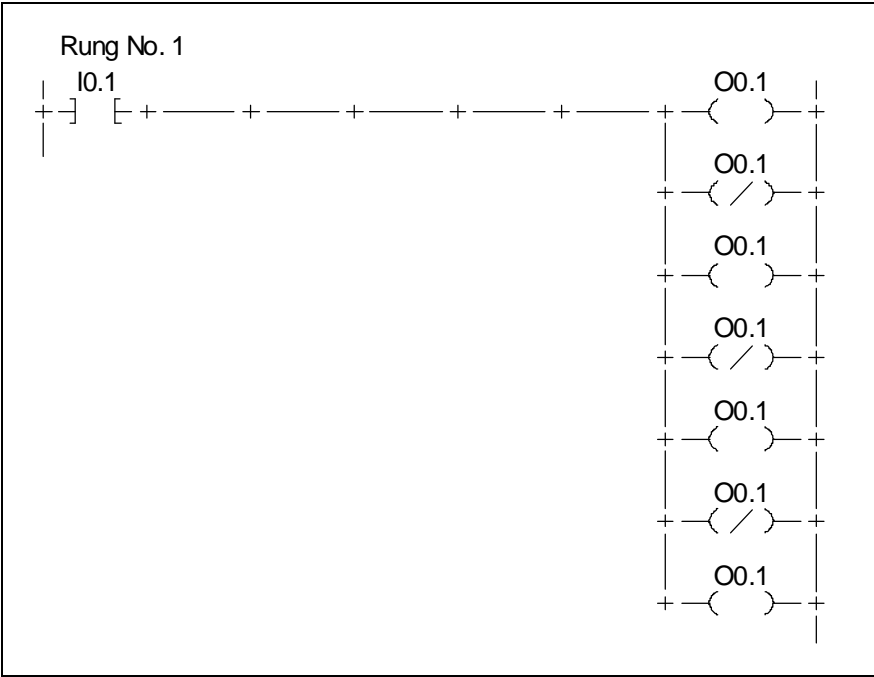


Fig. 4/7: Program

4. The structure of a ladder diagram program

The program does not make a lot of sense, as output O0.1 is simply switched on and off again continually. In the case of a controller with process image - for example the FPC 100 - the external output would not be switched on and off in this way. The output remains on if the input is logic 1 or off if the input is logic 0, because the last thing that happens in this command before updating of the process image is setting the output (if the condition is met).

On the other hand, a controller that accesses inputs and outputs directly will switch the external output on and off as programmed. This is necessary, so that in an environment with multiple processor modules, all modules have direct access to the inputs and outputs. In the case of the FPC 405, you can see the rapid flashing of the outputs using an oscilloscope. In most cases you can also see that the output light-emitting diode appears less bright.

4. The structure of a ladder diagram program

The following program can also be executed by any Festo PLC:

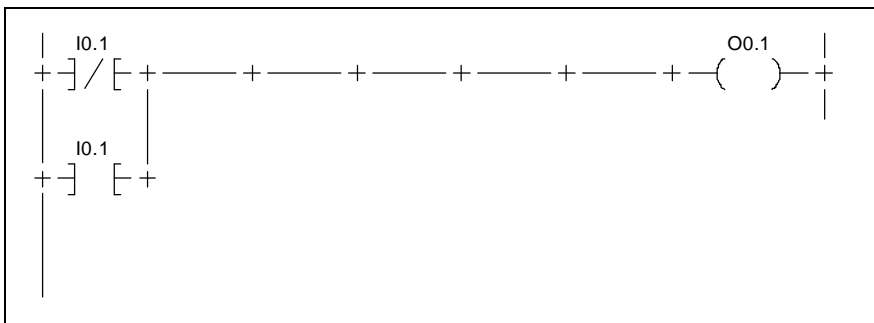


Fig. 4/8: Program

Here an input is interrogated for logic 1 and logic 0. A superfluous program, as a permanent 1 can be much more easily programmed with NOP. At first glance it appears that this condition will always be met as each input can only be logic 0 or logic 1.

However, this only applies for a controller with process image (FPC 100, IPC, FEC). Only then can you be sure that the logical status of the input cannot change between the two interrogations of the input.

In the case of the controllers with direct access to inputs (FPC 405) it is quite possible that the logical status of input I0.1 could change between the two interrogations. This does not happen very often, but in practice cannot be ignored.

4. The structure of a ladder diagram program

Advantages of the process image

On the one hand the process image method ensures that inputs cannot change their logical status during the program cycle.

Secondly, it is ensured that the outputs can only adopt the logical status last programmed. This allows safety or checking routines to be written at the end of the program to pick up programming errors. (If such check routines are absolutely necessary for controllers with direct input and output access, they are programmed with flags. Only at the end of the program are these flags written to the outputs as required by the check routines.)

And finally, program execution is much faster when a process image is used. However, when creating the process image, the cyclic time of the program must be checked.

4. The structure of a ladder diagram program

Advantages of the direct access method

Direct access makes it possible to operate several processor modules in parallel, that is to use a multiprocessor system.

Furthermore, the direct access method ensures that only the current status of an input is used. The cycle time does not need to take the time for updating the process image into account.

Finally, the rapid switching of outputs can be used for driving electronic peripherals (for example stepper motors, electronic counters, electronic displays, electronic multiplexers).

Which of the two methods is better for a specific application will depend on a number of criteria. In case of doubt it is good to know:

FPC 100, IPC and FEC are single processor systems and use the process image method.

FPC 405 is a multiprocessor system and works with direct access to inputs and outputs.

4. The structure of a ladder diagram program

4.5 HOW FAST IS A CONTROLLER?

Each contact and each coil involve time. Therefore it follows that the cycle time of a controller will increase with program length. This is the case with all modern PLCs.

In order to avoid excessively long cycle times, modern controllers offer the jump command. This allows unnecessary program parts to be skipped. This command is also available in Ladder diagram for the Festo controllers.

The cycle times of controllers vary greatly. In addition, as the time taken to process various commands can vary greatly (an arithmetic operation takes longer than simply switching an output), it is difficult to make a global statement about the cycle time of a controller. Brochure statements such as "6 milliseconds per 1000 statements" look good but have no value, as different statements require a different length of time for execution. For this reason the example programs in Appendix A contain simple programs with which you can measure the actual cycle time, the longest and shortest cycle time of the program and the reaction time (that is cycle time and input signal delay). Use one of these programs if you want to check the cycle time for a specific application.

5. Overview of operations

Chapter 5
Operations - an overview

5. Overview of operations

Contents

ARITHMETIC/LOGIC BOX	50
MODULE	50
SWAP	50
FUNCTION MODULE	50
INVERTING	50
CREATING COMPLEMENT	50
LOAD TO	50
LOGIC	51
Exclusive OR (EXOR)	51
Exclusive OR (EXOR) with 3 outputs	51
Exclusive OR (EXOR) with words	51
Identity	52
NOT (at output)	52
NOT (at input)	52
OR	53
OR with words	53
AND	53
AND with words	53
NOP	53
ARITHMETIC FUNCTIONS	
Adding	54
Dividing	54
Multiplying	54
Subtracting	54
ROTATING	
To left	54
To right	54
SHIFTING	
To left	54
To right	54
LATCHING (memory)	55

5. Overview of operations

JUMPING to a flag 55

ENTERING a jump destination 55

COILS..... 55

RESET (delete)..... 55

SET 55

Assignment..... 55

Negated assignment..... 55

COMPARATORS..... 55

Equal55

Greater than 55

Greater than or equal..... 56

Less than 56

Less than or equal 56

Unequal..... 56

NUMBERS

Conversion BCD to BINARY 56

Conversion BINARY to BCD..... 56

COUNTING

Interrogated preselect achieved? 57

Interrogated preselect not achieved? 57

Decrement..... 57

Increment..... 57

Set counter 57

TIME

Start switch-off delay 57

Switch-off delay time expired? 57

Switch-off delay time running?..... 58

Start switch-on delay..... 58

Switch-on delay time expired?..... 58

Switch-on delay time running? 58

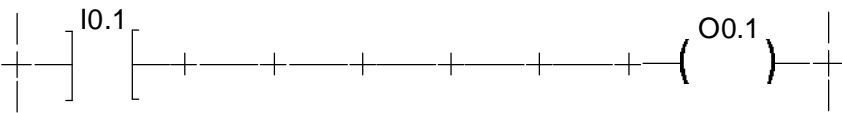
Start pulse timer 58

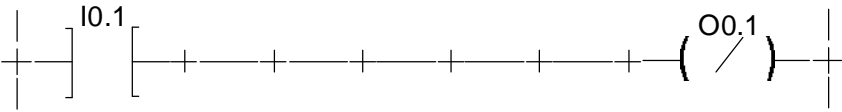
Pulse timer running? 58

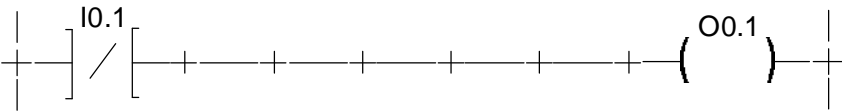
5. Overview of operations

Operation	Symbol	Function
Arithmetic/ logic box		This box allows arithmetic or logic word operations to be carried out.
Module		Call user modules or standard modules
Swap		Swap the high and low byte of the upper word (here FW4) and write them to the lower word (here R3).
Function module		Call Function module supplied by Festo.
Invert		Invert every bit of the upper word (here R2) individually and write the result to the lower word (here R2).
Complement		Write the two's complement of the upper word (here R2) (i.e. change sign).
LOAD TO		Load the value from the upper word (here 12) into the lower word (here CW4).

5. Overview of operations

Operation	Function
Logic: Identity	Direct transfer of the logic status of a bit to the output bit
	

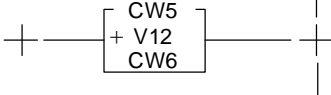
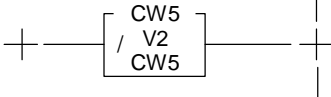
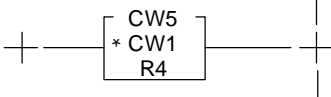
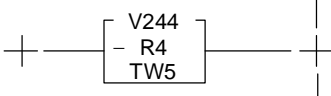
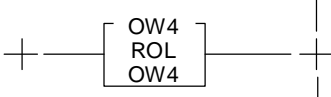
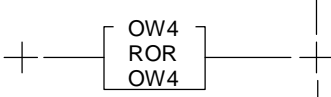
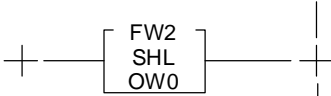
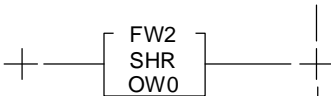
Operation	Function
Logic: NOT (at output)	Negation of the output signals
	

Operation	Function
Logic: NOT (at input)	Negation of the input signals
	

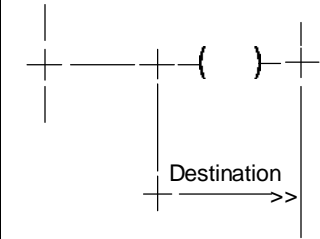
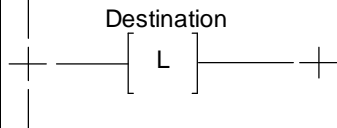
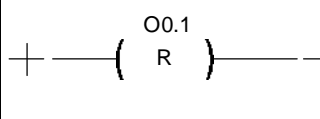
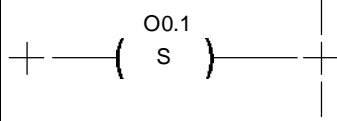
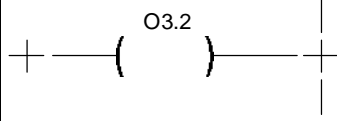
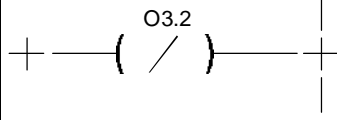
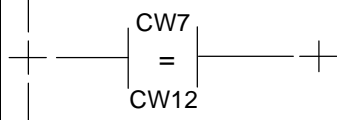
5. Overview of operations

Operation	Symbol	Function
Logic: OR		OR connection of binary signals
Logic: OR with words		Bitwise OR connection of two words
Logic: AND		Logical AND connection of binary signals
Logic: AND with words		Logical AND connection of the individual bits of two words
NOP		Non operation Produces permanent 1. The condition NOP is always fulfilled.

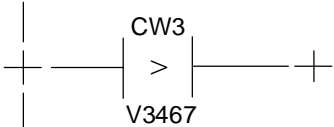
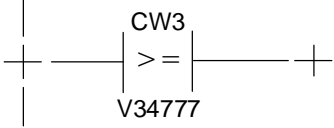
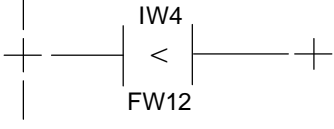
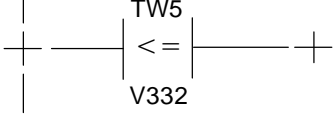
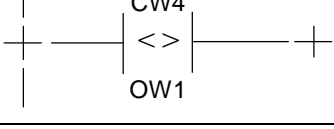
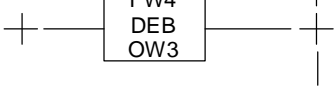
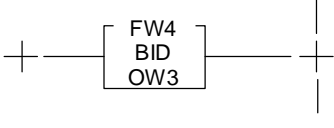
5. Overview of operations

Operation	Symbol	Function
Arithmetic: Addition		The addition of two whole numbers and storing the result
Arithmetic: Division		Dividing Ignoring places after decimal point
Arithmetic: Multiplication		Multiplies two whole numbers (places beyond decimal point are ignored)
Arithmetic: Subtraction		Subtract two whole numbers
Rotation: left		Rotates all word bits to the left
Rotation: right		Rotates all word bits to the right
Shift: left		Shifts the bits of a word one place to the left
Shift: right		Shifts the bits of a word one place to the right

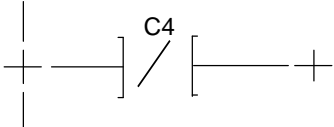
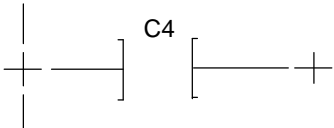
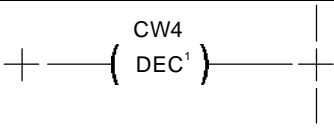
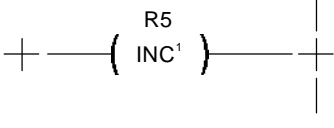
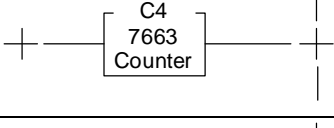
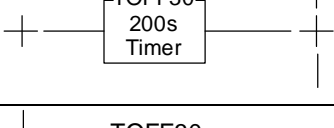
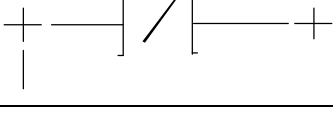
5. Overview of operations

Operation	Symbol	Function
Jump to label		Jump to label clearly indicated at some point in the program (here the label is DESTINATION).
Jump destination		Enter label that can be jumped to
Coils: RESET		Resetting, i.e. erasing a stored bit
Coils: SET		Setting a stored bit
Coils: Assignment		The output bit must always have the same logical status as the result of the previous condition.
Coils: Assignment negated		The output bit must always have the negated logical status of the result of the previous condition.
Comparison: Equal		The rung signal is 1, if both words are identical.

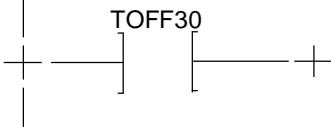
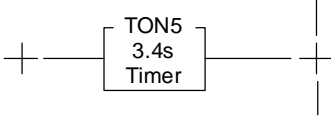
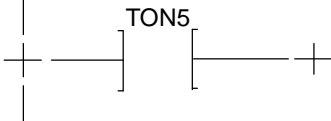
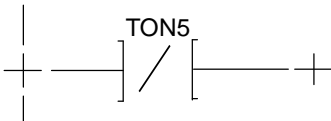
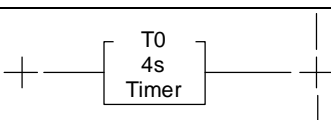
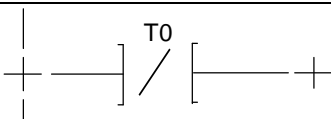
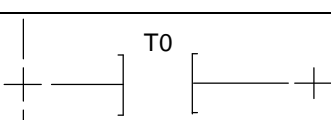
5. Overview of operations

Operation	Symbol	Function
Comparison: Greater		The rung carries a logic 1 if the upper word (here CW3) is greater than the lower word (here the value 3467).
Comparison: Greater/ equal		The rung carries a logic 1 if the upper word (here CW3) is greater than or equal to the lower word (here the value 34777).
Comparison: Less		The rung carries a logic 1 if the upper word (here IW4) is less than the lower word (here FW12).
Comparison: Less/ equal		The rung carries a logic 1 if the upper word (here TW5) is equal to or less than the lower word (here the value 332).
Comparison: Unequal		The rung carries a logic 1 if the two words (here CW4 and OW1) are unequal.
Numbers: CONVERT BCD TO BINARY		The upper word (here FW4) is interpreted as a decimal number and converted into binary. The result is written to the lower operand (here OW3).
Numbers: CONVERT BINARY TO BCD		The upper word (here FW4) is interpreted as binary and converted into BCD-code. The result is written into the lower operand (here OW3).

5. Overview of operations

Operation	Symbol	Function
Counting: Preselect interrogation achieved?		Rung is logic 1, if the counter (here No. 4) has reached or exceeded the preselect or has not yet been started.
Counting: Preselect interrogation not achieved?		Rung is logic 1, if the counter (here No. 4) has not reached the preselect.
Counting: Decrement		The upper word (here CW4) is decremented by 1.
Counting: Increment		The upper word (here R5) is incremented by 1.
Counting: Set counter		The counter preselect (here for No. 4) is set to the value indicated (here 7663), and the counter bit (here C4) becomes logic 1.
Time: Start switch-off delay		The timer (here No. 30) is set as a delay timer with the duration indicated (here 200 seconds).
Time: Switch-off delay Time expired?		Rung is logic 1, if the preselect time has expired.

5. Overview of operations

Operation	Symbol	Function
Time: Switch-off delay Time running?		The rung is logic 1, if the preselect time has not yet expired.
Time: Start switch-on delay		The timer (here No. 5) is set as a switch-on delay with the duration indicated (here 3.4 seconds).
Time: Switch-on delay Time expired?		Rung is logic 1, if the preselect time has expired.
Time: Switch-on delay Time running		Rung is logic 1, if the preselect time has not yet expired.
Time: Start pulse timer		The timer (here No. 0) is started as a pulse timer with the preselect indicated (here 4 seconds).
Time: Pulse timer Time expired?		Rung is logic 1, if the preselect time has expired or the timer has not been started.
Time: Pulse timer Time running		Rung is logic 1, if the preselect time has not yet expired.

6. Operations in detail

Chapter 6
The operations in detail
Examples for reference

6. Operations in detail

Contents

ARITHMETIC/LOGIC BOX	62
MODULE	65
SWAP	67
FUNCTION MODULE	69
INVERTING	71
CREATING COMPLEMENT	73
LOAD TO	75
LOGIC	
Exclusive OR (EXOR)	76
Exclusive OR (EXOR) with 3 outputs	77
Exclusive OR (EXOR) with words	79
Identity	81
NOT (at output)	82
NOT (at input)	83
OR	84
OR with words	85
AND	87
AND with words	88
NOP	90
ARITHMETIC FUNCTIONS	
Adding	91
Dividing	93
Multiplying	95
Subtracting	97
ROTATING	
To left	99
To right	101
SHIFTING	
To left	103
To right	105
LATCHING (memory)	107
	60

6. Operations in detail

JUMPING to a flag 110

ENTERING a jump destination 111

COILS

RESET (delete)..... 113

SET 114

Assignment..... 115

Negated assignment..... 116

COMPARATORS

Equal 117

Greater than 119

Greater than or equal..... 121

Less than 123

Less than or equal 125

Unequal..... 127

NUMBERS

Conversion BCD to BINARY 129

Conversion BINARY to BCD..... 131

COUNTING

Interrogated preselect achieved? 133

Interrogated preselect not achieved? 135

Decrement..... 137

Increment..... 138

Set counter 139

TIME

Start pulse timer 141

Interrogate pulse timer 142

6. Operations in detail

The Arithmetic/Logic Box

The Arithmetic/Logic Box is only used in the executive part. It allows coherent programming of word operations in contrast to the multi-bit boxes which only allow one command each.

The Arithmetic/Logic Box is indicated in the rung as a box. In the printout the contents of the box are printed immediately following the rung containing the Arithmetic/Logic Box.

During programming you can examine the contents of the box by positioning the cursor on the box and pressing the <Enter> key. The contents of the box are then displayed on the screen. You can cancel the display of contents by pressing the <ESC> or <Enter> key.

Independent of the number of command lines within the box, the Arithmetic/Logic Box is only regarded as a **single** box. **Up to 16 lines** can be entered into the Arithmetic/Logic Box. As only 5 boxes can be used in a rung, only 4 other boxes can be used in addition to an Arithmetic/Logic Box.

Within the Arithmetic/Logic Box, multi-bit commands are entered as text and therefore not always the same as the commands otherwise entered in Ladder diagram boxes. In addition, there is a space for comments below the Arithmetic/Logic Box.

6. Operations in detail

Each field of the Arithmetic/Logic Box may only contain **one** entry.

Example:

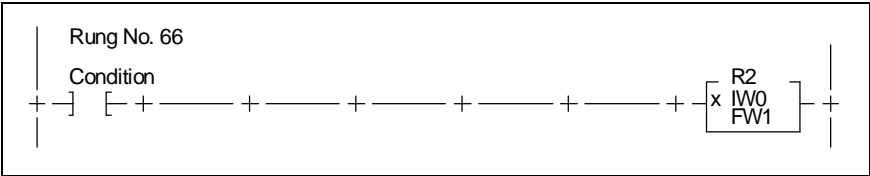


Fig. 6/1: Logic, EXOR using words

The same operation in an Arithmetic/Logic Box has the following structure:

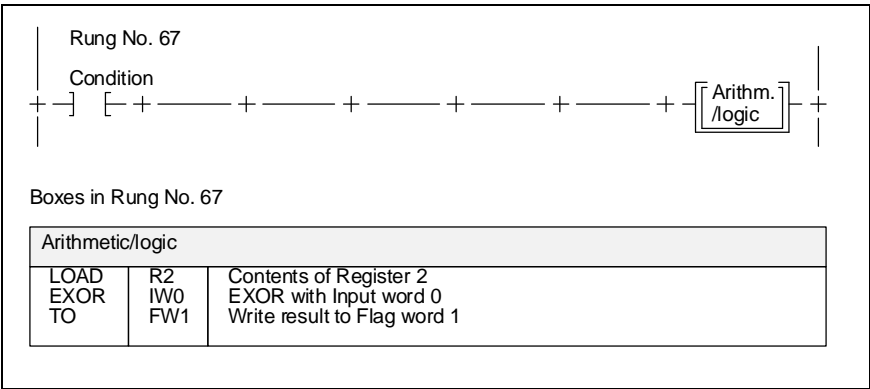


Fig. 6/2: Logic, EXOR with words in arithmetic box

6. Operations in detail

Figures 6/1 and 6/2 show an EXOR operation on two words. 6/1 in a multi-bit box and 6/2 in the Arithmetic/Logic Box. (For details of this operation see "Exclusive OR with words".) In the Arithmetic/Logic Box the multi-bit operation begins with the LOAD command. This indicates that the entered multi-bit operand is loaded into the processor module's multi-bit accumulator. The required operation is carried out in this multi-bit accumulator. The result is loaded from the multi-bit accumulator into any other multi-bit operand of your choice (command: TO). In the multi-bit box, EXOR is indicated by an x. In the Arithmetic/Logic Box EXOR is indicated by EXOR.

The result of the operation in the multi-bit box and in the Arithmetic/Logic Box is completely identical.

The Arithmetic/Logic Box is better when you want to use several multi-bit operations in a single rung, and/or when you want to enter comments.

6. Operations in detail

MODULE

Modules (also called program modules) are MODULES created by the user that can be called up at any point in a Ladder diagram program. Modules must be clearly identified as a module when calling them from the Ladder diagram editor.

These modules or program MODULES are nothing more than subroutines. Subroutines allow repetitive procedures in a program to be written once but called up many times.

The individual controllers support the following number of modules:

- **FPC 100B/AF:** 8 program modules
- **IPC:** 100 program modules
- **FEC:** 100 program modules
- **FPC 405:** 32 program modules (The program modules for the FPC 405 may only be written in assembler if they are to be called from the Ladder diagram program.)

For example, a text display may be connected to a controller. The transmission of data to this text display can be programmed by means of a program module.

6. Operations in detail

The contents of a text display need to be updated from various parts of the program: from the automatic program as well as from the manual program. An error message changes the text, as does an operator entry.

At this point in your program you call the MODULE that handles data transmission to the text display.

This makes the program shorter and easier to read.

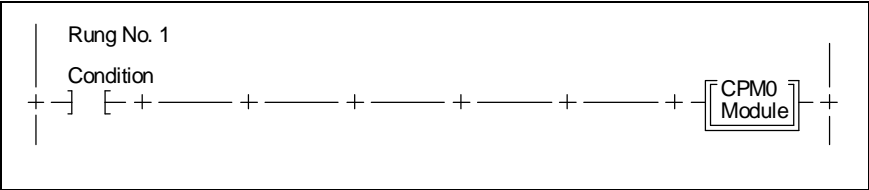


Fig. 6/3: Module

6. Operations in detail

SWAP

All operands in the Festo controllers are structured as 16-bit words. Each word consists of two 8-bit bytes, the low byte (right) and the high byte (left).

Sometimes it may be necessary to swap the two bytes of a word. This can be done with the Ladder diagram command SWAP.

The command SWAP changes:

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

to:

0	1	1	0	0	1	1	1	0	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

In Ladder diagram SWAP is programmed with a box:

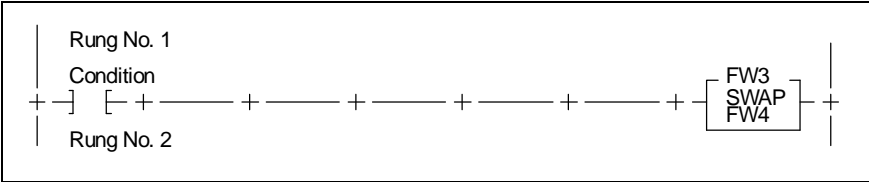


Fig. 6/4: SWAP bytes

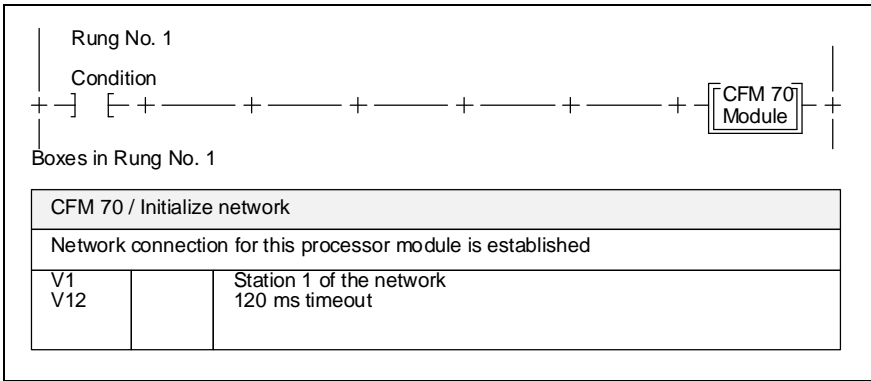
6. Operations in detail

FUNCTION MODULE

Function modules – as opposed to program modules – are sub-programs pre-programmed into the controller by Festo. So these are sub-routines in the accepted sense, or to be more precise procedures, as generally used in higher programming languages. These sub-routines are supplied with the controller or can be ordered separately from Festo.

One example of a function module (CFM) is for opening a network for the FPC 405. Function module CFM 70 is used for this purpose. The station number of the calling processor module and the desired timeout are passed to CFM 70 as parameters. CFM 70 leaves the message in function unit 32 indicating whether network setup was successful or not.

Fig. 6/6: Function module CFM 70



6. Operations in detail

Another typical function module is CFM 0 for the FPC 100B/AF. This function module facilitates deletion of all retentive operands, as required for many applications.

The following application is typical:

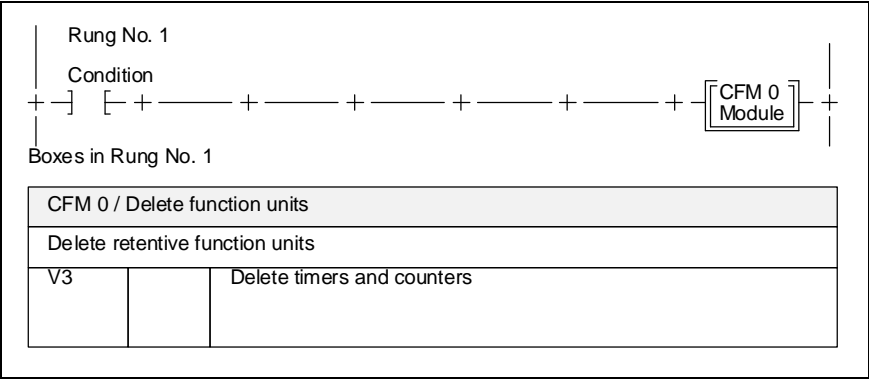


Fig. 6/7: Function module CFM 0

The function modules offered depend on the controller type. For details please see the system manual for the appropriate controller type.

6. Operations in detail

INVERTING

INVERT is an operation for use with multi-bit operands. INVERT negates each bit of a word.

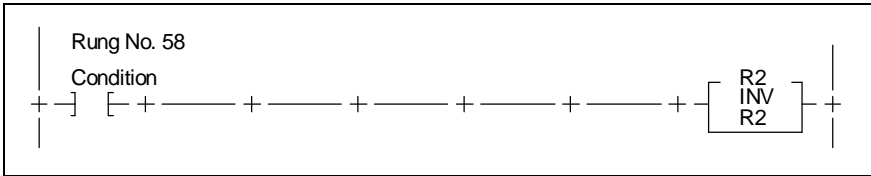


Fig. 6/8: Inverting

The same result can be achieved by using INVERT in the Arithmetic/Logic Box:

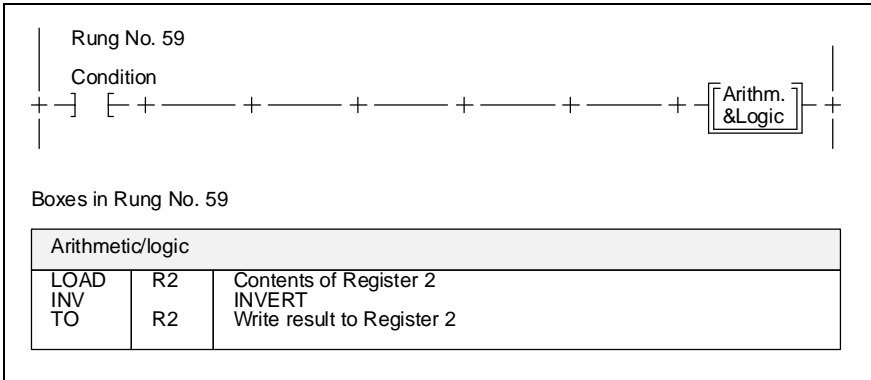


Fig. 6/9: Inverting in arithmetic box

In this example register 2 is inverted and the result is written back to register 2.

6. Operations in detail

30311₁₀, 30311₁₀ or 7667₁₆

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1

becomes -30312₁₀, 35224₁₀ or 8998₁₆.

Any multi-bit operand can be entered as the operand to be inverted (the upper operand in the box). The result can be returned to any multi-bit operand (the lower operand in the box) **with the exception of input words**.

If the operand at the top is not the same as the operand at the bottom, the contents of the upper operand are not changed.

6. Operations in detail

COMPLEMENT

COMPLEMENT is an operation for use with multi-bit operands.

The COMPLEMENT operation is used to change the sign of a number. The value -5 becomes 5, the value 2734 becomes -2734. The COMPLEMENT operation only makes sense if you are working with signed numbers.

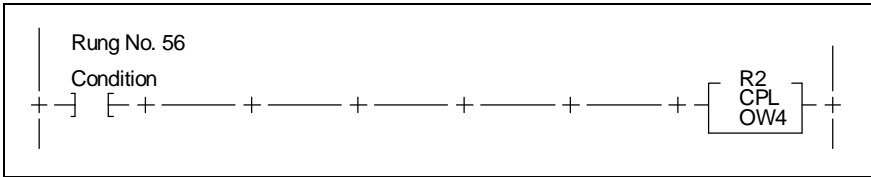
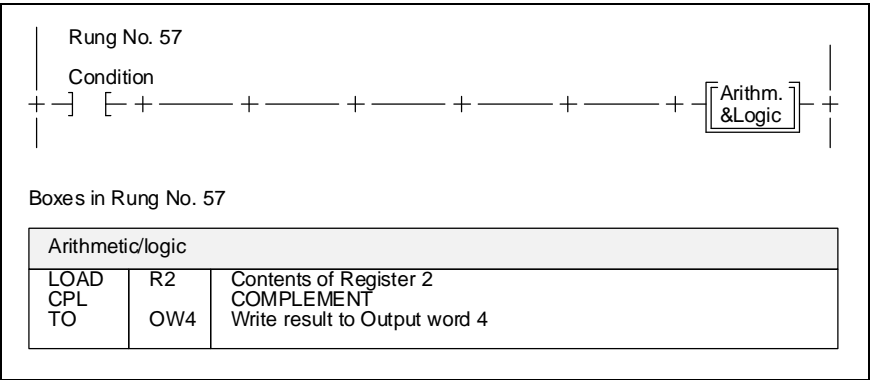


Fig. 6/10: Creating the complement

The same function is programmed in the



Arithmetic/Logic Box as follows:

Fig. 6/11: Creating the complement in arithmetic box

6. Operations in detail

In this example the COMPLEMENT of register 2 is written to output word 4 (register 2 remains unchanged).

30311₁₀, 30311₁₀ or 7667₁₆

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0

becomes -30311₁₀, 35225₁₀ or 8999₁₆.

The COMPLEMENT operation can be used for all multi-bit operands. The destination (lower operand) can be any multi-bit operand **with the exception of an input word**. If the upper operand (source) and lower operand (destination) are different, the upper operand is left unchanged.

6. Operations in detail

LOAD TO

LOAD TO is an operation for use with multi-bit operands.

LOAD TO allows the contents of a multi-bit operand to be loaded into another multi-bit operand. The source can be any multi-bit operand, the destination can also be any multi-bit operand **with the exception of an input word**.

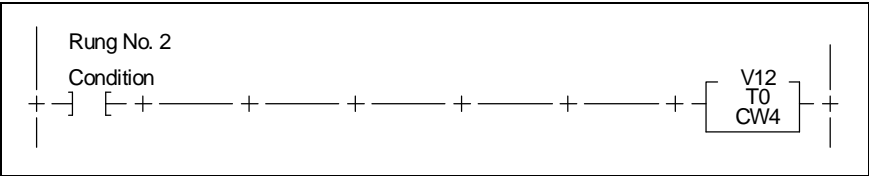
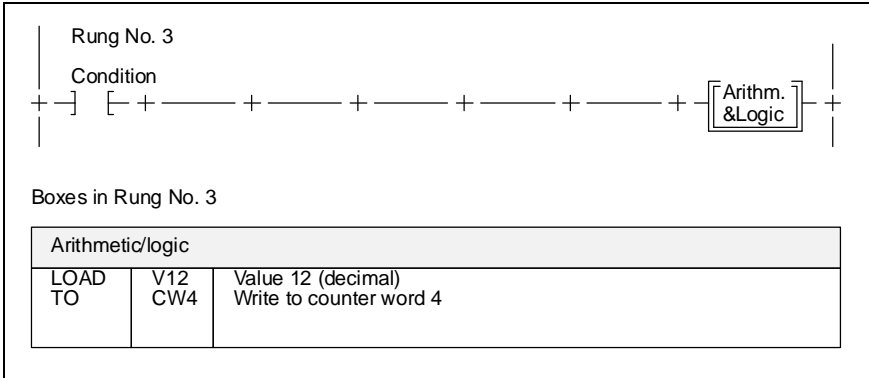


Fig. 6/12: Load to

The identical function is programmed using the Arithmetic/Logic Box as follows:

Fig. 6/13: Load to (Arithmetic/Logic Box)



In this example the decimal value 12 is transferred to counter word 4.

6. Operations in detail

LOGIC: EXCLUSIVE OR (EXOR)

EXOR is one of the common logic operations. The logic module is shown thus:

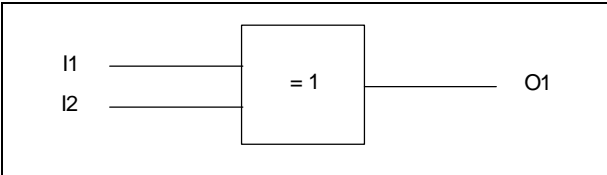


Fig. 6/14: Logic module

The truth table for EXOR is as follows:

I1	I2	O1
0	0	0
0	1	1
1	0	1
1	1	0

Ladder diagram does not offer an independent operation for EXOR. The EXOR must be constructed using a combination of parallel and series circuits and negations.

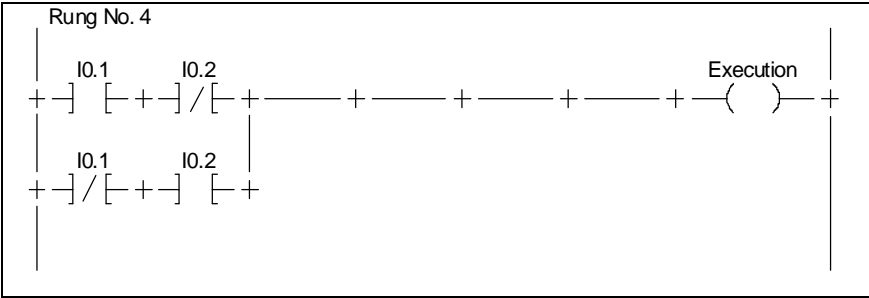


Fig. 6/15: LOGIC: Exclusive OR

6. Operations in detail

LOGIC: EXCLUSIVE OR (EXOR) with 3 inputs

There are different views on the truth table for an EXOR with three inputs. In many cases – particularly when programming with statement list – the EXOR can only be used with a maximum of two inputs.

Here the EXOR with three inputs is shown for the sake of completeness and to demonstrate how you can expand the logical function of EXOR (and any other logical function) to meet your requirements.

The logic element for EXOR with three inputs is shown thus:

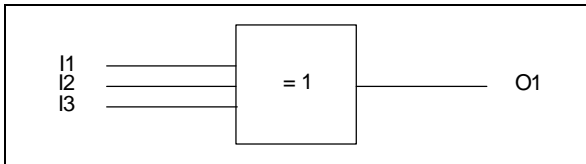


Fig. 6/16: Logic module

The truth table can also be expanded:

I1	I2	I3	O1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

6. Operations in detail

In Ladder diagram, this type of EXOR is programmed as follows:

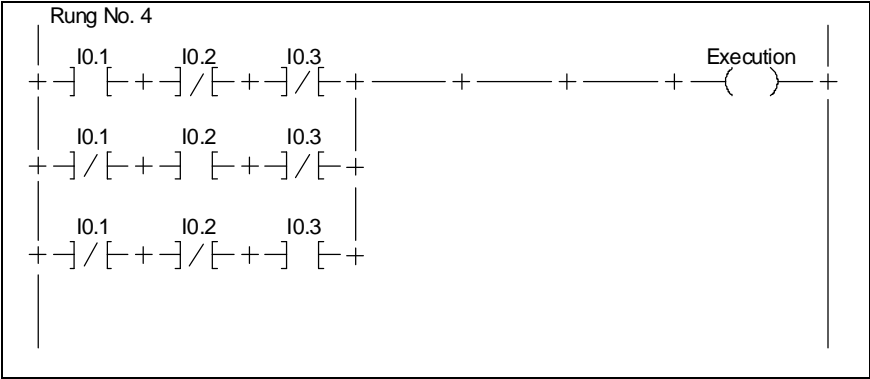


Fig. 6/17: LOGIC: Exclusive OR

6. Operations in detail

LOGIC: EXCLUSIVE OR (EXOR) with words

The EXOR that can be used with multi-bit operands is used in the executive part of the Ladder diagram. Thus, this EXOR does not represent a condition for an action, but is only called if the condition is met (the rung evaluated to logic 1).

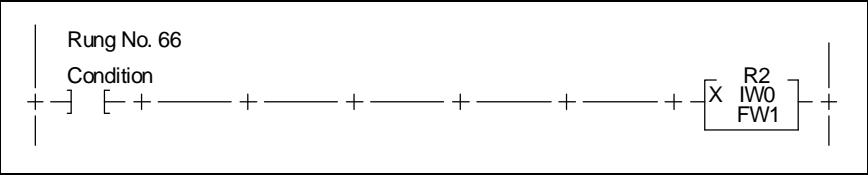


Fig. 6/18: LOGIC: Exclusive OR using words

This operation can be programmed identically with the Arithmetic/Logic Box as follows:

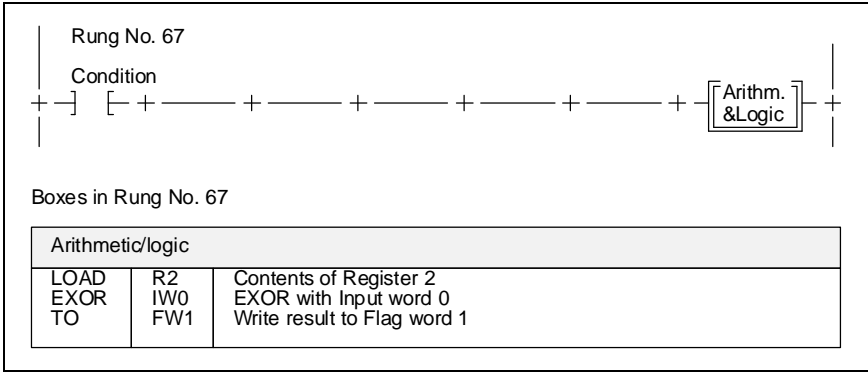


Fig. 6/19: OR with words in arithmetic box

6. Operations in detail

In this example an EXOR operation is carried out on each of the bits from register 2 with the corresponding bits of input word 0. This is carried out 16 times, each time with two inputs. The result is written to flag word 1, so that register 2 and input word 0 remain unchanged.

Register 2:

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

EXOR input word 0

1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

becomes:

1	0	0	1	1	1	1	0	1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

and is written to flag word 1.

The EXOR can be used for any multi-bit operands and the result can be written to any multi-bit operands **with the exception of an input word**.

6. Operations in detail

LOGIC: IDENTITY¹

The IDENTITY (yes function) is one of the basic logic operations. It has one input and one output (like negation).

The logic symbol for IDENTITY is shown thus:

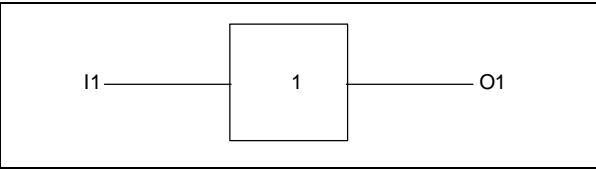


Fig. 6/20: Logic symbol for IDENTITY

The truth table for IDENTITY is as follows:

I1	O1
0	0
1	1

IDENTITY is programmed in Ladder diagram as follows:

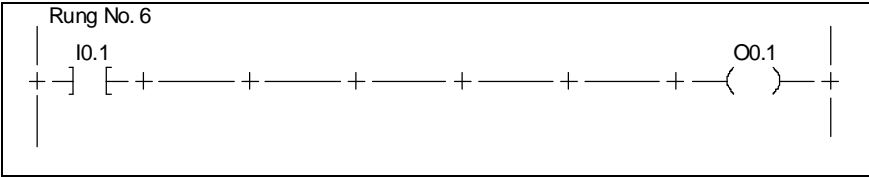


Fig. 6/21: LOGIC: IDENTITY

1. Logic: Identity with words, see under: LOAD TO

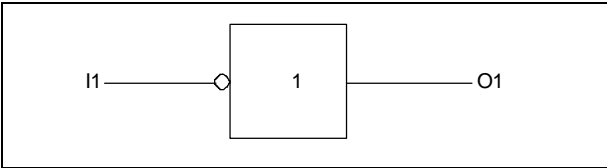
6. Operations in detail

LOGIC: NOT¹ (at input)

NOT or negation is the inverse of identity. The logic signal of an input is negated.

In Ladder diagram this negation can be used for the input or the output. For this reason the two methods are described separately here. The logic symbol for a NOT function (at input) is shown thus:

Fig. 6/24: Logic symbol for NOT function

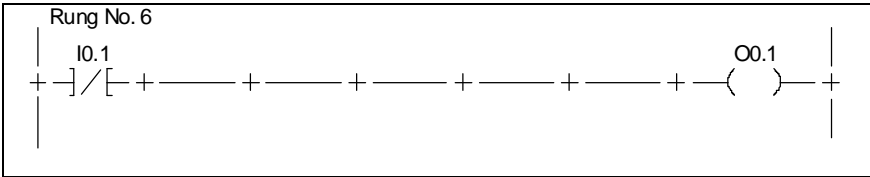


The truth table for the NOT function is as follows:

I1	O1
0	1
1	0

The NOT function is programmed in Ladder diagram as follows:

Fig. 6/25: LOGIC: NOT (at input)



The slant (oblique stroke) in the contact symbol indicates negation.

1. Logic: NOT with words, see under: INVERT

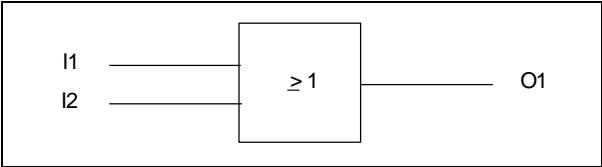
6. Operations in detail

LOGIC: OR

OR is one of the basic logic operations (like AND and NOT).

The logic symbol for OR is shown thus:

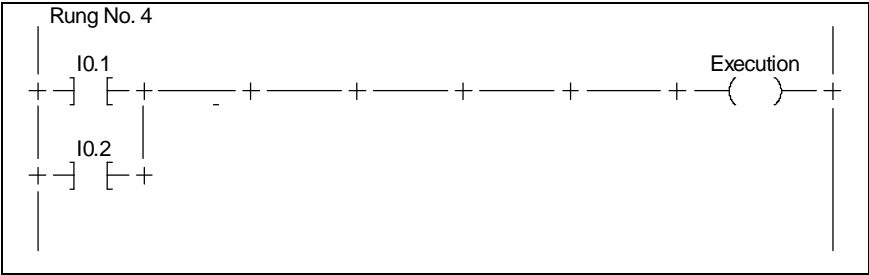
Fig. 6/26: Logic symbol OR



The truth table for the OR function is as follows:

I1	I2	O1
0	0	0
0	1	1
1	0	1
1	1	1

In Ladder diagram OR is represented by a



parallel branch:

Fig. 6/27: LOGIC: OR

6. Operations in detail

LOGIC: OR with words

The OR operation can be used for multi-bit operands. In this case it is not a condition for the command in the executive part.

When carrying out an OR operation using words, we speak of "masking". If for example you want the fifth bit of a word (counting from the least significant bit) to have the value logic 1 you OR this word with another word in which only the fifth bit has the value logic 1. The resulting word contains all the bits of the output word and guarantees that the fifth bit has the value 1.

Register 2:

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
v	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v

OR value 16

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

becomes:

0	1	1	1	0	1	1	0	0	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

and is written to flag word 1.

6. Operations in detail

In Ladder diagram you can OR-mask in the multi-bit box or in the Arithmetic/Logic Box:

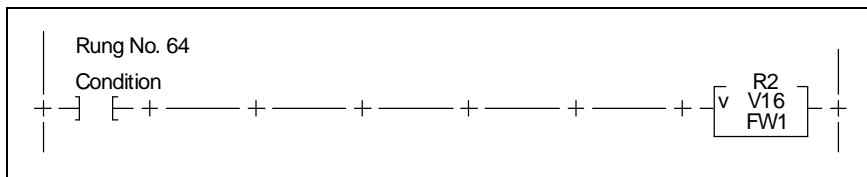


Fig. 6/28: LOGIC: OR using words

The same function can be programmed with the Arithmetic/Logic Box:

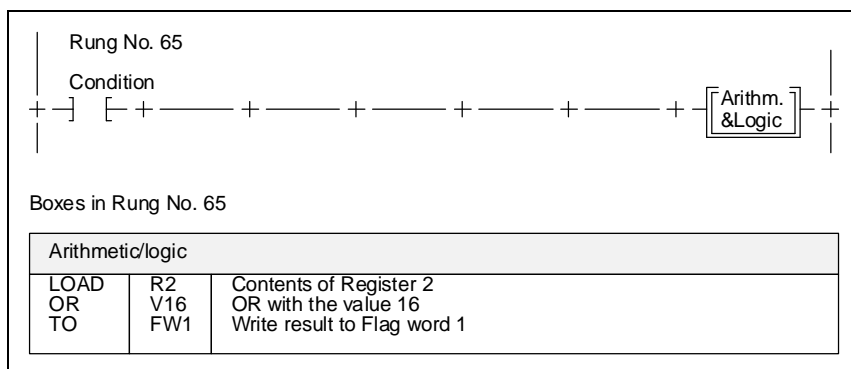


Fig. 6/29: LOGIC: OR using words in arithmetic box

Masking does not have the same significance for programmable logic controllers as it does in other areas of computing. You could simply set the fifth bit of the word by loading the word into a flag or output word. There you could change the fifth bit to logic 1 using the set command and, if required, load the word back.

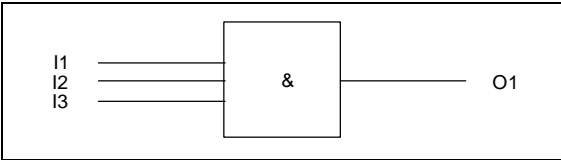
You can OR any two multi-bit operands. The destination for the result can be any multi-bit operand **with the exception of an input word**.

6. Operations in detail

LOGIC: AND

AND belongs to the basic logic operations (like OR and NOT).

The logic symbol for AND is as follows (shown



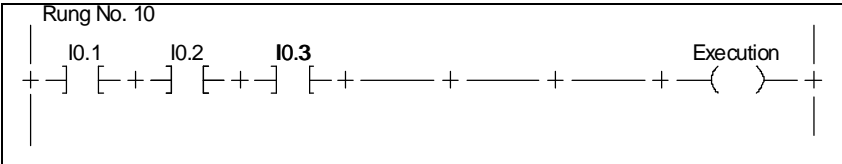
here with three inputs):

Fig. 6/30: Logic symbol AND

The truth table for AND is as follows:

I1	I2	I3	O1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

In Ladder diagram AND is represented by se-



ries connection:

Fig. 6/31: LOGIC: AND

6. Operations in detail

LOGIC: AND with words

The AND operation can also be used for multi-bit operands. In this case it is not a condition but a command in the executive part.

When carrying out an AND operation with words, we speak of "masking". For example, if you want to establish whether the fifth bit of a word (counting from the least significant bit) has the value logic 1, you AND this word with another word in which only the fifth bit has the value logic 1. If the result is 0, the fifth bit in the word you are testing is 0. If the result is greater than 0, the fifth bit has the value 1.

Register 2:

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^

AND value 16

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

becomes:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

and is written to flag word 1.

6. Operations in detail

In Ladder diagram you can AND in the multi-bit box or in the Arithmetic/Logic Box:

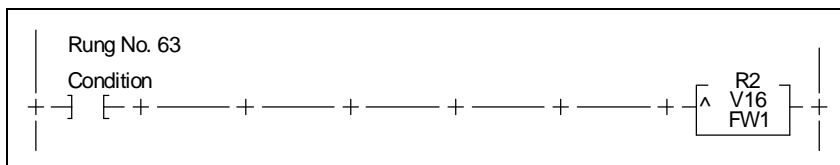


Fig. 6/32: LOGIC: AND with words

The same function can be programmed with the Arithmetic/Logic Box:

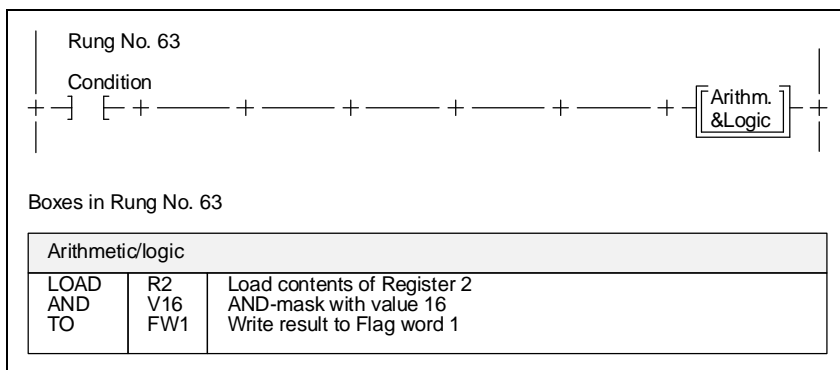


Fig. 6/33: LOGIC: AND with words in arithmetic box

Masking does not have the same significance for programmable logic controllers as it does in other areas of computing. You could simply interrogate the fifth bit of a word by loading the word into a flag or output word. From a flag or output word you can interrogate the fifth bit in binary as Fx.4 or Ox.4. You can AND any multi-bit operands. The destination for the result can be any multi-bit operand **with the exception of an input word**.

6. Operations in detail

NOP

NOP is the **null operation**, that is, an operation that "does nothing". All programming languages have null operations. In Ladder diagram, NOP can be used as an operand designation or contact but not for coils. If NOP is used, this contact is **always** switched through, which has the same effect as the negated interrogation of the initialization flag FI after the first cycle of the program.

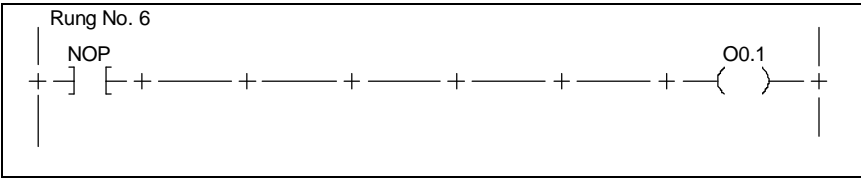
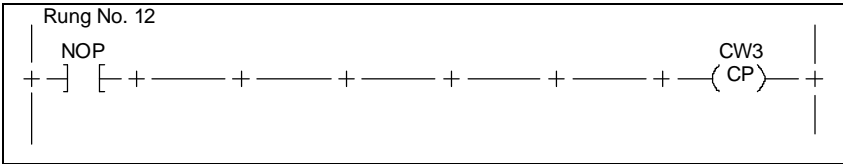


Fig. 6/34: NOP

In this example, output O0.1 is **always** switched through.

If NOP is used, this means that the rung is always active and that signal-edge recognition is switched off. In the following example counting takes place for **each cycle**, not only for the 0-



1 signal edge of the condition:

Fig. 6/35: NOP

6. Operations in detail

Arithmetic function: ADDING

The four basic arithmetic functions ADDING, subtracting, multiplying and dividing are available for all Festo controllers. These arithmetic functions can also be used in Ladder diagram.

In the Festo controller, these arithmetic functions are always used for **whole, signed numbers** in the range -32767.

The overflow cannot be interrogated.

If you use these numbers outside this range, no error message is given. Overflow is simply truncated (not rounded).

The FPC 405 also allows the use of function module 20 to program arithmetic operations with an overflow that can be interrogated.

ADDITION can be programmed as follows:

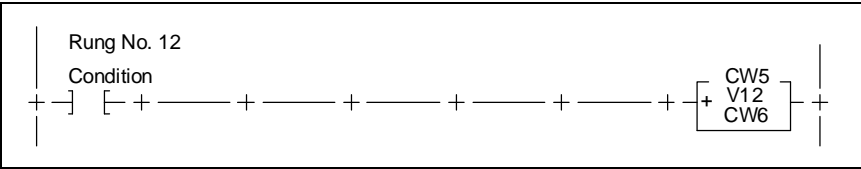


Fig. 6/36: Arithmetic function: Adding in the arithmetic box

6. Operations in detail

The same operation with the same function can be programmed with the Arithmetic/Logic Box as follows:

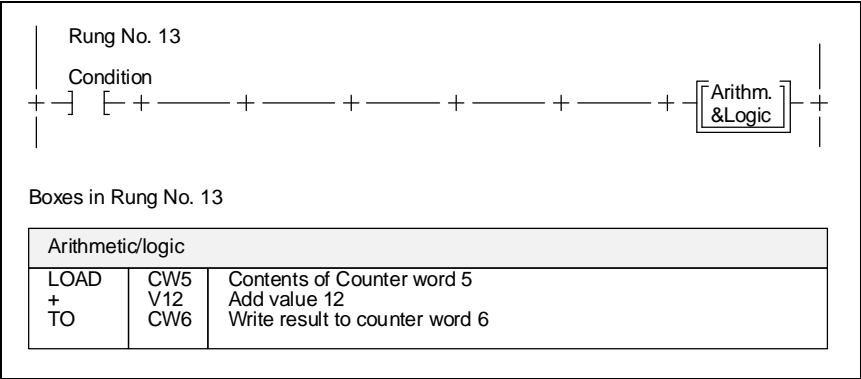


Fig. 6/37: Arithmetic: Adding in the arithmetic box

Please note that you always need at least three operands for an arithmetic operation. The first two operands are added and the result is written to the third operand. The contents of the first operands (or other operands included with arithmetic functions) are not changed, if neither of these operands is given in the result field. Any multi-bit operand can be used as a source; any multi-bit operand **with the exception of an input word** may be used as a destination.

6. Operations in detail

Arithmetic function: DIVIDING

The four basic arithmetic functions adding, subtracting, multiplying and DIVIDING are available for all Festo controllers. These arithmetic functions can also be used in Ladder diagram.

In the Festo controller, these arithmetic functions are always used for **whole, signed numbers** in the range -32767.

The overflow cannot be interrogated.

If you use these numbers outside this range, no error message is given. Overflow is simply truncated (not rounded).

The FPC 405 also allows the use of function module 20 to program arithmetic operations with an overflow that can be interrogated.

DIVISION can be programmed as follows:

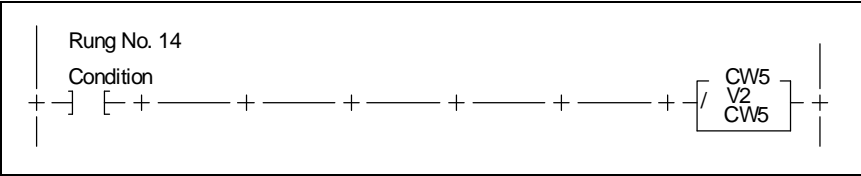


Fig. 6/38: Arithmetic function: Dividing

6. Operations in detail

The same operation with the same function can be programmed with the Arithmetic/Logic Box as follows:

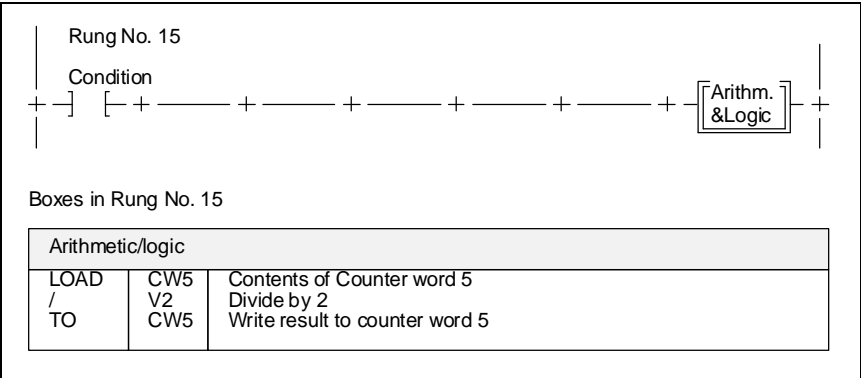


Fig. 6/39: Arithmetic function: Dividing in the arithmetic box

Please note that you always need at least three operands for an arithmetic operation. The first two operands are divided and the result is written to the third operand. The contents of the first operands (or other operands included with arithmetic functions) are not changed, if neither of these operands is given in the result field. Any multi-bit operand can be used as a source; any multi-bit operand **with the exception of an input word** may be used as a destination.

6. Operations in detail

Arithmetic function: MULTIPLYING

The four basic arithmetic functions adding, subtracting, MULTIPLYING and dividing are available for all Festo controllers. These arithmetic functions can also be used in Ladder diagram.

In the Festo controller, these arithmetic functions are always used for **whole, signed numbers** in the range -32767.

The overflow cannot be interrogated.

If you use these numbers outside this range, no error message is given. Overflow is simply truncated (not rounded).

The FPC 405 also allows the use of function module 20 to program arithmetic operations with an overflow that can be interrogated.

MULTIPLYING can be programmed as follows:

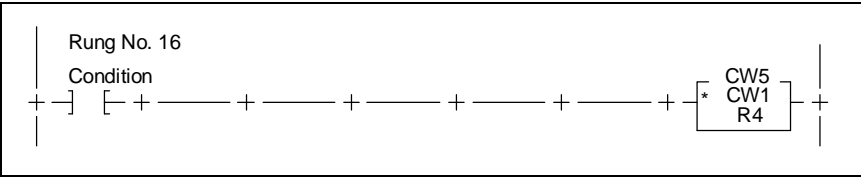


Fig. 6/40: Arithmetic function: Multiplying

6. Operations in detail

The same operation with the same function can be programmed with the Arithmetic/Logic Box as follows:

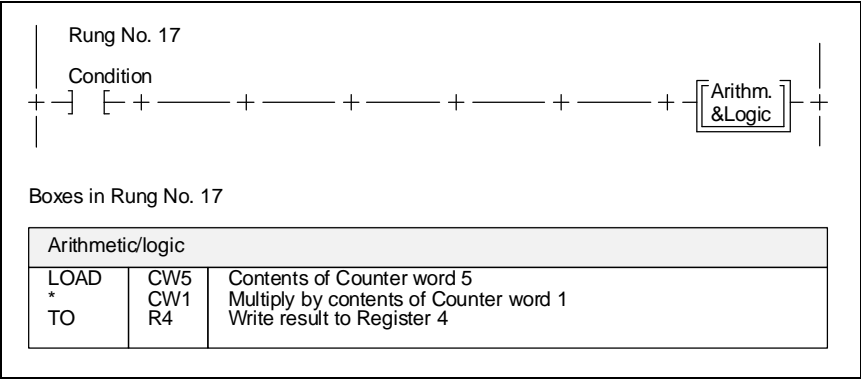


Fig. 6/41: Arithmetic function: Multiplying in the arithmetic box

Please note that you always need at least three operands for an arithmetic operation. The first two operands are multiplied and the result is written to the third operand. The contents of the first operands (or other operands included with arithmetic functions) are not changed, if neither of these operands is given in the result field. Any multi-bit operand can be used as a source; any multi-bit operand **with the exception of an input word** may be used as a destination.

6. Operations in detail

Arithmetic function: SUBTRACTING

The four basic arithmetic functions adding, SUBTRACTING, multiplying and dividing are available for all Festo controllers. These arithmetic functions can also be used in Ladder diagram.

In the Festo controller, these arithmetic functions are always used for **whole, signed numbers** in the range -32767.

The overflow cannot be interrogated.

If you use these numbers outside this range, no error message is given. Overflow is simply truncated (not rounded).

The FPC 405 also allows the use of function module 20 to program arithmetic operations with an overflow that can be interrogated.

SUBTRACTING can be programmed as fol-

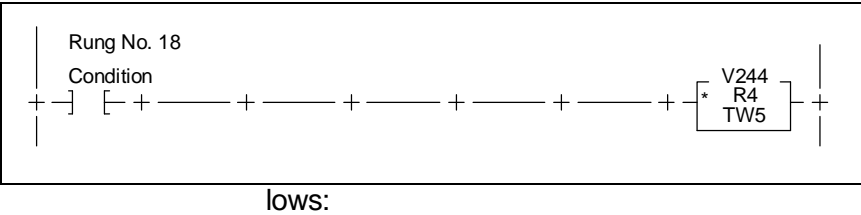


Fig. 6/42: Arithmetic function: Subtracting

6. Operations in detail

The same operation with the same function can be programmed with the Arithmetic/Logic Box as follows:

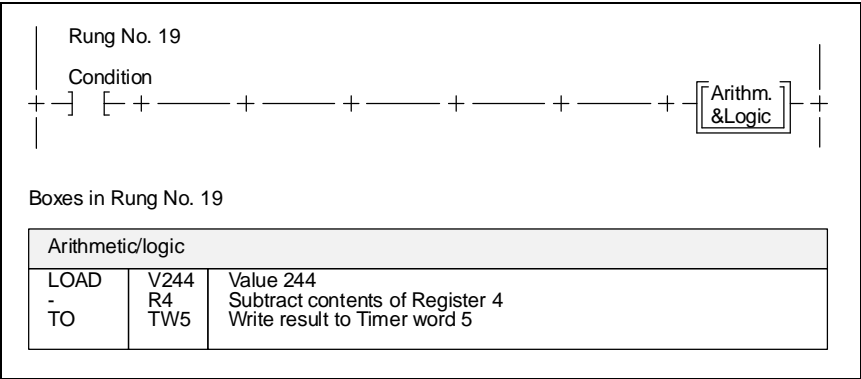


Fig. 6/43: Arithmetic function: Subtracting in the arithmetic box

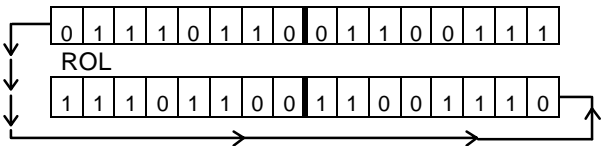
Please note that you always need at least three operands for an arithmetic operation. The first two operands are subtracted and the result is written to the third operand. The contents of the first operands (or other operands included with arithmetic functions) are not changed, if neither of these operands is given in the result field. Any multi-bit operand can be used as a source; any multi-bit operand **with the exception of an input word** may be used as a destination.

6. Operations in detail

ROTATE LEFT

ROTATE is one of the commands that shifts the individual bits of a word. In contrast to shift, ROTATE ensures that no bits are lost. The bit pushed off one end of the word is inserted at the other end. So if a 16-bit word is rotated 16 times, it returns to its original value.

In the case of ROTATE LEFT (ROL) the bits are shifted left (in the Festo controllers all 16 bits are always shifted). The most significant bit (extreme left) is inserted on the extreme right as the least significant bit.



ROTATE LEFT is programmed in Ladder diagram as follows:

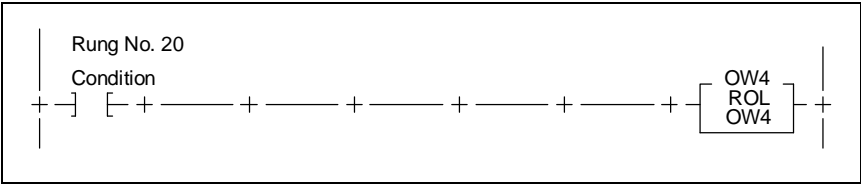


Fig. 6/44: Rotate left

6. Operations in detail

The same operation programmed with the Arithmetic/Logic Box looks like this:

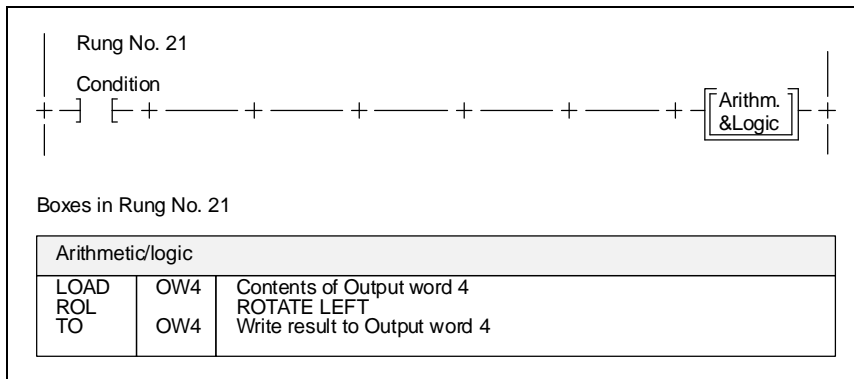


Fig. 6/45: Rotate left in arithmetic box

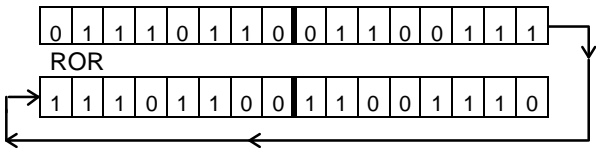
The ROTATE operation can be used for any multi-bit operand. The result can be written to any multi-bit operand **except an input word**.

6. Operations in detail

ROTATE RIGHT

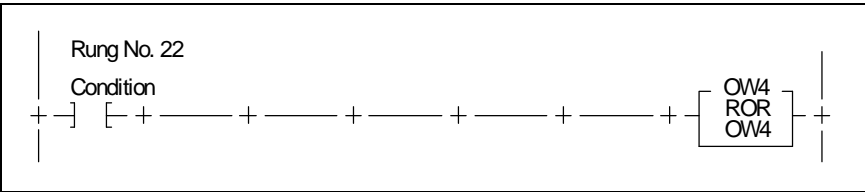
ROTATE is one of the commands that shifts the individual bits of a word. In contrast to shift, ROTATE ensures that no bits are lost. The bit pushed off one end of the word is inserted at the other end. So if a 16-bit word is rotated 16 times, it returns to its original value.

In the case of ROTATE RIGHT (ROR) the bits are shifted to the right (in the Festo controller all 16 bits are always shifted). The least significant bit (extreme right) is inserted on the extreme left as the most significant bit.



ROTATE RIGHT is programmed in Ladder diagram as follows:

Fig. 6/46: Rotate right



6. Operations in detail

The same operation programmed with the Arithmetic/Logic Box looks like this:

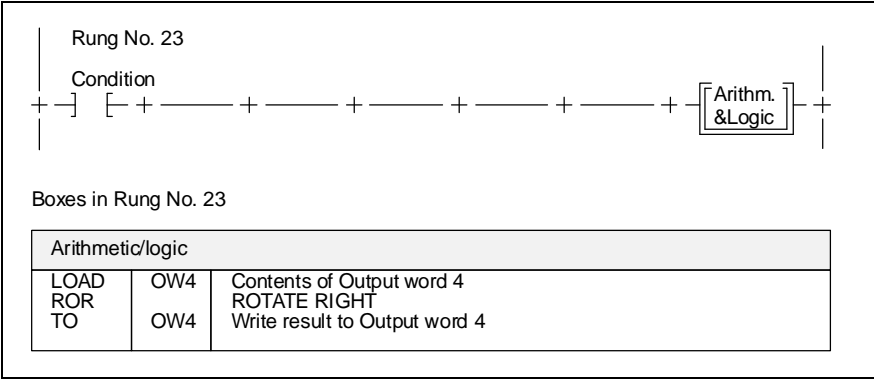


Fig. 6/47: Rotate right in arithmetic box

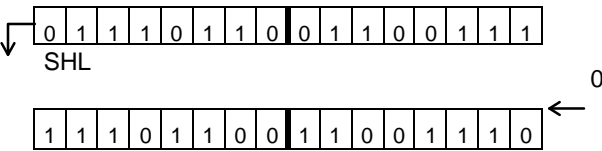
The ROTATE operation can be used for any multi-bit operand. The result can be written to any multi-bit operand **except an input word**.

6. Operations in detail

SHIFT LEFT

SHIFT is one of the commands that allows the individual bits of a word to be shifted. In contrast to rotate, SHIFT shifts one bit off the end of the word – this bit is lost. A 0 is inserted at the other end of the word. If a 16-bit word is shifted 16 times, all bits are set to 0. In the FPC 405 the bit shifted out can be interrogated with the local flag FI7.

SHIFT LEFT (SHL) shifts the bits to the left (in the Festo controller all 16 bits are always shifted). The most significant bit (extreme left) is shifted out, and a 0 is inserted at the extreme right.



In Ladder diagram SHIFT LEFT is programmed as follows:

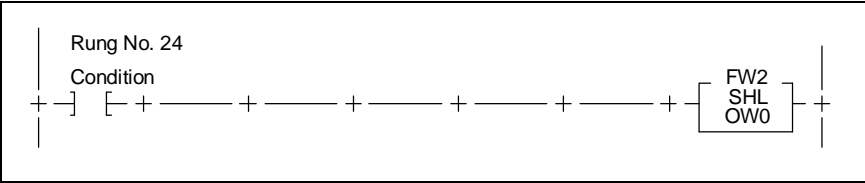


Fig. 6/48: Shift left

6. Operations in detail

The same operation programmed with the Arithmetic/Logic Box looks like this:

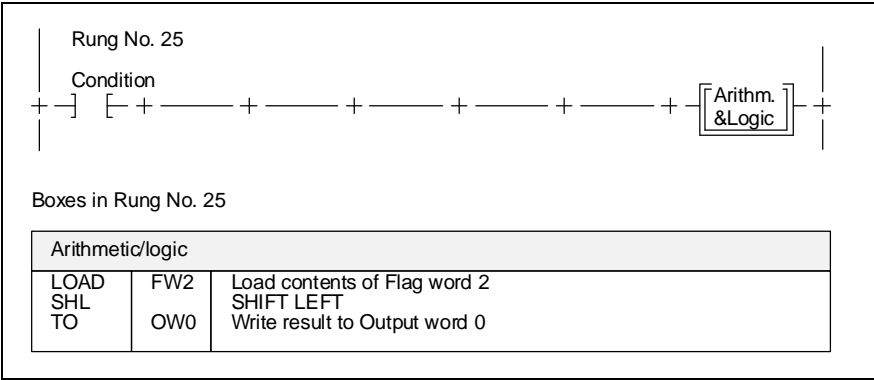


Fig. 6/49: Shift left in arithmetic box

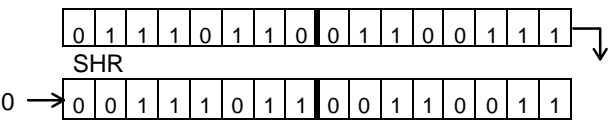
The SHIFT operation can be used to change any multi-bit operand. The result can be written to any multi-bit operand **except an input word**.

6. Operations in detail

SHIFT RIGHT

SHIFT is one of the commands that allows the individual bits of a word to be shifted. In contrast to rotate, SHIFT shifts one bit off the end of the word – this bit is lost. A 0 is inserted at the other end of the word. If a 16-bit word is shifted 16 times, all bits are set to 0. In the FPC 405 the bit shifted out can be interrogated with the local flag FI7.

SHIFT RIGHT (SHR) shifts the bits to the right (in the Festo controller all 16 bits are always shifted). The least significant bit (extreme right) is shifted out, and a 0 is inserted at the extreme left.



In Ladder diagram SHIFT RIGHT is programmed as follows:

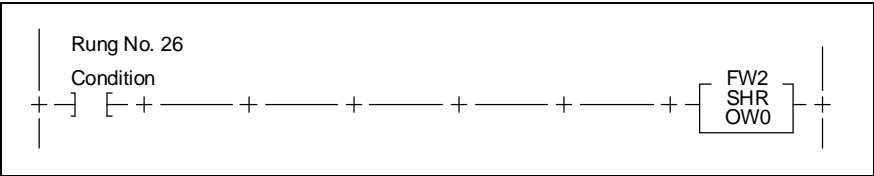


Fig. 6/50: Shift right

6. Operations in detail

The same operation programmed with the Arithmetic/Logic Box looks like this:

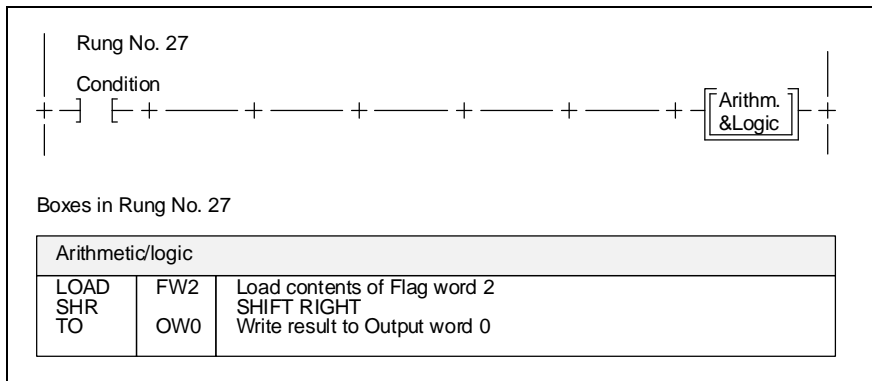


Fig. 6/51: Shift right in arithmetic box

The SHIFT operation can be used to change any multi-bit operand. The result can be written to any multi-bit operand **except an input word**.

6. Operations in detail

LATCH (memory)

The LATCH is the electrician's traditional method of constructing a one-bit memory with a relay. In the Festo controllers, the relay is replaced by a flag, allowing any LATCH to be constructed.

In programmable logic controllers, a one-bit memory can also be implemented with set and reset commands. However, the LATCH has not lost its significance.

We differentiate two types of latch: dominantly resetting and dominantly setting latches. The dominant resetting latch has the value logic 0 when switched on and off.

The logic symbol for the latch is shown by the flipflop:

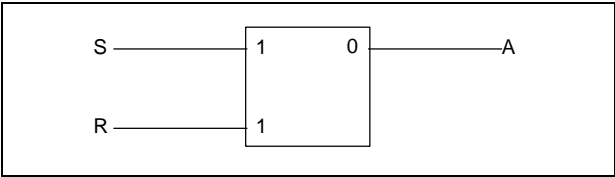


Fig. 6/52: Flipflop dominant resetting latch

6. Operations in detail

In Ladder diagram the dominantly resetting latch looks like this:

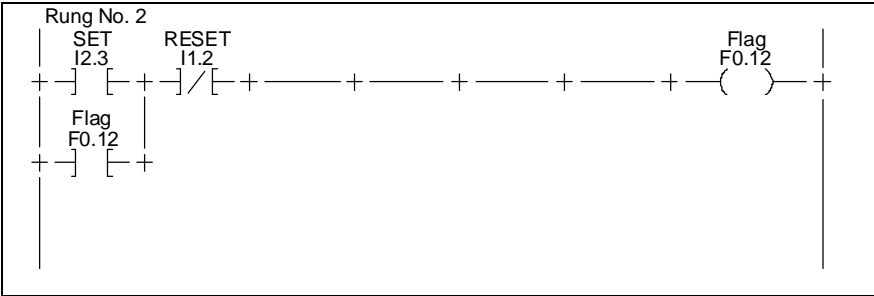


Fig. 6/53: Dominant resetting latch

The following method is also possible and probably more familiar to the electrician:

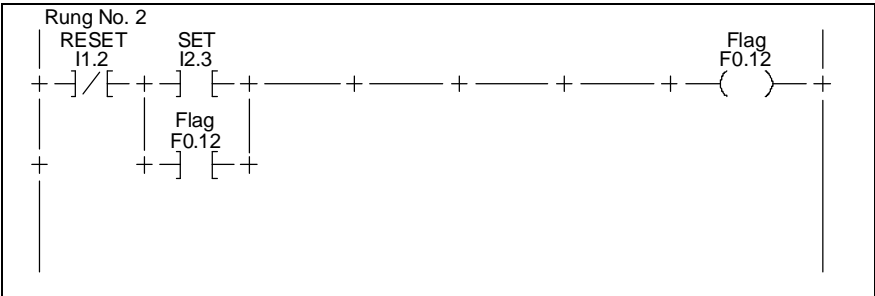


Fig. 6/54: Dominant resetting latch

6. Operations in detail

The dominantly setting latch is logic 1 at the output when switched on and off. The flipflop of the dominantly setting latch differs only from the dominant resetting latch in that it has a 1 at the output at the flipflop.

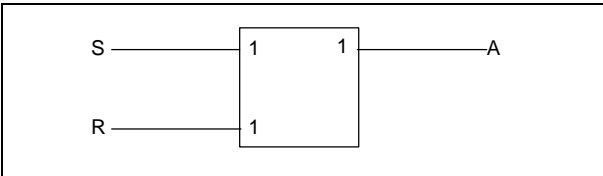


Fig. 6/55: Flipflop dominant setting latch

In Ladder diagram the resetting input is located in the latch itself in order to force dominant setting behaviour:

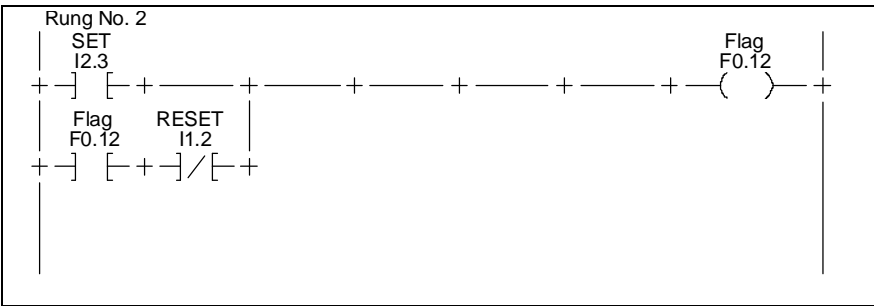


Fig. 6/56: Dominant setting latch

6. Operations in detail

JUMPING to a label

The JUMP command is one possibility of shortening the program cycle time by jumping over unneeded parts of the program. In Festo FST, any number of jumps may be programmed. JUMPS have a jump label as their destination, not a rung number. A label, that is a jump destination, is entered at the beginning of the conditional part of a rung. The jump command is inserted in the executive part of the rung and must contain the name of a jump label. This jump label must be contained in the program, or else a syntax error will be reported when translating in the program.

The JUMP command in Ladder diagram looks like this:

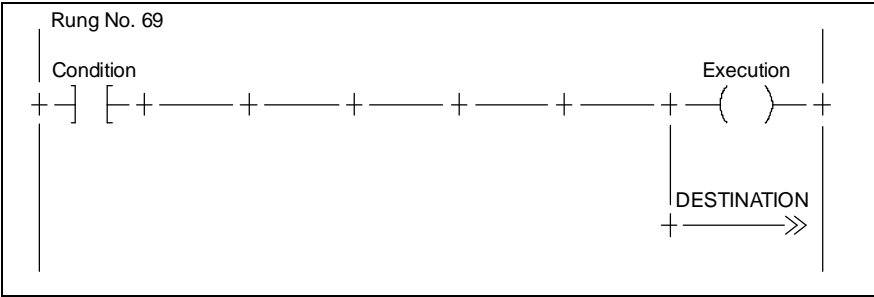


Fig. 6/57: Jump

6. Operations in detail

Many texts on programming differentiate between a conditional and unconditional JUMP. The jump command shown here is a conditional JUMP, but it can be easily converted into an unconditional jump:

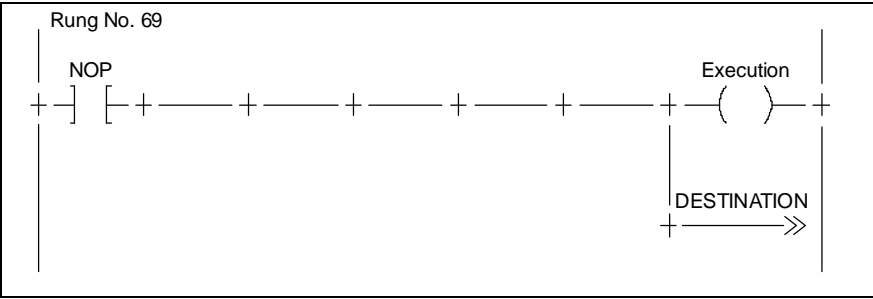


Fig. 6/58: Unconditional jump

Important: A jump command does not replace a contact. There must be at least one entry in each contact. If you only require a jump as a result of a condition, you must enter an (otherwise) unused flag as the operand in the contact.

6. Operations in detail

ENTERING A JUMP LABEL

The JUMP command is one possibility of shortening the program cycle time by jumping over unneeded parts of the program. In Festo FST, any number of jumps may be programmed. JUMPS have a jump label as their destination, not a rung number. A label or jump destination, is entered at the beginning of the conditional part of a rung. The jump command is inserted in the executive part of the rung and must contain the name of a jump label. This jump label must be contained in the program, or else a syntax error will be reported when translating in the program.

In Ladder diagram the JUMP label is entered as follows:

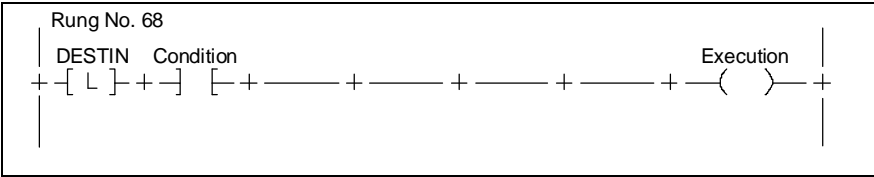


Fig. 6/59: Entering a jump label

Important: A jump command does not replace a contact. Every rung must have at least one contact, and a condition and executive part must be entered.

6. Operations in detail

Coils: RESET (cancel)

When coils are energised, basically we differentiate between stored and non-stored commands. The non-stored commands transfer the logic status of the condition (of the rung) directly to the coil. A coil reflects each change of the logic status of the rung.

In the case of stored commands RESET and set, the command is executed precisely if (and only if) the rung has logic status 1, i.e. the condition is therefore fulfilled.

If the rung assumes logic 0, i.e. the condition is not fulfilled, the coil which is programmed with a stored command is not changed. The status of the coil is only changed when the rung containing the opposite command becomes logic 1.

RESETTING of a coil is indicated by an R in the coil symbol:

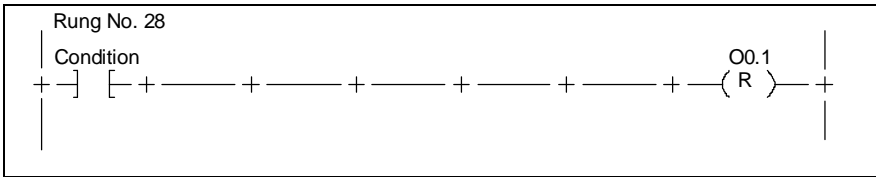


Fig. 6/60: Coil: Reset (Storing cancelled)

6. Operations in detail

Coils: SET

When coils are energised, basically we differentiate between stored and non-stored commands. The non-stored commands transfer the logic status of the condition (of the rung) directly to the coil. A coil reflects each change of the logic status of the rung.

In the case of stored commands SET and reset, the command is executed precisely if (and only if) the rung has logic status 1, i.e. the condition is therefore fulfilled.

If the rung assumes logic 0, i.e. the condition is not fulfilled, the coil which is programmed with a stored command is not changed. The status of the coil is only changed when the rung containing the opposite command becomes logic 1.

SETTING of a coil is indicated by an S in the coil symbol:

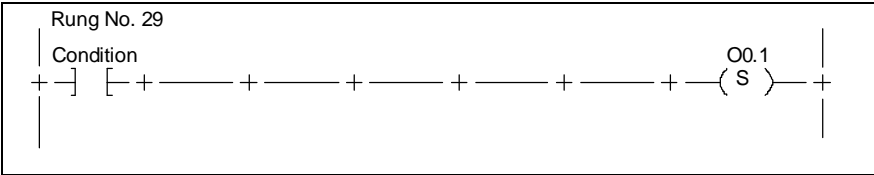


Fig. 6/61: Coil: Set

6. Operations in detail

Coils: ASSIGNMENT

When coils are energised, basically we differentiate between stored and non-stored commands. The non-stored commands – that is ASSIGNMENT or negated assignment – transfer the logic status of the condition (of the rung) directly to the coil. A coil reflects each change of the logic status of the rung.

In the case of stored commands set and reset, the command is executed precisely if (and only if) the rung has logic status 1, i.e. the condition is therefore fulfilled.

If the rung assumes logic 0, i.e. the condition is not fulfilled, the coil which is programmed with a stored command is not changed. The status of the coil is only changed when the rung containing the opposite command becomes logic 1.

ASSIGNMENT, that is the non-stored adoption of the logic status of the rung, is simply programmed by using the coil symbol.

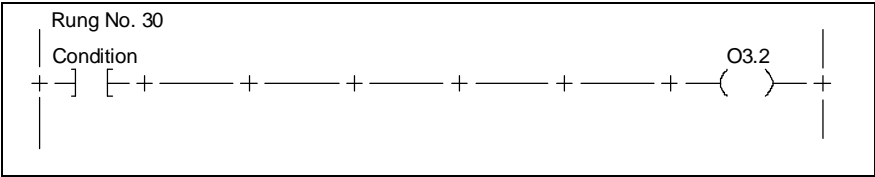


Fig. 6/62: Coil: Assignment

6. Operations in detail

Coils: NEGATED ASSIGNMENT

When coils are energised, basically we differentiate between stored and non-stored commands. The non-stored commands – that is assignment or **NEGATED ASSIGNMENT** – transfer the logic status of the condition (of the rung) directly to the coil. A coil reflects each change of the logic status of the rung.

In the case of stored commands set and reset, the command is executed precisely if (and only if) the rung has logic status 1, i.e. the condition is therefore fulfilled.

If the rung assumes logic 0, i.e. the condition is not fulfilled, the coil which is programmed with a stored command is not changed. The status of the coil is only changed when the rung containing the opposite command becomes logic 1.

In the case of **NEGATED ASSIGNMENT** the negation of the logic status of the rung is transferred to the coil (see also Logic: NOT).

NEGATED ASSIGNMENT, that is the non-stored adoption of the negated logic status of the rung, is programmed by using the negated coil symbol.

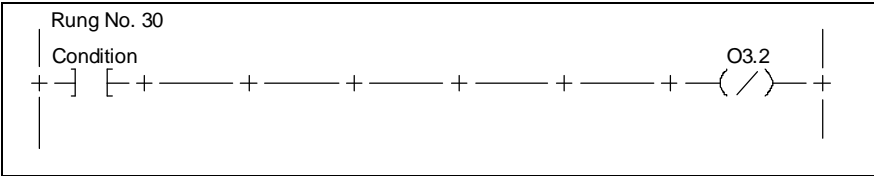


Fig. 6/63: Coil: Negated assignment

6. Operations in detail

Comparison: EQUAL

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- EQUAL
- Greater than
- Greater than or equal
- Less than
- Less than or equal
- Unequal

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The EQUAL operation is programmed as follows:

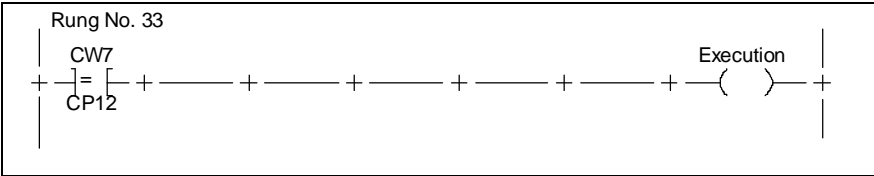


Fig. 6/64: Comparison: Equal

In this example the rung evaluates to logic 1 if counter word 7 has a value that is exactly equal to the value in counter preselect 12.

6. Operations in detail

Comparison: GREATER THAN

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- Equal
- GREATER THAN
- Greater than or equal
- Less than
- Less than or equal
- Unequal

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The operation GREATER THAN is programmed as follows:

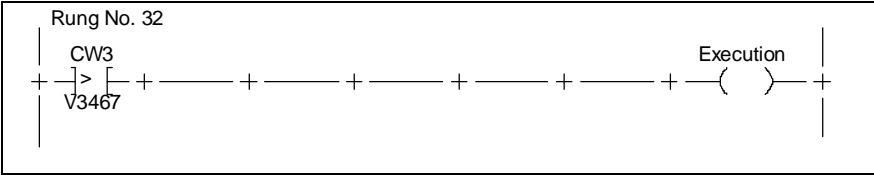


Fig. 6/65: Comparison: Greater than

In this example the rung evaluates to logic 1 if counter word 3 has a value that is greater than 3467 (for example 3468 or even 22341).

6. Operations in detail

Comparison: GREATER THAN OR EQUAL

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- Equal
- Greater than
- GREATER THAN OR EQUAL
- Less than
- Less than or equal
- Unequal

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The GREATER THAN OR EQUAL operation is programmed as follows:

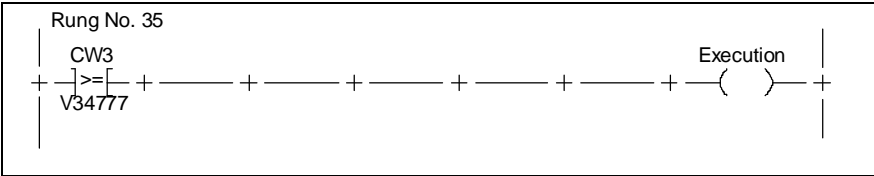


Fig. 6/66: Comparison: Greater than or equal

In this example the rung evaluates to logic 1 if counter word 3 has a value that is greater than or equal to 34777 (for example 34777, 34778 or greater).

6. Operations in detail

Comparison: LESS THAN

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- Equal
- Greater than
- Greater than or equal
- LESS THAN
- Less than or equal
- Unequal

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The LESS THAN operation is programmed as follows:

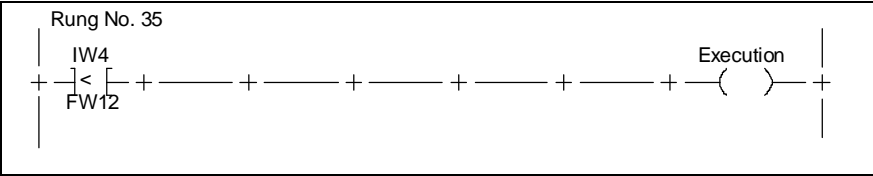


Fig. 6/67: Comparison: Less than

In this example the rung evaluates to logic 1 if input word 4 has a value that is less than the content of flag word 12.

6. Operations in detail

Comparison: LESS THAN OR EQUAL

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- Equal
- Greater than
- Greater than or equal
- Less than
- LESS THAN OR EQUAL
- Unequal

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The LESS THAN OR EQUAL operation is programmed as follows:

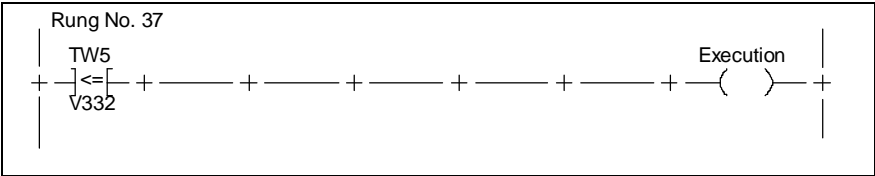


Fig. 6/68: Comparison: Less than or equal

In this example the rung evaluates to logic 1 if timer word 5 has a value that is less than or equal to 332 (that is 332, 331 or less).

6. Operations in detail

Comparison: UNEQUAL

Comparison operations are basic to every programmable logic controller. Comparisons allow processing of simple analogue values, multiple evaluation of counters, comparison of size etc.

Festo controllers allow all comparison operations to be programmed in Ladder diagram:

- Equal
- Greater than
- Greater than or equal
- Less than
- Less than or equal
- UNEQUAL

Comparison operations compare positive whole numbers between 0 and 65,535. If a negative value (for example V-10) is entered, the positive whole binary number corresponding to the bit pattern (of V-10) is compared. This does not apply to the FPC 405 if processing has been switched previously to signed numbers (from 0 to 32,767) with CFM20.

Comparison operations are always entered into a box in the conditional part.

6. Operations in detail

The value of the upper operand is always compared to the value of the lower operand. If the expression evaluates to true, the rung has the logic status 1. If the expression evaluates to false, the rung has the logic status 0.

The UNEQUAL operation is programmed as follows:

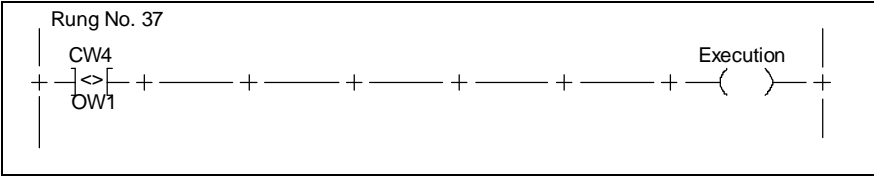


Fig. 6/69: Comparison: Unequal

In this example the rung evaluates to logic 1 if counter word 4 has a value that is unequal to the value of output word 1.

6. Operations in detail

Numbers: CONVERSION BCD to BINARY (DEB command)

When connecting coding switches to PLCs or when 7-segment displays are used, inputs and outputs require BCD-coded decimal numbers. Festo controllers have commands that allow conversion to and from BCD: the BID command for binary into decimal (that is, into BCD code) and the DEB command from decimal into binary (that is, from BCD code).

Please note that this operation always involves 16-bit words. An output word in BCD code consists of 4 digits each of which can be 0 -9.

The DEB command has the following effect:

7667_{BCD}

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

becomes 7667₂

0	0	0	1	1	1	0	1	1	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The conversion commands BID and DEB can be used in a multi-bit box or in the Arithmetic/Logic Box.

6. Operations in detail

The DEB command is programmed as follows:

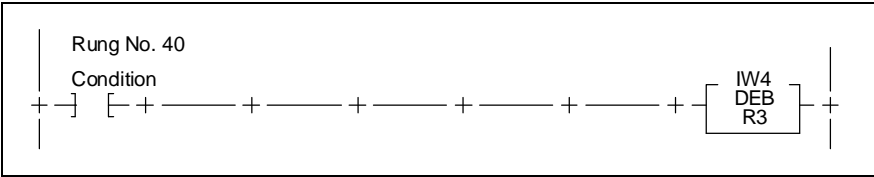


Fig. 6/70: Numbers: Convert BCD to binary

The following programming has exactly the same effect:

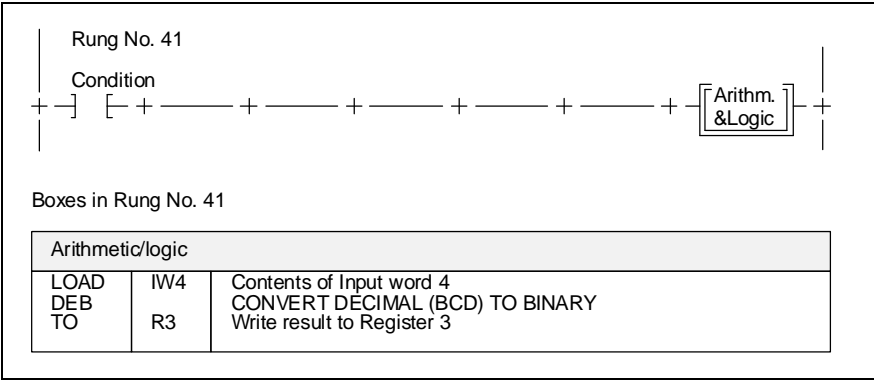


Fig. 6/71: Numbers: Convert BCD to binary in arithmetic box

The conversion commands allow conversion of the content of all multi-bit operands. The result can be written to any multi-bit operand except an input word.

In online mode you cannot display numerical values in BCD code. A BCD code is evaluated in binary (as a binary decimal number, a signed decimal number or as a hexadecimal number depending on the display mode).

6. Operations in detail

Numbers: CONVERSION BINARY to BCD (BID command)

When connecting coding switches to PLCs or when 7-segment displays are used, inputs and outputs require BCD-coded decimal numbers. Festo controllers have commands that allow conversion to and from BCD: the BID command for binary into decimal (that is, into BCD code) and the DEB command from decimal into binary (that is, from BCD code).

Please note that this operation always involves 16-bit words. An output word in BCD code consists of 4 digits each of which can be 0 -9.

The BID command has the following effect:

30311₂

0	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

becomes 0311_{BCD} (the leftmost digit is lost):

0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The conversion commands BID and DEB can be used in a multi-bit box or in the Arithmetic/Logic Box.

6. Operations in detail

The BID command is programmed as follows:

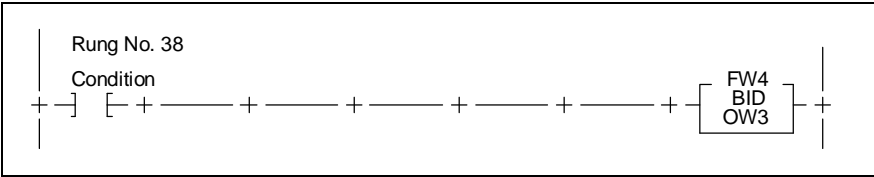


Fig. 6/72: Numbers: Convert digital to BCD

The following programming has exactly the same effect:

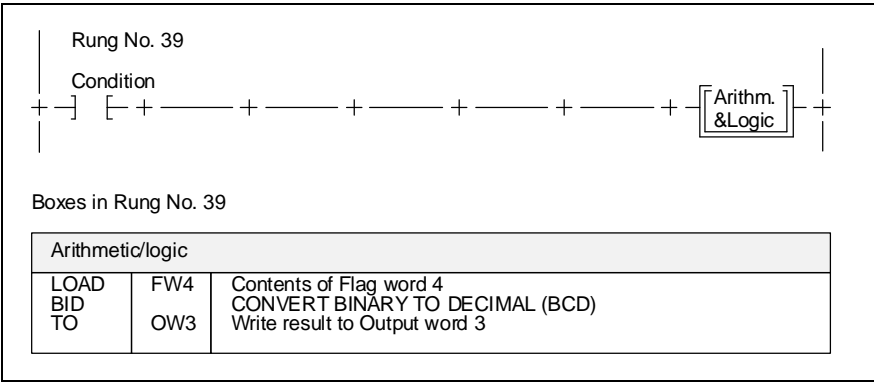


Fig. 6/73: Numbers: Convert binary to BCD in arithmetic box

The conversion commands allow conversion of the content of all multi-bit operands. The result can be written to any multi-bit operand except an input word.

In online mode you cannot display numerical values in BCD code. A BCD code is evaluated in binary (as a binary decimal number, a signed decimal number or as a hexadecimal number depending on the display mode).

6. Operations in detail

Counting: **INTERROGATED** **PRESELECT** **ACHIEVED**
?

The counters supplied in all Festo controllers are preselect counters. They consist of three elements: the counter preselect (nominal value), the counter word (actual value) and the counter bit (status of the counter).

Normally a counter counts from its initial value to its end value.

The one-bit operand C (counter) can be interrogated for logic 0 or logic 1.

The counter is logic 0 as soon as

- the counter word and the counter preselect (actual and nominal value) are identical or
- the counter word (actual value) becomes 0 (decremental counter) or
- the counter is not started.

C is logic 1 if – after starting a counter by a set command or the counter box or by loading a value into the preselect – the preselect has not been reached (incremental counter) or the actual value has not reached 0 (decremental counter).

6. Operations in detail

Interrogating whether the preselect has been reached (PRESELECT ACHIEVED) is therefore the same as interrogating whether the counter is inactive:

Compared to an electromechanical preselect counter the counter module has a normally closed contact (not a normally open contact or changeover contact) to indicate that the preselect has been reached.

Important: The condition of the rung shown is also fulfilled if the counter was not started.

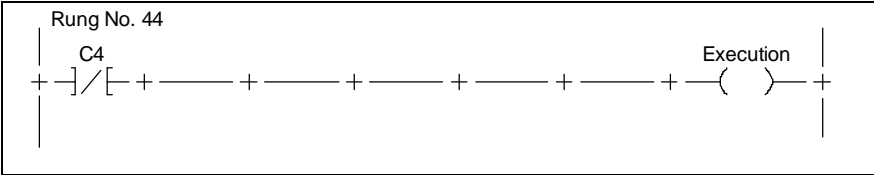


Fig. 6/74: Counter: Interrogated preselect achieved

6. Operations in detail

Counting: INTERROGATED PRESELECT NOT ACHIEVED?

The counters supplied in all Festo controllers are preselect counters. They consist of three elements: the counter preselect (nominal value), the counter word (actual value) and the counter bit (status of the counter).

Normally a counter counts from its initial value to its end value.

The one-bit operand C (counter) can be interrogated for logic 0 or logic 1.

The counter is logic 0 as soon as

- the counter word and the counter preselect (actual and nominal value) are identical or
- the counter word (actual value) becomes 0 (decremental counter) or
- the counter is not started.

C is logic 1 if – after starting a counter by a set command or the counter box or by loading a value into the preselect – the preselect has not been reached (incremental counter) or the actual value has not reached 0 (decremental counter).

6. Operations in detail

Interrogating whether the preselect has not been reached (PRESELECT NOT ACHIEVED) is the same as interrogating whether the counter is active:

Compared to an electromechanical preselect counter the counter module has a normally closed contact (not a normally open contact or changeover contact) to indicate that the preselect has been reached.

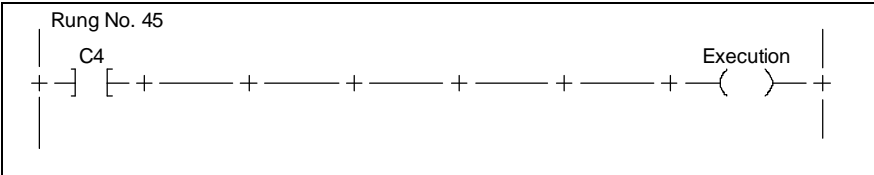


Fig. 6/75: Counter: Interrogated preselect not achieved

6. Operations in detail

Counting: DECREMENT

Counters can count forwards or backwards: incrementing or decrementing. Both of these types are supported by Festo Ladder diagram.

The counting commands can be used for any word except for inputs.

The one-bit operand C can be used during counting. In reality, the counter word (actual value) is changed.

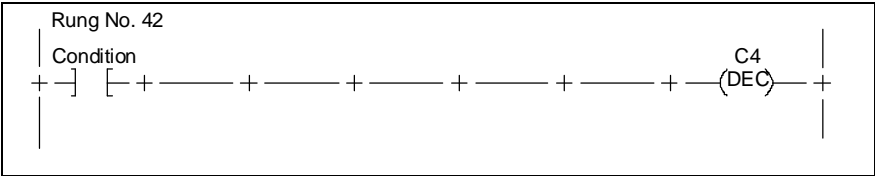
The counter command changes the value of the operand by 1: +1 for incrementer, -1 for decrementer.

The counting command is only executed when the logic status of the rung changes from 0 to 1, that is, only on a rising signal edge. This means that a Ladder diagram imitates the classical electromechanical counter which also counts the rising signal edge.

In Festo Ladder diagram, the counting commands are coil commands, not boxes.

In Ladder diagram, DECREMENT is programmed as follows:

Fig. 6/76: Counting: Decrement



6. Operations in detail

Counting: INCREMENT

Counters can count forwards or backwards: incrementing or decrementing. Both of these types are supported by Festo Ladder diagram.

The counting commands can be used for any word except for inputs.

The one-bit operand C can be used during counting. In reality, the counter word (actual value) is changed.

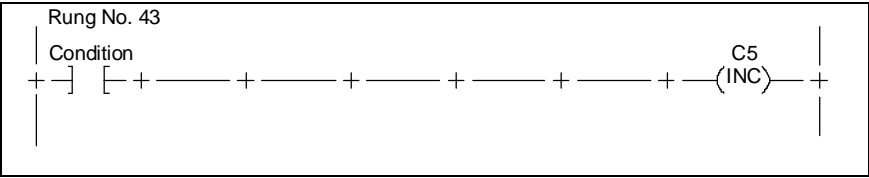
The counter command changes the value of the operand by 1: +1 for incrementer, -1 for decrementer.

The counting command is only executed when the logic status of the rung changes from 0 to 1, that is, only on a rising signal edge. This means that a Ladder diagram imitates the classical electromechanical counter which also counts the rising signal edge.

In Festo Ladder diagram, the counting commands are coil commands, not boxes.

In Ladder diagram, INCREMENT is programmed as follows:

Fig. 6/77: Counting: Increment



6. Operations in detail

Numbers: SET COUNTER

With a preselect counter it must be possible to set a definite starting value. It must be established whether it is an incremener (0 to preselect) or decremener (preselect to 0). A counter must be set at least once during the program and can then be reset to initial value any number of times with a simple set command.

The simplest case is the incremental counter. The nominal value is loaded into the preselect and the actual value set to 0. This can be done as follows:

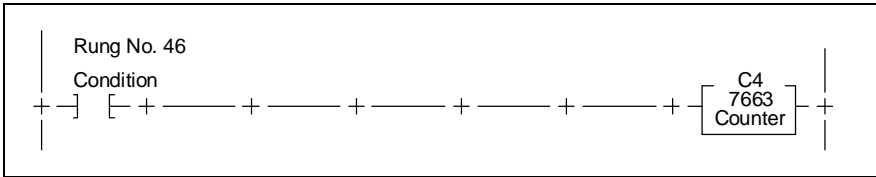


Fig. 6/78: Counting: Set incremental counter

A simple set of command now sets the counter (actual value) to this initial value. A reset command stops the counter, setting it to logic 0.

6. Operations in detail

Resetting the actual value (counter word) to 0 again is done with the SET command in the executive part of the rung, whereby the counter bit C becomes logic 1 (the normally closed contact closes again).

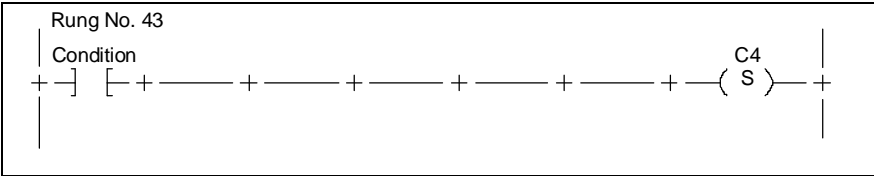


Fig. 6/79

In the case of a decrementer, the counter word must be loaded with the initial value as it will count from this initial value to 0:

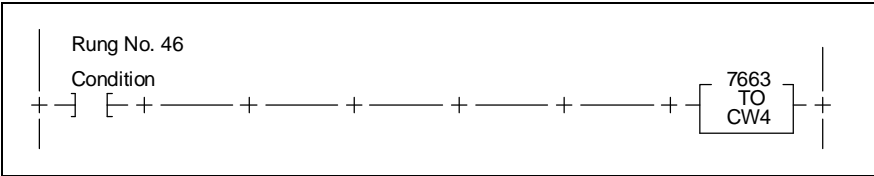


Fig. 6/80: Counting: Set decrementer counter

For more detailed examples, please see Chapter 9, "Programming of counters".

6. Operations in detail

Time: Start pulse timer

Each processor module of a Festo controller has a number of timer modules. Each of these timer modules can be used as a pulse timer. In addition, the FPC 100B/AF, IPC, FEC and FPC 405 controllers enable the incorporation of a switch-on and switch-off delay similar to the classic time relay.

A timer module consists of two elements:

- One element which is START TIMER.
- Another element which is interrogate timer.

START TIMER is an executive function, that is, a box, which replaces a coil.

Interrogate timer is a conditional function realized as a contact.

STARTING of a pulse timer is programmed, for example, as follows:

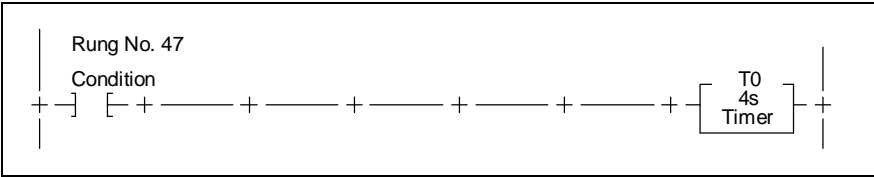


Fig. 6/81: Time: Start pulse timer

For details of timer programming see Chapter 8, "Programming of timers"

6. Operations in detail

Time: Interrogate pulse timer

Each processor module of a Festo controller has a number of timer modules. Each of these timer modules can be used as a pulse timer. In addition, the FPC 100B/AF, IPC, FEC and FPC 405 controllers enable the incorporation of a switch-on and switch-off delay similar to the classic time relay.

A timer module consists of two elements:

- One element which is start timer.
- Another element which is INTERROGATE TIMER.

Start timer is an executive function, that is, a box, which replaces a coil.

INTERROGATE TIMER is a conditional function realized as a contact.

INTERROGATION of a pulse timer is programmed, for example, as follows:

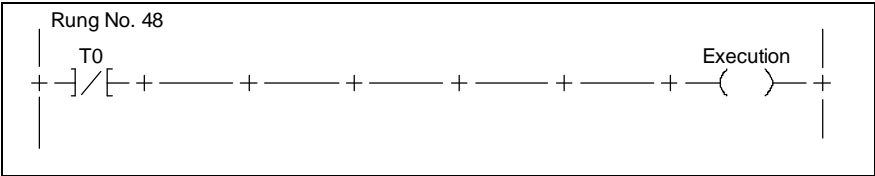


Fig. 6/82: Time: Pulse timer expired?

For details of timer programming see Chapter 8, "Programming of timers"

7. Addressing operands

Chapter 7
Addressing operands

7. Addressing operands

Contents

7.1	ABSOLUTE AND SYMBOLIC ADDRESSES.....	145
	Absolute address	145
	Symbolic address	148
	Mixing symbolic and absolute addresses.....	150
7.2	INFORMATION CONTENT OF THE ABSOLUTE ADDRESS.....	153

7. Addressing operands

When using a programmable logic controller, operands must be addressed.

7.1 ABSOLUTE AND SYMBOLIC ADDRESSES

The FST software differentiates between **absolute** and **symbolic** addresses.

Absolute address

Absolute addresses are the designators required by the hardware and operating system of a PLC. Generally these are addresses that are incorporated by the manufacturer and used by the operator with the aid of LEDs, digits and selector switches.

An absolute operand address comprises an operand identifier and an operand parameter.

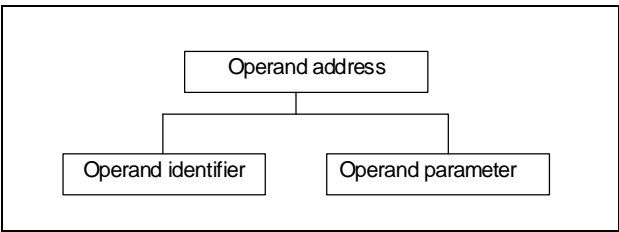


Fig. 7.1

7. Addressing operands

The operand identifier is an abbreviation normally consisting of one or two letters:

O	Output
OW	Output Word
CFM	Function Module
CMP	Program Module
I	Input
IW	Input Word
FU	Function Unit
E	Error
EW	Error Word
V	Value
V\$	Hexadecimal Value
F	Flag
FI	Flag Initialization
FW	Flag Word
NOP	Null Operation
P	Program
R	Register
T	Timer
TOFF	Switch-off delay timer
TON	Switch-on delay timer
TP	Timer Preselect
C	Counter
CP	Counter Preselect
CW	Counter Word

The absolute operand identifiers are **auto-**
matically recognized by the FST software.

7. Addressing operands

Most operand identifiers require an additional operand parameter, a number, which allows the specific input to be addressed.

A few operands do not allow an additional parameter:

FI	Flag Initialization
E	Error
NOP	Null Operation

When using FST software, there must be **no space** between the operand identifiers and the appropriate parameter.

The FST software automatically recognizes a valid absolute operand address and handles it appropriately.

7. Addressing operands

Symbolic address

During programming it is often very difficult and tedious to remember all operand addresses. A whole range of operands are used in various different applications. Commentaries could also help to make a program easier to read and understand.

This is the purpose of **symbolic operands**. These are the names given to operands by the user.

In FST software symbolic operands **may have up to 9 characters, no spaces, no period and a limited range of special characters**.

FST software symbolic operands therefore consist essentially of letters and numbers.

Some examples of useful names for symbolic operands are:

- Start
- Stop
- Left
- Right
- Motor1on
- Motor1off
- GRIPCLOSE
- GRIPOPEN

7. Addressing operands

Naturally, a symbolic operand name must be **unique** within **a project**.

Unique means also that a name given to a symbolic operand cannot be given as a jump label.

Important: *Symbolic operands must be clearly differentiated from absolute operands. The identifiers used for absolute operands (such as V, Y, F, etc.) must not be used as symbolic operand identifiers. For example, V12 or Y1 are absolute addresses. VALVE1, for example, would be permissible as a symbolic address.*

7. Addressing operands

Mixing symbolic and absolute addresses

FST software allows mixing of symbolic and absolute operand addresses. An allocation list into which you enter all the names can be created either before starting programming or during programming.

You can also switch off the allocation list function during programming and program without allocation list entry. This can be useful if you want to program a routine that is to be valid for various machines for which the absolute addresses are not yet known.

The following must always be taken into account however:

As the FST software **automatically** recognizes absolute addresses, symbolic addresses must be selected that do not conflict with absolute addresses.

A typical conflict in Ladder diagram is the use of the symbol "V" for a coil.

As "V" is the identifier for the absolute operand and Constant, "V12" is not interpreted by the FST software as a symbolic address but as an absolute address.

7. Addressing operands

Apart from this limitation, symbolic and absolute addresses can be mixed in FST programs. As an additional aid in Ladder diagram programming for FST software version 3.1 and later, both absolute and symbolic addresses are automatically shown where a symbolic address has been entered.

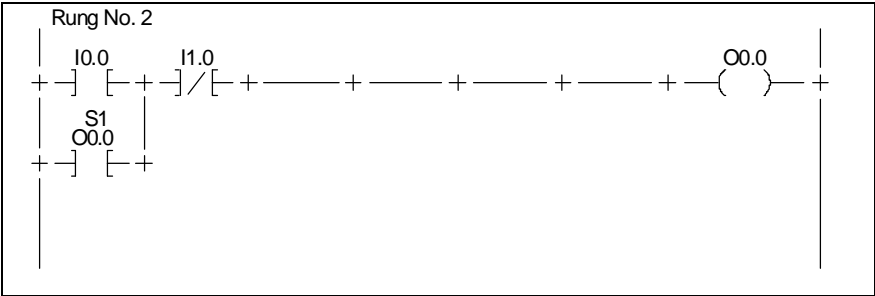


Fig. 7/2: Symbolic and absolute operands

In this example S1 was entered as a symbolic address for the latch, whereas in the output the absolute operand (O0.0) address has been entered.

7. Addressing operands

The allocation list lists all absolute and symbolic operands. The allocation list is sorted alphabetically by absolute operand identifier.

Operands of the allocation list		Comment
Absolute	Symbolic	
O0.0	H1	Indicate machine started
I0.0	Start	S1: Start pushbutton
I1.0	Stop	S2: Stop pushbutton

Practice has shown that only specific symbolic addresses such as "Start" are useful. The symbolic address "SV1CCS3bk" for "Solenoid valve 1 for cylinder C at station 3 back" is just as inconclusive as the absolute address "O3.2".

7. Addressing operands

7.2 INFORMATION CONTENT OF THE ABSOLUTE ADDRESS

The absolute address for Festo controllers always has the same structure of the operand parameter (number). The absolute address has the following structure:

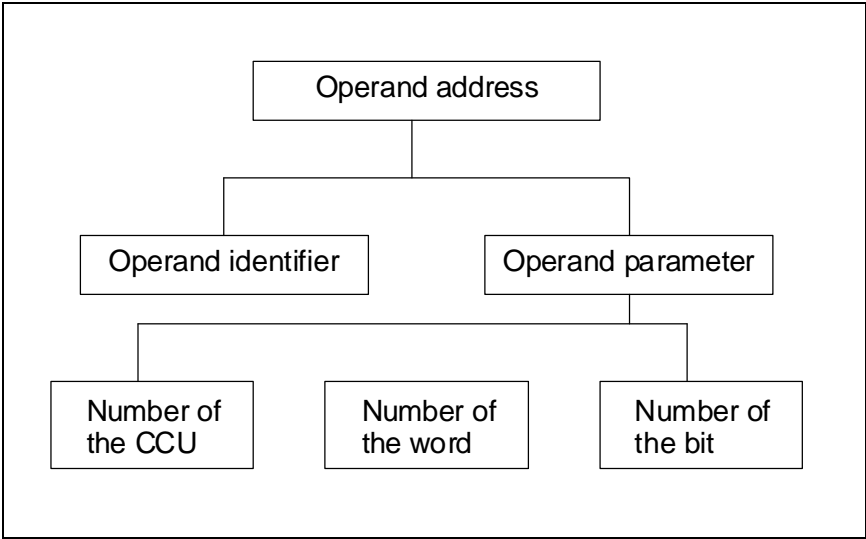


Fig. 7/3

Please note that the three parts of the operand parameter are separated by a period. Spaces are not permissible.

7. Addressing operands

Example

Addressing flags:

As the FPC 100B/AF, IPC and FEC have only one processor module in a system (single-processor system), it does not make sense to allocate a number for the processor module. Therefore, in these controllers flags can be addressed from

F0.0 to Fn.n.

On the other hand, the FPC 405 controller can manage several processor modules and processor modules can access the flag of other CCUs. For this reason addressing is from

F0.0.0 to Fn.n.n.

As these flags are one-bit operands, each bit is numbered individually.

However, if you are using a flag word, it is incorrect to give a bit number. For this reason the flag words for FPC 100B/AF, IPC and FEC are addressed from

FW0 to FWn.

In the case of the FPC 405 controller, flag words are addressed from FW0.0 to FWn.n as 6 processor modules can be used in one system.

7. Addressing operands

The following table provides an overview of the absolute addresses possible with the various Festo controllers.

Operand	FPC 100B/AF	FEC	IPC	FPC 405
Output (word)				
O	O0.0 - O1.5	O0.0 - O0.7*	O0.0 - O255.15*	O0.0 - O15.15
OW	OW0 - OW1	OW0 *	OW0 - OW255*	OW0 - OW15
Function module				
CFM	CFM0	CFM0 - CFM99	CFM0 - CFM99	CFM0 - CFM75
Program module				
CMP	CMP0 - CMP7	CMP0 - CMP99	CMP0 - CMP99	CMP0 - CMP75
Input (word)				
I	I0.0 - I2.7	I0.0 - I1.3*	I0.0 - I255.15*	I0.0 - I15.15
I ¹	—	—	—	I0 - I7
IW	IW0 - IW2	IW0 - IW1*	IW0 - IW255*	IW0 - IW15
IW ²				IW
Error (word)				
E	E	E	E	E
EW	EW	EW	EW	EW
Function unit				
FU	FU0 - FU23	FU0 - FU23	FU0 - FU23	FU0 - FU23
FU	FU32 - FU47	FU32 - FU47	FU32 - FU47	FU32 - FU47
Initialization flag				
FI	FI	FI	FI	FI
Flag (word)				
F	F0.0 - F15.15	F0.0 - F9999.15	F0.0 - F9999.15	F0.0.0-F5.63.15
FW	FW0 - FW15	FW0 - FW9999	FW0 - FW9999	FW0.0-FW5.63
FI ³				FI0 - FI7

1, 2, 3 see following page

* On FEC and IPC we are using logical addresses. Value range from 0 ... 255

7. Addressing operands

Operand	FPC 100B/AF	FEC	IPC	FPC 405
Null operation NOP	NOP	NOP	NOP	NOP
Program P	P0 –	P0 - P63	P0 - P63	P0 - P63
Register R	R0 - R63	R0 - R255	R0 - R255	R0 - R127
Timer T TOFF TON TP TW	T0 - T31 TOFF0-TOFF31 TON0 - TON31 TP0 - TP31 TW0 - TW	T0 - T255 TOFF0-TOFF255 TON0-TON255 TP0 - TP255 TW0 - TW255	T0 - T255 TOFF0–TOFF255 TON0–TON255 TP0 - TP255 TW0 - TW255	T0 - T63 TOFF0-TOFF63 TON0 - TON63 TP0 - TP63 TW0 - TW63
Counter C CP CW	C0 - C15 CP0 - CP15 CW0 - CW	C0 - C255 CP0 - CP255 CW0 - CW255	C0 - C255 CP0 - CP255 CW0 - CW255	C0 - C63 CP0 - CP63 CW0 - CW63

1. These refer to the internal inputs of the interrupt processor module of the FPC 405. These inputs are local to one processor module and can be interrupted, have practically no input signal delay and can be addressed as I0 to I7.
2. If internal inputs of the interrupt processor module of the FPC 405 are to be processed as a word, the absolute address is IW.
3. Each FPC 405 **program** has local flags addressed from FI0 to FI7.

7. Addressing operands

Naturally this comparison table is not of great interest if you are programming other makes of controller. This table is more relevant if you wish to use programs written for one type of Festo controller on another type.

Depending on which type of controller a program has been developed, it will be necessary to modify the operand parameters.

Notes

[illegible]

8. Programming timers

Chapter 8

Programming timers

8. Programming timers

Contents

8.1	GENERAL.....	161
	Timers	161
	Timer errors	162
	Shortest and longest time	163
	Components of a timer module	164
	Activating timer initialization	164
8.2	TIMER TYPES	165
	Switch-on delay timer	166
	Switch-off delay timer	167
	Pulse timer	168
8.3	EXAMPLES.....	169
	Switch-on delay	169
	Programming with TON.....	170
	Programming with TOFF	171
	Programming with T	171
	Stairwell lighting	172
	Programming with TON.....	172
	Programming with TOFF	173
	Programming with T	174
	Flashing lamp	175
	Programming with TON.....	175
	Programming with TOFF	176
	Programming with T	177
8.4	EVALUATING THE INDIVIDUAL TIMER ELEMENTS	178
	Programming flashing lamp with comparison	180

8. Programming timers

8.1 GENERAL

Timers

Timer modules are incorporated in all modern programmable logic controllers. All Festo controllers offer several timer modules for each processor module. Depending on the controller type, the number of timer modules range between 16 (FPC 100B/AF) and 64 (FPC 405) per processor module.

The timer modules are counters which count an internal clock pulse. Timer modules of Festo controllers use a clock pulse of 10 milliseconds, that is, 0.01 seconds. This clock pulse is fully process-independent and is available as soon as the processor module is powered up.

If a time is programmed, the preset counter is loaded with a number. The preset counter decrements by one for each clock pulse. When 0 is reached the time has expired.

This time measurement technique works very well and is widely used. The same technique is used in modern quartz clocks and watches: a counter counts clock pulses from a quartz crystal. This is known as digital time measurement.

8. Programming timers

Timer errors

Digital time measurement has a systematic minimum error of one clock pulse. This error results from the fact that the clock pulse is process-independent.

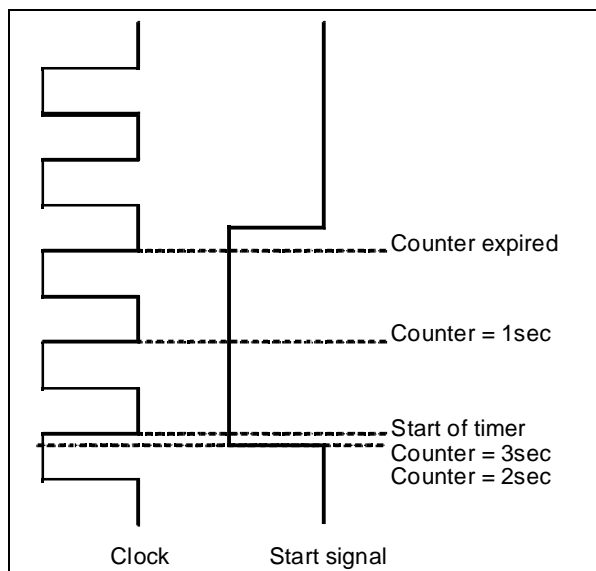


Fig. 8/1: Example of timer

In the above example the timer counts each trailing edge of the clock pulse. As the input signal changes to logic 1 "sometime", this means that if 3 pulses are programmed, it counts approximately 2 pulses up to the end of the time period.

This systematic error of event-driven time measurement has a maximum value of one clock pulse.

8. Programming timers

Shortest and longest time

The shortest time that can be measured by a timer of this type is two clock pulses. In this case the maximum error is 50%. (If a time of one clock pulse were programmed, the possible error would be 100%. If that were permissible, programming of the timer would be redundant.)

The longest time that can be measured with a timer of this type is dependent on the size of the counter. The counters in Festo controllers are all multi-bit operands of 16 bits. This type of counter can count up to a maximum of 65535. 65535 times 10 milliseconds is 655 seconds or 10 minutes and 55 seconds.

<i>A single timer module in a Festo controller can measure a time between 0.02 seconds and approx. 10 minutes.</i>

Longer time periods can be measured by cascading. Using this technique a single counter and a single timer module can be used to measure a period of up to 65,000 x 10 minutes.

8. Programming timers

Components of a timer module

Every timer must be started and interrogated. In Ladder diagram, timers are started in a timer box, which replaces a coil. Interrogation of the time is undertaken by a contact, just as in the circuit diagram.

Activating timer initialization

Timer initialization is a box replacing a coil in the Ladder diagram, that is, in the executive part of the rung (just as a coil of a time relay is the load in a circuit diagram).

The box is activated via a signal edge unless activated by a rung in which the contact is a NOP, i.e. the box operates once when the condition defined in the rung changes from 0 to 1. The condition can then remain logic 1 for any length of time – the timer continues to run. If the condition switches from 1 to 0, the switch-on delay timer module (TON) switches to logic 0 and the switch-off delay (TOFF) timer begins time measurement.

8. Programming timers

8.2 TIMER TYPES

Festo differentiates three different types of timer module:

- switch-on delay timer (TON),
- switch-off delay timer (TOFF) and
- pulse timer (T).

8. Programming timers

Switch-on delay timer

This timer is permanently set as long as the condition is not fulfilled (0 level), that is, the timer word is constantly loaded into the timer preselect. The timer is active (TON=0). When the condition is fulfilled (level 1) it is not reset, but begins running. When the word is 0, the timer is inactive (TON=1). The timer only becomes active again (TON=0) when the condition is no longer fulfilled.

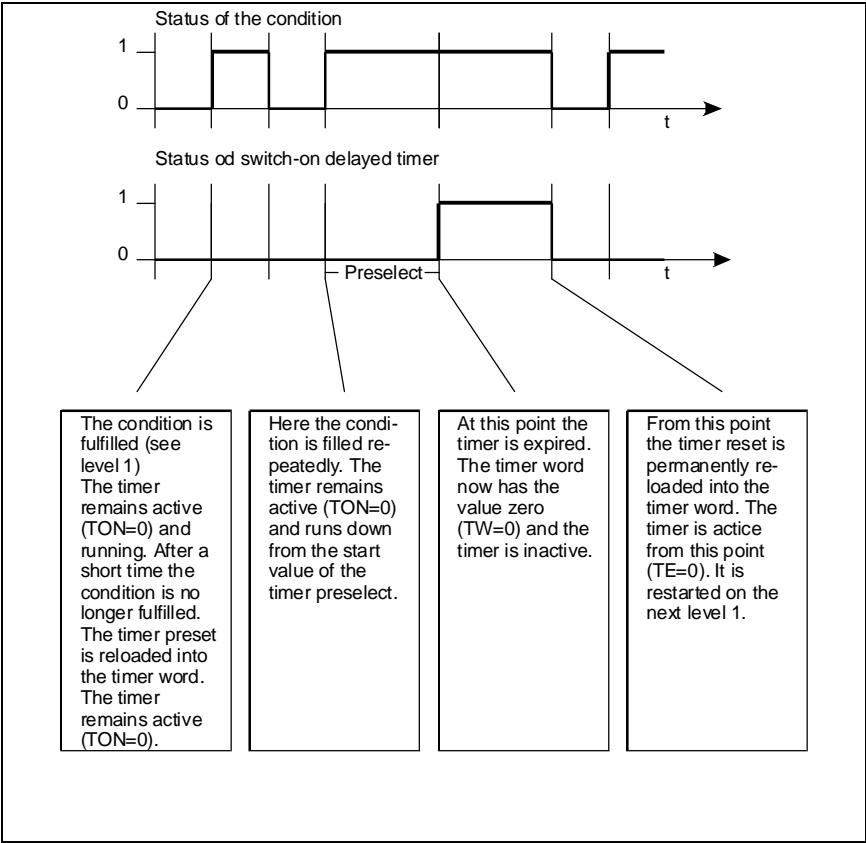


Fig. 8/2: Behaviour of switch-on delay timer

8. Programming timers

Switch-off delay timer

This timer reacts to level 1. This means that it is permanently set as long as the condition is fulfilled. The timer word is constantly loaded with the timer preselect; the timer is active ($TOFF=1$). When the condition is no longer fulfilled this timer is not reset, but begins to run. When the timer has expired ($TW=0$), it becomes inactive ($TOFF=0$). A timer does not become active again until the next level 1.

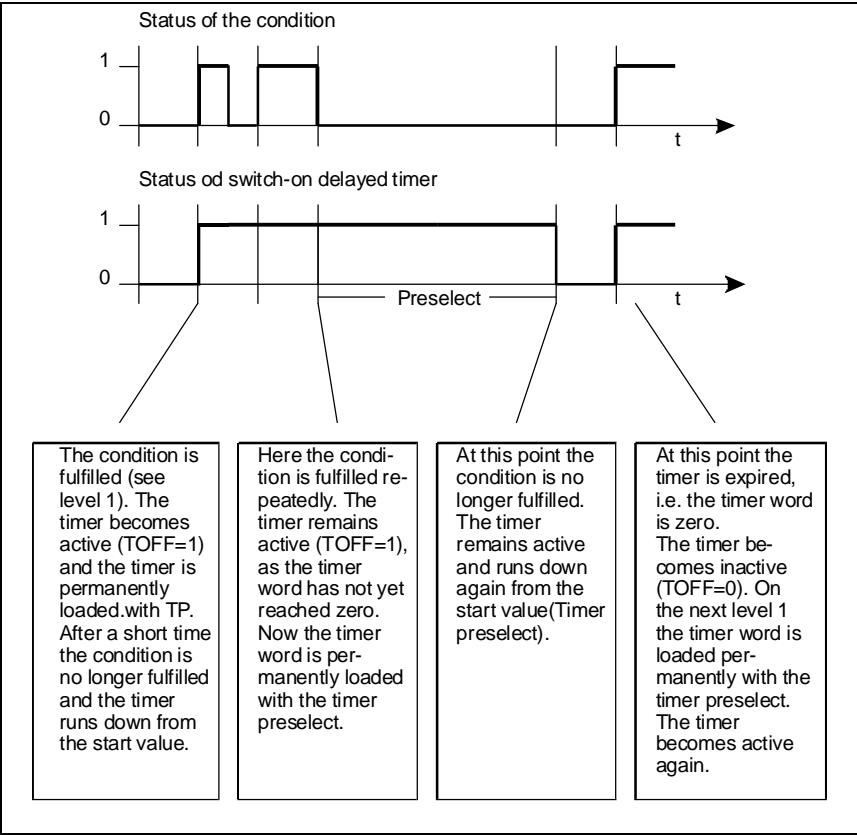


Fig. 8/3: Behaviour of switch-off delay timer

8. Programming timers

Pulse timer

This timer reacts to the rising edge of a condition. If the condition is fulfilled, the timer runs. As long as the condition is not fulfilled, the timer preselect is permanently loaded into the timer word.

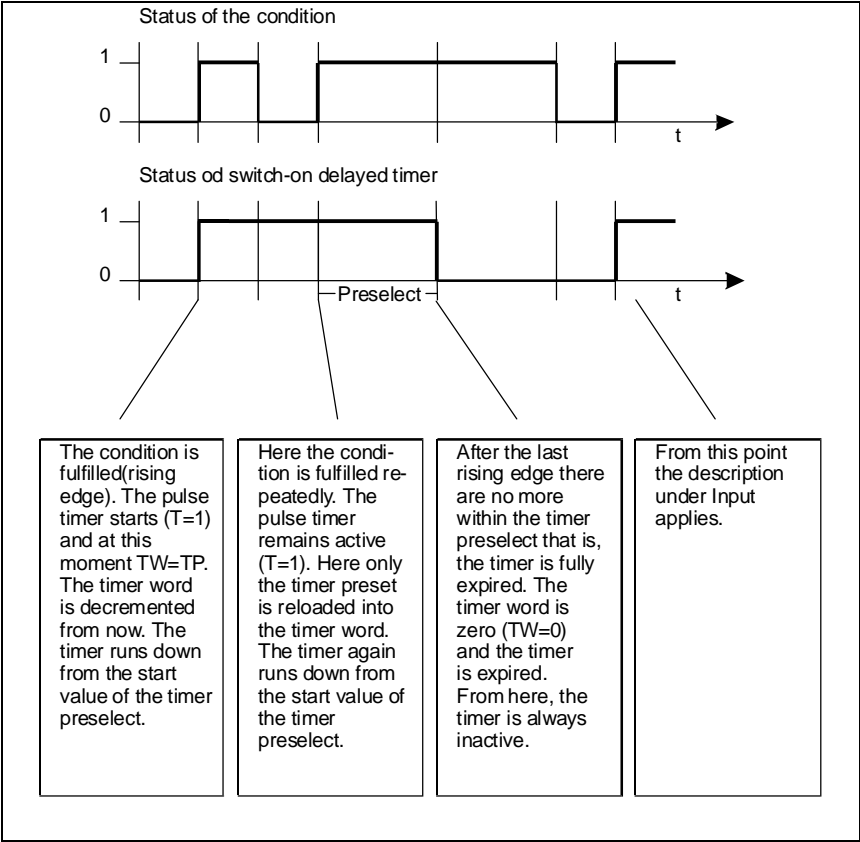


Fig. 8/4: Behaviour of pulse timer

8. Programming timers

8.3 EXAMPLES

The following examples are typical of time delays in control technology. All examples are shown with three solutions: with switch-on delay timer, switch-off delay timer and pulse timer.

Pick the solution that you like best.

Switch-on delay

A lamp is to illuminate after a push button has been pressed for 5 seconds. The lamp is not to illuminate if the push button is not pressed.

8. Programming timers

Programming with TON

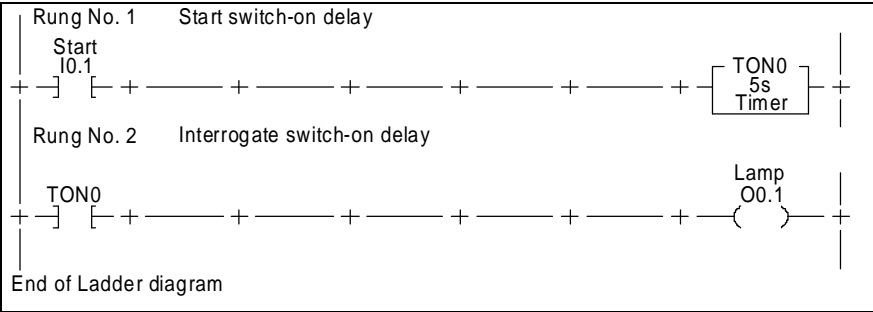


Fig. 8/5: Switch-on delay – programming with TON

The circuit diagram for the same function with a time relay looks like this:

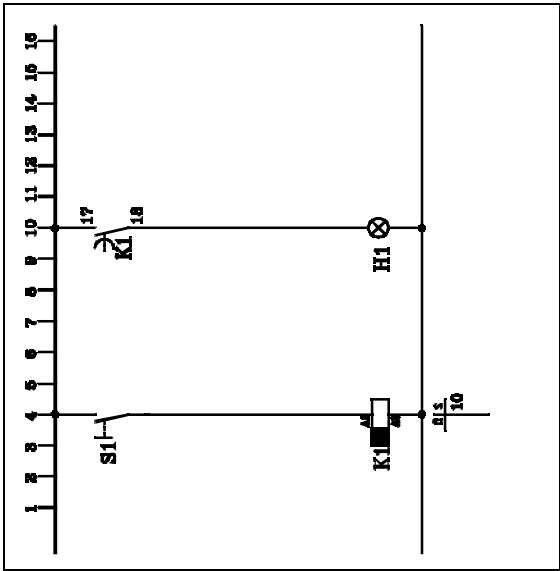
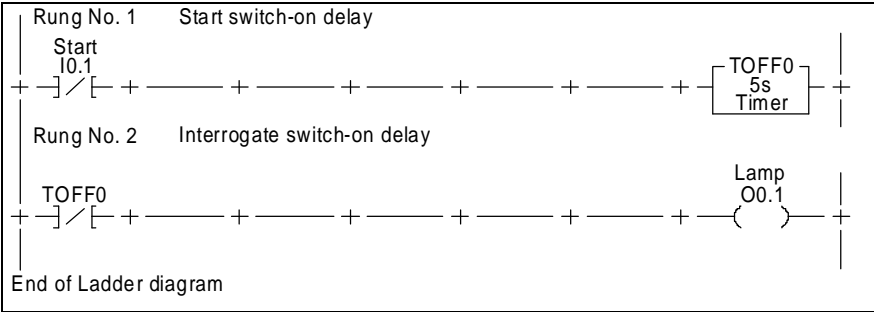


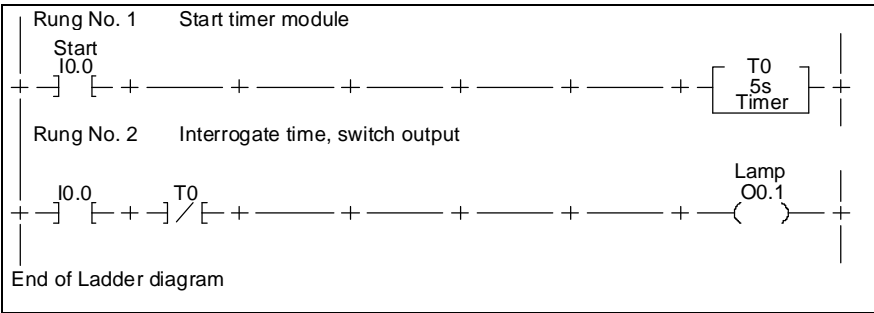
Fig. 8/6: Circuit diagram

8. Programming timers



Programming using TOFF

Fig. 8/7: Start and interrogate switch-off delay timer



Programming using T

Fig. 8/8: Start timer module/interrogate time, switch output

8. Programming timers

Stairwell lighting

A lamp is to illuminate as soon as the push button is pressed. When the push button is released, the lamp is to remain on for a further 5 seconds.

Programming using TON

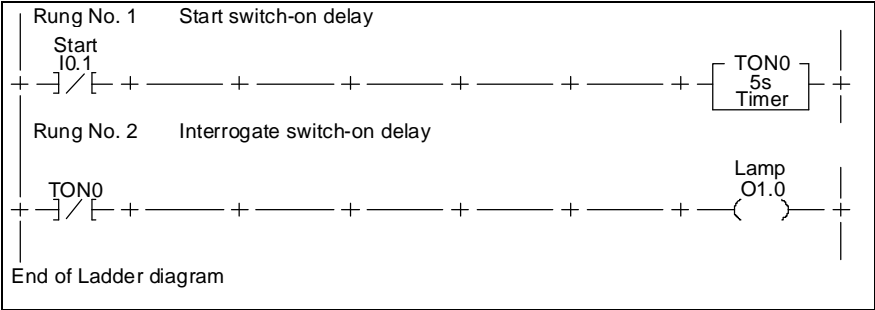


Fig. 8/9: Start and interrogate switch-on delay timer

8. Programming timers

Programming using TOFF

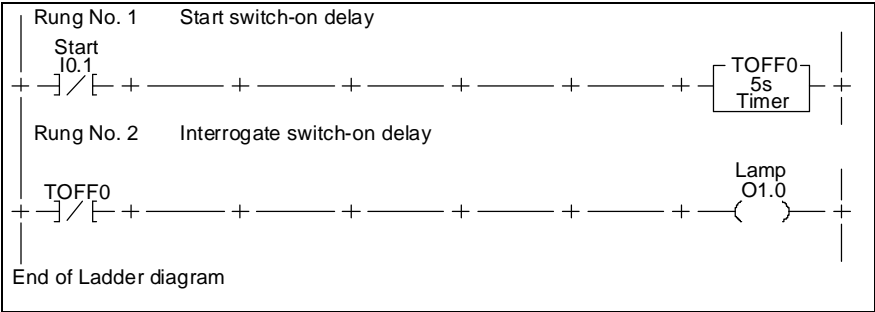


Fig. 8/10: Start and interrogate switch-off delay timer

The circuit diagram for the same function but with a timer relay looks like this:

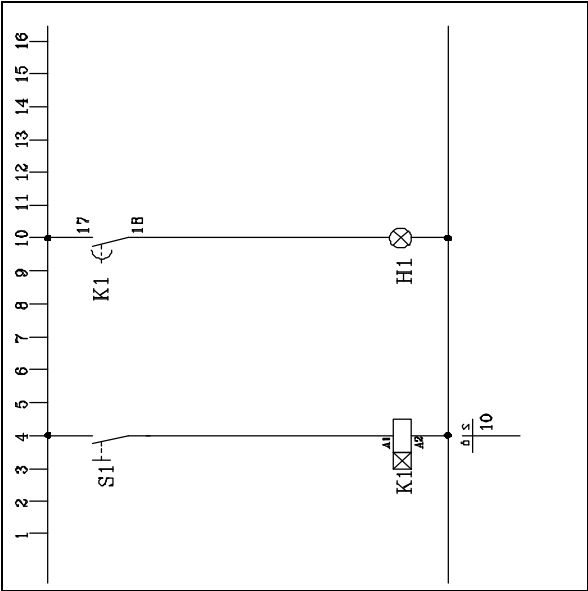


Fig. 8/11: Circuit diagram

8. Programming timers

Programming using T

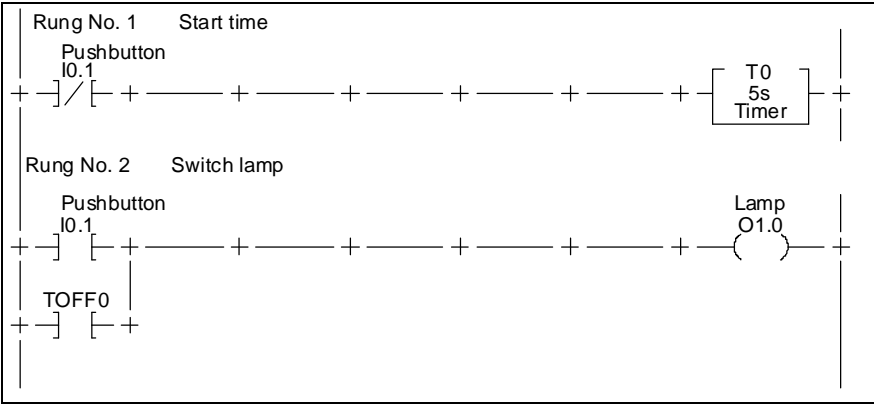


Fig. 8/12: Start time/switch lamp

8. Programming timers

Flashing lamp

A lamp is to flash on and off at one second intervals as long as the push button is pressed.

Programming using TON

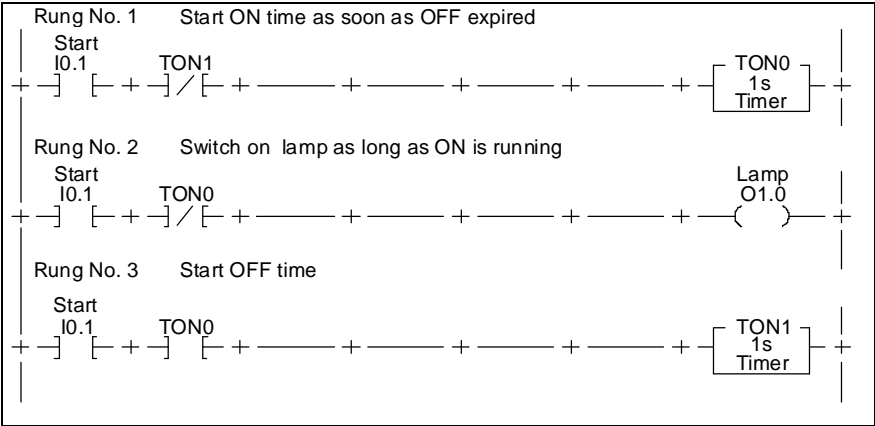


Fig. 8/13: Flashing lamp – programmed with TON

8. Programming timers

Programming using TOFF

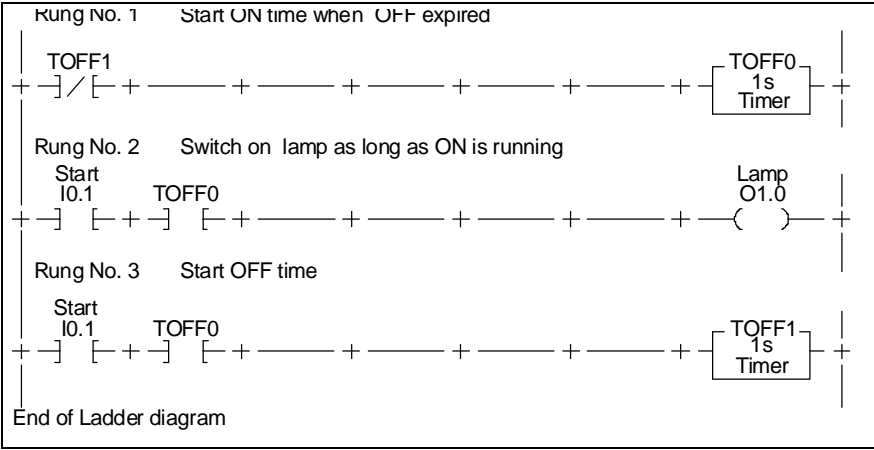


Fig. 8/14: Flashing lamp – programmed with TOFF

8. Programming timers

Programming using T

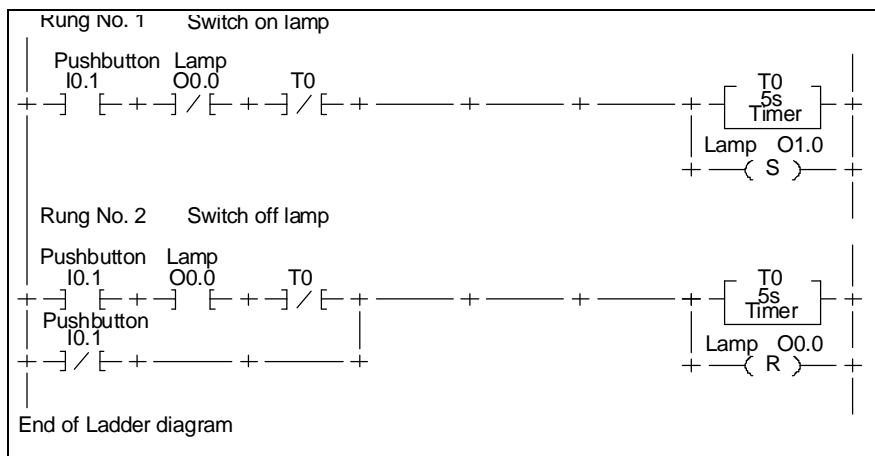


Fig. 8/15: Flashing lamp – programming with T

8. Programming timers

8.4 EVALUATING THE INDIVIDUAL TIMER ELEMENTS

All of the previous examples assume that the timer module consists of two elements: coil (to start the time) and contact (to interrogate the time). In practice however the structure of timer modules offers other possibilities.

Each timer module consists of a timer word, timer preselect and timer bit.

Timer word:

The time is actually counted in the timer word. The timer word is the counter that is decremented by 1 for each 0.01 second clock pulse. The timer word is the only multi-bit operand in a Festo PLC that cannot be used freely as a multi-bit operand, as it is always counting the clock pulses.

Timer preselect:

The timer preselect is a word into which the initial value of the timer module is stored. It therefore represents the period of the timer module. The timer preselect is retentive in all Festo controllers, that is, the contents remain unchanged when the power is switched off (if the back-up battery is functioning). The timer preselect can be calculated, counted, converted etc. like any other multi-bit operand.

8. Programming timers

Timer bit:

The timer bit can be interrogated for 1 and 0 like any one-bit operand, but can also be set and reset.

Interrogation for logic 1 and logic 0 results in an answer as described under the pulse timers, switch-on delay timer and switch-off delay timer.

SETTING to logic 1 causes restart of the timer using the period stored in the timer preselect. RESETTNG to logic 0 stops the timer as if the time had expired.

A good understanding of these individual elements enables very flexible and convenient timer programming. For example, the previous example of the flashing lamp could also be implemented with the aid of a comparative interrogation of the timer word:

8. Programming timers

Programming flashing lamp with comparison

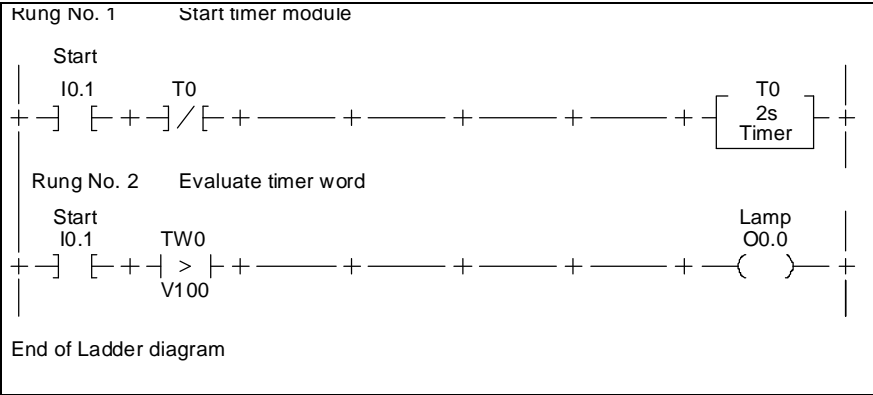


Fig. 8/16: Start timer module/evaluate timer word

Here the timer is set for 2 seconds and re-started as soon as it expires. The timer is interrogated to see if the timer word is "greater than 100", that is whether the timer word is counting from 200 to 100 (this takes exactly 1 second). As soon as the timer word reaches 100, the lamp is switched off.

The possibility of directly accessing all elements of a timer module allows any time construction to be programmed. The only limitation is your imagination.

9. Programming counters

Chapter 9

Programming counters

9. Programming counters

Contents

9.1	WHAT IS AN INCREMENTAL COUNTER?	184
9.2	ELEMENTS OF THE COUNTER	186
9.3	PROGRAMMING COUNTERS.....	191
	Programming incremental counters	191
	Programming decremental counters	193
	Programming a car parking system.....	195

9. Programming counters

Festo PLCs can all be programmed with incremental or decremental counters in Ladder diagram.

In addition, each element of the counter module can be accessed directly.

Similarly, all word operands with the **exception of input words and the timer word** can be used for counting. These are then not incremental counters but event counters whose value is compared to other values.

9. Programming counters

9.1 WHAT IS AN INCREMENTAL COUNTER?

Normally we speak of an incremental counter when we are using a counter that:

- Counts pulses and stores the result in its own memory (actual value).
- Offers the possibility of setting the actual value to an initial value (reset input).
- Has its own memory for a preselect (nominal value).
- Has a result output that can be interrogated to see whether the actual value has reached the nominal value.

9. Programming counters

In practice this could be:

A machining centre which is to manufacture 14,000 screws of a particular type.

The preselect (nominal value) is set to 14,000.

At the beginning of the operation, the operator (or an automatic mechanism) sets the counter to 0.

The counter is reset.

During the operation the counter is incremented by 1 for each screw. The result is shown on the display. The counter is reset.

The incremental counter shows the number of parts manufactured.

As soon as 14,000 screws have been manufactured, the machine is stopped. A new type, possibly with a new nominal value is entered.

A reaction takes place when nominal and actual values are equal.

9. Programming counters

9.2 ELEMENTS OF THE COUNTER

In order to allow flexibility in counter programming, Ladder diagram for a Festo PLC offers the following operands for counters:

The counter word is the operand in which the actual value is stored and is changed with each counting pulse. The counter word must be reset at the beginning of the counting operation. Counter words are retentive in all Festo PLCs (as long as the buffer battery is working).

The counter preselect is the operand in which the nominal value is stored.

The counter bit returns:

Logic 0:

The counter has expired or reached the preselect or reached the value 0.

Logic 1:

The counter is still counting, the preselect has not yet been reached.

Compared to an electromechanical counter, the counter bit is a normally closed contact (not a normally open contact or a changeover contact), with which the count can be output.

9. Programming counters

The counter bit is retentive in all Festo PLCs (as long as the back-up battery is working). This means that all counts are retained even in the case of a power failure.

The following **operations** facilitate use of the operands:

Counter initialization is called as a box instead of a coil. The counter number is entered as the operand and the number to which the counter is to count. This number is automatically loaded into the counter preselect (nominal value), the counter bit is set to logic 1 (the normally closed contact remains closed).

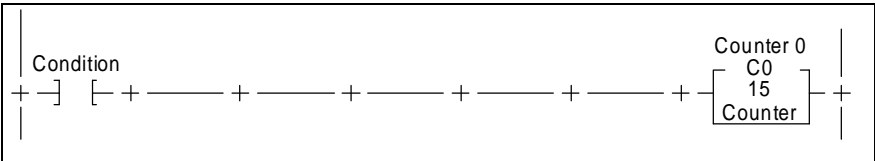


Fig. 9/1: Counter initialization

Counter set is a simple coil command. When this command is executed, the counter word (actual value) is set to 0 and the counter bit to logic 1 (the counter is active).

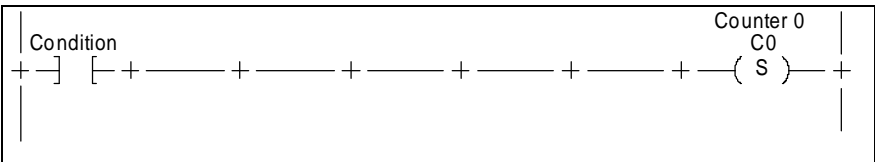


Fig. 9/2: Setting a counter

9. Programming counters

Counter reset – like set – is a simple coil command. If this command is executed, the counter bit has the value logic 0.

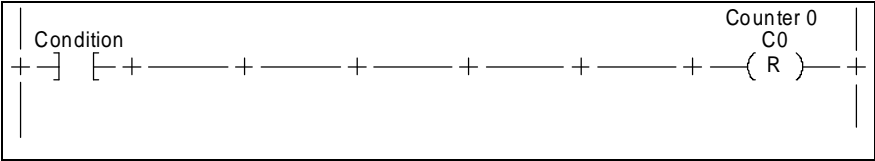
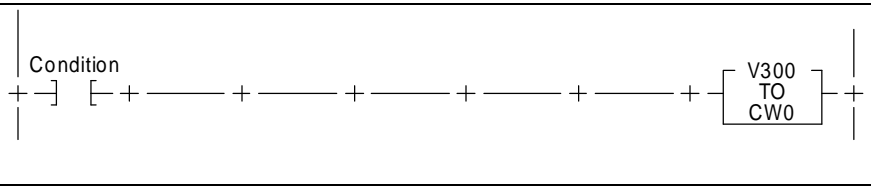


Fig. 9/3: Resetting a counter

Load counter word is a coil operation (in the executive part). The load counter word command loads the counter word (actual value) with a number and sets the counter bit to logic 1 (the normally closed contact opens). The counter bit reverts to logic 0 (the normally closed contact closes), when 0 is reached



(decremental counter).

Fig. 9/4: Loading a counter word

9. Programming counters

The counting operation is a typical executive operation and is entered in the coil of a Ladder diagram. When using INC (increment) the counter is incremented by 1 and when using DEC (decrement) the counter is decremented by 1.

Important: Only the signal edge is counted, not the continuous signal.

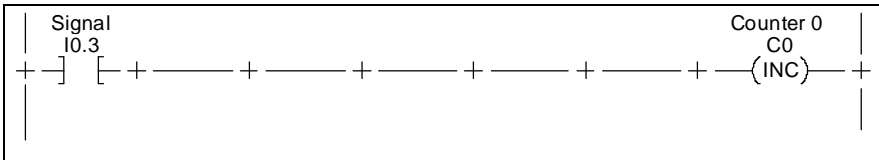


Fig. 9/5: Counting

Interrogating the counter for 1 is a contact operation. This condition is fulfilled (the normally closed contact is closed) when the counter is still counting, i.e. the preselect has not yet been reached.

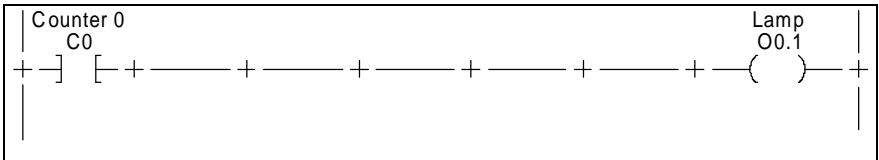


Fig. 9/6: Interrogating a counter for logic 1

9. Programming counters

Interrogating a counter for 0 is the opposite contact operation. The condition is fulfilled (the normally closed contact is opened) if the preselect has been reached, i.e. the counter is no longer counting or if the counter has reached the value 0.

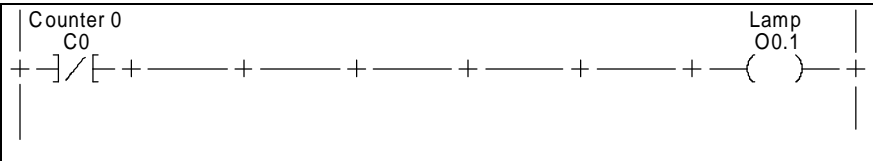
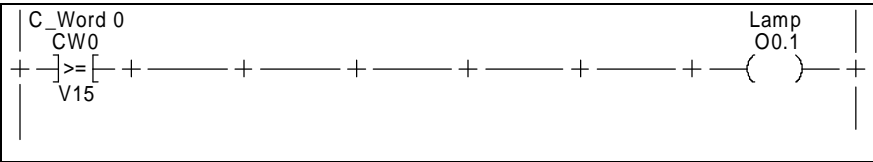


Fig. 9/7: Interrogating a counter for logic 0

Compare counter word or counter preselect are boxes which replace a contact. The actual value (counter word) or the nominal value (counter preselect) is compared with a number (constant) or with another word operand, in-



cluding other counter words.

Fig. 9/8: Compare counter word or counter preselect

9. Programming counters

9.3 PROGRAMMING COUNTERS

Programming incremental counters

The incremental counter is probably the most commonly used type of counter. The following program counts the signals at input I0.3 (symbolic name: Signal). After 10 signals, output O0.1 (symbolic name: Lamp) is switched on. A signal at input I0.1 (symbolic name: Reset) resets the current value of the counter to 0 and sets the counter bit back to logic 1.

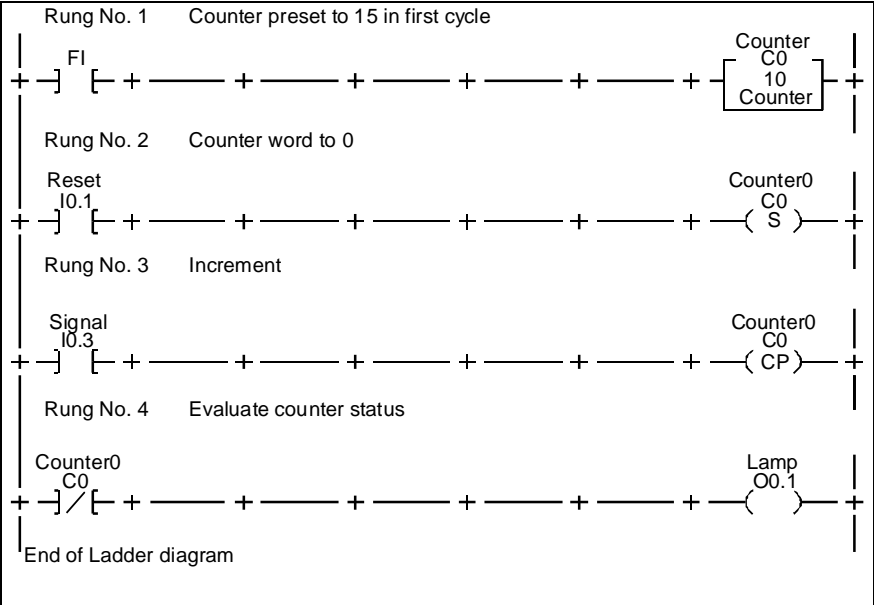


Fig. 9/9: Programming an incremental counter in LDR

9. Programming counters

Note: The first rung initializes the counter immediately after power-up (with FI). This is necessary because the interrogation in rung 4 whether the counter is "inactive" would otherwise be fulfilled immediately after power-up (for the FPC 405). The counter would only be set to logic 1 (and therefore the lamp out) after the first reset command (rung 2).

The circuit diagram for an electromechanical counter for the same function as the above program looks like this:

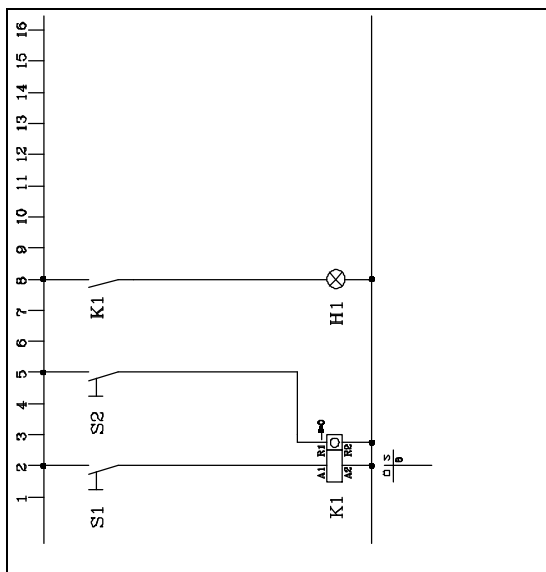
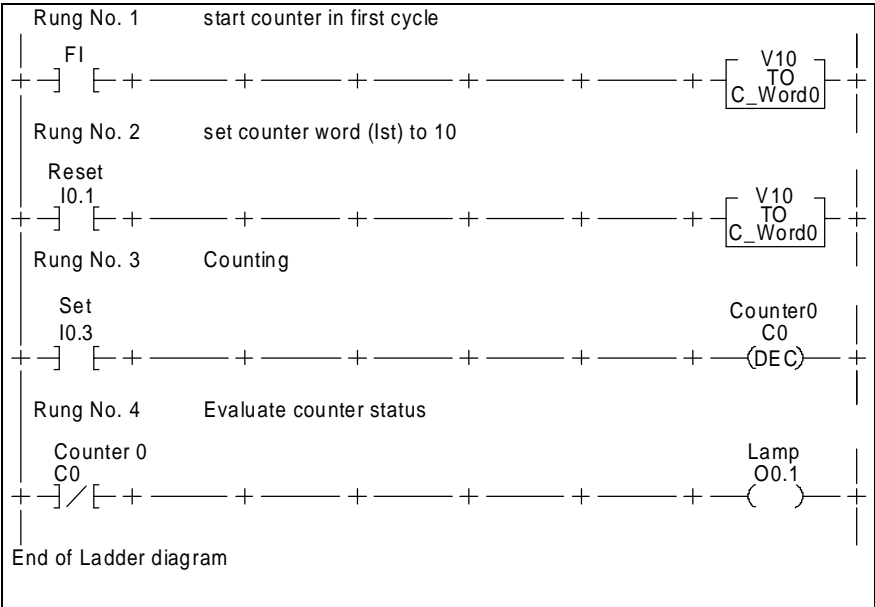


Fig. 9/10: Current diagram for incrementer

9. Programming counters

Programming decremental counters

In the case of the decremental counter, the actual value (counter word) must not be set to 0, but must start at the desired number. The counter bit then becomes logic 0 as soon as 0



is reached.

Fig. 9/11: Programming decremental counter in LDR

9. Programming counters

Note: *As the counter is a decremental counter, it cannot be reset by simply using the set command. A box (the Arithmetic/Logic Box can also be used) must be used to load the start value into the counter word.*

Furthermore, in this example the initialization flag ensures that the counter is active immediately after power-up of the controller. Otherwise the condition in rung 4 would be met immediately after power-up, and the lamp would be switched on.

9. Programming counters

Programming a car parking system

The car park is a classic example of this type of counter:

A car park has space for 10 cars. In front of the car park is a traffic light. The traffic light shows green as long as there is still space in the car park (and the counting system is working). The traffic light shows red as soon as the car park is full.

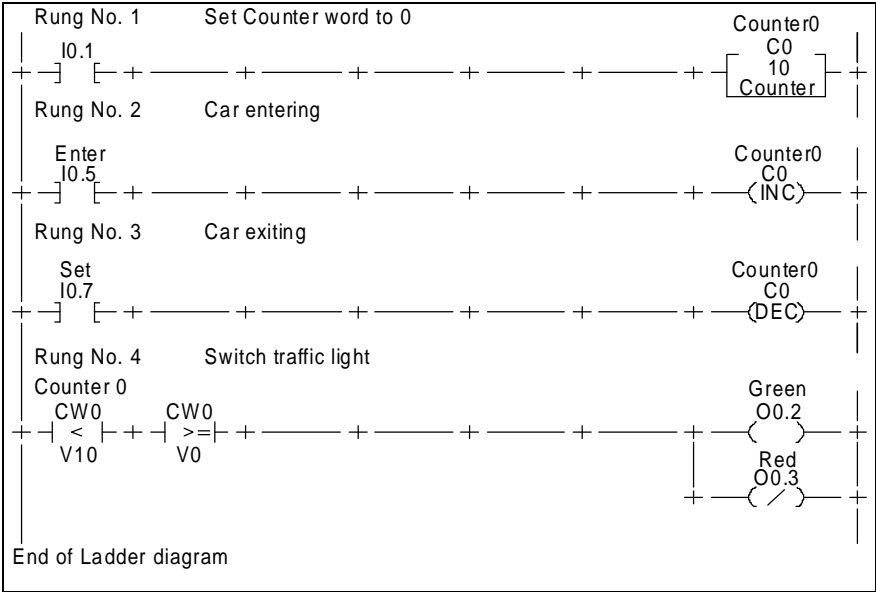


Fig. 9/12: Programming of car parking facility in LDR

9. Programming counters

Please note: *The only purpose of rung 1 is to set the counter to 0 if necessary (for example, if the counter still contains a count from a test, the controller is brand-new or the battery has just been replaced).*

As only counting commands and a comparison are used there is no reason to use a counter here. One could just as well use a flag word, a register, an output word etc. (anything except an input word).

10. Using registers

Chapter 10
Using registers

10. Using registers

Chapter 10

Using Registers 199

10. Using registers

In the Festo controllers registers are 16-bit word operands. Their contents can therefore be between 0 and 65,535 (unsigned), between 0 and 32,767 (signed), between \$0 and \$FFFF (hexadecimal) or between 0 and 9999 (BCD).

Arithmetic operators (+, -, /, *) assume unsigned numbers unless the FPC 405 is switched to signed numbers with CFM20.

In the case of comparison operations (=, >, >=, <, <=, <>) unsigned numbers are assumed unless the FPC 405 is switched to signed numbers with CFM20.

All bit-manipulating operations (ROL, ROR, SHL, SHR, INV, CPL) manipulate all 16 bits of the registers.

All bit-comparison operations (AND, OR, EXOR) compare all 16 bits.

A single bit cannot be accessed directly. If, for example, the 6th bit of a register is to be set to logic 1 it must either be OR masked or the register must be loaded into a flag word, the bit manipulated and the contents of the flag word reloaded into the register.

10. Using registers

All registers of Festo controllers (except for the FEC) are retentive, as long as the back-up battery is functioning.

For the FEC only registers 0 ... 127 are retentive.

Register 63 of the FPC 100B/AF cannot be used for general register applications. This register serves only for setting the input signal delay for the input of the FPC 100B/AF (see Appendix B.8).

11. Using flags

Chapter 11

Using flags

11. Using flags

Contents

11.1 FLAG BIT	203
11.2 FLAG WORD	207

11. Using flags

11.1 FLAG BIT

Flags fulfill the same function as the auxiliary relay in the circuit diagram: a relay that is not switched externally but only fulfills a function within the controller.

Flags can be used as outputs, except that outputs have an external connection and an external LED, where as flags are only present and visible within the system.

All operations that can be used on outputs can also be used on flags, for example assignment, setting, resetting, interrogating etc.

Flags are differentiated from outputs by the following points:

- Every Festo controller has retentive (non-volatile) flags whose status is retained after power loss or power
- failure (assuming the back-up battery is functioning).
- The status of the flag remains unchanged even if the appropriate processor group is switched to stop.
Outputs, on the other hand, are set to logic 0 (depending on the controller type).

11. Using flags

Like outputs, flags are organized as words with 16 bits each. For this reason an absolute flag address always consists of an indication of the flag word and the flag bits, for example:

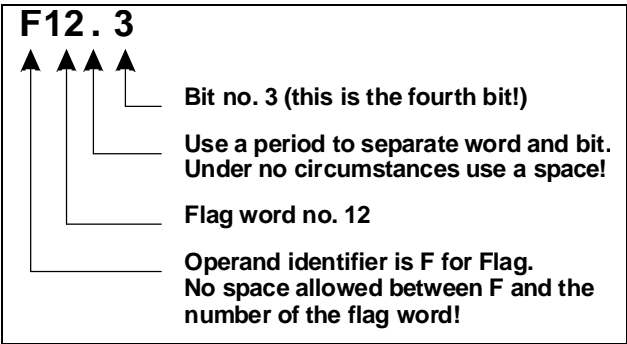


Fig. 11/1

11. Using flags

In the case of the multiprocessor controller FPC 405 the processor module can be indicated in addition to the flag word. Each program in each processor module can access the global flag of other processor modules by including the processor module number in the absolute address, for example:

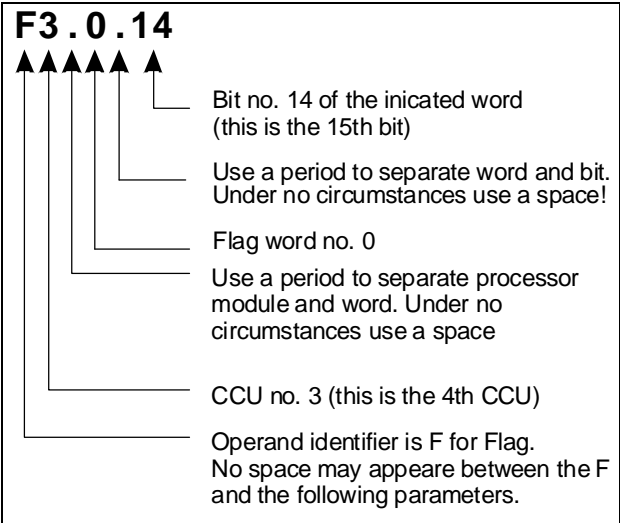


Fig. 11/2

11. Using flags

In the case of the FPC 405 there are an additional 8 flags for local use within the processor module. These flags are only available for the local program and are always set to logic 0 when the program is started. The flags are addressed as FI0 to FI7. FI0 to FI5 are freely usable within the program.

FI6 becomes logic 1 if a mathematical operation using CFM20 results in an overflow.

FI7 is used for the storage of a bit shifted out in a SHIFT operation. It can be interrogated. As this flag is a local flag it is not entered in the allocation list. The allocation list belongs to the project and can only manage local operands that belong to a processor module.

11. Using flags

11.2 FLAG WORD

As flags are organized in words of 16 bits, these words can also be addressed as words. In the absolute address the bit number and the period before it are not required. The operand identifier is FW for flag word, for example:

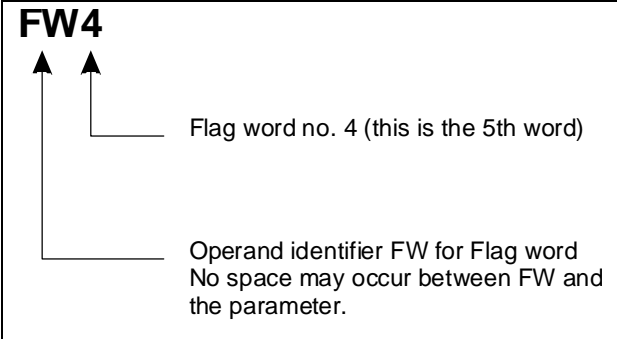


Fig. 11/3

11. Using flags

Like flag bits, flag words of remote processor modules can be manipulated when using a multi-processor system (FPC 405). In this case the CCU number is added to the absolute address, for example:

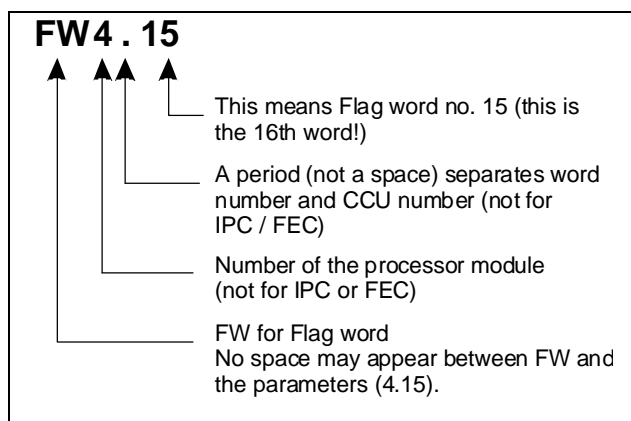


Fig. 11/4

Flag words can be manipulated with all operations used for words: Interrogation operations are comparison operations (=, >, >=, <, <=, <>); LOAD TO is an example of a word-manipulating operation; all arithmetic operations (+, -, *, /); the bit-manipulating operations AND, OR, EXOR, CPL, INV; and the counter operations INC and DEC (increment and decrement).

Appendix A - Example programs

Appendix A

Example programs

Appendix A - Example programs

Contents

A.1	SEQUENCE PROGRAMS	211
	Sequence program with existing flag chain, with storing commands	214
	Sequence program with existing flag chain and latching.....	215
	Sequence program with resetting sequence	217
	Sequence program with counter.....	219
A.2	MEASURING THE CYCLE TIME	220
	Number of cycles every 3 seconds ..	221
	Absolute duration of a cycle in 0.1 ms	223
	Measuring reaction time	225
	Measurement and display of the absolute reaction time.....	227
	Display of the longest cycle time	229
A.3	STRUCTURED PROGRAMMING IN LADDER DIAGRAM	233

Appendix A - Example programs

A.1 SEQUENCE PROGRAMS

Research shows that sequence programs occur in 80% of all automation systems. In most cases where programmable logic controllers are used a sequence program is likely to be at the heart of the system.

Using a simple and very well known example it will be demonstrated how a sequence program can be programmed using Ladder diagram. A "bedside lamp controller" is to be programmed. Press the pushbutton → lamp on; press the button again → lamp off.

In your bedside lamp the problem is probably solved mechanically. The switch has a mechanism that causes the current to be switched on and off. The electrician knows the problem of the hall way light. There a latching relay is used which also solves the problem mechanically.

What does the diagram of the bedside lamp control look like?

Appendix A - Example programs

The function diagram in accordance with DIN 40719 part 6 has the following structure:

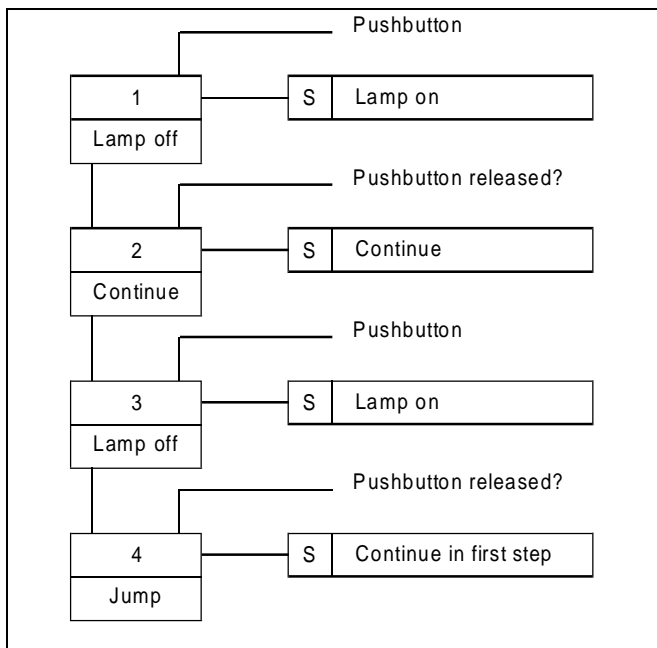


Fig. A/1: Function diagram to DIN 40719

There are several ways in which a sequence program can be programmed in Ladder diagram. Here, four **systematic** methods will be presented. The aim is the systematic development of a sequence program, not the shortest, most elegant or most popular means. If you don't have your own "preferred method", select the one that you like best. If you have to write a sequence program, write the basic structure, copy the rungs with which the steps are to be constructed, for as many steps as required and complete the rest. That's it!

Appendix A - Example programs

You should consider the counter method only if you have to program longer sequences. It may be that your controller has too few flags available. The FPC 100, for example, has 256 flags available. If your sequence has 150 steps, the number of flags required could be a problem.

Please note that in all examples the programs are such that after power-up the controller begins at the first step (even after a power failure). In order to ensure this even when retentive flags or a retentive counter word are used, the first step in the rung is always a reset with the aid of the initialization flag FI.

Please also note that the following program examples often use operand designations. The absolute designations are automatically added so that the symbolic designators act as a type of commentary in the program.

Appendix A - Example programs

Sequence program with existing flag chain, with storing commands

This programming method assumes that each step requires a flag. An additional flag is switched on (this is the reason for the name "existing flag chain", as in the last step all flags are "existing"). The outputs are storing and directly connected to the appropriate flags.

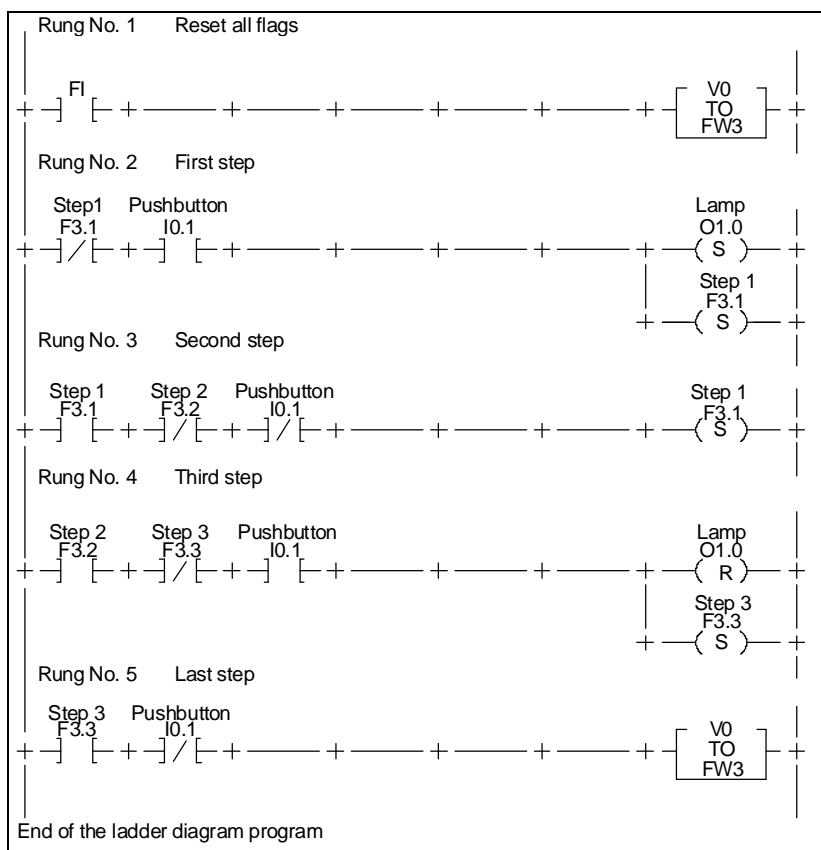


Fig. A/2: Ladder diagram

Appendix A - Example programs

Sequence program with existing flag chain and latching

This programming method is similar to the existing flag chain with storing commands. The difference is that no storing commands are used but latching. This method requires the separation of the step chain from the switching of outputs. The entire Ladder diagram is thus separated into a "Control part" and a "Power part". The control part only contains the step chain, the outputs are switched in the power part.

Appendix A - Example programs

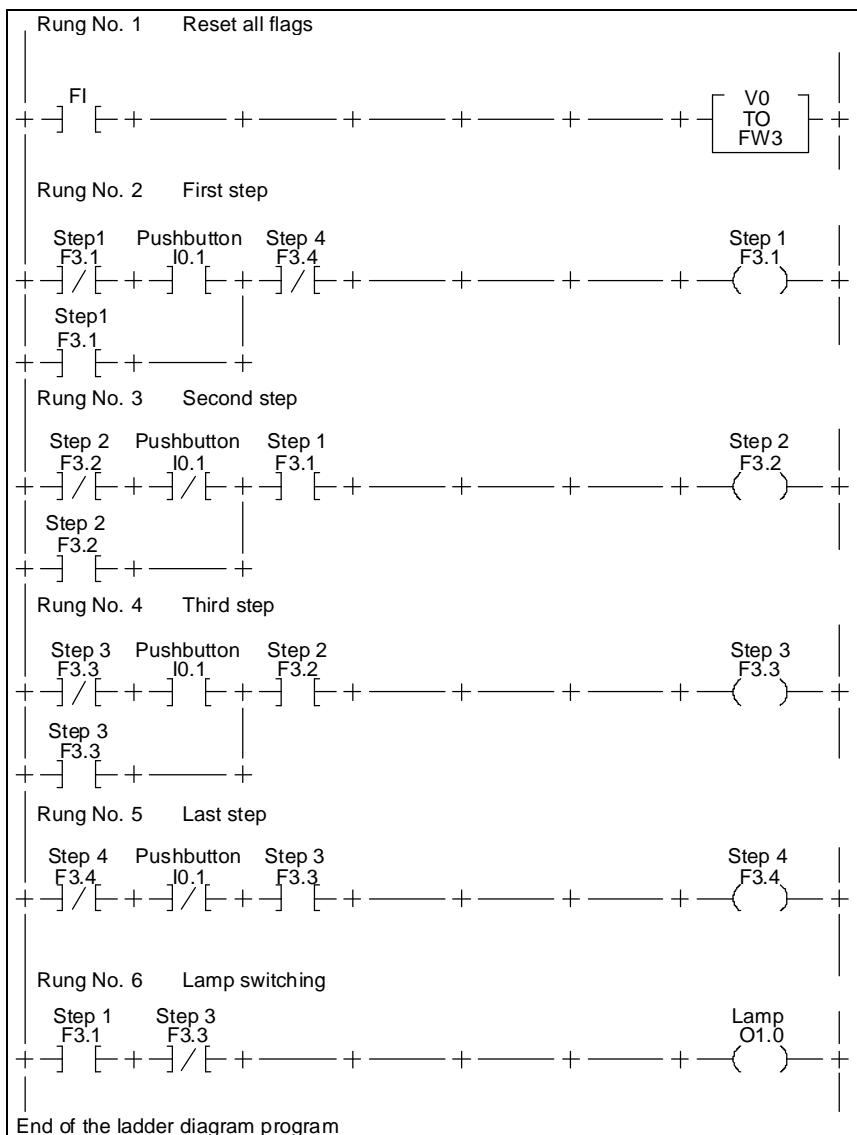


Fig. A/3: Ladder diagram

Appendix A - Example programs

Sequence program with resetting sequence

The resetting sequence is very popular in relay technology and has therefore been adopted by many electricians in Ladder diagram programming.

Here too the "Control part" and "Power part" are separated (not entirely necessary) and (in this programming example) programming is done entirely with latching.

Please note in the case of the resetting sequence that at the start of the sequence the last step must be set (if necessary a step without action). For this reason the initialization flag is used in the first rung to load value 16 into the flag word. This sets flag F3.4.

Appendix A - Example programs

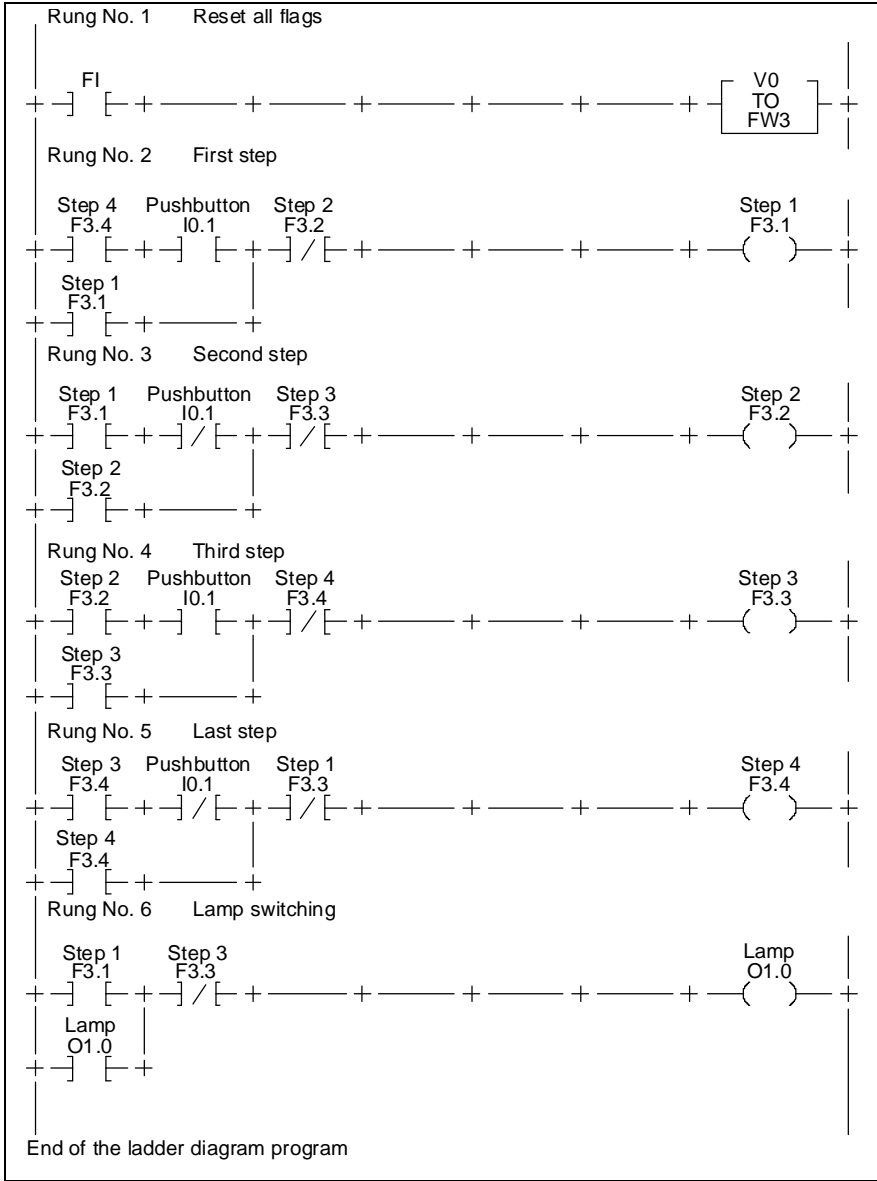


Fig. A/4: Ladder diagram

Appendix A - Example programs

Sequence program with counter

Programming of a sequence with a counter is totally unknown in relay technology as a counter cannot be interrogated for its current status. On the other hand the counter method allows significantly more flexible programming – particularly jumps can be very elegantly solved.

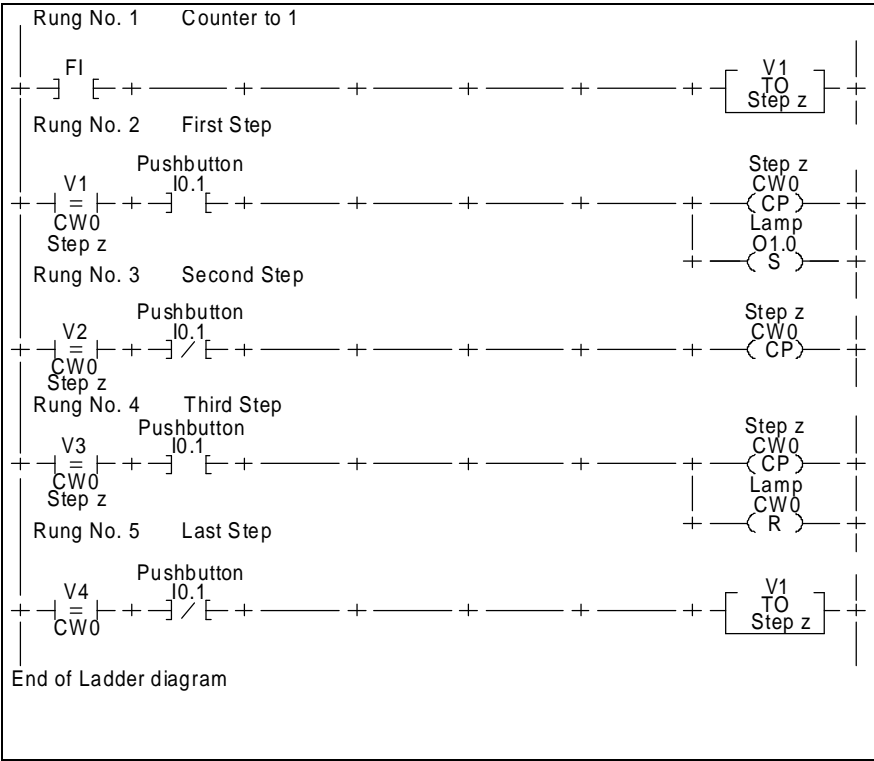


Fig. A/5: Ladder diagram

Appendix A - Example programs

A.2 MEASURING THE CYCLE TIME

It is often interesting to know the cycle time of a processor module. Although Festo gives data on the time requirement of individual commands for each controller type, it is almost impossible to calculate the cycle time of a particular program. It is a lot easier to write a small program in which the cycle time is written to a register. With the aid of the on-line function this register can then be examined.

There are only two restrictions that need to be taken into account.

On the one hand the measuring program also has a cycle time. The cycle time is easy to evaluate by doubling the measuring program and comparing the new cycle time with the previous cycle time.

The cycle time increase caused by the measuring programs are given. Deduct the time required for the measuring program from the measured result.

Secondly, data transfer for online mode also adds to the cycle time. But you need online mode to read the result of the cycle time measurement. This is also the reason why you can set the dynamic display in online mode to slower or faster. By changing this display speed you can clearly see the influence on the cycle time. Select a slow display speed when displaying dynamically, otherwise the values will be totally unrealistic.

Number of cycles every 3 seconds

This program counts how many cycles are executed within 3 seconds. The result is stored in register 1 and is updated every 3 seconds.

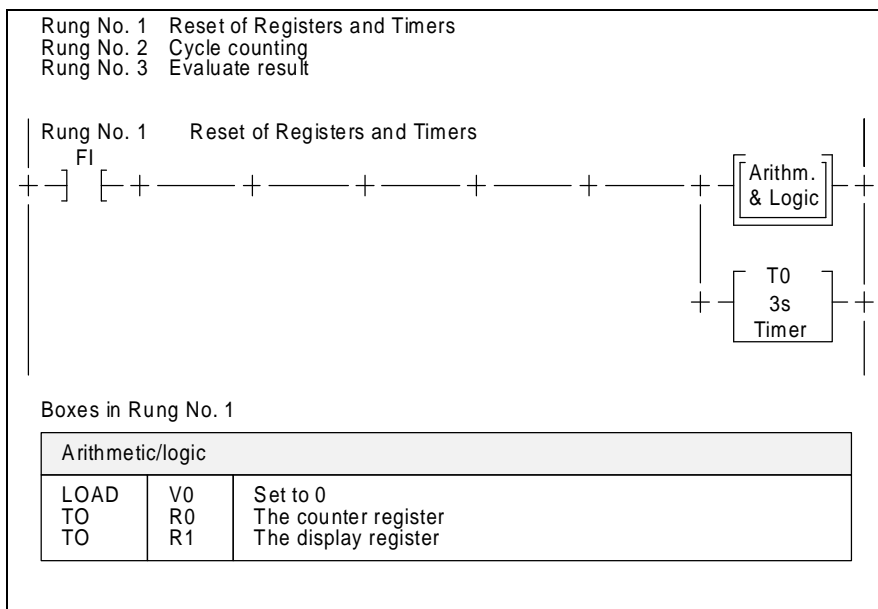


Fig. A/6: Ladder diagram

Appendix A - Example programs

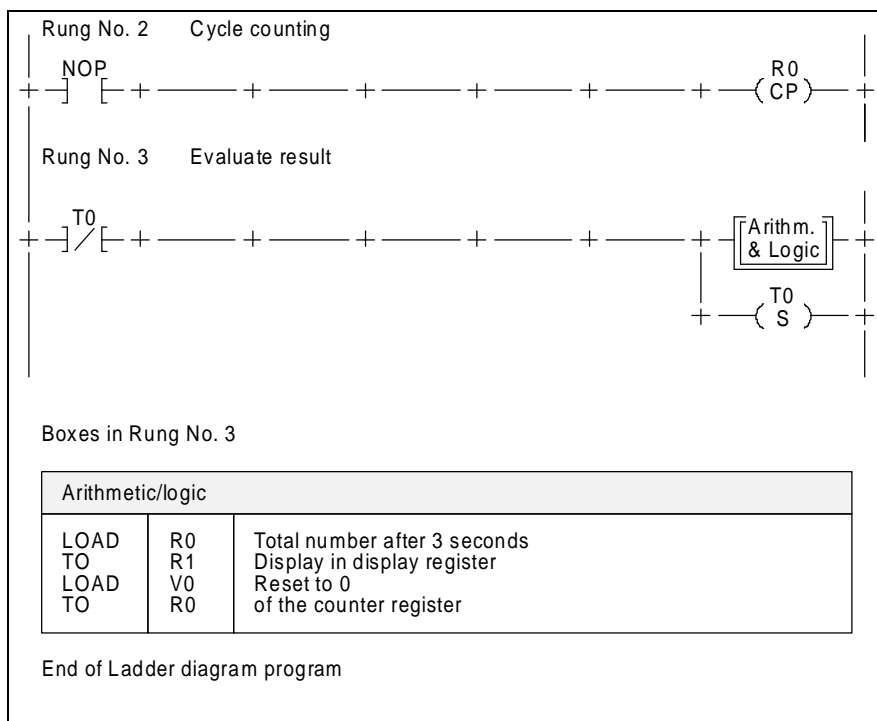


Fig. A/7: Ladder diagram – continued

This measurement program increases the actual cycle time for each processor module as follows:

- FPC 100B/AF by 0.1 ms
- IPC by *** ms
- FEC by *** ms
- FPC 405 by 0.15 ms

Appendix A - Example programs

Absolute duration of a cycle in 0.1 ms

As in the first program, this program counts how many cycles are executed in 3 seconds. This value is then converted into the absolute time per cycle. The value stored in register 2 must be multiplied by 0.1 ms in order to find the duration of an individual cycle.

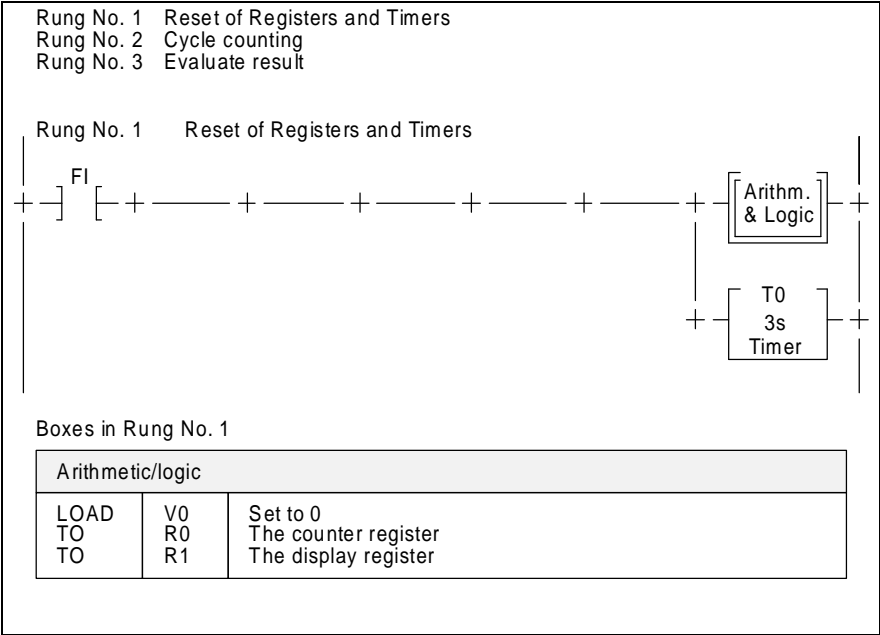


Fig. A/8: Ladder diagram

Appendix A - Example programs

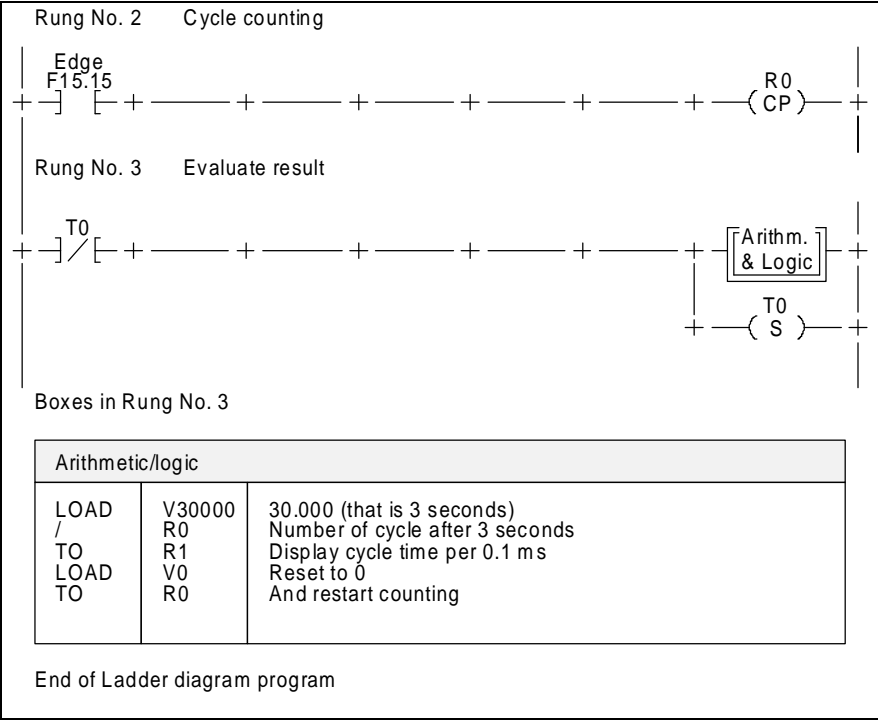


Fig. A/9: Ladder diagram – continued

This measuring program increases the actual cycle time of a processor module by approx. 1.0 ms.

Measuring reaction time

For this reason the following two programs do not use a flag for the edge to be counted but an input signal that is taken from the output of the controller.

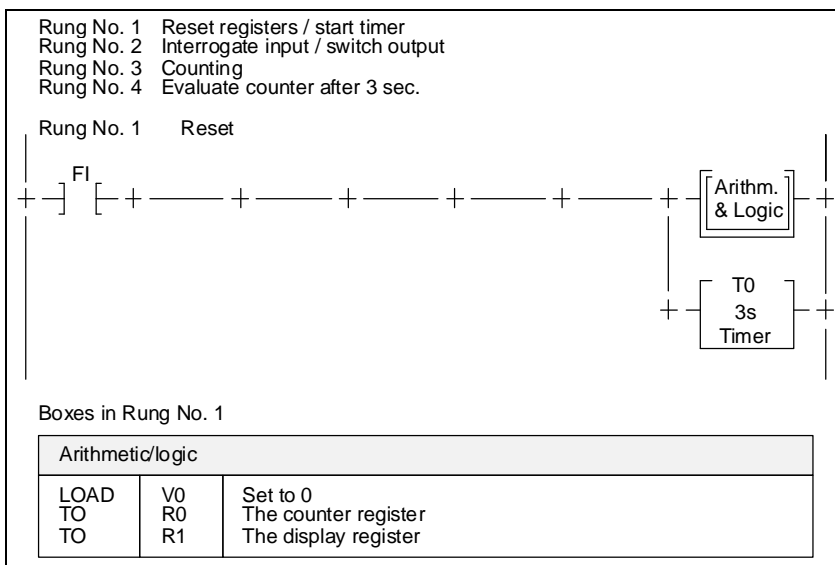


Fig. A/10: Ladder diagram

Appendix A - Example programs

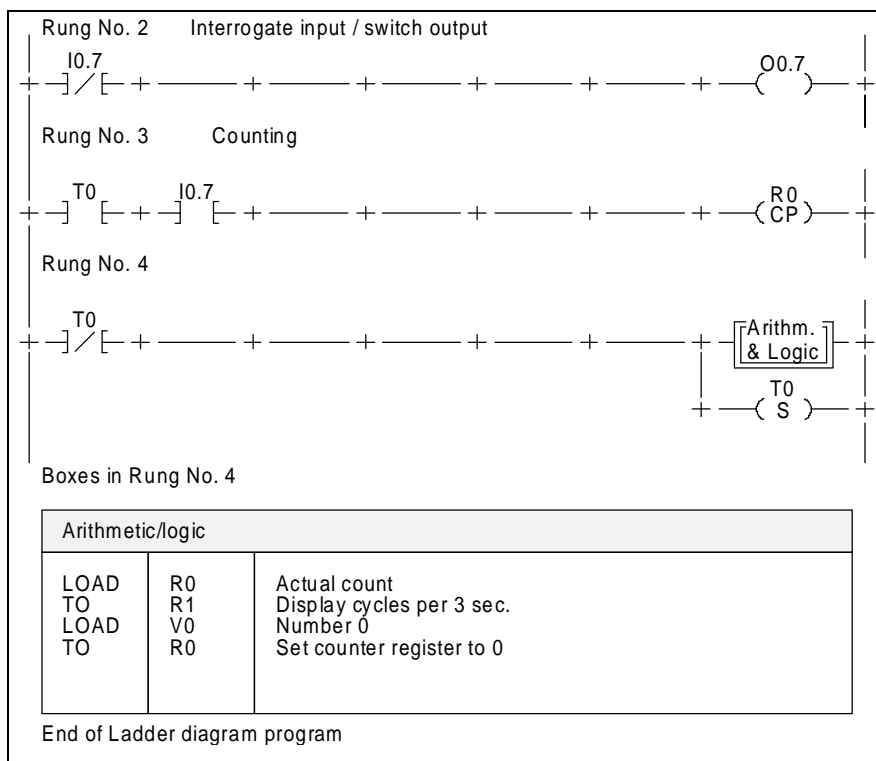


Fig. A/11: Ladder diagram – continued

This measuring program increases the actual cycle time of a processor module as follows:

- **FPC 100B/AF** by 0.2 ms
- **IPC** ***
- **FEC** ***
- **FPC 405** negligible compared to input signal delay

Appendix A - Example programs

Measurement and display of the absolute reaction time

This program measures the reaction time in the same way as the previous program. The result is converted into the absolute time for a reaction cycle, whereby the value in register 2 must be multiplied by 0.1 ms.

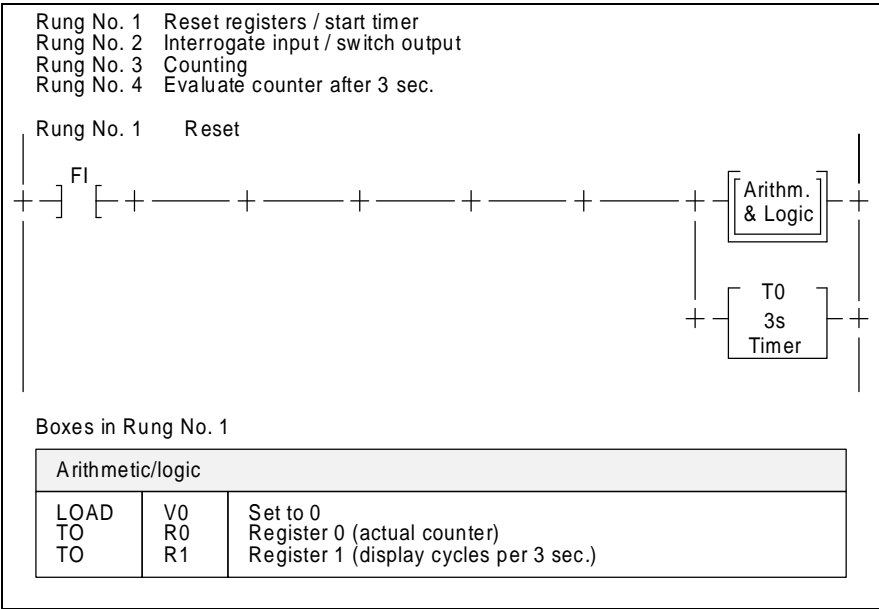


Fig. A/12: Ladder diagram

Appendix A - Example programs

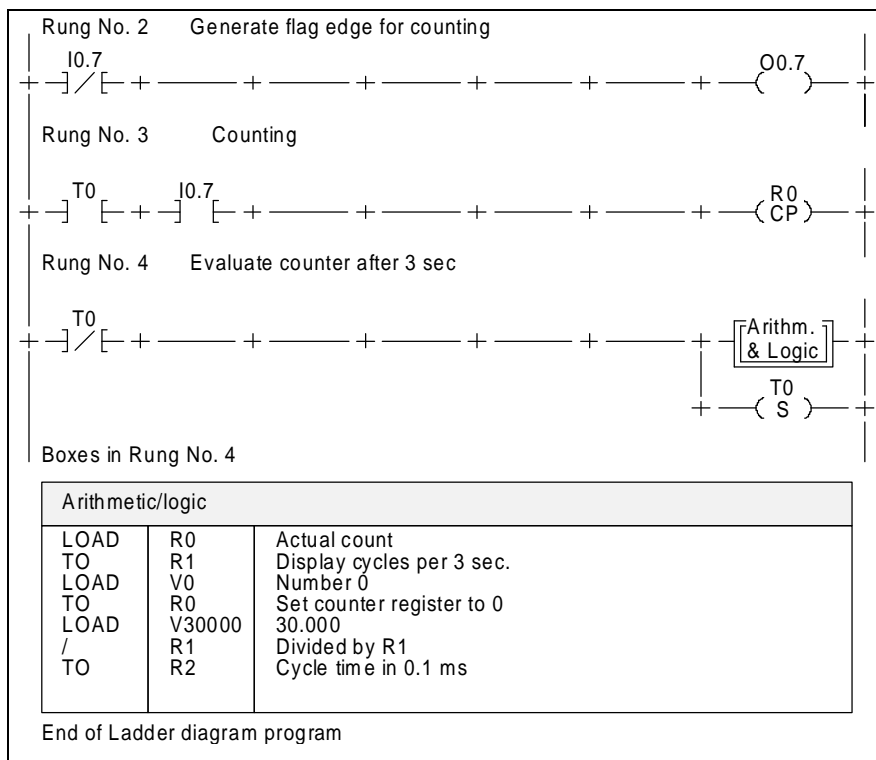


Fig. A/13: Ladder diagram – continued

Appendix A - Example programs

This measurement program increases the actual cycle time of a processor module as follows:

- **FPC 100B/AF** by 0.2 ms
- **IPC** ***
- **FEC** ***
- **FPC 405** negligible compared to input signal delay

Display of the longest cycle time

In many cases the cycle time varies considerably. For example if jump commands or modules are used in a program, if parallel programming of the FPC 405 is used or if the interrupt capability of the FPC 405 is used or if a BASIC program is used for this controller for process visualization or operator guidance, the cycle time can fluctuate considerably. In order to be able to measure the cycle time properly in such cases a register is used for the maximum value. This allows cycle time measurement to run in the background while the machine executes all possible program variants. Later the longest cycle time can be read from the registers.

Appendix A - Example programs

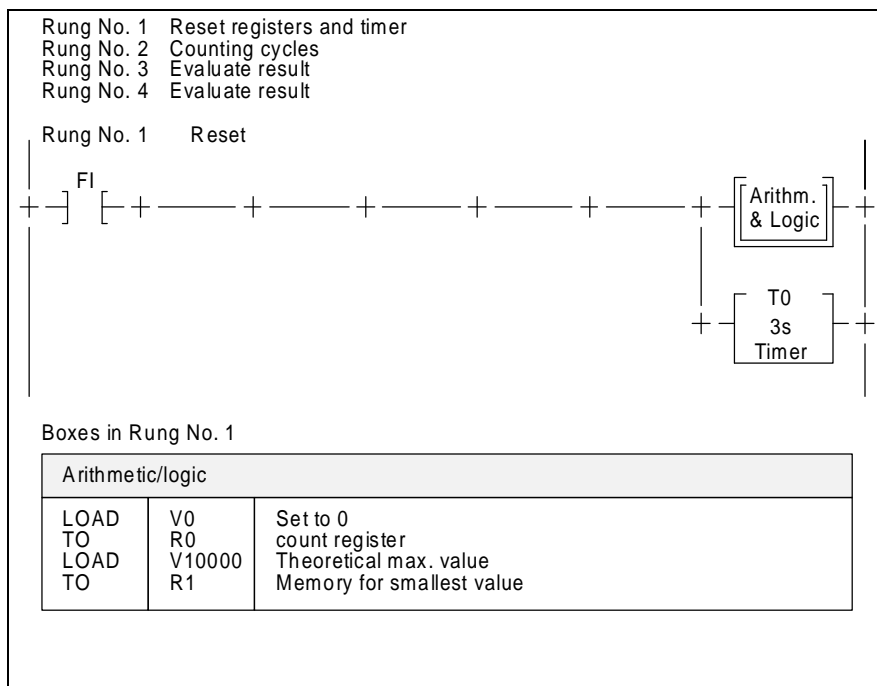


Fig. A/14: Ladder diagram

Appendix A - Example programs

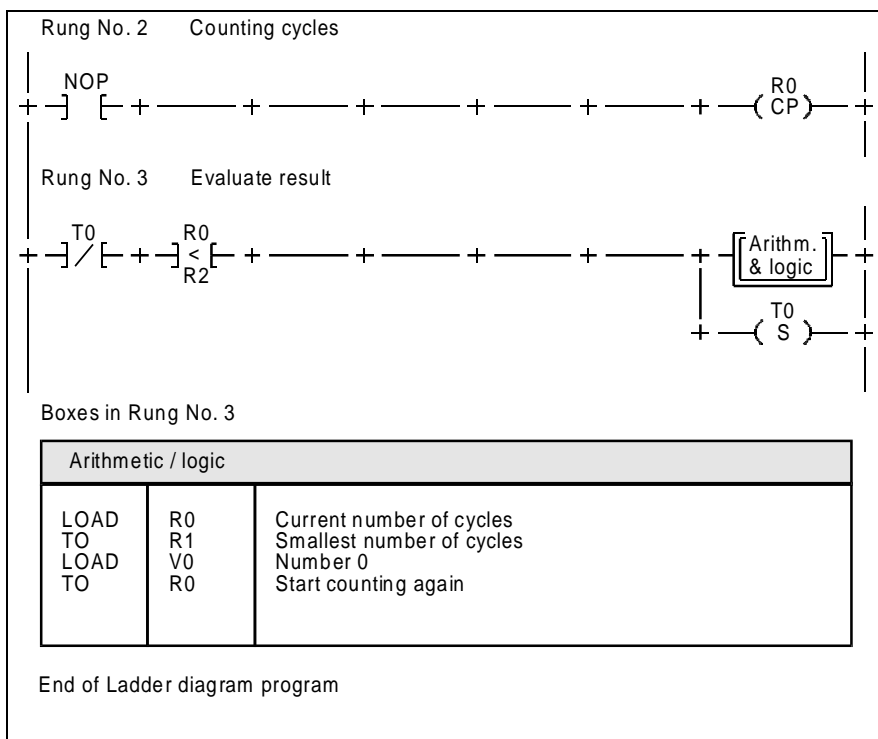


Fig. A/15: Ladder diagram – continued 1

Appendix A - Example programs

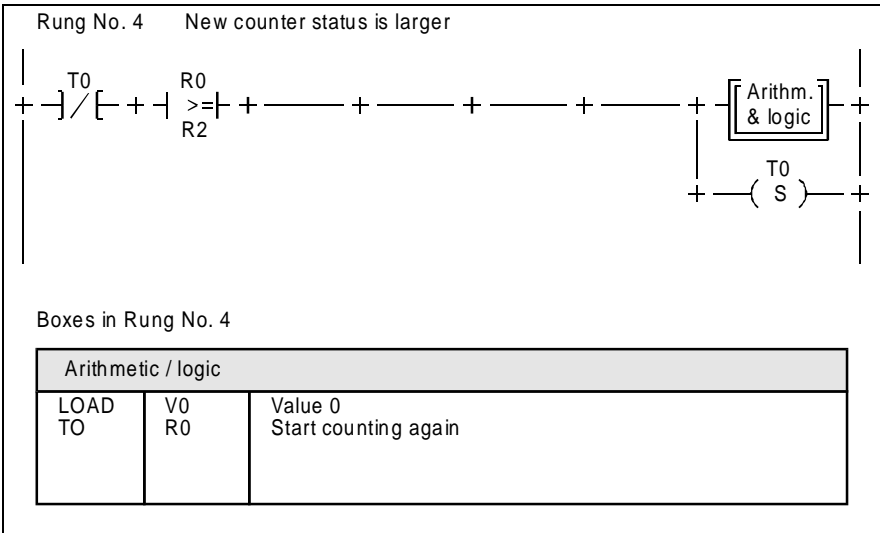


Fig. A/16: Ladder diagram – continued 2

Use of this measurement program increases the actual cycle time for a processor module as follows:

- FPC 100B/AF by 0.24 ms
- IPC by *** ms
- FEC by *** ms
- FPC 405 by 0.30 ms

Appendix A - Example programs

A.3 STRUCTURED PROGRAMMING IN LADDER DIAGRAM

Commonly used control programs for automation technology are very similar in their structure. For this reason a structure for a Ladder diagram program is shown here. This structure is not standardized and is not particularly elegant but helps to make a program easy to write and read. For this reason we suggest:

1st Rung

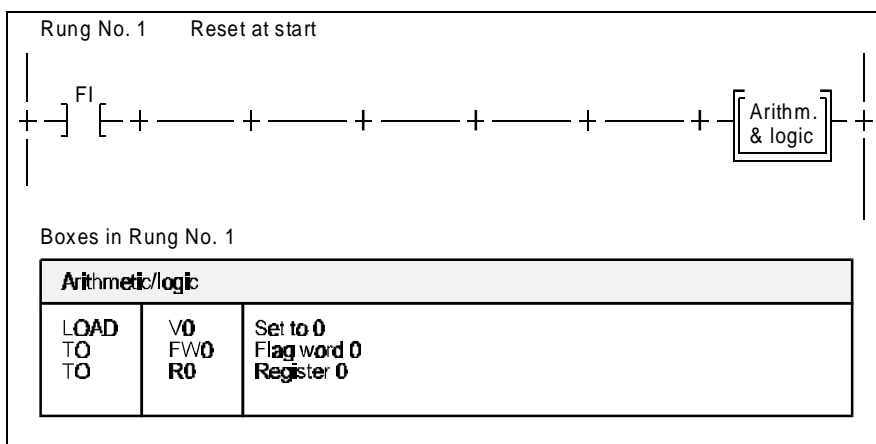


Fig. A/17: Ladder diagram

This rung carries out all settings that need to be carried out **once** when the controller is powered up. Often this means that retentive flags, counters etc. need to be reset, and an output with a ready lamp has to be switched on and that possibly a READY MESSAGE needs to be displayed on a text display.

Appendix A - Example programs

2nd Rung

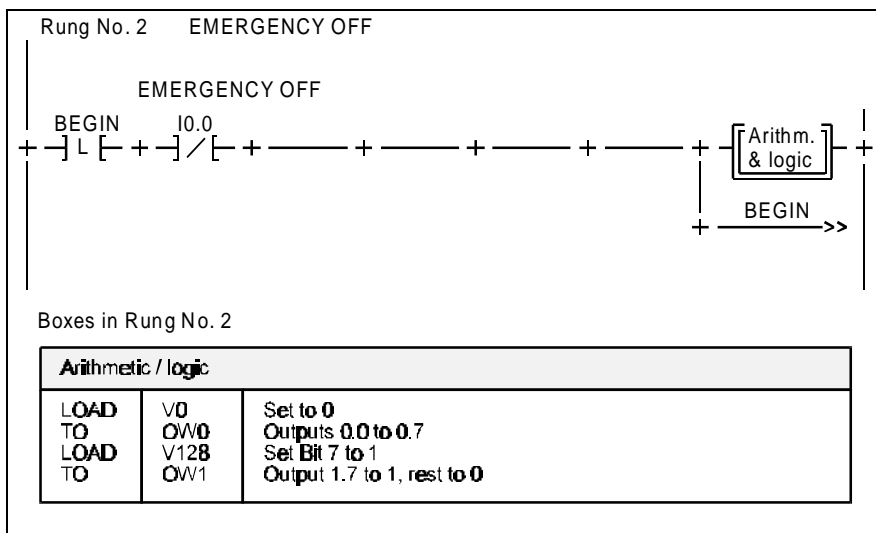


Fig. A/18: Ladder diagram – continued 1

The first entry here is the flag "Begin". This serves as a jump destination from any point in the program.

The second entry is interrogation of the EMERGENCY OFF, the most important error monitoring function. EMERGENCY OFF normally requires (in addition to hardware monitoring of EMERGENCY OFF in accordance with DIN VDE 57 113 part 1, IEC 204-1 or EN 60204 part 1) switching off of nearly all outputs, possibly switching on of a display lamp for EMERGENCY OFF etc. In addition, in this example, the program jumps back to "Begin". Nothing else needs to be done in the event of an EMERGENCY OFF.

Appendix A - Example programs

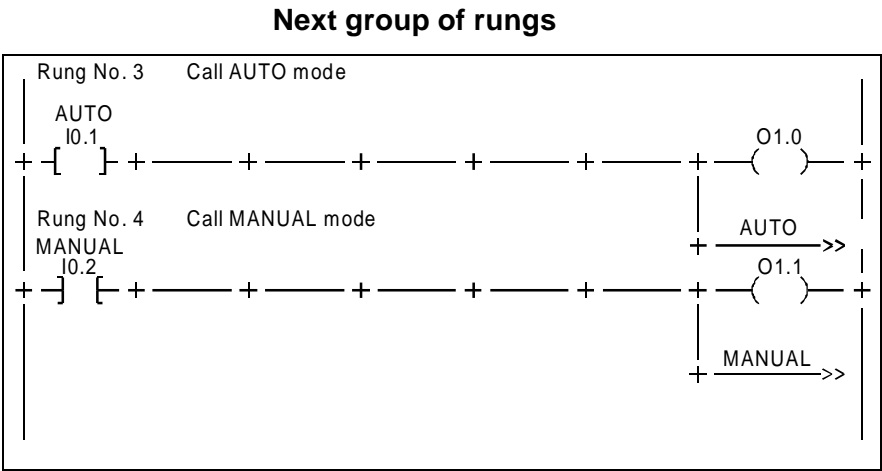


Fig. A/19: Ladder diagram – continued 2

This part of the program allows selection of the automatic/manual mode.
 If necessary, further error messages are monitored here. It is important that operating mode selection takes place in such a way that control can pass immediately to the rung containing the selected operating mode.

Appendix A - Example programs

Group: automatics

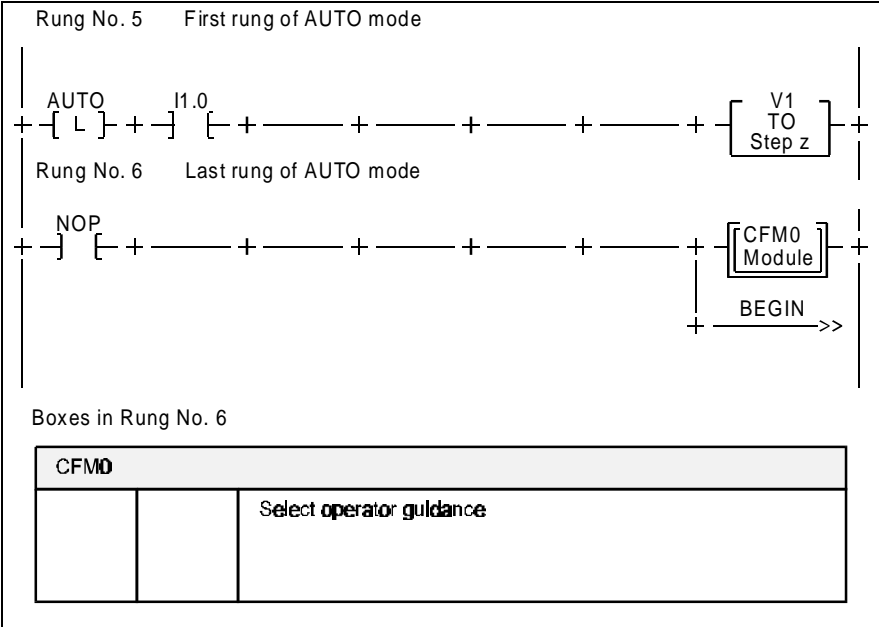


Fig. A/20: Ladder diagram – continued 3

Automatic operation is programmed in this rung group. This is normally a sequence program, for which reason a step counter is used. At the end of this section, control returns to the "Begin" flag.

Appendix A - Example programs

Group: manual

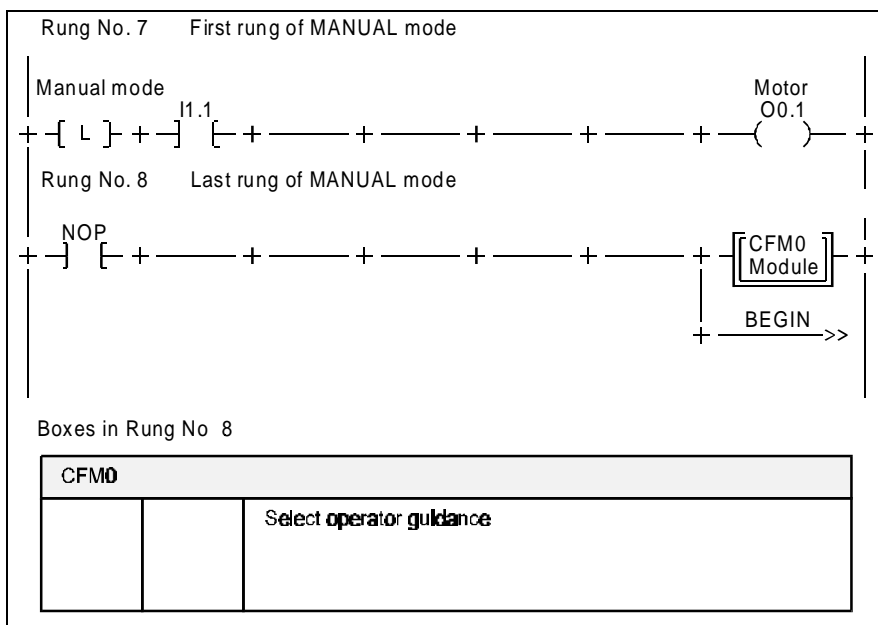


Fig. A/21: Ladder diagram – continued 4

This rung group has a similar structure to "automatic", except that there is no step counter as the manual mode is normally implemented as a parallel logic program.

The last rung of this group also passes control back to the "Begin" flag.

The more clearly a program is structured, the easier it is to find and eliminate errors. Only a well structured program is a help when searching for a defective sensor, a defective valve or a sticking relay.

Notes

[illegible]

Appendix B - Differences in program execution

Appendix B

Differences in program execution

Appendix B - Differences in program execution

Contents

B.1	PROCESS IMAGE.....	241
	Safety routines	242
	Cycle time	243
B.2	MULTITASKING.....	244
B.3	MULTIPROCESSING.....	246
B.4	INTERRUPT CAPABILITY.....	247
B.5	INTERNAL INPUTS AND FLAGS... 248	
	Internal inputs of the FPC 405.....	248
	FPC 405 local flags.....	249
	Local and global allocation list	250
B.6	SERIAL DATA PORT	251
	Hardware	251
	Active/passive interface	252
B.7	BEHAVIOUR AFTER BATTERY FAILURE	253
B.8	INPUT SIGNAL DELAY	255
	FPC 100B/AF	255
	FPC 405	257
	IPC	257
	FEC	257

Appendix B - Differences in program execution

This appendix examines the differences between the controllers offered by Festo. The list does not claim to be complete and is not any strict order. Nonetheless it should provide a useful source of information.

B.1 PROCESS IMAGE

The FPC 100B/AF, IPC and FEC controllers work with a process image of the inputs and outputs. The FPC 405 controller writes and reads directly to the appropriate input or output in each program line.

In practice this means that **before** processing the entire Ladder diagram, the FPC 100B/AF reads the status of all inputs into a memory area and the status of all outputs is read from a memory area to outputs.

This updating of the process image also occurs after each backward jump, so that there are no programs in which the process image is not updated.

If updating of a process image is required at a particular point in the program, a backward jump is inserted.

Appendix B - Differences in program execution

Safety routines

With the FPC 100B/AF, IPC and FEC (that is, with process image) an output is not switched directly as a result of a rung but only prior to the next execution of the program. For this reason it is both possible and useful to write monitoring routines **at the end of the program** which eliminate problems caused by a programming error earlier in the program. It is normal practice to first program rungs that prevent "the worst" from happening: a cylinder may only extend if another one has already retracted; a motor may only be switched on if it is disengaged; the direction of rotation of a motor must be unambiguous etc.

All further rungs are written **before** these last monitoring rungs. If a similar safety routine is to be used with a multiprocessor system (FPC 405), flags are programmed first instead of outputs. **At the end of the program** the flags are then transferred to the outputs taking the safety routines into account.

Appendix B - Differences in program execution

Cycle time

The example programs show several programs that measure cycle time. Where these do not measure reaction time but the pure cycle time, the time taken to execute a complete program is measured. In the case of controllers with process image the reaction time is calculated from the input signal delay, the (comparatively short) output signal delay and **twice** the cycle time. In the case of controllers without process image the single cycle time is sufficient.

The reason for this is that the change of a signal at an input can take place immediately after updating of the process image. The following cycle works with the old logic status. The process image is then updated (after one complete cycle) and the program continues executing with the new logic status and writes the new update to the output during the next cycle (the second cycle is complete).

Appendix B - Differences in program execution

B.2 MULTITASKING

The FPC 100B/AF, FEC, IPC and FPC 405 can store several programs (with different program numbers) in their program memory. Of these stored programs, several programs can be identified as a task and processed simultaneously (quasi-parallel processing).

The controllers support multitasking as follows:

- **FPC 100B/AF:** 1 program and 8 program modules
- **IPC:** up to 64 programs
- **FEC:** up to 64 programs
- **FPC 405:** up to 64 programs can be stored. Of these up to 8 Ladder diagram programs can be processed simultaneously. If only 1 Ladder diagram program is processed, an additional 48 Statement list programs can be processed. For further combinations see the FPC 405 manual.

Appendix B - Differences in program execution

All Festo controllers can process program modules, that is subroutines, created by the user. The controllers support program modules as follows:

- **FPC 100B/AF:** 8 modules
- **IPC:** 100 modules
- **FEC:** 100 modules
- **FPC 405:** 32 modules (can only be written in Assembler if they are to be called from a Ladder diagram program)

All processor modules contain function modules provided by Festo as follows:

- FPC 100B/AF: CMP0 - CMP7
- IPC: CMP0 - CMP99
- FEC: CMP0 - CMP99
- FPC 405: CMP0 - CMP31

Appendix B - Differences in program execution

B.3 MULTIPROCESSING

The FPC 405 allows several processor modules to work together via a system bus. Communication takes place with the aid of flags.

If a program is to be imported from an FPC 405 into FPC 100, FEC or IPC, absolute addressing of flags must be adapted accordingly (in addition to other changes). This is because in the FPC 405 the flag addresses contain the processor module number.

Each processor module can access the global inputs and outputs and global flags via the system bus. The Festo multiprocessor operating system manages bus access to the processor modules.

If two processor modules access the bus simultaneously, the operating system allows access in the order of priority: processor module number 0 (highest priority) to processor module number 5 (lowest priority).

Processor module 0 also tests the system bus on power-up. For this reason, each configuration must contain a CCU number 0. The remaining CCUs can then each be given a unique number in the range of 1 to 5.

Appendix B - Differences in program execution

B.4 INTERRUPT CAPABILITY

In order to allow rapid reaction to external signals, Festo controllers have inputs that allow an interrupt program to be called. This reaction is considerably faster than would be possible with other inputs and normal program execution.

The controllers have interrupt inputs as follows:

- **FPC 100B/AF:** 1 input with max. 3 kHz
- **IPC:** Input module
- **FEC:** 2 inputs with max. 4 kHz
- **FPC 405:** 8 interrupt inputs on the interrupt processor module

Appendix B - Differences in program execution

B.5 INTERNAL INPUTS AND FLAGS

The FPC 405 has so-called internal inputs and internal flags.

Internal inputs of the FPC 405

The interrupt processor module of the FPC 405 has 8 inputs directly on the CCU module.

These 8 inputs are called **internal inputs**, as they are only available to the local processor module. In contrast to these internal inputs, all other inputs can be addressed by any processor module (if they are working together via the system bus).

The internal inputs are fast inputs with an input signal delay of approx. 60 μ s (in contrast to approx. 5 ms for "normal" inputs).

The absolute addresses of these inputs are I0 to I7 or IW.

When using these 8 inputs for "normal" purposes, take into account that there is practically no input signal delay available. For this reason, these inputs must be connected via a shielded cable or an input signal delay must be programmed with a timer module.

Appendix B - Differences in program execution

FPC 405 local flags

The FPC 405 processor modules offer 640 local flags and 384 global flags. The local flags can only be addressed within the local processor module. The global flags can be addressed by any processor module via the system bus.

Addressing of FPC 405 flags is as follows:
F0.0.0 to F.23.15 are global flags.
F24.0 to F63.15 are local flags.

In this context, local means that these flags can only be addressed within the local processor module.

In addition there are 8 flags that are program-internal and are addressed as FI0 to FI7. These 8 flags always have the status logic 0 when the program is started. The first 6 of these flags are available for use within the program. Flags FI6 and FI7 have the following functions:

- FI6 becomes logic 1 in the event of an overflow when using a mathematical operation with CFM20.
- FI7 is used to store the bit shifted out during a SHIFT operation.

Only absolute addressing can be used for these 8 program-internal flags. They are not entered into the allocation list.

Appendix B - Differences in program execution

Local and global allocation list

Because of these inputs and flags (and other "local" operands such as the registers) FST software for the FPC 405 differentiates between the global allocation list that contains operands accessible to all processor modules and the local allocation list, containing operands for the local processor modules.

Note: *If you exchange programs between different systems (file import), then the different types of addressing of local operands must be observed.*

Appendix B - Differences in program execution

B.6 SERIAL DATA PORT

Hardware

All Festo PLCs are equipped with a port for serial data transmission, the command interpreter port. The port types are as follows:

- **FPC 100B/AF:** RS232C data port
- **IPC:** RS232C data port
- **FPC 405:** RS232C data port
- **FEC:** RS232C data port

These data ports (the IPC and FPC 405 also have serial data ports) allow connection of the programming and test device to the controller (industry-standard personal computer).

Appendix B - Differences in program execution

Active/passive interface

All Festo PLCs have a common command set for the serial data interface. This means that **external** addressing of all Festo PLCs is identical.

Festo also offers PLC modules to drive various text displays. These allow Festo PLCs to drive text displays, printers etc. directly from a Ladder diagram program with the aid of the serial interface.

In addition, the FPC 405 controller allows active control of the interface from the program. This controller can send data to or request data from another computer system. This can be done using Festo BASIC and the serial port belonging to the BASIC equipment of the controller. This is done by calling a simultaneously executing BASIC program from Ladder diagram. For details see the BASIC manual.

Appendix B - Differences in program execution

B.7 BEHAVIOUR AFTER BATTERY FAILURE

The FPC 100B/AF and the FPC 405 can be equipped with lithium batteries. The batteries back up the contents of the Random Access Memory of the processor module when the controller is switched off or following a power failure.

In practice this means that the battery is required to:

- back up the user program loaded in RAM (when you are not using an EPROM or EEPROM) and
- back up the status of flags, timer preselects, counters and registers.

When the controller is powered up the charge status of the battery is checked. An error message is generated in the event of the battery failure. In the case of the FPC 103 and FPC 405 controllers, a message is generated if the battery is "weak" and must be replaced as soon as possible.

Depending on the condition of the battery and the duration of a battery failure, a power failure can lead to a change or loss of contents of the RAM. The degree of change is random. Contents of memory are unreliable after a battery failure. However, as soon as the controller is powered up again, change of RAM contents is shown by means of a system error message.

Appendix B - Differences in program execution

For this reason application programs must be written to set all internal operands (flags, counters, timers and registers) to a defined state following a voltage **and** battery failure. A module that sets all internal operands to 0 is enabled for each controller.

In the case of the IPC, the user programs can be stored in a flash memory. The variable data (R, CW etc.) are written to a RAM memory safe against power failure. Because of these measures, the IPC does not require a battery.

In the case of the FEC, the user programs are stored in a flash memory. The variable data (R, CW etc.) are also written to the flash memory safe against power failure. Because of these measures, the FEC does not require a battery.

Appendix B - Differences in program execution

B.8 INPUT SIGNAL DELAY

Input signal delay is implemented for all Festo PLCs in order to prevent the user from having to take extensive interference protection measures for all the input signal ports.

FPC 100B/AF

In the case of the FPC 100B/AF input signal delay is software-implemented. The delay program is integrated into the operating system. Software implementation means that the input signal delay can be changed. The value for the input signal delay is stored in register 63. Changing the value of this register changes the delay for **all inputs** of the FPC 100B/AF.

In this instance only the low byte, i.e. the 8 low-order bits of the register, are evaluated. This means that the value can be set to between 1 and 255, with 1 as the shortest delay and 255 as the longest delay. The contents of the high byte are ignored.

R 63	Delay
0	Longest delay (as 255), approx. 25 ms
1	Shortest delay, less than 1 ms
10	Default delay (sets automatically after power on and battery failure), approx. 1 ms
255	Longest delay (as 0!), approx. 25 ms

Important: Increasing the input signal delay also increases the cycle time for the processor module. If a particular input really does have to be considerably delayed, programming of a timer is recommended rather than the use of register 63.

Appendix B - Differences in program execution

FPC 405

In the case of the FPC 405 input signal delay for the normal I/O modules is hardware-implemented (see FPC405 manual).

IPC

In the case of the IPC input signal delay is software-implemented (debounce time according to the cycle time can be switched ON/OFF by software).

FEC

In the case of the FEC input signal delay is software-implemented (5ms, additional debounce time according to the cycle time can be set by software).

Appendix C - Subroutines

Appendix C

Subroutines, Multitasking and Multiprocessing

Appendix C - Subroutines

Contents

C.1	MODULES	265
	Which controller uses modules?	265
	What are modules?	266
C.2	MULTITASKING.....	267
	Which controller supports multitasking?	267
	What is multitasking?	267
C.3	MULTIPROCESSING.....	272
	What is multiprocessing?.....	272

Appendix C - Subroutines

The possible applications of programmable logic controllers are countless. Cleaning machines for turned parts, baking machines for bread rolls, grinding machines for a coal-fired power station, burner controller for a gas-heated power station, handling equipment for tubs of margarine, automatic welding machines for hermetic compressors, automatic testing devices for heat exchangers.

The majority of these machines have features in common. If you examine the hardware and software of such systems, you will find similar structures – even if one system bakes bread rolls and the other manufactures pneumatic cylinders. Similarly there are common features in the operation of the systems. To some extent operation is based on national and international standards and to some extent on simple generally recognized principles for sensible operator interfaces.

In most cases automated machines have control elements such as

- EMERGENCY OFF
- Start
- Stop
- Automatic
- Manual
- Acknowledge fault
- Select type
(large or small rolls, large or small valves)

etc.

Appendix C - Subroutines

In most cases these automated systems have devices that assist the operators. Examples are displays and signals (lamps, acoustic signals, text displays, screens, flasher lamps, flashing lights, sirens etc.) that indicate

- Ready
- Machine error
- Controller error
- Manual mode
- Automatic mode
- Type being produced

etc.

A program for a programmable logic controller can be regarded as a series of rungs that are translated in a programming device to a series of program lines. Let us assume that your program for the bread making machine is 2000 rungs long. The machine works well with your program, the customer is happy, the rolls are wonderfully golden brown and taste delicious.

Appendix C - Subroutines

A baker must also get up early in the morning to start work. After producing many thousand rolls there is a fault: when the first batch of dough is to be processed a fault indicator illuminates, the machine stops and has to be repaired. It is 5.15 a.m.

What is to be done?

Our master baker fetches the printout of your program to find out why the fault indicator is on. Somewhere in the 2000-rung program must be an output called "Error" ...

We can leave the rest to your imagination.

Of course it would be helpful to give the master baker a list of all possible errors which would assist in providing a solution to our example. But even when the operating instruction and error list are written with greatest care and attention, at some stage it will be necessary to refer to the program to find the fault.

Appendix C - Subroutines

When this is necessary, it would be useful to have a program that is easy to read and understand, in which rungs which might be associated with the error can be located quickly. FST software offers several aids in this respect.

- You can enter a short comment for each rung. A list of all rungs with these comments is printed out at the beginning of your program forming a sort of table of contents for the program.

Rung number 1

Reset registers/start time

Rung number 2

Generate flag edge for counting

Rung number 3

Count

Rung number 4

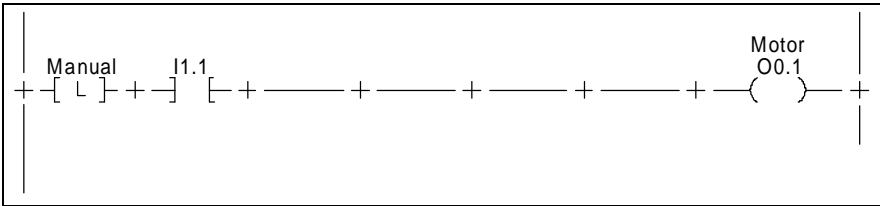
Interrogate counter after 3 seconds

- You can print out a cross-reference list of all operands. This indicates which operands are used in which rungs of the program.

Appendix C - Subroutines

FST cross-reference list											
FPC	CCU	P/B	REF	TYPE	Absolute Operand	Symbol. Operand	Line/Rung/Comment				
100	0	P00	5	LDR	F15.15	EDGE	2	2	3		
100	0	P00	5	LDR	FI		1				
100	0	P00	5	LDR	R0		1	3	4	4	4
							5	5	5		
100	0	P00	5	LDR	R1		1	4	5		
100	0	P00	5	LDR	R2		1	4	4	5	
100	0	P00	5	LDR	T0		1	3	4	4	5
							5				

- You can set jump labels in your program to indicate the beginning of a section. For example, the label "AUTO" is set at the beginning of the section for automatic sequencing. "MANUAL" indicates the beginning of the manual program section,



"ERROR" the error messages etc.

Fig. C/1: Manual operation

- You can use modules, which can be called up as subroutines at any point in your program.

Appendix C - Subroutines

- Within a project you can use various program numbers and program names (as symbolic addresses for a program), and call this as required.
- You can have several processor modules working together in one PLC as part of a project.
- A large system can be divided up into several individual projects interconnected with the aid of a network.

The use of subroutines is described in the section **modules**.

Structuring a project using several programs is described under the heading "**Multitasking**".

The use of several processor modules within a project is described under the heading "**Multiprocessing**".

The use of a network is described in a separate hand book.

Appendix C - Subroutines

C.1 MODULES

Which controller uses modules?

Modules can be used in all Festo PLCs. The number, selection and type of module is dependent on the type of controller:

Controller	Program module	Function module
FPC 100 B/AF	8 in any programming language	CFM0
IPC	100 in any programmig language	
FEC	100 in any programming language	
FPC 405	32 written in LDR or Assembler	CFM0 to CFM75

Appendix C - Subroutines

What are modules?

Modules are subroutines. This means that modules are programs that can be called from another program. When a processor module calls a module, control passes from the calling program to the module and returns to the calling program after processing of the module. Nesting of modules – calling one module from another module – is **not** allowed with the Festo PLCs. However, the FPC 405 does allow calling of the same module from simultaneously executing programs.

Modules are created, edited, tested and loaded independently to great advantage. Various versions of a module can be created and loaded as required.

Note that the use of JUMP commands within a module¹ can result in control never being passed back to the calling program. Particular care must be taken when using backward jumps within a module.

¹ and with the aid of a Step command or the PW command within a module written in Statement list.

Appendix C - Subroutines

C.2 MULTITASKING

Which controller supports multitasking?

Multitasking is the execution of several programs in parallel within a project. Multitasking is supported by the following controllers:

- **FPC 100B/AF:** 1 program 8 program modules
- **IPC:** 64 programs
- **FEC:** 64 programs
- **FPC 405:** Up to 8 programs simultaneously out of 64 stored programs

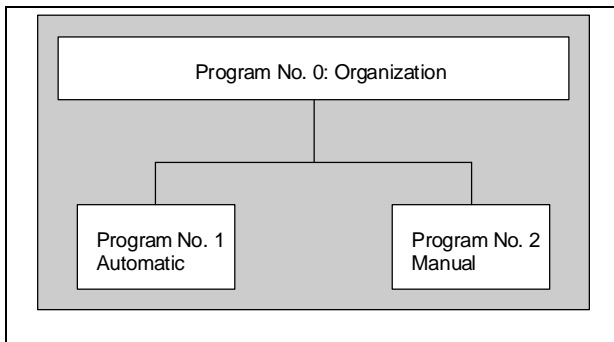
What is multitasking?

A conveyer system is to be operable in two modes "Manual" and "Automatic". A selector switch on the control console allows the operating mode to be selected.

Within a FST project we give the programs for these two operating modes two different numbers, for example program number 1 for automatic mode and program number 2 for manual mode.

Program number 0 is the organization program: dependent on the position of the selection switch and of the machine it manages which operating mode is active. For example, it must only be possible to switch the machine to automatic mode when it is in its initial position.

Appendix C - Subroutines



Therefore we have three programs:

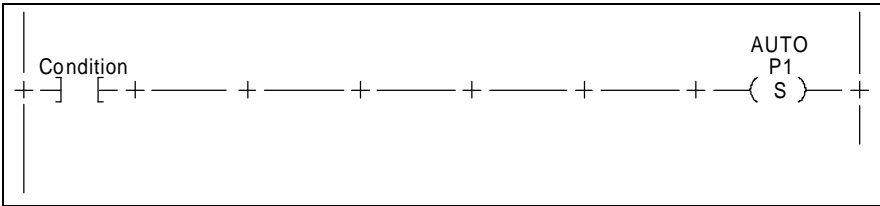
Fig. C/2: Programs

The two programs for automatic and manual mode are typical of programs that are mutually exclusive. A machine must never operate in both manual and automatic mode – the result would be chaotic.

The organization program must monitor the selector switch to see if the operating mode setting has been changed. This means that the organization program must be executing continuously, in parallel to either the manual or automatic mode program.

Appendix C - Subroutines

Additional Ladder diagram programs are always started from program 0. For this reason program 0 is generally the organization pro-



gram.

Fig. C/3: Organization program

The automatic program with the address P1 (absolute address) can be used like any other one-bit operand. It can be set, reset, or interrogated for logic 1 or logic 0.

*Please note that control of the program in the above rung is **signal-edge** controlled. In the above rung the program is set from logic 0 to logic 1 the first time the condition changes. Switching on again – for example after having been switched off from somewhere else – only takes place when the condition changes to logic 0 and then changes back to logic 1.*

Appendix C - Subroutines

When program P1 is set, programs P1 and P0 execute simultaneously. Switchover occurs after each complete cycle through the Ladder diagram program. In reality, the processor first executes one program completely, then the simultaneous program, then the first again etc.

In practice the most common case is a Ladder diagram program working in parallel with a BASIC program either for data transmission to a host computer or for driving a terminal for operator control.

FPC 405

This controller allows 64 programs to be stored in the processor module. Up to 8 Ladder diagram programs can be executed simultaneously. The FPC 405 manual contains a matrix indicating how many of which programs in which language mix can be executed simultaneously.

In practice it is likely there would be several Ladder diagram programs executing simultaneously with a BASIC program for operator guidance or a network.

FPC 100B/AF

This controller allows 1 program and 8 program modules can be stored in the processor module. Up to 1 Ladder diagram programs and 8 Ladder diagram program modules can be executed simultaneously.

Appendix C - Subroutines

IPC

This controller allows 64 programs to be stored in the processor module. Up to 64 Ladder diagram programs can be executed simultaneously.

FEC

This controller allows 64 programs to be stored in the processor module. Up to 64 Ladder diagram programs can be executed simultaneously.

Appendix C - Subroutines

C.3 MULTIPROCESSING

The following controller supports multiprocessing, that is the operation of multiple processor modules within a system (connected via a system bus):

FPC 405 up to 6 processor modules on one bus

What is multiprocessing?

Each processor module of a programmable logic controller consists basically of a micro-processor system. This includes the micro-processor itself and storage and peripheral modules. In addition there are the connections to inputs and outputs, to the programmer etc.

This then represents a system with a single processor. This processor executes each program line in turn in the order prescribed by the program. In addition the processor must handle administrative tasks, counting for counter modules, reacting to requests from programmers and test equipment etc. All this takes time.

Appendix C - Subroutines

The result: the processor module needs time to cycle through a program once. Several measuring programs to measure the cycle time of a processor module are shown in Appendix B, "Example programs".

Depending on the program size and CCU type, this time problem can be significant. The problem can be alleviated by allowing several processor modules to work in parallel.

In computer technology this method is used in parallel processors. But in the case of parallel processors, a single program is compiled in such way that it is automatically distributed to the various processors. The Festo PLCs offer you the possibility of allowing several processor modules to work within a common system.

Distribution of tasks and programs to the various processor modules is undertaken by the application program, that is by you.

Appendix C - Subroutines

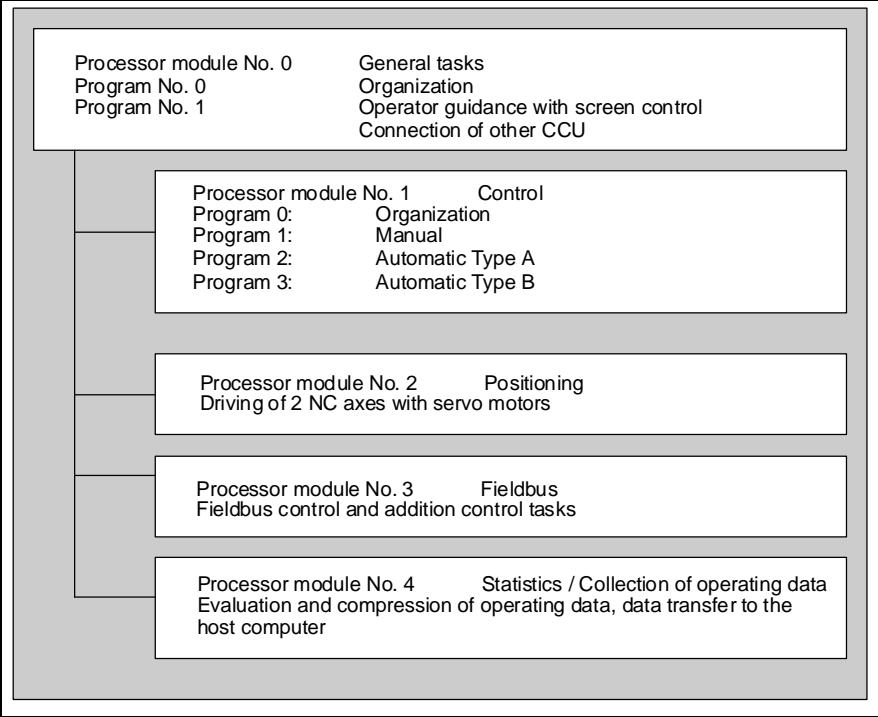


Fig. C/4: Example

Appendix C - Subroutines

Of course it is necessary for the various processor modules to be able to "talk to each other" – in other words exchange information.

This information exchange takes place in the Festo PLC with the aid of flags. The FPC 405 operand parameter consists of:

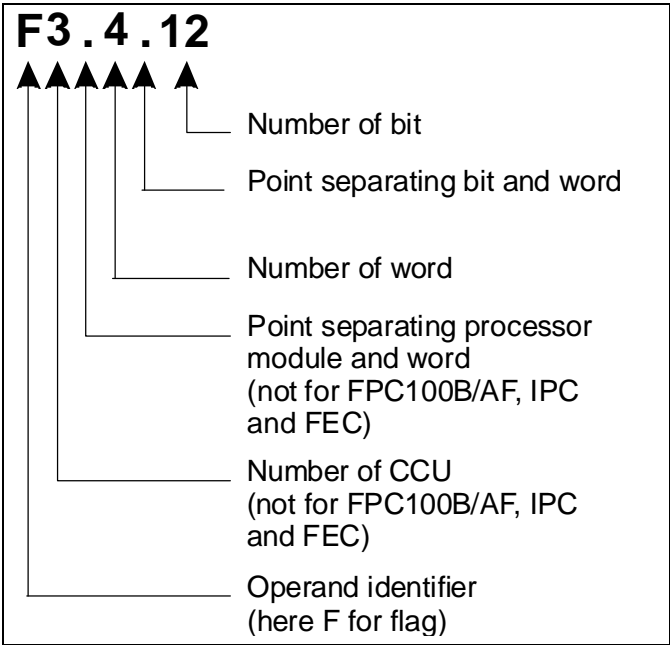


Fig. C/5: Operand parameters

Each processor module can access all flags of all processor modules (with the exception of the local flags of the FPC 405 processor module). This allows quick and easy data exchange.

Appendix C - Subroutines

In addition, the **FPC 405** allows each processor module to start and stop programs executing on other processor modules directly (that is without the use of flags).

This represents a simple and convenient way of dividing a machine project into various subtasks and distributing these to various processor modules, all of which can access the same inputs and outputs.

With the FPC 405 you can select a certain type of processor module to fit the task (e.g. field bus connection, additional serial interface, networking etc.).

Appendix D - Binary and hexadecimal numbers

Appendix D

The binary and hexadecimal number system

Appendix D - Binary and hexadecimal numbers

Contents

D.1	THE STRUCTURE OF NUMBER SYSTEMS	279
D.2	THE BINARY NUMBER SYSTEM ..	282
	BCD code	284
	The hexadecimal number system.....	286
	Signed numbers.....	288
D.3	DIFFERENTIATION IN LADDER DIAGRAM	289

Appendix D - Binary and hexadecimal numbers

D.1 THE STRUCTURE OF NUMBER SYSTEMS

The history of numbers and mathematics has resulted in a wide range of number systems. The oldest known written number is supposed to be 30,000 years old. A general characteristic of the most widespread system – the decimal system – is the structure of places and the significance of these places. For example the number 4344 has the following structure:

4 x 1000 plus 3 x 100 plus 4 x 10 plus 4 x 1

This means that the digit "4" on the left has a completely different meaning to the digit "4" on the right (9301 is significantly more than 1309). This is not the only type of number structure. The Roman number system worked in a completely different way. In the Roman system, a new symbol was used as soon as the number became too large. For example CXXI = 121. On the other hand, IXXC is incorrect as the sequence does not coincide with the meaning of the symbol.

Appendix D - Binary and hexadecimal numbers

Back to the decimal number system.

The basis for the decimal number system is the presence of 10 different digits (for this reason decimal, decimum (Latin) = 10). It follows that we can use 10 different symbols to count from 0 to 9. As soon as we want to go past 9, we carry over to the next position. This next position has the significance 10 (=10¹). In this next position we can again count up to 9 (that is 9 x 10¹ = 90). Then we need to carry over again to the next position with the significance 100 (=10²).

If we take the number 71718711 as an example:

10 ⁷ = 10 Mil	10 ⁶ = 1 Mil	10 ⁵ = 100000	10 ⁴ = 10000	10 ³ = 1000	10 ² = 100	10 ¹ = 10	10 ⁰ = 1
7	1	7	1	8	7	1	1

The table makes it clear that the 7 "on the left" means 7 times 10 million = 70 million, while the 7 in the third position from the right means 7 times 100 = 700.

Appendix D - Binary and hexadecimal numbers

The position on the extreme right is called the least significant position (because its value is least significant) and the position on the extreme left is called the most significant position (because its value is most significant).

Using the 8 positions as in our example the highest number we can create is 99999999, that is nearly 100 million. If we want to see the maximum number that can be represented with a given number of positions, we can simply take the value of the next highest position, subtract 1 and that is the answer.

Example:

The maximum number that can be represented by a 6-digit decimal number is $10^6 - 1 = 1000000 - 1 = 999999$. (10^6 is the seventh position as 10^0 not the 10^1 is the first position).

What does this mean for the Festo PLC? Each word (for example, a register) can store a 4-digit decimal number with the aid of BCD code. A 4-digit decimal number enables you representing numbers up to $10^4 - 1 = 10000 - 1 = 9999$.

We could use this same pattern to develop any amount of number systems. The basic structure of the decimal number system can be transferred to number systems with any number of digits. So it follows that all arithmetic operations and methods that we use for decimal number systems can be also used for other number systems.

Appendix D - Binary and hexadecimal numbers

D.2 THE BINARY NUMBER SYSTEM

Back in 1679 G. W. von Leibnitz fulfilled one of the basic conditions for the development of the computer by transferring the characteristics of the decimal number system to a form of calculation using only two digits. This is because the computer only operates with two digits: "current on", "current off". These two states can be represented as numbers: "1" and "0". Unless analogue signals are used, no other numbers can be represented.

If we limit ourselves to two digits (0 and 1) the number system has the following structure:

2^7 = 128	2^6 = 64	2^5 = 32	2^4 = 16	2^3 = 8	2^2 = 4	2^1 = 2	2^0 = 1
1	0	1	1	0	0	0	1

The principle is the same as for the decimal number. But as only two digits are available, the significance of a position is not calculated by 10^x but by 2^x . The least significant digit (extreme right) has the value 1 (exactly as in the decimal system), the next position has the value $2^1 = 2$ as opposed to $10^1 = 10$ in the decimal system.

Appendix D - Binary and hexadecimal numbers

Using 8 positions (as in our decimal number example) we can represent a number of up to a maximum of $2^8-1 = 255$. This would be represented by the number 11111111_2 .

Each position of the binary number can be occupied by the digit 0 or 1. The smallest unit of the binary system is a bit.

In the above example a number has been created from 8 bits, that is, the bits have been joined together to form a byte (represented in the computer as 8 electrical signals, each of which denotes "current on" or "current off").

In the Festo PLCs, words generally have a width of 16 bits, meaning that they can count from 0000000000000000_2 to 1111111111111111_2 . The highest possible number that can be represented with 16 bits in the binary system is $2^{16}-1 = 65536_{10}-1 = 65535_{10}$.

This is the reason why a timer can only measure a time of up to 65535 times 10 milliseconds = 655.35 seconds = 10 minutes and 55.35 seconds. This is also the reason why an individual counter can only count from 0 to 65536 (or from 65535 to 0).

Appendix D - Binary and hexadecimal numbers

BCD code

Normally, binary numbers are displayed to you as decimal numbers. The easiest method for this is to use the BCD code. In BCD code, each individual digit of the decimal number system is represented by the corresponding binary number:

0_{10}	=	0_2
1_{10}	=	1_2
2_{10}	=	10_2
3_{10}	=	11_2
4_{10}	=	100_2
5_{10}	=	101_2
6_{10}	=	110_2
7_{10}	=	111_2
8_{10}	=	1000_2
9_{10}	=	1001_2

It follows that 4 positions of the binary number system are required for the 9 digits of the decimal number system. This wastes space as 4 positions in the binary number system allow numbers up to $2^4-1 = 15_{10}$ to be represented. However, this is acceptable to achieve greater clarity.

Appendix D - Binary and hexadecimal numbers

The number 7133_{10} is represented in BCD code as:

0111 0001 0011 0011BCD.

This 4-digit decimal number therefore requires 16 bits for representation in BCD code. BCD-coded numbers of this type are often used in 7-segment displays or in encoding switches which pass numbers to the control system. For this reason the command set of the Festo PLCs contains the conversion commands DEB and BID. BID converts from binary to decimal and DEB converts from decimal to binary. Please note that only 16 bits are available in each case. This allows the representation of a 4-digit decimal number in BCD code, i.e. a maximum of

$9999_{10} = 1001 \quad 1001 \quad 1001 \quad 1001_{\text{BCD}}$.

If you convert a larger number into BCD code, the most significant positions that don't fit will be lost without a warning and without an error message.

Example:

DEB converts
 24000_{10}
to BCD code
 $0100 \ 0000 \ 0000 \ 0000_{\text{BCD}}$,
which represents the last 4 decimal digits of
 24000 , namely 4000_{10} .

Appendix D - Binary and hexadecimal numbers

In FPC on-line mode this number

0100 0000 0000 0000_{BCD}

is still understood as a binary number (only decimal and hexadecimal numbers can be displayed, not BCD), and this is therefore interpreted as

16384₁₀.

If we convert this back with DEB (decimal to binary) the number 400010 is written as a binary number, namely

0000111110100000₂.

This is also displayed in FPC on-line mode as 4000₁₀.

The hexadecimal number system

Reading binary numbers is very difficult for the inexperienced user. Reading numbers in BCD code is much simpler but wastes a lot of space. For this reason the octal and hexadecimal number system were developed. In the octal number system, 3 bits are joined together. 3 bits allow representation of numbers from 0₁₀ to 7₁₀, 8 different digits. In the hexadecimal number system, 4 bits are joined together. 4 bits allow the representation of numbers from 0₁₀ to 15₁₀. This requires 16 different digits. The well known digits 0 to 9 are used, followed by the letters A, B, C, D, E and F.

$F_{16} = 15_{10} = 1111_2 = 0001\ 0110_{BCD}$.

Appendix D - Binary and hexadecimal numbers

In the same way as the decimal and binary system, a hexadecimal number is read as:

16^3 = 4096	16^2 = 256	16^1 = 16	16^0 = 1
8	7	B	C
which means: $8 \cdot 16^3 + 7 \cdot 16^2 + 11 \cdot 16^1 + 12 \cdot 16^0 = 34748_{10}$			

As each hexadecimal digit consists of 4 bits, the 16 bits of a word of the Festo PLC can represent the hexadecimal number of up to 4 positions. Therefore the highest number that can be represented with 16 bits is

$$FFFF_{16} = 65535_{10} = 1111111111111111_2.$$

In order to differentiate between decimal and hexadecimal numbers, hexadecimal numbers are prefixed with a \$ when programming the Festo PLC.

Example:

$$\begin{aligned} V\$1A2 = V418 &= 0000000110100010_2 \\ &= 0000\ 0100\ 0001 \\ 1000_{BCD} \end{aligned}$$

or

$$\begin{aligned} V\$2210 = V8720 &= 0010001000010000_2 \\ &= 1000\ 0111\ 0010\ 0000_{BCD} \end{aligned}$$

Appendix D - Binary and hexadecimal numbers

Signed numbers

Up to now we have assumed positive whole numbers. We haven't taken into account that there is such a thing as negative numbers.

In order to work with negative numbers there is a convention that the most significant bit of a binary number is used to indicate the sign:
1 corresponds to -
0 corresponds to +.

As the most significant bit is used for the sign, signed numbers consist of one bit less, in our case maximum of 15 bits. It follows that the largest number that can be represented in a Festo PLC with a signed number is $\pm 2^{15} = +32767 \dots -32768$.

Appendix D - Binary and hexadecimal numbers

D.3 DIFFERENTIATION IN LADDER DIAGRAM

Ladder diagram for Festo PLCs has the following conventions for differentiation of number systems:

1. In comparison operations (>, <, =, >=, <=, <>) an unsigned positive binary number between 0 and 65535 is assumed. Negative numbers must be converted for comparison, unless the FPC 405 is switched to signed numbers with CFM20.
2. All arithmetic operators (+, -, /, *) assume signed binary numbers between 0 and ± 32767 .
3. All bit-manipulating operations (BID, DEB, INV, CPL, AND, OR, EXOR) take all 16 bits of a word into account. They do not treat the most significant bit as a sign, but as a bit.
4. Hexadecimal numbers are only unsigned, as only positive numbers are possible.

Notes

[illegible]

Index

Index

Index

!

0-1 signal edge.....	164 ; 189
7-segment display	129 ; 131 ; 285

Index

A

Active serial port252

Address

Absolute21 ; 145 ; 147 ; 153 ; 248

Absolute (list of all possible)155

Absolute operands (rules for).....147

Counter157

Error (word)155

Flag (word)156

Function module155

Function unit.....155

Initialization flag155

Input (word)155

Mixing absolute and symbolic150

Null operation157

Output (word).....155

Program157

Program module155

Register157

Representation in printout.....150

Rules for symbolic addresses.....148

Symbolic.....21 ; 145 ; 148

Timer157

Allocation list9 ; 15 ; 150 ; 152 ; 250

AND, see Logic86

Application program.....273

Arithmetic

Adding91

Dividing93

Functions208

Multiplying.....95

Operators23 ; 199 ; 289

Subtracting97

Arithmetic/Logic Box.....62

Assembler11

Automatic233

Auxiliary relay.....203

Index

B

BASIC	11 ; 252
Battery	
Buffering.....	33
Failure.....	33 ; 213 ; 253
BCD code	199 ; 281
Binary number	129 ; 131 ; 282
Bit	153 ; 283
Manipulation	289
Operand, see one-bit operand	24
Boolean operation.....	37
Boxes.....	34 ; 36 ; 62
Maximum number.....	34
Bus access	246
Byte	283

C

CCU	10
Central control unit	16
Check routine	44 ; 242
Circuit diagram	34 ; 37 ; 170 ; 172 ; 192 ; 203
Coding switches.....	129; 131
Coil	33; 177
Assignment.....	115
Negated assignment.....	116
Resetting.....	113
Setting.....	114
Command interpreter port.....	251
Comment.....	17 ; 64 ; 262
Comparison.....	208
Equal.....	117
Greater than.....	119
Greater than or equal	121

Index

Less than	123
Less than or equal.....	125
Operations	199 ; 289
Unequal	127
Complement	73
Conditional part	37
Contact	33 ; 177
Control part	215 ; 216
Counter	23 ; 183 ; 283
Actual value	185 ; 186
Bit	186
Compare	185
Counting.....	189
Decremental.....	23 ; 183 ; 188 ; 193 ; 195
Event.....	183
Incremental.....	23 ; 183 ; 191 ; 195
Initialize	187
Interrogate.....	190
Load counter word.....	188
Nominal value	185
Possible operands.....	187
Preselect.....	31 ; 185 ; 186
Programming	191 ; 193
Pulse	186
Reset	185 ; 188
Set	187
Setting preselect	139
Word	23 ; 31 ; 186 ; 190
Zero actual value.....	187
Counting	
Interrogate.....	189
Cross-reference list.....	9; 262
Cycle time	45 ; 46 ; 218 ; 221 ; 243 ; 273
Cycle time measurement.....	46

Index

D

Decimal number.....	23
Decrementing.....	137
Direct access	
Advantages	45
Disadvantages	44
Documentation	9 ; 18
Double-word.....	21

E

EEPROM	253
EMERGENCY OFF	232
Encoding switch	285
EPROM.....	253
Error.....	24 ; 147
List.....	9 ; 18
Word	23 ; 27
Executive part	37
Existing flag chain	214

Index

F

FI	23 ; 39 ; 192 ; 194 ; 213 ; 216 ; 231
Flag.....	203 ; 213 ; 275
Bit	203 ; 204
Word	204 ; 207
Flashing outputs	42
Function	
Chart.....	15
Diagram.....	212
Key	18
<F1>	17
<F10>	18
<F9>	18
Key allocation.....	18
Module	69 ; 70 ; 245 ; 265
Unit.....	27

G

Global allocation list	250
------------------------------	-----

H

Help key	18
Hexadecimal.....	29 ; 199 ; 286 ; 289
Number.....	23
High byte	67

Index

I

Import of data	9 ; 246
Incrementing.....	137 ; 138
Initialization flag.....	23 ; 39 ; 213 ; 216 ; 231
Input signal delay.....	200 ; 223 ; 248 ; 255
Input word	23 ; 27
Internal	
Flags	248
Inputs	248
Operands	248
Interrupt.....	227 ; 247
Capability	247
Inverting	71

J

Jump	46 ; 110 ; 227
Backward	241
Label	110 ; 112 ; 149 ; 263

Index

L

Ladder diagram	11 ; 33 ; 270
Editor	18
Program	16
Latch.....	107
Dominant resetting	108
Dominant setting.....	109
Latching	215 ; 216
Least significant position.....	281 ; 282
Line crossing.....	33
Lithium battery.....	253
Load.....	9; 37
LOAD command	64
LOAD TO.....	75
Local	
Allocation list	250
Input.....	248
Logic	
AND.....	86
AND with words	88
Exclusive OR.....	76
EXOR.....	76
EXOR with three inputs	77
EXOR with words	79
Identity	81
Identity with words, see "LOAD TO".....	81
Negation.....	82
Negation with words	71 ; 82
NOT (at input).....	83
NOT (at output).....	82
NOT with words	71
Operation	23 ; 37
OR	83
OR with words	85
Long-word	21
Low byte	67

Index

M

Machine code.....	38
Manual	233
Masking.....	85 ; 88
Matrix programming	11
Module	9 ; 23 ; 65 ; 69 ; 263 ; 265
/Program	16
Nesting.....	266
Number.....	16
Most significant bit	288
MS-DOS.....	15
Multi-bit operand, see Word operand.....	183
Multiprocessing	205 ; 246 ; 272
Multiprocessor system	45
Multitasking	227 ; 244 ; 267

N

Negation	
Of words.....	71
Network.....	264
NOP	90
NOT, see Logic	82
Null operation	90
Numbering	38
Numbers	
Conversion BCD to digital	129
Conversion digital to BCD	131
Interrogation preselect achieved	133

Index

O

Octal	286
One-bit operand	21
On-line mode	18 ; 218
Operand	10 ; 186
Address	145 ; 153
Enter	36
Identifier	145 ; 147 ; 153
Missing	36
Parameter	145 ; 153
Retentive	39
Operating mode	267
Operation	36
OR, see Logic	83
Organization program	267
Output	24
Word	23 ; 27

P

Page header	15
Parallel	
Branch	33 ; 35 ; 84
Coil	33 ; 35
Connection	37
Maximum number	33
Processor	273
Programs	267
Parameter	69
PC	251
Power	
Failure	33 ; 213 ; 253
Part	215 ; 216
Up	38 ; 246
Preselect, see Counter	31
Print	9
Printer drive	252

Index

Priority for bus access	246
Procedure	69
Process image	38 ; 40 ; 223 ; 241
Advantages	44
Disadvantages	45
Method	44
Processor module	10 ; 205 ; 218 ; 246 ; 248 ; 264
Program	9 ; 23 ; 244 ; 264
Load	18
Module	65 ; 245 ; 263 ; 265
Number	16
Save	18
Structure	231
Programming and test device	251
Project	9 ; 149 ; 264 ; 267
Name	15
Pulse timer, see Time	168

Q

Quasi-parallel processing	244
---------------------------------	-----

Index

R

Reaction time	223
Absolute in 0.1 ms	225
Longest possible time.....	227
Measuring	46 ; 223
Register	23 ; 29 ; 199
Register 63 (FPC 100)	255
Resetting sequence	216
Retentiveness	33 ; 177 ; 186 ; 200 ; 203 ; 213
ROL	99
ROR.....	101
Rotate	
Left.....	99
Right.....	101
RS232C interface	251
Run/Stop switch	38 ; 203
Rung	33 ; 34
Number.....	38
Section.....	33

Index

S

Safety routine..... 44 ; 242

Sequence

 Program 15 ; 211

 Program with counter.....217

 Program with jump217

Sequencer.....216

Serial data transmission.....251

Series connection37 ; 87

Shift

 Left (SHL).....103

 Register 103 ; 105

 Right (SHR)105

Sign.....73 ; 288

Signal-edge recognition.....164

Signed.....23 ; 199

Significance280 ; 282

Simultaneous.....244

 Execution, see Multitasking267

Standing 139 ; 43

Start/Stop switch.....38 ; 203

Statement list11

Structure chart15

Subdirectory.....15

Subroutine.....245 ; 263 ; 265

 Nesting.....266

Subroutines.....65 ; 69

SWAP67

Switch-off delay, see Time165

Switch-on delay, see Time165

Symbolic address, see address.....21 ; 145

Syntax test.....18

System bus246

Index

T

Task.....	244
Text display	252
Time.....	283
Box.....	164
Clock pulse.....	162
Compare timer word	178
Digital time measurement	162
Error in measurement	162
Flashing lamp	174
Initialization	164
Longest time.....	163
Module	177
Preselect.....	177
Pulse timer	165 ; 168
Quartz	161
Requirement for operations	218
Shortest time	163
Start	141 ; 142
Switch-off delay.....	164 ; 165 ; 167 ; 172
Switch-on delay.....	164 ; 165 ; 166 ; 169 ; 171
Word	177
Timer.....	23
Bit	178
Preselect.....	29
Word	31
Timer, see Time	141 ; 142 ; 161
Title page	15
TO.....	64
Twos complement.....	73

Index

<i>U</i>	
Unsigned	199

<i>V</i>	
Version.....	9
Number.....	17

<i>W</i>	
Word.....	23 ; 283
Operand	21 ; 27 ; 183 ; 199
Operation	62

<i>Y</i>	
Yes function	81