Modsoft AKF → Micro
Type: MICROAKF
Version 2.0 – 2.1
MICROAKF for Beginners

User Instruction
DOK-704105.33-1197

Translation of the German Description DOK-703411

Belongs to software kit E-No. 424 704 703

Documents in the software package

Documentation	Area of application
Installation User Instruction DOK-704104	Explains the usage and installation of the diskettes included.
MICROAKF for Beginners User Instruction DOK-704105	Serves to introduce new customers to MICROAKF. The user learns how to use the software in small steps.
Short Form Guide User Instruction DOK-704106	Tables for validity ranges and system markers, SFB-Formal operands for quick use onsite.
SFB Block Library DOK-704572	Description of the Standard Function Blocks.
Configuration User Instruction DOK-704107	Contains the new features of the current version and explains the functions of the individual software menus for the configurer.
Masterindex User Instruction DOK-704108	Index of all documentation.

Notes

Application Note

Caution The relevant regulations must be observed for control applications involving safety requirements.
For reasons of safety and to ensure compliance with documented system data, repairs to components should be performed only by the manufacturer.

Training
Schneider Automation offers suitable training that provides further information concerning the system (see addresses).

Data, Illustrations, Alterations
Data and illustration are not binding. We reserve the right to alter our products in line with our policy of continuous product development. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us by using the form on the last page of this publication.

Addresses
The addresses of our Regional Sales Offices, Training Centers, Service and Engineering Sales Offices in Europe are given at the end of this publication.

Copyright
All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including copying, processing or any information storage, without permission in writing by the Schneider Automation. You are not authorized to translate this document into any other language.

Trademarks
All terms used in this user manual to denote Schneider Automation products are trademarks of the Schneider Automation.

Terminology

Note This symbol emphasizes very important facts.

Caution This symbol refers to frequently appearing error sources.

Warning This symbol points to sources of danger that may cause financial and health damages or may have other aggravating consequences.

Expert This symbol is used when a more detailed information is given, which is intended exclusively for experts (special training required). Skipping this information does not interfere with understanding the publication and does not restrict standard application of the product.

Path This symbol identifies the use of paths in software menus.

Figures are given in the spelling corresponding to international practice and approved by SI (Système International d‘ Unités).
I.e. a space between the thousands and the usage of a decimal point
(e.g.: 12 345.67).

Abbreviations

ABS	absolute addressing
Adr.	Address (signal address)
AKF	Instruction List, Ladder Diagram, Function Block Diagram
IL	Instruction List
AWP	User program
DB 0 ... 9	Data block(= SYM/COM block)
D-Word	Double word
I/O	Input / Output siganls (e.g. of a module)
FB	Function block
FBD	Function Block Diagram
G-Word	Floating point word
HW	Hardware (e.g. PLC)
IB	Initialization block
LD	Ladder diagram
OB	Organisation block
PB	Program block
PADT	Programming and debugging tool (= Programming panel)
React.	Reaction during descriptions of steps (on screen)
SFB	Standard function block
PLC	Programmable controller system (= programmable controller)
SSP	Signal memory
SW	Software
SYM	Symbolic addressing
SYM/COM	Symbols and comments
SA	Node
<Return>	actuate the Return key
<Esc>	actuate the Esc key
<Ctrl>+<Alt>+<Del>	actuate the keys Ctrl, Alt and Del simultaneously (begin with Ctrl and end with Del)

Objectives

An introduction into structured programming is always provided.

Arrangement of This Guide

Chapter 1 This chapter gives a short overview of which components are used in programming with Modsoft AKF. Further, the basic functions of the software are given.

Chapter 2 After a general explanation the various block types in Modsoft AKF are explained. The different structure levels are demonstrated by means of a picture.

Chapter 3 This chapter gives a short explanation of the individual technical languages.

Chapter 4 In this chapter the most important functions and the performace of the individual block types is described.

Chapter 5 This chapter shows in the form of a program flowchart how you should handle system configurations.

Chapter 6 This chapter explains the program structure of the example program contained in the software.

Chapter 7 This chapter contains a small application example, executed in all details, for programming the Micro with MICROAKF.

Validity Note

This instruction manual is valid for the software MICROAKF, version 2.0 – 2.1.

Table of Contents

Chapter 1
Introduction

This chapter gives a brief overview of the important components available for Modsoft AKF programming. In addition, the basic software functions are listed.

1.1 General

The Modsoft AKF software is used for structured programming of programmable controller programs using modern windows technology.

For this purpose, three specialized technical languages are used for creating and depicting programs: the instruction list, the ladder diagram and the the functional block diagram (see also the draft standard IEC 1131).

The programs consist of different blocks which can be connected with each other, depending on the application and complexity of the job. The different block types - organization block, program block, function block, standard function block and data block DB0 ... DB9 (see also Chapter 4):

- Organization of the overall program
- Summary of technical program parts
- Facilitating repetitive programs (”subroutines”)
- Facilitating programming with predefined part programs
- Facilitating symbolic programming

Following the introduction into ”Structured Programming” with its program parts, further chapters will go into MICROAKF in greater detail. A brief review of the features is then followed by a sample application on which the basic skills can be practiced. A statement of requirements for the task to be solved by the sample program is also found on the MICROAKF diskettes.

1.2 Programming components

What you will need for programming your programmable controller?

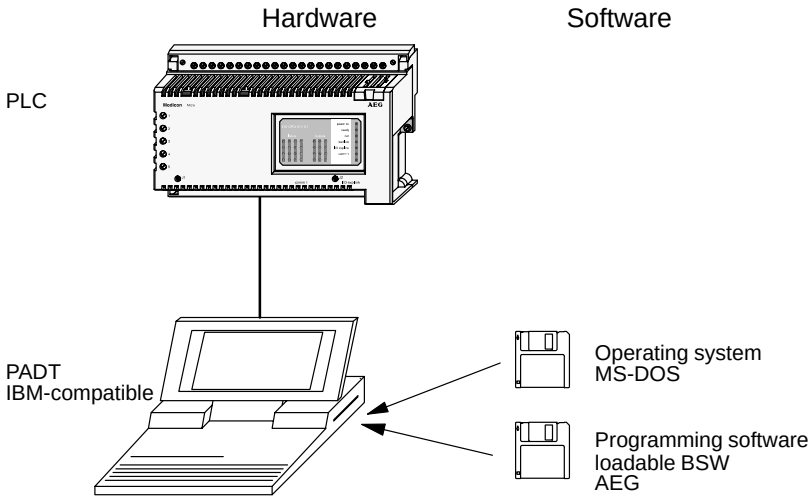


Figure 1 Components for programming a PLC

1.3 Basic functions

User programs in Modsoft AKF are created off-line and then transferred into the PLC. Subsequently, online functions are also possible, as for example an overview of the program during scan time.

The following basic functions are performed with Modsoft AKF:

❒ **Edit** (create/modify)	-	offline
❒ **Load** (from / to PLC)	-	off- or online
Compare (PADT / PLC)	-	online
❒ **Online**	-	online
❒ **Print**	-	offline
❒ **Special**	-	offline
❒ **Setup**	-	offline

Chapter 2
Structured Programming

Following a general description, various types of Modsoft AKF blocks will be explained. A figure illustrates the structure levels.

2.1 General

The performance and economy of a programming system are marked by a number of special features.

Structuring and standardization possibilities, the use of general-purpose personal computers as programming panels (PADT) and a convenient user interface add up to advantages that keep software and startup costs as low as possible.

Given the large quantities of information and the large programs common in programmable controller applications today, program partition is useful in optimizing scan times. Time-critical processes require rapid reactions. This is achievable by means of skillful configuration, which avoids processing those parts of the program which do not need immediate attention.

The division of a complex of tasks into smaller parts helps to make the bigger picture understandable.

Program parts as individual, closed software blocks are easier to produce and test. Processes integrated into large monolithic main programs, on the other hand, are considerably more difficult to understand in their entirety.

Software configuration with the convenient programming software is useful for avoiding difficult to manage complexes of branching instructions.

2.2 Program structure

The technical languages facilitate the structuring and writing of programs. Programs can be written and depicted in the form of instruction lists, ladder diagrams, and function block diagrams.

By ”structuring”, we mean the creation of clear, readily understandable, closed user-program parts, known as blocks.

Processes that are specific to one technical application, as well as processes that are repeatedly used, can be created and tested, then used many times within a system or as a technical block. Function blocks can be assembled into universal or user-specific programming libraries. Standard function blocks for complex control, data processing and operating functions, integrated into the programmable controller, form the basis for the easy buildup of software blocks into complex user-specific applications.

Blocks are composed of networks. These form the lowest structuring level. The networks contain the logic, which is made up of operations with parameters (known as ”instructions” in the context of the instruction list.

To help maintain clear order within the complex of tasks, there are the following five types of blocks (see Chapter 4).

The figure below depicts an example of the different structural levels.

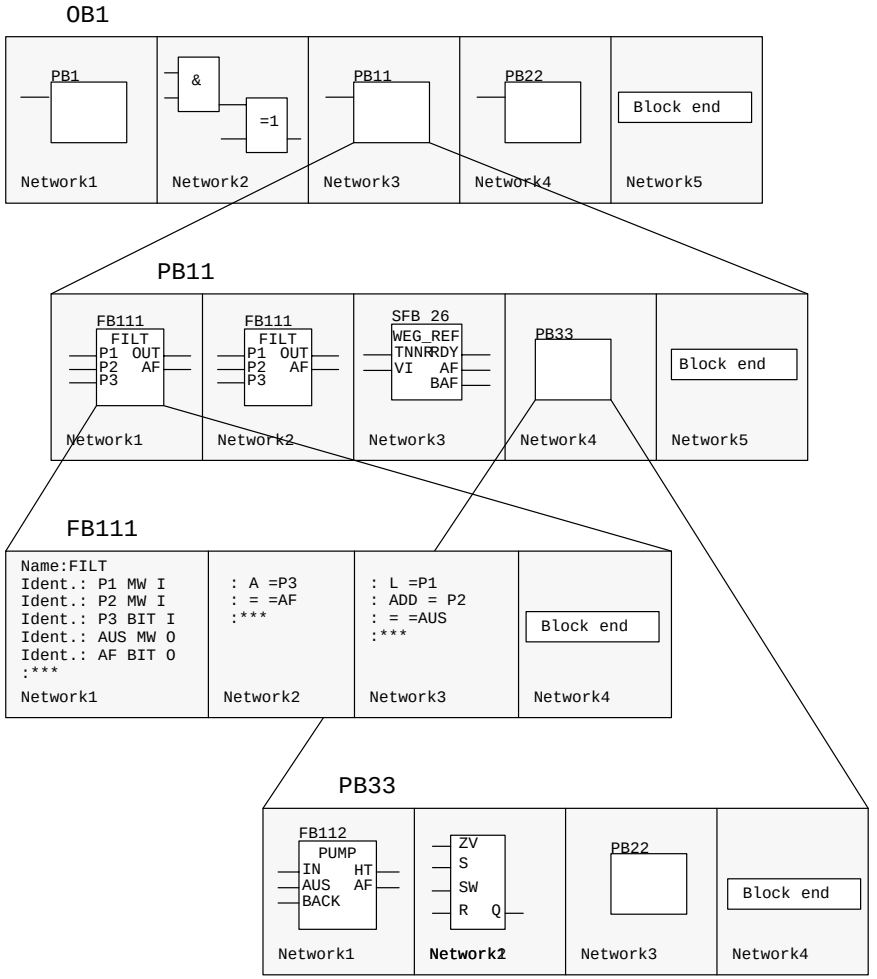


Figure 2 Sample excerpt from a structured program (Micro)

Chapter 3
Technical Languages in Programming

This chapter gives a short explanation of the special languages

- Instruction List IL
- Ladder Diagram LD
- Function Block Diagram FBD.

Modsoft AKF software is based on the structured programming system in the standardized technical languages. For information on the standard features (structure, etc.) see DIN 19239 or the draft standard IEC 1131.

- Instruction list (IL)
- Ladder diagram (LD)
- Function block diagram (FBD)

It is possible to represent programming blocks in a language other than the one they were programmed in .

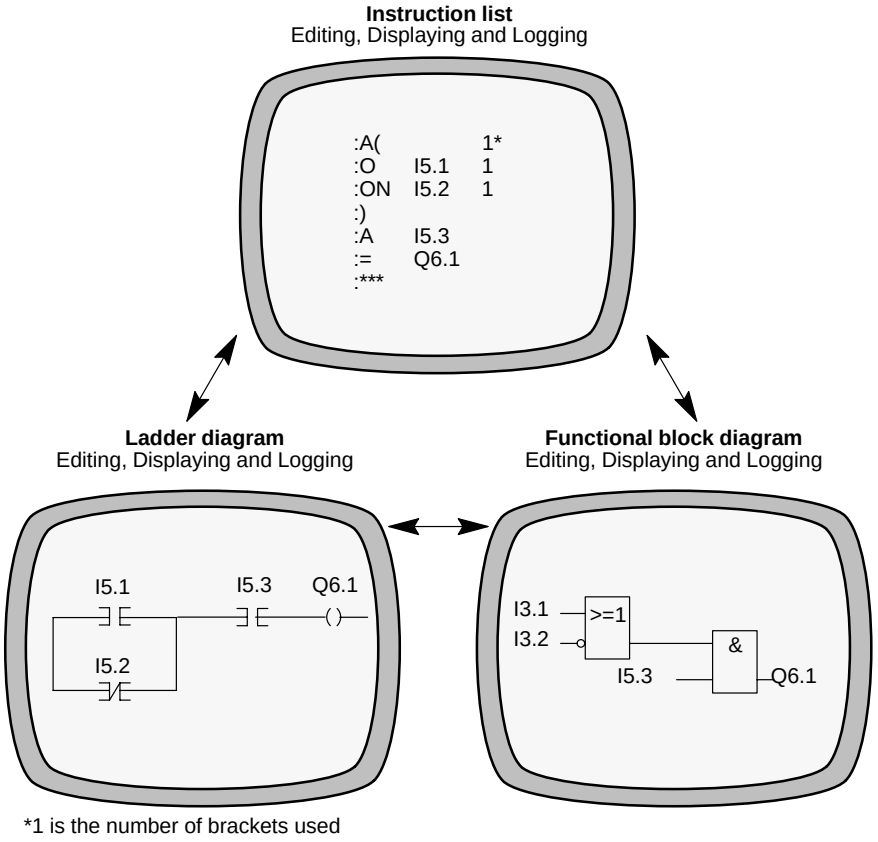


Figure 3 Presentation in different technical languages

3.1 Instruction list IL

The instruction list is a standard type of presentation in alphanumeric form.

Programs are generated in IL as a sequence of instruction lines.

There are two different types of instruction. The following two instructions are equivalent (AF is arbitrary):

: O I5.1 : O =AF

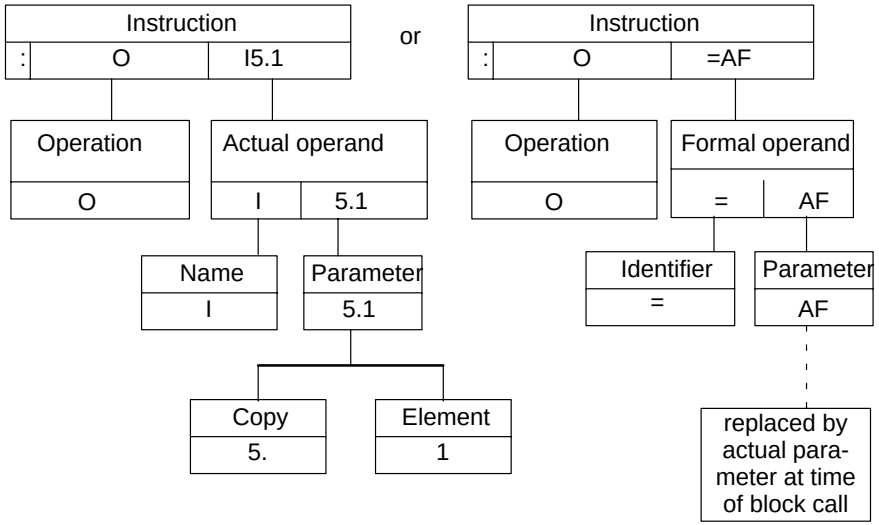


Figure 4 Explanation of terms in instruction list

A network's instruction list closes with the network end sign "***".
The end of a block is marked with "BE".

There are tables of operations and operands/data structures for the individual controls.

Organization blocks (OBs), program blocks (PBs) and function blocks (FBs) may be programmed in IL.

Branching and block calling instructions are possible in instruction list programming.

☞ **Note** More detailed information (e.g. how to generate programs in instruction list) is available in the User Instruction ”Configuration.

3.2 Ladder diagram LD

The ladder diagram is a standard type of graphic presentation.

The following basic symbols are available when generating ladder diagrams.

—]\[—	Normally open contact
—]/[—	Normally closed contact
———┤	Connection of parallel conductors
———	Continuation in parallel, but without any contact
—( )—	Output

Ladder diagram operations/data structures are parametred with operands.

See for explanations of terms Figure 4.

The end of a block is marked with the box ”Block end”.

OBs , PBs and FBs may be programmed in ladder diagram.

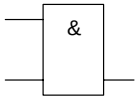
Branching instructions are not permitted in the ladder diagram, but block calls are.

☞ **Note** Further information (e.g. on generating programs in ladder diagram) is contained in the User Instruction ”Configuration.

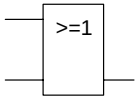
3.3 Function block diagram FBD

The function block diagram is a standard type of graphical presentation.

The following basic symbols are available for generating function block diagrams.



AND-block



OR-block

Input
Negated input

Function block diagrams are still parametered with operands/data structures.

In FBD, OBs, PBs and FBs may be programmed.

The end of a Blocks is marked by a box with ”Block end”.

See Figure 4 for explanations of terms.

Branching instructions are not permitted in the function block diagram, but block calls are.

☞ **Note** Further information (e.g. regarding the generation of programs in function block diagram) is available in the User Instruction ”Configuration.

Chapter 4
Software Blocks

Main functions and performance features of the different types of blocks will be described in the following chapter.

4.1 Types of blocks

- The **organization block OB**
  contains the rough structure and determines the sequence in which further blocks are to be processed.

- The **program block PB**
  organizes parts of the user program along technical criteria such as, for example, modules, parts of machines or sections of plants.

- The **function block FB**
  processes recurrent program parts as separate subprograms.

- The **standard function block SFB**
  performs the same task as the FB, and is an integral component of the range of standard functions of the PLC.

Individual networks in the IL, LD or FBD presentation are the ”substructures” which make up the program blocks and standard function blocks (PBs und SFBs). The individual networks make up the program with the sequence of instructions for the process control system.

The user-program in Modsoft AKF is composed of different blocks. Their selection will depend on the complexity of the task and on achieving the most economical configuration. The block technique serves for structured programming.

- The **SYM/KOM-block (known as data block DB0 to DB9)**
  contains and organizes the allocation of hardware addresses, symbolic addresses and comments. It cannot be linked into a network, being instead generated separately using the SYM/COM-editor.

4.2 Organization Block (OB)

In MICROAKF, you have an organization block, OB1. OB1 determines the framework for the entire user program.

The organization block can be generated in IL, LD or FBD.

The OB is processed in cyclical scans. Each scan begins by processing network 001 and ends by processing the last network in OB.

Program and function blocks (PB, FB and SFB) are called from the OB in the required sequence and processed.

PB's and FB's are arrayed here in a sequence of networks with continuous numbering, starting with network 001.

Every network contains either one PB, FB or SFB call (with the exception of IL) or one part of a user program in IL, LD or FBD.

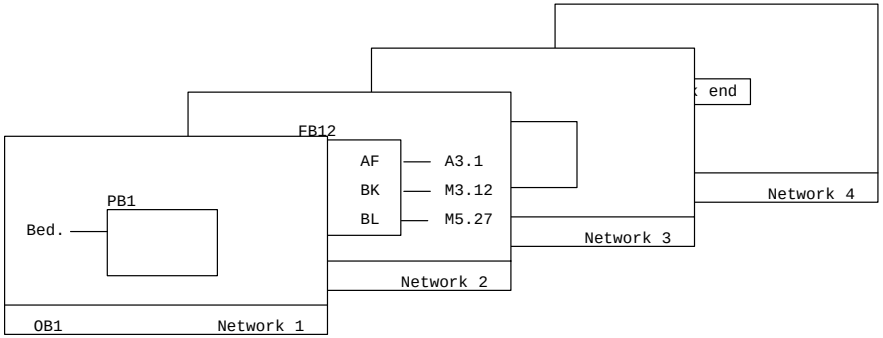


Figure 5 Example of networks in an organization block

Depending on the specified conditions, a software block call is followed by the processing of the block concerned (PB, FB or SFB). There then follows the return to the next network of the OB.

4.3 Program Block (PB)

A program block generally contains technically related parts of the user program, e.g. one of x different machines.

Program Blocks can be programmed in IL, LD or FBD.

Structure:
A PB consists of a sequence of networks with continuous numbering, begin with network 001. In the networks you can generate IL, LD or FBD program parts, or you can make conditional or unconditional calls of PB's, FB's and SFB's.

Call:
PBs may be called from the OB, from another PB or from a FB.

One PB may be called repeatedly.
A block to be called is depicted as a rectangle in the network (in LD/FBD). The PB number is displayed above the rectangle. For conditional PBs, the signal address of the condition is displayed to the left of the rectangle.
A PB which is not called from any place will never be processed.

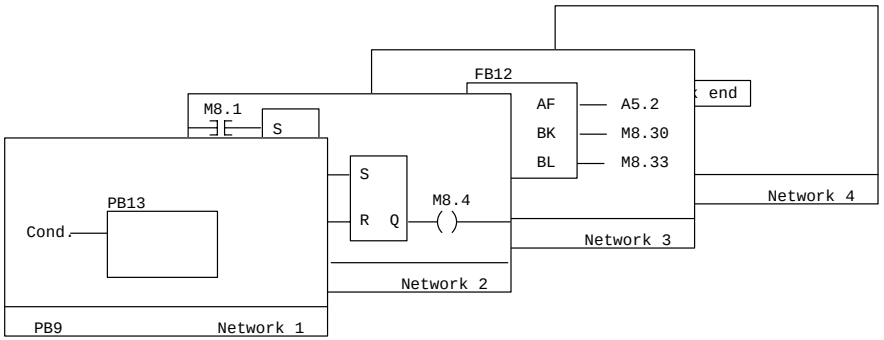


Figure 6 Example of networks in a program block

4.4 Function Block (FB)

Function blocks serve for creating frequently recurring parts of the program. They are parameterizable subroutines, i.e. an FB can be called up and parametrized a number of times at different points.

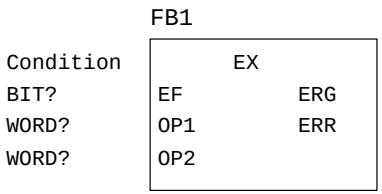
It is necessary to distinguish between a function block and the call of a function block. The function block contains a part of the user program. The call of an FB ensures that it is processed precisely when the corresponding call is initiated in the program. Before the processing of the block the parameterizing of the FB is transferred to the subroutine (formal operand replaced by actual operand). An FB that is not called up from any point is never processed.

Structure:
The program of the function block consists of the declaration part and the instruction part.

- **Declaration part**
  The declaration part is always found in the first network of a FB.
  Enter the name of the function block and the list of formal operands, giving types, in the declaration part.
  The declaration part also contains information on the graphical structure of the block rectangle and the parameter sequence.
  The declaration part can subsequently be modified, to a certain extent.

- **Instruction Part**
  The instruction part contains the program in IL, LD or FBD, which determines the algorithmic operation on the formal operands in the declaration part.
  The names of the formal parameters must always be preceded by a ”=” character in the instruction list. On the right, next to the instruction list, there may be a number stating the nesting depth of the particular line.

Call



Conditional call of a function block in ladder diagram or function block diagram. The formal operands are displayed in the block. Actual operands/data structures (M3.3, A5.1) must still be entered in place of the question marks.

Figure 7 Example of a conditional call

A FB can be called from another FB, a PB or an OB. The FB then appears in the LD/FBD as a rectangle in the network.

In the rectangle its designation is shown again in brief notation. Inside the rectangle on the left are the input formal operands, and to the left of the rectangle the actual operands, with a condition above, if applicable. Inside the rectangle on the right are the output formal operands, and to the right of the rectangle the actual operands. After calling the FB, you need only enter the parameters outside the rectangle.

If formal parameters are modified in the declaration part of an FB, all FB calls may have to be re-programmed. With the program summary or the global cross-reference list you can quickly find out where the calls of an FB are located in the user program.

It is possible to call an FB from the instruction part of another function block (nesting, recursion). The called function block can contain the formal parameters of the calling FB as actual parameters. As many as ten nestings are possible. During configuration it is necessary to ensure that this limit is precisely adhered to.

4.5 Standard Function Block (SFB)

Along with the Modsoft AKF software, you will receive a library of standard function blocks. These blocks are predefined and need only be called up (conditionally or unconditionally) at whichever points you wish and parameterized.

The declaration and instruction parts of the SFB are already present in the software and cannot be modified by the user. The formal parameters are prescribed. The designer calls up the block at the point required in the program and parameterizes it with the desired actual operands.

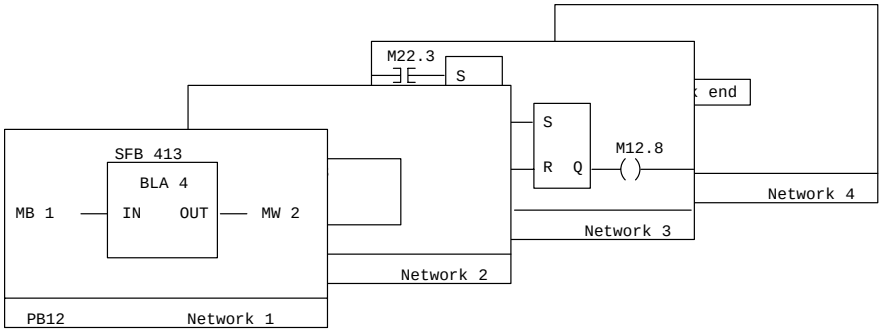


Figure 8 Example of a SFB call in the network of a PB (NW1)

4.6 Data Block DB

What is the relationship between an absolute address (inputs/outputs, markers, etc) and its technical function? Briefly, it is possible to provide absolute addresses with symbolic names and comments. The text of the symbolic names and comments is held in the SYM/KOM-block under the current station name.

After activation of the SYM/KOM-block in programming, the symbolic names entered in the SYM/KOM-block can be used in place of the absolute addresses. The SYM/KOM-block can also be documented.

Signal	Symbol	Comment
I5.1	ON	Motor 1 ON
I5.2	MOT_RI	Motor right on
I5.3	MOT_LE	Motor left on
I5.4	HALT	Emergency off button
I5.5	PUMP_1	Pump 1 on
I5.6	PUMP_2	Pump 2 on
I5.7		
I5.8		
I5.9		
I5.10		
I5.11	PINC_UP	Clamp up
I5.12	PINC_DO	Clamp down
I5.13		
I5.14		
I5.15		
I5.16		

Figure 9 Example of Entries in SYM/KOM-block

Chapter 5
Micro Working Guide

The following pages depict in program flowchart form how MICROAKF should be used in Micro system processing.

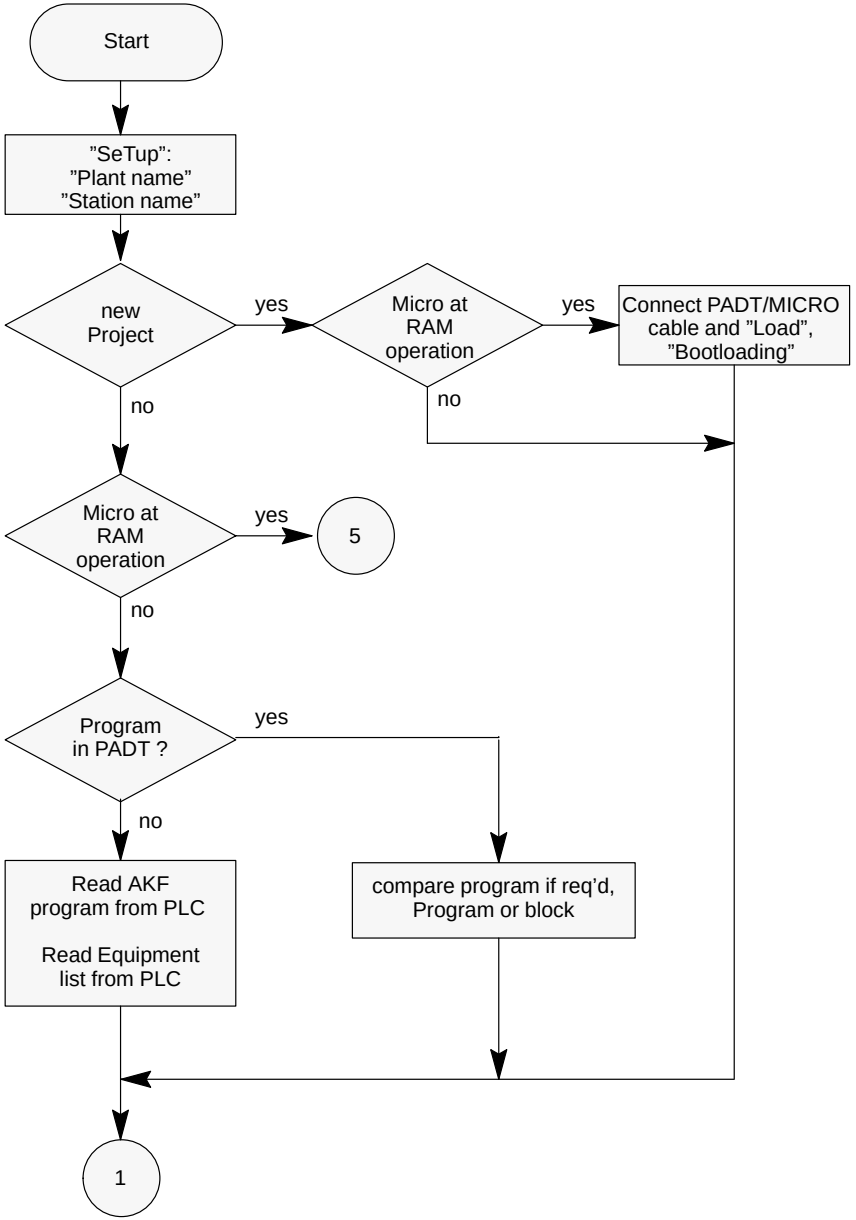


Figure 10 Flowchart System processing

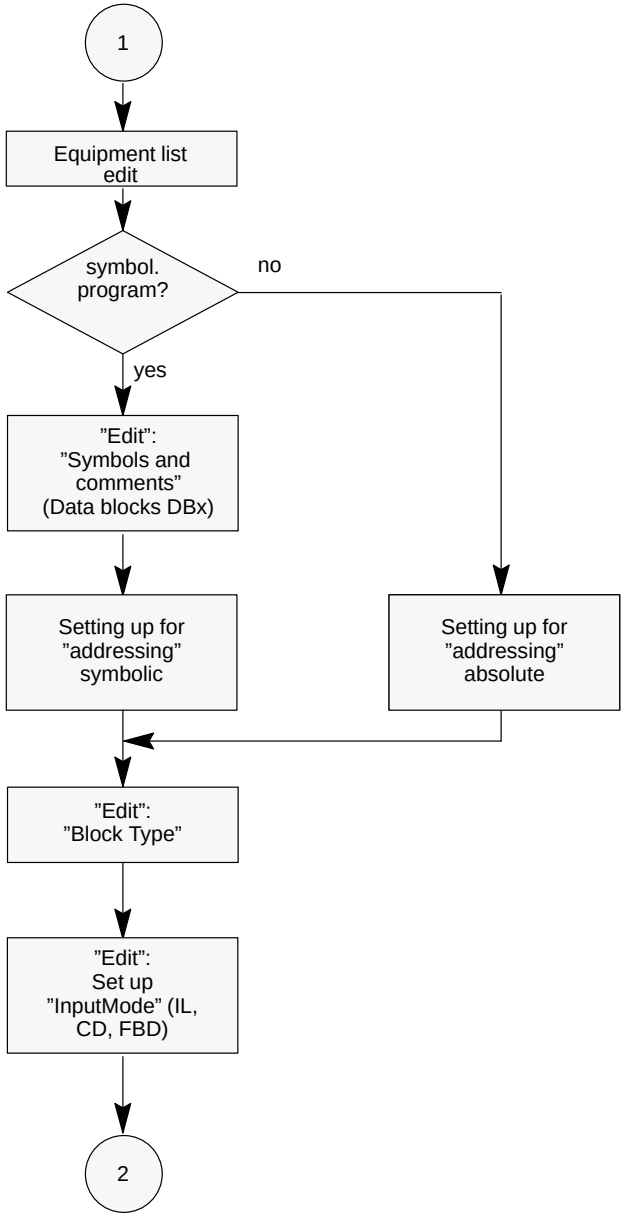


Figure 11 Flowchart System processing (continued from Figure 10)

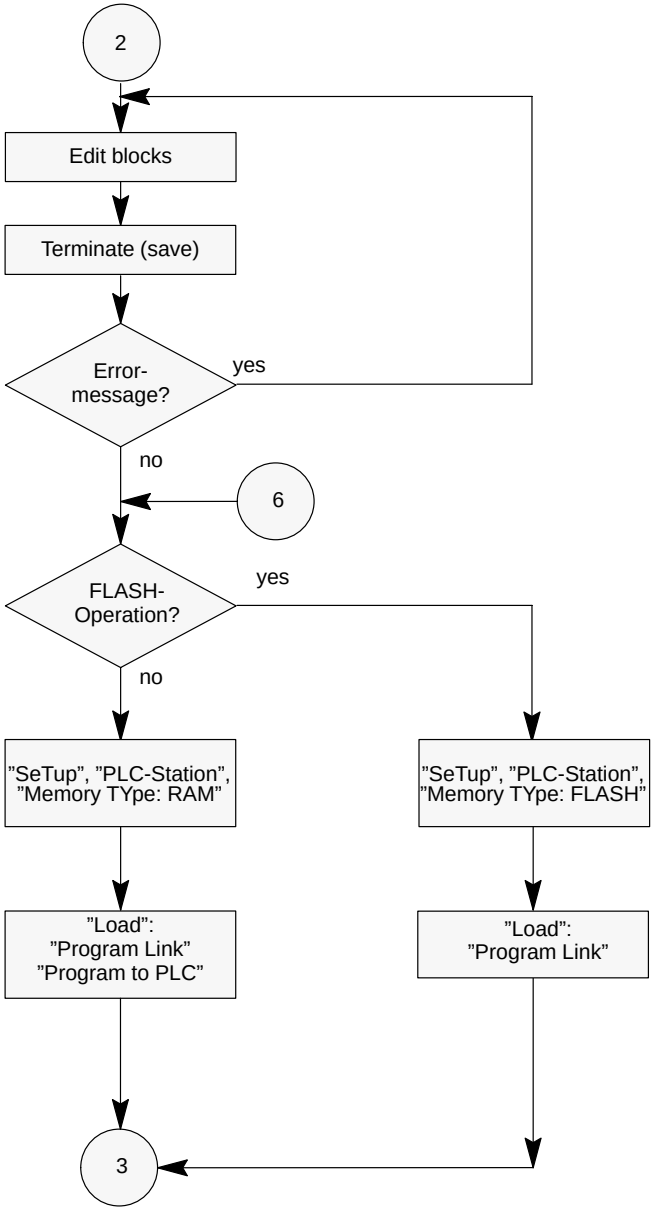


Figure 12 System processing flowchart (Continuation of Figure 11)

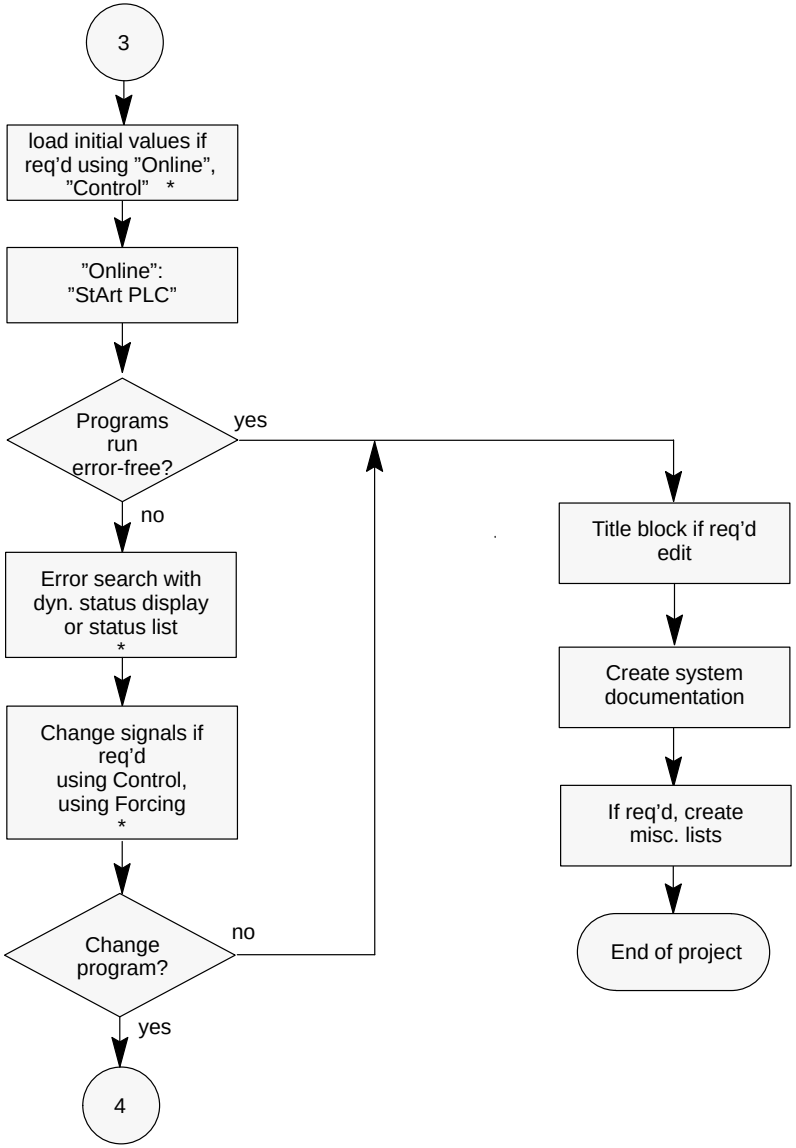


Figure 13 System processing flowchart (continued from Figure 12)

* not possible at EPROM-operation

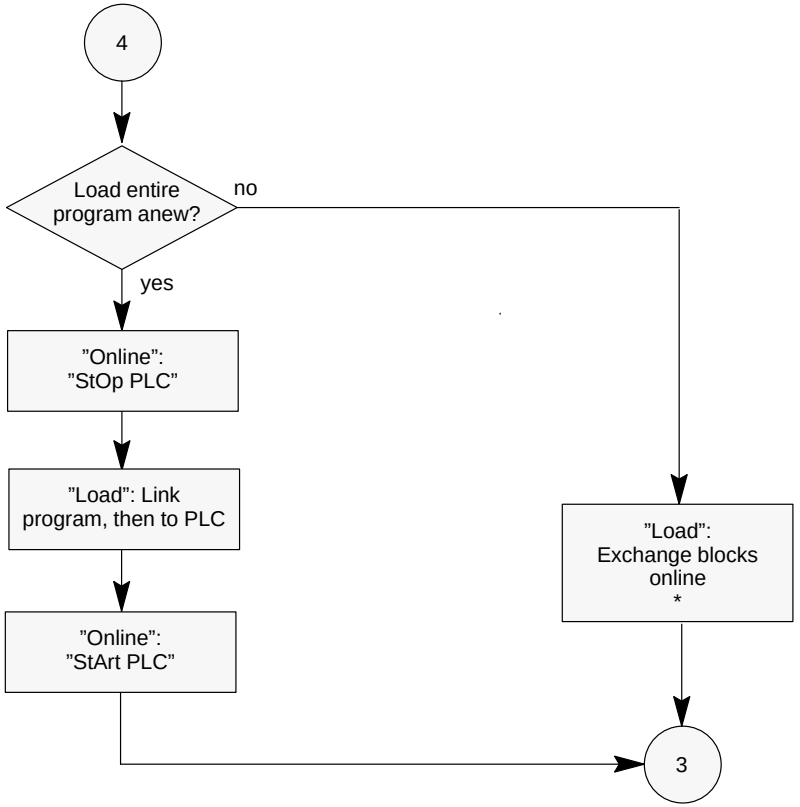


Figure 14 System processing Flowchart (Continued from Figure 13)

* not possible at EPROM-operation

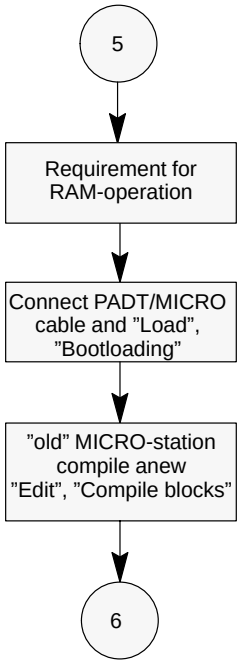


Figure 15 Procedure for ”old” AKF-program and RAM-operation

Chapter 6
Basic MICROAKF Programming Skills

This chapter contains a small, detailed example of how to program the Micro in MICROAKF.

6.1 General

This chapter is to teach you, the AKF beginner, the rudiments of programming an Micro. For this purpose, you will generate a simple program in MICROAKF, copy it into the programmable controller and then observe it, using the dynamic status display.

6.2 Preparations

You should already have carried out the following preparations:

- On the programming panel (in this case P810) you have installed the software in drive C: (see User manual ”Instalation”)

☞ **Note** The sample software program is installed simultaneously.

- For example, an Micro may be given:
  Please remember the connecting cable for PADT ↔ PLC.

6.3 Task Definition

A program is to be designed which will run an eight-bit pattern (”bit string”) across eight bits of an output module (”moving light”). The bit to be used is to be read in through the marker M2.1 to M2.8 and taken over at marker M1.2. It is possible to stop the output by setting M1.1 = 1 (all 8 outputs will be = 0) or to ”freeze” the current state by setting M1.3 = 0. Programming is to be done symbolically, in the technical language ”instruction list”.

The sample plant will be called ”PRACTICE”, the sample program ”MICRO”.

☞ **Note** The program logic is already complete; this is only an exercise for learning to use AKF.

6.4 Parameters of the Sample Program

The following parameters are used in the program:

Table 2 Operands in the Sample Program

Operand	Symbol	Remarks
M1.1	OFF	Off=1: all outputs off Off=0: display
M1.2	LOAD	Bit string is loaded on 0–>1 transition
M1.3	FREE	Free=0: freeze, Free=1: run
M2.1	BIT1	First bit of string
M2.2	BIT2	Second bit of string
M2.3	BIT3	Third bit of string
M2.4	BIT4	Fourth bit of string
M2.5	BIT5	Fifth bit of string
M2.6	BIT6	Sixth bit of string
M2.7	BIT7	Seventh bit of string
M2.8	BIT8	Eighth bit of string
M3.1	HELP1	Help marker for edge detection
M3.2	HELP2	Help marker for edge detection
M3.3	HELP3	Help marker for edge detection
M3.4	HELP4	Help marker for edge detection
MB72	ROTATE	This byte will contain the rotated info
SM15	CLOCK	10.0 Hz flashing cycle
Q1.1	RUN1	Outputs to which the bit string is to be sent in alternation (Moving light)
Q1.2	RUN2	
Q1.3	RUN3	
Q1.4	RUN4	
Q1.5	RUN5	
Q1.6	RUN6	
Q1.7	RUN7	
Q1.8	RUN8	

6.5 Programming

☞ **Note** Menu functions are given in quotation marks, e.g. ”Edit”, ”Block”. Entries which you must type in are written in `Courier`, e.g. `MICROAKF`. Key combinations and special keys are given in brackets, e.g. <Ctrl>+<S>.

6.5.1 Call Program

Step 1 Call the software from user drive C: by means of the following command: `MICRO`

☞ **Note** The marked, highlighted capital letters (reference letters) serve to call up a menu line directly.
The following steps are to be performed in the order given (even though the numbering begins with ”1” anew in each section).

6.5.2 Set Plant/Station

Step 1 Enter `T` for ”SeTup”

React. The setup menu opens

Step 2 Enter `L` for ”PLant”

React. A plant name will be demanded.

Step 1 Enter the following plant name
`C:\PRACTICE` , then <Return>

React. You will be asked whether the plant is to be created.

Step 2 Confirm with YES

React. A station name is requested.

Step 3 Enter the following station name:
`MICRO` , then <Return>

React. You will be be asked whether the station is to be generated.

Step 4 Confirm with YES

React. The main menu will be displayed on the screen.

6.5.3 Program Presettings

Let us now consider the setup settings.

Step 1 Enter `S` for ”PC-Station”

React. The presettings menu opens.

If the presettings are not already complete and correct, carry out the following presettings:

Step 2 Enter `A` for ”Addressng”; toggle until ”SYM” appears (symbolic programming)

Step 3 Enter `D` for ”Data block file”, and enter `0` . To finish, press <Return> again

Step 4 Enter `B` for ”Max. number of blocks”. Enter `20` <Return>

Step 5 Enter `M` for ”Link-mode”, select ”Complete Retranslation” by means of the cursor keys and press <Return> again

Step 6 Enter `Y` for ”Memory TYpe” and toggle until ”RAM” appears

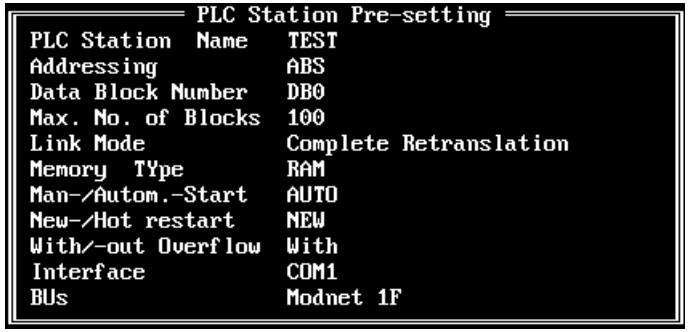
Step 7 Enter S for ”Manual/Autom. Start and toggle until ”AUTO” appears

Step 8 Enter H for ”New/Hot Restart” and toggle until ”NEW” appears

Step 9 Enter o for ”Without/With Overflow” and toggle until ”With” appears

Step 10 Enter I for ”Interface” and toggle until ”COM1” appears

Step 11 Enter U for ”BUs” and toggle until ”Modnet 1F” appears



Step 12 Press <Esc> twice

React. The menus are closed and the selection bar is on ”SeTup” in the main menu. The entered presettings have now been accepted under Setup.

6.5.4 Equipment List

Step 1 Move to ”Edit” using <←> and <Return>

React. The edit menu opens.

Step 2 Enter Q for ”EQuipment List”

React. The equipment list editor opens and can be seen.

Step 3 Press <Ctrl>+<Return> to open the menu

Step 4 Type D for module entry

Step 5 C CPU

Step 6 With <↓> to ”512”

Step 7 Confirm with <Return> entry

6.5.4.1 End equipment list and save

Step 1 Press <Ctrl>+<Return> to open the menu

Step 2 Press T (for Terminate) to save the equipment list and leave the equipment list editor.

React. The edit menu comes on the screen again.

6.5.5 Assign symbols and comments (SYMKOM-blocks)

As symbolic programming has been provided for, the signal symbols must be assigned. This is best done before creating the program.

☞ **Note** The numbering below will cover several chapters.

Step 1 Enter `S` for ”Symbols and Comments”

React. The ”Symbols and Comments” menu opens.

Step 2 Enter `S` for ”Start Entry”

React. The symbol and comment editor opens. All I/O’s which are not available, according to the equipment list, will be marked with a *. Under ”Symbol” will be upper-case script, under ”Comments” it will be lower-case script.

6.5.5.1 Assign input symbols

Step 3 Press <Ctrl>+<Return> to open the handling menu

Step 4 Enter `F` for ”Search Function”

React. A window will appear, into which you can enter the desired signal.

Step 5 Enter `M1.1` and <Return>

React. The cursor will jump to the ”Symbol” column of signal M1.1.

Step 6 Enter `OFF` and <Tab>

React. The cursor jumps to the ”Commentary” column

Step 7 Enter the following as comments: `Off=1: all outputs off, Off=0: display` and confirm with <Tab>

React. The entry took place in line M1.1 and the cursor is now in the line for the signal M1.2.

Step 8 Enter : `LOAD` <Tab> `Bit string is loaded on 0->1 transition` <Tab>

React. The entry took place in line M1.2, and the cursor is now in the line for the signal M1.3.

Step 9 Enter : `FREE` <Tab> `Free=0: freeze, Free=1: run` <Tab>

React. The entry took place in line M1.3 and the cursor is now in the line for the signal M1.4.

Step 10 Press <Ctrl>+<Return> to open the handling menu

Step 11 Enter `F` for ”Search Function”

Step 12 Enter `M2.1` and <Return>

React. In the editor, the cursor jumps to the ”Symbol” column of the given signal.

Step 13 Enter the following text for M2.1 to M2.8:
`BIT1` <Tab> `First bit of string` <Tab>
`BIT2` <Tab> `Second bit of string` <Tab>
`BIT3` <Tab> `Third bit of string` <Tab>
`BIT4` <Tab> `Fourth bit of string` <Tab>
`BIT5` <Tab> `Fifth bit of string` <Tab>
`BIT6` <Tab> `Sixth bit of string` <Tab>
`BIT7` <Tab> `Seventh bit of string` <Tab>
`BIT8` <Tab> `Eighth bit of string` <Tab>

React. The entries took place in lines M2.1 to M2.8 and the cursor is now positioned in the next signal line: M2.9.

Step 14 Press <Ctrl>+<Return> to open the handling menu

Step 15 Enter `F` for ”Search Function”

Step 16 Enter `M3.1` and <Return>

Step 17 Enter the following text for M3.1 to M3.4:
`HELP1 <Tab> Help marker for edge detection <Tab>`
`HELP2 <Tab> Help marker for edge detection <Tab>`
`HELP3 <Tab> Help marker for edge detection <Tab>`
`HELP4 <Tab> Help marker for edge detection <Tab>`

Step 18 Press <Ctrl>+<Return> to open the handling menu.

Step 19 Enter `F` for ”Search Function”

Step 20 Enter `MB72` and <Return>

React. The cursor jumps to the first symbol character of signal MB72.

Step 21 Enter at MB72:
`ROTATE` <Tab> `This byte will contain the rotated info`

Step 22 Press <Ctrl>+<Return> to open the handling menu.

Step 23 Enter `F` for ”Search Function”

Step 24 Enter `SM15` and <Return>

Step 25 Enter the following text : `CLOCK`

6.5.5.2 Assign Output Symbols

Step 26 Press <Ctrl>+<Return> to open the handling menu

Step 27 Enter `F` for ”Search function”

Step 28 Enter `Q1.1` and <Return>

React. In the editor, the cursor jumps to the ”symbol” column of the given signal.

Step 29 Enter the following text for Q1.1 to Q1.8:
`RUN1` <Tab> `Outputs Q1.1-1.8 will be used` <Tab>
`RUN2` <Tab> `for the moving light .`<Tab>
`RUN3` <Tab> `The string entered at M2.1-2.8` <Tab>
`RUN4` <Tab> `will start moving,` <Tab>
`RUN5` <Tab> `when a 0->1 transition` <Tab>
`RUN6` <Tab> `is detected.` <Tab>
`RUN7` <Tab> <Tab>
`RUN8`

Step 30 Enter <Ctrl>+<Return> to open the handling menu.

Step 31 Enter `T` for ”Terminate”

React. That ends the SYM/KOM-block editing, and the texts will now be saved.

Step 32 Enter <ESC>

React. The edit menu is again on display.

6.5.6 Edit program (blocks)

In this chapter, the user program will be entered in MICROAKF.

6.5.6.1 Open block editor

Step 1 Enter `B` for ”Blocks”

React. The block menu opens.

Step 2 Enter `B` for ”Block number”

Step 3 Enter `1` and <Return>

Step 4 Enter `N` for ”Network number”

Step 5 Enter `1` and <Return>

Step 6 Enter `T` for ”Block Type” and toggle until ”PB” appears

Step 7 Enter `I` for ”Input Mode” and toggle until ”IL” appears

Step 8 Enter `A` for ”Addressing” and toggle until ”SYM” appears

Step 9 Enter `S` for ”Start Input”

React. The block editor opens, and the last network of PB1 is displayed, with ”BE” for the block end.

6.5.6.2 Edit PB1

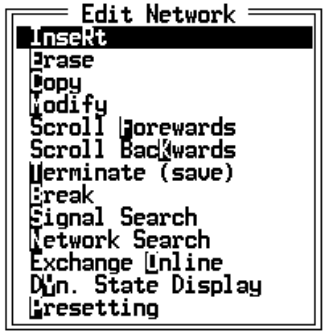
First the program block (PB) is edited, in which the program for the moving light is found. PB1 consists of network 1 with the program and network 2 with ”BE” for the block end.

❒ **Insert network**

An empty block always consists only of network 1, in which the block end is found. In order to edit the block, an empty network must first be inserted.

Step 1 Enter <Ctrl>+<Return>

React. The handling menu opens



Step 2 Enter R for ”Insert” (Network)

React. The ”Insert” function always inserts a new network before the current network. In this case, network 1 is empty (contains only :***), while Network 2 contains ”BE” for block end.

❒ **Edit network 1**

Now, the program instructions will be created in network 1. The program contains several jumps, which should normally be entered by instruction list (IL). For this reason, the entire program will be created in IL, although this is not absolutely necessary.

Step 1 Enter <Ctrl>+<Return>

React. The handling menu opens

Step 2 Enter R for ”Insert Line”

Step 3 Press <Ctrl>+<R> several times, so that more lines are inserted (the program contains a total of 24 lines)

Step 4 Enter the following lines now. While editing, you can insert further empty lines above the cursor by pressing <Ctrl>+<E>.

```
A <Tab> OFF <Return>
JF <Tab> =JUMP1 <Return>
LD <Tab> V0 <Return>
= <Tab> ROTATE <Return>
JI <Tab> =JUMP3 <Return>
```

Step 5 For the jump position JUMP1, move the cursor to the left edge of the screen, using the cursor keys:

```
JUMP1 <Tab> (to :) A <Tab> LOAD <Return>
```

Step 6 Enter the other lines as per Step 4 :

```
EDP <Tab> HELP1 <Return>
=C <Tab> HELP2 <Return>
JF <Tab> =JUMP2 <Return>
LBB <Tab> BIT1 <Return>
DBB <Tab> CNT8 <Return>
= <Tab> ROTATE <Return>
```

Step 7 Enter the following line as per Step 5 :

```
JUMP2 <Tab> (to :) A <Tab> CLOCK <Return>
```

Step 8 Enter the following lines as per Step 4 :

```
A <Tab> FREE <Return>
EDP <Tab> HELP3 <Return>
=C <Tab> HELP4 <Return>
JF <Tab> =JUMP4 <Return>
A <Tab> ROTATE <Return>
ROL <Tab> V1 <Return>
= <Tab> ROTATE <Return>
```

Step 9 Enter the following line as per Step 5 :

```
JUMP3 <Tab> (to :) LD <Tab> ROTATE <Return>
```

Step 10 Enter further lines as per Step 4 :

```
TBB <Tab> RUN1 <Return>
DBB <Tab> CNT8 <Return>
```

Step 11 Enter the following line as per Step 5 :
`JUMP4` <Tab> (to :) `NOP` <Return>
:***

❒ **Erasing superfluou empty instruction lines**

The network closes with :***. Superfluous empty lines can be erased by proceeding as follows:

Step 1 Move the cursor into the empty line, using cursor keys

Step 2 Press <Ctrl>+<Return>

Step 3 Enter `E` for ”Erase”

❒ **End network and save**

Step 1 Press <Ctrl>+<Return> to open the handling menu

Step 2 Enter `T` for ”Terminate”

❒ **End block and save**

Step 1 Press <Ctrl>+<Return> to open the handling menu.

Step 2 Press `T` for ”Terminate (Save)”

Block PB1 is now finished, and the ”Edit Block” menu again shows up on the screen. PB1 must now be linked into the organization block, since the latter holds all the keys in one hand, so to speak. Without the organization block, the program cannot be run.

6.5.6.3 Edit OB1

❒ **Open OB**

Step 1 Enter T for ”Block Type” and toggle until ”OB” appears

Step 1 Enter B for ”Block number”

Step 2 Press 1 and <Return>

Step 3 Enter S for ”Start Input”

React. The block editor opens, and the last network of OB1 appears, with ”BE” for the block end.

Step 4 Enter <Ctrl>+<Return> to open the handling menu.

Step 5 Enter R for ”Insert” (Network)

React. The ”Insert” function always inserts a new network before the current one. In this case, network 1 is empty (contains only :***), and network 2 contains ”BE” for the block end.

❒ **Call a PB in OB (unconditionally, i.e. the PB will be called for every scan)**

Step 1 Press <Ctrl>+<Return> to open the handling menu.

Step 2 Enter R for ”Insert Line”

Step 3 Enter the following text for the block call:
BC <Tab> PB1 <Return>

❒ **Terminate and save OB**

Step 1 Press <Ctrl>+<Return> to open the handling menu.

Step 2 Enter T for ”Terminate” (Network)

Step 3 Press <Ctrl>+<Return> to open the handling menu.

Step 4 Enter T for ”Terminate” (Block)

The program has now been entered.

Below, a printout of the new program is shown.

```
C:\PRACTICE\MICRO\OB1
AEG Modicon Modsoft AKF: Program Protocol

NETWORK: 0001

      :BC   PB 1
      :***

NETWORK: 0002

      :BE
```

```
C:\PRACTICE\MICRO\PB1
AEG Modicon Modsoft AKF: Program Protocol

NETWORK: 0001

      :A    OFF
      :JF   =JUMP1
      :LD   V 0
      :=    ROTATE
      :JI   =JUMP3
JUMP1 :A    LOAD
      :EDP  HELP1
      :=C   HELP2
      :JF   =JUMP2
      :LBB  BIT1
      :DBB  CNT 8
      :=    ROTATE
JUMP2 :A    CLOCK
      :A    FREE
      :EDP  HELP3
      :=C   HELP4
      :JF   =JUMP4
      :A    ROTATE
      :ROL  V 1
      :=    ROTATE
JUMP3 :LD   ROTATE
      :TBB  RUN1
      :DBB  CNT 8
JUMP4 :NOP
      :***

NETWORK: 0002

      :BE
```

6.6 Link program

Below, the program is prepared for the PLC.

Step 1 Go back to the main menu by leaving the opened menu windows using <Esc>.

Step 2 Enter L for ”Load”

React. The load menu opens

Step 3 Enter L for ”Link”

React. The program is linked

Step 4 Acknowledge the report

The program is now ready for transfer to the PLC.

6.7 Transferring the program to the PLC

The following function is used to transfer the program, the equipment list and the initial values into the PLC.

Step 1 Enter P for ”Program to PLC”

React. The program was transferred. It can now be started.

6.8 Starting the program

Step 1 Use < → > to change to ”Online”

React. The load menu closes and the online menu opens

Step 2 Enter A for ”StArt PLC”

React. You will be asked to confirm that you really want to start

Step 3 Confirm with YES

React. The run LED on the MICRO will light up.

6.9 Setting and Changing the Parameters

In order to effect an output on Q1.1 to Q1.8, proceed as follows.

6.9.1 Online list

First, a list will be created, in which you will enter and control the signals.

Step 1 After starting the PLC, you will be in the online menu. If not, leave the menu you are in by pressing <Esc> and enter 0 for ”Online”.

Step 2 Enter L for ”Online List”

React. The menu Online List Load/Erase will appear

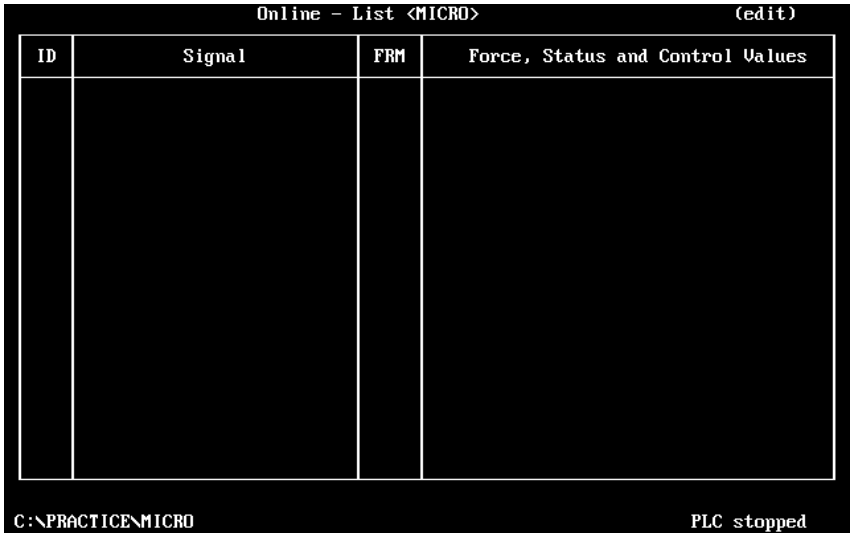
Step 3 Enter L for ”Online List Load”

Step 4 Enter MICRO and <Return>

React. You will be asked to confirm whether you want to start a new list.

Step 5 Confirm with YES

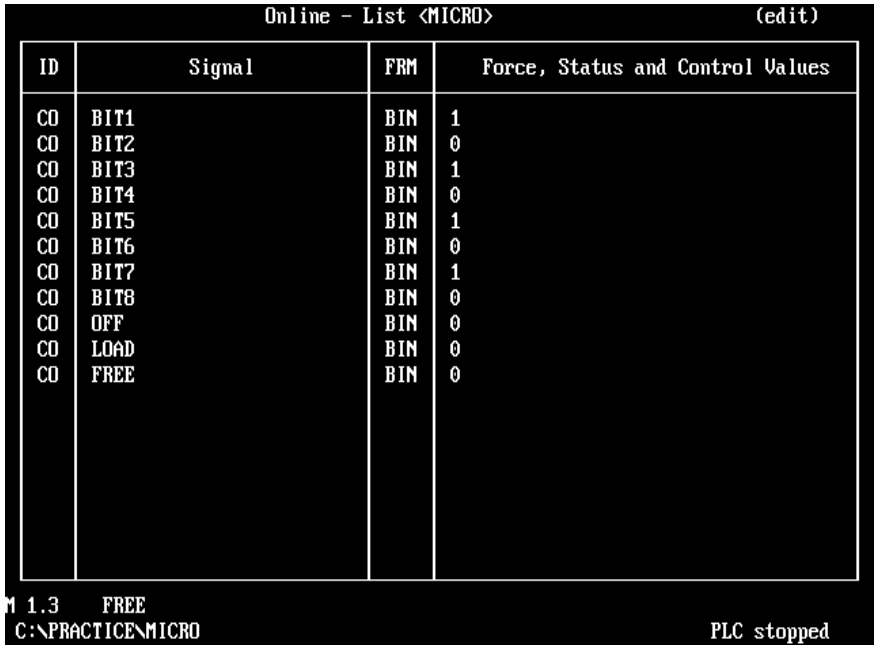
React. The window with the list editor will open. The cursor is positioned in the upper left column.



Step 6 Enter the following text in the sequence specified :

```
CO <Tab> BIT1  <Tab>  1 <Return>
CO  <Tab> BIT2 <Return> <Return>
FO  <Tab> BIT3 <Tab>  1  <Return>
CO  <Tab> BIT4 <Return> <Return>
CO  <Tab> BIT5 <Tab>  1  <Return>
CO  <Tab> BIT6 <Return> <Return>
CO  <Tab> BIT7 <Tab>  1  <Return>
CO  <Tab> BIT8 <Return> <Return>
CO  <Tab> OFF <Return> <Return>
CO  <Tab> LOAD <Return> <Return>
CO  <Tab>  FREE  <Return> <Return>
```

React. The signals have been completely entered.



Step 7 Move the cursor to the right column of FREE

Step 8 Set the value to ”1”

Step 9 Press <Ctrl>+<Return> to open the handling menu.

Step 10 Enter C to switch on ”Control”

Step 11 Set the value to ”0”

Step 12 Press <Ctrl>+<Return> to open the handling menu.

Step 13 Enter C to switch on ”Control”

React. After initialization (FREE = 1) all outputs are off

Step 14 FREE to ”1”

Step 15 Press <Ctrl>+<Return> to open the handling menu.

Step 16 Enter C to switch on ”Control”

React. The moving light is enabled (”run”)

Step 17 LOAD ”1” (0→1 edge)

Step 18 Press <Ctrl>+<Return> to open the handling menu.

Step 19 Enter C to switch on ”Control”

React. The bit array of the markers (BIT1...BIT8) is set to the outputs and will be rotated. Now the program is in its desired final status.

The bit array can be changed at any time and transfered new with LOAD.

Step 20 Press <Ctrl>+<Return> to open the handling menu.

Step 21 Enter T for ”Terminate”

React. You are asked whether you want to save the list.

Step 22 Confirm with YES

Step 23 Leave the menu ”Online List” with <ESC>

The Online List is now completed. Additionally, the signal status can now be seen in the dynamic status display.

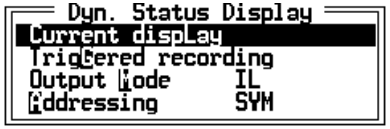
6.10 Online list/Dyn. Status Display

6.10.1 Dynamic Status Display

Now the dynamic status display (”current display”) will be carried out.

Step 1 In the online menu: Enter `D` for ”Dyn. Status Display”

React. The menu opens



Step 2 Enter `L` for ”Current Display”

Step 3 Enter `B` for ”Block number”

Step 4 Enter `1` and <Return>

Step 5 Enter `T` for ”Block Type” and toggle until ”PB” appears

Step 6 Enter `N` for ”Network number”

Step 7 Enter `1` and <Return>

Step 8 Enter S for ”Start Display”

React. Network 1 of the PB1 switches in

Now you can page through the network with the cursor keys, or use <PgDn> and <PgUp> to page to another network, Use <Ctrl>+<Return> to open a handling menu, in which several other functions are available.

In order to observe signal states one scan cycle at a time (e.g. for diagnostic purposes) use the ”Triggered Recording”.

6.11 Remarks on the Program-Documentation

In the ”Print” you can carry out the program documentation. With ”Complete Documentation”, all important data are output on one printout (with the printer’s contents listing).

You have the choice of sending the lists to a screen, into a file or to a printer.

The files can be processed with any ASCII editor. You will be in the current station directory. Assign the name of the file directly when selecting ”Output Unit”, ”File”.

When outputting to a printer, the printer must be reset. Resetting is carried out in the menu ”SeTup”, ”Print”.

☞ **Note** Please note the menu documentation in the user manual ”Configuration”.

6.12 Remarks on Backing Up

In the menu ”Special”, you can back up the entire station or restore it.

☞ **Note** Please note the menu documentation in the user manual ”Configuration”.

Chapter 7
Programming Example

The software contains a sample program of a three-station plant to explain possibilities of structuring using Modsoft AKF.
The structure of the program is explained in this chapter, but a detailed program listing is not provided.
These listings can be printed using the Modsoft AKF function ”Print”.

7.1 General

The plant in this example is a sequence control system for an automatic rotating table with three stations, each station being divided into eight steps of the sequence control. The following description focusses on aspects that illustrate the different procedures in configuration.

Three identical stations of an automatic rotating table are taken to show three approaches to creating a structured user-program.

This program is not intended to write fully the sequence control for an automatic rotating table, but to provide insight into the different structuring possibilities of programs in Modsoft AKF software.

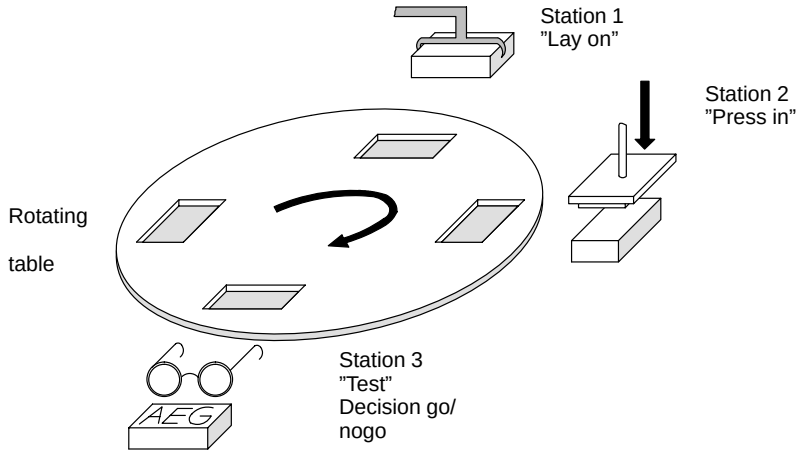


Figure 16 Illustration of Automatic Rotating Table

7.2 Task

The automatic rotating table consists of three stations and the rotation. The emphasis was placed on the stations. The mode selector is only shown by simplified representation of the subsystems, referred to the individual stations.

The rotary movement of the table is not dealt with either, because this is in principle like the functional sequence of a station.

The example is a pneumatically controlled system with, in part, 100% drive, the pince being open and in the top left end position in the de-energized state. When the machine is in the initial state ”Top/Left” and the mode selector is other than ”Hand”, the first step is started after actuation of the material-flow release button.

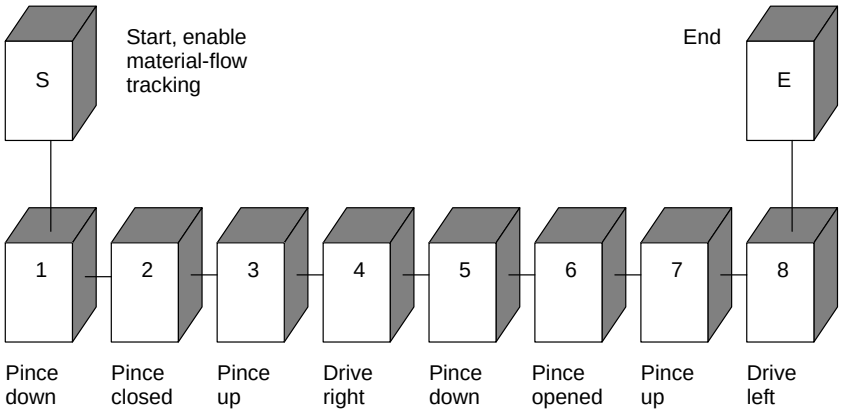


Figure 17 Sequence of Steps in Station

When the pince is open and again in the initial state ”Top/Left”, the ready signal of the particular station is issued for this sequence. When the rotation has been completed, the ready signals are cleared.

The eight steps are the same in each station. For this reason, each station is described with a different solution, all stations being implemented in a single organization block.

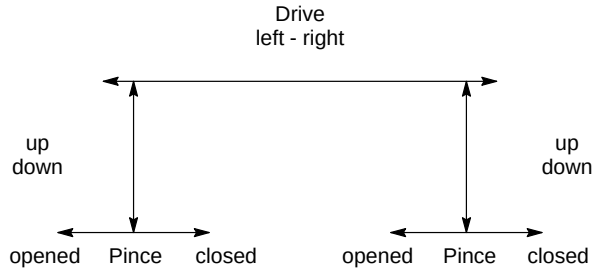


Figure 18 Movements of pince in station

Each program part (describing one station) has its advantages and drawbacks. The extent to which the person configuring should make use of the structuring possibilities that are offered is dependent on a number of criteria, e.g.

- the scan time,
- the program length (memory capacity),
- the available programming time.

The following shows a mode switch for each plant, which enables each station to be operated in another mode. The general mode selector will be neglected in the following descriptions of the three stations.

☞ **Note** The block numbers in the following schematics correspond to the station ”Example” that is supplied on the diskettes.

7.3 Station 1

Example 1 describes a project that amounts to the same as sequential processing without fine structures. This station only contains program blocks. In this way the structure of the station is most easily recognized. This program structure is advisable for smaller systems, where less emphasis is placed on speed. The configuring effort is greater compared to stations 2 and 3 because here there are no application-specific function blocks. However, no FB-local formal parameters have to be declared in this case. The degree of complexity remains low because everything runs purely sequentially.

But this also means that all blocks have to be processed and tested in the scan. The pince may not open, for instance, when the drive right/left is executed.

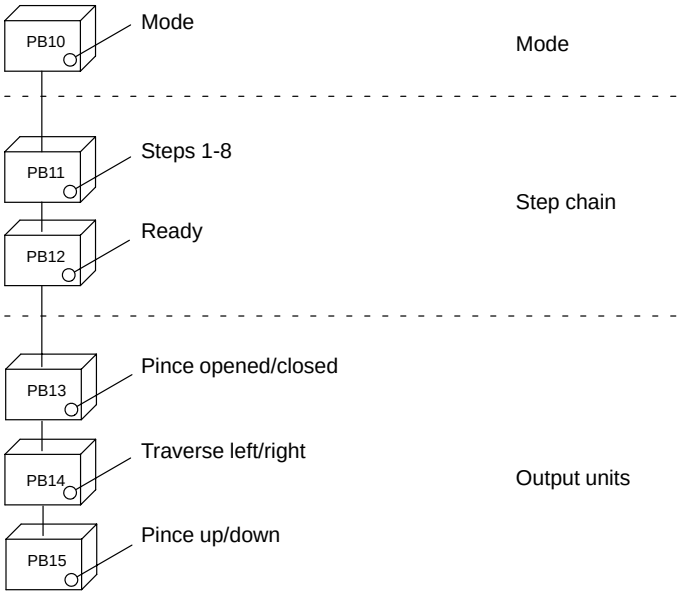


Figure 19 Schematic of Station 1

7.4 Station 2

Example 2 shows sensible structuring with program and function blocks. The processing of step chain 1-4 and 5-8 is event-controlled. This station offers optimal transparency for configuration and maintenance.

Example 2 is a configuration example for medium-sized and large systems. Application-specific function blocks are used, which only have to be parameterized after integration in a program block. Thus, compared to example 1, the configuration effort is reduced and less memory is necessary, because FB23 for instance is only programmed once (i.e. only occupies one memory location) but is called up three times with changing parameters. The event-controlled processing of the step chains (steps 1-4 and 5-8) permits the scan times to be reduced in large systems. As well, memory is saved by using application-specific function blocks.

The more steps are used, the better the utilization of memory becomes.

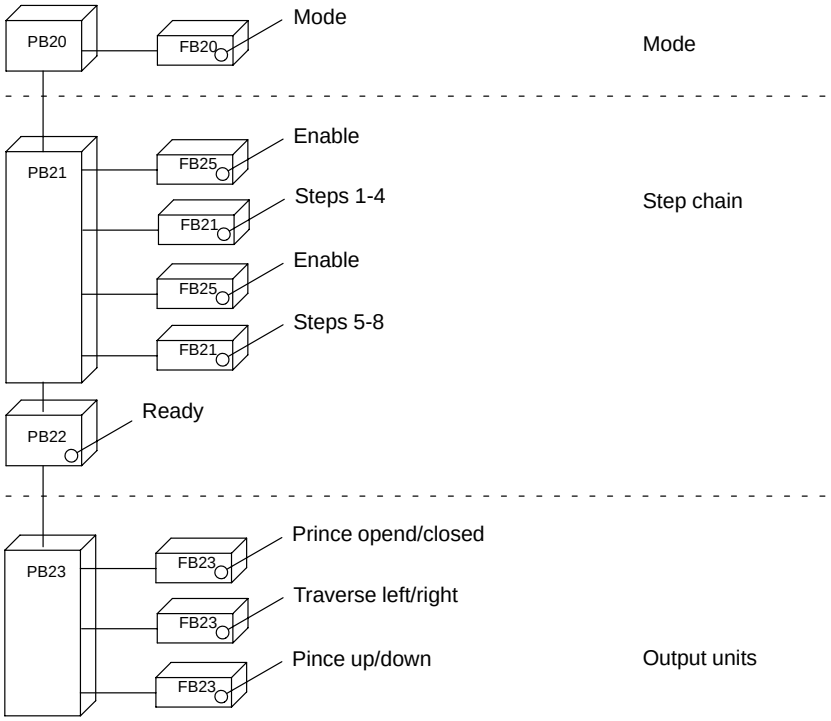


Figure 20 Schematic of Station 2

7.5 Station 3

Example 3 differs from example 2 by its finer structuring. Here too, as in example 2, the step chain is divided into two units, only one being enabled for event control at a time. In contrast to station 2, the output unit is made up of six block units. This structure is only meaningful for complex automation tasks where emphasis is also placed on the shortest possible scan times. Each of the six output units is enabled event-controlled. The planning effort for example 3 is greater than in example 2. In station 3 the enable blocks are parameterized directly in organization block OB1. This avoids calling up a program block. The tasks of the drive units have been divided up again, once more saving scan time compared to example 2. However then only parameter modifications are possible while the program is running if OB1 is exchangeable.

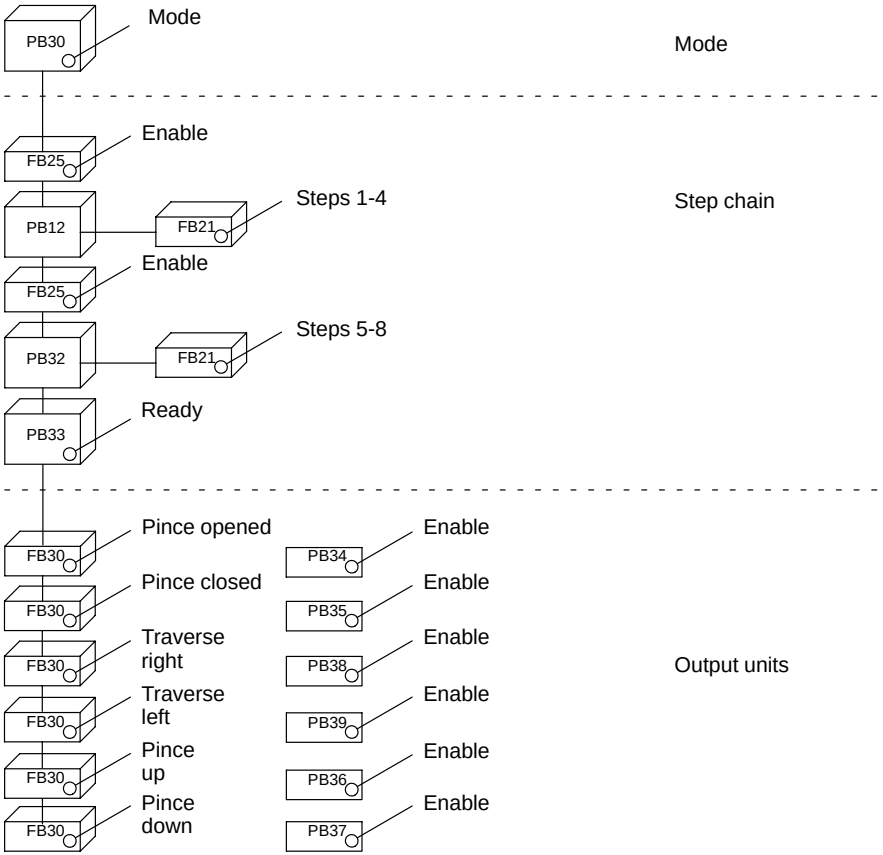


Figure 21 Schematic of Station 3

Index