Kuhnke Electronics
Programming Manual
for KUAX 644, 657P, 680C, 680I and KDT 680CT

E 417 GB 19 September 1996 / 71.317

This manual is primarily intended for the use of the designing, the project planning, and the developing engineers. It does not give any information about delivery possibilities. Data is only given to describe the product and must not be regarded as guaranteed properties in the legal sense. Any claims for damages against us – on whatever legal grounds – are excluded except in instances of deliberate intent or gross negligence on our part.
We reserve the rights for errors, omissions or modifications.

Table of contents

1. Introduction

1.1. Target systems

This instruction manual is concerned with the programming of the user programs for the following PLC systems:

- **KUAX 644 PC Control**
  is used as a slot card in the PC which communicates with decentralized I/Os and other controllers via its integrated PROFIBUS interface;

- **KUAX 657P Profi Control**
  modular design in a 19" rack, with configuration options of up to 1024 I/Os, networkable via PROFIBUS due to PROFIBUS module;

- **KUAX 680C Compact Control**
  compact mini-controller equipped with inputs/outputs and counters, can be extended by 4 modules for further I/Os and functions if bus board exists;

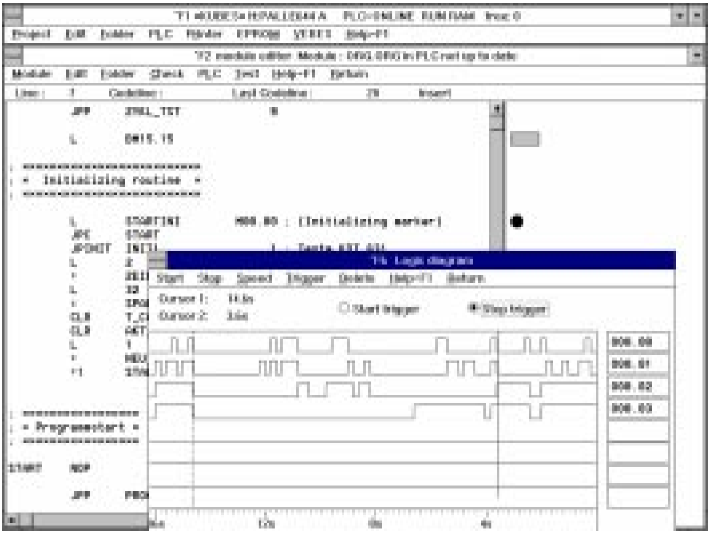
- **KDT 680CT Control Terminal**
  same as KUAX 680C but also designed as a operating terminal;

- **KUAX 680I Profi Control**
  modular mini-controller with slots for 4 or 8 modules, equipped with an integrated PROFIBUS interface.

All above devices have the same processor and the same programming aid is used for programming. This makes changing systems a lot easier.

1.2. Programming aid KUBES

KUBES is the programming tool. This is the Kuhnke User Software that runs on commercially available PCs under MS-Windows, the comfortable user interface.
Here you have the possibility of working in several windows to use various KUBES functions at the same time, a feature that is very convenient for commissioning and servicing:


Project Edit Folder PLC Window EPROM VERIFY Help-F1
Module Edit Folder Check PLC Test Help-F1 Return
T4: Logic diagram
Start Stop Speed Trigger Delete Help-F1 Return
○ Start trigger
● Stop trigger
M00.00
M00.01
M00.02
M00.03

```
        JPP     ZYKL_TST

        L       DW15.15

; ************************
; *  Initializing routine  *
; ************************

        L       STARTINI        M00.00 ; [Initializing marker]
        JPC     START

; * Programmstart *

START   NOP

        JPP
```

Figure above: screenshot of a typical KUBES screen with the KUBES Main Menu bar and the Module Editor visible as well as a dynamic (in this case: graphic) display and logic diagram for monitoring the signal changes of selected operands.

Please refer to the KUBES Beginner's Manual
E 327 GB, KUBES
to learn about the basic functionality of KUBES.
For further information please refer to the KUBES online help.

1.3. Working principle of CPU

The purpose of the CPU is to control the processing of the program. The microprocessor for the user program has two resource programs for its instructions:

- The monitor program
  which contains the system properties of the controller. This program is delivered with the device.

And the

- user program
  which contains the programs for controlling the machine or plant. These programs are written under the KUBES programming tool. Additional C tasks can be embedded into them.

1.3.1. Multitasking

The CPUs of the described controllers are based on so-called multitasking operations. A high-priority time control function ensures that all tasks have the same share of the overall processing time. Every task has a maximum of 1 ms available before it is interrupted to run the next task requesting processing time. In the next cycle, the task is resumed at the same point where it was interrupted in the previous cycle.
The following are typical tasks: user program, processing of KUBES commands, dialog terminal communication activities, C tasks (if implemented), PROFIBUS operations (if implemented).

1.4. Other programming aids

Apart from writing user programs as described in this manual, the system has other options for working out complex software solutions for controllers.

1.4.1. KUBES modules

Kuhnke provide a large number of KUBES modules as so-called black box program packages. A good many of them are delivered with KUBES. You can use them immediately after installation of this user software.
Other modules can be obtained as separate program packages on disk.

☞ *To learn more about KUBES modules please refer to the folllowing instruction manual:*
E 386 GB, KUBES Modules.

1.4.2. Programming in the C high-level programming language

Apart from the normal user program it is also possible to embed program sections that were written in the C high-level programming language. These programs are called C tasks. They are normally written by Kuhnke developers as standard software packages or customer-specific solutions for specific tasks. Upon request, Kuhnke offer training programmes for users interested in learning more about writing their own C tasks. A prerequisite for participation is a solid knowledge of C language programming.

1.4.3. Networking software

Controllers KUAX 644, 657P and 680I can be networked via PROFIBUS. That means that they can be incorporated in a network to communicate with other devices such as decentralized I/Os (e.g. KUAX 680S), frequency converters, other controllers etc.

☞ *PROFIBUS is not described in this manual. Please refer to the following instruction manual:*
E 365 GB, PROFIBUS

2. Safety information

2.1. Target group

This instruction manual contains all information necessary for the use of the described product (software, control device, modules) according to instructions. It addresses the **personnel of the construction, project planning, service and commissioning departments**. For proper understanding and error-free application of technical descriptions, instructions for use and particularly of notes of danger and warning, **extensive knowledge of automation technology** is compulsory.

2.2. Reliability

Reliability of Kuhnke controllers is brought to the highest possible standard by extensive and cost-effective means in their design and manufacture.

These comprise:
software specification
rough and detailed software design
extensive documentation of all steps of development
testing of individual software modules
integration test of the software package
application test software ↔ controllers
integration into the KUHNKE Quality Assurance system
transferring the finished software to error free installation disks.

Despite these measures, we cannot warrant a perfect functioning of our software, as there may, for example, be unexpected hardware configurations that might cause unforeseeable malfunctions.

2.3. Notes

Please pay particular attention to the additional notes which we have marked by symbols in this instruction manual:

2.3.1. Danger

⚠ *This symbol warns you of dangers which may cause death, (grievous) bodily harm or material damage if the described precautions are not taken.*

2.3.2. Danger caused by high contact voltage

⚡ *This symbol warns you of dangers of death or (grievous) bodily harm which may be caused by high contact voltage if the described precautions are not taken.*

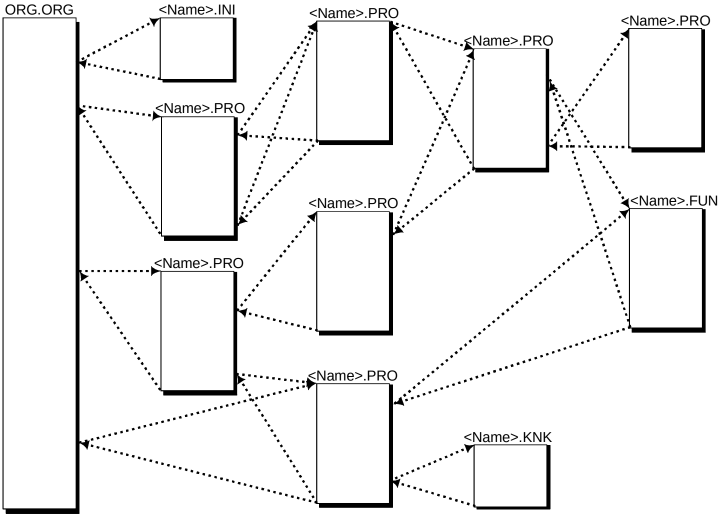
2.3.3 Important information / cross reference

☞ *This symbol draws your attention to important additional information concerning the use of the described product. It may also indicate a cross reference to information to be found elsewhere.*

3. Software modules

The user program of the controllers is written as a module structure. This gives you the opportunity to arrange the technological problem to be controlled as separate sub-tasks. The individual modules form a hierarchical system on a maximum of 5 levels. Modules on higher levels can call up modules on lower ones.
A program with this kind of structure is very clear and considerably helps understanding and reviewing finished programs.

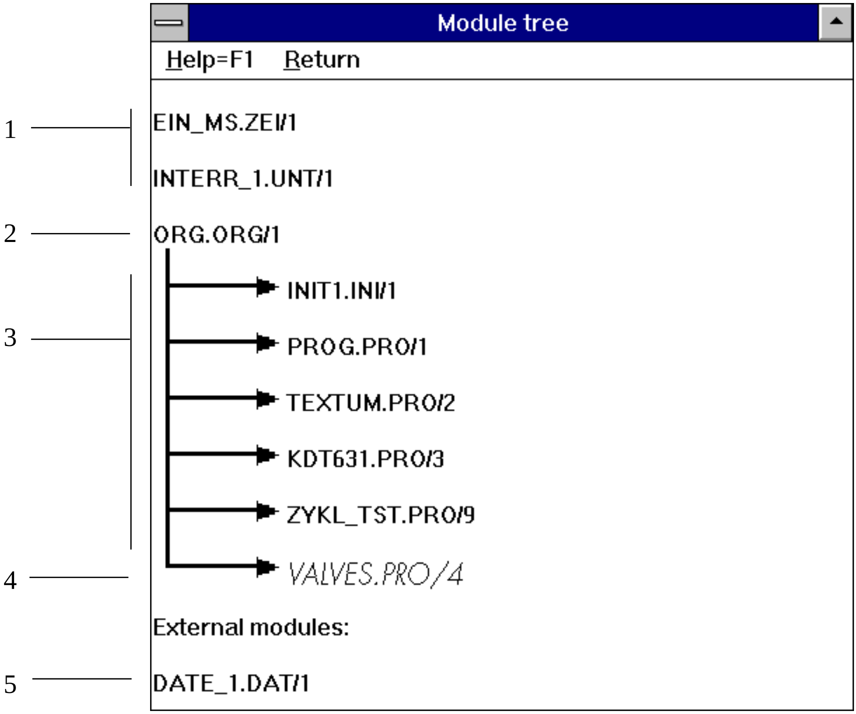
Example of a module hierarchy:



👉 *There are also modules which are independent of this module hierarchy. They are not called up by a command but by interrupts. This group of modules consists of timer and interrupt modules.*

KUBES module overview:

The KUBES programming software has a feature that allows displaying and printing the entire module hierarchy of a project. All modules that belong to the project are shown:



EIN_MS.ZEI/1

INTERR_1.UNT/1

ORG.ORG/1

- INIT1.INI/1
- PROG.PRO/1
- TEXTUM.PRO/2
- KDT631.PRO/3
- ZYKL_TST.PRO/9
- *VALVES.PRO/4*

External modules:

DATE_1.DAT/1

Legend:

1 Automatically called up modules: interrupt modules (.INT) and timer modules (.TIM)

2 Organization module (ORG.ORG)

3 Modules called up by command in the program: program (.PRO), function (.FUN) and initialization modules (.INI).

4 Virtual modules (in italics); they are called up by command, but they have not been edited (they are empty).

5 External modules are part of the project but they are not being called up at present.

Organization of module calls:
The modules containing program code (there are also the data module which are modules without program code) represent a kind of complete sub-programs. The way in which the modules are organized already takes care of the return jump to the location from where a module was called up. The user need not program the return. Interrupt-controlled modules work very much the same way.
Dies gilt auch für die interruptgesteuerten Bausteine.

⚠ *Program lines to jump back to the calling module are rejected by an error message. This also happens if you want a module to call itself up.*

Types of modules:
The following types of modules can be distinguished:

- Organization module
- Program modules
- Function modules
- Timer modules
- Interrupt modules
- Initialization modules
- Trigger modules
- KUBES modules
- Data modules

Watchdog:
The watchdog monitors the program runtime. The program run is stopped and an error message output (system error marker ERR00.00 = 3) if the program runtime is exceeded.

- Processing of the individual modules is monitored by a separate watchdog time. There are approximately 70 ms available for the processing of a module.
- Another watchdog time monitors the overall program (cycle time). A watchdog error is output if the organization module is not processed again after a maximum of 2 s.

☞ *You can measure the cycle time of your program yourself. In appendix "B. Measuring the cycle time" we are presenting the program "ZYKL_TST.PRO".*

3.1. Organization module

Features:

Name:	ORG.ORG
Number:	1
Length:	max. 253 lines incl. max. 128 code lines
Function:	organization of the overall program
Call:	automatically at the beginning of each program cycle

The organization module is the main program module of a project. KUBES automatically creates it when you create a new project (see KUBES Beginner's Manual, E 327 GB). This module contains the branching instructions to all other modules.
For reasons of expediency, the programming of the organization module should include program selection and calling up of the modules responsible for overall tasks.
All commands are applicable without limitations (commands of the initialization modules excluded).

Example (program listing):

```
======== Kubes ==================================== KUAX 680C =======
                         Organization module IL
Project : E205D
Module  : ORG       No.: 1      created : Nov 26 1991 16:08
User    : KUBES                 changed : Nov 26 1991 16:08
====================================================================

  1:          JPP     ONOFF               1
  2:
  3:          JPP     COUNTER              2
  4:
  5:          L        MOTOR          O00.00 ; (motor conveyor belt)
  6:          JPCP    NUMACT               3
  7:
  8:          LN       MOTOR          O00.00 ; (motor conveyor belt)
  9:          JPCP    NUMSUM               4
```

3.2. Program module

Features:

Name:	xxxxxxxx.PRO
Number:	255
Length:	max. 253 lines incl. 128 code lines
Function:	user program for a separate part of the overall problem; organization of the next module level
Call:	from the organization module or other program modules

Use KUBES to create program modules. They are managed in the project under a (max. 8-digit) name and a number.
All commands are applicable without limitations (commands of the initialization modules excluded).

Example (program listing):

```
======== Kubes ==================================== KUAX 680C =======
                        Program module  IL
Project : E205D
Module  : COUNTER   No.: 2       created : Nov 26 1991 16:15
User    : Kevin Kubes            changed : Nov 26 1991 16:15
Comment : COUNTER
=====================================================================

          L       COUNTER        C00.00 ; (piece counter)
          O       STOP           I00.01 ; (switch off motor)
          =       PULSE         PP00.00
          L       PULSE         PP00.00
          JPCP    SUM                 5
          L       ONMARKER       M00.00 ; (marker motor ON/OFF)
          =       COUNTER:12:V   C00.00 ; (piece counter)
          L       CIMP           I00.02 ; (counting impulse of initiator)
          =C      COUNTER        C00.00 ; (piece counter)
          L       CLEAR          I00.03 ; (key "clear count”)
          JPCP    NEW                 6
```

3.3. Function module

Features:

Name: xxxxxxxx.FUN
Number: 255
Length: max. 253 lines incl. 128 code lines
Parameters: max. 16
Function: general-purpose module. It is created by the user and can be equipped with parameters. The program contained in this type of module is written like that for the program modules.
Call: from the organization or program module

Up to 16 input and output parameters make it possible to execute the function with different variables (operands, constants). These parameters are entered into a table and are used in the program part like normal operands under their own names.
Multiple use with different parameters in one program is possible.
All commands are applicable without limitations (module calls and commands of the initialization modules excluded).

☞ *Programmable timers (PTxx.xx) are not permissible as input parameters. Not the logical timer output would be read but the value of the status byte.*
<u>Remedy:</u> Assign the timer output to a marker and then use the marker as input parameter.

Example (calling up a function module):

```
; convert ASCII characters into binary value

        JPF     ASC3BIN2 ,    ____
                  ASC_23_B -|    |- BIN_16,
                  ASC_BS_3 -|    |- ,
                  VALUE_22 -|____|-
```

3.4. Timer module

Features:

Name: xxxxxxxx.TIM
Number: 4
Length: max. 253 lines incl. 128 code lines
Function: Processing of sections of the program in intervals of quartz precision controlled by time interrupts. The following 4 time bases are available: 10 ms, 100 ms, 1 s, 10 s. The program contained in this type of module is written like that for the program modules.
Call: automatically by the assigned time interrupt

Amongst other things, the time interrupts serve the processing of programmable timers. The timer modules created by the user are called up and processed by these time interrupts.
All commands are applicable without limitations (module calls and commands of the initialization modules excluded).

⚠ *When creating an interrupt module please be aware of the fact that the module number decides on its function. See the table below.*

The timer modules are called up by the following time interrupts:

Timer Module No.	Called up by Time Interrupt
1	10 ms
2	100 ms
3	1 s
4	10 s

⚠ *The timer modules should be as short as possible to reduce the time load on the CPU to a minimum. You should therefore only include those operations in a timer module that you consider really necessary. Everything else can be taken care of in the program modules.*

3.5. Interrupt module

Interrupt modules are called up by interrupts which are signalled to the CPU via the control bus. Interrupts can be triggered by interrupt inputs, interrupt-controlling modules or by failure or error messages.

Features:

Name:	xxxxxxxx.INT
Number:	18
Legth:	max. 253 lines including max. 128 code lines
Function:	they serve quick reactions to events such as "Count complete", "Undervoltage" etc. The program contained in this type of module is written like that for the program modules.
Call:	automatically by the assigned interrupt

All commands are applicable without limitations (module calls and commands of the initialization modules excluded).

⚠ *When creating an interrupt module please be aware of the fact that the module number decides on its function. See the tables below.*

3.5.1. Call by error or failure messages

For certain system errors it is practical to provide certain means in the user program that allow keeping the effects of such errors as small as possible. To be able to react fast enough, the monitor program triggers an interrupt that calls up an interrupt module. You can program this module to contain the required reaction:

Trigger	**Controller**	**Interrupt Module**
undervoltage power supply	657P, 680I, 680C, 680CT	17
short-circuit on output		18

☞ *Refer to appendix "Error and failure messages" of the instruction manuals of the individual controllers to find explanations of the causes of these failures. There you will also find suggested actions to remedy the situation.*

3.5.2. Call by interrupt-controlling modules (KUAX 680..)

The following section applies to controllers
- KUAX 680I,
- KUAX 680C and
- KDT 680CT.

Function modules, e.g. counter modules, communicate with the user program via transfer addresses SLx... (in the KUAX 657 and 657P, these are the slave dual-port RAM addresses, hence SLx).
Each module slot is assigned 32 addresses (16 from every group of addresses, e.g. SLA and SLB for slot 0) which serve various functions depending on the type of module

Each address group can call up an interrupt module which means that one module can trigger up to 2 interrupt.

Trigger	Controller	Transfer Address Range	Interrupt Module
Module 0	680I, 680C, 680CT	SLA00.00...01.15 SLB00.00...01.15	1 2
Module 1		SLC00.00...01.15 SLD00.00...01.15	3 4
Module 2		SLE00.00...01.15 SLF00.00...01.15	5 6
Module 3		SLG00.00...01.15 SLH00.00...01.15	7 8
Internal interrupt inputs	680C, 680CT	SLJ00.00...01.15	10

3.5.3. Call by slave module (KUAX 657P)

Slave modules communicate with the user progra via dual-port RAM addresses SLA...SLP. The address range is set by the DIP switch on the module.
Every address range is assigned an interrupt module (1...16) which can be called up by interrupt-controlling slave modules:

Trigger		Controllers	Interrupt Module
Slave dual-port RAM:	SLA	657P	1
	SLB		2
	SLC		3
	SLD		4
	SLE		5
	SLF		6
	SLG		7
	SLH		8
	SLI		9
	SLJ		10
	SLK		11
	SLL		12
	SLM		13
	SLN		14
	SLO		15
	SLP		16

There is a wide range of different slave modules available for the KUAX 657P. They have different functions such as positioning, communication, regulation etc.

☞ *Please refer to the relevant instruction manuals to learn about programming of these modules and handling of the interrupt function in the user program.*

3.6. Initialization module

Features:

Name:	xxxxxxxx.INI
Number:	5
Length:	max. 253 lines incl. max. 128 code lines
Function:	Serves easy assignment of a certain value to operands without having to use logical operations, e.g. for presetting process parameters, tables, text fields, etc.
Call:	from the organization and the program module

Only a limited set of instructions is applicable (see chapter "5.15. Commands of the initialization modules").

To avoid extending the cycle time, initialization modules should only be called up when needed but not cyclically.

Example (program listing):

```
========  KUBES  =======================================================
                           Init. module  IL
Project : K631TEXT               Network  :
Module  : INIT      No.: 1        created : Jul 26 1994 13:43
User    : Kevin Kuax              changed : Jan 31 1996 13:56
Comment : INIT
========================================================================
  1: INIT       M00.00      BIT     1        ; (initialization)
  2: BC08.00                BYTE    "Preset value:{10/3/1}$“
  3: SBC00.00               BYTE    "Time:“..:..:...“$“
  4: SBC02.00               BYTE    „““$“
  5: SBC04.00               BYTE    „““,2,4,6,8,0,2,4,6,8,“$“
  6: HITKEY     BD15.09     BYTE    0
  7: LINES      BD15.12     BYTE    2        ; (no. of lines)
  8: CHARS      BD15.13     Byte    20       ; (no. of characters)
  9: KEY        BD15.14     WORD    1234     ; (key for test operation)
 10: IMGGO      BD15.10     TEXT    "Kuhnke GmbH Malente"
```

3.7. Trigger module

Features:

Name: xxxxxxxx.TRG
Number: 16
Length: max. 253 lines incl. max. 128 code lines
Function: Used in "Test mode with breakpoints" for triggering breakpoints.
Call: Under KUBES (see there) in test mode

All commands are applicable without limitations (module calls and commands of the initialization modules excluded).

Is used in test mode under KUBES only. The result (contents of the processor accu) at the end of the trigger module defines the trigger condition. With byte or word operations, bit 7 of the lowbyte in the accu is analysed.

Example (program listing):

```
; Trigger is I00.00 is on
; and PT00.00 is over

         L       I00.00
         A       PT00.00
```

3.8. KUBES module

Features:

Name:	<set>.KNK
Number:	255
Parameters:	max. 16
Function:	Module for special solutions.
Call:	from the organization and the program module

Written by Kuhnke in high-level programming language or Assembler and delivered in one or several libraries on diskette. By using the input and output parameters you can execute the function with different variables (operands, constants). Multiple use with different parmeters in one program is allowed.

☞ *Programmable timers (PTxx.xx) are not permissible as input parameters. Not the logical timer output would be read but the value of the status byte.*
Remedy: Assign the timer output to a marker and then use the marker as input parameter.

Example (calling up a KUBES module):

```
; Multiplication followed by division operation
         JPK     MULDIV32 ,    _____
                   MU_CAND  -|     |- RESULT,
                   MU_PLIER -|     |- DI_SOR_0,
                   DIVISOR  -|     |- QUOT_0,
                            -|_____|- ADDRESS
```

☞ *There is a separate instruction manual for the KUBES modules:* E 386 GB, KUBES Modules.

3.9. Data module

Many PLC applications require management of large amounts of data such as recipes, sets of parameters or positioning data. The available markers are often not enough.
It is the use of operating terminals in particular which has made data modules an increasing necessity. They give you the opportunity of editing or creating data in the actual system.
Implementation of data modules also allows connecting external development tools to the PLC via a data interface.

Features:

Name: xxxxxxxx.DAT

Number: max. 255, depending on the capacity of the user memory

Length: 256 byte

Function: Serves storing large amounts of data (tables, texts etc.) in the user program memory, i.e. either in the EPROM (read only) or in the RAM (read/write). Data modules are accessed from within the user program via data processing ranges DB0...DB7.

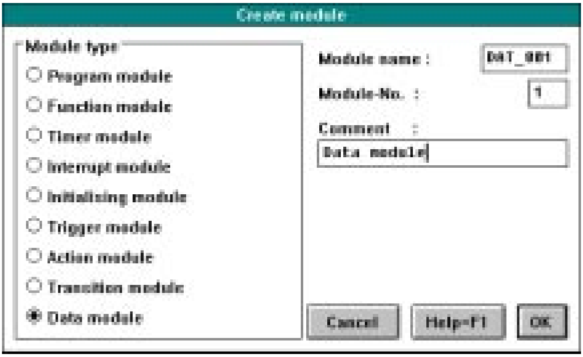
Call: from the organization, program, function, time, and interrupt modules using the LoadDB and StoreDB commands.

Prerequisites:

- KUBES4, version 4.10 or higher
- PLC monitor, version 4.17 or higher

3.9.1. Creating data modules

Use the Module Editor's "Create" command from the "Module" menu to create data modules. Choosing this command displays the following dialog box:



Create module

Module type
○ Program module
○ Function module
○ Timer module
○ Interrupt module
○ Initialising module
○ Trigger module
○ Action module
○ Transition module
● Data module

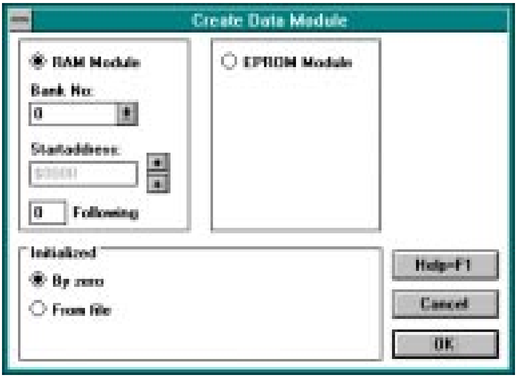
Module name : DAT_001
Module-No. : 1
Comment : Data module

Cancel | Help=F1 | OK

Select the correct type and name for your module.

If the module name ends in the numbers "001", the next KUBES dialog box will allow you to create several data modules simultaneously (in this case: DAT_002, DAT_003...).

Clicking on "OK" opens the next dialog box:



Create Data Module

● RAM Module
Bank No: 0
Startaddress: [illegible]
0 Following

○ EPROM Module

Initialized
● By zero
○ From file

Help=F1 | Cancel | OK

You are requested to make several entries:

- Decide whether you want to store the data module(s) in the program range (EPROM) or in the data range (RAM) of the user memory. Data modules stored in the program range (EPROM) cannot be changed while the controller is running. Data modules stored in the data range can be changed, however.
- Enter bank number and starting address within the bank for data modules stored in the RAM.

☞ *You have to reserve a data memory range for modules that you want to store in the RAM either before creating them or afterwards during online operation (KUBES Main Menu, PLC menu, command Set Memory Size).*

- In the box next to Following enter the number of additional data modules that you want to create. You have the option of making this entry only if the name you assigned to the data module to be created in the previous dialog box ended in 001.
- You can also specify a file from which you want to import existing data into the module you are creating (see ch. "3.9.5 Importing data into a data module"). If you do not specify a file, the data module will be filled with zeros. After you have created the data module it will be automatically taken over into the project.

3.9.2. Transmitting data modules to the PLC

Before transmitting the project you will be shown a list box from which you can select the data modules that you want to transmit to the PLC.
Optionally you can use a new Module Editor command to transmit individual data modules to the PLC: PLC menu, command Transmit Data Module.

3.9.3. Getting data modules from the PLC

Use the command Load Data Module from the Module Editor's PLC menu to get data modules from the PLC to write them to a file. This is also possible while adjusting the project. RAM modules are always reported to be modified without this preventing a successful adjustment however. The purpose is to be able to read back the RAM data modules via the adjustment dialog at any time.

3.9.4. Exporting data modules

Data stored in a data module on the PC can be written to a file using the Module Editor's command Export Data Module from the Module menu.

3.9.5. Importing data into a data module

Data stored in a file on the PC can be loaded into a data module using the Module Editor's command Import Data Module from the Module menu. Data to be imported can be simple text files or tables.

3.9.6. Editing data modules online

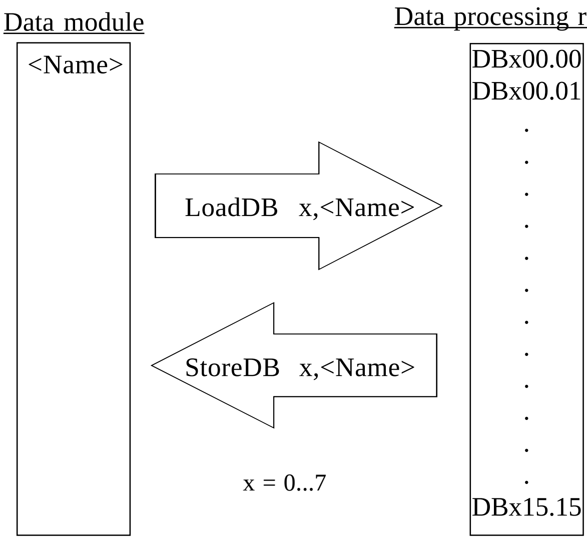
Use the Display Address Range to indirectly editing data modules. To do so load the contents of the data module into a data processing range (DBx, see ch. 3.9.7) (command: LoadDB...). This data processing range can be edited in the Display Address Range. The data can then be written back to the module upon the corresponding user program command (command: StoreDB...) as long as the destination range of the data module is in the RAM.

3.9.7. Data processing ranges

Data processing ranges provide a means of indirectly accessing the data modules. There are 8 data processing ranges (DB0...DB7) which you can treat just like byte marker ranges. Each range consists of 256 byte which exactly corresponds to the size of a data module:

DB000.00 - DB015.15
DB100.00 - DB115.15
DB200.00 - DB215.15
DB300.00 - DB315.15
DB400.00 - DB415.15
DB500.00 - DB515.15
DB600.00 - DB615.15
DB700.00 - DB715.15

Commands LoadDB and StoreDB are used for copying data between data module and data processing range



Storing the data module again (StoreDB) is only possible if the data module is in the RAM.

3.9.8. Programming

The contents of data modules must be loaded to a data processing range before they can be edited in the user program.
In the user program they can be read from and edited in every programmable module type.

3.9.8.1. Load commands

The purpose of the load commands is to load the contents of a data module into a data processing range. Once loaded they can be processed from within every other module type (organization, program, function modules etc.) using the normal PLC commands.

`LoadDB x,<Name>`
Loads data module <Name> into data processing range x (= 0...7).

`LoadDB A1,<Name>`
Loads data module <Name> into the data processing range whose number (= 0...7) is stored in byte marker A1.

`LoadDB A1,A2`
Loads the data module whose number (= 0...255) is stored in byte marker A2 into the data processing range whose number (= 0...7) is stored in byte marker A1.

`LoadDB x,A2`
Loads the data module whose number (= 0...255) is stored in byte marker A2 into data processing range x (= 0...7).

3.9.8.2. Store commands

The purpose of the store commands is to write the contents of a data processing range to a data module.
However, this is only possible if the data module is stored in the RAM.

`StoreDB   x,<Name>`
Copies data processing range x (= 0...7) to data module <Name>.

`StoreDB   A1,<Name>`
Copies the data processing range whose number (= 0...7) is stored in byte marker A1 to data module <Name>.

`StoreDB   A1,A2`
Copies the data processing range whose number (= 0...7) is stored in byte marker A1 to the data module whose number (= 0...255) is stored in byte marker A2.

`StoreDB   x,A2`
Copies data processing range x (= 0...7) to the data module whose number (= 0...255) is stored in byte marker A2.

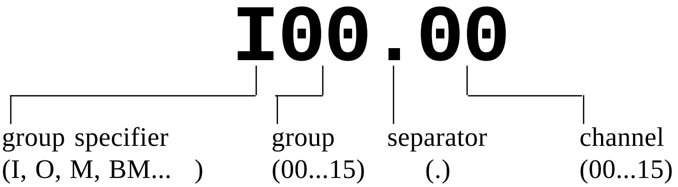
4. Operands

All addresses which can be addressed in the user program for signal processing or data storing are called operands. They are "operated" with.

On the following pages you will only find an overview of the different types of operands. For details please refer to the "Software" chapters of the instruction manuals of the various controllers.

4.1. Addressing

The operands are arranged in groups. Each group consists of a maximum of 255 operands. The way these operands are addressed is represented by the address of the first input operand:



Legend:

- Group specifier: indicates the function of many of the operands (I=input, O=output, M=marker, BM=byte marker)
- Group: from 00 through to 255 max. (256 groups). Individual groups can be smaller, e.g. timers, counters...
- Separator: separates group number and channel number. Input is compulsory.
- Channel: from 00 through to 255 max. (256 channels)

Input:

Leading zeros can be left out when inputting addresses in KUBES. Examples:

Input	*Representation*
I.	I00.00
O1.	O01.00
I1.1	I01.01

Offset addressing:

It is possible to indicate an offset for the absolute addresses of the local operands. The address is then made up by adding absolute address and offset.

L   BM00.00[BM00.01] means that the value in BM00.01 (offset) is added to the address of BM00.00. The resulting new address then responds to the load command.

⚠ *The value of the offset should be chosen in a way that excludes exceeding the corresponding operand range (max. 256 addresses).*

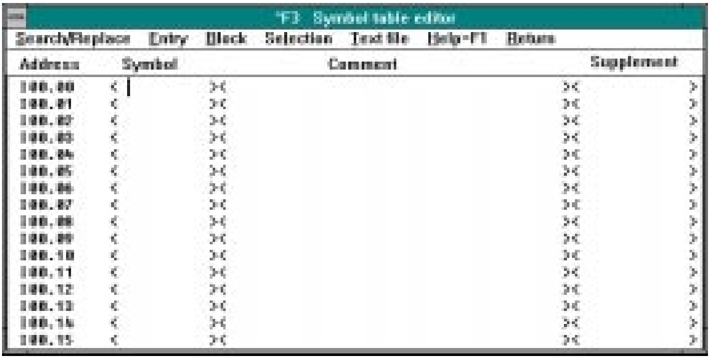
Reason:

Exceeding the operand range leads to reading (with read commands L,A,O...) from or writing (with assignment commands =, =N) into an operand from another range (see table on the right). This can lead to unintended machine functions or to program destruction.

☞ *In the instruction manuals of the controllers you will find a table called "address assignment of operands". Read this table to learn how the operand ranges are organized in the memory.*

4.2. Symbolic specification of operands

KUBES has a Symbol Table which offers you the possibility to specify names, or symbols, for the operands used. You can add an explanatory comment and a supplementary text.



```
Address     Symbol                 Comment                        Supplement
100.00      < |         ><                                  ><              >
100.01      <           ><                                  ><              >
100.02      <           ><                                  ><              >
100.03      <           ><                                  ><              >
100.04      <           ><                                  ><              >
100.05      <           ><                                  ><              >
100.06      <           ><                                  ><              >
100.07      <           ><                                  ><              >
100.08      <           ><                                  ><              >
100.09      <           ><                                  ><              >
100.10      <           ><                                  ><              >
100.11      <           ><                                  ><              >
100.12      <           ><                                  ><              >
100.13      <           ><                                  ><              >
100.14      <           ><                                  ><              >
100.15      <           ><                                  ><              >
```

Legend:

- Address
  Specification of the operand as preset by the system
- Symbol
  Identification of the operand as defined by the user. It should indicate the intended use of the operand. In the program , the symbol is listed together with the address. As data input you can use either the address or the symbol.
  Length: 8 characters (letters, numbers or <_>)
- Comment
  Explanatory text about the operand function. In the program the comment is listed together with the address.
  Length: 35 characters
- Supplement
  Supplementary text that only appears on the printout of the symbol table but not in the program listing. It be used, for example, to indicate the destination wiring point (terminal) of inputs or outputs. The printout of the symbol table can thus be used as a wiring diagram.

Advantages of the use of symbols:

- improved readability of the program
  the symbols refer to the actual use of the operand (the address in itself is neutral with regard to the application)
  Example:

Address	*Symbol*	*Comment*
I00.00	START_FF	start forward feed

- Review of the operands used already
  This is probably the most important aspect. The more operands you need the more important a permanently updated list becomes. Once you have specified all operands in the symbol table you have an almost perfect guarantee that no double assignment can occur. The Symbol Table ensures that.
- Reuse of program sections
  store parts of the program as a module in a folder under KUBES and you will be able to easily use it again for other projects. If you entered the operands in the symbol table it will be easy to reassign them to other addresses.
- Use as wiring diagram
  enter the connection point as a supplement to the operand so that you know to which point the input or output is to be wired and your symbol table will provide you with a perfect wiring diagram:

Address	*Symbol*	*Comment*	*Supplement*
I00.00	START_FF	start forward feed	X100/1
I00.01	POS_FF	forward feed into position	X101/1

etc.

⚠ *We recommend always using these advantages. We know from experience that the little plus in time you need for programming is more than compensated by much simplified or unnecessary trouble shooting.*

☞ *You will surely notice that for many of the examples of this manual we have done without the Symbol Table. The reason for this is that we wanted to make the direct address relation clear.*

4.3. Features of the operand groups

4.3.1. Digital inputs and outputs

- Inputs and outputs represent the process as a process image which is updated between two subsequent program cycles.
- Inputs "read" the signals of switches, key-switches, initiators etc. and report the signal status to the CPU via the control bus.
- Outputs output control signals to relays, contactors, magnets etc. in order to switch them on or off. Determined by the user program, the CPU transmits the signals to the output modules via the control bus. At the same time, the signals are also transmitted to RAM memory cells, which are addressed under the same address on the CPU. The processor accesses this memory cell to read the status of an output (commands: L, A, O...).

Configuration
You define your own configuration by plugging (or not plugging) the corresponding modules into the controller (if this is of a modular design).
The following systems are exceptions to this rule:

- KUAX 644 PC Control
  has no local inputs and outputs of its own. All peripheral signals are read via PROFIBUS from decentralized devices (KUAX 680S, KUAX 680I or other PROFIBUS stations).
- KUAX 680C Compact Control
  and Control Terminal KDT 680CT
  these are permanently configured with inputs and outputs. If there is a bus board you can also plug in modules to provide further inputs and outputs.

Addressing

in KUAX 657P groups with 16 channels each

Inputs: I00.00...15.15
SI00.00...15.15

Outputs: O00.00...15.15
SO00.00...15.15

in KUAX 680I with 8 channels each

Inputs: I00.00...xx.07

Outputs: O00.00...xx.07

in KUAX 680C and KDT 680CT groups with 8 and 2 channels each

Inputs: I00.00...xx.07

Counter inputs.: SI00.00...00.01

Interrupt inputs: SI01.00...01.01

Outputs: O00.00...xx.07

Access

Inputs and outputs are updated as a process image between program cycles. Commands of the user program access cells of this process image.

Exception: the KUAX 657P has no process image. Inputs are accessed directly during the reading process. Outputs are also assigned directly.

4.3.2. Analog inputs and outputs

- Inputs "read" the analog values of temperatures, liquid levels, speeds etc. Analog-to-digital conversion is done by the processor. The digital value can be processed in the program.
- Outputs output analog control signals for drives etc. in order to control these. Depending on the user program, the signals are transmitted to the control bus by the CPU. The digital-to-analog converter is on the module itself. The analog signal is tapped off the corresponding terminals.

Configuration

You define your own configuration by plugging (or not plugging) the corresponding modules into the controller (if this is of a modular design).

The following systems are exceptions to this rule:

- KUAX 644 PC Control
  has no local inputs and outputs of its own. All peripheral signals are read via PROFIBUS from decentralized devices.
- KUAX 680C Compact Control
  and Control Terminal KDT 680CT
  these are permanently configured with inputs and outputs. If there is a bus board you can also plug in modules to provide further inputs and outputs.

Addressing

<u>in KUAX 657P</u>

Analog inputs: via dual-port RAM of slave modules only
Analog outpus: AO00.00...03.15

<u>in KUAX 680I, 680C and KDT 680CT groups of 4 channels each</u>

Analog inputs: AI00.00...03, AI01.00...03, AI02.00...02
Analog outputs: AO00.00...xx.03

4.3.3. Bit markers and byte markers

There is a large number of byte markers available on the CPU for marking (storing) current data.
Some of these byte markers are remanent if the CPU is accu-buffered.

Addressing
Bit markers: M00.00...15.15
R00.00...15.15
etc.
Byte markers: BM00.00...15.15
BR00.00...15.15
DB000.00...DB715.15

☞ *For word operations (16 bit) you combine 2 byte markers.*

4.3.4. Programmable timers

By default, the controllers have 128 software timers available. The time range is from 10ms - 65535s. These timers can be programmed with raising or falling delay or as clock pulse or pulse generators respectively. If desired they can be remanent.

Addressing
Programmable timers: PT00.00...07.15

4.3.5. Software clock pulse

The system makes 4 software clock pulses available via byte operands. These operands are automatically incremented in the specified clock frequency (0...255). The user program can analyse this for time-controlled operations.

Addressing

10 ms:	T00.00
100 ms:	T00.01
1 s:	T00.02
10 s:	T00.03

4.3.6. Counters

32 counters with a counting depth of 16 bit (0-65535) can be programmed as up or down counters. They too can be remanent if desired.

Addressing

Counters:	C00.00...01.15

4.3.7. System error marker "ERR00.00"

Recognized system errors are written into byte operand (8 bit) "ERR00.00" by the monitor program. They can be read by the user program and then analysed correspondingly. Please refer to the actions recommended in appendix "D. Reactions to failures" of the instruction manuals of the controllers.

5. Description of the commands

The overview below describes all commands in plain text.

Application to operands for bit, byte and word operations:
A large portion of the commands can be used for bit and byte and word operations. Although this is a great advantage it can also be a source of danger if used unwisely:

⚠ *Please avoid logical operations with operands of different sizes if at all possible. Because if you do, the result of the operations cannot always be predicted as bit operands only influence bit 7 of the accumulator while all bits are analysed or influenced in byte and word operations.*

<u>Example:</u>

```
L  I00.00
```

writes the status of the input into bit 7 of the accumulator while leaving all other bits uninfluenced. These could have various values, depending on the historical situation.
If you then assign the value to a byte operand

```
=  BM00.00
```

only bit 7 of this operand will be set and defined, all other bits will be in or take on an undefined status.

Differences between the controllers:
As there may be differences in the use of the commands in the various control systems, the information given in this chapter is kept very general.

☞ *Please refer to appendix "A. References" of this manual to find a list of instruction manuals. For technical information about the commands please refer to chapter "Commands overview" of the instruction manual of the relevant controller.*

There you will find the following information:

- which types of operands and constants can be linked to the commands,
- how much address space do the commands occupy in the user memory,
- how long is the processing time for a command,
- whether and how the carry and zero bits are influenced.

5.1. Logical operations commands

Command	Function
L	Load Loads the value of the bit or byte operand into the accu
LN	Load with negation Loads the negated value of the bit or byte operand into the accu
LD	Load Loads the value of the word operand (16 bit) into the accu
U	logical AND operation Logical AND operation bit-by-bit between the value of the bit or byte operand and the contents of the accu: "1 A 1 = 1", "1 A Ø = Ø", "Ø A Ø = Ø", "Ø A 1 = Ø". The result is stored in the accu.
UN	logical NAND operation (AND not) Logical NAND operation bit-by-bit between the negated value of the bit or byte operand and the contents of the accu: "1 AN 1 = Ø", "1 AN Ø = 1", "Ø AN Ø = Ø", "Ø AN 1 = Ø". The result is stored in the accu.
UD	logical AND operation Logical AND operation bit-by-bit between the value of the word operand (16 bit) and the contents of the accu: "1 A 1 = 1", "1 A Ø = Ø", "Ø A Ø = Ø", "Ø A 1 = Ø". The result is stored in the accu.
O	logical OR operation Logical OR operation bit-by-bit between the value of the bit or byte operand and the contents of the accu: "1 O 1 = 1", "1 O Ø = 1", "Ø O Ø = Ø", "Ø O 1 = 1". The result is stored in the accu.
ON	logical NOR operation (OR not) Logical NOR operation bit-by-bit between the negated value of the bit or byte operand and the contents of the accu: "1 ON 1 = 1", "1 ON Ø = 1", "Ø ON Ø = 1", "Ø ON 1 = Ø". The result is stored in the accu.
OD	logical OR operation Logical OR operation bit-by-bit between the value of the word operand (16 bit) and the contents of the accu: "1 O 1 = 1", "1 O Ø = 1", "Ø O Ø = Ø", "Ø O 1 = 1". The result is stored in the accu.
XO	logical EXCLUSIVE-OR operation (antivalence) Logical EXCLUSIVE-OR operation bit-by-bit between the value of the bit or byte operand and the contents of the accu: "1 XO 1 = Ø", "1 XO Ø = 1", "Ø XO Ø = Ø", "Ø XO 1 = 1". The result is stored in the accu.
XON	logical EXCLUSIVE-OR operation (equivalence) Logical EXCLUSIVE-OR operation bit-by-bit between the value of the bit or byte operand and the contents of the accu: "1 XON 1 = 1", "1 XONØ = Ø", "Ø XON Ø = 1", "Ø XON 1 = Ø. The result is stored in the accu.

Examples:

```
L    I00.00
```

Load status (bit) of the input

```
L    BM00.00
```

Load contents (byte/8 bit) of the byte marker

```
LD   BM00.00
```

Load contents (word/16 bit) of byte markers BM00.00 and 01

```
L    200
```

Load constant value (8 bit, 0...255)

```
LD   2000
```

Load constant value (16 bit, 0...65535)

```
LN   I00.00
```

Load negated status (bit) of the input

```
LN   BM00.00
```

Load negated contents (byte/8 bit) of the byte marker

To read unoccupied input addresses

The note of warning below concerns:

- in KUAX 680I, 680C and KDT 680CT:
  all input channels I(SI)xx.08...15, as only the 8 channels I(SI)xx.00.00...07 are occupied per group,
- in all modular systems:
  all input addresses for which no module has been plugged in,
- in KUAX 644:
  all local inputs as this controller has no inputs of its own:

⚠ *Avoid reading input addresses with no corresponding hardware configuration. In such cases, the value read depends on the current status of the bus data line and is undefined (i.e. not defined "0").*

5.2. Assignments and set commands

Cmnd	Function
=, =D	assignment Writes the contents of the accu into the memory addressed by the operand.
S	conditional set Sets the value of the operand to log 1 if there is log 1 in the accu after the preceding operation; it remains unchanged if there is log 0 in the accu.
R	conditional reset Sets the value of the operand to log 0 if there is log 1 in the accu after the preceding operation; it remains unchanged if there is log 0 in the accu.
=1	unconditional set Sets the value of the bit operand to logical 1, regardless of what is in the accu.
=0	unconditional reset Sets the value of the bit operand to logical 0, regardless of what is in the accu.

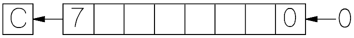
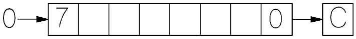
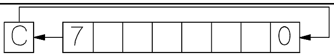
5.3. Arithmetic commands

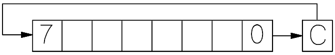
Cmnd	Function
ADD, ADDD	Addition Adds the value of the operand to the contents of the accu. The sum is stored in the accu after the operation.
SUB, SUBD	Subtraction Subtracts the value of the operand from the contents of the accu. The difference is stored in the accu after the operation.
MUL, MULD	Multiplication Multiplies the value of the operand by the contents of the accu. The product is stored in the accu after the operation.
DIV, DIVD	Division Divides the value of the operand by the contents of the accu. The quotient is stored in the accu after the operation.

5.4. Comparison commands

Cmnd	Function
CMP, CMPD	Comparison Compares the value of the operand to the contents of the accu. The operation sets internal flags. These lead to conditional jump instructions and are used for program branching.
CMP=, CMPD=	Compare if equal Like "Comparison" plus influence on the accu: if the comparison is "true" then the accu is set to 255 (logical 1), otherwise it is cleared.
CMP<>, CMPD<>	Compare if inequal Like "Comparison" plus influence on the accu: if the comparison is "true" then the accu is set to 255 (logical 1), otherwise it is cleared.
CMP<=, CMPD<=	Compare if smaller or equal Like "Comparison" plus influence on the accu: if the comparison is "true" then the accu is set to 255 (logical 1), otherwise it is cleared.
CMP>=, CMPD>=	Compare if greater or equal Like "Comparison" plus influence on the accu: if the comparison is "true" then the accu is set to 255 (logical 1), otherwise it is cleared.

5.5. Shift and rotation commands

Cmnd	Function
LSL, LSLD	logical shift left in the accu Shifts the contents of the accu by one binary position (has the same effect as a multiplication by 2). The result of the operation is stored in the accumulator.
LSLM, LSLDM	logical shift left in the operand Shifts the contents of the operand by one binary position (has the same effect as a multiplication by 2). The result of the operation is stored in the operand.
	
LSR, LSRD	logical shift right in the accu Shifts the contents of the accu by one binary position (has the same effect as dividing the contents by 2). The result of the operation is stored in the accumulator.
LSRM, LSRDM	logical shift right in the operand Shifts the contents of the operand by one binary position (has the same effect as dividing the contents by 2). The result of the operation is stored in the operand.
	
ROL, ROLD	Roll (end-around shift) left in the accu by Carry Shifts the contents of the accu by one binary position. The contents of the carry bit moves into the digit thus become vacant; the carry value is written into the carry bit.
ROLM, ROLDM	Roll (end-around shift) left in the operand by Carry Shifts the contents of the operand by one binary position. The contents of the carry bit moves into the digit thus become vacant; the carry value is written into the carry bit.
	
ROR, RORD	Roll (end-around shift) right in the accu by Carry Shifts the contents of the accu by one binary position. The contents of the carry bit moves into the digit thus become vacant; the carry value is written into the carry bit.
RORM, RORDM	Roll (end-around shift) right in the operand by Carry Shifts the contents of the operand by one binary position. The contents of the carry bit moves into the digit thus become vacant; the carry value is written into the carry bit.



5.6. Manipulation of bytes and flags

Cmnd	Function
INC, INCD	Increment Increments the value of the operand by one.
DEC, DECD	Decrement Decrements the value of the operand by one.
CLR	Clear The value of the operand becomes 0.
NOP	Do-nothing operation No operation, just forwarding to the next instruction.
SEC	Set CARRY Sets the CARRY bit to 1.
CLC	Clear CARRY Clears the CARRY bit.

5.7. Module calls

Cmnd	Function
JPP	unconditional call of a program module
JPCP	conditional call of a program module Calls the module up if the accu contains a bit operand set to logical 1 (bit 7 is read).
JPF	unconditional call of a function module
JPCF	conditional call of a function module Calls the module up if the accu contains a bit operand set to logical 1 (bit 7 is read).
JPK	unconditional call of a KUBES module
JPCK	conditional call of a KUBES module Calls the module up if the accu contains a bit operand set to logical 1 (bit 7 is read).
JPINIT	unconditional call of the initialization module

5.8. Jump commands

Jumps within a program module are carried out to a program line identified by a jump mark. Difference is made between the following types of jumps:

- unconditional jumps,
- conditional jumps that analyse the logical state of bit operands,
- conditional jumps that analyse the result of comparison operations.

Command	Function
JP	unconditional jump
JPC	conditional jump if yes (log. 1) Carries out the jump if the accu contains a bit operand set to logical 1 (bit 7 is read)
JPCN	conditional jump if no (log. 0) Carries out the jump if the accu contains a bit operand set to logical 0 (bit 7 is read)
Conditional jumps after comparison operations:	
	Carries out the jump if the contents of the accu, in relation to the compared value, is:
JP=	equal or 0
JP<>	inequal
JP<	smaller
JP>	greater
JP<=	smaller or equal
JP>=	greater or equal
Jumps depending on the state of Carry or Zero bit:	
JPCS, JPCC	Jumps if carry bit is set (1) or cleared (0)
JPZS, JPZC	Jumps if zero bit is set (1) or cleared (0)
Jumps depending on the sign bit in the accu:	
JP+, JP-	Jumps if value is positive or negative

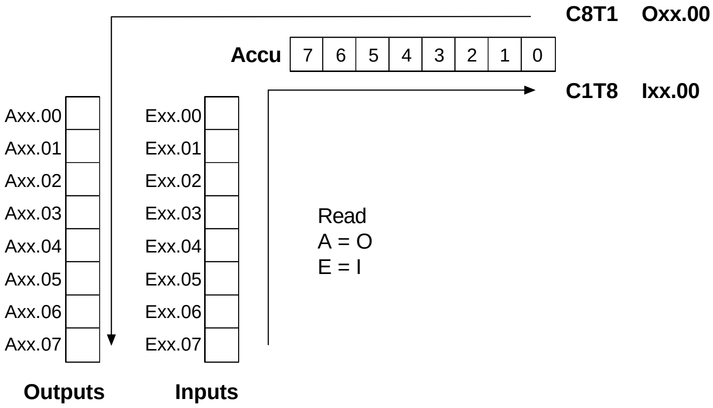
5.9. Copy commands

Command	Function
C1T8	copies the values of eight 1 bit operands into the 8bit accu
C1T16	copies the values of sixteen 1 bit operands into the 16bit accu
C8T1	copies the value in the 8bit accu into eight 1 bit operands
C16T1	copies the value in the 16bit accu into sixteen 1 bit operands

Example for 8 bit copy commands:

```
C1T8    Ixx.00     ;copies 8 inputs
                   ; into the accu
C8T1    Oxx.00     ;cop. contents of the accu
                   ; into 8 outputs
```

The figure below illustrates the function. The arrows indicate the direction:



5.10. Binary<->BCD conversion

Command	Function
BINBCD3	Binary-to-BCD conversion into a 3 decade BCD value Before the operation, the accu contains a 16bit binary value. After the operation, it contains the same value as a 3 decade BCD value.
BCDBIN3	BCD-to-binary conversion of a 3 decade BCD value Before the operation, the accu contains a 3 decade BCD value. Afterward, it contains the same value as a 16bit binary value.

Example:

```
LD        987       ;load binary value
=D        BM00.00   ; store in 2 byte markers
BINBCD3             ;binary-to-BCD conversion
=D        BM00.02   ; store in 2 byte markers
```

Use the Binary display mode of the Module Editor's Dynamic Display to view the effect of the individul commands. As the contents of the accu cannot be displayed directly, we use byte markers in our example:

The command **LD 987**
maps the binary value as follows:

```
                0000 0011 1101 1011
                 high byte    low byte
                 (BM00.01     BM00.00)
```

The command **BINBCD3**
converts the binary value into a BCD value:

```
                0000 1001 1000 0111
                        9      8     7   (decimal)
                 high byte    low byte
                 (BM00.01     BM00.00)
```

BCD-to-binary conversion follows the same principle, but the other way around.

⚠ *These commands can be used to convert values up to 999 max. There are KUBES modules available to convert 4-digit numbers.*

5.11. Programmable pulses (edge analysis)

Command		Function
L =	I00.00 PP00.00	The programmable pulse output is set when the status of the input changes from 0 to 1. The programmable pulse output is reset at the next run of this program part (next cycle).
L =N	I00.00 PP00.00	The programmable pulse output is set when the status of the input changes from 1 to 0. The programmable pulse output is reset at the next run of this program part (next cycle).
L =	PP00.00 O00.00	Loads the output signal of the programmable pulse into the accu and assigns it to an output.

⚠

After switching the controller on (or after RESET), the pulse has to be passed once at a value of 0 as the function cannot be guaranteed otherwise.
Recommendation: *Use a marker to assign the pulse value and map the input signal after it on the marker.*

Example:

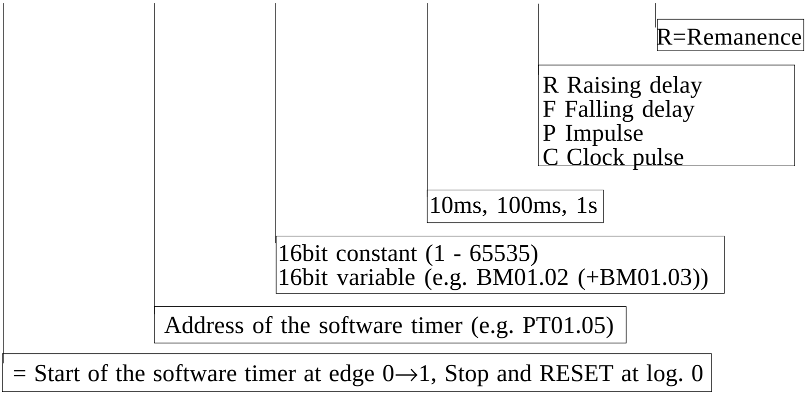
```
; To create a pulse signal
     L       M00.00    ;marker
     =       PP00.00   ; creates pulse
     L       I00.00    ;map input
     =       M00.00    ; on marker
; To analyse the pulse signal
     L       PP00.00   ;pulse signal
     JPCN    END       ; no -> jump
     INC     BM00.00   ;increment if signal

END  NOP
```

5.12. Programmable timers

You can program up to 128 software timers in the range of 10 ms - 65535s. These timers have the addresses PT00.00 -PT07.15.

To start a timer:

Assignent Address:Time value *Time basis :Function :Remanence *)



R=Remanence

R Raising delay
F Falling delay
P Impulse
C Clock pulse

10ms, 100ms, 1s

16bit constant (1 - 65535)
16bit variable (e.g. BM01.02 (+BM01.03))

Address of the software timer (e.g. PT01.05)

= Start of the software timer at edge 0→1, Stop and RESET at log. 0

Examples:

```
=      PT01.00:175*100ms:R:R
```

Start raising delay of 17.5 s with remanent actual value

```
=      PT01.00:BM04.06*100ms:F:R
```

Start falling delay with variable time value
(BM04.06/BM04.07 * 100 ms = preselection)

To scan an output:

```
L      PTxx.xx
```

Load logical time output into the accu

To scan the actual value (resid. value):

```
LD     PTxx.xx
```

Example:

```
LD     PT01.02
=D     BM06.02
```

Write residual value into BM06.02 and BM06.03

To halt the timer:

```
=TH    PTxx.xx
```

Example:

```
L      I01.00
=TH    PT01.03
```

The timer is halted while I01.00=1 (without RESET)

*) Entering “:R” for the zero-voltage guarantee of the actual timer value is optional.

5.13. Programmable counters

You can program up to 32 software counters in the range of 1-65535. These counters have the addresses C00.00 -C01.15.

To start a counter:

Assignment Address :Counter value :Function :Remanence [*)]

R=Remanence

F Forward count
B Backward count

16bit constant (1 - 65535)
16bit variable (e.g. BM01.02 (+BM01.03)

Address of the software counter (e.g. C01.05)

= Start of the software counter at edge 0→1, Stop and RESET at log. 0

Examples: = C00.00:175:F

Start forward counter with preset value 175

= C01.01:BM00.00:B:R

Load remanent backward counter with variable preset value (is in BM00.00 and BM00.01).

To scan an output: L Cxx.xx ;Count complete

Load logical counter output into the accu

To scan an actual value: LD Cxx.xx

Example:

```
LD  C01.01
=D  BM06.02
```

Write actual value into BM06.02 and BM06.03

To count/transfer the clock pulse: =C Cxx.xx

Example:

```
L   I00.01   ;Clock pulse
=C  C01.01
```

Input I00.01 represents the counting pulse. With each positive edge (0/1 transition), the count is increased by 1.

[*)] Entering “:R” for the zero-voltage guarantee of the actual counter value is optional.

5.14. Special commands

Command	Function
O_OFF	Outputs off. Deactivates the actuating elements of all outputs but does not change the internal status of the output "markers". The program keeps running. Use for example to react to short circuits (see instruction manual of the controller, appendix "Reactions to failures")
O_ON	Ouptuts on. Reactivates the actuating elements of all outputs. The program keeps running.
RESET	Reset and stop. Resets all non-remanent outputs, markers, timers and counters and stops program execution. Restart is only possible by switching the supply off and on again. Use for example to react to very extensive times of undervoltage (see instruction manual of the controller, appendix "Reactions to failures")
WAIT n	Wait for "n" * 10 milliseconds (interrupt module 17 only!). Defines an internal wait loop. The delay time is indicated in milliseconds (n = 1...6[* 10 ms], i.e. 60 ms max.). Program execution stops for this time. Program execution is resumed when the set time interval is over. Note: longer wait loops may lead to triggering the watchdog (see appendix "D.3. Watchdog"). Use: e.g. reaction to undervoltage with defined program exec. delay (see instruction manual of the controller, appendix "Reactions to failures")

5.15. Commands of the initialization modules

The initialization modules are a special variety of modules. None of the commands described previously in this chapter can be used here. On the other hand can the following commands only be used in the initialization modules.

Command	Function
BIT	Assigns logical values (signs 0/1) to one or several 1bit addresses (outputs or markers). Examples: O00.00 BIT 1 ;single bit O00.00 BIT 1,0,1,1.... ;bit string with different signals O00.00 BIT [16],1 ;bit string with [max. 255] equal signals
BYTE	Assings values to one or several (subsequent) byte addresses (8bit). Each value may be one of the range 0...255. Examples: BM00.00 BYTE 75 ;single byte as decimal value BM00.00 BYTE $4B ;single byte as hexadecimal value BM00.00 BYTE %01001011 ;single byte as binary value BM00.00 BYTE "K" ;single byte as ASCII character BM00.00 BYTE 1,18,0,125... ;byte string with different values BM00.00 BYTE [8],128 ;byte string with [max. 255] equal values
WORD	Assigns values to one or several word addresses (16bit = 2 byte). Each value may be one of the range 0...65535. Examples: BM00.00 WORD 19285 ;single word as decimal value BM00.00 WORD $4B55 ;single word as hexadecimal value BM00.00 WORD %0100101101010101 ;single word as binary value BM00.00 WORD +4.5V ;single word as voltage (-10...+10V) BM00.00 WORD 9.1mA ;single word as current (-20...+20mA) BM00.00 WORD 1,1800,10000,125... ;word string with different values BM00.00 WORD [8],10000 ;word string with [max. 128] equal values
TEXT	Assigns text to a number of byte addresses. Each individual text is filed in this form: <length><actual text><zero>. The length comprises itself and the last sign (zero). A text like this may be up to 253 characters long as the total length must not exceed 255. Examples: BM00.00 TEXT "KUHNKE" ;single text BM00.00 TEXT "KUHNKE", " MALENTE"... ;text string

5.16. Commands of the data modules

☞ *The commands of the data moduls are only listed again for reasons of completeness. Refer to chapter "3.9. Data module" to find a detailed description of these commands.*

Cmnd	Operand	Function
LoadDB	x,<name>	loads the contents of data module <name> into data processing range DBx00.00...15.15 (x = 0...7)
	byte1,<name>	loads the contents of data module <name> into data processing range DBx00.00...15.15 (x = value 0...7 in byte1)
	x,byte2	loads the contents of data module number y (y = value 1...255 in byte2) into data processing range DBx00.00...15.15 (x = 0...7)
	byte1,byte2	loads the contents of data module number y (y = value 1...255 in byte2) into data processing range DBx00.00...15.15 (x = value 0...7 in byte1)
StoreDB	x,<name>	stores the contents of data processing range DBx00.00...15.15 (x = 0...7) in data module <name>
	byte1,<name>	stores the contents of data processing range DBx00.00...15.15 (x = value 0...7 in byte1) in data module <name>
	x,byte2	stores the contents of data processing range DBx00.00...15.15 (x = 0...7) in data module number y (y = value 1...255 in byte2)
	byte1,byte2	stores the contents of data processing range DBx00.00...15.15 (x = value 0...7 in byte1) in data module number y (y = value 1...255 in byte2)

6. Programming examples

The examples given in this chapter are basically applicable to all controllers for which this manual was written. However, there are some structural differences which is why you should read the following notes:

- The KUAX 644
  has no local inputs or outputs. However, you can use most of the examples for internal markers and external operands (PROFIBUS) just the same.

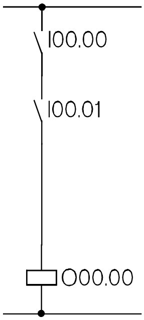
- The KUAX 657P
  has no process image for the local inputs and outputs. This means that the status of an input can change during one program cycle. You can thus switch an output on and off several times. This may lead to a jittering of the output.

- KUAX 680I, 680C and KDT 680CT
  have no input and output channels Ixx.08...15 and Oxx.08...15. You must therefore rewrite examples that make use of these channels. There is a process image for the local inputs and outputs. KUBES modules RD_IN (direct reading of inputs) and WR_OUT (direct writing into outputs) allow a direct access by by-passing the process image.

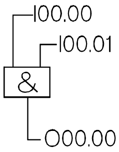
6.1. Basic functions

6.1.1. AND

Circuit diagram



Function diagram

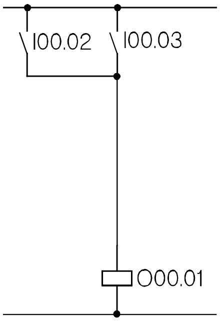


Instruction list

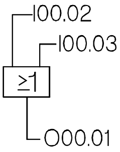
L	I00.00
A	I00.01
=	O00.00

6.1.2. OR

Circuit diagram



Function diagram

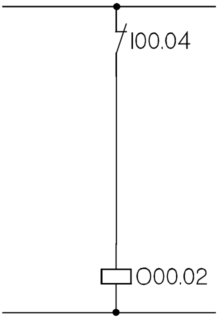


Instruction list

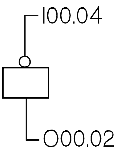
L	I00.02
O	I00.03
=	O00.01

6.1.3. Negation at input

Circuit diagram



Function diagram

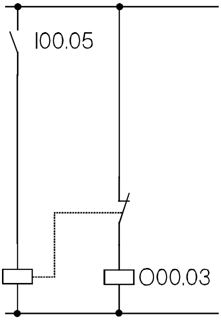


Instruction list

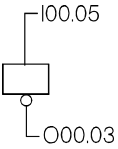
```
LN      I00.04
=       O00.02
```

6.1.4. Negation at output

Circuit diagram



Function diagram

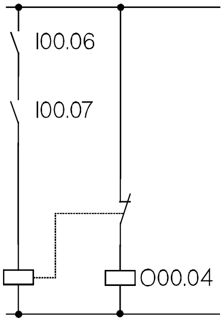


Instruction list

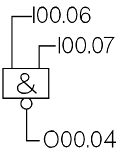
```
L       I00.05
=N      O00.03
```

6.1.5. NAND

Circuit diagram



Function diagram

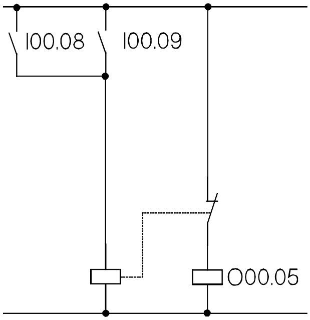


Instruction list

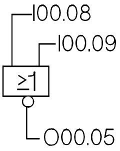
L	I00.06
A	I00.07
=N	O00.04

6.1.6. NOR

Circuit diagram



Function diagram

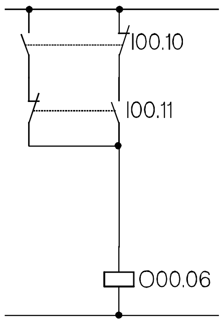


Instruction list

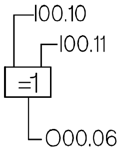
L	I00.08
O	I00.09
=N	O00.05

6.1.7. XO EXCLUSIVE-OR (non-equivalence)

Circuit diagram



Function diagram

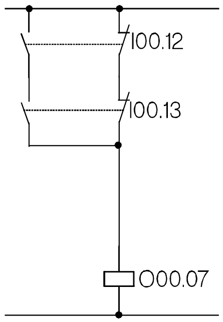


Instruction list

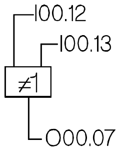
```
L      I00.10
XO     I00.11
=      O00.06
```

6.1.8. XON EXCLUSIVE-NOR (equivalence)

Circuit diagram



Function diagram

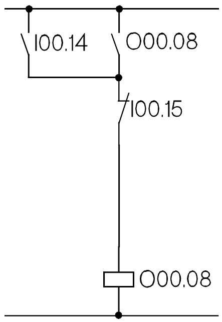


Instruction list

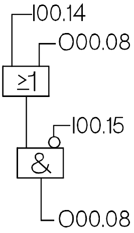
```
L      I00.12
XON    I00.13
=      O00.07
```

6.1.9. Self-locking circuit

Circuit diagram



Function diagram



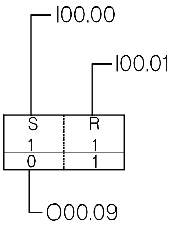
Instruction list

L	I00.14
O	O00.08
AN	I00.15
=	O00.08

6.2. Memory functions

6.2.1. With reset dominance

Circuit symbol

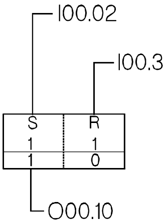


Instruction list

```
L      I00.00
S      O00.09 *)
L      I00.01
R      O00.09 *)

L      M00.00 *)
=      O00.09
```

6.2.2. With set dominance

Circuit symbol



Instruction list

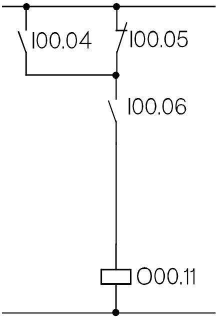
```
L  I00.02
R  O00.10 *)
L  I00.03
S O00.10 *)
```

**) Please note for KUAX 657P:*
This controller works without process image. Using the above example may therefore lead to a jittering of the output if the set and reset outputs are pressed at the same time.
Remedy: Store the result in a marker and assign to the output at the end of the sequence.

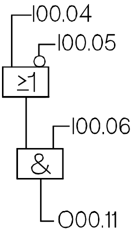
6.3. Combinational circuits

6.3.1. OR-AND circuit

Circuit diagram



Function diagram

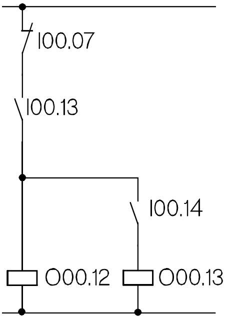


Instruction list

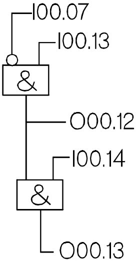
L	I00.04
ON	I00.05
A	I00.06
=	O00.11

6.3.2. Parallel circuit to output

Circuit diagram



Function diagram

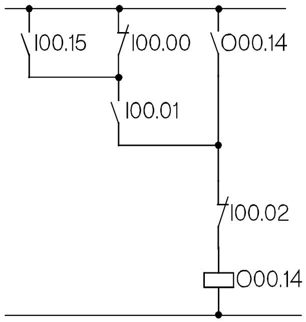


Instruction list

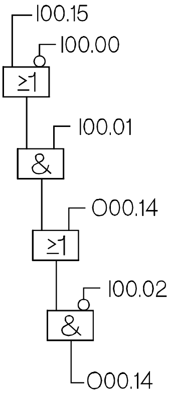
LN	I00.07
A	I00.13
=	O00.12
A	I00.14
=	O00.13

6.3.3. Network with one output

Circuit diagram



Function diagram

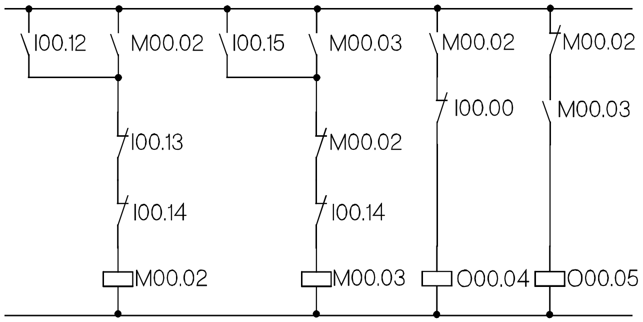


Instruction list

L	I00.15
ON	I00.00
A	I00.01
O	O00.14
AN	I00.02
=	O00.14

6.3.4. Network with outputs and markers

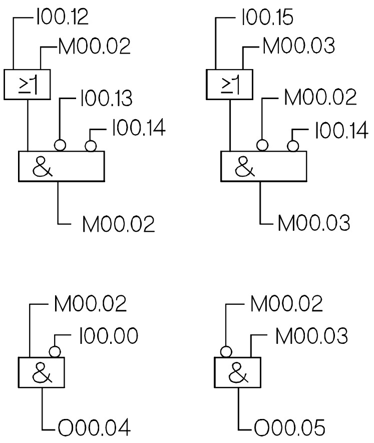
Circuit diagram



Instruction list

```
L       I00.12
O       M00.02
AN      I00.13
AN      I00.14
=       M00.02
L       I00.15
O       M00.03
AN      M00.02
AN      I00.14
=       M00.03
L       M00.02
AN      I00.00
=       O00.04
LN      M00.02
A       M00.03
=       O00.05
```

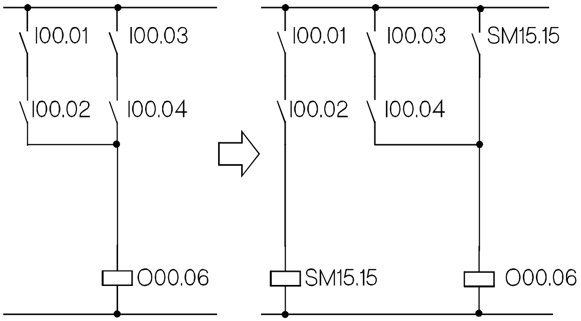
Function diagram



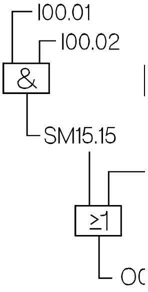
6.4. S-markers as AND/OR markers

6.4.1. Network with OR marker

Circuit diagram



Function diagram



Instruction list

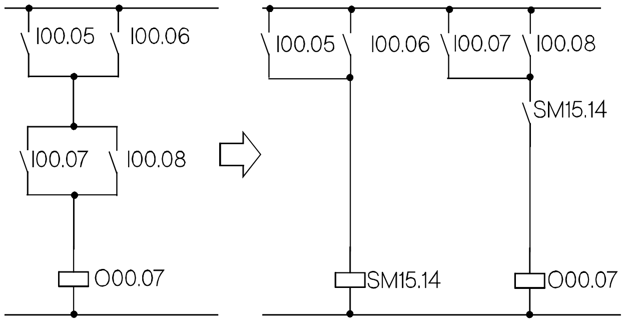
```
L       I00.01
A       I00.02
=       SM15.15
L       I00.03
A       I00.04
O       SM15.15
=       O00.06
```

In this example, a part result has to be stored temporarily.
Definition: S-marker SM15.15 is basically always used as OR marker. It can thus always be re-used in other networks.

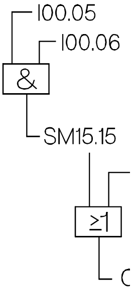
OR marker = SM15.15

6.4.2. Network with AND marker

Circuit diagram



Function diagram



Instruction list

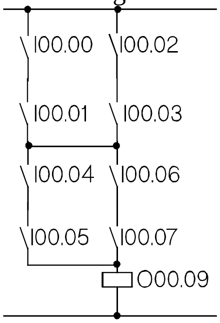
```
L       I00.05
O       I00.06
=       SM15.14
L       I00.07
O       I00.08
A       SM15.14
=       O00.07
```

In this example, a part result has to be stored temporarily.
Definition: S-marker SM15.14 is basically always used as AND marker. It can thus always be re-used in other networks.

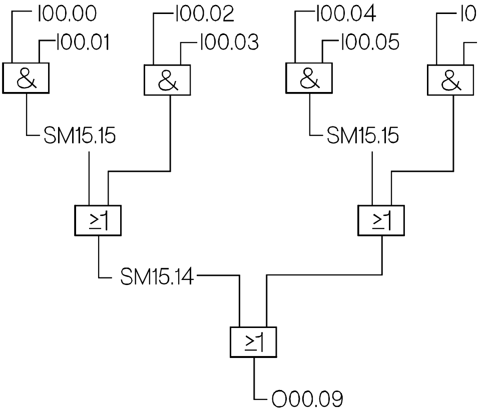
AND marker = SM15.14

6.4.3. Network with multiple use of the OR marker

Circuit diagram



Function diagram



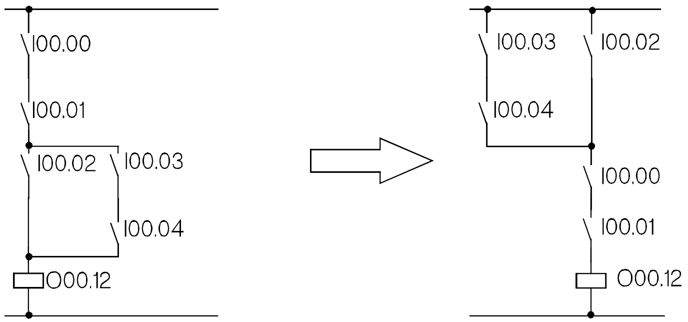
Instruction list

```
L       I00.00
A       I00.01
=       SM15.15  ;set OR marker
L       I00.02
A       I00.03
O       SM15.15
=       SM15.14  ;set AND marker
L       I00.04
A       I00.05
=       SM15.15  ;set OR marker
L       I00.06
A       I00.07
O       SM15.15
A       SM15.14
=       O00.09
```

6.5. Circuit conversion

Circuit diagram before

Circuit diagram after



Instruction list before

```
L       I00.00
A       I00.01
=       SM15.14
L       I00.02
=       SM15.15
L       I00.03
A       I00.04
O       SM15.15
A       SM15.14
=       O00.12
```

Instruction list after

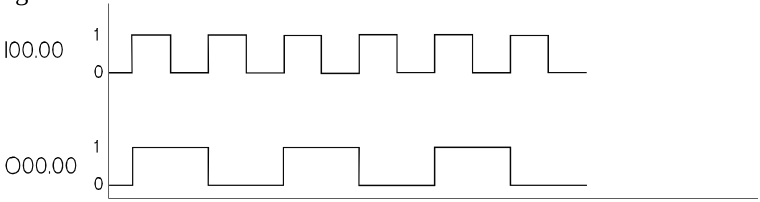
```
L       I00.03
A       I00.04
O       I00.02
A       I00.00
A       I00.01
=       O00.12
```

Circuit conversion leads to a different sequence of commands. Program generation is thus facilitated as the storing of part results becomes unnecessary.

6.6. Special circuits

6.6.1. Current surge relay

Signal course

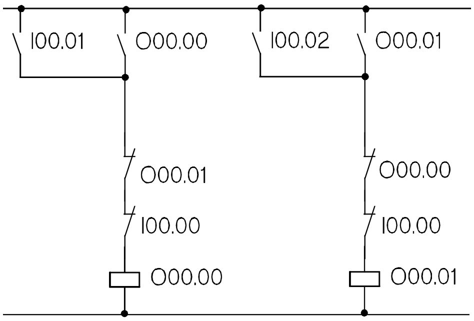


Instruction list

L	I00.00
=	PP00.00
L	PP00.00
XO	O00.00
=	O00.00

6.6.2. Reverse circuit (reverse contactor) with forced halt

Circuit diagram

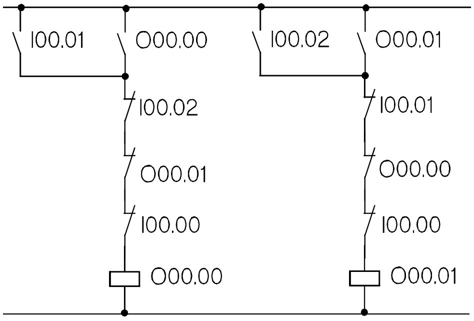


Instruction list *1)

```
L       I00.01    ;right key
O       O00.00    ;right contactor
AN      O00.01    ;left contactor
AN      I00.00    ;halt key *2)

=       O00.00    ;right contactor

L       I00.02    ;left key
O       O00.01    ;left contactor
AN      O00.00    ;right contactor
AN      I00.00    ;halt key *2)

=       O00.01    ;left contactor
```

6.6.3. Reverse circuit (reverse contactor) without forced halt

Circuit diagram



Instruction list *1)

```
L       I00.01    ;right key
O       O00.00    ;right contactor
AN      I00.02    ;left key
AN      O00.01    ;left contactor
AN      I00.00    ;halt key  *2)
=       O00.00    ;right contactor

L       I00.02    ;left key
O       O00.01    ;left contactor
AN      I00.01    ;right key
AN      O00.00    ;right contactor
AN      I00.00    ;halt key  *2)
=       O00.01    ;left contactor
```

*1) As the switching of the outputs is done very quickly we recommend providing a contactor interlock outside the PLC.

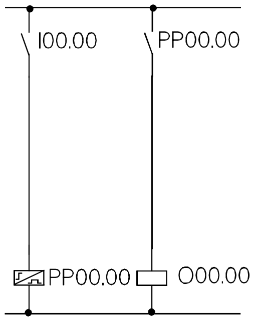
*2) If, for safety reasons, the halt key is already connected as n.c. switch outside the PLC, an A (AND) has to be programmed here.

6.7. Pulse edge analysis

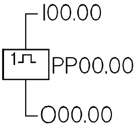
The controller has programmable pulses for status change recognition of logical signals (edge analysis). They can be used for both the positive and the negative edge.

6.7.1. Programmable pulse with positive edge

Circuit diagram



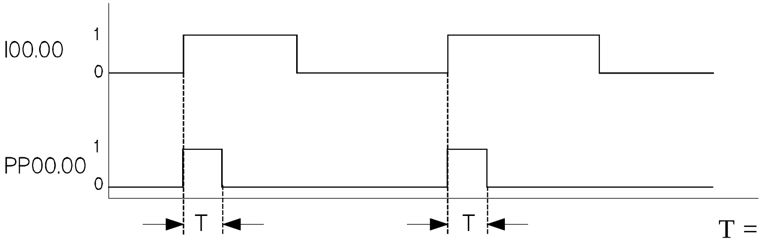
Switching symbol



Instruction list

L	I00.00
=	PP00.00
L	PP00.00
=	O00.00

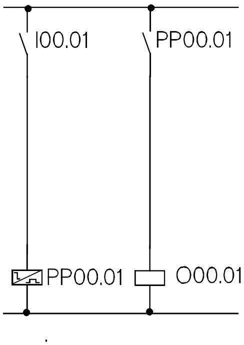
Signal course



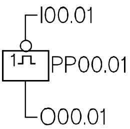
T = 1 cycle

6.7.2. Programmable pulse with negative edge

Circuit diagram



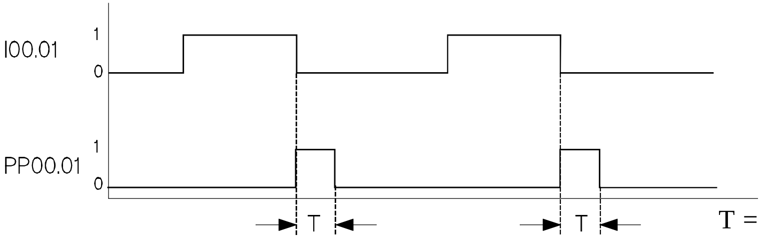
Switching symbol



Instruction list

L	I00.01
=N	PP00.01
L	PP00.01
=	O00.01

Signal course



T = 1 cycle

Behaviour of the progr. pulses after switching the controller on

After switching the controller on (or after RESET), the pulse has to be passed once at a value of 0 as the function cannot be guaranteed otherwise. Recommendation: Assign a pulse with a non-remanent marker and then set the input signal after it to the marker (see example below).

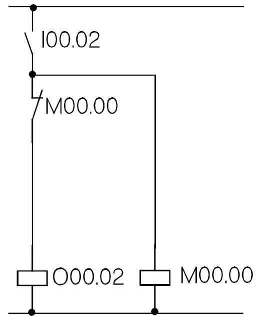
Example with positive pulse

L	M00.00
=	PP00.00
L	I00.00
=	M00.00
L	PP00.00
=	O00.00

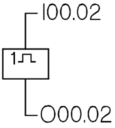
☞ *As opposed to the programmable pulses (see above) which are activated by edge changeovers, the signal status is evaluated in the two following examples. This causes a different behaviour when switching the controller on.*

6.7.3. Pulse with positive signal

Circuit diagram



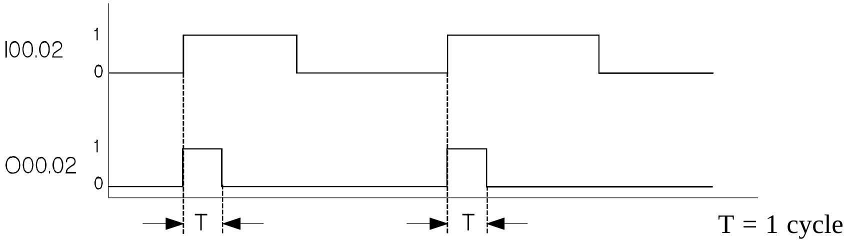
Switching symbol



Instruction list

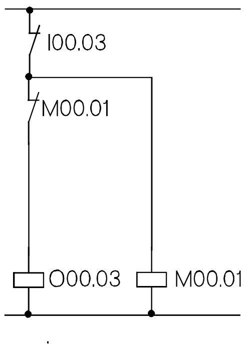
```
L       I00.02
=       SM15.14
AN      M00.00
=       O00.02
L       SM15.14
=       M00.00
```

Signal course

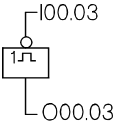


6.7.4. Pulse with negative signal

Circuit diagram



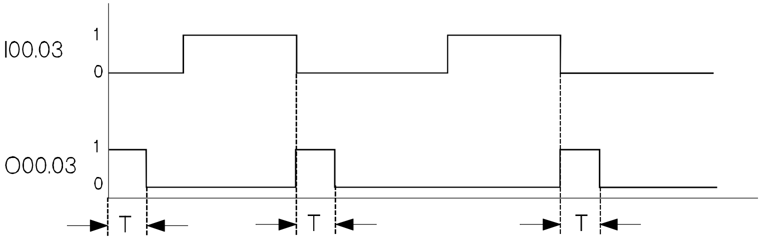
Switching symbol



Instruction list

LN	I00.03
=	SM15.14
AN	M00.01
=	O00.03
L	SM15.14
=	M00.01

Signal course

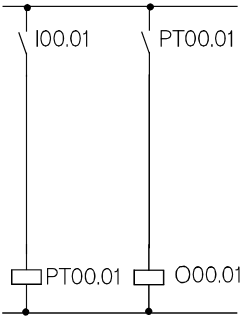


T = 1 cycle

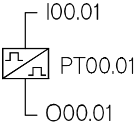
6.8. Software timers

6.8.1. Impulse at startup

Circuit diagram



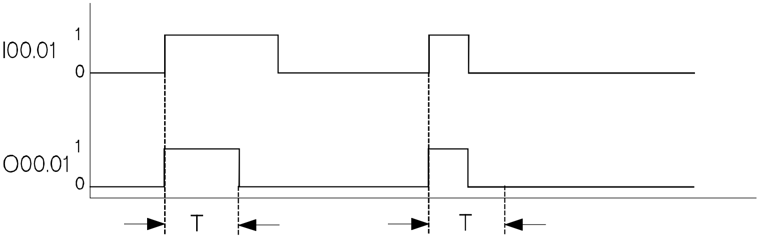
Switching symbol



Instruction list

```
L   I00.01
=   PT00.01:135*10ms:P
L   PT00.01
=   O00.01
```

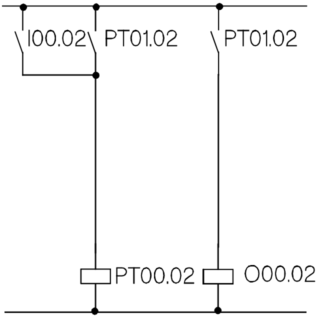
Signal course



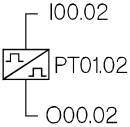
T= Time preselection (here: 1.35s)

6.8.2. Impulse of constant length

Circuit diagram



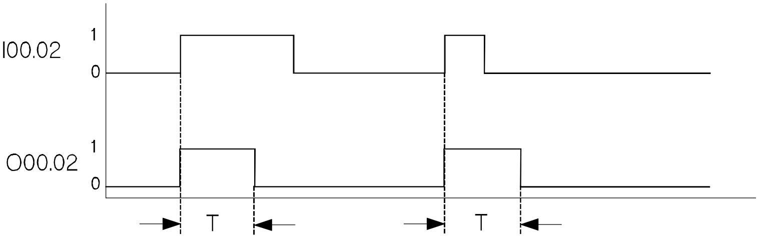
Switching symbol



Instruction list

L I00.02
O PT00.02
= PT00.02:123*100ms:P
L PT00.02
= O00.02

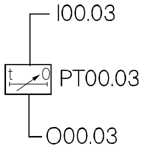
Signal course



T= Time preselection (here: 12.3s)

6.8.3. Raising delay

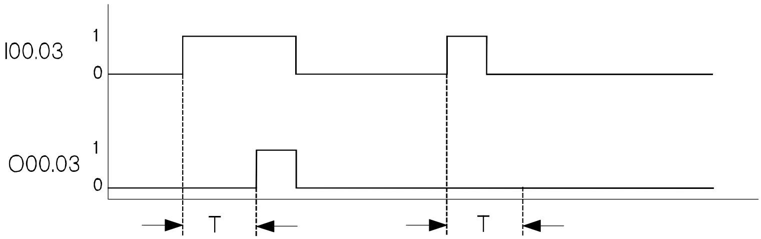
Switching symbol



Instruction list

```
L   I00.03
=   PT00.03:185*10ms:R
L   PT00.03
=   O00.03
```

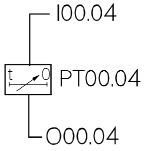
Signal course



T= Time preselection (here: 1.85s)

6.8.4. Falling delay

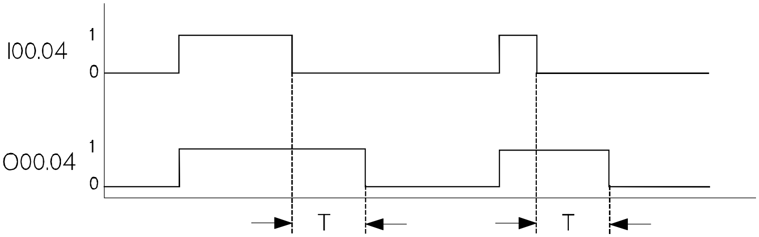
Switching symbol



Instruction list

```
L   I00.04
=   PT00.04:35*100ms:F
L   PT00.04
=   O00.04
```

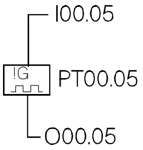
Signal course



T= Time preselection (here: 3.5s)

6.8.5. Impulse generator with pulse output

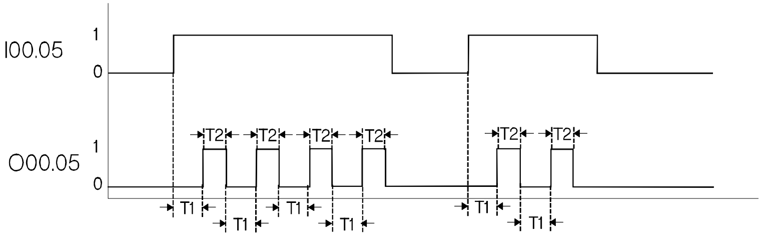
Switching symbol



Instruction list

```
L       I00.05
AN      O00.05
=       PT00.05:55*10ms:R
L       PT00.05
=       O00.05
```

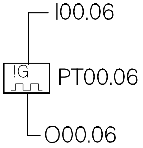
Signal course



T1= Time preselection (here: 0.55s)
T2= Cycle time

6.8.6. Flash generator with one timer

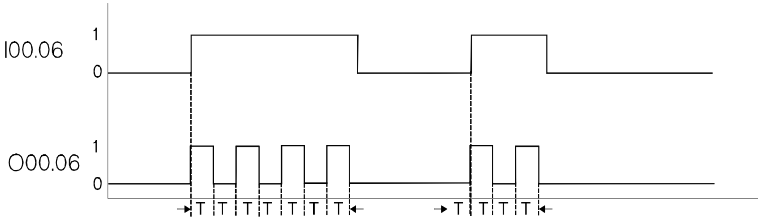
Switching symbol



Instruction list

```
L       I00.06
=       PT00.06:50*10ms:C
L       PT00.06
=       O00.06
```

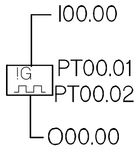
Signal course



T= Time preselection (here: 0.50s), Flash frequency = 1 Hz

6.8.7. Flash generator with two timers

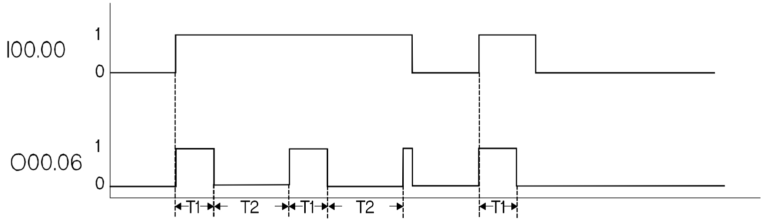
Switching symbol



Instruction list

```
L       I00.00
AN      PT00.02
=       PT00.01:5*100ms:P
L       PT00.01
=       O00.00
LN      PT00.01
=       PT00.02:10*100ms:P
```

Signal course



T1= Time preselection for switch-on (here: 500ms=0.5s)
T2= Time preselection for switch-off (here: 1,000ms=1s)

6.9. Programmable clock

Apart from the software timers, there are four programmable clock pulses available in the operands PC00.00 - PC00.03:

Operand	Clock pulse	Range
T00.00	10 ms	0-255
T00.01	100 ms	
T00.02	1 s	
T00.03	10 s	

Each of these operands is automatically incremented in the stated clock pulse. At 255, the next clock pulse causes a carry to 0.

<u>Example for an application:</u> Each part of the program is supposed to be passed only every 100 ms.

```
P1_STA  L      PC00.01     ;is 100 ms clock pulse memory
        CMP    BM03.14     ;    equal to the old value?
        JP=    P1_END      ;go to end of program if so
        =      BM03.14     ;    otherwise new = old
        .
        .                  ;this program is only
        .                  ; passed every 100 ms
        .
        .
        LN     O01.03      ;program for flash
        =      O01.03      ; generator 100 ms
        .
        .
P1_END  .
```

☞

```
at      L      PC00.01
        =      SM00.10
```

*the logical status of SM00.10 changes every 128 * 100 ms as bit 7 of PC00.01 is evaluated for the output of SM00.10.*

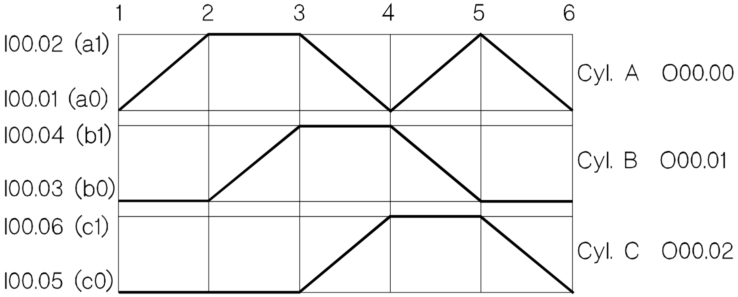
6.10. Software counters

Example: Forward counter to 12

```
L       I00.00        ;start counter
=       C00.00:12:F

L       I00.01        ;count (transfer clock pulse)
=C      C00.00

L       C00.00        ;scanning "Count completed”
=       A00.12

LD      C00.00        ;scan actual value
=D      BM00.00
```

6.11. Programming an operational sequence

Use the path-step diagram below to program a step chain (also called sequential function chart).

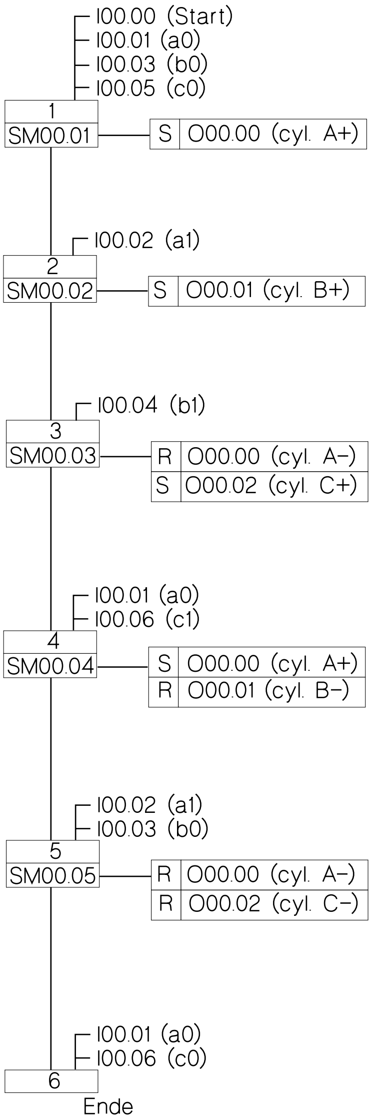


We will suggest two solutions:

6.11.1 Step chain with step markers

For this program solution you set a marker for every step of the program. The markers are cleared only at the end of the program and the step chain released for a new processing cycle.

Function diagram



Program

```
L       I00.00      ;Start
A       I00.01      ;Limit switch a0
A       I00.03      ;Limit switch b0
A       I00.05      ;Limit switch c0
AN      SM00.01     ;Step 1
S       SM00.01     ;Step 1
S       O00.00      ;Cylinder A+

L       I00.02      ;Limit switch a1
A       SM00.01     ;Step 1
AN      SM00.02     ;Step 2
S       SM00.02     ;Step 2
S       O00.01      ;Cylinder B+

L       I00.04      ;Limit switch b1
A       SM00.02     ;Step 2
AN      SM00.03     ;Step 3
S       SM00.03     ;Step 3
R       O00.00      ;Cylinder A-
S       O00.02      ;Cylinder C+

L       I00.01      ;Limit switch  a0
A       I00.06      ;Limit switch c1
A       SM00.03     ;Step 3
AN      SM00.04     ;Step 4
S       SM00.04     ;Step 4
S       O00.00      ;Cylinder A+
R       O00.01      ;Cylinder B-

L       I00.02      ;Limit switch a1
A       I00.03      ;Limit switch b0
A       SM00.04     ;Step 4
AN      SM00.05     ;Step 5
S       SM00.05     ;Step 5
R       O00.00      ;Cylinder A-
R       O00.02      ;Cylinder C-

L       I00.01      ;Limit switch a0
A       I00.05      ;Limit switch c0
A       SM00.05     ;Step 5
R       SM00.01     ;Step 1
R       SM00.02     ;Step 2
R       SM00.03     ;Step 3
R       SM00.04     ;Step 4
R       SM00.05     ;Step 5
```

6.11.2. Step chain with automatic status registration

This program solution is based on a byte operand that is used as step counter. Jump instructions are used to branch to the step that is currently active. Once the condition for the next step is fulfilled, the step counter is incremented by 1. When the last step has been completed, the step counter is cleared and the step chain is ready for the next processing cycle.

Program:

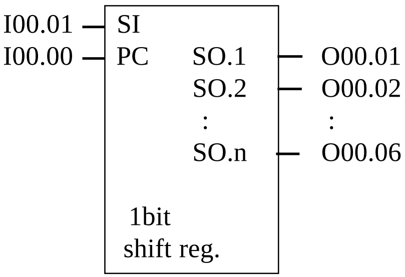
```
; Read step counter -> jump to next current step
         L       STEP_CNT        BM00.00 ; (step counter)
         CMP     0
         JP=     STEP_0                  ; jump to step no. 0
         CMP     1
         JP=     STEP_1                  ; jump to step no. 1
         CMP     2
         JP=     STEP_2                  ; jump to step no 2
         CMP     3
         JP=     STEP_3                  ; jump to step no 3
         CMP     4
         JP=     STEP_4                  ; jump to step no 4
         CMP     5
         JP=     STEP_5                  ; jump to step no 5
         CMP     6
         JP=     STEP_6                  ; jump to step no 6
         JP      END

; Step chain
STEP_0   L       START            I00.00 ; (start key)
         A       A0               I00.01 ; (limit switch A0)
         A       B0               I00.03 ; (limit switch B0)
         A       C0               I00.05 ; (limit switch C0)
         JPCN    END                     ; start condition fulfilled?
         INC     STEP_CNT        BM00.00 ; (step counter)
         JP      END
```

```
STEP_1   =1      CYL_A           O00.00 ; (cylinder A)
         L       A1              I00.02 ; (limit switch A1)
         JPCN    END                    ; condition for step fulfilled?
         INC     STEP_CNT       BM00.00 ; (step counter)
         JP      END
STEP_2   =1      CYL_B           O00.01 ; (cylinder B)
         L       B1              I00.04 ; (limit switch B1)
         JPCN    END                    ; condition for step fulfilled?
         INC     STEP_CNT       BM00.00 ; (step counter)
         JP      END

STEP_3   =0      CYL_A           O00.00 ; (cylinder A)
         =1      CYL_C           O00.02 ; (cylinder C)
         L       A0              I00.01 ; (limit switch A0)
         A       C1              I00.06 ; (limit switch C1)
         JPCN    END                    ; condition for step fulfilled?
         INC     STEP_CNT       BM00.00 ; (step counter)
         JP      END

STEP_4   =1      CYL_A           O00.00 ; (cylinder A)
         =0      CYL_B           O00.01 ; (cylinder B)
         L       A1              I00.02 ; (limit switch A1)
         A       B0              I00.03 ; (limit switch B0)
         JPCN    END                    ; condition for step fulfilled?
         INC     STEP_CNT       BM00.00 ; (step counter)
         JP      END

STEP_5   =0      CYL_A           O00.00 ; (cylinder A)
         =0      CYL_C           O00.02 ; (cylinder C)
         L       A0              I00.01 ; (limit switch A0)
         A       C0              I00.05 ; (limit switch C0)
         JPCN    END                    ; condition for step fulfilled?
         INC     STEP_CNT       BM00.00 ; (step counter)
         JP      END

STEP_6   CLR     STEP_CNT       BM00.00 ; (step counter)

END      NOP
```

6.12. Register circuits

6.12.1. 1bit shift register

In this example, the shift register is 6 steps long. The signal input is shifted from O00.01 to O00.06 when the shift clock pulse is applied from I00.00.



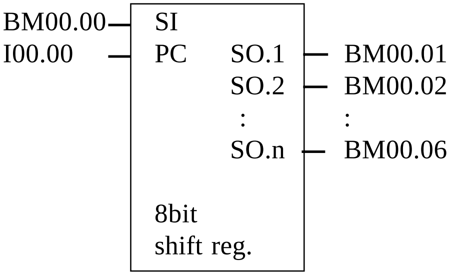
SI:	signal input	I00.01
PC:	shift clock pulse	I00.00
SO.1:	signal output 1	O00.01
SO.2:	signal output 2	O00.02
:	:	:
SO.n:	signal output n	O00.06

Instruction list

```
        L     I00.00       ;shift clock pulse
        =     PP00.00      ;pulse
        L     PP00.00      ;pulse
        JPCN  NORM         ;to normal program if no
        L     O00.05       ;step 5
        =     O00.06       ;step 6
        L     O00.04       ;step 4
        =     O00.05       ;step 5
        L     O00.03       ;step 3
        =     O00.04       ;step 4
        L     O00.02       ;step 2
        =     O00.03       ;step 3
        L     O00.01       ;step 1
        =     O00.02       ;step 2

        L     I00.01       ;signal input
        =     O00.01       ;step 1
NORM    :
        :                  ;normal program
```

6.12.2. 8bit shift register

In this example, the shift register is 6 steps long. The set information is shifted from BM00.00 to BM00.06 when the shift clock pulse is applied from I00.00.



SI:	signal input	BM00.01
PC:	shift clock pulse	I00.00
SO.1:	signal output 1	BM00.01
SO.2:	signal output 2	BM00.02
:	:	:
SO.n:	signal output n	BM00.06

Instruction list

```
          L       I00.00   ;shift clock pulse
          =       PP00.00  ;pulse
          L       PP00.00  ;pulse
          JPCN    NORM     ;to normal program if no
          L       BM00.05  ;step 5
          =       BM00.06  ;step 6
          L       BM00.04  ;step 4
          =       BM00.05  ;step 5
          L       BM00.03  ;step 3
          =       BM00.04  ;step 4
          L       BM00.02  ;step 2
          =       BM00.03  ;step 3
          L       BM00.01  ;step 1
          =       BM00.02  ;step 2

          L       BM00.00  ;signal input
          =       BM00.01  ;step 1
NORM      NOP
          :                ;normal program
          :
```

6.13. Bit-to-byte transfer

It is possible to transfer the contents of 8 or 16 1bit operands into byte operands in two operations. In the same way, the contents of byte operands can be copied directly into the 1bit range.

Example: To copy eight 1bit operands into one byte

```
C1T8  I00.00     ;copy contents of I00.00-I00.07 into the accumulator
=     BM00.00    ;assign contents of the accumulator to BM00.00
```

Example: To copy one byte into eight 1bit operands

```
L       BM00.01 ;load contents of BM00.01 into the accumulator
C8T1    O00.03  ;copy contents of the accu into operands O00.03-O00.10
```

Example: To copy sixteen 1bit operands into two bytes

```
C1T16   I01.00  ;load contents of I01.00-I01.15 into the accumulator
=D      BM00.02 ;copy contents of the accumulator into BM00.02-BM00.03
                ;(I01.00-I01.07 into BM00.02, I01.08-I01.15 into BM00.03)
```

Example: To copy two bytes into sixteen 1bit operands

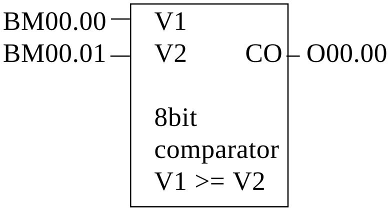
```
LD      BM00.04 ;load contents of BM00.04-BM00.05 into the accumulator
C16T1   O00.00  ;copy contents of the accu into the address O00.00-O00.15
                ;(BM00.04 into O00.00-O00.07,BM00.05 into O00.08-O00.15)
```

6.14. Comparator circuits

6.14.1. 8bit comparator

6.14.1.1. Result of the comparison: logical evaluation

The result of the comparison is evaluated as logical 1 or logical 0 by an assignment:



V1:	comparison value 1	BM00.00
V2:	comparison value 2	BM00.01
CO:	comparator output	O00.00

Program

```
L       BM00.00 ;compare V1 to V2
CMP>=   BM00.01 ; whether greater or equal *1)
=       O00.00  ;CA (becomes "1" if V1 is greater or equal, or
                     otherwise "0")
```

*1) further commands are: CMP=, CMP<>, CMP<=

6.14.1.2. Result of the comparison: evaluation with one jump

The result of the comparison is evaluated as a conditional jump, i.e. the jump is carried out if the result is "correct":

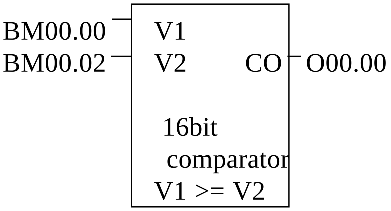
```
L           BM00.00
CMP         BM00.01
JP>=        MARK *2)
```

*2) further commands are: JP=, JP<>, JP<, JP<=, JP>

6.14.2. 16bit comparator

6.14.2.1. Result of the comparison: logical evaluation

The result of the comparison is as logical 1 or logical 0 in the accu and can be evaluated for example by an assignment.



V1:	comparison value 1 BM00.00+BM00.01
V2:	comparison value 2 BM00.02+BM00.03
CO:	comparator output O00.00

Program

```
LD        BM00.00 ;compare V1 to V2
CMPD      BM00.01 ; whether "greater or equal" *1)
=         O00.00  ;CA (is set if V1 is greater or equal)
```

*1) further commands are: CMPD=, CMPD<>, CMPD<=

6.14.2.2. Result of the comparison: evaluation with one jump

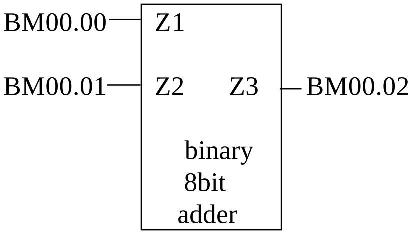
The result of the comparison is evaluated as a conditional jump, i.e. the jump is carried out if the result is "correct":

```
LD        BM00.00
CMPD      BM00.01
JP>=      MARK *2)
```

*2) further commands are: JP=, JP<>, JP<, JP<=, JP>

6.15. Arithmetic functions

6.15.1. Binary 8bit adder



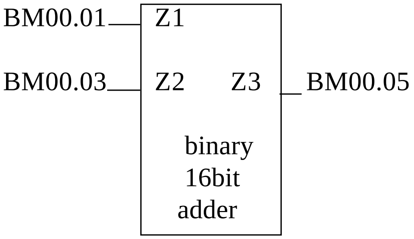
Z1:	1st summand BM00.00	8bit	0-255 ($FF)
Z2:	2nd summand BM00.01	8bit	0-255 ($FF)
Z3:	sum BM00.02	8bit	0-255 ($FF)

Program

```
L       BM00.00   ;Z1 1st summand
ADD     BM00.01   ;Z2 2nd summand
=       BM00.02   ;Z3 sum
```

In case of a carry, the carry bit is set.

6.15.2. Binary 16bit adder



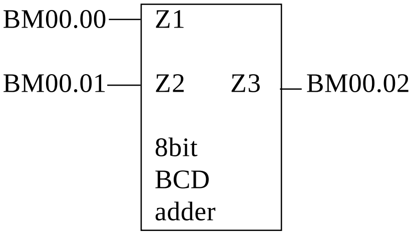
Z1: 1st summand 16bit 0-65535 ($FFFF)
BM00.01(HB)+BM00.00(LB)
Z2: 2nd summ. 16bit0-65535 ($FFFF)
BM00.03(HB)+BM00.02(LB)
Z3: sum 16bit 0-65535 ($FFFF)
BM00.05(HB)+BM00.04(LB)

Program

```
LD      BM00.00   ;Z1 1st summand
ADDD    BM00.02   ;Z2 2nd summand
=D      BM00.04   ;Z3 sum
```

In case of a carry, the carry bit is set.

6.15.3. 8bit BCD adder

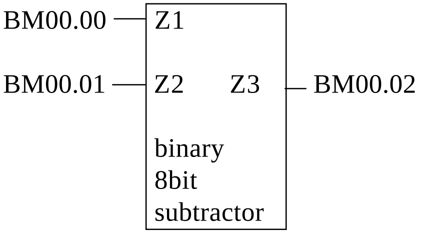


Z1:	1st summand BM00.00	8bit	0-99
Z2:	2nd summand BM00.01	8bit	0-99
Z3:	sum BM00.02	8bit	0-99

Program

```
    ****** BCD correction ******************************

          CLR   LBM00.01    ;marker for BCD correction
          L     BM00.00     ;Z1 1st summand
          A     %00001111   ;extract upper 4 bits
          =     LBM00.00    ;1st decade of this
          L     BM00.01     ;Z2 2nd summand
          A     %00001111   ;1st decade of this
          ADD   LBM00.00
          CMP   10          ;BCD correction necessary?
          JP<   ADDIT       ;jump if not
          L     6           ;load correction
          =     LBM00.01    ;value if yes

    ****** Addition ***********************************

ADDIT     L     LBM00.01
          ADD   BM00.00     ;Z1 1st summand
          ADD   BM00.01     ;Z2 2nd summand
          =     BM00.02     ;Z3 sum
```

6.15.4. Binary 8bit subtractor



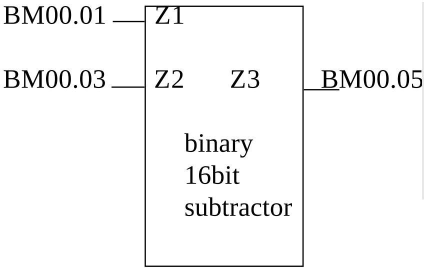
Z1: minuend 8bit 0-255 ($FF)
BM00.00
Z2: subtrahend 8bit 0-255 ($FF)
BM00.01
Z3: difference 8bit 0-255 ($FF)
BM00.02

⚠ *Z3 becomes negative and is filed as two’s complement if Z2 > Z1. Further evaluation of Z3 has to take this into account.*

Program

```
L       BM00.00   ;Z1 minuend
SUB     BM00.01   ;Z2 subtrahend
=       BM00.02   ;Z3 difference
```

6.15.5. Binary 16bit subtractor

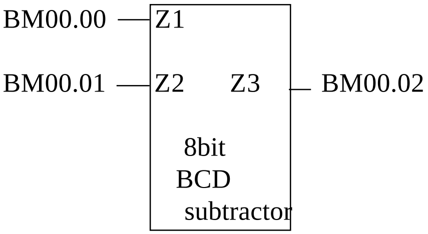


Z1: 1st minuend 16bit 0-65535 ($FFFF)
BM00.01(HB)+BM00.00(LB)
Z2: 2nd subtrahend 16bit 0-65535 ($FFFF)
BM00.03(HB)+BM00.02(LB)
Z3: difference 16bit 0-65535 ($FFFF)
BM00.05(HB)+BM00.04(LB)

Program

```
LD      BM00.00   ;Z1 minuend
SUBD    BM00.02   ;Z2 subtrahend
=D      BM00.04   ;Z3 difference
```

6.15.6. 8bit BCD subtractor

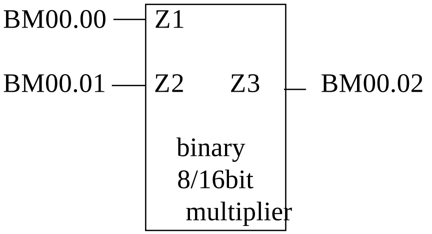


Z1:	minuend BM00.00	8bit	0-99
Z2:	subtrahend BM00.01	8bit	0-99
Z3:	difference BM00.02	8bit	0-99

Program

```
    ****** BCD correction ******************************

          L      BM00.00     ;Z1 minuend
          A      %00001111   ;mask upper 4 bits
          =      LBM00.00    ;1st decade of this
          L      BM00.01     ;Z2 subtrahend
          A      %00001111   ;1st decade of this
          CMP    LBM00.00    ;BCD correction necessary?
          JP<=   SUBTR       ;jump if not
          L      BM00.01     ;load correction
          ADD    6           ;value if yes
          =      BM00.01

    ****** Subtraction ***********************************

SUBTR     L      BM00.00     ;Z1 minuend
          SUB    BM00.01     ;Z1 subtrahend
          =      BM00.02     ;Z3 difference
```

6.15.7. Binary 8bit multiplier

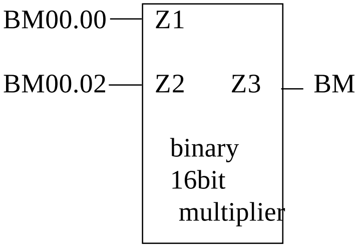


Z1: multiplicand 8bit 0-255 ($FF)
BM00.00
Z2: multiplier 8bit 0-255 ($FF)
BM00.01
Z3: product 16bit 0-65025 ($FE01)
BM00.03(HB)+BM00.02(LB)

Program

```
        L       BM00.00 ;Z1 multiplicand
        MUL     BM00.01 ;Z2 multiplier
        =D      BM00.02 ;Z3 product (16bit)
```

6.15.8. Binary 16bit multiplier

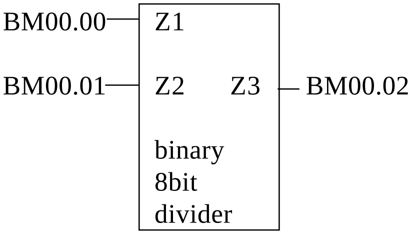


Z1: multiplicand 16bit 0-65535 ($FFFF)
BM00.01(HB)+BM00.00(LB)
Z2: multiplier 16bit 0-65535 ($FFFF)
BM00.03(HB)+BM00.02(LB)
Z3: product 16bit 0-65535 ($FFFF)
BM00.05(HB)+BM00.04(LB)

Program

```
        LD      BM00.00 ;Z1 multiplicand
        MULD    BM00.02 ;Z2 multiplier
        =D      BM00.04 ;Z3 product
```

6.15.9. Binary 8bit divider

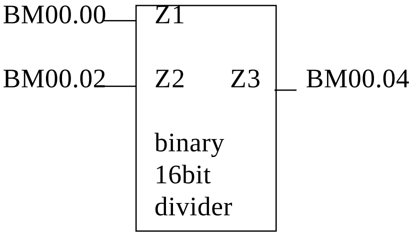


Z1:	dividend BM00.00	8bit	0-255 ($FF)
Z2:	divisor BM00.01	8bit	0-255 ($FF)
Z3:	quotient BM00.02	8bit	0-255 ($FF)

Program

```
L       BM00.00   ;Z1 dividend
DIV     BM00.01   ;Z2 divisor
=D      BM00.02   ;Z3 quotient
```

6.15.10. Binary 16bit divider



Z1:	dividend 16bit 0-65535 ($FFFF) BM00.01(HB)+BM00.00(LB)
Z2:	divisor 16bit 0-65535 ($FFFF) BM00.03(HB)+BM00.02(LB)
Z3:	quotient 16bit 0-65535 ($FFFF) BM00.05(HB)+BM00.04(LB)

Program

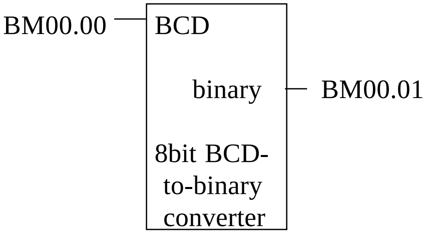
```
LD      BM00.00   ;Z1 dividend
DIVD    BM00.02   ;Z2 divisor
=D      BM00.04   ;Z3 quotient
```

The calculated quotient is integer. The remainder can be determined as follows:

```
LD      BM00.04   ;Z3 quotient
MULD    BM00.02   ;Z2 divisor
=D      LBM00.00  ;Z3 (whole number!) * Z2
LD      BM00.00   ;Z1 dividend
SUBD    LBM00.00
=D      BM00.06   ;remainder
```

6.16. Code converters

6.16.1. 8bit BCD-to-binary converter

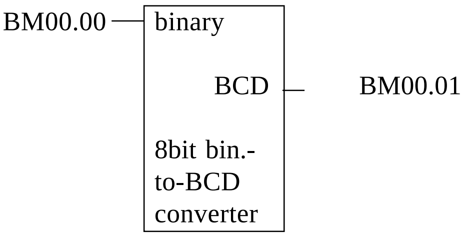


BCD:	8bit	0-99	BM00.00
binary:	8bit	0-99 ($63)	BM00.01

Program

```
L       BM00.00    ;load BCD value
LSR                ;shift
LSR                ; tens
LSR                ; to
LSR                ; digits
MUL     10         ;multiply
=       BM00.01    ;store temporarily
L       BM00.00    ;load BCD value
A       %00001111  ;mask tens
ADD     BM00.01    ;add binary tens
=       BM00.01    ;store binary value
```

6.16.2. 8bit binary-to-BCD converter

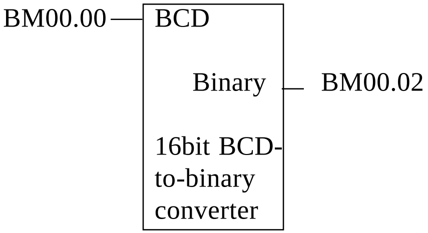


binary:	8bit	0-99	($63)	BM00.00
BCD:	8bit	0-99		BM00.01

Program

```
L         BM00.00     ;load binary value
DIV       10          ;determine and
=         LBM00.00    ;mark tens
MUL       10          ;calculate and mark down
=         LBM00.01    ;integer tens value
L         BM00.00
SUB       LBM00.01    ;determine and
=         LBM00.01    ;mark units
L         LBM00.00    ;shift
LSL                   ;  tens
LSL                   ;  into the
LSL                   ;  upper
LSL                   ;  nibble
O         LBM00.01    ;pack and output
=         BM00.01     ; BCD value
```

6.16.3. 16bit BCD-to-binary converter

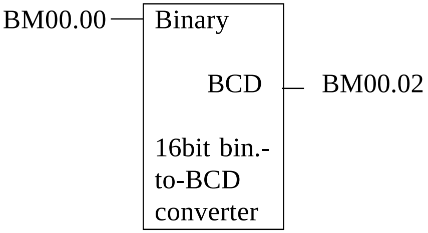


BCD:	16bit 0-9999 BM00.01(HB)+BM00.00(LB)
binary:	16bit 0-9999 ($270F) BM00.03(HB)+BM00.02(LB)

Program

```
CLR     BM00.03     ;clear because of LD  BM00.02
CLR     LBM00.03    ;clear because of LD  LBM00.02
L       BM00.00     ;separate units decade
A       %00001111
=       BM00.02     ;binary units
L       BM00.00     ;separate tens decade
LSR
LSR
LSR
LSR                 ;binary tens
MUL     10
ADD     BM00.02
=       BM00.02     ;units+tens
L       BM00.01     ;separate hundreds decade
A       %00001111
=       LBM00.02    ;binary hundreds
LD      LBM00.02    ;the same as word
MULD    100
ADDD    BM00.02
=D      BM00.02     ;units+tens+hundreds
L       BM00.01     ;separate thousands decade
LSR
LSR
LSR
LSR
=       LBM00.02    ;binary thousands
LD      LBM00.02    ;the same as word
MULD    1000
ADDD    BM00.02
=D      BM00.02     ;complete binary value
```

6.16.4. 16bit binary-to-BCD converter

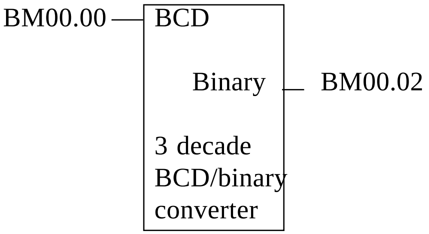


Binary:	16bit 0-9999 ($270F) BM00.01(HB)+BM00.00(LB)
BCD:	16bit 0-9999 BM00.03(HB)+BM00.02(LB)

Program

```
        CLR     BM00.02     ;set to zero
        CLR     BM00.03     ;ditto
THOU1   LD      BM00.00     ;load binary value
        CMPD    1000
        JP<     THOU2       ;smaller than a thousand?
        SUBD    1000        ;subtract 1000 if yes
        =D      BM00.00
        INC     BM00.03     ;count subtraction steps
        JP      THOU1       ;back to enquiry
THOU2   L       BM00.03     ;shift thousands
        LSL                 ;  into the upper
        LSL                 ;  nibble of the
        LSL                 ;  highbyte of the
        LSL                 ;  BCD output if no
        =       BM00.03     ;prepare highbyte
HUND    LD      BM00.00     ;remainder of binary value (thousands excl.)
        CMPD    100
        JP<     TEN1        ;smaller than a hundred?
        SUBD    100         ;subtract 100 if yes
        =D      BM00.00
        INC     BM00.03     ;count subtraction steps (in the lower nibble
                            ;  of the highbyte of the BCD output)
        JP      HUND        ;back to enquiry
TEN1    L       BM00.00     ;remainder of binary value (hundreds excl.)
        CMP     10
        JP<     TEN2        ;smaller than ten?
        SUB     10          ;subtract 10 if yes
        =       BM00.00
        INC     BM00.02     ;count subtraction steps
        JP      TEN1        ;back to enquiry
TEN2    L       BM00.02     ;shift tens to the
        LSL                 ;  upper nibble of the
        LSL                 ;  lowbyte of the
        LSL                 ;  BCD output
        LSL                 ;  if no
        ADD     BM00.00     ;units remainder into the lower nibble
        =       BM00.02     ;output from the lowbyte
```

6.16.5. 3 decade BCD-to-binary converter



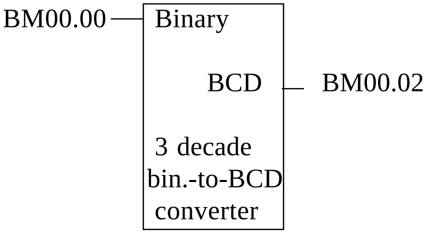
BCD:	16bit 0-999 BM00.01(HB)+BM00.00(LB)
binary:	16bit 0-999 ($03E7) BM00.03(HB)+BM00.02(LB)

Program

```
LD      BM00.00 ;load BCD value
BCDBIN3
=D      BM00.02 ;output binary value
```

☞ *If there are 3-decade BCD values to be calculated with arithmetically, we recommend first converting these into binary values by use of command BCDBIN3 and then executing the arithmetic operations with binary values.*

6.16.6. 3 decade binary-to-BCD converter



binary: 16bit 0-999 ($03E7)
BM00.01(HB)+BM00.00(LB)
BCD: 16bit 0-999
BM00.03(HB)+BM00.02(LB)

Program

```
LD          BM00.00     ;load binary value
BINBCD3
=D          BM00.02     ;output BCD value
```

6.16.7. 10bit analog-to-binary conversion

Analog values in the KUBES format require 16bit operands. The actual analog value is not flush in the 16bit word. Before you can calculate analog values or compare them to binary values you have to shift them right within the word.

In the case of 10bit analog values you need a shift by 5 digits:

Program

```
LD      AI00.00  ;analog input
LSRD             ;logical shift right
LSRD
LSRD
LSRD
LSRD
=D      BM00.00  ;write into byte marker as binary value
```

6.17. Module programming

Task (example):

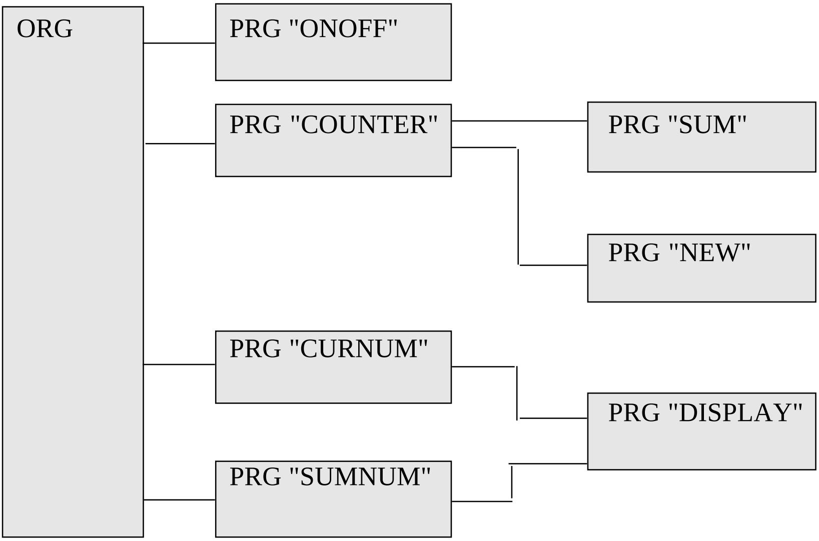
Sets of 12 pieces each are to be transported on a conveyor belt. The drive of the belt is operated by start and stop keys. The belt is stopped after every twelfth piece. Before leaving the belt, each piece triggers an impulse via an initiator which is used for counting.

A 3-digit BCD display is supposed to show:

- while the belt is running:
  the current piece number in the set (0...12)
- permanently:
  the sum total of pieces transported already (0...999)

You should be able to set the counter to zero via a cancel key.

The overall program is realized by a practical dividing it up into separate modules (see next page for a program printout):



Printout of program listing

```
======== Kubes ==================================== KUAX 680C =======
                         Project structure

Project    : E205GB
                                  created : Nov 19 1991 09:42
User       : Kevin Kubes          altered : Nov 21 1991 08:17
Comment    : Example "Module programming”
=======================================================================


ORG.ORG/1
|
*——>ONOFF.PRO/1
|
*——>COUNTER.PRO/2
|     |
|        *——>SUM.PRO/5
|     |
|         *——>NEW.PRO/6
|
*——>CURNUM.PRO/3
|      |
|        *——>DISPLAY.PRO/7
|
*——>SUMNUM.PRO/4
       |
         *——>DISPLAY.PRO/7
```

```
========  Kubes ====================================  KUAX  680C  ======
                         Organisation module   IL

Project   : E205GB
Module    : ORG        No.: 1      created : Nov 26 1991 16:08
User      : KUBES                  altered : Nov 26 1991 16:08

=======================================================================


  1:          JPP     ONOFF                 1
  2:
  3:          JPP     COUNTER               2
  4:
  5:          L        MOTOR           O00.00 ; (motor conveyor belt)
  6:          JPCP    CURNUM                3
  7:
  8:          LN       MOTOR           O00.00 ; (motor conveyor belt)
  9:          JPCP    SUMNUM                4
 10:
```

```
========  Kubes ====================================  KUAX  680C  =======
                         Program module   IL

Project    :  E205GB
Module     :  DISPLAY    No.: 7     created : Nov 26 1991 16:20
User       :  Kevin Kubes           altered : Nov 26 1991 16:20
Comment    :  DISPLAY
=======================================================================


  1:          LD      BM00.02
  2:          BINBCD3
  3:          C16T1   UNITS          S000.00 ; (display "digits”)
  4:
```

```
========  Kubes =====================================  KUAX  680C  =======
                          Program module  IL

Project    : E205GB
Module     : CURNUM     No.: 3     created: Nov 26 1991 16:20
User       : Kevin Kubes           altered: Nov 26 1991 16:20
Comment    : CURNUM
==========================================================================


  1:           LD     COUNTER         C00.00 ; (piece counter)
  2:           =D     BM00.02
  3:           JPP    DISPLAY              7
  4:
```

```
========  Kubes =====================================  KUAX  680C  =======
                          Program module  IL

Project    : E205GB
Module     : SUMNUM     No.: 4     created : Nov 26 1991 16:22
User       : Kevin Kubes           altered : Nov 26 1991 16:22
Comment    : SUMNUM
==========================================================================


  1:           LD     SUM            BM00.00 ; (current piece number)
  2:           =D     BM00.02
  3:           JPP    DISPLAY             7
  4:
```

```
======== Kubes =================================== KUAX 680C =======
                        Program module  IL

Project   : E205GB
Module    : ONOFF    No.: 1      created : Nov 26 1991 16:12
User      : Kevin Kubes          altered : Nov 26 1991 16:12
Comment   : ONOFF
====================================================================


  1:          L     START          I00.00 ; (start motor)
  2:          S     IOMARKER       M00.00 ; (marker motor ON/OFF)
  3:          L     STOP           I00.01 ; (stop motor)
  4:          ON    READY          M00.01
  5:          O     DONE           M00.02 ; (12 pieces counted)
  6:          R     IOMARKER       M00.00 ; (marker motor ON/OFF)
  7:          L     IOMARKER       M00.00 ; (marker motor ON/OFF)
  8:          =     MOTOR          O00.00 ; (motor conveyor belt)
  9:
```

```
======== Kubes =================================== KUAX 680C =======
                        Program module  IL

Project   : E205GB
Module    : NEW       No.: 6     created : Nov 26 1991 16:19
User      : Kevin Kubes          altered : Nov 26 1991 16:19
Comment   : NEW
====================================================================


  1:          LD     0
  2:          =D    SUM          BM00.00 ; (current piece number)
  3:
```

```
========  Kubes =====================================  KUAX  680C  =======
                         Program module  IL

Project   : E205GB
Module    : SUM       No.: 5     created : Nov 26 1991 16:18
User      : Kevin Kubes          altered: Nov 26 1991 16:18
Comment   : SUM
=========================================================================


  1:          LD     COUNTER          C00.00 ; (piece counter)
  2:          ADDD   SUM             BM00.00 ; (current piece number)
  3:          =D     SUM             BM00.00 ; (current piece number)
  4:
```

```
========  Kubes =====================================  KUAX  680C  =======
                         Program module  IL

Project   : E205GB
Module    : COUNTER   No.: 2     created : Nov 26 1991 16:15
User      : Kevin Kubes          altered : Nov 26 1991 16:15
Comment   : COUNTER
=========================================================================


  1:          L      COUNTER          C00.00 ; (piece counter)
  2:          O      STOP             I00.01 ; (motor off)
  3:          =      PULSE           PP00.00
  4:          L      PULSE           PP00.00
  5:          JPCP   SUM                   5
  6:          L      IOMARKER         M00.00 ; (marker motor ON/OFF)
  7:          =      COUNTER:12:V     C00.00 ; (piece counter)
  8:          L      CIMP             I00.02 ; (counting pulse of the initiator)
  9:          =C     COUNTER          C00.00 ; (piece counter)
 10:          L      CANCEL          I00.03  ; (key "clear counter”)
 11:          JPCP   NEW                   6
 12:
```

6.18. Data modules

6.18.1. Creating data modules offline

Many PLC applications require text management for displays, operating terminals etc. The texts can be written with a simple text editor that uses no control characters. Windows, in its Accessories group, provides NOTEPAD.EXE for such purposes.

6.18.1.1. Creating a data file with the text editor

- At a suitable location, create a sub-directory for your collection of data modules (in this case: C:\KUBESEXE\DATA).

- Start text editor NOTEPAD.EXE by double-clicking on its program icon.

- Write the following text without word wrapping (by pressing <Enter>):
  If on the manure the cock does crow, the weather stays calm or a strong gale will blow.

- Save File, set path to
  C:\KUBESEXE\DATA, filename **WEATHER.DAT**

6.18.1.2. Creating the data module

- KUBES, Module Editor, Create Module: data module called WEATHER and no.1, OK.
  The Create Data Module dialog is displayed.

- Specify a data range address in the 'RAM module' section:
  Bank number: **`0`**
  Start address: **`8000`**
  (Remember later when setting the memory size under KUBES!)

- In the Initialized section click on 'from file'
  The Open File dialog is displayed
  Select **`C:\KUBESEXE\DATA\WEATHER.DAT`**, OK. The Open File dialog is closed
  The Create Data Module dialog is now set to initialization from **`WEATHER.DAT`**, OK. Execution is confirmed by 2 messages:
  1st message:
  File is smaller than data module. Filled with 0.
  2nd message:
  Data module successfully created: WEATHER.

6.18.1.3. Importing a data module

If a data module already exists in your project, you can import data from a data file into the data module. The process is similar to the initial creation of the corresponding data module.

- KUBES, Module Editor, Module menu, Import data module
  The Import Data Module dialog is displayed.

- In the RAM Module section check the data range address:
  Bank number: **0**
  Start address: **8000**
  (Remember later when setting the memory size under KUBES!)

- Click on Select Source File
  The Open File dialog is displayed
  Select **C:\KUBESEXE\DATA\WEATHER.DAT**, OK. The Open File dialog is closed
  The Create Data Module dialog is now set to the initialization from **WEATHER.DAT**, OK. Execution is confirmed by 1 message:
  1st message:
  File is smaller than data module. Filled with 0.

6.18.1.4. Testing the data module

- KUBES, Module Editor, Load Module: ORG
  Write down the program line:
  ```
  LoadDB  0,WEATHER
  ```
  This loads data module WEATHER into data processing range 0.

- KUBES, PLC, Online (depending on interface and PLC type)

- KUBES, PLC, Stop and Reset:

- KUBES, PLC, Set memory size:
  Bank 0: 52 kB, data range remains empty.
  Bank 1: 0 kB, data range $0000 - $FFFF appears.

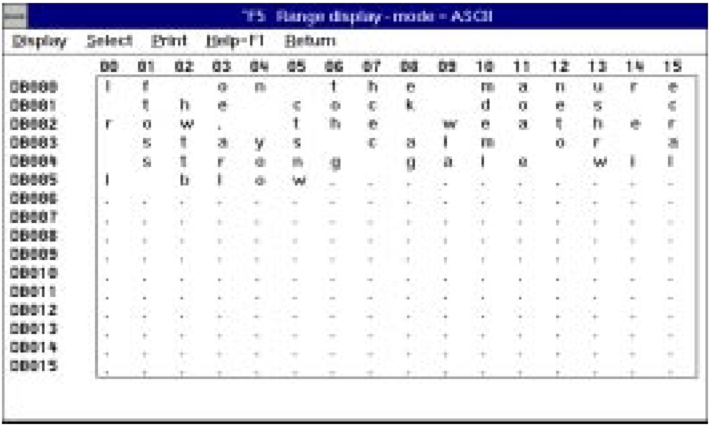
- KUBES, PLC, Transmit Program:
  The dialog 'Select data modules for transmitting' is displayed
  Select data module WEATHER in section 'Data modules in the project', accept. Highlight data module WEATHER in section 'Data modules to be transmitted', OK
  The project and all data modules are now transmitted to the controller.

- KUBES, PLC, Start:

- KUBES, PLC, Display address range, Select: Byte marker DB0, Display ASCII, Dynamic display:

The following should now be displayed on your screen.

☞ *see next page*



	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
DB000	I	f		o	n		t	h	e		m	a	n	u	r	e
DB001		t	h	e		c	o	c	k		d	o	e	s		c
DB002	r	o	w	.		t	h	e		w	e	a	t	h	e	r
DB003		s	t	a	y	s		c	a	l	m		o	r		a
DB004		s	t	r	o	n	g		g	a	l	e		w	i	l
DB005	l		b	l	o	w	.	.	.	.	.	.	.	.	.	.
DB006	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB007	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB008	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB009	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB010	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB011	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB012	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB013	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB014	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
DB015	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

6.18.2. Creating data modules online

6.18.2.1. Creating a data module

- KUBES, Module Editor, Create Module: Data module called TEST and No.2, OK.
  The Create Data Module dialog is displayed.

- Input data range address in the RAM Module section:
  Bank number: **0**
  Start address: **8100**
  (Remember later when setting the memory size under KUBES!)

- In section 'initialized' click on By Zero, OK. Execution is confirmed by one message:
  1st message: Data module successfully created: TEST.

6.18.2.2. Editing data modules in the display address range

- KUBES, Module Editor, Load Module: ORG
  Write down the program line:
  StoreDB 1,TEST
  This loads data processing range 1 into data module TEST.

- KUBES, PLC, ONLINE (depending on interface and PLC type)

- KUBES, PLC, Stop and Reset:

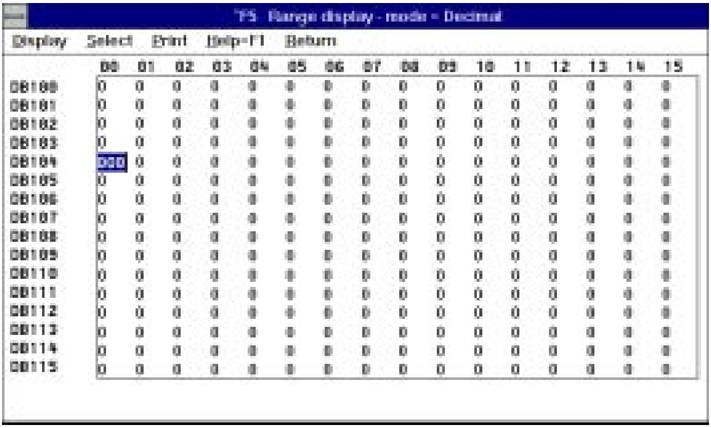
  KUBES, PLC, Set memory size:
  Bank 0: 52 kB, data area remains empty.
  Bank1: 0 kB, data range $0000 - $FFFF appears.

- KUBES, PLC, Transmit program:
  The dialog 'Select data modules for transmitting' is displayed Select data module TEST in section 'Data modules in the project', accept. Select data module TEST in section 'Data modules to be transmitted', OK.
  The project and all data modules is transmitted to the controller.

- KUBES, PLC, Start:

- KUBES, PLC, Display address range, Select: Byte marker DB1, Display decimal, dynamic off:
  Input 0,ENTER;1,ENTER;2,ENTER;... into addresses DB100.00...DB100.15.

The following should now be displayed on your screen.



	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
DB100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB103	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB104	000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB105	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB106	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB107	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB108	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB109	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB113	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB114	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DB115	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Command **`StoreDB   1,TEST`** writes the contents of data processing range DB1 in the PLC into data module TEST.

6.18.2.3. Loading data modules from the PLC

If you want to take over the contents of data module TEST you just created into the project, you have to download it from the controller

- KUBES, PLC online, Module Editor, PLC, Load data module
  The Load Data Module dialog is displayed.

- Select TEST, load.
  You are prompted: Target module already exists. Overwrite?

- Yes. Now data module TEST in the project has the edited contents. If you now remove command ’**`StoreDB   1,TEST`**’ from the program, data module TEST will remain unchanged in the PLC even after the next program transmissions.

6.18.2.4. Exporting data modules

If you want to archive a data module, make it available for other projects, or document its contents, it is also possible to export it.

- KUBES, Module Editor, Module, Export data module
  The Export Data Module dialog is displayed
  To select a data module: highlight TEST
  Click on destination file to select, the Save File As dialog is displayed.
  Set path to **`C:\KUBESEXE\DATA`**, input ‘TEST.DAT, OK.
  The Save Fiel As dialog is closed, and the Export Data Module dialog redisplayed, export.

- It is not possible to view file TEST.DAT with a text editor as only values 0...15 are contained. However, you can use DOS program debug.exe, for example, for viewing.
  - **`debug C:\KUBESEXE\DATA\ , ENTER`**
  - **`d , ENTER`**

  The contents of the first 128 bytes is shown.
  - **`d , ENTER`**

  The contents of the second 128 bytes is shown.

A. References to literature

Controllers

Instruction manual E 331 GB, KUAX 644 PC Control
Kuhnke GmbH, Malente

Instruction manual E 312 GB, KUAX 657P Profi Control
Kuhnke GmbH, Malente

Instruction manual E 399 GB, KUAX 680C Compact Control
Kuhnke GmbH, Malente

Instruction manual E 414 GB, Control Terminal KDT 680CT
Kuhnke GmbH, Malente

Instruction manual E 308 GB, KUAX 680I Profi Control
Kuhnke GmbH, Malente

Modules

Instruction manual E 357 GB, Modules of KUAX 657 and 657P
Kuhnke GmbH, Malente

Instruction manual E 326 GB, Modules of KUAX 680I, 680C and KDT 680CT
Kuhnke GmbH, Malente

Software

Beginner's manual E 327 GB, KUBES, Kuhnke User Software
Kuhnke GmbH, Malente

Instruction manual E 386 GB, KUBES modules
Kuhnke GmbH, Malente

Instruction manual E 365 GB, PROFIBUS
Kuhnke GmbH, Malente

B. Measuring the cycle time

This little program is perfect utility for measuring the cycle time of your program without requiring any other tools. We recommend embedding it in your program as a separate program module.

Definition:
The cycle time consists of program code, module calls, V.24 communication, watchdog monitoring and processing of timers and counters.

Program structure:

- Module "ZYKL_TST" (see next page) determines the cycle time of the controller in microseconds.
- The maximum cycle time that can be measured is 10000 µs (10 ms), or, optionally, 65 ms (see note at the end of the program).
- We recommend storing this module in a folder so that you can incorporate it in a project at any time.
- To call the module up in your project, include the instruction **`JPP ZYKL_TST`** in your ORG module program.
- If you are working with a KUAX 667 you have to replace the four **`LSLD`** commands (lines 16...19) by a **`MULD 16`** command.
- The utility measures the time (x * 10 ms (10000 µs)) needed for 625 cycles.. Multiplied by 16 (16 * 625 = 10000) this number is the cycle time.
- Module ZYKL_TST also extends the cycle time by approximately 40 µs.
- The result is written into BM15.00 as a 16bit value. Display it either via a dialog terminal or vir the dynamic display under KUBES.

Addresses BM15.00...07 must not be used for writing operations anywhere else in the user program. If they are, you must use other operands for calculating the cycle time.

```
========   KUBES   =====================================================
                         Program module  IL
Project : PALL_644              Network  :
Module  : ZYKL_TST   No.: 9       created : Feb 11 1991 15:18
User     : Heinz-Werner Panck    changed : Feb 09 1996 09:58
Comment : Testing the cycle time
========================================================================

  1: ; ****  Count the 10 ms pulses  ****
  2:          L       T00.00                ; Timer 10ms compare
  3:          CMP     BM15.06               ;  with old value
  4:          JP=     CONTINU1              ; jump if =
  5:          =       BM15.06               ; otherwise new value
  6:          INCD    BM15.02               ; 10ms counter + 1
  7:
  8: ; ****  Counting the cycles  ****
  9: CONTINU1 INCD    BM15.04               ; cycle counter + 1
 10:          LD      BM15.04               ; counted enough
 11:          CMPD    625                   ;  cycles ?
 12:          JP<     END                   ; else ->
 13:
 14: ; ****  Multiply by 16  *****
 15:          LD      BM15.02               ; 10ms counter
 16:          LSLD                          ; multiplied
 17:          LSLD                          ; by 16
 18:          LSLD                          ; [ 16 * 625 = 10000 ]
 19:          LSLD
 20: ; !!!!!!!!!!!!!!!! the next line shows the result !!!!!!!!!!!!!!!!
 21:          =D      BM15.00               ; cycle time in microseconds
 22:
 23: ; **** Delete old values  ****
 24:          LD      0
 25:          =D      BM15.02
 26:          =D      BM15.04
 27: END      NOP
```

☞ *In case the cycle time is longer than 10 ms, exchange the 10ms clock pulse by the 100ms clock pulse (T00.0 -> T00.01) and replace the four LSLD commands (lines 16...19) by a MULD 160 command.*

Index