Kuhnke Electronics
Instruction Manual

Compact Control KUAX 680C

E 399 GB 16 January 1996 / 67.977

Table of contents

Appendix

1. Introduction

Compact Control KUAX 680C is the compact controller of the Kuhnke GmbH. With its compact structure it resembles much the KUAX 680I, the difference being that already in its basic configuration, i.e. without any modules, the controller is equipped with a practical amount of inputs/outputs and interfaces:

2 interfaces RS 232 one of which can be set to work as a RS 485 (with separate connection)
16 digital inputs
2 counter inputs for fast event counting
2 interrupt inputs
4 analog inputs
16 digital outputs
2 analog outputs

Should you need more you can extend the configuration of the KUAX 680C by up to 4 modules. All modules of the KUAX 680I can be used with the sole exception of the event counter module.

The processor ensures almost complete software compatibility. It is the same type of CPU that has proved its worth in the controllers KUAX 680I, KUAX 644 and KUAX 657P. For programming you need KUBES, the Kuhnke user software, version 4.12 or higher.

2. Safety and Reliability

2.1. Target group

This instruction manual contains all information necessary for the use of the described product (control device, software, etc.) according to instructions. It is written for 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 standards by extensive and cost-effective means in their design and manufacture.

These include:
- selecting high-quality components,
- quality arrangements with our sub-suppliers,
- measures for the prevention of static charge during the handling of MOS circuits,
- Worst-Case dimensioning of all circuits,
- inspections during various stages of fabrication,
- computer aided tests of all assembly groups and their coefficiency in the circuit,
- statistic assessment of the quality of fabrication and of all returned goods for immediate taking of corrective action.

Despite these measures, the occurrence of errors in electronic control units - even if most highly improbable - must be taken into consideration.

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. Dangers 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.

2.4. Safety

Our product normally becomes part of larger systems or installations. The following notes are intended to help integrating the product into its environment without dangers for man or material/equipment.

2.4.1. To be observed during project planning and installation

- 24V DC power supply:
  Generate as electrically safely separated low voltage. Suitable devices are, for example, split transformers constructed to correspond to European standard EN 60742 (corresponds to VDE 0551)
- In case of power breakdowns or power fades: the program has to be structured in such a way as to create a defined state at restart that excludes dangerous states.
- Emergency switch-off installations or other emergency installations have to be realized in accordance with EN 60204/ IEC 204 (VDE 0113). They must be effective at any time.
- Safety and precautions regulations for qualified applications have to be observed.
- Please pay particular attention to the notes of warning which, at relevant places, will make you aware of possible sources of dangerous mistakes or failures.
- The relevent standards and VDE regulations are to be observed in every case.
- Control elements have to be installed in such a way as to exclude unintended operation.
- Control cables have to be layed in such a way as to exclude interference (inductive or capacitive) which could influence the operation of the controller.

To achieve a high degree of conceptual safety in planning and installing an electronic controller it is essential to follow the instructions given in the manual exactly because wrong handling could lead to rendering measures against dangerous failures ineffective or to creating additional dangers.

2.4.2. To be observed during maintenance and servicing

- Precaution regulation VBG 4.0 must be observed, and §8 (Admissible deviations during working on parts) in particular, when measuring or checking a controller in a power-up condition, .
- Repairs must only be executed by the trained Kuhnke personnel (usually in the main factory in Malente). Warranty expires in any other case.
- Spare parts:
  Only use parts approved of by Kuhnke. Only genuine Kuhnke modules must be used in modular controllers.
- Modules must only be connected to or disconnected from the controller with no voltage supplied. Otherwise they may be destroyed or (possibly not immediately recognizably!) detracted from their proper functioning.
- Always deposit batteries and accumulators as hazardous waste.

2.5. Electromagnetic compatibility

2.5.1. Definition

Electromagnetic compatibility is the ability of a device to function satisfactorily in its electromagnetic environment without itself causing any electromagnetic interference that would be intolerable to other devices in this environment.

Of all known phenomena of electromagnetic noise, only a certain range occurs at the location of a given device. This noise depends on the exact location. It is determined in the relevant product standards.

The international standard regulating construction and degree of noise resistance of programmable logic controllers is IEC 1131-2 which, in Europe, has been the basis for European standard EN 61131-2.

2.5.2. Resistance to interference

Electrostatic discharge, ESD
in accordance with IEC 801-2, 3rd degree of sharpness

Fast transient interference, Burst
in accordance with IEC 801-4, 3rd degree of sharpness

Irradiation resistance of the device, HF
in accordance with IEC 801-3, 3rd degree of sharpness

Immunity to damped oscillations
in accordance with IEC 255-4 (1 MHz, 1 kV)

2.5.3. Interference emission

Interfering emission of electromagnetic fields, HF
in accordance with EN 55011, limiting value class A, group 1

If the controller is designed for use in residential districts, then high-frequency emissions must comply with limiting value class B as described in EN 55011.
Appropriate means for keeping the corresponding limiting values are fitting the controller into a grounded metal cabinet and equipping the supply cables with filters.

2.5.4. General notes on installation

As component parts of machines, facilities and systems, electronic control systems must comply with valid rules and regulations, depending on the relevant field of application.

General requirements concerning the electrical equipment of machines and aiming at the safety of these machines are contained in Part 1 of European standard EN 60204 (corresponds to VDE 0113).

For safe installation of our control system please observe the following notes:

2.5.5. Protection against external electrical influences

Connect the control system to the protective earth conductor to eliminate electromagnetic interference. Ensure practical wiring and laying of cables.

2.5.6. Cable routing and wiring

Separate laying of power supply circuits, never together with control loops:

DC voltage	60 V ... 400 V
AC voltage	25 V ... 400 V

Joint laying of control loops is permissible:

data signals, shielded
analog signals, shielded

digital I/O lines, unshielded
DC voltages < 60 V, unshielded
AC voltages < 25 V, unshielded

2.5.7. Location of installation

Make sure that there are no impediments due to temperatures, dirt, impact, vibrations and electromagnetic interference.

Temperature
Consider heat sources such as general heating of rooms, sunlight, heat accumulation in assembly rooms or control cabinets.

Dirt
Use suitable casings to avoid possible negative influences due to humidity, corrosive gas, liquid or conducting dust.

Impact and vibrations
Consider possible influences caused by motors, compressors, transfer routes, presses, ramming machines and vehicles.

Electromagnetic interference
Consider electromagnetic interference from various sources near the location of installation: motors, switching devices, switching thyristors, radio-controlled devices, welding equipment, arcing, switched-mode power supplies, converters / inverters.

2.5.8. Particular sources of interference

Inductive actuators
Switching off inductances (such as from relais, contactors, solenoids or switching magnets) produces overvoltages. It is necessary to reduce these extra voltages to a minimum.
Reducing elements my be diodes, Z-diodes, varistors or RC elements. To provide suitably designed reducing elements, please pay attention to the technical specifications delivered by the manufacturer or supplier of the corresponding actuators.

3. Hardware

The KUAX 680C has a compact design. The basic configuration is as follows:

User memory

program and data:	112 KByte flash EPROM
data:	64 KByte RAM, buffered by accu

Internal inputs and outputs

16 digital inputs, 24 V DC, 5 ms
2 counter inputs, 24 V DC, counting frequency = 10 kHz
2 interrupt inputs, 24 V DC, 0.3 ms
4 analog inputs, 0...10 V, 10 bit, single ended
16 digital outputs, 24 V DC, 0.5 A
2 analog outputs, 0...10 V, 8 bit

Module slots

4 module slots
These are suitable for modules of the KUAX 680I (as from production date calendar week 27/95).

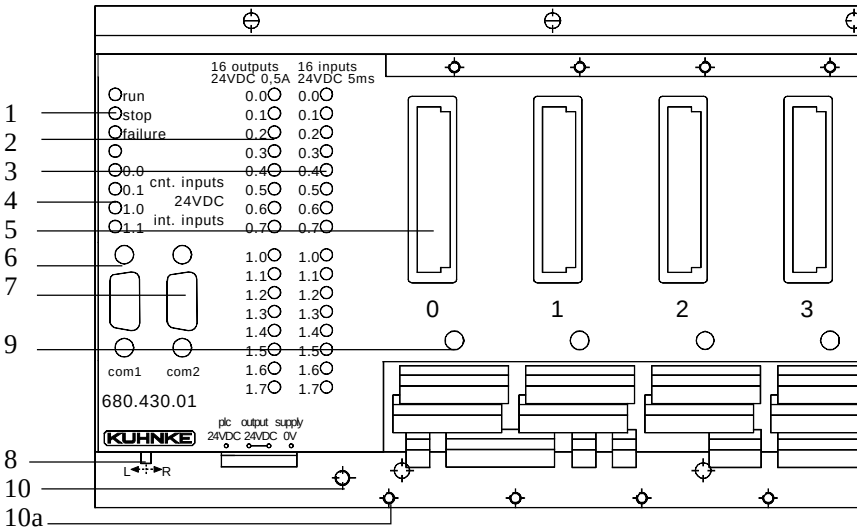
Communication ports

2 V.24 ports: RS 232/1 and RS 232/2
1 RS 485, to be activated by KUBES module

☞ *You cannot use RS 232/2 and RS 485 simultaneously.*

3.1. Layout of LEDs and connectors

3.1.1. Top view



Legend

1 3 system LEDs:
run (green), stop (red), failure (red)

2 16 LEDs "internal outputs" (red)

3 16 LEDs "internal inputs" (green)

4 4 LEDs "special inputs" (green):
SI0.0...0.1 counter inputs
SI1.0...1.1 interrupt inputs

5 4 module slots

6 V.24 port RS 232/1, 9pin Sub-D connector

7 V.24 port RS 232/2, 9pin Sub-D connector

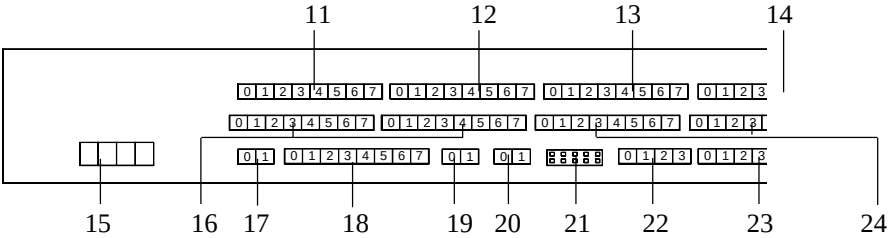
8 "normal program" / "load monitor" switch
pos. "L": normal program
pos. "R": load monitor
Don´t change the switch position in run mode! Otherwise the program run is interrupted.

9 plastic knobs to lock the modules into position

10 ground pin M4 x 15

10a 4 M3-size threaded bores for cable shields

3.1.2. Front view



Legend

11 signals from module 0, 8pin screw-type locking connector
12 signals from module 1, 8pin screw-type locking connector
13 signals from module 2, 8pin screw-type locking connector
14 signals from module 3, 8pin screw-type locking connector
15 power supply, 4pin screw-type locking connector
16 16 digital internal outputs, O0.0...7 and O1.0...07,
two 8pin screw-type locking connectors
17 no function
18 RS 485 interface, 8pin screw-type locking connector
19 2 internal counter inputs: SI0.0...0.1,
2pin screw-type locking connector
20 2 internal interrupt inputs, SI1.0...1.1,
2pin screw-type locking connector
21 test connector, 8pin socket
This connector serves test purposes in the factory only. Never connect to any supply, as this might destroy components!
22 2 internal analog outputs, 0...10 V, AO0.0...0.1,
4pin screw-type locking connector
23 4 internal analog inputs, 0...10 V, AI0.0...0.3,
8pin screw-type locking connector
24 16 internal digital inputs, I00.0...7 and I01.0...07,
two 8pin screw-type locking connectors

3.1.3. Screw-type locking connectors

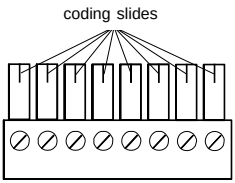
Power supply, inputs, outputs and the RS 485 interface are all connected by means of screw-type locking connectors (COMBICON of the Phoenix company):
Power supply: connector type COMBICON
matrix 5.08 mm, connector 0.2...2.5 mm², max. load 12 A
all other connectors: connector type MINI-COMBICON
matrix 3.81 mm, connector 0.14...1.5 mm² , max. load 8 A

The green screw-type locking connectors fit very tightly so that they are not shaken loose by vibration. If you find it impossible to pull them off with your hands you may use a flat object such as a screwdriver with a wide blade as a lever.

Never pull the leads to unplug locking connectors. You might otherwise accidentally pull them out of the terminals or rip them off.

3.1.3.1. Coding

The MINI-COMBICON screw-type locking connectors can be coded to prevent them from being accidentally plugged into the wrong positions (e.g. digital inputs into the RS 485 interface). To do so simply slot one or several coding profiles into the groove(s) provided for this purpose in the receiving connector part. Cut off the corresponding coding slide on the male connector part, e.g. by means of a side cutter.



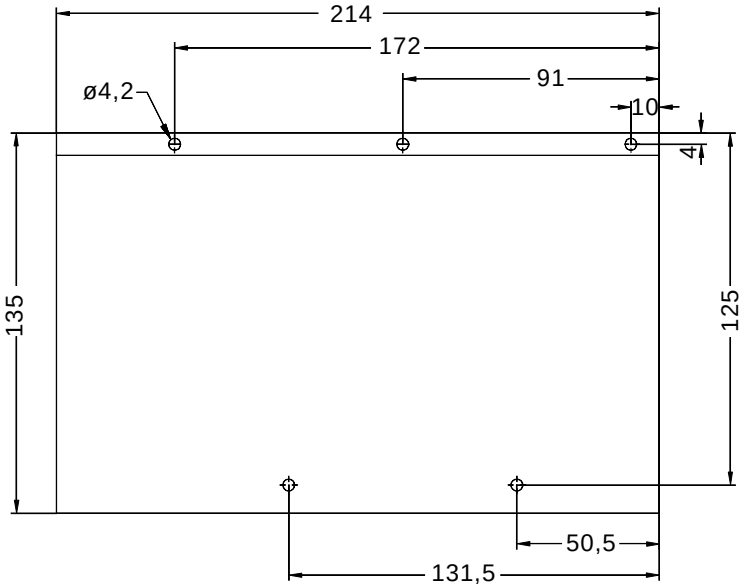
Some of the receiving connector parts are pre-coded in the factory. See the corresponding illustration to learn where there is a coding already and what it looks like.

3.2. Basic device: dimensions and mounting

L = 214 mm
W = 135 mm
H = 51 mm, with modules: 108 mm, when mounted on carrier rail: + 7.5 mm

3.2.1. Wall mounting

Use 5 M4 screws to attach the device. The illustration shows the positions of the drill holes in the base plate of the KUAX 680C.

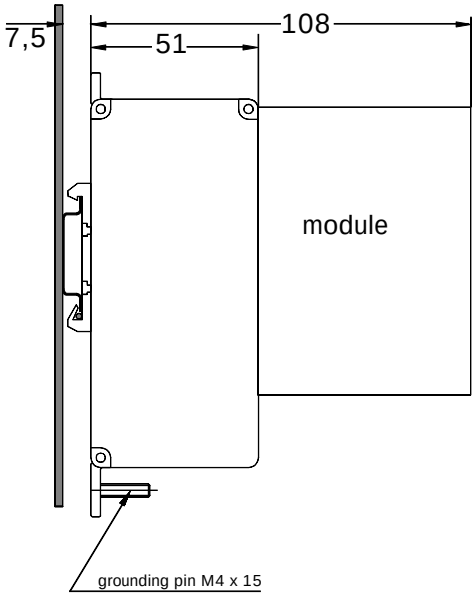


Dimensions and positions of the drill holes are the same as on the KUAX 680I (with 4 module slots).

3.2.2. Carrier rail mounting

The basic device can also be mounted on a carrier rail in accordance with DIN EN 50022 (35 x 7.5 mm).

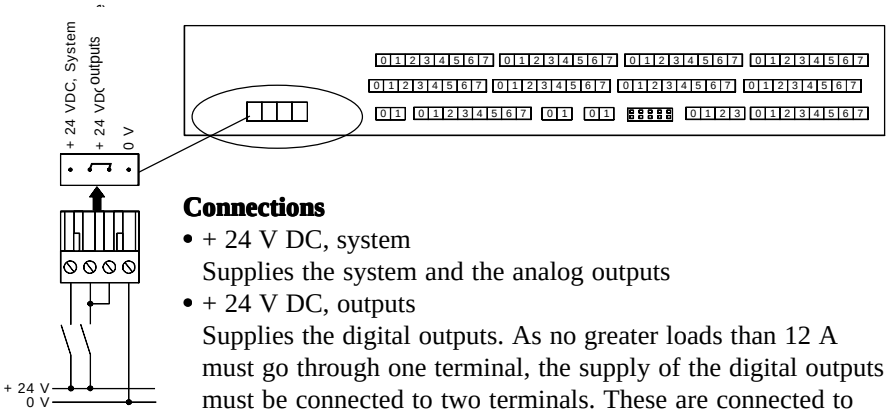
Screw 2 quick screw connectors for carrier rail mounting into the base plate for this purpose (use 3 for devices with 8 slots). These must be ordered separately (see appendix "B.1. Accessories").



Lengthways, the base plate provides 6 T-slots two of which are used as a casing for the nuts for screwing in the quick screw connectors (the illustration shows a fixation in the middle of the device). Remove one side-plate of the device to insert the nuts.

3.3. Power supply

The device is supplied with power via a 4pin screw-type locking connector (matrix 5,08).



Connections

- + 24 V DC, system
  Supplies the system and the analog outputs
- + 24 V DC, outputs
  Supplies the digital outputs. As no greater loads than 12 A must go through one terminal, the supply of the digital outputs must be connected to two terminals. These are connected to each other on the PCB.
- 0 V
  Reference potential for system, outputs and inputs.

If you supply inputs or outputs from other sources than the system, you must make sure to connect the 0 V connections of all power sources (equipotential bonding).
Otherwise the possibility of program execution errors or even the destruction of components by uncontrolled compensation currents cannot be excluded.

supply voltage 24 V DC -20%/+25%

wire diameter fexible leads, 2.5 mm² max.

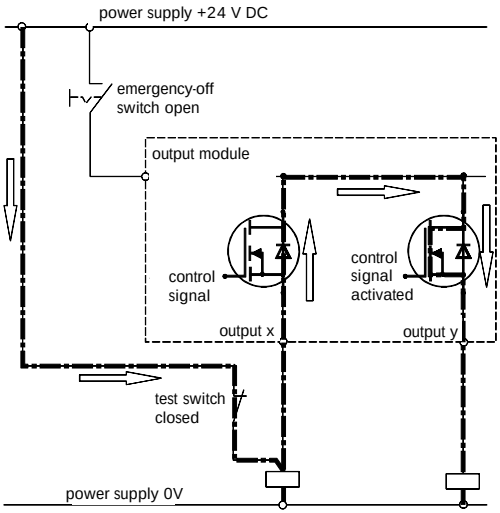
3.3.1. Emergency Off installation for the outputs

Due to the separate power supply of the outputs, these can be deactivated via a shared emergency off installation.
The advantage is that the system is still supplied with power, the inputs read and communication (with terminal, PC...) continued.

However, you must make sure of the following:

There must never be a backward power feed to the outputs while the output power supply is switched off (e.g. by emergency off installation)!
This is valid in cases where the system is still supplied with power.

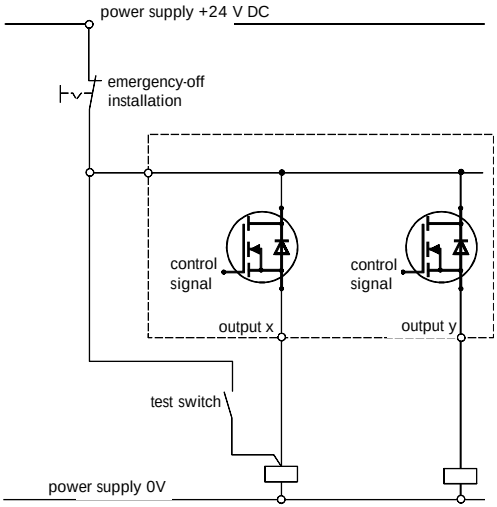
Wrong:



Outputs set by the program can be supplied via the protective diode of a back-fed output (in this case: via the test switch), thus rendering the emergency off function for these outputs ineffective. Also, the protective diode of the feeding output may be destroyed if exposed to high loads.

Correct:

To avoid this situation, also connect the supply of supplementary key switches or other switching elements (in this case: the test switch) to the emergency off installation if the switches are connected in parallel to the outputs:



3.3.2. Grounding

On the base plate of the device (see "3.1.1. Top view", pos. 10a) there is a threaded bolt of dimensions M4 x 15 mm with two nuts for connecting the grounding wire with the frame ground.

wire diameter	recommended:	4 mm^2
	minimum:	1.5 mm^2

Capacitive ground connection of the supply voltages
For reasons of operational safety, the 0V potential of the KUAX 680C has a capacitive connection to the frame ground. High-frequency interferences can bleed off via this connection.

Shielding of data lines
The connector casings of interfaces RS 232/1 and RS 232/2 and the "Shield" connectors of the RS 485 interface are directly connectrd to the frame ground. We recommend using these positions for connecting the shields of the data lines. Possible interference signals can thus bleed off without causing any damage.

☞ *Connect the basic device to ground via the grounding pin to render these measures effective. Always choose the shortest possible connection.*

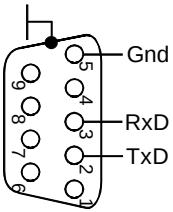
3.4. Interfaces for serial communication

The control device is equipped with 3 serial interfaces, two of which can be used at the same time (RS 232/1 and RS 232/2 or RS 232/1 and RS 485).

3.4.1. RS 232 (V.24)

The KUAX 680C has two V.24 interface connectors which can be accessed from above. Use these interfaces to program the device and to establish the connection for communication with PCs, user terminals or other machines.
The default configuration of these interfaces (i.e. the configuration without any influence by KUBES modules) is as follows:

protocol	KUBES protocol *)
tranfer rate	9600 baud
data bits	8
parity check	o (odd)
stop bits	1
type of connection	point-to-point
connector	9pin female Sub-D connector



The connector casing has a conductive connection to the frame ground (ground connection!). Connect the shield of the connecting cable to the connector casing.

**) To enable communication with devices which run other than the KUBES protocol, the interfaces can be set to a separate protocol by means of KUBES modules (see instruction manual E 386 GB "KUBES Modules").*

3.4.1.1. Interface RS 232/1

V.24 interface (RS 232) no. 1. The connector is on top of the device (see "3.1.1. Top view", pos. 6).
This interface uses a processor port of its own. It is therefore always available.

3.4.1.2. Interface RS 232/2

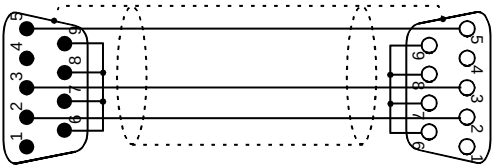
V.24 interface (RS 232) no. 2. The connector is on top of the device (see "3.1.1. Top view", pos. 7).
This interface shares one processor port with the RS 485 (see next page). You can use a KUBES module to disable this interface and to activate the RS 485.

3.4.1.3. Programming

Programs for the KUAX 680C are written using KUBES, the Kuhnke user software (as from version 4.12). KUBES runs on PCs under the Windows user interface (version 3.1 or better). As long as the interfaces are configured by default (see previous page), you can use one of the two above-mentioned V.24 interfaces for programming.

Programming cable
For the connection between KUAX 680C and PC we recommend using a prefabricated and ready-to-use programming cable which can also be used for all other controllers that can run KUBES programs:

KUAX 680C IBM AT

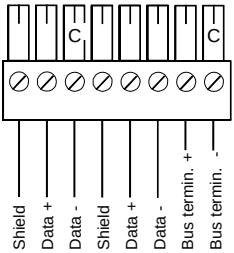


Order no. 657.151.03

3.4.2. Interface RS 485

Serial interface, used for networking with other devices. The connector is located at the front of the device (see "3.1.2. Front view", pos. 8).
Use a KUBES module (please ask us) to activate this interface. Activating the RS 485 deactivates interface RS 232/2.

protocol	selectable via KUBES module
type of connection	point-to-point or bus
connector	8pin screw-type locking, matrix = 3.81



C = remove coding element

<u>Connectors</u> (from left to right):

- Shield	cabel shield, incoming
- Data +	data line +, incoming
- Data -	data line -, incoming
- Shield	cable shield, outgoing
- Data +	data line +, outgoing
- Data -	data line -, outgoing
- Bus termin. +	bus termination resistor, connect to "Data +" if terminating station
- Bus termin. -	bus termination resistor, connect to "Data -" if terminating station

Connectors with the same functional designation (Shield, Data +, Data -) have a conducting connection inside the device. Shield is also directly connected to the chassis ground (grounding connection !). In bus connections, the shielding should be applied to both ends of the cable.

3.5. User memory

The KUAX 680C is equipped with a Flash-EPROM and RAM. Some of these memory resources are available for the user:

Program memory
Programs are mainly stored in the Flash-EPROM. Here the program is saved and stored safely without the use of electrical energy. The program is divided into modules (see ch. "4.7 Module programming"). A table of modules (module allocation table) contains references to the position of individual modules in the memory. When you modify the program of a module, the module will be saved as the last module in the Flash-EPROM. The reference in the table will be adapted correspondingly. The original program of the module will thus become invalid ("dead module"). Like this, some memory space is lost with every editing process. The free memory is indicated in the KUBES Main Status bar. Transmitting the entire changed program again reorganizes the memory and "dead modules" will be removed. This increases the free memory space.

The module allocation table is updated, i.e. overwritten, every time a module is changed. However, it is not possible to overwrite the contents of Flash-EPROMs which is why the module allocation table is stored in a buffered RAM.

When you have completed your program, make sure to transmit the module allocation table into the Flash-EPROM with the last program version. After longer periods of inactivity, the table might otherwise get lost due to discharging of the accumulator.

Use the "Copy to 644 Flash" command of the "EPROM" menu for transmission. The controller must be in Stop and Reset mode before you can use this command:

Data memory

Some of the Flash-EPROM space can also be used as data memory. It is then no longer available as program memory. Divide the memory into a program range and a data range for this purpose (see KUBES, "Set memory size" command of the "Main" menu).

The data memory of the Flash-EPROM can only be written into once: after "Delete program" or after "Transmit program" (KUBES commands). It is thus impossible to change the data while the controller is running.

Another possibility is to store data in the RAM. The RAM is buffered by a built-in accumulator. For times at which the device is not supplied with power, data security can therefore only be guaranteed for a limited amount of time (see appendix "A. Technical specifications"). We therefore recommend not to use the RAM as program memory.

Banks

The user memory occupies 3 banks:

Bank 0	48 KByte	Flash-EPROM
Bank 1	64 KByte	Flash-EPROM
Bank 2	64 KByte	RAM

3.6. System messages

The operating status of the KUAX 680C is indicated by three light emitting diodes (LEDs) which are located on the left side of the device (see chapter "3.1.1. Top view").

LED	Colour	Function
run	green	normal program operation
stop	red	program is stopped
failure	red	failure

The "failure" LED flashes in various rhythms to indicate different types of failures. Please refer to appendix "D. Reactions to failures" for explanations of the significance and the context of these messages.

3.7. Internal inputs and outputs

Inputs and outputs are used to lead signals from the machine or plant into the controller (inputs) or vice versa from the controller into the machine or plant (outputs).

These include:

digital input signals from
- switches
- key-switches
- sensors
- incremental actuators
- etc.

digital output signals to switch
- relays
- magnets
- solenoids
- etc.

analog signals such as
- temperature values
- liquid level values
- speeds
- etc.

In the chapter below you will find descriptions of the inputs and outputs that the basic device is equipped with when it comes to you.

Process image
The KUAX 680C has a process image for the digital inputs and outputs. The processor works with this process image when the program instructs it to read inputs or to write to outputs.
The status of inputs is requested between two subsequent program cycles and then entered as information into the process image.
Outputs are switched on or off between two subsequent program cycles, depending on the status found in the process image.

3.7.1. Digital inputs, 5 ms

These inputs are designed for registering digital signals from various sources. When working with proximity switches and semiconductor sensors in particular, you must make sure that they operate within the range of switching threshold values indicated below.

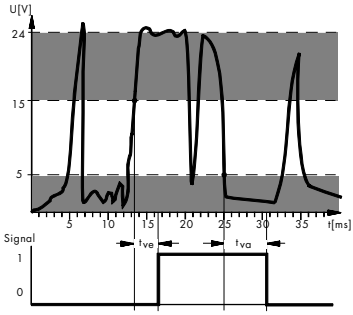
The input switching connections serve adapting the external signals to the system voltage.

Defined signals and switching thresholds

logical 0	≤ 5 V
logical 1	≥ 15 V
(hysteresis	1...4 V)

Signal delay

Voltage surges (noise impulses) are filtered out to avoid them being accepted as valid signals that might cause unintended switching processes. This delays signal recognition by 5 ms nominally:

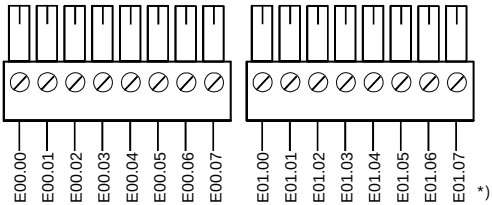


raising delay: t_{ve} = 3.0 ... 7.0 ms
falling delay: t_{va} = 4.0 ... 7.0 ms

Input signals are read between program cycles and then written into the process image. You must therefore add the program cycle time to the delay time to determine the mean signal availability for the user program.

Signal line connection

The input signal lines are connected to the front of the device via two 8pin screw-type locking connectors. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 24, to find the exact location of the connectors. Connect group 0 to the left connector and group 1 to the right one.



*) read "E" = "I"

Technical specifications

number of inupts 16
type (to IEC 1131) 1
potential separation ..... no
line interfacing ..... 8pin screw-type locking connector, 3.81 matrix
indicators ..... LEDs
  location ..... on top of the device (see "3.1.1. Top view", pos. 3)
  colour ..... green
  tapping point ..... in the input circuit
  signal state ..... 1: LED on
  0: LED off
addressing
  group 0 ..... I00.00...I00.07
  group 1 ..... I01.00...I01.07
input voltage ..... 24 V DC -20%/+25% (incl. residual ripple)
surge immunity ..... ≤ 60 V DC (≤ 30 min.)
signal recognition
  logical 0 ..... ≤ 5 V DC
  logical 1 ..... ≥ 15 V DC
power consumption/input ..... max. 10 mA

3.7.2. Counter inputs, 10 µs

The KUAX 680C has two fast counters. Each of these is assigned an input for recognising the counting signals. These inputs have a particularly short signal delay. They can also be read in the user program (via the process image) and treated like normal inputs.

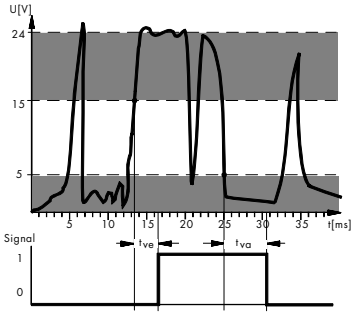
The input switching connections serve adapting the external signals to the system voltage.

Defined signals and switching thresholds

logical 0	≤ 5 V
logical 1	≥ 15 V
(hysteresis	1...4 V)

Signal delay

Voltage surges (noise impulses) are filtered out to avoid them being accepted as valid signals that might cause unintended switching processes. This delays signal recognition:

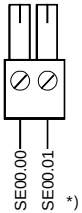


raising delay: t_{ve} = 0.003 ... 0.016 ms
falling delay: t_{va} = 0.007 ... 0.017 ms

Due to the very short signal delay, signal noise may not be filtered out sufficiently. Signal noise must therefore not be allowed to occur in the first place. Please take this into account when laying the cables. We urgently recommend using shielded cables. Connect the cable shield to the device (see chapter "3.1.1. Top view", pos. 10a).

Signal line connection

The input signal lines are connected to the front of the device via a 2pin screw-type locking connector. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 19, to find the exact location of the connectors:



*) read "SE" = "SI"

Technical specifications

number of inputs	2
function	counter inputs
type (to IEC 1131)	1
potential separation	no
line interfacing	2pin screw-type locking connector, 3.81 matrix
indicators	LEDs
location	on top of the device (see "3.1.1. Top view", pos. 4)
colour	green
tapping point	in the input circuit
signal state	1: LED on 0: LED off
frequency	max. 10 kHz
addressing	SI00.00...SI00.01
input voltage	24 V DC -20%/+25% (incl. residual ripple)
surge immunity	≤ 60 V DC (≤ 30 min.)
signal recognition	
logical 0	≤ 5 V DC
logical 1	≥ 15 V DC
power consumption/input	max. 10 mA

Counting function

Inputs

The counters work as event counters. The inputs are permanently allocated to the counters:

SI00.00	counter #1
SI00.01	counter #2

Transfer buffer memory

A memory area is used as transfer buffer for communication between user program and counters:

SLI00.00...00.15	counter #1
SLI01.00...01.15	counter #2

The operands of the transfer buffer memory have the following significance:

SLI0x.00	actual lowbyte value
SLI0x.01	actual highbyte value
SLI0x.04	preset lowbyte value
SLI0x.05	preset highbyte value
SLI0x.08	count: 0 = Stop, <>0 = Run
SLI0x.09	counting direction: 0 = down, <>0 = up
SLI0x.10	counting mode: 0 = positive edges, <>0 = positive and negative edges
SLI0x.11	<>0 = clear count
SLI0x.12	<>0 = accept preset value

x = **0** *for counter #1*
x = **1** *for counter #2*

3.7.3. Interrupt inputs, 300 µs

The KUAX 680C has two interrupt inputs for very fast recognition of external events. These inputs have a particularly short signal delay. They can be read like normal inputs in the user program (operands SI01.00...01.01) and trigger a processor interrupt.

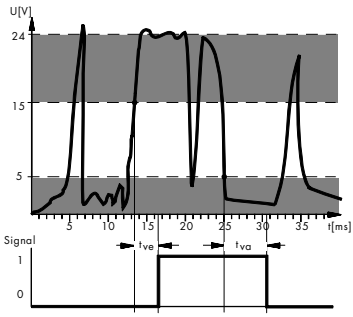
The input switching connections serve adapting the external signals to the system voltage.

Defined signals and switching thresholds

logical 0	≤ 5 V
logical 1	≥ 15 V
(hysteresis	1...4 V)

Signal delay

Voltage surges (noise impulses) are filtered out to avoid them being accepted as valid signals that might cause unintended switching processes. This delays signal recognition:

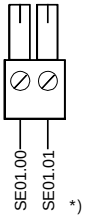


raising delay: t_{ve} = 0.05 ... 0.23 ms
falling delay: t_{va} = 0.10 ... 0.39 ms

Due to the very short signal delay, signal noise may not be filtered out sufficiently. Signal noise must therefore not be allowed to occur in the first place. Please take this into account when laying the cables. We urgently recommend using shielded cables. Connect the cable shield to the device (see chapter "3.1.1. Top view", pos. 10a).

Signal line connection

The input signal lines are connected to the front of the device via a 2pin screw-type locking connector. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 20, to find the exact location of the connectors:



*) read "SE" = "SI"

Technical specifications

number of inputs	2
function	counter inputs
type (to IEC 1131)	1
potential separation	no
line interfacing	2pin screw-type locking connector, 3.81 matrix
indicators	LEDs
location	on top of the device (see "3.1.1. Top view", pos. 4)
colour	green
tapping point	in the input circuit
signal state	1: LED on 0: LED off
frequency	max. 10 kHz
addressing	SI01.00...SI01.01
input voltage	24 V DC -20%/+25% (incl. residual ripple)
surge immunity	≤ 60 V DC (≤ 30 min.)
signal recognition	
logical 0	≤ 5 V DC
logical 1	≥ 15 V DC
power consumption/input	max. 10 mA

Interrupt function

Interrupt module no. 10
If one of the two inputs triggers an interrupt, this event immediately calls up interrupt module no. 10. Use this module for the program that defines the reaction to an interrupt event.

Transfer buffer memory SLJ00.00...01.15
The purpose of transfer buffer memories is to determine whether an input is to trigger an interrupt and which input if so. After the occurrence of an interrupt, the transfer buffer memory will contain the interrupt source.

The operands of the transfer buffer memory have the following significance:

User program reads:

SLJ00.00 ....255:	interrupt triggered by SI01.00
SLJ00.01 ....255:	interrupt triggered by SI01.01

User program writes:

SLJ01.00 ....255:	enable interrupt by ⎍ SI01.00
.....0:	disable
SLJ01.01 ....255:	enable interrupt by ⎍ SI01.01
.....0:	disable
SLJ01.04 ....255:	enable interrupt by ⎏ SI01.00
.....0:	disable
SLJ01.05 ....255:	enable interrupt by ⎏ SI01.01
.....0:	disable
SLJ01.08 ....255:	enable interrupt function
.....0:	disable
SLJ01.14 ....255:	transfer settings as above

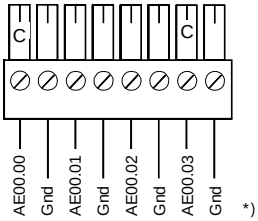
⎍ = positive edge of the input signal
⎏ = negative edge of the input signal

3.7.4. Analog inputs, 0…10 V, 10 bit, single-ended

In its basic configuration, the KUAX 680C is equipped with four analog inputs. Further analog inputs can be added as plug-in modules.

Signal line connection

The input signal lines are connected to the front of the device via an 8pin screw-type locking connector. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 23, to find the exact location of the connectors.



C = remove coding element

*) read "AE" = "AI"

It is obligatory to connect both lines – signal (AIxx.xx) and the Gnd line to the right of it – for each channel. The Gnd connectors are not directly connected to the device ground.

Shielding

Use shielded wires to connect the analog signal lines. Connect the shielding to the aluminium base profile of the controller using M3 screws (see chapter "3.1.1. Top view", pos. 10a).

Representation of the analog value

The read analog value is digitalised and the digital value written into a 16bit address as two's complements. In this address, the value is contained in bits 5...14. Bits 0...4 and 15 (sign bit) always have logical status 0:

Address bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status:	0	----------read value ----------										0	0	0	0	0

In the user program, the value is read in a double byte operation.

Example:

```
LD      AI00.00
CMPD>=  4V          ;input range 0...10.00V
=       M00.01
```

Settings

Analog conversion must be enabled by the user program if and when required. Enabling will be automatic if the device is configured with an additional analog input module.
You can also preset the conversion time. It is also valid for the analog input modules. Transfer buffer range SLK00.00...00.02 is used for setting. The operands of this range have the following significance:

SLK00.00	255 =	enable analog conversion
.....	0 =	disable analog conversion
SLK00.01	255 =	transfer settings
SLK00.02	255 =	conversion time 1 ms
.....	0 =	conversion time 10 ms

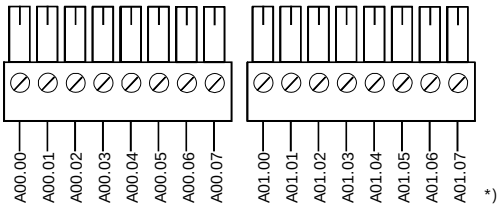
Technical specifications

number of inputs (channels) 4
potential separation ..... no
addressing ..... AI00.00...AI00.03
measuring range ..... 0...10 V
resolution ..... 10 bit, ~ 0.01 V / digit
conversion time ..... 10 ms or 1 ms
input voltage protection ..... 60 V
protection against noise impulses ..... by filters and buffers

3.7.5. Digital outputs, 500 mA

Digital outputs provide the connection to the external actuators (relays, contactors, solenoids, valves...).
Resistive or inductive loads can be applied. Free-wheeling diodes have been added to suppress inductive switch-off surges.
The switching state of the outputs is indicated by LEDs.

Signal line connection
The output signal lines are connected to the front of the device via two 8pin screw-type locking connectors. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 16, to find the exact location of the connectors. Connect group 0 to the left connector and group 1 to the right one:



*) read "A" = "O"

Reverse polarity protection
A diode has been installed to avoid a possible reversing of the polarity of the output supply voltage destroying any circuits.

Increased performance by parallel connection
There is a maximum load that can be applied to individual outputs (see "Technical specifications" below). However, it is permissible, to connect 2 outputs in parallel. This doubles the output performance.

You must only connect outputs in parallel which are within the same group of eight because the processor controls the outputs by byte (simultaneity).

Protection against short circuit and overload

The following means have been implemented to protect the outputs against destruction caused by overload or short circuit:

- the load current is limited to approx. 1.0...1.2 A
- a temperature monitoring system switches the output off after 0.1 to 1 s and notifies the CPU of the short circuit
- the CPU outputs a short circuit message,
- reports the short circuit by a flashing rhythm (1) of the "failure" LED,
- activates interrupt module 18
- see also appendix D.1.

Restart

- find the failure source
- make the device voltage free
- remove the failure
- switch supply voltage back on

Technical specifications

outputs	16
type	semiconductor
indicators	LEDs
location	on top of device (see chapter "3.1.1. Top view", pos. 2)
colour	red
tapping point	in the load current circuit
signal state	1: LED on 0: LED off
addressing	
group 0	O00.00...O00.07
group 1	O01.00...O01.07
output voltage:	24 V DC -20%/+25% (incl. residual ripple)
outputs current	max. 0.5 A
short circuit protection	yes

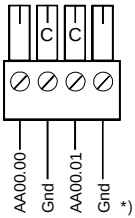
3.7.6. Analog outputs, 0…10 V, 8 bit

In its basic configuration, the KUAX 680C is equipped with two analog outputs. Further outputs can be added as plug-in modules.

The internal analog outputs described in this chapter are generated by the processor via the PWM outputs. However, these are also required for controlling stepping motors.

If you are working with a 2 channel stepping motor, none of the two analog outputs is available.
If you are working with a 1 channel stepping motor, only analog output O00.01 is available.

Signal line connection
The output signal lines are connected to the front of the device via a 4pin screw-type locking connector. Please refer to the illustration given in chapter "3.1.2. Front view", pos. 22, to find the exact location of the connectors:



C = remove coding elements

*) read "AA" = "AO"

It is obligatory to connect both lines – signal (AIxx.xx) and the Gnd line to the right of it – for each channel. The Gnd connectors are not directly connected to the device ground.

Shielding
Use shielded wires to connect the analog signal lines. Connect the shielding to the aluminium base profile of the controller using M3 screws (see chapter "3.1.1. Top view", pos. 10a).

Representation of the analog value

The user program must write the analog value to be output into a 16bit address in two's complements. In this addres, the value is contained in bits 7...14, the sign bit (bit 15) is 0. Bits 0...6 are not analysed:

Address bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status:	x	-- value to be output --								x	x	x	x	x	x	x

In the user program, the value is written with a double byte operation.

Example:

```
LD   5.5V      ;input range 0...±10.00V
=D   AO00.00
```

Technical specifications

number of outputs ..... 2, short circuit proof
potential separation ..... no
addressing ..... AO00.00...AO00.01
range ..... 0...+10 V
Value-changing time ..... 100 ms
max. output current ..... 5 mA
resolution ..... 8 bit, ~ 0.039 V / digit
sign ..... no

3.8. Module slots

The range of inputs and outputs can be extended by adding up to four modules. Use the modules – made as from calendar week 27/95 – that were developed for the KUAX 680I.

Modules produced before calendar week 27/95 do not fit into the slots of the KUAX 680C. They have no drilled hole at the bottom for locking the module into place (see chapter "3.2.2. Top view", pos. 9).

The four plug-in connectors for the modules are located on top of the device (see chapter "3.1.1. Top view", pos. 5). The module slots are numbered from left (slot #0) to right (slot #3).

The modules are described in a separate instruction manual:

Instruction manual Modules of KUAX 680I and 680C E 326 GB

3.8.1. Differences to the KUAX 680I

The instruction manual of the modules (E 326 GB) describes the use of the modules in the KUAX 680I.
When using the modules in the KUAX 680C, please observe the differences described below which are due to the fact that, in its basic configuration, the KUAX 680C has some I/Os.

Limited use

You cannot use the event counter module, order no. 680.454.03. Reason: the basic configuration of the device includes two event counters (internal inputs) already.
The stepping motor modules and the internal analog outputs share the same system resources, i.e. the PWM outputs of the processor:

PWM	analog output	stepping motor module
1	AO00.00	680.444.01 und .02
2	AO00.01	680.444.02

Thus, if you are working with a two-channel stepping motor module (680.440.02) none of the two internal analog outputs is available. If you are working with a one-channel stepping motor module (680.440.01) you can still use internal analog output AO00.01.

Addressing

Please take into account that some input and output groups are occupied by the internal inputs and outputs already. As in the KUAX 680I, the modules plugged into the device are numbered by groups from left to right. They start with different group numbers, however.

Modules	First group
digital inputs	I02
analog inputs	AI01
digital outputs	O02
analog outputs	AO01

3.8.2. Input / output configuration

Internal	Module slots			
	0	1	2	
Permanently equipped with:	Allowed configuration of all slots: 8/16 digital I/Os., 4 analog (interface or 2 multi-function counter. Illegal configuration: event counter module.			
2 serial interfaces, 16 digital inputs, 2 counter inputs, 2 interrupt inputs, 16 digital outputs, 4 analog inputs, 2 analog outputs *1		Additional: 2 x stepping motor (PWM) *1	Additional: 2 x stepping motor (PWM) or 2 x analog input *2	Add 4 x a *2

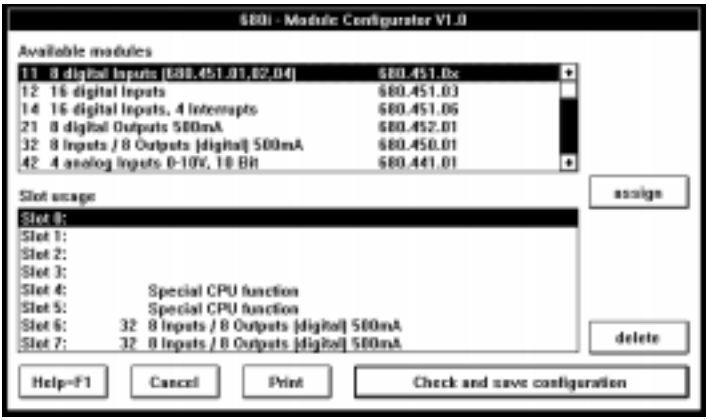
Enter the inputs and outputs that the device is equipped with into the KUBES Module Configurator when writing your project.

please continue overleaf

*1 The internal analog outputs and the stepping motor modules share the same PWM outputs of the processor. Working with a stepping motor module excludes the use of one or even both internal analog outputs (see chapter "3.8.1. Limitations...").
*2 These are the analog inputs (10 bit) that use the A/D converter of the processor.

3.8.2.1. The KUBES Module Configurator

Enter the inputs and outputs that the device is equipped with into the Module Configurator as required when writing your project under KUBES (see KUBES, Main menu, command "Configuration 680" of the "Edit" menu). The Configurator shows 8 slots. In the KUAX 680C, slots 4 to 7 are reserved for the internal configuration. The entries for slots 4 and 5 cannot be changed. The entries for slots 6 and 7 stand for the internal inputs and outputs. Enter I/O modules of the same configuration. You can change these entries so that you can also work with devices providing a different basic configuration (e.g. 16 inputs, 8 outputs).



The information entered for slots 6 and 7 must correspond to the number of internal digital inputs and outputs. Ohterwise, these cannot be addressed via the program.

4. Software

4.1. Working principle

The micro processor for the user program receives its program from two different program memories:

the

- Monitor program memory

and the

- User program memory

The monitor program contains all system features of the controller KUAX 680C. It is part of the basic configuration of the device when delivered.

The user program memory contains all programs for controlling the machine or plant. The programs are written under the KUBES programming software.
Another feature is so-called C-tasks which can be included in the user program. C-tasks are programs written in the C programming language. They contain solutions for complex control tasks (regulation, positioning etc.).

The following chapters will provide you with the information necessary to create user programs for the KUAX 680C.

The way in which programs are written is not described in this manual. For this please refer to the:

☞

Beginner's Guide KUBES 4 E 327 GB

4.2. Operands overview

Group	Input	Function	Type	Number	Input range from	Input range to	Comment
I	I_._	Internal inputs	Bit	16	I00.00	I01.07	With process chart (basic configuration of the device includes internal I/Os)
I	I_._	Module inputs		max.64	I02.00	I09.07	
SI	SI_._	Counter inputs		2	SI00.00	SI00.01	
SI	SI_._	Interrupt inputs		2	SI01.00	SI01.01	
O	O_._	Internal outputs		16	O00.00	O01.07	
O	O_._	Module outputs		max.64	O02.00	O09.07	
M SM LM FM SO	M__.__ SM__.__ LM__.__ FM__.__ SO__.__	Markers		256 256 256 256 256	M00.00 SM00.00 LM00.00 FM00.00 SO00.00	M15.15 SM15.15 LM15.15 FM15.15 SO15.15	
R SR	R__.__ SR__.__	Remanent markers		256 256	R00.00 SR00.00	R15.15 SR15.15	Buffered by accu in the device
BM SBM BO	BM__.__ SBM__.__ BO__.__	Byte markers		256 256 256	BM00.00 SBM00.00 BO00.00	BM15.15 SBM15.15 BO15.15	
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7	DB0__.__ DB1__.__ DB2__.__ DB3__.__ DB4__.__ DB5__.__ DB6__.__ DB7__.__	Byte markers to be used as "data processing ranges" for data modules	Byte	256 256 256 256 256 256 256 256	DB000.00 DB100.00 DB200.00 DB300.00 DB400.00 DB500.00 DB600.00 DB700.00	DB015.15 DB115.15 DB215.15 DB315.15 DB415.15 DB515.15 DB615.15 DB715.15	You can use these operands like normal byte markers
BR SBR ABM BC SBC BD SBD LBM FBM	BR__.__ SBR__.__ ABM__._ BC__.__ SBC__.__ BD__.__ SBD__.__ LBM__.__ FBM__.__	Remanent byte markers		256 256 256 256 256 256 256 256 256	BR00.00 SBR00.00 ABM00.00 BC00.00 SBC00.00 BD00.00 SBD00.00 LBM00.00 FBM00.00	BR15.15 SBR15.15 ABM15.15 BC15.15 SBC15.15 BD15.15 SBD15.15 LBM15.15 FBM15.15	Buffered by accu in the device (accu function test: write a bit map once into a byte marker and check at start of program)
C	C__.__	Progr. counters	Word	32	C00.00	C01.15	
PT	PT__.__	Progr. timers		128	PT00.00	PT07.15	
PC	PC__.__	Progr. clock	Byte	4	PC00.00	PC00.03	
PP	PP__.__	Progr. pulse	Bit	128	PP00.00	PP07.15	
PL	PL00.00 PL00.01	logical 0 logical 1		1 1	PL00.00 PL00.01		
AI	AI_._	Internal analog inputs	Word	4	AI00.00	AI00.04	
AI	AI_._	Analog module inputs		6	AI01.00	AI02.03	
AO	AO_._	Internal analog outputs		2	AO00.00	AO00.01	
AO	AO_._	Analog module outputs		16	AO01.00	AO04.03	
ERR	ERR00.00	System error	Byte	1	Notification of system errors (see appendix "D. Reactions to errors")		

4.2.1.1. Short description of the operands

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

Digital inputs and outputs

In its basic configuration the KUAX 680C is already equipped with inputs and outputs. This I/O range can be extended by modules which you can plug into slots 0...3 if and when required. You can define the configuration yourself by choosing the modules you need.

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.

Bit markers and byte markers

There are 1536 bit markers in 6 groups and 5120 byte markers in 20 groups available on the CPU (see chapter "4.2. Operands overview") for marking (storing) current data.

Of these, 512 bit markers and 2304 byte markers are remanent if the CPU is accu-buffered.

Timers

As a standard, the KUAX 680I has128 software timers available (PT00.00-PT07.15). 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.

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.

Analog inputs and outputs

In its basic configuration the KUAX 680C is already equipped with analog inputs and outputs. This I/O range can be extended by modules which you can plug into slots 0...3 if and when required. You can define the configuration yourself by choosing the modules you need.

Inputs "read" the analog values of temperatures, liquid levels, speeds etc. The analog-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-analog converter is on the module itself. The analog signal is tapped off the corresponding terminals.

System error marker "ERR00.00"

Recognized system errors are written into the byte operand (8 bit) "ERR00.00" by the monitor program. They can be read by the user program and then analysed correspondingly (see also appendix "D. Reactions to failures").

4.3. Commands overview

The following overview contains information about all available commands including the possible types of addressing, the necessary memory capacity and the processing time.

☞ *Please take special care of only linking operands of the same size (bit, byte or word). Mixed operations must be avoided as they may lead to wrong results.*

4.3.1. Logical operations commands

Logical operations commands are commands which serve the logical operation between operands including the assignment of results.

They can be executed with bit, byte and word operands.

☞ *In the following tables, the column "Time" and, in some tables, columns "Byte", "C" and "Z" have not been filled in. This is due to the fact that the relevant information was not yet available. It will be provided in the next edition of this manual. We are sorry for the inconvenience.*

Load commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
L	I00.00	4	0,63	load 1bit address	--	yes
	BM00.00	4	0,63	load 8bit address	--	yes
	100	4	0,5	load 8bit constant	--	yes
	I00.00[10]	8	1,13	load 1bit address with constant offset	--	yes
	I00.00[BM01.00]	14	4,32	load 1bit address with variable offset	--	yes
	BM00.00[10]	8	1,13	load 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	14	4,38	load 8bit slave address with variable offset	--	yes
LN	I00.00	6	0,83	load negation 1bit address	--	yes
	BM00.00	6	0,83	load negation 8bit address	--	yes
	I00.00[10]	10	1,38	load neg. 1bit address with constant offset	--	yes
	I00.00[BM01.00]	16	4,63	load neg. 1bit address with variable offset	--	yes
	BM00.00[10]	10	1,38	load neg. 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	16	4,63	load neg. 8bit slave addr. with variable offset	--	yes
LD	BM00.00	4	0,75	load 16bit address (even address)	--	yes
	BM00.01	10	3,75	load 16bit address (odd address)	--	yes
	10000	4	0,5	load 16bit constant	--	yes
	BM00.00[10]	16	4,25	load 16bit address with constant offset	--	yes
	BM00.00[BM01.00]	22	7,75	load 16bit slave address with variable offset	--	yes

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

AND commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
A	I00.00	4	0,63	AND 1bit address	--	yes
	BM00.00	4	0,63	AND 8bit address	--	yes
	100	4	0,5	AND 8bit constant	--	yes
	I00.00[10]	10	1,38	AND 1bit address with constant offset	--	yes
	I00.00[BM01.00]	16	4,63	AND 1bit address with variable offset	--	yes
	BM00.00[10]	10	1,38	AND 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	16	4,63	AND 8bit slave address with variable offset	--	yes
AN	I00.00	8	1,13	AND negation 1bit address	--	yes
	BM00.00	8	1,13	AND negation 8bit address	--	yes
	I00.00[10]	12	1,63	AND neg. 1bit address with constant offset	--	yes
	I00.00[BM01.00]	18	4,83	AND neg. 1bit address with variable offset	--	yes
	BM00.00[10]	12	1,63	AND neg. 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	18	4,83	AND neg. 8bit slave addr. with variable offset	--	yes
	BM00.00[SLA01.00]	18	4,83	AND neg. 8bit slave addr. with variable offset	--	yes
AD	BM00.00	6	1,0	AND 16bit address (even address)	--	yes
	BM00.01	10	3,75	AND 16bit address (odd address)	--	yes
	10000	4	0,5	AND 16bit constant	--	yes

*) influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

OR commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
O	I00.00	4	0,63	OR 1bit address	--	yes
	BM00.00	4	0,63	OR 8bit address	--	yes
	100	4	0,5	OR 8bit constant	--	yes
	I00.00[10]	10	1,38	OR 1bit address with constant offset	--	yes
	I00.00[BM01.00]	16	4,63	OR 1bit address with variable offset	--	yes
	BM00.00[10]	10	1,38	OR 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	16	4,63	OR 8bit slave address with varibale offset	--	yes
ON	I00.00	8	1,13	OR negation 1bit address	--	yes
	BM00.00	8	1,13	OR negation 8bit address	--	yes
	I00.00[10]	12	1,63	OR negation 1bit address with constant offset	--	yes
	I00.00[BM01.00]	18	4,83	OR negation 1bit address with variable offset	--	yes
	BM00.00[10]	12	1,63	OR negation 8bit address with constant offset	--	yes
	BM00.00[BM01.00]	18	4,83	OR neg. 8bit slave addr. with variable offset	--	yes
OD	BM00.00	6	1,0	OR 16bit address (even address)	--	yes
	BM00.01	10	3,75	OR 16bit address (odd address)	--	yes
	10000	4	0,5	OR 16bit constant	--	yes

*) Influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

EXCLUSIVE-OR commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
XO	I00.00	4	0,63	Exclusive-OR 1bit address	--	yes
	BM00.00	4	0,63	Exclusive-OR 8bit address	--	yes
	100	4	0,5	Exclusive-OR 8bit constant	--	yes
	I00.00[10]	10	1,38	Exclusive-OR 1bit addr. with constant offset	--	yes
	I00.00[BM01.00]	16	4,63	Exclusive-OR 1bit addr. with variable offset	--	yes
	BM00.00[10]	10	1,38	Exclusive-OR 8bit addr. with constant offset	--	yes
	BM00.00[BM01.00]	16	4,63	Excl.-OR 8bit slave addr. with variable offset	--	yes
XON	I00.00	8	1,13	Exclusive-OR negation 1bit address	--	yes
	BM00.00	8	1,13	Exclusive-OR negation 8bit address	--	yes
	I00.00[10]	12	1,63	Excl.-OR neg. 1bit addr. with constant offset	--	yes
	I00.00[BM01.00]	18	4,83	Excl.-OR neg. 1bit addr. with variable offset	--	yes
	BM00.00[10]	12	1,63	Excl.-OR neg. 8bit addr. with constant offset	--	yes
	BM00.00[BM01.00]	18	4,83	Excl.-OR neg. 8bit slave addr. with var. offset	--	yes

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

Assignments and set commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
=	I00.00	4	0,63	Assignment 1bit address	--	--
	BM00.00	4	0,63	Assignment 8bit address	--	--
	O00.00[10]	8	1,13	Assignment 1bit address with constant offset	--	--
	O00.00[BM01.00]	14	4,38	Assignment 1bit address with variable offset	--	--
	BM00.00[10]	8	1,13	Assignment 8bit address with constant offset	--	--
	BM00.00[BM01.00]	14	4,38	Assignment 8bit slave addr. with var. offset	--	--
=N	I00.00	16	2,13	Assignment negation 1bit address	--	yes
	BM00.00	16	2,13	Assignment negation 8bit address	--	yes
	O00.00[10]	20	2,63	Assignment neg. 1bit addr. with const. offset	--	yes
	O00.00[BM01.00]	26	6,0	Assignment neg. 1bit addr. with var. offset	--	yes
	BM00.00[10]	12	1,63	Assignment neg. 8bit addr. with const. offset	--	yes
	BM00.00[BM01.00]	18	4,83	Assignment neg. 8bit sl. addr. with var. offset	--	yes
=D	BM00.00	4	0,75	Assignment 16bit address (even address)	--	--
	BM00.01	18	5,25	Assignment 16bit address (odd address)	--	--
	BM00.00[10]	16	5,0	Assignment 16bit addr. with constant offset	--	--
	BM00.00[BM01.00]	22	8,25	Assignment 16bit slave addr. with var. offset	--	--
S	O00.00	12	1,75	Conditional set 1bit address	--	yes
R	O00.00	10	1,38	Conditional reset 1bit address	--	yes
=1	O00.00	8	2,5	Unconditional set 1bit address	--	yes
=0	O00.00	8	2,25	Unconditional reset 1bit address	--	yes

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.2. Arithmetic commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
ADD	BM00.00	4	0,63	Addition 8bit address	yes	yes
	100	4	0,5	Addition 8bit constant	yes	yes
ADDD	BM00.00	4	0,75	Addition 16bit address (even address)	yes	yes
	BM00.01	10	3,75	Addition 16bit address (odd address)	yes	yes
	10000	4	0,5	Addition 16bit constant	yes	yes
SUB	BM00.00	4	0,63	Subtraction 8bit address	yes	yes
	100	4	0,5	Subtraction 8bit constant	yes	yes
SUBD	BM00.00	4	0,75	Subtraction 16bit address (even address)	yes	yes
	BM00.01	10	3,75	Subtraction 16bit address (odd address)	yes	yes
	10000	4	0,5	Subtraction 16bit constant	yes	yes
MUL	BM00.00	8	4	Multiplication 8bit address	0	yes
	100	10	4,13	Multiplication 8bit constant	0	yes
MULD	BM00.00	8	3,63	Multiplication 16bit address (even address)	0	yes
	BM00.01	12	6,38	Multiplication 16bit address (odd address)	0	yes
	10000	8	3,38	Multiplication 16bit constant	0	yes
DIV	BM00.00	8	4,63	Division 8bit address	0	yes
	100	8	4,8	Division 8bit constant	0	yes
DIVD	BM00.00	8	4,25	Division 16bit address (even address)	0	yes
	BM00.01	12	7,0	Division 16bit address (odd address)	0	yes
	10000	8	4,0	Division 16bit constant	0	yes

~) Only approximative indication of time, as the time depends on the operand because of iterative processing

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.3. Comparison commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
CMP	BM00.00 100	4 4	0,63 0,5	Compare 8bit address Compare 8bit constant	yes yes	yes yes
CMPD	BM00.00 BM00.01 10000	6 10 4	1,0 3,75 0,6	Compare 16bit address (even) Compare 16bit address (odd) Compare 16bit constant	yes yes yes	yes yes yes
CMP=	BM00.00 100	16 16	1,75 1,5	Compare if equal 8bit address Compare if equal 8bit constant	yes yes	yes yes
CMPD=	BM00.00 BM00.01 10000	18 22 16	2,25 5,25 2,0	Compare if equal 16bit address (even) Compare if equal 16bit address (odd) Compare if equal 16bit constant	yes yes yes	yes yes yes
CMP<>	BM00.00 100	16 16	1,75 1,5	Compare if inequal 8bit address Compare if inequal 8bit constant	yes yes	yes yes
CMPD <>	BM00.00 BM00.01 10000	18 22 16	2,25 5,25 2,0	Compare if inequal 16bit address (even) Compare if inequal 16bit address (odd) Compare if inequal 16bit constant	yes yes yes	yes yes yes
CMP<=	BM00.00 100	16 16	1,75 1,5	Compare if < or = 8bit address Compare if < or = 8bit constant	yes yes	yes yes
CMPD <=	BM00.00 BM00.01 10000	18 22 16	2,25 5,25 2,0	Compare if < or = 16bit address (even) Compare if < or = 16bit address (odd) Compare if < or = 16bit constant	yes yes yes	yes yes yes
CMP>=	BM00.00 100	16 16	1,75 1,5	Compare if > or = 8bit address Compare if > or = 8bit constant	yes yes	yes yes
CMPD >=	BM00.00 BM00.01 10000	18 22 16	2,25 5,25 2,0	Compare if > or = 16bit address (even) Compare if > or = 16bit address (odd) Compare if > or = 16bit constant	yes yes yes	yes yes yes

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.4. Shift and rotation commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
LSL	Accu	2	0,25	Log. shift left in accu, 8bit	yes	yes
LSR	Accu	6	0,75	Log. shift right in accu, 8bit	yes	yes
LSLD	Accu	2	0,25	Log. shift left in accu, 16bit	yes	yes
LSRD	Accu	2	0,25	Log. shift right in accu, 16bit	yes	yes
LSLM	BM00.00	10	1,75	Log. shift left in 8bit address	yes	yes
LSRM	BM00.00	10	1,75	Log. shift right in 8bit address	yes	yes
LSLDM	BM00.00 BM00.01	10 14	1,75 7,25	Log. shift left in 16bit address Log. shift left in 16bit address (odd)	yes	yes
LSRDM	BM00.00 BM00.01	10 14	1,75 7,25	Log. shift right in 16bit address Log. shift right in 16bit address (odd)	yes	yes
ROL	Accu	2	0,25	Roll left in accu, 8bit	yes	yes
ROR	Accu	10	1,25	Roll right in accu, 8bit	yes	yes
ROLD	Accu	2	0,25	Roll left in accu, 16bit	yes	yes
RORD	Accu	20	3	Roll right in accu, 16bit	yes	yes
ROLM	BM00.00	10	1,75	Roll left in 8bit address	yes	yes
RORM	BM00.00	14	2,25	Roll right in 8bit address	yes	yes
ROLDM	BM00.00 BM00.01	10 14	1,75 7,25	Roll left in 16bit address Roll left in 16bit address (odd)	yes	yes
RORDM	BM00.00 BM00.01	26 34	3,0 7,5	Roll right in 16bit address Roll right in 16bit address (odd)	yes	yes

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.5. Byte and flag manipulation

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
INC	BM00.00	4	0,63	Increment 8bit address	--	yes
DEC	BM00.00	4	0,63	Decrement 8bit address	--	yes
INCD	BM00.00 BM00.01	10 14	1,75 7,25	Increment 16bit address Increment 16bit address (odd)	-- --	yes yes
DECD	BM00.00 BM00.01	10 14	1,75 7,25	Decrement 16bit address Decrement 16bit address (odd)	-- --	yes yes
CLR	BM00.00	4	0,5	Clear 8bit address	--	--
NOP		2	0,25	Do-nothing operation	--	--
SEC		2	0,25	Set Carry bit = 1	yes	--
CLC		2	0,25	Clear Carry bit = 0	yes	--

*) Influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

4.3.6. Module calls

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
JPP	Program module	14	21,5	Jump to program module	--	--
JPCP	Program module	18	22	Conditional jump if yes to program module	--	--
JPF	Function module	18	22	Jump to function module	--	--
JPCF	Function module	26	22,5	Conditional jump if yes to function module	--	--
JPK	KUBES module	18	22	Jump to KUBES module	--	--
JPCK	KUBES module	26	22,5	Conditional jump if yes to KUBES module	--	--
JPINIT	Init module	14	21,5	Jump to Init module	--	--

*) Influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

4.3.7. Jump commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
JP	Jump mark	4	0,5	Unconditional jump	--	--
JPC	Jump mark		0,5	Conditional jump if yes (logical 1)	--	--
JPCN	Jump mark	8	0,5	Conditional jump if no (logical 0)	--	--
JP=	Jump mark	4	0,5	Jump if equal	--	--
JP<>	Jump mark	4	0,5	Jump if inequal	--	--
JP<	Jump mark	4	0,5	Jump if smaller	--	--
JP>	Jump mark	4	0,5	Jump if greater	--	--
JP<=	Jump mark	4	0,5	Jump if smaller or equal	--	--
JP>=	Jump mark	4	0,5	Jump if greater or equal	--	--
JPCS	Jump mark	4	0,5	Jump if Carry bit = 1	--	--
JPCC	Jump mark	4	0,5	Jump if Carry bit = 0	--	--
JPZS	Jump mark	4	0,5	Jump if Zero bit = 1	--	--
JPZC	Jump mark	4	0,5	Jump if Zero bit = 0	--	--
JP+	Jump mark	4	0,5	Jump if positive (two's complement)	--	--
JP-	Jump mark	4	0,5	Jump if negative (two's complement)	--	--

#Times: the higher value is valid if there is a jump,
the smaller value is valid if there is no jump

*) Influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

4.3.8. Copy and BCD commands

Com-mand	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
C1T8	I00.00	16	16,5	Copy 1bit addresses to 8bit accu	--	--
C8T1	O00.00	16	13,5	Copy 8bit accu to 1bit addresses	--	--
C1T16	I00.00	16	29,5	Copy 1bit addresses to 16bit accu	--	--
C16T1	O00.00	16	23	Copy 16bit accu to 1bit addresses	--	--
BINBCD3	Accu	4	8,63	Binary-BCD conversion (3 decades)	++	++
BCDBIN3	Accu	4	8,0	BCD-binary conversion (3 decades)	++	++

*) Influence on (C)arry and (Z)ero bit:
-- no change
yes defined flag alteration
++ undefined flag alteration

4.3.9. Programmable pulses, timers and counters

Com.	Operand (example)	Byte	Time [µs]	Function	C *)	Z *)
=	PP00.00	16	5,13	programmable pulse at positive edge	++	++
=N	PP00.00	16	5,13	programmable pulse at negative edge	++	++
L A O	PP00.00	4 4 4	0,63 0,63 0,63	logical operation with pulse signal	--	++
LN AN ON	PP00.00	6 8 8	0,88 1,13 1,13	logical operation with pulse signal, negated	--	++
=	PT00.00:100*10ms:E:R	14	21	progr. time with constant time value (log.1=On)	++	++
=	PT00.00:BM00.00*10ms:E	20	24	progr. time with variable time value (log.1=On) **)	++	++
=N	PT00.00:100*10ms:E:R	18	22	progr. time with constant time value (log.0=On)	++	++
=N	PT00.00:BM00.00*10ms:E	24	26	progr. time with variable time value (log0=On) **)	++	++
=TH	PT00.00	14	15	time halt (actual value is stored)	++	++
L A O	PT00.00	4 4 4	0,63 0,63 0,63	logical operation with time output	--	++
LN AN ON	PT00.00	6 8 8	0,88 1,13 1,13	logical operation with time output, negated	--	++
=	C00.00:100:V:R	14	13,5	set progr. counter with const. cnt. value (log.1=On)	++	++
=	C00.00:BM00.00:V:R	20	14,5	set progr. counter with var. cnt. value (log.1=On)**)	++	++
=N	C00.00:100:V:R	18	15	set progr. counter with const. cnt. value (log.0=On)	++	++
=N	C00.00:BM00.00:V:R	24	16	set progr. counter with var. cnt. value (log.0=On)**)	++	++
=C	C00.00	14	14,5	transfer of clock pulse (count)	++	++
L A O	C00.00	4 4 4	0,63 0,63 0,63	logical operation with counter output	--	++
LN AN ON	C00.00	6 8 8	0,88 1,13 1,13	logical operation with counter output, negated	--	++

:R this entry at the last position renders counters and timers to become remanent (buffered by the accu on the CPU) when set.

**) No external operand (PROFIBUS) can be applied for the variable timer or counter value.

*) Influence on (C)arry and (Z)ero bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.10. Special commands

Com-mand	Operand	Byte	Time [µs]	Function	C *)	Z *)
O_OFF	-		1,5	deactivates the actuating elements of all outputs		
O_ON	-		1,5	activates the actuating elements of all outputs		
RESET	-		200	resets all non-remanent outputs, markers, timers and counters and stops program execution		
WAIT	n			wait loop. Programm. delay = n (1...6) * 10 ms]		

*) Influence on (C)arry-and (Z)ero-Bit:

--	no change
yes	defined flag alteration
++	undefined flag alteration

4.3.11. Commands for the initialisation 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 with the initialization modules.

Operand	Data type	Value	Byte	Time [µs]	Function
O00.00	BIT	1 1,0,1,1.... [16],1			Write logical value into 1bit address Write logical values into 1bit and subsequent addresses Write log. value into 1bit and the 15 following addresses
BM00.00	BYTE	75 *1) 1,18,0,125... [8],128 "KUHNKE"			Write (decimal) value into 8bit address Write values into 8bit and subsequent addresses Write value into 8bit and the following 7 addresses Write text into 8bit and subsequent addresses
BM00.00	WORD	19285 *2) 1,18,0,125... [8],13283			Write (decimal) value into 16bit address Write values into 16bit and subsequent addresses Write value into 16bit and the following 7 addresses
BM00.00	TEXT	"KUHNKE"			Write text as from the specified address
		"KUHNKE", " MALENTE"...			Write texts as from the specified address

4.3.12. Commands for the data modules

Com-mand	Operand	Byte	Time [µs]	Function	C *)	Z *)
LoadDB	x,<name>	12	220	load data module <name> into DBx00.00...15.15	++	++
	byte1,<name>			load data module <name> into DBx00.00...15.15 (x = value 0...7 in byte1)		
	x,byte2			load data module number y (y = value 1...255 in byte2) into DBx00.00...15.15 (x = 0...7)		
	byte1,byte2			load data module number y (y = value 1...255 in byte2) intoDBx00.00...15.15(x = value 0...7 in byte1)		
StoreDB	x,<name>			store DBx00.00...15.15 (x = 0...7) in data module <name>		
	byte1,<name>			store DBx00.00...15.15 (x = value 0...7 in byte1) in data module <name>		
	x,byte2			store DBx00.00...15.15 (x = 0...7) in data module y (y = value 1...255 in byte2)		
	byte1,byte2			store DBx00.00...15.15 (x = value 0...7 in byte1) in data module number y (y = value 1...255 in byte2)		

*) Influence on (C)arry and (Z)ero bit:	--	no change
	yes	defined flag alteration
	++	undefined flag alteration

4.4. Registers

As a matter of size, there are three types of operands in the KUAX 680C:

- 1bit operands
- 8bit operands (bytes)
- 16bit operands (words)

The accumulator in the CPU of the KUAX 680C can be used as a 1bit, 8bit or 16bit register.

☞ *Please do not confuse: the term "accu(mulator)" in the software part stands for a general-purpose register in the processor. In the hardware part it stands for a chargeable battery.*

1bit operands are used for internal byte operations. Only bit 7 of the 8bit accu is analysed, however.
For 16bit operands, a 16bit accu is used which uses the above-mentioned 8bit accu as lowbyte. Word processing is started by commands that have a "D" as their last sign.

☞ *In order to prevent mistakes, we recommend not to use different types of operands within operations that belong together.*

4.5. Addressing

The value of the operand can be assigned in two different ways:
absolute value (voltage or current values or constant)
contents of an operand (bit, byte, word)

4.5.1. Address mnemonics

The operand addresses are indicated as mnemonic symbols, e.g. BM00.00, O00.00, PT00.00. The actual address management of the processor remains invisible.
Thus, "L BM00.00" stands for loading the contents of a memory location which carries the mnemonic name "BM00.00".

4.5.2. 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.

Examples

L I00.00[5]	is the same as	L I00.05
= O01.00[6]	is the same as	= O01.06
= BM01.00[17]	is the same as	= BM02.01

4.5.3. Addresses occupied by the operands

Group	Addr. range (Hex.)			Group	Addr. range (Hex.)		
M	09000	-	090FF	C	0AE00	-	0AEFF
SM	09100	-	091FF		0AF00	-	0AFFF
O	09200	-	092FF	LM	0B000	-	0BFF0
reserved	09300	-	093FF	FM	0B100	-	0B1FF
R	09400	-	094FF	SO	0B200	-	0B2FF
SR	09500	-	095FF	*reserved*	0B300	-	0B3FF
I	09600	-	096FF	AO	0B400	-	0B4FF
reserved	09700	-	097FF	T / PL	0B500	-	0B5FF
PP	09800	-	098FF	SI	0B600	-	0B6FF
PT	09900	-	099FF	*reserved*	0B700	-	0B7FF
	09A00	-	09AFF	*reserved*	0B800	-	0B8FF
	09B00	-	09BFF	*reserved*	0B900	-	0B9FF
	09C00	-	09CFF	*BZ CPU*	0BA00	-	0BAFF
	09D00	-	09DFF	T	0BB00	-	0BBFF
	09E00	-	09EFF	*global variables*	0BC00	-	0BC0F
	09F00	-	09FFF	ERR00.00	0BC10	-	0BC10
BM	0A000	-	0AFF0	*global variables*	0BC11		0BCFF
SBM	0A100	-	0A1FF	*reserved*	0BD00	-	0BDFF
BC	0A200	-	0A2FF	*KBM*	0BE00	-	0BEFF
SBC	0A300	-	0A3FF		0BF00	-	0BFFF
BD	0A400	-	0A4FF	*reserved*	0C000	-	0CFF0
SBD	0A500	-	0A5FF		0C100	-	0C1FF
FBM	0A600	-	0A6FF		0C200	-	0C2FF
LBM	0A700	-	0A7FF		0C300	-	0C3FF
ABM	0A800	-	0A8FF	SLA-SLP	0C400	-	0C4FF
BR	0A900	-	0A9FF		0C500	-	0C5FF
SBR	0AA00	-	0AAFF	*RAM for SLA-SLP*	0C600	-	0C6FF
BI	0AB00	-	0ABFF		0C700	-	0C7FF
BO	0AC00	-	0ACFF	DB0-DB7	27800	-	27FFF
AI	0AD00	-	0ADFF				

Make sure in any case not to write into reserved ranges when using offset addressing.

4.5.4. Types of addressing: overview

The load command is taken as an example to give an overview of the different types of addressing.

To load the contents of an operand		
L	I00.00	1bit address
L	BM00.00	8bit address
LD	BM00.00	16bit address
To load a constant value		
8bit constant (0...255):		
L	100	decimal
L	$64	hexadecimal
L	%01100100	binary
L	´A´	ASCII
16bit constant (0...65535):		
LD	10000	decimal
LD	$3FEA	hexadecimal
LD	%0010011100010000	binary
LD	4.5V	voltage (-10...+10V)
LD	5mA	current(-20...+20mA)
To load the contents of an offset-addressed operand		
with constant offset (0...255):		
L	I00.00[10]	1bit address
L	BM00.00[10]	8bit address
LD	BM00.00[10]	16bit address
with variable offset in the local operand (0...255):		
L	I00.00[BM01.00]	1bit address
L	BM00.00[BM01.00]	8bit address
LD	BM00.00[BM01.00]	16bit address
with variable offset in the external operand (0...255)		
L	I00.00[BI03a00.]	1bit address
L	BM00.00[BI03a00.]	8bit address
LD	BM00.00[BI03a00.]	16bit address

4.6. Description of the commands

The following overview explains all commands in plain text.

4.6.1 Logical operations commands

Logical 8bit operations include bit-by-bit negation (one's complement) if an N is added to the command

4.6.1.1. Load and logical operations commands

Cmnd	Function
L, LD	load Loads the value of the operand into the accu
A, AD	logical AND operation Logical AND operation bit-by-bit between the value of the operand and the contents of the accu. The result of the operation is stored in the accu.
O, OD	logical OR operation Logical OR operation bit-by-bit between the value of the operand and the contents of the accu. The result of the operation is stored in the accu.
XO	logical exclusive-OR operation Logical exclusive-OR operation bit-by-bit between the value of the operand and the contents of the accu. The result of the operation is stored in the accu.

To read unoccupied input addresses

It should be avoided to read input addresses for which no module is plugged in. In such cases, the value read depends on the current status of the bus data line and is undefined (i.e. not defined "0").

4.6.1.2. 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.

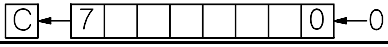
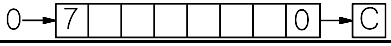
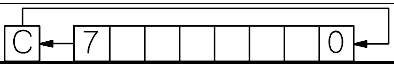
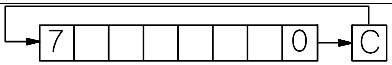
4.6.2 Arithmetic commands

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.

4.6.3 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.

4.6.4. 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.
	

4.6.5. Byte and flag manipulation

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.

4.6.6. 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

4.6.7. 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 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

4.6.8. Copy and BCD commands

The operating method of the copy commands is explained by the program examples (see chapter "6.13. Bit-to-byte transfer").

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 of 8bit accus into eight 1 bit operands
C16T1	copies the value of 16bit accus into sixteen 1 bit operands
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. After the operation, it contains the same value as a 16bit bin. valu

4.6.9. 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 control on (or after a RESET), the pulse has to be passed once at a value of 0 as the function cannot be guaranteed otherwise.
<u>Recommendation:</u> Assign the pulse with a marker and map the input signal after it on the marker.

Example:

```
L       M00.00    ;marker
=       PP00.00   ; creates a pulse
L       I00.00    ;map input
=       M00.00    ; on marker
```

4.6.10. 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.

<u>To start a timer:</u>

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)

<u>To scan an output:</u> L PTxx.xx

Load logical time output into the accu

<u>To scan the actual value (resid. value):</u> LD PTxx.xx

Example:

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

Write residual value into BM06.02 and BM06.03

<u>To halt the timer:</u> =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.

4.6.11. 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.

<u>To start a counter:</u>

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).

<u>To scan an output:</u> L Cxx.xx ;Count complete

Load logical counter output into the accu

<u>To scan an actual value:</u> LD Cxx.xx

Example: LD C01.01
=D BM06.02

Write actual value into BM06.02 and BM06.03

<u>To count/transfer the clock pulse:</u> =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.

4.6.12. 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 appendix "D. 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 appendix "D. 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 for example to react with a defined program execution delay to undervoltage (see appendix "D. Reactions to failures").

4.6.13. 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

4.6.14. Commands of the data modules

Data modules are stored in the user memory (either in the EPROM or the RAM). You can create up to 255 data modules of a capacity of 256 byte each.
Operand ranges DB0...DB7 are used for accessing the data modules from within the user program. Each of these operand ranges has a capacity of 256 byte and can be addressed like all other byte markers: DBx00.00...15.15 (x=0...7). Also, these operands work with all commands that were described previously as suitable for use with byte markers.

The LoadDB and StoreDB commands can only be used with data processing ranges DB0...DB7:

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)

4.7. Module programming

The user program of the KUAX 680C is built up as a module structure. The user is thus enabled to divide the technological problem he wants to control up into separate sub-tasks. The individual modules form a hierarchical system on a maximum of 5 levels in which modules on higher levels call up modules on lower ones. Such a program structure is very clear and considerably facilitates, amongst other things, the understanding or the maintenance of finished programs. The following types of modules can be distinguished:

- organization module
- program modules
- function modules
- timer modules
- interrupt modules
- initialization modules
- data modules
- trigger modules
- KUBES modules

The watchdog monitors the cycle time

The processing of the individual modules is monitored by a watchdog. The watchdog is triggered every time a module is called up. After that, 70ms are available for processing the module.
An additional watchdog time monitors the overall program: a watchdog error is also reported if the organization module is not re-passed (i.e. the program cycle completed) after a maximum of 2 s.

The modules are a kind of sub-program. That means that returning to the calling module is already defined by the inert module organization.

The return to the calling module must not be written as an instruction into the user program. Neither must modules be allowed to call themselves up.

4.7.1. Organization module

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 Guide, 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 of the modules responsible for overall tasks.
All commands are applicable without limitations (commands of the initialization modules excluded).

4.7.2. Program module

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).

4.7.3. Function module

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.
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.

4.7.4. Timer module

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
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).

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.

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

Module No.	Called by time interrupt
1	10 ms
2	100 ms
3	1 s
4	10 s

4.7.5. 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, other interrupt modules or by failure or error messages.

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.
Call:	automatically by the assigned interrupt

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

Call by modules that can trigger interrupts

Function modules, e.g. counter modules, communicate with the user program via the SLx... transfer addresses (in the KUAX 657, these indicate the slave dual-port RAM addresses, hence the name SLx).
Each module slot is assigned 32 addresses (16 from any one group of addresses, e.g. SLA and SLB for slot 0) which serve various functions depending on the type of module. Each group of addresses allows the call of an interrupt module so that one module can trigger up to 2 interrupts.

Assignment of transfer addresses and interrupt modules

Using/ triggering	Address range	Interrupt module
module 0	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 counters	SLI00.00...01.15	-
internal interrupt inputs	SLJ00.00...01.15	10
internal analog inputs	SLK00.00...01.15	-
error message voltage supply	-	17
error message short circuit	-	18

In the case of certain system errors, appropriate measures should be taken in the user program to minimalize the effects of such errors. In order to be able to react fast enough, the monitor program calls an interrupt module by interrupt. You can use this interrupt module to program the desired reactions.

☞ *Appendix "D. Reactions to failures" contains some recommendations and suggestions concerning this subject.*

4.7.6. Initialization module

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 "4.6.13. Commands of the initialization modules").

☞ *Initialization modules should only be called when needed but not cyclically (cycle time).*

4.7.7. Data module

Name:	xxxxxxxx.DAT
Number:	255
Length:	max. 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 all other types of modules using the LoadDB and StoreDB commands (see chapter "4.7.14. Commands of the data modules").

4.7.8. Trigger module

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. 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.

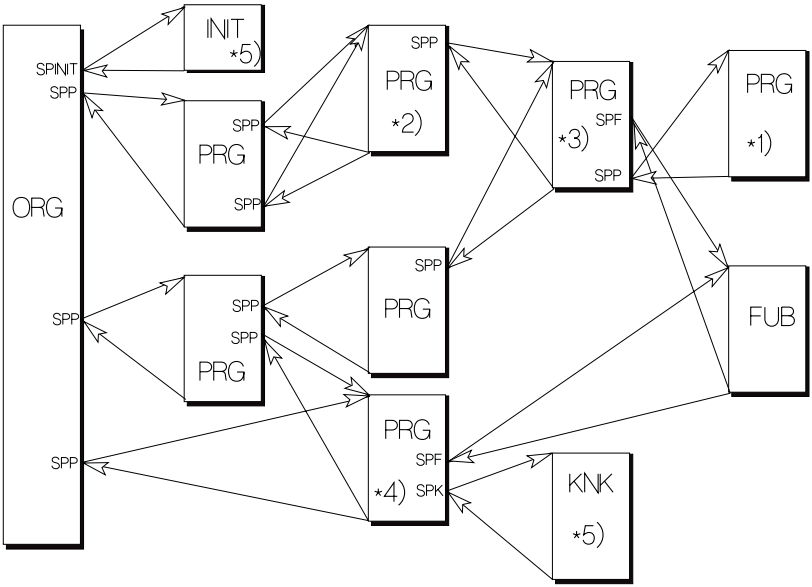
4.7.9. KUBES module

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

Created 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.

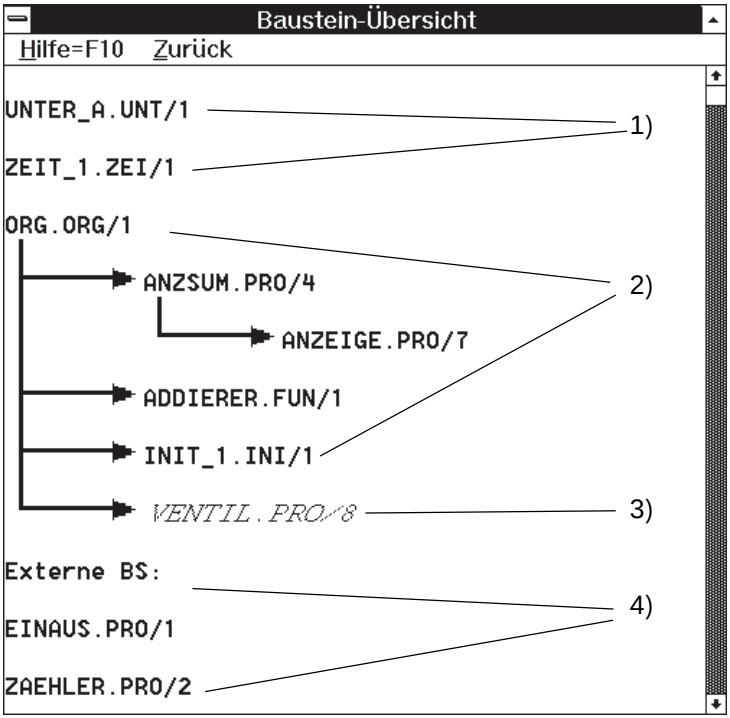
☞ *Programmable timers (PTxx.xx) are not permissible as input parameters. Not the logical timer output would be read but the value of the status byte.*
Remedy: Assign the timer output to a marker and then use the marker as input parameter.

4.7.10. Module hierarchy (example for different module calls)



KUBES module overview

Use the KUBES View Tree function to have the entire module hierarchy of a project displayed. All modules of the current project are shown. You can also print the tree.



1) Modules which are called up automatically
   - Interrupt modules (.INT); they are called up by function modules
   - Timer modules (.TIM); they are called up by time interrupts
2) Modules which are called up by a program command
3) Virtual modules (in italics); there is a command to call them up but they have not been edited yet (they are empty).
4) External modules
   - Modules which belong to the project but are not called at present
   - Trigger modules (.TRG); they are only called up in test mode.

5. Networking

The KUAX 680C was mainly developed as a local controller. It was not designed for use as a master in a PROFIBUS network.

However, the built-in serial RS 485 interface allows communication with less extended protocols.
Details concerning network communication were not available at the copy deadline of this edition of the instruction manual, though.
We will provide more information in this chapter of later editions.

Please feel free to contact us for further information.

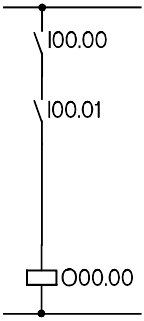
6. Programming examples

The following examples use operands that are partly not available in the KUAX 680C (inputs and outputs are counted octally; channels ".08" ... ".15" do not exist). They can be arbitrarily replaced, however.

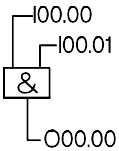
6.1. Basic functions

6.1.1. AND

Circuit diagram



Function diagram

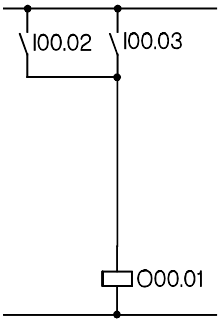


Instruction list

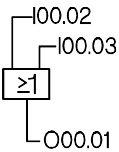
```
L    I00.00
A    I00.01
=    O00.00
```

6.1.2. OR

Circuit diagram



Function diagram

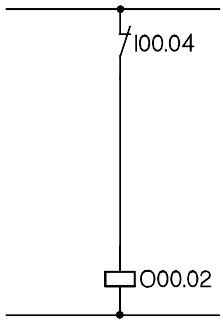


Instruction list

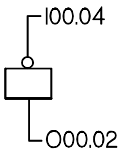
```
L    I00.02
O    I00.03
=    O00.01
```

6.1.3. Negation at input

Circuit diagram



Function diagram

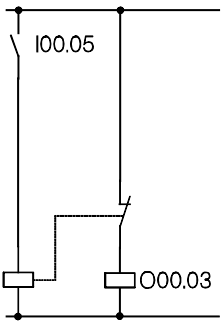


Instruction list

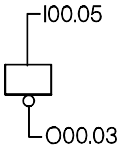
```
LN    I00.04
=     O00.02
```

6.1.4. Negation at output

Circuit diagram



Function diagram

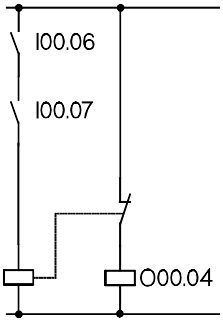


Instruction list

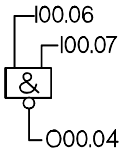
```
L     I00.05
=N    O00.03
```

6.1.5. NAND

Circuit diagram



Function diagram

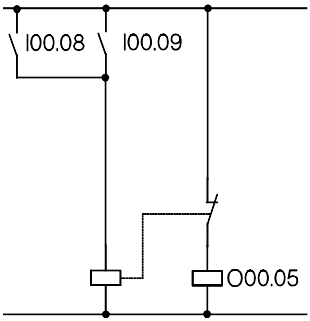


Instruction list

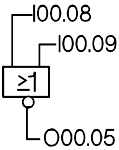
```
L     I00.06
A     I00.07
=N    O00.04
```

6.1.6. NOR

Circuit diagram



Function diagram

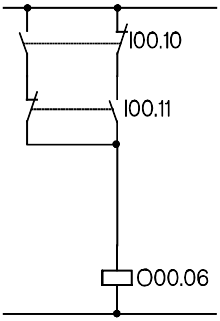


Instruction list

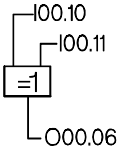
```
L     I00.08
O     I00.09
=N    O00.05
```

6.1.7. XO EXCLUSIVE-OR (non-equivalence)

Circuit diagram



Function diagram

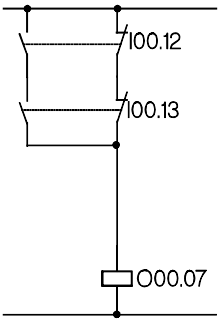


Instruction list

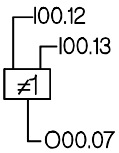
```
L     I00.10
XO    I00.11
=     O00.06
```

6.1.8. XON EXCLUSIVE-NOR (equivalence)

Circuit diagram



Function diagram

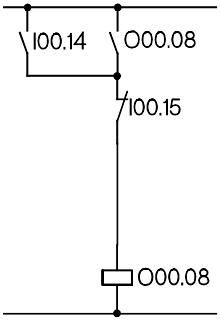


Instruction list

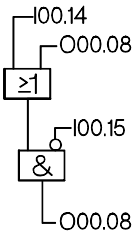
```
L     I00.12
XON   I00.13
=     O00.07
```

6.1.9. Self-locking circuit

Circuit diagram



Function diagram



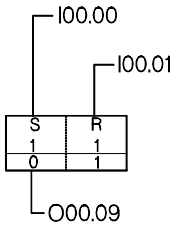
Instruction list

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

6.2. Memory functions

6.2.1. With reset dominance

Circuit symbol

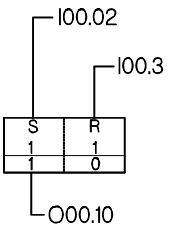


Instruction list

```
L    I00.00
S    M00.00
L    I00.01
R    M00.00

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

6.2.2. With set dominance

Circuit symbol



Instruction list

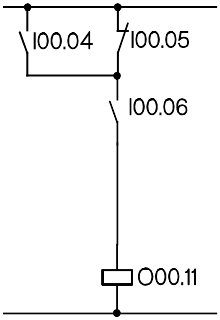
```
L  I00.02
R  M00.01
L  I00.03
S  M00.01

L  M00.01 *)
=  O00.10
```

*) If, in controls that work without process mapping, the set and reset inputs are activated simultaneously a jittering of the output may occur. In the KUAX 680C, the result must therefore be stored temporarily in a marker.

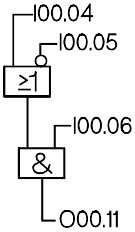
6.3. Combinational circuits

6.3.1. OR-AND circuit

Circuit diagram



Function diagram

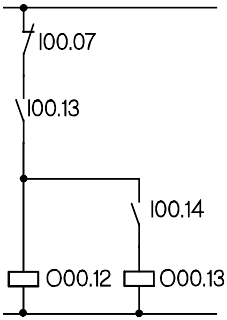


Instruction list

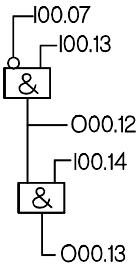
```
L     I00.04
ON    I00.05
A     I00.06
=     O00.11
```

6.3.2. Parallel circuit to output

Circuit diagram



Function diagram

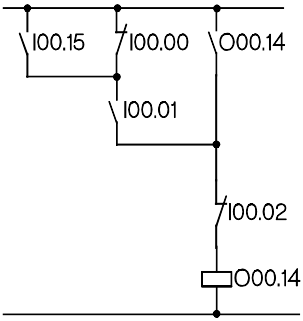


Instruction list

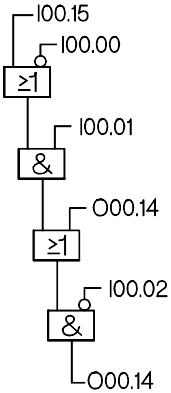
```
LN    I00.07
A     I00.13
=     O00.12
A     I00.14
=     O00.13
```

6.3.3. Network with one output

Circuit diagram



Function diagram

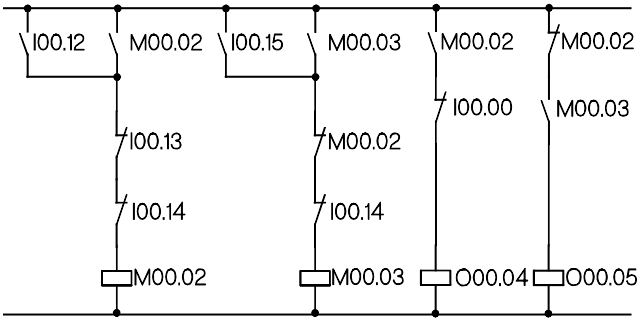


Instruction list

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

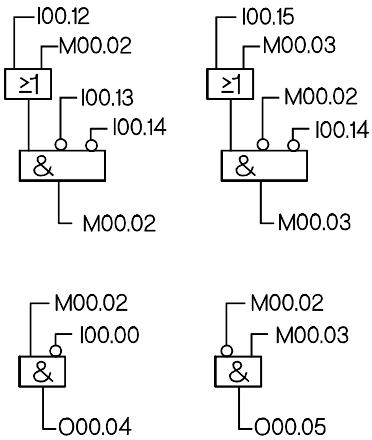
Circuit diagram



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
```

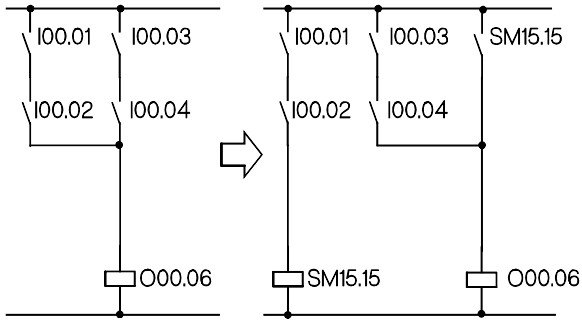
Function diagram



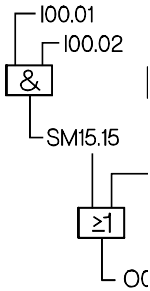
6.4. S-marker as AND / OR marker

6.4.1. Network with OR marker

Circuit diagram



Function diagram



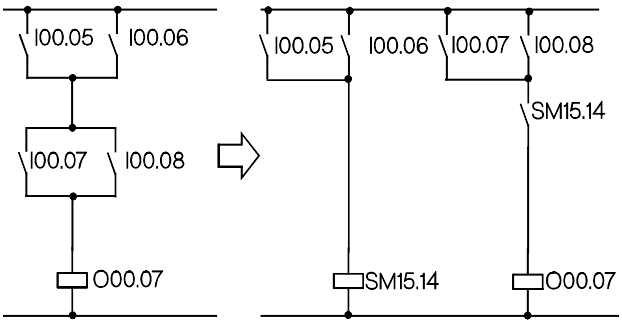
Instruction list

L	I00.01	Note:	In this example, a part result
A	I00.02		has to be stored temporarily.
=	SM15.15	Definition:	S-marker SM15.15 is basically
L	I00.03		always used as OR marker
A	I00.04		as it can always be re-used in
O	SM15.15		other networks.
=	O00.06		

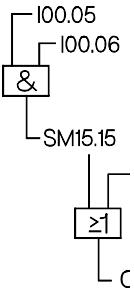
OR marker = SM15.15

6.4.2. Network with AND marker

Circuit diagram



Function diagram



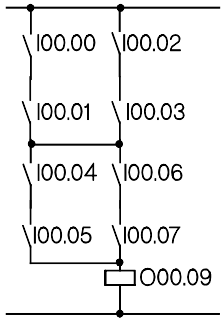
Instruction list

		Note:	In this example, too, a
L	I00.05		result has to be stored
O	I00.06		temporarily in an S-marker.
=	SM15.14		This marker is linked in
L	I00.07		an AND operation.
O	I00.08	Definition:	S-marker 15.14 is basically always
A	SM15.14		used as AND marker.
=	O00.07		

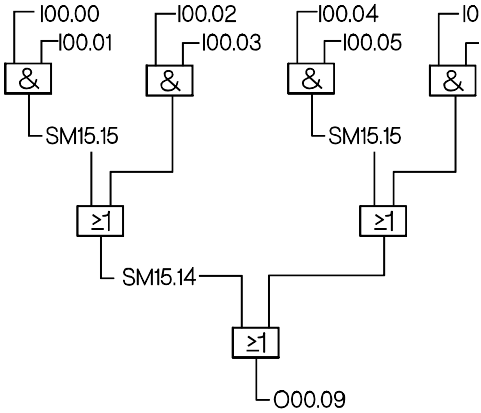
AND marker = SM15.14

6.4.3. Network with multiple use of the OR marker

Circuit diagram



Function diagram

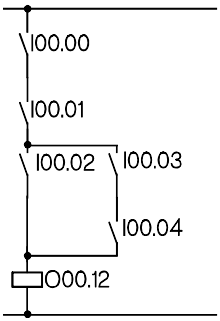


Instruction list

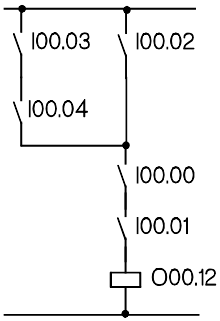
```
L     I00.00
A     I00.01
=     SM15.15  ;set OR marker
L     I00.02
A     I00.03
O     SM15.15
=     SM15.14  ;set AND marker
L     I00.04
A     I00.05
=     SM15.15  ;set OR marker
L     I00.06
A     I00.07
O     SM15.15
A     SM15.14
=     O00.09
```

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

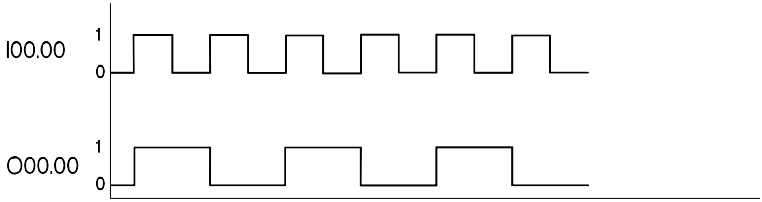
L	I00.03
A	I00.04
O	I00.02
A	I00.00
A	I00.01
=	O00.12

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

6.6. Special circuits

6.6.1. Current surge relay

Signal course

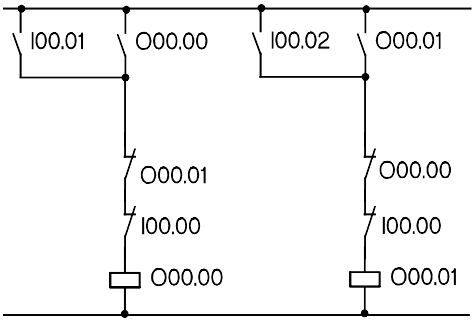


Instruction list

```
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
*1)

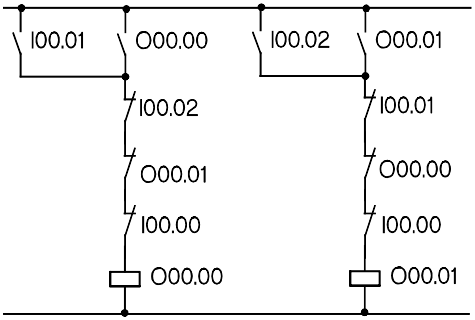


Instruction list

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

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

6.6.3. Reverse circuit (reverse contactor) without forced halt

Circuit diagram
*1)



Instruction list

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

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

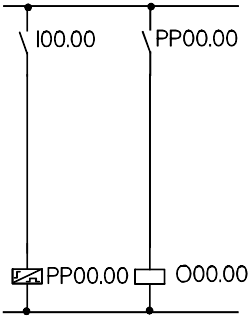
*1) As the switching of the outputs is done very quickly it is advisable to provide a contactor interlock outside the PLC.
*2) If, for safety reasons, the stop key is already connected as n.c. switch outside the PLC, an A (AND) has to be programmed here.

6.7. Pulse edge evaluation

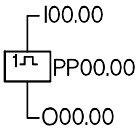
The KUAX 680C contains 128 programmable pulses for status change recognition of logical signals (edge evaluation). They can be used for both the positive and the negative edge.

6.7.1. Programmable pulse with positive edge

Circuit diagram



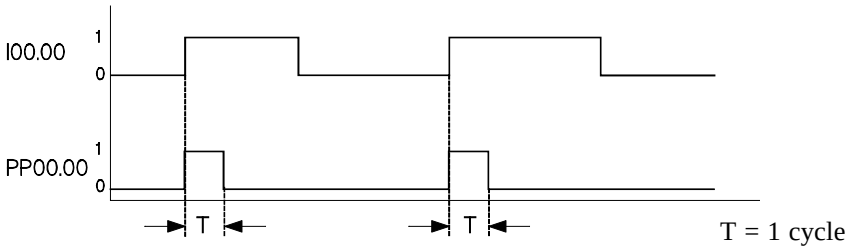
Switching symbol



Instruction list

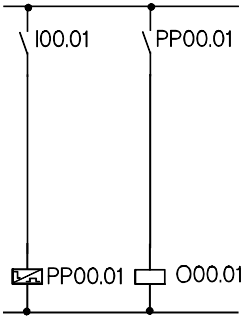
```
L    I00.00
=    PP00.00
L    PP00.00
=    O00.00
```

Signal course

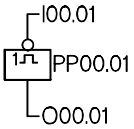


6.7.2. Programmable pulse with negative edge

Circuit diagram



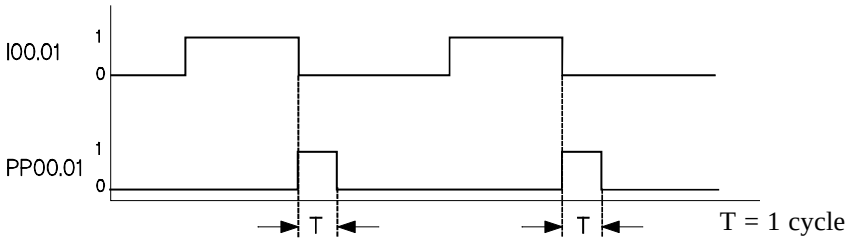
Switching symbol



Instruction list

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

Signal course



Behaviour of the progr. pulses after switching the controller on

After switching the controller on (or after a 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

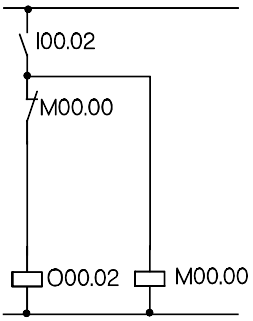
```
L     M00.00
=     PP00.00
L     I00.00
=     M00.00
L     PP00.00
=     O00.00
```

Examples

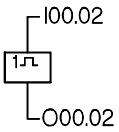
As opposed to the programmable pulses (see above) which are activated by edge reversals, the signal status is evaluated in the following two examples. This causes a different behavior when switching the control on.

6.7.3. Pulse with positive signal

Circuit diagram



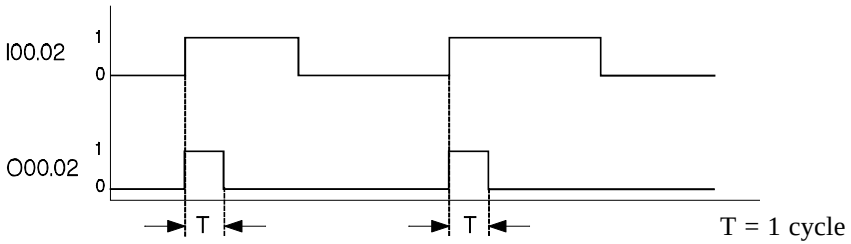
Switching symbol



Instruction list

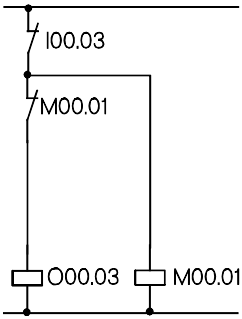
```
L    I00.02
=    SM15.14
AN   M00.00
=    O00.02
L    SM15.14
=    M00.00
```

Signal course

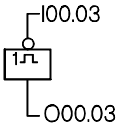


6.7.4. Pulse with negative signal

Circuit diagram



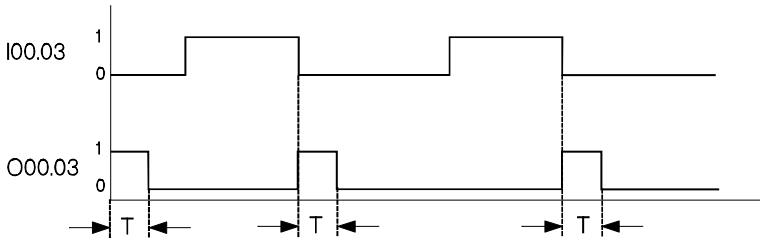
Switching symbol



Instruction list

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

Signal course

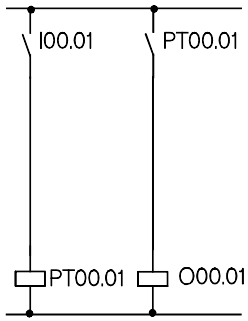


T = 1 cycle

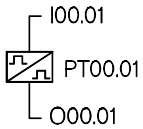
6.8. Software timers

6.8.1. Impulse at startup

Circuit diagram



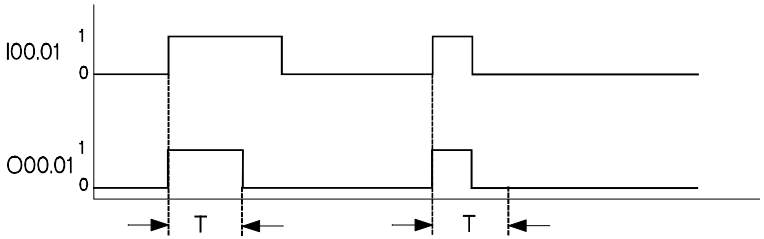
Switching symbol



Instruction list

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

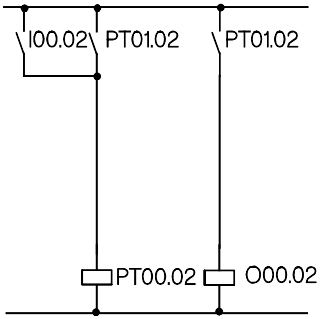
Signal course



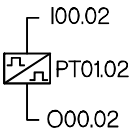
T= Time preselection (here: 1.35s)

6.8.2. Impulse with constant duration

Circuit diagram



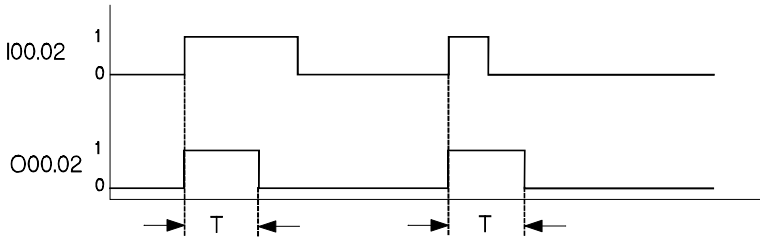
Switching symbol



Instruction list

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

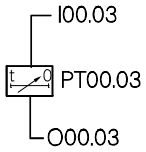
Signal course



T= Time preselection (here: 12.3s)

6.8.3. Raising delay

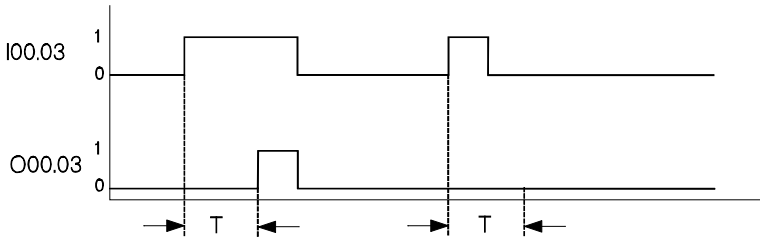
Switching symbol



Instruction list

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

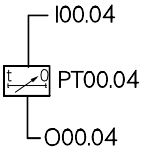
Signal course



T= Time preselection (here: 1.85s)

6.8.4. Falling delay

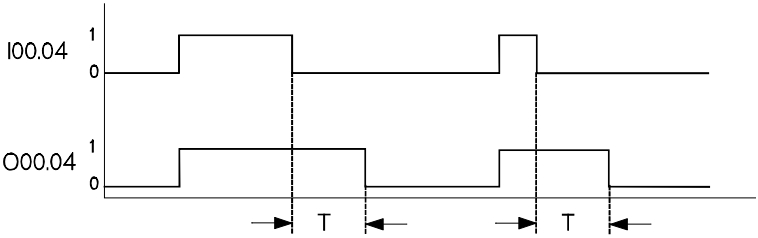
Switching symbol



Instruction list

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

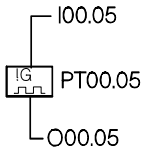
Signal course



T= Time preselection (here: 3.5s)

6.8.5. Impulse generator with pulse output

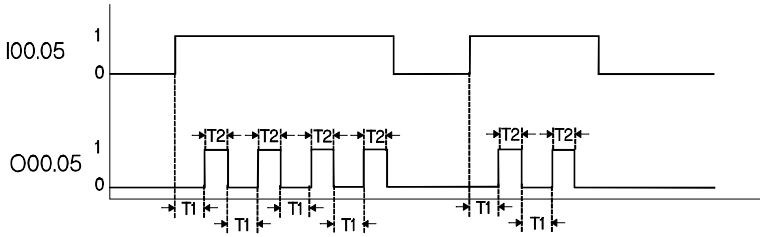
Switching symbol



Instruction list

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

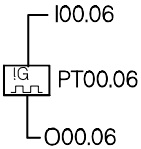
Signal course



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

6.8.6. Flash generator with one timer

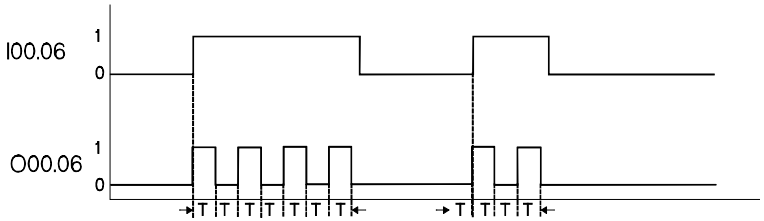
Switching symbol



Instruction list

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

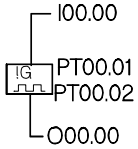
Signal course



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

6.8.7. Flash generator with two timers

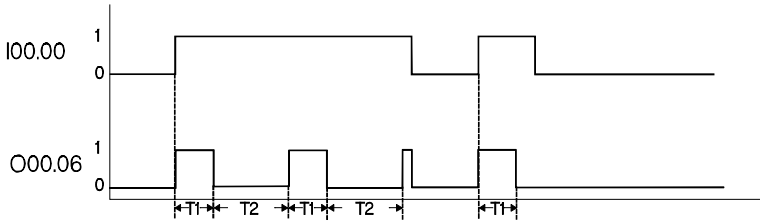
Switching symbol



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.

Example for an application: 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 yes
        =     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*

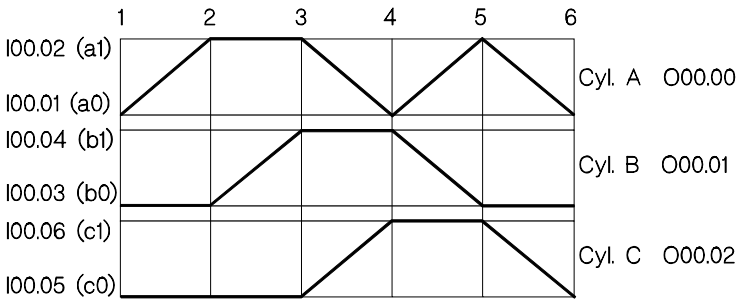
6.10. Software counters

Example: Forward counter to 12

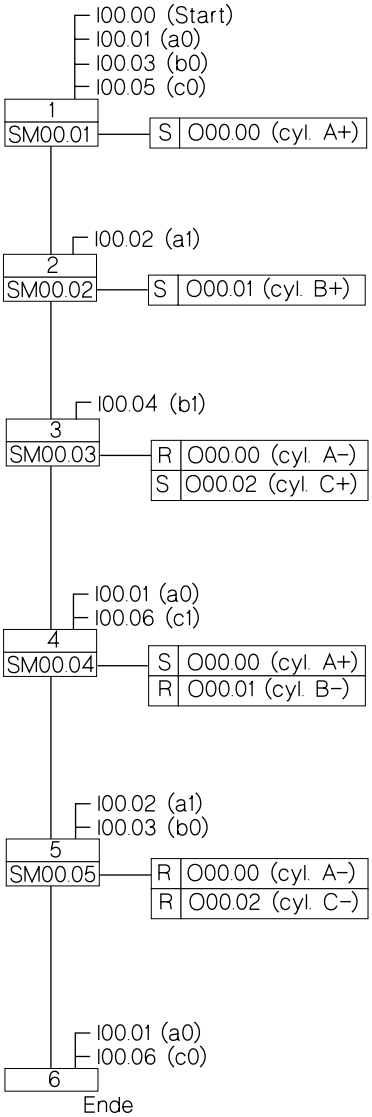
```
L     I00.00        ;start counter
=     C00.00:12:F

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 of an operational sequence

Path-step diagram



Function diagram



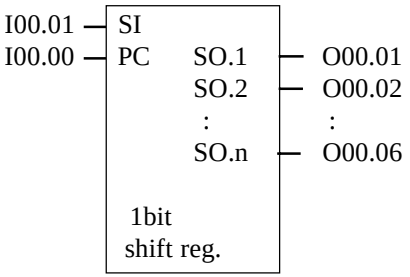
Program

```
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
```

6.12. 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.



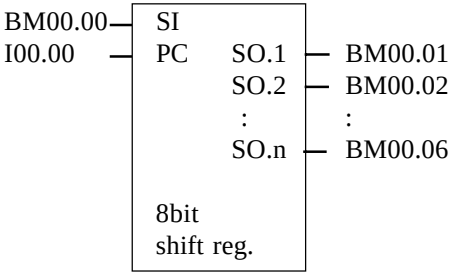
SI:	signal input		I00.01
PC:	shift clock	pulse	I00.00
SO.1:	signal output 1		O00.01
SO.2:	signal output 2		O00.02
:	:	:	
SO.n:	signal output n		O00.06

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

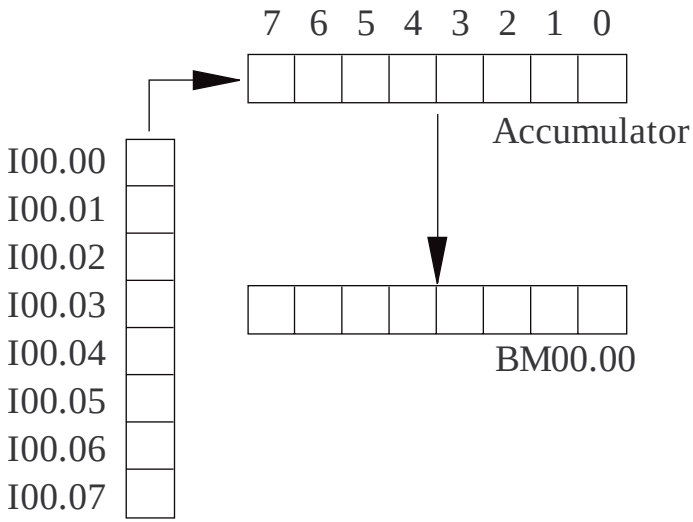
```
        L     I00.00    ;shift clock pulse
        =     PP00.00   ;pulse
        L     PP00.00   ;pulse
        JPCN  NORM      ;to normal program if no
        L     BM00.05   ;step 5
        =     BM00.06   ;step 6
        L     BM00.04   ;step 4
        =     BM00.05   ;step 5
        L     BM00.03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

6.13.1. 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
```



6.13.2.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
```

6.13.3.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)
```

6.13.4. To copy two bytes into sixteen 1bit operands

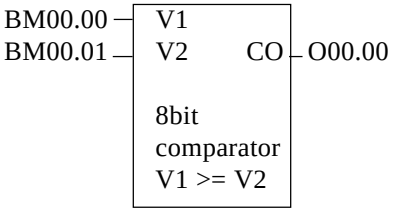
```
LD      BM00.04 ;load contents of BM00.04-BM00.05 into the accumulator
C16T1   O00.00  ;copy contents of the accu into the address O00.00-O00.15
                ;(BM00.04 into O00.00-O00.07,BM00.05 into O00.08-O00.15)
```

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":

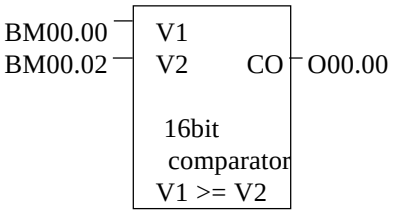
```
L        BM00.00
CMP      BM00.01
JP>=     MARK *2)
```

*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

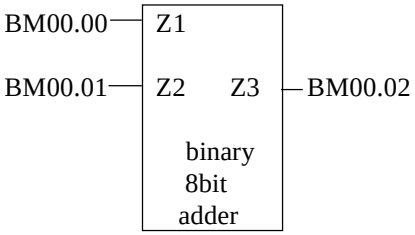
The result of the comparison is evaluated as a conditional jump, i.e. the jump is carried out if the result is "correct":

```
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



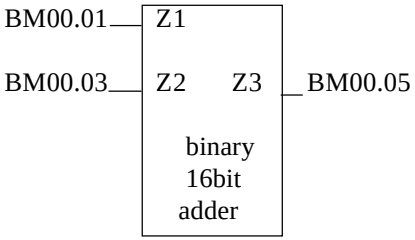
Z1:	1st summand BM00.00	8bit	0-255 ($FF)
Z2:	2nd summand BM00.01	8bit	0-255 ($FF)
Z3:	sum BM00.02	8bit	0-255 ($FF)

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



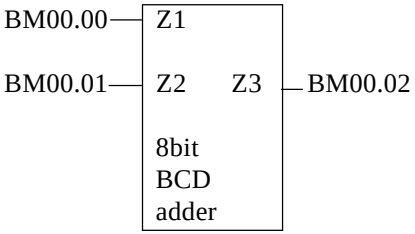
Z1:	1st summand BM00.01(HB)+BM00.00(LB)	16bit	0-65535 ($FFFF)
Z2:	2nd summ. BM00.03(HB)+BM00.02(LB)	16bit	0-65535 ($FFFF)
Z3:	sum BM00.05(HB)+BM00.04(LB)	16bit	0-65535 ($FFFF)

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

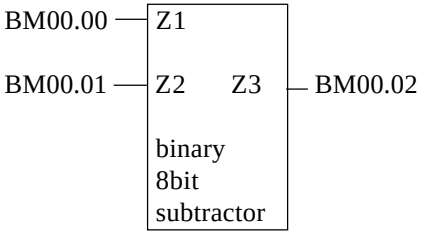


Z1:	1st summand BM00.00	8bit	0-99
Z2:	2nd summand BM00.01	8bit	0-99
Z3:	sum BM00.02	8bit	0-99

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



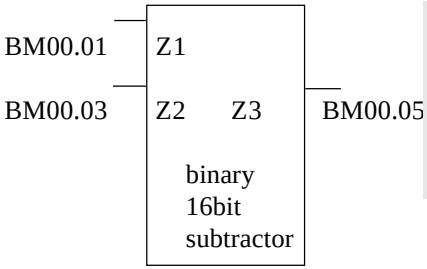
Z1:	minuend BM00.00	8bit	0-255 ($FF)
Z2:	subtrahend BM00.01	8bit	0-255 ($FF)
Z3:	difference BM00.02	8bit	0-255 ($FF)

Z3 becomes negative and is filed as two's complement if Z2 > Z1. Further evaluation of Z3 has to take this into consideration.

Program

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

6.15.5. Binary 16bit subtractor

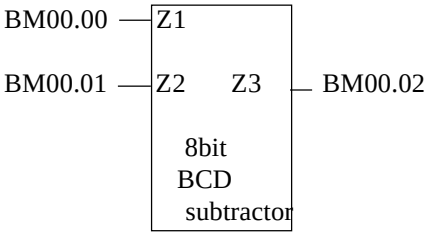


Z1:	1st minuend 16bit 0-65535 ($FFFF) BM00.01(HB)+BM00.00(LB)
Z2:	2nd subtrahend 16bit 0-65535 ($FFFF) BM00.03(HB)+BM00.02(LB)
Z3:	difference 16bit 0-65535 ($FFFF) BM00.05(HB)+BM00.04(LB)

Program

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

6.15.6. 8bit BCD subtractor

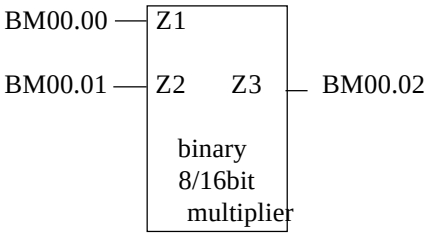


Z1:	minuend BM00.00	8bit	0-99
Z2:	subtrahend BM00.01	8bit	0-99
Z3:	difference BM00.02	8bit	0-99

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

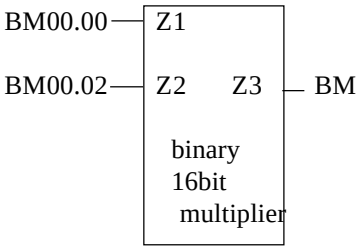


Z1:	multiplicand	8bit	0-255 ($FF)
	BM00.00		
Z2:	multiplier	8bit	0-255 ($FF)
	BM00.01		
Z3:	product	16bit	0-65025 ($FE01)
	BM00.03(HB)+BM00.02(LB)		

Program

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

6.15.8. Binary 16bit multiplier

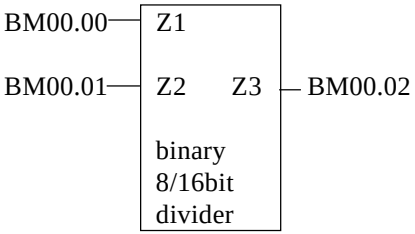


Z1:	multiplicand	16bit	0-65535 ($FFFF)
	BM00.01(HB)+BM00.00(LB)		
Z2:	multiplier	16bit	0-65535 ($FFFF)
	BM00.03(HB)+BM00.02(LB)		
Z3:	product	16bit	0-65535 ($FFFF)
	BM00.05(HB)+BM00.04(LB)		

Program

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

6.15.9. Binary 8bit divider

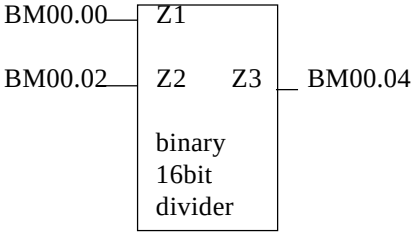


Z1:	dividend BM00.00	8bit	0-255 ($FF)
Z2:	divisor BM00.01	8bit	0-255 ($FF)
Z3:	quotient BM00.02	8bit	0-255 ($FF)

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

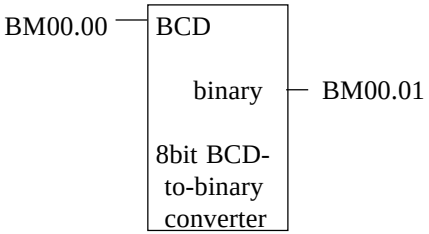
```
LD      BM00.00     ;Z1 dividend
DIVD    BM00.02     ;Z2 divisor
=D      BM00.04     ;Z3 quotient
```

The calculated quotient is integer. The remainder can be established 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

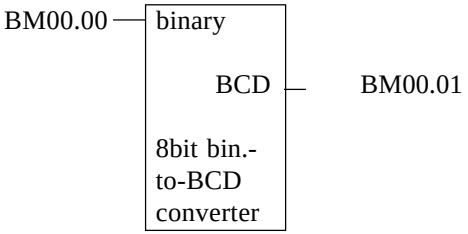


BCD:	8bit	0-99	BM00.00
binary:	8bit	0-99 ($63)	BM00.01

Program

```
L       BM00.00     ;load BCD value
LSR                 ;shift
LSR                 ;tens
LSR                 ;to
LSR                 ;units
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

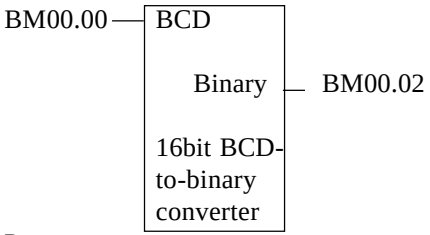


binary:	8bit	0-99 ($63)	BM00.00
BCD:	8bit	0-99	BM00.01

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

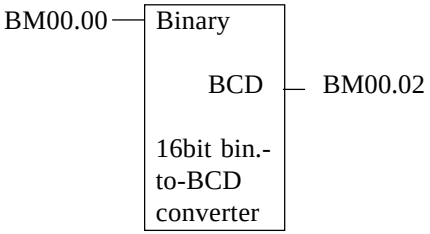


BCD:	16bit 0-9999 BM00.01(HB)+BM00.00(LB)
binary:	16bit 0-9999 ($270F) BM00.03(HB)+BM00.02(LB)

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



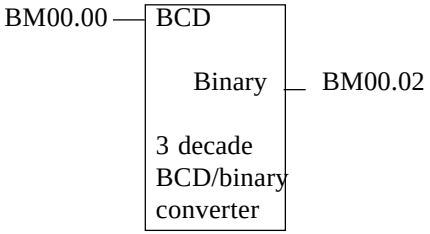
Binary:	16bit 0-9999 ($270F)
	BM00.01(HB)+BM00.00(LB)
BCD:	16bit 0-9999
	BM00.03(HB)+BM00.02(LB)

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



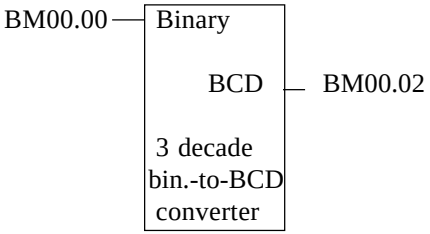
BCD:	16bit 0-999
	BM00.01(HB)+BM00.00(LB)
binary:	16bit 0-999 ($03E7)
	BM00.03(HB)+BM00.02(LB)

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, it is advisable to first convert these into binary values by use of the command BCDBIN3 and then to execute 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
```

6.17. Module programming

Task (example):

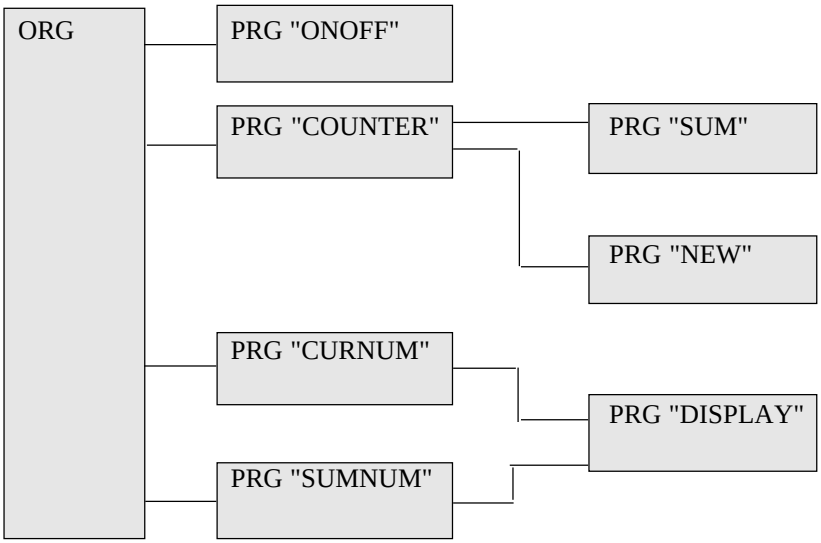
Sets of 12 pieces each are to be transported on a conveyor belt. The drive of the belt is operated by start and stop keys. The belt is stopped after every twelfth piece. Before leaving the belt, each piece triggers an impulse via an initiator which is used for counting.

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 657 ========
                        Project structure

Project     : E205GB
                                        created : Nov 19 1991 09:42
User        : Gerd Hildebrandt     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 657 ========
                        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 657 ========
                        Program module  IL

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


  1:          LD      BM00.02
  2:          BINBCD3
  3:          C16T1   UNITS          SO00.00 ; (display "units”)
  4:
```

```
======== Kubes ================================== KUAX 657 ========
                        Program module  IL

Project : E205GB
Module     : CURNUM     No.: 3     created: Nov 26 1991 16:20
User       : Gerd Hildebrandt     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 657 ========
                        Program module  IL

Project : E205GB
Module     : SUMNUM     No.: 4     created : Nov 26 1991 16:22
User       : Gerd Hildebrandt     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 657 ========
                        Program module  IL

Project : E205GB
Module     : ONOFF    No.: 1     created : Nov 26 1991 16:12
User       : Gerd Hildebrandt    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:
================================== KUAX 657 ========
                        Program module  IL

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


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

```
======== Kubes ================================== KUAX 657 ========
                        Program module  IL

Project : E205GB
Module     : SUM       No.: 5    created : Nov 26 1991 16:18
User       : Gerd Hildebrandt    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 657 ========
                        Program module  IL

Project : E205GB
Module     : COUNTER   No.: 2    created : Nov 26 1991 16:15
User       : Gerd Hildebrandt    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             00.02 ; (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:
```

A. Technical specifications

Admissible ambient conditions
- Storage temperature ..... -25...+70 °C
- Ambient temperature during oper. ..... 0...55 °C
- Relative humidity ..... 50...95 %

Power supply ..... 24 V DC -20%/+25%
- Current consumption of basic device 160 mA max. (without modules)

Test voltage ..... 500 V DC (to IEC 1131-2)

Protection class 1

Processor 80C166

Memory
- program memory ..... 112 KByte Flash-EPROM
- data memory ..... 64 KByte buffered RAM

Accumulator ..... for buffering the remanent operands and data memories
- buffer time ..... typ. 3 months
- charging time ..... max. 110 hours

An accu of a delivered device can be discharged according to uncertain storage time

Line interfacing
- Supply ..... screw-type locking term.s, matrix 5.08
- Inputs and outputs ..... screw-type locking term., matrix 3.81
- Interface(s)
  - RS 232/1 and RS 232/2 ..... female 9pin Sub-D connector
  - RS 485 ..... screw-type locking term., matrix 3.81

Digital inputs of the basic device

Number 16
Addressing ..... I00.00...00.07, I01.00...01.07
Supply voltage ..... 24 V DC - 20 % /+ 25 %
for further information ..... ☞ 3.7.1. Digital inputs

Counter inupts of the basic device

Number 2
Addressing ..... SI00. 00...00. 01
Supply voltage ..... 24 V DC - 20 % /+ 25 %
Number of counters 2
Counting frequency ..... 10 kHz max.
for further information ..... ☞ 3.7.2. Counter inputs

Interrupt inputs of the basic device

Number 2
Addressing ..... SI01.00...01.01
Supply voltage ..... 24 V DC - 20 % /+ 25 %
for further information ..... ☞ 3.7.3. Interrupt inputs

Analog inputs of the basic device

Number ..... 4 single-ended
Range ..... 0...10 V
Resolution ..... 10 bit
for further information ..... ☞ 3.7.4. Analog inputs

Digital outputs of the basic device

Number 16
Addressing ..... O00.00...00. 07, O01.00...01.07
Supply voltage ..... 24 V DC - 20 % /+ 25 %
for further information ..... ☞ 3.7.5. Digital outputs

Analog outputs of the basic device

Number 2
Range ..... 0...10 V
Resolution ..... 8 bit
for further information ..... ☞ 3.7.6. Analog outputs

Module slots 4
Usable modules ..... modules of KUAX 680I and 680C, produced as from calendar week 27/95
for further information ..... ☞ 3.8. Module slots

Program memory ..... built-in
Flash-EPROM ..... 112 KByte
RAM ..... 64 KByte
Data protection in the RAM ..... accu 110 mAh, buffer time 6 weeks min. (at 0...40 °C), charging time 72 h max.

Programming
Programming device ..... IBM-PC (or compatible PC)
Operating systems ..... MS-DOS, MS-WINDOWS version 3.1 or better
Programming software ..... KUBES version 4.12 or better
Type of programming IL
Program documentation ..... IL, FD, LD, SymT, CRL

B. Order specifications

Basic device 680.430.01

Screw-type locking terminals
- for the basic device (1 set) 680.180.11
- 8pin, with wiring and 3 m cable (1 pc.) 680.180.08

Simulator box for digital inputs (4 x 8pin) 680.155.50

Mounting material
- Quick screw connectors for carrier rail mounting (2 pcs.) 680.180.05

Digital input modules
- Input module, 24 V DC, 8 inputs 680.451.01
- Input module, 24 V DC, 8 inputs, 1 ms 680.451.04
- Input module, 24 V DC, 8 inputs, with realtime clock 680.451.02
- Input module, 24 V DC, 16 inputs 680.451.03
- Input module, 24 V DC, 16 inputs, triggers interrupts 680.451.06
- Input module, 24 V DC, 16 inputs, 1 ms 680.451.07

Digital output modules
- Output module, 24 V DC, 0.5 A, 8 outputs 680.452.01

Digital input/output modules
- Input/output module, 24 V DC, 8 inputs, 8 outputs 680.450.01

Analog input modules
- Analog input module, 0...10 V, 10 bit, 4 channels 680.441.01
- Analog input module, 0(4)...20 mA, 10 bit, 4 channels 680.441.02
- Analog input module, PT100, 0...300 °C, 10 bit, 4 channels 680.441.04
- Analog input module, thermo-couple Ni-Cr-Ni (K-type)
  0...1200 °C, 10 bit, 4 channels 680.441.07
- Analog input module, potentiometer, 10 bit, 4 channels 680.441.05

Analog output modules
- Analog output module, 0...10 V, 8 bit, 4 channels 680.442.01
- Analog output module, 0(4)...20 mA, 8 bit, 4 channels 680.442.02

Analog input/output modules

Analog input/output module, 2 I 0...10 V, 2 O 0...± 10 V 680.441.03
Analog input/output module, 2 I 0...20 mA, 2 O 0...±10 V 680.441.06
Analog input/output module, 2 I 0...10 V, 2 O 0...20 mA 680.441.08
Analog input/output module, 2 I 0...20 mA, 2 O 0...20 mA 680.441.09

Counter modules

Counter module, 1 multi-function counter, 24 bit 680.454.01
Counter module, 2 multi-function counters, 24 bit 680.454.02
SSI module, 2 absolute value devices, 24 bit 680.454.04

Communication modules

V.24 module 680.440.01
TTY module 680.440.02
RS485 module 680.440.03
SE_680I communication program 680.505.01

Positioning modules

Stepping motor module, 1 channel 680.444.01
Stepping motor module, 2 channels 680.444.02

Instruction manuals

Module of KUAX 680I and 680C (English) ..... E 326 GB
KUBES 4, User software (English) ..... E 327 GB

C. Literature and trademarks

C.1. References to literature

Instruction Manual E 326 GB, Modules of KUAX 680I and 680C
Kuhnke GmbH, Malente

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

Instruction Manual E 386 GB, KUBES Modules
Kuhnke GmbH, Malente

C.1.Trademarks

IBM
is a registered trade mark of the International Business Machines Corporation

MS-DOS
is a registered trade mark of the Microsoft Corporation

EPSON
is a registered trade mark of the Epson Corporation

D. Reactions to failures

The KUAX 680C monitors itself. Any occurring errors or failures are reported and lead to reactions in the control according to their dangerousness.
The errors and failures are numbered from 1 through max. 255. They can be indicated in several ways:

Failures overview

<table>
<tr><th colspan="2">Failure</th><th colspan="3">Type of indication</th></tr>
<tr><th>No.</th><th>Type</th><th>failure LED</th><th>Error byte ERR00.00</th><th>Interrupt module [no.]</th></tr>
<tr><td>1</td><td>short circuit (on output, mMotor...)</td><td rowspan="3">yes</td><td rowspan="2">yes</td><td>18</td></tr>
<tr><td>2</td><td>undervoltage</td><td>17 *1)</td></tr>
<tr><td>3</td><td>watchdog</td><td>no</td><td rowspan="13">-</td></tr>
<tr><td>4</td><td>-</td><td></td><td rowspan="4">yes</td></tr>
<tr><td>5</td><td>-</td><td></td></tr>
<tr><td>6</td><td>-</td><td></td></tr>
<tr><td>7</td><td>wrong module configuration</td><td rowspan="3">yes</td></tr>
<tr><td>8</td><td>checksum in the user program</td><td rowspan="4">no</td></tr>
<tr><td>9</td><td>hierarchy error</td></tr>
<tr><td>10</td><td>-</td><td></td></tr>
<tr><td>11</td><td>overtemperature in the device</td><td rowspan="3">no</td></tr>
<tr><td>12</td><td>short circuit (1) removed</td><td>0</td></tr>
<tr><td>13</td><td>voltage (2) ok again</td><td>0</td></tr>
<tr><td>14</td><td></td><td></td><td></td></tr>
<tr><td>15</td><td></td><td></td><td></td></tr>
<tr><td>16</td><td></td><td></td><td></td></tr>
<tr><td>17</td><td></td><td></td><td></td></tr>
<tr><td>18</td><td></td><td></td><td></td></tr>
</table>

*1) The interrupt module is only called up in case of undervoltage

Legend for the failures overview

LED "failure"

The red light emitting diode is located on the left side of the device. It flashes in a rhythm that indicates the failure number:

No.	Flash rhythm
1	
2	
3	
4	
etc.	

The counting impulses follow each other in a short sequence (250/250 ms). Then there is a break (1 s) whereupon the counting impulses are repeated.

<u>Exception:</u> If the switch "normal program - download monitor" is in the download position (position "R", see chapter "3.1.1. Top view"), LED "failure" is permanently on.

Error byte "ERR00.00"

The error number is written into an error byte (ERR00.00). It can be analysed in the user program:

Example:
```
L       ERR00.00
C8T1    O00.00  ;binary indication
                ; via 8 outputs
```

Interrupt module [no.]

The failure triggers an interrupt (IRQ). This causes the monitor to immediately call up the assigned interrupt module.

☞ *In the following sections, individual types of failures are described and suggested reactions are explained.*

D.1. Short circuit on an output (failure #1)

Cause

Short circuit
Overload

Indication

"failure" LED flashes
KUBES reports the failure in plain text
Error byte "ERR00.00" is set to 1
Event notification on the PROFIBUS

Reaction

the corresponding output is switched off thermally
interrupt module no. 18 is activated. Use this module to program the desired reactions of the controller: :

Example:
```
O_OFF          ;switch all outputs off
=1      Mxx.xx;set marker
```

the program run is continued, only the outputs are switched off externally (their internal status remains unaltered, however; the LEDs are switched off too, though)

Remedy

remove short circuit and then

<u>either</u>

switch the outputs back on again via the program (do not include this in the program of the interrupt module as this is only activated once when the failure occurs).

Example:
```
L       Mxx.xx  ;outputs switched off?
JPCN    RETURN  ; jump if not
L       Iyy.yy  ;input "SC removed"
JPCN    RETURN  ; jump if not
O_ON            ;switch outputs on
=0      Mxx.xx  ;reset marker
RETURN ....     ;normal program run
```

- all internally set outputs are switched back on, the program run is continued
"failure" LED extinguishes
error byte "ERR00.00" is reset

<u>or</u>

restart the controller:
via the hardware: switch the supply off and back on again
via the software: KUBES RESET command followed by the RUN command

D.2. Undervoltage (supply, failure #2)

Supply voltage: 24 V DC - 20%/+25%
A built-in voltage monitoring reacts in two steps to falling below certain limiting values:

1st Step

Cause

Supply voltage approx. < 19 V

Reaction

interrupt module no. 17 is activated
the program run is not yet interrupted

☞ *Buffered operands (markers, timers and counters) can be reset unvolutarily if the user program is processed further in this phase. The cause for this would be that, because of the undervoltage, inputs could perhaps recognize a 0 signal already.*

Indication

"failure" LED flashes
Error byte "ERR00.00" is set to 2

2nd Step

<u>1st alternative</u>

Cause

Supply voltage rises back to 24 V DC ± 20%

Reaction

"failure" LED extinguishes
Error byte "ERR00.00" is reset
The program is continued without interruption

<u>2nd alternative</u>

Cause

Supply voltage continues to fall, approx. < 17.5 V

Reaction

5 V system voltage is interrupted
=> - STOP: the program run is stopped
- RESET: outputs, error byte (ERR00.00) and unbuffered markers, timers and counters are reset
- all LEDs off

Remedy

as a precaution, in the user program, to save buffered operands:
Processing of the user program should be interrupted until step #2 of the cause has been reached or until a realistic waiting time has been exceeded.

program example for interrupt module no. 17:

```
        WAIT    5           ;wait 5 * 10 ms = 50
ms
        L       ERR00.00    ;scan error byte
        CMP     2           ;still undervoltage?
        JP<>    RETURN      ; jump if not
        RESET               ; else RESET and Stop
RETURN  NOP                 ;continue program run
```

☞ *The WAIT command starts a program loop whose length is entered in n * 10 ms. Should this time be longer than approx. 70 ms a watchdog error is recognized. This then switches the controller off. Practical waiting times are therefore approx. 50 ms max.*

D.3. Watchdog (program run time exceeded, failure #3)

Cause

run time of a module > 50...70 ms
or
run time of the overall program > 2 s

Indication

"failure" LED flashes
KUBES reports the failure in plain text
Event notification on the PROFIBUS

Reaction

STOP: program run is stopped
RESET: outputs and unbuffered markers, timers and counters are reset

Remedy

change the program architecture to achieve shorter run times
restart the controller:
via the hardware: switch supply off and back on again, or
via the software: KUBES RESET command followed by the RUN command

D.4. Checksum in the user program (failure #8)

During program generation, a checksum (CS) is generated over the entire user program memory according to a certain algorithm.

Cause

When starting the controller, the monitor re-calculates the checksum and compares it to the stored value. If the result is inequal, the monitor recognizes a failure.

Reaction

the controller does not start

Message

"failure" LED flashes
"Stop" LED lights up
Error byte "ERR00.00" is set to 8

Remedy

determine the cause of the failure and remove it
use the Transmit command of the KUBES PLC menu to tansmit the project again to the controller

D.5. Hierarchy error (failure #9)

Program calls and other module calls must not exceed certain hierarchy limits (see chapter "4.7. Module programming").

☞ *During programming, the controller reports a hierarchy error when it receives a program. At this stage, this is only a warning that there could be an error. Only at start-up does the controller check whether there is really a hierarchy error. If there is not, the error indication disappears again.*

Cause

When switching the controller on or after the KUBES start command, the monitor program checks whether there is a hierarchy error (a module calls up the module by which itself was called up or the nesting depth exceeds 5 levels)

Reaction

the controller does not start

Indication

"failure" LED flashes
"Stop" LED is on
Error byte "ERR00.00" is set to 9

Remedy

determine the cause of the error and remove it
use the Transmit command of the KUBES PLC menu to tansmit the project again to the controller

E. Versions

We will continue to develop the KUAX 680C further. At various stages of development we will release new versions.

E.1 Hardware

Laboratory sample (produced before calendar week 28/95)
First version, delivered with preliminary release:
In the following points, the hardware differs from the standard as planned:

connectors for RS 232/1 and RS 232/2: the plugs are in reverse order;
changeover switch "Normal program" / "Load monitor" (see chapter "3.1.1. Top view", pos. 8) is not there yet but a jumper instead;
the positions for the connectors of the alarm output and the RS 485 interface (see chapter "3.1.2. Front view", pos. 17 and 18) are in reverse order;
the marking of the casing is not yet complete.

Laboratory sample (produced as from calendar week 34/95)
Second version, delivered with preliminary release:
In the following points, the hardware differs from the standard as planned:

connectors for RS 232/1 and RS 232/2: the plugs are in the right order now;
changeover switch "Normal program" / "Load monitor" (see chapter "3.1.1. Top view", pos. 8) is now there;
the positions for the connectors of the alarm output and the RS 485 interface (see chapter "3.1.2. Front view", pos. 17 and 18) are now in keeping with the illustration;
improved marking of the device casing; however, the labelling of the LEDs of the counter inputs and the interrupt inputs are reversed (see chapter "3.1.1. Top view, pos. 4).

E.2. Software (monitor program)

The monitor program of the KUAX 680C is stored in the Flash-EPROM. This has the great advantage that new versions can be transferred into the controller easily by using a PC and the required program. Hardware changes are no longer necessary.

Monitor version 4.22
First released version.

Index

Symbols

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X