# Kuhnke Electronics
# Instruction Manual

## Eco Control 667E
## Small compact PLC

E 556 GB 3 December 1998 / 77.931

# Table of contents

# 1 Introduction

Eco Control 667E is a small high-performance PLC. Due to its compact design it is well-suited for all applications that expect a lot of "functionality" from a small machine.

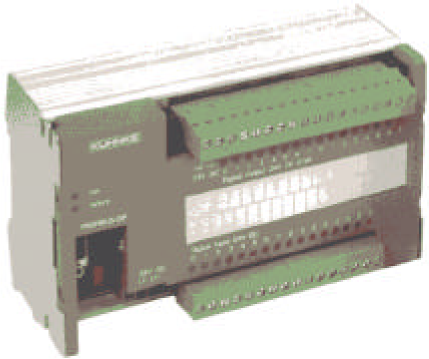


*Fig. 1: Eco Control 667E 16/16*

## 1.1 Features

➢ Easy installation due to the integrated snap-on device for carrier rails.

➢ Program and data memories are located in the built-in NV-RAM (non-volatile RAM).

➢ Program and remanent operands are permanently stored without any energy from outside (battery or accumulator).

➢ Set of operands:
- Inputs: 8, 16, 32 (depending on model)
– Outputs: 8, 16, 32 (depending on model)
– Bit markers: 1320, inc. 512 remanent markers
– Byte markers: 2816, inc. 2304 remanent markers
– Timers: 32, 10 ms...65535 min, quartz-precision
– Counters: 32, 0...65535

➢ Programming via PC, MS®Windows und KUBES

## 1.2 Successor to Pico/Compact Control KUAX 667

Eco Control 667E is the legitimate replacement for “Pico/ Compact Control KUAX 667“.

Apart from its software being compatible with the older types it also features a couple of major improvements:

- The device is smaller although its performance is the same.
- Installation is easier due to the integrated snap-on device.
- Modern manufacturing techniques ensure that you get a lot more value for more money.
- A plug-type memory module is no longer required because the program is stored in the built-in NV-RAM.
- No battery or accumulator because the NV-RAM safely stores programs and data.
- The controller is CE-certified.

# 2 Reliability, safety

## 2.1 Target group

This instruction manual contains all information necessary for the use of the described product (control device, control terminal, 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 agreements with our sub-suppliers,
- measures for the prevention of static charge during the handling of MOS circuits,
- worst case planning and design of all circuits,
- inspections during various stages of fabrication,
- computer aided tests of all assembly groups and their coefficiency in the circuit,
- statistical assessment of the quality of fabrication and of all returned goods for immediate taking of corrective action.

## 2.3 Notes

Despite the measures described in chapter 2.2, the occurrence of faults or errors in electronic control units - even if most highly improbable - must be taken into consideration.

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 humans or material/equipment.

### 2.4.1 Observe during planning and installation

- 24V DC power supply: Generate as electrically safely separated low voltage. Suitable devices are, for example, split transformers constructed in compliance with European standard EN 60742 (corresponds to VDE 0551).
- In case of power breakdowns or power fades: the program is to be structured in such a way as to create a defined state at restart that excludes dangerous states.
- Emergency switch-off installations must comply 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 faults.
- Relevent standards and VDE regulations are to be observed in every case.
- Control elements are to be installed in such a way as to exclude unintended operation.
- Control cables are to be layed in such a way as to exclude interference (inductive or capacitive) which could influence controller operation or its functionality.

*To achieve a high degree of conceptual safety in planning and installing an electronic controller it is essential to ex-*

*actly follow the instructions given in the manual because wrong handling could lead to rendering measures against dangers ineffective or to creating additional dangers.*

### 2.4.2 Observe during maintenance or servicing

➢ Precautions regulation VBG 4.0 must be observed, and section 8 (Admissible deviations during working on parts) in particular, when measuring or checking a controller in a power-up condition.

➢ Repairs must only be made by specially trained Kuhnke staff (usually in the main factory in Malente). Warranty expires in every other case.

➢ Spare parts:

➢ Only use parts approved of by Kuhnke. Only genuine Kuhnke modules must be used in modular controllers.

➢ In the case of modular systems: modules are to be dead when plugging or unplugging them. They may otherwise be destroyed or their functionality adversely affected, the latter without you necessarily noticing immediately.

➢ Dispose of any 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 defined 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 acc. with EN 61000-4-2, 3rd degree of sharpness
- Irradiation resistance of the device, HF
in acc. with EN 61000-4-3, 3rd degree of sharpness
- Fast transient interference, burst
in acc. with EN 61000-4-4, 3rd degree of sharpness
- Immunity to damped oscillations
in acc. with EN 61000-4-12 (1 MHz, 1 kV)

## 2.5.3 Interference emission

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

☞ *If the controller is designed for use in residential areas, then high-frequency emissions must comply with limiting value class B as described in EN 55011.*
*Fitting the controller into an earthed metal cabinet and equipping the supply cables with filters are appropriate means for maintaining the relevant limiting values*

## 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 current loops:

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

Joint laying of control current loops is allowed for:

- shielded data signals
- shielded analogue signals
- unshielded digital I/O lines
- unshielded DC voltages < 60 V
- unshielded AC voltage < 25 V

### 2.5.7 Location of installation

Make sure that there are no impediments due to temperatures, dirt, impact, vibration 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 vibration

Consider possible influences caused by motors, compressors, transfer lines, 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 relays, contactors, solenoids or switching magnets) produces overvoltages. It is necessary to reduce these extra voltages to a minimum. Reducing elements may be diodes, Z-diodes, varistors or RC elements. To find the best adapted elements, we recommend that you contact the manufacturer or supplier of the corresponding actuators for the relevant information.

# 3 Hardware

Eco Control 667E is a compactly built controller in a housing with an integrated snap-on device for installation on carrier rails

Inputs and outputs are connected to it by means of screw-type locking terminals. A female 9-pin D-Sub connector serves as the interface for communication with programming PCs or other devices such as dialogue terminals.

## 3.1 Model variants

The different variants vary in their I/O configuration.

- Eco Control 667E 8/8
  8 digital inputs
  8 digital outputs
  1 serial interface (V.24)
- Eco Control 667E 16/16
  16 digital inputs
  16 digital outputs
  1 serial interface (V.24)
- Eco Control 667E 32/32 (in preparation)
  32 digital inputs
  32 digital outputs
  1 serial interface (V.24)

## 3.2 Top view

This view tells you where the connectors and light emitting diodes (LEDs) are located on the device.

### 3.2.1 Eco Control 667E 8/8

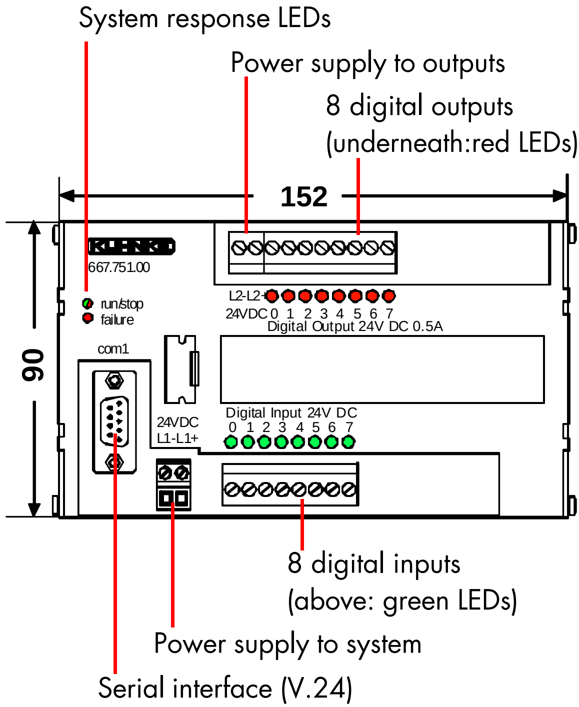



*Fig. 2: Top view of Eco Control 667E 8/8*

### 3.2.2 Eco Control 667E 16/16

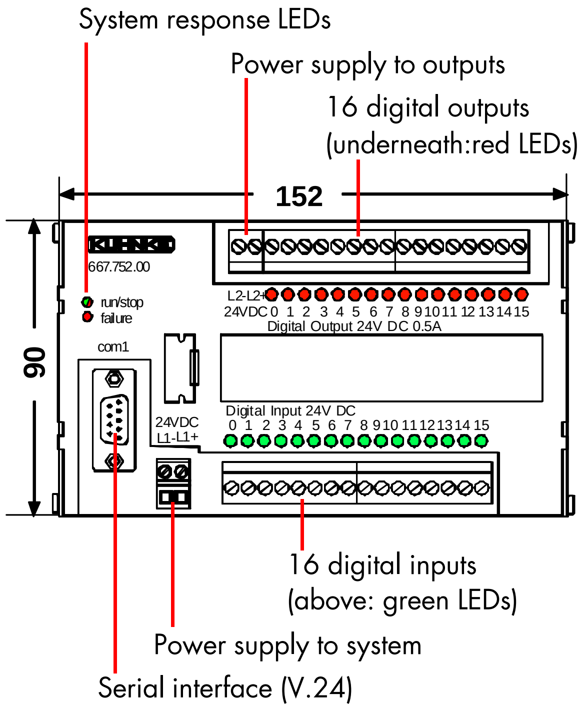



*Fig. 3: Top view of Eco Control 667E 16/16*

## 3.2.3 Eco Control 667E 32/32

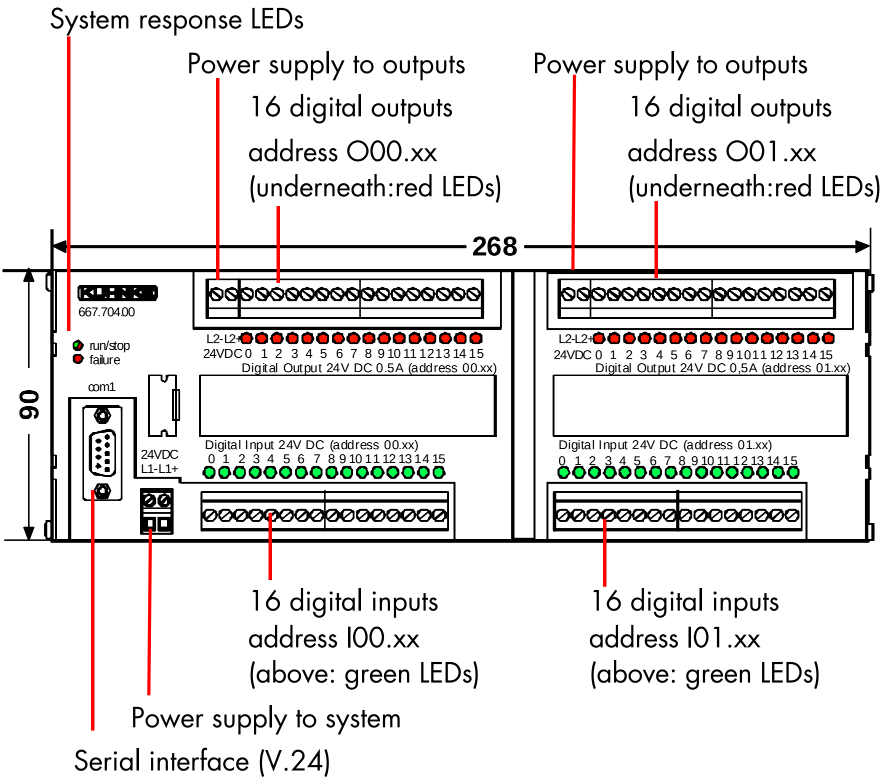



*Fig. 4: Top view of Eco Control 667E 32/32*

## 3.3 Mechanical design

The housing mainly consists of an aluminium profile with an integrated snap-on device for installation on carrier rails. The side walls of galvanised sheet metal steel are riveted to the aluminium profile. The hooks of the plastic cover snap into the appropriate holes in the side walls.

## 3.3.1 Installation

Eco Control 667E is designed for installation on carrier rails (in acc. with DIN EN 50022, 35 x 7.5 mm).

### Procedure

1. Push the device against the carrier rail such that the metal spring snaps into the space between carrier rail and installation surface (see illustration).

2. Push the device up against the installation wall until it snaps in.

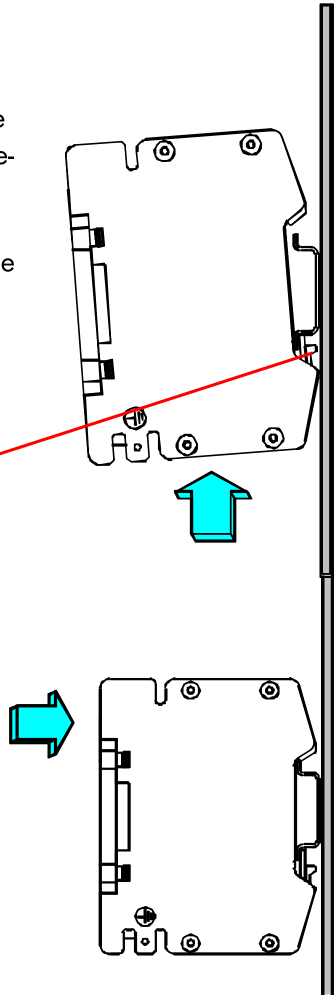



*Fig. 5: Installation on carrier rail*

## 3.3.2 Earthing

The metal housing is to be connected to earth. Each side wall has an earthing connector integrated into it (see arrow in illustration):

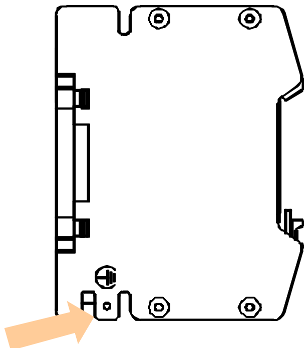


*Fig. 6: Earthing connector*

- Type of connector
  Connect plain plug 6.3 x 0.8 mm (fast-on) to at least 1 of the side walls
- Earthing lead
  Diameter: min. 2.5 $mm^2$
  Length: as short as possible
- Function: earth connection of functions.

  *No protection against high contact voltage. To ensure the protective function, make sure that the devices are supplied with safely separated small voltages (see chapter 2.4.1).*
- The casing of the COM1 connector for the serial port directly connects to the earth connection of functions. This is where you attach the cable shielding.
- The connectors for +24V DC and 0V supply are internally (by spring contacts on the PCB) and capacitively connected to the housing and, thus, to earth. High-frequency interference is conducted to earth via this channel.

## 3.4 Power supply

System and outputs are supplied via separate connectors (for the location of the connectors see chapter 3.2). This allows you to switch off all outputs without having to disconnect the controller from its power source.

*To ensure uninterrupted operation, lay the supply cables separately, using the shortest possible cables to connect the power source with the controller's supply terminals.*
*If you are using two different power source, you are obliged to equalise the potential between the 0V connectors.*

### 3.4.1 System power supply

The system power supply connects to a 2-pin plug-type terminal block.

- Connectors: L1- → 0V
  L1+ → +24V DC
- Voltage: 24 V DC -20%/+25%
- Power consumption: c. 100 mA

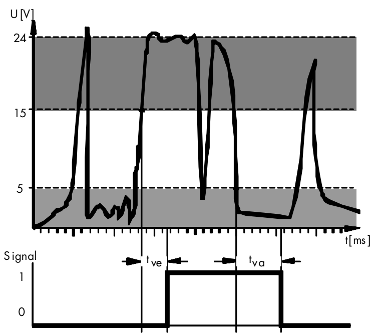
The outputs are supplied separately. However, potentials of system and output supplies are not separated.

*For a description of the output supply voltage connector refer to chapter 3.6.*

## 3.5 Digital inputs

The inputs pick up the digital signals of a variety of sources. They connect to the device by screw-type locking terminals (→ illustrations in chapter 3.2). Make sure that they work within the switching thresholds indicated below, which particularly applies to proximity switches and semiconductor sensors. The input circuitry adapts the incoming signals to the system voltage.

- Defined signals and switching thresholds
  Logical 0 ≤ 5 V
  Logical 1 ≥ 15 V
  (Hysteresis 1...4 V)
- Signal delay
  Peak voltages (noise impulses) are filtered to avoid them being considered as valid signals which might trip unintended switching actions. This delays signal detection by rated 5 ms:




Raising delay:
$t_{ve}$ = 3.0...7.0 ms
Falling delay:
$t_{va}$ = 4.0...7.0 ms

*Fig. 7: Input signal delay*

*Input signals are read between program cycles and written into the process image . To calculate the average availability of signals to the user program, you must therefore add the program cycle time to the specified delays.*

## 3.6 Digital outputs

Digital outputs are the connection to external actuators (relays, contactors, solenoids, valves...). They connect to the controller by screw-type locking terminals (→ illustrations in chapter 3.2). You can control resistive and inductive loads. Free-wheeling diodes suppress inductive switch-off peaks. The output status is indicated by LEDs.

### Output power supply

The output power supply connects to a 2-pin plug-type terminal block (→ illustrations in chapter 3.2).

- Connectors: L2- → 0V
  L2+ → +24V DC
- Voltage: 24 V DC -20%/+25%
- Power consumption: depends on the load on the outputs

### Protection against short circuit- and overload

Outputs are protected against destruction by overload or short circuit. In case of a fault, all outputs are disabled and the "failure" LED flashes (→ 7.1)

## 3.7 Serial interface COM1

The serial interface mainly provides a connection to programming PCs. Apart from that it can also be used for communication with other devices such as dialogue terminals, for example.

- Type
  V.24 (RS 232)
- Connector
  female 9-pin D-Sub connector
- Pin wiring
  2 TxD
  3 RxD
  5 Gnd
- Cable shielding
  connects to the plug's frame ground.

*The metal connector casing is directly connected to the frame and, thus, to earth if the device is properly earthed (see chapter 3.3.2).*

## 3.8 Light emitting diodes

Two LEDs indicate the system status:

- run/stop
  lights up green while the PLC program is running
  lights up red when the PLC program stops
- failure
  flashes red if there is a short at an output

## 3.9 Processor

The core unit of the controller is its single-chip microprocessor, type 80C535. It gets its commands from the monitor program and the user program (→ 3.10).

### 3.9.1 On-chip RAM

The microprocessor features an integrated on-chip RAM (→ 3.10.5) which allows very fast accesses.

## 3.10 Memory distribution

The controller has four types of memory:
Flash EPROM, NV-RAM, SRAM and on-chip RAM.

### 3.10.1 Operating system

The operating system is stored in the flash EPROM. It contains the system software and is loaded at the Kuhnke factory before delivery. Users cannot directly access this type of memory.

### 3.10.2 User program

The user program is safely stored in the NV-RAM (→ 3.10.4). The device reserves 32 kbyte for the user program.

By default, the user program is stored in machine language code and also in KUBES' intermediate code. The latter allows KUBES to retrieve the program from the controller.

## 3.10.2.1 Disable retrievability = increase capacity

Storing the intermediate code in memory can be disabled by writing into operand SLG14.05 via the user program. There are two effects:

- The capacity of the program memory is increased.
- The program is secured against unauthorised access because it can no longer be disassembled.

| | | |
|---|---|---|
| SLG14.05 | = 0 | enable intermediate code storage |
| | <> 0 | disable intermediate code storage |

Before the program is transmitted to the controller, KUBES (version 5.30 or higher) displays the following dialog:

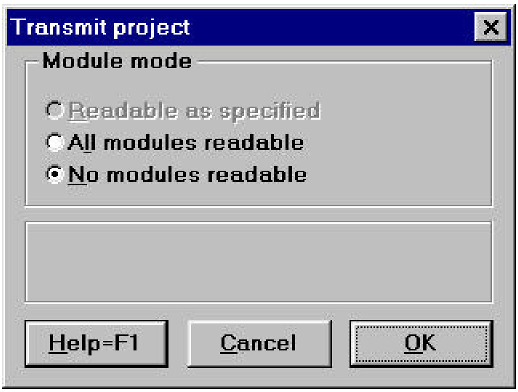



*Fig. 8: Program retrievability settings in KUBES*

KUBES automatically writes into SLG14.05

- Setting "All modules retrievable": [SLG14.05] ← 0
- Setting "No module retrievable": [SLG14.05] ← 255

## 3.10.3 Data memory

Data in this context comprises all operands (inputs and outputs, bit markers, byte marker, timers and counters). The monitor program also falls back on some parts of this memory for internal purposes.

8 kbyte of the NV-RAM described in chapter 3.10.4 are reserved for no-voltage protected operands (also called remanent operands).

The volatile (non-remanent) operands are stored in a 24 kbyte S-RAM. This type of memory is cleared when the device is being initialised to ensure that all memory cells have a defined status (0).

Inputs and output (Ixx.xx and Oxx.xx) as well as 40 markers (M00.00...M02.07) are mapped onto the microprocessor’s so-called on-chip RAM. These addresses can be accessed particularly fast (→ 3.10.5).

## 3.10.4 NV-RAM: special features

NV-RAM technology (non-volatile RAM) ensures that programs and data are stored safely without the use of external energy (accumulator or battery) even if you disconnect the device from the mains. They’re stored without any limitation in time no matter how long the device remains switched off for. They resume their previous status when you restart the controller.

### 3.10.5 On-chip RAM: special features

The on-chip RAM is part of the microprocessor. It can be addressed by individual bits. Accesses to this type of memory are about twice as fast as accesses to the external types of memory, i.e. S-RAM and NV-RAM. The on-chip RAM is therefore fully occupied. Addresses are assigned to the following operands:

- 32 inputs (I00.00...I01.15)
- 32 outputs (O00.00...O01.15)
- 40 markers (M00.00...M02.07)

# 4 Software

## 4.1 Operative approach

The microprocessor receives its program from two program memories:

- the memory containing the operating system
- the memory containing the user program

### Operating system memory

It contains the operating system and all system features of Eco Control 667E. It is permanently installed in the device (→ 3.10.1).

### User program memory

It contains the programs required for controlling the machine or system. The programs are written in KUBES, Kuhnke programming software package. The user program memory is permanently installed in the device (→ 3.10.2).

The next chapters only detail the knowledge you need to write user programs for Eco Control 667E.

The method of how to actually input the program is not explained. For a description refer to:

*Instruction manuel KUBES, E327GB*

## 4.1.1 PLC cycle

As a typical PLC, Eco Control 667E cyclically processes the user program in the program memory.

### Cycle time

The controller's overall action in time is indicated by the cycle time which is influenced by a variety of factors:

- command execution time
- length and structure of the program
- monitor functions
- self-test functions
- KUBES functions

#### 4.1.1.1 The 4 phases of a PLC cycle

- Update process image
  The status of the inputs is read and written into an internal RAM range (operand range I00.00 ...). The program uses these values in the next cycle.
  Exception:
  Operations with byte input markers BIxx.xx immediately read the inputs without waiting for the next update of the process image.
- Process program
  Program processing always starts with the first line of the ORG module and ends with the last line of the ORG module (see example "structured programming"). The calculated values (assignments) are written into the process image memory.
- Update outputs
  The output markers are copied to the outputs only at the end of a complete program processing cycle. Thus, even if the outputs have been changed by the program several times, only the last status will be output to the relevant terminal.
  Exception:
  Assignments to byte output markers BOxx.xx immediately write their result into the output memories without waiting for the process image to update the outputs.
- Internal PLC action
  In certain cases, the CPU has to respond to requests that are required for self-testing or for communication with the programming PC.

## 4.1.1.2 Minimum cycle time

The time it takes to complete a PLC cycle is shortest if the PLC just processes the program.

### Calculating the cycle time

- Sum total of execution times of module call commands
- Sum total of command execution times

(see table Set of Commands)

- Process image update: 25 μs

However, due to the possibility of using conditional module calls and conditional jump commands (JPC...), the cycle time also depends on the internal and external states used as conditions.

This gives the programmer the chance to optimise the program runtime by cleverly arranging his program.

A clear project structure ensures that the PLC is only engaged in operations that are relevant to the control process at that time.

Another benefit ensues from storing the most frequently used bit operands in the on-chip RAM, because accesses to this memory are twice as fast as accesses to other types of memory (→ 3.10.5).

## 4.1.1.3 Influence of timer interrupts on the cycle time

The programmable timers depend on highest precision.

This is ensured by a quartz crystal and the relevant frequency dividers that generat impulses which, in turn, generate interrupts at the intervals set by the programmable timers (10 ms, 100 ms, 1 s, 10 s).

If the timers are enabled, these timer interrupts lead to the current time values being incremented or decremented which means that the timer outputs may have to be adjusted. This is added to by the updating of the clock pulse markers (C00.00-03).

Processing of the current program is therefore to be interrupted, the contents of the CPU registers is to be saved and stored for continuation later.

### 4.1.1.3.1 Extension of the cyle time

The amount of time by which the cycle is extended due to the handling of timer interrupts depends on the number of currently active programmable timers.

#### Worst case

Every 10 ms, the PLC cycle is extended by c. 2 ms if all of the 32 possible timers have been programmed as clock pulses with the same time on the basis of 10 ms and if they are all enabled.

#### Best case

In the best case, the cycle time is extended by only 0.4 ms.

## 4.1.1.4 Influence of communication on the cycle time

Both programming and testing online in the KUBES environment and man-machine communication with operating terminals demand data exchange via the V.24 port. this communication is interrupt-controlled and can extend the cycle time by up to 10%, given a transfer rate of 9600 baud.

### Status information

In certain intervals, KUBES requests information from the controller even if there is no actual communication:

Frequency: every 5 s

Delay: 1 ms

### Dynamic displays

The single address and address range displays, the logic diagram or the dynamic display in the Module Editor allow you to permanently read and display up to 256 operands. You can reduce the resulting time load by either loading fewer program lines into the Module Editor or by using the single address instead of the address range display

Examples

➢ The dynamic display of 224 byte (14 lines with C1T16 SMxx.xx) in the Module Editor can extend a program cycle of 2 ms by another 2 ms (worst case).

➢ Having the address range display dynamically reading a complete marker range extends the cycle time by c. 0.5 ms.

### 4.1.1.5 Changing the program in run mode, transmitting a module

The user program can be changed without interrupting the program run. When you transmit a changed module, the controller needs some time to receive, interpret and insert the module as well as to calculate its checksum.

Extension of cycle time: c. 10 %

Duration: depends on the length of program and the cycle time

### 4.1.1.6 Restarting the controller after changes in Stop/ Reset mode

Modifying the program memory while the controller is in Stop or Reset mode also modifies the checksum required for the memory test. The checksum is calculated when you restart the controller (RUN). The controller will resume run mode only when the checksum test has been completed successfully.

## 4.1.1.7 Programming

For programming and testing, connect the programming PC with the appropriate interface as described in chapter 3.7.

### Requirements

➢ PC running MS®Windows

➢ Programming software KUBES (part no. 680.502.00) installed on the PC

➢ Programming cable (part no. 657.151.03)
– Connect with the PC's COM port as specified in KUBES (→ below, Fig. 9)
– Connect with the PLC's programming interface

### Data transfer rate

The transfer format is set to: 9600 bit/s,
8 data bits, 1 start bit, 1 stop bit, odd parity

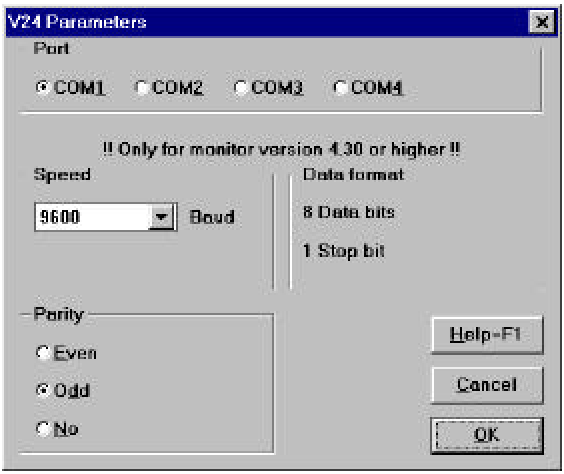



*Fig. 9: KUBES interface parameters*

## 4.2 Operand ranges

All addresses used by the program for signal processing or data storage are called operands. They are “operated” with.

Eco Control 667 provides a large number of operands. Please refer to the table in chapter 4.2.2.

## 4.2.1 Definitions

### Inputs

Signals that are fed into the controller and read by the user program.

### Outputs

Signals that are generated by the program in the controller and picked up externally as control signals. They switch on lamps, drives etc.

### Markers

Signals that are used inside the controller for storing states and supporting complex logical operations. There are two types of markers:

- bit markers (1-bit signals) and
- byte marker (8-bit signals).

### Timers

They control time processes.

### Counters

They count events or increments output by pulse generators.

## 4.2.2 Summary of operands

| *Operands* from | *Operands* to | *Name* | *Max. qty.* | *Bits* | *Description* |
|---|---|---|---|---|---|
| I00.00 | I01.15 | Digital inputs | 32 | 1 | Max. I/O configuration depends on the model variant (→ 3.1).<br>The process image of inputs and outputs is stored in the on-chip RAM (→ 3.10.5), therefore fast access |
| O00.00 | O01.15 | Digital outputs | 32 | 1 | |
| BI00.00<br>BO00.00 | BI00.03<br>BO00.03 | Byte inputs<br>Byte outputs | 4<br>4 | 8<br>8 | For reading inputs directly by byte (without process image).<br>For writing outputs directly by byte (without process image). |
| M00.00 | M02.07 | Fast bit markers | 40 | 1 | Bit markers in the on-chip RAM (→ 3.10.5), therefore fast access |
| SM00.00<br>FM00.00<br>LM00.00 | SM15.15<br>FM15.15<br>LM15.15 | Bit markers | 256<br>256<br>256 | 1<br>1<br>1 | Bit markers. Divided into groups of 256 for better differentiation. |
| R00.00<br>SR00.00 | R15.15<br>SR15.15 | Remanent bit markers | 256<br>256 | 1<br>1 | Bit markers, stored remanently in the NV-RAM (→ 3.10.4) |
| BM00.00<br>SBM00.00 | BM15.15<br>SBM15.15 | Byte markers | 256<br>256 | 8<br>8 | Byte markers. Divided into groups of 256 for better differentiation. |
| BR00.00<br>SBR00.00<br>BC00.00<br>SBC00.00<br>BD00.00<br>SBD00.00<br>FBM00.00<br>LBM00.00<br>ZBM00.00 | BR15.15<br>SBR15.15<br>BC15.15<br>SBC15.15<br>BD15.15<br>SBD15.15<br>FBM15.15<br>LBM15.15<br>ZBM15.15 | Remanent byte markers | 256<br>256<br>256<br>256<br>256<br>256<br>256<br>256<br>256 | 8<br>8<br>8<br>8<br>8<br>8<br>8<br>8<br>8 | Byte markers, stored remanently in the NV-RAM (→ 3.10.4) |
| PL00.00<br>PL00.01 | | Logical 0<br>Logical 1 | 1<br>1 | 1<br>1 | Programmed logical signals, changing not possible. |
| PC00.00 | PC00.03 | Clock pulse marker | 4 | 8 | Byte operands, incremented at pulse rates 10 ms, 100 ms, 1 s, 10 s. |
| PP00.00 | PP07.15 | Progr. pulse | 128 | 1 | Evaluates the 0/1 changeover (edges) of digital signals. |
| PT00.00 | PT01.15 | Timers | 32 | 16 | Programmable, range: 10 ms – 65535 s |
| C00.00 | C01.15 | Counters | 32 | 16 | Programmable, range: 10 – 65535 |
| SLF, SLG... | | Special functions | | 8 | Partly reserved for monitor, KUBES modules, additional modules |

## 4.2.3 Set operand functions

When you are planning your project, please take into account that some of the operands listed above have set functions:

### 4.2.3.1 Operands reserved for monitor functions

| *Operand* | *Function* |
|---|---|
| SLG14.00 | Internal use |
| SLG14.01 | Undervoltage monitoring (n * 10 ms) |
| SLG14.02 | Reads inputs in case of undervoltage |
| SLG14.03 | Internal use |
| SLG14.04 | Internal use |
| SLG14.05 | Generates the intermediate code |
| SLG14.06 | Transmitting projects: retains remanent data |

*These operands must not be used for any other purposes. Failure to obey may render the controller functions unsafe.*

### 4.2.3.2 Operands reserved for KUBES modules

Eco Control 667E has no KUBES module parameters. Individual KUBES modules use defined operands that you should reserve in case you wish to use these operands.

| *Operand* | *Used by KUBES module* |
|---|---|
| BM00.00...03 | WR_OFFS, RD_OFFS |
| BM01.00 | |
| FBM00.00...01 | |
| FBM01.00...09 | V24667IS, V24667IE, V24667IN |

*If you wish to embed one or several of the above KUBES modules in your project, you must make sure that the relevant operands are reserved for this purpse only.*

## 4.3 Description of commands

All operations are started by commands. They are executed in the accumulator of the CPU. Basic terms:

- Load commands load a value into the accumulator
- Logical operations link the operand value with the contents of the accumulator
- Assignments write the contents of the accumulator into the specified operands (in the case of bit operations: the status of bit 7)
- Set commands set (S) or reset/clear (R) the contents of the operand if the previous operation in the accumulator results in "logical 1".

## 4.4 Types of operands

Eco Control 667E differentiates between three types of operands which are marked by their size:

- Bit 1 bit
- Byte 8 bit
- Word 16 bit (2 byte)

The accumulator in the CPU of Eco Control 667E can be used as a bit, byte or word register.

Bit operations are carried out like byte operations, the difference being that only bit 7 of the 8 bit accumulator is evaluated.

Byte operations are executed in the same accumulator as bit operations.

Word operations use a 16-bit accu whose low byte contains the accumulator where bit and byte operations are executed. Word operations are started by commands whose last character is a D (not applicable to byte inputs BIxx and byte outputs Boxx).

*To avoid mistakes we recommend that you do not use different types of operands in operations that belong together.*

## 4.4.1 Addressing

There are two different ways of assigning operand values:

- absolute value (constant)
- contents of an operands

Operand specifiers are made up as follows:

BM00.00

Group mark | Group number | Channel number

You can use the mnemonic (symbolic name) previously assigned to an operands via KUBES' Symbol Table Editor.

Complete commands (instructions) consist of a command and an operand (rare exceptional cases have no operand):

### Example

L BM00.00

Loads the contents of byte marker BM00.00 into the accu.

## 4.4.2 Summary of commands

The purpose of commands is to “operate” with the operands (see "4.2 Operand ranges").

Eco Control 667E provides a large number of commands. They are listed and described in the tables starting on the next page.

### Memory requirements of commands

Normally, the user program is stored twice in the user program memory:

- as machine code which is read by the processor;
- as intermediate code which is used for transfer actions between PC and PLC in accordance with the KUBES protocol. Storing the intermediate code can be disabled by the relevant instruction in the user program (→ 3.10.2.1).

The “No. of bytes” table columns list the memory requirements for both cases.

## 4.4.2.1 Logical operation commands

| *Command* | *Operand* | *No. of bytes with interm. code* | *No. of bytes w/o interm. code* | *Processing time[ms]* | *Description* |
|---|---|---|---|---|---|
| L | I00.00 | 5 | 2 | 2.0 | Load bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 9 | 6 | 6.0 | Load bit operand |
| | BM00.00 | 9 | 6 | 6.0 | Load byte operand (8 bit) |
| | 100 | 6 | 4 | 3.0 | Load byte constant (8 bit) |
| LD | BM00.00 | 15 | 12 | 12.5 | Load word operand (16 bit or 2 byte) |
| | 1000 | 20 | 7 | 5.0 | Load byte constant (16 bit) |
| LN | I00.00 | 6 | 3 | 3.0 | Load and negate bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 10 | 7 | 6.0 | Load and negate bit operand |
| | BM00.00 | 10 | 7 | 7.0 | Load and negate byte operand (8 bit) |
| A | I00.00 | 5 | 2 | 2.0 | And bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 15 | 12 | 12.0 | And bit operand |
| | BM00.00 | 13 | 10 | 10.0 | And byte operand (8 bit) |
| | 100 | 8 | 6 | 5.0 | And byte constant (8 bit) |
| AN | I00.00 | 5 | 2 | 2.0 | And bit operand (negated) in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 16 | 13 | 13.0 | And bit operand (negated) |
| | BM00.00 | 14 | 11 | 11.0 | And byte operand (negated) (8 bit) |
| O | I00.00 | 5 | 2 | 2.0 | Or bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 15 | 12 | 12.0 | Or bit operand |
| | BM00.00 | 13 | 10 | 10.0 | Or byte operand (8 bit) |
| | 100 | 8 | 6 | 5.0 | Or byte constant (8 bit) |

| *Command* | *Operand* | *No. of bytes with interm. code* | *No. of bytes w/o interm. code* | *Processing time[ms]* | *Description* |
|---|---|---|---|---|---|
| ON | I00.00 | 5 | 2 | 2.0 | Or bit operand (negated) in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 16 | 13 | 13.0 | Or bit operand (negated) |
| | BM00.00 | 14 | 11 | 11.0 | Or byte operand (negated) (8 bit) |
| XO | I00.00 | 13 | 10 | 8.0 | Exclusive-Or (antivalence) bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 15 | 12 | 12.0 | Exclusive-Or (antivalence) bit operand |
| | BM00.00 | 13 | 10 | 10.0 | Exclusive-Or (antivalence) byte operand (8 bit) |
| | 100 | 8 | 6 | 5.0 | Exclusive-Or (antivalence) byte constant (8 bit) |
| XON | I00.00 | 14 | 11 | 8.0 | Equivalence bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 14 | 11 | 13.0 | Equivalence bit operand |
| | BM00.00 | 14 | 11 | 11.0 | Equivalence byte operand (8 bit) |

### 4.4.2.2 Assignments and store commands

| *Command* | *Operand* | *No. of bytes with interm. code* | *No. of bytes w/o interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| = | O00.00 | 5 | 2 | 2.0 | Equal (assignment) to bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 11 | 8 | 8.0 | Equal (assignment) to bit operand |
| | BM00.00 | 9 | 6 | 6.0 | Equal(assignment) to byte operand (8 bit) |
| =D | BM00.00 | 17 | 14 | 16.0 | Equal (assignment) to word operand (16 bit) |
| =N | O00.00 | 7 | 4 | 4.0 | Equal to negated bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 13 | 10 | 12.0 | Equal to negated bit operand |
| | BM00.00 | 11 | 8 | 8.0 | Equal to negated byte operand (8 bit) |
| S | O00.00 | 7 | 4 | 3.0 | Set bit operand in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 11 | 8 | 8.0 | Set bit operand |
| R | O00.00 | 7 | 4 | 3.0 | Reset bit opernd in on-chip RAM (→ 3.10.5) |
| | SM00.00 | 15 | 12 | 12.0 | Reset bit operand |

*Please also read the explanatory notes on the next page.*

## Notes on assignments and store commands

- Assignments (=...)

Assignments write the contents of the accumulator into the specified operand.

- Set command (S)

Writes “logical 1” into the specified operand if the preceding operation in the accu resulted in “logical 1”. There is no influence on the operand if the result in the accu was “logical 0”.

- Reset command (R)

Writes “logical 0” into the specified operand if the preceding operation in the accu resulted in “logical 1”. There is no influence on the operand if the result in the accu was “logical 1”.

### 4.4.2.3 Arithmetical operation commands

| *Command* | *Operand* | *No. of bytes with Interm. code* | *No. of bytes w/o Interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| ADD | BM00.00<br>100 | 11<br>6 | 4<br>8 | 8.0<br>3.0 | Add byte operand<br>Add byte constant |
| ADDD | BM00.00<br>1000 | 28<br>20 | 25<br>17 | 26.0<br>18.0 | Add word operand<br>Add word constant |
| SUB | BM00.00<br>100 | 13<br>7 | 10<br>5 | 8.0<br>3.0 | Subtract byte operand<br>Subtract byte constant |
| SUBD | BM00.00<br>1000 | 30<br>21 | 27<br>18 | 28.0<br>18.0 | Subtract word operand<br>Subtract word constant |
| MUL | BM00.00<br>100 | 12<br>9 | 9<br>7 | 11.0<br>7.5 | Multiply byte operand<br>Multipliy byte constant |
| MULD | BM00.00<br>1000 | 18<br>16 | 15<br>13 | variable<br>variable | Multiply word operand<br>Multiply word constant |
| DIV | BM00.00<br>100 | 17<br>11 | 14<br>9 | 12.5<br>7.5 | Divide byte operand<br>Divide byte constant |
| DIVD | BM00.00<br>1000 | 21<br>19 | 18<br>16 | variable<br>variable | Divide word operand<br>Divide word constant |

*The contents of the accu is arithmetically combined with the specified operand*

*The result of the operation is written into the accu. You can either use it for further operations or assign it to an operand.*

## 4.4.2.4 Comparison,- shift- and incrementation commands

| *Com-mand* | *Operand* | *No. of bytes* | | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| | | *with* | *w/o* | | |
| | | *Interm. code* | | | |
| CMP | BM00.00<br>100 | 24<br>19 | 21<br>17 | 16.0<br>11.0 | Compare with byte operand<br>Compare with byte constant |
| CMPD | BM00.00<br>1000 | 44<br>40 | 41<br>37 | 38.0<br>30.0 | Compare with word operand<br>Compare with word constant |
| LSL | No ope-rand | 6 | 5 | 3.0 | 8-bit shift left of contents of accu |
| LSR | No ope-rand | 6 | 5 | 5.0 | 8-bit shift right of contents of accu |
| INC | BM00.00 | 13 | 10 | 12.0 | Increment byte operand (con-tents + 1) |
| DEC | BM00.00 | 13 | 10 | 12.0 | Decrement byte operand (con-tents - 1) |
| INCD | BM00.00 | 45 | 42 | 45.0 | Increment word operand (con-tents+ 1) |
| DECD | BM00.00 | 45 | 42 | 45.0 | Decrement word operand (contents - 1) |
| CLR | BM00.00 | 14 | 11 | 14.0 | Clear byte operand |
| NOP | No ope-rand | 2 | 1 | 1.0 | Dummy instruction |

*Please also read the explanatory notes on the next page.*

## Notes on comparison, shift and incrementation commands

- Compare (CMP...)

Compares the contents of the accu with the contents of the operand. The result is set as internal flag which is evaluated by jump commands (see "4.4.2.5 Jump commands").

- Shift (LS...)

Shifts the contents of the accu by one place.

- Increment (INC...), Decrement (DEC...)

Increments or decrements the contents of the accu by 1.

## 4.4.2.5 Jump commands

| *Com-mand* | *Operand* | *No. of bytes with* | *w/o Interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| JP | Label | 12 | 10 | 5.0 | Unconditional jump to specified label |
| JPC | Label | 14 | 12 | 6.0 | Conditional jump (if logical 1) to specified label |
| JPCN | Label | 14 | 12 | 6.0 | Conditional jump (if logical 0) to specified label |
| JP= | Label | 12 | 13 | 6.0 | Jump to specified label if equal (after comparison) |
| JP<> | Label | 15 | 13 | 6.0 | Jump to specified label if not equal (after comparison) |
| JP< | Label | 18 | 16 | 7.5 | Jump to specified label if smaller (after comparison) |
| JP> | BM00.00 | 15 | 13 | 7.5 | Jump to specified label if greater (after comparison) |
| JP<= | Label | 18 | 16 | 7.5 | Jump to specified label if smaller or equal (after comparison) |
| JP>= | Label | 18 | 16 | 7.5 | Jump to specified label if greater or equal (after comparison) |
| JPP | Program module | 5 | 3 | 18.0 | Unconditional jump to specified program module |
| JPCP | Program module | 9 | 7 | 18.0 | Conditional jump (if logical 1) to specified program module |

| *Command* | *Operand* | *No. of bytes with Interm. code* | *No. of bytes w/o Interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| JPK | KUBES module | 7 | 3 | 18 | Unconditional jump to specified KUBES module |
| JPCK | KUBES module | 11 | 7 | 18 | Conditional jump (if logical 1) to specified KUBES module |

*Jumps in the program immediately move program processing to the destination line. This can be either a so-called label (i.e. a symbolic jump mark) or another module.*

➢ Conditional jumps (JPC...)

The jump is taken if the preceding operation resulted in "logical 1" or "logical 0" (JPCN).

➢ Jumps after comparison (JP= to JP>=)

The jump is taken if the contents of the accu has the specified mathematical relation to the operand.

## 4.4.2.6 Copy commands

| *Command* | *Operand* | *No. of bytes with* | *No. of bytes w/o* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| | | *Interm. code* | | | |
| C1T8 | I00.00 | 7 | 4 | 3/350 | Copy 8 bit operands from the on-chip RAM (→ 3.10.5) to the accu |
| | SM00.00 | 8 | 5 | 200 | Copy 8 bit operands to the accu |
| C8T1 | O00.00 | 5 | 2 | 1/400 | Copy the contents of the accu to 8 bit operands in the on-chip RAM (→ 3.10.5) |
| | SM00.00 | 8 | 5 | 200 | Copy the contents of the accu to 8 bit operands |
| C1T16 | I00.00 | 10 | 7 | 5/650 | Copy 16 bit operands from the on-chip RAM (→ 3.10.5) to the accu |
| | SM00.00 | 8 | 5 | 300 | Copy 16 bit operands to the accu |
| C16T1 | O00.00 | 8 | 5 | 3/750 | Copy the contents of the accu to 16 bit operands in the on-chip RAM (→ 3.10.5) |
| | SM00.00 | 8 | 5 | 300 | Copy the contents of the accu to 16 bit operands |

*Please also read the explanatory notes on the next page.*

## Notes on the copy commands

Copy commands are used to parallely load the contents of 8 or 16 bit operands into the accu or write the contents of the accu into 8 or 16 bit operands.

Practical applications:

- reading binary or BDC values via inputs
- controlling numerical displays (e.g. 7-segment display)

The time it takes to process copy commands depends on the last number of the bit operand’s channel number.

The channel number is indicated after the separating point:

I00.00

channel number

Processing time is shorter if the channel number ends with 0 or 8.

### Example 1

| | | | |
|---|---|---|---|
| C1T8 | I00.00 | => processing time: | 3 µs |

### Example 2

| | | | |
|---|---|---|---|
| C1T8 | I00.13 | => processing time: | 350 µs |

## 4.4.2.7 Programmable pulses , timers and counters

| *Command* | *Operand* | *No. of bytes with Interm. code* | *No. of bytes w/o Interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| = | PP00.00 | 11 | 8 | 42 | Activate pulse at positive edge (0/1) |
| =N | PP00.00 | 11 | 8 | 42 | Activate pulse at negative edge (0/1) |
| L | PP00.00 | 9 | 6 | 6 | Load pulse |
| A,O... | PP00.00 | 13 | 10 | 10 | Link pulse |
| = | PT00.00:1000*1s:E [1)] | 16 | 8 | 32 | Start timer with const. preset value |
| = | PT00.00:BM00.00*1s:E [1)] | 34 | 26 | ~60 | Start timer with variable preset value (BM00.00+01) |
| L | PT00.00 | 9 | 6 | 6 | Load timer output |
| A,O.. | PT00.00 | 13 | 10 | 10 | Link timer output |
| LD | PT00.00 | 15 | 12 | 12.5 | Load current timer value |
| =TH | PT00.00 | 26 | 23 | 22 | Halt timer (without clearing it) |
| = | C00.00:10000:V [1)] | 14 | 6 | 35 | Start counter with const. preset value |
| = | C00.00:BM00.00:V [1)] | 32 | 24 | ~60 | Start counter with var. preset value (BM00.00+01) |

| *Command* | *Operand* | *No. of bytes with Interm. code* | *No. of bytes w/o Interm. code* | *Proc. time [ms]* | *Description* |
|---|---|---|---|---|---|
| L | C00.00 | 9 | 6 | 6 | Load counter output (count at preset value) |
| A, O... | C00.00 | 13 | 10 | 10 | Link counter output |
| LD | C00.00 | 15 | 12 | 12.5 | Read current counter value |
| =C | C00.00 | 9 | 6 | 25 | Assign pulse signal |

1) Adding “R” to the operand declaration makes the current timer or counter value remanent (→ 3.10.4).
Example: “ = PT00.00:1000:1s:E:R”

*Please also read the explanatory notes on the next page.*

## Notes on programmable pulses, timers and counters

These are more or less special forms of the commands described earlier. For a more detailed description refer to chapter "6 Examples".

- Programmable pulse

When a wipe pulse has been set (=, =N...) and the corresponding code line is skipped, the output signal will be retained until the line is processed again.

- Remanence

The "R" operand supplements listed in the table are optional parameters. Add them if you wish a timer or counter to be remanent (when you stop or reset the controller, the current (time) count will be stored and retrieved when you restart the controller).

- Timers

Once started, timers run regardless of whether the corresponding code line is being processed or not.

## 4.5 Programming modules

The user program of Eco Control 667E is structured by modules. This helps you to break up the technological problem to be controlled into separate part tasks. The modules form a hierarchical system (at max. 5 levels) that allows modules at higher levels to call modules at lower ones. A program of this structure is very clear and helps a lot with understanding or updating of finished programs. The following types of modules are available:

- organisation module
- program modules
- KUBES modules

Processing of individual modules is monitored by a watchdog which is triggered every time a module is called. After that the system has 70 ms to process the module.

Program and KUBES modules are subroutines. The return to the calling module is ensured by the module organisation and must not be programmed separately. Modules must not call themselves.

The maximum length of a module is 128 instructions. To these you may add extra comment lines so that the maximum number of lines is 253.

## 4.5.1 Organisation module

| | |
|---|---|
| Function: | organises the other modules |
| Name: | ORG |
| Quantity: | 1 |

It is practical if the ORG module contains the program selection and calls of the modules that are relevant to the overall task. All PLC instructions can be used without limitation.

## 4.5.2 Program module

| | |
|---|---|
| Function: | PLC program module for a separate part task. Organises the next module level. |
| Name: | Optional |
| Quantity: | Max. 255 |

## 4.5.3 KUBES module

| | |
|---|---|
| Function: | Library module for the solution of a specific, defined basic task. KUBES modules are programmed by Kuhnke in a high-level language and added as code to a library. |
| Name: | Set |
| Quantity: | Max. 255 |

## 4.5.4 Module hierarchy

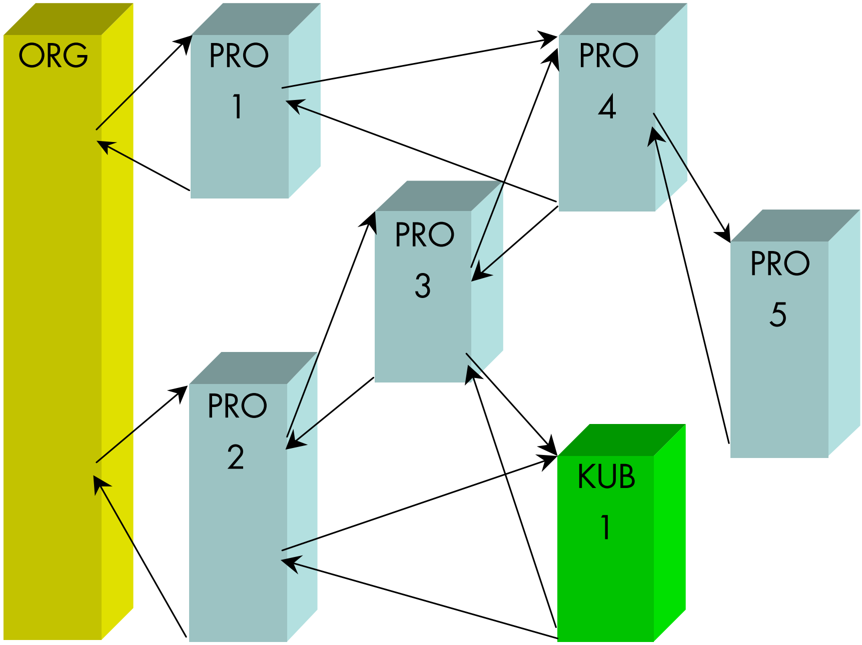



*Fig. 10: Module hierarchy, example*

Notes on the illustration

- Hierarchy levels

The example above uses all of the 5 available hiearchy levels by linking
ORG → PRO 2 → PRO 3 → PRO 4 → PRO 5

- Terminating modules

KUBES modules (here: KUB 1) are terminating modules. No other modules can be called from there.

# 5 KUBES modules

KUBES modules are subroutines translated into machine code. Their job is to solve compley tasks that program modules written by the user can solve only with difficulties or not at all.

## Reserved operands

The KUBES modules of Eco Control 667E accept no parameters. Data is exchanged via reserved operands (→ 4.2.3) which must not be assigned to any other addresses by the user program if the relevant KUBES modules are being used.

## Standard modules

A set of standard KUBES modules is automatically installed together with KUBES.

## Special modules

There is the option of delivering customised software solutions in the shape of KUBES modules. They are delivered separately and installed in the PC by means of BIBS, the library service program (part of the KUBES software package).

Feel free to contact us if and when required.

# 5.1 KUBES module libraries

KBUES modules are combined in libraries which are stored in the KUBES program root created when installing KUBES.

## Hard disk arrangement of KUBES

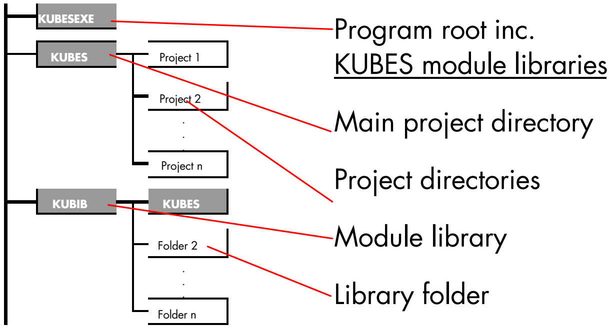



*Fig. 11: Hard disk arrangement of KUBES*

The KUBES module library is called:

- KULIB667.LIB

Other libraries are available. They apply to Kuhnke's other controllers which we do not want to discuss at this point.

KUBES automatically chooses the correct library for the project work. You are obliged to specify the type of controller when you open a new project. KUBES uses this information for library selection.

The type of controller to be chosen for Eco Control 667E is "667".

## 5.1.1 Contents of the KUBES module library

Library "KULIB667.LIB" not only applies to Eco Control 667E as described in this manual but also to the older types, Pico Control 667 and Compact Control 667.

Please note that some modules in the library can be used for the last two devices only, because they can be configured with an additional module if and when required.

The table lists the available modules in alphabetical order:

| *KUBES module* | *Used in Eco Control 667E* | *Function* |
|---|---|---|
| CNT_ENC | no | Counter functions for the add. "counter" module |
| CNT_EVENT | | |
| RD_OFFS | yes | Read with offset |
| SST667IN | | V.24 communication: send strings |
| V24667IE | | V.24 communication: receive individual char.s |
| V24667IS | | V.24 communication: send individual char.s |
| V24667ST | no | Communication via additional "V.24" module |
| V24667XE | | |
| V24667XS | | |
| WR_OFFS | yes | Write with offset |

The library can be viewed in KUBES:

- Module Editor
- Open "Module" menu
- Choose "KUBES modules"

## 5.1.2 Loading KUBES modules

The required KUBES module is started by a jump command at the appropriate place in the user program (organisation or program module):

➢ `JPK          <module name>`

Absolute jump. It is taken every time the microprocessor reads the program line. The module is not called if a jump skips the program line.

or

➢ `JPCK     <module name>`

Conditional jump. It is only taken if the preceding operation results in “logical 1”. The module is not called if a jump skips the program line.

## 5.2 Communication modules

Communication modules allow you to use the programming interface for simle data traffic.

There are three KUBES modules available for this task:

- V24667IS

Sends single characters

- V24667IE

Receives single characters

- V24667IN

Sends strings (data ranges)

The data transfer format is set and cannot be changed:

- 8 data bits
- 1 stop bit
- no parity check
- 1200 bit/s

## 5.2.1 Reserved operands

| *Suggested symbol* | *Address* | *Used by KUBES mod.* | *Value* [1] | *Function* |
|---|---|---|---|---|
| INIT_V24 | FBM01.00 | V24667IS,<br>V24667IE,<br>V24667IN | K:255 | V24 mode settings ok |
| RES_1 | FBM01.01 | | K:<n> | Internally used marker |
| KUBES | FBM01.02 | | <br>U:255<br>U:0 | V24 mode:<br>Programming/KUBES prot.<br>Communicating |
| | FBM01.03 | V24667IS | U:<Chr> | Char. to be sent |
| | FBM01.04 | V24667IS,<br>V24667IN | U:255<br>K: 0 | Start transfer<br>Acknowledge |
| REC_CHR | FBM01.05 | V24667IE | K:<Chr> | Char. to be received |
| | FBM01.06 | | K:255<br>U:0 | Character received<br>Acknowledge |
| | FBM01.07 | V24667IN | K:<n> | Internal counter of bytes sent |
| | FBM01.08 | | U:<n> | Qty. of data bytes (1...230) |
| SDATA | FBM01.09<br>to<br>FBM15.15 | | U:<Dat> | Data field to be sent |

[1] K: KUBES module writes
U: user writest

> *These operands are reserved for the described functions. They must not be used for any other purposes if the relevant KUBES modules are embedded in the program.*

## 5.2.2 V.24 mode settings

Reserved operand “FBM01.02” enables communication. This operand’s status decides whether the KUBES protocol in programming mode (also supporting communication with suitable dialogue terminals, for example) or the communication mode is activated:

| *Operand* | *Status* | *V.24 mode* |
|---|---|---|
| FBM01.02 | 255 | Programming (KUBES protocol) |
| | 0 | Communicating by means of the KUBES modules described below |

*To switch over to communication mode please make sure to use an external input as suggested in the example program below (**à** 5.2.6). Failure to comply may permanently disable the programming mode.*

The chosen mode becomes active as soon as at least one of the KUBES modules has been run.

➢ The KUBES module acknowledges the change of settings:
[FBM01.00] 🡐 255

*Clear operand (FBM01.00) at the start of the program because it is undefined when you switch on the controller.*

*Example program (**à** 5.2.6) lines 3...26*

## 5.2.3 Sending single characters (V24667IS)

| | |
|---|---|
| KUBES module: | V24667IS |
| Length: | 66 byte |
| Processing time: | c. 50 µs |
| Function: | send single character |

### 5.2.3.1 Program structure

1. User chooses V.24 mode (→5.2.2)
2. User verifies that no character is being sent
   [FBM01.04] → 0 ?
3. User specifies the character to be sent
   [FBM01.03] ← <character to be sent>
4. User starts data transfer
   [FBM01.04] ← 255
5. KUBES module acknowledges when transfer is done
   [FBM01.04] ← 0

Step 1 only needs to be taken once to enable communication. It is the same for sending and receiving data. Afterwards, steps 2...5 can be taken any number of times, also alternating with receiving actions.

*Example program (à 5.2.6) lines 34...50*

## 5.2.4 Receiving single characters (V24667IE)

| | |
|---|---|
| KUBES modules: | V24667IE |
| Length: | 106 byte |
| Processing time: | c. 90 µs |
| Function: | receive single character |

### 5.2.4.1 Program structure

1. User chooses V.24 mode(→5.2.2)
2. User checks whether a character was received
   [FBM01.06] ➔ 255 ?
3. User reads the character received
   [FBM01.05] ➔ <character received>
4. User acknowledges reception
   [FBM01.06] 🡐 0

Step 1 only needs to be taken once to enable communication. It is the same for sending and receiving data. Afterwards, steps 2…4 can be taken any number of times, also alternating with sending actions.

*Example program (à 5.2.6) lines 75...83*

# 5.2.5 Sending strings (SST667IN)

| | |
|---|---|
| KUBES module: | SST667IN |
| Length: | 104 byte |
| Processing time: | c. 60 µs |
| Function: | send strings (of characters) |

## 5.2.5.1 Program structure

1. User chooses V.24 mode (→5.2.2)
2. User verifies that no strings are being sent
   [FBM01.04] ➔ 0 ?
3. User writes the data to be sent into the data field
   [FBM01.09 ff] 🡐 <data bytes to be sent>
4. User specifies the quantity of data bytes
   [FBM01.08] 🡐 <quantity of data bytes to be sent)
5. User starts transfer
   [FBM01.04] 🡐 255
6. KUBES module acknowledges when transfer is done
   [FBM01.04] 🡐 0

Step 1 only needs to be taken once to enable communication. It is the same for sending and receiving data. Afterwards, steps 2...6 can be taken any number of times, also alternating with receiving single characters.

*Example program (**à** 5.2.6) lines 52...73*

## 5.2.6 Example program “serial communication“

This program uses all KUBES modules available for serial communication.

```
======== KUBES =======================================================

                        Project structure

Project  : 667_COMM              Network  :
                                 created : Aug 19 1998 10:03
User :                           changed : Aug 19 1998 16:39
Comment: 667E: Data communication via V.24
======================================================================

ORG.ORG/1
|
*------>SST667IN.KNK/6
|
*------>V24667IE.KNK/9
|
*------>V24667IS.KNK/10
```

```
======== KUBES =======================================================

                        Organisation module  IL

Project  : 667_COMM              Network  :
Module : ORG       No.: 1        created : Aug 19 1998 10:03
User :                           changed : Aug 19 1998 16:41
======================================================================

; ------------- Data communication test program -----------------
; Enable V.24 mode
; ================
; Clear operand FBM01.00 first (process once only)
         L       INI_MRK         M00.00 ; (initialisation marker)
         JPC     MODE_SEL
         L       0
         =       V24_OK        FBM01.00 ; (255 = V24 mode enabled)
         L       PL00.01
         =       INI_MRK         M00.00 ; (initialisation marker)
         JP      END_COM
; Choose mode (process cyclically)
MODE_SEL L       V24_MODE        I00.00 ; (0=programming, 1=communic.)
         JPCN    PROG
COMM     L       0                      ; communication mode
         =       KUBES         FBM01.02 ; (255 =programming, 0=communic.)
         JP      RUN_V24
PROG     L       255                    ; programming mode
         =       KUBES         FBM01.02 ; (255 =programming, 0=communic.)
; Start KUBES module "send single character"
RUN_V24  JPK     V24667IE               ; single character received
; Mode enabled?
         L       V24_OK        FBM01.00 ; (255 = V24 mode enabled)
         CMP     255
         JP<>    END_COM                ; no -> jump
; Send single characters or strings?
; ==================================
SND_MOD  L       SND_MODE        I00.01 ; (0=single char., 1=strings)
         JPCN    S_SINGLE               ; send single character
         JPC     S_STRING               ; module: send strings
```

```
; Send single characters
; =====================
; Specify transfer interval (every second)
S_SINGLE L       T00.02                  ; current value
         CMP     SBM00.00                ; old value
         JP=     REC                     ; second not passed yet
         =       SBM00.00                ; store new value
; Send
SEND     JPK     V24667IS                ; KUBES module
         L       SND_RUN        FBM01.04 ; (255 =start transfer, 0=ackn.)
         JPC     END_SNGL                ; still sending
         L       PC00.02                 ; clock gen. value as s_char.
         =       SND_CHR        FBM01.03 ; (character to be sent)
         C8T1    O00.08                  ; show SND_CHR at outputs
         L       255
         =       SND_RUN        FBM01.04 ; (255 =start transfer, 0=ackn.)
END_SNGL JP      REC

; Send strings
; ============
; Specify data to be sent (here: "<STX>PLC<ETX>")
S_STRING JPK     SST667IN                ; KUBES module
         L       $02                     ; STX (Start of Text)
         =       SDATA          FBM01.09 ; (start of s_data field)
         L       'S'
         =       FBM01_10       FBM01.10 ; (data to be sent)
         L       'P'
         =       FBM01_11       FBM01.11 ; (data to be sent)
         L       'S'
         =       FBM01_12       FBM01.12 ; (data to be sent)
         L       $03                     ; ETX (End of Text)
         =       FBM01_13       FBM01.13 ; (data to be sent)
LENGTH   L       5                       ; length of data to be sent
         =       SDAT_LEN       FBM01.08 ; (qty. of s_data bytes)
; Send
SEND_STR L       SND_RUN        FBM01.04 ; (255 =start transfer, 0=ackn.)
         JPC     END_STRG                ; still sending
         L       255
         =       SND_RUN        FBM01.04 ; (255 =start transfer, 0=ackn.)
END_STRG NOP
```

```
; Receive single character
; ========================
REC       L       REC_RUN       FBM01.06 ; (255 =receive char., 0=ackn.)
          CMP     255
          JP<>    END_REC
          L       REC_CHR       FBM01.05 ; (received character)
          C8T1    A00.00                 ; show REC_CHR at outputs
          CLR     REC_RUN       FBM01.06 ; (255 =char. received, 0=ackn.)
END_REC   NOP
; End of communication program
; ============================
END_COM   NOP
```

## 5.3 Copying data (blocks)

The two KUBES modules described next serve the following purposes:

- KUBES module "RD_OFFS"
reads a specified operand range
- KUBES module "WR_OFFS"
writes into a specified operand range
- both KUBES modules
copy data from one operand range to another

The operand range is accessed via its intermediate code address (→ 5.3.2). You can add an offset to address 1 (or two in the case of word operands) particular operand in the range.

### 5.3.1 Reserved operands

| *Suggested symbol* | *Address* | *Used by KUBES module* | *Function* |
|---|---|---|---|
| WR_SRC | BM00.00...01 | WR_OFFS | Data source (2 byte) |
| RD_DEST | BM00.02...03 | RD_OFFS | Data destination (2 byte) |
| OFFSET | BM01.00 | RD_OFFS and WR_OFFS | Pointer (offset) to an address in the operand range |
| ADDRESS | FBM00.00...01 | RD_OFFS WR_OFFS | First address of the operand range |

*These operands are reserved for the described functions. They must not be used for any other purposes if the relevant KUBES modules are embedded in the program.*

## 5.3.2 Operands' intermediate code addresses

The operands can only be accessed directly if the normal mnemonic (M00.00, SBM03.15...) is used.

Internal markers can only be accessed indirectly by means of their intermediate code addresses:

| *Operand range* | | *Intermediate code address (hexadecimal)* | |
|---|---|---|---|
| *Start* | *End* | *Start* | *End* |
| R00.00 | R15.15 | $0100 | $01FF |
| BM00.00 | BM15.15 | $0200 | $02FF |
| SBM00.00 | SBM15.15 | $0300 | $03FF |
| BR00.00 | BR15.15 | $0400 | $04FF |
| SBR00.00 | SBR15.15 | $0500 | $05FF |
| - | - | $0600 | $06FF |
| ABM00.00 | ABM15.15 | $0700 | $07FF |
| FM00.00 | FM15.15 | $0800 | $08FF |
| - | - | $0900 | $09FF |
| SR00.00 | SR15.15 | $0A00 | $0AFF |
| BC00.00 | BC15.15 | $0B00 | $0BFF |
| SBC00.00 | SBC15.15 | $0C00 | $0CFF |
| BD00.00 | BD15.15 | $0D00 | $0DFF |
| SBD00.00 | SBD15.15 | $0E00 | $0EFF |
| LBM00.00 | LBM15.15 | $0F00 | $0FFF |

## 5.3.3 Reading data (RD_OFFS)

| | |
|---|---|
| KUBES module: | RD_OFFS |
| Length: | 58 byte |
| Processing time: | c. 89 µs |
| Function: | read operand data |

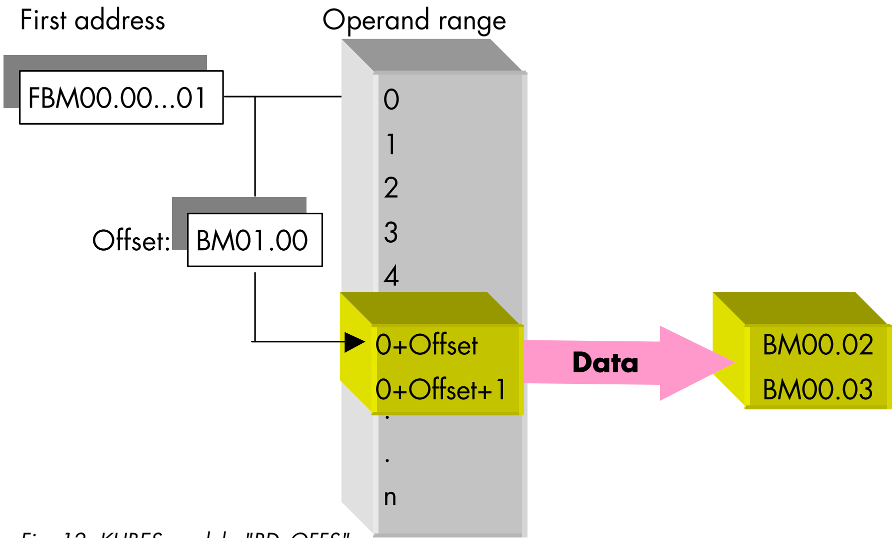



*Fig. 12: KUBES module "RD_OFFS"*

### 5.3.3.1 Program structure

➢ Specify the first address of the source operand range for reading
[FBM00.00...01] ← <intermediate code address>

➢ Set the offset of an address in the range (0 = first)
[BM01.00] ← <offset>

➢ Start KUBES module RD_OFFS

➢ Evaluate 1 or 2 byte of data
[BM00.02...03] ➔ <evaluation>

*Example program (**à** 5.3.5)*

## 5.3.4 Writing data (WR_OFFS)

| | |
|---|---|
| KUBES module: | WR_OFFS |
| Length: | 74 byte |
| Processing time: | c. 33 µs |
| Function: | write data into operand |

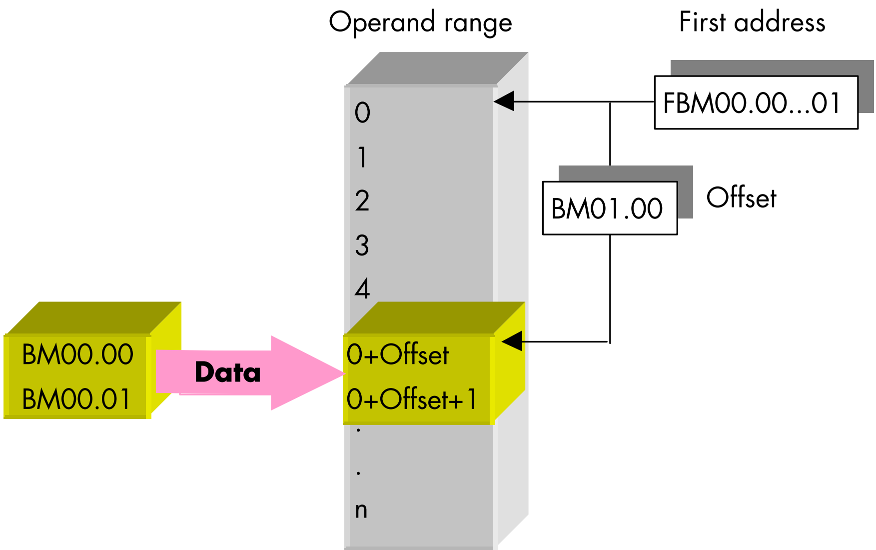



*Fig. 13: KUBES module "WR_OFFS"*

### 5.3.4.1 Program structure

➢ Provide 1 or 2 byte of data
[BM00.00...01] ← <data>

➢ Specify the first address of the target operand range for writing
[FBM00.00...01] ← <intermediate code address>

➢ Set the offset of an address in the range (0 = first)
[BM01.00] ← <offset>

➢ Start KUBES module WR_OFFS

*Example program (à 5.3.5)*

## 5.3.5 Example program "copy data block"

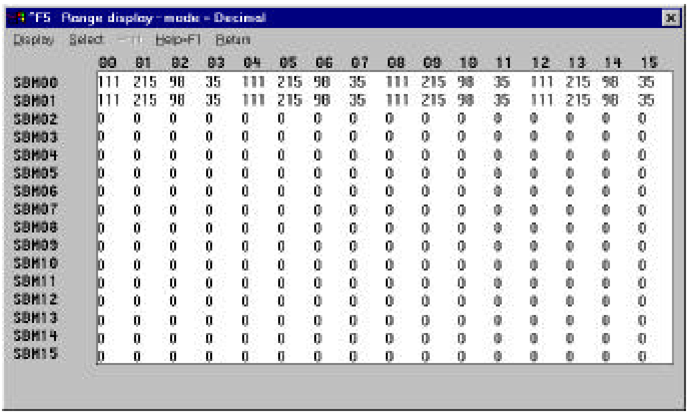
This program is to copy 16 byte of data. The source and target ranges are very near each other so that the program can be easily tested by means of KUBES' Address Range display function:

| *Data ranges* | | *First address (interm. code)* |
|---|---|---|
| Source | SBM00.00...SBM00.15 | $0300 |
| Target | SBM01.00...SBM01.15 | $0310 |

We took the first intermediate code address of every range from the table in chapter (→ 5.3.2).

Proceed as follows:

- ➢ Write and transmit program (see next page)
- ➢ Start controller (RUN)
- ➢ Display Address Range, choose "byte markers SBM"




- ➢ Set and reset input I00.00.

After a short while you will find the data in SBM01.00...15 and SBM00.00...15 are identical (see screen dump).

```
======== KUBES ========================================================

                        Project structure
Project  : 667_COPY                 Network  :
                                    created : Aug 17 1995 11:28
User :                              changed : Aug 17 1998 09:59
Comment: 667E: Copy data block
======================================================================

ORG.ORG/1
|
*------>RD_OFFS.KNK/3
|
*------>WR_OFFS.KNK/4
|
*------>DAT_SET.PRO/1
```

```
======== KUBES ========================================================
                  Organisation module  IL
Project  : 667_COPY            Network  :
Module   : ORG       No.: 1      created : Aug 17 1995 11:28
User :                           changed : Aug 17 1998 08:47
======================================================================
; ------------------ Test program "Copy data blocks" -----------------
***************************************************************************
; To be able to easily repeat the test, the provided data is modified
; by a randomizer via I00.00 (see module "DAT_SET").
***************************************************************************

; Jump to "Provide data"
; ----------------------
         L       SET_DAT         I00.00 ; (set data)
         JPCP    DAT_SET              1

; First steps
; -----------
; Clear offset memory to be on the safe side
         CLR     OFFSET         BM01.00 ; (points to oper. in data field)

; Specify length of data fields (here: BM01.01 chosen)
         L       16                     ; 16 byte
         =       DAT_LEN        BM01.01 ; (length of data fields)

; Program for copying (designed as a loop)
; ----------------------------------------
LOOP     NOP
; Specify start of source memory (SBM00.00)
SOURCE   LD      $0300                  ; interm. code of 1st source addr.
         =D      ADDR_1        FBM00.00 ; (1st addr. of operand range)
         JPK     RD_OFFS                ; source data to BM00.02...03
; Transfer data
         LD      DATA_RD        BM00.02 ; (data from source memory, byte)
         =D      DATA_WR        BM00.00 ; (data to target memory, byte 1)
; Specify start of target memory (SBM01.00)
DEST     LD      $0310                  ; interm. code of 1st target addr.
         =D      ADDR_1        FBM00.00 ; (1st addr. of operand range)
         JPK     WR_OFFS                ; data from BM00.00...03 to target
; Increment offset (twice because your're copying by word
INCOFFS  INC     OFFSET         BM01.00 ; (pointer to oper. in data field)
         INC     OFFSET         BM01.00 ; (pointer to oper. in data field)
; Complete range copied?
         L       OFFSET         BM01.00 ; (pointer to oper. in data field)
         CMP     DAT_LEN        BM01.01 ; (length of data fields)
         JP<     LOOP                   ; no -> get next set of data
END_COPY NOP
```

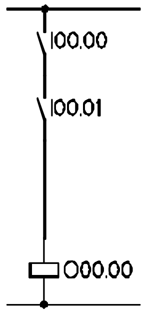
```
======== KUBES ========================================================
                        Program module  IL
Project  : 667_COPY                 Network  :
Module   : DAT_SET   No.: 1         created : Aug 14 1998 16:47
User :                              changed : Aug 19 1998 09:37
Comment: Provide data
======================================================================

; A kind of randomizer writes the contents of the internal clock pulse
; generators into byte markers SBM00.00...15 while the module is running
; ---------------------------------------------------------------------
        L       PC00.00
        =       SBM00.00
        L       PC00.01
        =       SBM00.01
        L       PC00.02
        =       SBM00.02
        L       PC00.03
        =       SBM00.03
        L       PC00.00
        =       SBM00.04
        L       PC00.01
        =       SBM00.05
        L       PC00.02
        =       SBM00.06
        L       PC00.03
        =       SBM00.07
        L       PC00.00
        =       SBM00.08
        L       PC00.01
        =       SBM00.09
        L       PC00.02
        =       SBM00.10
        L       PC00.03
        =       SBM00.11
        L       PC00.00
        =       SBM00.12
        L       PC00.01
        =       SBM00.13
        L       PC00.02
        =       SBM00.14
        L       PC00.03
        =       SBM00.15
```

# 6 Examples

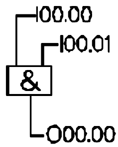
## 6.1 Basic functions

### 6.1.1 AND

Wiring diagram




Logic diagram

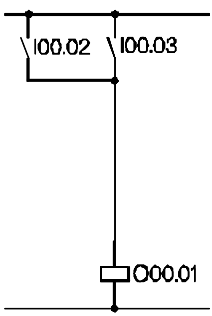



Instruction list

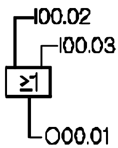
| | |
|---|---|
| L | I00.00 |
| A | I00.01 |
| = | O00.00 |

### 6.1.2 OR

Wiring diagram




Logic diagram

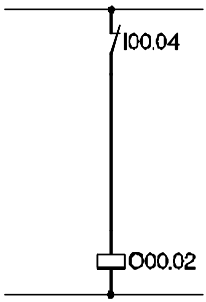



Instruction list

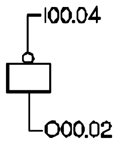
| | |
|---|---|
| L | I00.02 |
| O | I00.03 |
| = | O00.01 |

## 6.1.3 Negated input

Wiring diagram




Logic diagram

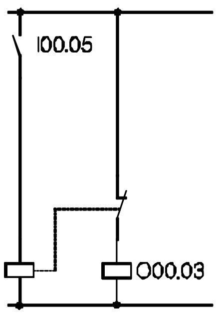



Instruction list

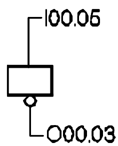
| | |
|---|---|
| LN | I00.04 |
| = | O00.02 |

## 6.1.4 Negated output

Wiring diagram




Logic diagram

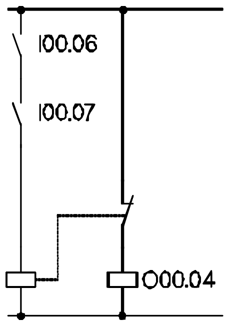



Instruction list

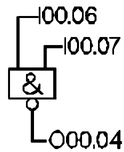
| | |
|---|---|
| L | I00.05 |
| =N | O00.03 |

## 6.1.5 NAND

Wiring diagram




Logic diagram

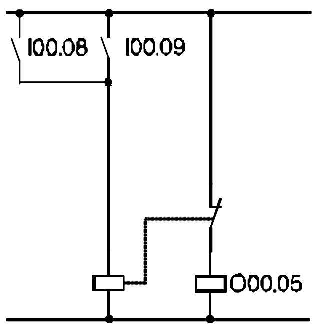



Instruction list

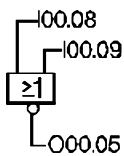
| | |
|---|---|
| L | I00.06 |
| A | I00.07 |
| =N | O00.04 |

## 6.1.6 NOR

Wiring diagram




Logic diagram

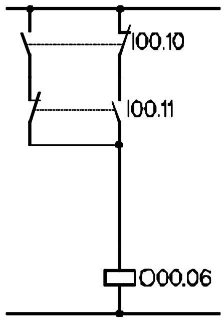



Instruction list

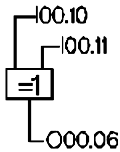
| | |
|---|---|
| L | I00.08 |
| O | I00.09 |
| =N | O00.05 |

## 6.1.7 XO: exclusive OR (antivalence)

Wiring diagram




Logic diagram

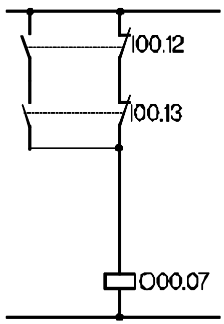



Instruction list

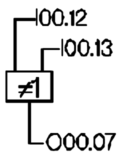
| | |
|---|---|
| L | I00.10 |
| XO | I00.11 |
| = | O00.06 |

## 6.1.8 XON: exclusive NOR (equivalence)

Wiring diagram




Logic diagram

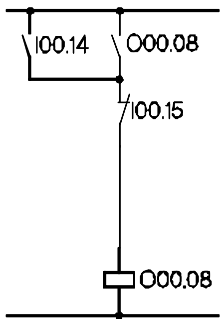



Instruction list

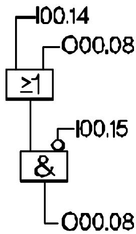
| | |
|---|---|
| L | I00.12 |
| XON | I00.13 |
| = | O00.07 |

# 6.1.9 Seal-in circuit

Wiring diagram




Logic diagram



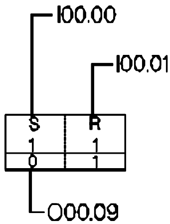

Instruction list

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

# 6.2 Memory functions

## 6.2.1 Mainly resetting

Logic diagram

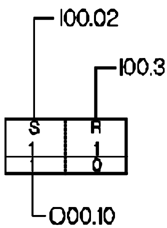



Instruction list

| | |
|---|---|
| L | I00.00 |
| S | O00.09 |
| L | I00.01 |
| R | O00.09 |

## 6.2.2 Mainly setting

Logic diagram



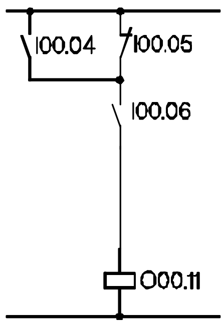

Instruction list

| | |
|---|---|
| L | I00.03 |
| R | O00.10 |
| L | I00.02 |
| S | O00.10 |

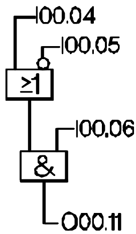
# 6.3 Switching circuits

## 6.3.1 OR-AND circuit

Wiring diagram




Logic diagram

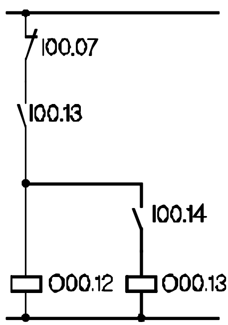



Instruction list

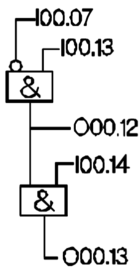
| | |
|---|---|
| L | I00.04 |
| ON | I00.05 |
| A | I00.06 |
| = | O00.11 |

## 6.3.2 Parallel circuit to output

Wiring diagram




Logic diagram

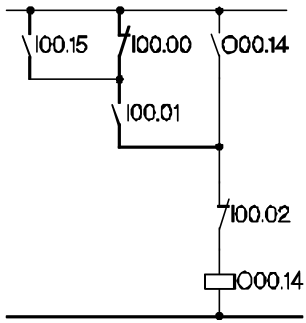



Instruction list

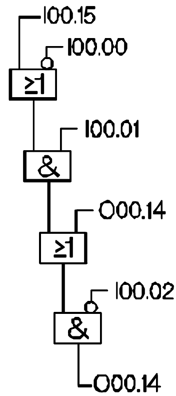
| | |
|---|---|
| LN | I00.07 |
| A | I00.13 |
| = | O00.12 |
| A | I00.14 |
| = | O00.13 |

## 6.3.3 Network with one output

Wiring diagram




Logic diagram

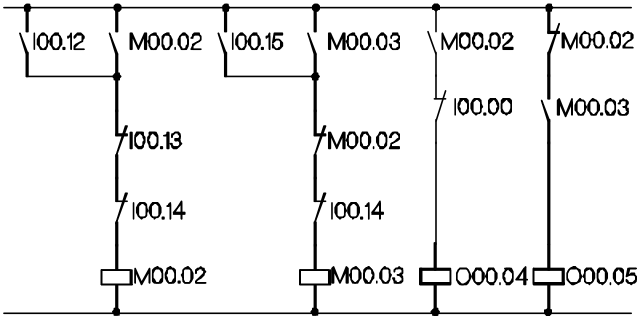



Instruction list

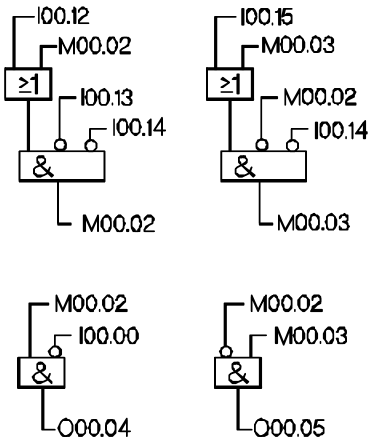
| | |
|---|---|
| L | I00.15 |
| ON | I00.00 |
| A | I00.01 |
| O | O00.14 |
| AN | I00.02 |
| = | O00.14 |

## 6.3.4 Network with outputs and markers

Wiring diagram




Logic diagram



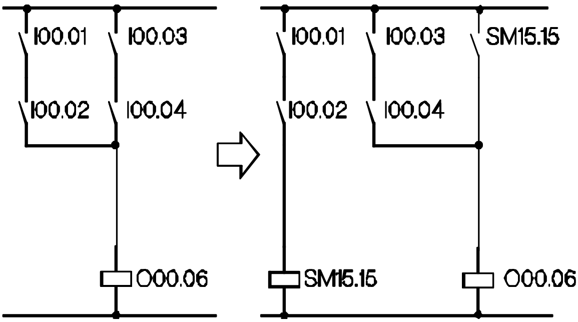

Instruction list

| | |
|---|---|
| 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 |

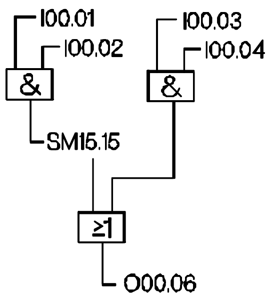
# 6.4 Special markers used as AND/OR marker

## 6.4.1 Network with OR marker

Wiring diagram




Logic diagram




Instruction list

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

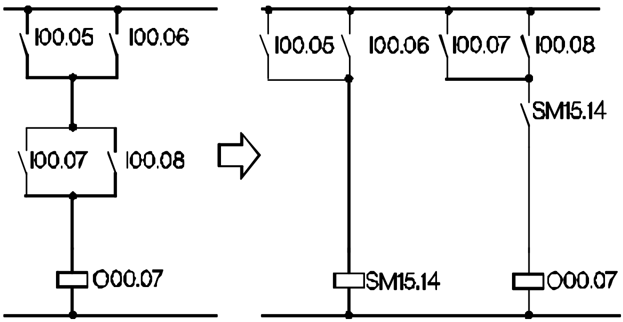
Note: In this example, a part result is to be briefly stored.

Definition: Always use special marker SM15.15 because it can be used again in other networks.

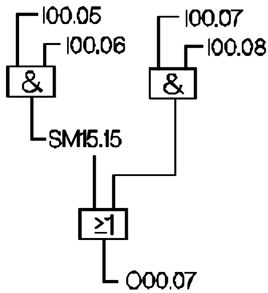
OR marker = SM 15.15

## 6.4.2 Network with AND marker

Wiring diagram




Logic diagram




Instruction list

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

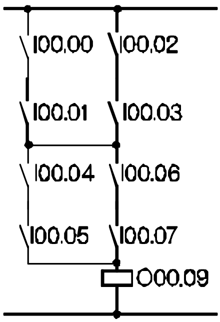
Note: In this example, too, a result is to be stored briefly in a special marker which is AND connected.

Definition: Always use special marker SM15.14 as AND marker.

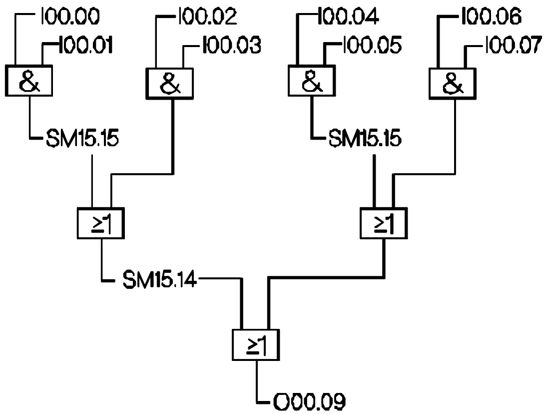
AND marker = SM 15.14

## 6.4.3 Network with multiple use of the OR marker

Wiring diagram




Logic diagram



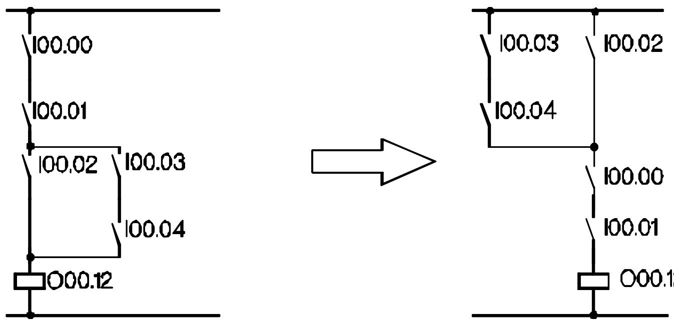

Instruction list

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

## 6.5 Circuit conversion

Wiring diagram before

Wiring 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 |

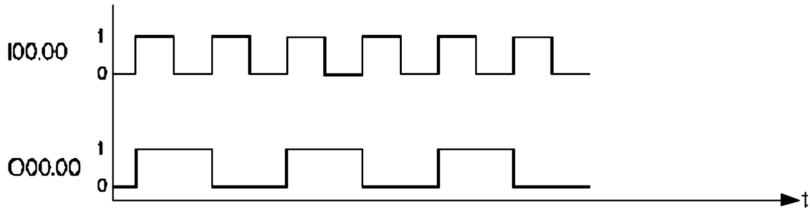
Circuit conversion leads to another sequence of instructions. This facilitates program creation because you can do without some of the markers for part results.

The length of the program is considerably reduced.

# 6.6 Special-purpose circuits

## 6.6.1 Impulse relay

Wiring diagram

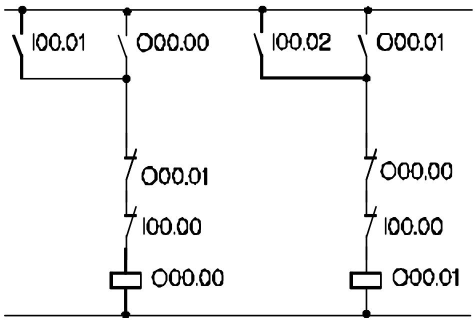



Instruction list

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

## 6.6.2 Reversing circuit (reversing starter) with forced stop

Wiring diagram




Instruction list

| | | |
|---|---|---|
| L | I00.01 | :right push-button |
| O | O00.00 | :right contactor |
| AN | O00.01 | :left contactor |
| AN | I00.00 | :Stop push-button*) |
| = | O00.00 | :right contactor |
| L | I00.00 | :left push-button |
| O | O00.01 | :left contactor |
| AN | O00.00 | :right contactor |
| AN | I00.00 | :Stop push-button*) |
| = | O00.01 | :left contactor |

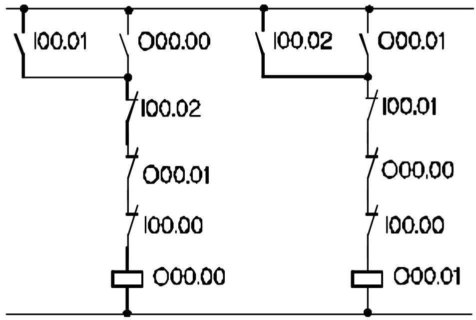
Notes:

We recommend that you provide a contactor interlock outside the PLC because switching between outputs is very fast.

*) Type A (AND) at this point if an n.c. Stop button has been connected outside the controller for safety reasons.

## 6.6.3 Reversing circuit (reversing starter) without forced stop

Wiring diagram




Instruction list

| | | |
|---|---|---|
| L | I00.01 | ; right push-button |
| O | O00.00 | ; right contactor |
| AN | I00.02 | ; left push-button |
| AN | O00.01 | ; left contactor |
| AN | I00.00 | ; Stop push-button*) |
| = | O00.00 | ; right contactor |
| L | I00.02 | ; left push-button |
| O | O00.01 | ; left contactor |
| AN | I00.01 | ; right push-button |
| AN | O00.00 | ; right contactor |
| AN | I00.00 | ; Stop push-button*) |
| = | O00.01 | ; left contactor |

Notes: We recommend that you provide a contactor interlock outside the PLC because switching between outputs is very fast.

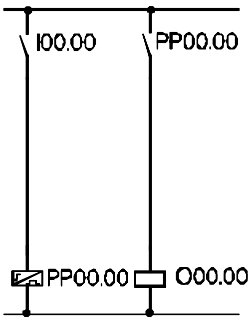
*) Type A (AND) at this point if an n.c. Stop button has been connected outside the controller for safety reasons.

## 6.7 Edge evaluation (wiping pulse)

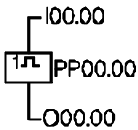
ECO Control 667E has 128 programmable wiping pulses for the detection of status changes of logical signals (edge evaluation). The pulses can be used for both rising and falling edges.

### 6.7.1 Programmable wiping pulse at rising edge

Wiring diagram



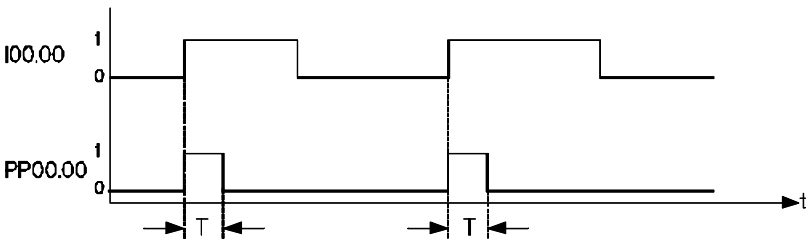

Circuit symbol




Instruction list

| | |
|---|---|
| L | I00.00 |
| = | PP00.00 |
| L | PP00.00 |
| = | O00.00 |

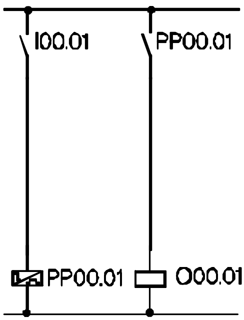
Signal curve



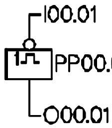

T = cycle time

## 6.7.2 Programmable wiping pulse at falling edge

Wiring diagram



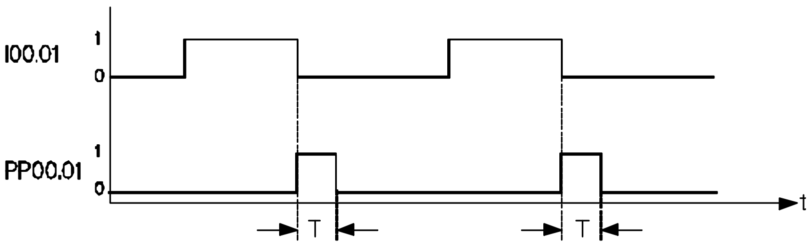

Circuit symbol




Instruction list

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

Signal curve

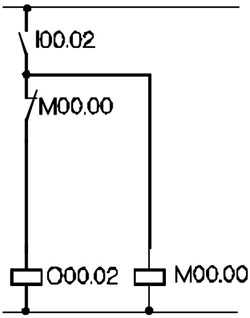



T = cycle time

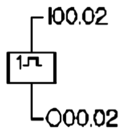
As opposed to the programmable wiping pulses of the previous examples, which were activated by a change of edge, the next two examples evaluate the signal state. This influences the start-up behaviour.

## 6.7.3 Wiping pulse at positive signal

Wiring diagram



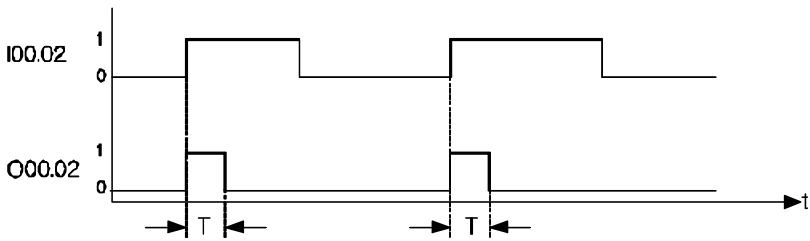

Circuit symbol




Instruction list

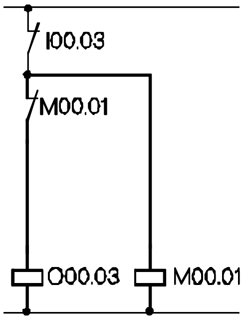
| | |
|---|---|
| L | I00.02 |
| AN | M00.00 |
| = | O00.02 |
| L | I00.02 |
| = | M00.00 |

Signal curve

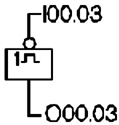


## 6.7.4 Wiping pulse at negative signal

Wiring diagram



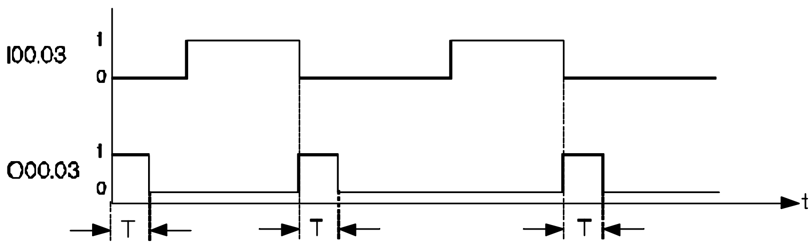

Circuit symbol




Instruction list

| | |
|---|---|
| LN | I00.03 |
| AN | M00.01 |
| = | O00.03 |
| LN | I00.03 |
| = | M00.01 |

Signal curve



# 6.8 Software timers

## 6.8.1 Mnemonics

You can program up to 32 software timers in the range between 10 ms and 65535 s. Timer addresses are PT00.00 – PT01.15.

**Start timer**

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

:**R** remanence

:**R** raising delay

:**F** falling delay

:**P** impulse

:**C** clock pulse

**10 ms** (or *100 ms, or *1s

16 bit constant (**1 – 65535** ) or

16 bit variable (e.g. **BM01.02** (+BM01.03

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

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

*) Note: Adding the "R" parameter (remanence of current timer value) is optional.

| | | | |
|---|---|---|---|
| • **Read output** | L | PTxx.xx | "1"= time run out |
| • **Read current value (remaining time)** | LD | PTxx.xx | 16 bit value of remaining time |
| • **Stop timer** | =TH | PTxx.xx | stop without RESET |

### 6.8.1.1 Syntax examples

Start raising delay of 17.5 s with remanent current value:

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

Start falling delay with variable timer value (timer value in BM06.02/03):

```
=       PT01.01:BM04.06*100ms:F
```

Read timer value and store in BM06.02/03:

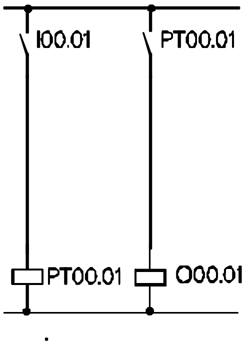
```
LD      PT01.02
=D      BM06.02
```

Stop timer while I01.00 is on:

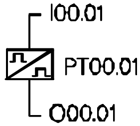
```
L       I01.00
=TH     PT01.03
```

## 6.8.2 Impulse at start-up

Wiring diagram



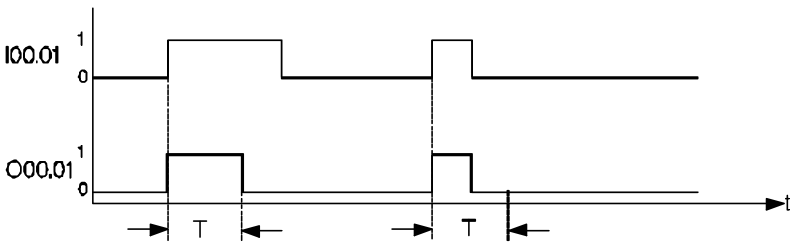

Circuit symbol




Instruction list

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

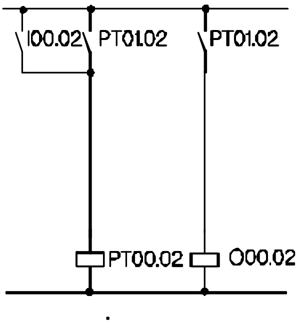
Signal curve



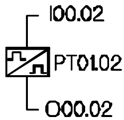

T = set time (here: 1.35s)

## 6.8.3 Impulse of constant duration

Wiring diagram



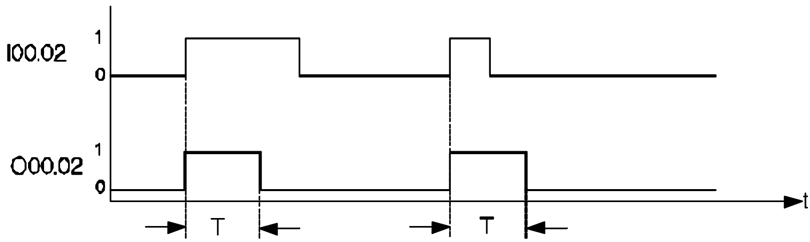

Circuit symbol




Instruction list

```
L   I00.02
O   PT01.02
=   PT01.02:123*100ms:P
L   PT01.02
=   O00.02
```

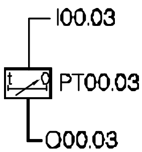
Signal curve




T = set time (here: 12.3s)

## 6.8.4 Raising delay

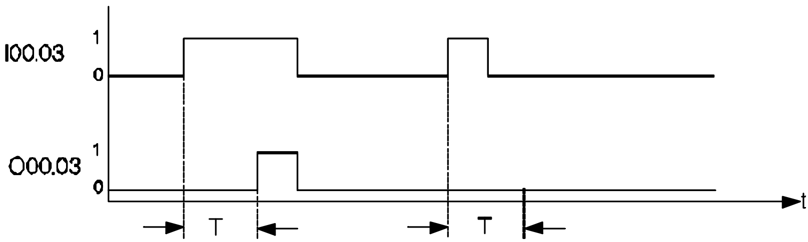
Circuit symbol




Instruction list

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

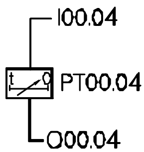
Signal curve




T = set time (here: 1.85s)

## 6.8.5 Falling delay

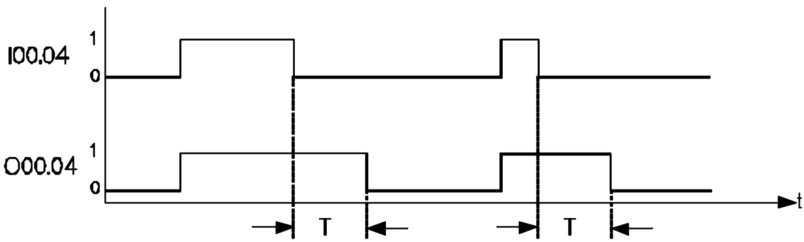
Circuit symbol




Instruction list

| | |
|---|---|
| L | I00.04 |
| = | PT01.04:35*100ms:F |
| L | PT01.04 |
| = | O00.04 |

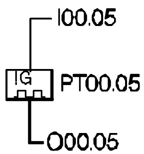
Signal curve




T = set time (here: 3.5s)

## 6.8.6 Pulse generator with wiping pulse output

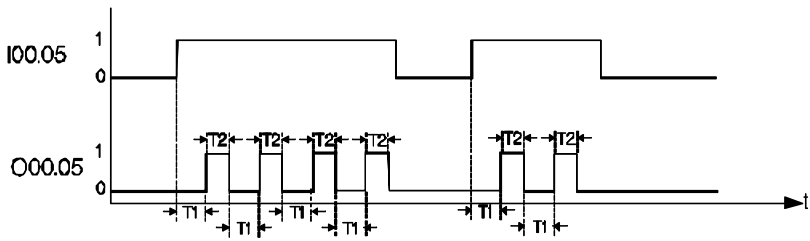
Circuit symbol




Instruction list

| | |
|---|---|
| L | I00.05 |
| AN | O00.05 |
| = | PT00.00:55*10ms:R |
| L | PT00.05 |
| = | O00.05 |

Signal curve

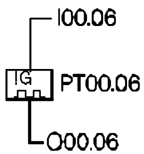



T1 = set time (here: 0.55s)

T2 = cycle time

## 6.8.7 Flash generator with one timer

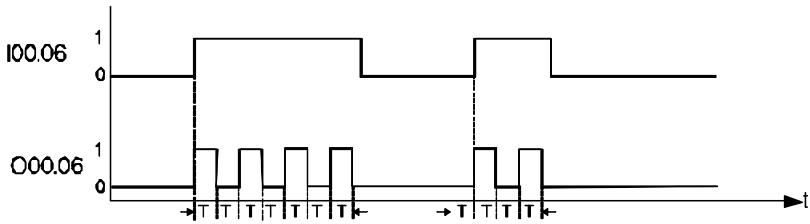
Circuit symbol




Instruction list

| | |
|---|---|
| L | I00.06 |
| = | PT01.06:50*10ms:C |
| L | PT01.06 |
| = | O00.06 |

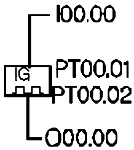
Signal curve




T = set time (here: 0.5s), flashing frequency = 1Hz

## 6.8.8 Flash generator with two timers

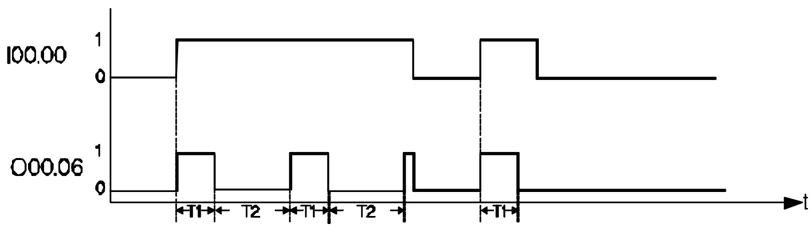
Circuit symbol




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 curve




T1 = set switch-on time (here: 500ms=0.5s)

T2 = set switch-off time (here 1000ms=1s)

## 6.9 Programmable clock

Apart from the software timers there are also four 8-bit operands available which are incremented at set clock pulses.

Operand addresses are PC00.00-PC00.03:

| Operand | Clock pulse | Range |
|---|---|---|
| | 10 ms | 0...255 |
| PC00.01 | | |
| PC00.02 | | |
| | 10 s | |

The pulse markers are incremented by 1 in the range from 0 to 255 at the specified clock pulse. When the count reaches 255, the next clock cycle sets the operand back to 0.

### Application example

One part of the program is to be processed only every 100 ms.

```
L       PC00.01     ;if 100 ms clock pulse memory is
CMP     BM03.14     ;the same as the old value?
JP=                 ;go to end of program 1 if yes
=       BM03.14     ;else new = old
 ″
 ″
″                   ;this part of the program is processed
″                   ;only every 100 ms
″
″
″
LN      O01.03      ;program for 100 ms flash generator
 =       O01.03
 ″
 ″
 ″
 ″
```

```
L     PCxx.xx
=
```

changes the bit marker’s logical state every (128*clock pulse time) because the status of bit 7 in the accumulator is used for bit processing.

# 6.10 Software counters

## 6.10.1 Mnemonics

You can program up to 32 software counters in the range from 1 to 65535. Counter addresses are C00.00-C01.15.

```
|Assignment  |Address  |:Counter val.|:Function  |:Remanence *)
|            |         |             |           |:R remanence
|            |         |             |           |
|            |         |             |:F   count up
|            |         |             |:B   count down
|            |         |
|            |         |16 bit constant (1 – 65535 ) or
|            |         |16 bit variable (e.g. BM01.02 (+BM01.03))
|            |
|            |Address of software counter (e.g. C00.05)
|
|= Starts the software counter at edge 0 → 1, RESET at log. 0
```

*) Note: Adding the "R" parameter (remanence of current counter value) is optional.

| | | | |
|---|---|---|---|
| →**Read output** | L | Cxx.xx | "1"= count complete |
| →**Read current value (remaining value)** | LD | Cxx.xx | 16 bit value of the current count |
| → **Count (transfer pulse)** | L | | stop without RESET |

### 6.10.1.1 Syntax examples

Start forward counter to 175 with remanent current value:

```
=      C00.00:175:F:R
```

Start down-counter with non-remanent, variable counter value (the set value is stored in BM04.06/ BM04.07)

```
=      C00.03:BM04.06:B
```

Transfer counting pulse (count)

```
L      I02.03          ;pulse
=C     C00.03
```

Read counter output (set count complete?)

```
L      C01.00
```

Read count:

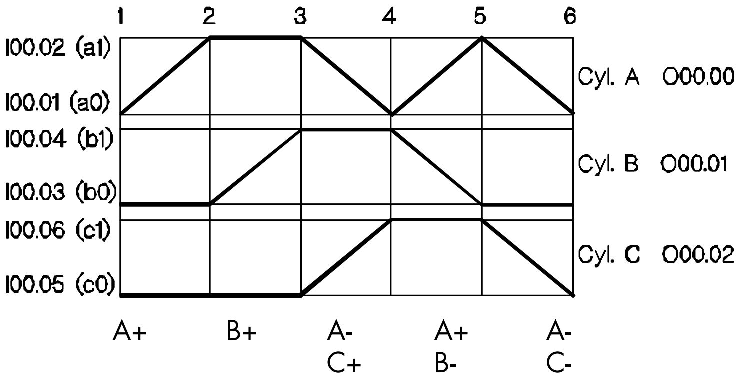
```
LD     C01.00
```

## 6.10.2 Up-counter to 12

```
L      I00.00          ;start counter
=      C00.00:12:V
L      I00.01          ;counter (transfer pulse)
=C     C00.00
L      C00.00          ;read "count complete"
=      O00.12
LD     C00.00          ;read current value
=D     BM00.00
```

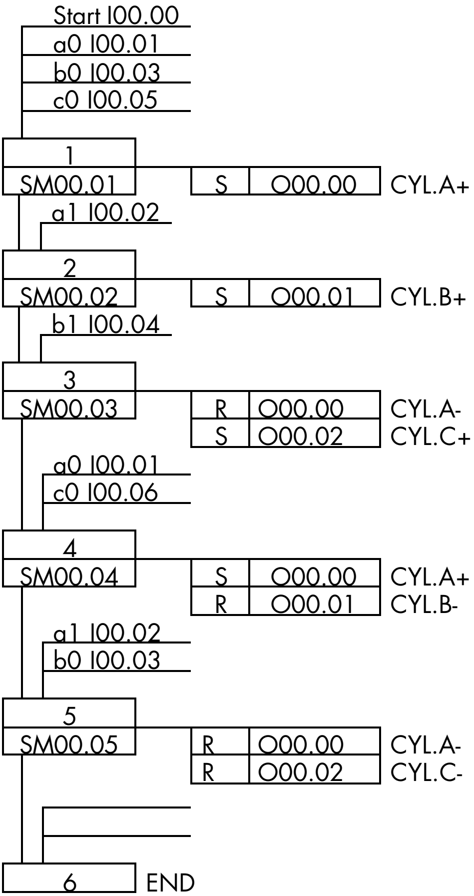
# 6.11 Programming a sequential process

Path-step diagram



Examples

## Logic diagram



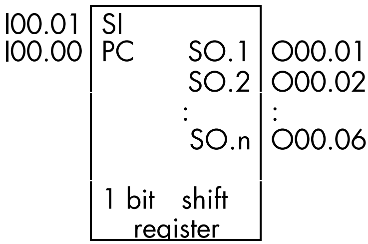

## Program

```
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 A+

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
A    I00.03    ;limit switch b0
A    SM00.04
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 1-bit 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 comes in from I00.00.




Circuit symbol

| | | |
|---|---|---|
| SI: | signal input | I00.01 |
| PC: | | 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     ;wiping pulse
        L       PP00.00     ;wiping pulse
        JPCN    NORM        ;go to normal programif 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 8-bit 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 comes in from I00.00.

Circuit symbol

```
BM00.00 | SI            |
 I00.00 | PC       SO.1 | BM00.01
        |          SO.2 | BM00.02
        |           :   | :
        |          SO.n | BM00.06
        |               |
        | 8 bit   shift |
        |   register    |
```

| | | |
|---|---|---|
| SI: | signal input | BM00.01 |
| PC: | | I00.00 |
| SO.1: | signal output 1 | BM00.01 |
| SO.2: | signal ouput 2 | BM00.02 |
| : | : | |
| SO.n: | signal output n | BM00.06 |

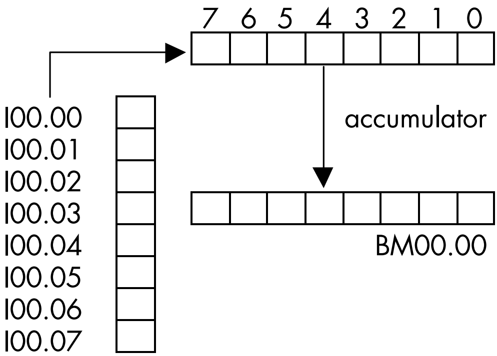
Instruction list

```
        L       I00.00     ;shift clock pulse
        =       PP00.00    ;wiping pulse
        L       PP00.00    ;wiping pulse
        JPCN    NORM       ;go to normal program if not
        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       I00.01     ;signal input
        =       BM00.01    ;step 1
NORM    :
        :                  ;normal program
```

# 6.13 Copy commands (bit-to-byte transfer)

## 6.13.1 Copy eight 1-bit operands to one byte

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




## 6.13.2 Copy one byte to eight 1-bit operands

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

## 6.13.3 Copy sixteen 1-bit operands to two bytes

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

## 6.13.4 Copy two byte to sixteen 1-bit operands

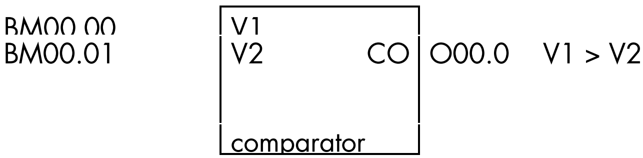
```
LD      BM00.04   ;load contents of BM00.04-BM00.05 into accumulator
C16T1   O00.00    ;copy contents of accu to addresses O00.00-O00.15
                  ;(BM00.04 to O00.00-O00.07,
                  ; BM00.05 to O00.08-O00.15)
```

# 6.14 Comparator circuits

## 6.14.1 8-bit comparator

The program in this example compares the contents of two 8-bit markers. The result (greater, smaller, or equal) is evaluated by conditional jumps (see jump operations). In this case, O00.00 is set if reference value 1 is greater than reference value 2.

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



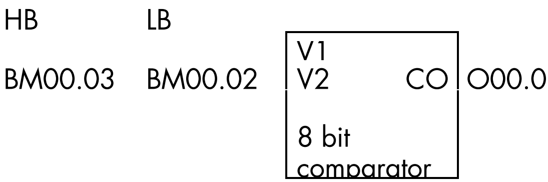

Program

```
        L     BM00.00   ;compare V1
        V     BM00.01   ;with V2
        JP>   MARK1     ;jump if V1 greater than V2
        L     PL00.00   ;log. 0
        JP    MARK2     ;jump to CO
MARK1   L     PL00.01   ;logical 1
MARK2   =     O00.00    ;CO
```

## 6.14.2 16-bit comparator

The program in this example compares the contents of two 16-bit markers. The result (greater, smaller, or equal) is evaluated by conditional jumps (see jump operations). In this case, O00.00 is set if reference value 1 is greater than reference value 2.

| | | HB | LB | |
|---|---|---|---|---|
| V1: | reference value 1 | BM00.01 | BM00.00 | HB: high byte |
| V2: | reference value 2 | BM00.03 | BM00.02 | LB: low byte |
| CO: | comparator output | O00.00 | | |



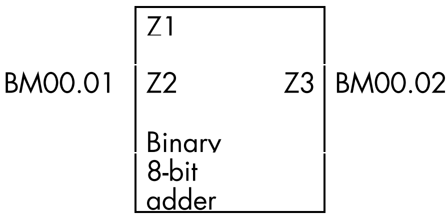

<u>Program</u>

```
         LD     BM00.00   ;compare V1
         CMPD   BM00.02   ;with V2
         JP<    MARK3     ;jump if V1 smaller than V2
         L      PL00.00   ;log. 0
         JP     MARK4     ;to CO
MARK3    L      PL00.01   ;logical 1
MARK4    =      O00.00    ;CO
```

# 6.15 Arithmetic functions

## 6.15.1 Binary 8-bit adder

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



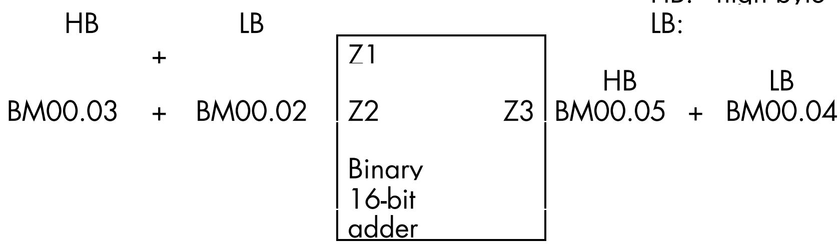

Program

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

## 6.15.2 Binary 16-bit adder

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

HB: high byte
LB:

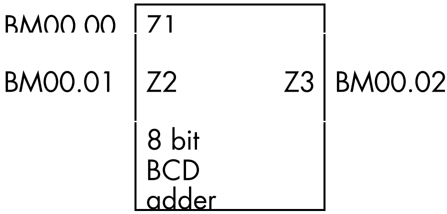



Program

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

## 6.15.3 8-bit BCD adder

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



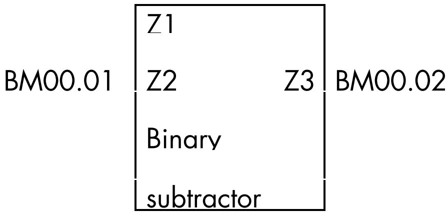

Programm

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

        CLR     LBM00.01   ;marker for BCD correction
        L       BM00.00    ;Z1 1st addend
        A       15
        =       LBM00.00   ;1st decade
        L       BM00.01    ;Z2 2nd addend
        A       15         ;davon 1. Dekade
        ADD     LBM00.00
        CMP     10         ;BCD correction required?
        JP<     ADDIT      ;jump if not
        L       6          ;if so:
        =       LBM00.01   ;load correction
****** Addition ********************************

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

## 6.15.4 Binary 8-bit subtractor

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

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



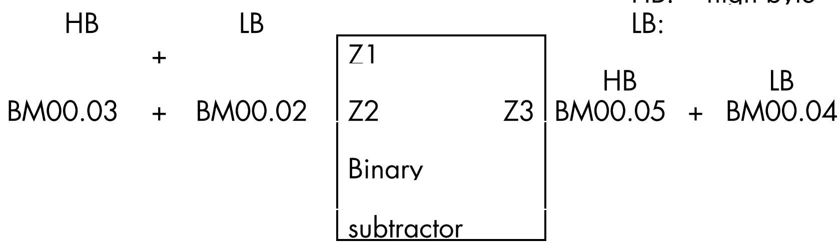

Programm

```
L       BM00.00   ;Z1 minuend
SUB     BM00.02   ;Z2 Ssbtrahend
=       BM00.04   ;Z3 difference
```

## 6.15.5 Binary 16-bit subtractor

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

HB: high byte
LB:

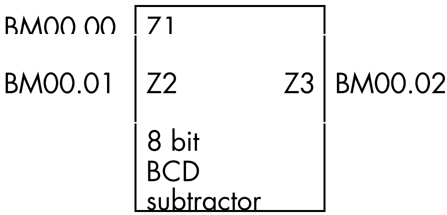



Program

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

## 6.15.6 8-bit BCD subtractor

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



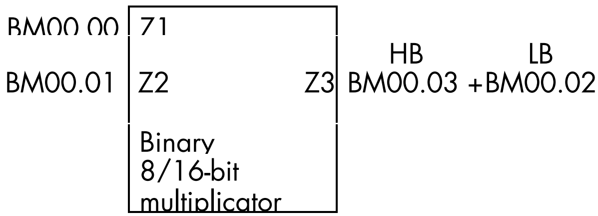

Programm

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

          L       BM00.00    ;Z1 minuend
          A       15
          =       LBM00.00   ;1st decade
          L       BM00.01    ;Z2 subtrahend
          A       15         ;1st decade
          CMP     LBM00.00   BCD correction required?
          JP<=    SUBTR      ;jump if not
          L       BM00.01    ;if so:
          ADD     6          ;load correction value
          =       BM00.01

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

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

## 6.15.7 Binary 8-bit multiplicator

| | | | | | |
|---|---|---|---|---|---|
| Z1: | multiplicand | 8 bit | 0-255 ($FF) | BM00.00 | |
| Z2: | multiplicator | 8 bit | 0-255 ($FF) | BM00.01 | |
| | | | | HB | LB |
| Z3: | product | 16 bit | 0-65025 ($FI01) | BM00.03+ | BM00.02 |

HB: high byte
LB: low byte



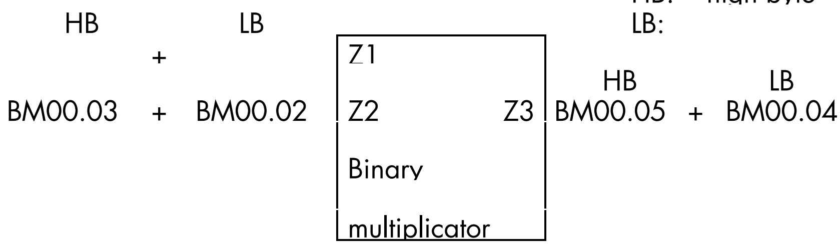

Program

```
L     BM00.00   ;Z1 multiplicand
MUL   BM00.01   ;Z2 multiplicator
=D    BM00.02   ;Z3 product
```

## 6.15.8 Binary 16-bit multiplicator

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

HB: high byte
LB:

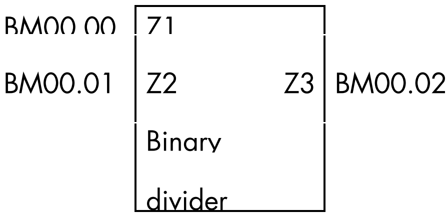



Program

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

## 6.15.9 Binary 8-bit divider

| | | | |
|---|---|---|---|
| Z1: | dividend | 8 bit | BM00.00 |
| Z2: | divisor | 8 bit | BM00.01 |
| Z3: | quotient | 8 bit | BM00.02 |

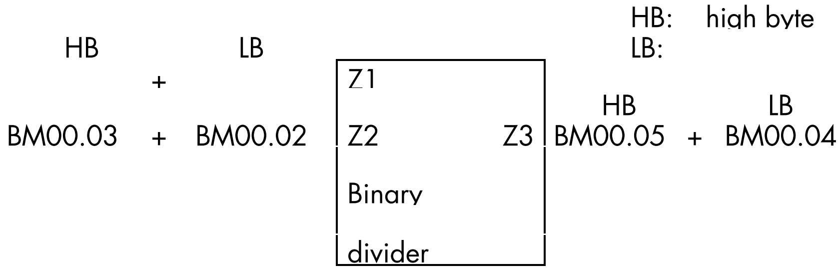



<u>Programm</u>

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

## 6.15.10 Binary 16-bit divider

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




Programm

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

The resulting quotient is an integer number. Proceed as follows to find the rest:

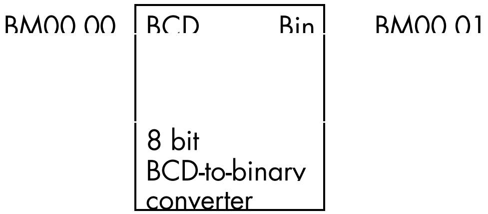
```
LD      BM00.04   ;Z3 quotient
MULD    BM00.02   ;Z2 divisor
=D      LBM00.00  ;Z3(integer!) *Z2
LD      BM00.00   ;Z1 dividend
SUBD    LBM00.00
=D      BM00.06   ;rest
```

# 6.16 Code converters

## 6.16.1 BCD-to-binary converter, 8-bit

BCD: 8-bit 0-99 BM00.00

Binary: 8-bit 0-99($63) BM00.01



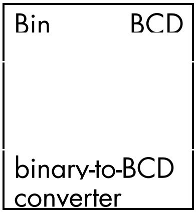

<u>Program</u>

```
L      BM00.00   ;load BCD value
LSR              ;shift
LSR              ;tens
LSR              ;to
LSR              ;ones
MUL    10        ;multiply
=      BM00.01   ;store
L      BM00.00   ;load BCD value
A      15        ;extract tens
ADD    BM00.01   ;add binary tens
=      BM00.01   ;store binary value
```

## 6.16.2 Binary-to-BCD converter, 8-bit

Binary: 8-bit 0-99($63) BM00.00

BCD: 8-bit 0-99 BM00.01



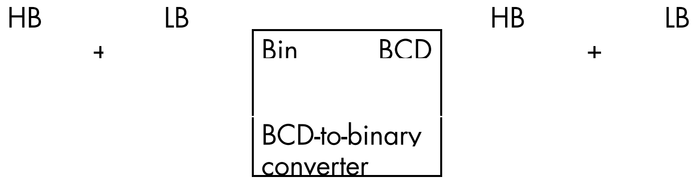

Program

```
L     BM00.00   ;load binary value
DIV   10        ;find and
=     LBM00.00  ;store tens
MUL   10        ;calculate and register
=     LBM00.01  ;integer amount of tens
L     BM00.00   ;
SUB   LBM00.01  ;find and
=     LBM00.01  ;store tens
L     LBM00.00  ;shift
LSL             ;tens
LSL             ;into the upper
LSL             ;nibble
LSL             ;
O     LBM00.01  ;compress and
=     BM00.01   ;output BCD value
```

### 6.16.3 BCD-to binary converter, 16 bit

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

HB:
LB: low byte



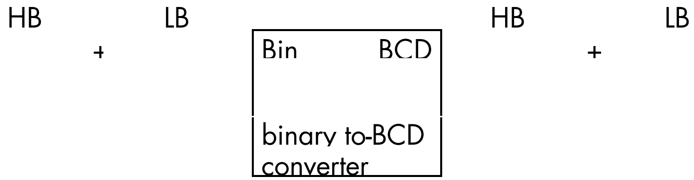
Program

```
CLR     BM00.03    ;clear because of LD BM00.02
CLR     LBM00.03   ;clear because of LD LBM00.02
L       BM00.00    ;separate ones decade
A       15
=       BM00.02    ;binary ones
L       BM00.00    ;separate tens decade
LSR
LSR
LSR
LSL                ;binary tens
MUL     10
ADD     BM00.02
=       BM00.02    ;ones + tens
L       BM00.01    ;separate hundreds decade
A       15
=       LBM00.02   ;binary hundreds
LD      LBM00.02   ;same as word
MULD    100
ADDD    BM00.02
=D      BM00.02    ;ones + tens + hundreds
L       BM00.01    ;separate thousands decade
LSR
LSR
LSR
LSR
=       LBM00.02   ;binary thousands
LD      LBM00.02   ;same as word
MULD    1000
ADDD    BM00.02
=D      BM00.02    ;complete binary value
```

### 6.16.4 Binary-to-BCD converter, 16 bit

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

HB:
LB: low byte



Program

```
        CLR   BM00.02  ;set to zero
        CLR   BM00.03  ;"
THOU1   LD    BM00.00  ;load binary value
        CMPD  1000
        JP<   THOU2    ;smaller than one-thousand ?
        SUBD  1000     ;if yes: subtract 1000
        =D    BM00.00
        INC   BM00.03  ;count subtraction steps
        JP    THOU1    ;check again
THOU2   L     BM00.03  ;if not: shift thousands
        LSL            ;to the upper
        LSL            ;nibble of the
        LSL            ;BCD output's
        LSL            ;high byte
        =     BM00.03  ;prepare high byte
HUND    LD    BM00.00  ;remaining binary value (no thousands)
        CMPD  100
        JP<   TEN1     ;smaller than one-hundred?
        SUBD  100      ; if yes: subtract 100
        =D    BM00.00
        INC   BM00.03  ;count subtraction steps (in lower
                       ;nibble of BCD output's high byte)
        JP    HUND     ;check again
TEN1    L     BM00.00  ;rem. binary value (no hundreds either)
        V     10
        JP<   TEN2     ;smaller than ten ?
        SUB   10       ;if yes: subtract 10
        =     BM00.00
        INC   BM00.02  ;count subtraction steps
        JP    TEN1     ;check again
TEN2    L     BM00.02  ;if not: shift tens
        LSL            ;into the upper
        LSL            ;nibble of the
        LSL            ;BCD output's
        LSL            ;low byte
        ADD   BM00.00  ;remaining ones into lower nibble
        =     BM00.02  ;output low byte
```

# 6.17 Modular programming

### Task

Sets of 12 pieces each are to be transported on a conveyor belt. The belt drive 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. The impulse is used for counting.

A binary display is to show:

- while the belt is moving:

the current number in the set (0…12)

- else:

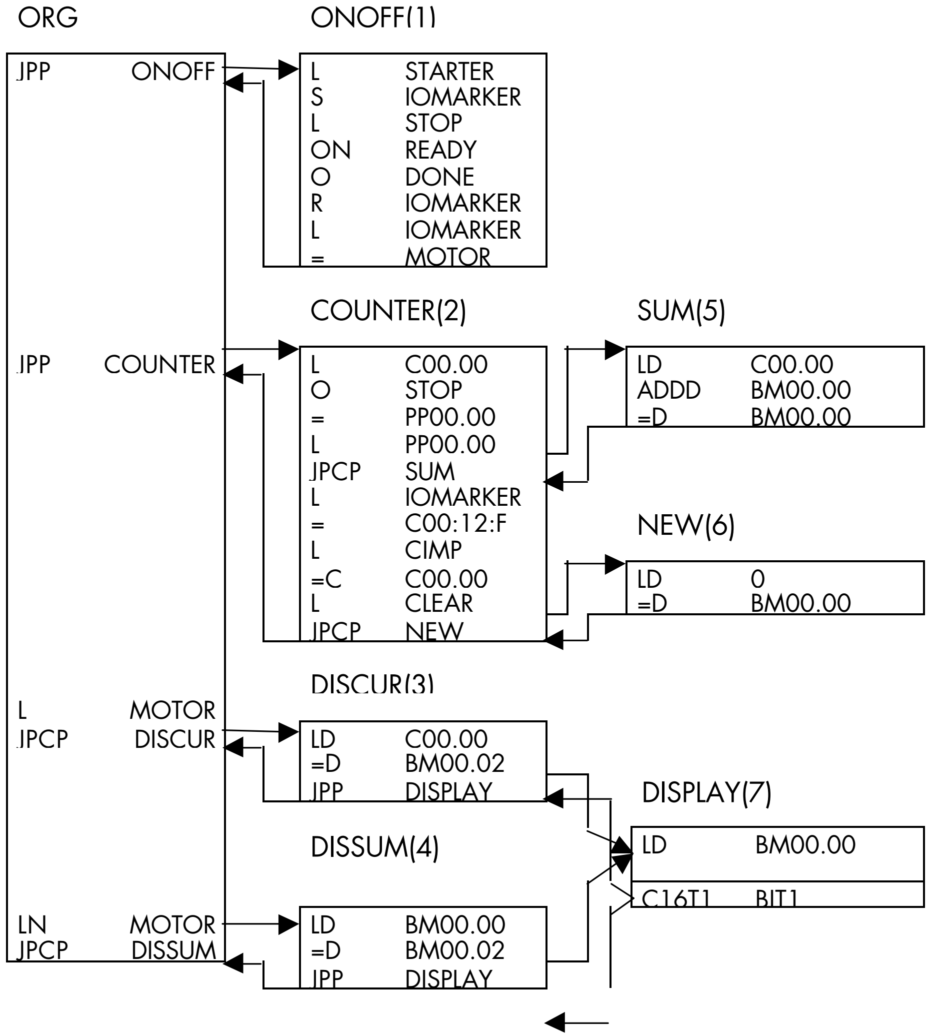
the sum total of all parts transported already (0...65536)

You should be able to reset the counter via the Clear keys.

## 6.17.1 Part task definition

The part tasks are to be defined under technological aspects and aim for clearly arranged modules that can be used several times. Our example only indicates an understanding of the modules' interaction.

### 6.17.1.1 Module structure



## 6.17.1.2 Documentation

```
======== KUBES ========================================================

                        Symbol table

Project  : E556D                  Network  :
                                  created : Jul 20 1998 09:53
User : Virginia Lehmann           changed : Jul 20 1998 09:53
Comment: Example "Modular programming"
======================================================================
Address:    Symbol:     Comment:                           Supplement:
I00.00      START       belt drive on                      X10/1-1
I00.01      STOP        belt drive off                     X10/1-2
I00.02      READY       system ready                       X10/2-1
I00.03      CIMP        counting pulse of initiator        X10/2-2
I00.04      CLEAR       clear sum                          X10/2-3
O00.00      MOTOR       motor protection
O00.01      BIT1        binary display
O00.02      BIT2        binary display
O00.03      BIT3        binary display
O00.04      BIT4        binary display
O00.05      BIT5        binary display
O00.06      BIT6        binary display
O00.07      BIT7        binary display
O00.08      BIT8        binary display
O00.09      BIT9        binary display
O00.10      BIT10       binary display
O00.11      BIT11       binary display
O00.12      BIT12       binary display
O00.13      BIT13       binare anzeige
O00.14      BIT14       binary display
O00.15      BIT15       binary display
O01.00      BIT16       binary display
M00.00      IOMARKER    „motor on/off“ marker
BM00.00     BM00_00     low byte counting register
BM00.01     BM00_01     high byte counting register
BM00.02     BM00_02     high byte display register
BM00.03     BM00_03     low byte display register
C00.00      COUNTER     up-counter
PP00.00     DONE        set count complete
```

```
======== KUBES ========================================================

                        Project structure

Project  : E556D                 Network  :
                                 created : Jul 20 1998 09:53
User : Virginia Lehmann          changed : Jul 20 1998 09:53
Comment: Example “Module programming"
======================================================================
ORG.ORG/1
|
*------>ONOFF.PRO/1
|
*------>COUNTER.PRO/2
|       |
|       *------>SUM.PRO/5
|       |
|       *------>NEW.PRO/6
|
*------>DISCUR.PRO/3
|       |
|       *------>DISPLAY.PRO/7
|
*------>DISSUM.PRO/4
        |
        *------>DISPLAY.PRO/7
```

```
======== KUBES ========================================================

                        Organisation module  IL

Project  : E556D                 Network  :
Module   : ORG       No.: 1      created : Jul 20 1998 09:53
User : Virginia Lehmann      changed : Jul 20 1998 10:27
======================================================================

  1:          JPP     ONOFF               1
  2:          JPP     COUNTER              2
  3:          L       MOTOR           O00.00 ; (motor protection)
  4:          JPCP    DISCUR               3
  5:          LN      MOTOR           O00.00 ; (motor protection)
  6:          JPCP    DISSUM               4
  7:
```

```
======== KUBES =======================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : ONOFF    No.: 1        created : Jul 20 1998 10:40
User : Virginia Lehmann           changed : Jul 20 1998 10:40
Comment: ONOFF
 ======================================================================
  1:          L       START           I00.00 ; (belt drive on)
  2:          S       IOMARKER        M00.00 ; („motor on/off“ marker)
  3:          L       STOP            I00.01 ; (belt drive off)
  4:          ON      READY           I00.02 ; (system ready)
  5:          O       DONE           PP00.00 ; (set count complete)
  6:          R       IOMARKER        M00.00 ; („motor on/off“ marker)
  7:          L       IOMARKER        M00.00 ; („motor on/off“ marker)
  8:          =       MOTOR           O00.00 ; (motor protection)
  9:
```

```
======== KUBES =======================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : COUNTER   No.: 2       created : Jul 20 1998 10:42
User : Virginia Lehmann           changed : Jul 20 1998 10:42
Comment: COUNTER
======================================================================

  1:          L       COUNTER         C00.00 ; (up-counter)
  2:          O       STOP            I00.01 ; (belt drive off)
  3:          =       DONE           PP00.00 ; (set count complete)
  4:          L       DONE           PP00.00 ; (set count complete)
  5:          JPCP    SUM                  5
  6:          L       IOMARKER        M00.00 ; („motor on/off“ marker)
  7:          =       COUNTER:12:F    C00.00 ; (up-counter)
  8:          L       CIMP            I00.03 ; (counting pulse of initiator)
  9:          =C      COUNTER         C00.00 ; (up-counter)
 10:          L       CLEAR           I00.04 ; (clear sum)
 11:          JPCP    NEW                  6
 12:
```

```
======== KUBES =======================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : DISCUR    No.: 3       created : Jul 20 1998 10:45
User : Virginia Lehmann           changed : Jul 20 1998 10:45
Comment: DISCUR
======================================================================

  1:          LD      COUNTER         C00.00 ; (up-counter)
  2:          =D      BM00_02        BM00.02 ; (high byte display register)
  3:          JPP     DISPLAY              7
  4:
```

```
======== KUBES ==========================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : DISSUM    No.: 4       created : Jul 20 1998 10:48
User : Virginia Lehmann           changed : Jul 20 1998 10:48
Comment: DISSUM
=========================================================================

  1:          LD      BM00_00        BM00.00 ; (low byte counting register)
  2:          =D      BM00_02        BM00.02 ; (high byte display register)
  3:          JPP     DISPLAY              7
  4:


======== KUBES ==========================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : SUM     No.: 5         created : Jul 20 1998 10:49
User : Virginia Lehmann           changed : Jul 20 1998 10:49
Comment: SUM
=========================================================================

  1:          LD      COUNTER         C00.00 ; (up-counter)
  2:          ADD     BM00_00        BM00.00 ; (low byte counting register)
  3:          =D      BM00_00        BM00.00 ; (low byte counting register)
======== KUBES ==========================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : NEW       No.: 6       created : Jul 20 1998 10:50
User : Virginia Lehmann           changed : Jul 20 1998 10:50
Comment: NEW
=========================================================================

  1:          LD      0
  2:          =D      BM00_00        BM00.00 ; (low byte counting register)
  3:


======== KUBES ==========================================================

                        Program module  IL

Project  : E556D                  Network  :
Module   : DISPLAY   No.: 7       created : Jul 20 1998 10:50
User : Virginia Lehmann           changed : Jul 20 1998 10:50
Comment: DISPLAY
=========================================================================

  1:          LD      BM00_00        BM00.00 ; (low byte counting register)
  2:          C16T1   BIT1            O00.01 ; (binary display)
  3:
```

# 7 Troubleshooting

## 7.1 “Failure“ LED flashing? → Short circuit

- Indication:
  “failure“ LED: flashing red light
- Cause:
  Short circuit or overload at an output.
- Reaction:
  All outputs are disabled.
- Corrective action:
  - Find short circuit (e.g. by disconnecting all outputs and reconnecting them one by one).
  – Remove short circuit
  – Restart PLC

## 7.2 LEDs „run/stop“ and „failure“ light up red → Undervoltage

- Indication:
  „run/stop“ LED: permanent red light
  „failure“ LED: permanent red light
- Cause
  The system supply voltage falls below a threshold somewhere between 16 and19 V.
- Reaction:
  The user program stops, all non-remanent operands and outputs =0.
- Corrective action:
  - Switch supply voltage off and back on again.

## 7.3 No online connection to KUBES?

The following error message may be displayed when you are trying to go online with the PLC (via V.24):

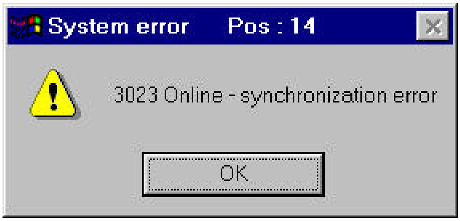



*Fig. 14: V.24-synchronisation error message*

If it does, please check whether:

- the PLC is switched on,
- the programming cable is connected to the PLC,
- the programming cable is properly connected to the PC (check port! the standard port is COM1),
- the cable is a genuine KUBES programming cable (part no.: 657.151.03).

If all of the above points are okay, but the PLC still does not react, it could be that the PLC no longer accesses the port.

- Switch the supply voltage off and back on again.

In some cases, the PLC still does not react. The following causes are possible:

- PLC defective
- program error (CPU no longer accepts KUBES' online message)
- wrong V.24 parameter settings

## Ultimate chance of correcting the fault

- Switch off all supply voltages, i.e. both the system supply and the supply of the outputs (→ 3.4).
- Take off the lid of the housing
  The lid snaps into the device's side walls. Carefully push out one side wall to unlock the lid so that you can take it off.
- Pull off the jumper located above the V.24 interface connector.
- Switch on the system supply.
  ➔ the PLC indicates "stop" (→ 3.8). (Repeat the procedure if not.).
- Choose "Online V.24" in KUBES.

  *Hand in the PLC for repairs if there's still no online connection.*
- Choose "Delete program".
- Transmit a new and unbugged program.
- Switch the power supply off.
- Put the jumper back in.
- Close the lid.
- Switch all supply voltages back on.

# 8 Data summary

## 8.1 Technical data

### 8.1.1 Design

| | |
|---|---|
| Type | open |
| Dimensions (L x W x H) | depend on model variant |
| Eco Control 667E 8/8 | 152 x 90 x 73 mm |
| Eco Control 667E 16/16 | 152 x 90 x 73 mm |
| Eco Control 667E 32/32 | 268 x 90 x 73 mm |
| Installation | on carrier rail |
| Weight | depends on model variant |
| Eco Control 667E 8/8 | c. 570 g |
| Eco Control 667E 16/16 | c. 580 g |
| Eco Control 667E 32/32 | c. 970 g |
| Admissible ambient conditions | |
| Storage temperature | -25...+70 °C |
| Ambient temp. during operation | 0...55 °C |
| Relative humidity | 50...95% |

### 8.1.2 System power supply

| | |
|---|---|
| Voltage | 24 V DC -20% / +25% |
| Power consumption | 100 mA |
| Connectors | clamp-screw term. up to 2.5mm² |
| L1+ | + 24 V DC |
| L1- | 0 V |

### 8.1.3 System status indicators

| | |
|---|---|
| Type | light emitting diodes, class 1 (in acc. with EN 60825-1) |
| Run/stop (duo-LED, green/red) | program running/stopped |
| Failure (LED, red) | failure indicator |

### 8.1.4 Serial interface

| | |
|---|---|
| Type | V.24 (RS 232) |
| Connector | female, 9-pin D-Sub |
| Function | programming and data communication |
| Maximum baud rate | 9.6 kbit/s |
| Transfer format | 8 data bits, 1 start bit, 1 stop bit |

### 8.1.5 Programming

| | |
|---|---|
| Programming device | PC with MS®Windows |
| Programming software | KUBES (version 5.30 or higher) |
| Programming cable | 657.151.03 |

## 8.1.6 Digital inputs

| | |
|---|---|
| Provided | via internal process image |
| Amount | depends on model variant |
| Eco Control 667E 8/8 | 8 |
| Eco Control 667E 16/16 | 16 |
| Eco Control 667E 32/32 | 32 |
| Type (in acc. with IEC 1131) | 1 |
| Galvanic separation | none |
| Indicators | light emitting diodes, class 1 (in acc. with EN 60825-1) |
| Colour | green |
| Tapping point | in input circuit |
| Signal state | 1: LED on<br>2: LED off |
| Addressing | depends on model variant |
| Eco Control 667E 8/8 | I00.00...07 |
| Eco Control 667E 16/16 | I00.00...15 |
| Eco Control 667E 16/16 | I00.00...15, I01.00...15 |
| Input voltage | 24 V DC -20%/+25% (inc. residual ripple) |
| Surge immunity | ≤ 40 V DC (≤ 30 min) |
| Signal detection | |
| Logical 0 | ≤ 5 V DC |
| Logical 1 | ≥ 15 V DC |
| Power consumption/input | max. 10 mA |

## 8.1.7 Digital outputs

| | |
|---|---|
| Control | via internal process image |
| Amount | depends on model variant |
| Eco Control 667E 8/8 | 8 |
| Eco Control 667E 16/16 | 16 |
| Eco Control 667E 32/32 | 32 |
| Type | semiconductor |
| Indicators | light emitting diodes, class 1 (in acc. with EN 60825-1) |
| Colour | red |
| Tapping point | in the load circuit |
| Signal state | 1: LED on<br>2: LED off |
| Addressing | depends on model variant |
| Eco Control 667E 8/8 | O00.00...07 |
| Eco Control 667E 16/16 | O00.00...15 |
| Eco Control 667E 32/32 | O00.00...15, O01.00...15 |
| Output supply | 24 V DC -20%/+25% |
| Connectors | clamp-screw term. up to 2.5mm$^2$ |
| L1+ | + 24 V DC |
| L1- | 0 V |
| Output current/output | max. 0.5 A |
| Short-circuit protection | yes |

## 8.1.8 Processor and memory

| | |
|---|---|
| Microprocessor | 80C535 |
| Memory | |
| Operating system | Flash-EPROM |
| User program | NV-RAM, 32 kbyte |
| Data, remanent | NV-RAM, 8 kbyte |
| Data, non-remanent | S-RAM, 24 kbyte |

## 8.1.9 Operands

| | |
|---|---|
| Programmable timers | remanent if required |
| Amount/range | 32/10 ms ... 65535 s |
| Programmable counters | remanent if required |
| Amount/range | 32/0...65535 |
| Inputs and outputs | → 8.1.4 und 0 |
| Bit markers | 1320, inc. 512 remanent |
| Byte markers | 2816, inc. 2304 remanent |

## 8.2 Order specifications

### 8.2.1 Controllers

| *Product* | *Part number* |
|---|---|
| Eco Control 667E 8/8<br>8 digital inputs, 8 digital outputs | upon request |
| Eco Control 667E 16/16<br>16 digital inputs, 16 digital outputs | 667.752.00 |
| Eco Control 667E 32/32<br>32 digital inputs, 32 digital outputs | 667.704.00 |

### 8.2.2 Accessories

| *Product* | *Part number* |
|---|---|
| Simulator plug for 8 digital inputs | 667.155.50 |
| Starter kit Eco Control 667E, German, containing:<br>KUBES light (programming software just for Eco Control 667), programming cable 657.151.03, instruction manuals E 327 D and E 556 D.<br>→ Available as from week 44/98 | 667.502.00 |
| Starter kit Eco Control 667E, English, containing:<br>KUBES light (programming software just for Eco Control 667), programming cable 657.151.03, instruction manuals E 327 GB and E 556 GB.<br>→ Available as from week 51/98 | 667.502.11 |

# 9 Index

# 10