Table of Contents

1. Introduction to B-CONW for Windows.

B-CONW for Windows is a programming language, designed to allow engineers to create control sequences and configuration software for the System 2000 and RTU8/RTU-COM/RTU-870 range of products from Brodersen Control Systems A/S.

B-CONW can be used by both the engineer, who may not be formally educated in programming, and the experienced programmer alike. B-CONW unlocks the power of the System 2000 and RTU modules, enabling them to perform complex control functions, alarm monitoring and in the case of the RTU8/RTU-COM compact outstation, the logging of data.

The IEC1131 standard for programming software defines a range of software with the basic idea of achieving some common standards being different hardware/software vendors. B-CONW uses the IEC1131-3 Instruction List standard.

The B-CONW software development package is run from a PC, where the programme is created, compiled into code, capable of running in the System 2000 and RTU modules, tested for correct operation, and finally downloaded into the module.

In the development of B-CONW great importance has been attached to achieving an efficient programming cycle, editing, compiling, and testing.

The creation of a programme consists of the following steps.

- The programme is created using the Instruction List format in the editor, to create the source code.
- The source code is then translated into a binary code by the compiler.
- The code can then be tested for correct operation using the Debug facilities.
- The resulting code is then downloaded, via the COM port or via the field bus using the IOTOOL32. (Please see manual for information about IOTOOL32.)

Licence conditions.

Which types of Brodersen modules can be programmed?

The B-CONW can be used to programme Brodersen modules which contain the B-CON facility.

Brodersen modules with B-CON facility:

BITBUS slave modules with I/O with revision numbers 3.00 or higher.
BITBUS slave modules with I/O containing /DL
RTU modules with type numbers containing UCR
REMOTE master modules with type numbers containing /RM

1.1 Installation/Get started.

The B-CONW Programming Tool is a bundled part of the IOTOOL32 Pro software package.
To install the B-CONW you must install the full IOTOOL32 Pro package using the installation programme supplied on the CD ROM.

Please refer to the IOTOOL32 Pro Programme Installation described in the IOTOOL32 Pro User's Guide supplied with the IOTOOL32 Pro Package.

Start up of B-CONW:

B-CONW is started from the I/O Explorer by clicking on the icon B-CON.

2. Programming.

2.1 Data types and formats

A few basic points have to be explained before starting the programming:

- **Specification of data types and formats**
- **Numbering of input and output**
- **Use of symbols**
- **Set up of data which should be available on the serial port of the module for communication with external devices (not RM and BM modules)**

Specification of data types and formats.

B-CONW operates with **3 different data formats**:

-	1 Bit	Basic logic unit	Value range: 0 or 1
B	1 Byte = 8 Bits		Value range: -128 to 127
W	1 Word =2 Bytes=16 Bits		Value range:-32768 to 32767

B-CONW operates with **5 different data types:**

I	Input variable
O or Q	Output variable
M	Memory marker (internal relay/register)
C	Constant

Constants can be entered as **different value types**:

Bit constants can be entered as **1 or 0 without extension**.
Byte/Word constants can be entered as **decimal value without extension or hexadecimal value with extension H.**

Examples:

I	= Bit input
BI	= Byte input
WO	= Word output
C1	= Bit constant value1
BC100	= Byte contant decimal value 100
WC4376	= Word constant decimal value 4376
WC0fffH	= Word constant hexadecimal value 0fff (Decimal 4095)

2.2 Numbering of inputs and outputs

The inputs and outputs of the Brodersen modules with B-CON facility are **separated in 8 types:**

DI	**DO**	Digital inputs/outputs
AI	**AO**	Analogue inputs/outputs
ZI	**ZO**	Aux. inputs/outputs
YI	**YO**	Aux. inputs/outputs (communication registers)

Each **input/output type** has its **own numbering range** for addressing.

Numbering range for:

DI/DO	0-1999	Bytewise
AI/AO	2000-3999	Bytewise
ZI/ZO	4000-5999	Bytewise
YI/YO	6000-7999	Bytewise

Examples of syntax for addressing:

I0.2 Digital input bit 2 on input byte 0

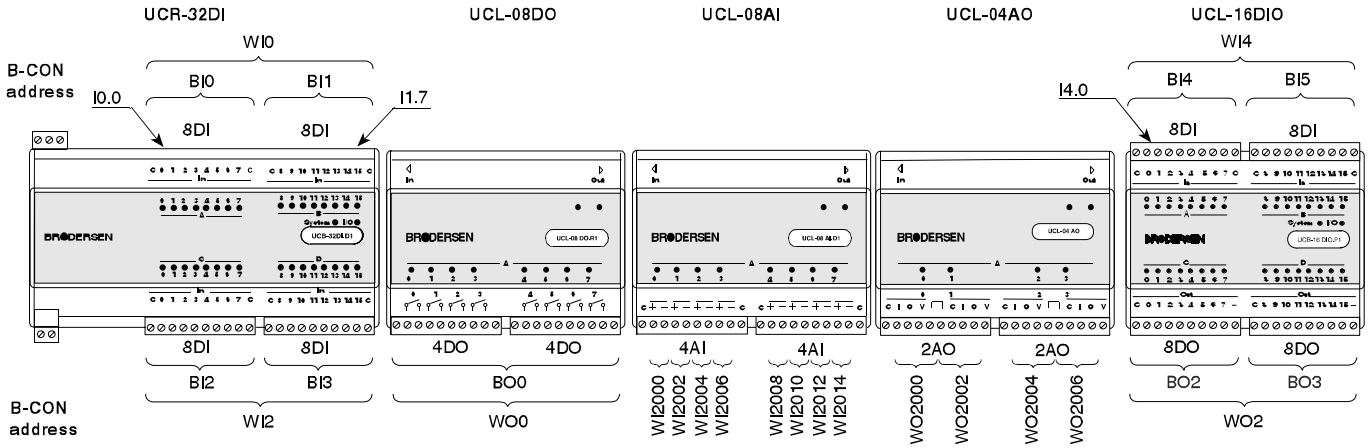
BO0 Digital output byte 0

WI0 Digital input word 0. **Covers both input byte 0 AND byte 1**

WI2 Digital input word 2. **Covers both input byte 2 AND byte 3**

WI2000 Analogue input 0

Example of an island



2.3 Use of symbols

To ease the overview of a source code, it can be an advantage to use symbols instead of the basic names and numbers described earlier.

A symbol can be a synonym for an input, output address, or a memory marker.
To define symbols, the following syntax has to be used:

```
#define    <symbol> <I/O address or memory marker>
```

Example:

```
#define    pump      i0.3
#define    temp      wi2000
#define    scale     wm22
```

"pump", "temp" and "scale" can now be used in the B-CON programme instead of the standard I/O addresses and memory markers and ease both the overview of writing the B-CON source code and the readability afterwards.

The define statements should be entered **before** the "main:" label. (See 2.5 for “How to build a B-CON programme”)

2.4 Set up of data

Set up of data which should be available on the serial port of the module for communication with external devices (not RM and BM modules).

Communication with the Master (typically a PC).

To communicate with the Master it is necessary to define which data should be accessable from the Master to and from the B-CON slave, Remote slave, or RTU8 via the communication port on the module (Bitbus or Modbus).

The syntax for definition of the accessable data is:

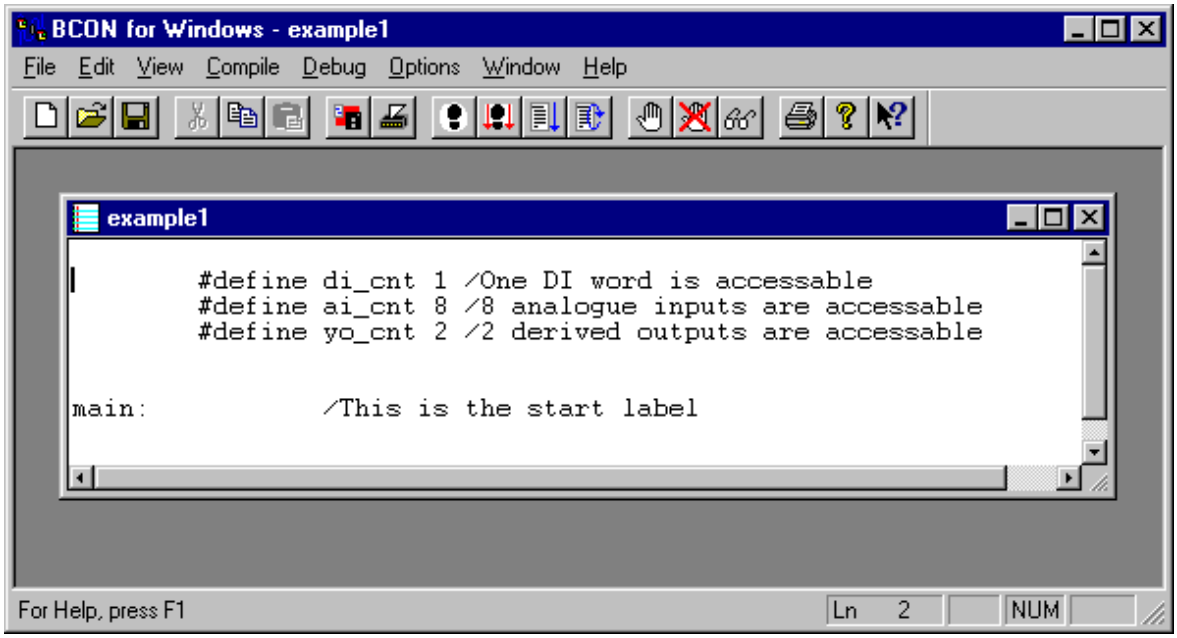
```
#define  <type of data>_cnt <number of words>
```

Example:

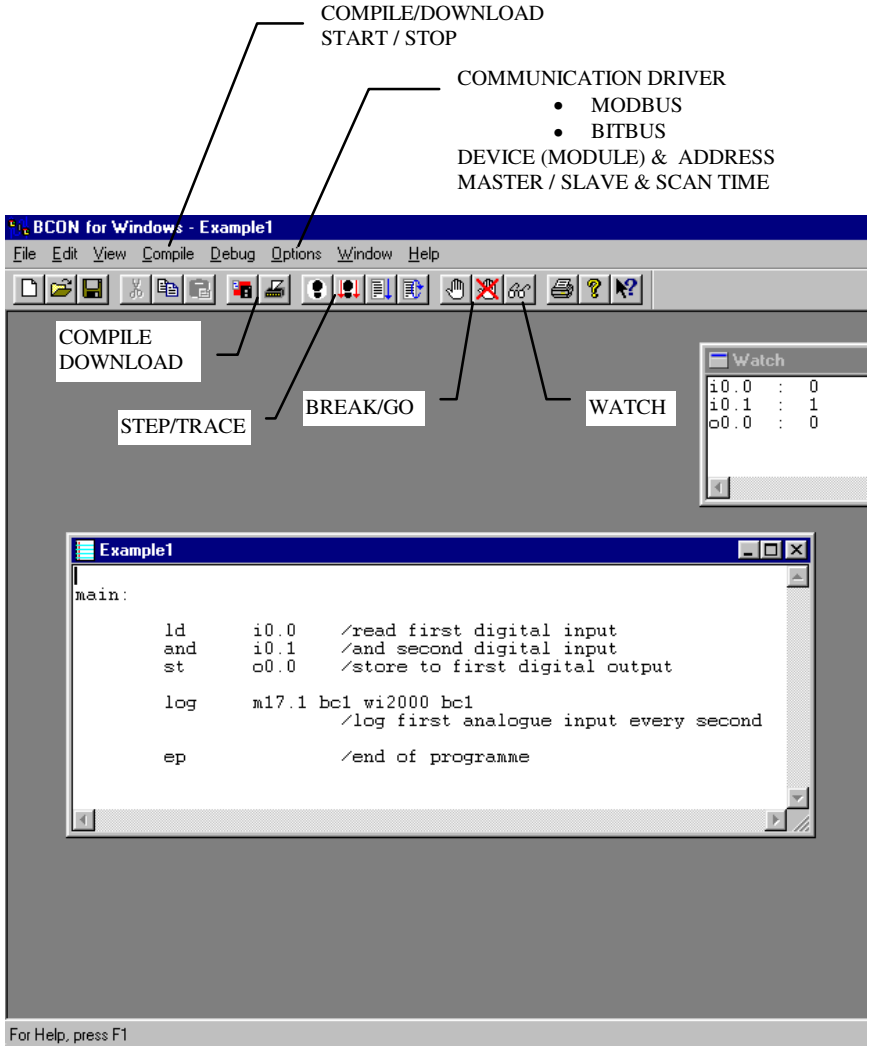
```
#define di_cnt 1
#define ai_cnt 8
#define yo_cnt 2
```

These define statements should be entered **before** the "main:" label. (See 2.5 for “How to build a B-CON programme”)

Example:

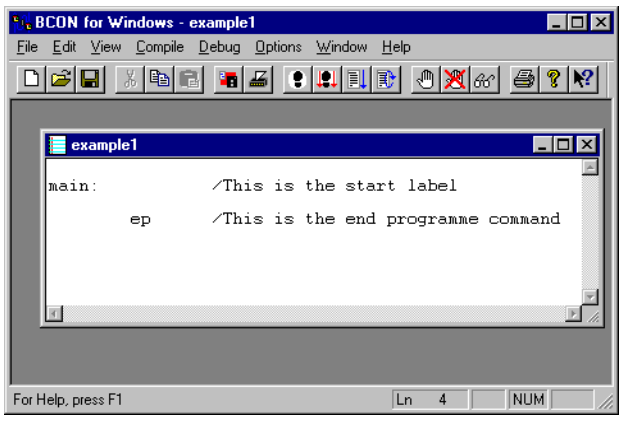


2.5 B-CON Environment/editer



How to build a B-CON programme.

Click the icon "File" and choose "New".
You can now start editing the instruction list source code for your B-CON programme.
The label "**main:**" and an end command "**ep**" frames the programme. (See example below).

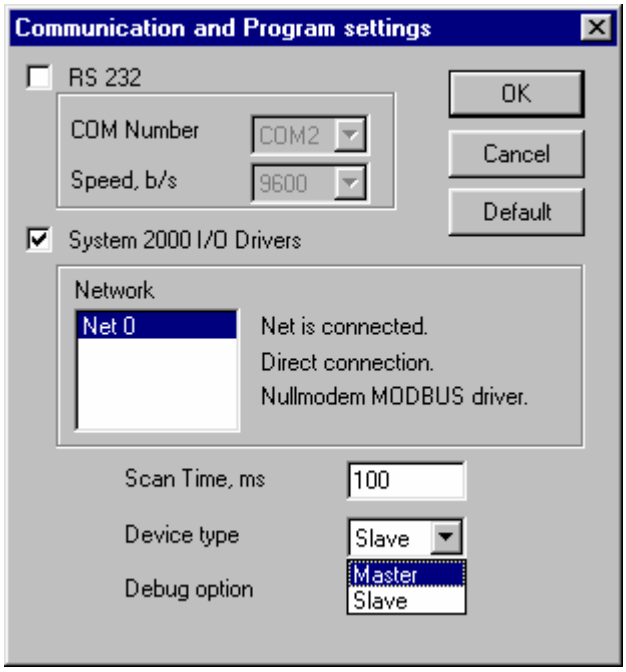


Comments can be added to the source text as documentation for the code.
All comments should start with "/". Any text entered after a "/" in a line will not be compiled to CPU code.

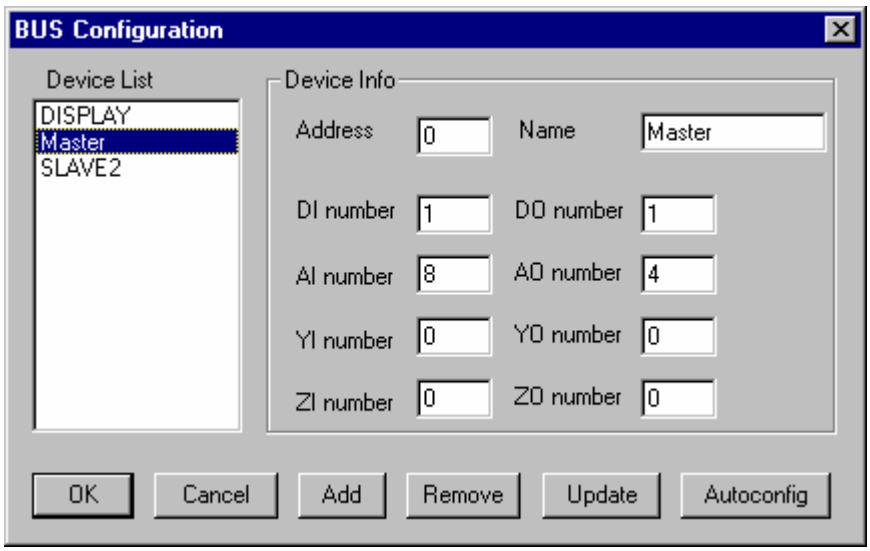
Important features when programming a Master.

When programming a Master you must first make a "Device Table" for the modules which are connected to the Network.

1. Choose the menu **"Options"** and choose **"Communication and Program"**.
   Select "Master" and Click "OK".



2. Enter **"Options"** again and choose **"Device"**.
   For the Master and each slave a table has to be filled out. See Example.

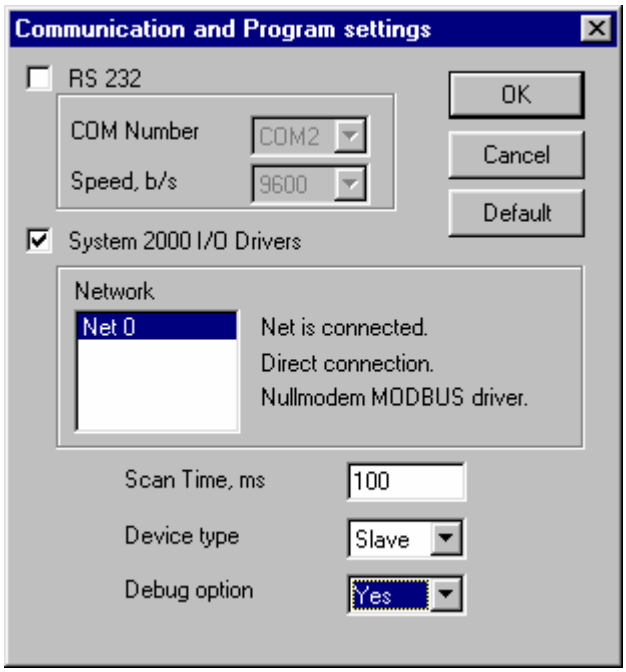


Please note that the Master **must** have the address No. 0 in the device table. Address switch on the module should be set to 1.
The numbers which must be entered are the number of words of each type.
If all modules are connected you can also let the B-CON programme read the configuration if you press **"Autoconfig"**.

2.6 Debugging a B-CON programme.

To debug your B-CON program you can use the debug facilities available in the “Debug” menu.

Note! Before you enter the “Debug” menu you must choose “Debug Option” in the “Communication and Program” menu under “Option”.



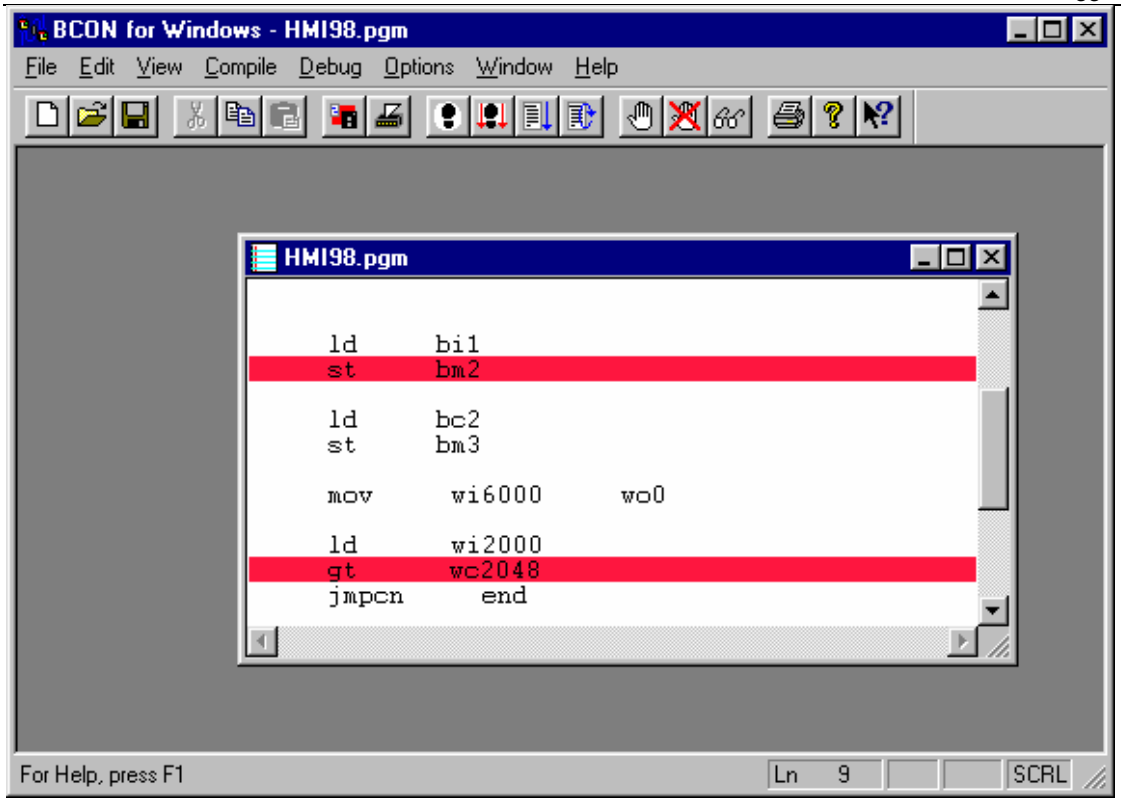
When choosing “Yes” for “Debug option” only a part of the application program will be down loaded to the System 2000 module, and the execution of the application program will be carried out in the PC instead.

In the “Debug” menu you will find different ways of executing the B-CON application:

STEP:	Each time F7 is pressed, the next line in the B-CON program is executed.
TRACE:	When F8 is pressed a slow execution of the B-CON application is started. In “Trace” mode it is possible to follow the execution of the B-CON program line by line.
RUN:	Execution of the program as fast as possible in the PC.
RESTART :	Press shift F2 to restart the execution of the application program.
STOP DEBUGGER:	Stops the Debug function.

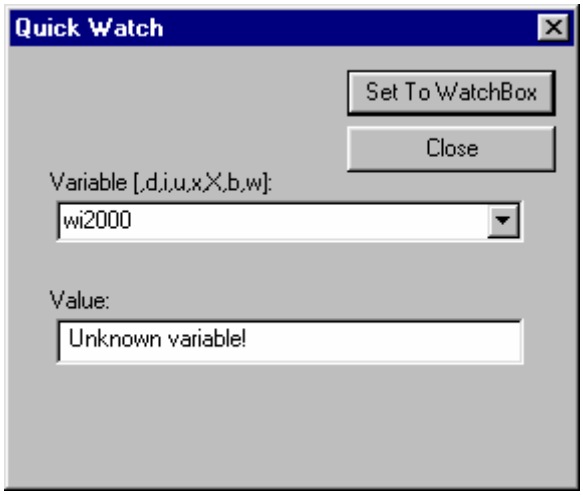
To help debugging it is possible to make **break points** in the program to stop the **“Trace”** function at specific lines in the program.

Place the cursor in the line where the **break point** should be and press **F4.**



To monitor the M registers or inputs or outputs it is possible to activate Watch windows which will show the contents of the chosen registers during the execution of the application program.

To set up a Watch window you can either **press shift F7** or select the menu **"Watch"**.



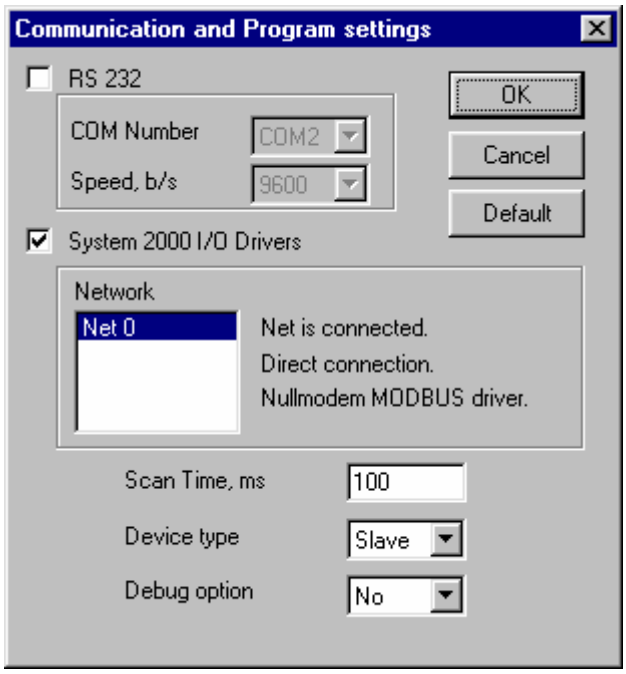
You can now enter the names of the variables you want to watch/monitor as they are placed in a separate window on the screen.

2.7 Compilation and Download.

After finishing the Instruction list programming you should convert the code for use in the Remote Slave, RTU8, or I/O slave module with B-CON facility.

It is carried out if you follow the below procedure:

1. Click the menu **"Option"** then **"Communication and Program"**.



2. Choose the right communication way.

   "RS232" is used if you want to **download directly via one of the COM ports** (used if no System 2000 drivers are installed).

   "System 2000 I/O Drivers" is used if you want to **download via one of the IOTOOL32 drivers.**

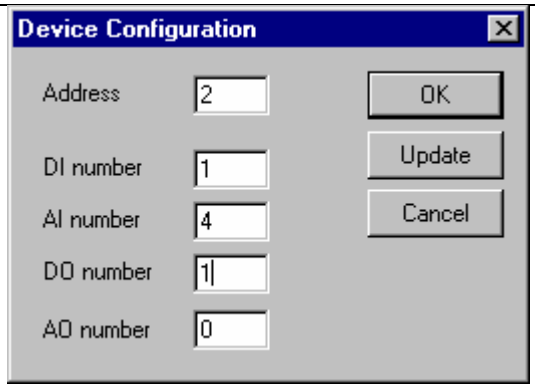
3. Check the settings and press **"OK"**.

4. Choose the right **"Device type"**, **Master** or **Slave** as well as if you want to operate with the **"Debug option"** or not.

5. If you choose **"Slave"** you have to specify the **"Scan time"** for your B-CON programme.

6. Click **"OK"** when the right choices have been made.

7. Before downloading to a **slave**, it is necessary to choose the correct slave **address**.
   Click **"Options"** and enter **"Device"**.
   Enter the address of the slave you want to download to and click **"Update"**.
   Click **"OK"** when the configuration of the Slave module has been read.



When downloading to a Master, keep the **"Device table"** already defined in chapter 2.5 **"Programming a Master"**.

8. Enter the menu **"Compile"**. Choose the command "Compile".
Correct the errors if the compiler returns an Error file and repeat the compilation procedure until the programme is error free.

9. When the programme is compiled and free of errors, you can use the **"Down load"** command to transfer it to the module.

10. Use the **"Start"** command to activate the down loaded programme in the module.

3. Programming

3.1 Description of the Language

IEC 1131 is an international standard for the development of PLC controllers and PLC programming languages. It was agreed upon in 1990 and is generally applied for new PLC systems.
The standardization of instructions and command patterns contributes to the fact that programmers do not need to change their familiar programming techniques when a PLC system is changed.

The programming language was developed according to direction IEC 1131 of working group 65A, which sets the standard for programmable controls. The standard comprises 3 types of presentation:

- Structured text
- Instruction list
- Function sets (macros)

In the following description of the **instruction language**, all the available IEC 1131 commands that are supported by B-CON are discussed in detail.

3.1.1 Organization of Programs

A control program consists of a number of instructions. Each instruction is terminated with <RETURN>.
An instruction line can be supplemented by a comment.
In editing a program it must be noted that there must be an indentation of at least one <SPACE> or better a tab before an instruction is written. In this free place the editor expects a jump label.

Commands
An instruction contains an OPERATOR (command) along with an optional MODIFIER, then follows a <SPACE> or a <TAB> and, if necessary, one or more operands separated from each other by commas or <SPACE>.

Comments
A comment can be written either before the beginning of the program or in the instruction after the last element in the line. It is introduced by the sign (/).

Labels
An instruction can be preceded by a label for the identification of this place. A label can have a length of up to 8 characters including the end character. Its first sign must be a letter, not a figure. Special characters are not permitted. The label must be terminated by a colon as end character (:). In the jump instruction, however, this colon must not be contained.

Example, section of the program:

```
        LD IO.1         /load input variable
        JMPC MARK1      /jump target without colon

MARK1   LD  M1.1        /jump label with colon
        AND M5.5
```

The instruction (instruction line) was preceded by a mark called „MARK1“.

Data types of operands
Operands can take the following data types:

- Input variable
- Output variable
- Memory variable (marker variable)
- Constant

The range of values for a variable depends on the selected operand format (bit, byte or word format).

Formats of operands
Operands can have 3 data formats:

Bit It has the data length of 1 bit. This data format has 2 values, namely „logical true“ = 1 or „logical false“ = 0.

Byte It has the data length of 8 bits, i.e. 1 Byte. This data format has a value range between +127 and -128, each value with a sign.

Word It has the data length of 16 bits, i.e. 2 bytes. This data format has a value range between +32767 and -32768, each value with a sign.

Define symbols
The application programmer can define symbolically his own operand characteristics for variables and constants via the operator #DEFINE. Thus he produces a relation to the process to be operated. In this way the control program becomes more comprehensible.

It is a common practice to place DEFINE directives at the beginning of the program. Thus the programmer has good control of the process inputs, process outputs and the assignment to markers. DEFINE directives can also be given within a program. This, however, is not recommendable.

Examples:

```
#DEFINE terminator switch_S1   I1.1/assignment input bit I1.1
#DEFINE overload forward current_F1   I1.2/assignment input bit I1.2
#DEFINE pump_M1 O2.0 /assignment output bit O2.0
```

Application programming commands
The commands of the programming language according to IEC 1131 are described in detail. For each command a sample program of general validity was developed referring in its application to a control module with input and output ports.

In order to access control modules from the B-CON card a reference protocol has to be defined in the program head declaring the symbolical name of the device and the name of the module control program as illustrated above.
When inputs, outputs or markers of a module are to be operated from the master card, the symbolical name of the device must always precede the variable definition separated by the point operator.

In the following the changes are shown necessary to transform a program of general validity (B-CON program) with module-integrated inputs and outputs into a program capable of running under B-CON M:

B-CON sample program:

```
/P_AND.PGM
/Example: AND conjunction in bit format

ld   i0.1   /OP1: load input
and  i0.2   /OP2: AND conjunction OP1 with OP2
st   o0.0   /RES: store in output
ep          /end of program
```

Logic instructions
The current programming language according to the directions of IEC 1131 provides an extended set of instructions which, contrary to other PLC languages, permits instructions with nesting and the application of operands as stack.

The nesting rule consists in an instruction which represents the result of operand 1, operator (command) and operand 2.
The general nesting rule is determined as follows:

Result:= operand 1 operator operand 2

<ACC> <ACC>

Example: (after preceding DEFINE directives!)

```
LD   OPERAND1   /load OPERAND1 in ACC
AND  OPERAND2   /AND nesting with OPERAND2
ST   RESULT     /output of result
```

Operand 1 must principally be loaded into the ACC before the execution of a command or it is deliberately used as the result of a previous operation.

According to the instruction this contents of the ACC is then connected with operand 2 and kept in the ACC as the new result.

Stack operations (without operand signifies)
Instructions (operations) with stack operands are a special option in the programming language at hand.
When there is no following operand or only a format signifies in an instruction, it is understood that this operand (operand 2) is in the stack and a connective of the contents of the ACC and the stack is produced. Such instructions are permitted in bit, byte and word format.
The allocation of the operands (which is which?) differs from the general pattern of instructions (sequence of operand allocation) in operations with a stack operand.

Example: (after previous DEFINE directives!)

```
LD   OPERAND1  /load OP1
LD   OPERAND2  /load OP2 as stack operand
AND  B         /AND connective OP1 and OP2 as stack
 operand
ST   RESULT    /output of the result
```

Instructions for the different formats

Bit format	AND	/without operand signifier
Byte format	AND B	/operand sifnifier B=byte
Word format	AND W	/operand signifier W=word

Operator modifiers
Operator modifier „N“ refers to the boolean negation of operand 2, which is carried out before the execution of the instruction.

Example: (after previous DEFINE directives!)

```
LD       OPERAND1     /load OPERAND1 into ACC
ANDN     OPERAND2     /AND connective with OPERAND2,
                      /which is negated before the operation
ST       RESULT       /output of the result
```

Jump label modifiers
Jump operators can be modified with modifiers „C“ and in combination with „CN“. Modifier „C“ means that the jump instruction is carried out when the result of the connective is „logical true“, i.e. the boolean value is „1“ (contents of the ACC).

Modifier „CN“ means that the jump instruction is carried out when the result of the connective is „logical false“, i.e. the boolean value is „0“ (contents of the ACC).

Examples:

```
JMPC  MARK1/jump to MARK1 when
          /ACC contents is „1“
          /i.e. condition „TRUE“
JMPCN MARK1/jump to MARK1 when
          /ACC contents is „0“
          /i.e. condition „false“
JMP   MARK3/jump to MARK3 unconditional
 /i.e. without any condition
```

End of program
Every program has to be terminated with the instruction „EP“.

Macros
Macros are defined sequences of commands which are similar to the operation of functions. They are invited by a macro name with arguments as pass parameters. Different from functions they do not put out a return code.
A macro is not an invitation of special function routines but it is created by the internal combination of different B-CON command sequences (B-CON operators).
A macro can be compared to a subprogram (subroutine) invited by the main program and passing on the necessary parameters.

The general invitation of a macro is declared as follows:

M_name[(][ARG1[,arg2[,...,argn]]][)]

Explanations:

M_name	macro name to address and operate the macro
arg1,arg2 ... argn	macro arguments prescribed by the corresponding macro
[...]	elements in these brackets are optional arguments of a macro

Include
The include directive is used to include a files in current program position.

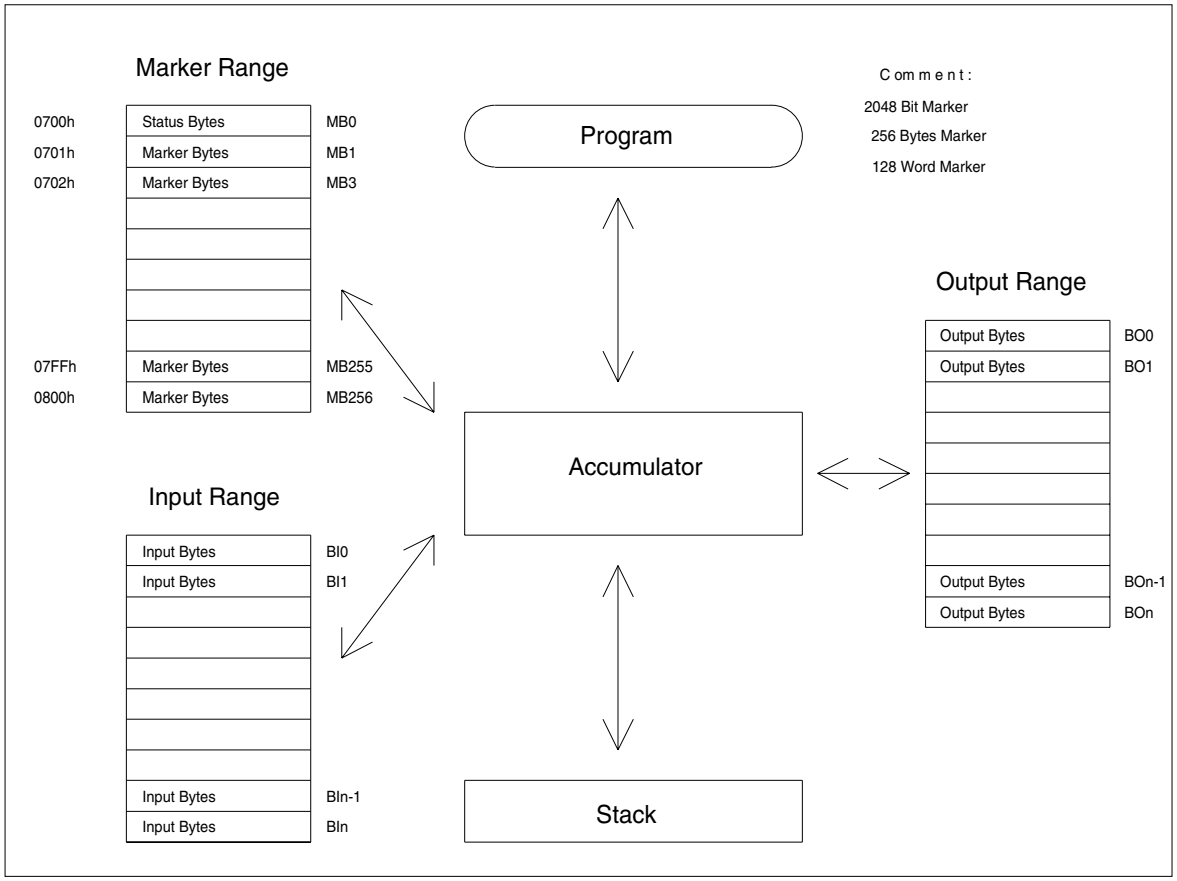
Using “#INCLUDE” files with program, definitions or data can included in the user program. The parameters, defined in the file, that is included can be used as a standard parameters.
It is impossible to debug the file, included in the program. However, the values of parameters can be changed and watched.

Examples:

```
#include PrgmFl.pgm
#include DataFl.pgm
```

3.1.2 Elements for Program Control

In order to define how the control is to work, the handling of the programming language and the concept of the control must be known.
Concerning the development of the user program it must be reagarded that the control consists of the following main elements:



Accumulator
The accumulator is a logical unit wich consists either in loaded operand 1 or in the result after the execution of an instruction.
According to the data format used it has to be noted that the ACC has a format length of 8 or 16 bits.
The format length of 1 bit is represented by 1 byte, only bit Nr 0 being evaluated and bits 1 to 7 remaining unregarded.

Stack
The stack, also called nesting storage or push-down store, is a logical unit which stores results or operands temporarily.
The stack has a fixed data length of 8 bits. During the execution of the program a total of 8 bytes can be stored in the stack.

The operating mode of the stack is according to the LIFO principle (last in - first out), i.e. that the value stored last is unloaded first.
Cooperation with the ACC takes place automatically, e.g. by loading or unloading with each loading or storing.
Physically the stack is organized in such a way that its beginning is with the address assigned uppermost. Its symbolical address 1 thus lies on the storing address assigned uppermost and its symbolical address 8 lies on the lowest storing address.

With a load command the latest contents of the ACC is pushed into the stack and the stack address is decremented.
With store commands the stack contents loaded last is transported back into the ACC and the stack address is incremented.
The format length of 1 bit principally takes byte format, i.e. 8 bits. Only bit 0 is evaluated and bits 1 to 7 remain unregarded.

Inputs/Outputs
Inputs and outputs are process input signals which correspond to the control module applied. Their values are copies of the hardware input/output channels.

They can be defined in the following operand formats:

Format	Preset Identification
Bit	Without
Byte	B
Word	W

I/O addresses

In B-CON the 4 different input/output types are accessed by means of different address ranges as followed:

I/O Type	System 2000 reference	Address Range
Digital	DI/DO	0000 - 1999
Analog	AI/AO	2000 - 3999
Cnt/GW/etc	ZI/ZO	4000 - 5999
Derived	YI/YO	6000 - 7999

The actual useable addresses relates of cause to the actual physical installation (number and type of expansion modules etc.) and the defined registers for communication with the master (YI/YO in B-CON S see below).

ZI/ZO variables are normally not available in BCON-S.
In order to write YIxx variable an instruction ST WOxx should be used.
In order to read YOxx variable an instruction LD WIxx should be used.

In order to make local Inputs available to the master, their number should be defined using “#DEFINE” directive.

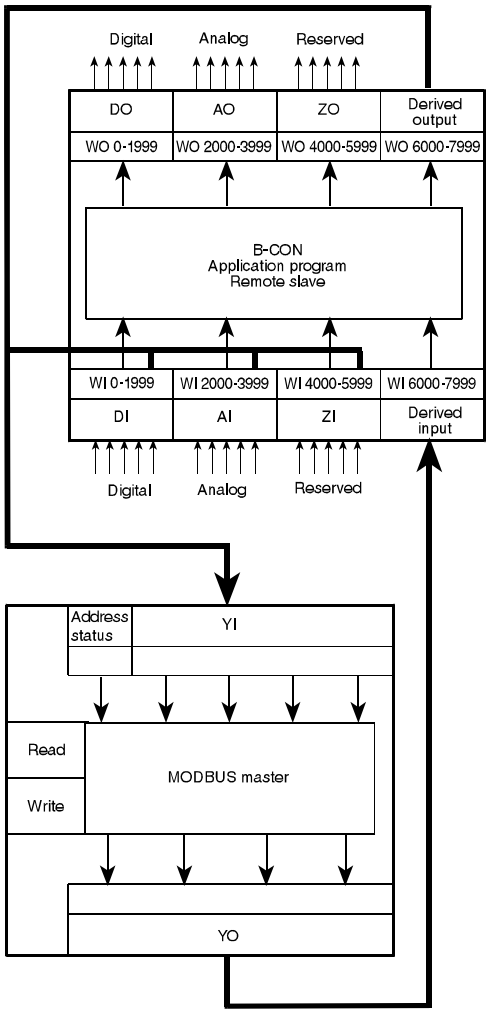
Example:

```
#DEFINE DI_CNT 1          // Master can read 1 DI Word
#DEFINE AI_CNT 2          // Master can read 2 AI Words
#DEFINE YI_CNT 16         // Master can read 16 YI Words
#DEFINE YO_CNT 1          // Master can write 1 YO Word
```

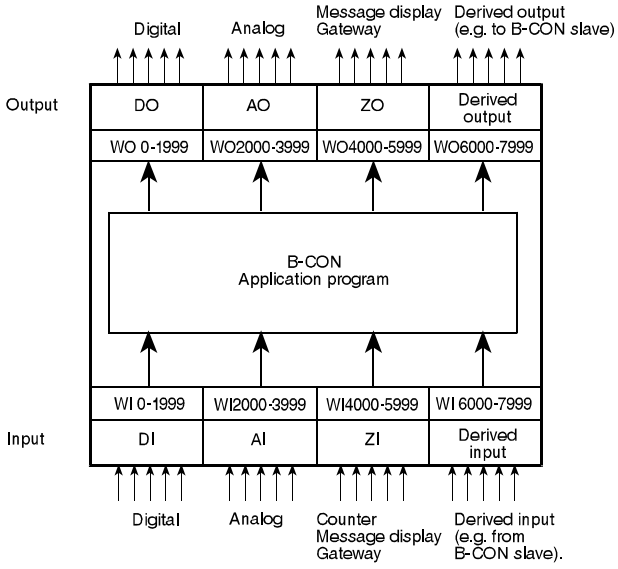
If the actual installation contains more inputs than specified only the lowest inputs will be transferred, e.g.: DI_CNT 1 will transfer first 16 inputs (word 0) whereas following inputs will not be transferred.

The user cannot define Digital and Analog outputs to be available to the master as only the local application program in the slave can control the outputs.
In case outputs should be controlled “directly” from the master the programmer must copy the data form YO (LD wi6000..) to DO (ST wo0...).

B-CON S Database interface/register layout



B-CON M Database interface/register layout



B-CON registers are numbered in units of bytes whereas elsewhere in SYSTEM 2000 the general unit is words, example:

System 2000	B-CON	
DI 0	BI0 BI1	WI 0
DI 1	BI2 BI3	WI 2
DI 2	BI4 BI5	WI 4

Memory variables (Markers)
Markers are storage cells in the internal memory of the control module applied with a memory address of fixed allocation.

Thus are available:

512 byte markers or
256 word markers or
4096 bit markers.

The address allocation of the markers is organized by the operating system according to the following convention:

Marker byte	Memory address
BM0	Error/overflow
BM1	System indicator
BM2	Internal registers
BM3 BM4 BM5	Reserved for modem control (B-CON RM, RS and RTU8).
BM6 . . BM19	Used by RTU8 modules
BM20 . . BM511	(can be used as bit, byte or words)
BM512 . . BM1024	(can be used as bit, byte or words, RTU8 only)

Please observe that the first 2 memory bytes (BM0 and BM1) is reserved for error indication, warnings and special facilities. They must therefore NOT be used as normal internal registers.

The operand formats can be selected in the same way as with inputs and outputs:

Format	Preset Identification
Bit	Without
Byte	B
Word	W

Constants
Constants enable the presetting of fixed variables for operands.
They are also available in the usual operand formats:

Format	Syntax	Value range
Bit	Without	log. „0“ or „1“
Byte	B	-128 0 +127
Word	W	-32768 0 +32767

The corresponding value range always has to be adjusted to the further application (bit, byte or word operation).

Operand calls
The programming language uses 3 labels identities for operands calls, i.e. operand addresses:

- operand format (bit, byte, word)
- operand type (input, output, marker or constant)
- operand counting number in the control system

Format	Syntax
Bit	Without
Byte	B
Word	W

Data type	Syntax
Input variable	I
Output variable	O or Q
Marker (storage variable)	M
Constant	C

As constants can take different data value types, they must be stated by an extended label. This label is placed after the address C.

Value type	Extended label
Binary value	Default boolean integer
Decimal value	Defaults
Hexadecimal	H

Examples:

```
c1          = binary constant of boolean value „1“
c0          = binary constant of boolean value „0“
bc127       = byte constant of decimal value +127
bc-128      = byte constant of decimal value -128
bch0A       = byte constant of value 0Ahex (decimal 10)
bch0  FF    = byte constant of value FFhex (decimal 255)
```

The addresses of operands and operators can be written in small, capital or mixed characters in the source program.
For reasons of clearness using small or capital characters is recommended.

Examples of correct generation of operand addresses:
The following examples show the generation of correct operand addresses:

Operand	Address	Explanation
1.2	input bit variable	input 1, bit No 2
I0.7	input bit variable	input 0, bit No 7
Bi3	input byte variable	input 3
Wi	input word variable	input 0
o1.3	output bit variable	output 1,bit No 3
Q0.5v	output bit variable	output 0, bit No 5
Bo2	output byte variable	output 2
m1.2	bit memory variable	marker 1, bit No 2 (0701hex)
BM3	byte memory variable	marker 3 (0703hex)
wM31	word memory variable	marker 31 (0731hex)
Wm11	word memory variable	marker 11 (0711hex)
C1	bit constant	boolean value 1 (TRUE)
c0	bit constant	boolean value 0 (FALSE)
bc12	byte constant	decimal value 12
bch0a	byte constant	hex value 0A (decimal 10)
wc-1234	word constant	decimal value -1234

Examples of false operand addresses:

Operand	Address	Explanation
1.2		Missing parameter for operand
I0.8	Bit	No greater than 7
bm1.2	Byte	Format with bit marker
dBM3	Undefined	Marker "d"
C11	Bit	Constant grater than 1
bc 128	Byte	Constant greater than 127
wC123.4	Word	Constant with bit marker

Generation of instructions
The generation of an instruction is carried out by selecting an operator followed by an operand as parameter.
The format of the selected operand defines the bit range in which the operation is to be executed. As you know all operand types (inputs, outputs, markers and constants) can also take all formats (bit,byte or word).
As everywhere, there are exceptions here,too, i.e. with some operators (commands) only certain operand formats are permitted.

instructions for bit processing (1-bit length)
AND C0 /AND conn. ACC with bit constant, value log. 0
AND /AND conn. ACC with stack
AND I1.1 /AND conn. with input bit 1.1

Instructions for byte processing (8-bit length)
AND BM1 /AND conn. ACC with marker byte 1
AND B /AND conn. ACC with stack
AND BO2 /AND conn. with output byte 2

Generation of instructions for word processing (16-bit length)
AND WM3 /AND conn. ACC with marker word 3
AND W /AND conn. ACC with stack
AND WI1 /AND conn. with input word

List of commands

In the following list of commands all the commands (operators) are listed with a short description also mentioning the legal operand formats. Derivative commands such as commands in subordinating brackets are not listed. Their handling is illustrated in the corresponding detailed descriptions.

Command	Format	Description
LD		
LDN	all	operand is loaded into ACC
	all	operand is negated and loaded intoACC
ST	all	ACC contents are loaded into operand
STN	all	ACC contents are negated and loaded into op.
S	bit	operand is set „1“ storing
R	bit	operand is reset „0“ storing
AND/&	all	log. AND conjunction, result in ACC
ANDN/&N	all	OP2 negated, log.AND conjunction, result in ACC
OR	all	log. OR conjunction, result in ACC
OR	all	OP2 negated, log. OR conjunction, result in ACC
XOR	all	log. XOR conjunction, result in ACC
XORN	all	OP2 negated, log. XORN conjunction,result in ACC
JMP	bit	unconditional jump to label
JMPC	bit	jump to label if condition log. 1
JMPCN	bit	jump to label if condition log. 0
ADD	byte, word	add operation, result in ACC
SUB	byte, word	subtract operation, result in ACC
MUL	byte, word	multiply operation, result in ACC
DIV	byte, word	divide operation, result in ACC
ROR	byte, word	rotate ACC contents from „High“ to „Low“ by n digits
ROL	byte, word	rotate ACC contents from „Low“to „High“ by n digits
GT	byte, word	compare signed to >, result: Boolean value in ACC
GE	byte, word	compare signed to =>,result: Boolean value in ACC
EQ	byte, word	compare signed to =,result: Boolean value in ACC
LE	byte, word	compare signed to <=,result: Boolean value in ACC
LT	byte, word	compare signed to <,result: Boolean value in ACC
UGT	byte, word	compare unsigned to >, result: Boolean value in ACC
UGE	byte, word	compare unsigned to >=,result: Boolean value in AC
UEQ	byte, word	compare unsigned to =, result: Boolean value in ACC
ULE	byte, word	compare unsigned to <=,result: Boolean value in ACC
ULT	byte, word	compare unsigned to <,result: Boolean value in ACC
DUP	all	duplicate ACC contents into stack
DUPN	all	duplicate negated ACC contents into stack
NOP	_	no operation (no command)
EP	_	end of program

List of functions (macros)
In the following list of macros all the macro notations are listed with a short task description. Their operation is illustrated in the corresponding detailed descriptions.

Macro Notation	Short Description
RTR STR	RS trigger, reset dominant RS trigger, set dominant
CNTUP	counter up counting range: 0..+.32767 max. frequency: 35 Hz
CNTDN	counter down counting range: +32767...0 max. frequency: 35 Hz
CNTUD	counter up - down counting range up: 0...+32767 counting range down: +32767...0 max. frequency: 35 Hz
UPLS DNPLS	positive edge recognition, creates single output pulse negative edge recognition, creates single output pulse
DTR	data trigger, synchronises input signal with system clock max. frequency: 35 Hz
MEQ	Masked equal
LIM	Inlimit check
MOV MVM	Move value Move masked
COPYB COPYW	Move multiple bytes Move multiple words

Timers

Function Notation	Short Description
TMRx	universal timer function x = Timer No number: max. 4 byte argument 0 = timer as edge recognition byte argument 1 = timer as on delay byte argument 2 = timer as off delay

Special functions

Function Notation	Short Description
LOG	LOG instruction (RTU8 only)
CODE	Insert assembler code in program
FUNCTION/ SUBROUTINE	Subroutine