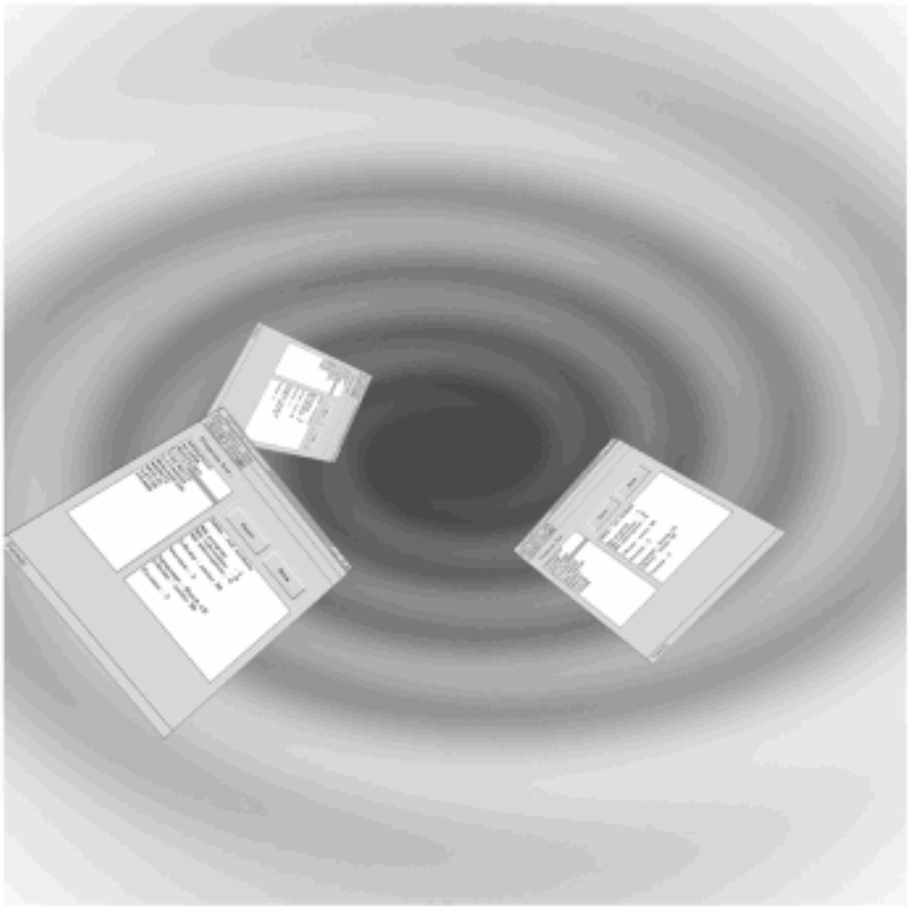
Software manual

Advant Controller 31

AC31GRAF
Programming Software

1SBC006099R1001 C - 03/07



ABB

General table of contents

A USER'S GUIDE

1 Getting started A-3
2 Using the project manager A-4
3 Making a modular project A-11
4 Using editors A-15
5 Editor common tools A-48
6 Control panel A-62
7 User's library A-75

B LANGUAGE REFERENCE

1 Project architecture B-3
2 Variables B-6
3 FBD language B-7
4 LD and Quick LD languages B-10
5 SFC language B-20
6 IL language B-32

C FUNCTION BLOCK DESCRIPTION

1 Libraries C-2
2 Basic operators/functions C-8
3 Program control functions C-68
4 CS31 functions C-84
5 Communication functions C-116
6 Regulation functions C-154
7 Format conversion functions C-174
8 Standard double word functions C-200
9 High order functions C-216
10 Memory access functions C-291
11 Special Functions C-317
12 Historical values C-323
13 Runtimes C-325

LEXICON

Action List of statements or assignments executed when a step of an SFC program is active.

Activity of a step Attribute of a step which is marked by an SFC token. The actions attached to the step are executed according to its activity.

Analog Type of variables. These are continuous integer.

Beginning step First step of the body of a macro step. A beginning step is not linked to any preceding transition.

Boolean Type of variables. Such variables con only take true or false values.

Boolean action SFC action: a boolean variable is assigned with the activity signal of a step.

Cell Elementary area of the graphic matrix for graphic languages such as SFC, FBD or LD.

Clearing a transition Run time operation: all the tokens existing in the preceding steps are removed. A token is created into each of the following steps.

Coil Graphic component of an LD program, used to represent the assignment of an output variable.

Comment Text included in a program, having no impact on the execution of the program.

Comment (SFC) Text attached to an SFC step or transition, having no impact on the execution of the program.

Common Range of defined words. Such objects can be used in any program of any project.

Condition (for a transition) Boolean expression attached to an SFC transition. The transition cannot be cleared when its condition is false.

Contact Graphic component of an LD program. It represents the status of an input variable.

Cross references Information calculated by the AC31GRAF about the dictionary of variables, and where those variables are used in a project.

Current result (IL) Result of an instruction in an IL program. The current result can be modified by an instruction, or used to set a variable.

Cycle timing Duration of the central unit execution cycle.

Diary Text file which contains all the notes about the changes made to one program. Each note is completed with its editing date.

Dictionary Set of internal, input or output variables, and defined words, used in the programs of one project.

Edge Change of a boolean variable. A rising edge means a change from false to true. A falling edge means a change from true to false.

Ending step Last step of the body of an SFC macro step. An ending step is not linked to any following transition.

FBD Functional Block Diagram language.

Function block Graphic component of the FBD language, which represents a standard elementary function from the AC31GRAF libraries.

Functional Block Diagram Graphic language: the equations are built with standard elementary blocks from the AC31GRAF library, linked together in the diagram.

Global Range of variables or defined words. Such objects can be used in any program of one project.

Identifier Unique word used to represent a variable or a constant expression in the programming.

IL Instruction List language.

Initial situation Set of the initial steps of an SFC program, which represents the context of the program when it is started.

Initial step Special step of an SFC program, which is activated when the program starts.

Input Variables linked to an input device.

Instruction Elementary operation of an IL program, entered on one line of text.

Instruction List Low level literal language, entered as a sequential list of elementary operations.

Integer Class of analog variables, stored in a signed integer 16 bit format.

Internal Variable not linked to an input or output device.

Jump to a step SFC graphic component, which represents a link, from a transition to a step. The graphic symbol of a jump is an arrow, numbered with the reference of the destination step.

Keyword Reserved word of the language.

Label (IL) Identifier put at the beginning of an IL instruction line, which identifies the instruction, and can be used as an operand for the JMP operations.

Ladder Diagram Graphic language mixing contacts and coils, for the design of boolean equations.

LD Ladder Diagram language.

Level 1 of the SFC Main description of an SFC program. Level 1 groups the chart (steps and transitions), and the attached comments.

Level 2 of the SFC Detailed description of an SFC program. It is the description of the actions within the steps, and the boolean conditions attached to the transitions.

Library Set of hardware or software resources, which can be directly inserted in any application.

Local Range of variables or defined words. Such objects can be used in only one program of one project.

Macro step SFC graphic component. A macro step is a unique group of steps and transitions, represented as a unique symbol in the main chart, and described separately.

Message Character strings used for ASCII communication.

Modbus Master-Slave protocol. The CS31 central unit can be a Modbus slave for the link with an external system (such as supervisory systems) in a complete architecture.

Modifier (IL) Single character put at the end of an IL operation keyword, which modifies the meaning of the operation.

Non-stored action SFC action: it is a list of statements, executed at each central unit cycle, when the corresponding step is active.

Operand (IL) Variable or constant expression processed by an elementary IL instruction.

Operation (IL) Basic instruction of the IL language. An operation is generally associated to an operand in an instruction.

Output Variables linked to an output device.

Power rail Main left and right vertical rails at the extremities of an ladder diagram.

Program Basic programming unit in a project. A program is described with one language, and is placed in the hierarchy architecture of the project in case of modular project.

Project Programming area, which groups all the information (programs, variables, ...).

Pulse action SFC action: it is a list of statements executed only once when the corresponding step is activated.

Reference number (SFC) Decimal number (from 1 to 65535) which identifies an SFC step or transition in an SFC program.

Register (IL) Current result of an IL sequence.

Separator Special character (or group of characters) used to separate the identifiers in a literal language. A separator may represent an operation.

Sequential Function Graphic language: the process is described as a set of steps, linked by transitions. Actions are attached to the steps.

Chart	Transitions are detailed as boolean conditions.
Sequential section	Group of the programs of a project. The execution of those programs follows the dynamic rules of the SFC language.
SFC	Sequential Function Chart language.
Step	Basic graphic component of the SFC language. A step represents a steady situation of the process, and is drawn as a square. A step is referenced by a number. The activity of a step is used to control the execution of the corresponding actions.
Sub-program	Program written with any language excepted SFC, and called by another program, called its owner program.
Token (SFC)	Graphical marker used to show the active steps of an SFC program.
Toolbox	Small child window of an graphic editing tool window, which groups the main buttons for the selection of the graphic components.
Transition	Basic graphic SFC component. A transition represents the condition between different SFC steps. A transition is referenced by a number. A boolean condition is attached to each transition.
Validity of a transition	Attribute of a transition. A transition is validated when all the preceding steps are actives.
Variable	Unique representation of elementary data which is processed in the programs of project.

A User’s guide

A USER'S GUIDE

1 Getting started A-3
2 Using the project manager A-4
2.1 Project manager description A-4
2.2 Printing a project document A-7
3 Making a modular project A-11
4 Using editors A-15
4.1 Using the FBD/LD editor A-15
4.1.1 Basics of the FBD/LD languages A-15
4.1.2 Entering a FBD diagram A-18
4.1.3 Working on an existing diagram A-19
4.1.4 Displaying options A-21
4.1.5 Other AC31GRAF tools A-22
4.1.6 Style and modification tracking A-23
4.2 Using the SFC editor A-26
4.2.1 SFC language main topics A-26
4.2.2 Entering SFC chart A-29
4.2.3 Working on an existing SFC chart A-30
4.2.4 Entering the level 2 programming A-31
4.2.5 Selecting a variable from list A-35
4.2.6 Commands of the "Tools" menu A-36
4.2.7 Using the SFC gallery A-36
4.3 Using the Quick LD editor A-37
4.3.1 Basics of the LD language A-37
4.3.2 Entering a LD diagram A-39
4.3.3 Working on an existing diagram A-42
4.3.4 Display options A-43
4.3.5 Calling other AC31GRAF tools A-43
4.3.6 Selecting a variable from variable list A-44
4.4 Using the IL editor A-45
4.4.1 File commands A-45
4.4.2 Editing commands A-46
4.4.3 Options A-47
4.4.4 Selecting a variable from list A-47
5 Editor common tools A-48
5.1 Declaring variables A-48
5.1.1 Using the variable list in declaration mode A-48
5.1.2 Using the variable list in selection mode A-49
5.2 Cross References A-53
5.3 Build the application A-55
5.4 Creating graphics A-56
5.4.1 Drawing chart A-56
5.4.2 Object description A-58
5.4.3 Commands of the "File" menu A-60
5.4.4 Options A-60
6 Control panel A-62
6.1 Using the control panel A-62
6.2 Time diagrams A-66
6.3 On line list A-68

6.4 Status / Diagnosis A-71
6.5 Configuration A-72
7 User's library A-75
7.1 User's function A-75
7.2 Variables for a user's function A-76
7.3 Compiling a user's function A-77
7.4 User library access and rights control A-77

1 Getting started

This chapter covers the installation of the AC31GRAF workbench. It also includes a short example of an AC31GRAF application, giving the user a brief outline of its main features and enabling the immediate use of AC31GRAF.

Installing AC31GRAF

This chapter covers the installation of the AC31GRAF Workbench and how to set up the computer for application development.

Hardware and software requirements

The AC31GRAF Workbench can be installed on any computer meeting the minimum qualifications for Windows Version 3.1. However, the following hardware is recommended for application development:

- A personal computer using an 80486 or higher microprocessor
- 8 megabytes of conventional and extended memory
- One 3.5-inch (1.44 megabyte) disk drive
- One hard disk with at least 20 megabytes of available space
- A graphic VGA or SVGA adapter and compatible monitor
- A mouse (required for graphic development tools)
- A parallel LPT1 port

Before installing the AC31GRAF workbench, one of the following software should already be included on the system:

- Windows Version 3.1 running in 386 enhanced mode
- Windows 95
- Windows NT Version 3.51 or 4.00

Using the installation program

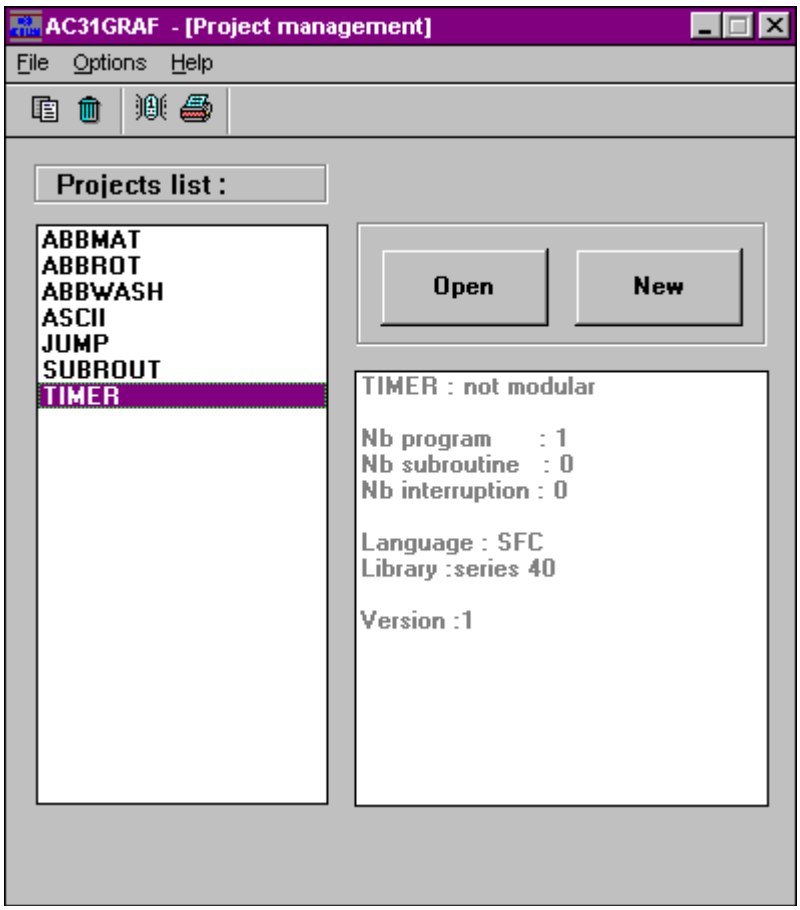
The AC31GRAF Workbench is installed by using SETUP. This program copies the AC31GRAF software from the AC31GRAF disks onto the user's hard disk.

SETUP is a Windows program.
To install AC31GRAF, the following steps must be performed:

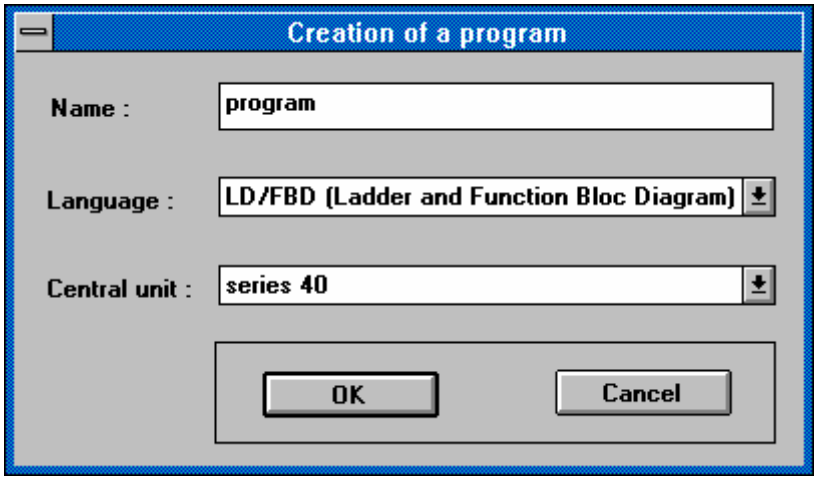
- Insert Disk 1 into the appropriate drive
- From Program Manager, select the "SETUP.EXE" to start the installation.

2 Using the project manager

2.1 Project manager description



To create a new project, push the "**New**" button. Following this, a dialog box appears allowing the user to give a name to the project, to select the CPU type (serie 40, series 50, Controller, series 90 ,serie 94 or series 30), and finally to select the language (SFC, FBD/LD, Quick LD , IL or modularize).



The new project name must conform to the following rules :

- the name cannot exceed 8 characters
- the first character must be a letter
- the following characters can be letters or digits.

It is possible to take the modularize choice instead of a specific language in order to build a modular project. If the project is not modular, the editor of the selected language is opened and the program has the name «main» by default.
On the other hand, if the project is created as modularize, the *Program management* window is opened in order to create programs.

Editing the project descriptor

The ”**File / Project descriptor**” command is used to edit the project text descriptor.
This document fully identifies the project from the others on the project list. The project descriptor can also be used to record any remarks during the project lifetime.

The History of modifications

AC31GRAF stores any information relative to a component of a project in a history file. Each modification is identified in the history by a title, a date and a time. There is one history file for each project.
The “**File / History**” command allows the user to view or print the history of modifications for the selected project. The user can select one or more items in the main list, and press the following buttons:

OK..... closes the window.
Print..... sends the contents of the list to the printer.
Selected ..... removes the selected items from the list.
Erase /All..... clears the complete list.
Search/Find..... find a pattern in the list.

Make a project document

The “**File / Print**” command allows the user to build and print a complete document about the selected project. This document can group any component (program, variable, parameters...) of the selected project.
See description at the chapter «Printing a project document»

Using the library management

The “**File / Library**” command launches the user library manager.

Renaming a project

The “**File / Rename**” command allows the user to change the name of a selected project.

Copying a project

The “**File / Copy**” command allows the user to copy all the contents of the selected project in the same CPU or to another one. When the user enters the name of the copy, according to the rules of above, he is able to choose the CPU.

Deleting a project

The “**File / Delete**” command deletes the entire contents of the selected project.

Modularizing a project

The “**File / Modularize**” command allows the user to edit the *Program management* window in order to create other programs, subroutines or interruptions.

Closing an application

The “**File / Exit**” command closes the application.

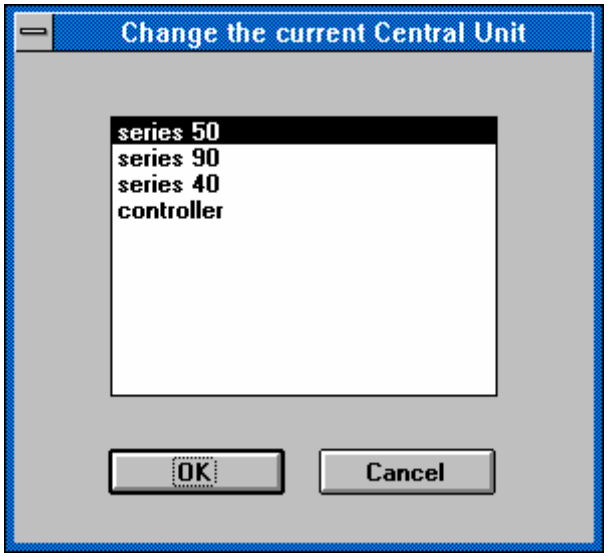
Uploading an application

The “**Options / Upload application**” command allows the user to upload the code of a CPU. First, the user creates a new project and next, he has to select the time out value and the COM port he wants to use, then the *Control panel* window is editing and displays a default choice of the program start and a program end of the upload address that the user can change.

Finally, the code is uploaded and the original project edited.
The uploaded project consists of a main program and, if any, of subroutines and interruption programs. The IL editor is necessary to read the code of these different programs.
All the variables used in the uploaded project are recorded in the global variables file.

Changing the Central Unit

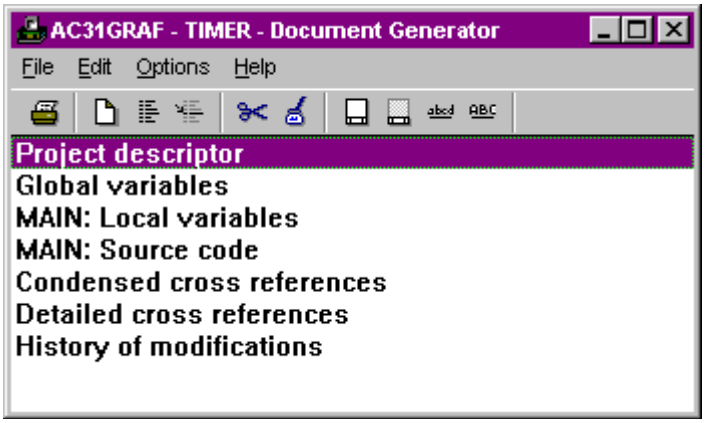
The “**Options / Change the Central Unit**” command allows the user to change the current CPU of the selected project.
According to the new CPU, the behavior of the project will be affected (possibility of subroutines, interruptions, area addresses for the variables, new list of blocks)



2.2 Printing a project document

The AC31GRAF Document Generator allows the user to build and print a complete document for the selected project. Unlike the "**Print**" commands from the other windows of the AC31GRAF Workbench, the Document Generator can be used to print more than one component of the project in the same document, with global format and page numbering.

The Document Generator is selected from the **Project manager** with :
File / Print



The "**File / Print**" command of the **Document Generator** generates the document and send it to the printer, according to the specified table of contents. The "Print" job may take few minutes to build and format the document. It is highly recommended to wait until "Printing Job" is done in the AC31GRAF Document Generator window, before running other commands of the AC31GRAF Workbench. Building the whole document may require a large space on the hard disk. An error message will be displayed if the disk is full. In such a case, the user will have to either free up disk space by removing files, or reduce the size of the print job. When the "**File / Print**" command is run, a dialog box appears. It allows the user to enter a note describing the actual print command. Those notes are stored in a history file, and will be printed on the first page of any future document (including the present one).

The commands of the "**Edit**" menu are used to define the elements of the project that must be inserted in the document. A choice of commands allow the user to use a default table (with all the components of the project), build a specific table (with only some components) or move items in the table and modify it. Any information about the project may be inserted in the project document. No information from another project or from AC31GRAF libraries may appear in this document.

Inserting a new item

When the "**Edit / Insert**" command is run, the "**Add item**" dialog box appears. It allows the user to insert items (components of the project) into the table of contents.

For an item relative to a program, use the "**Program**" combo box to select a program name. Press the "**Add**" button to insert the selected item to the table of contents. The same item can appear only once in the table.

Clearing table

The "**Edit / Clear**" command resets the table of contents, so that it can be totally rebuilt using single item insertion.

Default table

The "**Edit / Default list**" command defines a standard table of contents for the document, which includes all the components of the project. The standard table consists of:

- Project descriptor
- Global variables
- MAIN: Local variables
- MAIN: Source code
- Condensed cross references
- Detailed cross references
- History of modifications

Cut and paste

The "**Edit / Cut**" and "**Edit / Paste**" commands move items in the list, in order to customize the order of the table. The Document Generator allows multiple selection so that a group of items may be cut and pasted.

The commands of the "**Options**" menu are used to define and customize the format of the generated document.

Page format

The "**Options / Page format**" command is used to define the main parameters operated by the Document Generator when formatting a page. The following parameters can be specified:

- **Left margin**: (1 or 2 centimeters, or no margin)
- **Page border**: When this option is selected, a border is drawn around any printed page.

Page title

The "**Options / Page Title**" command is used to define the contents of the title box printed at the bottom of any page. The standard layout of this box is as follows :

ABB	Text1	AC31GRAF - Project 'PrName'	date
	Text2		
	Text3		page

The first line of the main title (with the name of the AC31GRAF project), the current date and the page number are automatically generated by the Document Manager, and cannot be changed.

The three lines of text on the left side of the box (text1, text2, text3) and the second line of the main title are user defined. The user also can change the logo printed in the box on the left. To use another logo, the user has to specify the pathname of a bitmap image file (**.BMP**). The image can have any dimension. It will be stretched or shrunk, according to the exact dimensions of the printed page. Clicking on the logo area, in the dialog box, shows the new specified image. The image file must be on the disk (at the specified directory and with the specified filename) when the "**File / Print**" command is run.

Selecting character fonts

The "**Options / Text font**" and "**Options / Title font**" commands are used to define the fonts of characters used when printing text, and titles for any item of the document. The size and style of characters may also be selected for text and titles. The selection of a font is made with the standard dialog box defined by Windows. Any text (literal programs, names within diagrams...) will be printed with the selected size, style and font of characters. Only titles will be printed with the font selected for titles.

If the fonts of characters are not defined, the standard font of the printer will be used for any text, with the following styles:

- "Normal" style for texts and names within diagrams
- "Bold" style for titles

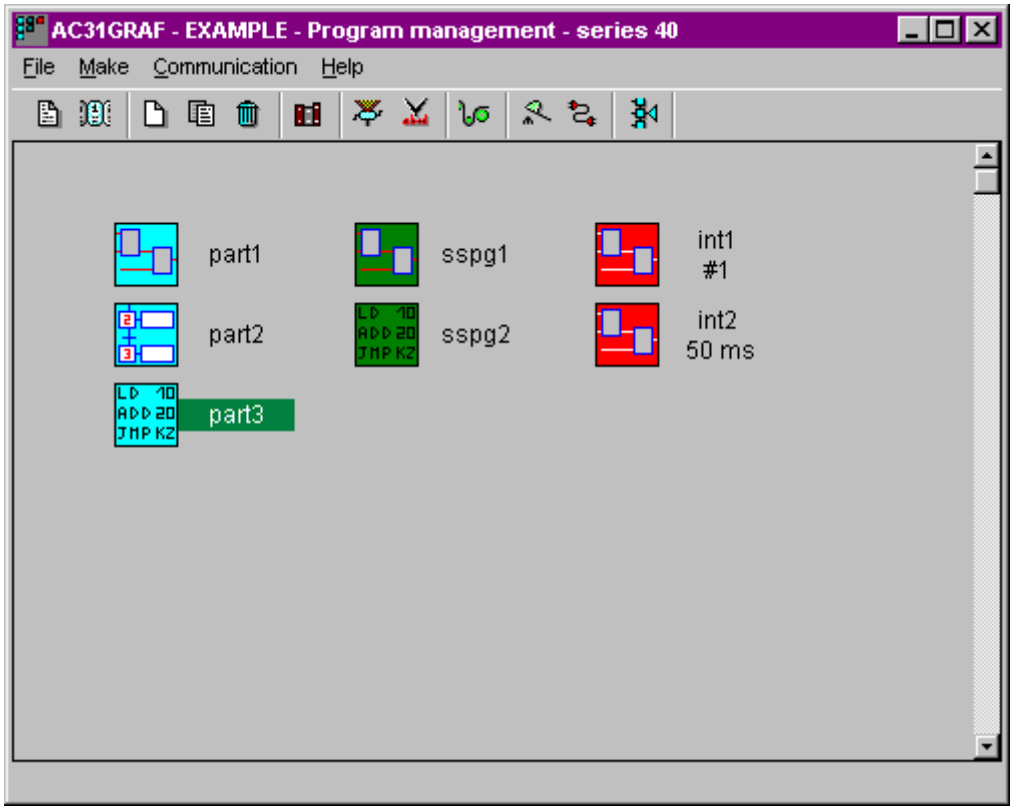
Separate SFC levels

The "**Options / Separate SFC levels**" option directs the system to print, for each SFC program, first the level 1 of the SFC (chart and comments), and then the level 2 programming. When this option is not checked, levels 1 and 2 appear together on the same printout.

3 Making a modular project

You can create modular projects using one of two methods:
-when creating a project, select **modularize** in the dialog box
-when in a non-modular project, select the Modularize menu item

The program management window is edited and you are able to create programs.



Creating a new program

The "**File / New**" command enables you to create a new program, subroutine, or interruption for the project.
For a subroutine and an interruption, the SFC editor cannot be used.

The user has to enter the name of the program, the language and the type of the program.

If a program is created, a blue icon appears in the window program management, for a subroutine, the icon is green and for an interruption the icon is red. The name of the program is set at the right of the icon.

Icon for IL language :

Icon for FBD/LD language :

Icon for SFC language : 

Icon for Quick LD language: 

A project can contains only three interruption programs : two hard interruptions and a soft one.
For a interruption program, the name contains the type of the interruption (#1 or #2 for a hard interruption, and the cycle time for the soft interruption).

Some CPU do not support subroutine and interruption program, they are : serie 30 , serie 90 and serie 94.

The Subroutines and interruption tasks are called in the main program (Il or FBD language) with directly their name as a function block or a format parameter.

Editing a program

The "**File / Edit**" command displays the editor of the selected program. It has the same result as the button **open**.

Editing the variables list

The "**File / Variable list**" command allows to edit the list of variable window.
The global variables and the local variables of the current program are shown.

Editing a diary file

The "**File / Program descriptor**" command allows the user to start editing the diary file of the current edited project program. This is a text file which contains all the notes about the modifications made to the program during its time life.

Setting an interruption program

The "**File / Parameters**" command allows the user to change the cycle time of the soft interruption of the project.

Renaming a program

The "**File / Rename**" command allows the user to change the name of the selected program.
The new program name must conform to the following rules :

- the name cannot exceed 8 characters
- the first character must be a letter
- the following characters can be letters or digits.

Deleting a program

The “**File / Delete**” command deletes the selected program/subroutine/interruption from the current application.

Copying a program

The “**File / Copy**” command allows the user to copy the selected program to the same project, so the user has to give a name for the copy and then nothing happens for the variables.
Furthermore, the user can copy the selected program to an other project. First, there is a detection of conflict name, and then the program is added to those of the target project. There is a detection on variables conflict.
Each variable not used in the target project will be added to the global variables file of this project.

Closing the program manager

The “**File / Exit**” command closes the program management window.

Making a graphic

The “**Make / Graphics**” command runs the graphic editor. This tool allows the user to define graphic images that will be refreshed during debug, based on the state of the application variables. The images are built with standard windows bitmap (**.BMP**) and icon (**.ICO**) files. This requires additional graphic editing tools, such as **PaintBrush**, to create bitmaps and icon files.

Building the application code

The “**Make / Code generation**” command starts the project code generation.
Before generating the code, any program that is still not verified is checked to detect the syntax errors.

Verifying a program

The “**Make / Verify**” command allows the user to verify the syntax of the program currently selected. When a program is verified, with no error detected, it is not re-verified during the code generation.

Running the cross reference editor

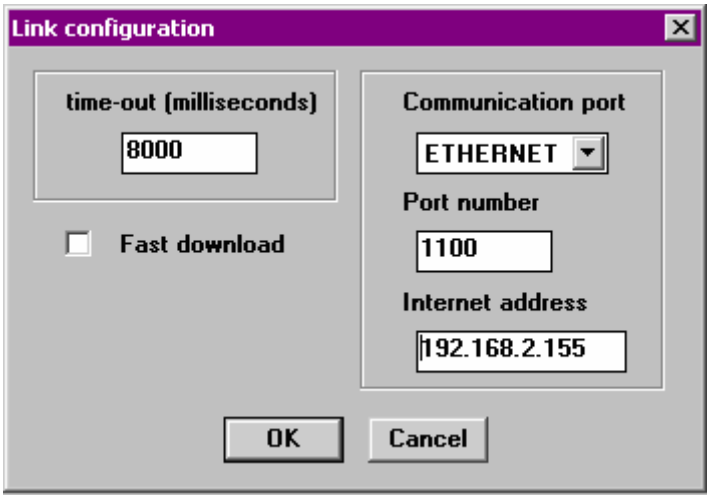
The “**Make / Cross references**” command allows the user to calculate, view or print the cross references of the project. The cross references show the user all the occurrences of each variable in the source code of the programs, in the entire project. This function is very useful to detect an access to a variable or any global source, or to the list all the occurrences of a global variable in the source code.

Running the communication

The “**Communication / Run communication**” command opens the communication main window, and closes the program management.
This open is then re-opened in debug mode as soon as the communication is established between the debugger and the target application.

Setting the communication parameters

The “**Communication / Communication parameters**” edits the dialog box enables the user to define the parameters of the link for communication between the debugger on the host PC and the target system.



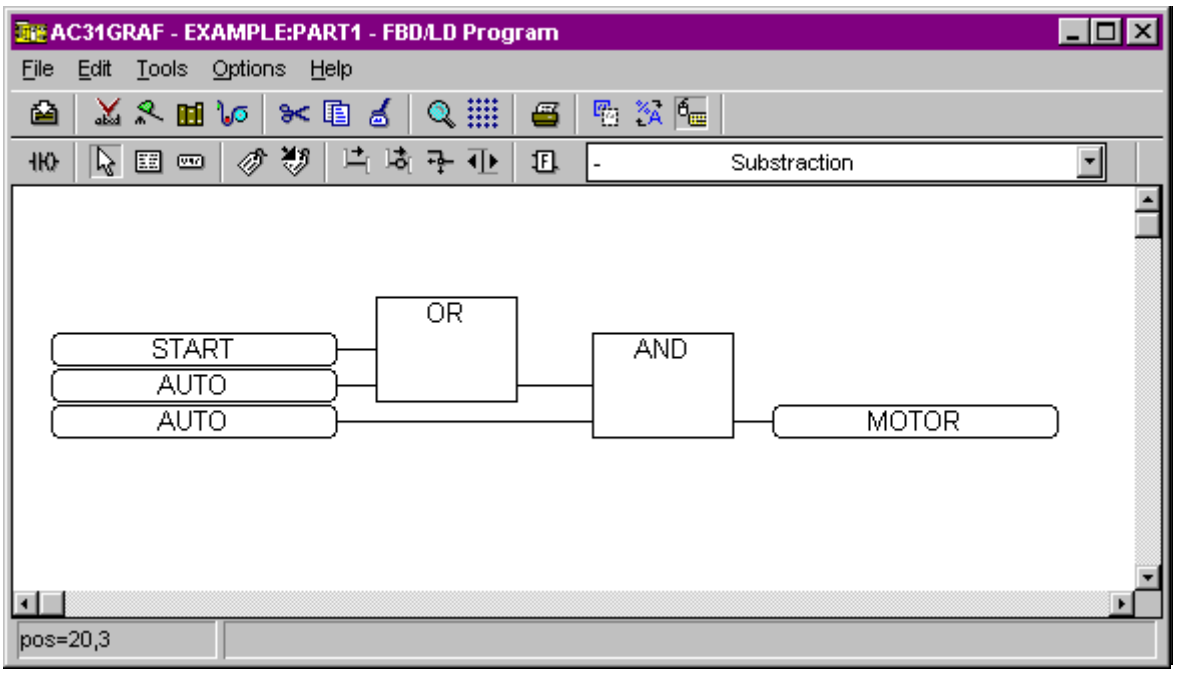
The communication parameters are the following:

- communication port
- time-out (ms)
- an option for fast download. This option allows the communication to transfer a program in the PLC approximately seven times faster.
  WARNING: Occasionally, when using a modem, an RS485,.. communication malfunctions. In these cases, do not select this option.
- port number to use for TCP-IP communication (for use with the Ethernet communication port only)
- Internet address to use for TCP-IP communication (for use with the Ethernet communication port only)

4 Using editors

4.1 Using the FBD/LD editor

The AC31GRAF FBD/LD graphic editor allows the user to enter complete FBD programs, which may includes parts in LD. It combines graphic and text editing capabilities, so both diagrams and corresponding inputs and outputs can be entered. As the editor is more dedicated to FBD language, pure LD diagrams should rather be entered using AC31GRAF Quick LD editor.



4.1.1 Basics of the FBD/LD languages

The **FBD** language is a graphic representation of many different types of equations. **Operators** are represented by rectangular function boxes. Function inputs are connected to the left side of the box. Function outputs are connected to the right side. Diagram inputs and outputs (**variables**) are connected to the function boxes with **logical links**. An output of a function box may be connected to the input of another box.

The **LD** language enables graphic representation of boolean expressions. Boolean **AND**, **OR**, **NOT** operators are explicitly represented by the diagram topology. Boolean input variables are attached to graphic **contacts**. Boolean output variables are attached to graphic **coils**. Contacts and coils are connected together and to left and right power rails by **horizontal lines**. Each line segment has a boolean state of **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

LD and FBD diagrams are always interpreted from the left to the right, and from the top to the bottom. Refer to the AC31GRAF Manual Part B for more details about LD and FBD languages. These are the basic graphic components of the LD and FBD languages, such as supported by the FBD/LD editor:

 **Left power rail**

Rungs must be connected on the left to a **left power rail**, which represents the initial "TRUE" state. AC31GRAF FBD editor also allows to connect any boolean symbol to a left power rail.

 **Right power rail**

Coils may be connected on the right to a **right power rail**. This is an optional feature when using the AC31GRAF FBD/LD editor. If a coil is not connected on the right, it includes a right power rail in its own drawing.

 **LD vertical "OR" connection**

LD vertical connection accepts several connections on the left and several connections on the right. Each connection on the right is equal to the OR combination of the connections on the left.

 **Contacts**

A contact modifies the boolean data flow, according to the state of a boolean variable. The name of the variable is displayed upon the contact symbol. The following types of contacts are supported by AC31GRAF FBD/LD editor:

 ..... direct contact
 ..... negated contact
 ..... contact with positive (rising) edge detection
 ..... contact with negative (falling) edge detection

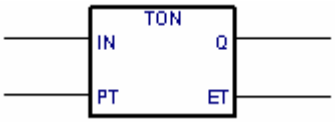
 Coils

A coil represents an action. It must be connected on the left to a boolean symbol such as a contact. The name of the variable is displayed upon the coil symbol. The following types of coils are supported by AC31GRAF FBD/LD editor:

 ..... direct coil
 ..... negated coil
 ..... "set" action coil
 ..... "reset" action coil

Function blocks

A block in an FBD diagram can represent a function, a function block, a sub-program or an operator. Inputs and outputs must be connected to variables, contacts or coils, or other block inputs or outputs. Formal parameter names are displayed inside of the block rectangle.



Labels

Labels can be placed everywhere in the diagram. Labels are used as targets for jump instructions, to change the execution order in the diagram. Labels are not connected to other elements. It is highly recommended to place labels on the left of the diagram, in order to increase the diagram readability.

Jumps

A jump symbol always refers to a label, placed elsewhere in the diagram. Its left connection must be linked to a boolean point. When the left connection is TRUE, the execution of the diagram directly jumps to this target label. Note that backward jumps are dangerous as they may lead to a blocking of the PLC loop in some cases.

Variables

Variables in the diagram are represented inside small rectangles, connected on the left or on the right to other elements of the diagram.

Connection links

Connection links are drawn between elements put in a diagram. Links are always drawn from an output to an input point (in the direction of the data flow).

Connection links with boolean negation

Some boolean links are represented with a small circle on their right extremity. This represents a boolean negation of the information transported by the link.

User defined corners

User defined points may be defined on links. They allow the user to manually control the routing of a link. If no corner is placed, the AC31GRAF FBD/LD editor uses a default routing algorithm.

4.1.2 Entering a FBD diagram

To enter a diagram, you have to place elements (blocks, variables, contacts, coils...) in the graphic area, and draw links between them.

Inserting objects

To insert an object in a diagram, select the corresponding button in the toolbar and click in the graphic area, where you want to insert it.

Selecting objects

Selecting graphic objects is needed for most of the editing commands. The AC31GRAF LD/FBD graphic editor enables the selection of one or more existing objects in the diagram area. To select objects, the "**select**" (button with an arrow) choice must be checked in the editor toolbar. To select one object, the user only has to click on its symbol. To select a list of objects, drag the mouse in the diagram and select a rectangle area. All the graphic objects that intersect the selection rectangle are marked as "**selected**". A selected object is drawn with little black squares around its graphic symbol. By making a new selection, all previously selected objects are unselected. To remove the existing selection, simply click with the mouse on an empty area, outside of the rectangle which borders the selected objects.

Inserting comments

Comments may be inserted anywhere in the diagram. Comments have no influence on the program execution. They allow a higher readability of the diagram. To insert a comment block, select this button in the toolbar, and drag the mouse to select the rectangle area where comment must be drawn. Then enter the text of the comment. No special leading or trailing characters such as "**(***" and "***)**" are needed when entering the text of a comment block. A comment block may be resized by dragging the corners of its border when it is selected.

Moving objects

To move objects in the diagram, you have to select them, and drag the mouse to move the selected area in the diagram. To move connected objects, the user simply has to move the graphic symbols put on the diagram. The AC31GRAF LD/FBD editor will automatically redraw the connection lines between the objects that were moved, based on their new location.

Drawing links

Select one of these buttons in the toolbar to draw a link between connection points of existing elements. If you draw a link from a connection point to an empty location in the diagram, it is automatically terminated by a user defined corner, so that you can continue drawing another segment.

Changing link drawing

The "**Tools / Move line**" command is used when a link is selected in the diagram to change its automatic routing. This command has no effect when the link is connected to a user defined corner. When a link is drawn as three segments, this command changes the position of the second segment. Below are examples:

Changing the type of a link

You can easily change the type of link (with or without boolean negation) by double clicking with the mouse on its right extremity.

Drawing LD rungs

To draw a new LD rung, first insert the left power rail. Then place a coil: it will be automatically linked to the power rail. Other contacts and vertical OR connections may be directly inserted on the rung line, without drawing any new connection link.
When a new LD contact or coil is inserted in an empty space of the editing area, the new horizontal rung line is automatically drawn from the new inserted element to the existing power rails on the left and on the right. This line is not automatically drawn if the new contact or coil is not placed between power rails. The new inserted contact or coil can then be freely moved on the drawn rung. The horizontal lines created by the editor while inserting a LD contact or coil symbol can be selected and deleted. You can insert a new LD contact or coil symbol on the horizontal line of an existing rung. The editor automatically cuts the rungs and connects it to the left and right connection points of the new inserted contact or coil.

Multiple connections

A multiple connection can be created on the right of any **output** point. It means that the information is **broadcasted** to several other points in the diagram. The same state is propagated on each extremity on the right. The number of lines drawn at the right of an output connection point is not limited. Two connection lines cannot have their right extremity connected on the same **input** point, except for the following LD symbols:

- ..... right power rail
- ..... multiple connection on the left (OR) operator

These LD symbols can have an unlimited number of inputs.

4.1.3 Working on an existing diagram

The commands of the "**Edit**" menu are used to change or complete an existing diagram. Most of these commands act on the elements currently selected in the diagram.

Correcting a diagram

The DEL key can be used to remove the selected elements. Pending links are deleted with selected elements. Use "**Edit / Undo**" command to restore elements after a DEL command. The DEL command can also be applied to a group of elements selected in the diagram. The "**Cut**", "**Copy**", "**Paste**" commands of the "**Edit**" menu are used to move or copy selected elements.

Find and replace

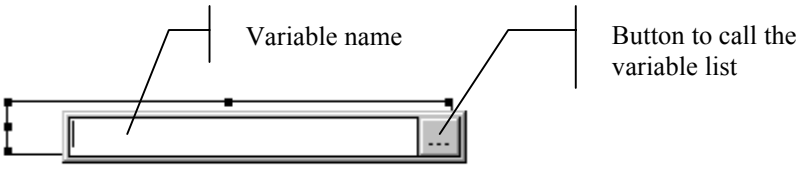
The "**Edit / Find**" and "**Edit / Replace**" menu commands are used to find and replace texts in the diagram. Only complete names can be found. Research acts on contacts, coils, block names, variables and labels. It cannot be used to find a string in a comment text. The Replace command cannot be used to change the name of a block. The research can be made upward or downward, starting at the current selection position. It "loops" when the limits of the diagram are reached.

Displaying the execution order

When an FBD diagram includes backward loops, the execution order cannot follow the single left to right / top to bottom method. In order to avoid confusion, use the "**Tools / Show execution order**" command or press **Control + F1** keys to display the execution order that will be used at compiling time. Tags numbered from 1 to N are displayed close to symbols that lead to an action (coils, set variables and function blocks).

Entering symbols and texts

Double click with the mouse on an element to enter the associated symbol or text. This applies to variables, contacts and coils, comment texts and labels. When used on a contact or coil, this also allows to change its type (direct, negated...).
If the "**Auto input**" mode is checked in the "**Options**" menu, the variable symbol must be entered immediately each time a new contact or coil is inserted.
The symbol must always be entered immediately when a variable or label is inserted.



If the "**Manual keyboard input**" mode is selected in the "**Options**" menu, the variable name is directly catch in a field. Enter the new text and hit **Enter** to validate, or **Esc** to give up. The field used for this mode can not be closed with the mouse.
When you enter the variable name and you hit **Enter**, if the variable does not already exists, the variable list is opened to complete the variable definition.
When you click on the button to call the variable list, if the variable already exists, the variable list will be opened on this variable.

Selecting function block type

Double click with the mouse on a block is used to change its type. The block type is selected from the list of available operators, functions and function blocks. This command also allows to change the number of input points in the case of a commutative operator (e.g. AND, OR, ADD, MUL...).

Getting free space

When you press the right button of the mouse in the FBD drawing area, a popup menu is displayed. It contains the following commands that can be used to insert or remove free space at the location of the mouse cursor:

Insert rows..... This command inserts horizontal free space, made of 4 rows according to the grid step, starting at the position of the mouse cursor where popup menu is open.

Delete rows..... This command removes unused horizontal space (rows) starting at the position of the mouse cursor where popup menu is open. This command cannot be used to remove FBD elements.

When popup menu is open, a gray line in the FBD drawing area indicates where empty space will be inserted or removed.

4.1.4 Displaying options

The commands of the "**Options**" menu are used to customize the drawing of the FBD diagram on the screen.

Layout customization

The "**Options / Layout**" command opens a dialog box where are grouped all the parameters and options concerning the editor workspace and the drawing of the graphic diagram. Use the check boxes in the "Workspace" groupbox to display or hide editor toolbars and status bar. Options of the "Document" groupbox allows you to show or hide points of the editing grid, and to enable/disable the use of colors for the drawing.

Switching name and address

The “**Options / Name/address**” command switches the symbol of a variable by his address and vice versa.

Selecting LD toolbar

The “**Options / LD toolbar**” command selects the LD toolbar.

Selecting FBD toolbar

The “**Options / FBD toolbar**” command selects the FBD toolbar.

Auto input variable list manager

The “**Options / Auto input**” command edit the variable list manager when a variable is put.

Manual keyboard input

The “**Options / Manual keyboard input**” command allows the user to enter directly the symbol of the variable.

Setting the communication parameters

The “**Options / Link configuration**” command allows the user to modify the communication parameters.

Zoom

Options of the "Zoom" groupbox allows you to select a main zoom ratio. You can also use the "zoom" button in the editor toolbar to swap between default zoom ratios.

4.1.5 Other AC31GRAF tools

Verifying (compiling) the program

The "**File / Verify**" command runs the AC31GRAF code generator to verify the programming syntax of the currently edited program. In case of SFC language, both level 1 and 2 are checked.

Running debugger

The "**File / PLC communication**” command runs the AC31GRAF graphic debugger real connected mode, and re-opens the edited FBD/LD program in debug mode. Used in debug mode, no modification can be entered in the program.

Editing the variable list

The "**File / Variable list**" command is used to edit the list of variables for the current application and the current program.

Making a Graphic

The “**File / Graphics**” command runs the graphic editor. This tool allows the user to define graphic images that will be refreshed during debug, based on the state of the application variables. The images are built with standard windows bitmap (**.BMP**) and icon (**.ICO**) files. This requires additional graphic editing tools, such as **PaintBrush**, to create bitmaps and icon files.

Printing the program

The "**File / Print**" command outputs the edited program on a printer. This command produces a draft listing of the program. More detailed information is given when the project document generator is used.

Selecting a variable from the variables list

When editing a text program the "**Edit / Insert variable**" allows the selection of a declared variable name to be inserted at the current position of the caret. When editing LD or FBD programs, variable selection is required for the description of contacts coils, block I/O parameters or FBD variable boxes. In both cases, the "Select" dialog box is open to select a declared variable.

To select a variable, click on its name in the list. Its name and comment are then displayed on the top of the list. Then press the "**OK**" button to confirm its selection. It is also possible to directly enter a variable name in the edit control without using the list.

For the graphicals programs (SFC, FBD and Quick LD) the "**Edition / Copy drawing (metafile)**" copy an image of a program in metafile form into the Window clipboard. So, it can be copied in others applications. For the SFC programs, only the information of the level 1 are copied into the image (graph, references and comments).

4.1.6 Style and modification tracking

The AC31GRAF LD/FBD editor enables you to assign a graphic style to any component of a LD/FBD diagram. A style is mainly defined as a special diagram colouring. But AC31GRAF also used styles to enable modification tracking in diagram for version control purpose.

Note that styles are not visible during simulation or on-line debug, as colours (red and blue) are used in that mode to highlight TRUE / FALSE states of spied variables.

Predefined style

The following styles are pre-defined:

Normal..... Default drawing (black). For modification tracking, "normal" style indicates that elements having that style are part of the original diagram. "Normal" style elements are normally scanned during execution.

Modified ..... Elements marked as "modified" are painted in pink. For modification tracking, the "modified" style is used to highlight elements that have been added or changed after the original release of the diagram. "Modified" style elements are normally scanned during execution.

Deleted ..... Elements marked as "deleted" are painted in gray, with dashed lines. Such elements are not taken into account for the execution of the diagram. This style is used to keep a track of elements removed after the original release when version control is required.

Custom..... In addition to predefined style, AC31GRAF LD/FBD editor allows you to select any color to be applied to a part of the diagram. Such elements are considered as having a "Custom" style. The use of "Custom" style has no effect on the diagram execution at run time.

Use the commands of "**Style**" sub-menu in "**Edit**" menu to manually apply a style to selected elements.

Modification tracking

The use of styles, and the availability of the "Deleted" style allows automatic modification tracking in an existing diagram. Use the "**Mark modifications**" command in "**Edit/Style**" menu to enable or disable modification tracking.

When the "**Mark modifications**" option is set, all elements changed in or added to the diagram are automatically set with "**Modified**" style. When an element is deleted, using "**Delete**" or "**Cut**" commands, they are not visually removed from the diagram, but simply marked with "**Deleted**" style. This enables the user to automatically keep a trace of all modifications entered in the diagram.

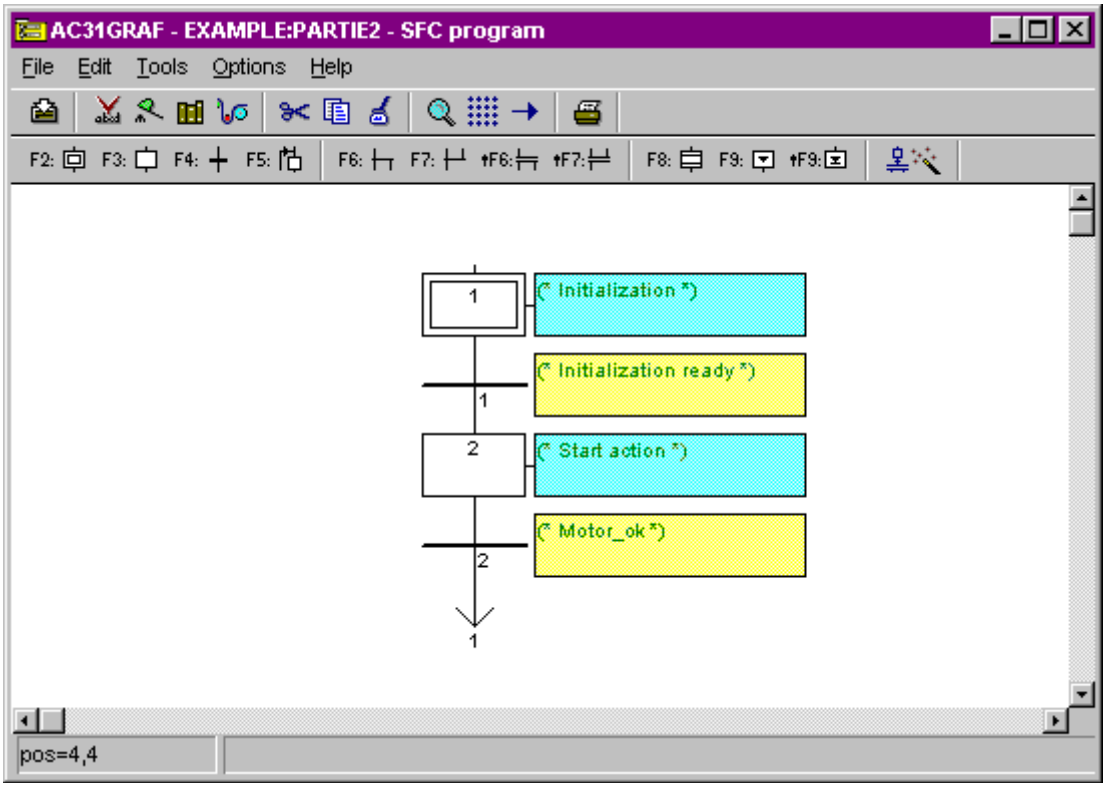
Use the "**Edit/Style/Remove all deleted items**" to actually remove all elements marked with "**Deleted**" style from the LD/FBD diagram. This command does not take care of the current selection, and always applies to the entire diagram.

To "restore" one element marked with the "**Deleted**" style, select the desired element and apply to it the "**Normal**" style, the "**Modified**" style or any "**Custom**" style. Such operation may lead to invalid connections (more than one link connected to the same input point) that will be detected during next program verification.

For graphic programs (SFC, FBD), you can also use the "**Edit / Copy drawing**" command to copy in the clipboard the drawing of the chart in metafile format, so that it can be pasted in other applications such as word processors. For SFC programs, only the level 1 information (chart, numbering and level 1 comments) appears on the copied metafile.

4.2 Using the SFC editor

The SFC language is used to describe operations of a sequential process. It uses a simple graphic representation for the different steps of a process, and conditions that enable the change of active steps. An SFC program is entered by using the AC31GRAF graphic SFC editor. SFC is the core of the IEC 1131-3 standard. The other languages usually describe the actions within the steps and the logical conditions for the transitions. The AC31GRAF graphic SFC editor allows the user to enter complete SFC programs. It combines graphic and text editing capabilities, thus allowing the entry of both the SFC chart, and the corresponding actions and conditions.



4.2.1 SFC language main topics

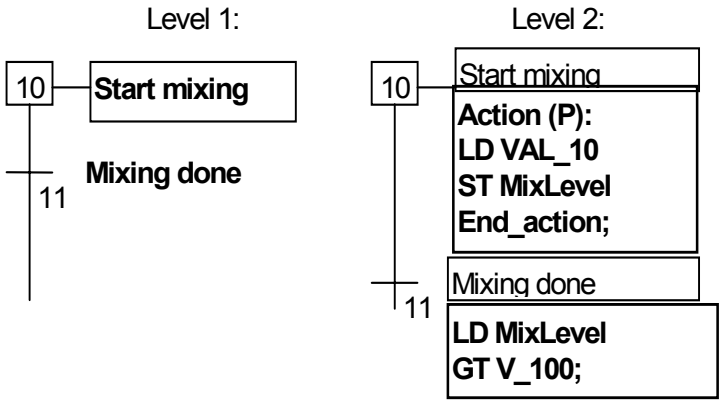
The SFC language is used to represent sequential processes. It divides the process cycle into a number of well-defined successive **steps** (self-contained situations), separated by **transitions**. Refer to the AC31GRAF Languages Reference Manual PART B for more details on the SFC language.

SFC components are joined by **oriented lines**. The default orientation of a line is **up** to **down**. These are the basic graphic components used to build an SFC chart:

- ..... Initial step
- ..... Step
- ..... Transition
- ..... Jump to a step
- ..... Macro step

☐..... Macro beginning step
☐..... Macro ending step

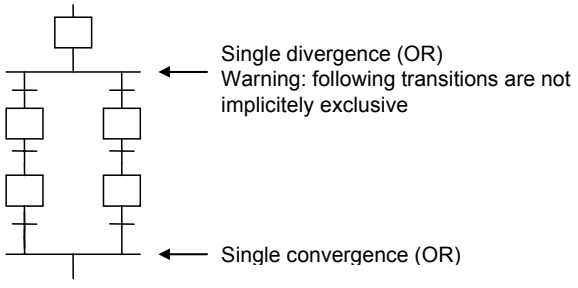
The SFC programming is usually separated into two different levels: The **Level 1** shows the graphic chart, reference numbers of the steps and the transitions, and comments attached to the steps and the transitions. The **Level 2** is the **IL** programming of the actions within the steps and **IL** or **Quick LD** for the conditions attached to the transitions. Actions or conditions may refer to **sub-programs** written in other languages (**FBD**, **LD**, **Quick LD** or **IL**). Below is an example of level 1 and level 2 programming:



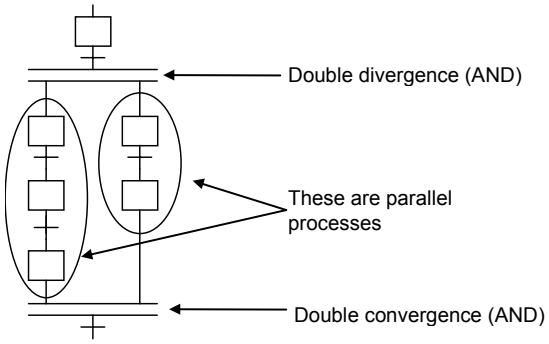
The level 2 programming of a step is entered in a text editor. It can include action blocks programmed in IL. The level 2 programming of a transition can be entered either in IL or Quick LD languages.

Divergences and convergences

Divergences and convergences are used to represent **multiple links** between steps and transitions. Simple divergences or convergences represent different **inclusive** possibilities between different sub parts of the process.

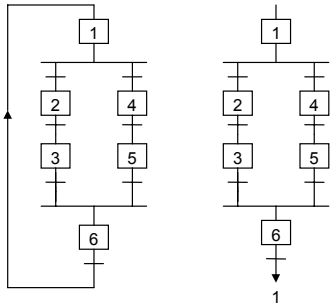


Double divergences represent **parallel** processes.



Jump to a step

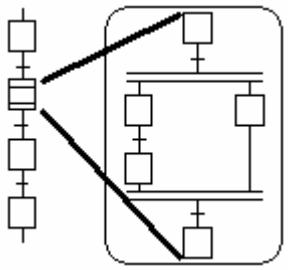
The SFC editor only allows the user to draw links in the **up** to **down** direction. A **jump** to a step can be used to represent a link to an upper part of the chart. Following charts are equivalent:



Jump to a transition is forbidden, and must be explicitly represented as a double (AND) convergence.

Macro steps

A macro step is a **unique** representation of a **stand-alone** group of steps and transitions. A macro step begins with a **beginning step** and terminates with an **ending step**.



The detailed representation of a macro step must be described in the same SFC program. The macro-step symbol must have the same **reference number** as the macro beginning step. A macro step description may contain another macro step.

4.2.2 Entering SFC chart

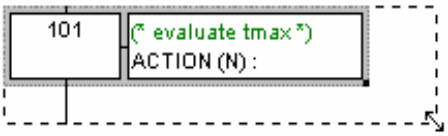
To draw an SFC chart, the user simply has to introduce the significant components of the chart. All the single lines joining two elements (horizontally or vertically) are drawn automatically by the SFC editor. To place an SFC component on the chart, the user has to move the selection to appropriate location and select the type of the component in the editor toolbar. The symbol is inserted at the current position. The following keyboard sequences can also be used:

F2: ..... Insert an initial step
F3: ..... Insert a single step
F4: ..... Insert a transition
F5: ..... Insert a jump to a step
F6: F7: ..... Insert an OR divergence or convergence / Add branches
↑F6: ↑F7: ..... Insert an AND divergence or convergence / Add branches
F8: ..... Insert a macro step
F9: ↑F9: ..... Insert begin or end step for the body of a macro step

(The "↑" symbol indicates a combination with SHIFT key)

The **editing grid** shows **matrix cells**. An editor option allows the user to show or hide the grid during chart input. The grid is very useful for initial layout of SFC chart, or selecting sub-parts of the chart. Use the "**Options / Layout**" command to display or hide the grid.

The AC31GRAF SFC editor always shows the current position in the matrix. The focused cell is marked in gray. The small square on its bottom right corner can be used to freely resize the cells. The X/Y ratio of the cells can also be changed this way.



Creating a divergence or convergence

Divergences and convergences are always drawn **from the left to the right**. To draw a divergence or a convergence, its **left corner** has to be placed on the chart area. The type of drawing (simple or double) is set by selecting one of these buttons in the toolbar.

F6: F7: ..... Insert an OR divergence or convergence / Add branches
↑F6: ↑F7: ..... Insert an AND divergence or convergence / Add branches

Adding branches to divergences

The **start** and **stop** position of each **auxiliary branch** is placed on the divergence or convergence line using these buttons in the toolbar. The left corner of the divergence or convergence must be present before inserting new branches. The right corners have the same style (simple or double) as the main left corner. Right corners cannot be placed if the main left corner has not been added.

F6: F7: ..... Insert an OR divergence or convergence / Add branches
*F6: *F7: ..... Insert an AND divergence or convergence / Add branches

F8: Inserting a macro step

This button is used to insert a macro step in the main chart. The body of the macro step must be entered elsewhere in the same SFC program.

F9: *F9: Body of a macro step

Macro steps must be described in the same SFC program as the main chart. A macro step must start with a **beginning step** and stop with an **ending step**. The sub-chart described as the macro implementation must be **self-contained**. The macro beginning step must have the same **reference** as the macro-step symbol of the main branch.

4.2.3 Working on an existing SFC chart

You can use either the mouse or keyboards arrows to select a rectangle area in the chart. The whole selected area is marked in gray. The commands of the "**Edit**" menu can then used:

Cut / copy / delete / paste commands

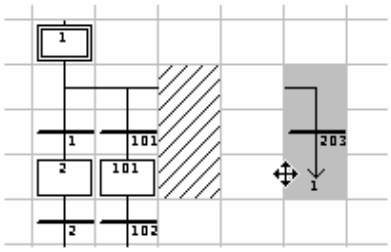
The following commands are available from the "**Edit**" menu when the "**arrow**" button is selected in the editor toolbar:

Cut ..... Move selected rectangle from the screen to the SFC clipboard
Copy ..... Copy selected rectangle from the screen to the SFC clipboard
Delete ..... Clear (delete) selected rectangle
Paste ..... Insert contents SFC clipboard at the current position

The "**Edit / Paste**" copies SFC clipboard to the screen. Copy / Paste commands work on both SFC chart and step/transition level 2 programming. It is also possible to copy a chart in a program and paste it in another SFC program. Elements are inserted before the currently selected position.

Move elements

When SFC elements are selected in the SFC chart, you can move them to another location of the chart by dragging the selection with the mouse. While you drag the selection, the initial location of selected elements is hatched.



The destination area for moved elements must be empty. No insertion is possible while moving SFC symbols.

Renumbering steps and transitions

Each step or transition is identified by a logical number in the SFC chart. The "**Edit / Renumber**" command allows the user to automatically set up numerically sequential reference numbers for any of the steps and the transitions of the currently edited SFC program. When a step number is changed, all the jumps to this step are automatically updated with the new reference number. (this also applies to macro steps and beginning steps)

Direct access to a step or transition

The "**Edit / Go to**" command allows the user to access an existing step or transition. The scrolling position is automatically adapted so that the step or transition is visible.

Find and replace texts

The "**Edit / Find Replace**" command can be used to find or replace text strings in the complete program (all steps and transitions). The Find/Replace dialog box is used to enter a searched text and directly open the level 2 programming section where text occurs.

4.2.4 Entering the level 2 programming

To enter the Level 2 text, the user must double click on the step or transition symbol. The level 2 programming is displayed on the right of the SFC window. The separation line between SFC chart and level 2 programming can be freely moved.

You can open one or two level 2 areas at the same time. The following commands are available from keyboard, mouse or the "Edit" menu:

	Keyboard	*Mouse*	*"Edit" menu*
Open in last default window	Enter	Double Click	Edit level 2
Open in separate window	Ctrl+Enter	Ctrl + DoubleClick	Edit Level 2 in separate window

When two level 2 windows are visible, the separation between them can be freely moved. The button on the right of the level 2 title bar is used to close a level 2 window.

The language for Level 2 programming is IL (Instruction List) or Quick LD. An independent window is opened to enter the level 2 programming. You can open several level 2 windows at the same time.

When working on a level 2 programming window, you can still access the commands of the "**Edit**" menu in the main window to work on the active level 2 text or diagram.

The "**Options / Refresh**" can be used at any time when level 2 windows are open to refresh the main SFC chart with modified level 2 programs.

Inserting a variable name

When programming in text language, press this button to select a variable declared in the list of variables and insert its name at the current position of the caret.

Inserting a Pulse action block in step

When programming the level 2 of a step, press this button to insert the template of a Pulse action block at the current position of the caret. Below is the format of a Pulse action block:

```
Action (P) :
   IL statement;
   ...
End_Action;
```

Pulse actions are instructions which are executed only once when the step becomes active. Refer to the AC31GRAF language reference for further details on SFC programming.

Inserting a Non stored action block in step

When programming the level 2 of a step, press this button to insert the template of a Non stored action block at the current position of the caret. Below is the format of a Non stored action block:

```
Action (N) :
   IL statement;
   ...
End_Action;
```

Non stored actions are instructions which are executed on every PLC cycle when the step is active. Refer to the AC31GRAF language reference for further details on SFC programming.

New P0 and P1 action qualifiers

AC31GRAF supports new **P0** and **P1** action qualifiers. When programming the level 2 of a step, press these buttons to insert the template of a P0 or P1 action block at the current position of the caret. Below is the format of such blocks:

Action (P0) :	**Action (P1) :**
IL statement;	IL statement;
...	...
End_Action;	**End_Action;**

P1 actions are instructions which are executed only once when the step becomes active (same as Pulse). P0 actions are instructions which are executed only once when the step becomes inactive. Refer to the AC31GRAF language reference for further details on SFC programming.

Boolean actions

Other text semantics are available to directly act on a boolean variable according to the step activity. Such actions consist of attaching the **step activity signal** to an internal or output boolean variable. This is the syntax of the basic boolean actions:

var (N); ..... assigns the step activity signal to the variable
var; ..... same effect (N attribute is optional)
/ var; ..... assigns the negation of the step activity signal to the variable

example :
%O62.00 (N); ..... assigns the step activity signal to the output %O62.00
%O62.00; ..... same effect (N attribute is optional)
/ %M00.00; assigns the negation of the step activity signal to the variable %M00.00

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

var (S); .....sets the variable to TRUE when the step activity signal becomes TRUE
var (R); ..... resets the variable to FALSE when the step activity signal becomes TRUE

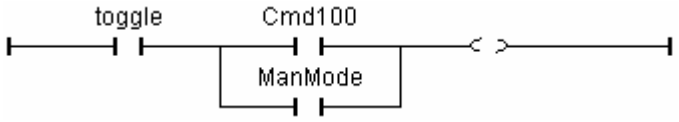
example
%O62.00 (S); ..... sets %O62.00 to TRUE when the step activity signal becomes TRUE
%M00.00 (R); ..... resets %M00.00 to FALSE when the step activity signal becomes TRUE

Transitions written in IL

The level 2 of a transition is a boolean expression. To program it in IL language, just enter the boolean condition according to the IL syntax. Optionally, a semi colon may be added at the end of the expression.

Transitions written in Quick Ladder

Quick LD editor is available to program the level 2 condition of a transition. In this case, the diagram is made of just one rung, with only one coil which represents the transition. The name of the transition is not repeated with the coil symbol. Below is an example of transition condition programmed in Quick LD.



When programming in Quick LD, use the keyboard arrows to move the selection in the programming logical grid, and then use the following shortcuts to insert symbols:

F2..... insert a contact the before selected symbol / initiate the rung
F3..... insert a contact after the selected symbol
F4..... insert a contact in parallel with the selected symbol
F6..... insert a block after the selected symbol
F7..... insert a block before the selected symbol
F8..... insert a block in parallel with the selected symbol

You can also click on the function key bar at the bottom of the level 2 window instead of hitting function keys.

Hit RETURN when the selection is on a contact or a block I/O parameter to select a variable or enter a constant value. Hit RETURN when the selection is on a function block to select the type of the function block. You can also double click on a symbol for the same effect.

Hit SPACE bar when a contact is selected to change the type of contact (direct, negated or with pulse detection). Refer to the chapter "Using the Quick LD editor" in this document for more details about Quick LD capabilities.

Verifying a program

The "**File / Verify**" command runs the AC31GRAF code generator to verify the programming syntax of the currently edited program. In case of SFC language, both level 1 and 2 are checked. When syntax verification is complete, the code generator window must be closed to continue work on the program.

Running the communication

The "**File / PLC communication**" command runs the AC31GRAF graphic debugger in connected mode, and re-opens the edited SFC program in debug mode. Used in debug mode, no modification can be entered in the program.

Editing variable list

The "**File / Variable list**" command is used to edit the list of variables for the current application and the current program.

Making a graphic

This "**File / Graphics**" command runs the graphic editor. This tool allows the user to define graphic images that will be refreshed during debug, based on the state of the application variables. The images are built with standard windows bitmap (.**BMP**) and icon (.**ICO**) files. This requires additional graphic editing tools, such as **PaintBrush**, to create bitmaps and icon files.

Printing the program

The "**File / Print**" command outputs the edited program on printer. This command produces a draft listing of the program. More detailed information is given when the project document generator is used.

For graphic programs (SFC, FBD), you can also use the "**Edit / Copy drawing**" command to copy in the clipboard the drawing of the chart in metafile format, so that it can be pasted in other applications such as word processors. For SFC programs, only the level 1 information (chart, numbering and level 1 comments) appears on the copied metafile.

4.2.5 Selecting a variable from list

When editing a text program (IL) the "**Edit / Insert variable**" allows the selection of a declared variable name to be inserted at the current position of the caret. When editing LD or FBD programs, variable selection is required for the description of contacts coils, block I/O parameters or FBD variable boxes. In both cases, the "Select" dialog box is opened to select a declared variable.

To select a variable, click on its name in the list. Its name and comment are then displayed on the different fields. Then press the "**OK**" button to confirm its selection. It is also possible to enter a variable name with a double click on the selected line in the list, or manually enter its symbol without using the list.

4.2.6 Commands of the "Tools" menu

The following commands are available in the Tools menu. They are used to display information in a small text list at the bottom of the SFC window:

Find in steps and transitions find occurrences of a text in all steps and transitions and list them in the output window
Hide output window close the output list window

When error messages or occurrences are displayed in the output window, double click on a line to directly open the level 2 programming window at the corresponding location.

4.2.7 Using the SFC gallery

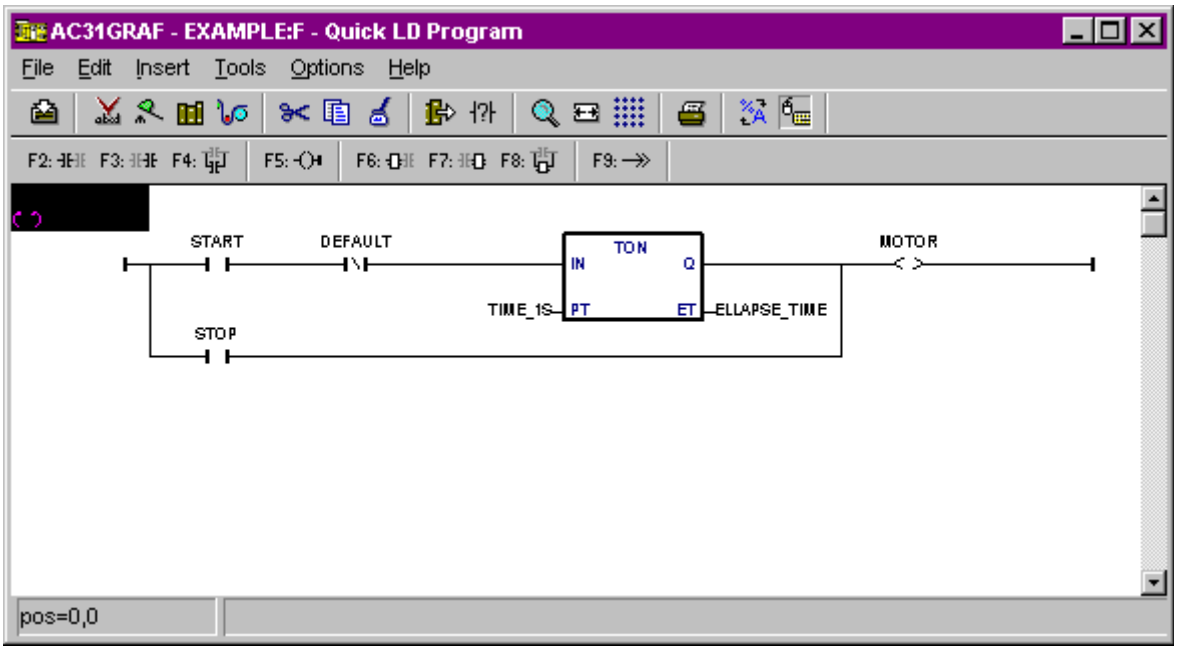
The AC31GRAF SFC editor manages an SFC gallery: it is a collection of SFC structures that can be inserted in any SFC chart. Elements of the SFC gallery can optionally embed the level 2 programming of steps and transitions. Use the following commands of the "**Tools**" menu:

Copy to SFC gallery ..... copy selected elements to SFC gallery
Paste from SFC gallery .... paste an SFC gallery element at the current location

When copying to SFC gallery (i.e. creating a new SFC gallery element), you can optionally ask to embed level 2 programming of selected SFC symbols.

4.3 Using the Quick LD editor

The LD language enables graphic representation of boolean expressions. Boolean AND, OR, NOT operators are explicitly represented by the diagram topology. Boolean input variables are attached to graphic contacts. Boolean output variables are attached to graphic coils. The AC31GRAF Quick LD editor provides easy LD diagram entering using either keyboard or mouse. Elements are automatically linked and arranged on rungs by the Quick LD editor. No connection is drawn manually by the user. The Quick LD editor also arranges rungs in the diagram so that the space filled by the diagram is always optimized.



4.3.1 Basics of the LD language

A LD program is expressed as a list of **rungs** where contacts and coils are arranged. Below are the basic components of an LD diagram:

Rung head (left power rail)

Each rung begins with a left power rail, which represents the initial "TRUE" state. AC31GRAF Quick LD editor automatically creates the left power rail when the first contact of the rung is placed by the user. Each rung may have a logical name, which can be used as a label for jump instructions.

Contacts

A contact modifies the boolean data flow, according to the state of a boolean variable. The name of the variable is displayed upon the contact symbol. The following types of contacts are supported by AC31GRAF Quick LD editor:

- ..... direct contact
- ..... negated contact

⊣P⊢ ..... contact with positive (rising) edge detection
⊣N⊢ ..... contact with negative (falling) edge detection

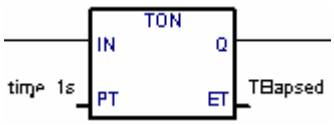
⊣()⊢ Coils

A coil represents an action. The rung state (state of the link on the left of the coil) is used to force a boolean variable. The name of the variable is displayed upon the coil symbol. The following types of coils are supported by AC31GRAF Quick LD editor:

⊣( )⊢ ..... direct coil
⊣(\)⊢ ..... negated coil
⊣(S)⊢ ..... "set" action coil
⊣(R)⊢ ..... "reset" action coil
⊣(P)⊢ ..... coil with positive (rising) edge detection
⊣(N)⊢ ..... coil with negative (falling) edge detection

Function blocks

A block in an LD diagram can represent a function, a function block, a sub-program or an operator. Its first input and output parameters are always connected to the rung. Other input and output parameters are literally written outside of the block rectangle.



Rung end (right power rail)

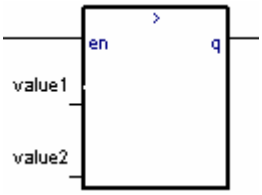
A rung ends with a right power rail. Using the Quick LD editor, the right power rail is automatically inserted when a coil is placed by the user.

Jump symbol

A jump symbol always refers to a rung label, id. the name of a rung defined somewhere in the same LD diagram. It is placed at the end of a rung. When the rung state is TRUE, the execution of the diagram directly jumps to this target rung. Note that backward jumps are dangerous as they may lead to a blocking of the PLC loop in some cases.

The "EN" input

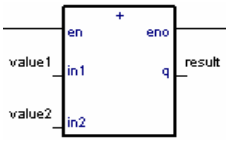
On some operators, functions or function blocks, the first input does not have boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE. Below is the example of a comparison operator, and the equivalent expression:



IF en is true
the comparison is valided and scanned else
q := FALSE;

The "ENO" output

On some operators, functions or function blocks, the first output does not have boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "**ENO**". The **ENO** output always takes the same state as the first input of the block.
On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in pseudo language:



IF en is true, the addition is valided and scanned
en = eno
(* continue rung with eno state *)

Limitations of Quick LD editor

The AC31GRAF Quick LD editor does not allow to continue a rung (insert other contacts or coils) on the right of a coil. If several outputs have to be made on the same rung, the corresponding coils must be drawn in parallel.

4.3.2 Entering a LD diagram

All the editing commands of the Quick LD editor may be achieved either with the keyboard or with the mouse.

The editing grid

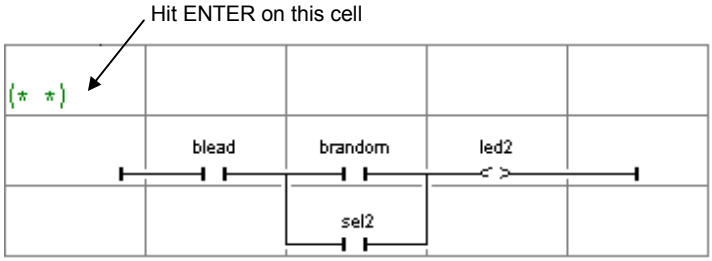
The LD diagram is entered in a logical matrix. Each cell of the matrix may contain up to one LD symbol. Use the arrows of the keyboard, or click on a cell to move the current selection. The selected cell is marked in reverse. For some cut/copy/paste operations, it is possible to select several cells. To do that with the mouse, just drag the mouse cursor in the diagram. With keyboard, use arrow keys with SHIFT key pressed.

Starting a new rung

To add a new rung to a diagram, move the selection after the last existing rung and insert a contact (hit F2 or press the corresponding button in the function key toolbar). A new rung with one contact and one coil is created.

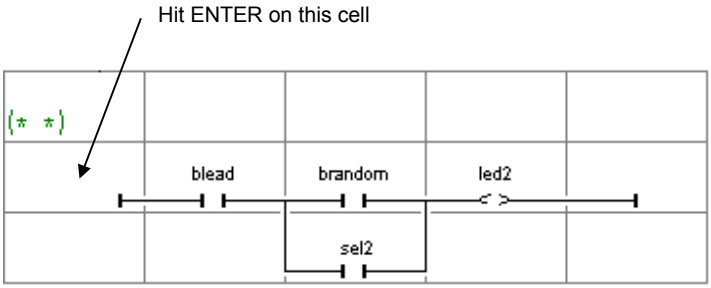
Entering the rung comment

Each rung may be documented with up to two lines of text. To enter a rung comment text, move the selection on the cell upon the rung and hit ENTER key, or double click on this cell with the mouse:



Entering the rung label

Each rung may be identified by a name. This name can be used as a target label for jump operations. To enter or change the label of a rung, move the selection on rung head and hit ENTER key, or double click on this cell with the mouse:



The AC31GRAF Quick LD editor keeps the memory of the rung labels you already entered, whether it has been specified for a rung name or a jump operation. The "Jump/Label"' dialog box gives you the possibility either to enter a new label, or to select an existing one.

If you enter a new name, it will automatically be added to the list. The "**Remove**" button is used to remove the selected name from the list. It does not remove the label on the rung you selected in the diagram. To do this, just press **OK** when the edit box is empty.

Inserting symbols on a rung

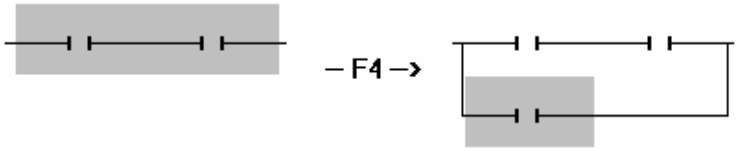
The insertion of symbols (contacts, coils, blocks...) on an existing rung is always made according to the current selection. You have to select a valid cell position within the rung and hit one of the following function keys to insert:

F2..... a contact before the selected symbol (on the left)
F3..... a contact after the selected symbol (on the right)
F4..... a contact in parallel with the selected symbol
F6..... a block before the selected symbol (on the left)
F7..... a block after the selected symbol (on the right)
F8..... a block in parallel with the selected symbol

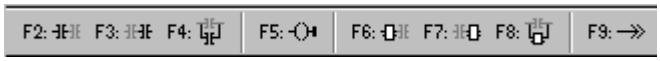
The following commands are valid when the selection is on the rung output (coil):

F5..... add a coil in parallel with the selected one
F9..... add a "Jump" symbol in parallel with the selected one
Shift + F9..... add a symbol "Return" in parallel with the selected one

For parallel insertion (F4/F8), if several contacts of a rung are selected together, the symbol is inserted in parallel with the group of selected elements. Below is an example:



To insert symbols in the diagram, you can also use the commands of the "**Insert**" menu. With the mouse, you can click on the key tool bar located at the bottom of the screen, on the type of symbol you want to insert:



Entering symbols

To associate a variable symbol to a contact or a coil, select it and hit ENTER. With the mouse, double click on the contact or coil. A variable selection box appears. Refer to chapter "More about program editors" in this document for further information about how to use this box. To associate a function, function block or operator to a block, hit ENTER when the selection is on the inside its rectangle. To associate a variable symbol to an input or output block parameter the selection must be on the corresponding location, **outside** the rectangle of the block.

If the mode “**Manual keyboard Input**” is selected in the “**Options**” menu, the variable name is directly catch in a field. Enter the new text and hit **Enter** to validate, or **Esc** to give up. The field used for this mode can not be closed with the mouse.

Changing the type of contacts and coils

The "**Edit / Change coil/contact type**" changes the type of the selected contact or coil. A contact may be direct, negated, with positive or negative edge detection. A coil may be direct, negated, set or reset, with positive or negative edge detection. Hitting the SPACE bar has the same effect.

Inserting a rung in a diagram

The "**Edit / Insert rung**" command insert a new rung in the diagram, before the selected one. The rung is initiated with one contact and one coil.

4.3.3 Working on an existing diagram

The commands of the "**Edit**" menu are used to change or complete an existing diagram. Most of these commands act on the elements currently selected in the diagram.

Correcting a diagram

The DEL key can be used to remove the selected elements. It is not possible to remove a coils, a jump or return symbol when it is the only output of a rung. Use "**Edit / Undo**" command to restore elements after a DEL command. The DEL command can also be applied to a group of elements selected in the diagram. The DEL command can be used when selection is on the rung comment text to reset it. The DEL command, used when the selection is on the rung head, remove the entire rung.

Copying symbols

The "**Cut**", "**Copy**", "**Paste**" commands of the "**Edit**" menu are used to move or copy selected elements. These commands do not act on rung comments. The "**Edit / Paste special**" command gives you the choice to insert the pasted elements:

- before the selected element (on the left)
- after the selected element (on the right)
- in parallel with the selected element

Managing entire rungs

All editing commands (delete, copy, cut...) act on the entire rung if the selection is on the rung header (left power rail). This provides an easy way to arrange rungs in the diagram, just by moving the selection in the first column. It is also possible to extend the selection vertically so that it includes several rung headers. In this case edition commands may be applied to a list of entire rungs.

Find and replace

The "**Edit / Find**" and "**Edit / Replace**" menu commands are used to find and replace texts in the diagram. Only complete names can be found. Search acts on contacts, coils, block names, block parameters and run labels. It cannot be used to find a string in a rung comment. The Replace command cannot be used to change the type of a block. The research can be made upward or downward, starting at position of the current selection. It "loops" when the limits of the diagram are reached. The following shortcuts are also available for quick research of variable names:

ALT + F2 finds the next element with the same variable name as the element currently selected. This feature can also be applied to function blocks and rung labels.

ALT + F5 finds the next coil with the same variable name as the element currently selected. This feature is mainly used in debug mode, to quickly find out the rungs which forces a suspicious variable.

4.3.4 Display options

The commands of the "**Options**" menu are used to customize the drawing of the LD diagram on the screen, and to hide or display some types of information.

Rung comments

Use the "**Options / Rung comments**" command to hide or display the rung comments in the whole diagram. Hiding the rung comments can be required to have a more condensed view on a huge diagram, as each comment consumes one row in the editing matrix. This option does not affect the contents of the existing rung comments, and can be swapped at any time.

Switching name and address

Each variable, when associated to a contact, a coil or a block I/O parameter is identified by its symbolic name. When using the menu "**Options / Name/address**" the symbol appears to replace the address or vice-versa

Rawing options

The "**Options / Layout**" command opens a dialog box where are grouped all the parameters and options concerning the editor workspace and the drawing of the graphic LD diagram.

Use the check boxes in the "Workspace" groupbox to display or hide editor tool bar, status bar and function key toolbar. Options of the "Document" groupbox allows you to show or hide points of the editing grid, and to enable/disable the use of colors for the drawing.


Options of the "Zoom" groupbox allows you to select a main zoom ratio. You can also use the "zoom" button in the editor toolbar to swap between default zoom ratios.


You can also customize the X/Y aspect ratio of cells in the editing grid. This last option can be used to reduce the default cell width, if you commonly use short names for variables. You can also use the "width" button in the editor toolbar to change the X/Y aspect ratio without entering the Layout dialog box.

4.3.5 Calling other AC31GRAF tools

Verifying a program

The "**File / Verify**" command runs the AC31GRAF code generator to verify the programming syntax of the currently edited program. In case of SFC language, both level 1 and 2 are checked. When syntax verification is complete, the code generator window must be closed to continue work on the program. If there is only one program in the application (the edited one) the application code is generated if no syntax error is detected. The "**Options / Compiling options**" command is used to set compiling

and optimizing parameters. Refer to chapter "Using the code generator" in this document for further information about compiling and code generation.

Running debugger

The "**File / PLC Communication**" commands run the AC31GRAF graphic debugger in real connected mode, and re-opens the edited program in debug mode. Used in debug mode, no modification can be entered in the program.

Editing the variable list

The "**File / Variable list**" command is used to edit the variable list of variables for the current application and the current program. It also contains the entry points to edit the user defined words. The **local** declarations or defined words relate to the currently edited program.

Printing the program

The "**File / Print**" command outputs the edited program on printer. This command produces a draft listing of the program. More detailed information is given when the project document generator is used.

For graphic programs (SFC, FBD and Quick LD) You can also use the "**Edit / Copy drawing**" command to copy in the clipboard the drawing of the chart in metafile format, so that it can be pasted in other applications such as word processors. For SFC programs, only the level 1 information (chart, numbering and level 1 comments) appear on the copied metafile.

4.3.6 Selecting a variable from variable list

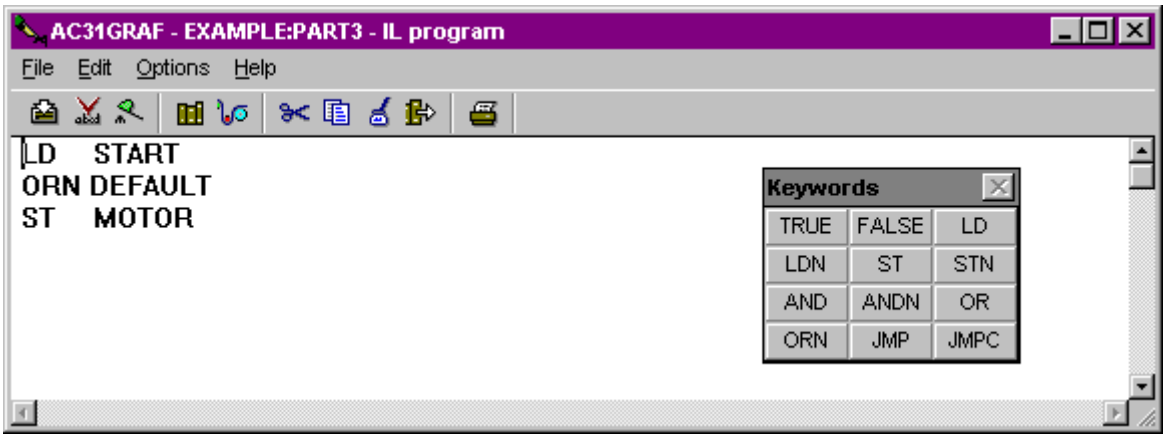
When editing LD or FBD programs, variable selection is required for the description of contacts coils, block I/O parameters or FBD variable boxes. In both cases, the "Select" dialog box is open to select a declared variable.

To select a variable, click on its name in the list. Its name and comment are then displayed on the top of the list. Then press the "**OK**" button to confirm its selection. It is also possible to directly enter a variable name in the edit control without using the list.

4.4 Using the IL editor

This chapter only describes features and commands of the AC31GRAF text editor, particularly when used to enter the source code of IL programs.

The AC31GRAF text editor is also used to enter the project descriptor, to edit diary files, technical notes (on line documentation) for library elements, and each time a text document has to be entered by the user.



4.4.1 File commands

Verifying a program

The "**File / Verify**" command runs the AC31GRAF code generator to verify the programming syntax of the current edited program. When syntax verification is complete, the code generator window must be closed to continue work on the program.
CAUTION: Verify doesn't compile: only the syntax of each instruction is verified
Missing instruction is not detected.

Running debugger

The "**File / PLC communication**" command runs the AC31GRAF graphic debugger in connected mode, and re-opens the edited IL program in debug mode. Used in debug mode, no modification can be entered in the program.

Editing the variable list

The "**File / Variable list**" command is used to edit the list of variables for the current application and the current program.

Making a Graphic

The "**File / Graphic**" command runs the graphic editor. This tool allows the user to define graphic images that will be refreshed during debug, based on the state of the application variables. The images are built with standard bitmap (**.BMP**) windows and

icon (**.ICO**) files. This requires additional graphic editing tools, such as **PaintBrush**, to create bitmaps and icon files.

Printing the program

The "**File / Print**" command outputs the edited program on a printer. This command produces a draft listing of the program. More detailed information is given when the project document generator is used.

4.4.2 Editing commands

The commands of the "**Edit**" menu are used to work on the edited text. Most of these commands act on the characters currently selected in the diagram, or perform an action at the current location of the caret.

Cut, paste, and copy

The DEL key can be used to remove the selected text. Use "**Edit / Undo**" command to restore elements after a DEL command. The "**Cut**", "**Copy**", "**Paste**" commands of the "**Edit**" menu are used to move or copy text in the program, or to insert pieces of texts copied in the clipboard by other applications.

Find and replace

The "**Edit / Find**" and "**Edit / Replace**" menu commands are used to find and replace texts in the program. Any character string can be found. Research can be performed upward or backward, starting at the current location of the caret. It does not "loops" when the limits of the program are reached.

Go to line

The "**Edit / Go to line**" command is used to move the caret to a specific line number. This can be very useful to have access to a line with an error detected by the AC31GRAF compiler in an IL program, and referenced by a line number.

Inserting a variable

Use the "**Edit / Insert variable**" command to insert at the caret position the symbol of a variable or object declared in the list of variables. Symbol is selected through the common variable selection box described in chapter "More about program editors" in this document.

Inserting a file

The "**Edit / Insert file**" command inserts the whole contents of a file at the current location of the caret. Note that only pure ASCII text files can be handled by this command.

4.4.3 Options

The commands of the "**Options**" menu are used to display or hide editor toolbars, and select the character font. The selected character font will be used for any text editing in all AC31GRAF Workbench.

When used to enter the source code of an IL program, the "**Options / Show keywords**" command is used to show or hide a toolbox that groups the most common keywords of IL language. Click on a button in the toolbar to insert the corresponding keyword or operator at the current location of the caret.

4.4.4 Selecting a variable from list

 When editing a text program (IL) the "**Edit / Insert variable**" allows the selection of a declared variable name to be inserted at the current position of the caret. When editing LD or FBD programs, variable selection is required for the description of contacts coils, block I/O parameters or FBD variable boxes. In both cases, the "Select" dialog box is open to select a declared variable.

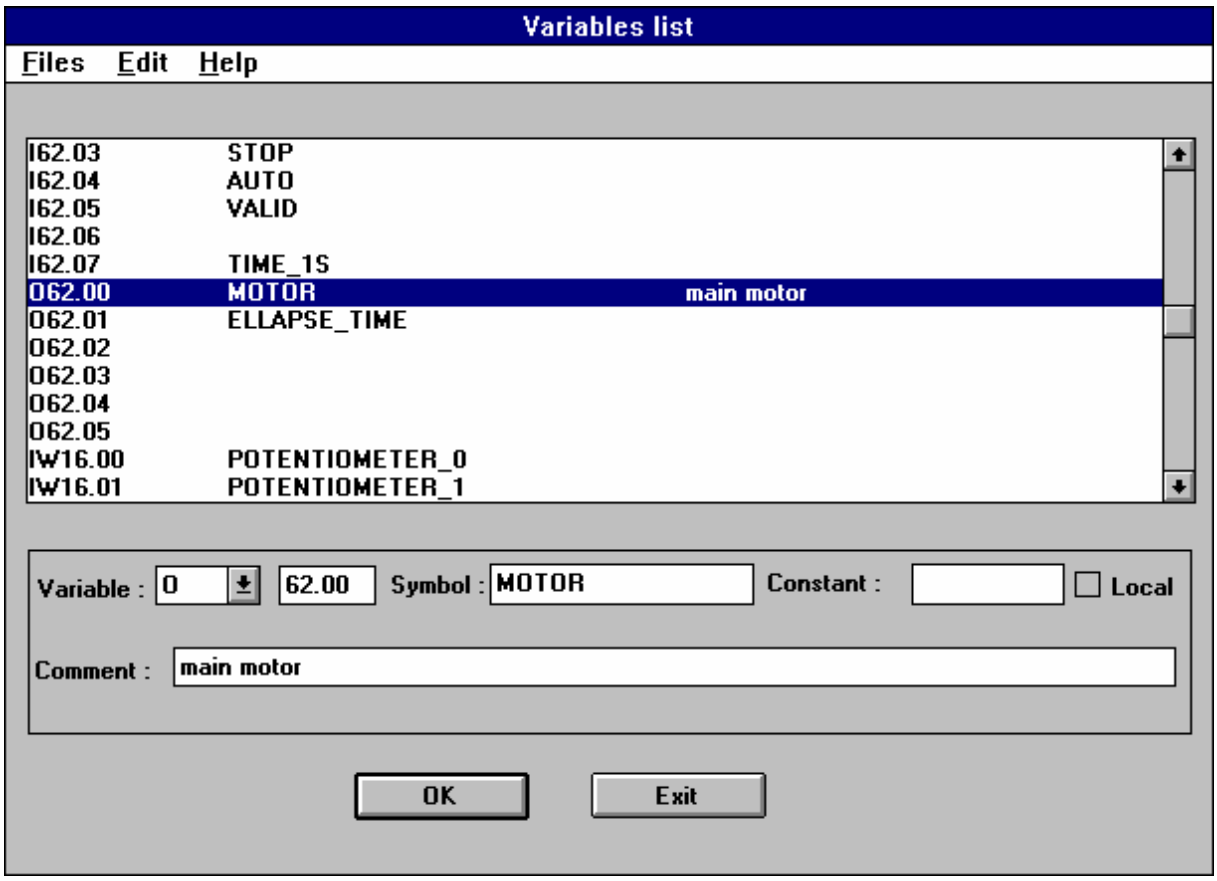
To select a variable, click on its name in the list. Its name and comment are then displayed on the different fields. Then press the "**OK**" button to confirm its selection. It is also possible to enter a variable name with a double click on the selected line in the list, or manually enter its symbol without using the list.

Instruction List, or **IL** is a low level language. It is highly effective for smaller applications or for optimizing parts of an application. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

5 Editor common tools

5.1 Declaring variables

The AC31GRAF variable list editor is an editing tool for the declaration of the internal variables of the application.



5.1.1 Using the variable list in declaration mode

When the variable list is edited, all the fields are filled with the characteristic of the last used variable, and the concerned field has just to be changed. So, the user can:

- change the symbol, the comment or the value of the variable constant in order to modify it or to change its address to enter a new variable.
- Select an address type among the list and to create a new variable.

The variable will be added to the list when the OK button will be pressed.
When leaving the variable list, the user is able to save all modification occurred to the list.

5.1.2 Using the variable list in selection mode

When the variable list is edited, by default, the last selected variable is highlighted in the list. The user can:
- modify the fields of this variable in order to create an other
- double-click on the variable in the list to take it
- click on the variable in the list and to press OK to take it.

The variable will be added to the list.

Fast way to enter a variable:
(for the select and declare mode)

When the list of variables is edited, the cursor is set on the variable type field.
The user can enter manually the type he wants without using the mouse and with the 'tab' key, he can reach the next field in order to complete the address…Finally, he press OK to enter the variable in the list.

Format of the variable list file
Each variable is described on one line of text. It is not possible to insert an empty line.
Here is the exact format of the different variable records.

<type>→ <address>→<symbol>→<comment>↵
<type>→ <address>→ → <comment>↵
<type>→ <address>→<symbol>↵
<type>→ <address>↵

type..... Variable type (ex : I,O, IW, …..).
address ..... Variable address (ex: 62.01).
symbol ..... Variable symbol (ex : MOTOR). If it is a constant, add "**=<constant_value>**" (ex : MOTOR=1)
Comment ..... Variable comment (ex : Motor command).
→ ..... Tabulation.
↵..... Carriage return.

What you have to know on the variable creation:

- **Address**

For TXT, # and #H type, there is no address.
For I, O, A, E, IW, OW, EW and AW type, the address format is : "cc.cc" or "cc,cc" where c is a digit. For the other type, the address format is "ccc.cc" or "ccc,cc".
For more information, to range for one type, select the type choose "**Edit / Range on".**
The maximum size for an address is 6 characters.
You can not have twice the same address.
You can not create a variable outside of its declaration area (see " **Edit / Range on** ").

- **Symbol**

The first character must be a letter (“a”-“z” or “A”-“Z”). The next characters must be some letters or digits.
A symbol must not begin by ‘GT”, “GS” or “LG”.
The words BVARTMP, SFC_EVOL and BVARTMP2 are reserved and are forbidden for symbol.
The maximum size for a symbol is 16 characters.
You can not have twice the same symbol.

- **Constant**

A constant type is : K, KW and KD. For these types, the field “constant” must be filled.
The value of a constant KW is between –32768 and 32767.
The value of a constant KD is between –2147483648 and 2147483647
The value of a constant K is 0 or 1.
The maximum size for a constant is 11 characters.

- **Comment**

The maximum size for a comment is 100 characters.

- **Variable scope**

A “local” variable is a variable visible only in a program, a sub-routine or a interrupt program.
A “global” variable is visible in an application. If a variable has no symbol, it is considered as a global variable for the application.

For the user library:

For the P, PW, PD, Q, QW, QD type, an address and a symbol are necessary.
For types I, IW, O, OW, S: no symbol, just an address.
For the M, MW, MD, KW, KD, just a symbol is necessary.

Saving the variable list

The command “**File / Save**” allows to save the state of the variable list.

Making the cross references

The command “**File / Cross references**” runs the cross reference tool.

Importing a variable list

The command “**File / Import**” allows to select a variable file on the disk and add all the variables on the bottom of the list. The source file can have a PC33 file format, in this case, the checkbox "PC33 format" must be selected.
If an address is already used in the project, a message is displayed to inform the user and the variable is not set in the list, the same thing happens when a variable is not inside the range value.
If the file is in an 907PC331 format, it is set in the new one.

- **Exporting a variable list**

The command “**File / Import**” allows to select variables in the list, record them in a file on a selected directory. (the file extension is ".dcl")

- **Cut variables**

The command “**Edit / Cut**” erases the selected line(s) in the list and record it in the clipboard.

- **Copying variables**

The command “**Edit / Copy**” records the selected line(s) in the list in the clipboard.

- **Paste variables**

The command “**Edit / Paste**” writes the contain of the clipboard in the variable list.

- **Inserting an empty line**

The command “**Edit / Insert**” inserts an empty line above the selected one.

- **Removing a symbol**

The command “**Edit / Remove symbol**” removes the symbol of the selected variable.

- **Removing a comment**

The command “**Edit / Remove comment**” removes the comment of the selected variable.

- **Searching an address**

The command “**Edit / Search**” allows to search a defined address and highlight the line where the variable is found.

- **Sorting the variable list**

The command “**Edit / Sort**” allows to set the list in the alphabetic order on the variable addresses (I, IW, K, KW,…). If you save the sorted list and you open again the variable list editor, the variables will be sorted in the alphabetic order. But the global variables will be displayed in first and the local variables after.

- **Authorized values area**

The command “**Edit / Range on type**” allows to show the authorized values area for a selected type. Before to use this command, select a variable in the list with the correct type (for example the variable “O 62.00” to see the authorized values area of the type “O”).

Creating a variables area

The command ‘’**Edit / Variables area**” runs a dialog box containing the current address type and allowing to select an area of addresses. When clicking on the OK button, all the variables are added to the bottom of the list. If an address is already used in the project, a message is displayed to inform the user and the variable is not set in the list, the same thing happens when a variable is not inside the range value.

Ex:

If a user selects the M type and the following area:
from: 2.3
to: 2.6

the new addresses added to the list are:
M 002.03
M 002.04
M 002.05
M 002.06

5.2 Cross References

The AC31GRAF includes a cross-reference editor which provides user with a total view of the declared variables in the project programs, and where they are used. The aim of a cross reference is to list all the variables declared in the project, and to localize at the source of each program the parts of source code where those variables are used. The cross references are very useful for a global view of one variable life cycle. They help to localize side effects, and to reduce the time to understand the project during the maintenance. The cross references may also be used for a global view of the complete list of variables of a project, so unused variables are easily found and the complexity of the project measured.

From the variable list editor, the command “**File / Cross references**” runs the Cross reference editor.

The list on the left shows the declared objects of the project (programs, variables and defined words), and the library elements (functions and function blocks) referenced in the project. The list on the right shows the occurrences in the programs of the object currently selected in the first list.

- **Object type selection**

Because a project can group a huge number of declared objects, the combo box in the editor toolbar is used to select the type of objects which must be listed in the window. This allows the user to have access to selected information.

Each time the cross references are re-calculated, the selection is reset to "**All objects**" in order to present the complete list.

- **Re-calculating cross references**

The "**File / Re-calculate**" command can be used at any time to update the cross references according to the modifications entered in other AC31GRAF editing windows.

- **Exporting cross references**

The "**File / Export**" command is used to write the complete listing of the cross references in an ASCII text file. This file can then be opened with other applications such as Windows **NotePad** or word processors.

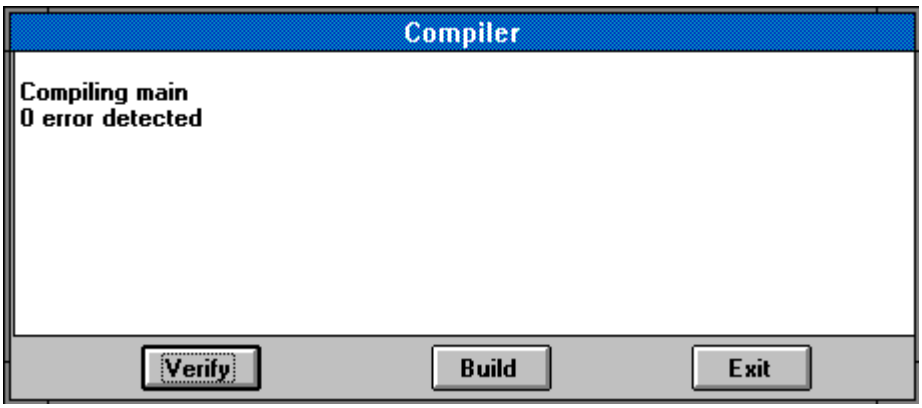
- **Searching an object**

The "**Edit / Search**" command allows the user to directly select an variable in the editor list. The searched variable cannot be found if it is not actually listed (when using a selected display). It is recommended to activate the "**All**" selection in the toolbar before searching for a variable.

Opening a program

The list on the right contains the occurrences of the selected object in the source files and I/O connection of the open project. The "**Edit / Open program**" command enables the user to directly open a program where the object appears. It is also possible to double click the mouse on an occurrence (in the occurrence list) to open the corresponding program.

5.3 Build the application



The window of build application is automatically open with the "**File / Verify**" command from the editors or with the "**Make / Verify**" command from *the program management* window for a modularized project.

Verifying a program

The "**Verify**" button allows the user to verify the syntax of the program currently selected (in the program management window or the program currently selected).

Building a project

The "**Build**" button allows the user to generate the code for a project.
When building a project, a verified program will not be re-verified, until its contents has not changed.
Indeed, during the build phase, every program that is still not verified is checked to detect the syntax error.
If no error occurs during the build phase, the code is generated and ready to be downloaded.

Errors detected

If there is an error, a double-click on a line error sets the cursor
- on the line containing the error for the IL language
- on the picture containing the error for the FBD and LD language
- or on the current line error of the step or transition in cause for the SFC language.

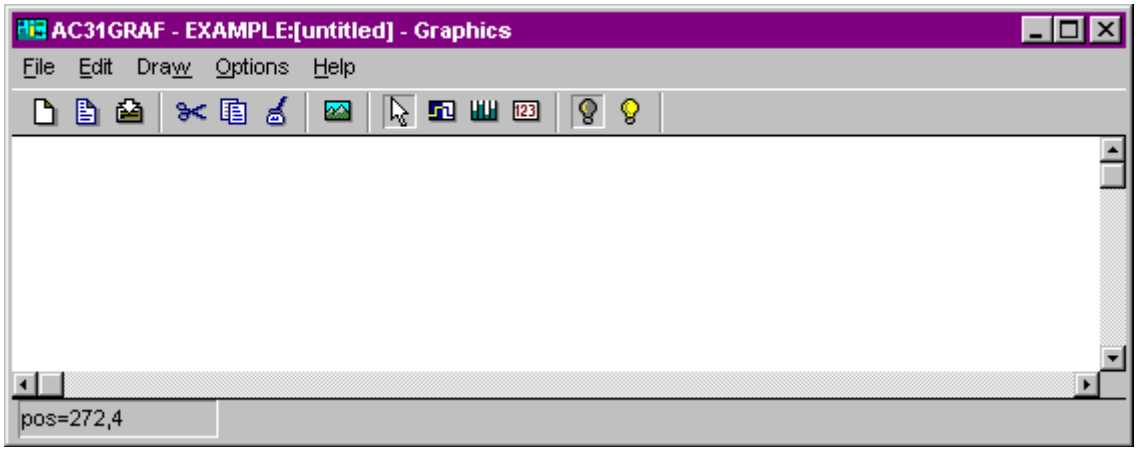
The window can be resized, so the user can read both his errors and his source code.

Exit

The "**Exit**" button closes the compiler window.

5.4 Creating graphics

AC31GRAF contains a graphic editor, which allows the user to define graphic pictures that can be animated during debug. Graphic objects must be linked to the variables of the AC31GRAF project. The graphic picture is defined "off line", and cannot be modified once the communication is running.



5.4.1 Drawing chart

A chart is made of one background picture, and of a set of graphic objects that will be animated during debug. To enter the chart, the following operations must be performed:

- Select a background picture
- Insert graphic objects
- Link objects to the variables of the project

The following graphic objects can be inserted in the chart :

Icon (boolean)..... Boolean display: two icons are used to display either FALSE or TRUE state of the linked variable.

Bar graph (integer)..... Rectangle filled with one solid color. The size of the rectangle is adjusted according to the value of the linked analog variable.

Numerical (integer) ..... Numerical display corresponding to the value of the linked analog variable.

The background picture

The background picture is contained in a "bitmap" (.**BMP**) file. This picture either is built with another tool, or can be copied from an existing graphic library. The AC31GRAF graphic editor does not include a painting tool. The most common tool to build bitmap files is **Paintbrush**, delivered with **Windows**. The background picture can have any size. If the AC31GRAF graphic editing window is larger than the background picture, unused space can be filled with any solid color. The "**Draw /**

Background picture" command is used to select the bitmap file and also to select a solid color for empty space. The bitmap file must exist before it is inserted in the chart. Press the "**Browse**" button of the dialog box to select an existing pathname on the disk. A background color can be selected to fill unused space, if the selected bitmap is smaller than the editing window.

Note: Bitmaps consume a large amount of memory. It is highly recommended to correctly size the picture, and limit the unused space inside the bitmap rectangle.

Icons

Icons used to display boolean objects are stored in standard icon files (.**ICO**). These are 32x32 pixel/16 color pictures, which can use the "transparent" color. Any icon editor can be used to create .**ICO** files.

Bar graph and numerical displays

"Bar graph" and "numerical" objects can be linked to analog (or integer) variables. The sizing direction of "bar graph" objects can be to the left, to the right, to the top or to the bottom. Any variable of the AC31GRAF project (global or local to one program) can be linked to a graphic object.

Inserting objects

To insert a graphic object in the chart, the following operations have to be performed:
- Select the type of object in the editor toolbar or in the "**Draw**" menu
- Click at the insert position in the graphic area
- Enter the definition of the object

Analog "bar graph" and "numerical" objects have a variable size. To insert such an object, the user has to define the boundary of the rectangle, by dragging the mouse in the drawing area. The following buttons of the editor toolbar are used to select a type of graphic object:

..... Selection mode (no object can be inserted)
..... Insert boolean icon
..... Insert analog bar graph
..... Insert analog numerical field

Selecting objects

Selecting graphic objects is needed for most of the editing commands. The AC31GRAF graphic editor enables the selection of one or more existing objects in the chart area. To select objects, the "**select**" (button with an arrow) choice must be checked in the editor toolbar. You can run the "**Draw / Select**" command or hit the **ESCAPE** key to set the selection mode. To select one object, the user simply has to click on its symbol. To select a list of objects, drag the mouse in the drawing area to select a rectangle area. All the graphic objects that intersect the selection rectangle are marked as "**selected**". A selected object is drawn with little black squares around its graphic symbol.

By making a new selection, any previously selected objects are unselected. To remove the existing selection(s), simply click with the mouse on an empty area outside of the rectangle which borders the selected objects.

Moving and resizing objects

To move one or more objects in the edited chart, select them and drag them using the mouse to another location. Some objects, such as analog bar graphs and numerical fields, have variable height and width. The AC31GRAF graphic editor enables the user to change the size of such an existing object. To do this, click on the border of the selected object, and drag the mouse to change the object size. Boolean "icon" objects have a fixed size (32x32 pixels) and cannot be resized.

Cut and paste

The commands of the "**Edit**" menu are used to work on currently selected objects. Below is the list of available commands:

- Cut the selected objects (and copy them in the clipboard)
- Clear (delete) the selected objects
- Copy the selected objects in the clipboard
- Paste the clipboard in the diagram

Pasted objects are marked as **selected** objects, so they can be moved to another place in the diagram.

Note: The AC31GRAF graphic editor uses a dedicated clipboard, which contains logical information about linked graphic files and variable names. This is not the standard graphic or text clipboard of Windows.

Icons and bitmaps

Icon and bitmap files are created using other tools. The only information stored in the AC31GRAF graphic definition file is the pathname of these files. It is better to store icon and bitmap files in the AC31GRAF project directory. Therefore, only the filenames (and not complete pathnames) are stored. This ensures that the AC31GRAF graphic file can be transported to another platform, which may have different directory pathnames. If the AC31GRAF Workbench has been installed in the "**c:\AC31GRAF**" root directory, the directory which contains the project named "**ppp**" has the following pathname :

c:\AC31GRAF\apl\ppp

Warning: Compressed bitmap files (.**BMP** files with Run Length Encoding compression) are not recognized by the AC31GRAF graphic editor.

5.4.2 Object description

The parameters of an existing object can be modified, by double clicking on its symbol in the graphic area. When the "**Name**" field is the active field, pressing the "**Browse**" button enables the user to find the names of the variables already declared

in the list of variables. When an "**Icon file**" field is the active field (for a boolean icon), pressing the "**Browse**" button enables the user to find the pathname of any of the existing icon files.

Below is the list of fields that must be entered when defining a boolean "icon" object:

Name ..... Name of the AC31GRAF boolean variable linked to the graphic object.
"False" icon file ..... Pathname of the icon (.ICO) file used to display the False" state of the linked boolean variable.
"True" icon file ..... Pathname of the icon (.ICO) file used to display the "True" state of the linked boolean variable.
Default value ..... Value used to display the object before it is first refreshed by the debugger.
Command variable ..... If this option is checked, the user can modify the value of the linked variable during debug by double clicking on the corresponding graphic symbol.

Below is the list of fields that must be entered when defining an analog "bar graph" object:

Name ..... Name of the AC31GRAF analog variable linked to the graphic object.
Range – minimum ..... Indicates the minimum value that can be displayed. If the value is less than this limit, the bar graph is not displayed.
Range – maximum ..... Indicates the maximum value that can be displayed. If the value is greater or equal to this limit, the bar graph rectangle is completely filled.
Default value ..... Value used to display the object before it is first refreshed by the debugger.
Direction ..... Indicates the growing direction of the filled part of the bounded rectangle. When the variable value increases, the filled rectangle must grow to the left, to the right, to the top or to the bottom.
Color ..... Color used to draw the filled part of the moving bar.
Command variable ..... If this option is checked, the user can modify the value of the linked variable during debug by double clicking on the corresponding graphic symbol.

Below is the list of fields that must be entered when defining an analog "numerical" object:

Name ..... Name of the ABB AC31GRAF analog variable linked to the graphic object.

Format ..... Indicates whether the value is displayed as an integer or real value. The decimal part of a displayed real value is truncated if the "integer" format is selected.

Justification ..... Indicates how the text (numerical value) must be justified (left, right or center) in the bounded rectangle.

Default value ..... Value used to display the object before it is refreshed by the debugger.

Command variable ..... If this option is checked, the user can modify the value of the linked variable during debug by double clicking on the corresponding graphic symbol.

5.4.3 Commands of the "File" menu

The "**File**" menu contains the commands which allow the user to manage the complete chart as a file.


The "**File / New**" command starts the editing of a new graphic chart. The number of charts defined for a project is not limited by AC31GRAF. Before editing the new chart, the previously opened chart is closed. The AC31GRAF graphic editing window cannot be used to edit several charts at once. However, multiple AC31GRAF graphic editing windows can be opened simultaneously with each used to edit a different chart.


The "**File / Open**" command allows the user to close the currently edited chart and to start editing another chart of the current project. The new selected chart replaces the current one in the editing window. When selecting the new chart, the "**Delete**" button can be used to delete an existing chart, in order to clean up the project directory. Icon and bitmap files referenced in a chart are not erased when the chart is deleted.


The "**File / Save**" command stores the currently edited chart on the disk. If the chart is a new untitled document, the user must give it a name before saving it. Naming a chart must conform to the following rules:

- The length of the name cannot exceed **8** characters
- The first character must be a **letter**
- The following ones must be **letters**, **digits** or **underscore** characters
- Naming is case insensitive

The "**File / Save as**" command allows the user to store the currently edited chart under another name.

5.4.4 Options

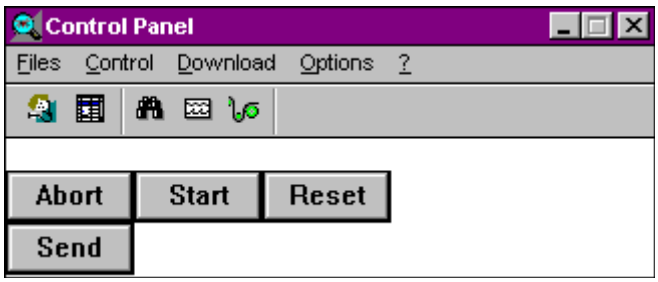
The "**Options**" menu contains the graphic editing options for displaying the toolbar, cursor coordinates, or selecting the state of displayed boolean objects.


The "**Options / View false state**" and "**Options / View true state**" commands are used to select the state (false or true) of the displayed boolean "icon" objects. By selecting one of these options, the user can simulate the state of the inserted

boolean objects, in order to display either "false" state or "true" state icons. These commands apply to all inserted objects. Object states cannot be changed individually.

6 Control panel

6.1 Using the control panel



This window is the main window of the **PLC communication** menu.
It can be reduced to the state of an icon, or to the state of a minimum graphic as below :

In this state, the window is set at the right corner at the top of the window.

 Close the communication

 Display the extended graphic

 Control menu

 Transfer menu

In its normal state there four buttons below the menu items, these are:

 Download the program and the constant (the user program is stored into the flashEprom). The configuration is not sent

 Run the program

 Warm restart of the program

 Abort the running program

Editing the on line list

The “**File / On line List**” edits the on-line list window. For further information see the “On line list” chapter.

Making time diagrams

The “**File / Time diagrams**” command runs the time diagram tool. This tool allows the user to watch time diagrams corresponding to the changes in the lists of variables.

Making a graphic

The "**File / Graphics**" command runs the graphic editor. This tool allows the user to define graphic images that will be refreshed during debug, based on the state of the application variables. The images are built with standard windows bitmap (.**BMP**) and icon (.**ICO**) files. This requires additional graphic editing tools, such as **PaintBrush**, to create bitmaps and icon files.

Displaying the central unit status

The “**File / Status**” command edits the status window. This windows display some information about the central unit. For further information see the “Status / diagnosis” chapter.

Configuration

The “**File / Configuration**” command edits the configuration window. This window allows the user to accede to central unit parameters. For further information see the “Configuration” chapter.

Aborting a PLC program

The command “**Control / Abort PLC program**” aborts the program in the central unit. It is no longer running, when launching this command.

Starting a PLC program

The command “**Control / Start PLC program**” runs the program in central unit.

Restarting a PLC program

The command “**Control / Reset (Warm) of central unit program**” restarts the program in its current state in the central unit.

Cold-restarting a PLC program

The command “**Control / Cold restart of central unit program**” restarts the program in its initial state (memory and configuration are initialized) in the central unit.

Delete PROM

The command “**Control / Delete PROM**” deletes the program on the **FlashEPROM**.

- **Save PROM contents**

The command “**Control / Save PROM contents**” cancels the **flashEPROM** of the central unit.

- **Reactivating a former program**

The command “**Control / Reactivate program**” reactivates the former program.

- **Erasing PROM contents**

The command “**Control / Erase PROM contents**” fills the central unit program memory with **NOP** instructions.

- **Optimizing a PLC program**

The command “**Control / Optimize program on PLC**” suppresses the **NOP** instructions.

- **Sending a program**

The command “**Control / Send program**” downloads the program (prog.ABB) into the central unit.

- **Sending constants**

The command “**Control / Send constants**” sends the constants of the application.(not the system constants) into the central unit.

- **Sending all**

The command “**Control / Send all**” downloads the program (prog.ABB) and the constants of the application, except the system constants into the central unit.

- **Download / Send system constants**

The command “**Control / Send system constants**” sends the system constants (It is also be done from the configurator editor) into the central unit.

- **Minimizing the window**

The option “**Options / Minimize window**” takes the shape as described on the top.

- **Window always visible**

If the option “**Options / Always on top**” is selected, the window is always above all other windows.

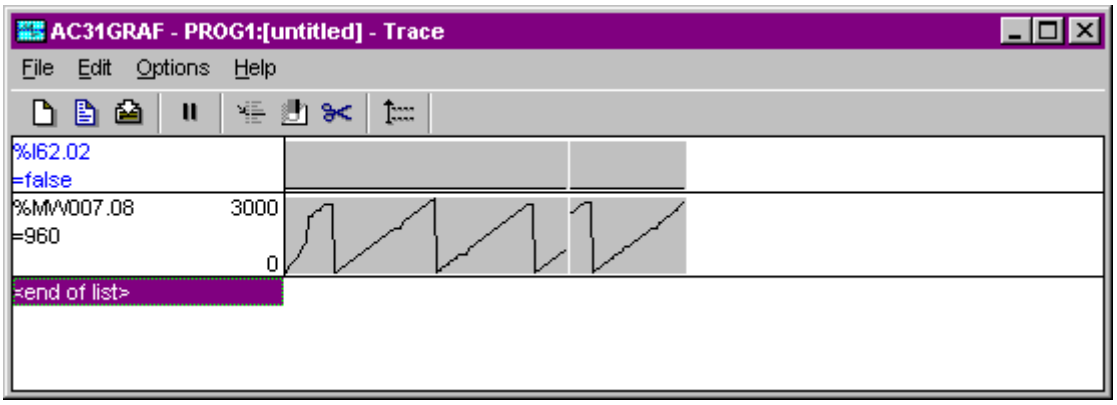
On the next selection of this menu item, the window no longer on this state.

- **Editing the historic of the errors**

The command option “**Options / Error list**” edits the historic of the errors occurred with the date and the time.

6.2 Time diagrams

The "**File / Time diagrams**" command of the Communication window enables the user to watch time diagrams corresponding to the changes in the lists of variables. Lists are built when debugging the application. They can be stored on the disk and opened again during other debug sessions. A time diagram list can contain up to **8** variables. Variables of different types can be mixed in the same list. Global and local variables can be inserted in a list. A list is dedicated to one particular project.



Tracing variables is very useful for the functional testing of an application. Even though the displayed time diagrams are not precise measurements, they allow the user to control the synchronization of process events.

Saving lists on hard disk

The commands of the "**File**" menu are used to create, open and save the lists of time diagrams. The number of lists for one project is not limited by AC31GRAF. While naming the lists of variables to be saved on the disk, the following rules have to be followed:

- name cannot exceed **8** characters
- the first character must be a **letter**
- the following characters can be **letters**, **digits**
- naming of lists is case insensitive

The time diagram editor cannot display more that one list of variables at a time in the same window. However, the time diagram editor can be run more than once, in order to spy different lists simultaneously.

Inserting variables in the list

The "**Edit / Insert**" command inserts another variable in the list of time diagrams. The variable name is selected in the list of variables defined in the project list variables. The user does not have to manually enter the identifier. The variable is inserted before the variable currently selected in the list. The list cannot contain more than **8** diagrams. The same variable cannot appear more than once in the same list. If the

project is modular, the user can choose the program for the local variables he wants to see.

Changing the selected variable

The "**Edit / Modify**" command replaces the selected variable by another variable. You can also use the "**Cut**" command to remove the selected variable from the list.

Diagram scaling

The "**Edit / Set scale**" command sets up the minimum and maximum values for the timing diagram of the selected variable. This command has no effect if the selected diagram shows a boolean variable.

Setting refresh duration

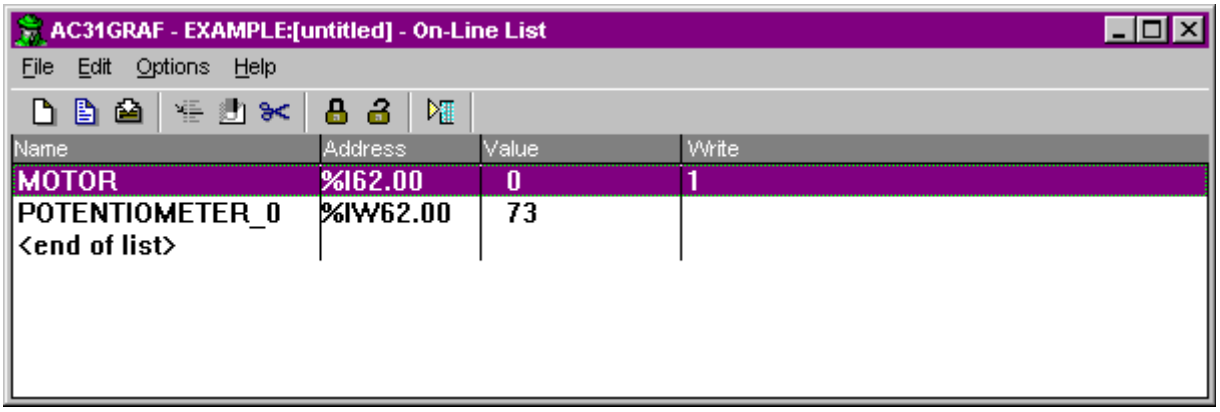
The "**Options / Cyclic refresh duration**" command enables the user to change the time scale of the timing diagrams. The time scale is defined by entering the cyclic refresh duration, which is the time elapsed between two consecutive points of a timing diagram.

Pause and resume

Another possibility is to pause the cyclic diagram refresh. The **ESCAPE** key can also be used to pause or restart refreshing diagrams.

6.3 On line list

The "**On line list**" command enables the user to build non-contiguous lists of variables which are refreshed with their current values. Lists are built when debugging the application. The lists can be stored on the disk and opened again during other debug sessions. Variables of different types may be mixed in the same list. Global and local variables can be inserted in a list. A list of variables is dedicated to one particular project. The on line list is very useful for the functional testing of an application. It allows the user to watch the changes of a limited part of the controlled process, independent of the corresponding source code in the application programs. The on line list is also useful while debugging IL text programs. The user can easily group in a list the set of variables used in a program, in order to control or monitor the execution of the programmed instructions.



To modify a value, double click on a variable and enter a new value or erase the current value. The new value appears in the column “**write**”.

Creating a variable list

The “**File / New**“ command creates a new list of variables.

Opening a variable list

The “**File / Open**“ command allows the user to select and open a variable list.

Saving a variable list

The "**File / Save**" command save a variable list. The number of lists for one project is not limited by AC31GRAF. While naming the lists of variables to be saved on the disk, the following rules have to be followed:

- a name cannot exceed **eight** characters
- the first character must be a **letter**
- the following characters can be **letters**, **digits**
- naming of lists is case insensitive

The list editor cannot display more that one list of variables at a time in the same window. However, the list editor can be run more than once, in order to spy different lists simultaneously.

Inserting a variable

The “**Edit / Insert**” command inserts one or more variables into the On line list which are selected from the list of variables displaying the global and local variables for a selected program. In the list of variables, you select continuous variables by selecting them while holding down the **Shift** key and non-continuous variables by selecting them while holding down the **Ctrl** key. Inserting variables eliminates the need to manually enter identifiers. Selected variables are inserted before the variable currently selected in the On line list. A variable cannot appear more than once in a given On line list.

If a variable has no symbol, its address appears in the column name and column address. Whereas, if a variable is defined with a symbol, the latter is added in the column name.

Modifying a variable

The “**Edit / Modify**” command replaces the selected variable by another variable. You can also use the **" Edit / Cut"** command to remove the selected variable from the list.

Cut a variable

The “**Edit / Modify**” command deletes the current selected line.

Sending new values

The “**Edit / Send new values**” command allows to download to the central unit the new value entered in the column “**write**” to the selected variable.

Resetting new values

The “**Edit / Reset new values**” command erases the column “**write**”.

Locking a selected variable

The “**Edit / Lock selected variable**” command is similar to the send command, but the variable is locked.
Only for the input/output variables.

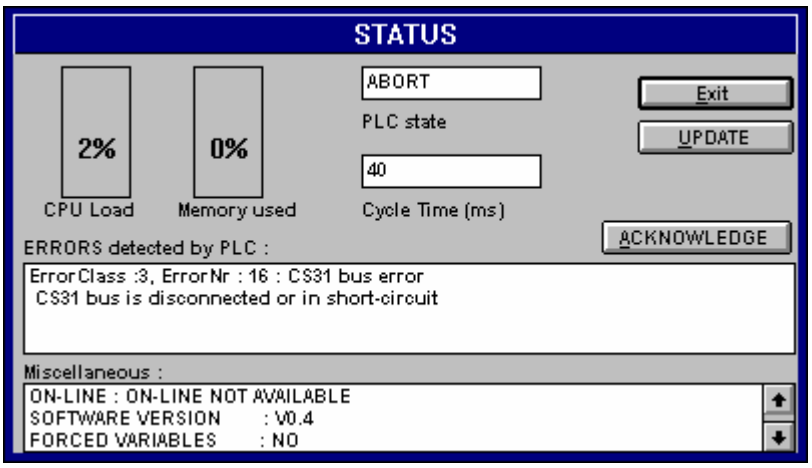
Unlocking selected variable

After having selected the “**Edit / Unlock selected variable**” command, the selected variable stops to be locked.
Only for the input/output variables.

Adding list of locked variables

The “**Edit / Get list of locked variables**” adds the list of locked variables to the current list in the central unit.

6.4 Status / Diagnosis

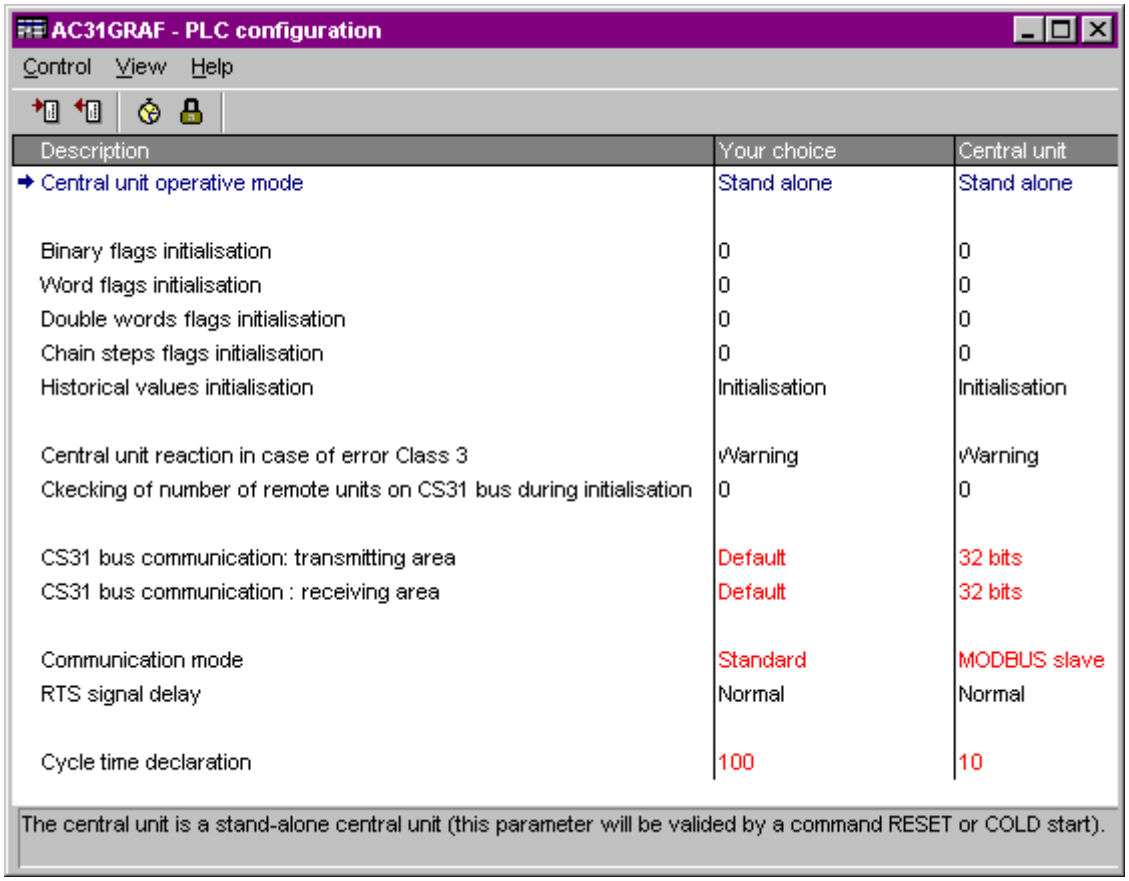


The diagnostic command is run from the control panel and is always available when the AC31GRAF is used in On Line mode. It displays diagnostic information from the central unit.

Cycle Time (ms)..... the central unit cycle time is displayed in ms
PLC State ..... the central unit state (RUN or ABORT) is displayed
Bar-graph ..... one represents the CPU load and the other one, the proportion of the memory used.
Update ..... refresh the bar-graph and the different messages displayed.
Errors detected by PLC ... the error messages emitted by the central unit are analyzed by AC31GRAF and converted into text strings. Each error is identified by a class number, an error number in the class and a meaning. If an address number is indicated with the error message, you can double clicked on the message to open the file of the IL code program (prog.txt).
Acknowledge ..... acknowledge the central unit error.
Miscellaneous..... display different information according to the central unit type.

Refer to the technical description of each central unit for more details.

6.5 Configuration



The configuration tool provides the way to display, enter, upload, download and verify main central unit parameters.

The list of parameters depends on the central unit type and this list is automatically selected at the beginning of the project

Parameters are stored in system constants in the central unit. Such constants are not visible from the program.

The configuration tool may be used in both off-line and on-line modes.

A template of default values is created when a new project is created.

The parameters are displayed in a listbox :
The meaning of each parameter is described on the bottom of the window

Your choice column of the listbox contains values registered on PC. When you leave the configuration tool (by clicking on 'EXIT' choice), your choice is automatically saved.

The central unit column displays the values that are stored on the central unit. Each time the configuration tool is launched, the system constants are read and the result is displayed inside the «central unit» column.

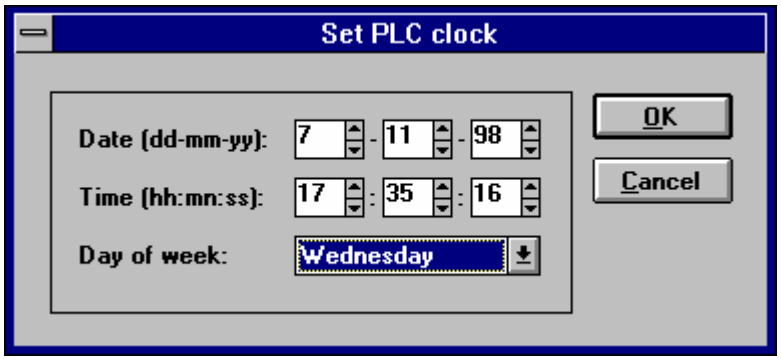
Sending system constants

The “**Control / Download**” command sends to the central unit the system constant according your choice.
Once the sending is ended, if you click on refresh, you can verify the central unit contains the system constant similar to your choice.

Uploading system constants

The purpose of that choice is to register on PC the values that are stored on central unit .
The “**Control / Download**” command copies the content of the column “Central unit” into the column “Your choice”.

Setting PLC clock



The “**Control / Set PLC clock**” command changes the central unit time (available for serie 30, serie 50 and serie 90).

Setting the password protection

The “**Control / Password protection**” command changes the password stored on central unit (available for serie 40, serie 50 and serie 30)

Exit

The “**Control / Exit**” command saves the content of the column “Your choice” and exit the configuration tool

- **Showing ToolBar**

The “**View / Show ToolBar**” command displays or removes the toolbar inside the configuration tool.

- **Showing title bar**

The “**View / Show title bar**” command displays or removes the listbox title bar

- **Showing status bar**

The “**View / Show status bar**” command displays or removes the status bar, at the bottom of the window.

- **Refreshing the constants**

The “**View / Reads**” command reads again the system constants on the central unit, and display the result by refreshing the column “Central unit”.

Refer to the technical description of each central unit for more details.

- **Automatic launch of editors in debug mode**

The following commands have to be set for a direct use of the debug mode

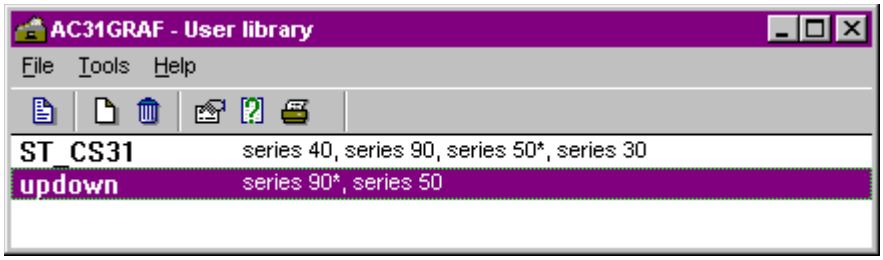
- To start one project in debug mode
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME
-To start one project in debug mode with GRAF1 graphic
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME -G=GRAF1
  -To start one project in debug mode with GRAF1, GRAF2 graphics
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME -G=GRAF1 -G=GRAF2
-To start one project in debug mode with LIST1 list of variables
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME -L=LIST1
-To start one project in debug mode with GRAF1, GRAF2 graphics, and LIST1 list of variables
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME -G=GRAF1 -G=GRAF2 -L=LIST1
-To start one project in debug mode with TIME1 chronogram
  C:\AC31GRAF\EXE\CORIDA7.EXE -D=PROJECT_NAME -T=TIME1

7 User's library

7.1 User's function

A user's function is developed by the user for a specific and repetitive application. The user's library enables the creation and storage of user's functions.
The "**File / Library**" command from the "**project management**" window runs the library tool.

- The code of a user function is written in FBD/LD (Function Bloc Diagram / Ladder Diagram) and duplicated on each call.
- A user function is created for one CPU (series 30, series 40, series 50, series 90, or controller).



Creating a function

The "**File / New**" command allows to create a new function (name, language and CPU used)

Editing the source

The "**File / Open**" command allows to edit the source file of the function.

Renaming a function

The "**File / Rename**" command allows to change the name of a function.

Deleting a function

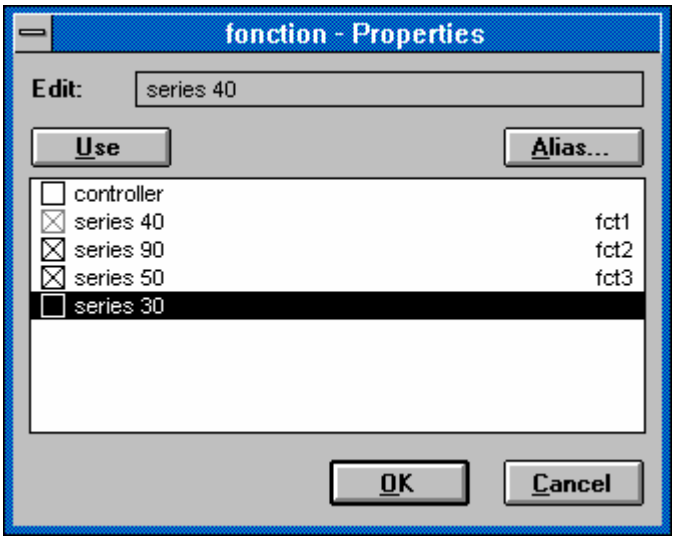
The "**File / Delete**" command allows to delete a function from the list.

Showing properties

The "**File / Properties**" command shows the availability for the other CPUs (maybe using another name).

Creating an alias

With the "**File / Properties/Alias**" command, while entering the properties of a function, you can define an alias for the function when used in a project. Thus a function created under a name can appear under another name in the function list when editing a project. Different aliases may be defined for each type of PLC.



A particular use of aliases is the capability to define several functions with different source code, and thus different names, and to provide all of them under the same name, so that the end user can use a generic service, without any regard on the coding differences due to various PLC specific features.

Editing a technical note

The "**Tools / Technical note**" command allows to edit the AC31GRAF text editor line technical note.

Importing or exporting a function

The "**Tools / Import and Export**" command allows to retrieve or set a user library on a floppy disk (everything is copied to one file).

7.2 Variables for a user's function

A user's function is programmed with specific variables. The type P, PW, PD are used as input parameters and Q, QW, QD as output parameters. There is no address affected to them but these types are numbering from 0 to N, furthermore, they must contain a symbol.
The internal variables such as M and K have only a name (symbol) and no address.
The direct global variables have only an address, but are not recommended.
Warning: Labels similar to an operator are not allowed (for example: LD, AND, OR etc...)

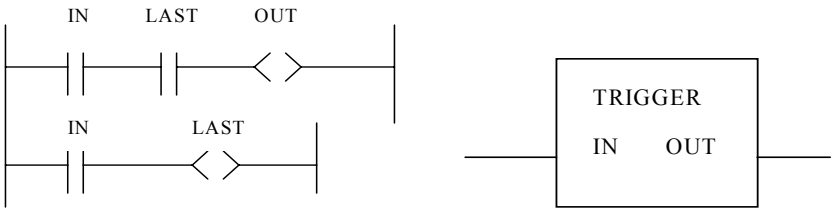
7.3 Compiling a user's function

A function must be compiled before to be used in a project.
Never forget to archive a user library on the disk, and the fact that the debug is not available.
A user's function is automatically inserted into the standard AC31GRAF library.

Below is an example of a user's function :

TRIGGER

Source :



Declaration :
P0 IN
Q0 OUT
M LAST

7.4 User library access and rights control

You can choose to apply up to three levels of access and rights control for users of the user library:

- Level 1 allowing users to access the library editor
- Level 2 allowing users to access technical information on functions as well as view their source code (in read-only mode)
- Level 3 allowing users to access technical information on functions, edit the source code of user functions, and create or delete user functions

Note: The level 1 password is independent from the level 2 and level 3 passwords. You can only set a level 2 password when a level 3 password is set.

For all three levels of access and rights control, when a user attempts to access the user library, a dialog box prompts them to enter a password.

You set up access and rights control for the user library from within the library window by choosing the **Tools/Set Password** option displaying the Password setting editor. In the Password setting editor, for each access and rights control level to implement, you need to provide a password and password confirmation.

B Language reference

B LANGUAGE REFERENCE

1 Project architecture B-3
1.1 Programs B-3
1.2 Modular architecture B-3
1.3 No-modular architecture B-5
2 Variables B-6
2.1 Variables are sorted according to their type. B-6
2.2 Variables are sorted according to their range B-6
3 FBD language B-7
3.1 FBD diagram main format B-7
3.2 Jumps and labels B-8
3.3 Boolean negation B-9
3.4 Calling function or function blocks from the FBD B-9
4 LD and Quick LD languages B-10
4.1 Power rails and connection lines B-10
4.2 Multiple connection B-11
4.3 Basic LD contacts and coils B-12
4.3.1 Direct contact B-12
4.3.2 Inverted contact B-13
4.3.3 Contact with rising edge detection B-13
4.3.4 Contact with falling edge detection B-14
4.3.5 Direct coil B-14
4.3.6 Inverted coil B-15
4.3.7 SET coil B-15
4.3.8 RESET coil B-16
4.3.9 Coil with rising edge detection B-16
4.4 Jumps and labels B-18
4.5 Blocks in LD B-18
4.5.1 The "EN" input B-19
4.5.2 The "ENO" output B-19
4.5.3 Using both "EN" and "ENO" B-19
5 SFC language B-20
5.1 SFC chart main format B-20
5.2 SFC basic components B-20
5.2.1 Steps and initial steps B-20
5.2.2 Transitions B-21
5.2.3 Oriented links B-21
5.2.4 Jump to a step B-22
5.3 Divergences and convergences B-22
5.3.1 Single divergences B-23
5.3.2 Double divergences B-24
5.4 Macro steps B-25
5.5 Actions within the steps B-26
5.5.1 Boolean actions B-26
5.5.2 Pulse actions B-27
5.5.3 Non-stored actions B-27
5.5.4 Calling function and function blocks from an action B-28
5.5.5 IL convention B-28

5.5.6 Activation duration of a step B-29
5.6 Conditions attached to transitions B-29
5.6.1 LD convention B-30
5.6.2 IL convention B-30
5.7 SFC dynamic rules B-30
6 IL language B-32
6.1 IL main syntax B-32
6.1.1 Labels B-32
6.1.2 Operator modifiers B-32
6.2 IL operators B-33
6.2.1 LD operator B-34
6.2.2 ST operator B-34
6.2.3 S operator B-34
6.2.4 R operator B-35
6.2.5 JMP operator B-35
6.2.6 Calling sub-programs and interruptions B-36
6.2.7 Calling function blocks: CAL, !BA 0 operator B-36
6.3 Main differences between IEC IL and ABB IL B-37

1 Project architecture

An AC31GRAF project is divided into several programming units called **programs**. The programs of a project are linked together in a tree-like architecture. Programs can be described using any of **FBD**, **LD, SFC, Quick LD** or **IL** graphic or litteral languages.

1.1 Programs

A **program** is a logical programming unit, that describes operations between variables of the process.

There is three kinds of programs :
- main programs
- subroutines
- interruptions

The main programs are executed according to the order defined by the project architecture.
The subroutines can be called from each other programs of the project.
The call of subroutines and interruption programs can be done by all editors

Programs are linked together in a hierarchy tree. Programs placed on the top of the hierarchy are activated by the system. Sub-routines (lower level of the hierarchy) are activated by their father. A program can be described with any of the available graphic or literal following languages:

Sequential Function Chart (SFC) for high level programming
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Quick Ladder Diagram (QLD) for boolean operations with function block insertion
Instruction List (IL) for low level operations

The same program can be written with several languages. LD and FBD are combined in one diagram.

1.2 Modular architecture

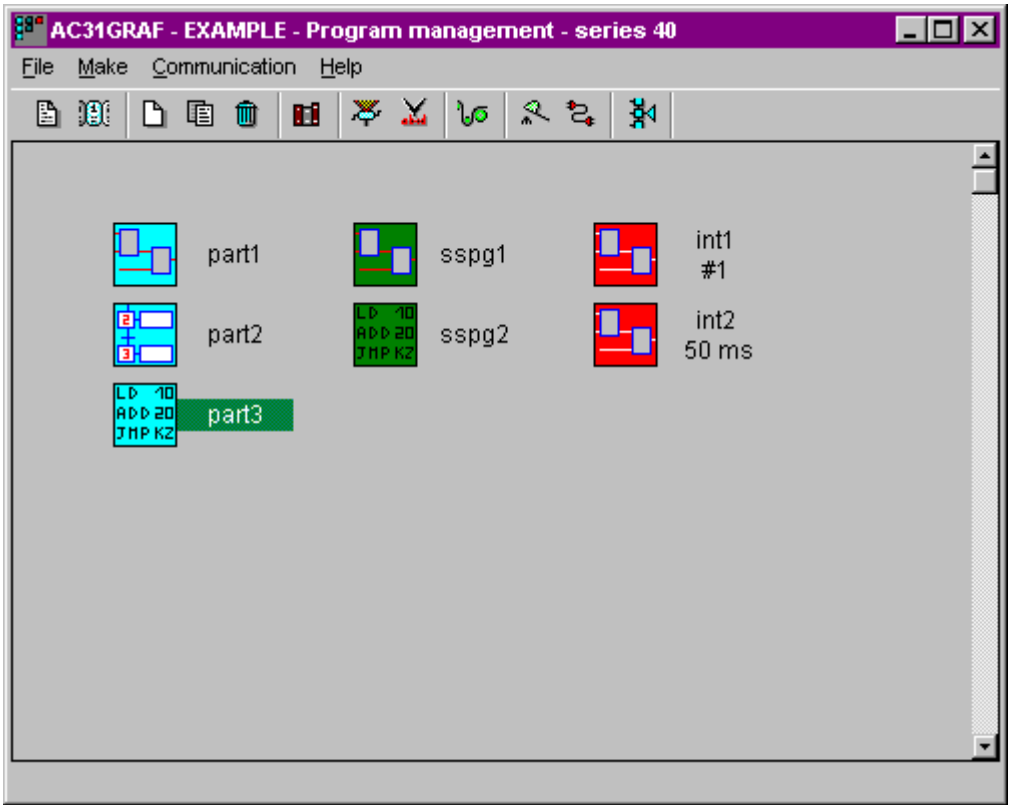
A project with a modular architecture contains more than one program.
It could be also composed of several programs , subroutines and interruptions program. A language is affected to each of them.

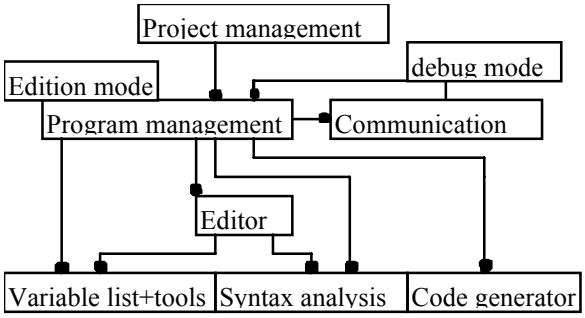
The **open** command of the project manager allows to edit the program manager of the selected project.

The edition of the programs of a modular project is realized from the program manager.

If a modular project contains only one program, it becomes a no-modular project.

Here is the diagram for the call of the main tools in a modular project :





Inside the program management, you can create subroutines, interruptions or other programs.

Each of them are represented by an icon:
- a blue icon for the programs
- a green icon for the subroutines
- a red icon for the interruptions

The icons representing a program are on the left of the client area of the program management, the subroutines in the middle and the interruptions on the right.

A project can contained only three interruption programs : two hard interruptions, (#1 and #2) and a soft one.
The icons representing the interruption programs allow, thanks to their name, to make the difference between the hard interruption, (icon name + #1 or #2) and the soft interruption (icon name + value of the cycle).

1.3 No-modular architecture

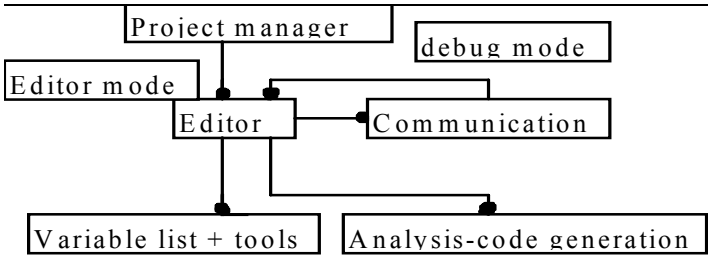
A project with a no-modular architecture contains only one program.
It does not allow to be composed of several programs and subroutines. A language is affected to the project.

The **open** command of the project manager allows to run the edition of the program which composes the project.

A no-modular project can be set modular by using the **modularize** command. This function of the files menu runs the program manager and allows the creation of other programs or subroutines.

In this case, the program manager contains only one icon which is representing the program of the project.
By default, its name is «main».

Here is the diagram for the call of the main tools in a no-modular project :



2 Variables

2.1 Variables are sorted according to their type.

These are basic types of variables :

BOOLEAN ..... true/false binary values
WORD..... analog values
DOUBLEWORD..... double word values
CONSTANT ..... binary, analog, double word values
TEXT..... string characters

Binary variable type	I	Binary input
	O	Binary output
	S	Binary step
	K	Binary constant
	M	Binary flag
Word variable type	IW	Analog input
	OW	Analog output
	KW	Analog constant
	MW	Analog flag
Double word variable type	KD	Double word constant
	MD	Double word flag
Text variable	TXT	Input string
Direct constant	#	Value in decimal
	#H	Value in hexadecimal

2.2 Variables are sorted according to their range

GLOBAL..... the variable can be used by all programs of a project
LOCAL..... the variable can be used by only one program

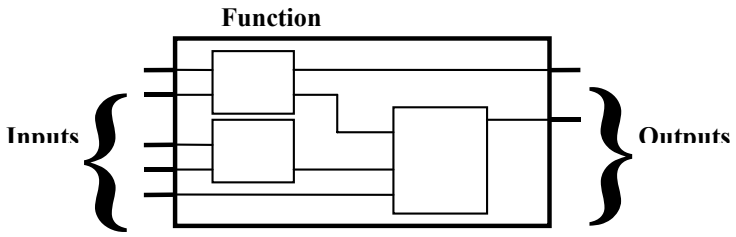
Two variables can have the same symbol, but they have to be local in two different files.
A global variable can not have the same symbol as a local variable file.
A variable defined without a symbol is global.

3 FBD language

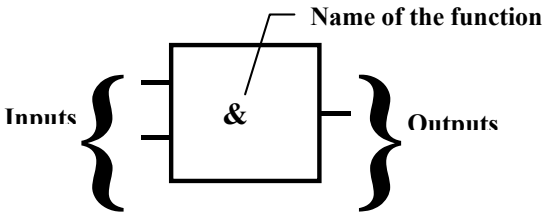
The **Functional Block Diagram** (FBD) is a graphic language. It allows the programmer to build complex procedures by taking existing **functions** from the AC31GRAF library and **wiring** them together in the graphic diagram area.

3.1 FBD diagram main format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of **elementary function blocks**. Input and output variables are connected to blocks by **connection lines**. An output of a function block may also be connected to an input of another block.



An entire function operated by an FBD program is built with standard **elementary** function blocks from the AC31GRAF library. Each function block has a fixed number of **input connection points** and a fixed number of **output connection points**. A function block is represented by a single **rectangle**. The inputs are connected on its **left** border. The outputs are connected on its **right** border. An elementary function block performs a single **function** between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well defined **type**.



Input variables of an FBD program must be connected to input connection points of function blocks. The type of each variable must be the same as the type expected for the associated input. An Input for FBD diagram can be a **constant** expression, any **internal** or **input** variable, or an **output** variable.

Output variables of an FBD program must be connected to output connection points of function blocks. The type of each variable must be the same as the type expected for the associated block output. An Output for FBD diagram can be any **internal** or **output** variable, or the name of the program (for **sub-programs** only). When an

output is the name of the currently edited sub-program, it represents the assignment of the return value for the sub-program (returned to the calling program).
Input and output variables, inputs and outputs of the function blocks are wired together with **connection lines**. Single lines may be used to **connect** two logical points of the diagram:

- An input variable and an input of a function block
- An output of a function block and an input of another block
- An output of a function block and an output variable

The connection is **oriented**, meaning that the line carries associated data from the left extremity to the right extremity. The left and right extremities of the connection line must be of the **same type**.

Multiple right connection can be used to broadcast an information from its left extremity to each of its right extremities. All the extremities of the connection must be of the same type.

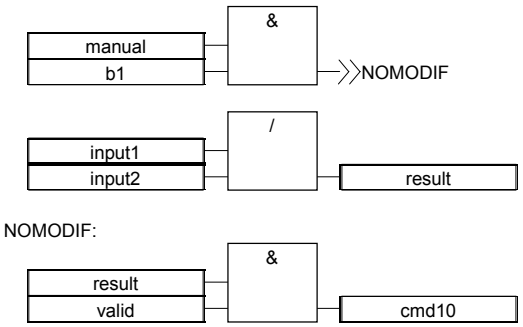
3.2 Jumps and labels

Labels and jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notations are used:

>>LAB ..... jump to a label (label name is "LAB")
LAB: ..... definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the boolean state **TRUE**, the execution of the program directly jumps after the corresponding label symbol.

(* Example of a FBD program using labels and jumps *)



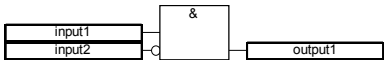

```
(* IL Equivalence: *)
        ld      manual
        and     b1
        jmpc    NOMODIF
        ld      input1
        or      input2
        st      result
NOMODIF:ld      result
```

and valid
st cmd10

3.3 Boolean negation

A single connection line with its right extremity connected to an input of a function block can be terminated by a **boolean negation**. The negation is represented by a small circle. When a boolean negation is used, the left and right extremities of the connection line must have the **BOOLEAN** type.

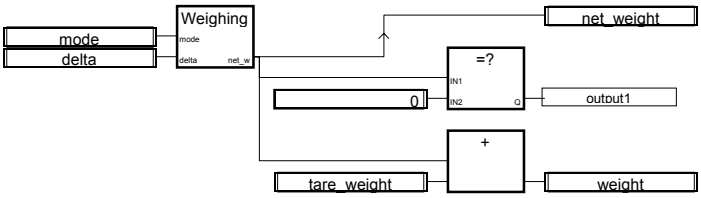
(* Example of a FBD program using boolean negation *)



3.4 Calling function or function blocks from the FBD

The FBD language enables the calling of sub-programs, functions or function blocks. A sub-program, or function or function block is represented by a function box. The name written in the box is the name of the sub-program or function or function blocks.
A function block can have more than one outputs.

(* Example of a FBD program using an user function block (Weighing) *)



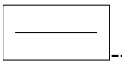
4 LD and Quick LD languages

Ladder Diagram (LD) is a graphic representation of boolean equations, combining **contacts** (input arguments) with **coils** (output results). The LD language enables the description of tests and modifications of **boolean** data by placing **graphic symbols** into the program chart. LD graphic symbols are organized within the chart exactly as an electric contact diagram. LD diagrams are connected on the left side and on the right side to vertical **power rails**.

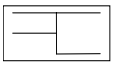
These are basic graphic components of an LD diagram:

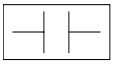

---------- Left vertical power rail

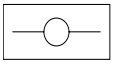

---------- Right vertical power rail


---------- Horizontal connection line


---------- Vertical connection line

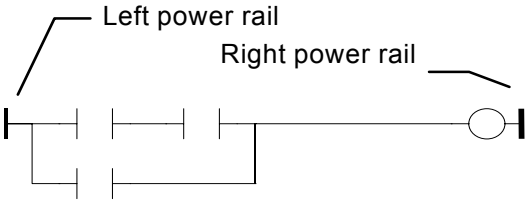

---------- Multiple connection lines (all connected together)


---------- Contact associated with a variable

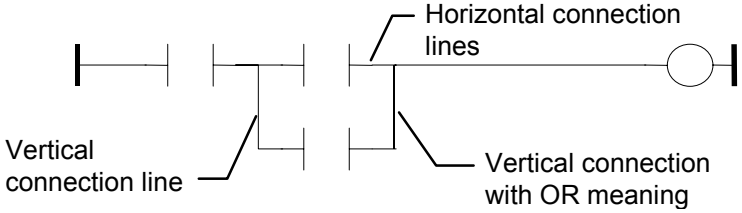

---------- Coil associated to an output or to an internal variable

4.1 Power rails and connection lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.



LD diagram graphic symbols are connected to power rails or to other symbols by **connection lines**. Connection lines are horizontal or vertical.



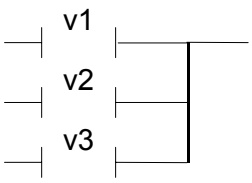
Each line segment has a boolean state **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

4.2 Multiple connection

The boolean state given to a single horizontal connection line is the same on the left and on the right extremities of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The boolean state of the extremities of a multiple connection follows logic rules.

A **multiple connection on the left** combines **more than one** horizontal lines connected on the **left** side of a vertical line, and **one** line connected on its **right** side. The boolean state of the right extremity is the **LOGICAL OR** between all the left extremities.

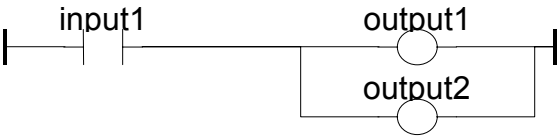
(* Example of multiple LEFT connection *)



(* right extremity state is (v1 OR v2 OR v3) *)

A **multiple connection on the right** combines **one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of the left extremity is propagated into each of the right extremities.

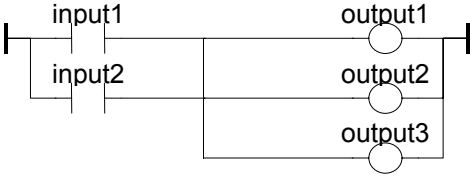
(* Example of multiple RIGHT connection *)



(* means: *)
output1 = input1
output2 = input1

A **multiple connection on the left and on the right** combines **more than one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of each of the right extremities is the **LOGICAL OR** between all the left extremities

(* Example of multiple LEFT and RIGHT connection *)



(* means: *)
output1 = input1 OR input2
output2 = input1 OR input2
output3 = input1 OR input2

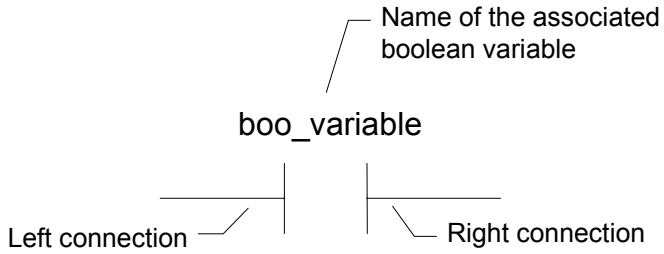
4.3 Basic LD contacts and coils

There are several symbols available for input contacts

There are several symbols available for output coils:

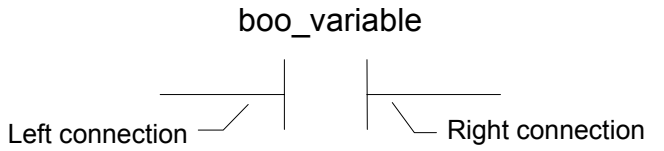
Direct coil
Inverted coil
SET coil
RESET coil
Coils with rising edge detection
Coils with falling edge detection

The name of the variable is written above any of these graphic symbols:



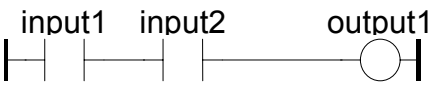
4.3.1 Direct contact

A direct contact enables a **boolean operation** between a **connection line** state and a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the value of the variable associated with the contact.

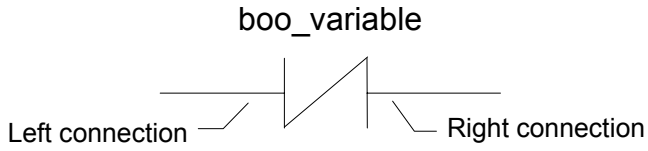
(* Example using DIRECT contacts *)



(* means: *)
output1 = input1 AND input2

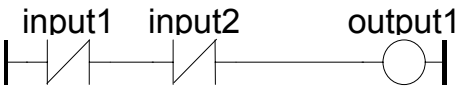
4.3.2 Inverted contact

An inverted contact enables a **boolean operation** between a **connection line** state and the boolean negation of a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the **boolean negation** of the value of the variable associated with the contact.

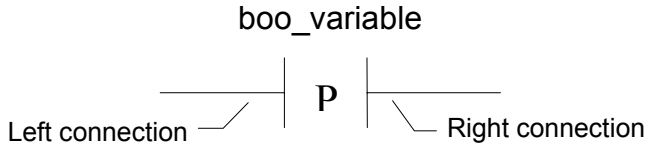
(* Example using INVERTED contacts *)



(* means: *)
output1 = NOT (input1) AND NOT (input2)

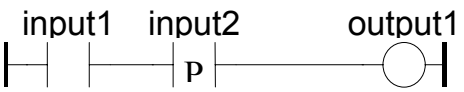
4.3.3 Contact with rising edge detection

This contact (positive) enables a **boolean operation** between a **connection line** state and the rising edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **rises** from FALSE to TRUE. It is reset to FALSE in all other cases.

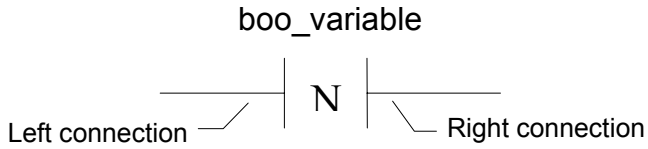
(* Example using RISING EDGE contacts *)



(* means: *)
output1 = input1 AND (input2 AND NOT (input2prev))
(* input2prev is the value of input2 at the previous cycle *)

4.3.4 Contact with falling edge detection

This contact (negative) enables a **boolean operation** between a **connection line** state and the falling edge of a boolean **variable**.



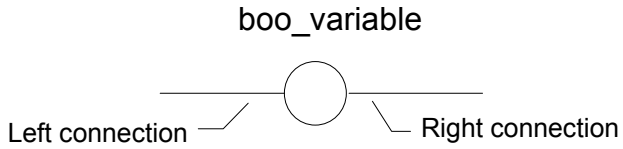
The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **falls** from TRUE to FALSE. It is reset to FALSE in all other cases.

(* Example using FALLING EDGE contacts *)

(* means: *)
output1 = input1 AND (NOT (input2) AND input2prev)
(* input2prev is the value of input2 at the previous cycle *)

4.3.5 Direct coil

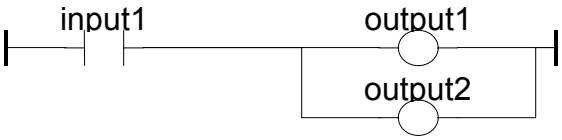
Direct coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is assigned with the boolean **state of the left connection**. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

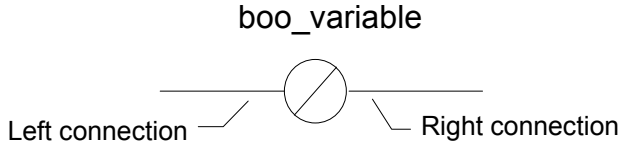
(* Example using DIRECT coils *)



(* means: *)
output1 = input1
output2 = input1

4.3.6 Inverted coil

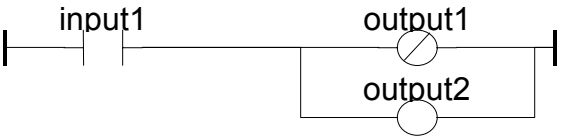
Inverted coils enable a **boolean output** according to the boolean **negation** of a **connection line** state.



The associated variable is assigned with the boolean **negation** of the **state of the left connection**. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

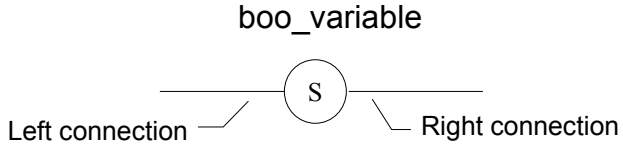
(* Example using INVERTED coils *)



(* means: *)
output1 = NOT (input1)
output2 = input1

4.3.7 SET coil

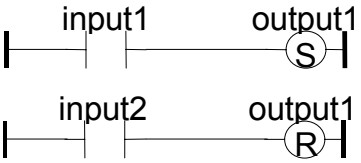
"Set" coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is **SET TO TRUE** when the boolean **state of the left connection** becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

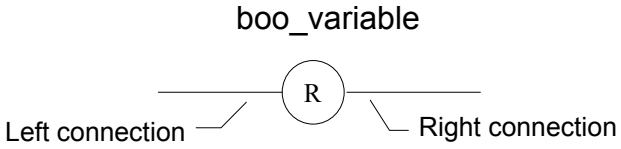
(* Example using "SET" and "RESET" coils *)



(* means: *)
IF input1 THEN output1 = TRUE
IF input2 THEN output1 = FALSE

4.3.8 RESET coil

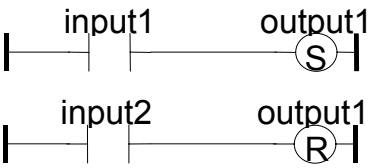
"Reset" coils enable **boolean output** of a **connection line** boolean state.



The associated variable is **RESET TO FALSE** when the boolean **state of the left connection** becomes **TRUE**. The output variable keeps this value until an inverse order is made by a "SET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

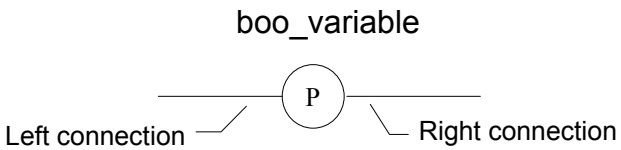
(* Example using "SET" and "RESET" coils *)



(* means: *)
IF input1 THEN output1 = TRUE
IF input2 THEN output1 = FALSE

4.3.9 Coil with rising edge detection

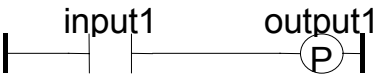
"Positive" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



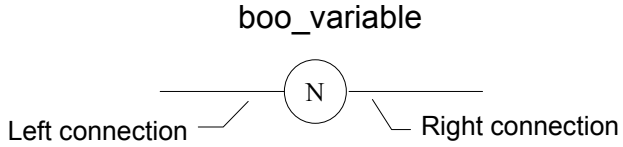
The associated variable is set to **TRUE** when the boolean **state of the left connection** rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)



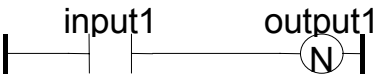
(* means: *)
IF (input1 and NOT(input1prev)) THEN output1 = TRUE
ELSE output1 = FALSE
(* input1prev is the value of input1 at the previous cycle *)
Coil with falling edge detection
"Negative" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)



(* means: *)
IF (NOT(input1) and input1prev) THEN output1 = TRUE
ELSE output1 = FALSE
(* input1prev is the value of input1 at the previous cycle *)

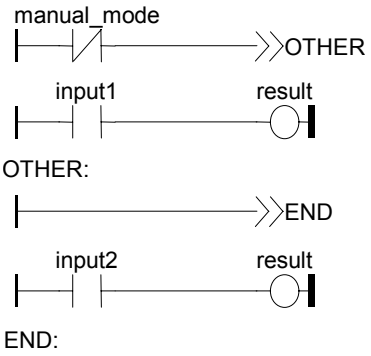
4.4 Jumps and labels

Labels, conditional and unconditional JUMPS symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB ..... jump to label named "LAB"
LAB: ..... definition of the label named "LAB"

If the **connection on the left** of the jump symbol has the **TRUE** boolean state, the program execution is driven after the label symbol.

(* Example using JUMP and LABEL symbols *)



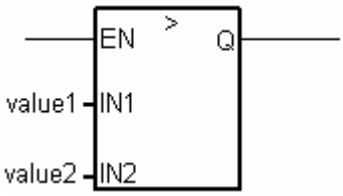

```
(* IL Equivalence: *)
        ldn     manual_mode
        jmpc    other
        ld      input1
        st      result
        jmp     END
OTHER:  ld      input2
        st      result
END:  (* end of program *)
```

4.5 Blocks in LD

Using the Quick LD editor, you connect function boxes to boolean lines. A function can actually be an operator, a function block or a function. As all blocks do not have always a boolean input and/or a boolean output, inserting blocks in an LD diagram leads to the addition of new parameters EN, ENO to the block interface. The EN, ENO parameters are not added if you use the FBD/LD editor as you can connect the variable with the required type.
A function block with duplicated inputs can not be used in QUICK LADDER.It has to be used in the other language as FBD

4.5.1 The "EN" input

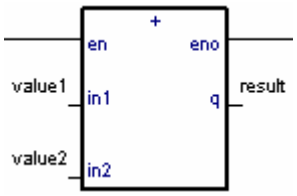
On some operators, functions or function blocks, the first input does not have boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE. Below is the example of a comparison operator, and the equivalent expression



IF en is true then the function is valided
else Q = false

4.5.2 The "ENO" output

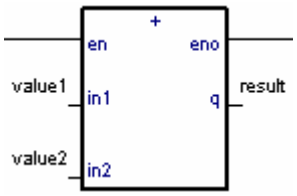
On some operators, functions or function blocks, the first output does not have boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "**ENO**". The **ENO** output always takes the same state as the first input of the block. On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in pseudo language:



IF en is true , the addition is valided and
scanned else not
en = eno
continue rung with eno state...

4.5.3 Using both "EN" and "ENO"

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in pseudo language



IF en is true , the addition is valided and
scanned else not
en = eno
continue rung with eno state...

5 SFC language

Sequential Function Chart (SFC) is a **graphic** language used to describe **sequential operations**. The process is represented as a set of well defined **steps**, linked by **transitions**. A **boolean condition** is attached to each transition. **Actions** within the steps are detailed by using IL language.

5.1 SFC chart main format

An SFC program is a graphic set of **steps** and **transitions**, linked together by **oriented links**. Multiple connection links are used to represent divergences and convergences. Some parts of the complete program may be separated and represented in the main chart by a single symbol, called **macro steps**. The basic **graphic rules** of the SFC are:

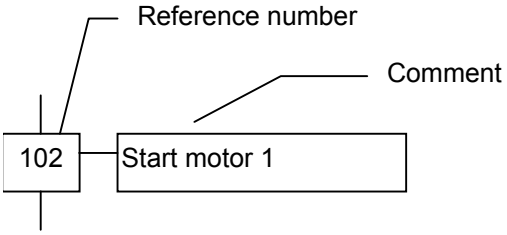
- A step cannot be followed by another step
- A transition cannot be followed by another transition

5.2 SFC basic components

The basic components (graphic symbols) of the SFC language are: steps and initial steps, transitions, oriented links, and jumps to a step.

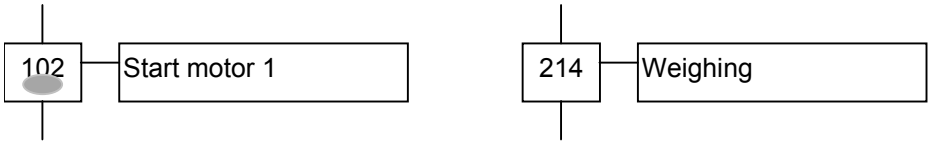
5.2.1 Steps and initial steps

A step is represented by a single **square**. Each step is **referenced** by a number, written in the step square symbol. A main description of the step is written in a rectangle linked to the step symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the step:



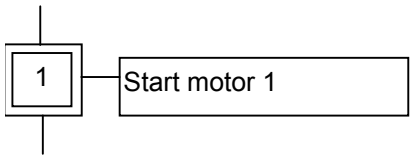
At run time, a **token** indicates that the step is **active**:

Active step: Inactive step:



The **initial situation** of an SFC program is expressed with **initial steps**. An initial step has a **double bordered** graphic symbol. A token is automatically placed in each initial step when the program is started.

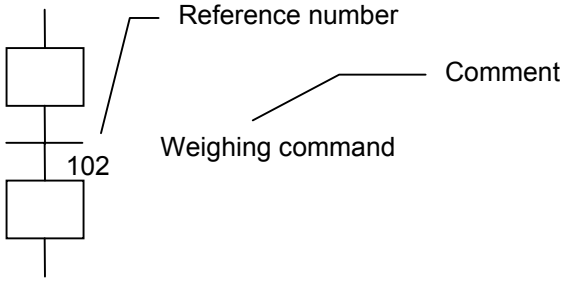
Initial step:



An SFC program must contain **at least one** initial step.

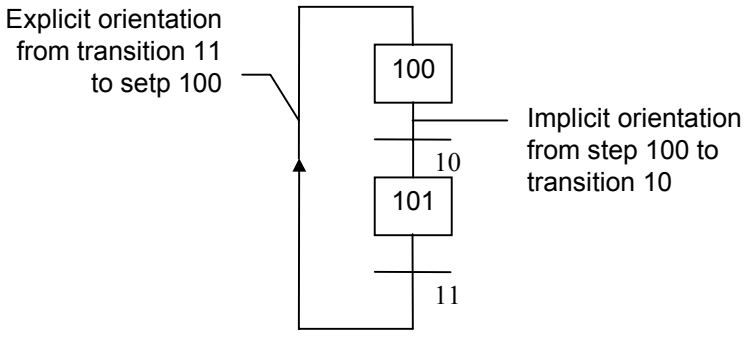
5.2.2 Transitions

Transitions are represented by a small horizontal bar that crosses the connection link. Each transition is **referenced** by a number, written next to the transition symbol. A main description of the transition is written on the right side of the transition symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the transition:



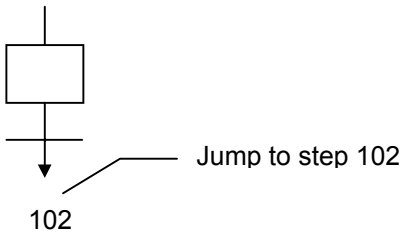
5.2.3 Oriented links

Single lines are used to link steps and transitions. These are oriented links. When the orientation is not explicitly given, the link is oriented from the top to the bottom.

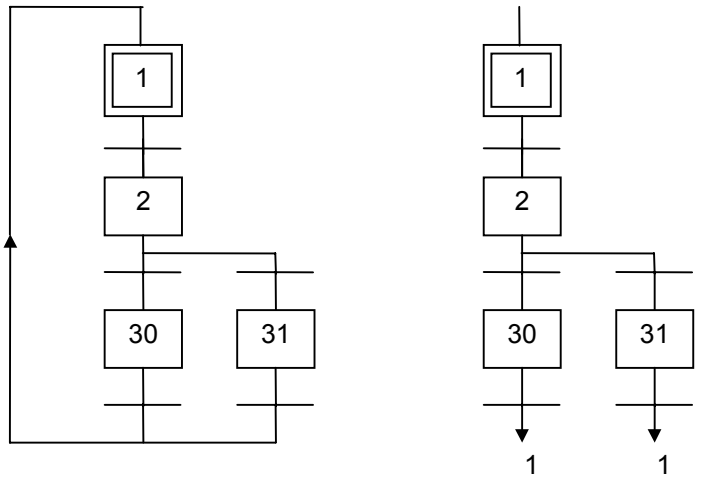


5.2.4 Jump to a step

Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the number of the destination step:



A jump symbol cannot be used to represent a link from a step to a transition. Example of jumps - the following charts are equivalent:

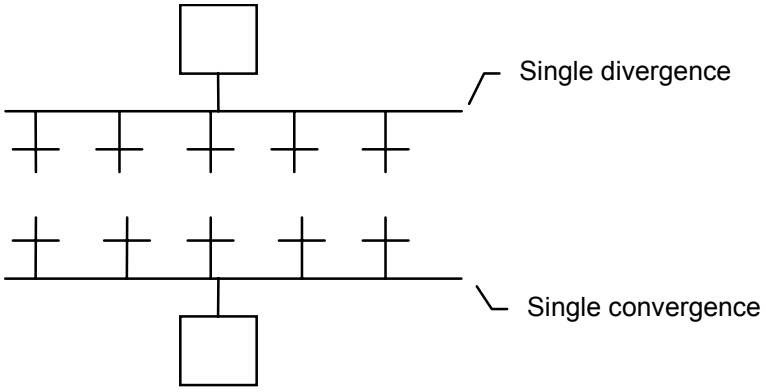


5.3 Divergences and convergences

Divergences are **multiple connection links** from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

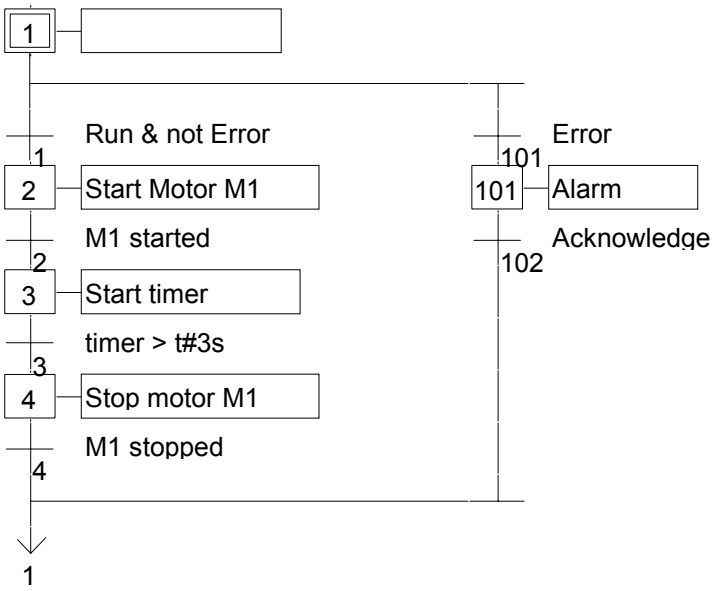
5.3.1 Single divergences

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence. Single divergences and convergences are represented by single horizontal lines.



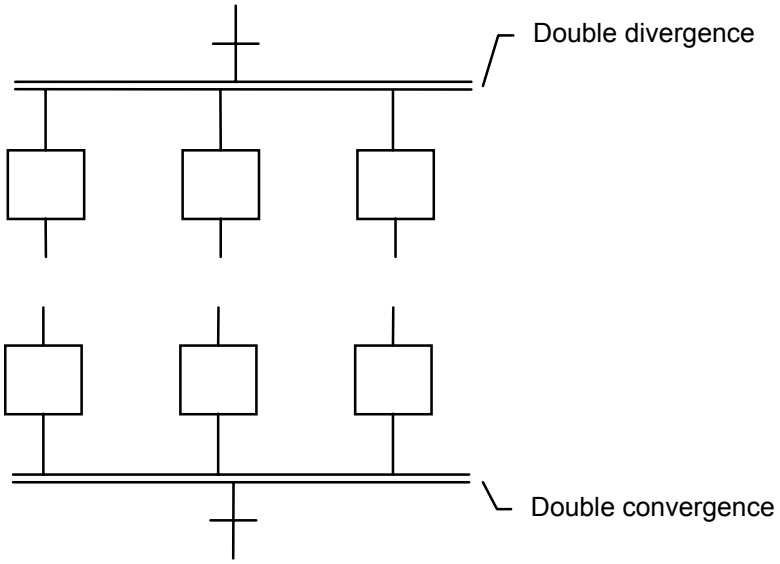
Warning: The conditions attached to the different transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of the transitions to ensure that only one token progresses in one branch of the divergence at run time. Below is an example of single divergence and convergence:

(* SFC program with single divergence and convergence *)



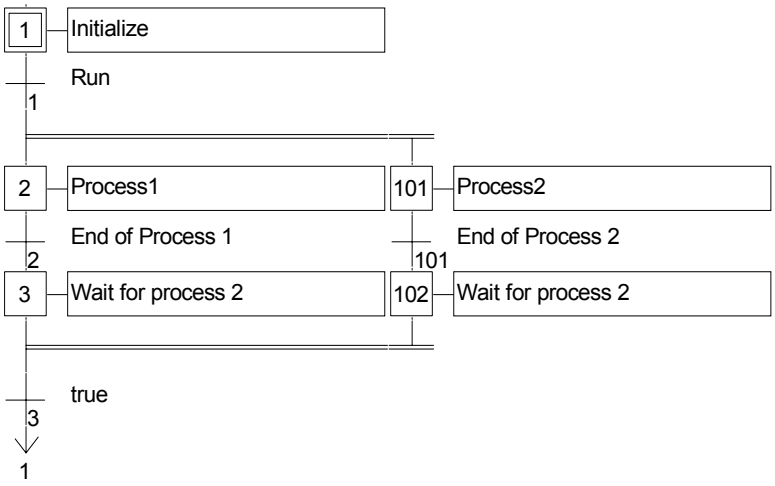
5.3.2 Double divergences

A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence. Double divergences and convergences are represented by double horizontal lines.



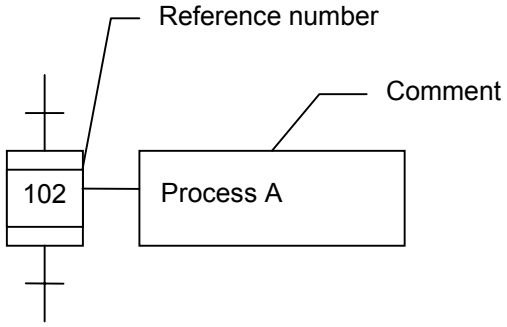
Example of double divergence and convergence:

(* SFC program with double divergence and convergence *)



5.4 Macro steps

A macro step is a unique representation of a unique group of steps and transitions. The body of the macro step is described separately, elsewhere in the same SFC program. It appears as a single symbol in the main SFC chart. This is the symbol used for a macro step :

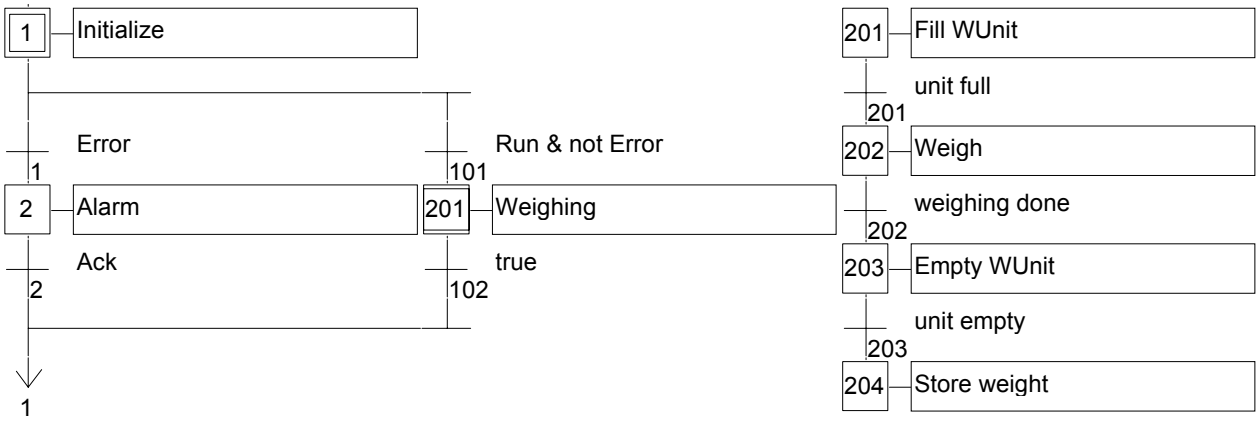


The reference number written in the macro step symbol is the reference number of the first step in the body of the macro step. The macro step body must begin with a **beginning step** and terminate with an **ending step**. The chart must be self-contained. A beginning step has no upper link (no backward transition). An ending step has no lower link (no forward transition). A macro step symbol may be put in the body of another macro step.

Warning: Because macro step is a **unique** set of steps and transitions, the same macro step cannot be used more than once in an SFC program.

Example of macro step:

(* Main chart *) **(* Body of the macro step *)**



5.5 Actions within the steps

The **level 2** of an SFC step is the detailed description of the **actions** executed **during the step activity**. This description is made by using **SFC literal features**, and other languages. The basic types of actions are:
Boolean actions
Pulse actions programmed in IL
None-stored actions programmed in IL

Several actions (with same or different types) can be described in the same step. The special features that enable the use of any of the other languages are:

Calling function and function blocks from an action
IL convention

5.5.1 Boolean actions

Boolean actions assign a boolean variable with the activity of the step. The boolean variable can be an output or an internal. It is assigned each time the step activity starts or stops. This is the syntax of the basic boolean actions:

var (N); .....assigns the step activity signal to the variable
var; .....same effect (N attribute is optional)
/ var; .....assigns the negation of the step activity signal to the variable

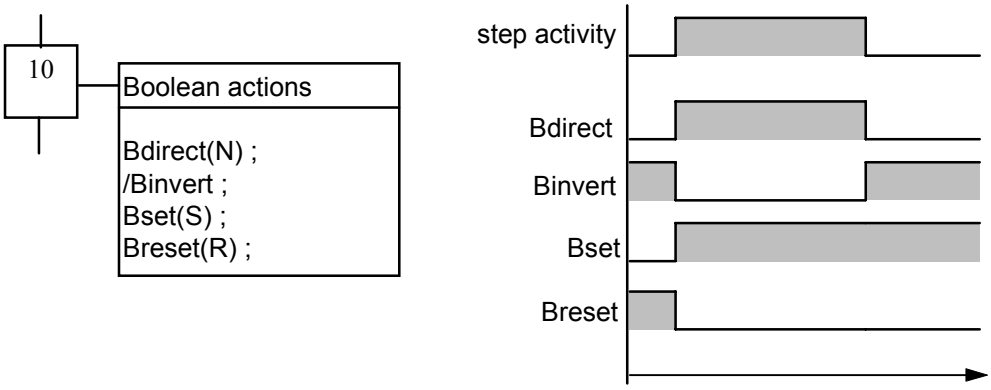
example :
%O62.00 (N); assigns the step activity signal to the output %O62.00
%O62.00;.....same effect (N attribute is optional)
/ %M00.00; ..assigns the negation of the step activity signal to the variable %M00.00

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

var (S); .....sets the variable to TRUE when the step activity signal becomes TRUE
var (R); .....resets the variable to FALSE when the step activity signal becomes TRUE

example
%O62.00 (S); sets %O62.00 to TRUE when the step activity signal becomes TRUE
%M00.00 (R); resets %M00.00 to FALSE when the step activity signal becomes TRUE

The boolean variable must be an OUTPUT or an INTERNAL. The following SFC programming leads to the following behavior:

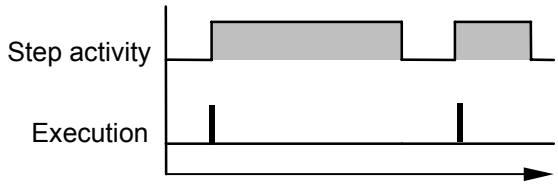


5.5.2 Pulse actions

A pulse action is a list of IL instructions, which are executed only **once** at the **activation** of the step. Instructions are written according to the following SFC syntax:

ACTION (P) :
(* IL statements *)
END_ACTION ;

The following shows the results of a pulse action:

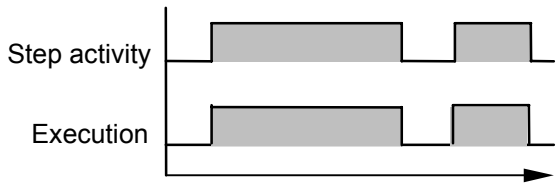


5.5.3 Non-stored actions

A non-stored (normal) action is a list of IL instructions which are executed **at each cycle** during the whole **active** period of the step. Instructions are written according to the following SFC syntax:

ACTION (N) :
(* IL statements *)
END_ACTION ;

The following is the results of a non-stored action:



5.5.4 Calling function and function blocks from an action

Subroutines, or function blocks (written in IL, LD or FBD language) can be directly called from an SFC action block, based on the following syntax :

For subroutines

```
ACTION (P) :
        CAL_FB(subroutineX, bit)
END_ACTION;
```

or

```
ACTION (N) :
        CAL_FB(subroutineX, bit)
END_ACTION;
```

For function blocks in IL

```
ACTION (P) :
        !BA 0
        BlocX
        Param1
        Param2
        ...
END_ACTION;
```

or

```
ACTION (N) :
        !BA 0
        BlocX
        Param1
        Param2
        ...
END_ACTION;
```

Warning: The instructions written between ACTION(P) or (N) and END_ACTION are scanned and executed only if the step is activated.

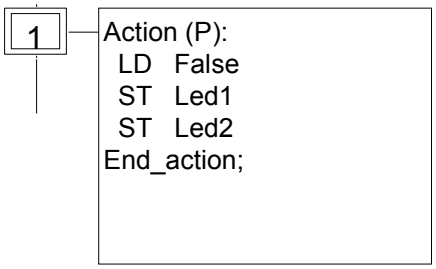
5.5.5 IL convention

Instruction List (IL) programming may be directly entered in an SFC action block, based on the following syntax:

```
ACTION (P) :        (* or N *)
   <instruction>
   <instruction>
   ....
END_ACTION;
```

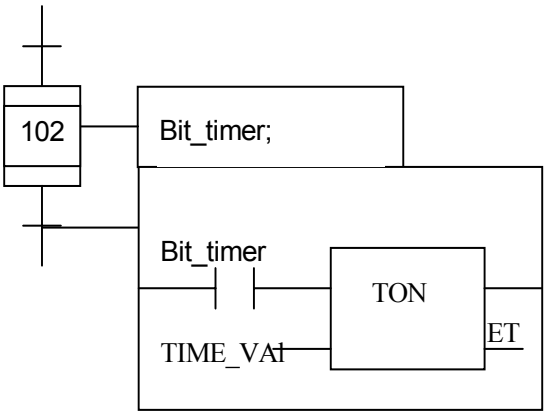
Below is an example of an IL program in an action block:

(* SFC program with an IL sequence in an action block *)



5.5.6 Activation duration of a step

An action can be actived only for a time.
The timer function has to be used in an transition.
It cannot be used in a action (N) because the timer function cannot be reset when the step is deactivated.
The start timer bit has to be written in the action .
Check the following example:



The bit-timer is set to 1 by the step 102 and in the same time the timer TON is started.
After the time value the next step is activated , the bit-timer is reset and the timer TON is stopped.

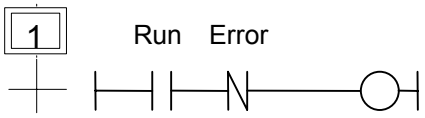
5.6 Conditions attached to transitions

At each transition, a **boolean expression** is attached that conditions the clearing of the transition. The condition is usually expressed with LD language. This is the **Level 2** of the transition. Other structures may, however, be used:

Warning: When no expression is attached to the transition, the default condition is **TRUE**.

5.6.1 LD convention

The **Ladder Diagram** (LD) language can be used to describe the **condition** attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transitions value.
Below is an example of LD programming for transitions:



5.6.2 IL convention

Instruction List (IL) programming may be directly used to describe an SFC transition.

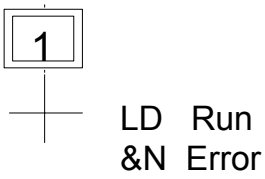
```
<instruction>
<instruction>
....
```

The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the transition:

current result = 0 ➔ condition is **FALSE**
current result <> 0 ➔ condition is **TRUE**

Below is an example of IL programming for transitions:

(* SFC program with an IL program for transitions *)



5.7 SFC dynamic rules

The **five** dynamic rules of the SFC language are:

Initial situation
The initial situation is characterized by the **initial steps** which are, by definition, in the active state at the beginning of the operation. **At least one** initial step must be present in each SFC program.

Clearing of a transition
A transition is either **enabled** or **disabled**. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are **active**, otherwise it is disabled. A transition cannot be **cleared** unless:
- it is enabled, and
- the associated transition condition is true.

Changing of state of active steps
The clearing of a transition simultaneously leads to the active state of the immediately following steps and to the inactive state of the immediately preceding steps.

Simultaneous clearing of transitions
Double lines may be used to indicate transitions which have to be cleared simultaneously. If such transitions are shown separately, the activity state of preceding steps (GSnnn.x) can be used to express their conditions.

Simultaneous activation and deactivation of a step
If, during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

6 IL language

Instruction List, or **IL** is a low level language. It is highly effective for smaller applications or for optimizing parts of an application. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

6.1 IL main syntax

An IL program is a list of **instructions**. Each instruction must begin on a new line, and must contain an **operator**, completed with optional **modifiers** and, if necessary, for the specific operation, one or more **operands**, separated with commas (**','**). A **label** followed by a colon (**':'**) may precede the instruction. If a **comment** is attached to the instruction, it must be the last component of the line. Comments always begin with **'(*'** and ends with **'*)'**. Empty lines may be entered between instructions. Comments may be put on empty lines.

Below are examples of instruction lines:

```
Label  Operator     Operand       Comments

Start: LD      BUTTON1           (* push button *)
       ANDN    %I62.02           (* command is not forbidden *)
       ST      START-MOTOR       (* start motor *)
```

6.1.1 Labels

A **label** followed by a colon (**':'**) may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Naming labels must conform to the following rules:

- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters**, **digits** or **'-'** character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a variable.

6.1.2 Operator modifiers

The available operator modifiers are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

N boolean negation of the operand
C conditional operation

The **'N'** modifier indicates a boolean negation of the operand. For example, the instruction **ORN %I62.00** is interpreted as: **OR NOT %I62.00**

The **'C'** modifier indicates that the attached instruction must be executed only if the current result has the boolean value TRUE (different than 0 for non-boolean values). The **'C'** modifier can be combined with the **'N'** modifier to indicate that the instruction must be executed only if the current result has the boolean value FALSE (or 0 for non-boolean values).

6.2 IL operators

The following table summarizes the standard operators of the IL language:

Operator	*Modifiers*	*Operand*	*Description*
LD	N	Variable, constant	Loads operand
!	N	Variable, constant	Loads operand
ST	N	Variable	Stores current result
S		Variable	Sets to TRUE
R		Variable	Resets to FALSE
!BA 0		+ function block name ->	Calls a function block
CAL		FB instance name	Calls a function block
CAL_FB		Calls a subroutine	
VTASK		Calls an interruption	
JMP	C N	Label	Jumps to label
AND	N	BOO	boolean AND
&	N	BOO	boolean AND
OR	N	BOO	boolean OR
/	N	BOO	boolean OR
ADD		variable, constant	Addition
+		variable, constant	Addition
SUB		variable, constant	Subtraction
-		variable, constant	Substraction
MUL		variable, constant	Multiplication
*		variable, constant	Multiplication
DIV		variable, constant	Division
:		variable, constant	Division
GT		variable, constant	Test: >
>		variable, constant	Test: >
GE		variable, constant	Test: >=
>=		variable, constant	Test: >=
EQ		variable, constant	Test: =
=?		variable, constant	Test: =
LE		variable, constant	Test: <=
<=		variable, constant	Test: <=
LT		variable, constant	Test: <
<		variable, constant	Test: <
NE		variable, constant	Test: <>

<>	variable, constant	Test: <>

In the next section, only operators which are specific to the IL language are described, other standard operators can be found in the section "standard operators, function blocks and functions".

6.2.1 LD operator

Operation loads a value in the current result

Allowed modifiers N

Operand constant expression
internal, input or output variable

Example:

```
                        (* EXAMPLES OF LD OPERATIONS *)
LDex::      LD    false       (* result = FALSE boolean constant *)
            LD    true        (* result = TRUE boolean constant *)
            LD    boo_var1    (* result = boolean variable *)
            LD    ana_var1    (* result = analog variable *)
            LDN   boo_var2    (* result = NOT ( boolean variable ) *)
```

6.2.2 ST operator

Operation stores the current result in a variable
the current result is not modified by this operation

Allowed modifiers N

Operand internal or output variable

Example:

```
                    (* EXAMPLES OF ST OPERATIONS *)
STboo:      LD    false
            ST    boo_var1    (* boo_var1 = FALSE *)
            STN   boo_var2    (* boo_var2 = TRUE *)
STana:      LD    %IW3.3
            ST    ana_var1    (* ana_var1 = value of %IW3.3 *)
```

6.2.3 S operator

Operation: stores the boolean value TRUE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation.

Allowed modifiers: (none)

Operand: output or internal boolean variable

Example:

```
                (* EXAMPLES OF S OPERATIONS *)
SETex:    LD    true        (* current result = TRUE *)
          S     var1        (* var1 = TRUE *)
                            (* current result is not modified *)
          LD    false       (* current result = FALSE *)
          S     var1        (* nothing done var1 unchanged *)
```

6.2.4 R operator

Operation stores the boolean value FALSE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation.

Allowed modifiers (none)

Operand output or internal boolean variable

Example:

```
                (* EXAMPLES OF R OPERATIONS *)
RESETex:  LD    true        (* current result = TRUE *)
          R     var1        (* var1 = FALSE *)
                            (* current result is not modified *)
          ST    var2        (* var2 = TRUE *)
          LD    false       (* current result = FALSE *)
          R     var1        (* nothing done  var1 unchanged *)
```

6.2.5 JMP operator

Operation jumps to the specified label

Allowed modifiers C N

Operand label defined in the same IL program

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)      *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with      *)
(* the JMPC operator *)

JMPex:
          LD        selector    (* selector is 0 or 1 or 2 *)
          EQ        value=0     (* %KW1.0=0*)
          JMPC      test1       (* if selector = 0 then *)
          LD        true
          ST        bo0         (* bo0 = true *)
          JMP       JMPend      (* end of the program *)
```

```
test1:
            LD          selector
            EQ          value=1     (* %KW1.1=1 *)
            JMPC        test2       (* if selector = 1 then *)
            LD          true
            ST          bo1         (* bo1 = true *)
            JMP         JMPend      (* end of the program *)
test2:
            LD          true        (* last possibility *)
            ST          bo2         (* bo2 = true *)
JMPend:                             (* end of the IL program *)
```

6.2.6 Calling sub-programs and interruptions

A sub-program is called from the IL language, using its name as an operator.

Operation executes a sub-program

Allowed modifiers (none)

Operand The first calling parameter must be the name of the subroutine
The following one, separated by a coma, is the call bit of the subroutine.

Example:

```
CAL_FB(sub1,%I2.2)      (* call the subroutine named sub1 *)
```

An interruption is called from the IL language, using its name as an operator.

Operation executes an interruption

Allowed modifiers (none)

Operand The first calling parameter must be the name of the interruption
The following one, separated by a coma, is the call bit of the interruption.

Example:

```
VTASK(int, %I2.2)  (* call the hard interruption named int *)
```

6.2.7 Calling function blocks: CAL, !BA 0 operator

Operation calls a function block
CAL has to be used for blocks not extended.

Allowed modifiers None

Operand Name of the function block instance.
Output parameters are known if used.

Example1:

(* Calling function block TON*)

CAL TON (IN, PT, ET, Q)

is equivalent to :

```
!BA0
TON
IN
PT
ET
Q
```

example 2:

(* Calling function block CTUH *)

```
CAL
CTUH(#1,%I62.00,%I62.01,%MW2.0,%I62.02,%I62.03,%I62.04,%O62.00,%OW62.0
1,%OW62.02)
```

is equivalent to :

```
!BA 0
CTUH
#1
%I62.00
%I62.01
%MW2.0
%I62.02
%I62.03
%I62.04
%O62.00
%OW62.01
%OW62.02
```

6.3 Main differences between IEC IL and ABB IL

With IL ABB, an instruction list must be followed by an operator **ST**:

```
LD %I62.01
AND %I62.02
ST %M2.0
```

In order to use the accumulator %M2.0, we have to load it:

```
LD %I62.01
```

```
AND %I62.02
ST %M2.0
LD %M2.0
OR %M3.0
…
```

With IL IEC, there no need to load the accumulator (the current result), so we can write:

```
LD %I62.01
AND %I62.02
ST %M2.2
OR %I62.03
…
```

Call of subroutines, interruptions and function blocs:

Function blocs:
The following syntax is just used for IL ABB in order to call function blocs

```
!BA 0
CTUH
…
```

but it is possible to use the IL IEC as follow:

```
CAL CTUH(…)
```

Subroutines and interruption:
To call a subroutine:
CAL_FB
and an interruption:
VTASK

This syntax is specific to the IL ABB

C Function block description

C FUNCTION BLOCK DESCRIPTION

1 Libraries C-2
2 Basic operators/functions C-8
2.1 Binary functions C-8
2.2 Timer functions C-16
2.3 Counter functions C-33
2.4 Comparison functions, word C-39
2.5 Arithmetic functions, word C-48
2.6 Logical functions, word C-60
3 Program control functions C-68
4 CS31 functions C-84
5 Communication functions C-116
6 Regulation functions C-154
7 Format conversion functions C-174
8 Standard double word functions C-200
8.1 Comparison functions, double word C-200
8.2 Arithmetic functions, double word C-203
8.3 Logical functions, double word C-212
9 High order functions C-216
10 Memory access functions C-291
11 Special Functions C-317
12 Historical values C-323
12.1 Definition C-323
12.2 Historical value table C-323
13 Runtimes C-325
13.1 Definition C-325
13.2 Runtime table (time in µs) C-325

1 Libraries

Each central unit serie has its own library of functions. The libraries are specified in the following table : the "x" symbol indicates the functions available in the different central unit series.

Binary functions	from pages C-8 to C-16	serie C^{tler}	40	50	90	30
&, AND	And	x	x	x	x	x
/, OR	Or	x	x	x	x	x
=	Allocation	x	x	x	x	x
=1	Exclusive OR	x	x	x	x	x
=R	Allocation reset memory	x	x	x	x	x
=S	Allocation set memory	x	x	x	x	x
I+	Pulse (positive edge)	x	x	x	x	x
I-	Pulse (negative edge)	x	x	x	x	x
MAJ	Majority				x	
RS	Set memory dominating	x	x	x	x	x
SR	Reset memory dominating	x	x	x	x	x

Timer functions	from pages C-16 to C-33	serie C^{tler}	40	50	90	30
ASV	OFF delay		x	x	x	x
ESV	ON delay		x	x	x	x
MOA	Monostable element "abort"		x	x	x	x
MOAT	Monostable element "abort" with time		x	x		
MOK	Monostable element "constant"		x	x	x	x
PDM	Pulse duration modulator		x	x	x	x
TIME_W	Time_word conversion		x	x		
TOF	OFF delay timing	x	x	x		
TON	ON delay timing	x	x	x		
TP	Pulse timing	x	x	x		
VVZ	Variable delay element				x	
W_TIME	Word_time conversion		x	x		

Counter functions	from pages C-33 to C-39	serie C^{tler}	40	50	90	30
CTU	Up counter	x	x	x		
CTUH	Hardware counter for encoder		x	x		
VRZ	Up/down counter	x	x	x	x	x
VRZD	Up/down counter, double word				x	

Comparison functions, word	from pages C-39 to C-48	serie C^{tler}	40	50	90	30
<	Less than	x	x	x	x	x
<=	Less than or equal	x	x	x	x	x
<>	Unequal	x	x	x	x	x
=?	Equal	x	x	x	x	x
>	Greater than	x	x	x	x	x

>=	Greater than or equal to	x	x	x	x	x
VGL3P	Comparator with 3-point response				x	
VGLEH	Comparator with unilateral hysteresis				x	
VGLUH	Comparator with asymmetrical hysteresis				x	

Arithmetic functions, word	from pages C-48 to C-60	serie C[tler]	40	50	90	30
+	Addition	x	x	x	x	x
-	Subtraction	x	x	x	x	x
*	Multiplication	x	x	x	x	x
DIV	Division	x	x	x	x	x
*: / MULDI	Multiplication with division	x	x	x	x	x
=W	Allocation	x	x	x	x	x
BETR	Absolute value generator		x	x	x	x
COS1	Cosinus				x	
MUL2N	Multiplication by 2 to the power of N		x	x	x	x
NEG	Negation		x	x	x	x
SIN1	Sinus				x	
SQRT	Square root		x	x	x	
ZUDKW	Allocation direct constant to word variable		x	x	x	x

Logical functions, word	from pages C-60 to C-68	serie C[tler]	40	50	90	30
MASKE	Mask				x	
SHIFT	Shift block				x	
WAND	AND combination, word		x	x	x	x
WOR	OR combination, word		x	x	x	x
WXOR	Exclusive OR combination, word		x	x	x	x

Program control functions	from pages C-68 to C-84	serie C[tler]	40	50	90	30
=PE	Conditional program end		x	x	x	x
ABORT	Program abort				x	
CAL_FB	Subroutine call			x		
CALLUP	Subroutine call for an assembler program				x	
DI	Read direct input		x	x		
DIN	Read direct inputs				x	
DO	Write direct output		x	x		
DOUT	Write direct outputs				x	
IOCON	Input/output configuration				x	
LZB	Run number block				x	
VTASK	Interrupt task validation		x	x		

CS31 functions	from pages C-84 to C-116	serie C[tler]	40	50	90	30
CONFIO1	1 analog channel configuration		x	x		
CONFIO4	4 analog channels configuration		x	x		
CONFIO8	8 analog channels configuration		x	x		
CS31CO	Configure CS31 module			x	x	x

CS31QU	Acknowledge CS31 error	x	x	x	
MT_CS31	data sent by CS31 master	x	x	x	
MR_CS31	data received by CS31 master	x	x	x	
ST_CS31	data sent by CS31 slave	x	x	x	
SR_CS31	data received by CS31 slave	x	x	x	

Communication functions	serie C[tler] from pages C-116 to C-154	40	50	90	30
AINIT	Initialisation of the ARCnet controller			x	
APOLL	Transfer of the data package to the ARCnet controller			x	
AREC / ARECitem	ARCnet data package receiving			x	
ASEND / ASEND+	ARCnet data package sending			x	
MODBUS	MODBUS master	x	x		x
MODBMASTK	MODBUS master for several interfaces	x	x	94	
REC / EMAS and RECvars	Receiving of ASCII characters and HEX values through a serail interface	x	x	X	x
SEND / DRUCK	Sending of ASCII characters and HEX values through a serail interface	x	x	X	x
SINIT	Initialization and configuration of the serial interfaces	x	x	X	x

Regulation functions	serie C[tler] from pages C-154 to C-174	40	50	90	30
DT1	Differentiator with delay of the 1st order			X	
INTK	Integrator (extended)			X	
PI	Proportional-integral controller	x	x	X	x
PIDT1	PIDT1 controller	x	x	X	
PT1	PT1 element			X	

Format conversion functions	serie C[tler] from pages C-174 to C-200	40	50	90	30
BCDDUAL / BCDBIN	BCD to binary conversion	x	x	X	x
BCDDUALD / BCDDW	BCD to binary conversion, double word			X	
DUALBCD / BINBCD	Binary to BCD conversion	x	x	X	x
DUABCDD / DWBCD	Binary to BCD conversion, double word			X	
DWW	Double word to word conversion	x	x	X	x
PACK4	Pack 4 binary variables in a word	x	x	X	x
PACK8	Pack 8 binary variables in a word	x	x	X	x
PACK16	Pack 16 binary variables in a word	x	x	X	x
PACKD4	Pack 4 binary variables in a double word			X	
PACKD8	Pack 8 binary variables in a double word			X	
PACKD16	Pack 16 binary variables in a double			X	

	word				
PACKD24	Pack 24 binary variables in a double word			X	
PACKD32	Pack 32 binary variables in a double word			X	
UNPACK4	Unpacking a word into 4 binary variables	x	x	X	x
UNPACK8	Unpacking a word into 8 binary variables	x	x	X	x
UNPACK16	Unpacking a word into 16 binary variables	x	x	X	x
UNPACKD4	Unpacking a double word into 4 binary variables			X	
UNPACKD8	Unpacking a double word into 8 binary variables			X	
UNPACKD16	Unpacking a double word into 16 binary variables			X	
UNPACKD24	Unpacking a double word into 24 binary variables			X	
UNPACKD32	Unpacking a double word into 32 binary variables			X	
WDW	Word to double word conversion	x	x	X	x

Comparison functions, double word	serie C[tler] / from pages C-200 to C-203	40	50	90	30
<D / VKLD	Less than, double word	x	x	X	
=?D / VGLD	Equal, double word	x	x	X	
>D / VGRD	Greater than, double word	x	x	X	

Arithmetic functions, double word	serie C[tler] / from pages C-203 to C-212	40	50	90	30
+D / ADDD	Addition, double word	x	x	X	
-D / SUBD	Subtraction, double word	x	x	X	
*D / MULD	Multiplication, double word	x	x	x	
:D / DIVD	Division, double word	x	x	x	
=D / ZUWD	Allocation, double word	x	x	x	
BETRD	Absolute value generator, double word			x	
MUL2ND	Double word multiplication by 2 to the power of N			x	
NEGD	Negation, double word			x	
SQRT	Square root	x	x	x	

Logical functions, double word	serie C[tler] / from pages C-212 to C-216	40	50	90	30
DWAND	AND combination, double word	x	x	x	
DWOR	OR combination, double word	x	x	x	
DWXOR	Exclusive OR combination, double word	x	x	x	
MASKED	Mask, double word			x	
SHIFT	Shift block			x	

High order functions	serie C^{tler} from pages C-216 to C-291	40	50	90	30
ADRWA	Address selection			x	
AMELD	Analog value change annonciator			x	
AMELDD	Analog value change annonciator, double word			x	
ANAI4_20	Read analog value 4...20 mA (07KT92)			x	
AWM	Selection multiplexer			x	
AWT	Selection gate, word	x	x	x	x
AWTB	Binary selection gate	x	x	x	x
AWTD	Selection gate, double word			x	
BEG	Limiter	x	x	x	x
BEGD	Limiter, double word			x	
BITSU	Bit searcher			x	
BMELD	Binary value change annunciator	x	x	x	
DMUX	Demultiplexer			x	
DMUXD	Demultiplexer, double word			x	
DWUMC	Double word decoder			x	
FEHSU	Error searcher with automatic deletion			x	
FIFO	Stack, first-in / first-out			x	
FKG	Function generator	x	x	x	
HLG	Ramp function generator			x	
IDLB	Read binary variable, indexed	x	x	x	
IDLm / IDL	Read word variable, indexed	x	x	x	x
IDSB	Write binary variable, indexed	x	x	x	
IDSm / IDS	Write word variable, indexed	x	x	x	x
INITS	Initialize memory area in the operand memory with zero			x	
INITV	Initialize variables			x	
LDT	Illumination pushbutton control			x	
LIFO	Stack, last-in / first-out			x	
LIZU	List allocator	x	x	x	x
MAX	Maximum value generator	x	x	x	x
MAXD	Maximum value generator, double word			x	
MAZ	Maximum value generator as a function of time			x	
MAZD	Maximum value generator as a function of time, double word			x	
MIN	Minimum value generator	x	x	x	x
MIND	Minimum value generator, double word			x	
MUXR	Multiplexer with reset			x	
MUXRD	Multiplexer with reset, double word			x	
NPULSE		x	x		
SFEHSU	Error searcher with storage			x	
UHR	Clock	x	x	x	x
USM	Switchover multiplexer			x	
UST	Switchover gate			x	
USTD	Switchover, double word			x	
USTR	Switchover gate with reset			x	

USTRD	Switchover with reset, double word			x	
WDEC	Word decoder			x	
WUMC	Word recoder			x	

Memory Access	serie C[tler] from pages C-291 to C-317	40	50	90	30
COPY	Copying memory areas	X	x	x	x
DWAES	Write double word in the event of value change			x	
DWOL	Read double word with enable			x	
DWOS	Write double word with enabling			x	
FDEL	Delete data segment in Flash EPROM			x	
FRD	Read data segment from the Flash EPROM			x	
FWR	Write data segment to the Flash EPROM			x	
IOR	Read byte value from I/O address			x	
IOW	Write byte value to I/O address			x	
RDB	Read binary values from historical values memory			x	
RDDW	Read double word values from historical values memory			x	
RDW	Read word values from historical values memory			x	
WAES	Write word in the event of value change			x	
WOL	Read word with enabling	X	x	x	x
WOS	Write word with enabling			x	
WRB	Write binary values into historical values memory			x	
WRDW	Write double word values to historical values memory			x	
WRW	Write word values to historical values memory			x	

Special functions	serie C[tler] from pages C-317 to C-323	40	50	90	30
5F_ARC94	Arcnet for 07KT94			94	
COUNTB	Test of number of bits in a word/double word			94	
COUNTW	Fast counter on 07KT94			94	
DWWW	One double word in 2 words conversion			94	
IDENT	Identification			94	
MODMASTK	MODBUS master	x	X	94	
SETB	Set a bit in a word/double word			94	
TESTB	Test a bit in a word/double word			94	
WWDW	2 words in one double word conversion			94	

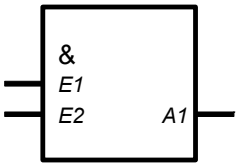
2 Basic operators/functions

2.1 Binary functions

Binary functions	from pages C-8 to C-16	serie C[tler]	40	50	90	30
&, AND	And	x	x	x	x	x
/, OR	Or	x	x	x	x	x
=1	Exclusive OR	x	x	x	x	x
=	Allocation	x	x	x	x	x
=R	Allocation reset memory	x	x	x	x	x
=S	Allocation set memory	x	x	x	x	x
I+	Pulse (positive edge)	x	x	x	x	x
I-	Pulse (negative edge)	x	x	x	x	x
MAJ	Majority				x	
RS	Set memory dominating	x	x	x	x	x
SR	Reset memory dominating	x	x	x	x	x

& AND

FBD



IL (compatible with IEC 1131-3)

```
LD    E1
AND   E2
ST    A1
```

PARAMETERS

E1	BINARY	%I, %M, %O, %S, %K	Operand 1
E2	BINARY	%I, %M, %O, %S, %K	Operand 2 **capable of duplication**
A1	BINARY	%M, %O, %S	Result of AND combination

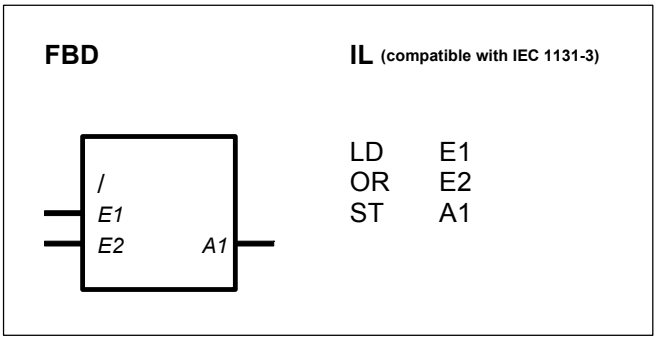
DESCRIPTION

This connection element realizes a logical AND combination of the operands at the inputs. The result is allocated to the operand at the output.

Truth table :

E1	E2	A1
0	0	0
1	0	0
0	1	0
1	1	1

/ OR



PARAMETERS

E1	BINARY	%I, %M, %O, %S, %K	Operand 1
E2	BINARY	%I, %M, %O, %S, %K	Operand 2 **capable of duplication**
A1	BINARY	%M, %O, %S	Result of the OR combination

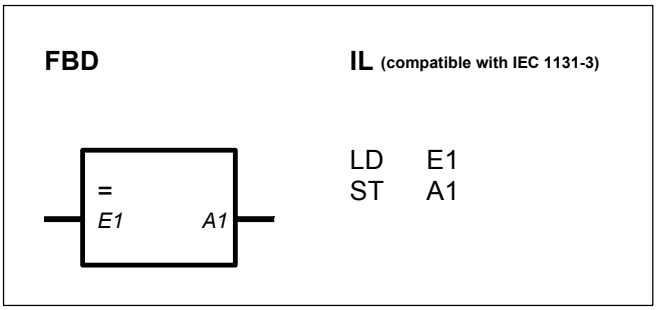
DESCRIPTION

This connection element realizes a logical OR combination of the operands at the inputs. The result is allocated to the operand at the output.

Truth table:

E1	E2	A1
0	0	0
1	0	1
0	1	1
1	1	1

= ALLOCATION



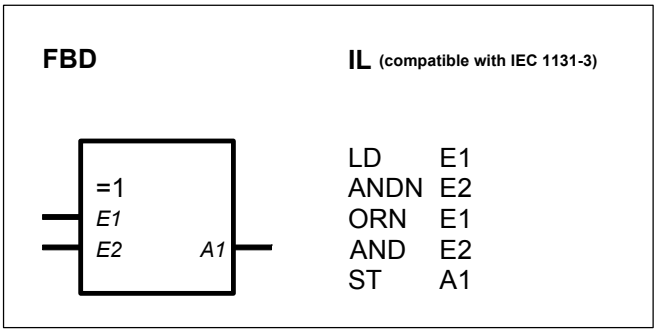
PARAMETERS

E1	BINARY	%I, %M, %O, %K,%S	Source
A1	BINARY	%M, %O, %K,%S	Target

DESCRIPTION

This connection element allocates the value of the operand at the input E1 to the operand at the output A1.

=1 EXCLUSIVE OR



PARAMETERS

E1	BINARY	%I, %M,%O,%S,%K	Operand 1
E2	BINARY	%I,%M,%O,%S,%K,	Operand 2
A1	BINARY	%M,%O,%S	Result of the XOR combination

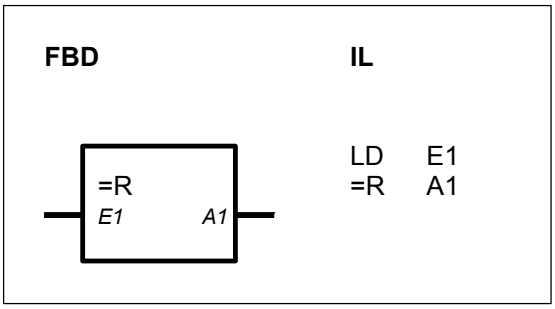
DESCRIPTION

This connection element realizes a logical EXCLUSIVE OR combination of the operands at the inputs. The result is allocated to the operand at the output.

Truth table:

E1	E2	A1
0	0	0
1	0	1
0	1	1
1	1	0

=R ALLOCATION RESET MEMORY



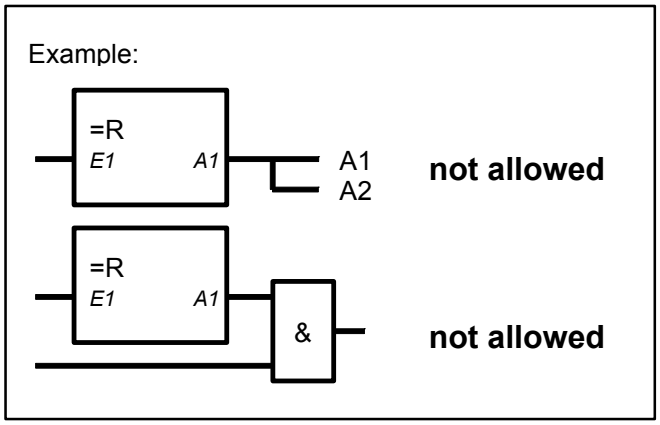
PARAMETERS

E1	BINARY	%I, %M, %O, %S, %K	Reset condition
A1	BINARY	%M, %O	Store variable

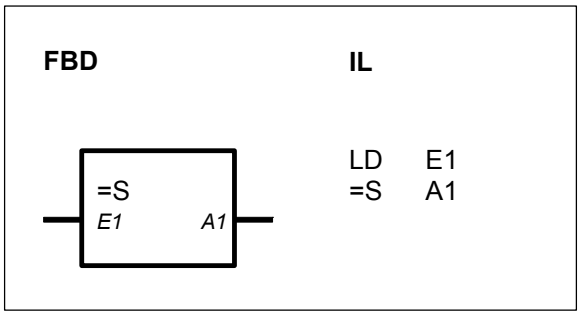
DESCRIPTION

A state 1 at the input sets the operand at the output to a state 0. A state 0 at the input has no influence on the operand at the output.

IMPORTANT :
This function block must only be used as an output function block : i. e., in the FBD, it must not be connected further by a line on the output side.
An operand (%M or %O) *must* be specified at the output.



=S ALLOCATION SET MEMORY



PARAMETERS

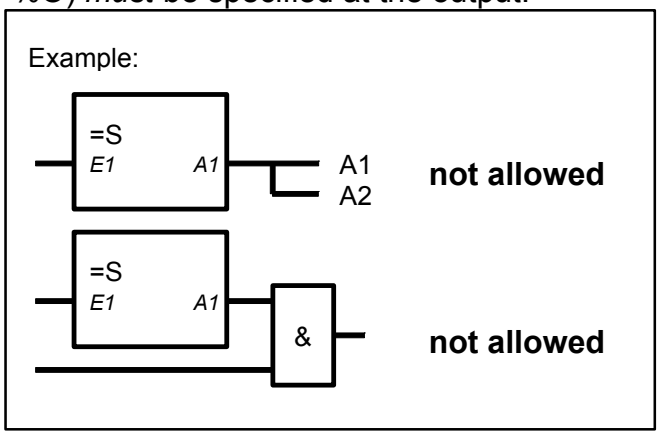
E1	BINARY	%I, %M, %O, %S, %K	Set condition
A1	BINARY	%M, %O	Storage variable

DESCRIPTION

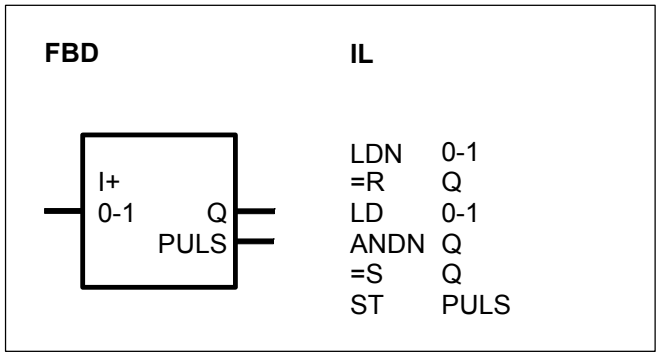
A state 1 at the input sets the operand at the output to a state 1. A state 0 at the input has no influence on the operand at the output.

IMPORTANT :
This function block must only be used as an output function block : i. e., in the FBD, it must not be connected further by a line on the output side.

An operand (%M or %O) *must* be specified at the output.



I+ PULSE (POSITIVE EDGE)



PARAMETERS

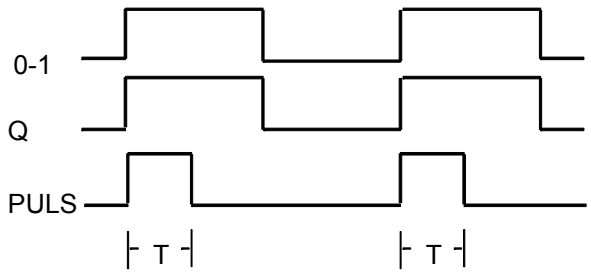
0-1	BINARY	%I, %M, %O, %S, %K	Input for 0 - 1 edge
Q	BINARY	%O, %M	Output for edge detection
PULS	BINARY	%O, %M	Pulse output

DESCRIPTION

A positive edge (0->1) at the input 0-1 generates a pulse with the duration of one PLC program cycle at the PULS output.

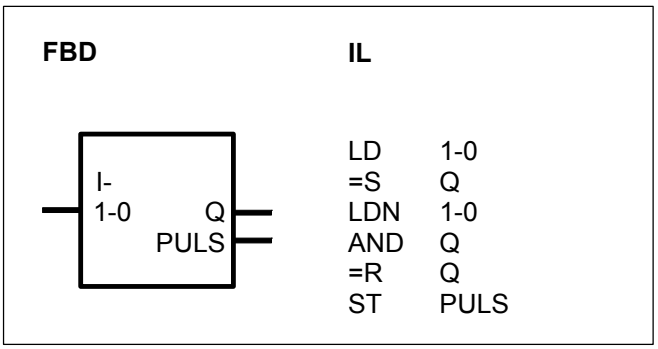
The output Q is needed for edge detection. This flag must not be used again in the PLC program.

Duration of the pulse : From recognition of the 0-1 edge by the connection element until renewed processing of this connection element in the next program cycle.



T: Program cycle time.

I- PULSE (NEGATIVE EDGE)



PARAMETERS

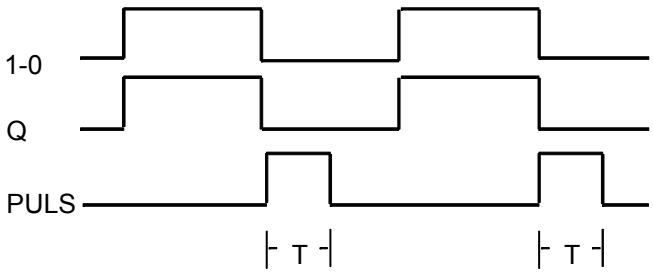
1-0	BINARY	%I, %M, %O, %S, %K	Input for 1 - 0 edge
Q	BINARY	%O, %M	Output for interrogation of the direct flag
PULS	BINARY	%O, %M	Pulse output

DESCRIPTION

A negative edge (1->0) at the input 1-0 generates a pulse at the output PULS which has the duration of 1 PLC program cycle.

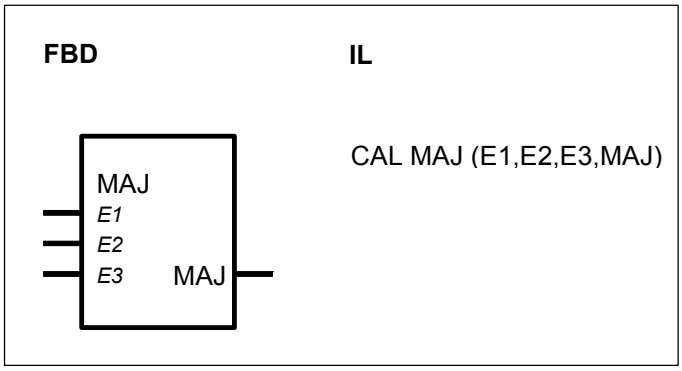
The output Q is needed for edge detection. This flag must not be used again in the PLC program.

Duration of the pulse : From recognition of the 1-0 edge by the connection element up to renewed processing of this connection element in the next program cycle.



T: Program cycle time of the PLC

MAJ MAJORITY



PARAMETERS

E1	BINARY	%I, %M, %O, %S, %K	Operand 1
E2	BINARY	%I, %M, %O, %S, %K	Operand 2
E3	BINARY	%I, %M, %O, %S, %K	Operand 3
MAJ	BINARY	%M, %O	Result

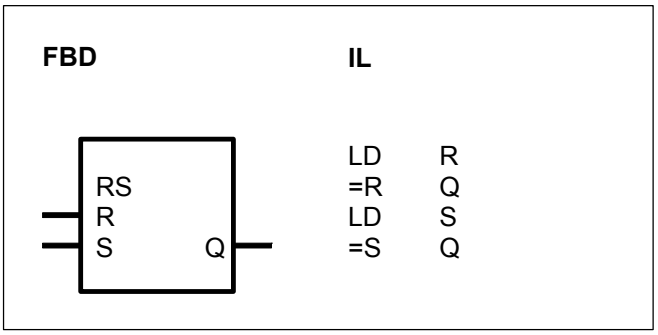
DESCRIPTION

This connection element realizes a MAJORITY element.

If at least 2 of the 3 binary operands at the inputs E1, E2 and E3 have the state 1, then the state 1 is allocated to the binary operand at the output MAJ.

If it is not the case, the state 0 is allocated to the binary operand at the output MAJ.

RS SET MEMORY, DOMINATING



PARAMETERS

R	BINARY	%I, %M, %O, %S, %K	Reset input
S	BINARY	%I, %M, %O, %S, %K	Set input
Q	BINARY	%M, %O	Flip-flop output

DESCRIPTION

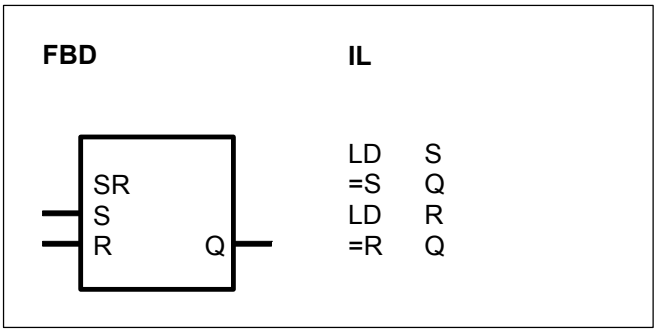
A status 1 at the input R resets the operand Q to the status 0.

A status 1 at the input S sets the operand Q to the status 1.

A simultaneous 1 status at the inputs S and R sets the operand Q to a status 1 (dominating set).

A status 0 at the input S or R has no influence on the operand Q.

SR RESET MEMORY, DOMINATING



PARAMETERS

S	BINARY	%I, %M, %O, %S, %K	Set input
R	BINARY	%I, %M, %O, %S, %K	Reset input
Q	BINARY	%M, %O	Flip-flop output

DESCRIPTION

A status 1 at the input S sets the operand Q to the status 1.

A status 1 at the input R resets the operand Q to the status 0.

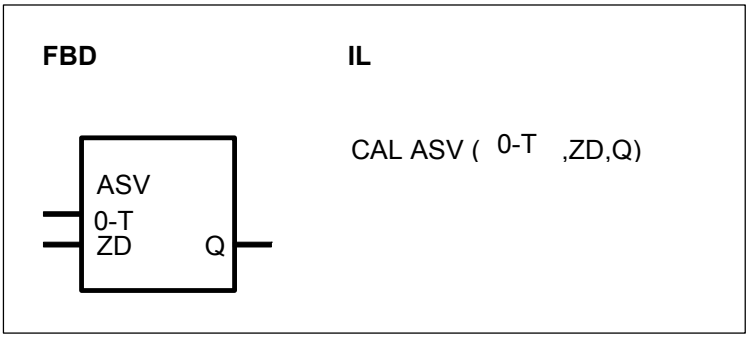
A simultaneous 1 status at the inputs S and R resets the operand to a status 0 (dominating reset).

A status 0 at the input S or R has no influence on the operand Q.

2.2 Timer functions

Timer functions	from pages C-16 to C-33	serie C[tler]	40	50	90	30
ASV	OFF delay		x	x	x	x
ESV	ON delay		x	x	x	x
MOA	Monostable element "abort"		x	x	x	x
MOAT	Monostable element "abort" with time		x	x		
MOK	Monostable element "constant"		x	x	x	x
PDM	Pulse duration modulator		x	x	x	x
TIME_W	Time_word conversion		x	x		
TOF	OFF delay timing	x	x	x		
TON	ON delay timing	x	x	x		
TP	Pulse timing	x	x	x		
VVZ	Variable delay element				x	
W_TIME	Word_time conversion		x	x	x	

ASV OFF DELAY



PARAMETERS

0 -T	BINARY	%I, %M, %O, %S, %K	Input signal
ZD	DOUBLE WORD	%MD, %KD	Delay time
Q	BINARY	%M, %O	Delayed signal

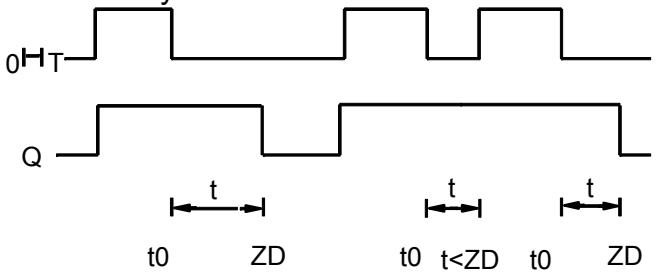
DESCRIPTION

The 1-0 edge of the input 0⊢⊣T is delayed by the time ZD and is output as a 1-0 edge at the output Q. If the input 0⊢⊣T returns to 1 level before expiry of the time ZD, the output Q retains 1 level.

The time is specified in milliseconds. Only integral multiple of 5 ms are admissible (Examples: 5 ms, 500 ms, 100 000 ms, ...). Time range which can be specified : 5 ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time

Meaningful range for ZD: > 1 cycle time.



General response

• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

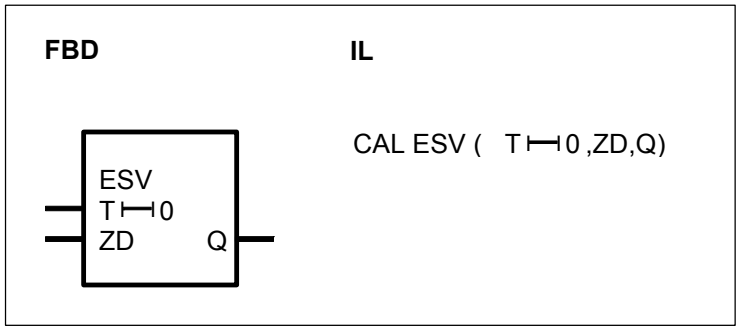
Important note :

Up to versions 07 KR 91 index f and 07 KT 92 index f, the behaviour of the timers is the following : processing of a timer in the old PLC's operating system is *not* influenced by the following commands :

- Abort program
- Start program
- Stop program
- Continue program

That is to say, processing of a started timer is continued in the PLC's operating system even if the affiliated PLC program is aborted, restarted or stopped and continued again.

ESV ON DELAY



PARAMETERS

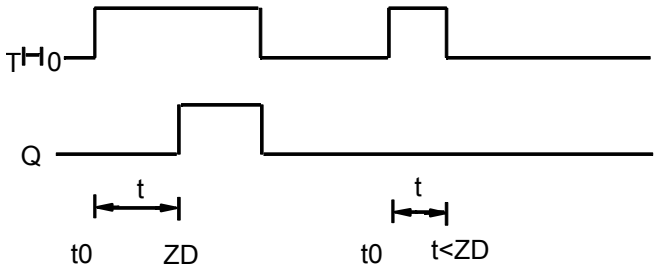
T ⊢ 0	BINARY	%I, %M, %O, %S, %K	Input signal
ZD	DOUBLE WORD	%MD, %KD	Delay time
Q	BINARY	%M, %O	Delayed signal

DESCRIPTION

The 0-1 edge of the input T⊢ 0 is delayed by the time ZD and is output as a 0-1 edge at the output Q. The output Q retains 0 level if the input T⊢ 0 returns to 0 level before the time ZD has elapsed.

The time is specified in milliseconds in a double word %MD or %KD. Only integral multiple of 5 ms are admissible (Examples: 5 ms, 500 ms, 100 000 ms, ...). Time range which can be specified is : 5 ms ... 24.8 days.

Maximum time offset at the output : < 1 cycle time
Meaningful range for ZD : > 1 cycle time



General response

• Started timers are processed by the PLC operating system and are completely independent of processing of the PLC program.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

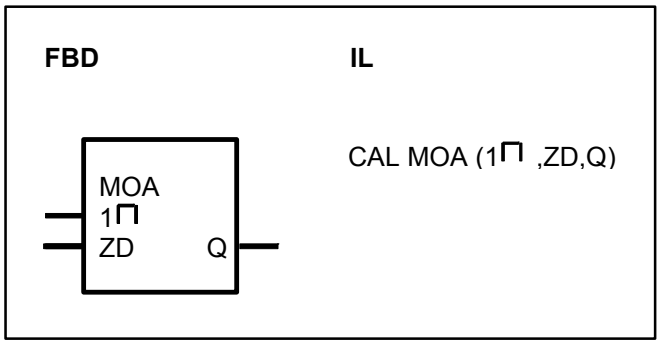
Important note :

Up to versions 07 KR 91 index f and 07 KT 92 index f, the behaviour of the timers is the following : processing of a timer in the old PLC's operating system is *not* influenced by the following commands :

- Abort program
- Start program
- Stop program
- Continue program

That is to say, processing of a started timer is continued in the PLC's operating system even if the affiliated PLC program is aborted, restarted or stopped and continued again.

MOA MONOSTABLE ELEMENT ”ABORT”



PARAMETERS

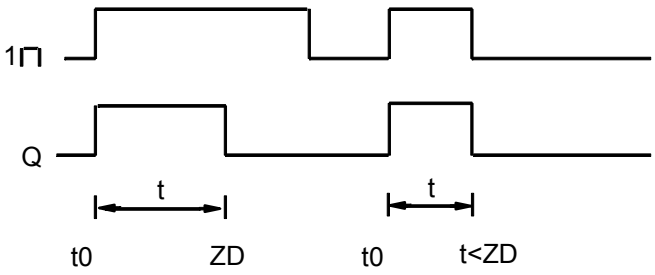
1⊓	BINARY	%I, %M, %O, %S, %K	Input signal
ZD	DOUBLE WORD	%MD, %KD	Pulse length
Q	BINARY	%M, %O	Output signal

DESCRIPTION

A 0-1 edge at the input 1⊓ produces a 0-1 edge at the output Q. If the input 1⊓ remains at 1 level, a 1-0 edge is output on output Q after duration ZD has elapsed.
The output Q is also set back to 0 level if the input 1⊓ should return to 0 level before expiry of time TD.

The time is specified in milliseconds. Only integral multiple of 5 ms are admissible (Examples: 5 ms, 500 ms, 100 000 ms, ...). Time range which can be specified: 5ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time
Meaningful range for ZD: > 1 cycle time



General response

• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP

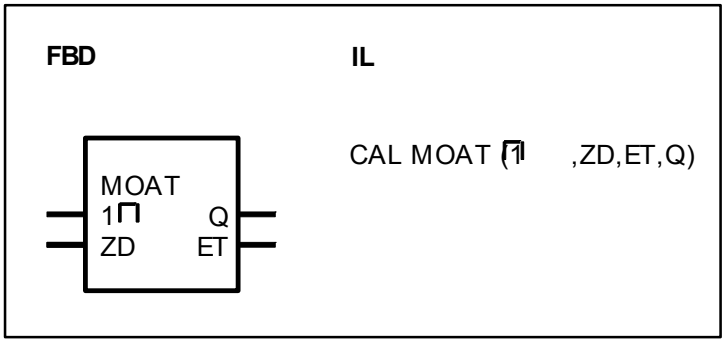
- Warm or cold start

Important note :
Up to versions 07 KR 91 index f and 07 KT 92 index f, the behaviour of the timers is the following : processing of a timer in the old PLC's operating system is *not* influenced by the following commands :

- Abort program
- Start program
- Stop program
- Continue program

That is to say, processing of a started timer is continued in the PLC's operating system even if the affiliated PLC program is aborted, restarted or stopped and continued again.

MOAT MONOSTABLE ELEMENT "ABORT" with time



PARAMETERS

1⊓	BINARY	%I, %M, %O, %S, %K	Input signal
ZD	WORD	%KW & %KW+1 %MW & %MW+1	Pulse length
	DOUBLE WORD	%MD, %KD	
Q	BINARY	%M, %O	Delayed signal
ET	WORD	%MW & %MW+1	Time visualization
	DOUBLE WORD	%MD	

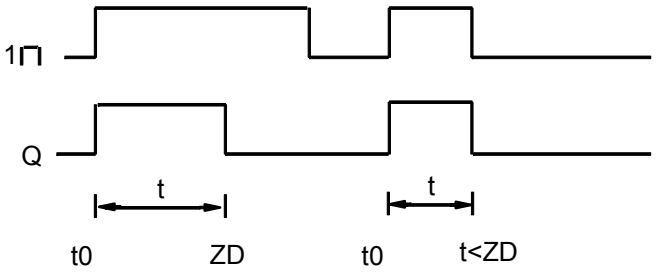
DESCRIPTION

A 0-1 edge at the input 1⊓ produces a 0-1 edge at the output Q. If the input 1⊓ remains at 1 level, a 1-0 edge is output on output Q after duration ZD has elapsed.
The output Q is also set back to 0 level if the input 1⊓ should return to 0 level before expiry of time TD.

The time elapsed can be consulted at the output ET and the pulse lenght at the input ZD can be modified when the timer is running. The pulse lenght is specified in milliseconds. The time range which can be specified is : 1 ms ... 24.8 days.

Maximum time offset at the output : < 1 cycle time

Meaningful range for PT : > 1 cycle time.



Note :
If the time is less than 65s, a word can be used for the preset time PT. Then the PT input can be used :
- with all the other word functions
- from the central unit potentiometer
- or for MODBUS communication (double word are not allowed in MODBUS)
directly without double word to word conversion.

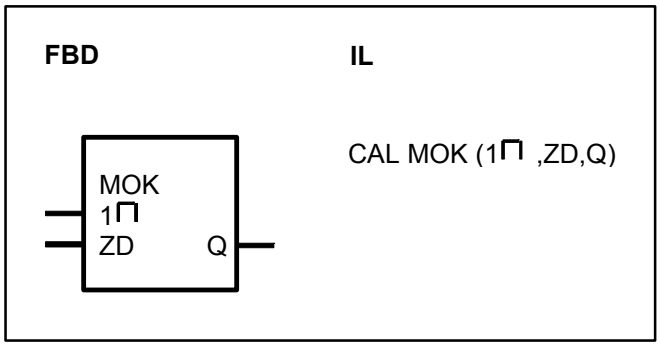
If word variables (%MW or %KW) are used for the parameter PT, two consecutive addresses are necessary. Never use %MW+1 or %KW+1 in your program in this case.

General response
• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

MOK MONOSTABLE ELEMENT ”CONSTANT”



PARAMETERS

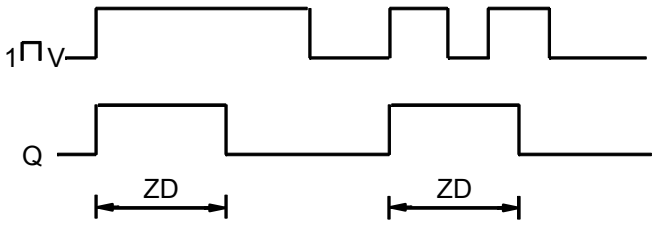
1⊓ V	BINARY	%I, %M, %O, %S, %K	Input signal
ZD	DOUBLE WORD	%MD, %KD	Pulse length
Q	BINARY	%M, %O	Output signal

DESCRIPTION

A 0-1 edge at the input 1⊓ produces a 0-1 edge at the output Q. The output Q is reset to 0 level after expiry of the period ZD. A second 0-1 edge of the input 1⊓ V before the time ZD has elapsed is ignored.

The time is specified in milliseconds. Only integral multiple of 5 ms are admissible (Examples: 5 ms, 500 ms, 100 000 ms, ...). Time range which can be specified: 5 ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time
Meaningful range for ZD: > 1 cycle time



General response

• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

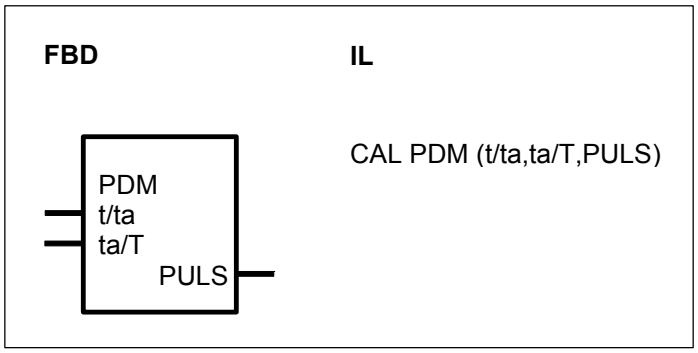
Important note :

Up to versions 07 KR 91 index f and 07 KT 92 index f, the behaviour of the timers is the following : processing of a timer in the old PLC's operating system is *not* influenced by the following commands :

- Abort program
- Start program
- Stop program
- Continue program

That is to say, processing of a started timer is continued in the PLC's operating system even if the affiliated PLC program is aborted, restarted or stopped and continued again.

PDM PULSE DURATION MODULATOR



PARAMETERS

t/ta	WORD	%IW, %MW, %OW, %KW	Duty ratio
ta/T	WORD	%IW, %MW, %OW, %KW	Period referred to the cycle time
PULS	BINARY	%O, %M	Pulse duration modulated signal

DESCRIPTION

This function block generates a pulse duration-modulated binary signal at its PULS output.

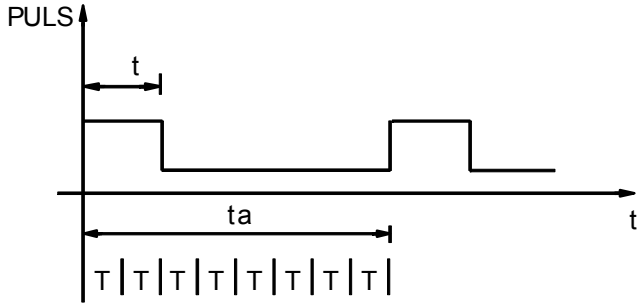
The duty ratio is specified at the t/ta input and the period for the output signal is specified at the ta/T input.
In case of serie 40 and 50 and cycle time equal to 0 , T is equal to 10ms

t/ta WORD
The required duty ratio for the output signal PULS is specified at the input t/ta.
- ta is the period of the signal at the output PULS
- t is the time within the period ta during which the output signal assumes a 1 level.

The specified value for the required duty ratio at the input t/ta must be specified in scaled form. To do this, the required duty ratio must be multiplied by the value 32767 and rounded to a whole number. The resulting numerical value is then specified at the input t/ta.

Marginal condition for t : t > T
That is to say, the required duty cycle of the output signal must be higher than the

cycle time of the PLC program.



The following relationship applies to specification of the keying ratio at the input t/ta:

Scaled value at the t/ta input	**Results in duty ratio at the output PULS**
Negative value	0 (0 %)
0 (0 * 32767)	0 (0 %)
.	.
.	.
16384 (0,5 * 32767)	0,5 (50 %)
.	.
.	.
32767 (1 * 32767)	1 (100 %)

ta/T WORD
The required period ta for the signal at the output PULS is specified at the input ta/T. At the same time, the period ta must be scaled to the cycle time T.
Marginal condition for ta:

- ta = n * T : ta must be an integral multiple of T
- ta >> T > 0; the higher ta is in relation to T, the more exactly the required duty ratio is kept to.
  E.g. ta $\geq$ 10 * T -> inaccuracy of the duty ratio at the output PULS $\leq$ 10%.

If a value ta/T < 0 is specified for ta/T, the function block automatically replaces this meaningless value by 32767.

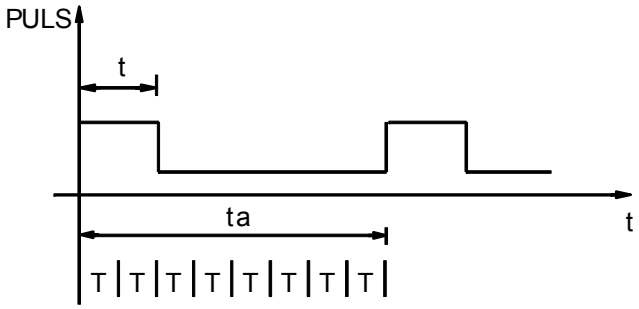
PULS BINARY
The pulse duration modulated signal is available at the PULS output.

EXAMPLE
Required :

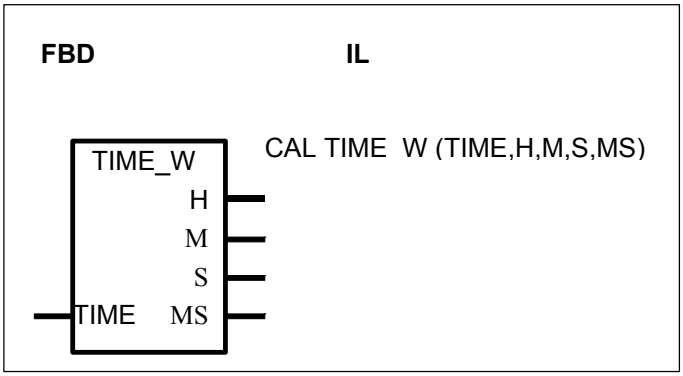
- Duty ratio : t/ta = 0,25 (25 %)
- Period : ta = 800 ms (only an integral multiple of the PLC cycle time is possible)
- Cycle time : T = 100 ms

Block parameters to be specified :

- Value at the input t/ta : 8192 (0,25 * 32767)
- Value at the input ta/T : 8 (800 ms/100 ms)



TIME_W TIME_WORD CONVERSION



PARAMETERS

TIME	DOUBLE WORD	%MD, %KD	Time value
H	WORD	%MW ,%OW	Hour value
M	WORD	%MW ,%OW	Minute value
S	WORD	%MW ,%OW	Second value
MS	WORD	%MW ,%OW	Millisecond value

DESCRIPTION

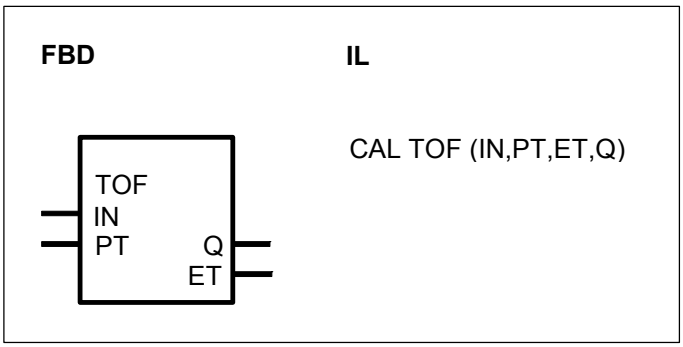
This functions is used to manage time value with word variables
The Time value TIME is converted in hours, minutes, seconds and milliseconds.

The time value is in millisecond

The maximum value is
0<=TIME<=986399999
0<= H <= 273
0<= M <= 59
0<= S <= 59
0<= MS<= 999

IF TIME is greater than 986399999 then H, M, S, MS are set to the maximum value
IF TIME is negative then H, M, S, MS are set to zero

TOF OFF DELAY



PARAMETERS

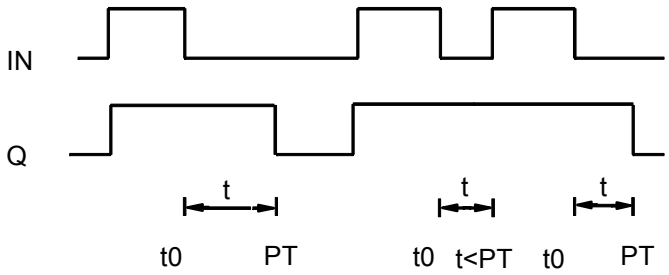
IN	BINARY	%I, %M, %O, %S, %K	Input signal
PT	WORD	%KW & %KW+1 %MW & %MW+1	Preset time
	DOUBLE WORD	%MD, %KD	
Q	BINARY	%M, %O	Delayed signal
ET	WORD	%MW & %MW+1	Time visualization
	DOUBLE WORD	%MD	

DESCRIPTION

The 1-0 edge of the input IN is delayed by the time PT at the output Q. If the input IN returns to 1 level before expiry of the time PT, the output Q retains 1 level.

The time elapsed can be consulted at the output ET and the preset time value at the input PT can be modified when the timer is running. The preset time is specified in milliseconds. The time range which can be specified is : 1 ms ... 24.8 days.

Maximum time offset at the output : < 1 cycle time
Meaningful range for PT : > 1 cycle time.



Note :
If the time is less than 65s, a word can be used for the preset time PT. Then the PT input can be used :
- with all the other word functions
- from the central unit potentiometer

- or for MODBUS communication (double word are not allowed in MODBUS) directly without double word to word conversion.

If word variables (%MW or %KW) are used for the parameter PT, two consecutive addresses are necessary. Never use %MW+1 or %KW+1 in your program in this case.

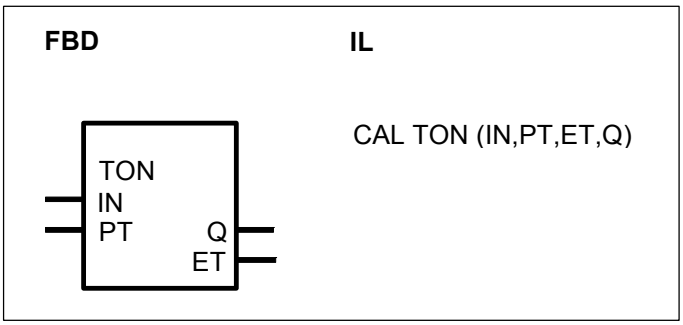
General response

• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

TON ON DELAY



PARAMETERS

IN	BINARY	%I, %M, %O, %S, %K	Input signal
PT	WORD	%KW & %KW+1 %MW & %MW+1	Preset time
	DOUBLE WORD	%MD, %KD	
Q	BINARY	%M, %O	Delayed signal
ET	WORD	%MW & %MW+1	Time visualization
	DOUBLE WORD	%MD	

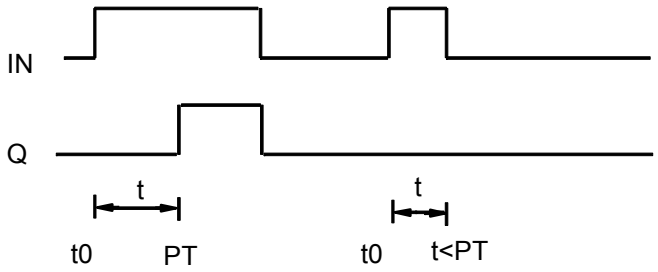
DESCRIPTION

The 0-1 edge of the input IN is delayed by the time PT at the output Q. The output Q retains 0 level if the input IN returns to 0 level before the time PT has elapsed.

The time elapsed can be consulted at the output ET and the preset time value at the input PT can be modified when the timer is running. The preset time is specified in

milliseconds. The time range which can be specified is : 1 ms ... 24.8 days.

Maximum time offset at the output : < 1 cycle time
Meaningful range for PT : > 1 cycle time



Note :
If the time is less than 65s, a word can be used for the preset time PT. Then the PT input can be used :
- with all the other word functions
- from the central unit potentiometer
- or for MODBUS communication (double word are not allowed in MODBUS)
directly without double word to word conversion.

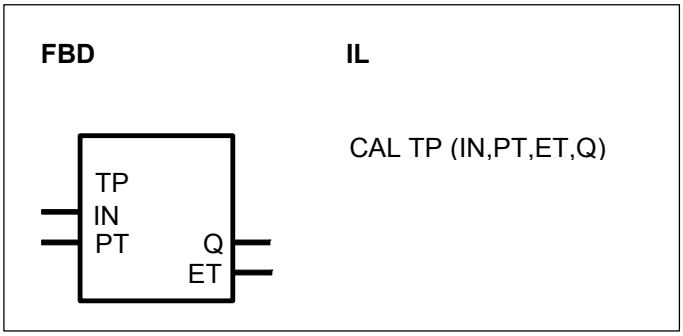
If word variables (%MW or %KW) are used for the parameter PT, two consecutive addresses are necessary. Never use %MW+1 or %KW+1 in your program in this case.

General response
• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

• Processing of a timer in the PLC operating system is influenced by the following commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

TP MONOSTABLE ELEMENT ”CONSTANT”



PARAMETERS

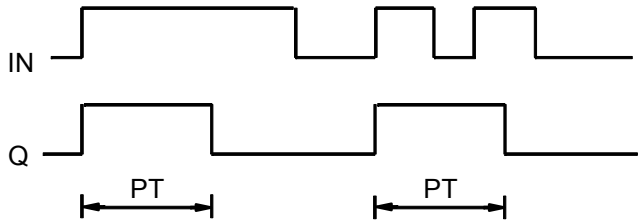
IN	BINARY	%I, %M, %O, %S, %K	Input signal
PT	WORD	%KW & %KW+1 %MW & %MW+1	Preset time
	DOUBLE WORD	%MD, %KD	
Q	BINARY	%M, %O	Delayed signal
ET	WORD	%MW & %MW+1	Time visualization
	DOUBLE WORD	%MD	

DESCRIPTION

A 0-1 edge at the input IN produces a 0-1 edge at the output Q. The output Q is reset to 0 level after expiry of the period PT. A second 0-1 edge of the input IN before the time PT has elapsed is ignored.

The time elapsed can be consulted at the output ET and the preset time value at the input PT can be modified when the timer is running. The preset time is specified in milliseconds. The time range which can be specified is : 1 ms ... 24.8 days.

Maximum time offset at the output : < 1 cycle time
Meaningful range for PT : > 1 cycle time



Note :
If the time is less than 65s, a word can be used for the preset time PT. Then the PT input can be used :
- with all the other word functions
- from the central unit potentiometer
- or for MODBUS communication (double word are not allowed in MODBUS)
directly without double word to word conversion.

If word variables (%MW or %KW) are used for the parameter PT, two consecutive addresses are necessary. Never use %MW+1 or %KW+1 in your program in this case.

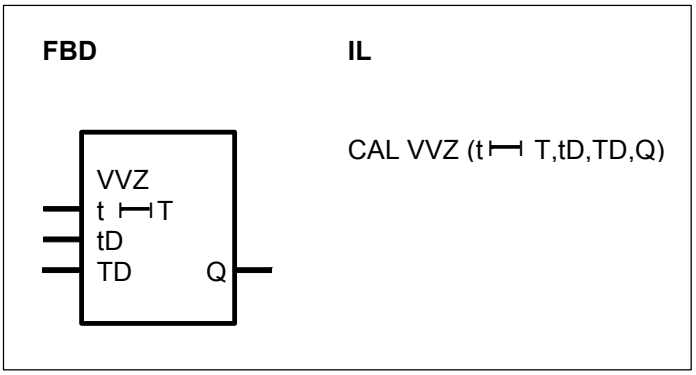
General response
• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program. An appropriate message of the operating system is not issued to the affiliated timer block in the PLC program until the timer has elapsed.

• Processing of a timer in the PLC operating system is influenced by the following

commands. All running timers are stopped and initialized when one of the following actions occurs:

- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

VVZ VARIABLE DELAY ELEMENT



PARAMETERS

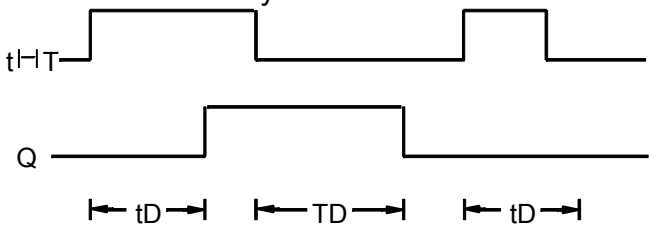
t⊢T	BINARY	%I, %M, %O, %S, %K	Input signal
tD	DOUBLE WORD	%MD, %KD	Delay time 0-1 edge
TD	DOUBLE WORD	%MD, %KD	Delay time 1-0 edge
Q	BINARY	%M, %O	Output signal

DESCRIPTION

A 0-1 edge at the input t⊢T longer or equal to tD activates the output Q during the time TD. The output Q is delayed from the input t⊢T by the time tD.

The time is specified in milliseconds. Only integral multiple of 5 ms are admissible (Examples: 5 ms, 500 ms, 100 000 ms, ...). Time range which can be specified : 5 ms ... 24.8 days.

Maximum time offset at the output: < 1 cycle time
Meaningful range for tD and TD: > 1 cycle time



General response

• Started timers are processed by the PLC operating system and are therefore completely independent of processing of the PLC program.

• Processing of a timer in the PLC operating system is influenced by the following

commands. All running timers are stopped and initialized when one of the following actions occurs:

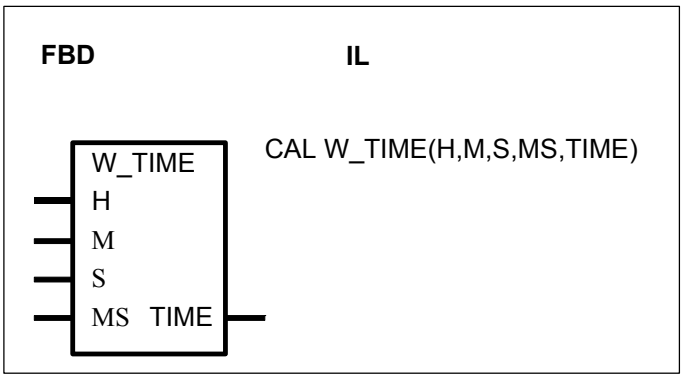
- Abort PLC program
- RUN/STOP switch from RUN -> STOP
- Warm or cold start

Important note :
Up to versions 07 KR 91 index f and 07 KT 92 index f, the behaviour of the timers is the following : processing of a timer in the old PLC's operating system is *not* influenced by the following commands :

- Abort program
- Start program
- Stop program
- Continue program

That is to say, processing of a started timer is continued in the PLC's operating system even if the affiliated PLC program is aborted, restarted or stopped and continued again.

W_TIME WORD_TIME CONVERSION



PARAMETERS

H	WORD	%IW, %KW, %MW, %OW	Hour value
M	WORD	%IW, %KW, %MW, %OW	Minute value
S	WORD	%IW, %KW, %MW, %OW	Second value
MS	WORD	%IW, %KW, %MW, %OW	Millisecond value
TIME	DOUBLE WORD	%MD	Time value

DESCRIPTION

This functions is used to set a TIME value through words
The words values are converted in a double word for timer functions

The time value is in millisecond

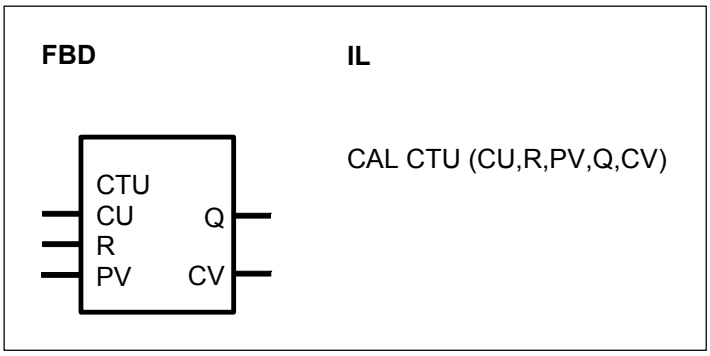
The maximum value is
0<= H <= 273
0<= M <= 32767
0<= S <= 32767
0<= MS<= 32767

If one parameter is set to a negative value, the value used for the internal calculation is 0

2.3 Counter functions

Counter functions	from pages C-33 to C-39	serie C[tler]	40	50	90	30
CTU	Up counter	x	x	x		
CTUH	Hardware counter for encoder		x	x		
VRZ	Up/down counter	x	x	x	x	x
VRZD	Up/down counter, double word				x	

CTU UP COUNTER



PARAMETERS

CU	BINARY	%I, %M, %O, %K, %S	Pulse input
R	BINARY	%I, %M, %O, %K, %S	Counter reset input
PV	WORD	%IW, %MW, %OW, %KW	High counter limit
Q	BINARY	%M, %O,	Limit indicator
CV	WORD	%OW, %MW	Counter value

DESCRIPTION

This function block serves to count pulses. Each positive edge (0->1 edge) at the input CU increases the current counter value specified at output CV by 1.

CU BINARY
The pulse signal is allocated to the input CU. The positive edge of the pulse is evaluated in each case.

R BINARY
A 1 signal at the input R sets the counter content to the value 0.
The reset input R has the highest priority.

PV WORD
The high limit of the counter is specified at the input PV.

Q BINARY
The output Q indicates if the counter value is higher or not than the value at the input PV.
CV ≥ PV -> Q = 1

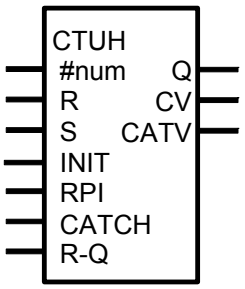
CV < PV -> Q = 0

CV WORD
The current counter value is available at the output CV.
If the counter reaches the positive or negative limit of the number range, the counter is limited to this value.

Number range
Integer word (16 bits)
• Low limit : 0
• High limit : $7FFF_H$ + 32767

CTUH HIGH SPEED COUNTER

FBD



IL

```
CAL   CTUH  (#num,R,S,INIT,RPI,
            CATCH,R-Q,Q,CV,CATV)
```

PARAMETERS

#num	DIRECT CONSTANT	#, #H	Counter mode
R	BINARY	%I, %M, %O, %K, %S	Counter reset input
S	BINARY	%I, %M, %O, %K, %S	Counter set input
INIT	WORD	%IW, %MW, %OW, %KW	Set value
RPI	BINARY	%I, %M, %O, %K, %S	Reset point indicator
CATCH	BINARY	%I, %M, %O, %K, %S	Catch counter value
R-Q	BINARY	%I, %M, %O, %K, %S	Reset bit overflow
Q	BINARY	%M, %O	Overflow
CV	WORD	%OW, %MW	Counter value
CATV	WORD	%OW, %MW	Catched counter value

DESCRIPTION

The CTUH function block allows the counting of high speed counters of units serie 40 and 50.

Units serie 40 and 50 have two high speed counters that can be used in the following modes :

- C1 : counting on the %I 62.00 input
  Counting start : on positive edge (0->1 edge)
  Representation : 1 word (16 bits)

Overflow : when passing from -1 to 0.
Capture : on positive edge of the %I 62.02 input

- C2 : counting on the %I 62.01 input
Counting start : on positive edge (0->1 edge)
Representation : 1 word (16 bits)
Overflow : when passing from -1 to 0.
Capture : on positive edge of the %I 62.03 input

- Incremental encoder : counting on the %I 62.00 and %I 62.01 inputs
Counting start : on positive edge (0->1 edge)
Representation : 1 word (16 bits)
Overflow : when passing from -1 to 0 or from 0 to -1
Capture : on positive edge of the %I 62.02 input
In case of defective channel (ie one input not connected) the value increases of +1 and decreases of -1.

#num DIRECT CONSTANT
The counting mode is specified at the input #num.
#1 = counter C1
#2 = counter C2
#3 = incremental encoder
>3 -> the block is not processed

R BINARY
A 1 signal at the input R resets the counter value and the capture register to the value 0. The reset input R has the highest priority.
If R = 1 then CV = 0 and CATV = 0

S BINARY
A 1 signal at the input S loads the counter value with the preset value specified at the input INIT.
If S = 1 then CV = INIT

INIT WORD
The preset value is specified at the input INIT.

RPI BINARY
A 1 signal at the input RPI validates the counter value capture and the counter reset during the capture. The RPI input has a higher priority than CATCH.
RPI = 1 Capture is valid on all the counters.
If there is a positive edge on %I 62,02 or %I 62,03, there is a hard capture of the counter. The counter is reset to 0.

CATCH BINARY
A 1 signal at the input CATCH validates the counter value capture.
CATCH = 0 Capture not valid
CATCH = 1 Capture is valid on all the counters.
If there is a positive edge on %I 62,02 or %I 62,03, there is a

hard capture of the counter. The counter is not reset to 0.

R-Q BINARY
A 1 signal at the input R-Q resets the overflow to the value 0.
If R-Q = 1 then Q = 0

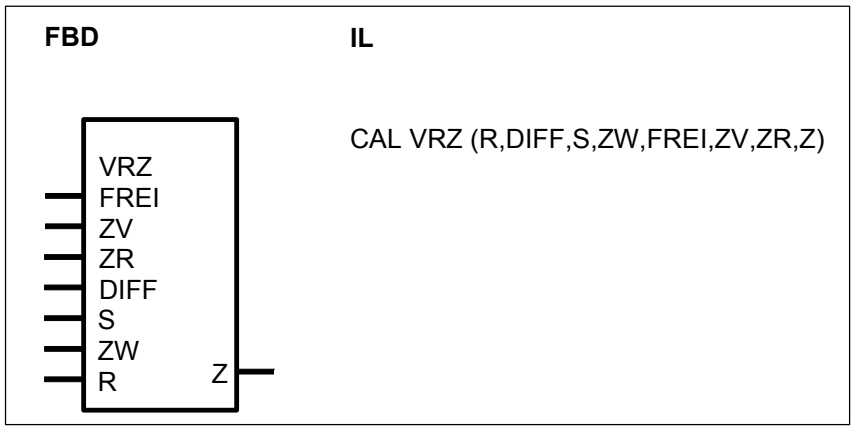
Q BINARY
The overflow is specified at the input Q.
Q = 1 when CV passes from -1 to 0 or from 0 to -1.

CV WORD
The current counter value is available at the output CV.

CATV WORD
The counter value when CATCH =1 is available at the output CATV.

Number range
Integer word (16 bits)
- Low limit : 8000_H - 32768
- High limit : $7FFF_H$ + 32767

VRZ UP / DOWN COUNTER



PARAMETERS

FREI	BINARY	%I, %M, %O, %K, %S	Enable block processing
ZV	BINARY	%I, %M, %O, %K, %S	Pulse input, up counting
ZR	BINARY	%I, %M, %O, %K, %S	Pulse input, down counting
DIFF	WORD	%IW, %MW, %OW, %KW	Counter content change per positive edge (increment)
S	BINARY	%I, %M, %O, %K, %S	Set counter to an intermediate value
ZW	WORD	%IW, %MW, %OW, %KW	Intermediate value
R	BINARY	%I, %M, %O, %K, %S	Reset counter
Z	WORD	%OW, %MW	Output for counter content

DESCRIPTION

This function block serves to count pulses. When counting enabled (FREI = 1), the positive edge of the pulse is evaluated in each case. The counter is capable of counting both up and down and the counting increment can be specified. It is possible to preset the counter content to an intermediate value.

FREI BINARY
Counting is enabled or disabled by means of the FREI input. The following applies :
FREI = 0 -> Counting disabled
FREI = 1 -> Counting enabled

ZV BINARY
Each positive edge (0->1 edge) at the input ZV increases the current counter content by the increment specified at the DIFF input.

ZR BINARY
Each positive edge (0->1 edge) at the input ZR decreases the current counter content by the increment specified at the DIFF input.

DIFF WORD
The increment for the counting operation is specified at the DIFF input. The increment is the value by which the counter is changed at the input ZV or ZR with each positive edge.

S BINARY
By means of a 1 signal at the input S, the counter content is set to the value specified at the input ZW. Counting is blocked as long as 1 signal is present at the input S. Setting is also effective when a 1 signal is present at the FREI input.

ZW WORD
The value to which the counter content is set by a 1 signal at the input S is specified at the input ZW.

R BINARY
A 1 signal at the input R sets the counter content to the value 0. The reset input R has the highest priority of all inputs.

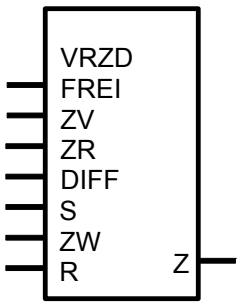
Z WORD
The current counter content is available at the output Z.
If the counter reaches the positive or negative limit of the number range, the counter is limited to this value.

Number range
Integer word (16 bits)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

Note for serie 90 :
The output is limited to -32767.
The value -32768 is an inadmissible value for word inputs.

VRZD UP / DOWN COUNTER, DOUBLE WORD

FBD



IL

```
CAL VRZD (R,DIFF,S,ZW,FREI,ZV,ZR,Z)
```

PARAMETERS

FREI	BINARY	%I, %M, %O, %K, %S	Enable block processing
ZV	BINARY	%I, %M, %O, %K, %S	Pulse input, up counting
ZR	BINARY	%I, %M, %O, %K, %S	Pulse input, down counting
DIFF	DOUBLE WORD	%MD, %KD	Counter content change per positive edge (increment)
S	BINARY	%I, %M, %O, %K, %S	Set counter to an intermediate value
ZW	DOUBLE WORD	%MD, %KD	Intermediate value
R	BINARY	%I, %M, %O, %K, %S	Reset counter
Z	DOUBLE WORD	%MD	Output for counter content

DESCRIPTION

This function block serves to count pulses. When counting enabled (FREI = 1), the positive edge of the pulse is evaluated in each case. The counter is capable of counting both up and down and the counting increment can be specified. It is possible to preset the counter to an intermediate value.

FREI BINARY

Counting is enabled or disabled by means of the FREI input. The following applies :

FREI = 0 -> Counting disabled
FREI = 1 -> Counting enabled

ZV BINARY

Each positive edge (0->1 edge) at the input ZV increases the current counter content by the increment specified at the DIFF input.

ZR BINARY

Each positive edge (0->1 edge) at the input ZR decreases the current counter content by the increment specified at the DIFF input.

DIFF DOUBLE WORD
The increment for the counting operation is specified at the DIFF input. The increment is the value by which the counter is changed at the input ZV or ZR with each po–sitive edge.

S BINARY
By means of a 1 signal at the input S, the counter content is set to the value specified at the input ZW. Counting is blocked as long as 1 signal is present at the input S. Setting is also effective when a 1 signal is present at the FREI input.

ZW DOUBLE WORD
The value to which the counter content is set by a 1 signal at the input S is specified at the input ZW.

R BINARY
A 1 signal at the input R sets the counter content to the value 0. The reset input R has the highest priority of all inputs.

Z DOUBLE WORD
The current counter content is available at the output Z.
If the counter reaches the positive or negative limit of the number range, the counter is limited to this value.

Number range
Integer double word (32 bits)

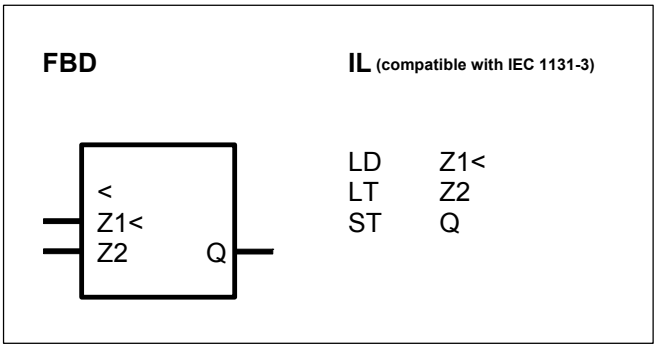
• Low limit :	$8000\ 0001_H$	-2 147 483 647
• High limit :	$7FFF\ FFFF_H$	+2 147 483 647
• Inadmissible value :	$8000\ 0000_H$	---

All blocks for double word arithmetic subject values to be processed for admissibility. If the inadmissible value occurs, it is corrected to the admissible value $8000\ 0001_H$ (-2 147 483 647).

2.4 Comparison functions, word

Comparison functions, word	from pages C-39 to C-48	serie C[tler]	40	50	90	30
<	Less than	x	x	x	x	x
<=	Less than or equal	x	x	x	x	x
<>	Unequal	x	x	x	x	x
=?	Equal	x	x	x	x	x
>	Greater than	x	x	x	x	x
>=	Greater than or equal to	x	x	x	x	x
VGL3P	Comparator with 3-point response				x	
VGLEH	Comparator with unilateral hysteresis				x	
VGLUH	Comparator with asymmetrical hysteresis				x	

< LESS THAN



PARAMETERS

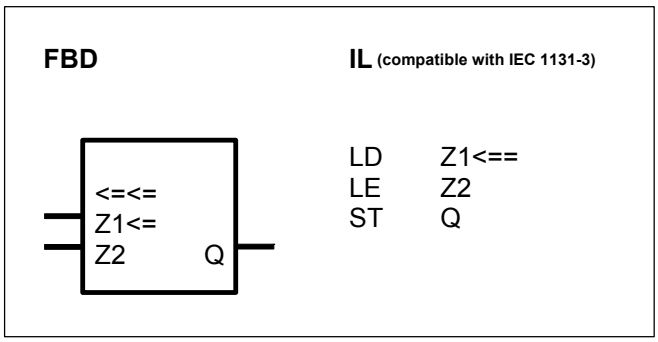
Z1<	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1< is compared to the value of the operand at the input Z2.
If the value at Z1< is less than the one at Z2, the state 1 is allocated to the operand at the output Q. The state 0 is allocated to Q if Z1< is equal to or greater than Z2.

Number range
Integer word (16 Bit)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

<= LESS THAN OR EQUAL



PARAMETERS

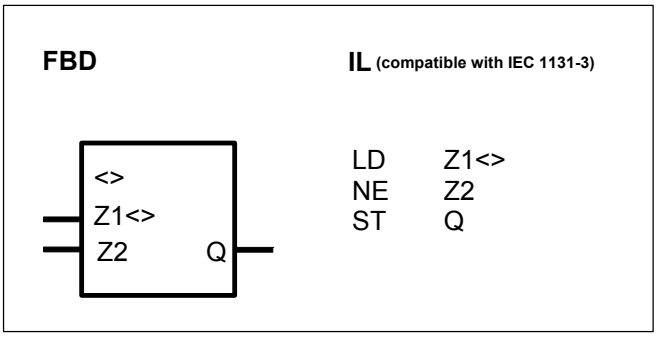
Z1<=	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1<= is compared to the value of the operand at the input Z2.
The state 1 is allocated to the operand at the output Q if the value at Z1<= is less than or equal to the value at Z2. The state 0 is allocated to Q if Z1<= is greater than Z2.

Number range
Integer word (16 Bit)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

<> UNEQUAL



PARAMETERS

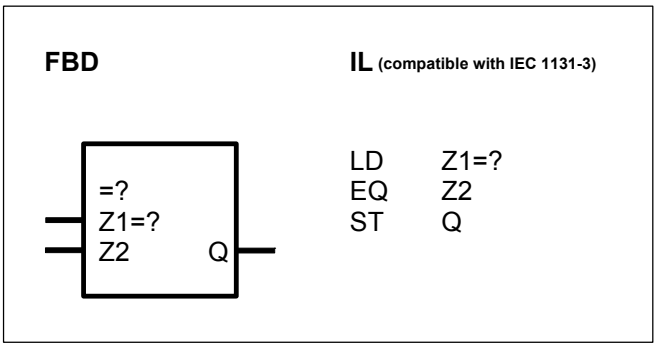
Z1<>	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1<> is compared to the value of the operand at the input Z2.
If the vaue at Z1<> is greater than or less than the one at Z2, the state 1 is allocated to the operand at the output Q. The state 0 is allocated to Q if Z1<> is equal to Z2.

Number range
Integer word (16 bits)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

=? EQUAL



PARAMETERS

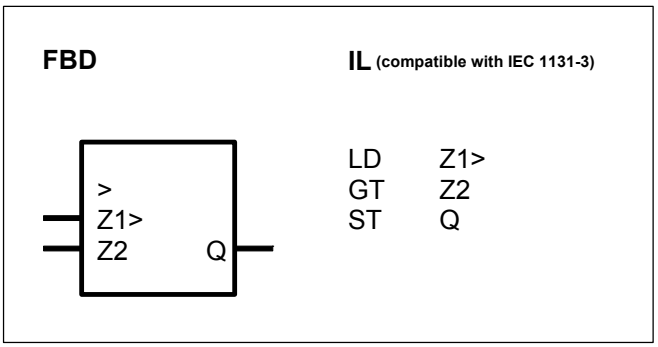
Z1=?	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1=? is compared to the value of the operand at the input Z2. If the value at Z1=? is equal to the one at Z2, the state 1 is allocated to the operand at the output Q. The state 0 is allocated to Q if Z1=? is unequal to Z2.

Number range
Integer word (16 Bit)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

> GREATER THAN



PARAMETERS

Z1>	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

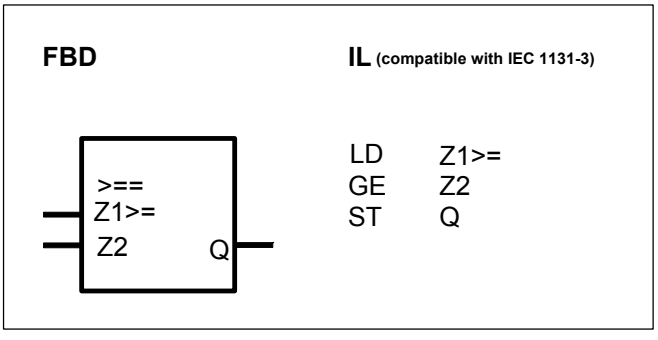
The value of the operand at the input Z1> is compared to the value of the operand at the input Z2. If the value at Z1> is greater than the one at Z2, the state 1 is allocated to the operand at the output Q. The state 0 is allocated to Q if Z1> is equal to or less than Z2.

Number range

Integer word (16 Bit)

- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

>= GREATER THAN OR EQUAL TO



PARAMETERS

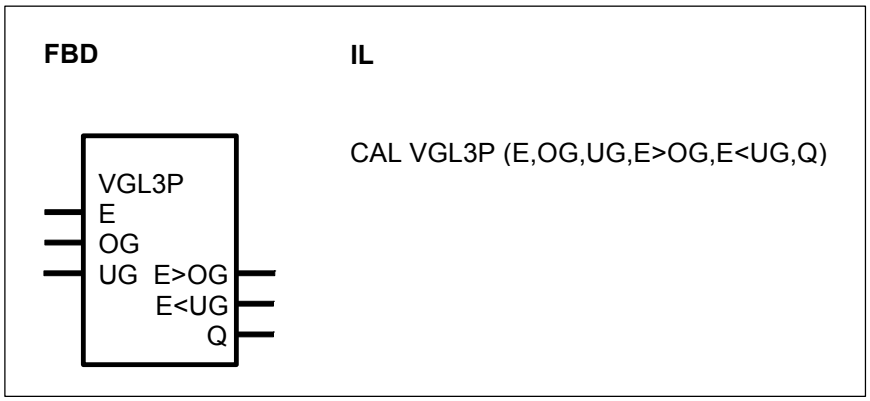
Z1>=	WORD	%IW, %MW, %OW, %KW	Value to be compared
Z2	WORD	%IW, %MW, %OW, %KW	Comparison value
Q	BINARY	%O, %M, %S	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1>= is compared to the value of the operand at the input Z2. The state 1 is allocated to the operand at the output Q if the value at Z1>= is greater than or equal to the value at Z2. The state 0 is allocated to Q if Z1>= is less than Z2.

Number range
Integer word (16 Bit)
- low limit : 8000_H (-32768)
- high limit : $7FFF_H$ (+32767)

VGL3P COMPARATOR WITH 3-POINT RESPONSE



PARAMETERS

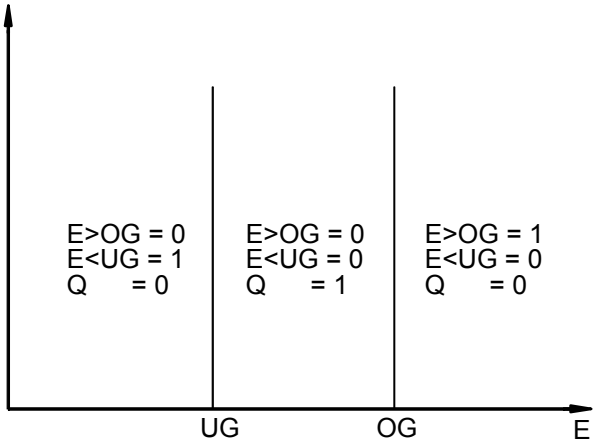
E	WORD	%IW, %MW, %OW, %KW	Input value
OG	WORD	%IW, %MW, %OW, %KW	High limit
UG	WORD	%IW, %MW, %OW, %KW	Low limit
E>OG	BINARY	%O, %M	Value > high limit
E<UG	BINARY	%O,%M	Value < low limit
Q	BINARY	%O,%M	Low limit ≤ input limit value ≤ high

DESCRIPTION

The value of the operand at the input E is compared to the values of the operands at the inputs OG and UG.
The possible results are signalled at the outputs E>OG, E<UG and Q.

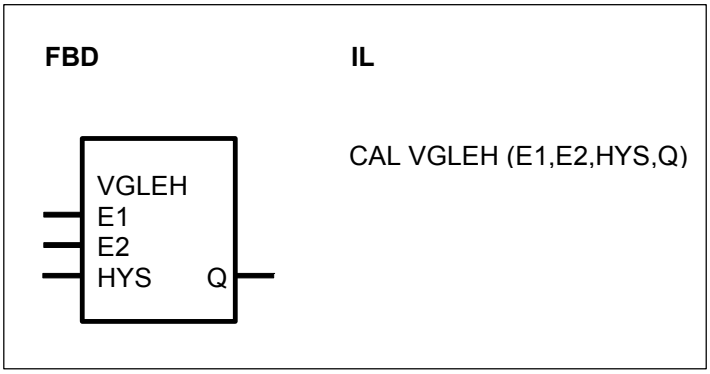
The following applies:

E < UG	---> E>OG = 0, E<UG = 1, Q = 0
UG ≤ E ≤ OG	---> E>OG = 0, E<UG = 0, Q = 1
E > OG	---> E>OG = 1, E<UG = 0, Q = 0



Number range
Integer word (16 bits)
- Low level : 8000_H -32768
- High level : $7FFF_H$ +32767

VGLEH COMPARATOR WITH UNILATERAL HYSTERESIS



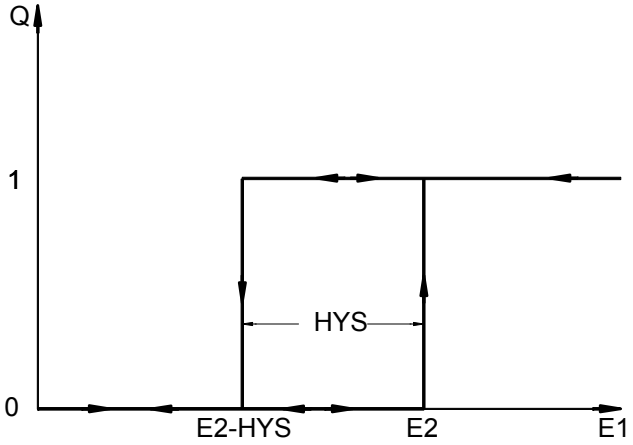
PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input value 1
E2	WORD	%IW, %MW, %OW, %KW	Input value 2
HYS	WORD	%IW, %MW, %OW, %KW	Hysteresis
Q	BINARY	%O, %M	Result of the comparison

DESCRIPTION

The values of the operands at the inputs E1 and E2 are compared to each other. Taking the hysteresis at the input HSY into account, the result is signalled at the output Q.

The following applies :
E1 ≥ E2 ---> Q = 1
E1 < E2 - HYS ---> Q = 0
E2 - HYS ≤ E1 < E2 ---> Q as in the previous cycle



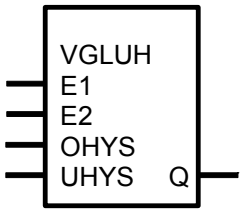
HYS = Hysteresis

Number range
Integer word (16 bits)
- Low limit : 8001_H -32767
- High limit : $7FFF_H$ +32767
- Inadmissible value : 8000_H ---

The following especially applies here to the specification for the left edge of the hysteresis : E2 - HYS ≥ -32 767 (8001_H)

VGLUH COMPARATOR WITH ASYMMETRICAL HYSTERESIS

FBD



IL

```
CAL VGLUH (E1,E2,OHYS,UHYS,Q)
```

PARAMETERS

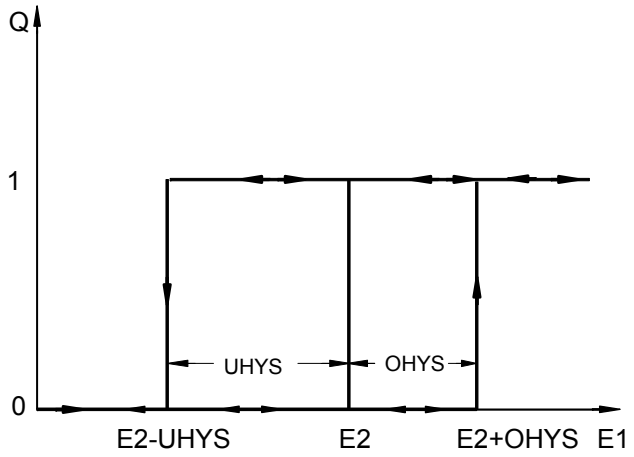
E1	WORD	%IW, %MW, %OW, %KW	Input value 1
E2	WORD	%IW, %MW, %OW, %KW	Input value 2
OHYS	WORD	%IW, %MW, %OW, %KW	High hysteresis
UHYS	WORD	%IW, %MW, %OW, %KW	Low hysteresis
Q	BINARY	%O, %M	Output

DESCRIPTION

The values of the operands at the inputs E1 and E2 are compared to each other. Taking the hystereses at the inputs OHYS (high hysteresis) and UHYS (low hysteresis) into account, the result is signalled at the output Q.

The following applies :

E1 < E2 - UHYS ---> Q = 0
E1 ≥ E2 + OHYS ---> Q = 1
E2-UHYS ≤ E1 ≤ E2+OHYS ---> Q as in the previous cycle



Number range

Integer word (16 bits)

- Low limit : 8001_H -32767
- High limit : $7FFF_H$ +32767
- Inadmissible value : 8000_H ---

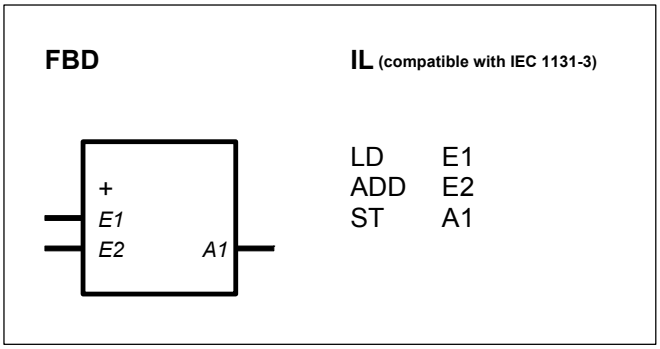
The following especially applies here :

E1 ≥ 8000_H (-32768)
E2 - UHYS ≥ 8001_H (-32767)

2.5 Arithmetic functions, word

Arithmetic functions, word	from pages C-48 to C-60	serie C[tler]	40	50	90	30
+	Addition	x	x	x	x	x
-	Subtraction	x	x	x	x	x
*	Multiplication	x	x	x	x	x
DIV	Division	x	x	x	x	x
*: / MULDI	Multiplication with division	x	x	x	x	x
=W	Allocation	x	x	x	x	x
BETR	Absolute value generator		x	x	x	x
COS1	Cosinus				x	
MUL2N	Multiplication by 2 to the power of N		x	x	x	x
NEG	Negation		x	x	x	x
SIN1	Sinus				x	
SQRT	Square root		x	x	x	
ZUDKW	Allocation direct constant to word variable		x	x	x	x

+ ADDITION



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Summand 1
E2	WORD	%IW, %MW, %OW, %KW	Summand 2; **duplicable**
A1	WORD	%MW, %OW	Total

DESCRIPTION

The values of the operands at the inputs are added and the result is allocated to the operand at the output.

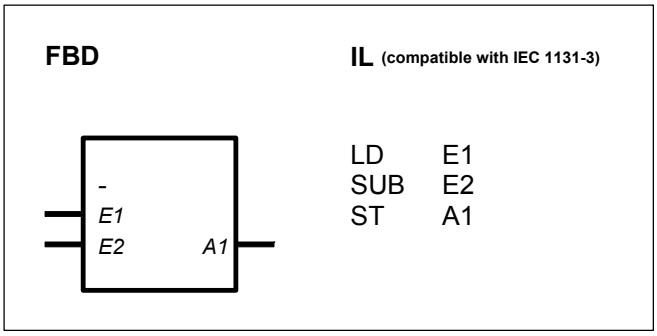
Number range

Integer Word (16 Bits)

- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 :
The output is limited to -32767.
The value -32768 is an inadmissible value for the input E1. If the value 8000_H (-32768) is present at E1, it is automatically corrected to the permissible value 8001_H (-32767) before processing.

- SUBTRACTION



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Minuend
E2	WORD	%IW, %MW, %OW, %KW	Subtrahend; **duplicable**
A1	WORD	%MW, %OW	Result (difference)

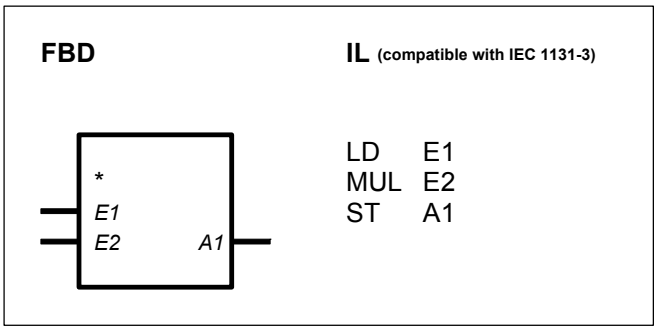
DESCRIPTION

The value of the operand at the input E2 is subtracted from the value of the operand at the input E1 and the result is allocated to the operand at the output A1.
The input E2 is capable of duplication (E2...En). If it is duplicated, all values of the operands at the inputs E2...En are subtracted from the operand at the input E1.

Number range
Integer Word (16 Bits)
- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 :
The output is limited to -32767.
The value -32768 is an inadmissible value for the input E1. If the value 8000_H (-32768) is present at E1, it is automatically corrected to the permissible value 8001_H (-32767) before processing.

* MULTIPLICATION



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Multiplicand
E2	WORD	%IW, %MW, %OW, %KW	Multiplier; **duplicable**
A1	WORD	%MW, %OW	Result (Product)

DESCRIPTION

The values of the operands at the inputs of this connection element are multiplied by each other and the result is allocated to the operand at the output.

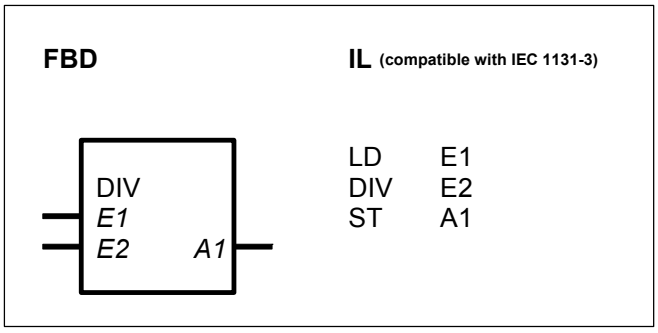
Number range

Integer Word (16 Bits)

- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 : The output is limited to -32767.

DIV DIVISION



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Dividend
E2	WORD	%IW, %MW, %OW, %KW	Divisor
A1	WORD	%MW, %OW	Result (quotient)

DESCRIPTION

The value of the operand at the input E1 is divided by the value of the operand at the input E2 and the result is allocated to the operand at the output A1. The following applies if input E2 is duplicated : E1 : E2 : E3 ... : En = A1. The result of division is always a whole number. Rounding (inclusion of the digits after the decimal point) does not take place.

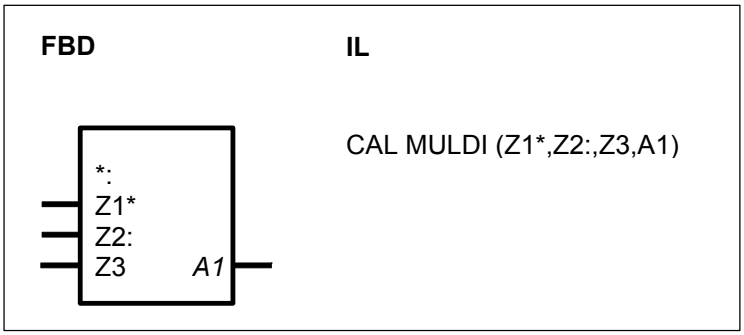
Number Range

Integer Word (16 Bits)

- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 : The output is limited to -32767.

*: MULTIPLICATION WITH DIVISION



PARAMETERS

Z1*	WORD	%IW, %MW, %OW, %KW	Multiplicand
Z2:	WORD	%IW, %MW, %OW, %KW	Multiplier
Z3	WORD	%IW, %MW, %OW, %KW	Divisor
A1	WORD	%MW,%OW	Result

DESCRIPTION

The value of the operand at the input Z1 is multiplied by the value of the operand at the input Z2, the intermediate result is divided by the value of the operand at the input Z3 and then the result is allocated to the operand at the output A1.

Internally, this function block operates with double word accuracy (32 bits) when multiplying and dividing. Only when allocating the result to the output A1 is the value limited to word accuracy (16 bits). The result is rounded up if the remainder of division is $\geq$ 0.5. If a number overflow occurs during division (e.?g. division by 0), the limit value of the number range is allocated to the output A1 with the correct sign.

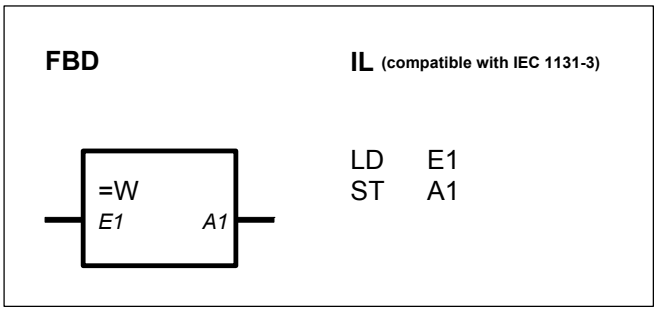
Number range

Integer Word (16 Bits)

- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 :
The output is limited to -32767.
The value -32768 is an inadmissible value for the word inputs. If the value 8000_H (-32768) is present at an input, it is automatically corrected to the permissible value 8001_H (-32767) before processing.

=W ALLOCATION



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Source
A1	WORD	%MW, %OW	Target

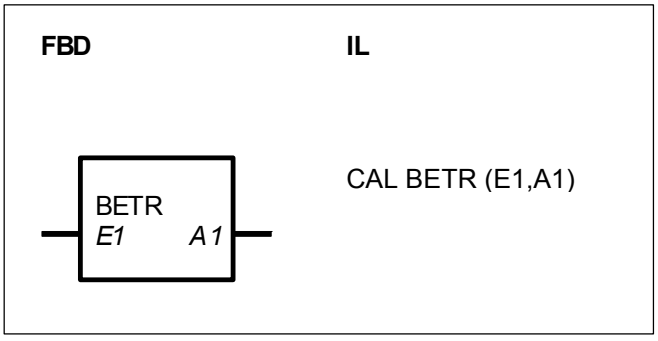
DESCRIPTION

This connection element allocates the value of the operand at the input E1 to the operand at the output A1.

Number range
Integer word (16 bits).
- Low limit : -32768
- High limit : 32767

Note for serie 90 : If the value 8000_H (-32768) is present at the input E1, the permissible value 8001_H (-32767) is allocated to the output A1.

BETR ABSOLUTE VALUE GENERATOR



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input value
A1	WORD	%MW, %OW	Absolute value of the input value

DESCRIPTION

The absolute value of the word operand at the input E1 is generated and is allocated to the operand at the output A1.

Number range

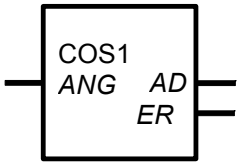
Integer Word (16 Bits)

- Low limit : 8000_H -32768
- High limit : $7FFF_H$ +32767

Note for serie 90 : The output is limited to -32767.
The value -32768 is an inadmissible value for the word inputs. If the value 8000_H (-32768) is present at the input E1, the maximum possible value $7FFF_H$ (+32767) is allocated to the output A1.

COS1 COSINE

FBD



IL

```
CAL COS1 (ANG,AD,ERR)
```

PARAMETERS

ANG	WORD	%IW, %MW, %OW	Angle 0...3600 (0.0°....360.0°)
AD	DOUBLE WORD	%MD	Cosine of the input value
ERR	BINARY	%M, %O	Error if ANG is out of range

DESCRIPTION

The function block calculates the cosine of the value at the input ANG and assigns it to the output AD.

The input value has to be in the following range : $0 \leq ANG \leq 3600$

0000 -> 0°
0001 -> 0.1°
0010 -> 1.0°

...
3600 -> 360.0°

If the value at ANG is negative or greater than 3600 (360°), the output AD is set to 0 and the error output ERR is set to 1.

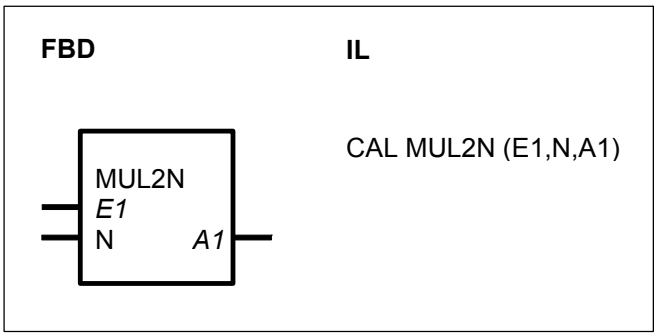
0 ≤ ANG ≤ 3600 => ERR = 0 and AD = cos (ANG)
ANG < 0 or ANG > 3600 => ERR = 1 and AD = 0

The result is within the range of -100 000 to +100 000.

Examples :

cos (0) = 100000
cos (450) = 70711
cos (900) = 0
cos (1800) = -100000
cos (2360) = -55919
cos (2700) = 0
cos (3600) = 100000

MUL2N MULTIPLICATION BY 2 TO THE POWER OF N



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input operand
N	WORD	%IW, %MW, %OW, %KW	Quantity
A1	WORD	%MW, %OW	Result

DESCRIPTION

The value of the operand at the input E1 is shifted bit-by-bit N times.

If the value at the input N is positive, the value is shifted to the left. For each shift by 1 bit position, this corresponds to multiplication of the current value by 2.

If the value at the input N is negative, it is shifted to the right. For each shift by 1 bit position, this corresponds to division of the current value by 2.

If N = 0, the value at the input E1 is passed directly to the output A1.

The result is allocated to the operand at the output A1.

Meaningful range for N: $-14 \leq N \leq +14$

Sign of the value at the input E1:
The sign of the value E1 is not influenced by the shift operation. That is to say, the sign of the output value is always identical with the sign of the input value.

Shift to the left (Multiplication):
When the value at the input is shifted to the left, the released bit 0 is filled with 0. The sign bit (bit 15) is not changed because limiting to the limit of the number range takes place beforehand.

Limiting of the value at the output A1 when shifting to the left:

• The following applies to positive values at the input E1:
If bit 14 has a “1” and if shift operations still have to be carried out on the basis of the value at the input N, these are no longer executed. Instead, the output is set to the limit of the positive number range. That is to say, the limit has been reached in any case at the latest after 14 shifts.
Limit value: Output A1 = +32767 ($7FFF_H$).

• The following applies to negative values at the input E1:
If bit 14 has a “0” and shift operations still have to be carried out because of the value at the input N, these will no longer be executed. Instead, the output is set to the limit of the negative number range. That is to say, the limit has been reached in any case at the latest after 14 shifts.
Limit: Output A1 = -32767 (8001_H)

Shift to the right (Division):
When shifting to the right, every bit moves to the right by one position. At the same time, the sign bit (bit 15) always retains its value. The released bit (bit 14) is filled in each case with the value of the sign bit.

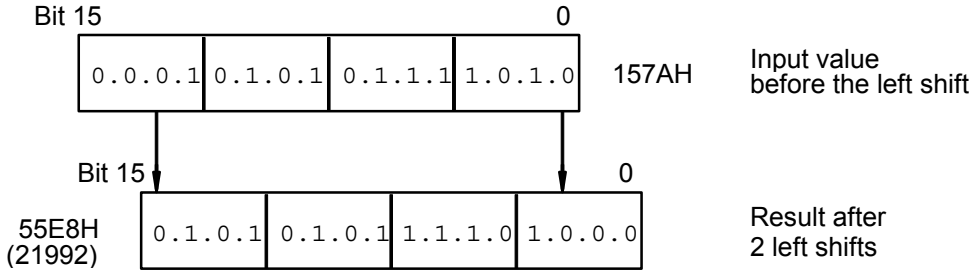
Limiting of the value at the output when shifting to the right:

• The following applies to positive values at the input E1:
If now only bit 0 has a “1” and shift operations still have to be carried out because of the value at the input N, the output will be set to the value 0. That is to say, the value 0 has been reached in any case at the latest after 14 shifts.
Output A1 = 0.

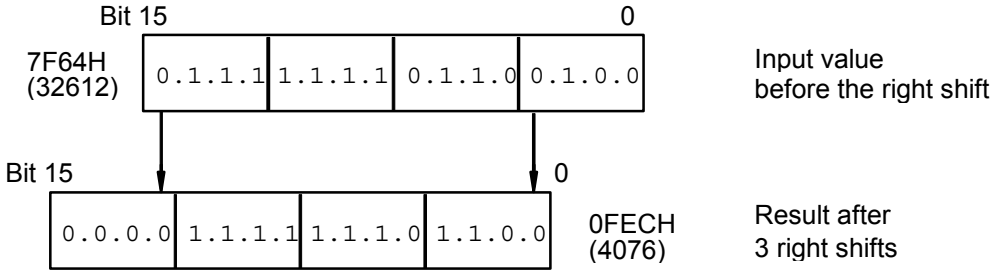
• The following applies to negative values at the input E1:
If bit 0 ... bit 15 has a “1” as the result of the shift, the limit value (-1) has been reached. Further shifts have no effect. That is to say, the value -1 has been reached at the latest after 15 shifts.
Output A1 = -1 ($FFFF_H$).

Examples:

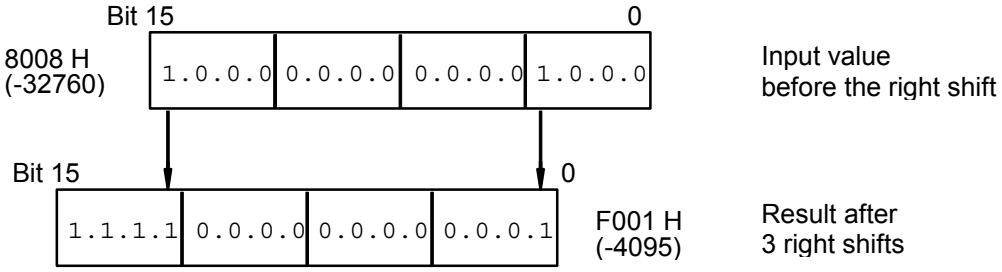
1. Input value E1 = 5498 ($157A_H$)
   Exponent N = 2 -> 2 * Left shift



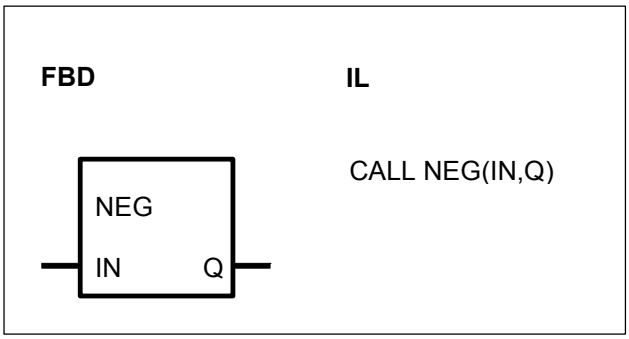
2. Input value E1 = 32612 ($7F64_H$)
   Exponent N = -3 -> 3 * Right shift



3. Input value E1 = -32612 (8008_H)
   Exponent N = -3 -> 3 * Right shift



NEG NEGATION



PARAMETERS

IN	WORD	%IW, %MW, %OW, %KW	Word value
Q	WORD	%OW, %MW	Negation

DESCRIPTION

The NEG fonction block allocates the negation of the variable at the input IN to the output Q.

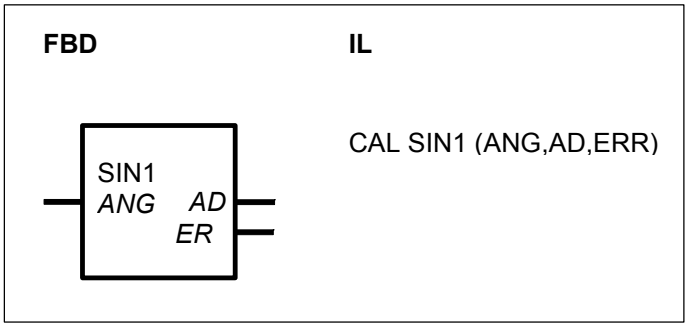
Number range

Integer word (16 bits)

- Low limit : 8000_H - 32768
- High limit : $7FFF_H$ + 32767

Note : if IN = -32768 then Q = +32767.

SIN1 SINE



PARAMETERS

ANG	WORD	%IW, %MW, %OW	Angle 0...3600 (0.0°....360.0°)
AD	DOUBLE WORD	%MD	Sine of the input value
ERR	BINARY	%M, %O	Error if ANG is out of range

DESCRIPTION

The function block calculates the sine of the value at the input ANG and assigns it to the output AD.

The input value has to be in the following range : $0 \leq ANG \leq 3600$

0000 -> 0°
0001 -> 0.1°
0010 -> 1.0°
...
3600 -> 360.0°

If the value at ANG is negative or greater than 3600 (360°), the output AD is set to 0 and the error output ERR is set to 1.

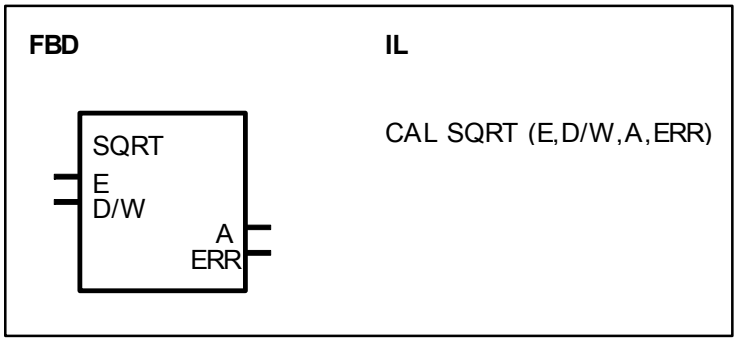
$0 \leq ANG \leq 3600$ => ERR = 0 and AD = cos (ANG)
ANG < 0 or ANG > 3600 => ERR = 1 and AD = 0

The result is within the range of -100.000 to +100.000.

Examples :

sin (0) = 0
sin (450) = 70711
sin (900) = 100000
sin (1800) = 0
sin (2360) = -82904
sin (2700) = -100000
sin (3600) = 0

SQRT SQUARE ROOT



PARAMETERS

E	WORD, DOUBLE WORD	%IW, %OW, %MW, %KW, %MD, %KD	Input
D/W	BINARY	%I, %O, %M, %K, %S	Format of the input/output
A	WORD, DOUBLE WORD	%OW, %MW, %MD, %KD	Square root of the input value
ERR	BINARY	%O, %M	Error if the input value is negative

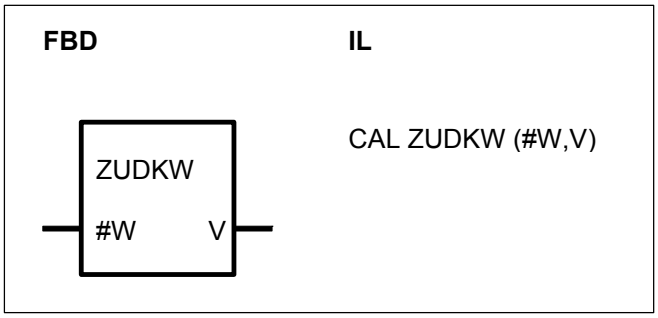
DESCRIPTION

The function block SQRT generates the square root of the value at the input E. The result is available at the output A and is always *rounded down* to an integral number.

The value at the input E must be a positive number. If the value at the input is negative, the value 'zero' is output through the output A and the value '1' is output through the ERR output. The ERR output indicates whether the value of the input operand E is positive (≥ 0) or negative (< 0).
Input E ≥ 0 -> ERR = 0 and A = square root
Input E < 0 -> ERR = 1 and A = 0

The input D/W defines the format of the input/output operand.
D/W = 0 -> WORD
D/W = 1 -> DOUBLE WORD

ZUDKW ALLOCATION DIRECT CONSTANT TO WORD VARIABLE



PARAMETERS

#W	DIRECT CONSTANT	#, #H	Numerical value which is to be allocated to the word variable at output V
V	WORD	%MW, %OW	Word variable to which the numerical value of input # is to be allocated

DESCRIPTION

The function block serves to allocate a numerical value to a word variable. The numerical value is specified as a direct constant.

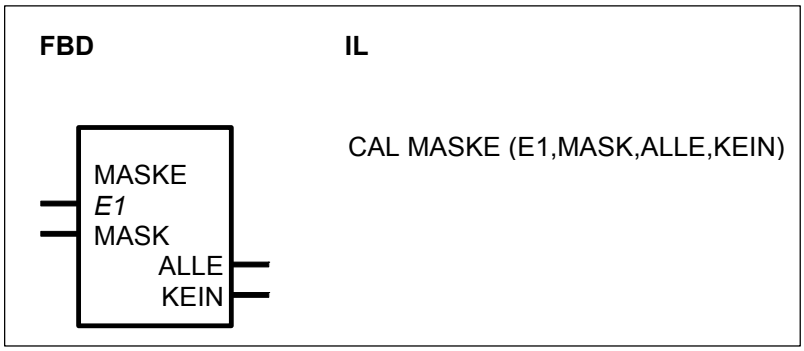
#W DIRECT CONSTANT (#, #H)
This numerical value is allocated to the word variable at output V.

V WORD
Word variable to which the numerical value of input #W is allocated.

2.6 Logical functions, word

Logical functions, word	from pages C-60 to C-68	serie C[tler] 40	50	90	30
MASKE	Mask			x	
SHIFT	Shift block			x	
WAND	AND combination, word	x	x	x	x
WOR	OR combination, word	x	x	x	x
WXOR	Exclusive OR combination, word	x	x	x	x

MASKE MASK



PARAMETERS

E1	WORD	%IW, %OW, %MW, %KW	Input value
MASK	WORD	%IW, %OW, %MW, %KW	Mask
ALLE	BINARY	%O, %M	All bits agree
KEIN	BINARY	%O, %M	No bit agrees

DESCRIPTION

The individual bits of the operand at the input E1 are compared to the bits of the operand at the input MASK. The result of the comparison is signalled at the outputs ALLE and KEIN.

If at least *all* bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 1
KEIN = 0

If *none* of the bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 0
KEIN = 1

If only *some* of the bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 0
KEIN = 0

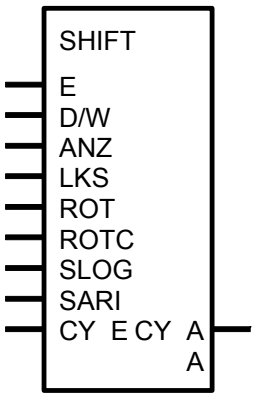
Example

```
E1   : X1111XX11XXXX11X  ALLE = 1
MASK : 0111100110000110  KEIN = 0

E1   : 00XXXXX0XXX0XXXX  ALLE = 0
MASK : 1100000100010000  KEIN = 1

E1   : 1X1X1XXX10XXXX10  ALLE = 0
MASK : 0010100011000011  KEIN = 0
```

X: These bits may have any value (don't care)

SHIFT SHIFT BLOCK

FBD



IL

```
CAL   SHIFT  (E,D/W,ANZ,LKS,ROT,ROTC,
             SLOG,SARI,CY E,CY A,A)
```

PARAMETERS

E	WORD, DOUBLE WORD	%IW, %OW, %MW, %KW, %KD, %MD	Operand to be shifted Word or double word
D/W	BINARY	%I, %O, %M, %K, %S	Format selection (W or DW)
ANZ	WORD	%IW, %OW, %MW, %KW	Number of bit positions to be shifted
LKS	BINARY	%I, %O, %M, %K, %S	Shift direction, left or right
ROT	BINARY	%I, %O, %M, %K, %S	Shift-Art: ROTATE
ROTC	BINARY	%I, %O, %M, %K, %S	Shift-Art: ROTATE by the CARRY FLAG
SLOG	BINARY	%I, %O, %M, %K, %S	SHIFT type: LOGICAL SHIFT
SARI	BINARY	%I, %O, %M, %K, %S	SHIFT type: ARITHMETIC SHIFT
CY_E	BINARY	%I, %O, %M, %K, %S	Initial value for the CARRY FLAG in the case of shift type ROTC
CY_A	BINARY	%O, %M	Status of the CARRY FLAG after the shift

A	WORD, DOUBLE WORD	%OW, %MW, %MD	Result of the shift

DESCRIPTION

This function block shifts the operand present at the input by a specified number of bit positions to the left or right. The result of the shift and the CARRY flag are available at the block's outputs. The operand at the input remains unchanged.

The required shift type is planned at the inputs :

- ROT
- ROTC
- SLOG
- SARITH

If several shift types are specified simultaneously, the shift type located furthest to the front in the sequence of the block's inputs applies.

The block can shift both word and double word operands.

Important: In certain circumstances, the value 8000_H or $8000\ 0000_H$ forbidden for arithmetic operations may be present at this block's output.

E WORD/DOUBLE WORD
Operand to be shifted
The planned SHIFT operation is applied to the input operand, which is not changed.

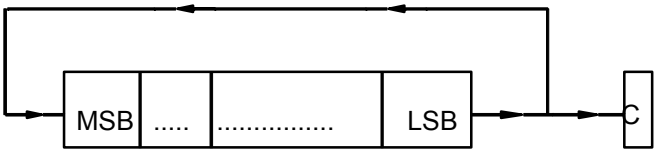
D/W BINARY
Format selection for the input operand
D/W = 0 -> WORD
D/W = 1 -> DOUBLE WORD

ANZ WORD
Number of bit positions to be shifte. Meaningful range for the quantity n :
Word operands : $0 \leq n \leq 16$
Double word operands :$0 \leq n \leq 32$

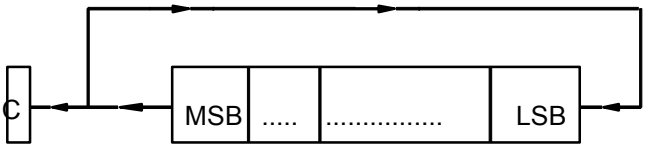
LKS BINARY
Direction in which shifting takes place
LKS = 0 -> shift to right
LKS = 1 -> shift to left

ROT BINARY
Shift type: ROTATE
The bit position released by the shift is replaced by the bit shifted out. The contents of the CARRY FLAG are additionally replaced by the bit shifted out. After the shift, the result and the contents of the CARRY FLAG are available at the block's outputs.

ROT to right :



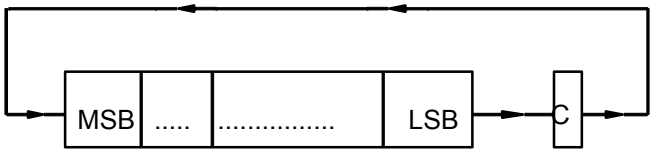
ROT to left :



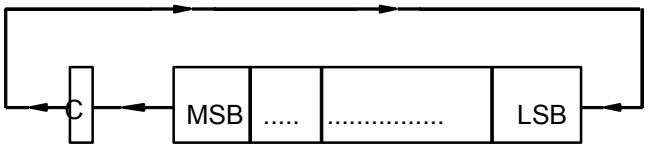
LSB : Least significant bit
MSB : Most significant bit

ROTC BINARY
Shift type: ROTATE by the CARRY FLAG
The bit position released by the shift is replaced by the contents of the CARRY FLAG. The CARRY FLAG is then replaced by the bit shifted out. After the shift, the result and the contents of the CARRY FLAG are available at the block's outputs.

ROTC to the right :



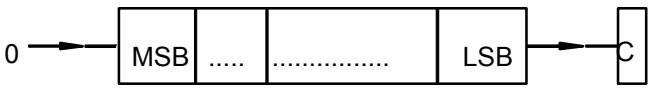
ROTC to the lef :



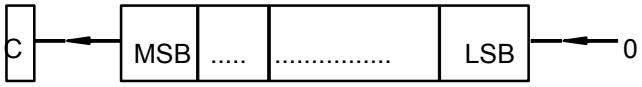
LSB : Least significant bit
MSB : Most significant bit

SLOG BINARY
Shift type: LOGICAL SHIFT
The bit position released by the shift is replaced by the value 0.
The contents of the CARRY FLAG are replaced by the bit shifted out.
After the shift, the result and the contents of the CARRY FLAG are available at the block's outputs.

SLOG to the right :



SLOG to the left :



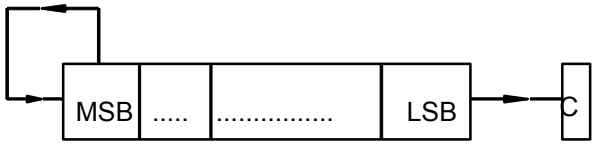
LSB : Least significant bit
MSB : Most significant bit

SARI BINARY
Shift type: ARITHMETIC SHIFT

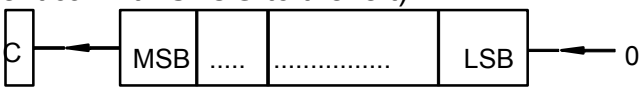
ARITHMETIC SHIFT to the right:
The bit position 15 (MSB, sign bit) released by the shift operation is replaced by itself. The contents of the CARRY FLAG are replaced by the bit shifted out. After the shift, the result and the contents of the CARRY FLAG are available at the block's outputs

ARITHMETIC SHIFT to the left (identical with SLOG left):
The bit position 0 released by the shift is replaced by the value 0. The contents of the CARRY FLAG are replaced by the bit shifted out. After the shift, the result and the contents of the CARRY FLAG are available at the block's outputs.

SARI to the right :



SARI to the left : (Identical with SLOG to the left)



LSB : Least significant bit
MSB : Most significant bit

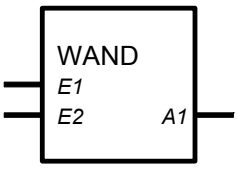
CY_E BINARY
Initial value for the CARRY FLAG with shift type ROTC. For the shift type "ROTATE by the CARRY FLAG", an initial value is needed for the CARRY FLAG. This initial value is specified at the CY_E input.

CY_A BINARY
State of the CARRY FLAG after the shift. After the shift, the current value of the CARRY FLAG is available at this output.

A WORD/DOUBLE WORD
Result of the shift. After the shift, the result is available at this output.

WAND AND COMBINATION, WORD

FBD



IL

```
CAL WAND (E1,E2,A1)
```

PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Operand 1
E2	WORD	%IW, %MW, %OW, %KW	Operand 2
A1	WORD	%MW, %OW	Result of the AND combination

DESCRIPTION

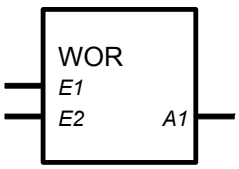
This function block generates the bit-by-bit AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

Example :

E1	0.0.0.0	0.0.1.1	0.0.1.0	0.1.1.0
E2	1.0.0.1	0.0.0.0	0.0.1.0	1.1.1.1
A1	0.0.0.0	0.0.0.0	0.0.1.0	0.1.1.0

WOR OR COMBINATION, WORD

FBD



IL

```
CAL WOR (E1,E2,A1)
```

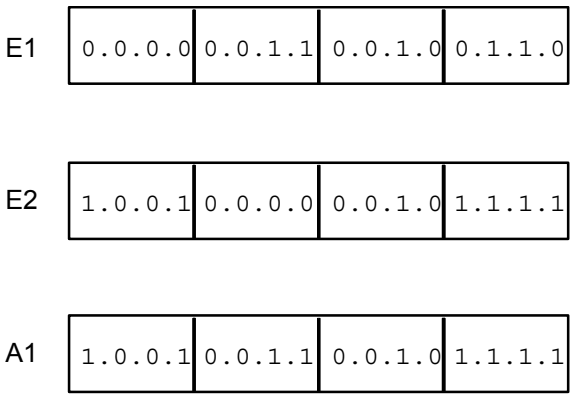
PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Operand 1
E2	WORD	%IW, %MW, %OW, %KW	Operand 2
A1	WORD	%MW, %OW	Result of the OR combination

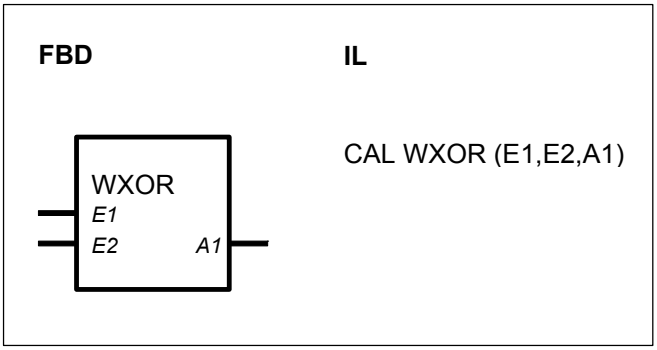
DESCRIPTION

This function block generates the bit-by-bit OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

Example :



WXOR EXCLUSIVE OR COMBINATION, WORD



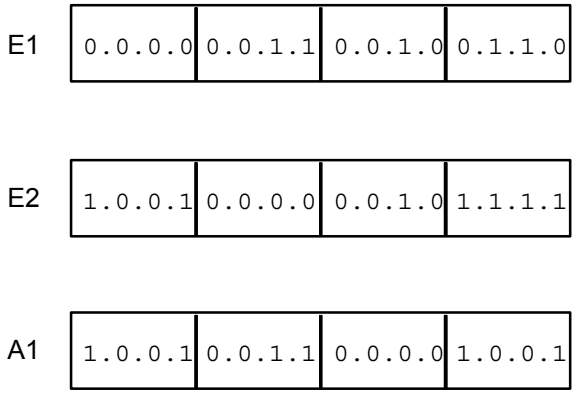
PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Operand 1
E2	WORD	%IW, %MW, %OW, %KW	Operand 2
A1	WORD	%MW, %OW	Result of the XOR combination

DESCRIPTION

This function block generates the bit-by-bit XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

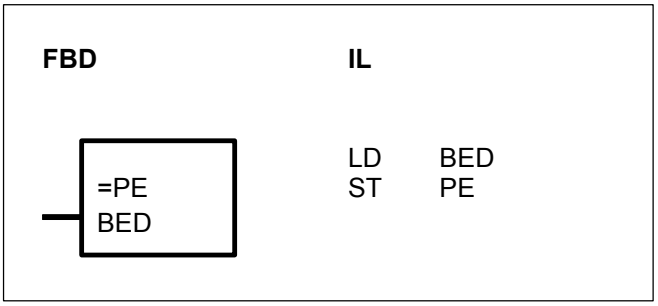
Example :



3 Program control functions

Program control functions	from pages C-68 to C-84	serie C[tler] 40	50	90	30
=PE	Conditional program end	x	x	x	x
ABORT	Program abort			x	
CAL_FB	Subroutine call		x		
CALLUP	Subroutine call for an assembler program			x	
DI	Read direct input	x	x		
DIN	Read direct inputs			x	
DO	Write direct output	x	x		
DOUT	Write direct outputs			x	
IOCON	Input/output configuration			x	
LZB	Run number block			x	
VTASK	Interrupt task validation	x	x		

=PE CONDITIONAL PROGRAM END



PARAMETERS

BED	BINARY	%I, %M, %O, %S, %K	Condition for program end

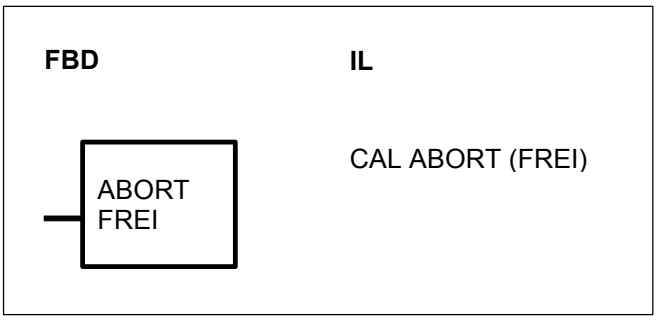
DESCRIPTION

Depending on the status of the operand at the input BED, processing of the PLC program is ended or is not ended here. :

BED = 0 : The PLC program is processed up to the end.

BED = 1 : The PLC program is processed only up to this point. The subsequent part of the PLC program is *not* processed.

ABORT PROGRAM ABORT



PARAMETERS

FREI	BINARY	%I, %M, %O, %K	Block enable

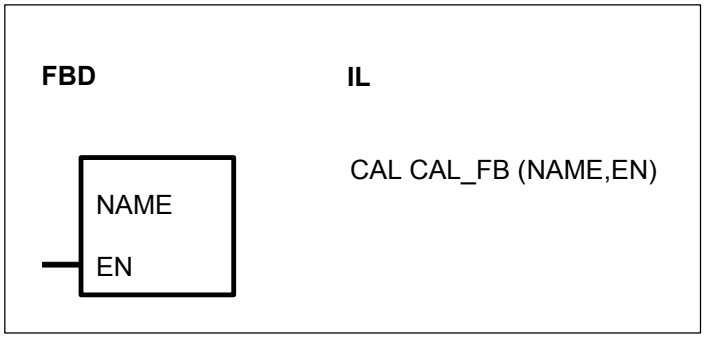
DESCRIPTION

The function block serves to abort a PLC program. All outputs are set to 0 if the PLC program is aborted.

The block can be used in particular for aborting the PLC program if a specific error of error class 3 occurs.

The function block is activated by a 1 signal at its enable input FREI.
If a 0 signal is applied to the enable input, the function block has no effect.
The following thus applies :
FREI = 0 : --> the block has no effect
FREI = 1 : --> the block aborts the program

CAL_FB SUBROUTINE CALL



PARAMETERS

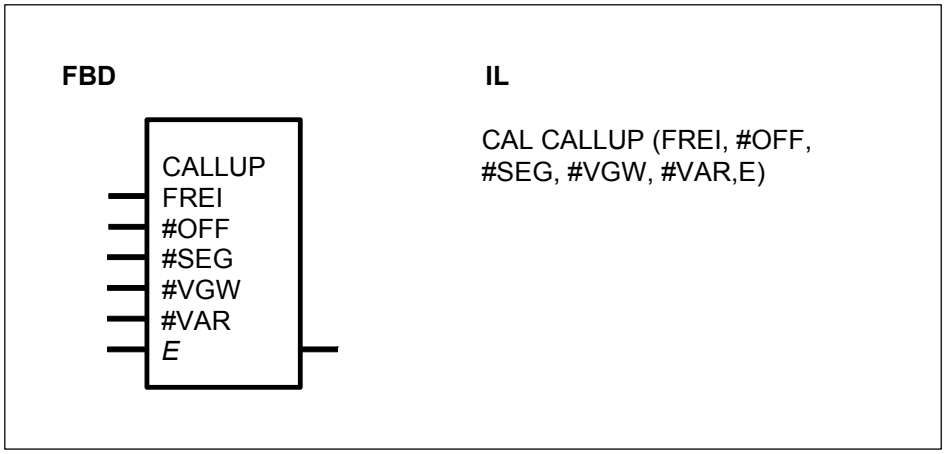
NAME	TXT		Subroutine name
EN	BINARY	%I, %M, %O, %K, %S	Call of the subroutine

DESCRIPTION

The CAL_FB function validates the execution of a subroutine whose name is specified in NAME variable

In function block diagram, the block has the name of the subroutine.
In instruction list, the function is CAL_FB with NAME and EN as parameters.

CALLUP SUBROUTINE CALL FOR AN ASSEMBLER PROGRAM



PARAMETERS

FREI	BINARY	%I, %M, %O, %K, %S	Block enable
#OFF	DIRECT CONSTANT	#, #H	Offset address of the subroutine
#SEG	DIRECT CONSTANT	#, #H	Segment address of the subroutine
#VGW	DIRECT CONSTANT	#, #H	Number of historical values of the subroutine
#VAR	DIRECT CONSTANT	#, #H	Number of input/output variables of the subroutine
E	X	%I, %M, %O, %K, %S, %IW, %MW, %OW, %KW, %MD, %KD	As an input variable;
		%M, %O, %MW, %OW %MD	As an output variable **capable of duplication**

DESCRIPTION

The function block calls a subroutine stored on the PLC in INTEL machine code. The machine code must be compatible with the processor of the relevant PLC.

After processing of the subroutine, the program branches back to the block. Before the subroutine is called, all registers are automatically saved and restored after the return.

FREI BINARY
Enable function block processing
FREI = 0 : Block is not processed
FREI = 1 : Block is processed

#OFF DIRECT CONSTANT
#SEG DIRECT CONSTANT
The start address of the assembler subroutine is specified at these two inputs. The start address consists of an offset and a segment address. The offset and segment addresses are specified as direct constants.

If the value “0” is specified as the segment address, the block expects the subroutine to have been stored in the user program memory. The offset address then represents the byte spacing between start of user program and start of subroutine. The subroutine is automatically saved as well when saving the PLC program on EPROM or flash EPROM.

#VGW DIRECT CONSTANT
The number of historical values used in the subroutine is specified at the input #VGW. This is specified as a direct constant.

#VAR DIRECT CONSTANT
The total number of input and output variables of the subroutine is specified at the input #VAR. Therefore, this is the total of all E1...En and A1...An. This is specified as a direct constant.

E BINARY, WORD, DOUBLE WORD
The input E is capable of duplication (E1...Em).
The input variables for the subroutine are specified at the inputs E1...Em. The subroutine performs “reading” access to these variables. When the subroutine accesses these variables, the assembler programmer must pay attention to the defined data format. Direct constants (value transfer) are not allowed at the inputs E1...Em.
The output variables for the subroutine are specified at the outputs Em+1...En. The subroutine performs “writing” access to these variables, i.e. values calculated in the subroutine are provided to the main program with the aid of the output variables Em+1...En. The assembler programmer must pay attention to the defined data format when accessing these variables.

Application notes

Store subroutine
The best possibility to store the subroutine is to store it in the user program memory (after PE). The subroutine can be loaded e. g. with a HEX file loader or the PLC monitor.
If the value “0” is specified as the segment address, the block expects the subroutine to have been stored in the user program memory. The offset address then represents the byte spacing between start of user program and start of subroutine.
The subroutine is automatically saved as well when saving the PLC program on EPROM or flash EPROM.

Loading the subroutine
The subroutine can be loaded into the PLC via the serial interface COM2 with monitor command “Read INTEL-HEX-FILE”. (See Description of the monitor functions, command “R”). This can be done only in connection with PLCs which have two serial

interfaces (e. g. 07 KT 92).

Subroutine creation

1. Register

When creating the subroutine, use can be made of the following registers to suit requirements :

AX, BX, CX, DX, BP

The following registers are intended for access to variables :

DS, ES, DI, SI

However, these registers can also be changed because they are corrected again after execution of the subroutine in the CALLUP block.

2. Start address

Subroutine's start address must be specified at the #OFF and #SEG inputs in the call of the block. The offset and segment are each specified as direct constants.

3. Access to input and output variables

Access to the variables is by way of the pointers ES:SI and DS:BX.
When the subroutine is entered, the pointer to the address of the first variable in the call of the block is located in (ES:SI) and the pointer to the address of the second variable is located in (ES:SI+2) etc. Access to the contents of these variables is then by means of DS:BX

Example 1 :
The values of the first two *word variables* in the block call are to be fetched to the registers AX and CX.

```
MOV   BX,ES:[SI]      ;Fetch address for 1st variable and store in BX
MOV   AX,DS:[BX]      ;fetch value to AX
MOV   BX,ES:[SI+2]    ;fetch address for second variable and store in BX
MOV   CX,DS:[BX]      ;fetch value to CX
```

Example 2 :
The value of the first two *binary variables* in the call of the block are to be fetched to the registers AL and CL.

```
MOV   BX,ES:[SI]      ;Fetch address for 1st variable and store in BX
MOV   AL,DS:[BX]      ;Fetch value to AL
MOV   BX,ES:[SI+2]    ;Fetch address for second variable and store in BX
MOV   CL,DS:[BX]      ;Fetch value to CL
```

Binary variables occupy a whole byte. This byte may only have 0 or 1 as its contents.

4. Access to historical values/RAM

If RAM space is needed in the subroutine for intermediate values, use can be made of the historical values memory.

The intermediate values are retained and are again available in the next program cycle for further processing. For each call of the subroutine, its own memory locations

are available in the historical values memory. If a subroutine is called three times, for instance, in total 3 times as many historical values are occupied in the historical values memory as are specified in the call. The number of memory words used (1 word = 16 bits) in the historical values memory must be specified as a direct constant in the call of the block. If too many historical values are occupied, the PLC issues an error message when the command PA (program editing) is used or when the program is started.

The historical values are addressed by way of the pointers DS:DI
In doing so, DS:DI points to the first available historical value when entering the subroutine.

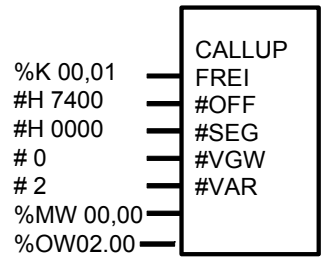
Example :
The value fetched to register AX in point 3. is to be compared to the first word in the historical values memory.

```
CMP   AX,DS :[DI]        ;Compare value to historical value
```

5. The stack has a depth of 40 words. In this way, up to 40 PUSH commands can be executed successively.

6. Return from the subroutine
The return is achieved by means of **RET FAR** (8086 machine code : CB). **RET FAR** is generated by the 8086 assembler by virtue of the fact that the subroutine is declared as **PROC FAR.**

FBD



IL

```
CAL CALLUP (%K00,01,#H7400,
#H0000, #0, #2, %MW00,00,
%OW02,00)
```

In a subroutine, the contents of the flag word %MW 01,00 are to be incremented and the contents of the flag word %MW 02,00 are to be decremented.

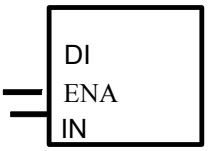
Subroutine

```
TEST   PROC  FAR

MOV    BX,ES:[SI]      ;address of %MW 01,00 to BX
MOV    AX,DS:[BX]      ;load value from %MW 01,00 to register AX
INC    AX              ;increment value
MOV    DS:[BX],AX      ;again write value back to %MW 01,00
MOV    BX,ES:[SI+2]    ;address of %MW 02,00 to BX
MOV    AX,DS:[BX]      ;load value from %MW 02,00 to register AX
DEC    AX              ;decrement value
MOV    DS:[BX],AX      ;again write value back to %MW 02,00
RET
TEST   ENDP
```

The subroutine is stored in the program memory 2. In case of this example the distance between the start of the subroutine and the start of the user program memory amounts to 7400H byte. It must be enable for execution (%K00,01=1). No historical values are needed and 2 variables are used.

DI READ DIRECT INPUT

FBD



IL

```
CAL DI (ENA,IN)
```

PARAMETERS

ENA	BINARY	%I, %M, %O, %K, %S	enable input
IN	BINARY	%I	Input to be updated

DESCRIPTION

This function can only be used with serie 40 & 50 central units.
The function block DI reads *ONE direct* input of the central unit and its extensions. The direct input to be read is specified at the input IN.

The function block is useful : if the cycle time is long
if the capacity utilization of the central unit is high

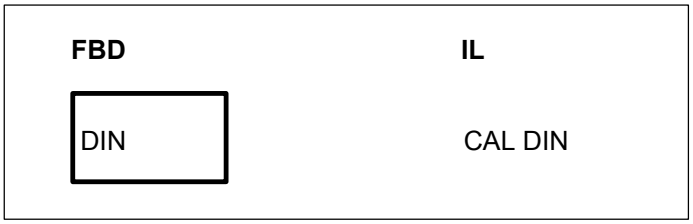
The PLC processor automatically builds an *updated* process image of the inputs at the start of each program cycle. The DI function block is able to get the physical input value specified at the input IN *within* a program cycle. This may be required in conjunction with specific applications, in order to detect and process the input signal changes more than once per program cycle.

- *If the input to be read is on the central unit :*
the new input value can be read instantaneously.

- *If the input to be read is on an extension unit :*
the DI function reads the present input value and generates at the same time a refreshment of the extension bus. The DI function block is usefull if it is used in a time interruption knowing that the present input value read in one interruption is the input value refreshed by the DI function of the previous interruption.

- Remote inputs are refreshed every cycle time. The DI function block can not be used for remote units.

DIN READ DIRECT INPUTS



DESCRIPTION

This function can only be used with serie 90 central units.
With the help of the CS31 bus processor, the function block DIN reads all of the *direct* inputs of the PLC processor. The *direct* inputs are the inputs which are directly available at the terminals of the PLC processor.

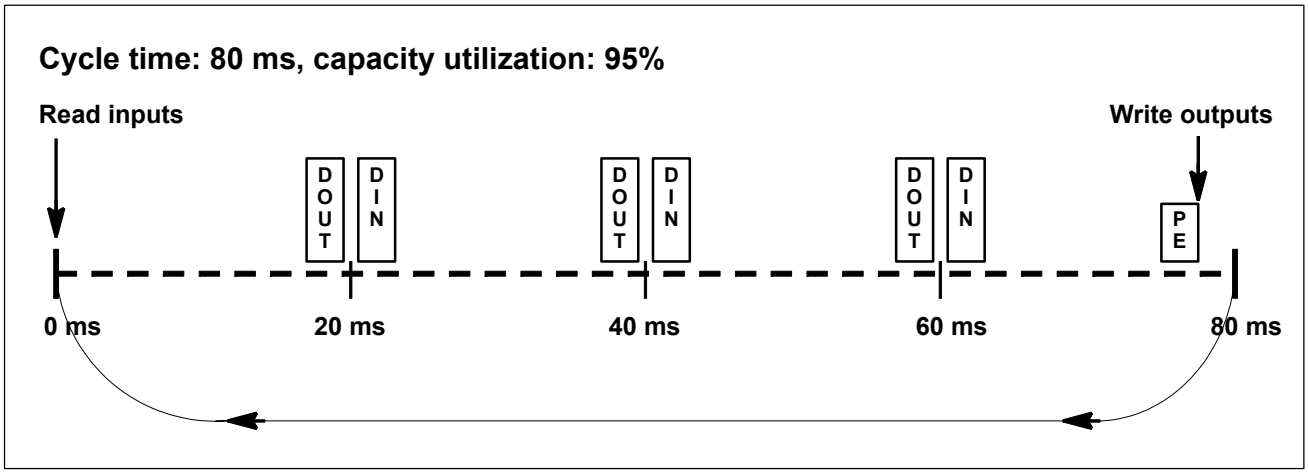
The function block is useful : with stand-alone PLC processors
if the cycle time is long
if the capacity utilization of the PLC is high

The PLC processor automatically builds an *updated* process image of the *direct* inputs at the start of each program cycle. The DIN function block is able to create additional updates of the process image of the *direct* inputs *within* program cycles. This may be required in conjunction with specific applications, in order to detect and process signal changes of the *direct* inputs more than once per program cycle.

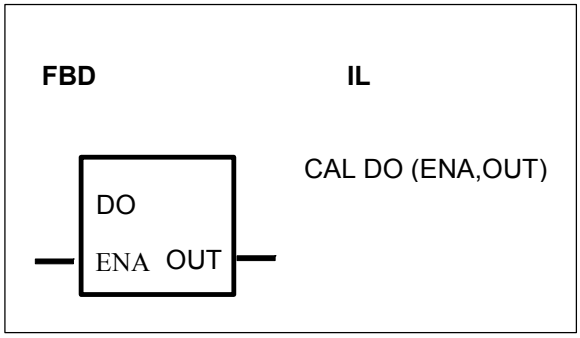
If the DIN and DOUT function blocks are used, the terminal–to–terminal reaction time between *direct inputs and direct outputs* can be reduced.

Example :
- Cycle time : 80 ms
- Capacity utilization : 95%
- 3 DIN + 3 DOUT blocks distributed regularly over the PLC program (i.e. after 20 ms, 40 ms and 60 ms)

In this example, the direct inputs of the PLC processor are read every 20 ms. The updated values of the direct outputs are provided with the same frequency. See the figure below for illustration.



DO WRITE DIRECT OUTPUT



PARAMETERS

ENA	BINARY	%I, %M, %O, %K, %S	enable input
OUT	BINARY	%O	Output to be updated

DESCRIPTION

This function can only be used with serie 40 & 50 central units.
The function block DO writes *ONE direct* output of the central unit or its extension. The direct output to be written is specified at the output OUT.

The function block is useful if the cycle time is long
if the capacity utilization of the central unit is high

The PLC processor automatically outputs a process image of the *direct* outputs, which was *updated* during the program cycle, at the end of each program cycle. The

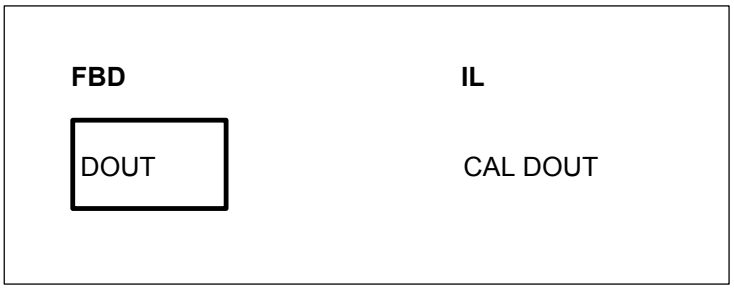
DO function block is able to set the physical output specified at the OUT parameter *within* program cycles. This may be required in conjunction with specific applications, in order to have available the output signal changes more than once per program cycle.

- If the output to be written is on the central unit :
the new output value can be write instantaneously.

- If the output to be written is on an extension unit :
the new output value can be write with a delay of max 2 ms.

- Remote outputs are refreshed every cycle time. The DO function block can not be used for remote outputs.

DOUT WRITE DIRECT OUTPUTS



DESCRIPTION

This function can only be used with serie 90 central units.
With the help of the CS31 bus processor, the function block DOUT writes all of the *direct* outputs of the PLC processor. The *direct* outputs are the outputs which are directly available at the terminals of the PLC processor.

The function block is useful : with stand-alone PLC processors
if the cycle time is long
if the capacity utilization of the PLC is high

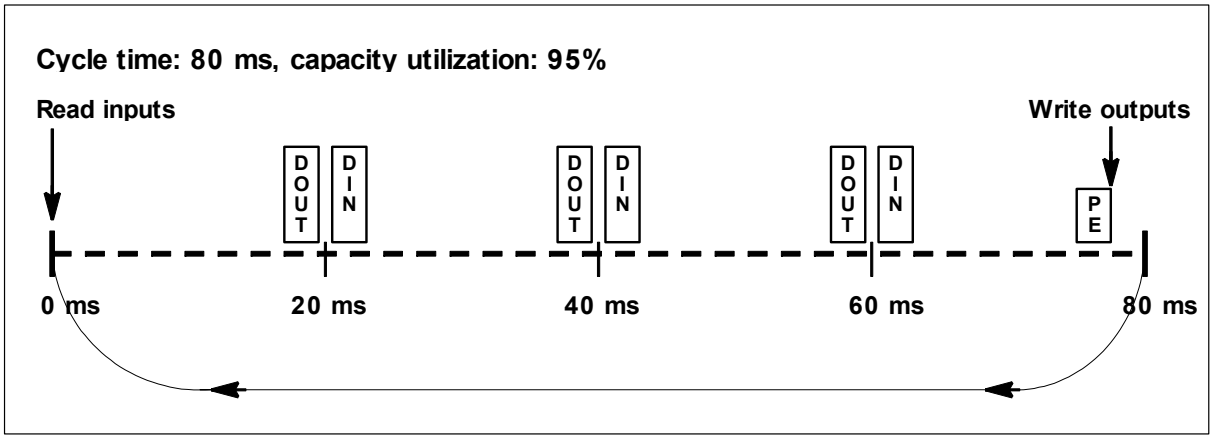
The PLC processor automatically outputs a process image of the *direct* outputs, which was *updated* during the program cycle, at the end of each program cycle. The DOUT function block is able to output additional updates of the process image of the *direct* outputs *within* program cycles. This may be required in conjunction with specific applications, in order to have available signal changes of the *direct* outputs more than once per program cycle.

If the DIN and DOUT function blocks are used, the terminal–to–terminal reaction time between *direct inputs and direct outputs* can be reduced.

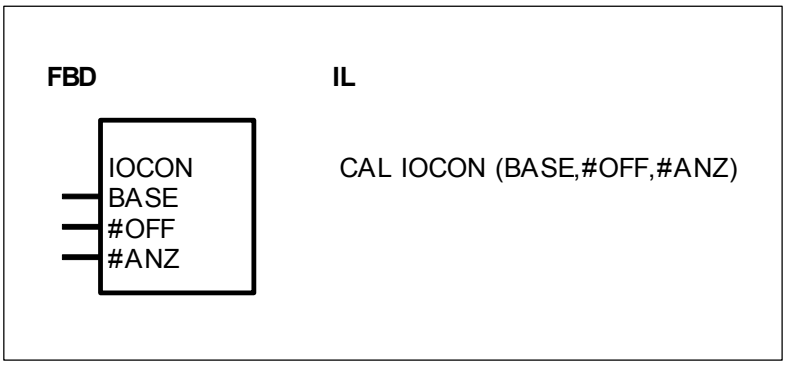
Example :
- Cycle time : 80 ms
- Capacity utilization : 95%
- 3 DIN + 3 DOUT blocks distributed regularly over the PLC program (i.e. after 20 ms, 40 ms and 60 ms)

In this example, the direct inputs of the PLC processor are read every 20 ms. The

updated values of the direct outputs are provided with the same frequency. See the figure below for illustration.



IOCON INPUT/OUTPUT CONFIGURATION



PARAMETERS

BASE	BINARY	%I, %O	Base of input or output identifier
#OFF	DIRECT CONSTANT	#, #H	Offset of the desired group number (related to BASE)
#ANZ	DIRECT CONSTANT	#, #H	Number of channels to be configured

DESCRIPTION

The PLC processor only operates those *binary* inputs and outputs whose identifier are explicitly configured in the PLC program.

The IOCON function block can enable a defined range of binary inputs or outputs for operation, whose identifier *do not exist* in the PLC program. Normally, those inputs and outputs are not processed automatically. This enable function is, for instance, required, when binary inputs are to be read with the IDLB function block or binary outputs are to be written with the IDSB function block, respectively.

The input or output range is specified at the function block. The range is configured at the program start in conjunction with the program preparation. The IOCON block has no function within the program cycle itself.

BASE BINARY
The identifier of a binary input or output is given at the input BASE. This identifier is the base which #OFF and #ANZ relate to.

#OFF DIRECT CONSTANT
Range of values : #OFF ≥ 0
At the input #OFF, it is specified where the input or output range, which is to be operated additionally, starts.
The beginning of the additionally configured range is as follows :
The value specified at the input #OFF is added to the group number of the configured input or output identifier, which is given at the input BASE. The new input or output identifier, calculated in this way, marks the beginning of the input or output range which is to be configured additionally.

The beginning of the range is calculated as follows :
GroupNo._beginning_of_range = GroupNo._BASE + #OFF
Channel_number_beginning_of_range = Channel_number_BASE
where : GroupNo._beginning_of_range : first group number of the range
Channel_number_beginning_of_range : first channel number of the range
#OFF : value at the input #OFF
GroupNo._BASE : group number of the identifier at BASE
Channel_number_BASE : channel number of the identifier at BASE

Example :
- Input BASE : %O 01,04, i.e. group number = 01 and channel number = 04
- Input #OFF : 3, i.e. the range to be configured starts with : %O (01+3),04 = %O 04,04

#ANZ DIRECTE CONSTANT
Range of values : #ANZ ≥ 1
The input #ANZ specifies the number of input or output channels to be configured in the additional range.

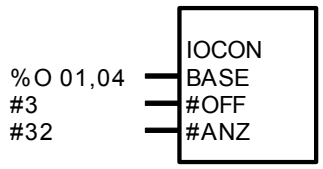
The end of the range is calculated as follows :
Equation : (#ANZ-1 + channel_number_BASE) :16 = DG,
Remainder = channel_number_of_end_of_range

GroupNo_end_of_range = GroupNo._beginning_of_range + DG
where : GroupNo_end_of_range : last group number of the range
Channel_number_of_end_of_range : last channel number of the range
GroupNo_beginning_of_range : first group number of the range
Channel_number_BASE : channel number of the identifier at BASE

Example :
Configured : BASE : %O 01,04
#OFF : 3
#ANZ : 32
It follows : Beginning of range : %O (01+3),04 = %O 04,04
End of range : Equation : (32 -1 + 4) :16 = 2, Remainder 3

--> %O (04+2),03 = %O 06,03
Range : %O 04,04 ... %O 06,03

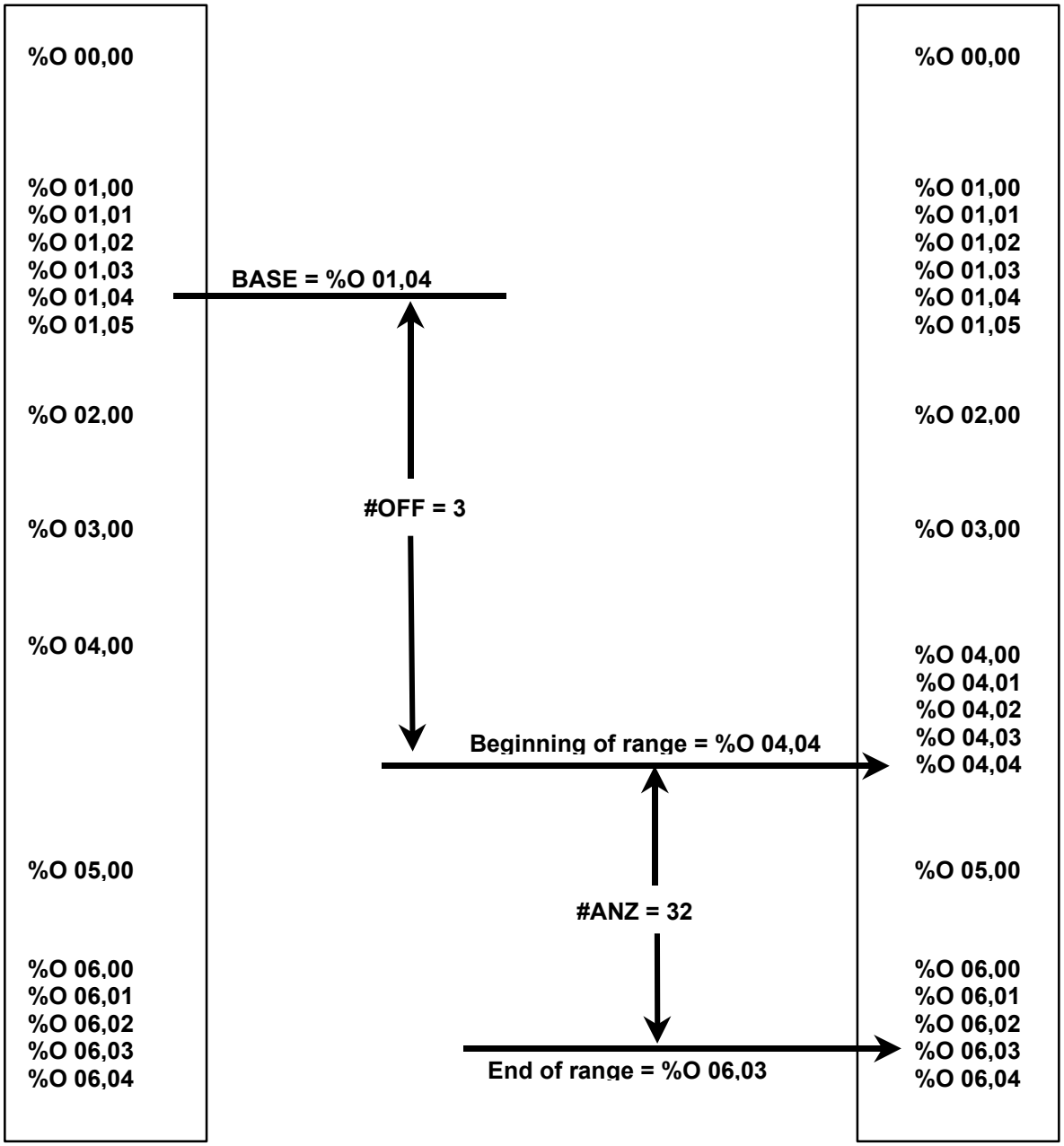
FBD



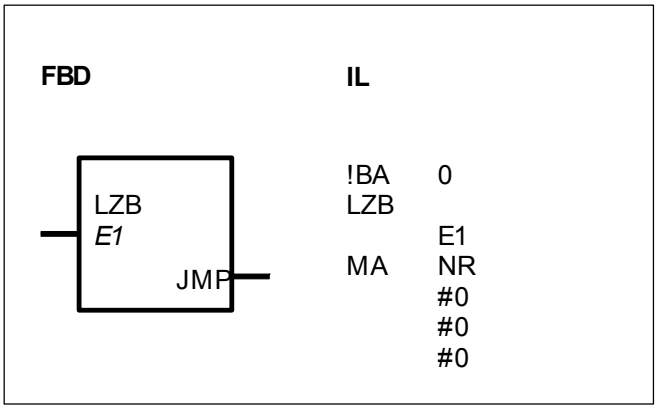
IL

```
CAL IOCON (%O01,04,#3,#32)
```

The outputs %O 04,04 ... %O 06,03 are output, even if their names are not configured in the PLC



LZB RUN NUMBER BLOCK



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Run number
JMP	SPECIAL		Target label

DESCRIPTION

This function block controls processing of a program part. This program part is called run number block and begins with the function block LZB and ends with the affiliated target label This program part is processed as follows depending on the value of the operand at the input E1 :

E1 = 0 : Program part is not processed
E1 = 1 : Program part is processed during every cycle
E1 = 2 : Program part is processed during every second cycle
E1 = n : Program part is processed during every nth cycle

FBD

Designation of the target label : JUMP label
The output JMP has to be connected to a jump label

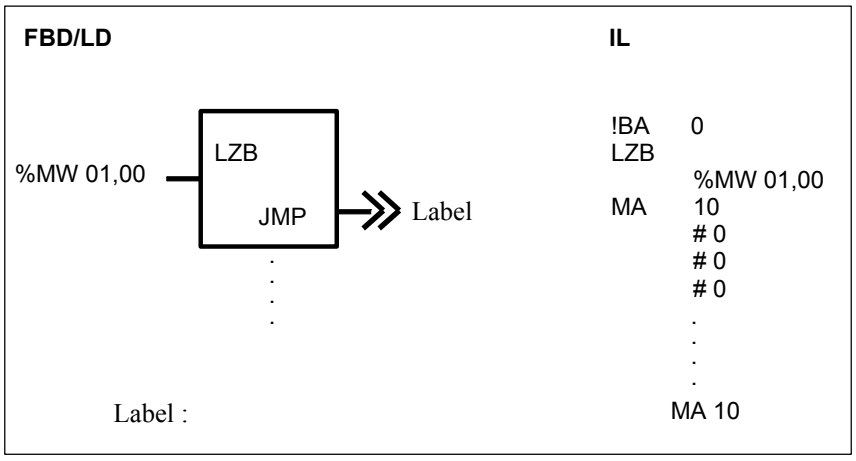
IL

Designation of the target label : MA No.

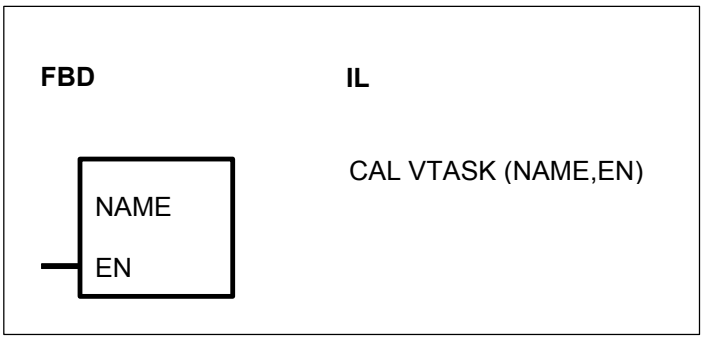
Where :	MA	Key word
	0 < Nr < 999	Number of the label

The PLC automatically calculates :
• The number of historical values to be skipped
• The address of the target label
• The pointer to the own historical value
The PLC enters the 3 computed values in the intended positions of the LZB block. When planned in IL, the user can basically fill these 3 program words with any contents. However, it is suggested to enter here the numerical value 0.

Caution : The 3 program words should not be written with NOPs because these are removed when optimizing the user program.


FBD/LD
IL
%MW 01,00
LZB
JMP
Label
Label :
!BA 0
LZB
%MW 01,00
MA 10
0
0
0
MA 10

VTASK INTERRUPTION TASK VALIDATION



PARAMETERS

NAME	TXT		Interruption name
EN	BINARY	%I, %M, %O, %K, %S	Interruption validation

DESCRIPTION

The VTASK function validates the execution of an interruption whose name is specified in NAME variable

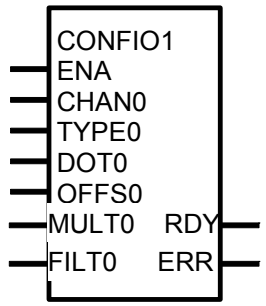
In function block diagram, the block has the name of the interruption.
In instruction list, the function is VTASK with NAME and EN as parameters.

4 CS31 functions

CS31 functions	from pages C-84 to C-116	serie C[tler]	40	50	90	30
CONFIO1	1 analog channel configuration		x	x		
CONFIO4	4 analog channels configuration		x	x		
CONFIO8	8 analog channels configuration		x	x		
CS31CO	Configure CS31 module			x	x	x
CS31QU	Acknowledge CS31 error			x	x	x
MT_CS31	data sent by CS31 master			x	x	x
MR_CS31	data received by CS31 master			x	x	x
ST_CS31	data sent by CS31 slave			x	x	x
SR_CS31	data received by CS31 slave			x	x	x

CONFIO1 1 ANALOG CHANNEL CONFIGURATION

FBD



IL

```
CAL CONFIO1(ENA, CHAN0, TYPE0, DOT0,
OFFS0, MULT0, FILT0, RDY, ERR)
```

PARAMETERS

ENA	BINARY	%I, %O, %M, %S,%K	Enable block processing
CHAN0	WORD	%IW, %OW	Channel identification
TYPE0	WORD	%MW, %OW, %KW	Type of analog channel
DOT0	WORD	%MW, %OW, %KW	Position of the dot of the display value
OFFS0	WORD	%MW, %OW, %KW	Value of the offset for the display value
MULT0	WORD	%MW, %OW, %KW	Value of the multiplication for the display value
FILT0	WORD	%MW, %OW, %KW	Filtering time
RDY	BINARY	%O, %M	Processing of the configuration is completed
ERR	BINARY	%O, %M	An error is detected

DESCRIPTION

The function block CONFIO1 is used to

-configure the type (voltage, current or PT100/Pt1000) of one analog channel on the AC31 extensions.

-change the filtering time of the analog input

-change the scale of the display value.

-lock or unlock the configuration for all analog channels of one analog extension.

The analog channel configuration is set through the function block instead of the pushbutton on the front plate of the analog extension.

The configuration is stored in an internal EEPROM in the analog extension.

The scale of the display value is set according to the formula

Display value = word value * MULT0/ 32767 + OFFS0

The latest configured channel number of one analog extension is displayed

ENA Binary
The function block is processed when ENA is on the rising edge 0->1

CHAN0 Word
The analog channel to be configured is directly set
For example, %IW00.00 for the analog input 0 on the analog extension at the address 0, %OW65.01 for the analog output 1 on the analog extension at the address 65

TYPE0 Word
Type of analog signal:
0 : the channel is set to +/- 10 V
1 : the channel is set to 0-20mA
2 : the channel is set to 4-20mA
3 : the channel is set to Pt100
4 : the channel is set to Pt1000
5 : the channel is set to Pt100 3 wires
6 : the channel is set to Pt1000 3 wires
8 : the channel is set to Ni 1000
9: the channel is set to BALCO500
14 : unlock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy
15 : lock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy

The configuration is automatically unlocked after a power supply ON

DOT0 Word

Position of the dot on the display

DOT0=0 4 digits are displayed without dot
Example :value =1234 display value is 1234
DOT0=1 4 digits are displayed with dot on position 1
Example :value =1234 display value is 123.4
DOT0= 2 4 digits are displayed with dot on position 2
Example: value =1234 display value is 12.34
DOT0= 3 4 digits are displayed with dot on position 3
Example: value =1234 display value is 1.234

If DOT0 <0 or DOT0>3, the configuration is the same than DOT0=0

OFFS0 Word
Value of the offset
-32767 <= OFFS0 <= 32767

MULT0 Word
Value of the multiplication
-32767 <= OFFS0 <= 32767
If MULT0 =0, the parameters OFFS0 and DOT0 are not used.
In this case, the scale is set to the factory setting scale.
The parameter MULT0=0 can be used to set a channel value to the display

FILT0 Word

0 : internal filter according to the documentation of analog extension
1-127 : integration number

160 : Fast refresh time (50ms instead of 120 ms in standard)
192 : 60Hz Filter
224 : 50Hz Filter

All channels of one extension will be affected by this parameter.

The time filter formula is
K=FILT0
Vn= result (T)
Vn-1=result (T-1)
Vins=analog value without filtering

$Vn=\Sigma n/ K$
With $\Sigma n=(Vins - Vn-1) + \Sigma n-1$
Initial value is : V1=Vins
$\Sigma 1=K\ V1$

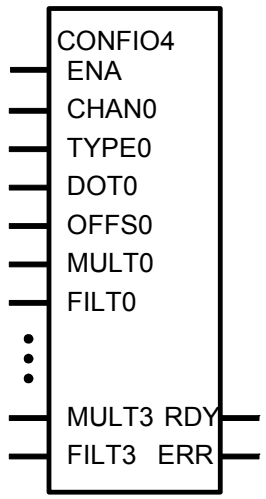
RDY Binary
This bit is set to 0 during the function processing

ERR Binary
This bit is set to 1 during one cycle (the bit RDY is set to 1 in the same time)
An error is detected if:
-one parameter value is wrong
-the analog channel doesn't exist
-communication problem between central unit and analog extension

Note: 3 historical values are used by the function CONFIO1

CONFIO4 4 ANALOG CHANNELS CONFIGURATION

FBD



IL

```
CAL CONFIO4(ENA, CHAN0, TYPE0, DOT0,
OFFS0, MULT0, FILT0,…,FILT3, RDY, ERR)
```

PARAMETERS

ENA	BINARY	%I, %O, %M, %S,%K	Enable block processing
CHAN0	WORD	%IW, %OW	Channel identification
TYPE0	WORD	%MW, %OW, %KW	Type of analog channel
DOT0	WORD	%MW, %OW, %KW	Position of the dot of the display value
OFFS0	WORD	%MW, %OW, %KW	Value of the offset for the display value
MULT0	WORD	%MW, %OW, %KW	Value of the multiplication for the display value
FILT0	WORD	%MW, %OW, %KW	Filtering time
CHAN1	WORD	%IW, %OW	Channel identification
TYPE1	WORD	%MW, %OW, %KW	Type of analog channel
…..	….	….	…..
MULT3	WORD	%MW, %OW, %KW	Value of the multiplication for the display value
FILT3	WORD	%MW, %OW, %KW	Filtering time
RDY	BINARY	%O, %M	Processing of the configuration is completed
ERR	BINARY	%O, %M	An error is detected

DESCRIPTION

The function block CONFIO4 is used to

-configure the type (voltage, current or PT100/Pt1000) of four analog channels on the AC31 extensions. The 4 channels can be located on different extensions.

-change the filtering time of the analog input

-change the scale of the display value.

-lock or unlock the configuration for all analog channels of one analog extension.

The analog channel configuration is set through the function block instead of the pushbutton on the front plate of the analog extension.

The configuration is stored in an internal EEPROM in the analog extension.

The scale of the display value is set according to the formula

Display value = word value * MULT0/ 32767 + OFFS0

The latest configured channel number of one analog extension is displayed

ENA Binary
The function block is processed when ENA is on the rising edge 0->1

CHAN0 Word
The analog channel to be configured is directly set
For example, %IW00.00 for the analog input 0 on the analog extension at the address 0, %OW65.01 for the analog output 1 on the analog extension at the address 65

TYPE0 Word
Type of analog signal:

0 : the channel is set to +/- 10 V
1 : the channel is set to 0-20mA
2 : the channel is set to 4-20mA
3 : the channel is set to Pt100
4 : the channel is set to Pt1000
5 : the channel is set to Pt100 3 wires
6 : the channel is set to Pt1000 3 wires
8 : the channel is set to Ni 1000
9: the channel is set to BALCO500
14 : unlock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy
15 : lock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy

The configuration is automatically unlocked after a power supply ON

DOT0 Word

Position of the dot on the display

DOT0=0 4 digits are displayed without dot
Example :value =1234 display value is 1234
DOT0=1 4 digits are displayed with dot on position 1
Example :value =1234 display value is 123.4
DOT0= 2 4 digits are displayed with dot on position 2
Example: value =1234 display value is 12.34
DOT0= 3 4 digits are displayed with dot on position 3
Example: value =1234 display value is 1.234

If DOT0 <0 or DOT0>3, the configuration is the same than DOT0=0

OFFS0 Word
Value of the offset
-32767 <= OFFS0 <= 32767

MULT0 Word
Value of the multiplication
-32767 <= OFFS0 <= 32767
If MULT0 =0, the parameters OFFS0 and DOT0 are not used.
In this case, the scale is set to the factory setting scale.
The parameter MULT0=0 can be used to set a channel value to the display

FILT0 Word

0 : internal filter according to the documentation of analog extension
1-127 : integration number

160 : Fast refresh time (50ms instead of 120 ms in standard)
192 : 60Hz Filter
224 : 50Hz Filter

All channels of one extension will be affected by this parameter.

The time filter formula is
K=FILT0
Vn= result (T)
Vn-1=result (T-1)
Vins=analog value without filtering

$Vn=\Sigma n/ K$
With $\Sigma n=(Vins - Vn-1) + \Sigma n-1$
Initial value is : V1=Vins
$\Sigma 1=K\ V1$

CHAN1 Word

The analog channel to be configured is directly set
For example, %IW00.00 for the analog input 0 on the analog extension at the address 0, %OW65.01 for the analog output 1 on the analog extension at the address 65. The channel can be located on a other extension.
For example %IW00.00 on CHAN0 and %OW02.01 on CHAN1

….

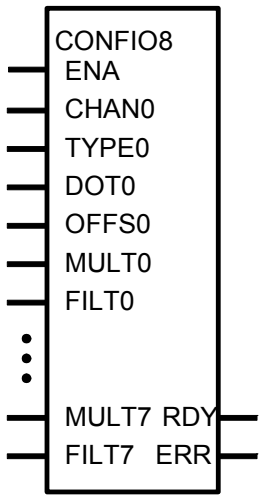
RDY Binary
This bit is set to 0 during the function processing

ERR Binary
This bit is set to 1 during one cycle (the bit RDY is set to 1 in the same time)
An error is detected if:
-one parameter value is wrong
-the analog channel doesn't exist
-communication problem between central unit and analog extension.

Note: 3 historical values are used by the function CONFIO4

CONFIO8 8 ANALOG CHANNELS CONFIGURATION

FBD



IL

```
CAL CONFIO4(ENA, CHAN0, TYPE0, DOT0,
OFFS0, MULT0, FILT0,…,FILT7, RDY, ERR)
```

PARAMETERS

ENA	BINARY	%I, %O, %M, %S,%K	Enable block processing
CHAN0	WORD	%IW, %OW	Channel identification
TYPE0	WORD	%MW, %OW, %KW	Type of analog channel
DOT0	WORD	%MW, %OW, %KW	Position of the dot of the display value
OFFS0	WORD	%MW, %OW, %KW	Value of the offset for the display value
MULT0	WORD	%MW, %OW, %KW	Value of the multiplication for the display value
FILT0	WORD	%MW, %OW, %KW	Filtering time
CHAN1	WORD	%IW, %OW	Channel identification
TYPE1	WORD	%MW, %OW, %KW	Type of analog channel
…..	….	….	…..
MULT7	WORD	%MW, %OW, %KW	Value of the multiplication for the display value
FILT7	WORD	%MW, %OW, %KW	Filtering time
RDY	BINARY	%O, %M	Processing of the configuration is completed
ERR	BINARY	%O, %M	An error is detected

DESCRIPTION

The function block CONFIO8 is used to

-configure the type (voltage, current or PT100/Pt1000) of eight analog channels on the AC31 extensions. The 8 channels can be located on different extensions.

-change the filtering time of the analog input

-change the scale of the display value.

-lock or unlock the configuration for all analog channels of one analog extension.

The analog channel configuration is set through the function block instead of the pushbutton on the front plate of the analog extension.

The configuration is stored in an internal EEPROM in the analog extension.

The scale of the display value is set according to the formula

Display value = word value * MULT0/ 32767 + OFFS0

The latest configured channel number of one analog extension is displayed

ENA Binary

The function block is processed when ENA is on the rising edge 0->1

CHAN0 Word

The analog channel to be configured is directly set

For example, %IW00.00 for the analog input 0 on the analog extension at the address 0, %OW65.01 for the analog output 1 on the analog extension at the address 65

TYPE0 Word

Type of analog signal:

0 : the channel is set to +/- 10 V

1 : the channel is set to 0-20mA

2 : the channel is set to 4-20mA

3 : the channel is set to Pt100

4 : the channel is set to Pt1000

5 : the channel is set to Pt100 3 wires

6 : the channel is set to Pt1000 3 wires

8 : the channel is set to Ni 1000

9: the channel is set to BALCO500

14 : unlock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy

15 : lock the configuration through the pushbutton on the analog extension front plate at the address xx when CHAN0 is %Iwxx.yy or %Owxx.yy

The configuration is automatically unlocked after a power supply ON

DOT0 Word

Position of the dot on the display

DOT0=0 4 digits are displayed without dot
Example :value =1234 display value is 1234
DOT0=1 4 digits are displayed with dot on position 1
Example :value =1234 display value is 123.4
DOT0= 2 4 digits are displayed with dot on position 2
Example: value =1234 display value is 12.34
DOT0= 3 4 digits are displayed with dot on position 3
Example: value =1234 display value is 1.234

If DOT0 <0 or DOT0>3, the configuration is the same than DOT0=0

OFFS0 Word
Value of the offset
-32767 <= OFFS0 <= 32767

MULT0 Word
Value of the multiplication
-32767 <= OFFS0 <= 32767
If MULT0 =0, the parameters OFFS0 and DOT0 are not used.
In this case, the scale is set to the factory setting scale.
The parameter MULT0=0 can be used to set a channel value to the display

FILT0 Word

0 : internal filter according to the documentation of analog extension
1-127 : integration number

160 : Fast refresh time (50ms instead of 120 ms in standard)
192 : 60Hz Filter
224 : 50Hz Filter

All channels of one extension will be affected by this parameter.

The time filter formula is
K=FILT0
Vn= result (T)
Vn-1=result (T-1)
Vins=analog value without filtering

$Vn=\Sigma n/ K$
With $\Sigma n=(Vins - Vn-1) + \Sigma n-1$
Initial value is : V1=Vins
$\Sigma 1=K\ V1$

CHAN1 Word

The analog channel to be configured is directly set

For example, %IW00.00 for the analog input 0 on the analog extension at the address 0, %OW65.01 for the analog output 1 on the analog extension at the address 65. The channel can be located on a other extension.
For example %IW00.00 on CHAN0 and %OW02.01 on CHAN1

….

RDY Binary
This bit is set to 0 during the function processing

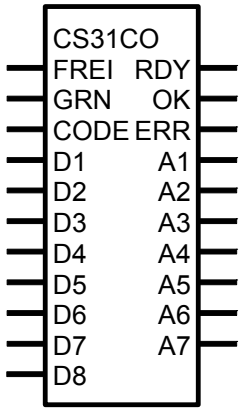
ERR Binary
This bit is set to 1 during one cycle (the bit RDY is set to 1 in the same time)
An error is detected if:

-one parameter value is wrong
-the analog channel doesn't exist
-communication problem between central unit and analog extension.

Note: 3 historical values are used by the function CONFIO8

CS31CO CONFIGURE CS31 MODULES

FBD



IL

```
CAL CS31CO  (FREI,GRN,CODE,D1,D2,D3,D4,D5,D6,D7,
            RDY,OK,ERR,A1,A2,A3,A4,A5,A6,A7)
```

PARAMETERS

FREI	BINARY	%I, %O, %M, %S	Enable (0->1 edge) for processing the block
GRN	WORD	%IW, %OW, %MW, %KW	Group number of the remote module to which the job refers
CODE	WORD	%IW, %OW, %MW, %KW	Identification of the job to be performed
D1	WORD	%IW, %OW, %MW, %KW	1st parameter of the job
D8	WORD	%IW, %OW, %MW, %KW	8th parameter of the job
RDY	BINARY	%O, %M	Processing of the job is completed
OK	BINARY	%O, %M	It has been possible to process the job correctly
ERR	WORD	%OW, %MW	Error message/status message

A1	WORD	%OW, %MW	1st parameter of the response
A7	WORD	%OW, %MW	7th Parameter of the response

DESCRIPTION

The function block serves to configure the CS31 remote modules. The block can both send configuration parameters to the remote modules and also scan their currently set configuration.
Apart from configuration of the CS31 remote modules, the function block can also process further jobs (see List of jobs).

Enable for processing a job once is triggered by a 0->1 edge at input FREI. The required job identification is specified at input CODE. The parameters required for the job are planned at inputs D1...D8. Status messages are signalled at outputs RDY, OK and ERR. The response data of the job are available at outputs A1...A7.
It may take several PLC cycles to process the job.

FREI BINARY
Processing of the block is controlled via input FREI.

FREI = 0 :	*All* block outputs are set to value “0”. However, this is not valid, if a job is *currently being processed*, i. e. processing of a job which is currently being processed, is not influenced by FREI = 0.
FREI = 0→1 edge :	Processing of the job is enabled. Input FREI is *no longer* evaluated *during* processing of the job.
FREI = 1 :	Block is *not* processed, i. e. it no longer changes its outputs. However, this is not valid, if a job is *currently being processed*.

GRN WORD
Group number with which the remote module is addressed by the PLC program.
Range : 0...63
Example : On binary input %I 12,08, “12” is the group number and “08” is the channel number.

CODE WORD
The identification of the job to be executed is specified at input CODE. (See List of jobs on next page).

D1...D8 WORD
The parameters required for the job are preset at inputs D1...D8. The number of parameters is dependent upon the job to be executed. There are also jobs requiring no parameters (see List of jobs on next page).

RDY BINARY
Output RDY signals that processing of the job currently being processed is completed. Output RDY does not indicate whether processing of the job has been *successful* or not. Output RDY has therefore always to be considered together with output OK.

RDY = 1 and OK = 1 : Processing of the job is completed without errors. A new job can be started with a 0->1 edge at the input FREI.
RDY = 1 and OK = 0 : During processing of the job *an error* has been detected. The corresponding error identification is present at the ERR output. A new job can be started with a 0->1 edge at the input FREI.
RDY = 0 : Processing of an enabled job has not yet been completed (job is still being processed) *or* the output RDY has been reset with FREI = 0.

OK BINARY
Output OK signals whether the job has been handled successfully or whether an error has been detected during processing. In the event of an error, an error number is indicated at output ERR. Output OK is not valid until the job has been completed, i.e. when RDY = 1.
The following applies :
If RDY = 1 *and* OK = 1 : Job has been processed successfully
OK = 0 : An error has been detected during processing of the job.

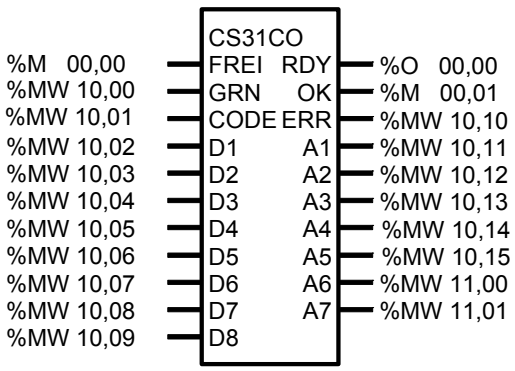
ERR WORD
Status and error identifications are output at output ERR. The status identifications are output *during processing* of a job in order to signal in what stage of processing the job currently is. After enabling a job, status identifications are signalled only for as long as RDY = 0.
The error identifications are output *after completion* of the job processing if an error has occurred. Error identifications are thus not signalled until RDY = 1 and OK = 0.

Error identifications
ERR = 1 : An illegal job identification has been specified at input CODE.
ERR = 2 : Incorrect parameters have been specified at inputs D1...D8 (e.g. a group number for which there is no remote module on the CS31 bus).
ERR = 3 The addressed CS31 remote module does not accept the job.

Status identifications
ERR = 8 : The function block is waiting since a job of another user is currently being processed.
ERR = 10 : The job has been sent to the addressee and the block is waiting for its response.

A1...A7 WORD
After completion of job processing, the response is available at outputs A1...A7. The number of response parameters depends upon the job performed (see List of jobs).

FBD



IL

```
CAL CS31CO (%M00,00, %MW10,00, %MW10,01, %MW10,02,
            %MW10,03, %MW10,04, %MW10,05, %MW10,06,
            %MW10,07, %MW10,08, %MW10,09, %O 00,00
            %M00,01, %MW10,10, %MW10,11, %MW10,12,
            %MW10,13, %MW10,14, %MW10,15, %MW11,00,
    %MW11,01)
```

Example of configuration of the analog input %IW 00,03 for current range 4 - 20 :

%MW 10,00	= 0	Group
%MW 10,01	=170	Job identifier for "configure analog unit channel"
%MW 10,02	=3	Channel
%MW 10,03	=49	Identification for current range 4 - 20 mA

These flags may be set to these values, e.g. with function "overwrite" in the PLC.

List of jobs

Processing a job consists of :
- transferring the job and
- supplying the OK response or not-OK response

The OK response is described in connection with the corresponding job.

The not-OK response of the individual jobs always looks as follows :

▪ **Not-OK response**
The following basically applies for the not-OK response :
RDY : 1
OK : 0

ERR :	1	inadmissible job identification
	2	wrong parameter; e.g. group number to which there exists no remote module
	3	remote module does not accept job
A1 ... A7 :	0	

● **Updating of the maximum number of remote modules detected**

Input word %IW 07,15 contains, amongst other things, the maximum number of remote modules detected in the past. The actual number of remote modules which exist at the moment may be less. This command is used to update this value. The modules which exist are counted and the value is stored. The user can scan this value in the PLC program. It is located in input word %IW 07,15, bits 8...15.

▪ **job**

GRN : 255 (Master PLC with bus)
CODE : 132
D1 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Scanning the open-circuit monitoring of an input to establish whether it is activated or deactivated**

▪ **job**

GRN : group number 0 ... 63
CODE : 32
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY :	1	
OK :	1	
A1 :	47	open-circuit monitoring ON
	32	open-circuit monitoring OFF
A2 ... A7 :	0	

● **Scanning the open-circuit monitoring of an output to establish whether it is activated or deactivated**

▪ **job**

GRN : group number 0 ... 63
CODE : 33
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY :	1	
OK :	1	
A1 :	47	open-circuit monitoring ON
	32	open-circuit monitoring OFF
A2 ... A7 :	0	

● **Deactivating or activating open-circuit monitoring of an input**

▪ **job**

GRN : group number 0 ... 63
CODE : 224 open-circuit monitoring ON
160 open-circuit monitoring OFF
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Deactivating or activating open-circuit monitoring of an output**

▪ **job**

GRN : group number 0 ... 63
CODE : 225 open-circuit monitoring ON
161 open-circuit monitoring OFF
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Scanning a channel to establish whether it is configured as input or input/output**

▪ **job**

GRN : group number 0 ... 63
CODE : 34
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 : 34 input
35 input/output
A2 ... A7 : 0

● **Configuration of a channel as input or input/output**

▪ **job**

GRN : group number 0 ... 63
CODE : 162 input
163 input/output
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Scanning the input delay of a channel**

▪ **job**

GRN : group number 0 ... 63
CODE : 38
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 : input delay :
2 2 ms
4 4 ms
.
.
.
30 30 ms
32 32 ms
A2 ... A7 : 0

● **Setting the input delay of a channel**

▪ **job**

GRN : group number 0...63
CODE : 166
D1 : channel number
D2 : input delay :
2 2 ms
4 4 ms
.
.
.
30 30 ms
32 32 ms

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Acknowledging errors on remote module**

This command can be used to reset the error messages registered on the selected remote module. Reset is possible only if the cause of the error is no longer operative.

▪ **job**

GRN : group number 0 ... 63
CODE : 232
D1 : lowest channel number on the module :
0 lowest channel number on the module is 0 (≤7)
8 lowest channel number on the module is 8 (>7)
D2 : module type :
0 binary input
1 analog input
2 binary output

	3	analog output
	4	binary input/output
	5	analog input/output
	Note :	Bit : even number (0, 2, 4)
		Word : odd number (1, 3, 5)

D3 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Acknowledging errors on remote module and resetting configuration values to default setting**

In addition to the job 'Acknowledging errors on remote module', all configurable settings are reset to the default setting.

▪ **job**

GRN : group number 0 ... 63
CODE : 233
D1 : first channel number on the module :

0	first channel number on the module is 0 (≤7)
8	first channel number on the module is 8 (>7)

D2 : module type :

0	binary input
1	analog input
2	binary output
3	analog output
4	binary input/output
5	analog input/output
Note :	Bit : even number (0, 2, 4)
	Word : odd number (1, 3, 5)

D3 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Scanning configuration of an analog input**

▪ **job**

GRN : group number 0 ... 63
CODE : 42
D1 : channel number
D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1

A1 :	50	input 0 ... 20 mA
	49	input 4 ... 20 mA

A2 ... A7 : 0

● **Scanning configuration of an analog output**

▪ **job**
GRN : group number 0 ... 63
CODE : 43
D1 : channel number
D2 ... D8 : not used
▪ **OK response**
RDY : 1
OK : 1
A1 : 50 output 0 ... 20 mA
49 output 4 ... 20 mA
51 output ± 10V
A2 ... A7 : 0

● **Configuration of an analog input**
▪ **job**
GRN : group number 0 ... 63
CODE : 170
D1 : channel number
D2 : 50 input 0 ... 20 mA
49 input 4 ... 20 mA
D3 ... D8 : not used
▪ **OK response**
RDY : 1
OK : 1
A1 ... A7 : 0

● **Configuration of an analog output**
▪ **job**
GRN : group number 0 ... 63
CODE : 171
D1 : channel number
D2 : 50 output 0 ... 20 mA
49 output 4 ... 20 mA
51 output ± 10V
D3 ... D8 : not used
▪ **OK response**
RDY : 1
OK : 1
A1 ... A7 : 0

● **Scanning bus configuration**
The bus interface of the Master PLC has a list which stores specific data of the remote modules. The remote modules are numbered in this list in the order in which they can be found on the CS31 bus. The internal number of the modules must be specified with this command. The response to this command is the group number stored under this number and status information on the corresponding module.
▪ **job**
GRN : not evaluated
CODE : 80
D1 : number from module list (1 ... 31)

D2 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 : status of the remote module :
Bit 0 ... 3 : number of process data bytes (binary module) or words (word module), which the module sends to the master
Bit 4 ... 7 : number of process data bytes (binary module) or words (word module), which the master sends to the module
A2 : group number (0 ... 63)
A3 : Bit 0 : 0 lowest channel number ≤7
1 lowest channel number > 7
Bit 1 : 0 binary module
1 word module
A4 ... A7 : 0

● **Read 1 ... 6 bytes**

▪ **job**

GRN : group number 0 ... 63
CODE : 49 read 1 byte
50 read 2 bytes
51 read 3 bytes
52 read 4 bytes
53 read 5 bytes
54 read 6 bytes
D1 : first channel number on the module :
0 first channel number on the module is 0 (≤7)
8 first channel number on the module is 8 (>7)
D2 : module type :
0 binary input
1 analog input
2 binary output
3 analog output
4 binary input/output
5 analog input/output
Note : bit : even number (0, 2, 4)
word : odd number (1, 3, 5)
D3 : Byte start address (Low Byte)
D4 : Byte start address (High Byte)
D5 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 : value of 1st byte
A2 : value of 2nd byte or 0
A3 : value of 3rd byte or 0
A4 : value of 4th byte or 0
A5 : value of 5th byte or 0
A6 : value of 6th byte or 0
A7 : 0

● **Read 1 bit of 1 byte**

▪ **job**

GRN : group number 0 ... 63
CODE : 63
D1 : first channel number on the module :
0 first channel number on the module is 0 (≤7)
8 first channel number on the module is 8 (>7)
D2 : module type :
0 binary input
1 analog input
2 binary output
3 analog output
4 binary input/output
5 analog input/output
Note : bit : even number (0, 2, 4)
word : odd number (1, 3, 5)
D3 : byte start address (Low Byte)
D4 : byte start address (High Byte)
D5 : bit position within bytes 0 ... 7
D6 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 : bit value (0 or 1)
A2 ... A7 : 0

● **Write 1 ... 4 bytes**

▪ **job**

GRN : group number 0 ... 63
CODE : 65 write 1 byte
66 write 2 bytes
67 write 3 bytes
68 write 4 bytes
D1 : first channel number on the module :
0 first channel number on the module is 0 (≤7)
8 first channel number on the module is 8 (>7)
D2 : module type :
0 binary input
1 analog input
2 binary output
3 analog output
4 binary input/output
5 analog input/output
Note : bit : even number (0, 2, 4)
word : odd number (1, 3, 5)
D3 : byte start address (Low Byte)
D4 : byte start address (High Byte)
D5 : value of 1st byte
D6 : value of 2nd byte or not used

D7 : value of 3rd byte or not used
D8 : value of 4th byte or not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

● **Write 1 bit of 1 byte**

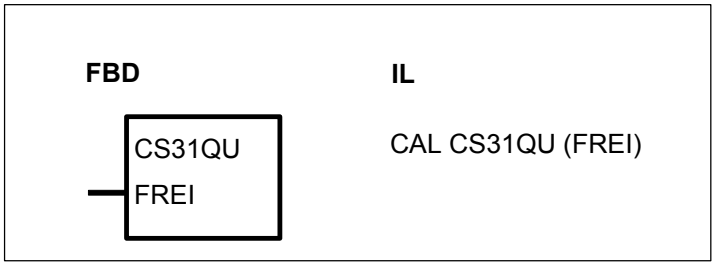
▪ **job**

GRN : group number 0 ... 63
CODE : 79
D1 : first channel number on the module :
0 first channel number on the module is 0 (≤7)
8 first channel number on the module is 8 (>7)
D2 : module type :
0 binary input
1 analog input
2 binary output
3 analog output
4 binary input/output
5 analog input/output
Note : bit : even number (0, 2, 4)
word : odd number (1, 3, 5)
D3 : byte start address (Low Byte)
D4 : byte start address (High Byte)
D5 : bit position within bytes 0 ... 7
D6 : bit value (0 or 1)
D7 ... D8 : not used

▪ **OK response**

RDY : 1
OK : 1
A1 ... A7 : 0

CS31QU ACKNOWLEDGE CS31 ERRORS



PARAMETERS

FREI	BINARY	%I, %O, %M, %S	Enable block processing

DESCRIPTION

This function block permits error messages of CS31 remote modules to be acknowledged automatically.
Error messages are stored on the CS31 remote modules until they are acknowledged. Even if the error has been remedied, the error message is still pending on the module until acknowledgment and is also signalled to the PLC until such time as it is acknowledged.
Processing of the block is enabled with a 1 signal at input FREI, and the block then acknowledges CS31 errors continuously. It may take several PLC cycles to acknowledge an error on a CS31 module.
If the function block is enabled, it constantly checks whether a CS31 error of class 3 or 4 has occurred and acknowledges this error.

1. A CS31 error of class 3 has occurred :
The block acknowledges the error on the CS31 remote module signalling the error *and* also clears the error message on the PLC, i.e. the error flag %M 255,13 is reset and LED FK3 is deactivated.

Example of an FK3 error :
- a remote module is disconnected from the CS31 bus.

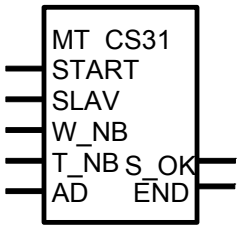
Note : In PLC configuration menu of the Control Panel in the CS31Graf, the "central unit reaction in case of FK3 error" must be "warning" in order to permit errors of class 3 to be processed. If "abort" is selected, the PLC automatically aborts execution of the PLC program when an FK3 error is detected.

2. A CS31 error of class 4 has occurred :
The block acknowledges the error on the CS31 remote module signalling the error *and* also clears the error message on the PLC, i.e. error flag %M 255,14 is reset.

Examples of an FK4 error :
- a remote module signals open circuit

MT_CS31 DATA SENT BY A MASTER TO A SLAVE

FBD



IL

user function block(IL not available)

PARAMETERS

START	BINARY	%I, %O, %M, %S	Enable block processing

SLAV	WORD	%MW, %OW, %KW	Slave number of the slave CPU
W_NB	WORD	%MW, %OW, %KW	Word number to exchange per sending
T_NB	WORD	%MW, %OW, %KW	Total word number to transmit
AD	WORD	%MW, %OW, %KW	First variable address of the data area
S_OK	BINARY	%O, %M	Slave CPU is ready to receive data
END	BINARY	%O, %M	Data area has been transmitted

DESCRIPTION

The function block MT_CS31 in the master has to be used with the function block SR_CS31 in the slave; and MR_CS31 in the master with ST_CS31 in the slave.

These function blocks allow to exchange a data area between a master central unit and a slave central unit on the CS31 bus, with the possibility to keep a quick data transfer between the master and the slave.
The transmission or receiving has to be configured in the slave central unit as follows
Size of the transmitting and sending area on CS31 bus for slave central unit.: 8 words

A maximum of 7 words are allowed to exchange in one direction per sending. The 8th word is reserved for the handshake.
Only the address of the 1st variable of the data area has to be given.
Words are then automatically transfered 7 by 7 (or 6 by 6,...) from the data area to OW xx,01 to OW xx,07 (or OW xx,01 to OW xx,06;...)
OW xx,00 is always reserved for the handshake to send; and IW xx,07 for the handshake to be received.
When words are sent 6 by 6 (or 5 by 5,...), OW xx,07 (and OW xx,06;...) stays available for a quick data transfer between the master and the slave.
(xx = the slave number of the slave central unit : $0 \leq xx \leq 61$ for a 50 serie and $0 \leq xx \leq 5$ for a 90 serie).

START Binary
The transfert is started. The data area is trasmitted only one time .
A new transmit is possible when END=1

SLAV Word
(For the function blocks in the master only)
Slave number of the slave central unit
The value is : $0 \leq SLAV \leq 61$ for 50 serie and $0 \leq SLAV \leq 5$ for 90 & 30 serie

W_NB Word
Number of words to transmit or receive per sending
The value is : $1 \leq W\_NB \leq 7$
The value has to be the same than the W_NB of the corresponding function block used
in the other central unit.

ex : if W_NB=6 in the master then W_NB=6 in the slave.
In the master, OW xx,01 to OW xx,06 are used for the transmission of words.
OW xx,00 and IW xx,07 are reserved for the handshake.

OW xx,07 is free and available for direct control of the slave I/O for example.

T_NB Word
(For a transmission only)
Total word number to transmit
The value has to be a multiple of W_NB : T_NB = n * W_NB

Note : n has to be chosen not to exceed addresses allowed in the central unit selected.

AD Word
Address of the first variable of the data area. Only the offset address is requested.

The Segment is 0 for 30 & 50 serie and 30C2H for a 90 serie
A 90 serie with 8 kinstructions memory can not use this function block.
The segment (30C2H) has to be modified in the code with the user library.

Please refer to the CPU respective documentation to find the hexadecimal addresse of variable,.

S_OK Binary
This bit is set by the slave.
It can be used to start a new transmit of data area

END Binary
This bit is set when the data area is completly transmitted to the slave CPU.
This bit is reset when START=0. Information that the slave has received this bit is necessary to start again the transfer. The bit S_OK can be use (it can be managed by the slave).

CAUTION

The data are sent every three cycle times between the master and the slave.

The value of W_NB in the slave has to be the same than W_NB in the master.

T_NB = n * W_NB
Pay attention not to exceed addresses allowed in the central unit selected.
The data area has to be chosen without any reserved area.

OW xx,00 to OW xx,07 and IW xx,00 to IW xx,07 are used following the function block used and the value of W_NB.

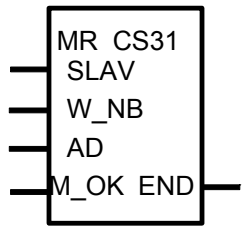
The data exchange is an exchange of words or double words. In the case of the 30 serie & 50 serie , the data area can be a bit area.

Several slaves can be used for the data exchange.

This function block used 431 words in the user program memory.

MR_CS31 DATA RECEIVED BY A MASTER FROM A SLAVE

FBD



IL

user function block(IL not available)

PARAMETERS

SLAV	WORD	%MW, %OW, %KW	Slave number of the slave CPU
W_NB	WORD	%MW, %OW, %KW	Word number to exchange per receiving
AD	WORD	%MW, %OW, %KW	First variable address of the data area
M_OK	BINARY	%I, %O, %M	CPU is ready to receive data
END	BINARY	%O, %M	Data area has been transmitted

DESCRIPTION

The function block MR_CS31 in the master has to be used with the function block ST_CS31 in the slave; and MT_CS31 in the master with SR_CS31 in the slave.

These function blocks allow to exchange a data area between a master central unit and a slave central unit on the CS31 bus, with the possibility to keep a quick data transfer between the master and the slave.
The transmission or receiving has to be configured in the slave central unit as follows
Size of the transmitting and sending area on CS31 bus for slave central unit.: 8 words

A maximum of 7 words are allowed to exchange in one direction per sending. The 8th word is reserved for the handshake.
Only the address of the 1st variable of the data area has to be given.
Words are then automatically transfered 7 by 7 (or 6 by 6,...) from IW xx,00 to IW xx,06 (or from IW xx,00 to IW xx,05;...) to the data area.
IW xx,07 is always reserved for the handshake to be received; and OW xx,00 for the handshake to send.
When words are sent 6 by 6 (or 5 by 5,...), IW xx,06 (and IW xx,05;...) stays available for a quick data transfer between the master and the slave.
(xx = the slave number of the slave central unit : $0 \leq xx \leq 61$ for a 50 serie and $0 \leq xx \leq 5$ for a 90 serie).

SLAV Word
(For the function blocks in the master only)

Slave number of the slave central unit
The value is : $0 \leq SLAV \leq 61$ for 50 serie and $0 \leq SLAV \leq 5$ for 30 1 90 series

W_NB Word
Number of words to transmit or receive per sending
The value is : $1 \leq W\_NB \leq 7$
The value has to be the same than the W_NB of the corresponding function block used
in the other central unit.

ex : if W_NB=6 in the master then W_NB=6 in the slave.
In the master, OW xx,01 to OW xx,06 are used for the transmission of words.
OW xx,00 and IW xx,07 are reserved for the handshake.
OW xx,07 is free and available for direct control of the slave I/O for example.

AD Word
Address of the first variable of the data area. Only the offset address is requested.

The Segment is 0 for 30 & 50 series and 30C2H for a 90 serie
A 90 serie with 8 kinstructions memory can not use this function block.
The segment (30C2H) has to be modified in the code with the user library.

Please refer to the CPU respective documentation to find the hexadecimal addresse of variable,.

M_OK Binary
This bit is set when the master central unit is ready to receive data area from the CPU slave.
It can be used by the slave CPU to start a new transmit of data area

END Binary
This bit is set when the data area is completly transmitted from the slave CPU.
This bit is reset by the slave CPU

CAUTION

The data are sent every three cycle times between the master and the slave.

The value of W_NB in the master has to be the same than W_NB in the slave.

OW xx,00 to OW xx,07 and IW xx,00 to IW xx,07 are used following the function block used and the value of W_NB.

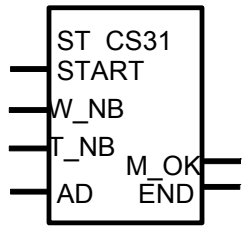
The data exchange is an exchange of words or double words. In the case of the 30 serie & 50 serie, the data area can be a bit area.

Several slaves can be used for the data exchange.

This function block used 230 words in the user program memory.

ST_CS31 DATA SENT BY A SLAVE TO A MASTER

FBD



IL

user function block(IL not available)

PARAMETERS

START	BINARY	%I, %O, %M, %S	Enable block processing
W_NB	WORD	%MW, %OW, %KW	Word number to exchange per sending
T_NB	WORD	%MW, %OW, %KW	Total word number to transmit
AD	WORD	%MW, %OW, %KW	First variable address of the data area
M_OK	BINARY	%O, %M	Master CPU is ready to receive data
END	BINARY	%O, %M	Data area has been transmitted

DESCRIPTION

The function block ST_CS31 in the slaver has to be used with the function block MR_CS31 in the master.

These function blocks allow to exchange a data area between a master central unit and a slave central unit on the CS31 bus, with the possibility to keep a quick data transfer between the master and the slave.
The transmission or receiving has to be configured in the slave central unit as follows
Size of the transmitting and sending area on CS31 bus for slave central unit.: 8 words

A maximum of 7 words are allowed to exchange in one direction per sending. The 8th word is reserved for the handshake.
Only the address of the 1st variable of the data area has to be given.
Words are then automatically transfered 7 by 7 (or 6 by 6,...) - from the data area to OW 00,00 to OW 00,06 (or OW 00,00 to OW 00,05;...)
OW 00,07 is always reserved for the handshake to send; and IW 00,00 for the handshake to be received.
When words are sent 6 by 6 (or 5 by 5,...), OW 00,06 (and OW 00,05;...) stays available for a quick data transfer between the master and the slave.

START Binary

The transfert is started. The data area is trasmitted only one time .
A new transmit is possible when END=1

W_NB Word
Number of words to transmit or receive per sending
The value is : $1 \leq W\_NB \leq 7$
The value has to be the same than the W_NB of the corresponding function block used in the other central unit.

ex : if W_NB=6 in the master then W_NB=6 in the slave.
In the master, OW xx,01 to OW xx,06 are used for the transmission of words.
OW xx,00 and IW xx,07 are reserved for the handshake.
OW xx,07 is free and available for direct control of the slave I/O for example.

T_NB Word
(For a transmission only)
Total word number to transmit
The value has to be a multiple of W_NB : T_NB = n * W_NB

Note : n has to be chosen not to exceed addresses allowed in the central unit selected.

AD Word
Address of the first variable of the data area. Only the offset address is requested.

The Segment is 0 for 30 & 50 serie and 30C2H for a 90 serie
A 90 serie with 8 kinstructions memory can not use this function block.
The segment (30C2H) has to be modified in the code with the user library.

Please refer to the CPU respective documentation to find the hexadecimal addresse of variable,.

M_OK Binary
This bit is set by the master.
It can be used to start a new transmit of data area

END Binary
This bit is set when the data area is completly transmitted to the master CPU.
This bit is reset when START=0. Information that the master has received this bit is necessary to start again the transfer. The bit M_OK can be use (it can be managed by the master).

CAUTION

The data are sent every three cycle times between the master and the slave.

The value of W_NB in the slave has to be the same than W_NB in the master.

T_NB = n * W_NB
Pay attention not to exceed addresses allowed in the central unit selected.
The data area has to be chosen without any reserved area.

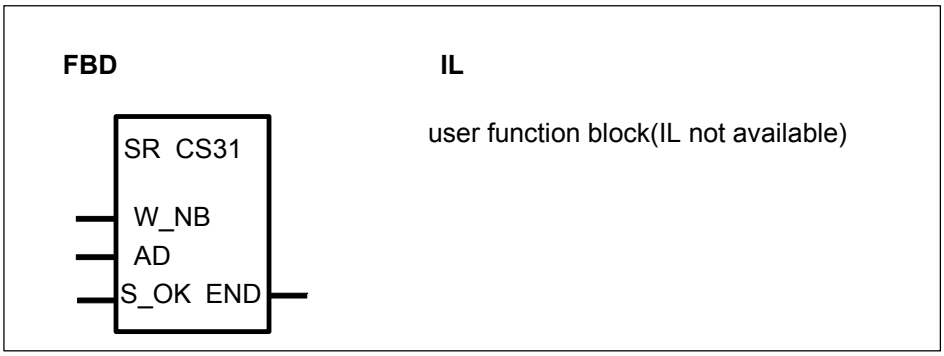
OW xx,00 to OW xx,07 and IW xx,00 to IW xx,07 are used following the function block used and the value of W_NB.

The data exchange is an exchange of words or double words. In the case of the 30 serie & 50 serie , the data area can be a bit area.

Several slaves can be used for the data exchange.

This function block used 431 words in the user program memory

SR_CS31 DATA RECEIVED BY A SLAVE FROM A MASTER



PARAMETERS

W_NB	WORD	%MW, %OW, %KW	Word number to exchange per receiving
AD	WORD	%MW, %OW, %KW	First variable address of the data area
S_OK	BINARY	I%, %O, %M	CPU is ready to receive data
END	BINARY	%O, %M	Data area has been transmitted

DESCRIPTION

The function block SR_CS31 in the slave has to be used with the function block MT_CS31 in the master

These function blocks allow to exchange a data area between a master central unit and a slave central unit on the CS31 bus, with the possibility to keep a quick data transfer between the master and the slave.
The transmission or receiving has to be configured in the slave central unit as follows
Size of the transmitting and sending area on CS31 bus for slave central unit.: 8 words

A maximum of 7 words are allowed to exchange in one direction per sending. The 8th word is reserved for the handshake.
Only the address of the 1st variable of the data area has to be given.
Words are then automatically transfered 7 by 7 (or 6 by 6,...) from IW 00,01 to IW 00,07 (or from IW 00,01 to IW 00,06;...) to the data area.
IW 00,00 is always reserved for the handshake to be received and OW 00,07 for the handshake to send.
When words are sent 6 by 6 (or 5 by 5,...), IW 00,07 (and IW 00,06;...) stays available for a quick data transfer between the master and the slave.

W_NB Word
Number of words to transmit or receive per sending
The value is : $1 \leq W\_NB \leq 7$
The value has to be the same than the W_NB of the corresponding function block used in the other central unit.

ex : if W_NB=6 in the master then W_NB=6 in the slave.
In the master, OW xx,01 to OW xx,06 are used for the transmission of words.
OW xx,00 and IW xx,07 are reserved for the handshake.
OW xx,07 is free and available for direct control of the slave I/O for example.

AD Word
Address of the first variable of the data area. Only the offset address is requested.

The Segment is 0 for 30 & 50 series and 30C2H for a 90 serie
A 90 serie with 8 kinstructions memory can not use this function block.
The segment (30C2H) has to be modified in the code with the user library.

Please refer to the CPU respective documentation to find the hexadecimal addresse of variable,.

S_OK Binary
This bit is set when the slave central unit is ready to receive data area from the CPU master.
It can be used by the master CPU to start a new transmit of data area

END Binary
This bit is set when the data area is completly transmitted from the master CPU.
This bit is reset by the master CPU

_

CAUTION

The data are sent every three cycle times between the master and the slave.

The value of W_NB in the master has to be the same than W_NB in the slave.

OW xx,00 to OW xx,07 and IW xx,00 to IW xx,07 are used following the function block used and the value of W_NB.

The data exchange is an exchange of words or double words. In the case of the 30 serie & 50 serie, the data area can be a bit area.

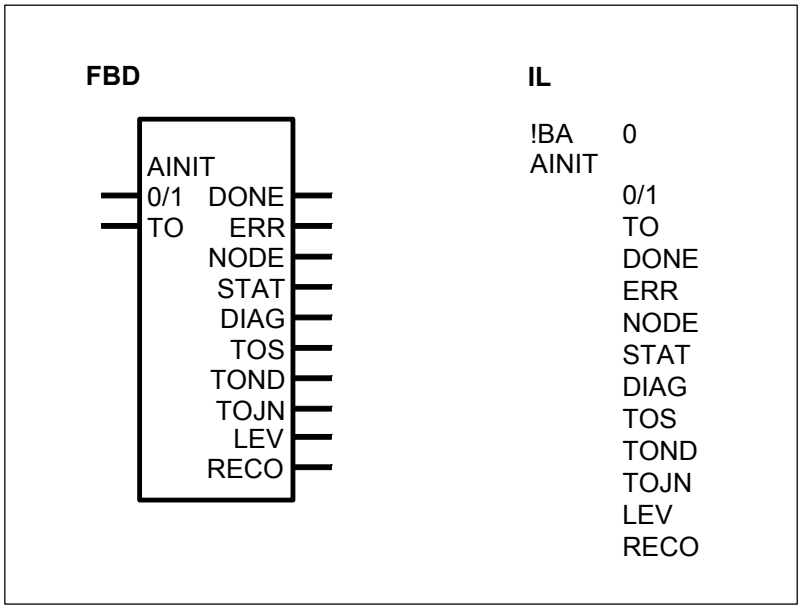
Several slaves can be used for the data exchange.

This function block used 431 words in the user program memory.

5 Communication functions

Communication functions	from pages C-116 to C-154	serie C[tler] 40	50	90	30
AINIT	Initialisation of the ARCnet controller			x	
APOLL	Transfer of the data package to the ARCnet controller			x	
AREC / ARECitem	ARCnet data package receiving			x	
ASEND / ASEND+	ARCnet data package sending			x	
MODBUS	MODBUS master	x	x		x
MODMASTK	MODBUS master for several interfaces	x	x	94	
REC / EMAS and RECvars	Receiving of ASCII characters and HEX values through a serail interface	x	x	x	x
SEND / DRUCK	Sending of ASCII characters and HEX values through a serail interface	x	x	x	x
SINIT	Initialization and configuration of the serial interfaces	x	x	x	x

AINIT INITIALIZATION OF THE ARCNET CONTROLLER



PARAMETERS

0/1	BINARY	%I, %M, %O, %K, %S	Initialization of the ARCnet controller with 0!1 edge
TO	WORD	%IW, %MW, %OW, %KW	Timeout in ms when sending data packages
DONE	BINARY	%O, %M	Initialization terminated
ERR	BINARY	%O, %M	Error has occurred
NODE	WORD	%OW, %MW	Own node number (station

			address)
STAT	WORD	%OW, %MW	Status register of the ARCnet controller
DIAG	WORD	%OW, %MW	Diagnosis register of the ARCnet controller
TOS	BINARY	%O, %M	Timeout has occurred during send operation
TOND	WORD	%OW, %MW	Node number of lost data package after timeout
TOJN	WORD	%OW, %MW	Job number of lost data package after timeout
LEV	WORD	%OW, %MW	Level of the send buffer
RECO	BINARY	%O, %M	Network reconfiguration is running (after loss of token)

The outputs STAT, DIAG, TOS, TOND, TOJN, LEV and RECO are updated after a successful initialization with each block call.

DESCRIPTION

ARCnet communication function block.

The function block AINIT initializes the ARCnet controller in the following way :
- Interrupt after reception of a data package
- Only short packages (a short package = 256 bytes)
- Data packages to all stations (broadcasts)

Important note :
If a PLC is used with an ARCnet interface, a certain section of the PLC TURBO program memory No. 2 is reserved for ARCnet.
If programs with more than 2 k instructions are executed, the system–dependent capacity utilization can possibly be increased by reason of the reduced TURBO memory No. 2 when *changes* are made *to a running program.*

There are *no* problems if
- the capacity utilization is less than 80 % *before* making changes to a running program or if
- the program length is less than 2 k instructions.

0/1 BINARY
A 0-1 edge at the input 0/1 causes a single initialization of the ARCnet controller. As long as the initialization has not been terminated (DONE = 0), a new 0-1 edge at the input 0/1 is ignored.

TO WORD
The timeout waiting time for sending data packages is specified in ms at the input TO. If a data package cannot be sent within this period, the sending operation of this data package is aborted and the package is lost. The output TOS indicates the loss of the package.

DONE BINARY
The output DONE indicates that the initialization has been terminated. This output has always to be considered together with the output ERR.
The following applies :

DONE = 1 **and** ERR = 0 :	The initialization has been terminated. No error has occurred.
DONE = 1 **and** ERR = 1 :	An error has occurred during initialization. The ARCnet controller has not responded within a period of 100 ms.

ERR BINARY
The output ERR indicates that the ARCnet controller has not responded after an initialization command within a period of 100 ms. This output has always to be considered together with the output DONE.
If an error has occurred, the following applies :
DONE = 1 **and** ERR = 1.

NODE WORD
The output NODE indicates the node number (station address) of its own after a successful initialization.

STAT WORD
The output STAT indicates the content of the status register of the ARCnet controller after a successful initialization.

DIAG WORD
The output DIAG indicates the content of the diagnosis register of the ARCnet controller after a successful initialization.

TOS BINARY
The output TOS indicates that the sending of a data package was not feasible within the timeout period (input TO) and that the data package is lost. Both the node number and the job number of the lost data package are available at the outputs TOND and TOJN, respectively.

TOND WORD
The output TOND indicates the node number of the lost data package after the timeout period has elapsed.

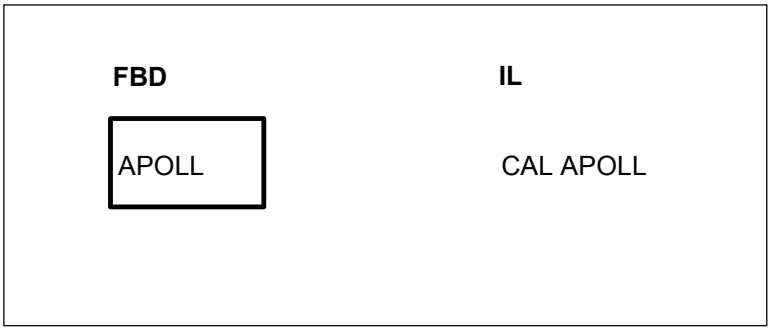
TOJN WORD
The output TOJN indicates the job number of the lost data package after the timeout period has elapsed.

LEV WORD
The output LEV indicates the level of the sending buffer after a successful initialization.

RECO BINARY
The output RECO indicates that the network is reconfiguring itself (RECO = 1) after a loss of token. The transition from RECO = 1 to 0 means that the reconfiguring

procedure has been terminated.
The outputs STAT, DIAG, TOS, TOND, TOJN, LEV and RECO are updated after a successful initialization with each block call.

APOLL TRANSFER DATA PACKAGES TO THE ARCNET CONTROLLER



DESCRIPTION

ARCnet communication function block.

The function block APOLL is intended for the following two tasks :
- It hands over a data package from the buffer storage to the ARCnet controller in order to send it off. The data packages are stored in the buffer storage by the function block ASEND.
- It monitors whether or not this data package is sent off within the timeout period configured in the function block AINIT.

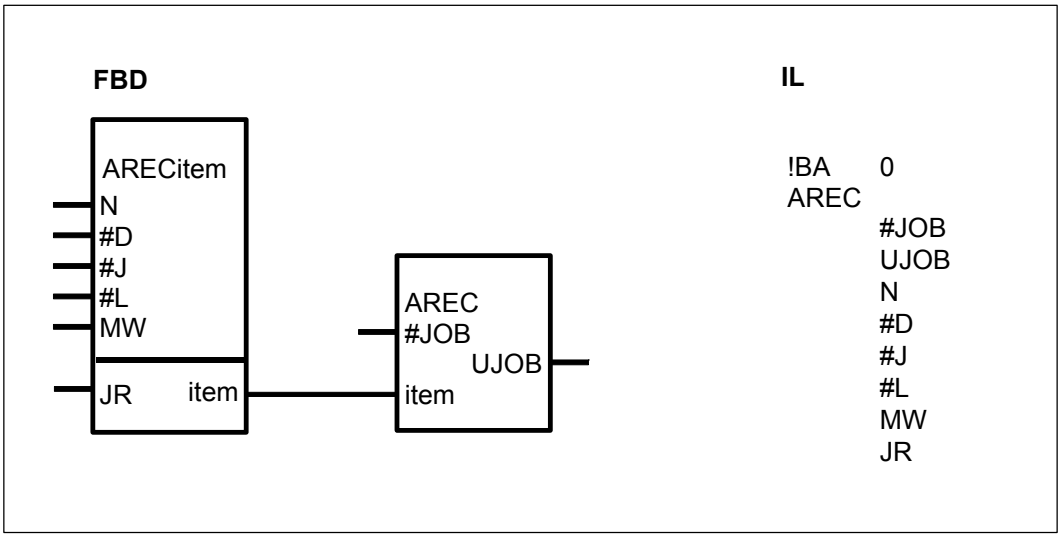
If the data package cannot be sent off by the ARCnet controller within the configured timeout period, the function block aborts the sending procedure. The data package is lost then. The loss of the data package is indicated by the output TOS of the AINIT block with the next block call.

If several APOLL blocks are configured in the user program, also several data packages can be sent off within one program cycle.

The ARCnet controller must have been initialized by the function block AINIT before a data package can be handed over from the storage buffer to it by the function block APOLL.

Important note :
If a PLC is used with an ARCnet interface, a certain section of the PLC TURBO program memory No. 2 is reserved for ARCnet.
If programs with more than 2 k instructions are executed, the system–dependent capacity utilization can possibly be increased by reason of the reduced TURBO memory No. 2 when *changes* are made *to a running program.*
There are *no* problems, if
- the capacity utilization is less than 80 % *before* making changes to a running program or if
- the program length is less than 2 k instructions.

AREC RECEIVE ARCNET DATA PACKAGES



PARAMETERS

#JOB	DIRECT CONSTANT	#, #H	Total number of jobs configured in the block
UJOB	BINARY	%O, %M	Unknown job received
N	WORD	%IW, %MW, %OW, %KW	Node number (station address) of the sender
#D	DIRECT CONSTANT	#, #H	DIN identification
#J	DIRECT CONSTANT	#, #H	Job number
#L	DIRECT CONSTANT	#, #H	Number of words of the user data to be received
MW	WORD	%IW, %MW, %OW, %KW	First word variable, as of the received user data are stored
JR	BINARY	%O, %M	Job received

In FBD : The item input can be duplicable. Each item input is directly linked to a ARECitem function block that corresponds to 1 data package.
In IL : N, #D, #J, #L, MW, JR can be duplicated all together by pack of 6 parameters.

DESCRIPTION

ARCnet communication function block.

The AREC function block has always to be used with the ARECitem function block.
The ARCnet controller must have been initialized by the function block AINIT before using the function block AREC.

The operating system reads the received data packages from the ARCnet controller interrupt-controlled and stores them into a storage buffer. The size of the storage buffer is 31 data packages.
The block reads the whole storage buffer contents and assigns the data packages to the configured jobs in the block.

With the same job the receiver gets user data which belong together logically. For unmistakable identification of a job the following parameters are necessary :

Node number of the sender,
DIN identification and
Job number.

The parameter values of the received data packages are compared to the parameter values of the configured jobs. If both values are equal, the user data of the data package beginning from the word variable MW are stored continuously and the output JR is set to 1. If several data packages are equal, the function block only evaluates that data package which has been received last (the latest data package).
If no data package from the storage buffer can be assigned to a configured job, the output belonging to this configured job is set to 0 (JR = 0).
The output UJOB indicates that a data package existing in the storage buffer could not be assigned to a job.
A user program must **not** contain **more than one** block call of AREC.

Important note :
If a PLC is used with an ARCnet interface, a certain section of the PLC TURBO program memory No. 2 is reserved for ARCnet.
If programs with more than 2 k instructions are executed, the system–dependent capacity utilization can possibly be increased by reason of the reduced TURBO memory No. 2 when *changes* are made *to a running program.*
There are *no* problems, if - the capacity utilization is less than 80 % *before* making changes to a running program or if
- the program length is less than 2 k instructions.

#JOB **DIRECT CONSTANT**
The total number of jobs configured in this function block is assigned to the input #JOB.
The following is valid : $1 \leq$ #JOB

UJOB **BINARY**
Output UJOB indicates that a data package was stored in the storage buffer which could not be assigned to any job.
The following applies :
UJOB = 1 : unknown job received
UJOB = 0 : no unknown job received

N **WORD**
The node number (station address) of the sender is specified for the configured job at input N.
The following is valid : $0 \leq N0...Nn-1 \leq 255$

#D **DIRECT CONSTANT**
The DIN identification is specified for the configured job at input #D.
The following is valid : $0 \leq$ #D ≤ 127
not allowed : 111 ($6F_H$)

#J **DIRECT CONSTANT**
The job number is specified for the configured job at input #J.
The following is valid : -32767 ≤ #J ≤ +32767

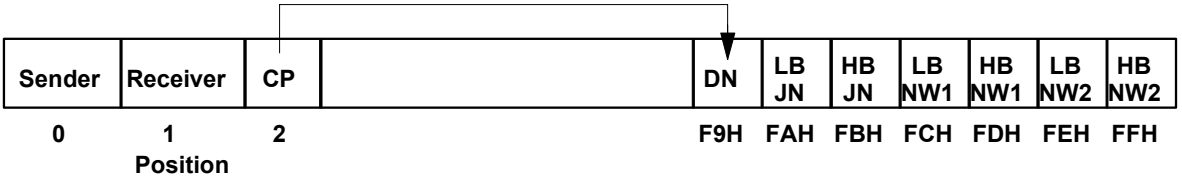
#L **DIRECT CONSTANT**
The number of user data words is specified for the configured job at input #L.
The following is valid : 0 ≤ #L ≤ 125

MW **WORD**
The starting word flag of the user data is specified for the configured job at input MW.

JR **BINARY**
Output JR (job received) indicates that a data package from the storage buffer has been assigned to this job. The user data of the data package are stored continuously beginning from the word variable MW.
The following applies :
JR = 1 : job received
JR = 0 : no job received

Structure of the ARCnet data package (short package = 256 bytes)

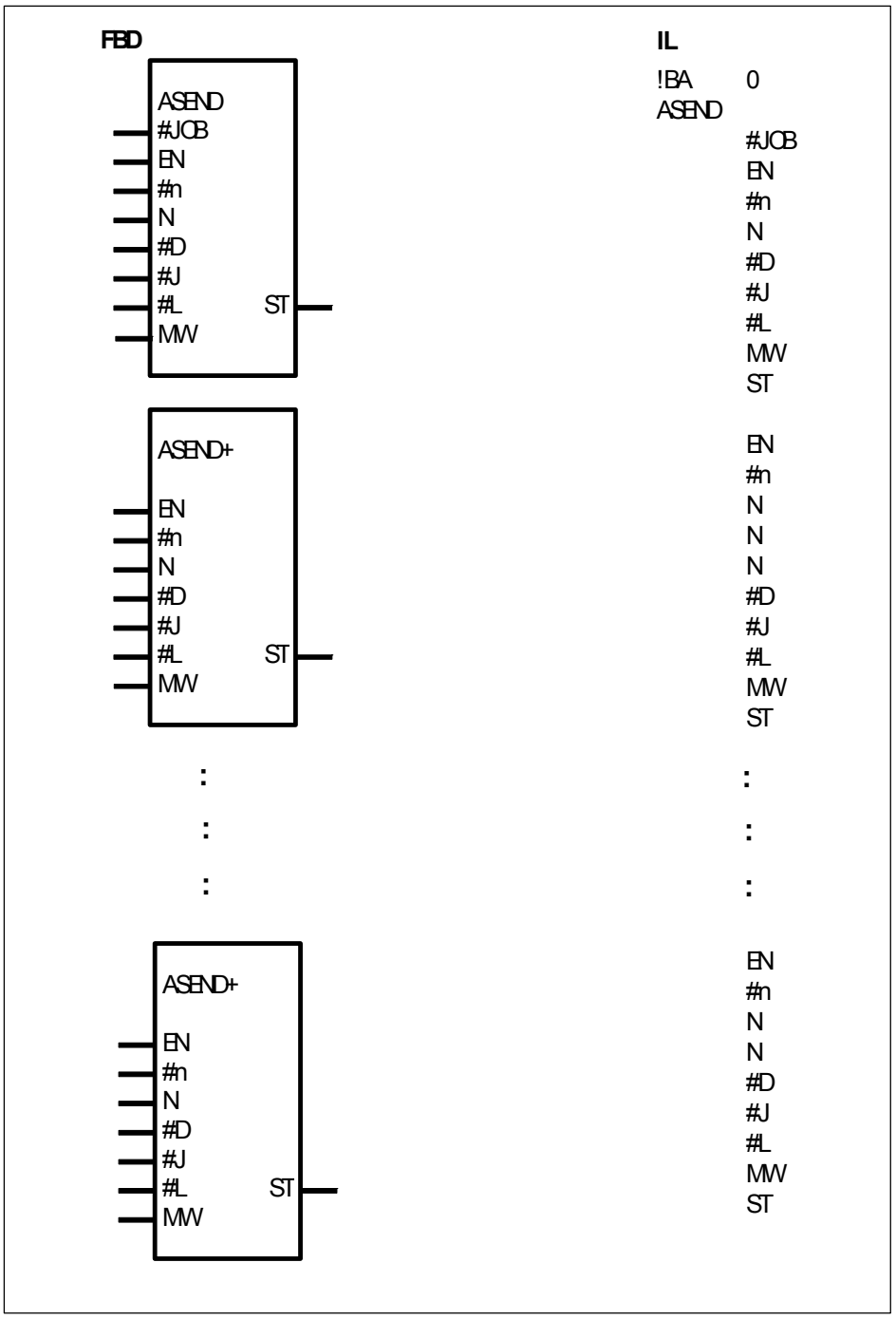
The ARCnet data package consists in :
- 3 bytes of control data : sender, receiver, CP (continuous pointer)
- max. 253 bytes of user data consisting in :
1 byte for DIN identification + 2 bytes for job number + 125 words for user data



Example of a configured job :
N : %KW 01,00 = 5
#D : #127 (7FH)
#J : #10 (0AH)
#L : #2
MW : %MW 10,00 = 258 (102H)
%MW 10,01 = 772 (304H)

Sender : Station address of the sender (05H)
Receiver : Station address of the receiver
CP : Continuous pointer, contains the position (F9H) of the first user data byte
DN : DIN identification (7FH)
LB JN : Low byte job number (0AH)
HB JN : High byte job number (00H)
LB NW1 : Low byte user data word 1 (02H)
HB NW1 : High byte user data word 1 (01H)
LB NW2 : Low byte user data word 2 (04H)
HB NW2 : High byte user data word 2 (03H)

ASEND SEND ARCNET DATA PACKAGES



PARAMETERS

#JOB	DIRECT CONSTANT	#, #H	Total number of jobs configured in the block
EN	BINARY	%I, %M, %O, %K, %S	Enable : Send job
#n	DIRECT CONSTANT	#, #H	Number of receivers of the data package
N	WORD	%IW, %MW, %OW, %KW	Node number (station address) of the receiver. The input can be duplicated.

#D	DIRECT CONSTANT	#, #H	DIN identification
#J	DIRECT CONSTANT	#, #H	Job number
#L	DIRECT CONSTANT	#, #H	Number of user data words to be sent
MW	WORD	%IW, %MW, %OW, %KW	First word variable of the user data words
ST	BINARY	%O, %M	Job has been stored in the storage buffer

The inputs EN, #n, N, #D, #J, #L and MW and the output ST are the parameters, that define a job. These parameters therefore appear both in the ASEND block and in the ASEND+ extension block.

DESCRIPTION

ARCnet communication function block.

The ASEND function block is intended for sending off jobs (data packages) via the ARCnet network. The configured jobs are stored in a buffer storage. From here, they are transferred to the ARCnet controller for sending off. The transport from the buffer storage to the ARCnet controller is managed by the APOLL function block. The transport is also carried out by the operating system in the idle period between two PLC cycles.
The maximum size of the storage buffer is 31 data packages.
The output ST indicates that the configured job has been stored in the storage buffer.

Important note
A user program must **not** contain **more than one** block call of ASEND. If more than *one* job have to be sent off, the ASEND block must be extended by the ASEND+ block. For *each* further job, one ASEND+ must be configured in the FBD *directly following* ASEND.
ASEND+ is only an extension block for ASEND, but not autonomous. For that reason, all used ASEND+ blocks must be configured directly following an ASEND block or then following each other. Under *no circumstances* may other CEs be inserted between ASEND and ASEND+ or ASEND+ and a following ASEND+ respectively.
For the same reason, the output ST may not be connected by a line with another CE. A variable must always be defined at the output ST.
ASEND+ blocks only exist in FBD. During the translation, the ASEND block and all the following ASEND+ are combined into *one* ASEND-IL function block.
The ARCnet controller must have been initialized by the function block AINIT before data packages can be stored in the storage buffer by the function block ASEND.

Important note :
If a PLC is used with an ARCnet interface, a certain section of the PLC TURBO program memory No. 2 is reserved for ARCnet.
If programs with more than 2 k instructions are executed, the system–dependent capacity utilization can possibly be increased by reason of the reduced TURBO

memory No. 2 when *changes* are made *to a running program.*
There are *no* problems, if - the capacity utilization is less than 80 % *before* making changes to a running program or if
- the program length is less than 2 k instructions.

#JOB DIRECT CONSTANT
The *total number* of jobs configured in the ASEND and ASEND+ function blocks is specified at the input #JOB.
The following is valid : 1 ≤ #JOB

EN BINARY
Dependent on the enable input EN, the configured job is stored as a data package in the storage buffer and sent off then.
The following applies :

EN = 0 :	The configured job is not stored in the storage buffer and thus not sent off.
EN = 0-1 edge :	The configured job is stored as a data package in the storage buffer and sent off then.
EN = 1 :	The configured job is stored in the storage buffer and sent off, *only if* the user data of the job have changed.

#n DIRECT CONSTANT
The number of receivers for the configured job is specified at the input #n. The node numbers of the receivers are configured at the inputs N, where : 1 ≤ #n ≤ 15

N WORD
If the same job has to be sent to several receivers, the node numbers (station addresses) of the receivers are defined at the inputs N.
N0 : Node number = 0 :
If the node number is ”0” at the input N0, the job is sent to all stations (Broadcast).
N1...Nn-1 : Node number = 0 :
If the node number is ”0” at the inputs N1...Nn-1, no broadcast is sent.
If one of the receivers is not able to receive data, the data package is lost for this receiver.
The following applies : 0 ≤ N ≤ 255

#D DIRECT CONSTANT
The DIN identification is specified for the configured job at input #D. The value 111 ($6F_H$) is not allowed here, because it is reserved for programming and test telegrams.
The following applies : 0 ≤ #D ≤ 127 not allowed : 111 ($6F_H$)

#J DIRECT CONSTANT
The job number is specified for the configured job at input #J.
The following applies : -32767 ≤ #J ≤ +32767

#L DIRECT CONSTANT
The number of user data words is specified for the configured job at input #L.
The following applies : 0 ≤ #L ≤ 125

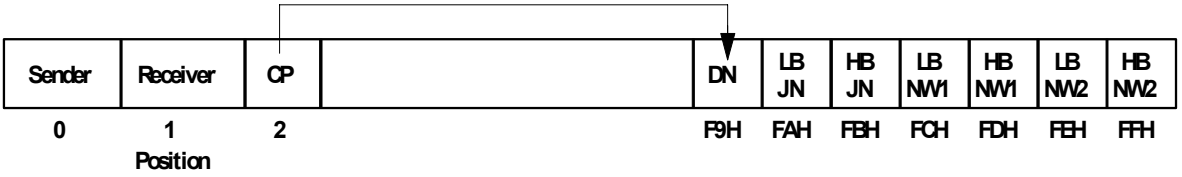
MW WORD
The starting word flag of the word flag area to be sent off is specified at the input MW. The data of this flag area are sent off as user data in this job.

ST BINARY
The output ST (status) indicates that the configured job has been stored in the storage buffer.
The following applies :
ST = 0 : The configured job has not been stored in the storage buffer.
ST = 1 : The configured job has been stored in the storage buffer.

Note
A variable must be specified at the output ST, i.e. the output may not be connected by a line with another CE.

Structure of the ARCnet data package (short package = 256 bytes)

The ARCnet data package consists in :
- 3 bytes of control data : sender, receiver, CP (continuous pointer)
- max. 253 bytes of user data consisting in :
1 byte for DIN identification + 2 bytes for job number + 125 words for user data

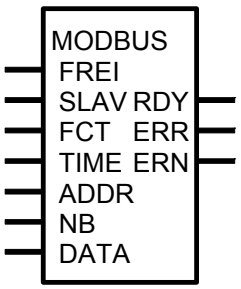


Example of a configured job :
N : %KW 01,00 = 5
#D : #127 (7FH)
#J : #10 (0AH)
#L : #2
MW : %MW 10,00 = 258 (102H)
%MW 10,01 = 772 (304H)

Sender : Station address of the sender (05H)
Receiver : Station address of the receiver
CP : Continuous pointer, contains the position (F9H) of the first user data byte
DN : DIN identification (7FH)
LB JN : Low byte job number (0AH)
HB JN : High byte job number (00H)
LB NW1 : Low byte user data word 1 (02H)
HB NW1 : High byte user data word 1 (01H)
LB NW2 : Low byte user data word 2 (04H)
HB NW2 : High byte user data word 2 (03H)

MODBUS MODBUS MASTER

FBD



IL

```
CAL MODBUS (FREI,SLAV,FCT,TIME,ADDR,
            NB,DATA,RDY,ERR,ERN)
```

PARAMETERS

FREI	BINARY	%I,%O,%M,%K,%S	Enable signal for one communication (rising edge)
SLAV	WORD	%IW,%OW,%MW,%KW	Slave number
FCT	WORD	%IW,%OW,%MW,%KW	Function code
TIME	WORD	%IW,%OW,%MW,%KW	Time–out for MODBUS communication
ADDR	WORD	%IW,%OW,%MW,%KW	Address in the slave
NB	WORD	%IW,%OW,%MW,%KW	Number of data to send or to read
DATA	WORD BINARY	%IW,%OW,%MW,%KW %I,%O,%M	Data to send to the slave or to write with the data received from a slave MODBUS
RDY	BINARY	%O,%M	Ready, communication in progress
ERR	BINARY	%O,%M	Communication error
ERN	WORD	%OW,%MW	Communication error, detail of error

DESCRIPTION

MODBUS master communication function block

The central unit is a master on a MODBUS network and can communicate with other products with MODBUS protocol.
The function MODBUS MASTER in the central unit is valided by :
- system constant KW 00,06 = 100
- connection between pins 7 and 6 on the connector of the serial interface

Several function blocks MODBUS can be used in one user program.

The MODBUS protocol is a master/slave protocol. The master sends a frame to a slave and waits for the answer (a time–out is defined). Binary or numeric data can be read or written in a slave.

The area of data in the master is chosen by the address of the first variable. The size of area is necessary for sending or receiving. Reading or writing data are done automatically from these areas.

FREI BINARY
A rising edge at the FREI input leads to an output of a request to a slave MODBUS, provided that the block is ready to do this (RDY = 1).
If a rising edge appears at the FREI input although the RDY output is equal to 0, i.e. the block is not ready for a new MODBUS communication, the rising edge will be ignored. Therefore, no new MODBUS communication can be started as long as the RDY output is 0.

SLAV WORD
Address of the slave which receives the request.
Value : $0 \leq ADDR \leq 255$
In case of address 0 (ADDR = 0), all slaves on the MODBUS network will read the frame.

FCT WORD
The function depends on the type of parameters and if it is reading or writing.
Value :
- 1 (01H): reading n bits
- 2 (02H): reading n bits
- 3 (03H): reading n words
- 4 (04H): reading n words
- 5 (05H): writing one bit
- 6 (06H): writing one word
- 7 (07H): fast reading of 8 bits
- 8 (08H): Diagnosis/initialization
- 15 (0FH): writing n bits
- 16 (10H): writing n words

The other function codes are not supported by by the central units 30serie, 40 serie and 50 serie. In case of a wrong function code, an error 1 is generated in the word ERN.

TIME WORD
Time–out for the communication (maximum time for an answer of the slave MODBUS).
The value is given in milliseconds.
Cycle time (KD 00,00) < TIME < 32767
In case of a time–out, the output ERN provides the value of 9.

ADDR WORD
Address of data in the slave memory to read or write.

NB WORD
Number of data to read or write in the slave.

This number defines also the size of the data area in the master to send to the slave or to receive from the slave.

DATA WORD, BINARY
DATA defines the first variable of the data area in the master. The size of this area depends on the NB parameter.
Different cases are possible according to the function code and the operand :

Reading :	ADDR	DATA	result
	word	word	idem
	bit	bit	idem
	bit	word	same as the function block PACK
	word	bit	The first bit of the word is ranged in the first bit of the group number. Example : data = M 00,07 : The first bit of the read word will be written in M 00,00.
Writing :	ADDR	DATA	result
	word	word	idem
	bit	bit	idem
	bit	word	same as the function block PACK
	word	bit	same as the function block UNPACK

RDY BINARY
The output RDY (ready) indicates whether a MODBUS communication is in progress or not. As long as a communication is in progress, the output RDY is equal to 0. The function block can only be used if RDY = 1.

ERR BINARY
The output ERR indicates an error occurred during communication. The word output ERN indicates the details of the error.

If	ERR = 1	->	error,
	ERR = 0	->	no error or communication in progress.

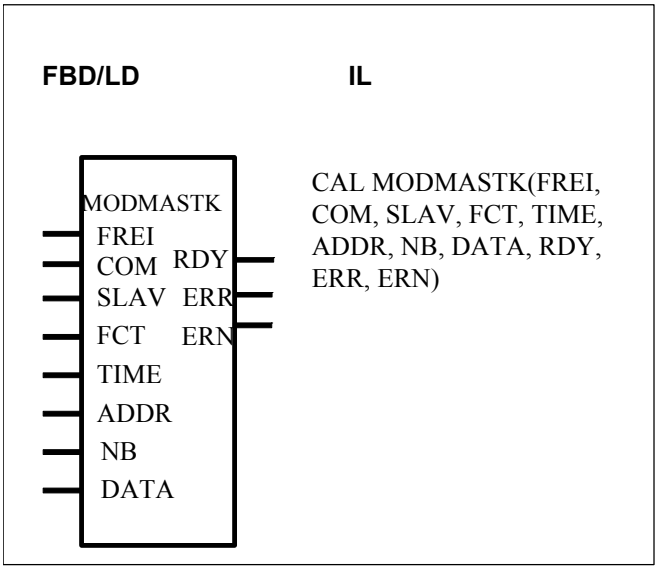
The error is clear after one cycle time.

ERN WORD
Details of error :

0 :	no error
1 :	unknown function code
2 :	address error
3 :	data error
9 :	time–out
10 :	checksum error

The error is clear after one cycle time.

MODMASTK MODBUS master several interfaces



PARAMETERS

FREI	BINARY	%I,%O,%M,%K,%S	Enable signal for one communication (rising edge)
COM	WORD	%IW,%OW,%MW,%KW	Communication interface (always=2)
SLAV	WORD	%IW,%OW,%MW,%KW	Slave number
FCT	WORD	%IW,%OW,%MW,%KW	Function code
TIME	WORD	%IW,%OW,%MW,%KW	Time–out for MODBUS communication
ADDR	WORD	%IW,%OW,%MW,%KW	Address in the slave
NB	WORD	%IW,%OW,%MW,%KW	Number of data to send or to read
DATA	WORD BINARY	%IW,%OW,%MW,%KW, %I,%O,%M	Data to send to the slave or to write with the data received from a slave MODBUS
RDY	BINARY	%O,%M	Ready, communication in progress
ERR	BINARY	%O,%M	Communication error
ERN	WORD	%OW,%MW	Communication error, detail of error

DESCRIPTION

MODMASTK master communication function block

The central unit is a master on a MODBUS network and can communicate with other products with MODBUS protocol.
The function MODMASTK in the central unit is valided by :
- system constant KW 00,06 = 100

The MODBUS protocol is a master/slave protocol. The master sends a frame to a slave and waits for the answer (a time–out is defined). Binary or numeric data can be read or written in a slave.

The area of data in the master is chosen by the address of the first variable. The size of area is necessary for sending or receiving. Reading or writing data are done automatically from these areas.

FREI BINARY
A rising edge at the FREI input leads to an output of a request to a slave MODBUS, provided that the block is ready to do this (RDY = 1).
If a rising edge appears at the FREI input although the RDY output is equal to 0, i.e. the block is not ready for a new MODBUS communication, the rising edge will be ignored. Therefore, no new MODBUS communication can be started as long as the RDY output is 0.

COM WORD
This parameter is always set to 2

SLAV WORD
Address of the slave which receives the request.
Value : 1 ≤ ADDR ≤ 254

FCT WORD
The function depends on the type of parameters and if it is reading or writing.
Value :
- 1 : reading n bits
- 2 : reading n bits
- 3 : reading n words
- 4 : reading n words
- 5 : writing one bit
- 6 : writing one word
- 7 : fast reading
- 15 : writing n bits
- 16 : writing n words

TIME WORD
Time–out for the communication (maximum time for an answer of the slave MODBUS).
The value is given in milliseconds.
Cycle time (KD 00,00) < TIME < 32767
In case of a time–out, the output ERN provides the value of 9.and ERR is 1

ADDR WORD
Address of data in the slave memory to read or write.

NB WORD
Number of data to read or write in the slave.
This number defines also the size of the data area in the master to send to the slave or to receive from the slave. NB=1….96

DATA WORD, BINARY
DATA defines the first variable of the data area in the master. The size of this area depends on the NB parameter.

RDY BINARY
The output RDY (ready) indicates whether a MODBUS communication is in progress or not. As long as a communication is in progress, the output RDY is equal to 0. The function block can only be used if RDY = 1.

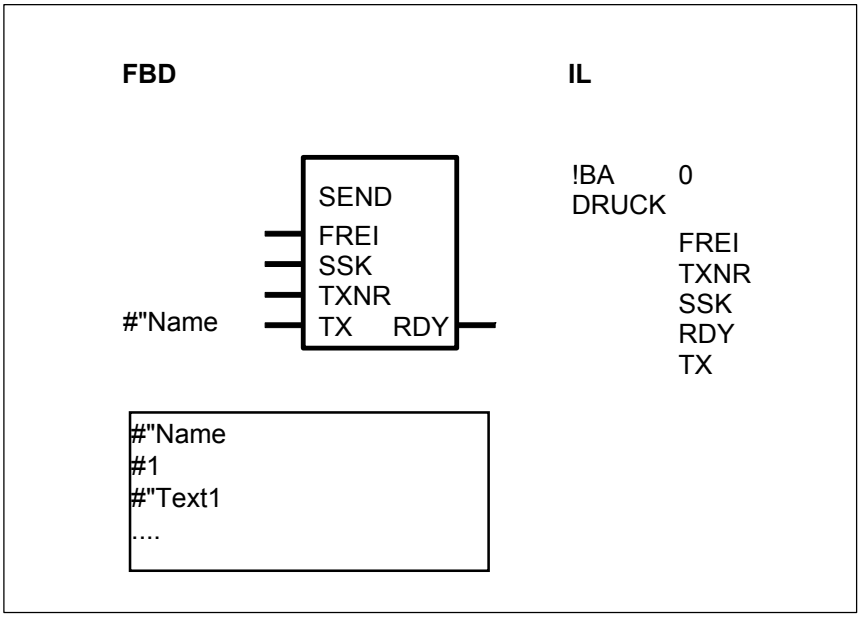
ERR BINARY
The output ERR indicates an error occurred during communication. The word output ERN indicates the details of the error.
If ERR = 1 -> error,
ERR = 0 -> no error or communication in progress.
The error is clear after one cycle time.

ERN WORD
Details of error :

0 :	no error
1 :	unknown function code
2 :	address error
3 :	data error
9 :	time–out
10 :	checksum error

The error is clear after one cycle time.

SEND SENDING OF ASCII CHARACTERS AND HEX VALUES THROUGH A SERIAL INTERFACE



PARAMETERS

FREI	BINARY	%O, %I, %S, %M	Enable signal for output of *one* text (0->1-edge)

SSK	WORD	%OW, %IW, %MW, %KW	Serial interface identification
TXNR	WORD	%OW, %IW, %MW, %KW	Number of the text to be output
TX	TXT	texts	Texts/operands **capable of duplication**
RDY	BINARY	%O, %I, %M	Ready
#"Name	TXT	texts	Comment name (in FBD)

DESCRIPTION

ASCII communication fonction block.

A CS31 central unit can send ASCII messages through its RS232 serial interface with the SEND function block.
Each message has an identification number and can be composed of ASCII texts and operands values. The texts and operand identifiers to be output are stored in the user program in the PLC directly by the SEND block. The numerical values to be output are conditioned for a diversity of representations by specifying a format identifier.

IMPORTANT NOTE : Initialization of the serial interface
Before using the SEND block, the serial interface used has to be initalized with the SINIT block.

Communication between several SEND blocks and the same serial interface
Several SEND blocks can use *the same* serial interface. If the serial interface is engaged by one of the SEND blocks, the other SEND blocks automatically wait until it is free again. The priority access to the serial interface when several SEND blocks simultaneously have access to the same interface, corresponds to the sequence in which the SEND blocks are called in the user program : the first SEND block located at the beginning of the user program is given access first. The processing sequence must be planned by an appropriate mutual interlocking of the SEND blocks.

Communication by a SEND and REC block with the same serial interface
A SEND block and a REC block (receiving of ASCII messages) can use the *same* serial interface without any special precaution.

FREI BINARY
If the block is ready (RDY = 1) and a 0 -> 1 edge appears at the FREI input, the text identified at the input TXNR is sent through the serial interface specified at the SSK input.

If a 0 -> 1 edge appears at the FREI input although the RDY output is equal to 0 (i.e. the block is not ready yet for a new transfer), the 0 -> 1 edge is *ignored*. Therefore, no new text transfer is started as long as the RDY signal is 0.

SSK WORD
Central units can have one or two RS232 serial interface. The number of the interface through which the text is to be output is specified at the SSK input :

SSK = 1 for COM1
SSK = 2 for COM2

TXNR WORD
The number of the text to be output is specified at the TXNR input : 1≤ TXNR ≤ 99

The number of the text to be output must be present at the TXNR input until the block indicates the end of text transfer with a 1 signal at its RDY output.

RDY BINARY
In the program cycle in which the block is called for the first time, and during the time when a text is output, RDY is equal to 0. As long as RDY is equal to 0, no new text output can be activated and all 0 -> 1 edge present at the FREI input are ignored and lost.

After the first call of the block or after termination of a text output, RDY is equal to 1 and the block is ready again for output of a new text.

FREI	RDY	TXNR	SSK	MEANING
0/1 edge	1	2	1	The text with the number 2 is output through interface 1
0/1 edge	0	2	1	0/1 edge is ignored because the block is not ready
no 0/1 edge	x	x	x	No output of a new text

=> The RDY signal can be used for example at the FREI input to activate a new text transfer.

TX ALL
In IL :
Texts and operands to be output are directly written at the TX inputs. The way to write the messages is described here below.

In FBD :
Text and operands to be sent are written in a comment window. Several comment windows can be created and each comment window has a "Name" that corresponds to the "Name" written at the TX inputs.

A comment window is called by the  button in the tool bar. You have to drag the mouse to select the rectangle area where the text must be written.
Comment may be inserted anywhere in the program.

One comment window can be composed of several messages.
The name of the comment window is NOT sent with the text and operands.
The way to write the messages is described here below and an example is given at the end.

- Storage of texts in the central unit :
Quantity : 1...99 messages (in one or several comment windows in FBD)
Length : Up to 256 characters (owing to the send buffer length = 256)

| Serie 90 | Serie 40 & 50 |

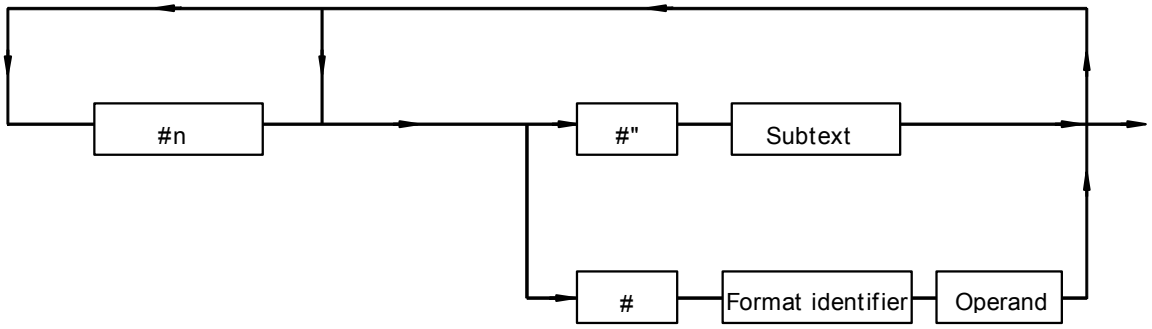
- Each text character counts as	1 character	1 character
- Each format identifier counts as	3 characters	
- Each bit operand counts as	1 character	1 character
- Each word operand counts as	2 characters	1 to 6 characters *
- Each double word operand counts as	4 characters	10 to 11 characters *

* depends on the format

When the program is started, the PLC checks the texts to determine whether or not the maximum length is exceeded.

- Syntax of texts

A text for the SEND block consists of :
- The text number
- One or several subtexts (optional)
- Operands with format identifier (optional)



#n : Number of the text to be entered as a direct constant (1...99).

#" : Start identifier for text input.

Subtext : All ASCII characters (from hexadecimal code 00 up to 7F).
The character ASCII <NUL> is an exception for central units serei 90. This character is used as a prefix for the format identifier in the send buffer.

Operand : Binary, word or double word operands whose values are output depending on the display format.

Format identifier : The PLC is capable of presenting the numerical values to be output on a screen or a printer in diverse ways. The format identifier specifies the type of the operand and the display format of its value. This is planned directly *before* the operand to be output and consists of three digits. The 1st digit from the left specifies the operand type. There are 3 operand types :

Binary : 1
Word : 2
Double word : 3

Numbers 2 and 3 define the display format.

Examples :

- Format identifier 103
  Digit 1 : 1 : Binary operand
  Digits 2 and 3 : 03 : Display format 03 (see table)

-Format identifier 204
  Digit 1 : 2 : Word operand
  Digits 2 and 3 : 04 : Display format 04 (see table)

-Format identifier 341
  Digit 1 : 3 : Double word operand
  Digits 2 and 3 : 41 : Display format 41 (see table)

- Possible display formats

All possible display formats are listed in the following table.
For series 40 & 50 :
- identifiers applicable to word data types : 01 to 16, 21 to 26, 33 to 36, 42 to 51, and 99
- identifiers applicable to double word data types : 05 to 16 and 41 to 62

For serie 90 : With the exception of the special indentifiers 98 and 99, all identifiers are applicable to the word and double word data types.

There are formats
- with leading zeros and
- with leading zeros substituted by blanks which are indicated in the table here below by -.

Format identifier	in series 40 & 50	in serie 90	Numerical example	ASCII output
01	x	x	0012345678	8
02	xx	xx		78
03	xxx	xxx		678
04	xxxx	xxxx		5678
05	xxxxx	xxxxx		45678
06	xxxx,x	xxxx,x	1102215	0221,5
07	xxx,xx	xxx,xx		022,15
08	xx,xxx	xx,xxx		02,215
09	x,xxxx	x,xxxx		0,2215
10	,xxxxx	,xxxxx		,02215
11	+/- xxxxx	+/- xxxxx	00331	+00331
12	+/- xxxx,x	+/- xxxx,x		+0033,1
13	+/- xxx,xx	+/- xxx,xx		+003,31
14	+/- xx,xxx	+/- xx,xxx		+00,331
15	+/- x,xxxx	+/- x,xxxx		+0,0331
16	+/- ,xxxxx	+/- ,xxxxx		+,00331
17		x	00234	----4
18		xx		---34
19		xxx		--234

Format identifier	in series 40 & 50	in serie 90	Numerical example	ASCII output
20		xxxx		--234
21	xxxxx	xxxxx		--234
22	xxxx,x	xxxx,x	00347	--347
23	xxx,xx	xxx,xx		--3,47
24	xx,xxx	xx,xxx		--,347
25	x,xxxx	x,xxxx		-,0347
26	,xxxxx	,xxxxx		,00347
27		+/- xxxxx		+--347
28		+/- xxxx,x		+--34,7
33	+/-xxx,xx		00347	+--3,47
34	+/-xx,xxx			+--,347
35	+/-x,xxxx			+-,0347
36	+/-,xxxxx			+,00347
37		xxxxxx	0012345678	345678
38		xxxxxxx		2345678
39		xxxxxxxx		12345678
40		xxxxxxxxx		012345678
41	xxxxxxxxxx	xxxxxxxxxx		0012345678
42	xxxxxxxxx,x	xxxxxxxxx,x	0011223344	001122334,4
43	xxxxxxxx,xx	xxxxxxxx,xx		00112233,44
44	xxxxxxx,xxx	xxxxxxx,xxx		0011223,344
45	xxxxxx,xxxx	xxxxxx,xxxx		001122,3344
46	xxxxx,xxxxx	xxxxx,xxxxx		00112,23344
47	xxxx,xxxxxx	xxxx,xxxxxx		0011,223344
48	xxx,xxxxxxx	xxx,xxxxxxx		001,1223344
49	xx,xxxxxxxx	xx,xxxxxxxx		00,11223344
50	x,xxxxxxxxx	x,xxxxxxxxx		0,011223344
51	,xxxxxxxxxx	,xxxxxxxxxx		,0011223344
52	+/- xxxxxxxxx	+/- xxxxxxxxx	0055667788	+0055667788
53	+/- xxxxxxxxx,x	+/- xxxxxxxxx,x		+005566778,8
54	+/- xxxxxxxx,xx	+/- xxxxxxxx,xx		+00556677,88
55	+/- xxxxxxx,xxx	+/- xxxxxxx,xxx		+0055667,788
56	+/- xxxxxx,xxxx	+/- xxxxxx,xxxx		+005566,7788
57	+/- xxxxx,xxxxx	+/- xxxxx,xxxxx		+00556,67788
58	+/- xxxx,xxxxxx	+/- xxxx,xxxxxx		+0055,667788
59	+/- xxx,xxxxxxx	+/- xxx,xxxxxxx		+005,5667788
60	+/- xx,xxxxxxxx	+/- xx,xxxxxxxx		+00,55667788
61	+/- x,xxxxxxxxx	+/- x,xxxxxxxxx		+0,055667788
62	+/- ,xxxxxxxxxx	+/- ,xxxxxxxxxx		+,0055667788
63		x	0087654321	---------1
64		xx		--------21
65		xxx		-------332

Format identifier	in series 40 & 50	in serie 90	Numerical example	ASCII output
66		xxxx		------4321
67		xxxxx		-----54321
68		xxxxxx		----654321
69		xxxxxxx		---7654321
70		xxxxxxxx		--87654321
71		xxxxxxxxx		--87654321
72		xxxxxxxxxx		--87654321
73		xxxxxxxxx,x	0012345678	--1234567,8
74		xxxxxxxx,xx		--123456,78
75		xxxxxxx,xxx		--12345,678
76		xxxxxx,xxxx		--1234,5678
77		xxxxx,xxxxx		--123,45678
78		xxxx,xxxxxx		--12,345678
79		xxx,xxxxxxx		--1,2345678
80		xx,xxxxxxxx		--,12345678
81		x,xxxxxxxxx		-,012345678
82		,xxxxxxxxxx		,0012345678
83		+/- xxxxxxxxxx		+--12345678
84		+/- xxxxxxxxx,x		+--1234567,8

Special format : Output of a word operand HEX value :

The value of a word operand is output directly as a hexadecimal value. Therefore, the value is not converted to ASCII before output.

98 Only the LOW BYTE (8 bits) of the word operand is output

99 The LOW BYTE of the word operand is output first, followed by its HIGH BYTE

Important : This special format is only permissible for the “WORD” data type.
Permissible format : 298 and 299
Inadmissible format : 198, 199, 398 and 399

- Input of texts

The following parts of the overall message are treated as independent operands :

- the text number e.g. # 1
- a subtext e.g. #” Text1
- a format identifier e.g. # 203
- an operand e.g. %MW 002,03
- a further subtext e.g. #” Text2

Input of special characters for screen or printer control :

Control characters such as “line feed” <LF> or “carriage return” <CR> are needed to arrange the message when output to a screen or printer. These special characters can be placed anywhere within a subtext. In the programming system, these special characters are entered by means of :

\Numerical value of the character

The numerical value of the character is specified as a three-digit decimal number.

Example :

The following output is to be made on a printer :

First line
Blank line
Second line

To do this, the following text must be planned :

First line <CR> <LF> <LF> second line

The following applies :

<CR> = 013
<LF> = 010

The text input in the programming system is as follows :

#”First line\013\010\010second line

Notes :

- The characters with the ASCII code $>20_H$ ($=32_D$) must be entered with the keyboard. As an example, the character ”!” can be entered with the keyboard, and not as \033.
- Characters, which are not special characters and which can not be entered with the keyboard, can be generated in the following way :

Press and hold down the <ALT> key, now type the numerical code (decimal code) on the numerical keypad of the keyboard, then release the <ALT> key.

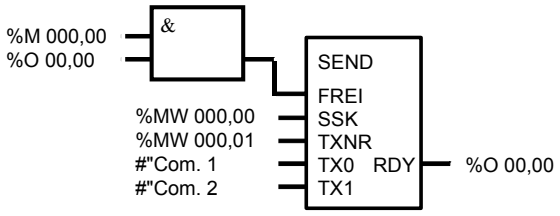
- The character with the ASCII code 255 is reserved for internal use of the programming software and must not be used otherwise.

- Example :

Text 1 : The machine is ready.
Text 2 : The machine is not ready.
Text 3 : Level is (%MW 001,01)m and temperature is (%MW 001,00)°C.

Note : Don't forget the SINIT function block to initialize the serial port

FBD



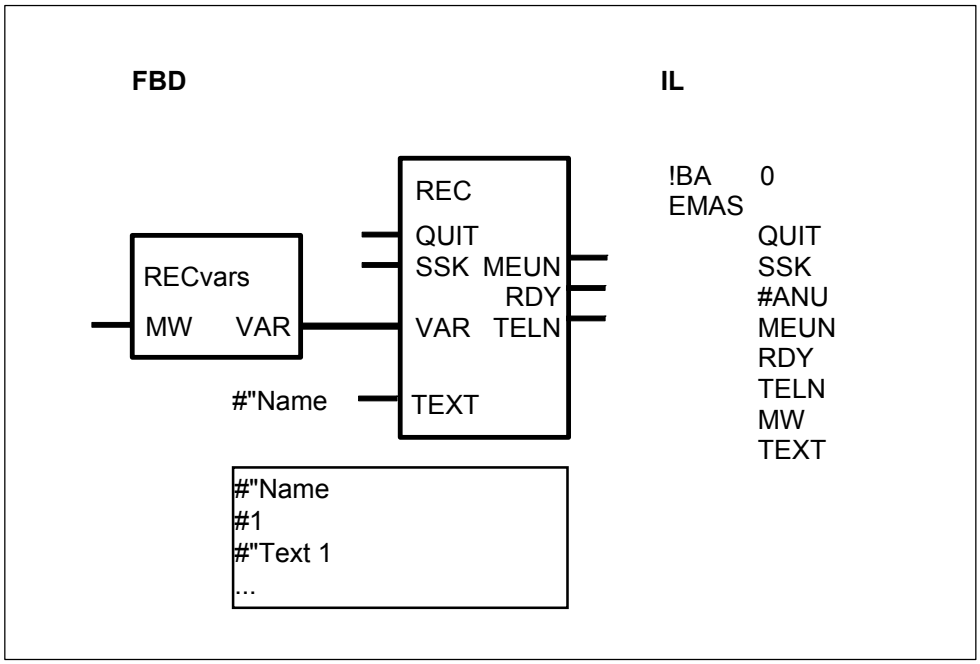

```
#"Com. 1
#1
#"\010\013The machine is ready.
#2
#"\010\013The machine is not ready.
```

```
#"Com. 2
#3
#"\010\013Level is
#202
%MW 001,01
#"m and temperature is
#203
%MW 001,00
#"°C.
```

IL

```
LD      %M 000,00
AND     %O 00,00
ST      %M 238,00

!BA     0
DRUCK
        %M 238,00
        %MW 000,00
        %MW 000,01
        %O 00,00
        #1
        #"\010\013The machine is ready.
        #2
        #"\010\013The machine is not ready.
        #3
        #"\010\013It is
        #202
        %MW 001,01
        #"m and temperature is
        #203
        %MW 001,00
        #"°C.
```

REC RECEIVING OF ASCII CHARACTERS AND HEX VALUES THROUGH A SERIAL INTERFACE



PARAMETERS

QUIT	BINARY	%O, %E, %M, %S, %K	Reception of telegrams not enabled.
SSK	WORD	%OW, %IW, %MW, %KW	Serial interface identification
#ANU	DIRECT CONSTANT	#, #H	Number of VAR for user information
TEXT	TXT	texts	Comparison telegram; **capable of duplication**
MEUN	BINARY	%O, %M	Data invalid
RDY	BINARY	%O, %M	Ready : telegram received
TELN	WORD	%OW, %MW	Number of the comparison telegram with which the received one agrees
MW	WORD	%OW, %MW	User data; **capable of duplication**
#"Name	TXT	Text	Comment name (in FBD)

DESCRIPTION

ASCII communication function block.

A CS31 central unit can receive ASCII messages through its RS232 serial interface with the REC function block. The REC function block is always linked to the RECvars function block.

The REC function block :
- receives telegrams through a serial interface of the PLC

- compares these telegrams to comparison telegrams stored in the user program
- and, if these agree, provides the user data of the telegram received at the block's outputs.

The received telegrams are fetched from the serial interface by an interface driver and are provided in a BUFFER for further processing by REC. The driver recognizes the end of the telegram by the end of telegram character. This end of telegram character is planned in the SINIT block.

IMPORTANT NOTE : Initialization of the serial interface
Before using the REC block, the serial interface used has to be initalized with the SINIT block.

Communication between several REC blocks and the same serial interface
- REC blocks of a user program which access *the same* serial interface must be interlocked so that only ever *one* REC block is active. If this is not done, telegrams may be processed by the wrong REC and declared invalid.
- If both user program 1 and also user program 2 contain REC blocks which access *the same* serial interface, they must be interlocked so that only ever *one* REC block is active. If this is not done, telegrams may be processed by the wrong REC and declared invalid.
A telegram loss can be avoided by interlocking of the REC blocks. Interlocking must be planned so that only the REC block is enabled for which the telegram arriving through the interface is intended.

Communication by an REC block and a SEND block with the same serial interface
An REC and a SEND block can use the *same* serial interface without special precautions having to be taken.

QUIT BINARY
The input QUIT controls reception of telegrams and also serves the purpose of acknowledgement in the event of an error occurring.
QUIT = 0 : Reception of telegrams enabled.
QUIT = 1 : Reception of telegrams *not* enabled. Acknowledgment after reception of an invalid telegram.
If agreement with *none* of the stored comparison telegrams is ascertained on comparison of a received telegram, the REC automatically assumes the "error" state. In this case, REC no longer processes any new telegrams until the error is acknowledged with a 1 signal at the input QUIT and reception of telegrams is enabled again (next cycle) with a 0 signal at the input QUIT.

SSK WORD
The number of the interface through which the block receives its telegrams is specified at the SSK input (interface identifier).
The following applies :
COM1 : number = 1
COM2 : number = 2

#ANU DIRECT CONSTANT
The number of outputs MW at which the block provides the received user information is specified at the input #ANU (number of user information items). This is specified as a direct constant(only used in Instruction List).

MEUN BINARY
The output MEUN (flag invalid) indicates whether or not the data at the outputs MW is valid or invalid.
If a telegram is received and processed properly, the data at the outputs MW is declared *valid*. The data at the outputs MW is declared *invalid* if the received telegram does not agree with any of the stored comparison telegrams or if the received telegram cannot be processed properly.
MEUN = 0 -> Data at the outputs MW is *valid*
MEUN = 1 -> Data at the outputs MW is *invalid*

RDY BINARY
The output RDY (ready) indicates that a telegram has been received and processed.
The output RDY does not provide any information as to whether or not a valid or invalid telegram has been received.
RDY = 0 -> Still no telegram has been received
RDY = 1 -> A telegram has been received and processed

Brief overview of the block's parameters:
QUIT
RDY
MEUN

QUIT	MEUN	RDY	Meaning
1	0	0	The EMAS is disabled by QUIT = 1. In doing so, the outputs MEUN and RDY are permanently set to 0.
0	1	0	EMAS is enabled for reception, but still no telegram has been received and eva luated.
0	0	1	EMAS has received a valid telegram and is ready to receive a new telegram.
0	1	1	EMAS has received an invalid telegram. An acknowledgement at the QUIT input is necessary in order to be able to receive a new telegram. QUIT: 0->1
0->1	0	0	Acknowledgement after reception of an invalid telegram. After acknowledge ment, the EMAS is enabled again by QUIT = 0

TELN WORD
If a valid telegram is received, the number of the affiliated comparison telegram is output through the output TELN (telegram number).

MW WORD
The output MW can be duplicated. The user data communicated in the telegram currently received is output through the MW parameters. This user data may consist of numerical values or any characters. This depends on which kind of dummy parameters have been planned in the comparison telegram. The user data of a telegram is stored beginning with the first MW parameter and in the sequence in

which they are planned in the comparison telegram. As many outputs MW must be provided as are sufficient for the telegram with the *most* user data.

TEXT ALL
In IL :
The comparison telegrams to be stored in the PLC program are specified at the inputs TEXT. The block is capable of processing from 1 to 99 telegrams. One telegram occupies 2 inputs, each telegram number being specified at one input and the actual telegram text being specified at the next one. The exact syntax and handling of the comparison telegrams are described here below.

In FBD :
Text and operands to be stored are written in a comment window. Several comment windows can be created and each comment window has a "Name" that corresponds to the "Name" written at the TX inputs.

A comment window is called by the button in the tool bar. You have to drag the mouse to select the rectangle area where the text must be written.
Comment may be inserted anywhere in the program.

One comment window can be composed of several messages.
The name of the comment window is NOT stored with the text and operands.
The way to write the messages is described here below and an example is given at the end.

DETAILED DESCRIPTION OF COMPARISON TELEGRAMS
1...99 comparison telegrams are stored directly after the REC block.

The comparison telegrams serve to identify
- the current telegrams received
- and the user data contained in the telegrams received

Each stored comparison telegram has a telegram number to identify it and may comprise up to 255 characters.

The comparison telegrams consist of :
- ASCII characters serving only to identify the telegram received,
- Dummy parameters for the user information to be received and to be output through the block's outputs.

As regards the dummy parameters for the user information, REC function block distinguishes between dummy parameters for *digits* and dummy parameters for *characters*.
Dummy parameter for digits : # (1 # per digit)
Dummy param. for characters : * (1 * per character/byte)

Dummy parameter for digits :
For each dummy digit parameter (#) of the comparison telegram, REC expects precisely *one* ASCII coded decimal digit in the telegram to be received. Up to 5 dummy digits constitute one dummy parameter group. Such a group of dummy digits represents the numerical value of a decimal number comprising up to 5 digits.

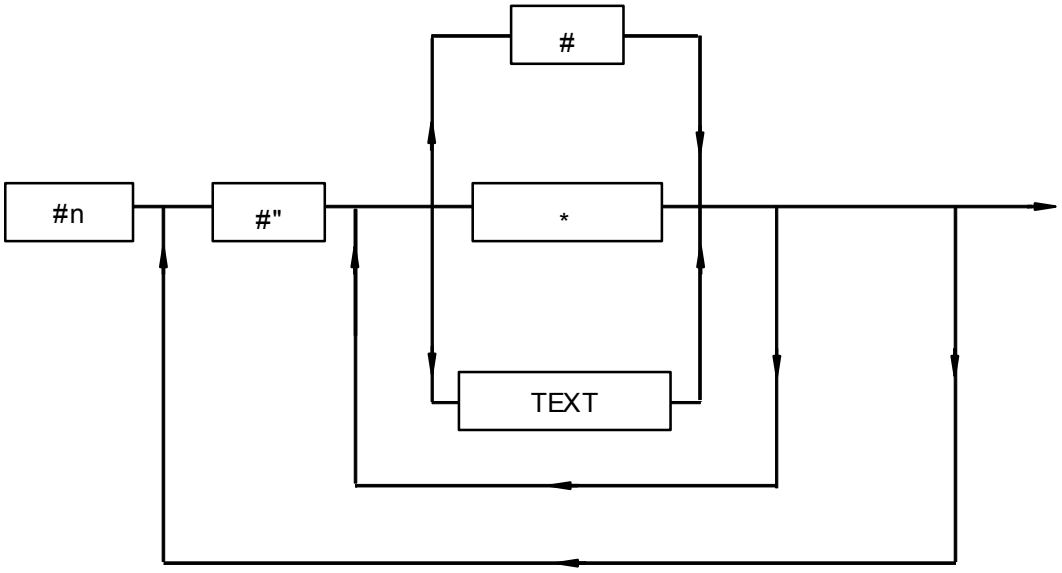
No dummy parameter is specified for the decimal number's sign because REC takes it into account automatically. The REC allocates one user information output to each numerical value belonging to a dummy parameter group.

E.g. :	Decimal number	Dummy parameters
	1234	####
	+1234	####
	-1234	####

The REC block checks the received decimal number in relation to its significant range. Only numbers within the ±32767 range can be processed in the PLC. If the received decimal number exceeds the significant range, the REC will automatically insert the maximum respective limit. The limit for a positive number is +32767 and the limit is -32767 for a negative number.

Dummy parameter for characters :
For each dummy character (*) of the comparison telegram, REC expects *any* one character/byte in the telegram to be received. These may comprise ASCII characters of letters, but also all other hex values from 0...FF.
The length of a dummy character group is up to 255. If this were the case, the complete comparison telegram would consist of dummy characters only.
REC allocates the characters/bytes received without change and successively to its user information outputs MW.

Syntax diagram : Structure of comparison telegrams



#n : Successive telegram number (direct constant 1...99)

#" : Start identifier for text input

** :* Dummy parameter for character/byte

 # : Dummy parameter for digits

TEXT : All ASCII characters 01 to FF except * and #

Input of comparison telegrams

- Each comparison telegram consists of :
  the telegram number
  the telegram text

The telegram number and the telegram text are two separate operands. This is why the telegram number and the telegram text occupy separate inputs in the FBD symbol of the REC block. Therefore, two inputs are needed for one comparison telegram.

Example :
First TEXT Input : #1 (No. of the first comparison telegram)
Second TEXT Input : #”PRINT### IDENTIFIER**** (Text of the first comparison telegram)

- Apart from the ASCII characters for * and #, all ASCII characters are possible in the telegram text.

- When entering special ASCII characters such as “start of line” <CR>, the following must be observed : special characters are entered by :
  \Numerical value of the character

The character’s numerical value is specified as a three-digit decimal number.

Example :
The following telegram text is to be compared : Temperature <CR> boiler 1
The following applies : <CR> = 013
The text input in the programming system is as follows :
  #”temperature\013boiler 1

Note :
- The characters with the ASCII code $>20_H$ or $>32_D$ must be entered with the keyboard. As an example, the character ”!” must be entered with the keyboard, and not as \033.
- Characters, which are no special characters and which also could not be entered with the keyboard, can be generated in the following way :
Press and hold down the <ALT> key, now type the numerical code (decimal code) on the numeric keypad of the keyboard, then release the <ALT> key.
- The character with the ASCII code 255 is reserved for internal use of the programming software and must not be used otherwise.

RECEIVING / PROCESSING TELEGRAMS
Receiving telegrams are controlled by the QUIT input which also acknowledge transmission errors.

The REC compares character by character a received telegram to the comparison telegrams stored. In doing so, agreement between the dummy parameters in the comparison telegram and the affiliated current characters in the received telegram is checked.

Important : For REC, the telegram end identifier indicates the end of the telegram to be received. This is why the telegram end identifier must *not* occur *within* a telegram. This also applies to user characters received on the basis of a dummy parameter (# or *).

Agreement
If the REC ascertains agreement between the received telegram and one of its comparison telegrams, the following applies to the outputs :
MEUN = 0, because it has been possible to receive and process the telegram *properly*
RDY = 1, because a telegram has been *received* and *processed*
TELN = Number of the relevant comparison telegram
MW = Current user data from the telegram received

The coherent digits defined by a dummy digit group in the comparison telegram are read out of the received telegram, are combined in *one* numerical value and this is output to *one* word operand (user information output). Text characters marked by dummy text characters (*) in the comparison text are read out of the received telegram and each *single* character/byte is output without change to a word operand (user information output).

The user information is allocated to the outputs MW in the same sequence as the user information occurs in the currently received telegram (see also examples).

The maximum number of user information word operands to be output is specified at the input #ANU. The maximum number results from the telegram with the most user information word operands to be reserved (up to 256 word operands, in which case the telegram would consist of user information *only*).

Special case :
If more user information is planned in a comparison telegram than user information outputs are available on the block, i.e. specified at the input #ANU, the received user data is output through the user information outputs until the end of the outputs has been reached. The remaining user data received will not be output.

The following applies in this error case :
MEUN = 1, because it has *not* been possible to process the telegram properly
RDY = 1, because a telegram has been received and processed
TELN = Text No. of the relevant comparison telegram
MW = The part of the current user data from the received telegram for which the number of planned user information outputs suffices.

No agreement
The received telegram agrees with *none* of the stored comparison telegrams.
The following then applies to the outputs :
MEUN = 1, because the received telegram is invalid
RDY = 1, because a telegram has been received and processing of it has been terminated
The REC is now in the “ERROR” state. In this case, REC does not process any new telegram until the error is acknowledged with a 1 signal at the QUIT input and the reception of telegrams is enabled again (next cycle) with a 0 signal at the QUIT input.

Acknowledgement of the error
Set QUIT = 1 and then, in the next cycle QUIT = 0.
After reception of telegrams has been enabled, it may happen that still no telegram is available for reception or that a telegram has not yet been received and evaluated completely. In this case, the outputs are set as follows
• MEUN = 1 and
• RDY = 0
until a telegram has been received and processed completely.

EXAMPLE OF A COMPARISON TELEGRAMS

Text 1 : RO104
Text 2 : DO##”#”##

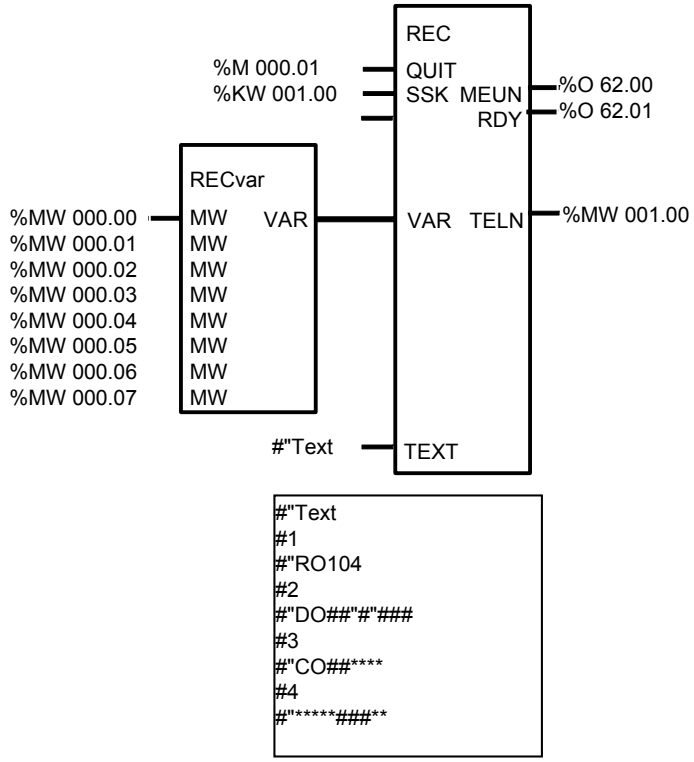
Special case : terminate text input and restart
If the text character (”) is used and a dummy parameter for digits (#) follows this text character, the input must be terminated after the text character (”). A subsequent text or dummy parameter begins with the input identifier of a new text input (#”).

Text 3 : CO##****
Text 4 : *****###**

Note : The texts designated TEXT1, TEXT2 and TEXT3 in the example are permanently planned texts which REC doesn’t output to user information outputs. They serve exclusively to identify an arriving telegram.
Any sequence of texts and dummy parameters for characters and digits is possible.

Note : don't forget the SINIT function block to initialize the serial port

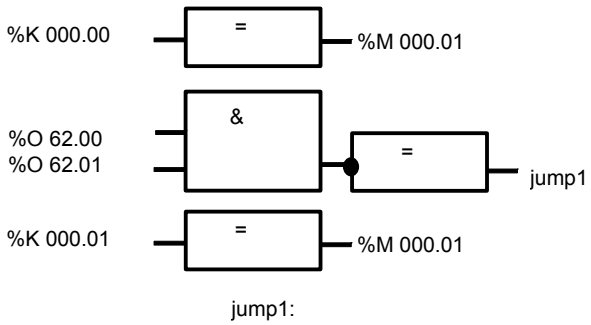
FBD



IL

```
!BA     0
EMAS
        %M 000.01
        %KW 001.00
        #1
        %O 62.00
        %O 62.01
        %MW 001.00
        %MW 000.00
        %MW 000.01
        %MW 000.02
        %MW 000.03
        %MW 000.04
        %MW 000.05
        %MW 000.06
        %MW 000.07
        #1
        #"RO104
        #2
        #"DO##"#"###
        #3
        #"CO##****
        #4
        #"*****###**
```

Example of acknowledgment with the QUIT parameter :



The following telegrams have been received and evaluated :
Example 1
RO104<CR>
The number 1 is output through the TELN output.
The MW0...MWn parameters are *not* written because no user information is planned.
As the received telegram has been evaluated and agrees with a comparison telegram, the following applies :
MEUN = 0
RDY = 1
TELN = 1

Example 2
DO+12”457”<CR>
The number 2 is output through the TELN output.
The value 12 is present at the MW0 parameter as the contents of a word flag.
The value 457 is present at the MW1 parameter as the contents of a word flag.
As the received telegram has been evaluated and agrees with a comparison telegram, the following applies :
MEUN = 0
RDY = 1
TELN = 2
MW0 = 12
MW1 = 457

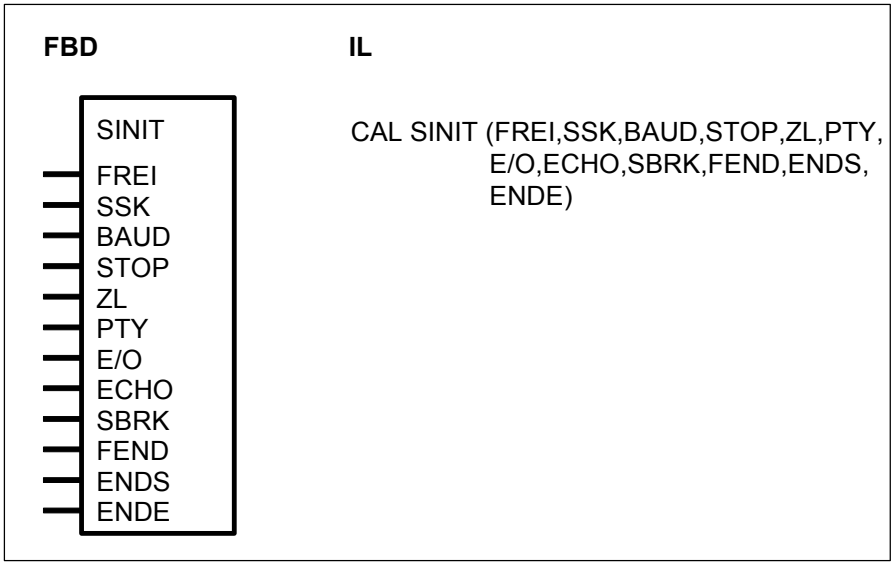
Example 3
CO-11AUTO<CR>
The number 3 is output through the TELN output.
-11 is present at the MW0 parameter as the contents of a word flag.
41_H (A in ASCII) is present at the MW1 parameter as the contents of a word flag.
55_H (U in ASCII) is present at the MW2 parameter as the contents of a word flag.
54_H (T in ASCII) is present at the MW3 parameter as the contents of a word flag.
$4F_H$ (O in ASCII) is present at the MW4 parameter as the contents of a word flag.
As the received telegram has been evaluated and agrees with a comparison telegram, the following applies :
MEUN = 0
RDY = 1
TELN = 3
MW0 = -11
MW1 = A
MW2 = U
MW3 = T
MW4 = O

Example 4
TEMPO120KM<CR>
The number 4 is output through the TELN output.
54_H (T in ASCII) is present at the MW0 parameter as the contents of a word flag.
45_H (E in ASCII) is present at the MW1 parameter as the contents of a word flag.
$4D_H$ (M in ASCII) is present at the MW2 parameter as the contents of a word flag.
50_H (P in ASCII) is present at the MW3 parameter as the contents of a word flag.
$4F_H$ (O in ASCII) is present at the MW4 parameter as the contents of a word flag.
The value 120 is present at the MW5 parameter as the contents of a word flag.
$4B_H$ (K in ASCII) is present at the MW6 parameter as the contents of a word flag.
$4D_H$ (M in ASCII) is present at the MW7 parameter as the contents of a word flag.
As the received telegram has been evaluated and agrees with a comparison telegram, the following applies :
MEUN = 0
RDY = 1
TELN = 4

MW0 = T
MW1 = E
MW2 = M
MW3 = P
MW4 = O
MW5 = 120
MW6 = K
MW7 = M

Example 5
XY25OTTO<CR>
The received telegram has been evaluated, but agrees with *none* of the comparison telegrams.
Therefore the following applies :
MEUN = 1
RDY = 1
TELN = is not allocated
Reception of further telegrams is blocked until reception of the invalid telegram is acknowledged at the QUIT input.

SINIT INITIALIZATION AND CONFIGURATION OF THE SERIAL INTERFACES



PARAMETERS

FREI	BINARY	%O, %I, %S, %M, %K	Enable processing of the block, 0 -> 1 edge
SSK	WORD	%OW, %IW, %MW, %KW	Interface identifier (1 or 2)
BAUD	WORD	%OW, %IW, %MW, %KW	Baud rate; 300 ... 9600 Baud
STOP	WORD	%OW, %IW, %MW, %KW	Number of stop bits; input has no effect
ZL	WORD	%OW, %IW, %MW, %KW	Character length, 7 or 8 data bits per character
PTY	BINARY	%O, %I, %S, %M, %K	Parity, enable/disable
E/O	BINARY	%O, %I, %S, %M, %K	Parity even/odd

ECHO	BINARY	%O, %I, %S, %M, %K	Echo, on/off
SBRK	BINARY	%O, %I, %S, %M, %K	Send break character
FEND	BINARY	%O, %I, %S, %M, %K	Enable end of text character for transmitting direction
ENDS	WORD	%OW, %IW, %MW, %KW	End of text character for transmitting direction
ENDE	WORD	%OW, %IW, %MW, %KW	End of text character for receiving direction

DESCRIPTION

Serial interface initialization function block.

The serial interface COM1 is available to the user. In case of 07 KT 92 and 07 KT 93, COM2 is additionally available. These interfaces can be operated by the PLC program (e.g. with the DRUCK and EMAS blocks).

Before using one of these interfaces, the interface must be initialized. The function block SINIT is available for this purpose.

The function block SINIT is processed *once* with every 0 -> 1 edge at the FREI input. It initializes the serial interface specified at the SSK input (COM1, COM2).

FREI BINARY
The block is run through *once* when a 0->1 edge is specified at the FREI input. As the result of this, the serial interface whose number is specified at the SSK input is initialized and the interface is then operable.

SSK WORD
The number of the interface to be initialized is specified at the input SSK.
The following applies :
COM1 : number = 1
COM2 : number = 2

BAUD WORD
The value for the baud rate is specified at the BAUD input.
Baudrate : 300 ... 9600 Baud

STOP WORD
The number of stop bits is set to 1 and can not be modified. The value for the number of stop bits specified at the input STOP has no significance.

ZL WORD
The input ZL specifies the required character length. The character length signifies the number of data bits per character.
7 or 8 data bits per character are possible.

PTY BINARY
The input PTY specifies whether a character is transferred with or without a parity bit.
PTY = 0 -> Transfer without parity bit

PTY = 1 -> Transfer with parity bit

E/O BINARY
The input E/O specifies whether an even or odd parity bit is required.
E/O = 0 -> Odd parity bit
E/O = 1 -> Even parity bit

ECHO BINARY
The input ECHO specifies whether the characters received through the applicable interface are to be reflected (echoed) by the PLC. In this way, the sender of a character, for example, can determine whether or not it has arrived correctly in the PLC.
ECHO = 0 -> No echo, character is not reflected
ECHO = 1 -> Echo, character is reflected

SBRK BINARY
The state of the transmit line TxD can be influenced at the input SBRK (send break character).
SBRK = 0 -> Normal state of the transmit line TxD for transfer of characters
SBRK = 1 -> Transmit line TxD set to “0”

FEND BINARY
The input FEND specifies whether or not the end of text character planned at input ENDS is output at the same time (enable end character).
FEND = 0 -> End of text character in transmitting direction is not output
FEND = 1 -> End of text character in transmitting direction is output

ENDS WORD
A freely selectable end of text character for the transmitting direction can be specified at the ENDS input. This end character is then appended automatically to every text (telegram) which the DRUCK block sends to the outside world through the serial interface. However, a precondition is that input FEND is enabled.
The end of text character is specified as a numeric value.

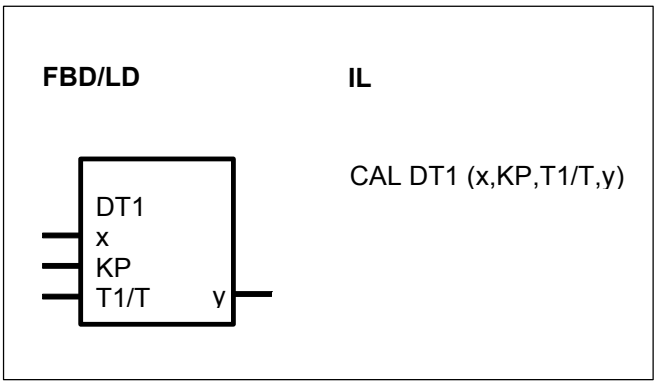
Example : 3 or 03_H signifies <ETX>
4 or 04_H signifies <EOT>
13 or $0D_H$ signifies <CR>
10 or OA_H signifies <LF>
32 or 20_H signifies <SP>
...

ENDE WORD
A freely selectable end of text character for the receiving direction can be specified at the ENDE input. When a telegram is received through the serial interface, the PLC recognizes the end of the telegram by virtue of this end character. The end character is specified in the same way as in the case of the ENDS input.

6 Regulation functions

Regulation functions	from pages C-154 to C-174	serie C^{tler} 40	50	90	30
DT1	Differentiator with delay of the 1st order			x	
INTK	Integrator (extended)			x	
PI	Proportional-integral controller	x	x	x	x
PIDT1	PIDT1 controller	x	x	x	
PT1	PT1 element			x	

DT1 DIFFERENTIATOR WITH DELAY OF THE 1ST ORDER



PARAMETERS

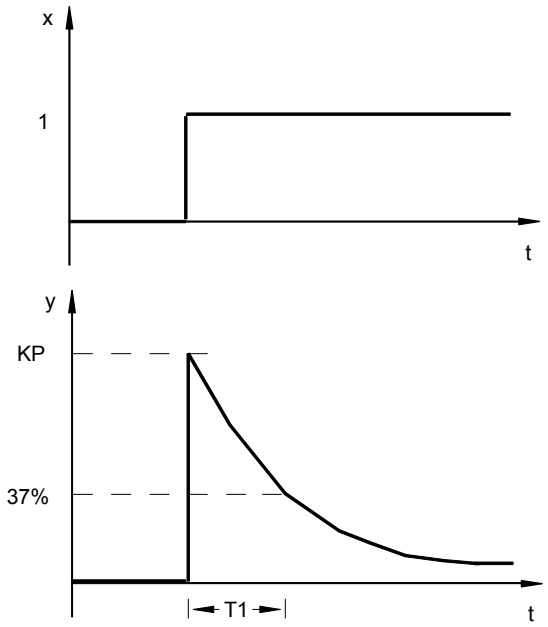
x	WORD	%IW, %OW, %MW, %KW	Controlled variable
KP	WORD	%IW, %OW, %MW, %KW	Proportional coefficient, specified as a percentage
T1/T	WORD	%IW, %OW, %MW, %KW	Time constant scaled to cycle time
y	WORD	%OW, %MW	Manipulated variable

DESCRIPTION

The controlled variable x is multiplied by the proportional coefficient KP. The proportional coefficient is specified as a percentage. If the controlled variable no longer changes, the manipulated variable y moves towards the value 0 in an exponentiel function. The time constant T1 specifies the time when the step response has dropped to approximately 37% of its initial value. The value has dropped below 1% after 5*T1.

Transfer function : $$F(s) = \frac{KP * s * T1}{s * T1 + 1}$$

Transfer function:



x WORD
The operand for the controlled variable (input value for the DT1 function) is specified at the input x.

KP WORD
The proportional coefficient is specified at the input KP. This value is specified as a percentage and can be positive or negative.

Example :	1	=	1	Percent
	55	=	55	Percent
	100	=	100	Percent
	1000	=	1000	Percent
	-100	=	-100	Percent

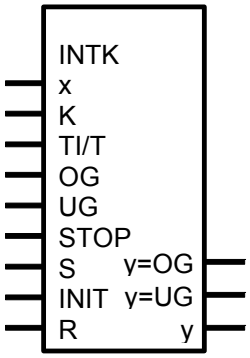
100 percent means that the input x is *not* influenced by the proportional coefficient.

T1/T WORD
The time constant of the DT1 function is specified at the input T1/T.To do this, the time constant T1 must be scaled to the PLC cycle time T.
The following applies to T1/TZ : $10 \leq T1/TZ \leq 32767$ -> T1 > 10 * TZ
If a negative value is specified for T1/T, then the PLC automatically sets T1/T to the maximum positive value +32767.

y WORD
The manipulated variable (output value of the DT1 function) is output through the output y.

INTK INTEGRATOR (EXTENDED)

FBD/LD



IL

```
CAL INTK (x,TI/T,R,STOP,S,INIT,K,
          OG,UG,y,y=OG,y=UG)
```

PARAMETERS

x	WORD	%IW, %OW, %MW, %KW	Controlled variable
K	WORD	%IW, %OW, %MW, %KW	Proportional coefficient, output as a percentage
TI/T	WORD	%IW, %OW, %MW, %KW	Integration time scaled to the cycle time
OG	WORD	%IW, %OW, %MW, %KW	High limit for the manipulated variable y
UG	WORD	%IW, %OW, %MW, %KW	Low limit for the manipulated variable y
STOP	BINARY	%I, %O, %M, %S, %K	Integrator stop
S	BINARY	%I, %O, %M, %S, %K	Set output to INIT value
INIT	WORD	%IW, %OW, %MW, %KW	Initial value
R	BINARY	%I, %O, %M, %S, %K	Reset output y to the value 0
y=OG	BINARY	%O, %M	Output y has reached top limit
y=UG	BINARY	%O, %M	Output y has reached low limit
y	WORD	%OW, %MW	Manipulated variable

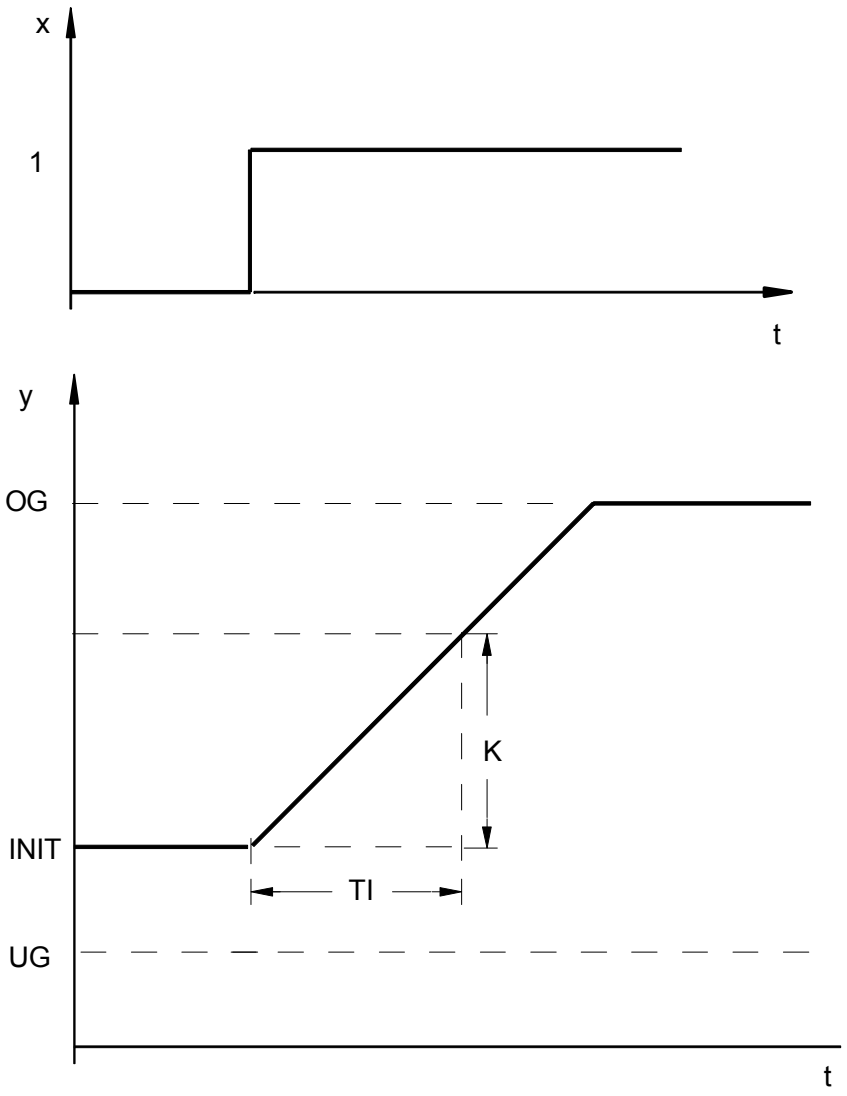
DESCRIPTION

This block generates the integral of the controlled variable x multiplied by the proportinal coefficient K.

The integrator's output y can be manipulated as follows :
- It can be set to the value 0 by a 1 signal at the input R (reset)
- It can be latched to the current value by a 1 signal at the input STOP
- It can be set to the initial value at the INIT input by a 1 signal at the input S (set)
- It can be limited to a maximum value specified at the input OG (high limit)
- It can be limited to a minimum value specified at the input UG (low limit)

Transfer function :

$$F(S) = \frac{K}{S * T}$$



x WORD
The operand for the controlled variable is specified at the input x.

K WORD
The proportional coefficient is specified at the input K. It serves to weight the controlled variable at the input x. Weighting is achieved by multiplying the controlled variable by the proportional coefficient.
The proportional coefficient is specified as a percentage.

Example :	1	=	1	Percent
	55	=	55	Percent
	100	=	100	Percent
	1000	=	1000	Percent
	-100	=	-100	Percent

• 1 percent means that the block multiplies the value at input x by the factor 0.01
• 100 percent means that the block multiplies the value at input x by the factor 1
• 1000 percent means that the block multiplies the value at input x by the factor 10

TI/T WORD
The integration time is specified at the input TI/T. It must be scaled to the cycle time. During the time TI the output y of the integrator changes by the value K * x.
Value range : 0 ≤ TI/T ≤ 328
• If values which are beyond the admissible value range are specified, the PLC uses the value 328.
• A large integration time (TI) can be achieved by choosing a great cycle time, too. If the block is used within a run number block, the cycle time of the run number block is valid for block INTK and not the cycle time (%KD 0,0) of the PLC program.

OG WORD
The manipulated variable y can be limited to a value range. The high limit for the manipulated variable y is specified at the input OG.

UG WORD
The manipulated variable y can be limited to a value range. The low limit for the manipulated variable y is specified at the input UG.

STOP BINARY *)
Integration can be stopped with the STOP input.
STOP = 0 -> integration is not stopped
STOP = 1 -> integration is stopped, i.e. the output y no longer changes.

S BINARY *)
By means of the input S, the manipulated variable y can be set to the initial value specified at the input INIT. Integration then again begins as from the initial value.
S = 0 -> No setting
S = 1 -> Output y is set to the specified initial value

INIT WORD
The initial value to which the output y must be set when required is specified at the input INIT.

R BINARY *)
With the input R, the output y can be reset to the value 0. Integration then again begins as from the value 0.

*) Priority sequence for the inputs STOP, S and R :

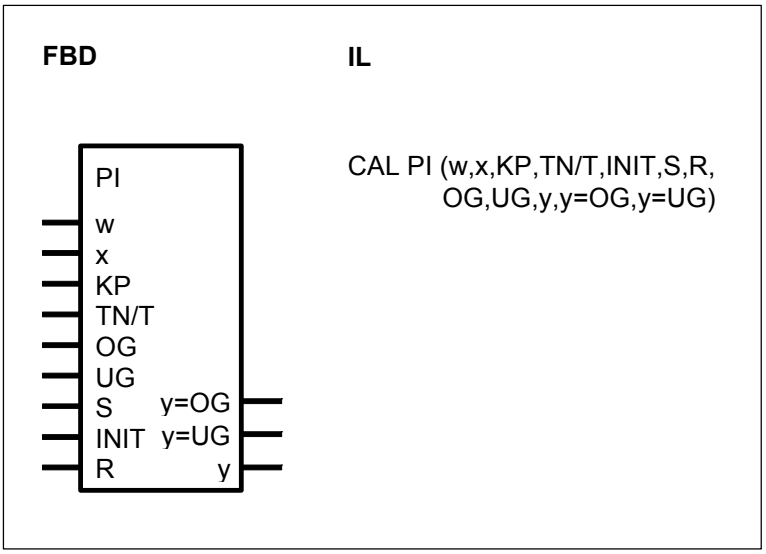
R	Highest priority
STOP	
S	Lowest priority

y=OG BINARY
Whether the value at the output y has reached the specified top limit is signalled at the output y=OG. Integration is stopped automatically when the limit is reached.
y=OG = 0 -> y has not reached the limit
y=OG = 1 -> y has reached the limit

y=UG BINARY
Whether the value at the output y has reached, the specified lower limit is signalled at the output y=UG. Integration is stopped automatically when the limit is reached.
y=UG = 0 -> y has not reached the limit
y=UG = 1 -> y has reached the limit

y WORD
The manipulated variable (output value of the integrator) is output through the output y.

PI PROPORTIONAL-INTEGRAL-CONTROLLER



PARAMETERS

w	WORD	%IW, %OW, %MW, %KW	Command variable (setpoint)
x	WORD	%IW, %OW, %MW, %KW	Controlled variable (actual value)
KP	WORD	%IW, %OW, %MW, %KW	Proportional coefficient; specified as a %
TN/T	WORD	%IW, %OW, %MW, %KW	Integral action time scaled to the PLC cycle time
OG	WORD	%IW, %OW, %MW, %KW	High limit for the manipulated variable y
UG	WORD	%IW, %OW, %MW, %KW	Low limit for the manipulated variable y
S	BINARY	%I, %O, %M, %S, %K	Enabling for setting the manipulated variable y to the initial value INIT
INIT	WORD	%IW, %OW, %MW, %KW	Initial value for the manipulated variable y
R	BINARY	%I, %O, %M, %S, %K	Reset of the manipulated variable y to the value 0
y=OG	BINARY	%O, %M	High limit has been reached
y=UG	BINARY	%O, %M	Low limit has been reached

y	WORD	%OW, %MW	Output for the manipulated variable y

DESCRIPTION

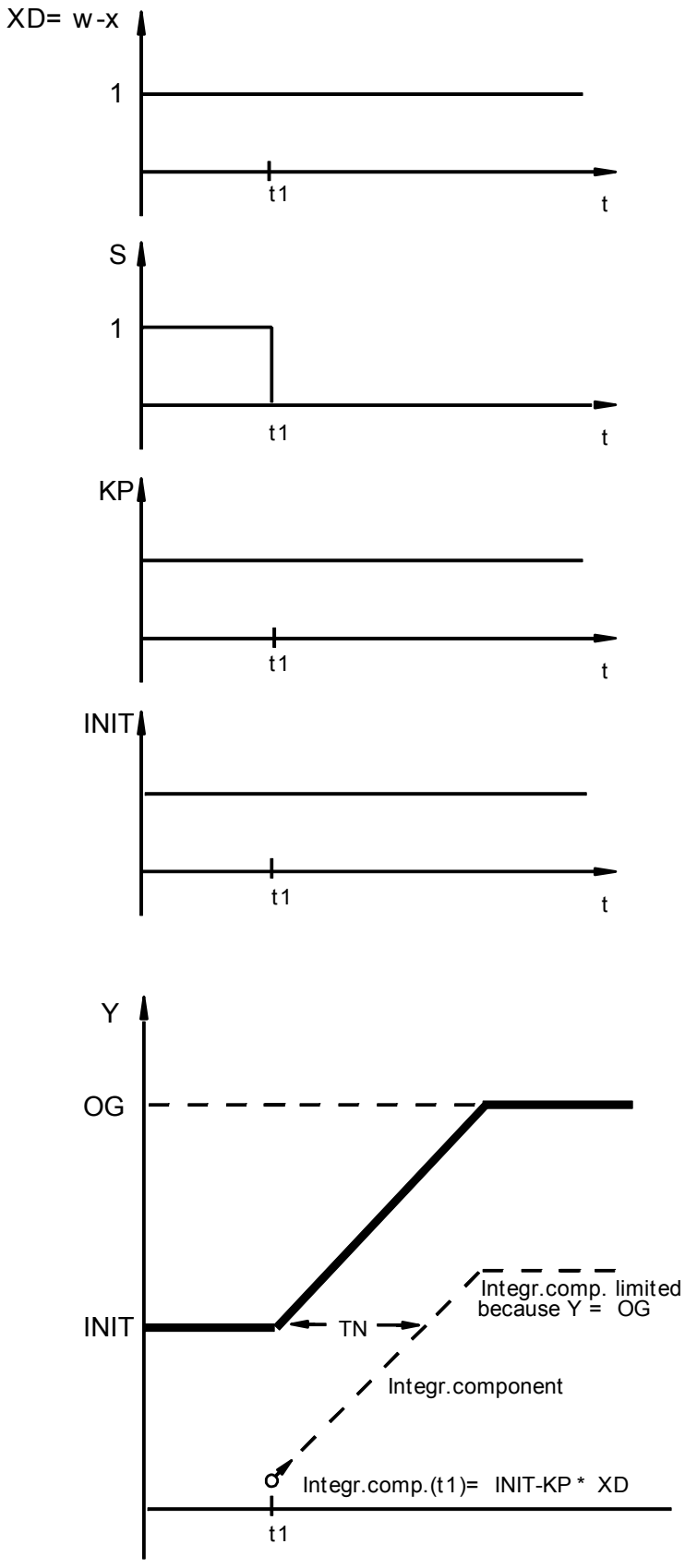
The PI controller changes the value at its output y (manipulated variable) until the value at the input x (controlled variable) is equal to the value at the input w (command variable).

Control algorithm : Simple rectangle rule

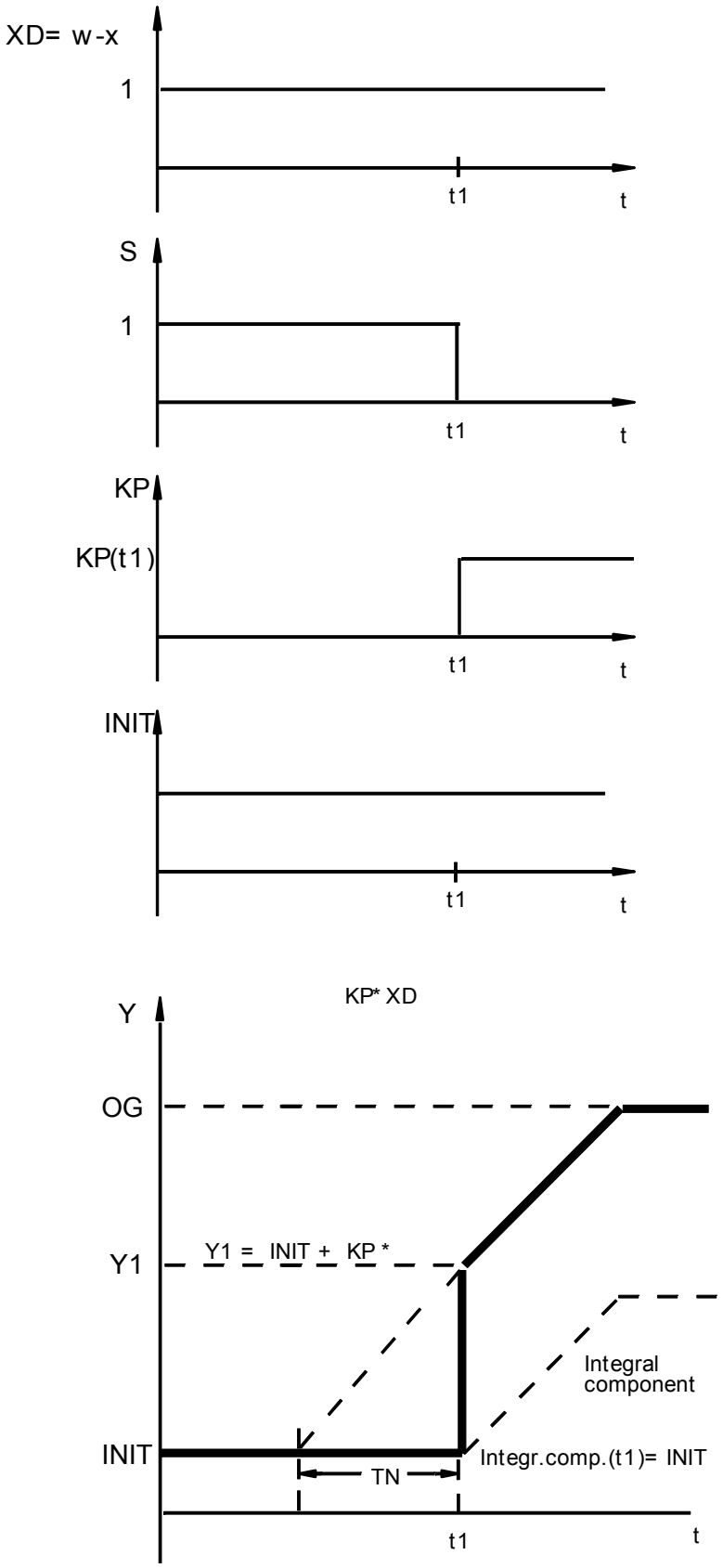
$$Y = \frac{KP}{100} * \frac{w - x}{TN / T} + YI(z-1) + \frac{KP}{100} * (w - x)$$

Where :YI(z-1) is the integral component from the previous program cycle

Transfer function : $F(s) = KP * (1 + \frac{1}{s * TN})$



PI controller : Surge-free transition from the specified initial value to control operation



PI controller : Surging transition from the specified initial value to control operation

w WORD

The command variable (setpoint) is specified at the input w.

x WORD
The controlled variable (actual value) is specified at the input x.

KP WORD
The proportional coefficient is specified at the input KP. This value is specified as a percentage and may be positive or negative.

Example :	1	=	1	Percent
	55	=	55	Percent
	100	=	100	Percent
	1000	=	1000	Percent
	-500	=	-500	Percent

• 1 percent means that the block multiplies the system deviation by the factor 0.01 (see also control algoritm)
• 100 percent means that the block multiplies the system deviation by the factor 1 (see also control algoritm)
• 1000 percent means that the block multiplies the system deviation by the factor 10 (see also control algoritm)
Generally, proportional coefficients of more than 1000% are not meaningful in control systems.

TN/T WORD
The integral action time TN is scaled to the PLC cycle time T and is specified at the input TN/T.
Value range : $0 \leq TN/T \leq 328$
• If values which are beyond the admissible value range are specified, the PLC uses the value 328.
• A large integral action time TN can be achieved by choosing a great cycle time T, too. If the block is used within a run number block, the cycle time of the run number block is valid for block PI and not the cycle time (%KD 0,0) of the PLC program.

Limiting the manipulated variable y
OG WORD
UG WORD
The output y (manipulated variable) of the controller can be limited
- To a maximum value by specifying a limit at the input OG (high limit);
- To a minimum value by specifying a limit at the input UG (low limit)
The high and low limits *also* apply to the controller's internal I component. That is to say, the I component can only assume values between the high and low limits. If the manipulated variable y reaches one of the two limits, the controller's I component is *no* longer altered. This prevents the I component from running amock in the event of limiting of the controller output y, assuming meaningless values from the point of view of control and, in certain circumstances, not returning to the operating range until after a very long time. This response of a controller is also referred to as a "special anti-reset windup measure".

Setting and resetting the controller
S BINARY
INIT WORD

R BINARY
Setting the controller to an initial value
- The output y of the controller is set to the initial value specified at the INIT input by means of a 1 signal at the input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0 (see above).

Surge-free setting/resetting
- The output y of the controller is set to the initial value specified at the input INIT by means of a 1 signal at the binary input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0.

In doing so, adjustment to the initial value takes place internally in the controller. This adjustment is a shift of the controller output from the momentary value to the required initial value. Now, the controller continues operating from this initial value precisely as it would have done at the old operating point before the shift, i.e. without surges. The controller's I component is defined so that the sum of the P component and the I component just results in the initial value.

Advantage of surge-free operation :
- Control as from the new initial value is devoid of surges.

Disadvantage of surge-free operation :
- The following equation applies : I_component = INIT - P_component
In certain circumstances, the I component is set to high values and may take very long before this "wrong" I component from the point of view of control is dissipated again.

Surging setting/resetting
- The output y of the controller is set to the initial value specified at the INIT input by means of a 1 signal at the input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0.

In the event of surging setting or resetting of the controller the I component is set equal to the initial value. To do this, the P component must be suppressed during setting.
Where : I component = INIT
Surging setting to an initial value is achieved by the following measure during setting:
- Specifying the value 0 at the input KP.
This measure renders the P component inactive. The controller output y assumes the initial value during the set cycle.

The P component is enabled again after the set cycle. From the initial value, the controller output y jumps according to the P component of the controller.

Advantage of surging setting :
- The I component is not set to "wrong" values from the point of view of control.

Disadvantage of surging setting :
- No freedom from surging

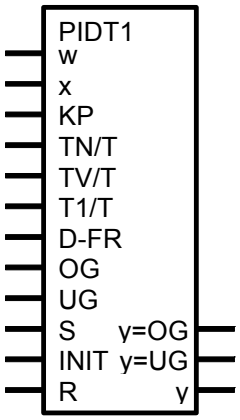
y WORD
The controller's manipulated variable y is output through the output y.

y=OG BINARY
The output y=OG signals whether or not the value at the output y has exceeded the specified high limit.
y=OG = 0 limit has *not* been reached.
y=OG = 1 limit has been reached.

y=UG BINARY
The output y=UG signals whether or not the value at the output y has reached the specified low limit.
y=UG = 0 limit has *not* been reached.
y=UG = 1 limit has been reached.

PIDT1 PIDT1 CONTROLLER

FBD



IL

```
CAL PIDT1 (w,x,KP,TN/T,TV/T,T1/T
          D-FR,INIT,S,R,OG,UG,y
          y=OG,y=UG)
```

PARAMETERS

w	WORD	%IW, %OW, %MW, %KW	Command variable (setpoint)
x	WORD	%IW, %OW, %MW, %KW	Controlled variable (actual value)
KP	WORD	%IW, %OW, %MW, %KW	Proportional coefficient, specified as a percentage
TN/T	WORD	%IW, %OW, %MW, %KW	Integral action time scaled to the PLC cycle time
TV/T	WORD	%IW, %OW, %MW, %KW	Derivative action time scaled to the PLC cycle time
T1/T	WORD	%IW, %OW, %MW, %KW	Returning time scaled to the PLC cycle time
D-FR	BINARY	%I, %O, %M, %K	Enable DT1 component
OG	WORD	%IW, %OW, %MW, %KW	High limit for the manipulated variable y
UG	WORD	%IW, %OW, %MW, %KW	Low limit for the manipulated

			variable y
S	BINARY	%I, %O, %M, %S, %K	Enable for setting to initial value INIT
INIT	WORD	%IW, %OW, %MW, %KW	Initial value for the manipulated variable y
R	BINARY	%I, %O, %M, %S, %K	Reset the manipulated variable y to the value 0
y=OG	BINARY	%O, %M	High value has been reached
y=UG	BINARY	%O, %M	Low value has been reached
y	WORD	%OW, %MW	Output for manipulated variable y

DESCRIPTION

The PI controller changes its output y (manipulated variable) until the input x (controlled variable) is equal to the input w (command variable).

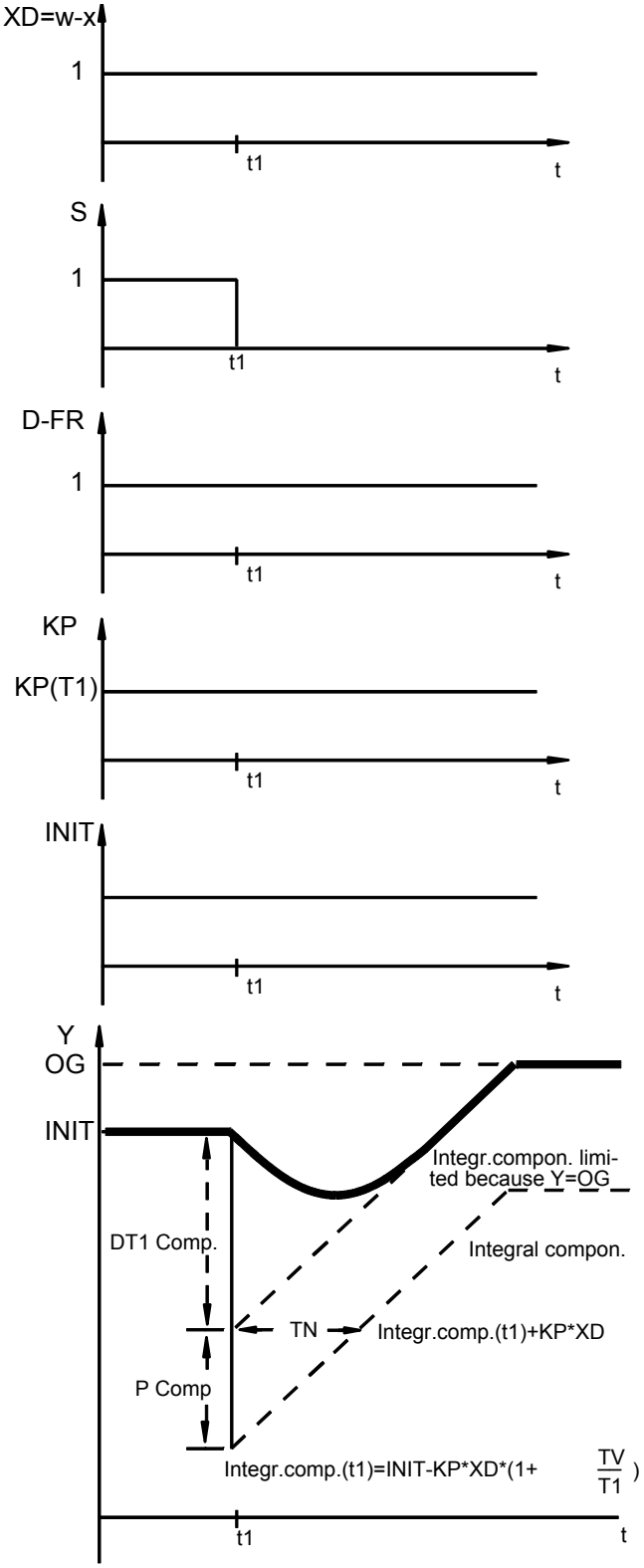
Transfer function : $$F(s) = KP * (1 + \frac{1}{s * TN} + \frac{s * TV}{1 + (s * T1)})$$

Control algorithm : simple rectangle rule :

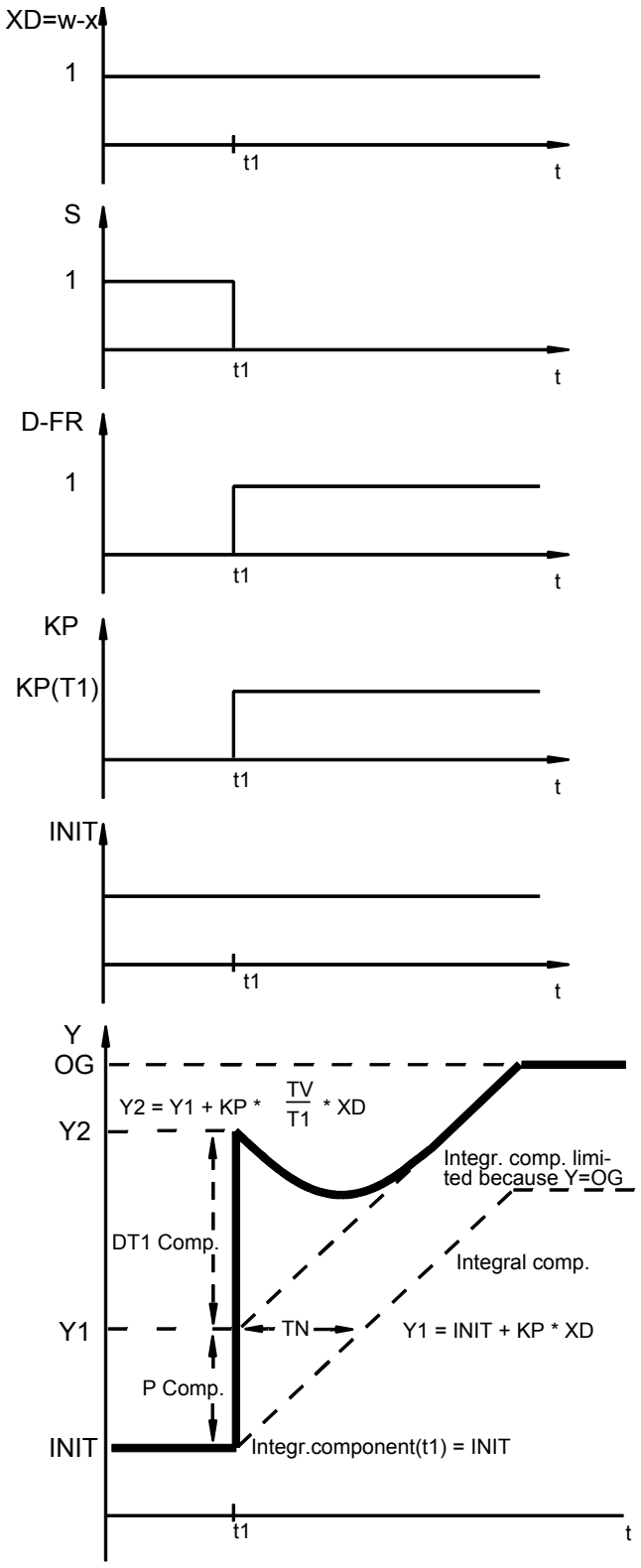
$$y = \frac{KP * XD}{100} + \frac{KP}{100} * \frac{XD}{TN/TZ} + YI(z-1) + \frac{T1/TZ}{1+(T1/TZ)} * \left(YDT1(z-1) + \frac{1}{T1/TZ} * \frac{TV}{TZ} * \frac{KP}{100} * (XD - XD(z-1)) \right)$$

where :

YI(z-1) : integral portion from the previous program cycle
YDT1(z-1) : differential portion from the previous program cycle
XD(z-1) : control system difference from the previous program cycle



PIDT1 controller : Surge-free transition from the specified intitial value to control mode



PIDT1 controller : Surging transition from the specified initial value to control mode

w WORD
The command variable (setpoint) is specified at the input w.

x WORD
The controlled variable (actual value) is specified at the input x.

KP WORD
The proportional coefficient is specified at the input KP. This value is specified as a percentage and may be positive or negative.

Example :	1	=	1	Percent
	55	=	55	Percent
	100	=	100	Percent
	1000	=	1000	Percent
	-500	=	-500	Percent

• 1 percent means that the block multiplies the system deviation by the factor 0.01 (see also control algoritm)
• 100 percent means that the block multiplies the system deviation by the factor 1 (see also control algoritm)
• 1000 percent means that the block multiplies the system deviation by the factor 10 (see also control algoritm)
Generally, proportional coefficients of more than 1000% are not meaningful in control systems.

TN/T WORD
The integral action time TN is scaled to the PLC cycle time T and is specified at the input TN/T.
Value range : $0 \leq TN/T \leq 328$
• If values which are beyond the admissible value range are specified, the PLC uses the value 328.
• A large integral action time TN can be achieved by choosing a great cycle time T, too. If the block is used within a run number block, the cycle time of the run number block is valid for block PIDT1 and not the cycle time (%KD 0,0) of the PLC program.

TV/T WORD
The derivative action time TV is scaled to the PLC cycle time T and is specified at the input TV/T($0 < TV/T \leq 32767$).

T1/T WORD
The returning time T1 is scaled to the PLC cycle time T and is specified at the input T1/T($0 < T1/T \leq 32767$). The returning time is the time in which the DT1 component has decreased to approximately 37% of its initial value.
Inadmissible time parameters :
Every time value is set to the maximum positive value 32767 if the time value at the input is erroneously specified as less than or equal to “0”.

D-FR WORD
The DT1 component of the controller can be connected or deactivated by means of the D-FR input.
D-FR = 0 The DT1 component is deactivated -> pure PI controller
D-FR = 1 The DT1 component is connected -> PIDT1 controller

In the following cases, from the control point of view it is often disturbing and not meaningful for the DT1 component to be active :
- During activations
- In the event of large system deviations
- When setting the controller to a specified initial value
- When resetting the controller to the value 0

The command and controlled variables can be compared outside of the controller. Depending on this comparison, the DT1 component can be activated or deactivated specifically by way of the D-FR input.

For example, activation can be restricted to ensuring that the system deviation is within a required bandwidth. That is to say, the DT1 component is only active if the controlled variable fluctuates around the setpoint within a specific bandwidth. The DT1 component is deactivated if the controlled variable leaves this tolerance band.

Limiting the maniulated variable y
OG WORD
UG WORD
The output y (manipulated variable) of the controller can be limited
- To a maximum value by specifying a limit at the input OG (high limit);
- To a minimum value by specifying a limit at the input UG (low limit)
The high and low limits *also* apply to the controller's internal I component. That is to say, the I component can only assume values between the high and low limits. If the manipulated variable y reaches one of the two limits, the controller's I component is *no longer* altered.

This prevents the I component from running amock in the event of limiting of the controller output y, assuming meaningless values from the point of view of control and, in certain circumstances, not returning to the operating range until after a very long time. This response of a controller is also referred to as a "special anti-reset windup measure (ARW)".

Setting and resetting the controller
S BINARY
INIT WORD
R BINARY
Setting the controller to an initial value
- The output y of the controller is set to the initial value specified at the INIT input by means of a 1 signal at the input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0 (see above).

Surge-free setting/resetting
- The output y of the controller is set to the initial value specified at the input INIT by means of a 1 signal at the binary input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0.
In doing so, adjustment to the initial value takes place internally in the controller. This adjustment is a shift of the controller output from the momentary value to the required

initial value. Now, the controller continues operating from this initial value precisely as it would have done at the old operating point before the shift, i.e. without surges. The controller's I component is defined so that the sum of the P component, I component and DT1 component just results in the initial value.

Advantage of surge-free operation :
- Control as from the new initial value is devoid of surges.

Disadvantage of surge-free operation :
- The following equation applies :

I component = INIT - P component - DT1 component

In certain circumstances, the I component is set to high values and may take very long before this "wrong" I component from the point of view of control is dissipated again.

Surging setting/resetting
- The output y of the controller is set to the initial value specified at the INIT input by means of a 1 signal at the input S (set).
- A 1 signal at the input R (reset) is equivalent to specifying the initial value 0.
- In the event of surging setting or resetting of the controller, the I component is set equal to the initial value. To do this, the P and DT1 components must be suppressed during setting.

Where : I component = INIT

Surging setting to an initial value is achieved by the following measures during setting :
- Deactivation of the DT1 component via the D-FR control input and
- Specifying the value 0 at the input KP.

These measures render the P component and the DT1 component inactive during setting of the controller.
The controller output assumes the initial value during the set cycle.
The P and DT1 components are enabled again after the set cycle. From the initial value, the controller output y jumps according to the P and DT1 components of the controller.

Advantage of surging setting :
- The I component is not set to "wrong" values from the point of view of control.

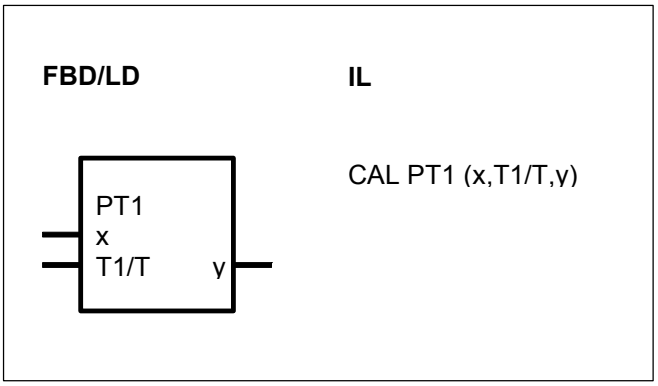
Disadvantage of surging setting :
- No freedom from surging

y WORD
The controller's manipulated variable y is output through the output y.

y=OG BINARY
The output y=OG signals whether or not the value at the output y has reached the specified high limit.
y=OG = 0 limit has *not* been reached.
y=OG = 1 limit has been reached.

y=UG BINARY
The output y=UG signals whether or not the value at the output y has reached the specified low limit.
y=UG = 0 limit has *not* been reached.
y=UG = 1 limit has been reached.

PT1 PT1 ELEMENT



PARAMETERS

x	WORD	%IW, %OW, %MW, %KW	Controlled variable
T1/T	WORD	%IW, %OW, %MW, %KW	Time constant
y	WORD	%OW, %MW	Manipulated variable

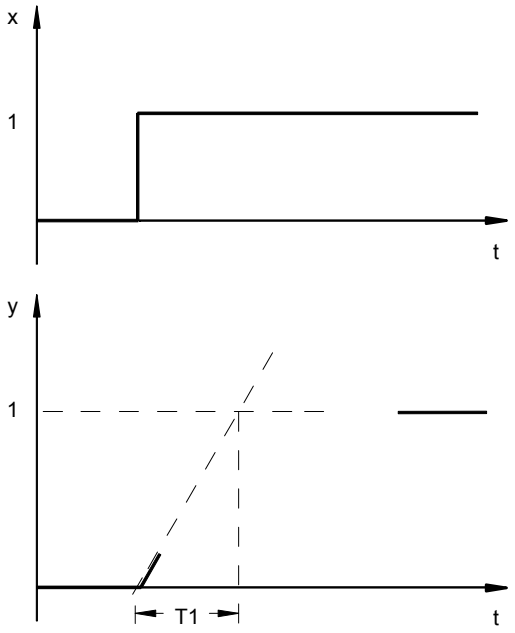
DESCRIPTION

This function block realizes a delay element of the first order.

Transfer function :

$$F = \frac{1}{1 + T1 * S}$$

Transfer funktion:



x WORD
The controlled variable is specified at the input x.

T1/T WORD
The delay time T1 is specified at the input T1/T. At the same time, the delay time T1 must be scaled to the cycle time T.
T1/T ≥ 0 must apply
If a negative time value is specified erroneously, the PLC automatically sets the value 32767 for T1/T.

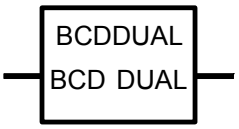
y WORD
The result of the delay element (manipulated variable) is output through the output y.

7 Format conversion functions

Format conversion functions	from pages C-174 to C-200 serie C[tler]	40	50	90	30
BCDDUAL / BCDBIN	BCD to binary conversion	x	x	x	x
BCDDUALD / BCDDW	BCD to binary conversion, double word			x	
DUALBCD / BINBCD	Binary to BCD conversion	x	x	x	x
DUABCDD / DWBCD	Binary to BCD conversion, double word			x	
DWW	Double word to word conversion	x	x	x	x
PACK4	Pack 4 binary variables in a word	x	x	x	x
PACK8	Pack 8 binary variables in a word	x	x	x	x
PACK16	Pack 16 binary variables in a word	x	x	x	x
PACKD4	Pack 4 binary variables in a double word			x	
PACKD8	Pack 8 binary variables in a double word			x	
PACKD16	Pack 16 binary variables in a double word			x	
PACKD24	Pack 24 binary variables in a double word			x	
PACKD32	Pack 32 binary variables in a double word			x	
UNPACK4	Unpacking a word into 4 binary variables	x	x	x	x
UNPACK8	Unpacking a word into 8 binary variables	x	x	x	x
UNPACK16	Unpacking a word into 16 binary variables	x	x	x	x
UNPACKD4	Unpacking a double word into 4 binary variables			x	
UNPACKD8	Unpacking a double word into 8 binary variables			x	
UNPACKD16	Unpacking a double word into 16 binary variables			x	
UNPACKD24	Unpacking a double word into 24 binary variables			x	
UNPACKD32	Unpacking a double word into 32 binary variables			x	
WDW	Word to double word conversion	x	x	x	x

BCDDUAL BCD TO BINARY CONVERSION

FBD



IL

```
CAL BCDDUAL(BCD,DUAL)
or
!BA0
BCDBIN
BCD
DUAL
```

PARAMETERS

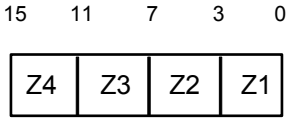
BCD	WORD	%IW, %MW, %OW, %KW	BCD-coded number
DUAL	WORD	%OW,%MW	Binary number

DESCRIPTION

The positive BCD coded number at the input BCD is converted to a binary number and is allocated to the operand at the DUAL output.

Definition :
The significance of the digits in a BCD coded number and a hexadecimal number is defined as follows :

BCD-NUMBER

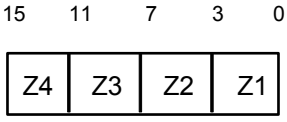


Numerical value:

Z1 * 1
Z2 * 10
Z3 * 100
Z4 * 1000

0 < Zi < 9

HEXDEC-NUMBER



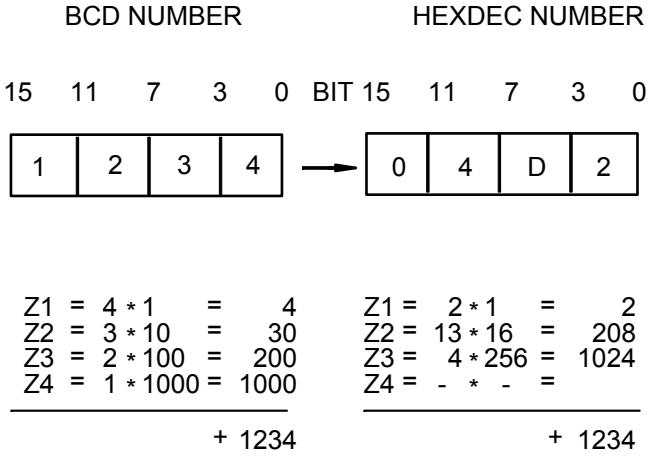
Numerical value:

Z1 * 1
Z2 * 16
Z3 * 256
Z4 * 4096

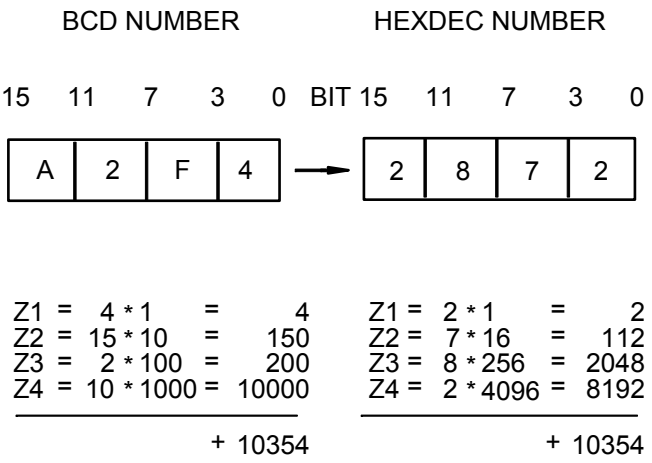
0 < Zi < F

Note :
At the BCD input, the block additionally also accepts digits to which the following applies : $0 \leq Zi \leq F$

Example 1



Example 2

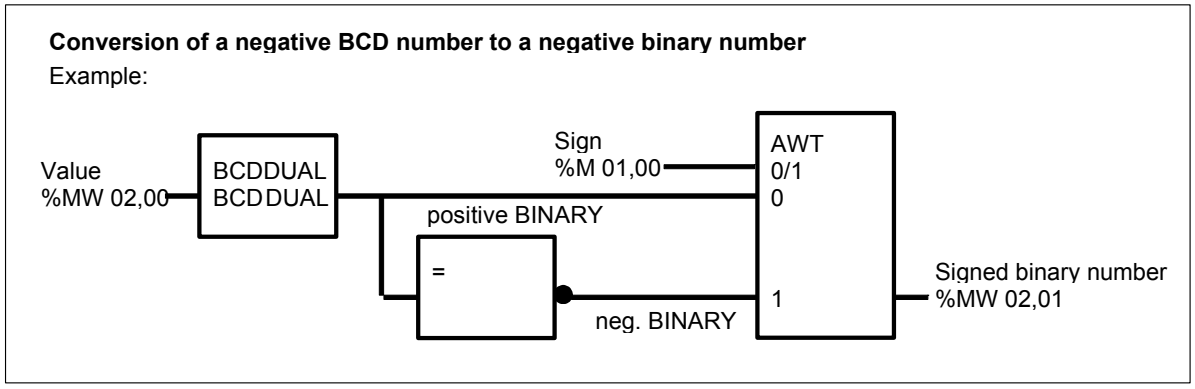


Representation of a negative BCD number

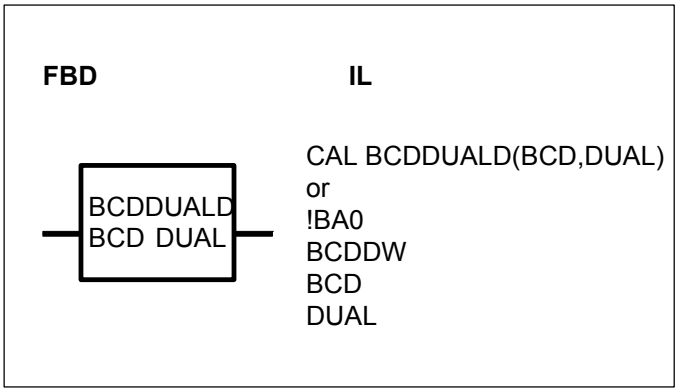
A negative BCD number can be represented in the PLC by *separate* representation of the value and the sign. In doing so, the value of the BCD number is stored in a word variable and the information about the sign is stored in a binary variable.

Example :
Representation of the BCD number -7 :

%MW 02,00 = 7 Value of the number, BCD coded
%M 01,00 = 1 Sign of the number • Positive : 0
• Negative : 1



BCDDUALD BCD TO BINARY CONVERSION, DOUBLE WORD



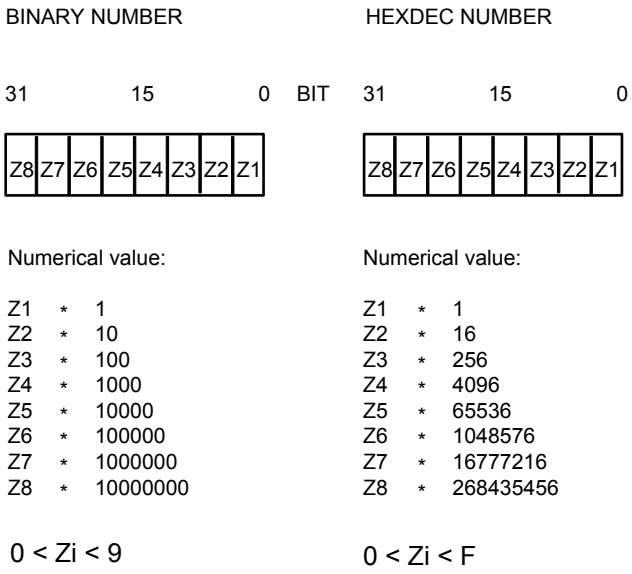
PARAMETERS

BCD	DOUBLE WORD	%MD, %KD	BCD coded number
DUAL	DOUBLE WORD	%MD	Binary number

DESCRIPTION

The positive BCD coded number at the input BCD is converted to a binary number and is allocated to the operand at the DUAL output.

Definition
The significance of the digits in a BCD coded number and a hexadecimal number is defined as follows :



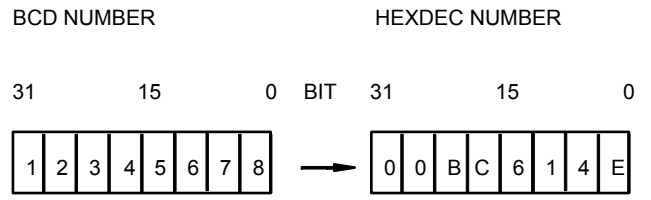
Note :
The block also accepts digits at the BCD input to which the following applies :

$0 \leq Zi \leq F$

Representation of a negative BCD number
See function block BCDDUAL

Conversion of a negative BCD number to a negative binary number
See function block BCDDUAL

Example 1:



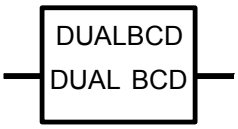
Z1 = 8 * 1	=	8	Z1 = 14 * 1	=	14
Z2 = 7 * 10	=	70	Z2 = 4 * 16	=	64
Z3 = 6 * 100	=	600	Z3 = 1 * 256	=	256
Z4 = 5 * 1000	=	5000	Z4 = 6 * 4096	=	24576
Z5 = 4 * 10000	=	40000	Z5 = 12 * 65536	=	786432
Z6 = 3 * 100000	=	300000	Z6 = 11 * 1048576	=	11534336
Z7 = 2 * 1000000	=	2000000	Z7 = 0 * 16777216	=	0
Z8 = 1 * 10000000	=	10000000	Z8 = 0 * 268435456	=	0
	+	12345678		+	12345678

Example 2:

Z1 = 4 * 1	=	4	Z1 = 2 * 1	=	2
Z2 = 15 * 10	=	150	Z2 = 13 * 16	=	208
Z3 = 2 * 100	=	200	Z3 = 7 * 256	=	1792
Z4 = 10 * 1000	=	10000	Z4 = 12 * 4096	=	49152
Z5 = 4 * 10000	=	40000	Z5 = 5 * 65536	=	327680
Z6 = 3 * 100000	=	300000	Z6 = 4 * 1048576	=	4194304
Z7 = 11 * 1000000	=	11000000	Z7 = 1 * 16777216	=	16777216
Z8 = 1 * 10000000	=	10000000	Z8 = 0 * 268435456	=	0
	+	21350354		+	21350354

DUALBCD BINARY TO BCD CONVERSION

FBD



IL

```
CAL DUALBCD (DUAL,BCD)
or
!BA0
BINBCD
DUAL
BCD
```

PARAMETERS

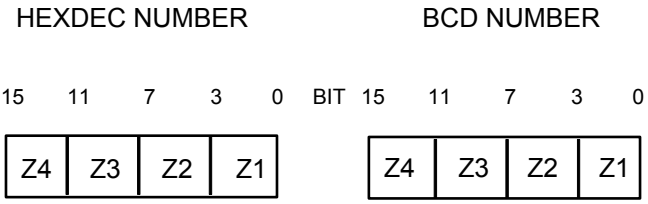
DUAL	WORD	%IW, %MW, %OW, %KW	Binary number
BCD	WORD	%OW,%MW	BCD coded number

DESCRIPTION

The binary number at the DUAL input is converted to a BCD coded number and is allocated to the operand at the output BCD.

The binary number is represented in 16 bits and must lie within the range $0 \leq DUAL \leq 270F_H$ (corresponding to BCD 9999). The BCD number is limited to 9999 if it lies outside this range. The BCD number is stored in a 16-bit word.

Definition :
The significance of the digits in a hexidecimal number and a BCD coded number is defined as follows :



Numerical value:

Z1 * 1
Z2 * 16
Z3 * 256
Z4 * 4096

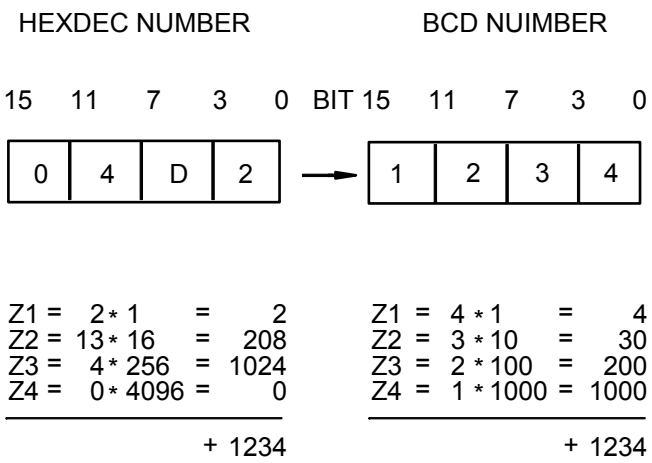
0 < Zi < F

Numerical value:

Z1 * 1
Z2 * 10
Z3 * 100
Z4 * 1000

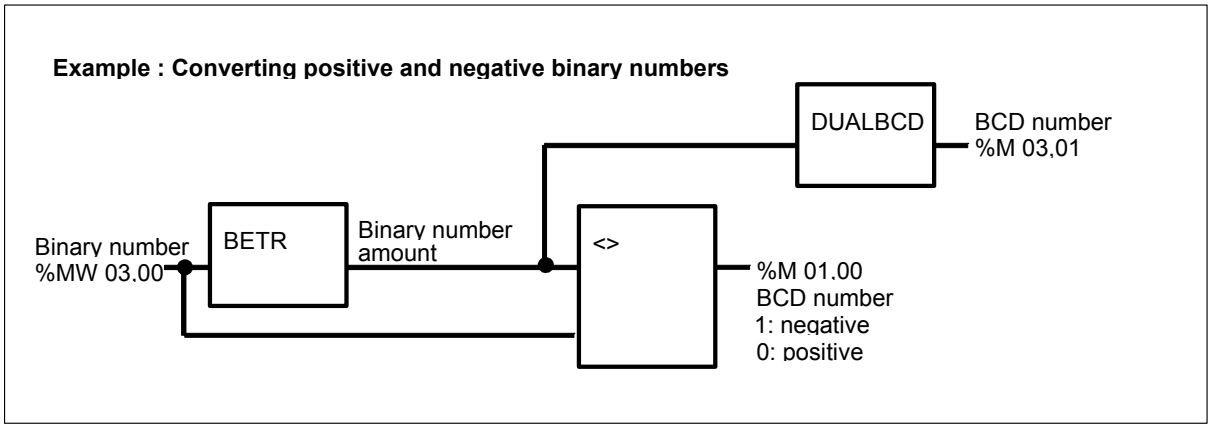
0 < Zi < 9

Example :



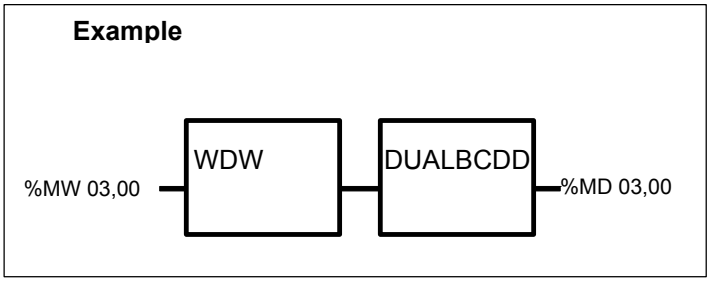
Conversion of a negative binary number to a BCD number

A negative binary number with an amount less than $270F_H$ can be converted to a BCD number, whereby the value and the sign of the BCD number are each stored in one flag.



Converting a binary number with an amount higher than $270F_H$

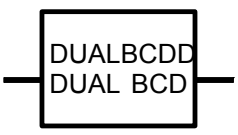
Binary numbers with an amount higher than $270F_H$ are first of all converted to a double word (function block WDW). They are then converted from BINARY to BCD by means of the DUALBCDD function block.



The same procedure as above applies if a sign has to be taken into account.

DUALBCDD BINARY TO BCD CONVERSION, DOUBLE WORD

FBD



IL

```
CAL DUALBCDD(DUAL,BCD)
or
!BA0
DWBCD
DUAL
BCD
```

PARAMETERS

DUAL	DOUBLE WORD	%MD, %KD	Binary number
BCD	DOUBLE WORD	%MD	BCD coded number

DESCRIPTION

The binary number at the input DUAL is converted to a BCD coded number and is allocated to the operand at the output BCD.

The binary number is represented in 32 bits and must lie within the range
$0 \leq$ DUAL $\leq 5F5E0FF_H$ (corresponding to BCD 99 999 999).
if the BCD number lies outside of this range, it is limited to 99 999 999. The BCD number is stored in a 32-bit word.

Definition

The significance of the digits in a hexadecimal number and a BCD coded number is defined as follows :

HEXDEC NUMBER

31 15 0 BIT

Z8	Z7	Z6	Z5	Z4	Z3	Z2	Z1

Numerical value:

Z1 * 1
Z2 * 16
Z3 * 256
Z4 * 4096
Z5 * 65536
Z6 * 1048576
Z7 * 16777216
Z8 * 268435456

0 < Zi < F

BCD NUMBER

31 15 0

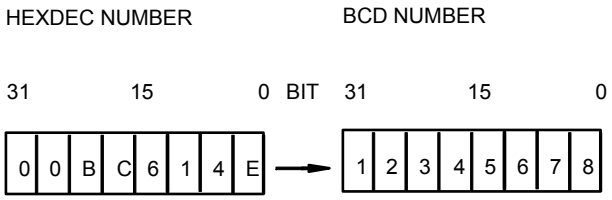
Z8	Z7	Z6	Z5	Z4	Z3	Z2	Z1

Numerical value:

Z1 * 1
Z2 * 10
Z3 * 100
Z4 * 1000
Z5 * 10000
Z6 * 100000
Z7 * 1000000
Z8 * 10000000

0 < Zi < 9

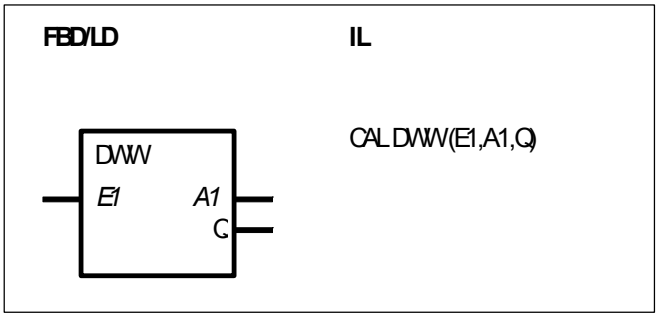
Example:



1 = 14 * 1	=	14	Z1 = 8 * 1	=	8
2 = 4 * 16	=	64	Z2 = 7 * 10	=	70
3 = 1 * 256	=	256	Z3 = 6 * 100	=	600
4 = 6 * 4096	=	24576	Z4 = 5 * 1000	=	5000
5 = 12 * 65536	=	786432	Z5 = 4 * 10000	=	40000
6 = 11 * 1048576	=	11534336	Z6 = 3 * 100000	=	300000
7 = 0 * 16777216	=	0	Z7 = 2 * 1000000	=	2000000
8 = 0 * 268435456	=	0	Z8 = 1 * 10000000	=	10000000
		+ 12345678			+ 12345678

Converting a negative binary number to a BCD number
See function block DUALBCD

DWW DOUBLE WORD TO WORD CONVERSION



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Double word variable to be converted
A1	WORD	%OW, %MW	Result of conversion, word variable
Q	BINARY	%O, %M	Result limited

DESCRIPTION

The value of the double word operand at the input E1 is converted to a word variable and the result is allocated to the word operand at the output A1.

The result is limited to the maximum or minimum number range.

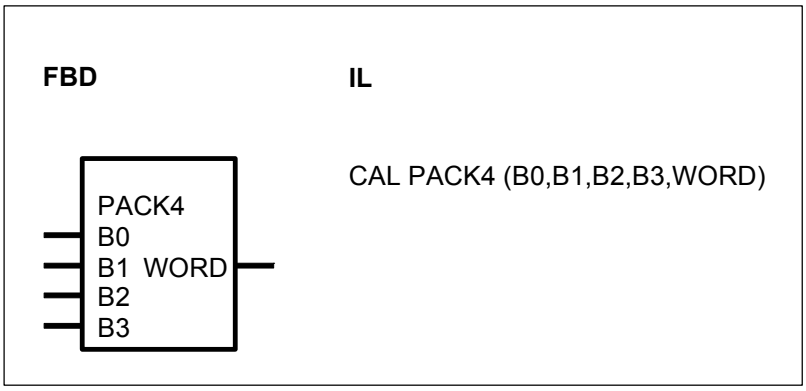
Max. number range : +32 767 ($7FFF_H$)

Min. number range : -32 767 (8001_H)

If limiting has taken place, a 1 signal is allocated to the binary operand at the output Q. If no limiting has taken place, a 0 signal is allocated to the binary operand at the output Q.

If the value of the operand at the input E1 lies outside of the permissible number range (value 8000 0000_H), the converted value is set to -32 767 (8001_H).

PACK4 PACK 4 BINARY VARIABLES IN A WORD



PARAMETERS

B0...B3	BINARY	%I, %M, %O, %K, %S	The 4 binary variables to be packed;
WORD	WORD	%OW, %MW	Word variable

DESCRIPTION

This function block packs 4 binary variables in one word variable.

B0-B3 BINARY
The binary variables to be packed are specified at the inputs B0...B3.

WORD WORD
The value of each binary variable at the inputs B0...B3 is loaded into the corresponding bit (bit 0 ... bit 3) of the variable at the output WORD.

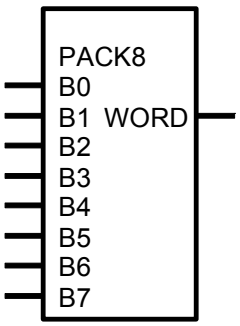
Affiliations

B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable
B2 -> bit 2 of the output variable
B3 -> bit 3 of the output variable

Note : The bits 4 to 15 of the output variable are occupied with the value 0.

PACK8 PACK 8 BINARY VARIABLES IN A WORD

FBD



IL

```
CAL PACK8 (B0,B1,B2,B3,B4,B5,B6,B7,WORD)
```

PARAMETERS

B0...B7	BINARY	%I, %M, %O, %K, %S	The 8 binary variables to be packed;
WORD	WORD	%OW, %MW	Word variable

DESCRIPTION

This function block packs 8 binary variables in one word variable.

B0-B3 BINARY
The binary variables to be packed are specified at the inputs B0...B7.

WORD WORD
The value of each binary variable at the inputs B0...B7 is loaded into the corresponding bit (bit 0 ... bit 7) of the variable at the output WORD.

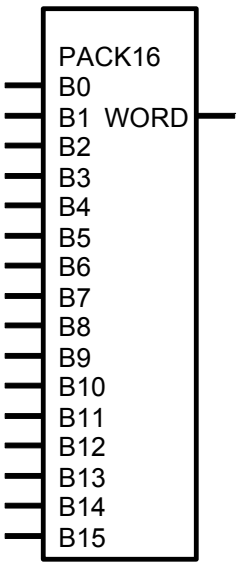
Affiliations

B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable
. . . .
B7 -> bit 7 of the output variable

Note : The bits 8 to 15 of the output variable are occupied with the value 0.

PACK16 PACK 16 BINARY VARIABLES IN A WORD

FBD



IL

```
CAL  PACK16 (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,
             B10,B11,B12,B13,B14,B15,WORD)
```

PARAMETERS

B0...B15	BINARY	%I, %M, %O, %K, %S	The 16 binary variables to be packed;
WORD	WORD	%OW, %MW	Word variable

DESCRIPTION

This function block packs 16 binary variables in one word variable.

B0-B3 BINARY
The binary variables to be packed are specified at the inputs B0...B15.

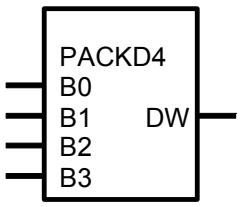
WORD WORD
The value of each binary variable at the inputs B0...B15 is loaded into the corresponding bit (bit 0 ... bit 15) of the variable at the output WORD.

Affiliations

B0	-> bit 0 of the output variable
B1	-> bit 1 of the output variable
.	. . .
B15	-> bit 15 of the output variable

PACKD4 PACK 4 BINARY VARIABLES IN A DOUBLE WORD

FBD



IL

```
CAL PACKD4 (B0,B1,B2,B3,DW)
```

PARAMETERS

B0...B3	BINARY	%I, %M, %O, %K, %S	The 4 binary variables to be packed;
DW	DOUBLE WORD	%MD	Double word variable

DESCRIPTION

This function block packs 4 binary variables in one double word variable.

B0-B3 BINARY
The binary variables to be packed are specified at the inputs B0...B3.

DW DOUBLE WORD
The value of each binary variable at the inputs B0...B3 is loaded into the corresponding bit (bit 0 ... bit 3) of the variable at the output DW.

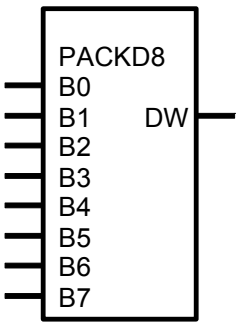
Affiliations

B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable
B2 -> bit 2 of the output variable
B3 -> bit 3 of the output variable

Note : The bits 4 to 31 of the output variable are occupied with the value 0.

PACKD8 PACK 8 BINARY VARIABLES IN A DOUBLE WORD

FBD



IL

```
CAL PACKD8 (B0,B1,B2,B3,B4,B5,B6,B7,DW)
```

PARAMETERS

B0...B7	BINARY	%I, %M, %O, %K, %S	The 8 binary variables to be packed;
DW	DOUBLE WORD	%MD	Double word variable

DESCRIPTION

This function block packs 8 binary variables in one double word variable.

B0-B7 BINARY
The binary variables to be packed are specified at the inputs B0...B7.

DW DOUBLE WORD
The value of each binary variable at the inputs B0...B7 is loaded into the corresponding bit (bit 0 ... bit 7) of the variable at the output DW.

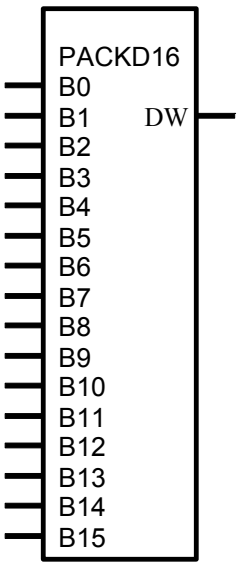
Affiliations

B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable
. . . .
B7 -> bit 7 of the output variable

Note : The bits 8 to 31 of the output variable are occupied with the value 0.

PACKD16 PACK 16 BINARY VARIABLES IN A DOUBLE WORD

FBD



IL

```
CAL PACKD16 (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,
             B10,B11,B12,B13,B14,B15,DW)
```

PARAMETERS

B0...B15	BINARY	%I, %M, %O, %K, %S	The 16 binary variables to be packed;
DW	DOUBLE WORD	%MD	Double word variable

DESCRIPTION

This function block packs 16 binary variables in one double word variable.

B0-B15 BINARY
The binary variables to be packed are specified at the inputs B0...B15.

DW DOUBLE WORD
The value of each binary variable at the inputs B0...B15 is loaded into the corresponding bit (bit 0 ... bit 15) of the variable at the output DW.

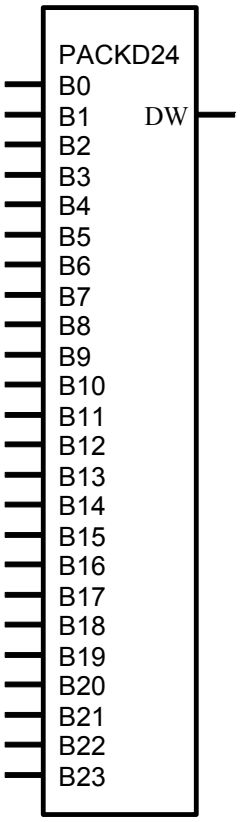
Affiliations

B0	-> bit 0 of the output variable
B1	-> bit 1 of the output variable
.	. . .
B15	-> bit 15 of the output variable

Note : The bits 16 to 31 of the output variable are occupied with the value 0.

PACKD24 PACK 24 BINARY VARIABLES IN A DOUBLE WORD

FBD



IL

```
CAL PACKD24 (B0,B1,B2,B3,B4,B5,B6,B7,B8
,B9,B10,B11,B12,B13,B14,B15,B16,B17,B18,
B19,B20,B21,B22,B23,DW)
```

PARAMETERS

B0...B23	BINARY	%I, %M, %O, %K, %S	The 24 binary variables to be packed;
DW	DOUBLE WORD	%MD	Double word variable

DESCRIPTION

This function block packs 24 binary variables in one double word variable.

B0-B23 BINARY
The binary variables to be packed are specified at the inputs B0...B23.

DW DOUBLE WORD
The value of each binary variable at the inputs B0...B23 is loaded into the corresponding bit (bit 0 ... bit 23) of the variable at the output DW.

Affiliations

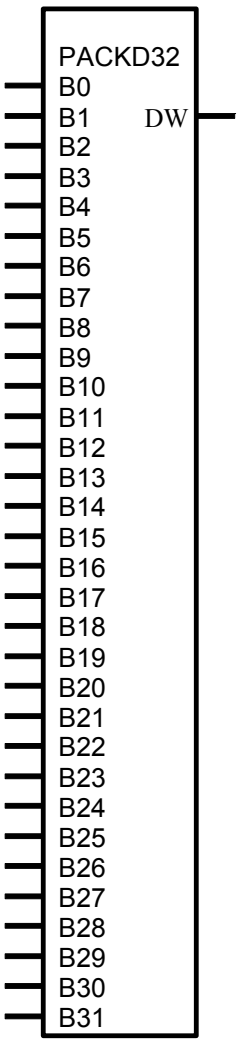
B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable

. . . .
B23 -> bit 23 of the output variable

Note : The bits 24 to 31 of the output variable are occupied with the value 0.

PACKD32 PACK 32 BINARY VARIABLES IN A WORD

FBD



IL

```
CAL PACKD32 (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,
            B11,B12,B13,B14,B15,B16,B17,B18,
            B19,B20,B21,B22,B23,B24,B25,B26,
            B27,B28,B29,B30,B31,DW)
```

PARAMETERS

B0...B31	BINARY	%I, %M, %O, %K, %S	The 32 binary variables to be packed;
DW	DOUBLE WORD	%MD	Double word variable

DESCRIPTION

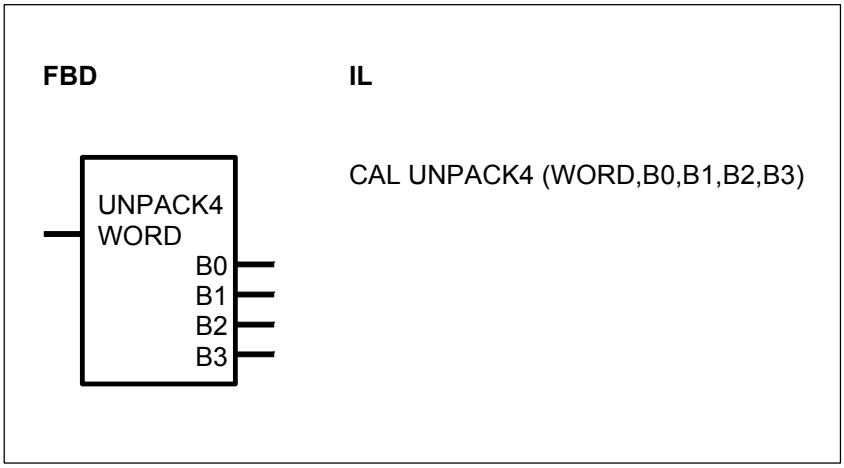
This function block packs 32 binary variables in one double word variable.

B0-B31 BINARY
The binary variables to be packed are specified at the inputs B0...B31.

DW DOUBLE WORD
The value of each binary variable at the inputs B0...B31 is loaded into the corresponding bit (bit 0 ... bit 31) of the variable at the output DW.

Affiliations
B0 -> bit 0 of the output variable
B1 -> bit 1 of the output variable
. . . .
B31 -> bit 31 of the output variable

UNPACK4 UNPACKING A WORD INTO 4 BINARY VARIABLES



PARAMETERS

WORD	WORD	%IW, %OW, %MW, %KW	Word variable to be unpacked
B0...B3	BINARY	%O, %M	The 4 binary output variables unpacked

DESCRIPTION

This function block unpacks the word variable at the input WORD. The bit 0 to bit 4 of the input variable is allocated to the corresponding output B0...B3.

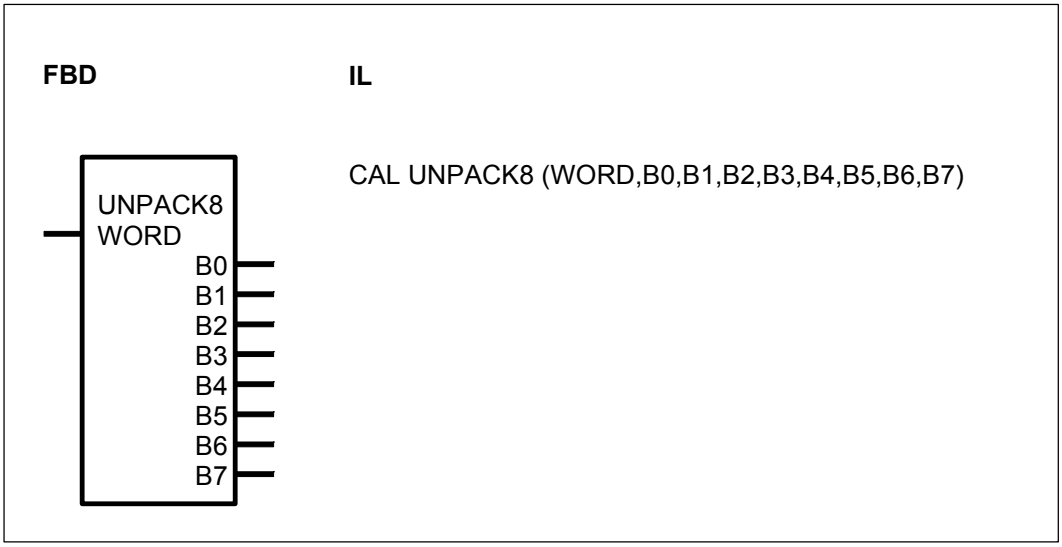
WORD WORD
The variable to be unpacked is specified at the input WORD. Each bit (bit 0...bit 3) of this input variable is allocated to the affiliated output variable (B0...B3).

B0-B3 BINARY
The affiliated bits of the variable at the input WORD are allocated to the binary outputs B0...B3.

Affiliation
Input variable Bit0 -> B0

Input variable Bit1 -> B1
Input variable Bit2 -> B2
Input variable Bit3 -> B3

UNPACK8 UNPACKING A WORD INTO 8 BINARY VARIABLES



PARAMETERS

WORD	WORD	%IW, %OW, %MW, %KW	Word variable to be unpacked
B0...B7	BINARY	%O, %M	The 8 binary output variables unpacked

DESCRIPTION

This function block unpacks the word variable at the input WORD. The bit 0 to bit 7 of the input variable is allocated to the corresponding output B0...B7.

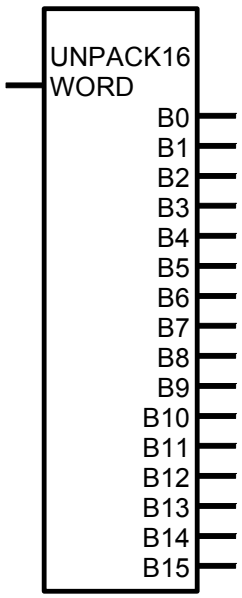
WORD WORD
The variable to be unpacked is specified at the input WORD. Each bit (bit 0...bit 7) of this input variable is allocated to the affiliated output variable (B0...B7).

B0-B7 BINARY
The affiliated bits of the variable at the input WORD are allocated to the binary outputs B0...B7.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit7 -> B7

UNPACK16 UNPACKING A WORD INTO 16 BINARY VARIABLES

FBD



IL

```
CAL UNPACK16 (WORD,B0,B1,B2,B3,B4,B5,B6,B7,
              B8,B9,B10,B11,B12,B13,B14,B15)
```

PARAMETERS

WORD	WORD	%IW, %OW, %MW, %KW	Word variable to be unpacked
B0...B15	BINARY	%O, %M	The 16 binary output variables unpacked

DESCRIPTION

This function block unpacks the word variable at the input WORD. The bit 0 to bit 15 of the input variable is allocated to the corresponding output B0...B15.

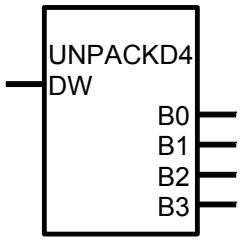
WORD WORD
The variable to be unpacked is specified at the input WORD. Each bit (bit 0...bit 15) of this input variable is allocated to the affiliated output variable (B0...B15).

B0-B15 BINARY
The affiliated bits of the variable at the input WORD are allocated to the binary outputs B0...B15.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit15 -> B15

UNPACKD4 UNPACKING A DOUBLE WORD INTO 4 BINARY VARIABLES

FBD



IL

```
CAL UNPACKD4 (DW,B0,B1,B2,B3)
```

PARAMETERS

DW	DOUBLE WORD	%MD	Double word variable to be unpacked
B0...B3	BINARY	%O, %M	The 4 binary output variables unpacked

DESCRIPTION

This function block unpacks the double word variable at the input DW. The bit 0 to bit 4 of the input variable is allocated to the corresponding output B0...B3.

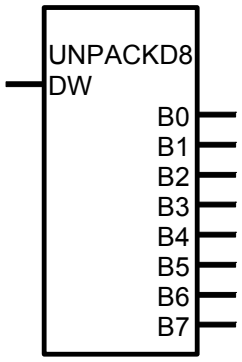
DW DOUBLE WORD
The variable to be unpacked is specified at the input DW. Each bit (bit 0...bit 3) of this input variable is allocated to the affiliated output variable (B0...B3).

B0-B3 BINARY
The affiliated bits of the variable at the input DW are allocated to the binary outputs B0...B3.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
Input variable Bit2 -> B2
Input variable Bit3 -> B3

UNPACKD8 UNPACKING A DOUBLE WORD INTO 8 BINARY VARIABLES

FBD



IL

```
CAL UNPACKD8 (DW,B0,B1,B2,B3,B4,B5,B6,B7)
```

PARAMETERS

DW	DOUBLE WORD	%MD	Double word variable to be unpacked
B0...B7	BINARY	%O, %M	The 8 binary output variables unpacked

DESCRIPTION

This function block unpacks the double word variable at the input DW. The bit 0 to bit 7 of the input variable is allocated to the corresponding output B0...B7.

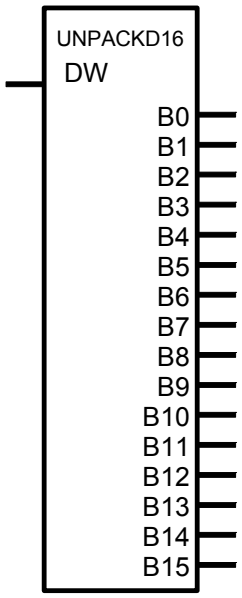
DW DOUBLE WORD
The variable to be unpacked is specified at the input DW. Each bit (bit 0...bit 7) of this input variable is allocated to the affiliated output variable (B0...B7).

B0-B7 BINARY
The affiliated bits of the variable at the input DW are allocated to the binary outputs B0...B7.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit7 -> B7

UNPACKD16 UNPACKING A DOUBLE WORD INTO 16 BINARY VARIABLES

FBD



IL

```
CAL UNPACKD16 (DW,B0,B1,B2,B3,B4,B5,B6,B7,
               B8,B9,B10,B11,B12,B13,B14,B15)
```

PARAMETERS

DW	DOUBLE WORD	%MD	Double word variable to be unpacked
B0...B15	BINARY	%O, %M	The 16 binary output variables unpacked

DESCRIPTION

This function block unpacks the word variable at the input DW. The bit 0 to bit 15 of the input variable is allocated to the corresponding output B0...B15.

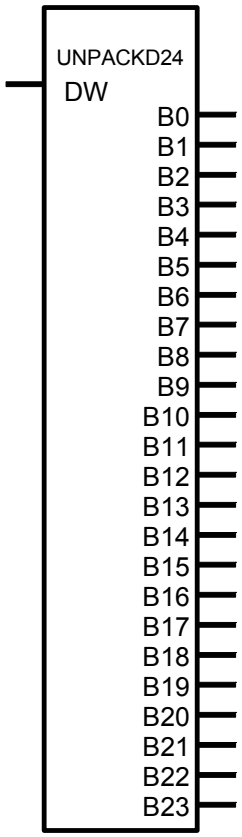
DW DOUBLE WORD
The variable to be unpacked is specified at the input DW. Each bit (bit 0...bit 15) of this input variable is allocated to the affiliated output variable (B0...B15).

B0-B15 BINARY
The affiliated bits of the variable at the input DW are allocated to the binary outputs B0...B15.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit15 -> B15

UNPACKD24 UNPACKING A DOUBLE WORD INTO 24 BINARY VARIABLES

FBD



IL

```
CAL UNPACKD24 (DW,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,
               B10,B11,B12,B13,B14,B15,B16,B17,
               B18,B19,B20,B21,B22,B23)
```

PARAMETERS

DW	DOUBLE WORD	%MD	Double word variable to be unpacked
B0...B23	BINARY	%O, %M	The 24 binary output variables unpacked

DESCRIPTION

This function block unpacks the word variable at the input DW. The bit 0 to bit 23 of the input variable is allocated to the corresponding output B0...B23.

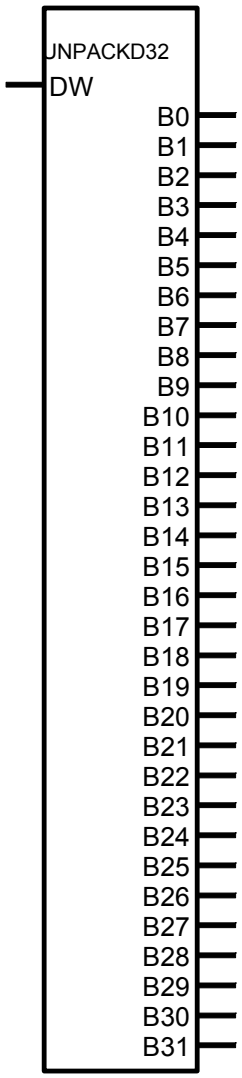
DW DOUBLE WORD
The variable to be unpacked is specified at the input DW. Each bit (bit 0...bit 23) of this input variable is allocated to the affiliated output variable (B0...B23).

B0-B23 BINARY
The affiliated bits of the variable at the input DW are allocated to the binary outputs B0...B23.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit23 -> B23

UNPACKD32 UNPACKING A DOUBLE WORD INTO 32 BINARY VARIABLES

FBD



IL

```
CAL UNPACKD32 (DW,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,
               B10,B11,B12,B13,B14,B15,B16,B17,
               B18,B19,B20,B21,B22,B23,B24,B25,
               B26,B27,B28,B29,B30,B31)
```

PARAMETERS

DW	DOUBLE WORD	%MD	Double word variable to be unpacked
B0...B31	BINARY	%O, %M	The 32 binary output variables unpacked

DESCRIPTION

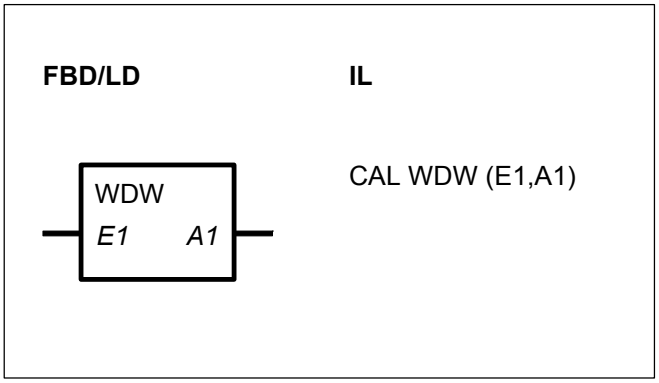
This function block unpacks the word variable at the input DW. The bit 0 to bit 31 of the input variable is allocated to the corresponding output B0...B31.

DW DOUBLE WORD
The variable to be unpacked is specified at the input DW. Each bit (bit 0...bit 31) of this input variable is allocated to the affiliated output variable (B0...B31).

B0-B31 BINARY
The affiliated bits of the variable at the input DW are allocated to the binary outputs B0...B31.

Affiliation
Input variable Bit0 -> B0
Input variable Bit1 -> B1
. . .
Input variable Bit31 -> B31

WDW WORD TO DOUBLE WORD CONVERSION



PARAMETERS

E1	WORD	%IW, %OW, %MW, %KW	Word quantity to be converted
A1	DOUBLE WORD	%MD	Result of conversion, double word quantity

DESCRIPTION

The value of the word operand at the input E1 is converted to a double word quantity and the result is allocated to the double word operand at the output A1.

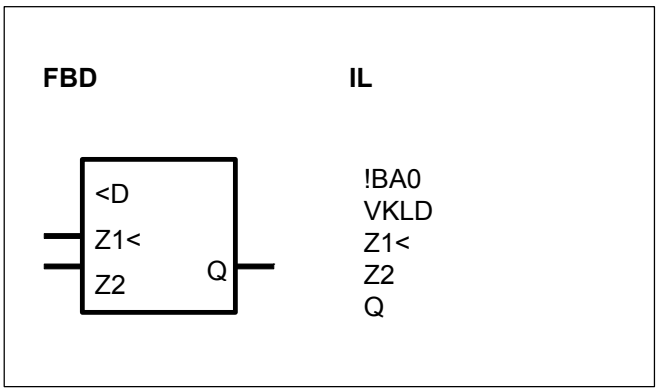
Value range for E1 : $8000_H \leq E1 \leq 7FFF_H$
$-32768 \leq E1 \leq 32767$

8 Standard double word functions

8.1 Comparison functions, double word

Comparison functions, double word		serie C[tler]	40	50	90	30
	from pages C-200 to C-203					
<D / VKLD	Less than, double word		x	x	x	
=?D / VGLD	Equal, double word		x	x	x	
>D / VGRD	Greater than, double word		x	x	x	

<D LESS THAN, DOUBLE WORD



PARAMETERS

Z1>	DOUBLE WORD	%MD, %KD	Value to be compared
Z2	DOUBLE WORD	%MD, %KD	Comparison value
Q	BINARY	%O, %M	Result of the comparison

DESCRIPTION

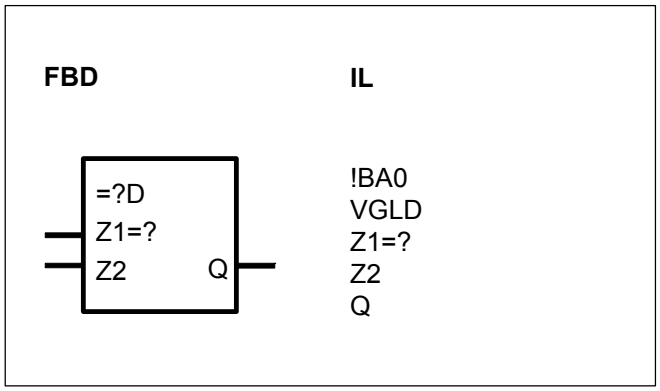
The value of the operand at the input Z1< is compared to the value of the operand at the input Z2.
The state 1 is allocated to the operand at the output Q if the value at Z1< is less than the one at Z2. The state 0 is allocated to Q if Z1< is equal to or greater than Z2.

Number range

Integer double word (32 Bit)

• low limit :	$8000\ 0000_H$	-2 147 483 648
• high limit :	$7FFF\ FFFF_H$	+2 147 483 647

=?D EQUAL, DOUBLE WORD



PARAMETES

Z1=?	DOUBLE WORD	%MD, %KD	Value to be compared
Z2	DOUBLE WORD	%MD, %KD	Comparison value
Q	BINARY	%O, %M	Result of the comparison

DESCRIPTION

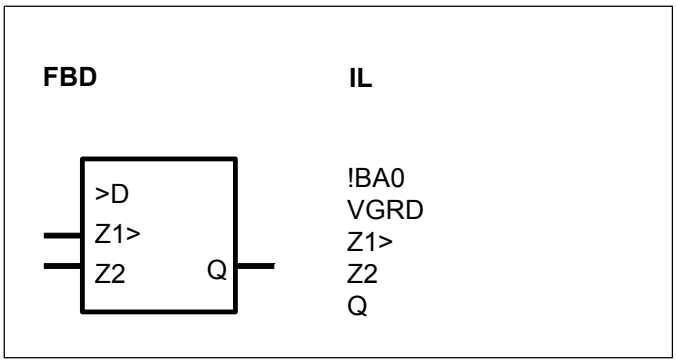
The value of the operand at the input Z1=? is compared to the value of the operand at the input Z2. If the vaule at Z1=? is identical to the one at Z2, the state 1 is allocated to the operand at the output Q. The state 0 is allocated to Q if Z1=? is unequal to Z2.

Number range

Integer double word (32 Bit)

• low limit :	$8000\ 0000_H$	-2 147 483 648
• high limit :	$7FFF\ FFFF_H$	+2 147 483 647

>D GREATER THAN, DOUBLE WORD



PARAMETERS

Z1>	DOUBLE WORD	%MD, %KD	Value to be compared
Z2	DOUBLE WORD	%MD, %KD	Comparison value
Q	BINARY	%O, %M	Result of the comparison

DESCRIPTION

The value of the operand at the input Z1> is compared to the value of the operand at the input Z2.
The state 1 is allocated to the operand at the output Q if the value at Z1> is greater than the one at Z2. The state 0 is allocated to Q if Z1> is equal to or less than Z2.

Number range

Integer double word (32 Bit)

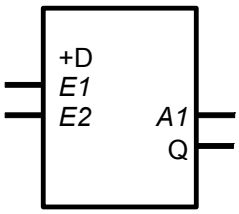
• low limit :	$8000\ 0000_H$	-2 147 483 648
• high limit :	$7FFF\ FFFF_H$	+2 147 483 647

8.2 Arithmetic functions, double word

Arithmetic functions, double word	from pages C-203 to C-212	serie C^{tler} 40	50	90	30
+D / ADDD	Addition, double word	x	x	x	
-D / SUBD	Subtraction, double word	x	x	x	
*D / MULD	Multiplication, double word	x	x	x	
:D / DIVD	Division, double word	x	x	x	
=D / ZUWD	Allocation, double word	x	x	x	
BETRD	Absolute value generator, double word			x	
MUL2ND	Double word multiplication by 2 to the power of N			x	
NEGD	Negation, double word			x	
SQRT	Square root	x	x	x	

+D ADDITION DOUBLE WORD

FBD



IL

```
!BA0
ADDD
E1
E2
A1
Q
```

PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Summand 1
E2	DOUBLE WORD	%MD, %KD	Summand 2
A1	DOUBLE WORD	%MD	Total
Q	BINARY	%O, %M	Total, limited

DESCRIPTION

The value of the operand at the input E1 is added to the value of the operand at the input E2 and the result is allocated to the operand at the output A1.

The result is limited to the maximum or minimum value of the number range. If limiting has taken place, a 1 signal is allocated to the binary operand at the output Q. If no limiting has taken place, a 0 signal is allocated to the binary operand at the output Q.

Number range
Integer, double word (32 bits).

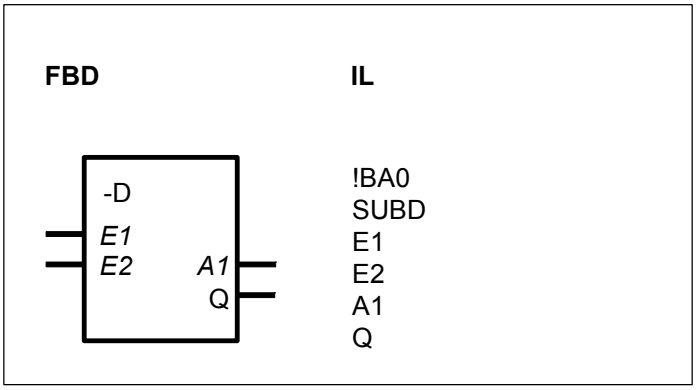
The following particularly applies here to the inputs E1 and E2 :

• Low limit :	8000 0000$_H$	- 2 147 483 648
• High limit :	7FFF FFFF$_H$	+2 147 483 647

The following applies to the output A1 :

• Low limit :	8000 0001$_H$	-2 147 483 647
• High limit :	7FFF FFFF$_H$	+2 147 483 647

-D SUBTRACTION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Minuend
E2	DOUBLE WORD	%MD, %KD	Subtrahend
A1	DOUBLE WORD	%MD	Result (difference)
Q	BINARY	%O, %M	Result, limited

DESCRIPTION

The value of the operand at the input E2 is subtracted from the value of the operand at the input E1 and the result is allocated to the operand at the output A1.

The result is limited to the maximum or minimum value of the number range.
If limiting has taken place, a 1 signal is allocated to the binary operand at the output Q. If no limiting has taken place, a 0 signal is allocated to the binary operand at the output Q.

The value of the operand at the input E2 is checked before subtraction to determine whether or not it lies outside of the permissible number range (8000 0000$_H$). If this is the case, calculation is done with the value -2 147 483 647 (8000 0001$_H$) instead of this inadmissible value.

Number range
Integer double word (32 bits).

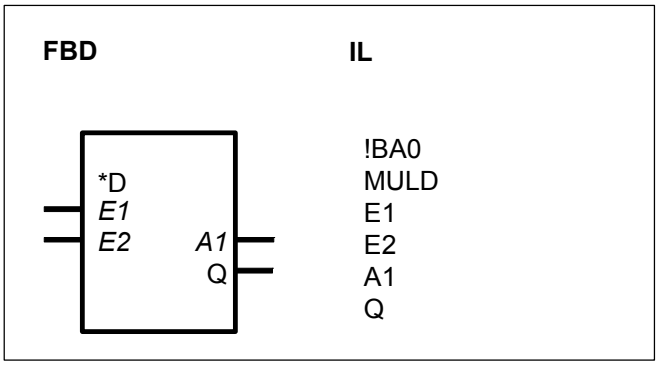
The following especially applies here to the input E1 :

• Low limit :	8000 0000$_H$	- 2 147 483 648.
• High limit :	7FFF FFFF$_H$	+ 2 147 483 647

The following applies to the input E2 and to output A1 :

• Low limit :	8000 0001$_H$	- 2 147 483 647
• High limit :	7FFF FFFF$_H$	+ 2 147 483 647
• Inadmissible value :	8000 0000$_H$	

*D MULTIPLICATION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Multiplicand
E2	DOUBLE WORD	%MD, %KD	Multiplier
A1	DOUBLE WORD	%MD	Result (Product)
Q	BINARY	%O, %M	Result limited

DESCRIPTION

The value of the operand at the input E1 is multiplied by the value of the operand at the input E2 and the result is allocated to the operand at the output A1.

The result is limited to the maximum or minimum value of the number range.
If limiting has taken place, a 1 signal is allocated to the binary operand at the output Q. If no limiting has taken place, a 0 signal is allocated to the binary operand at the output Q.

Number range
Integer double word (32 Bits)

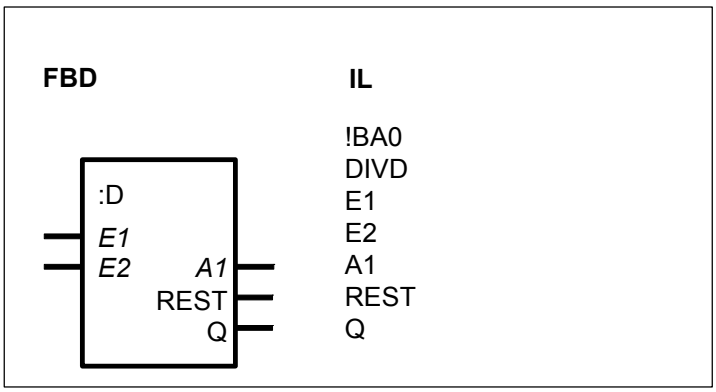
The following specially applies here to inputs E1 and E2 :

• Low limit :	8000 0000$_H$	-2 147 483 648
• High limit :	7FFF FFFF$_H$	+2 147 483 647

The following applies to the output A1 :

• Low limit :	8000 0001$_H$	-2 147 483 647
• High limit :	7FFF FFFF$_H$	+2 147 483 647

:D DIVISION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Dividend
E2	DOUBLE WORD	%MD, %KD	Divisor
A1	DOUBLE WORD	%MD	Result (quotient)
REST	DOUBLE WORD	%MD	Rest
Q	BINARY	%O, %M	Result limited

DESCRIPTION

The value of the operand at the input E1 is divided by the value of the operand at the input E2 and the result is allocated to the operand at the output A1, the remainder being allocated to the operand at the output REST.

If a remainder is produced, the result will always be rounded down.

If the result lies outside of the permissible number range, it will be limited to the maximum or minimum value of the number range.
If limiting has taken place, a 1 signal is allocated to the binary operand at the output Q and the value 0 is allocated to the output REST. If no limiting has taken place, a 0 signal is allocated to the binary operand at the output Q.

Division by “zero” is therfore also signalled at the binary output Q.

Remainder handling

If division results in a remainder, this is available at the double word output REST. The result of division is always rounded down if a remainder occurs.

Example : 3 : 3 = 1 Remainder 0
4 : 3 = 1 Remainder 1
5 : 3 = 1 Remainder 2
6 : 3 = 2 Remainder 0

As the remainder is available at the output REST, the user can compare this to the divisor and can round the result at the output A1 to suit his own requirements.
Example : Remainder $\geq$ divisor/2 -> round up the result at A1.

Division by ”zero”
If the divisor has the value “zero”, the positive or negative limit of the number range is allocated to the output A1.
The following applies to division by “zero” :
A1 = -2 147 483 647 (8000 0001_H) if the dividend is negative.
A1 = +2 147 483 647 (7FFF $FFFF_H$) if the dividend is positive.
REST = 0 Output for the remainder
Q = 1 Output to signal that the value at the output A1 has been limited

Invalid result value
If the invalid value 8000 0000_H is the result of division, this will be corrected to the permissible limit 8000 0001_H (-2 147 483 647), the binary output Q will be set to the value 1 and the output REST will be set to the value 0.

Number range
Integer double word (32 bits)

Here, the following particularly applies to the inputs E1 and E2 :

• Low limit :	8000 0000_H	-2 147 483 648
• High limit :	7FFF $FFFF_H$	+2 147 483 647

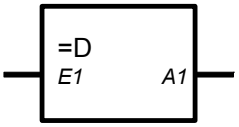
The following applies to the outputs A1 and REST :

• Low limit :	8000 0001_H	-2 147 483 647
• High limit :	7FFF $FFFF_H$	+2 147 483 647

Caution : the limit of E2 for 40&50 serie is –32767, +32767

=D ALLOCATION, DOUBLE WORD

FBD



IL

```
!BA0
ZUWD
E1
A1
```

PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Source
A1	DOUBLE WORD	%MD	Target

DESCRIPTION

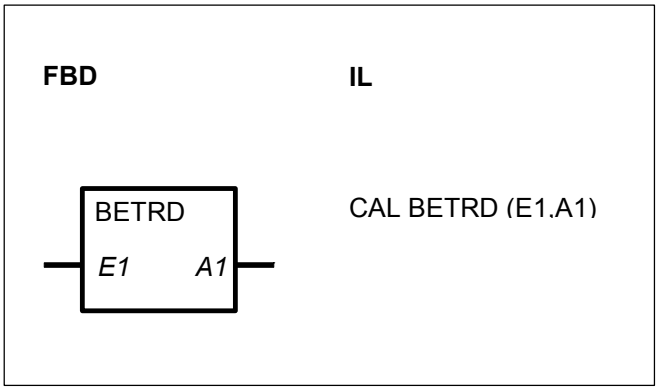
The value of the operand at the input E1 is allocated to the operand at the output A1.

If the inadmissible value 8000 0000_H should appear at the input for any particular reason, the permissible value 8000 0001_H (-2 147 483 647) will be allocated to the output A1. Therefore, the inadmissible value will be corrected.

Number range
Integer double word (32 bits)

• Low limit :	8000 0001_H	-2 147 483 647
• High limit :	7FFF $FFFF_H$	+2 147 483 647
• Inadmissible value :	8000 0000_H	---

BETRD ABSOLUTE VALUE GENERATOR, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input value
A1	DOUBLE WORD	%MD	Absolute value of the input value

DESCRIPTION

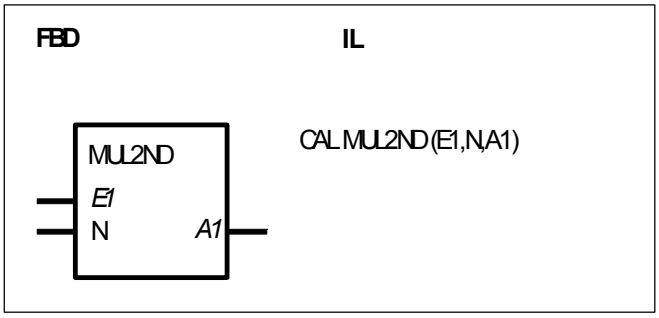
The absolute value of the operand at the input E1 is generated and the result is allocated to the operand at the output A1.

If, for any particular reason, the invalid value 8000 0000_H (-2 147 483 648) is present at the input E1, the value 7FFF $FFFF_H$ (+2 147 483 697) is allocated to the output A1. Therefore, the invalid value 8000 0000_H is first of all corrected to the valid value 8000 0001_H and only then the absolute value is generated.

Number range
Integer double word (32 bits)

• low limit :	8000 0001_H	-2 147 483 647
• high limit :	7FFF $FFFF_H$	+2 147 483 647
• invalid value :	8000 0000_H	---

MUL2ND DOUBLE WORD MULTIPLICATION BY 2 TO THE POWER OF N



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input operand
N	WORD	%IW, %OW, %MW, %KW	Quantity
A1	DOUBLE WORD	%MD	Result

DESCRIPTION

The value of the operand at the input E1 is shifted N times and bit-by-bit.
If the value at the input N is positive, shifting takes place to the left. For each shift by 1 bit position, this corresponds to multiplying the current value by 2.
If the value at the input N is negative, shifting takes place to the right. For each shift, this corresponds to division of the current value by 2.
The result is allocated to the operand at the output A1.

Meaningful range for N : $-30 \leq N \leq +30$
If N = 0, the value at the input E1 is passed directly to the output A1.

Sign of the value at the input E1 :
The sign of the value E1 is not influenced by the shift operation. That is to say, the sign of the output value is always identical to that of the input value.

Left shifting (Multiplication) :
When the value at the input is shifted to the left, the bit 0 released in each case is filled up with 0. The sign bit (bit 31) is not changed because setting to the limit of the number range takes place beforehand.

Limiting of the value at the output A1 during left shifting :

• The following applies to positive values at the input E1 :
If bit 30 is assigned a “1” and if shift operations still have to be carried out as the result of the value at the input N, these will no longer be executed. Instead, the output will be set to the limit of the postivie number range. That is to say, the limit has been reached in any case at the latest after shifting 30 times.
Output A1 = +2 147 483 647 ($7FFF\ FFFF_H$).

• The following applies to negative values at the input E1 :
If bit 30 is assigned a “0” and if shift operations still have to be carried out as the result of the value at the input N, these will no longer be executed. Instead, the output will be set to the limit of the postivie number range. That is to say, the limit has been reached in any case at the latest after shifting 30 times.
Output A1 = -2 147 483 647 ($8000\ 0001_H$).

Shifting to the right (division) :
When shifting to the right, each bit is moved to the right by one position. At the same time, the sign bit (bit 31) always retains its value. The bit released (bit 30) is filled up with the value of the sign bit in each case.

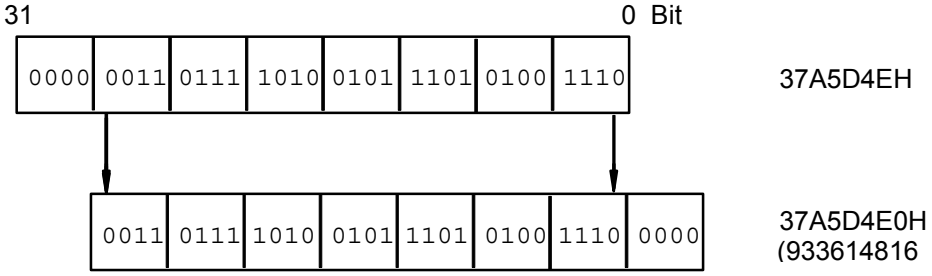
Limiting of the value at the output when shifting to the right :

• The following applies to positive values at the input :
If now only bit 0 has a “1” and if shift operations still have to be executed as the result of the value at the input N, the output will be set to the value 0. That is to say, the value 0 is reached in any case at the latest after shifting 30 times.
Output A1 = 0.

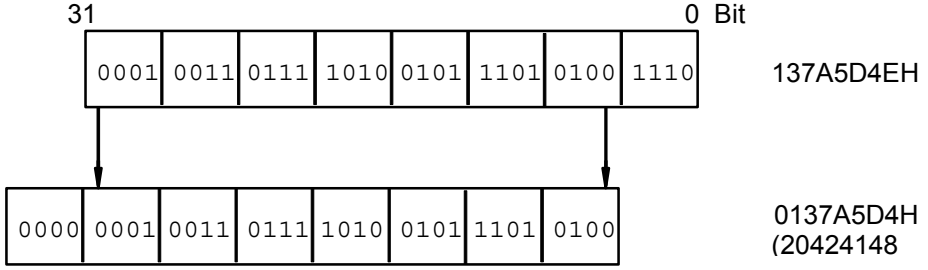
• The following applies to negative values at the input E1 :
If bits 0...31 have a “1” as the result of shifting, the limit (-1) has been reached. Further shifts have no effect. That is to say, the value -1 has been reached in any case at the latest after shifting 31 times.
Output A1 = -1 ($FFFF\ FFFF_H$).

Examples

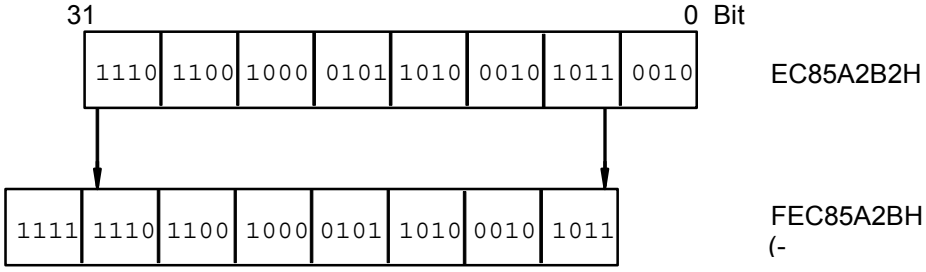
1. Input value E1 = 58350926 ($37A5D4E_H$)
   Exponent N = 4 --> 4 * Left shift



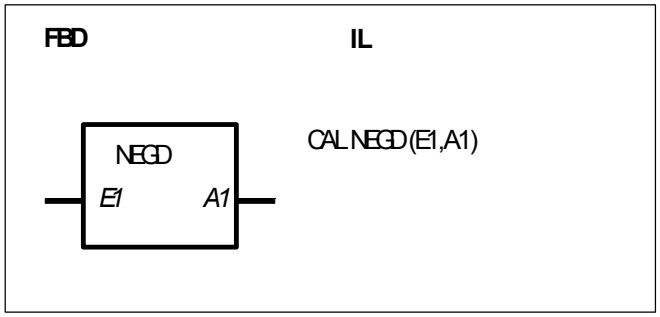
2. Input value E1 = 326786382 ($137A5D4E_H$)
   Exponent N = - 4 --> 4 * Right shift



3. Input value E1 = -326786382 ($EC85A2B2_H$)
Exponent N = - 4 --> 4 * Right shift



NEGD NEGATION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input value
A1	DOUBLE WORD	%MD	Negated value

DESCRIPTION

The value of the operand at the input E1 is negated and the result is allocated to the operand at the output A1.

If the inadmissible value 8000 0000_H (-2 147 483 648) is present at the input E1, the value 7FFF $FFFF_H$ (+2 147 483 647) is allocated to the output A1. Therefore, the inadmissible value is replaced by the admissible value 8000 0001_H (-2 147 483 647) before negation.

Number range
Integer double word (32 Bit).

• Low limit :	8000 0001_H	-2 147 483 647
• High limit :	7FFF $FFFF_H$	+2 147 483 647
• Inadmissible value :	8000 0000_H	

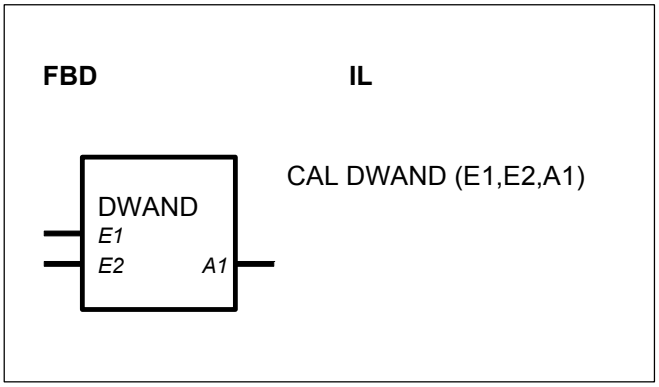
SQRT SQUARE ROOT

Please refer to the description of the SQRT function block from the chapter :
"2.5. Arithmetic functions, word".

8.3 Logical functions, double word

Logical functions, double word	from pages C-212 to C-216	serie C[tler] 40	50	90	30
DWAND	AND combination, double word	x	x	x	
DWOR	OR combination, double word	x	x	x	
DWXOR	Exclusive OR combination, double word	x	x	x	
MASKED	Mask, double word			x	
SHIFT	Shift block			x	

DWAND AND COMBINATION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Operand 1
E2	DOUBLE WORD	%MD, %KD	Operand 2
A1	DOUBLE WORD	%MD	Result of the AND combination

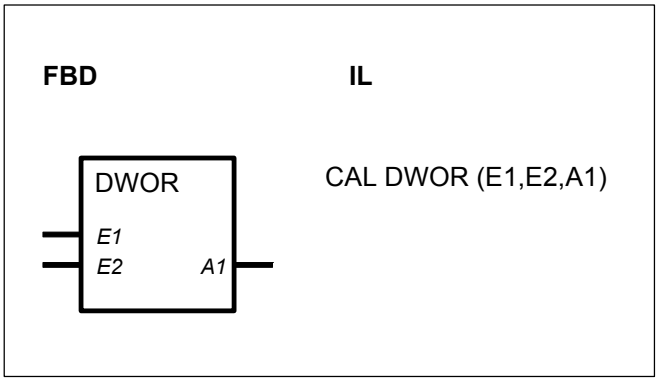
DESCRIPTION

This function block generates the bit-by-bit AND combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

Example

E1	1.0.0.0	0.0.1.1	0.0.1.0	0.1.1.0	1.0.1.0	1.1.0.0	0.0.1.1	0.1.0.1
E2	1.0.0.1	0.1.1.0	0.0.1.0	1.1.1.1	1.1.1.1	0.0.0.0	0.1.1.0	1.1.0.0
A1	1.0.0.0	0.0.1.0	0.0.1.0	0.1.1.0	1.0.1.0	0.0.0.0	0.0.1.0	0.1.0.0

DWOR OR COMBINATION, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Operand 1
E2	DOUBLE WORD	%MD, %KD	Operand 2
A1	DOUBLE WORD	%MD	Result of the OR combination

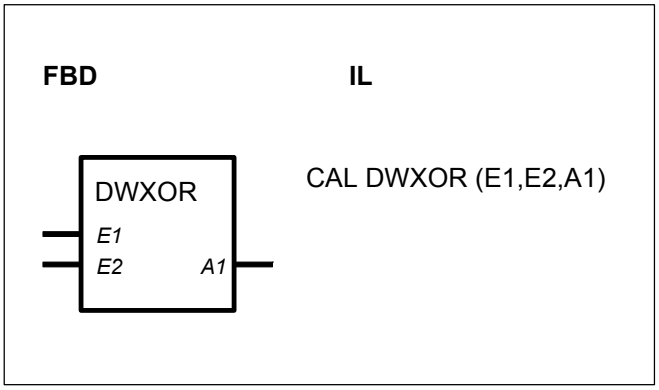
DESCRIPTION

This function block generates the bit-by-bit OR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

Example

E1	1.0.0.0	0.0.1.1	0.0.1.0	0.1.1.0	1.0.1.0	1.1.0.0	0.0.1.1	0.1.0.1
E2	1.0.0.1	0.1.1.0	0.0.1.0	1.1.1.1	1.1.1.1	0.0.0.0	0.1.1.0	1.1.0.0
A1	1.0.0.1	0.1.1.1	0.0.1.0	1.1.1.1	1.1.1.1	1.1.0.0	0.1.1.1	1.1.0.1

DWXOR EXCLUSIVE OR COMBINATION, DOUBLE WORD



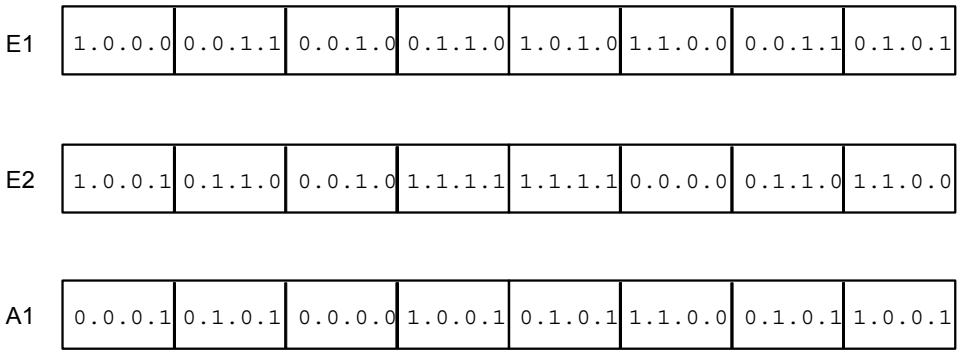
PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Operand 1
E2	DOUBLE WORD	%MD, %KD	Operand 2
A1	DOUBLE WORD	%MD	Result of the XOR combination

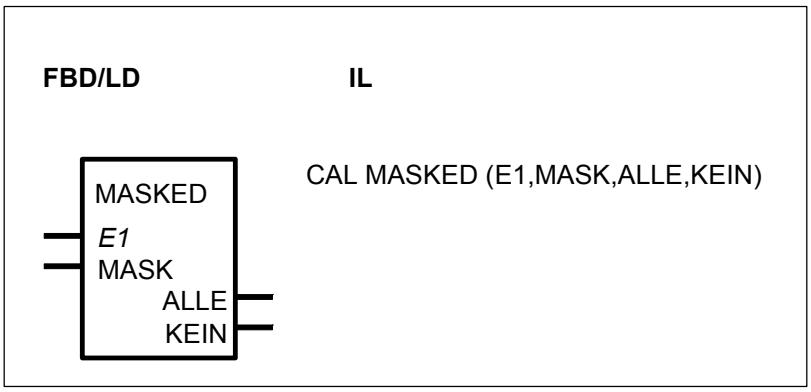
DESCRIPTION

This function block generates the bit-by-bit XOR combination of the operands present at the inputs E1 and E2. The result is allocated to the operand at the output A1.

Example



MASKED MASK, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input value
MASK	DOUBLE WORD	%MD, %KD	Mask
ALLE	BINARY	%O, %M	All bits agree
KEIN	BINARY	%O, %M	No bit agrees

DESCRIPTION

The individual bits of the operand at the input E1 are compared to the bits of the operand at the input MASK. The result of the comparison is signalled at the outputs ALLE and KEIN.

If at least *all* bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 1
KEIN = 0

If *none* of the bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 0
KEIN = 1

If only *some* of the bits, which are set on the operand at the input MASK, are set on the operand at the input E1, the following applies to the outputs : ALLE = 0
KEIN = 0

Example

```
E1     :  X1111XX11XXXX11X  X1XXXXX1XXX1XXXX  ALLE = 1
MASK   :  0111100110000110  0100000100010000  KEIN = 0


E1     :  X0000XX00XXXX00X  X0XXXXX0XXX0XXXX  ALLE = 0
MASK   :  0111100110000110  0100000100010000  KEIN = 1


E1     :  X1011XX10XXXX11X  X1XXXXX0XXX1XXXX  ALLE = 0
MASK   :  0111100110000110  0100000100010000  KEIN = 0
```

X : These bits may have any values (don't care).

SHIFT SHIFT BLOCK

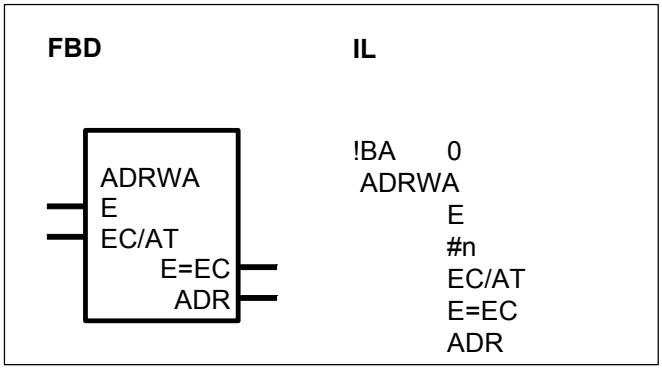
Please refer you to the description of the SHIFT function block from the chapter : "2.6. Logical functions, word".

9 High order functions

High order functions	serie C[tler] from pages C-216 to C-291	40	50	90	30
ADRWA	Address selection			x	
AMELD	Analog value change annonciator			x	
AMELDD	Analog value change annonciator, double word			x	
ANAI4_20	Read analog value 4...20 mA (07KT92)			x	
AWM	Selection multiplexer			x	
AWT	Selection gate, word	x	x	x	x
AWTB	Binary selection gate	x	x	x	x
AWTD	Selection gate, double word			x	
BEG	Limiter	x	x	x	x
BEGD	Limiter, double word			x	
BITSU	Bit searcher			x	
BMELD	Binary value change annunciator	x	x	x	
DMUX	Demultiplexer			x	
DMUXD	Demultiplexer, double word			x	
DWUMC	Double word decoder			x	
FEHSU	Error searcher with automatic deletion			x	
FIFO	Stack, first-in / first-out			x	
FKG	Function generator	x	x	x	
HLG	Ramp function generator			x	
IDLB	Read binary variable, indexed	x	x	x	
IDLm / IDL	Read word variable, indexed	x	x	x	x
IDSB	Write binary variable, indexed	x	x	x	
IDSm / IDS	Write word variable, indexed	x	x	x	x
INITS	Initialize memory area in the operand memory with zero			x	
INITV	Initialize variables			x	
LDT	Illumination pushbutton control			x	
LIFO	Stack, last-in / first-out			x	
LIZU	List allocator	x	x	x	x
MAX	Maximum value generator	x	x	x	x
MAXD	Maximum value generator, double word			x	
MAZ	Maximum value generator as a function of time			x	
MAZD	Maximum value generator as a function of time, double word			x	
MIN	Minimum value generator	x	x	x	x
MIND	Minimum value generator, double word			x	
MUXR	Multiplexer with reset			x	
MUXRD	Multiplexer with reset, double word			x	
NPULSE	Pulses generator	x	x		
SFEHSU	Error searcher with storage			x	
UHR	Clock	x	x	x	x
USM	Switchover multiplexer			x	

UST	Switchover gate	x
USTD	Switchover, double word	x
USTR	Switchover gate with reset	x
USTRD	Switchover with reset, double word	x
WDEC	Word decoder	x
WUMC	Word recoder	x

ADRWA ADDRESS SELECTION



PARAMETERS

E	WORD	%IW, %OW, %MW, %KW	Input value
#n	DIRECT CONSTANT	#, #H	Quantity EC or AT
EC/AT	WORD	%IW, %OW, %MW, %KW	Input code/output table; **capable of duplication**
E=EC	BINARY	%O, %M	Input value = input code
ADR	WORD	%OW, %MW	indirect address

DESCRIPTION

This block generates indirect addresses at the output ADR.

Definition : An indirect address is an operand whose value is the address of another operand.

E WORD

Input value to be compared with the input codes.

EC/AT WORD

The input EC/AT can be duplicated. The input codes for the comparison with the input E are specified at the inputs EC/AT_0...EC/AT_{n-1} The output table of operands whose indirect addresses are to be generated are specified at the inputs EC/AT_n ... EC/AT_{2n-1}.

The block compares the value at the input E successively against the values at the inputs EC/AT_0 ... EC/AT_{n-1}. The comparison is restarted each time the block is called, i.e. it begins with the input EC/AT_0.

If the value at the input E agrees with one of the values at the inputs EC/AT_0...EC/AT_{n-1}:
- The output E=EC is set to 1 (hit),
- The *allocated* operand is selected from the inputs EC/AT_n ... EC/AT_{2n-1}

If the value at the input E does not agree with one of the values at the inputs EC/AT_0...EC/AT_{n-1}:
- The output E=EC is set to 0 (no hit)
- *no* operand is selected from the output table EC/AT_n ... EC/AT_{2n-1} and accordingly no indirect address is generated either.

Convention for allocation between EC/AT_0...EC/AT_{n-1} and EC/AT_n ... EC/AT_{2n-1}:

EC/AT0 -> EC/AT_n
EC/AT1 -> EC/AT_{n+1}
. .
EC/AT_{n-1} -> EC/AT_{2n-1}

#n DIRECT CONSTANT
Only used in IL. The number of input codes EC/AT_0...EC/AT_{n-1} is specified at the input #n. It is specified as a direct constant.

Example: EC/AT0, ..., EC/AT5 are planned
-> input codes = EC/AT0, ..., EC/AT2 and output table = EC/AT0, ..., EC/AT5
-> #n = 3.

E=EC BINARY
The output E=EC indicates whether or not the value at the input E agrees with one of the values at the inputs EC/AT_0...EC/AT_{n-1}.

E=EC = 0 ->	No agreement
E=EC = 1 ->	The value at the input E agrees with one of the values at the inputs EC/AT_0...EC/AT_{n-1}.

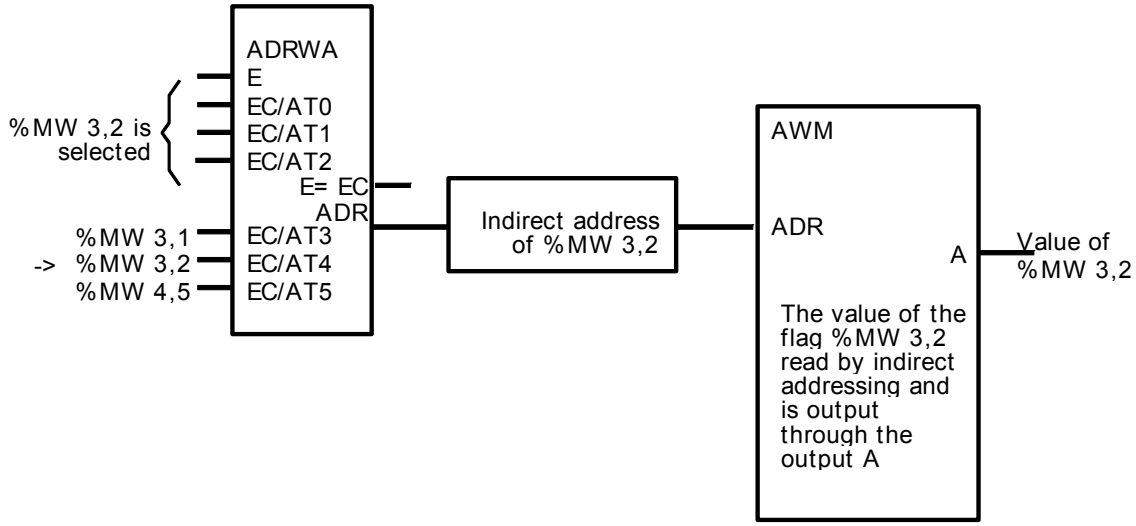
ADR WORD
The operand specified at the ADR output is the indirect address of the operand selected from the output table EC/AT_n ... EC/AT_{2n-1}. The address of the operand selected from the output table is therefore allocating as a value to the operand specified at the ADR output.
If no agreement between the input E and the inputs EC/AT_0...EC/AT_{n-1} is determined during comparison, no value is allocated to the ADR output. In that case, the ADR output is not updated.

Reading/writing operands indirectly
The function blocks AWM or USM use the indirect address generated by the block ADRWA in order to read or write the operand selected with the block ADRWA. Therefore, the block ADRWA *and* the block AWM or USM is needed to read or write operands indirectly. To do this, the operands to be read or written are listed at the

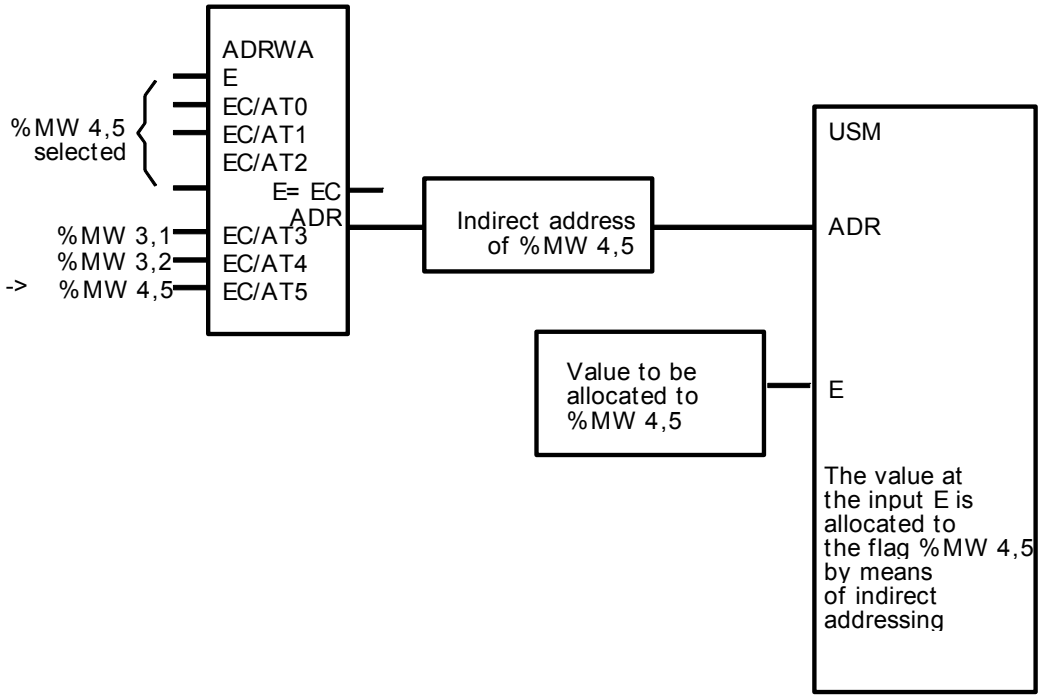
inputs EC/AT_n ... EC/AT_{2n-1} of the ADRWA block and the read or write access is then performed by the AWM or USM block.

Advantages of indirect addressing :
- In suitable applications, the PLC program is simplified substantially, thus reducing the planning effort.
- Access to any number of operands (multiplex function) is achieved with only *one* block (AWM or USM). In this process, the ADRWA block represents a powerful tool with which the operands to be accessed can be selected in a very flexible manner.

Example: Indirect reading of the flag %MW 3,2

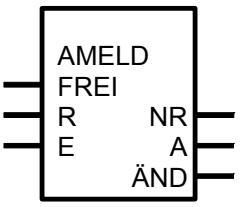


Example: Indirect writing of the flag %MW 4,5



AMELD ANALOG VALUE CHANGE ANNUNCIATOR

FBD



IL

```
!BA     0
AMELD
        FREI
        R
        #n
        E
        NR
        A
        ÄND
```

PARAMETERS

FREI	BINARY	%O, %I, %M, %S, %K	Block enabling
R	BINARY	%O, %I, %M, %S, %K	Reset
#n	DIRECT CONSTANT	#, #H	Number of input values
E	WORD	%IW, %OW, %MW, %KW	Input values, **duplicable**
NR	WORD	%OW, %MW	Number of the input value
A	WORD	%OW, %MW	Current input value
ÄND	BINARY	%O, %M	Change detected

DESCRIPTION

This function block monitors the change of the analog values present at the inputs E (#E0 ... #En-1).

Recognition of a change

Each time the block is processed, the current input values at the inputs E0...En-1 are successively compared against the historical values (input values from the previous processing of the block). If a change is recognized at one of the inputs E0...En-1 :
- this is indicated at the ÄND output
- the number of the input where the change was recognized is output through the NR output
- the changing input value is output through the A output

Each time the block is processed, a change at *one* input only is recognized. If a change is recognized, the inputs following the one where the change was previously discovered are monitored the next time the block is processed.

Initialization of historical values

The first time the block is processed after PLC initialization (FREI = 1) or enabling of processing after it had been disabled (FREI changes from 0 to 1), all current input values are assumed once as historical values and all outputs are set to the value 0. These initialized historical values now represent the starting basis for recognition of changes.

FREI BINARY

Processing of the block is enabled with the FREI input.
FREI = 0 -> Block is not processed
FREI = 1 -> Processing of the block is enabled
If FREI = 0, the outputs of the block are also no longer updated.

R BINARY
The block can be reset with the R input.
R = 0 -> No reset
R = 1 -> Reset of the block
Reset signifies :
- Adoption of the current values at the inputs E0...En-1 as historical values.
- All outputs are set to the value 0

#n DIRECT CONSTANT
ONLY in IL language
The number of values to be monitored at the inputs E0...En-1 is specified at the input #n. The number is specified as a direct constant.
Range for #n : $1 \leq \#n \leq 127$

E WORD
The input E can be duplicated (E0...En-1).
The operands to be monitored for a change are specified at the inputs E.

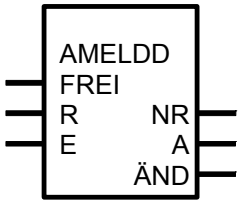
NR WORD
The serial number of the input E0...En-1 where a change has been discovered is output through the output NR.
If *no* output change is discovered during processing of the block, the number of the input changing last is still output through the output NR.
The following affiliations apply :
Change discovered at E0 -> NR = 0
Change discovered at E1 -> NR = 1
Change discovered at En-1 -> NR = n-1

A WORD
If a change is discovered at one of the inputs E0...En-1, the changing input value is allocated to the output A.
If *no* change is discovered at the inputs E0...En-1 during processing of the block, the value of the input changing last is still output through the output A.

ÄND BINARY
The output ÄND indicates whether or not a change has been discovered at the inputs E0...En-1.
ÄND = 0 -> No change discovered
ÄND = 1 -> Change discovered

AMELDD ANALOG VALUE CHANGE ANNUNCIATOR DOUBLE WORD

FBD



IL

```
!BA       0
AMELDD
          FREI
          R
          #n
          E
          NR
          A
          ÄND
```

PARAMETERS

FREI	BINARY	%O, %I, %M, %S, %K	Block enable
R	BINARY	%O, %I, %M, %S, %K	Reset
#n	DIRECT CONSTANT	#, #H	Number of input values
E	DOUBLE WORD	%MD, %KD	Input values; **duplicable**
NR	WORD	%OW, %MW	Number of the input value
A	DOUBLE WORD	%MD	Current input value
ÄND	BINARY	%O, %M	Change detected

DESCRIPTION

This function block monitors the change of the analog values present at the inputs E (#E0 ... #En-1).

Recognition of a change

Each time the block is processed, the current input values at the inputs E0...En-1 are successively compared against the historical values (input values from the previous processing of the block). If a change is recognized at one of the inputs E0...En-1 :

- this is indicated at the ÄND output
- the number of the input where the change was recognized is output through the NR output
- the changing input value is output through the A output

Each time the block is processed, a change at *one* input only is recognized. If a change is recognized, the inputs following the one where the change was previously discovered are monitored the next time the block is processed.

Initialization of historical values

The first time the block is processed after PLC initialization (FREI = 1) or enabling of processing after it had been disabled (FREI changes from 0 to 1), all current input values are assumed once as historical values and all outputs are set to the value 0.

These initialized historical values now represent the starting basis for re–cognition of changes.

FREI BINARY
Processing of the block is enabled with the FREI input.
FREI = 0 -> Block is not processed
FREI = 1 -> Processing of the block is enabled
If FREI = 0, the outputs of the block are also no longer updated.

R BINARY
The block can be reset with the R input.
R = 0 -> No reset
R = 1 -> Reset of the block
Reset signifies :
- Adoption of the current values at the inputs E0...En-1 as historical values.
- All outputs are set to the value 0.

#n DIRECT CONSTANT
ONLY used in IL language
The number of values to be monitored at the inputs E0...En-1 are specified at input #n. Range for #n : $1 \leq \#n \leq \#63$

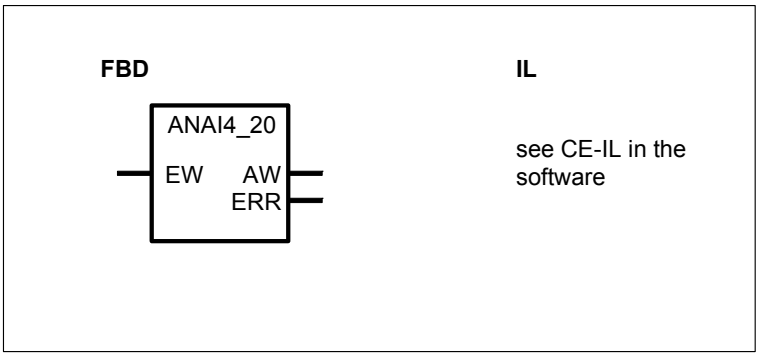
E DOUBLE WORD
The input E can be duplicated (E0...En-1).
The operands to be monitored for a change are specified at the inputs E0...En-1.

NR WORD
The serial number of the input E0...En-1 where a change has been discovered is output through the output NR.
If *no* change is discovered during processing of the block the number of the input changing last is still output through the output NR.
The following affiliations apply :
Change discovered at E0 -> NR = 0
Change discovered at E1 -> NR = 1
Change discovered at En-1 -> NR = n-1

A DOUBLE WORD
If a change is discovered at one of the inputs E0...En-1, the changing input value is allocated to the output A.
If *no* change is discovered at the inputs E0...En-1 during processing of the block, the value of the input changing last is still output through the output A.

ÄND BINARY
The output ÄND indicates whether or not a change has been discovered at the inputs E0...En-1.
ÄND = 0 -> No change discovered
ÄND = 1 -> Change discovered

ANAI4_20 READ ANALOG VALUE 4...20 mA (07 KT 92)



PARAMETERS

EW	WORD	%IW, %MW	Direct analog input
AW	WORD	%MW, %OW	Word variable, to which the result of the evaluation is assigned (0...32760 <-> 4...20 mA)
ERR	BINARY	%M, %O	Output open–circuit monitoring (1 = open circuit)

DESCRIPTION

The connection element ANAI4_20 is intended for reading current values of a 4...20 mA range. It is used with the direct analog inputs IW06,00...IW06,03 of the central unit 07 KT 92. The input range of 4...20 mA is converted into an internal numerical range of 0...32760. There is also an open–circuit monitoring.

The analog inputs, for which the connection element ANAI4_20 is used, have to be configured as current inputs. A separate connection element ANAI4_20 is required for each 4...20 mA current input.

The analog inputs are designed for currents of 0...20 mA. The analog input words IW06,00...IW06,03 generate values of a 0...32760 range corresponding to 0...20 mA.

The connection element ANAI4_20 transforms the numerical range at EW of 6552...32760 (4...20 mA) into a numerical range of 0...32760 at AW.
One step of the output value AW then amounts to 10 (4,883 mA). An input value at EW which is lower than 6552 (4 mA) leads to an output value at AW of 0.

EW WORD Direct analog input
The input EW is assigned to the analog input to be read.

AW WORD Analog value in internal representation
The word output AW provides the analog signal converted into the internal numerical range. In case of an open-circuit, the output is set to 0. The diagram 1 shows the relationship between the input current at the direct analog input and the output value at AW.

ERR BINARY Open–circuit monitoring
The binary output ERR indicates a signal 1 if the connection element ANAI4_20 has detected an open–circuit. If the input current of an analog input of the 07 KT 92 is lower than 3.6 mA, the open–circuit monitoring assumes an open–circuit (see diagram 2). AW is set to 0, if an open–circuit is detected.

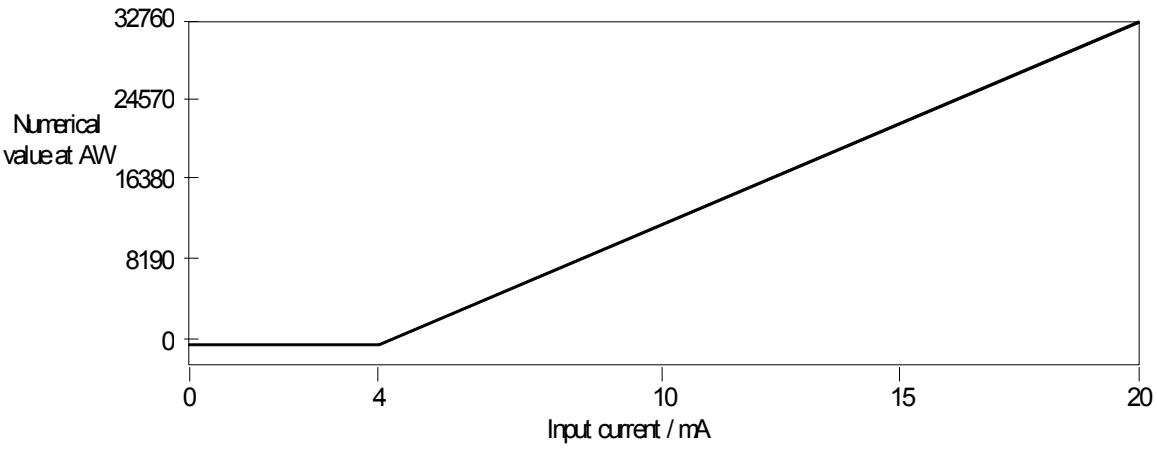


Diagram 1: Relationship between the input current at the direct analog input and the output value AW

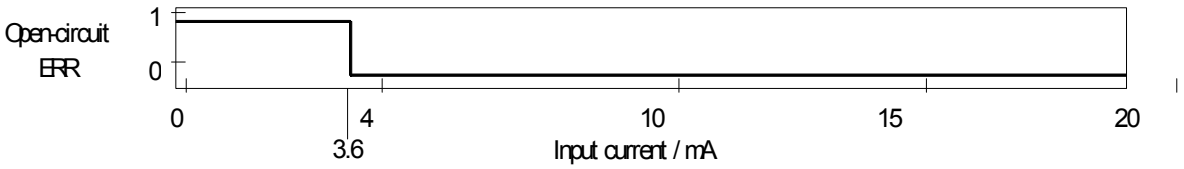
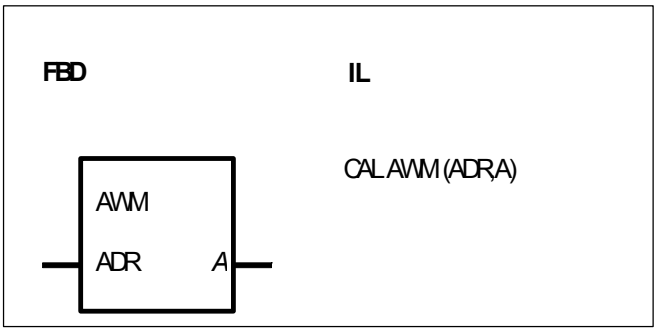


Diagram 2: The output ERR indicates an open-circuit signal if the input current is lower than 3.6 mA

AWMSELECTION MULTIPLEXER



PARAMETERS

ADR	WORD	%IW, %OW, %MW, %KW	Indirect address of the operand to be read
A	WORD	%OW, %MW	Value of the operand read

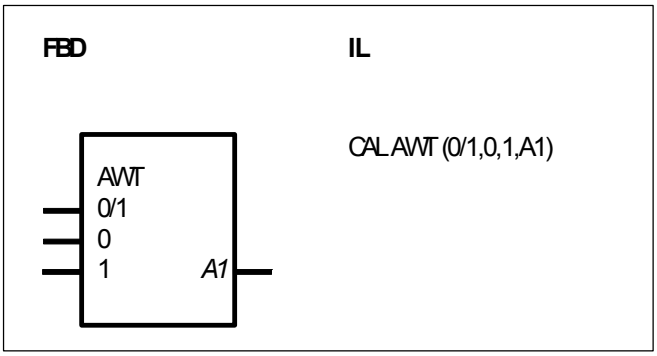
DESCRIPTION

This function block reads the value of an operand, using the method of indirect addressing. The value read is allocated to the output A

Note : The AWM block can only be used meaningfully in conjunction with the ADRWA block.
The value of the operand at the input ADR is interpreted as an address of the operand to be read (indirect addressing). Therefore, the operand at the input ADR represents an indirect address together with its value. This indirect address is generated by the ADRWA function block.

Note : An explanation of the indirect addressing method and the possibilities of using the AWM function block are described in the section dealing with the ADRWA function block.

AWT SELECTION GATE, WORD



PARAMETERS

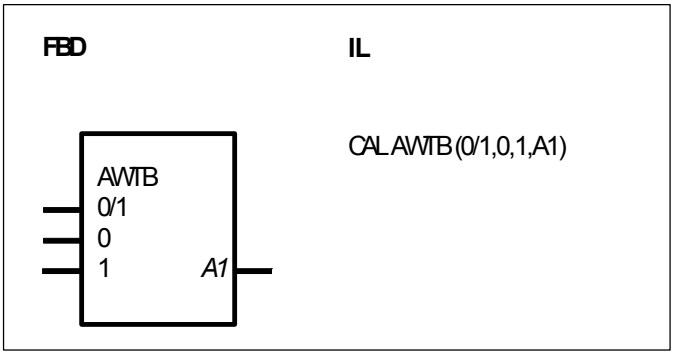
0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
0	WORD	%IW, %MW, %OW, %KW	Word input for 0/1 = 0
1	WORD	%IW, %MW, %OW, %KW	Word input for 0/1 = 1
A1	WORD	%OW, %MW	Word output

DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the word operand at the input 0 to the word operand at the output A1.

A 1 signal at the binary input 0/1 allocates the value of the word operand at the input 1 to the word operand at the output A1.

AWTB BINARY SELECTION GATE



PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
0	BINARY	%I, %M, %O, %S, %K	Binary input for 0/1 = 0
1	BINARY	%I, %M, %O, %S, %K	Binary input for 0/1 = 1
A1	BINARY	%O, %M	Binary output

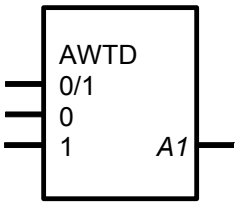
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the status of the operand at the input 0 to the operand at the output A1.

A 1 signal at the binary input 0/1 allocates the status of the operand at the input 1 to the operand at the output A1.

AWTD SELECTION GATE, DOUBLE WORD

FBD



IL

```
CAL AWTD (0/1,0,1,A1)
```

PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
0	DOUBLE WORD	%MD, %KD	Double word input for 0/1 = 0
1	DOUBLE WORD	%MD, %KD	Double word input for 0/1 = 1
A1	DOUBLE WORD	%MD	Double word output

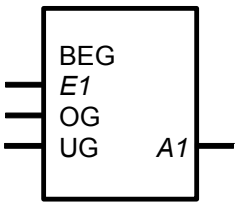
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the double word operand at the input 0 to the double word operand at the output A1.

A 1 signal at the binary input 0/1 allocates the value of the double word operand at the input 1 to the double word operand at the output A1.

BEG LIMITER

FBD



IL

```
CAL BEG (E1,OG,UG,A1)
```

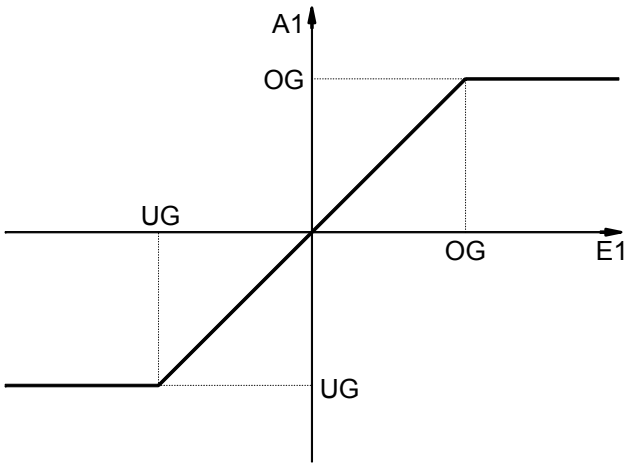
PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input value
OG	WORD	%IW, %MW, %OW, %KW	High limit
UG	WORD	%IW, %MW, %OW, %KW	Low limit
A1	WORD	%OW, %MW	Limited value

DESCRIPTION

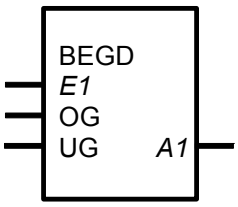
The value of the operand at the input E1 is limited to the range between the high and low limits.
The high limit is specified by the operand at the OG input and the low limit is specified by the one at the UG input.

The following applies : A1 = UG for E1 $\leq$ UG
A1 = E1 for UG $\leq$ E1 $\leq$ OG
A1 = OG for E1 $\geq$ OG



BEGD LIMITER, DOUBLE WORD

FBD



IL

```
CAL BEGD (E1,OG,UG,A1)
```

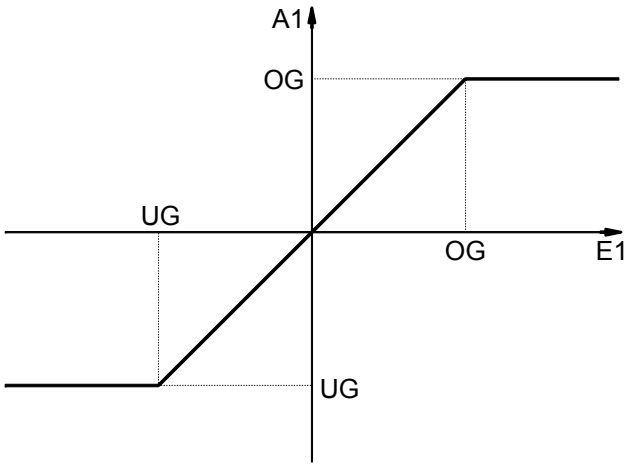
PARAMETERS

E1	DOUBLE WORD	%MD,	Input value
OG	DOUBLE WORD	%MD, %KD	High limit
UG	DOUBLE WORD	%MD, %KD	Low limit
A1	DOUBLE WORD	%MD	Limited value

DESCRIPTION

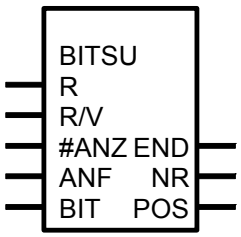
The value of the operand at the input E1 is limited to the range between the high and low limits.
The high limit is specified by the operand at the OG input and the low limit is specified by the one at the UG input.

The following applies : A1 = UG for E1 $\leq$ UG
A1 = E1 for UG $\leq$ E1 $\leq$ OG
A1 = OG for E1 $\geq$ OG



BITSU BIT SEARCHER

FBD



IL

```
CAL BITSU (R,R/V,#ANZ,ANF,BIT,END,NR,POS)
```

PARAMETERS

R	BINARY	%O, %I, %M, %S, %K	Reset of the block and new search
R/V	BINARY	%O, %I, %M, %S, %K	Search direction, up/down
#ANZ	DIRECT CONSTANT	#, #H	Field length, number of word operands
ANF	WORD	%IW, %OW, %MW, %KW	Field start, 1st word operand
BIT	WORD	%IW, %OW, %MW, %KW	Number of set bits to be skipped
END	BINARY	%O, %M	Field end reached
NR	WORD	%OW, %MW	Number of the word operand containing the set bit
POS	WORD	%OW, %MW	Bit position within the word operand

DESCRIPTION

This function block searches through a bit field for a set bit. The bit field consists of successive word operands. If a set bit is found, this is indicated at the block's outputs.

The first word operand in the bit field is specified at the input ANF. After the start of the PLC program (1st program cycle), the block sets all outputs to the value 0 and immediately begins searching for set bits. Therefore, the search is already active in the first program cycle. If the block finds a set bit, its position is indicated at the block's outputs and the search is ended. The next time the block is called in the next program cycle, it continues the search, doing so at the bit position directly following the bit found last. If the end of the bit field is reached during the search, this is indicated at the output END and a new search can be started again by means of a reset signal. The new search again begins at the start of the bit field.

R BINARY

The input R serves to reset the block and start a new search from the beginning of the bit field.
R = 0 -> No reset
R = 1 -> Reset is triggered

A 1 signal at the input R resets all of the block's outputs to the value 0 *and* results in immediate starting of a new search from the beginning of the bit field. Therefore, the new search begins in the same processing cycle in which the block's outputs are reset. The search within the reset cycle ends either at the first set bit or, if no bit is set, at the end of the field. The next time the block is called, the reset input must be set to 0 so that the block will be able to continue searching through the bit field for set bits. The reset input is dominant with respect to the other inputs.

R/V BINARY
The searching direction is specified at the input R/V.
R/V = 0 -> Search down : The *start* of the bit field within the meaning of the search is identical with the *physical start* of the bit field.
R/V = 1 -> Search up : The *start* of the bit field within the meaning of the search is identical with the *physical end* of the bit field.

#ANZ DIRECT CONSTANT
The number of word operands in which the bit field consists is specified at the input #ANZ. This is specified as a direct constant.

ANF WORD
The word operand with which the bit field physically begins is specified at the input ANF. The entire bit field consists in the operand at the input ANF and in the subsequent operands corresponding to the operand numbering. The total number of word operands is specified at the input #ANZ.

BIT WORD
The way in which the function block is to indicate the bits set in the bit field is specified at the input BIT.
BIT = 1 -> *Each* set bit in the bit field is indicated at the outputs NR and POS.
BIT = 2 -> The *first* set bit in the bit field is indicated and then only every *second* set bit of the other set bits in the bit field is indicated.
BIT = 3 -> The *first* set bit in the bit field is indicated and then only every *third* set bit of the other set bits in the bit field is indicated.
BIT = n -> The *first* set bit in the bit field is indicated and then only every *n-th* set bit of the other set bits in the bit field is indicated.

- Special function : BIT = 0
- If the block finds a set bit, in the next program cycle the search is *not automatically* continued at the subsequent bit position. The search is interrupted at this point until the bit found has assumed the significance 0. During interruption of the search, the position of the bit found last continues to be indicated at the outputs. If the bit found last assumes the value 0, the search is continued at the next bit position. If a further bit is set in the selected search direction, this bit's position is indicated and the search is interrupted again. If no more bits are set in the search direction, the position of the bit found last is indicated. The output END is set to 1.
- If no bits are set after a reset, the block runs through the bit field up to its end and stops there. The output END is set to one. The outputs NR and POS are set to 0.
- If bits are set after a reset, the block indicates the first set bit in the search direction and stops at this bit.

END BINARY
Whether or not the end of the bit field has been reached during the search is indicated at the END output.
END = 0 -> End of the bit field not reached
END = 1 -> End of the bit field reached
Search down : The end of the bit field is defined by bit position 15 in the *last* word operand of the physical bit field.
Search up : The end of the bit field is defined by bit position 15 in the *first* word operand of the physical bit field.
If the bit is set at the last bit position of the field and it is found and indicated during a search, the output END is not yet set to the value 1. This is not done until the next processing cycle of the block. Therefore, the prerequisite for setting the output END is that *no* set bit has been found during the current search.

NR, POS WORD
If the block finds a set bit in the planned bit field, its position is indicated at the outputs NR and POS.
Meanings :
NR : Current number of the word operand in which the set bit has been found.
POS : Position of the set bit within the word operand.

Current number of the word operand
NR = 0 -> 1st word operand of the bit field
NR = 1 -> 2nd word operand of the bit field
. . .
. . .
NR = n-1 -> nth word operand of the bit field

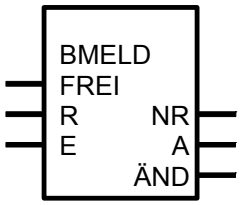
The word operands are numbered in the sense of the search.
Therefore, the following applies to down searching :
First word operand -> first word operand of the physical bit field
Last word operand -> last word operand of the physical bit field

Therefore, the following applies to up searching :
First word operand -> last word operand of the physical bit field
Last word operand -> first word operand of the physical bit field

Position of the set bit within the word operand
Numbering within a word operand is from 0...15. At the same time, position 0 corresponds to the least significant bit and position 15 to the most significant bit within the word operand.
If the end of the field is reached during a search without a set bit having been found, the position of the bit found last continues to be output through the output NR and the output POS. This takes place until a new search is compelled from the start of the bit field by a reset at the input R.
If no bits are set at all within the entire bit field, the outputs assume the following values at the end of the first search : END = 1
NR = 0
POS = 0
This state can be terminated again by a 1 signal at the input R (reset).

BMELD BINARY VALUE CHANGE ANNUNCIATOR

FBD



IL

```
!BA     0
BMELD
        FREI
        R
        #n
        E
        NR
        A
        ÄND
```

PARAMETERS

FREI	BINARY	%O, %I, %M, %S, %K	Block enable
R	BINARY	%O, %I, %M, %S, %K	Reset
#n	DIRECT CONSTANT	#, #H	Number of input values
E	BINARY	%O, %I, %M, %S, %K	Input values; **duplicable**
NR	WORD	%OW, %MW	Number of the input value
A	BINARY	%O, %M	Current input value
ÄND	BINARY	%O, %M	Change detected

DESCRIPTION

This function block monitors the change of the analog values present at the inputs E (#E0 ... #En-1).

Detection of a change

Each time the block is processed, the current input values at the inputs E0...En-1 are compared to the historical values (input values from the previous time the block was processed). If a change is detected at one of the inputs E0...En-1 :

- this is signalled at the output ÄND
- the number of the input at which the change was discovered is output through the output NR
- the input value that has changed is output through the output A.

Each time the block is processed, a change at only *one* input is recognized. When a change is recognized, the inputs following the input where the change was previously discovered are monitored the next time the block is processed.

Initialization of historical values

The first time the block is processed after PLC initialization (Frei = 1) or processing is enabled after it had been disabled (FREI changes from 0 to 1), all current input values are taken over once as historical values and all outputs are set to the value 0. These initialized historical values now represent the starting basis for recognition of changes.

FREI BINARY

Processing of the block is enabled with the FREI input.
FREI = 0 -> Block is not processed
FREI = 1 -> Processing of the block is enabled
If FREI = 0, the block's outputs are also no longer updated.

R BINARY
The block can be reset with the R input.
R = 0 -> No reset
R = 1 -> Reset of the block
Reset signifies :
- Adoption of the current values at the inputs E0...En-1 as historical values.
- All outputs are set to the value 0.

#n DIRECT CONSTANT
ONLY used in IL language
The number of values to be monitored at the inputs E0...En-1 are specified at the #n input. This is specified as a direct constant.
Range for #n : $\leq$ #n $\leq$ 127

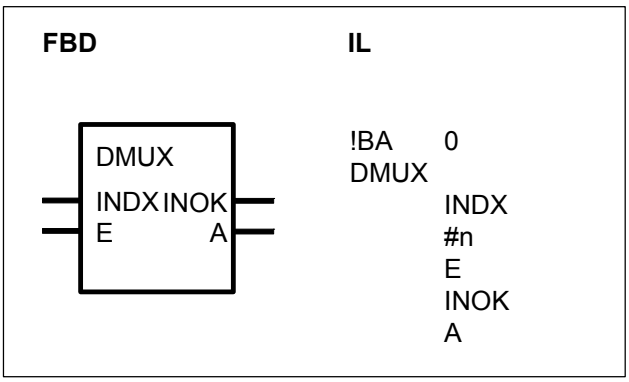
E BINARY
The input E is capable of duplication (E0...En-1).
The operands to be monitored for a change are specified at the inputs E0...En-1.

NR WORD
The serial number of the input E0...En-1 where a change has been discovered is output through the output NR.
If *no* change is discovered during processing of the block, the number of the input that changed last continues to be output through the output NR.
The following affiliations apply :
Change discovered at E0 -> NR = 0
Change discovered at E1 -> NR = 1
Change discovered at En-1 -> NR = n-1

A BINARY
If a change is discovered at one of the inputs E0...En-1, the input value that has changed is allocated to the output A.
If *no* change is discovered at the inputs E0...En-1 during processing of the block, the value of the input that has changed last continues to be output through the output A.

ÄND BINARY
The output ÄND indicates whether or not a change has been discovered at the inputs E0...En-1.
ÄND = 0 -> No change discovered
ÄND = 1 -> Change discovered

DMUX DEMULTIPLEXER



PARAMETERS

INDX	WORD	%IW, %OW, %MW, %KW	Index input
#n	DIRECT CONSTANT	#, #H	Quantity n of word inputs E0 ... En-1
E	WORD	%IW, %OW, %MW, %KW	Word input, **duplicable**
INOK	BINARY	%O, %M	Range monitoring of the index input
A	WORD	%OW, %MW	Word output to which one of the inputs E0...E_{n-1} is switched through.

DESCRIPTION

This function block connects one of the inputs E0...En-1 to the output A depending on the input INDX.
The value at the input INDX is monitored for validity.
The output A is set to 0 if the word input INDX is not within the valid range.

Relationship between E0...En-1, INDX and A :
The input INDX is used to define which of the inputs E0...En-1 is connected to the output A.

The following applies :

INDX = 1 : E0 -> A
INDX = 2 : E1 -> A
INDX = 3 : E2 -> A
: : :
INDX = n : E_{n-1} -> A

where : $1 \leq INDX \leq n \leq 32767$ (theoretically)

The output A is set to 0 if the input INDX = 0.

INDX WORD
Index input for selection of one of the inputs E0...En-1.
The following applies : $1 \leq INDX \leq n$ (Number of inputs E0...En-1)
Note : INDX = 0 -> Initialization of the output A with 0.

#n DIRECT CONSTANT
Only used in IL.
Quantity n of word inputs E0...En-1

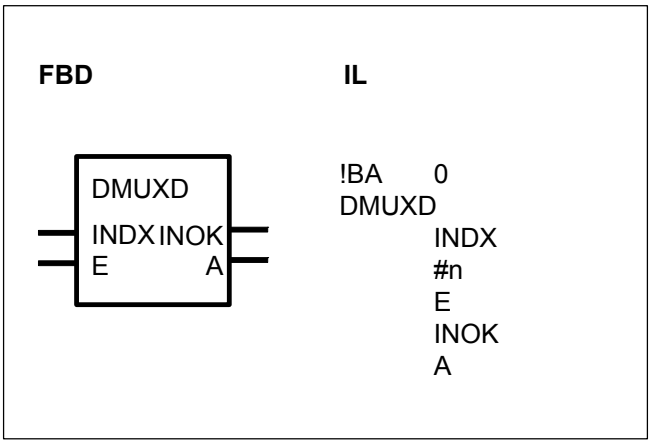
E WORD
Input E capable of duplication (E0 ... E_{n-1})
One of the inputs E0 ... E_{n-1} is connected to the output A.

INOK BINARY
Range monitoring for the input INDX
The output INOK indicates whether or not the input INDX is within the valid range.
The outputs INOK and A are set to 0 if the word input INDX is not within the valid range.

The following applies : INOK = 1 -> INDX in the valid range
INOK = 0 -> INDX in the invalid range -> A = 0
Valid range for the index : 1 ≤ INDX ≤ n

A WORD
Output to which one of the inputs E0...En-1 is switched through.
By means of the input INDX, one of the inputs E0...En-1 is selected and its value is allocated to the output A.
The output A is set to 0 if the word input INDX is not within the valid range.

DMUXD DEMULTIPLEXER DOUBLE WORD



PARAMETERS

INDX	WORD	%IW, %OW, %MW, %KW	Index input
#n	DIRECT CONSTANT	#, #H	Quantity n of the double word inputs E0 ... E_{n-1}
E	DOUBLE WORD	%MD, %KD	Double word inputs E0 ... E_{n-1}; **capable of duplication**.
NOK	BINARY	%O, %M	Range monitoring of the index input
A	DOUBLE	%MD	Double word output to which

WORD	one of the inputs E0 ... E_{n-1} is switched through.

DESCRIPTION

This function block connects one of the inputs E0...En-1 to the output A depending on the input INDX.
The value at the input INDX is checked for validity. The output A is set to 0 if the word input INDX is not within the valid range.

Relationship between E0 ... E_{n-1}, INDX and A :
The input INDX is used to define which of the inputs E0 ... E_{n-1} is connected to the output A.

The following applies :

INDX = 1 : E0 -> A
INDX = 2 : E1 -> A
INDX = 3 : E2 -> A
: : : :
INDX = n : E_{n-1} -> A

where : $1 \leq INDX \leq n \leq 32767$ (theoretically)

Note : The output A is is set to 0 if the input INDX = 0.

INDX WORD
Index input for selection of one of the inputs E0 ... E_{n-1}.
The following applies : $1 \leq INDX \leq n$ (number of inputs E0 ... E_{n-1})
Note : INDX = 0 -> Initialization of the output A with 0.

#n DIRECT CONSTANT
Only used in IL.
Quantity n of double word inputs E0 ... E_{n-1}.

E DOUBLE WORD
Input E capable of duplication (E0 ... E_{n-1})
One of the n inputs E0 ... E_{n-1} is connected to the output A.

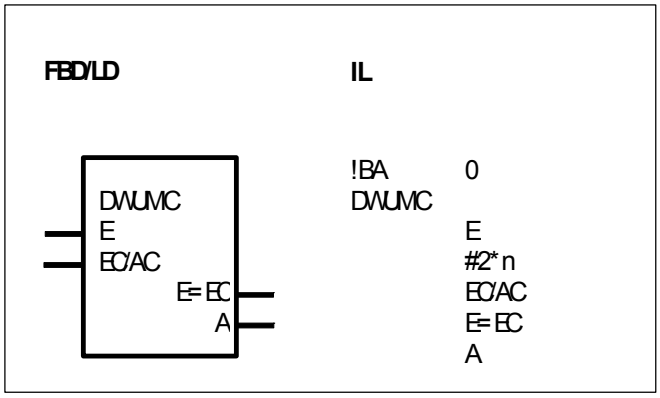
INOK BINARY
Range monitoring for the input INDX
The output INOK indicates whether or not the input INDX is within the valid range. The outputs INOK and A are set to 0 if the word input INDX is not within the valid range.

The following applies : INOK = 1 -> INDX within the valid range
INOK = 0 -> INDX in the invalid range -> A = 0
Valid range for the index : $1 \leq INDX \leq n$

A DOUBLE WORD

Output to which one of the n inputs E0...En-1 is switched through.
By means of the input INDX, one of the inputs E0...En-1 is selected and its value is allocated to the output A.
The output A is set to 0 if the input INDX is not within the valid range.

DWUMC DOUBLE WORD RECODER



PARAMETES

E	DOUBLE WORD	%MD, %KD	Input
#2*n	DIRECT CONSTANT	#, #H, %KW	Quantity n of reference values (multiplied by 2)
EC/AC	DOUBLE WORD	%MD, %KD	Reference value / Output code; can be duplicated
E=EC	BINARY	%O, %M	Coincidence indication
A	DOUBLE WORD	%MD	Output of the output code's value

DESCRIPTION

This function block compares the value of the operand at the input E to the reference values of the operands at the inputs EC/AC. If the input E agrees with at least one of the reference values EC/ACi, the output E=EC is set to 1. The output A receives the value of the output code EC/ACn+i, which is allocated to the reference value EC/ACi found.
An operand for the output code EC/ACn+i is allocated to each reference value at the inputs EC/ACi. Affiliation of EC/ACi to EC/ACn+i is recognizable by the index i.

Note : In **IL** language, the number of inputs EC/AC must be specified as a direct constant at the input #2*n.

E DOUBLE WORD
The operand whose value is to be compared to the values of the n reference values (EC0...ECn-1) is specified at the input E.

#2*n DIRECT CONSTANT (#,#H)
Only used in IL.
The total number (2*n) of the reference values and output codes (EC/AC) is specified

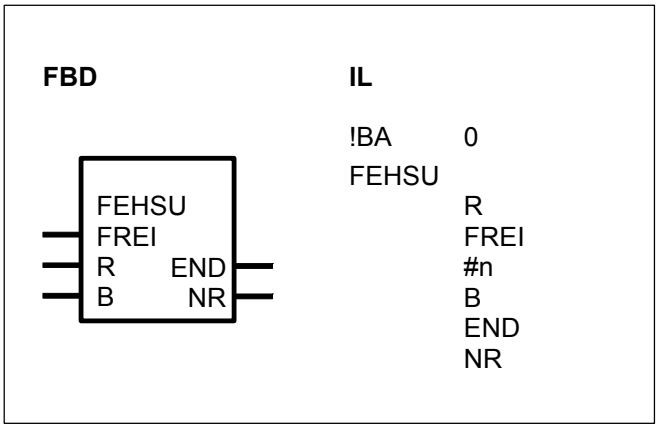
at the input #2*n. This is specified as an indirect constant.

EC/AC DOUBLE WORD
The input EC/AC must be duplicated according to the number of reference values needed. The operands for the reference values are specified at the inputs EC/AC0...EC/ACn-1. The output codes are specified at the inputs EC/ACn...AC/AC2n-1.
The value of the operand at the input E1 is compared to the reference values.
The output code EC/ACn+i is output through the output A if the input E agrees with one of the reference values EC/ACi.
Affiliation between reference values and output codes :
EC/AC0 <-> EC/ACn
EC/AC1 <-> EC/ACn+1
. .
. .
EC/AC_{n-1} <-> EC/ACn+(n-1)

E=EC BINARY
Agreement between the operand value of the input E and one of the reference values is signalled at the output E=EC.
The following applies: E=EC = 0 -> No agreement
E=EC = 1 -> Agreement

A DOUBLE WORD
The output code EC/ACn+i is output through the output A if the input E agrees with one of the reference values EC/ACi.
The following applies: A = 0 -> No agreement
A = EC/ACn+i -> Agreement

FEHSU ERROR SEARCHER WITH AUTOMATIC DELETION



PARAMETERS

FREI	BINARY	%I, %O, %M, %S, %K	Enable search
R	BINARY	%I, %O, %M, %S, %K	Reset
#n	DIRECT CONSTANT	#, #H	Number of binary variables
B	BINARY	%I, %O, %M	List of binary variables;

			capable of duplication
END	BINARY	%O, %M	List end reached
NR	WORD	%OW, %MW	List number of the variables found

DESCRIPTION

This function block searches through a list of binary variables (%I, %O, %M, %S) for set binary variables. If a set binary variable is found, its serial number referred to the list is indicated at the input NR. The set binary variable at the input is deleted.

FREI BINARY
The search for a set binary variable is enabled with the FREI input.
FREI = 0 Search is not enabled
FREI = 1 Search is enabled.

The search is continued until
- a set binary variable is found or
- the end of the variable list is reached.

If a binary variable with the value 1 is found in the list, the search is ended, the variable is deleted and its list number is output through the output NR.
The search is continued again when the block is called again (in the next program cycle). The binary variable directly following the one found is now examined first.
The search ends at the end of the list if no further set variable is found.
The value 1 is then allocated to the END output and the number of the variable found last during the search is indicated at the output NR.
If no set variable is found during a search from the start of the list, the search is automatically repeated from the start of the list when the block is called again. This takes place until a set variable is found.

R BINARY
By means of the input R, the function block is set to a defined state (reset) in order to be able to start a search from the start of the variable list.
R = 0 -> No reset of the block
R = 1 -> Reset of the block for preparation of a search from the start of the list

Note : Even when the block is called for the very first time after starting the PLC program, a reset *must* first of all be executed in order to create the correct marginal conditions for the first search.
A reset of the block results in the following :
- Output END = 0
- Output NR = 0
- The next search takes place from the start of the list.

Priority : The input R has higher priority than the input FREI, i. e. no search is executed as long as there is a 1 signal at the input R.

#n DIRECT CONSTANT
ONLY used in IL language
The number of binary variables planned at the inputs B0...Bn-1 is specified at the

input #n. This is specified as a direct constant.

B BINARY
The input B can be duplicated (B0...Bn-1).
The list of binary variables to be examined is specified at the inputs B0...Bn-1. Owing to the reset mechanism, step operands (%S) and indirect constants (%K) are not allowed.

END BINARY
END = 0 -> List end not reached
END = 1 -> List end reached *without* a set binary variable having been found in the current search

NR WORD
The list number of the binary variable found last is output through the NR output.
The following affiliations apply :

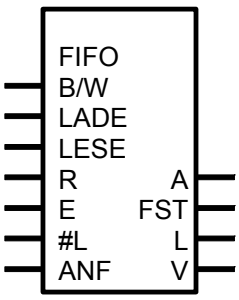
Variable at the input	List number
B0	1
B1	2
.	.
.	.
.	.
B_{n-1}	n

If the end of the list is reached during the search without a set variable having been found, the number of the variable last found continues to be output through the output NR.

Searching from the start of the list without a variable being found :
If no set binary variable is found during a search from the start of the list, the state 1 is assigned to the output END when the end of the list is reached and the value 0 is assigned to the output NR. When the block is called again in the next program cycle, the search automatically takes place again from the start of the list. Automatic searching from the start of the list continues until a set binary variable is found once.

FIFO STACK, FIRST IN/FIRST OUT

FBD



IL

```
CAL FIFO (R,LADE,LESE,E,B/W,#L,ANF,A,FST,L,V)
```

PARAMETERS

B/W	BINARY	%I, %O, %M, %S, %K	Binary/word data
LADE	BINARY	%I, %O, %M, %S, %K	Load FIFO
LESE	BINARY	%I, %O, %M, %S, %K	Read FIFO
R	BINARY	%I, %O, %M, %S, %K	Reset FIFO, 0->1 edge
E	BINARY, WORD	%I, %O, %M, %S, %K %IW, %OW, %MW, %KW	Input of data in the FIFO
#L	DIRECT CONSTANT	#, #H	Number of memory locations (bytes)
ANF	BINARY WORD	%I, %O, %M, %K %IW, %OW, %MW, %KW	Start of the FIFO in the flag area
A	BINARY, WORD	%O, %M %OW, %MW	Output of data from the FIFO
FST	WORD	%OW, %MW	Filling level of the FIFO
L	BINARY	%O, %M	FIFO empty
V	BINARY	%O, %M	FIFO full

DESCRIPTION

This function block realizes a stack for binary or word data from which the data written in first is again read out first (first in/first out).

B/W BINARY

The input BW serves to preselect binary or word processing.
B/W = 0 -> word processing
B/W = 1 -> binary processing
Binary or word processing is defined once and must not be changed during operation of the FIFO.

LADE BINARY

The value present at the input E is transferred to the next free position of the FIFO by means of a 1 signal at the LADE input. If the FIFO is “full” and a “load” signal is present, the new value will *not* be read in. A new value can only be read if a value has been read out first. It is then transferred to the first position of the FIFO.

LESE BINARY
A 1 signal at the input LESE results in output of the current FIFO value through the output A. If the FIFO is “empty”, a pending read order is ignored and the value 0 is output through the output A. The output A is always set to 0 if there is no read order.

LADE and **LESE**
If load and read orders are present simultaneously, the value to be loaded is forwarded directly from the input E to the output A. Then the FIFO is empty.
If the FIFO is not empty, the current FIFO value is output and the new load information is transferred to it.
The FIFO does not change its filling level in this mode. The value output through the output FST remains constant. If the FIFO is empty, the output L remains 1 and the output FST is set to 0.

R BINARY
A 0->1 edge at the input R results in the reset of the block. Therefore, values read in before are no longer available.
The output L assumes the value 1 and the outputs FST and A assume the value 0.

R and **LADE**
If R and LADE signals are present simultaneously, the reset is performed first and then directly afterwards the load operation. Therefore, the FIFO is reset and the new value to be read in is then immediately stored in the FIFO as the first value.

R and **LADE** and **LESE**
The value at the input E is forwarded directly to the output A. The output L is set permanently to 1 and the output FST is set permanently to 0.

E BINARY/WORD
The value to be transferred into the FIFO is specified at the input E.

#L DIRECT CONSTANT
The number of required memory locations (bytes) of the FIFO is specified at the input A. This quantity is specified as a direct constant and results from the following formula :

BINARY data : #L = Number of binary values to be stored
WORD data : #L = 2 * Quantity of word values to be stored
E.g. : BINARY data : 3 values -> #L = 3
WORD data : 3 values -> #L = 6

The FIFO length parameter is subjected to a plausibility check for the value 0 and also, in word processing mode, for an odd byte parameter. If the parameter specified at the input #L is incorrect, the FIFO will assume the initial state (as after R).

ANF BINARY/WORD
The FIFO memory start address is specified as a binary or word flag at the input ANF. The FIFO begins with the specified flag.

A BINARY/WORD
When the FIFO is read, the current value is available at the output A. If *no* read order

is available, the value 0 is output.

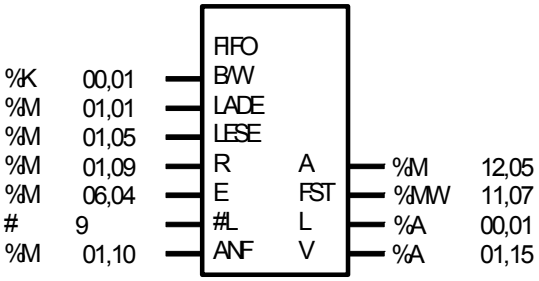
FST WORD
The output FST indicates the current filling level of the FIFO at any time. The filling level is the number of binary or word values stored in the FIFO.

L BINARY
The output L indicates whether or not the FIFO is empty.
L = 0 -> FIFO is not empty
L = 1 -> FIFO is empty

V BINARY
The output V indicates whether or not the FIFO is full.
V = 0 -> FIFO is not full
V = 1 -> FIFO is full
No more values can be read in if the FIFO is full. A value can only be read in once a value has been read out. This then takes place as from the start of the FIFO.
Example

FIFO for 9 binary data, stored as of %M 01,10

FBD

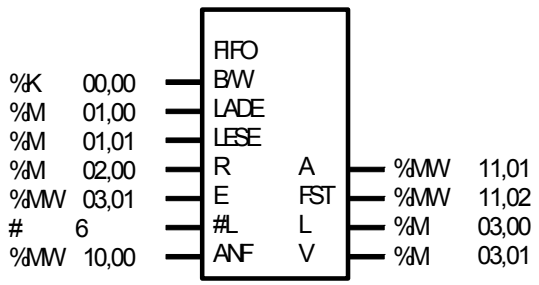


IL

```
CAL     LIFO    (%M01,09, %M01,01, %M01,05, %M06,04, %K00,01, #9, %M01,10,
                 %M12,05, %MW11,07, %A00,01, %A01,15)
```

FIFO for 3 word data, stored as of %MW 10,00

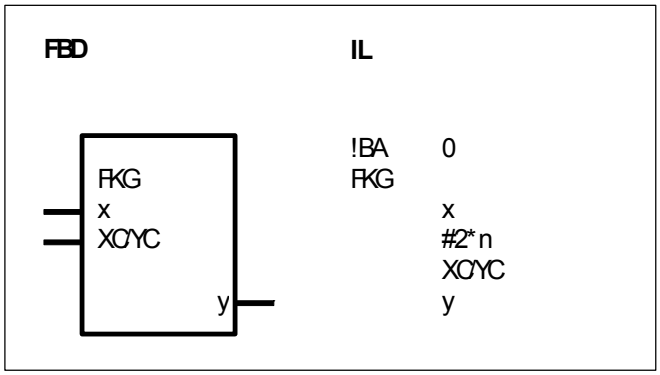
FBD



IL

```
CAL     LIFO    (%M02,00, %M01,00, %M01,01, %MW03,01, %K00,00, #6, %MW10,00,
                 %MW11,01, %MW11,02, %M03,00, %M03,01)
```

FKG FUNCTION GENERATOR



PARAMETERS

x	WORD	%IW, %OW, %MW, %KW	Input for the x value of the polygon
#2*n	DIRECT CONSTANT	#, #H	n : number of interpolation points
XC/YC	WORD	%IW, %OW, %MW, %KW	Input for x/y-axis values of the interpolation points; **duplicable**
y	WORD	%OW, %MW	Output for the y value of the polygon

DESCRIPTION

In an x/y coordinate system, a polygon is defined by n coordinate points X0/Y0...Xn-1/Yn-1. For each value at the input x, the function block outputs the affiliated y value of the polygon through the output y.

The following applies to the x-axis coordinates :

$X/Y0 < X/Y1 < X/Y2 ... < X/Y_{n-1}$

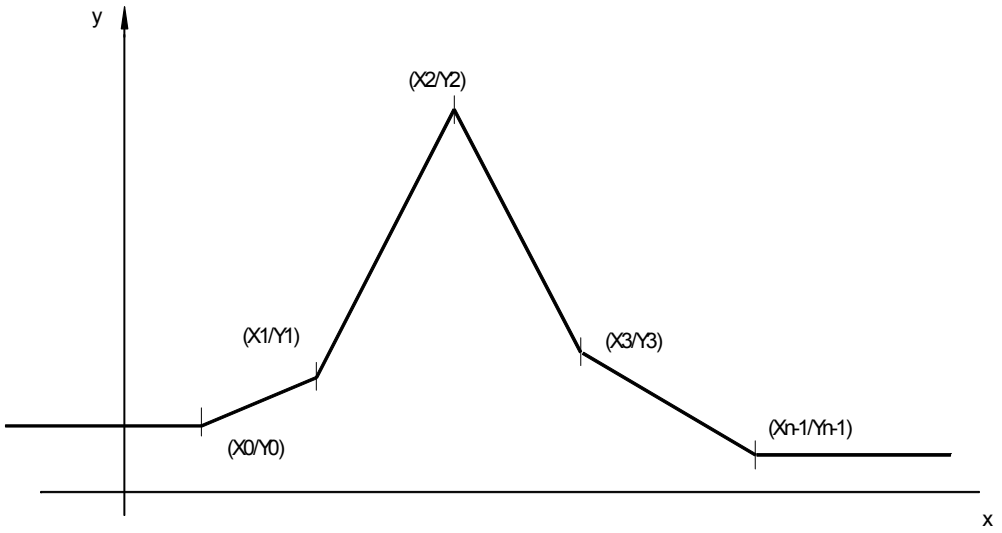
$2 \leq n \leq 32767$

This function block make a linear interpolation between the interpolation points. The resulting polygon representing the relationship between x and y is :

$$y = \frac{(x - X_{i-1}) * (Y_i - Y_{i-1})}{X_i - X_{i-1}} + Y_{i-1}$$

Note : The result of division is always rounded down, i.e. any division remainder is ignored.

The following applies to the range outside of the interpolation points :
$y = Y0$ for $x \leq X0$
$y = Y_{n-1}$ for $x \geq X_{n-1}$



x WORD
The current x coordinate is specified at the input x. The block then defines the y coordinate affiliated by the polygon.

#2*n DIRECT CONSTANT
Only used in IL Language
The number of x/y coordinates needed to define the polygon is specified at the input #2*n. This is specified as a direct constant.

n	=	number of interpolation points
2*n	=	number of x/y coordinates for n interpolation points

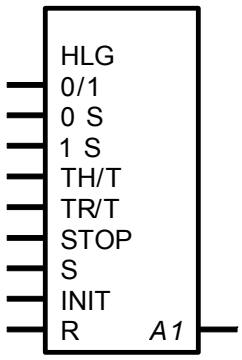
XC/YC WORD
This input has to be duplicated by a multiple of 2 : 2*n
- XC/YC0...XCYCn-1 specify the x coordinates of the n interpolation points.
- XC/YCn...XC/YC2n-1 specify the y coordinates of the n interpolation points.

Affiliation between (Xi,Yi) and XC/YCi :
(Xi,Yi) <-> (XC/YCi,XC/YCn+i)

y WORD
The y coordinate affiliated by the polygon of the specified x coordinate is output through the output y.

HLG RAMP-FUNCTION GENERATOR

FBD



IL

```
CAL HLG (0/1,0 S,1 S,TH/T,TR/T,STOP,S,INIT,R,A1)
```

PARAMETERS

0/1	BINARY	%I, %O, %M, %S, %K	Selection of the setpoint 0 or the setpoint 1
0 S	WORD	%IW, %OW, %MW, %KW	Setpoint 0
1 S	WORD	%IW, %OW, %MW, %KW	Setpoint1
TH/T	WORD	%IW, %OW, %MW, %KW	Start up time scaled to the cycle time
TR/T	WORD	%IW, %OW, %MW, %KW	Return time scaled to the cycle time
STOP	BINARY	%I, %O, %M, %S, %K	Latching of the output at the current value
S	BINARY	%I, %O, %M, %S, %K	Setting of the output to the INIT value
INIT	WORD	%IW, %OW, %MW, %KW	INIT value to which the output can be set
R	BINARY	%I, %O, %M, %S, %K	Reset of the output to the value 0
A1	WORD	%MW, %OW	Output

DESCRIPTION

The ramp-function generator serves to provide ramp-shaped adaption of the current actual value at the output to a specified setpoint.

The value at the output of the HLG is adapted linearly from the current actual value to the specified setpoint with the slope y'. In doing so, the value at the output precisely runs through the amount of the setpoint during the time TH or TR. If the value at the output of the HLG has reached the setpoint, it no longer changes unless a new setpoint is specified.

The slope y' of the ramp results from the specified time TH (start up time) or TR (return time) and the amount of the setpoint :

$$\text{Slope } y' = \frac{\text{Setpoint amount}}{\text{TH or TR}}$$

The slope is - positive if setpoint > actual value
- negative if setpoint < actual value
- 0 if setpoint = 0

Therefore, the specified setpoint has two functions :
- its amount defines the slope of the ramp together with the specified time TH or TR;
- it represents the value to which the current actual value must be adapted in a ramp shape.

The user may specify the start up time TH and the return time TR separately. The direction of the slope is defined on the basis of the setpoint. The direction of the slope then defines whether or not the running time TH or TR is used.
Slope y' positive -> TH, i.e. the ramp runs upwards.
Slope y' negative -> TR, i.e. the ramp runs downwards.

The start up time TH and the return time TR must be scaled to the program cycle time TZ, i.e. the following must be specified at the corresponding inputs of the block :
- Start up time : TH/TZ
- Return time : TR/TZ

The times are specified in milliseconds.
The following applies to the time constants TH or TR : $0 \leq TH \leq 32767$
$0 \leq TR \leq 32767$

Two setpoints can be planned (0 S and 1 S), whereby one of these setpoints is selected by the binary input 0/1 (setpoint selection). The setpoints may assume the following values : $-32767 \leq \text{setpoint} \leq +32767$

At any time, the output of the ramp-function generator can be
- stopped at the current value
- set to an initial value
- reset (output = 0)

The STOP input has the highest priority and the R input has the lowest.

The values at the inputs of the HLG can be altered at any time in the user program. In this way, any (non-linear) adaptation to the setpoint can be realized on the basis of the linear adaptation of the actual value.

Important
Setpoint = 0 means that the slope of the ramp is also 0, i.e. the current actual value does not change. If it is intended to switch from an actual value unequal to 0 to an actual value of 0, a setpoint unequal to 0 must be specified and the output of the ramp-function generator must be limited to 0 by a subsequent limiter. (On interpolation, the rounding transitions are based on calculation of integral numbers only).

0/1 BINARY
One of the two setpoints is selected with the input 0/1.
0/1 = 0 -> setpoint 0 S
0/1 = 1 -> setpoint 1 S

0 S WORD
The setpoint 0 is specified at the input 0 S.

1 S WORD
The setpoint 1 is specified at the input 1 S.

TH/T WORD
The startup time is specified at the input TH/T. At the same time, the start up time TH must be scaled to the cycle time T.

TR/T WORD
The start up time is specified at the input TH/T. At the same time, the start up time TH must be scaled to the cycle time T.

STOP BINARY
The output can be latched to the current value by means of the STOP input.
STOP = 0 -> Output not latched
STOP = 1 -> Output is latched
The STOP input has higher priority than the inputs S and R.

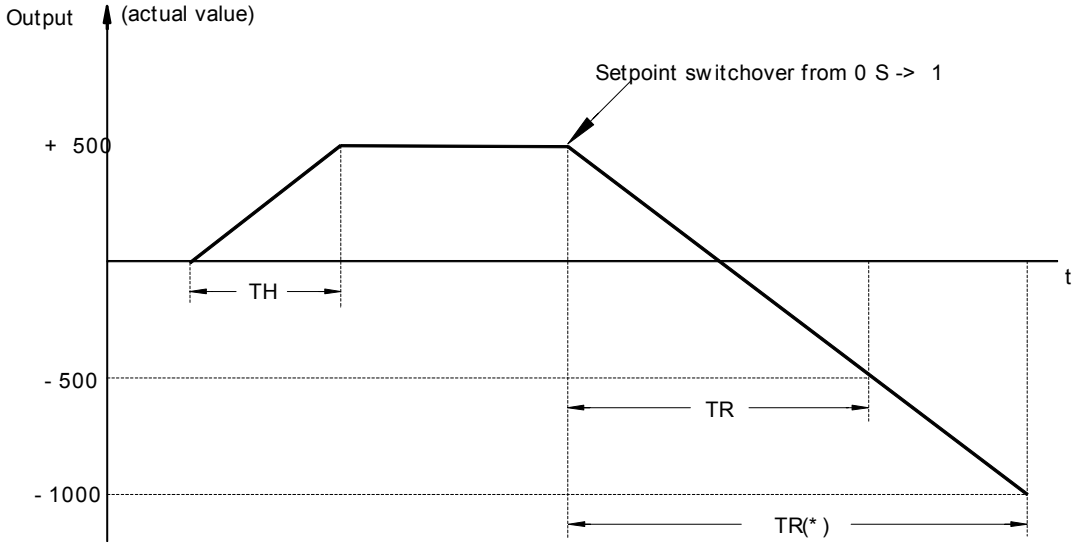
S BINARY
With the input S, the output can be set to the initial value specified at the input INIT.
S = 0 -> Output is not set to initial value.
S = 1 -> Output is set to initial value.

INIT WORD
The initial value to which the output is to be set if required is specified at the input INIT.

R BINARY
The output can be set to the value 0 with the input R.
R = 0 -> Output is not reset
R = 1 -> Output is reset to the value 0.

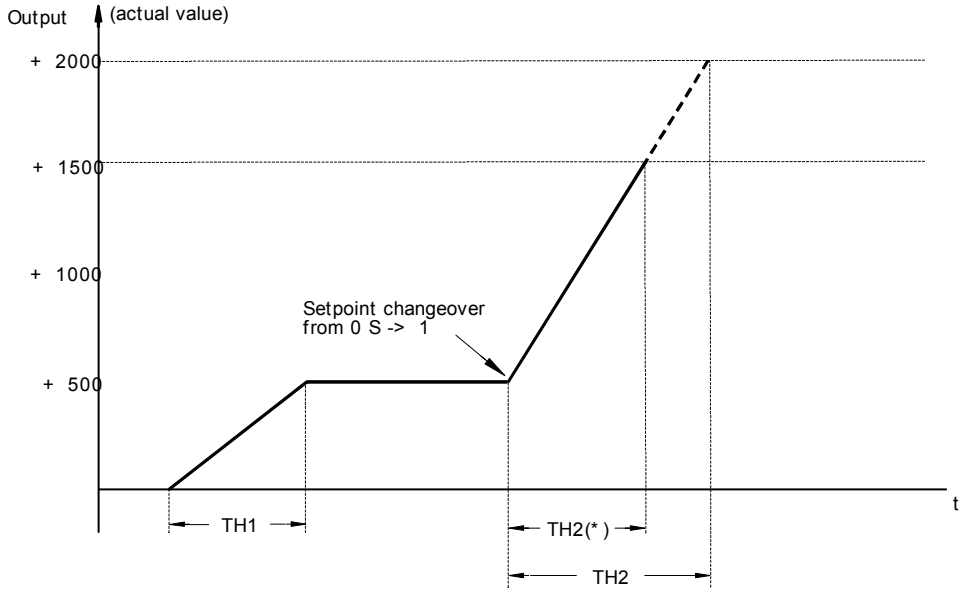
A1 WORD
Output of the block.

Example 1
The actual value is to be changed from 0 to the setpoint + 500 (setpoint amount 500) and then from +500 to the setpoint -1000 (setpoint amount 1000).
The start up time is TH and the return time TR.
0 S : 500
1 S : -1000



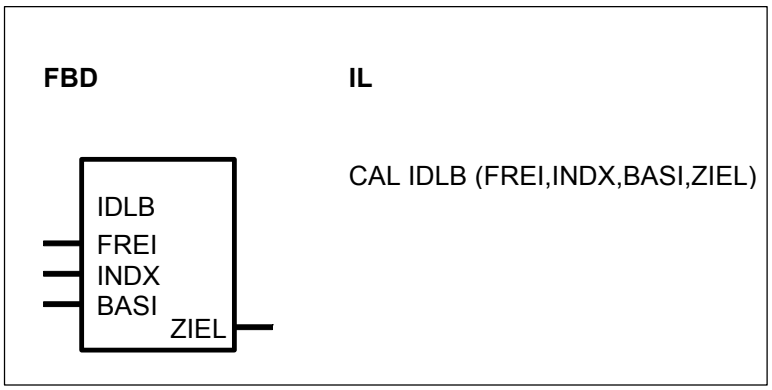
TR(*) is the actual time until the actual value has reached -1000.
During the time TH or TR, the actual value changes by the amount of the applied setpoint.

Example 2
The actual value is to be changed from 0 to the setpoint +500 (setpoint amount 500) and then from +500 to the setpoint 1500 (setpoint amount 1500).
The startup time is TH and the return time TR.
0 S : 500
1 S : 1500



TR(*) is the actual time until the actual value has reached 1500.
During the time TH or TR, the actual value changes by the amount of the applied setpoint.

IDLB READ BINARY VARIABLE, INDEXED



PARAMETERS

FREI	BINARY	%I, %O, %M, %K, %S	Enable for the block
INDX	WORD	%IW, %OW, %MW, %KW	The index and the basic variable result in the source variable
BASI	BINARY	%I, %O, %M, %K	Basic variable
ZIEL	BINARY	%O, %M	Target variable

DESCRIPTION

This function block serves the purpose of indexed reading of binary variables.
The source variable to be read is obtained from indexing the basic variable. The value of the source variable read is allocated to the target variable.

The group and channel numbers of the source flag (source variable) are determined from the basic flag and the index INDX.
The source flag is : M (G_Basis + A) , (K_Basis + B)
where : G_Basis : Group number of the basic flag
K_Basis : Channel number of the basic flag

Formula : $A = \frac{INDX}{16}$ Remainder B

Group No. of the source flag : Group No. of the basic flag + A
Channel No. of the source flag : Channel No. of the basic flag + B

Example :
Basic variable : M 000,00
INDX = 10
-> A = 10 : 16 -> A = 0, Remainder B = 10
->Source variable : M (000+A),(00+B) = M (000+0),(00+10) = M 000,10

Further examples :

Basic variable	INDX	Source variable
M 000,00	0	M 000,00
M 000,00	2	M 000,02
M 000,00	16	M 001,00
M 000,02	18	M 001,04
I 00,00	3	I 00,03
I 01,01	5	I 01,06
O 05,05	6	O 05,11

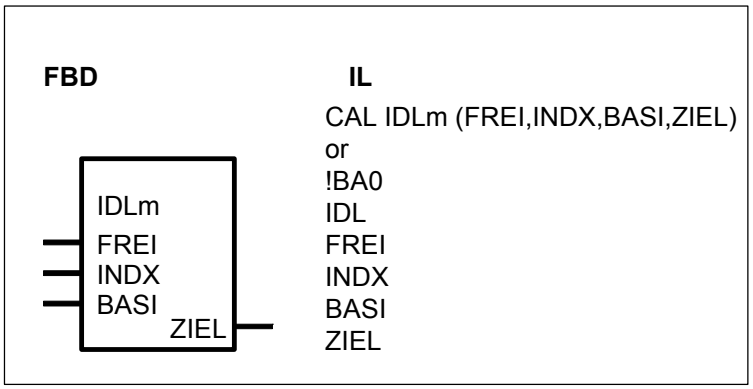
FREI BINARY
Enable block
FREI= 0 -> Block is not processed
FREI= 1 -> The value of the source variable is read and allocated to the target variable ZIEL.

INDX WORD
The index value is specified at the input INDX. The source variable (see above for a calculation) results from the index INDX and the basic variable.
Value range : -16383 ≤ INDX ≤ +16383

BASI BINRAY
The basic variable is specified at the input BASI. The source variable (see above for a calculation) results from the index INDX and the basic variable.

ZIEL BINARY
The target variable is specified at the output ZIEL. The value of the selected source variable is allocated to the target variable ZIEL.

IDLm READ WORD VARIABLE, INDEXED



PARAMETERS

FREI	BINARY	%I, %O, %M, %K, %S	Enable for the block
INDX	WORD	%IW, %OW, %MW, %KW	The index and the basic variable result in the source variable

BASI	WORD	%IW, %OW, %MW, %KW	Basic variable
ZIEL	WORD	%OW, %MW	Target variable

DESCRIPTION

This function block serves the purpose of indexed reading of word variables.
The source variable to be read is obtained from indexing the basic variable. The value of the source variable read is allocated to the target variable.

The group and channel numbers of the source flag (source variable) are determined from the basic flag and the index INDX.
The source flag is : MW (G_Basis + A) , (K_Basis + B)
where : G_Basis : Group number of the basic flag
K_Basis : Channel number of the basic flag

Formula : $A = \frac{INDX}{16}$ Remainder B

Group No. of the source flag : Group No. of the basic flag + A
Channel No. of the source flag : Channel No. of the basic flag + B

Example :
Basic variable : MW 000,00
INDX = 10
-> A = 10 : 16 -> A = 0, Remainder B = 10
->Source variable : MW (000+A),(00+B) = MW(000+0),(00+10) = MW 000,10

Further examples :

Basic variable	INDX	Source variable
MW 000,00	0	MW 000,00
MW 000,00	2	MW 000,02
MW 000,00	16	MW 001,00
MW 000,02	18	MW 001,04
IW 00,00	3	IW 00,03
IW 01,01	5	IW 01,06
OW 05,05	6	OW 05,11

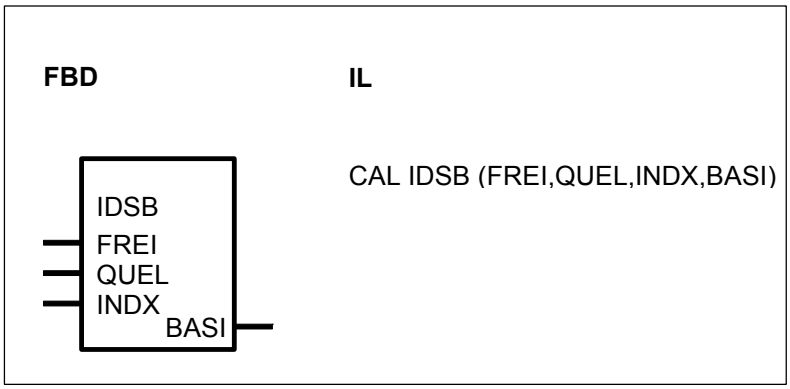
FREI BINARY
Enable block
FREI= 0 -> Block is not processed
FREI= 1 -> The value of the source variable is read and allocated to the target variable ZIEL.

INDX WORD
The index value is specified at the input INDX. The source variable (see above for a calculation) results from the index INDX and the basic variable.
Value range : -16383 ≤ INDX ≤ +16383
If INDX is out of range, the function block is not processed.

BASI WORD
The basic variable is specified at the input BASI. The source variable (see above for a calculation) results from the index INDX and the basic variable.

ZIEL WORD
The target variable is specified at the output ZIEL. The value of the selected source variable is allocated to the target variable ZIEL.

IDSB WRITE BINARY VARIABLE, INDEXED



PARAMETERS

FREI	BINARY	%I, %O, %M, %K, %S	Enable block
QUEL	BINARY	%I, %O, %M, %K	Source variable
INDX	WORD	%IW, %OW, %MW, %KW	The current target variable results from the index and the basic variable
BASI	BINARY	%O, %M	Basic variable

DESCRIPTION

This function block serves the purpose of indexed wri–ting of binary variables.
When the block is enabled, the value of the source variable is read and allocated to the target variable. The target variable is defined by indexing the basic variable.

The group and channel number of the target flag (target variable) are determined on the basis of the basic flag and the index INDX.
The target flag is called : M (G_Basis+A),(K_Basis+B) where :
G_Basis : Group number of the basic flag
K_Basis : Channel number of the basic flag

Formula : $A = \frac{INDX}{16}$ Remainder B

Group No. of the target flag : Group No. of the basic flag + A
Channel No. of the target flag : Channel No. of the basic flag + B

Example :
Basic variable : M 000,00

INDX = 10
-> A = 10 : 16 -> A = 0, Remainder B = 10
-> Target variable : M(000+A), (00+B) = M (000+0),(00+10) = M 000,10

Further examples :

Basic variable	INDX	Target
M 000,00	0	M 000,00
M 000,00	2	M 000,02
M 000,00	16	M 001,00
M 000,02	18	M 001,04
O 00,00	3	O 00,03
O 01,01	5	O 01,06
O 05,05	6	O 05,11

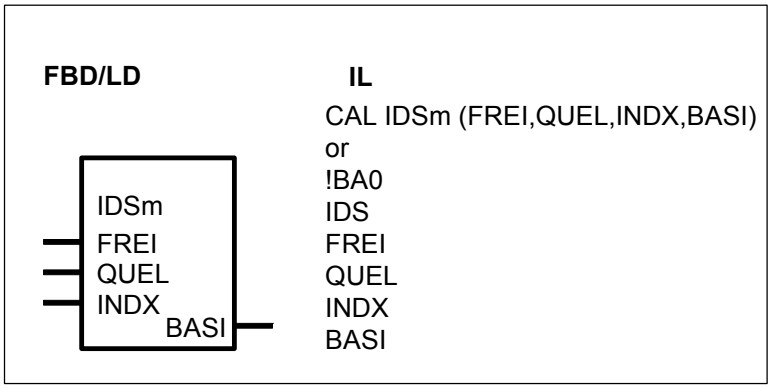
FREI BINARY
Enable block
FREI=0 : -> Block is not processed
FREI=1 : -> The value of the source variable is read and allocated to the target variable ZIEL.

QUEL BINARY
The source variable is specified at the input QUEL. The value of this variable is read and allocated to the target variable.

INDX WORD
The index value is specified at the input INDX. The target variable (see above for a calculation) results from the index INDX and the basic variable.
Value range : -16383 ≤ INDX ≤ +16383

BASI BINARY
The basic variable is specified at the input BASI. The target variable (see above for a calculation) results from the index INDX and the basic variable.

IDSm WRITE WORD VARIABLE, INDEXED



PARAMETERS

FREI	BINARY	%I, %O, %M, %K, %S	Enable block
QUEL	WORD	%IW, %OW, %MW, %KW	Source variable
INDX	WORD	%IW, %OW, %MW, %KW	The current target variable

			results from the index and the basic variable
BASI	WORD	%OW, %MW	Basic variable

DESCRIPTION

This function block serves the purpose of indexed wri–ting of word variables.
When the block is enabled, the value of the source variable is read and allocated to the target variable. The target variable is defined by indexing the basic variable.

The group and channel number of the target flag (target variable) are determined on the basis of the basic flag and the index INDX.
The target flag is called : MW (G_Basis+A),(K_Basis+B) where :
G_Basis : Group number of the basic flag
K_Basis : Channel number of the basic flag

Formula : $A = \frac{INDX}{16}$ Remainder B

Group No. of the target flag : Group No. of the basic flag + A
Channel No. of the target flag : Channel No. of the basic flag + B

Example :
Basic variable : MW 000,00
INDX = 10
-> A = 10 : 16 -> A = 0, Remainder B = 10
-> Target variable : MW(000+A), (00+B) = MW (000+0),(00+10) = MW 000,10

Further examples :

Basic variable	INDX	Target
MW 000,00	0	MW 000,00
MW 000,00	2	MW 000,02
MW 000,00	16	MW 001,00
MW 000,02	18	MW 001,04
OW 00,00	3	OW 00,03
OW 01,01	5	OW 01,06
OW 05,05	6	OW 05,11

FREI BINARY
Enable block
FREI=0 : -> Block is not processed
FREI=1 : -> The value of the source variable is read and allocated to the target variable ZIEL.

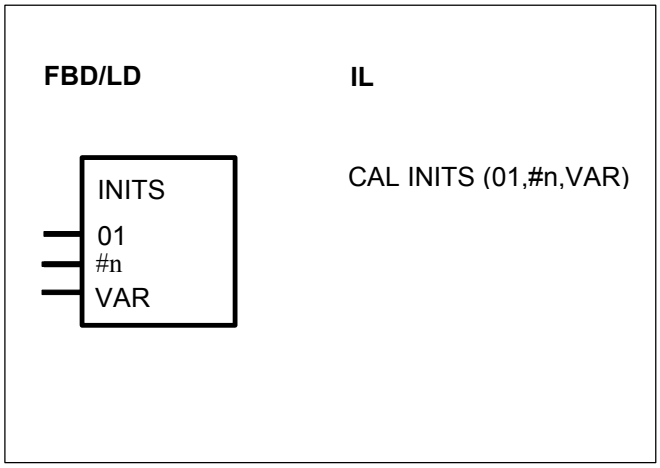
QUEL WORD
The source variable is specified at the input QUEL. The value of this variable is read and allocated to the target variable.

INDX WORD
The index value is specified at the input INDX. The target variable (see above for a

calculation) results from the index INDX and the basic variable.
Value range : -16383 ≤ INDX ≤ +16383
If INDX is out of range, the function block is not processed.

BASI WORD
The basic variable is specified at the input BASI. The target variable (see above for a calculation) results from the index INDX and the basic variables.

INITS INITIALIZE MEMORY AREA IN THE OPERAND MEMORY WITH ZERO



PARAMETERS

01	BINARY	%I, %O, %M, %S	Enabling for non-recurring processing of the block (0/1 edge)
#n	DIRECT CONSTANT	#, #H	Quantity (n) of memory words to be initialized
VAR	BINARY, WORD, DOUBLE WORD	%I, %O, %M, %K, %S, %IW, %AM, %MW, %KW, %MD, %KD	The n memory words specified at the input #n are initialized as from this variable.

DESCRIPTION

This function block serves the purpose of word-by-word initialization of memory areas in the operand memory with the value 0.

The block is run through precisely once in the event of a 0/1 edge at the input 01. In doing so, *n memory words* are initialized with the value 0 as from the variable specified at the input VAR. The variable specified at this input can be a BINARY, WORD or DOUBLE WORD operand.
The following must be observed :
BINARY operand : Occupies 1 BYTE in the operand memory.
WORD operand : Occupies 1 WORD in the operand memory.
DOUBLE WORD operand : Occupies 2 WORDS in the operand memory.

0-1 BINARY
Block enable.
The block is run through precisely *once* when a 0/1 edge appears at this input.

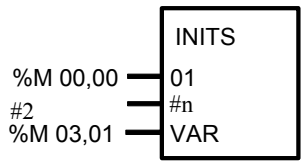
#n DIRECT CONSTANT
Quantity (n) of *memory words* to be initialized specified as a DIRECT CONSTANT.
Value range : $0 \leq n \leq 65535$
No initialization is realized if the value 0 is specified.

VAR BINARY, WORD, DOUBLE WORD
n memory words are initialized with the value 0 as from this variable (inclusive).

Example

The 4 binary flags
%M 03,01
%M 03,02
%M 03,03
%M 03,04
are to be initialized with the value 0.
A binary flag in memory takes up 1 byte. Therefore, there is a total of 4 bytes = 2 words to be initialized.
For this reason, the value 2 must be specified at the input #n as a DIRECT CONSTANT and the flag %M 03,01 at the input VAR.

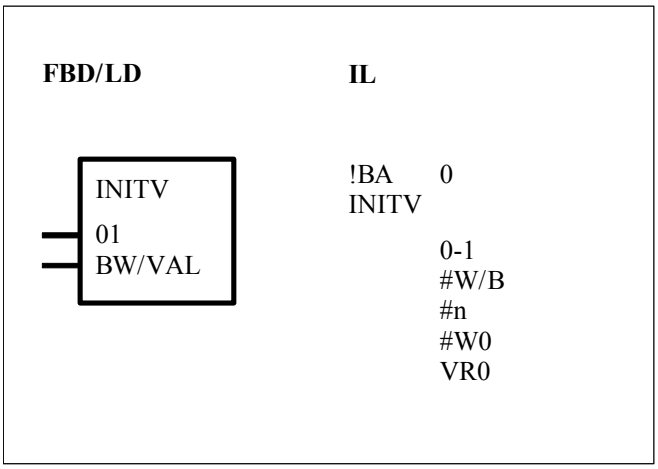
FBD/LD



IL

```
CAL INITS( %M00.00, #2, %M03.01
```

INITV INITIALIZE VARIABLES



PARAMETERS

0-1	BINARY	%I, %O, %M, %S	Enabling for non-recurring processing of the block (0/1 edge)
BW/VAL	BINARY, WORD DIRECT CONSTANT #,#H	%I, %O, %M, %IW, %OW, %MW	Initialization value for the I subsequent variable and variable (the format is internally recognised)The inputs BW/val can be duplicated in pairs
#W/B	DIRECT CONSTANT	#, #H	Format specification
#n	DIRECT CONSTANT	#, #H	Number of variables to be initialized
#W0	DIRECT CONSTANT	#, #H	Initialization value for the subsequent variable; The inputs #W0 and VR0 can be duplicated in pairs
VR0	BINARY, WORD	%I, %O, %M, %IW, %OW, %MW	Variable to be initialized; The inputs VR0 and #W0 can be duplicated in pairs

DESCRIPTION

This function block serves to initialize BINARY and WORD variables with numeric values.
The initialization values are specified as direct constants. The block is run through precisely once when a 0/1 edge appears at the input 0-1.

01 BINARY
The block is run through precisely *once* when a 0/1 edge appears at the input 0-1.

BW/VAL BINARY/WORD/DIRECT CONSTANT
The variable to be initialized is specified at the input BW/val and the second parameter is its value as a direct constant
This parameter has to be duplicated in pair

#W/B DIRECT CONSTANT
The format of the variable to be initialized is specified as a direct constant at the input #W/B. The following applies :

#W/B = 1 : Word variable (%IW, %OW, %MW)
#W/B = 0 : Binary variable (%I, %O, %M)

#n DIRECT CONSTANT
The quantity n of variables to be initialized is specified as a direct constant at the input #n.

#W0 DIRECT CONSTANT
The initialization value for the subsequent variable is specified as a direct constant at the input #W0.
The inputs #W0 and VR0 can be duplicated in pairs.
The following affiliations apply :

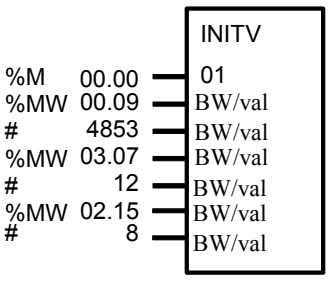
Value 0 -> Variable 0
Value n-1 -> Variable n-1

VR0 BINARY/WORD
The first variable to be initialized is specified at the input VR0.
The inputs VR0 and #W0 can be duplicated in pairs.

Example

The following variables are to be initialized:
%MW 00,09 = 4853
%MW 03,07 = 12
%MW 02,15 = 8

FBD/LD

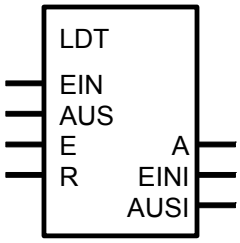


IL

```
!BA     0
INITV
        %M      00.00
        #           1
        #           3
        #        4853
        %MW     00.09
        #          12
        %MW     03.07
        #           8
        %MW     02.15
```

LDT ILLUMINATION PUSHBUTTON CONTROL

FBD



IL

```
CAL LDT (EIN,AUS,E,R,A,EINI,AUSI)
```

PARAMETERS

EIN	BINARY	%I, %O, %M, %K, %S	Dynamic input for activating output A
AUS	BINARY	%I, %O, %M, %K, %S	Dynamic input for deactivating output A
E	BINARY	%I, %O, %M, %K, %S	Dynamic input for activating and deactivating output A
R	BINARY	%I, %O, %M, %K, %S	Static input for deactivating output A
A	BINARY	%O, %M	Output
EINI	BINARY	%O, %M	Pulse for the duration of one PLC program cycle when activating output A
AUSI	BINARY	%O, %M	Pulse for the duration of one PLC program cycle when deactivating output A

DESCRIPTION

This function block changes the state at the output A with every 0/1 edge at the input E (like a toggle flip-flop). However, this only applies if the input R is inactive. The output A can be influenced additionally with the inputs EIN, AUS and R.
All inputs except R are edge-controlled, i. e. they react only to a 0/1 edge. The input R is active for as long as a static 1 signal is present at it. Brief pulses are available at the output EINI and AUSI when a status change of the output A occurs.

EIN	AUS	E	R	A
X	X	X	H	L
X	X	>	L	NA
X	>	<	L	L
>	<	<	L	H

NA	: Inverting of the previous state of A
X	: State is ignored
>	: 0/1 edge
<	: no 0/1 edge
L	: State 0
H	: State 1

EIN BINARY
The output A is set with every 0/1 edge at the input EIN. However, this only applies

when the inputs AUS, E and R are inactive. Therefore, of all inputs the input EIN has the lowest priority.

AUS BINARY
The output A is reset with every 0/1 edge at the input AUS. Therefore, the input AUS has a higher priority than the input EIN, but a lower priority than the inputs E and R.

E BINARY
With every 0/1 edge at the input E, the block changes the status at the output A. However, this only applies if the input R is inactive. Therefore, the input E has a higher priority than the inputs EIN and AUS, but a lower priority than the input R.

R BINARY
The output A is reset when a static 1 signal is present at the input R.
The input R has a higher priority than the inputs E and EIN. A priority conflict with the input AUS is not possible because both inputs have the same effect on the outputs.

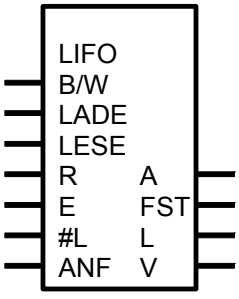
A BINARY
The output A is changed corresponding to the definition of the inputs.

EINI BINARY
A pulse is generated at the output EINI with every 0/1 edge of the output A. The length of this pulse is 1 program cycle.

AUSI BINARY :
A pulse is generated at the output AUSI with every 1/0 edge of the output A. The length of this pulse is 1 program cycle.

LIFO STACK, LAST IN/FIRST OUT

FBD



IL

```
CAL LIFO (R,LADE,LESE,E,B/W,#L,ANF,A,FST,L,V)
```

PARAMETERS

B/W	BINARY	%I, %O, %M, %S, %K	Binary data/word data
LADE	BINARY	%I, %O, %M, %S, %K	Load LIFO
LESE	BINARY	%I, %O, %M, %S, %K	Read LIFO
R	BINARY	%I, %O, %M, %S, %K	Reset LIFO, 0/1 edge
E	BINARY	%I, %O, %M, %S, %K	Enter data in the LIFO

	WORD	%IW, %OW, %MW, %KW	
#L	DIRECT CONSTANT	#, #H	Number of memory locations (bytes)
ANF	BINARY	%I, %O, %M, %K	Start of the LIFO in the flag area
	WORD	%IW, %OW, %MW, %KW	
A	BINARY	%I, %M	Output of data from the LIFO
	WORD	%OW, %MW	
FST	WORD	%OW, %MW	Filling level of the LIFO
L	BINARY	%O, %M	LIFO empty
V	BINARY	%O, %M	LIFO full

DESCRIPTION

This function block realizes a stack for binary or word data from which the data written in last is read out first (last in/first out).

B/W BINARY
The input B/W serves to preselect binary or word processing.
B/W = 0 -> word processing
B/W = 1 -> binary processing
Binary or word processing is defined once and must not be changed during operation of the LIFO.

LADE BINARY
The value present at the input E is transferred to the next free position of the LIFO by means of a 1 signal at the LADE input. If the LIFO is “full” and a “load” signal is present, the new value will *not* be read in. A new value can only be read in if a value has been read out first. It is then transferred to the first position of the LIFO.

LESE BINARY
A 1 signal at the input LESE results in output of the current LIFO value through the output A. If the LIFO is “empty”, a pending read order is ignored and the value 0 is output through the output A. The output A is always set to 0 if there is no read order.

LADE and **LESE**
If load and read orders are present simultaneously, the value to be loaded is forwarded directly from the input E to the output A. At the same time, the LIFO does not store the value. The LIFO does not change its filling level in this mode. The value output through the output FST remains constant. If the LIFO is empty, the output L remains 1 and the output FST is set to 0.

R BINARY
A 0->1 edge at the input R results in the reset of the block. Therefore, values read in before are no longer available. The output L assumes the value 1 and the outputs FST and A assume the value 0.

R and **LADE**
If R and LADE signals are present simultaneously, the reset is performed first and then directly afterwards the load operation. Therefore, the LIFO is reset and the new value to be read in is then immediately stored in the LIFO as the first value.

R and **LADE** and **LESE**
The value at the input E is forwarded directly to the output A. The output L is set permanently to 1 and the output FST is set permanently to 0.

E BINARY/WORD
The value to be transferred into the LIFO is specified at the input E.

#L DIRECT CONSTANT
The number of required memory locations (bytes) of the LIFO is specified at the input #L. This quantity is specified as a direct constant and results from the following formula :

BINARY data : #L = Number of binary values to be stored
WORD data : #L = 2 * Quantity of word values to be stored
E.g. : BINARY data : 3 values -> #L = 3
WORD data : 3 values -> #L = 6

The LIFO length parameter is subjected to a plausibility check for the value 0 and also, in word processing mode, for an odd byte parameter. If the parameter specified at the input #L is incorrect, the LIFO will assume the initial state (as after R).

ANF BINARY/WORD
The LIFO memory start address is specified as a binary or word flag at the input ANF. The LIFO begins with the specified flag.
A BINARY/WORD
When the LIFO is read, the current value is available at the output A. If *no* read order is available, the value 0 is output.

FST WORD
The output FST indicates the current filling level of the LIFO at any time. The filling level is the number of binary or word values stored in the LIFO.

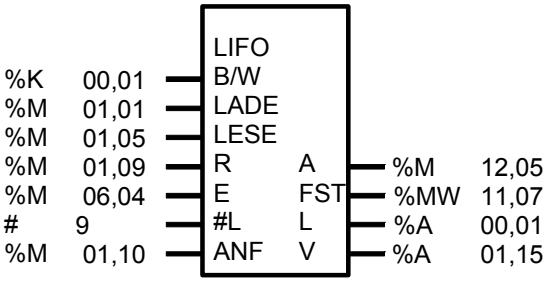
L BINARY
The output L indicates whether or not the LIFO is empty.
L = 0 -> LIFO is not empty
L = 1 -> LIFO is empty

V BINARY
The output V indicates whether or not the LIFO is full.
V = 0 -> LIFO is not full
V = 1 -> LIFO is full
No more values can be read in if the LIFO is full. A value can only be read in again once a value has been read out. This then takes place as from the start of the LIFO.

Example

LIFO for 9 binary data, stored as of %M 01,10

FBD

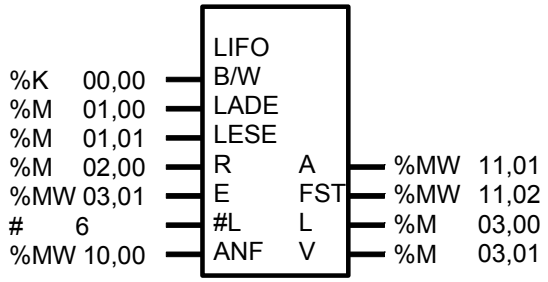


IL

```
CAL     LIFO    (%M01,09, %M01,01, %M01,05, %M06,04, %K00,01, #9, %M01,10,
                 %M12,05, %MW11,07, %A00,01, %A01,15)
```

LIFO for 3 word data, stored as of %MW 10,00

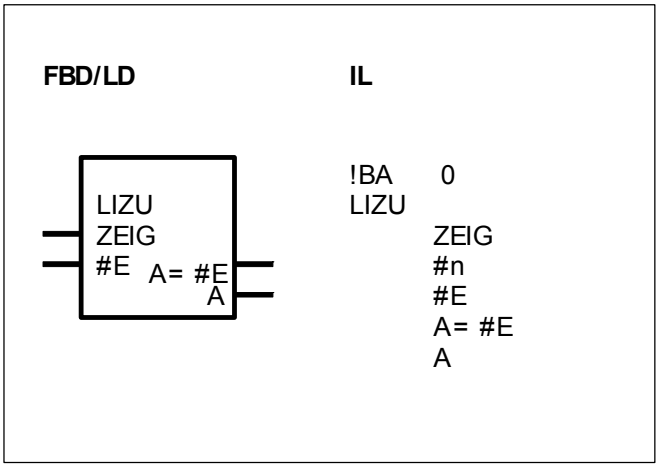
FBD



IL

```
CAL     LIFO    (%M02,00, %M01,00, %M01,01, %MW03,01, %K00,00, #6, %MW10,00,
                 %MW11,01, %MW11,02, %M03,00, %M03,01)
```

LIZU LIST ALLOCATOR



PARAMETERS

ZEIG	WORD	%OW, %IW, %MW, %KW	Pointer to the list of direct constants
#n	DIRECT CONSTANT	#, #H	Number of direct constants in the list
#E	DIRECT CONSTANT	#, #H	List of direct constants; **capable of duplication**
A	WORD	%OW, %MW	Selected direct constant
A=#E	BINARY	%O, %M	0 ≤ ZEIG < #n, i. e. pointer in the valid range

DESCRIPTION

This function block has a list of direct constants at its inputs #E (#E0... #En-1). With a list pointer, it selects a constant out of this list and outputs it through its output A.

ZEIG WORD

The pointer to the direct constant to be selected from the list is specified at the input ZEIG. The following affiliations apply :
ZEIG = 0 -> Direct constant at #E0
ZEIG = 1 -> Direct constant at #E1
. . . .
ZEIG = n-1 -> Direct constant at #En-1

The value at the input ZEIG is subjected to a validity check. The result of this range check is signalled at the output A=#E.
Allowed range : 0 ≤ ZEIG ≤ n-1
Where n : Number of the inputs #E0...#En-1.
No allocation to the output A takes place if the value at the input ZEIG is outside the allowed range.

#n DIRECT CONSTANT

ONLY used in IL language

The number n of the direct constants planned at the inputs #E is specified at the input #n. This is specified as a direct constant.

#E DIRECT CONSTANT

The input #E is capable of duplication (#E0...#En-1). One of the direct constants specified at the inputs #E is selected with the value at the input ZEIG and allocated to the output A.

A=#E BINARY

The output A=#E specifies whether the value of the list pointer (input ZEIG) is within the allowed range.

Allowed range : $0 \leq ZEIG \leq n-1$
where n : Number of the direct constants at #E0...#En-1.

The following applies :

ZEIG in the allowed range -> A=#E = 1

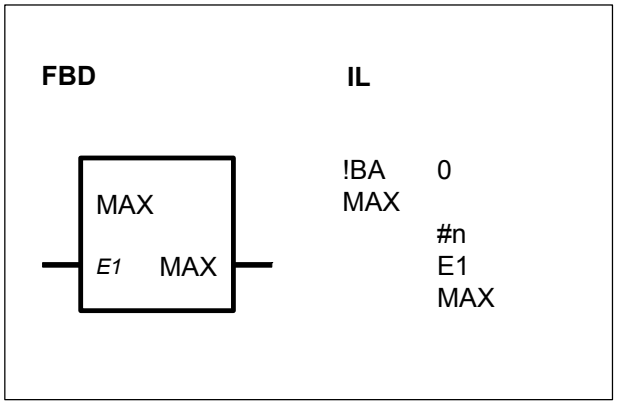
ZEIG in the forbidden range -> A=#E = 0

If the list pointer has a forbidden value, no constant can be selected nor allocated to the output A. In this case, the output A is *not* updated.

A WORD

The value of the selected direct constant is allocated to the operand at the output A.

MAX MAXIMUM VALUE GENERATOR



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs
E1	WORD	%IW, %MW, %OW, %KW	1st input value; **capable of duplication**
MAX	WORD	%OW, MW	Output (maximum value)

DESCRIPTION

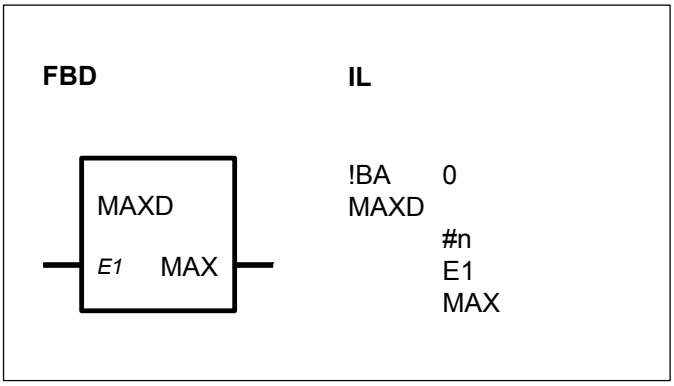
From n operands, this function block generates the maximum value and allocates it to the output.

#n DIRECT CONSTANT (#, #H)
Only used in IL.
The number of operands from which the maximum value is to be determined is specified at the input #n. This is specified as a direct constant.
n > 0 applies.

E1 WORD
The input must be duplicated as many times as necessary until the number of inputs specified at the input #n exists. On the basis of the values of the operands at these inputs, the maximum value is determined and is allocated to the output MAX.

MAX WORD
The maximum value from the n input operands is available at the output MAX.

MAXD MAXIMUM VALUE GENERATOR, DOUBLE WORD



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs
E1	DOUBLE WORD	%MD, %KD	1st input value; **capable of duplication**
MAX	DOUBLE WORD	%MD	Output (maximum value)

DESCRIPTION

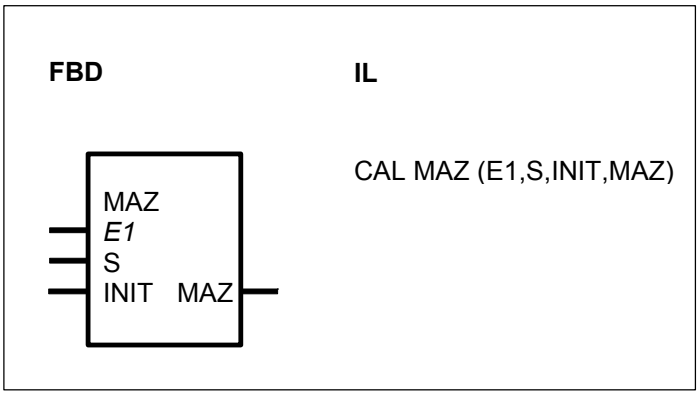
From n operands, this function block generates the maximum value and allocates it to the output.

#n DIRECT CONSTANT (#, #H)
Only used in IL.
The number of operands from which the maximum value is to be determined is specified at the input #n. This is specified as a direct constant.
n > 0 applies.

E1 DOUBLE WORD
The input must be duplicated as many times as necessary until the number of inputs specified at the input #n exists. On the basis of the values of the operands at these inputs, the maximum value is determined and is allocated to the output MAX.

MAX DOUBLE WORD
The maximum value from the n input operands is available at the output MAX.

MAZ MAXIMUM VALUE GENERATOR AS A FUNCTION OF TIME



PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input value whose maximum time is to be determined
S	BINARY	%I, %M, %O, %S, %K	Set input
INIT	WORD	%IW, %MW, %OW, %KW	Initial value
MAZ	WORD	%OW, %MW	Maximum value

DESCRIPTION

This function block determines, on the basis of the time progression of a signal, its maximum value occurring up to the current point in time.

The value of the operand at the input E1 is compared to the previously occurring maximum value.

If the input value E1 is higher than the previously occurring maximum, the input value is the new maximum value and is allocated to the operand at the output MAZ.

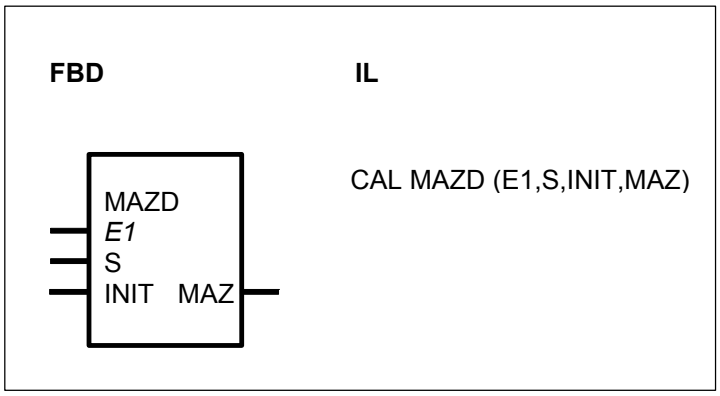
If the input value E1 is less than the previously occurring maximum value, the previous maximum value is allocated to the output.

The output MAZ is set to the value of the operand at the input INIT (initial value) with the 0->1 edge at the binary input S.

The following applies :

E1 < MAZ	-> MAZ = MAZ
E1 ≥ MAZ	-> MAZ = E1
S = 0->1 edge	-> MAZ = INIT

MAZD MAXIMUM VALUE GENERATOR AS A FUNCTION OF TIME, DOUBLE WORD



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input value whose maximum time is to be determined
S	BINARY	%I, %M, %O, %S, %K	Set input
INIT	DOUBLE WORD	%MD	Initial value
MAZ	DOUBLE WORD	%MD	Maximum value

DESCRIPTION

This function block determines, on the basis of the time progression of a signal, its maximum value occurring up to the current point in time.

The value of the operand at the input E1 is compared to the previously occurring maximum value.

If the input value E1 is higher than the previously occurring maximum, the input value is the new maximum value and is allocated to the operand at the output MAZ.

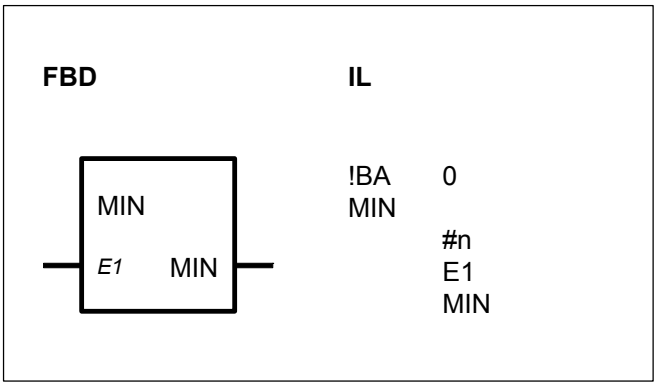
If the input value E1 is less than the previously occurring maximum value, the previous maximum value is allocated to the output.

The output MAZ is set to the value of the operand at the input INIT (initial value) with the 0->1 edge at the binary input S.

The following applies :

E1 < MAZ	-> MAZ = MAZ
E1 ≥ MAZ	-> MAZ = E1
S = 0->1 edge	-> MAZ = INIT

MIN MINIMUM VALUE GENERATOR



PARAMETER

#n	DIRECT CONSTANT	#, #H	Number of inputs
E1	WORD	%IW, %MW, %OW, %KW	1st input value; **capable of duplication**
MIN	WORD	%OW, %MW	Output (minimum value)

DESCRIPTION

This function block determines the minimum value from n operands and allocates it to the output.

#n DIRECT CONSTANT (#, #H)

Only used in IL.
The number of operands from which the minimum value is to be determined is specified at the input #n. This is specified as a direct constant.
n > 0 applies.

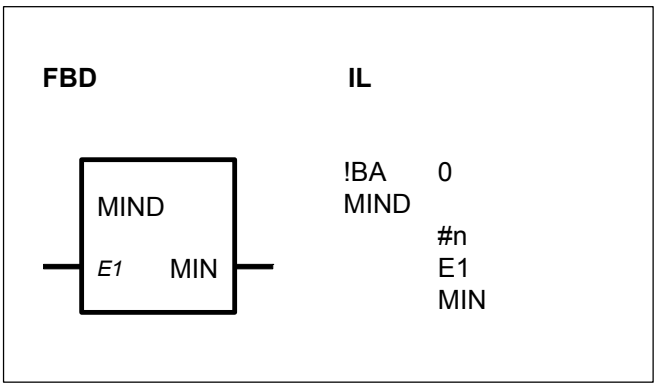
E1 WORD

The input E1 must be duplicated as often as the number of inputs specified at the input #n exists. On the basis of the values of the operands at these inputs, the minimum value is determined and is allocated to the output MIN.

MIN WORD

The minimum value of the n input operands is available at the output MIN.

MIND MINIMUM VALUE GENERATOR, DOUBLE WORD



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs
E1	DOUBLE WORD	%MD, %KD	1st input value; **capable of duplication**
MIN	DOUBLE WORD	%MD	Output (minimum value)

DESCRIPTION

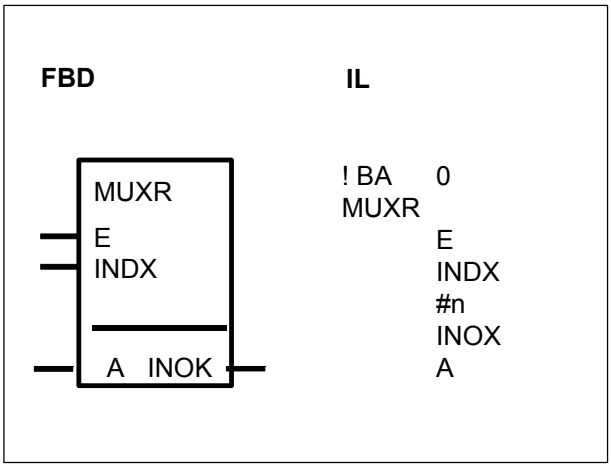
This function block determines the minimum value from n operands and allocates this value to the output.

#n DIRECT CONSTANT (#, #H)
Only used in IL.
The number of operands from which the minimum value is to be determined is specified at the input #n. This is specified as a direct constant.
n > 0 applies.

E1 DOUBLE WORD
The input E1 must be duplicated until the number of inputs specified at the input #n exists. On the basis of the values of the operands at these inputs, the minimum value is determined and is allocated to the output MIN.

MIN DOUBLE WORD
The minimum value of the n input operands is available at the output MIN.

MUXR MULTIPLEXER WITH RESET



PARAMETERS

E	WORD	%IW, %OW, MW, KW	Input
INDX	WORD	%IW, %OW, MW, KW	Index input
#n	DIRECT CONSTANT	#,#H	Quantity n of word outputs A0 ... An-1
INOK	BINARY	%O, M	Range monitoring for input INDX
A	WORD	%OW, MW	Word outputs A0 ... An-1; **capable of duplication**

DESCRIPTION

This function block connects the input E to one of the outputs A0...An-1 depending on the input INDX.
The word outputs that are not connected are set to 0.
The validity of the value at the input INDX is checked.

Relationship between E, INDX and A0...An-1 :
The input INDX is used to define with which of the outputs A0...A_{n-1} the input E is connected.

The following apply : INDX = 1 : E -> A0
INDX = 2 : E -> A1
INDX = 3 : E -> A2
: : :
INDX = n : E -> An-1
where 1 ≤ INDX ≤ n ≤ 32767 (theoretically)

E WORD
Input which is switched through to one of the outputs A0...An-1.

INDX WORD
Index input for selection of one of the outputs A0...An-1.

Value range : 1 ≤ INDX ≤ n
Note : INDX = 0 can be used to initialize the outputs A0 ... A_{n-1} (A0 ... An-1 = 0).

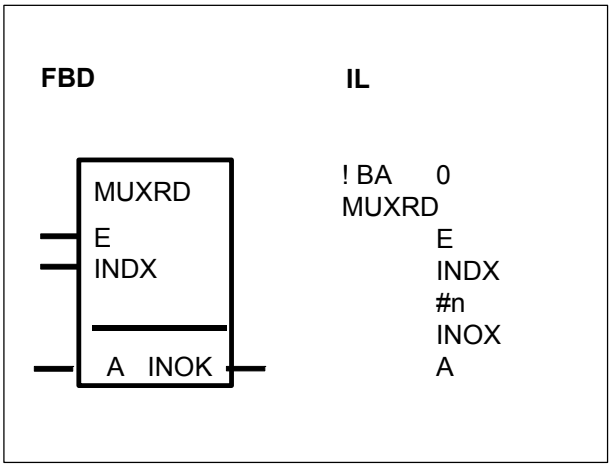
#n DIRECT CONSTANT
Only used in IL.
Quantity n of word outputs A0 ... A_{n-1}.

INOK BINARY
Range monitoring of the INDX input
The output indicates whether or not the input INDX is within the valid range.
Valid range : 1 ≤ INDX ≤ n
INOK = 1 -> Index input INDX within the valid range
INOK = 0 -> Index input INDX in the invalid range -> A0...A_{n-1} = 0

If the word input INDX is not within the valid range, all word outputs A0 to A_{n-1} are set to 0. Thus, for example, INDX = 0 can be used to initialize the outputs (A0...A_{n-1} = 0).

A WORD
The output A is capable of duplication (A0...A_{n-1}). The input E is allocated to one of the n outputs A0...A_{n-1}.

MUXRD MULTIPLEXER, DOUBLE WORD WITH RESET



PARAMETERS

E	DOUBLE WORD	%MD, %KD	Input
INDX	WORD	%IW, %OW, %MW, %KW	Index input
#n	DIRECT CONSTANT	#,#H	Quantity n of double word outputs A0 ... An-1
INOK	BINARY	%O, %M	Range monitoring for input INDX
A	WORD	%MD	Double word outputs A0...An-1; **capable of duplication**

DESCRIPTION

This function block connects the input E to one of the outputs A0...An-1 depending on the input INDX.
The double word outputs that are not connected are set to 0.
The validity of the value at the input INDX is checked.

Relationship between E, INDX and A0...An-1 :
The input INDX is used to define with which of the outputs A0...A_{n-1} the input E is connected.

The following applies :

INDX = 1 : E -> A0
INDX = 2 : E -> A1
INDX = 3 : E -> A2
: : : :
INDX = n : E -> An-1
where : 1 ≤ INDX ≤ n ≤ 32767 (theoretically)

E DOUBLE WORD
Input which is switched through to one of the outputs A0...An-1.

INDX WORD
Index input for selection of one of the outputs A0...An-1.
Value range : 1 ≤ INDX ≤ n
Note : INDX = 0 can be used to initialize the outputs A0...A_{n-1} (A0...An-1 = 0).

#n DIRECT CONSTANT
Only used in IL.
Quantity n of double word outputs A0...An-1.

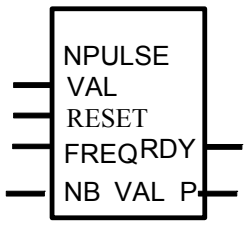
INOK BINARY
Range monitoring of the input INDX
The output indicates whether or not the input INDX is within the valid range.
Valid range : 1 ≤ INDX ≤ n
INOK = 1 -> Index input INDX within the valid range
INOK = 0 -> Index input INDX in the invalid range -> A0...An-1 = 0

If the word input INDX is not within the valid range, all double word outputs A0 to A_{n-1} are set to 0. Therefore, for example, INDX = 0 can be used to initialize the outputs (A0...A_{n-1} = 0).

A DOUBLE WORD
The output A is capable of duplication (A0...A_{n-1}). The input E is allocated to one of the n outputs A0...A_{n-1}.

NPULSE

FBD



IL

```
CAL NPULSE (VAL,RESET,FREQ,
NB,RDY,VAL_P)
```

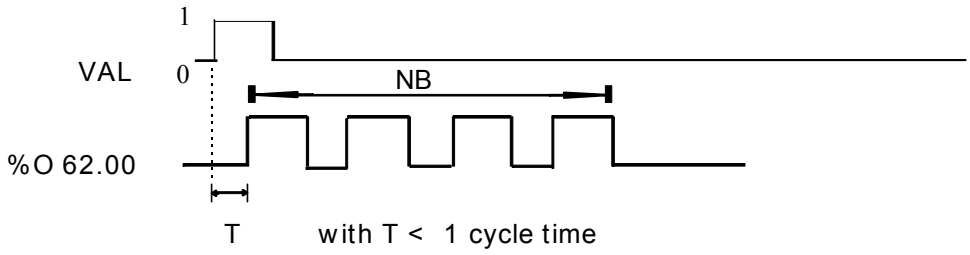
PARAMETERS

VAL	BINARY	%I, %M, %O, %S	Validation bit
RESET	BINARY	%I, %M, %O, %S	Reset
FREQ	WORD	%IW, %MW, %OW, %KW	Frequnecy
NB	WORD	%IW, %MW, %OW, %KW	Number of pulses
RDY	BINARY	%M, %O	Ready
VAL_P	WORD	%OW, %MW	Number of pulses elapsed

DESCRIPTION

The %O 62.00 output of central units serie 40 and 50 can be used for a pulse generator.

Pulses are generated at the output %O 62.00.
The rising edge at the input VAL starts the pulse generator from the beginning.
The first period starts with a high signal.



VAL BINARY
Validation bit
VAL 0->1 The pulse mode is validated. %O 62.00 can not be used normally.

RESET BINARY
Reset =1 stops the pulse generator, the output is set to 0 till reset =1

FREQ WORD
The frequency is defined with : $1/((256\text{-FREQ})*384\ 10^{-6})$ Hz

Pulse frequency : 10 Hz ≤ Frequency ≤ 2,604 kHz

FREQ = 0 Frequency = 10.173Hz
FREQ = 1 Frequency = 11.212Hz
...
FREQ = 255 Frequency = 2,604Khz

If FREQ <0 then Frequency = 10.173Hz
If FREQ > 255 then Frequency = 2,604Khz

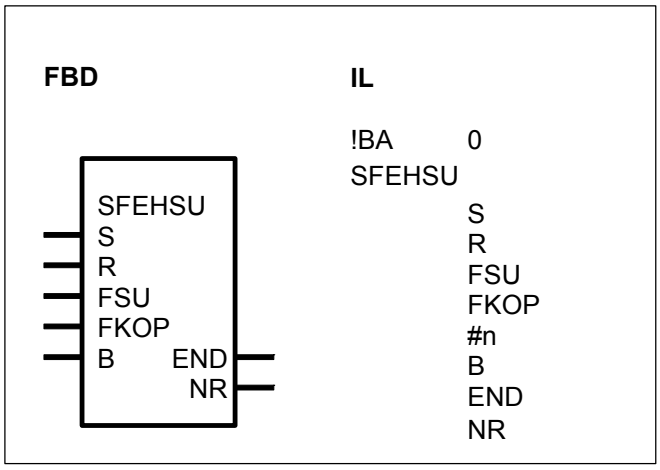
NB WORD
Number of pulses to be generated.
If NB<0 then the function NPULSE generates continiuously pulses till RESET input is valided.

RDY BINARY
Ready bit
RDY = 0 Counting
RDY = 1 No counting

VAL_P WORD
Number of pulses elapsed
This number is estimated by the function bloc every cycle time. It doesn't represent the exact value.

The internal memory bit %O62.00 is disabled during pulses are generating

SFEHSU ERROR SEARCHER WITH STORAGE



PARAMETERS

S	BINARY	%I, %O, %M, %S, %K	Set
R	BINARY	%I, %O, %M, %S, %K	Reset
FSU	BINARY	%I, %O, %M, %S, %K	Enable search
FKOP	BINARY	%I, %O, %M, %S, %K	Enable copy
#n	DIRECT CONSTANT	#, #H	Number of binary variables

B	BINARY	%I, %O, %M, %S, %K	List of binary variables; **capable of duplication**
END	BINARY	%O, %M	List end reached
NR	WORD	%OW, %MW	List number of the variable found

DESCRIPTION

This function block successively searches through a list of binary variables (%I, %O, %M, %S) for set binary variables. If a set binary variable is found, its number is output through the output NR. In doing so, the block does not directly search through the input list for the set binary variables, but through its image, which it stores in an internal list. After a set binary variable has been found in the internal list and its number has been output through the output NR, this binary variable is deleted from the internal list.

S BINARY

A 1 signal at the input S results in the following :
– All set binary variables of the input list are additionally entered in the internal list (ORed with the internal list).
– The block is prepared for a search from the start of the internal list, i. e. the internal list's pointer is set to its start.
– The output END (list end reached) is set to 1 and the output NR (number of the binary variable) is set to 0.
– If a set variable has been found during the previous search in the internal list, this is deleted from the internal list in order to avoid a double message.

R BINARY

– All set binary variables in the internal list are deleted when the input R has a 1 signal. The set binary variables at the block's inputs are *not* affected by this.
– The pointer to the internal list is *not* changed, i. e. a subsequent search begins as from the point to which the pointer of the internal list pointed before deletion.
– The output END is set to 1 and the output NR is set to 0.

FSU BINARY

– A 1 signal enables the search for set binary variables in the internal list.
– A 0 signal disables the search; in doing so, the old values at the outputs are reallocated during each cycle.
– If a binary variabe with the value 1 has been found in the internal list, the number of this set binary variable is output through the output NR. The binary variable is then cleared from the *internal* list. The output END (list end reached) is set to 0 if the binary variable is not the last one in the list.
– Numbering of the binary variables begins with 1.
– The output END retains the value 0 as long as the block has not reached the end of the internal list during the search.
– Each time it is called again, the block continues the search in the internal list, beginning with the next binary variable a fter the one found last. *Beforehand*, the binary variable found last is deleted from the internal list in order to avoid a double message.
– The following applies if the end of the internal list is reached during the search :

• the output END (list end reached) is set to 1.
• the number of the binary variable found last in the internal list is output through the output NR.
• each further time the block is called, the internal list is searched through as from the point of the binary variable found last. The search ends when a set binary variable is found or when the end of the list is reached.
- If the last binary variable of the list is set and has been found during the search, the search stops there until a new search from the start of the internal list is prepared by means of the input S.
– If no set binary variable has been found during the course of a search from the start of the list, the output END (list end reached) is set to 1 and the output NR (number) retains the value 0. Each time the block is called again, the whole list is always searched through until a binary variable assumes the value 1 and is found.

FKOP BINARY
When the input FKOP has a 1 signal, all set binary variables of the input list are additionally entered in the internal list. At the same time, the binary variables already set in the internal list are retained. Updating is realized by “ORing” of the input list with the internal list.
Restart when searching after updating
Updating of the internal list has no influence on the next search. The search begins precisely at the point in the internal list where it would also have been begun if no updating had taken place.

#n DIRECT CONSTANT
ONLY used in IL language
The number of binary variables planned at the inputs B0...Bn-1 is specified at the input #n. This is specified as a direct constant.
Important : The quantity at the input #n *must* be an integral multiple of 16. Dummy operands may also be planned (e.g. %K0,0 is specified at all inputs not needed) in order to keep to this stipulation.

B BINARY
The input B is capable of duplication (B0...Bn-1). The binary variables to be examined are specified at the inputs B0...Bn-1. The number of variables must always be an integral multiple of 16. To achieve this, assign %K0,0 = 0 to inputs Bi that are not needed.

END BINARY
Whether or not the end of the list has been reached during the search is signalled at the output END.
END = 0 -> List end not reached
END = 1 -> List end reached
If the last variable of the list is set *and* has been found during the search just carried out, its number is output through the output NR and the value 1 is additionally allocated to the output END.

NR WORD
The list number of the variable found last is output through the output NR. The following affiliations apply :

Variable at the input	List number
B0	1
B1	2
.	.
.	.
.	.
B_{n-1}	n

If the end of the list is reached during the search without a new variable having been found, the number of the variable found at last continues to be output through the output NR.
The value 0 is output through the output NR if *no* variable is set in the list.

Priorities of the inputs S, R, FSU, FKOP
- The set input S has the highest priority. No other input is processed as long as the input S has a 1 signal.
- The reset input R has the second highest priority. The subsequent inputs are not processed as long as the input R has a 1 signal.
- The update input FKOP has priority over the enable search input FSU.

Updating of the internal list (input FKOP) :
Case 1 : Updating enabled (FKOP = 1)
If updating is enabled, all set binary input variables are additionally entered in the internal list. If the search is enabled at the input FSU, it is carried out immediately after updating of the internal list.
Case 2 : Updating not enabled (FKOP = 0)
If updating is not enabled, the search is carried out immediately if enabled at the input FSU.
If the search is not enabled in both cases, the old values are output through the outputs END and NR.

UHR CLOCK

FBD

IL

```
CAL   UHR   (FREI,S,SEC,MIN,H,TAG,MON,JHR,
            WTG,AKT,FEHL,ASEC,AMIN,AH,
            ATAG,AMON,AJHR,AWTG)
```

PARAMETERS

FREI	BINARY	%I, %O, %M, %K, %S	Enable block processing
S	BINARY	%I, %O, %M, %K, %S	0/1 edge sets the time and date
SEC	WORD	%IW, %OW, %MW, %KW	Set input for the seconds
MIN	WORD	%IW, %OW, %MW, %KW	Set inputs for the minutes
H	WORD	%IW, %OW, %MW, %KW	Set inputs for the hours
TAG	WORD	%IW, %OW, %MW, %KW	Set inputs for the days
MON	WORD	%IW, %OW, %MW, %KW	Set inputs for the months
JHR	WORD	%IW, %OW, %MW, %KW	Set inputs for the years
WTG	WORD	%IW, %OW, %MW, %KW	Set inputs for the weekdays
AKT	BINARY	%O, %M	Topicality (usefulness) of the data at the outputs
FEHL	WORD	%OW, %MW	Error identifier
ASEC	WORD	%OW, %MW	Seconds output
AMIN	WORD	%OW, %MW	Minutes output
AH	WORD	%OW, %MW	Hours output
ATAG	WORD	%OW, %MW	Days output
AMON	WORD	%OW, %MW	Months output
AJHR	WORD	%OW, %MW	Years output
AWTG	WORD	%OW, %MW	Weekday No. output

DESCRIPTION

This function block allows users to set and display the current time and the current date.
The clock is set by means of the set inputs for the time and date. The values present at the set inputs are adopted by a 0/1 edge at the input S. As long as a 1 signal is present at the FREI input, the current date and time are indicated at the block's outputs.

FREI BINARY

Block enable
FREI = 0 : The block is not processed. The AKT and FEHL outputs are set to 0. The time and date outputs are no longer changed by the block.
FREI = 1 : Block is processed

S BINARY
Set inputs for date and time
In the event of a 0/1 edge at the input S, the clock is set to the values present at the time and date inputs. During the setting, the time and date at the block's output are invalid (output AKT = 0).
If the specified set values are inadmissible, the AKT output is set to 0 and an error message appears at the FEHL output. The values present at the time and date outputs are invalid in this case. The clock has to be set again.

SEC WORD
Set input for the seconds.
Value range : 0...59.

MIN WORD
Set input for the minutes.
Value range : 0...59.

H WORD
Set input for the hours.

The clock operates in 24 hour mode, i.e. it changes from 23 :59 :59 h to 0 :0 :0 h.
Value range : 0...23.

TAG WORD
Set input for the days (which day of the month)

Note for 07 KR 31:
The clock knows the number of days depending on the months and leap years. For the clock, a leap year exists when the year number is an integral multiple of 4. The maximum value for the days (28, 29, 30, 31) depends on the month.
Value range : 1...28, 29, 30, 31.

MON WORD
Set input for the month.
Value range : 1...12.

JHR WORD
Set input for the years.
The clock only indicates the years and decades.
Value range : 0...99.

WTG WORD
Set input for the number of the weekday.
Value range : 1...7.

Example :
The clock is set on Friday, 01.07.88. If the value 6 is entered for the week day number, Friday is now the 6th day of the week and Sunday is defined as the 1st day of the week.

AKT BINARY
Indication of the topicality (usefulness) of the outputs.
AKT is 1 if :
- The date and time outputs were updated in the current cycle;
- The values at the outputs are consistent, i.e. none of the values at the date or time outputs has changed during updating. They all originate from the same clock pulse;
- The clock was set correctly;

AKT = 1 -> FEHL = 0 : Date/time are *valid.*
AKT = 0 -> FEHL > 0 : Date/time are *invalid.*

The reason why is displayed at the output FEHL as error identifier.

FEHL WORD
In the event of an error, the relevant error identifier is available at the output FEHL.

Meanings of the error identifiers :
- No error has occurred :

FEHL = 0 : No error has occurred *or* FREI = 0,i.e. block disabled

- Error when setting the clock :

FEHL = 1 : $0 \leq SEC \leq 59$ has not been obeyed
FEHL = 2 : $0 \leq MIN \leq 59$ has not been obeyed
FEHL = 3 : $0 \leq H \leq 23$ has not been obeyed
FEHL = 4 : $1 \leq TAG \leq 28, 29, 30, 31$ (depending on the month, not tested in 07 KR 31) has not been obeyed
FEHL = 5 : $1 \leq MON \leq 12$ has not been obeyed
FEHL = 6 : $0 \leq JHR \leq 99$ has not been obeyed
FEHL = 7 : $1 \leq WTG \leq 7$ has not been obeyed

The following errors are not used in the 07 KR 31 :

FEHL = 8 : The transmission mailbox is currently occupied by another user. The block waits till the mailbox is free and thereafter sets date/time.
FEHL = 9 : Date/time at the outputs are *invalid.*
FEHL = 10 : Date/time are currently being set; this may take several PLC cycles.
FEHL = 11 : Setting was not successful, please repeat (unknown request code).
FEHL = 12 : Setting was not successful, please repeat (invalid mail parameter).
FEHL = 13 : Setting was not successful, please repeat (request code cannot be excecuted).

- Error when displaying date and time :

FEHL = 9 : Date/time at the outputs are *invalid.*

Outputs for date and time
The outputs are updated whenever a 1 signal is present at the FREI input *and* the clock has been set once. *During* the setting the outputs for date and time are *invalid.*

If the AKT output is equal to 1, the outputs for the date and time are valid. In the event of an error, an error identifier is output through the output FEHL.

ASEC WORD
Seconds output.
Value range : 0...59

AMIN WORD
Minutes output.
Value range : 0...59.

AH WORD
Hours output.
Value range : 0...23.

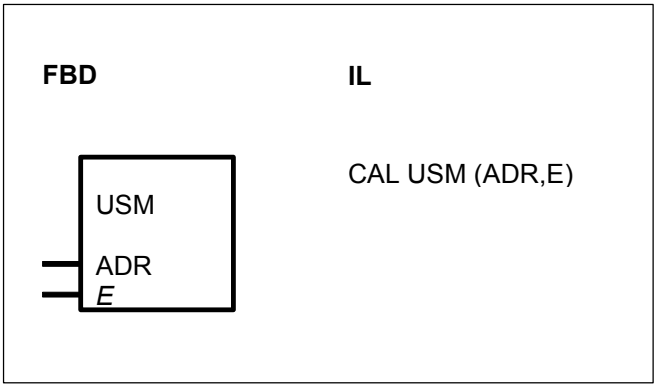
ATAG WORD
Days output.
Value range : 1...28, 29, 30, 31.

AMON WORD
Months output.
Value range : 1...12.

AJHR WORD
Years output.
Value range : 0...99.

AWTG WORD
Weekday No. output.
Value range : 1...7.

USM SWITCHOVER MULTIPLEXER



PARAMETERS

ADR	WORD	%IW, %OW, %MW, %KW	Indirect address of the operand to be written
E	WORD	%OW, %MW	Value to be allocated to the operand

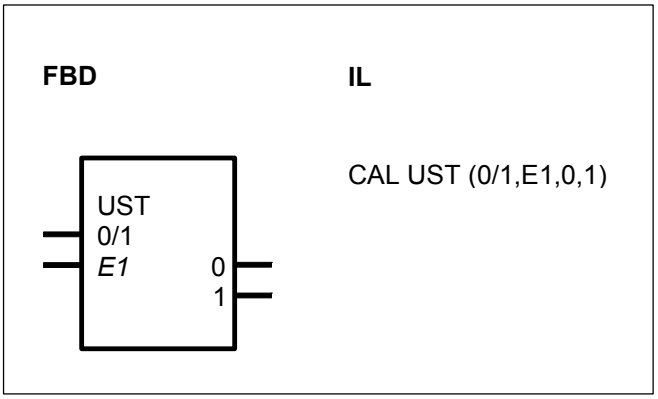
DESCRIPTION

This function block allocates the value from the input E to an operand, using the method of indirect addressing.

Note : The USM block can only be used meaningfully in conjunction with the ADRWA block.
The value of the operand at the input ADR is interpreted as the address of the operand to be written (indirect addressing). Therefore, the operand at the input ADR and its value represent an indirect address. This indirect address is generated by the ADRWA function block.

Note : Refer to the function block ADRWA for an explanation of the method of indirect addressing and the possibilities of using the USM function block.

UST SWITCHOVER GATE



PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
E1	WORD	%IW, %MW, %OW, %KW	Input
0	WORD	%OW, %MW	Output for 0/1 = 0
1	WORD	%OW, %MW	Output for 0/1 = 1

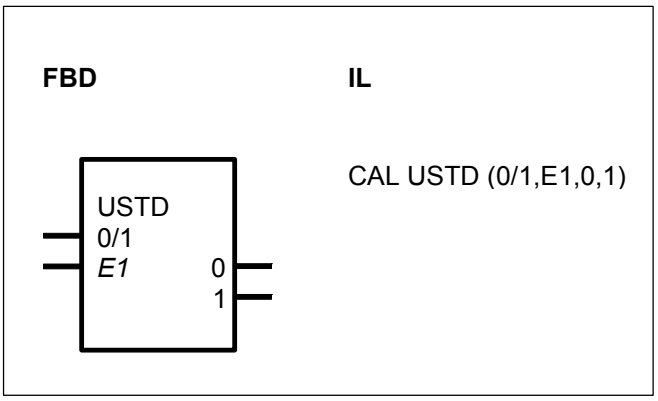
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the word operand at the input E1 to the word operand at the output 0.

A 1 signal at the binary input 0/1 allocates the value of the word operand at the input E1 to the word operand at the output 1.

The respective output that is not allocated retains its old value, but the old value is not updated.

USTD SWITCHOVER GATE, DOUBLE WORD



PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
E1	DOUBLE WORD	%MD, %KD	Input
0	DOUBLE WORD	%MD	Output for 0/1 = 0
1	DOUBLE WORD	%MD	Output for 0/1 = 1

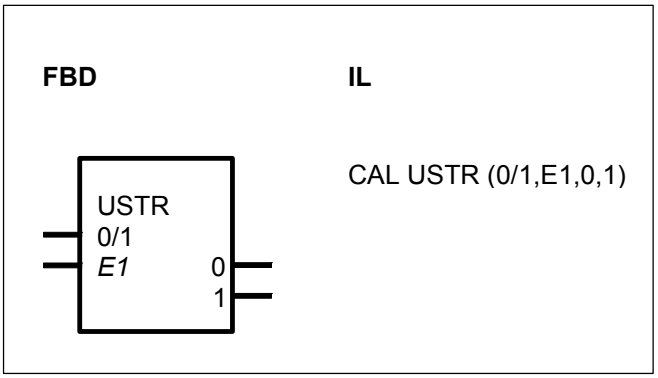
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the double word operand at the input E1 to the double word operand at the output 0.

A 1 signal at the binary input 0/1 allocates the value of the double word operand at the input E1 to the double word operand at the output 1.

The respective output that is not allocated retains its old value, but the old value is not updated.

USTR SWITCHOVER GATE WITH RESET



PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input

E1	WORD	%IW, %MW, %OW, %KW	Input
0	WORD	%OW, %MW	Output for 0/1 = 0
1	WORD	%OW, %MW	Output for 0/1 = 1

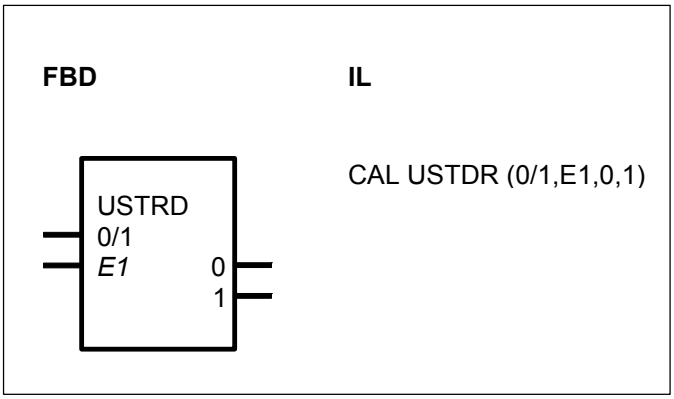
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the word operand at the input E1 to the word operand at the output 0.

A 1 signal at the binary input 0/1 allocates the value of the word operand at the input E1 to the word operand at the output 1.

The respective output that is not allocated is set to 0.

USTRD SWITCHOVER GATE WITH RESET, DOUBLE WORD



PARAMETERS

0/1	BINARY	%I, %M, %O, %S, %K	Switchover input
E1	DOUBLE WORD	%MD, %KD	Input
0	DOUBLE WORD	%MD	Output for 0/1 = 0
1	DOUBLE WORD	%MD	Output for 0/1 = 1

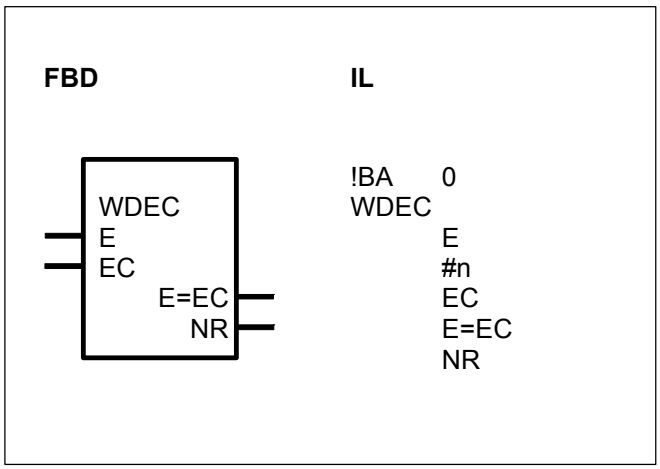
DESCRIPTION

A 0 signal at the binary input 0/1 allocates the value of the double word operand at the input E1 to the double word operand at the output 0.

A 1 signal at the binary input 0/1 allocates the value of the double word operand at the input E1 to the double word operand at the output 1.

The respective output that is not allocated is set to 0.

WDEC WORD DECODER



PARAMETERS

E	WORD	%IW, %OW, %MW, %KW	Input
#n	DIRECT CONSTANT	#, #H	Number of reference values
EC	WORD	%IW, %OW, %MW, %KW	Reference value; **duplicable**
E=EC	BINARY	%O, %M	Coincidence indication
NR	WORD	%OW, %MW	Number of the reference value in the event of coincidence

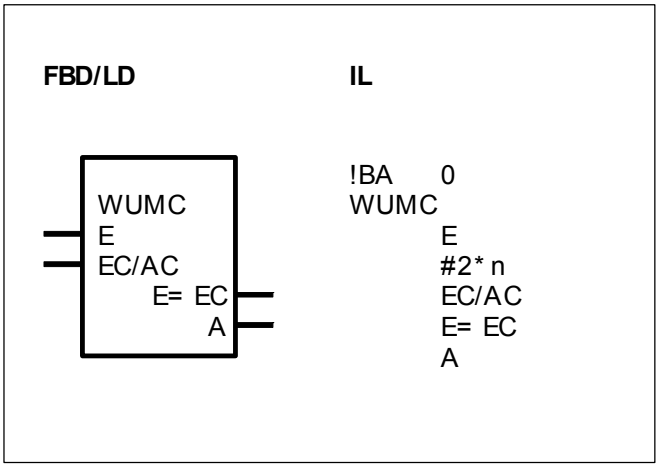
DESCRIPTION

This function block compares the value of the operand at the input E to the reference values of the operands at the inputs EC (EC0 ... ECn-1). The result of the comparison is signalled at the outputs.
If the input E agrees with at least 1 of the n reference values EC, the output E=EC is set to 1. The number of the 1st reference value EC agreeing with the input E is allocated to the operand at the output NR. Therefore, the number may assume a value from 1 to n.The outputs E=EC and NR are set to 0 if no agreement between the input value E and the reference value EC is determined.

No coincidence -> The outputs are : E=EC = 0 and NR = 0
Coincidence -> The outputs are : E=EC = 1 and NR = n, where n $\geq$ 1
E = EC0 -> NR = 1
E = EC2 -> NR = 2
. . .
E = ECn-1 -> NR = n

Note : In **IL** language, the number of inputs EC must be specified as a direct constant at the input #n.

WUMC WORD RECODER



PARAMETERS

E	WORD	%IW, %OW, %MW, %KW	Input
#2*n	DIRECT CONSTANT	#, #H	Quantity n of reference values (multiplied by 2)
EC/AC	WORD	%IW, %OW, %MW, %KW	Reference value and output code; **duplicable**
E=EC	BINARY	%O, %M	Coincidence indication
A	WORD	%OW, %MW	Output of the output code's value

DESCRIPTION

This function block compares the value of the operand at the input E to the reference values of the operands at the inputs EC/AC. If the input E agrees with at least one of the reference values EC/ACi, the output E=EC is set to 1. The output A receives the value of the output code EC/ACn+i, which is allocated to the reference value EC/ACi found. At each reference value at the inputs EC/ACi is assigned an operand for the output code EC/ACn+1.

Note : In **IL** language, the number of inputs EC must be specified as a direct constant at the input #2*n.

E WORD
The input E specify the value to be compared to the values of the n reference values EC/ACi.

#2*n DIRECT CONSTANT (#, #H)
Only used in IL language
The total number (2*n) of the reference values (EC/AC0...EC/ACn-1) and output codes (EC/ACn...EC/AC2n-1) is specified at the input #2*n. This is specified as an indirect constant.

EC/AC WORD
The input EC/AC must be duplicated according to the required number of reference values. The operands for the reference values are specified at the inputs EC/AC0...EC/ACn-1. The value of the operand at the input E is compared to the reference values. The output codes are specified at the inputs EC/ACn...EC/AC2n-1. The output code EC/ACi+n is output through the output A if the input E agrees with one of the reference values EC/ACi.
Affiliations between the reference values and output codes :
EC/AC0 <-> EC/ACn
EC/AC1 <-> EC/ACn+1
. .
. .
EC/AC_{n-1} <-> EC/ACn+(n-1)

E=EC BINARY
Agreement between the operand value of the input E and one of the reference values is signalled at the output E=EC.
The following applies : E=EC = 0 -> No coincidence
E=EC = 1 -> Coincidence

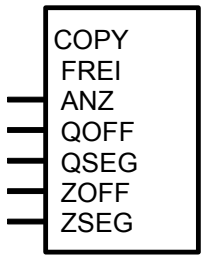
A WORD
The output code EC/ACn+i is output through the output A if the input E agrees with one of the reference values EC/ACi.
The following applies : A = 0 -> No coincidence
A = EC/ACn+i -> Coincidence

10 Memory access functions

Memory Access	serie C[tler] from pages C-291 to C-317	40	50	90	30
COPY	Copying memory areas	x	x	x	x
DWAES	Write double word in the event of value change			x	
DWOL	Read double word with enable			x	
DWOS	Write double word with enabling			x	
FDEL	Delete data segment in Flash EPROM			x	
FRD	Read data segment from the Flash EPROM			x	
FWR	Write data segment to the Flash EPROM			x	
IOR	Read byte value from I/O address			x	
IOW	Write byte value to I/O address			x	
RDB	Read binary values from historical values memory			x	
RDDW	Read double word values from historical values memory			x	
RDW	Read word values from historical values memory			x	
WAES	Write word in the event of value change			x	
WOL	Read word with enabling	x	x	x	x
WOS	Write word with enabling			x	
WRB	Write binary values into historical values memory			x	
WRDW	Write double word values to historical values memory			x	
WRW	Write word values to historical values memory			x	

COPY COPYING MEMORY AREAS

FBD



IL

```
CAL COPY
```

PARAMETERS

FREI	BINARY	%I,%O,%M,%K,%S	Block enable
ANZ	WORD	%IW,%OW,%MW,%KW	Quantity (n) of words to be copied
QOFF	WORD	%IW,%OW,%MW,%KW	Offset address of the start of the source area
QSEG	WORD	%IW,%OW,%MW,%KW	Segment address of the start of the source area
ZOFF	WORD	%IW,%OW,%MW,%KW	Offset address of the start of the target area
ZSEG	WORD	%IW,%OW,%MW,%KW	Segment address of the start of the target area

DESCRIPTION

This function block copies n words from a source memory area into a target memory area. The contents of the source memory area are not changed.
In each case, the start of the source and target memory areas is specified at the block's inputs by means of the offset and segment addresses.

FREI BINARY
Block enable
FREI = 0 -> The block is not processed
FREI = 1 -> The block is processed

ANZ WORD
Quantity n of words to be copied.
The following applies : $0 \leq n \leq +8000_H$
n = 0 :No copying
n = 8000_H : A whole segment (64 kbytes) is copied

QOFF WORD
Offset address of the start of the source area

QSEG WORD
Segment address of the start of the source area

ZOFF WORD
Offset address of the start of the target area

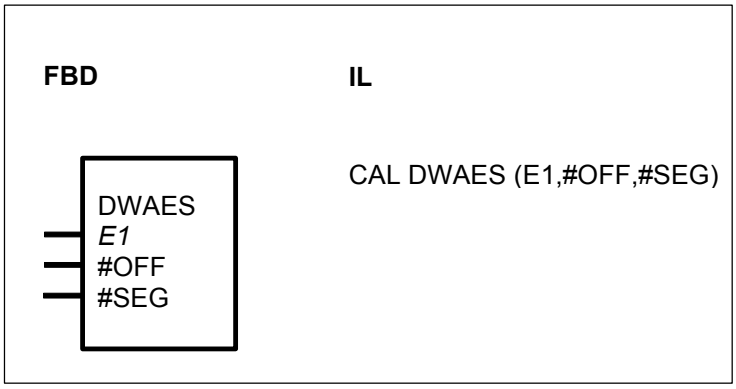
ZSEG WORD
Segment address of the start of the target area

Example for 07 KR 91 central unit :
96 words are to be copied :
- from MW 00,00 to MW 05,15 :
MW 00,00 address is : offset = $A410_H$ and segemnt = $30C2_H$
- to AW 00,00 to AW 05,15 :
AW 00,00 address is : offset = 9310_H and segemnt = $30C2_H$

	serie 90
Number of words to be copied :	96
Offset address of the source memory :	$A410_H$
Segment address of the source memory :	$30C2_H$
Offset address of the target memory :	9310_H
Segment address of the target memory :	$30C2_H$

Warning : The data addresses vary from a central unit to another.

DWAES WRITE DOUBLE WORD IN THE EVENT OF VALUE CHANGE



PARAMETERS

E1	DOUBLE WORD	%MD, %KD	Input for the operand to be read
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location to which the value of E1 must be written in the event of a change.
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location to which the

value of E1 must be written in the event of a change.

DESCRIPTION

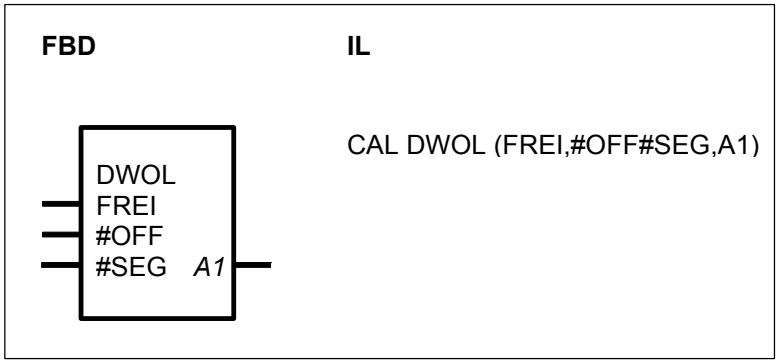
If the value of the operand at the input E1 changes in comparison with the value during previous processing of the block, the value of the operand at the input E1 is written to the specified physical address.
The physical address consists of a segment and an offset.

E1 DOUBLE WORD
If the operand at the input E1 changes, its value is written to the address specified at the inputs #OFF and #SEG.

#OFF DIRECT CONSTANT (#,#H)
The offset component of the address to be written is specified at the #OFF input. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#,#H)
The segment component of the address to be written is specified at the input #SEG. This is specified as a direct constant.

DWOL READ DOUBLE WORD WITH ENABLE



PARAMETERS

FREI	BINARY	%I, %O, %M, %S, %K	Block enable
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location whose double word value is to be read.
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location whose double word value is to be read.
A1	DOUBLE WORD	%MD	Output to which the value read is allocated.

DESCRIPTION

When a 1 signal is present at the FREI input, the value of the specified physical address is read and is allocated to the operand at the output A1.

No double word is read if there is a 0 signal at the FREI input.
The physical address consists of a segment and an offset.

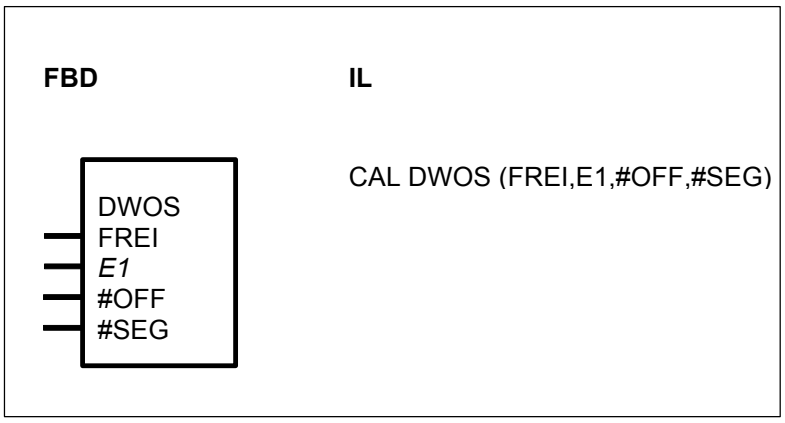
FREI BINARY
Processing of the block is enabled or disabled with the operand at the input FREI. The following applies:
FREI = 0 -> Processing disabled
FREI = 1 -> Processing enabled

#OFF DIRECT CONSTANT (#, #H)
The offset component of the address to be read is specified at the input #OFF. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#, #H)
The segment component of the address to be read is specified at the input #SEG. This is specified as a direct constant.

A1 WORD
The value read is allocated to the operand at the output A1.

DWOS WRITE DOUBLE WORD WITH ENABLING



PARAMETERS

FREI	BINARY	%I, %M, %O, %S, %K	Block enable
E1	DOUBLE WORD	%MD, %KD	Input for the operand to be written
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location to which the value of E1 must be written.
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location to which the value of E1 must be written.

DESCRIPTION

If there is a 1 signal at the input FREI, the value of the operand at the input E1 is read and then written to the specified physical address.

No double word is written if there is a 0 signal at the FREI input.
The physical address consists of a segment and an offset.

FREI BINARY
Processing of the block is enabled or disabled with the operand at the input FREI. The following applies : FREI = 0 -> Processing disabled
FREI = 1 -> Processing enabled

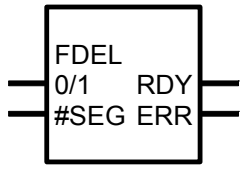
E1 DOUBLE WORD
The operand at the input E1 is read and its value is written to the address defined by the inputs #OFF and #SEG.

#OFF DIRECT CONSTANT (#,#H)
The offset component of the address to be written is specified at the input #OFF. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#,#H)
The segment component of the address to be written is specified at the input #SEG. This is specified as a direct constant.

FDEL DELETE DATA SEGMENT IN FLASH EPROM

FBD/LD



IL

```
CAL FDEL (0/1,#SEG,RDY,ERR)
```

PARAMETERS

0/1	BINARY	%I, %M, %O, %K, %S	Deletion of *one* data segment by a 0/1 edge
#SEG	DIRECT	#, #H	Data segment number in the FLASH EPROM
RDY	BINARY	%O, %M	Write procedure completed
ERR	BINARY	%O, %M	Error

DESCRIPTION

This function block deletes a data segment in the Flash EPROM. All data in this data segment are lost after deletion.

The input #SEG defines the data segment in the Flash EPROM. The deletion procedure in the Flash EPROM can take several PLC cycles. A 0->1 edge at the input 0/1 starts the deletion procedure *once*. Until the procedure has not been finished (RDY = 1), the input 0/1 is not evaluated again. After completion of the deletion procedure all function block outputs are updated. If then RDY = 1 and ERR =

0, the deletion was successful. If the outputs show RDY = 1 and ERR = 1, the data segment could not be deleted.

Important notes :
An access to the Flash EPROM is only permitted by using the function blocks FWR and FRD. It is not allowed to access the Flash EPROM by other function blocks (WOL, WOS, COPY,...).

0/1 BINARY
The input 0/1 controls the processing of the function block.

0/1 = 0 :	All outputs are set to the value of ”0”. This is not valid during a deletion procedure.
0/1= 0->1 edge :	Deletion of the data segment is started *once*. Until the procedure has not been finished (RDY = 1), the input 0/1 is not evaluated again.
0/1 = 1 :	The function block is not processed, i.e. the function block does no longer change its outputs. This is not valid during a deletion procedure.

#SEG DIRECT CONSTANT
The number of the data segment in the Flash EPROM is given at the input SEG.
In the Flash EPROM, 4 data segments are available :

#0 corresponds to data segment 0
#1 corresponds to data segment 1
#2 corresponds to data segment 2
#3 corresponds to data segment 3

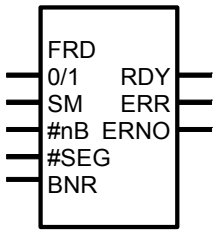
RDY BINARY
The output RDY indicates that the deletion procedure has been completed. This output has always to be considered together with the output ERR.

RDY = 1 and ERR = 0 :	The deletion procedure has been completed. The data segment has been deleted successfully.
RDY = 1 and ERR = 1 :	An error has occurred during the deletion procedure. The data segment could not be deleted.

ERR BINARY
The output ERR indicates whether or not an error has occurred during the deleting procedure. This output has always to be considered together with the output RDY.
If the data segment could not be deleted, the outputs have the following status :
RDY = 1 and ERR = 1

FRD READ DATA SET FROM FLASH EPROM

FBD/LD



IL

```
CAL FRD (0/1,SM,#nB,#SEG,BNR,RDY,ERR,ERNO)
```

PARAMETERS

0/1	BINARY	%I, %M, %O, %K, %S	Reading of *one* data set by a 0/1 edge
SM	BINARY WORD DOUBLE WORD	%I, %M, %O, %K, %S %IW, %MW, %OW, %KW %MD, %KD	Start flag of the location of the data set
#nB	DIRECT CONSTANT	#, #H	Block number of the data set
#SEG	DIRECT CONSTANT	#, #H	Data segment number in the Flash EPROM
BNR	WORD	%IW, %MW, %OW, %KW	Block number in the data segment
RDY	BINARY	%O, %M	Reading procedure completed
ERR	BINARY	%O, %M	Error indication
ERNO	WORD	%OW, %MW	Error number

DESCRIPTION

This function block reads a data set from a data segment in the Flash EPROM and stores the read data set beginning at the start flag defined by SM. The data of the data set had been stored in the Flash EPROM by the function block FWR or by the operating command FWR.

The inputs SM and #nB define which data are read from the Flash EPROM. The input #SEG defines the data segment in the Flash EPROM. The number of data, which are read from a block, depends on input SM.
Either 32 binary data or 16 word data or 8 double word data are read per block. The data of each block are secured by a checksum.
The reading procedure is carried out once by a 0/1 edge at the input 0/1. If there was no error when reading the data, the output RDY is set to ”1” and the outputs ERR and ERNO are set to ”0”. The data set is stored beginning at the defined start flag SM.
Storing the data set can take several PLC cycles.
If an error occurs during the reading procedure, RDY and ERR are both set to ”1”. The error type is indicated at the output ERNO.
The outputs RDY, ERR and ERNO are set to ”0” by a signal 0 at the input 0/1.

Important note:
An access to the Flash EPROM is only permitted by using the function blocks FWR and FRD. It is not allowed to access the Flash EPROM by other function blocks (WOL, WOS, COPY,...).

0/1 BINARY
The processing of the block is controlled by the input 0/1.
0/1 = 0 : The outputs RDY, ERR and ERNO are set to ”0”.
0/1= 0->1 edge : The reading procedure of the data set is carried out *once*.
0/1 = 1 : The block is not processed, i.e. it does not change its outputs any more.

SM BINARY / WORD / DOUBLE WORD
The first binary / word / double word flag for storing the data set is given at input SM.

#nB DIRECT CONSTANT
The block number of the data set is given at input #nB. The number of data, which are read by a block, depends on the input SM. Either 32 binary data or 16 word data or 8 double word data are read per block.
Examples:
SM = M 01,00 and #nB = 1 : Storing the data from M 01,00 to M 02,15 (1 block = 32 binary data)
SM = M 01,00 and #nB = 2 : Storing the data from M 01,00 to M 04,15 (2 blocks = 64 binary data)
SM = MW 02,00 and #nB = 1 : Storing the data from MW 02,00 to MW 02,15 (1 block = 16 word data)
SM = MW 02,00 and #nB = 2 : Storing the data from MW 02,00 to MW 03,15 (2 blocks = 32 word data)
SM = MD 03,00 and #nB = 1 : Storing the data from MD 03,00 to MD 03,07 (1 block = 8 double word data)
SM = MD 03,00 and #nB = 2 : Storing the data from MD 03,00 to MD 03,15 (2 blocks = 16 double word data)

#SEG DIRECT CONSTANT
The number of the data segment in the Flash EPROM is given at the input SEG.
In the Flash EPROM, 4 data segments are available.
#0 corresponds to data segment 0
#1 corresponds to data segment 1
#2 corresponds to data segment 2
#3 corresponds to data segment 3

BNR WORD
The number of the block in the data segment is given at the input BNR.
Valid values: 0...480

RDY BINARY
The output RDY indicates, that the reading procedure has been completed. The output has always to be considered together with the output ERR.
RDY = 1 and ERR = 0: The reading procedure has been completed. The data set is stored beginning at the definition at the input SM.

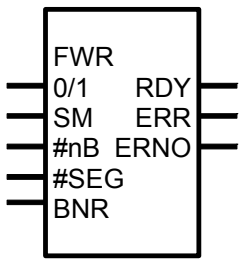
RDY = 1 and ERR = 1: An error has occurred during the reading procedure. The output ERNO indicates the error number.

ERR BINARY
The output ERR indicates whether or not an error has occurred during the reading procedure. The output has always to be considered together with the output RDY.
If there was an error, the outputs have the following status :
RDY = 1 and ERR = 1

ERNO WORD
In case of error, the output ERNO indicates the error number coded binary.
ERNO = 0: There was no error.
ERNO = 1: Block number and number of blocks is greater than 480 (bit 0 of ERNO = 1).
ERNO = 2: Data segment is greater than 3 (bit 1 of ERNO = 1).
ERNO = 4: Checksum error of read data. The data are not entered into the flag area (bit 2 of ERNO = 1).
This output has always to be considered together with the outputs RDY and ERR.

FWR WRITE DATA SET TO FLASH EPROM

FBD/LD



IL

```
CAL FWR (0/1,SM,#nB,#SEG,BNR,RDY,ERR,ERNO)
```

PARAMETERS

0/1	BINARY	%I, %M, %O, %K, %S	Saving of *one* data set by a 0/1 edge
SM	BINARY WORD DOUBLE WORD	%I, %M, %O, %K, %S %IW, %MW, %OW, %KW %MD, %KD	Start flag of the location of the data set
#nB	DIRECT CONSTANT	#, #H	Block number of the data set
#SEG	DIRECT CONSTANT	#, #H	Data segment number in the Flash EPROM
BNR	WORD	%IW, %MW, %OW, %KW	Block number in the data segment
RDY	BINARY	%O, %M	Writing procedure completed
ERR	BINARY	%O, %M	Error indication
ERNO	WORD	%OW, %MW	Error number

DESCRIPTION

The function block writes a data set to a data segment in the Flash EPROM. For this purpose, there are 4 data segments (0...3) available in the Flash EPROM. A deletion procedure (function block FDEL) always deletes a complete data segment. Each data segment consists of 481 blocks (0...480). After deletion, each of these 481 blocks can store data only **once**. If a block containing data is to be overwritten by new data, the entire data segment has to be deleted beforehand. In doing so, all data in this segment are lost.

The inputs SM and #nB define, which data are written to the Flash EPROM. The input #SEG defines the data segment in the Flash EPROM. The number of data, which can be stored in the block, depends on input SM.
Either 32 binary data or 16 word data or 8 double word data are written per block. The data of each block are secured by a checksum.
When a writing procedure of a data set is started (0->1 edge at input 0/1), the data of the data set must not be changed until the end of the writing procedure (RDY = 1).
Storing the data set in the Flash EPROM can take several PLC cycles.
A 0->1 edge at input 0/1 starts the writing procedure of the data set *once*. The input 0/1 is no longer evaluated until the storing of the data set has been completed (RDY = 1).
After completion of the writing procedure the block outputs RDY, ERR and ERNO are updated. If RDY = 1 **and** ERR = 0, the procedure was successful. If RDY = 1 **and** ERR = 1, an error had occurred. The output ERNO indicates the error type then.
After storing the data set in the Flash EPROM, the block outputs RDY, ERR and ERNO are set to ”0” by a signal 0 at input 0/1. A new 0/1 edge at input 0/1 starts a new writing procedure. Since without a previous deletion of the data segment no new data can be written to blocks which already contain data, the input BNR must point to the next free block for the next writing procedure.

Important note:
An access to the Flash EPROM is only permitted by using the function blocks FWR and FRD. It is **not allowed** to access the Flash EPROM by other function blocks (WOL, WOS, COPY,...).
For each user data set, a separate function block FWR as well as a separate function block FRD have to be planned in the PLC program.

0/1 BINARY
The processing of the block is controlled by the input 0/1.

0/1 = 0 :	The outputs RDY, ERR and ERNO are set to ”0”. This is not valid during a writing procedure.
0/1= 0->1 edge :	The writing procedure of the data set is carried out *once*. The 0/1 is no longer evaluated until the writing procedure has been completed (RDY =1)
0/1 = 1 :	The block is not processed, i.e. it does not change its outputs any more. This is not valid during a writing procedure.

SM BINARY / WORD / DOUBLE WORD
The first binary / word / double word flag for storing the data set is given at input SM.
When the writing procedure of a data set has been started (0->1 edge at input 0/1),

the data of the data set must not been changed until the procedure has been completed (RDY = 1).

#nB DIRECT CONSTANT
The block number of the data set is given at input #nB. The number of data, which are read by a block, depends on the input SM. Either 32 binary data or 16 word data or 8 double word data are read per block.
Examples:
SM = M 01,00 and #nB = 1 : Storing the data from M 01,00 to M 02,15
(1 block = 32 binary data)
SM = M 01,00 and #nB = 2 : Storing the data from M 01,00 to M 04,15
(2 blocks = 64 binary data)
SM = MW 02,00 and #nB = 1 : Storing the data from MW 02,00 to MW 02,15
(1 block = 16 word data)
SM = MW 02,00 and #nB = 2 : Storing the data from MW 02,00 to MW 03,15
(2 blocks = 32 word data)
SM = MD 03,00 and #nB = 1 : Storing the data from MD 03,00 to MD 03,07
(1 block = 8 double word data)
SM = MD 03,00 and #nB = 2 : Storing the data from MD 03,00 to MD 03,15
(2 blocks = 16 double word data)

#SEG DIRECT CONSTANT
The number of the data segment in the Flash EPROM is given at the input SEG.
In the Flash EPROM, 4 data segments are available.
#0 corresponds to data segment 0
#1 corresponds to data segment 1
#2 corresponds to data segment 2
#3 corresponds to data segment 3

BNR WORD
The number of the block in the data segment is given at the input BNR.
Valid values: 0...480

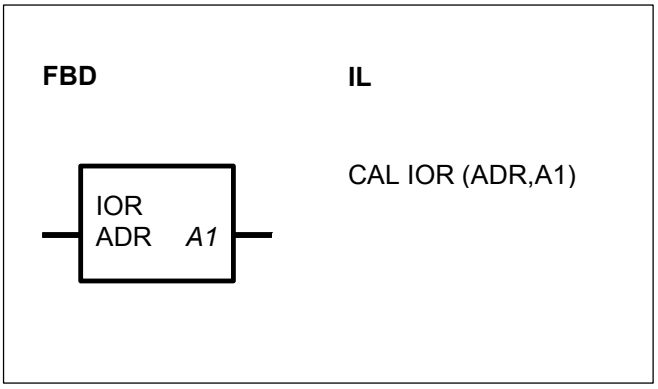
RDY BINARY
The output RDY indicates, that the writing procedure has been completed. The output has always to be considered together with the output ERR.
RDY = 1 and ERR = 0: The writing procedure has been completed. The data set is stored beginning at the definition at the input SM.
RDY = 1 and ERR = 1: An error has occurred during the writing procedure. The output ERNO indicates the error number.

ERR BINARY
The output ERR indicates whether or not an error has occurred during the writing procedure. The output has always to be considered together with the output RDY.
If there was an error, the outputs have the following status :
RDY = 1 and ERR = 1

ERNO WORD
In case of error, the output ERNO indicates the error number coded binary.
ERNO = 0: There was no error.

ERNO = 1: Block number and number of blocks is greater than 480 (bit 0 of ERNO = 1).
ERNO = 2: Data segment is greater than 3 (bit 1 of ERNO = 1).
ERNO = 4: Checksum error of read data. The data are not entered into the flag area (bit 2 of ERNO = 1).
This output has always to be considered together with the outputs RDY and ERR.

IOR READ BYTE VALUE FROM I/O ADDRESS



PARAMETERS

ADR	WORD	%IW, %OW, %MW, %KW	Address from the I/O area whose BYTE value is to be read.
A1	WORD	%OW, MW	The BYTE value read is allocated to the output A1.

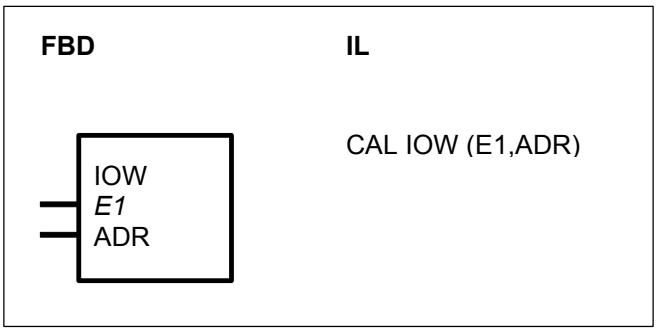
DESCRIPTION

This function block reads a byte out of the I/O area and allocates it to the operand at the output.

ADR WORD
The value of the operand at the input ADR represents the I/O address to be read.

A1 WORD
The byte read out of the I/O area is allocated to the LOW BYTE of the operand at the output A1.

IOW WRITE BYTE VALUE TO I/O ADDRESS



PARAMETERS

E1	WORD	%IW, %OW, %MW, %KW	Operand whose LOW byte is read and written into the I/O area
ADR	WORD	%IW, %OW, %MW, %KW	I/O address to which the value of the operand is written.

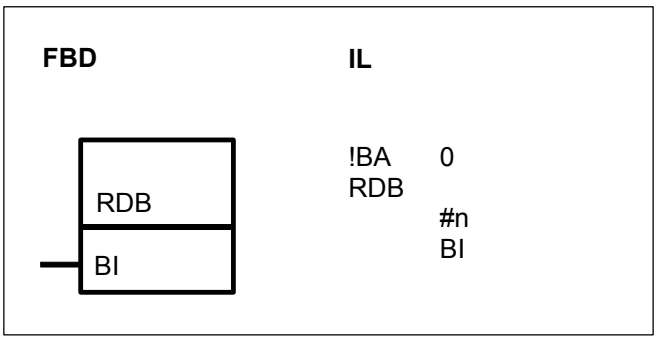
DESCRIPTION

This function block writes the byte specified at the input E1 into the I/O area.

E1 WORD
The LOW BYTE of the operand at the input E1 is written into the I/O area.

ADR WORD
The value of the operand at the input ADR represents the I/O address to which the value is written.

RDB READ BINARY VALUES FROM HISTORICAL VALUES MEMORY



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of outputs BI0 ... BIn-1
BI	BINARY	%O, %M	Output for the binary values, capable of duplication

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input BI0...BIn-1 are stored in the historical values memory of the RDB block by the affiliated WRB block. The function block RDB reads these values out of the historical values memory again at the start of the program part.

The function blocks WRB and RDB always occur in pairs.

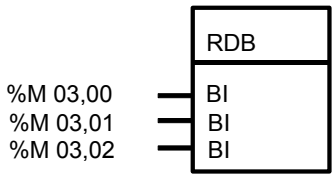
#n DIRECT CONSTANT
Only used in IL. The number of outputs BI0...BIn-1 is specified at the input #n.

BI BINARY
The BI output can be duplicated (BI0...BIn-1). The block allocates the values read out of the historical values memory to the outputs BI0...BIn-1.

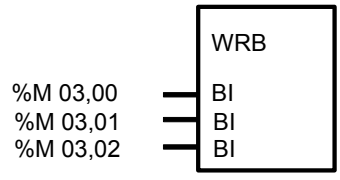
Note : The number n of BI outputs must agree with the number of inputs of the affiliated WRB block.

Example :

FBD/LD



The program part, which is used multiply and in which the variables %M 03,00 ... %M 03,02 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRB block and are read out again in the next cycle by means of the RDB block.

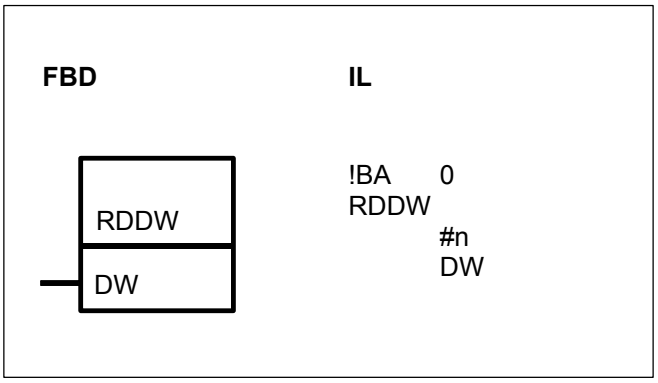


IL

```
!BA     0
RDB
        #3
        %M 03,00
        %M 03,01
        %M 03,02
```

```
!BA     0
WRB
        #0
        #3
        %M 03,00
        %M 03,01
        %M 03,02
```

RDDW READ DOUBLE WORD VALUES FROM HISTORICAL VALUES MEMORY



PARAMETERS

#n	DIRECT CONSTANT #, #H	Number of outputs
DW	DOUBLE WORD %MD	DW0 ... DWn-1 Output for the double word values, capable of duplication

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input DW0...DWn-1 are stored in the historical values memory of the RDB block by the affiliated WRB block. The function block RDB reads these values out of the historical values memory again at the start of the program part.

The function blocks WRDW and RDDW always occur in pairs.

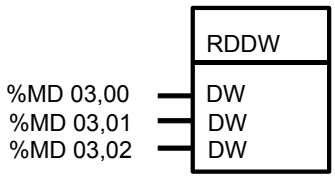
#n DIRECT CONSTANT
Only used in IL. The number of the outputs DW0...DWn-1 is specified at the input #n.

DW DOUBLE WORD
The DW output can be duplicated (DW0...DWn-1). The block allocates the values read out of the historical values memory to the outputs DW0...DWn-1.

Note : The number n of DW outputs must agree with the number of inputs of the affiliated WRDW block.

Example :

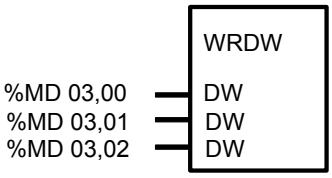
FBD/LD



The program part, which is used multiply and in which the variables %MD 03,00 ... %MD 03,02 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRDW block and are read out again in the next cycle by means of the RDDW block.

IL

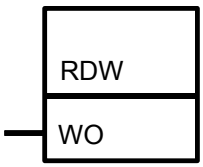
```
!BA     0
RDDW
        #3
        %MD 03,00
        %MD 03,01
        %MD 03,02
```




```
!BA     0
WRDW
        #0
        #3
        %MD 03,00
        %MD 03,01
        %MD 03,02
```

RDW READ WORD VALUES FROM HISTORICAL VALUES MEMORY

FBD



IL

```
!BA     0
RDW
        #n
        WO
```

PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of outputs WO0 ... WOn-1
WO	WORD	%OW, MW	Output for the word values, capable of duplication

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input WO0...WOn-1 are stored in the historical values memory of the RDB block by the affiliated WRB block. The function block RDB reads these values out of the historical values memory again at the start of the program part.

The function blocks WRW and RDW always occur in pairs.

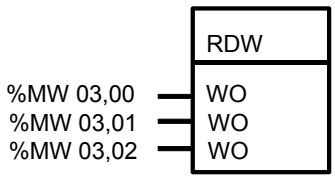
#n DIRECT CONSTANT
Only used in IL. The number of outputs WO0...WOn-1 is specified at the input #n.

WO WORD
The output WO can be duplicated (WO0...WOn-1). The block allocates the values read out of the historical values memory to the outputs WO0...WOn-1.

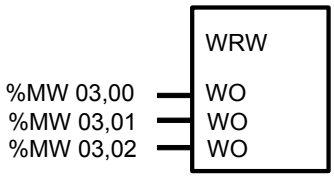
Note : The number n of WO outputs must agree with the number of inputs of the affiliated WRW block.

Example

FBD/LD



The program part, which is used multiply and in which the variables %MW 03,000 ... %MW 03,002 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRW block and are read out again in the next cycle by means of the RDW block.



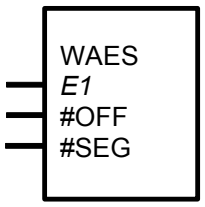
IL

```
!BA     0
RDW
        #3
        %MW 03,00
        %MW 03,01
        %MW 03,02
```

```
!BA     0
WRW
        #0
        #3
        %MW 03,00
        %MW 03,01
        %MW 03,02
```

WAES WRITE WORD IN THE EVENT OF VALUE CHANGE

FBD



IL

```
CAL WAES (E1,#OFF,#SEG)
```

PARAMETERS

E1	WORD	%IW, %MW, %OW, %KW	Input for the operand to be read
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location to which the value of E1 must be written in the event of a change.
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location to which the value of E1 must be written in the event of a change.

DESCRIPTION

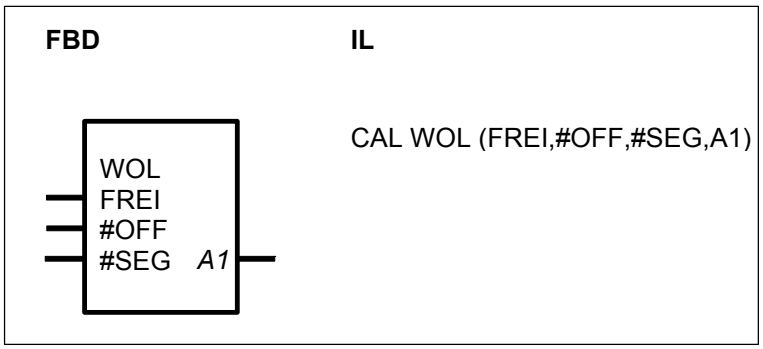
If the value of the operand at the input E1 compared to the value during previous processing of the block changes, the value of the operand at the input E1 is written to the specified physical address.
The physical address consists of a segment and an offset.

E1 WORD
If the operand at the input E1 changes, its value is written to the address specified at the inputs #OFF and #SEG.

#OFF DIRECT CONSTANT (#, #H)
The offset of the address to be written is specified at the input #OFF. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#, #H)
The segment of the address to be written is specified at the input #SEG. This is specified as a direct constant.

WOL READ WORD WITH ENABLING



PARAMETERS

FREI	BINARY	%I, %M, %O, %S, %K	Enable block
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location whose value must be read
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location whose value must be read
A1	WORD	%OW, %MW	Output to which the read value is allocated.

DESCRIPTION

When the FREI input has a 1 signal, the value of the specified physical address is read and is allocated to the operand at the output A1.
No reading and no allocation take place if the input FREI has a 0 signal.
The physical address consists of a segment and an offset.
The attainable address space is 1 MByte.

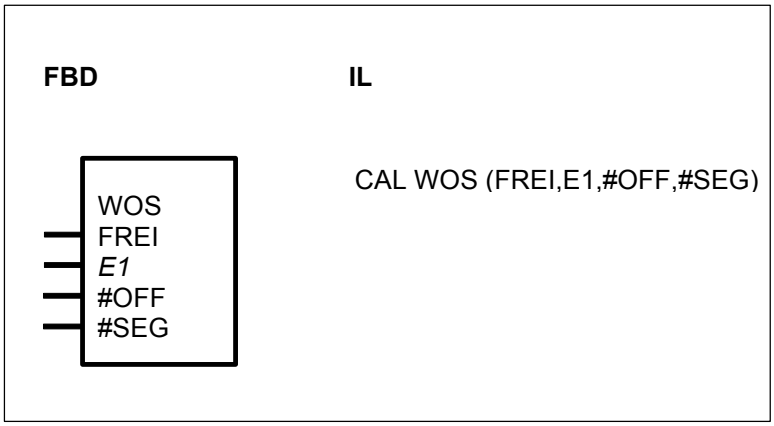
FREI BINARY
Processing of the block is enabled or disabled with the operand at the FREI input. The following applies:
FREI = 0 -> Processing disabled
FREI = 1 -> Processing enabled

#OFF DIRECT CONSTANT (#,#H)
The offset of the address to be read is specified at the input #OFF. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#,#H)
The segment of the address to be read is specified at the input #SEG. This is specified as a direct constant.

A1 WORD
The read value is allocated to the operand at the output A1.

WOS WRITE WORD WITH ENABLING



PARAMETERS

FREI	BINARY	%I, %M, %O, %S, %K	Enable block
E1	WORD	%IW, %MW, %OW, %KW	Input for the operand to be written
#OFF	DIRECT CONSTANT	#, #H	Offset address of the memory location to which the value of E1 must be written
#SEG	DIRECT CONSTANT	#, #H	Segment address of the memory location to which the value of E1 must be written

DESCRIPTION

When the input FREI has a 1 signal, the value of the operand at the input E1 is read and is then written to the specified physical address. The block is not processed if there is a 0 signal at the FREI input. The physical address consists of a segment and an offset. The attainable address area is 1 MByte.

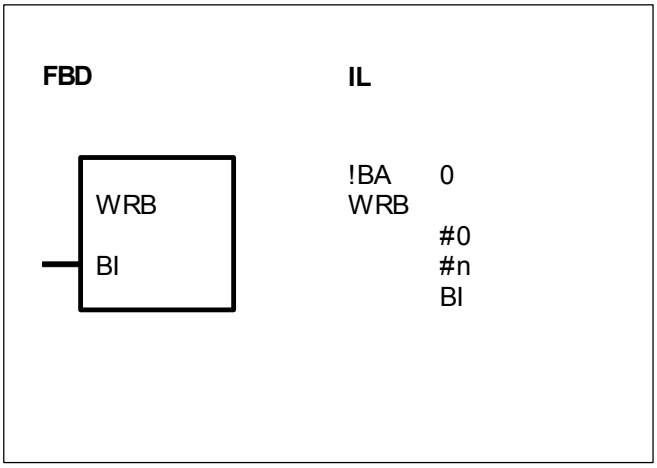
FREI BINARY
Proces sing of the block is enabled or disabled with the operand at the input FREI.
The following applies : FREI = 0 -> Processing disabled
FREI = 1 -> Processing enabled

E1 WORD
The operand at the input E1 is read and its value is written to the address defined by the inputs #OFF and #SEG.

#OFF DIRECT CONSTANT (#,#H)
The offset of the address to be written is specified at the input #OFF. This is specified as a direct constant.

#SEG DIRECT CONSTANT (#,#H).
The segment of the address to be written is specified at the input #SEG. This is specified as a direct constant.

WRB WRITE BINARY VALUES INTO HISTORICAL VALUES MEMORY



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs BI0 ... BIn-1
BI0	BINARY	%O, %M	Input for the binary values to be written, capable of duplication

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input BI0...BIn-1 are stored in the historical values memory of the affiliated RDB block by the WRB block. The affiliated function block RDB reads these values out of the historical values memory again at the start of the program part.
The function blocks WRB and RDB always occur in pairs.

#0 DIRECT CONSTANT
Only used in IL. The PLC enters the pointer to the historical values of the affiliated RDB block at this point.

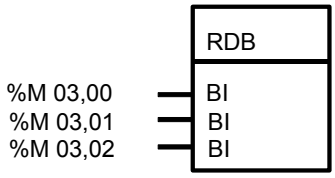
#n DIRECT CONSTANT
Only used in IL. The number of inputs BI0...BIn-1 is specified at the input #n as a direct constant.

Note : The value specified at the input #n must also agree with the number of outputs belonging to the affiliated RDB block.

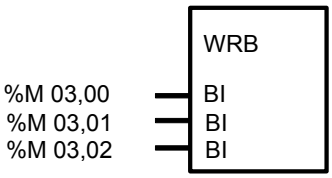
BI BINARY
The input BI is capable of duplication (BI0...BIn-1). The values of the operands specified at the inputs BI0...BIn-1 are written into the historical values memory of the affiliated RDB block.

Example

FBD/LD



The program part, which is used multiply and in which the variables %M 03,00 ... %M 03,02 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRB block and are read out again in the next cycle by means of the RDB block.

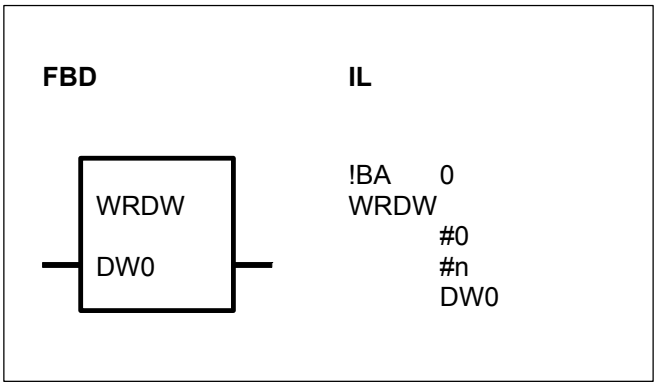


IL

```
!BA     0
RDB
        #3
        %M 03,00
        %M 03,01
        %M 03,02
```

```
!BA     0
WRB
        #0
        #3
        %M 03,00
        %M 03,01
        %M 03,02
```

WRDW WRITE DOUBLE WORD VALUES INTO HISTORICAL VALUES MEMORY



PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs DW0 ... DWn-1
DW0	DOUBLE WORD	%MD	Input for the double word values to be written, capable of

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input DW0...DWn-1 are stored in the historical values memory of the affiliated RDB block by the WRB block. The affiliated function block RDB reads these values out of the historical values memory again at the start of the program part.

The function blocks WRDW and RDDW always occur in pairs.

#0 DIRECT CONSTANT
Only used in IL. At this point, the PLC enters the pointer to the historical values of the affiliated RDDW block.

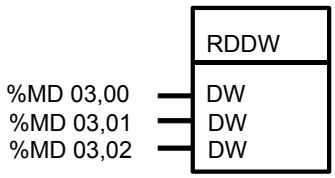
#n DIRECT CONSTANT
Only used in IL. The number of inputs DW0...DWn-1 is specified at the input #n.

Note : The value specified at the input #n must also agree with the number of outputs of the affiliated RDDW block.

D DOUBLE WORD
The input DW is capable of duplication (DW0...DWn-1). The values of the operands specified at the inputs DW0...DWn-1 are written into the historical values memory of the affiliated RDDW block.

Example

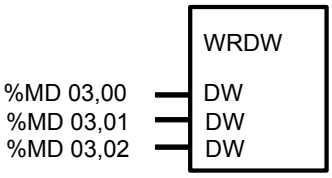
FBD/LD



IL

```
!BA     0
RDDW
        #3
        %MD 03,00
        %MD 03,01
        %MD 03,02
```

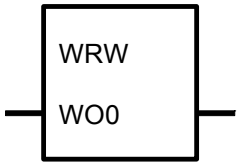
The program part, which is used multiply and in which the variables %MD 03,00 ... %MD 03,02 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRDW block and are read out again in the next cycle by means of the RDDW block.




```
!BA     0
WRDW
        #0
        #3
        %MD 03,00
        %MD 03,01
        %MD 03,02
```

WRW WRITE WORD VALUES TO HISTORICAL VALUES MEMORY

FBD



IL

```
!BA     0
WRW
        #0
        #n
        WO0
```

PARAMETERS

#n	DIRECT CONSTANT	#, #H	Number of inputs WO0 ... WOn-1
WO0	WORD	%OW, %MW	Input for the word values to be written, capable of duplication

DESCRIPTION

This function block allows to use ready-made program parts several times in one user program. It works with local variables within this part of the program. The local variables lose their validity outside of this program part.

At the end of the program part, the local variables values at the input WO0...WOn-1 are stored in the historical values memory of the affiliated RDB block by the WRB block. The affiliated function block RDB reads these values out of the historical values memory again at the start of the program part.

The function blocks WRW and RDW always occur in pairs.

#0 DIRECT CONSTANT
Only used in IL. The PLC then enters the pointer to the historical values of the affiliated RDW block at this point.

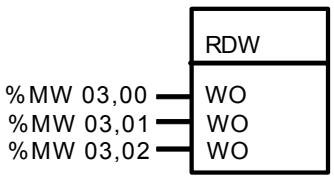
#n DIRECT CONSTANT
Only used in IL. The number of the inputs WO0...WOn-1 is specified at the input #n as a direct constant.

Note : The value specified at the input #n must also agree with the number of the outputs belonging to the affiliated block RDW.

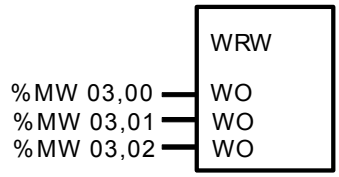
WO BINARY
The input WO0 can be duplicated (WO0...WOn-1). The values of the operands specified at the inputs WO0...WOn-1 are written into the historical values memory of the affiliated RDW block.

Example

FBD/LD



The program part, which is used multiply and in which the variables %MW 03,000 ... %MW 03,002 are needed in the next cycle, is located here. For this purpose, the values are written into the historical value memory by means of the WRW block and are read out again in the next cycle by means of the RDW block.



IL

```
!BA     0
RDW
        #3
        %MW 03,00
        %MW 03,01
        %MW 03,02
```

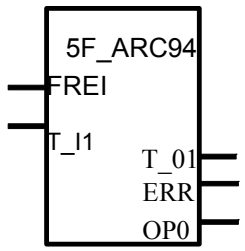
```
!BA     0
WRW
        #0
        #3
        %MW 03,00
        %MW 03,01
        %MW 03,02
```

11 Special Functions

Special functions	from pages C-317 to C-323	serie C[tler] 40	50	90	30
5F_ARC94	Arcnet for 07KT94			94	
COUNTB	Test of number of bits in a word/double word			94	
COUNTW	Fast counter on 07KT94			94	
DWWW	One double word in 2 words conversion			94	
IDENT	Identification			94	
MODMASTK	MODBUS master			94	
SETB	Set a bit in a word/double word			94	
TESTB	Test a bit in a word/double word			94	
WWDW	2 words in one double word conversion			94	

5F_ARC94 ARCNET for 07KT94

FBD



IL

CAL 5F_ARC94(FREI, T_I1, T_01, ERR, OP0)

PARAMETERS

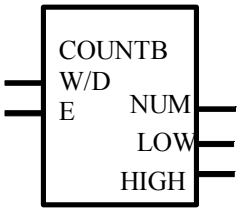
FREI	BINARY	%I, %M, %O, %S, %K	Enable block
T_I1	WORD	%IW,%OW,%KW,%MW	Offset address of the memory
T_01	WORD	,%OW, %MW	Segment address of the memory
ERR	WORD	,%OW, %MW	Error number
OP0	BINARY, WORD, DOUBLE WORD		

DESCRIPTION

This funtion is used for ARCNET communication . Refer to 07KT94 and ARCNET documentation for a complete description.

COUNTB TEST of NUMBER of BITS in a WORD/DW

FBD/LD **IL**




```
CAL COUNTB (W/D, E,
NUM, LOW, HIGH)
```

PARAMETERS

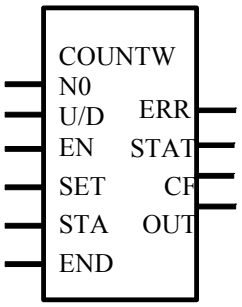
W/D	BINARY	%I, %O, %M, %K	Word or Double word selection
E	WORD	%IW,%OW,%MW,%KW	Word/Double word for test of number of bits
NUM	WORD	%OW,%MW	Number of bits in E
LOW	WORD	%OW,%MW	Position of lower bit
HIGH	WORD	%OW, %MW	Position of higher bit

DESCRIPTION

This function block defines the number of bits in a word or double word.
The position of the lower bit (0…31) is in LOW and the position of higher bit (0..31) is in HIGH.

COUNTW Fast counter on 07KT94

FBD/LD **IL**




```
CAL COUNTW(N0, U/D,
SET, STA, END, ERR,
STAT, CF, OUT)
```

PARAMETERS

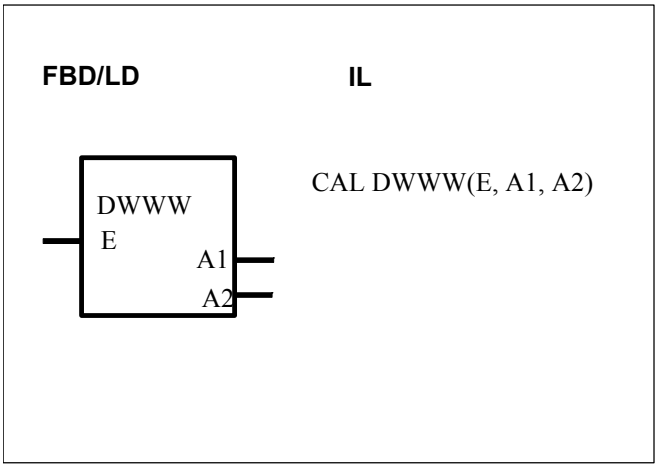
NO	WORD	%IW,%OW,%MW,%KW	Counter number 0 or 1
U/D	BINARY	%I, %O, %M, %K	0=Up / 1=Down
EN	BINARY	%I, %O, %M, %K	1=Enable counting

SET	BINARY	%I, %O, %M, %K	1=Set start value
STA	WORD	%IW,%OW,%MW,%KW	Star value
END	WORD	%IW,%OW,%MW,%KW	End value
ERR	BINARY	%O, %M	Error
STAT	WORD	%OW,%MW	Status
CF	BINARY	%O, %M	Carry Flag (1= End value reached)
OUT	WORD	%OW,%MW	Actual value output

DESCRIPTION

This function block defines the fast counter on the central unit 07KT94.
Refer to the relevant documentation for a complete description

DWWW One double word in 2 words conversion



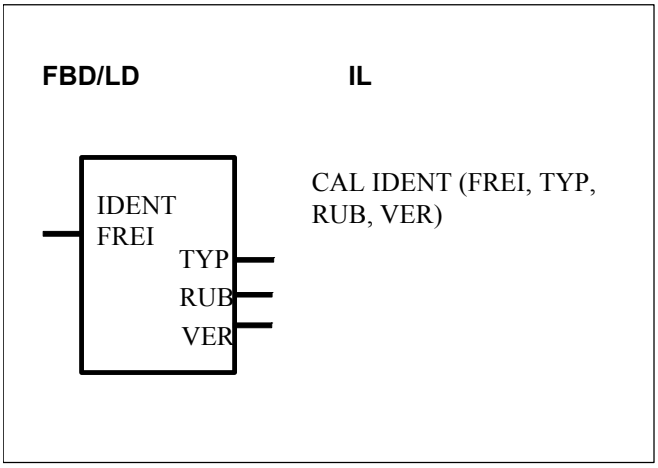
PARAMETERS

E	DOUBLE WORD	%KD, %MD	Double word to slip into 2 words
A1	WORD	%OW,%MW	Low word
A2	WORD	%OW,%MW	High word

DESCRIPTION

The Low word of the input E (double word) is set to the output A1. and the High word of the input E is set to A2

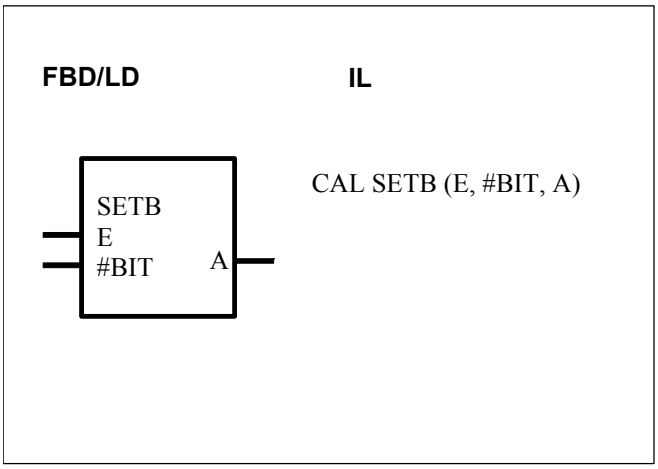
IDENT Identification



PARAMETERS

FREI	BINARY	%I, %O, %M, %K	Enable input
TYP	WORD	%OW,%MW	94 for 07KT94
RUB	WORD	%OW,%MW	Rubric =101 for R0101
VER	WORD	%OW,%MW	Sofware version =100 for V1.00

SETB Set a bit in a word/double word



PARAMETERS

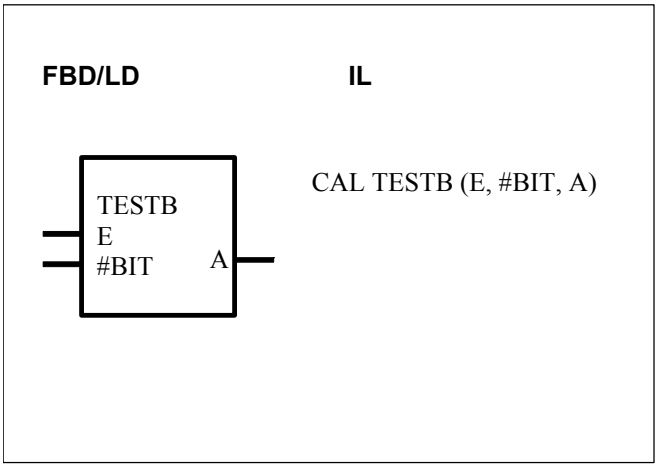
E	BINARY	%I, %O, %M, %K	Bit value
#BIT	DIRECT CONSTANT		Position of the bit into the word/double word
A	WORD	%OW,%MW	output

DESCRIPTION

A bit can be set in a word or double word
The bit number defined by #BIT in the output is set to the value of E

The value of #BIT is 0…31
The output A is a word
If #BIT is higher than 15 then the following address is used
For example :
A=%MW00.00 and #BIT=16 -> bit 0 of %MW00.01 will be set

TESTB Test a bit in a word/double word



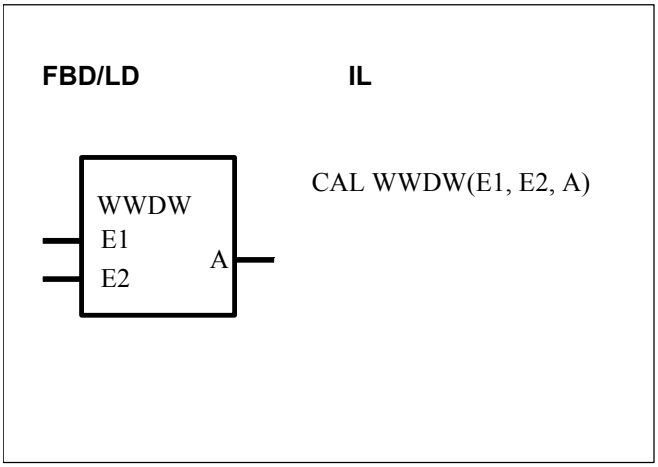
PARAMETERS

E	WORD	%KW, %IW, %OW, %MW	Word to test
#BIT	DIRECT CONSTANT		Position of the bit into the word/double word
A	BINARY	%O,%M	Result of the test

DESCRIPTION

A bit can be tested in a word or double word
The bit number defined by #BIT in the input E is set to output A
The value of #BIT is 0…31
The output A is a bit
If #BIT is higher than 15 then the following address is used
For example :
E=%MW00.00 and #BIT=16 -> bit 0 of %MW00.01 will be set on the output A

WWDW 2 words in one double word conversion



PARAMETERS

E1	WORD	%KW, %IW, %OW,%MW	E1 is the lower word in A
E2	WORD	%KW, %IW, %OW,%MW	E2 is the higher word in A
A	DOUBLE WORD	%MD	Output A

DESCRIPTION

The Low word of the input E1 and Higher word E2 is set in the double word A

12 Historical values

12.1 Definition

Some specific functions need several program cycles to be processed. An historical value is an internal register used by the function and during the function processing, to stock the function result of the (n-1) program cycle.

A maximum number of historical values can be used in a program :
- for serie 40 : 1000
- for serie 50 : 1000
- for serie 90 : 1024
- for serie 30 : 128

In case of serie 40, 50, 30 the number of timer function allowed in a program is illimited, but a maximum of 42 timers can be processed in the program at the same time.

Important note concerning ONLINE modifications :
The insertion of a function with historical values modify the historical value registers. For this reason, it is not possible to insert in a *running* program a function with historical values before other functions with historical values; the new program would work with wrong historical values.

12.2 Historical value table

	serie 40 &50	serie 90	serie 30
AMELD	-	#n + 3	-
AMELDD	-	4 + (2 * #n)	-
ASV	2	1	1
BITSU	-	9	-
BMELD	3+nb E	if #n even : 2 + #n/2 if #n odd : 2 + (#n+1)/2	if #n even : 2 + #n/2 if #n odd : 2 + (#n+1)/2
CALLUP	-	#VGW	-
CONFIO1	3		
CONFIO4	3		
CONFIO8	3		
CS31CO	1	1	1
CS31QU	-	1	1
CTU	2	-	-
CTUH	2	-	-
DT1	-	4	-
DWAES	-	2	-
ESV	2	1	1
FEHSU	-	4	-
FIFO	-	4	-

	serie 40 &50	serie 90	serie 30
HLG	-	2	-
INITS	-	1	-
INITV	-	1	-
INTK	-	3	-
LDT	-	1	-
LIFO	-	3	-
LZB	-	1	-
MAZ	-	2	-
MAZD	-	3	-
MOA	2	1	1
MODBUS	2	1	1
MODMASTK	2		
MOK	2	1	1
NPULSE	1	-	-
PDM	1	3	1
PI	3	4	6
PIDT1	5	9	-
PT1	-	2	-
RDB	-	if #n even : #n/2 if #n odd : (#n+1)/2	-
RDDW	-	2 * #n	-
RDW	-	#n	-
REC/EMAS	1	1	1
SEND/DRUCK	2	1	1
SFEHSU	-	3 + #n/16	-
SINIT	1	1	1
TOF	2	-	-
TON	2	-	-
TP	2	-	-
UHR	1	1	1
VGLEH	-	1	-
VGLUH	-	1	-
VRZ	3	2	3
VRZD	-	3	-
VVZ	-	2	-
WAES	-	1	-

13 Runtimes

13.1 Definition

The runtime (in µs) is the time for the function to be read by the program during one cycle.
The program cycle time (in ms) is then the runtime addition of all the functions in the program.

13.2 Runtime table (time in µs)

Binary functions	Controller, serie 40, serie 50	serie 90 (standard time + time per additional output)	serie 30 (standard time + time per additional output)
&	5.1	6.6 + 2.3	20 + 6
/	6	6.6 + 2.3	??
=	3.9	4.3 + 2.3	14 + 8
=1	8.8	10.9	32
=R	4.4	4.3 + 2.3	15 + 9
=S	4.45	4.3 + 2.3	15 + 9
I-	10.6	13.2	45
I+	8.95	13.2	45
MAJ	-	36	-
RS	8.8	8.6	30
SR	8.8	8.6	30

Timer functions	Controller, serie 40, serie 50	serie 90 (standard time; time in case of edge at input signal)	serie 30 (standard time; time in case of edge at input signal)
ASV	124	45; 541	97; 551
ESV	124	38; 400	97; 554
MOA	167	36; 600	97; 552
MOK	213	20; 380	97; 554
PDM	640	35	1300
TIME_W			
TOF	167	-	-
TON	210	-	-
TP	208	-	-
VVZ	-	46; 566 if 0-1 edge; 283 if 1-0 edge	-
W_TIME			

Counter functions	Controller, serie 40, serie 50	90 serie	30 serie
CTU	430	-	-
CTUH	560	-	-

VRZ	190	92	241
VRZD	-	38...113 according to the mode	-

Comparison functions, word	Controller, serie 40, serie 50	90 serie	30 serie
<	13.1	<12	??
<=	12.4	<12	??
<>	13.3	<12	<49
=?	13.3	<12	<47
>	12.4	<12	??
>=	13.1	<12	??
VGL3P	-	48	-
VGLEH	-	56	-
VGLUH	-	58...63	-

Arithmetic functions, word	Controller, serie 40, serie 50	90 serie (standard time + time per additional parameter)	30 serie (standard time + time per additional parameter)
+	12.8	<12 + 5	<46 + 18
-	13.8	<12 + 5	<49 + 21
*	31.8	<30 + 23	??
:	142	<31 + 24	<781 + 753
*: / MULDI	186	84	1430
=	8.1	7 + 5	28 + 14
BETR	23.3	28	42
COS1	-	??	-
MUL2N	36.2	30...35 + 4 per shift	72 + 15 per shift
NEG	10.6	-	-
SIN1	-	??	-
SQRT	572	55...193	-
ZUDKW	17	10	20

Logical functions, word	Controller, serie 40, serie 50	90 serie	30 serie
MASKE	-	35...38	-
SHIFT	-	37...78 according to the shift type + 1 per bit position to be shifted	-
WAND	22.7	28	46
WOR	22.7	29	46
WXOR	22.6	28	46

Program control functions	Controller, serie 40, serie 50	90 serie	30 serie

=PE	100	2	4
ABORT	-	10	-
CAL_FB		-	-
CALLUP		29	12
DI		-	-
DIN	-	<200 (KR91,KT93 : stand-alone); <1000 (KT92 : stand-alone) up to 4800 with master or slave CPU	-
DO		-	-
DOUT	-	<200 (KR91,KT93 : stand-alone); <1000 (KT92 : stand-alone) up to 4800 with master or slave CPU	-
IOCON	-	ca. 0	-
LZB	-	29	-
VTASK		-	-

CS31 functions	Controller, serie 40, serie 50	90 serie	30 serie
CONFIO1			
CONFIO4			
CONFIO8			
CS31CO	180	100	
CS31QU	27.5	60	

Communication functions	Controller, serie 40, serie 50	90 serie	30 serie (standard time + time per character)
AINIT	-	?x	-
APOLL	-	?x	-
AREC	-	?x	-
ASEND	-	?x	-
ASEND+	-	?x	-
MODBUS		-	
REC / EMAS		?	332 + 135
SEND / DRUCK		?	374 + 63
SINIT	100	25	865

Regulation functions	Controller, serie 40, serie 50	90 serie	30 serie
DT1	-	200	-
INTK	-	189...197	-
PI	1600	280	3600
PIDT1	1600	318 without DT1;	-

PT1	-	564 with DT1 106	-

Format conversion functions	Controller, serie 40, serie 50	90 serie (standard time + time per planned binary variable)	30 serie (standard time + time per planned binary variable)
BCDDUAL / BCDBIN	72.5	88	365
BCDDUALD / BCDDW	-	310	-
DUALBCD / BINBCD	107	28 (value=0) up to 162 (value=9999)	548
DUALBCDD / DWBCD	-	36...288	-
DWW	97	36...38	154
PACK4	355	56	168
PACK8	650	86	292
PACK16	1220	153	540
PACKD4	-	67	-
PACKD8	-	103	-
PACKD16	-	175	-
PACKD24	-	247	-
PACKD32	-	319	-
UNPACK4	325	61.5	164
UNPACK8	615	99.5	296
UNPACK16	1200	175.5	560
UNPACKD4	-	76	-
UNPACKD8	-	112	-
UNPACKD16	-	184	-
UNPACKD24	-	256	-
UNPACKD32	-	328	-
WDW		26	51

Comparison function, double word	Controller, serie 40, serie 50	90 serie	30 serie
<D / VKLD	107	35...36	-
=?D / VGLD	110	35...37	-
>D / VGRD	108	35...37	-

Arithmatic function, double word	Controller, serie 40, serie 50	90 serie	30 serie
+D / ADDD	114	49...52	-
-D / SUBD	116	50...55	-
*D / MULD	380	117...120	-
:D / DIVD	504	300...324	-
=D / ZUWD	40.5	31	-
BETRD		32...36	-

MUL2ND		34 + 6 per shift	-
NEGD		34	-
SQRT	604	x	-

Logical function, double word	Controller, serie 40, serie 50	90 serie	30 serie
DWAND	38	49	-
DWOR	39	35	-
DWXOR	38	35	-
MASKED	-	42	-
SHIFT	-	x	-

High order functions	Controller, serie 40, serie 50	90 serie per planned input E1	30 serie per planned input E1
ADRWA	-	44 + 8 per operand EC0...ECn-1	-
AMELD	-	57 + 31 per input E0...En-1	-
AMELDD	-	64 + 41 per input E0...En-1	-
ANAI4_20	-	180	-
AWM	-	26	-
AWT	22	30	40
AWTB	38.4	32	74
AWTD	-	41	-
BEG		32...35	147
BEGD	-	30...38	-
BITSU	-	136 + 26 per word variable	-
BMELD	1430	61 + 31 per input E0...En-1	?
DMUX	-	36	-
DMUXD	-	38	-
DWUMC	-	53 + 10 per entered ref. value	-
FEHSU	-	90 + 10 per input B0...Bn-1	-
FIFO	-	64 + 0...114	-
FKG	-	136 + 9 per interpolation point	-
HLG	-	126	-
IDLB	209	35	-
IDLm / IDL	27.4	35	63
IDSB	201	35	-
IDSm / IDS	38.6	35	99
INITS	-	40 + 2.6 per memory word	-
INITV	-	34 + 11 per additional VR1...VRn-1	-

LDT	-	x	-
LIFO	-	64 + 0...114	-
LIZU	139	19	127
MAX	426	22 + 9	74 + 64
MAXD	-	29 + 13	-
MAZ	-	36...40	-
MAZD	-	49...58	-
MIN	430	35 + 7	74 + 64
MIND	-	24 + 14	-
MUXR	-	44.5 + 7.5 per additional output A1...An-1	-
MUXRD	-	43 + 10 per additional output A1...An-1	-
NPULSE	386	-	-
SFEHSU	-	83 + 22 per input B0...Bn-1	-
UHR	430	60 to display; 200 to set	409 to display; 4697 to set
USM	-	225	-
UST	-	31	-
USTD	-	35	-
USTR	-	35	-
USTRD	-	43	-
WDEC	-	22 + 4max. per entered ref.value	-
WUMC	-	40 + 7.5 per entered comparison value	-

Memory access	Controller, serie 40, serie 50	90 serie	30 serie
COPY	258	59 + 2 per copied word	146 + 30 per copied word
DWAES	-	27	-
DWOL	-	26.5 if enable; 19 if not	-
DWOS	-	38 if enable; 19 if not	-
FDEL	-	x	-
FRD	-	x	-
FWR	-	x	-
IOR	-	24	-
IOW	-	24	-
RDB	-	48 (with time WRB) + 14 per var.	-
RDDW	-	34 (with WRDW time) + 28 per additional output	-
RDW	-	38 (with WRW time) +	-

		19 per additional output	
WAES	-	23 no reading; 33 reading	-
WOL	21.5	20 no reading; 33 reading	146 + 48 if reading
WOS	-	25 no reading; 39 reading	-
WRB	-	x	-
WRDW	-	x	-
WRW	-	x	-
Special functions	Controller, serie 40, serie 50	90 serie	30 serie
5F_ARC94	-	x	-
COUNTB	-	x	-
COUNTW	-	x	-
DWWW	-	x	-
IDENT	-	x	-
MODMASTK	-	x	-
SETB	-	x	-
TESTB	-	x	-