SINUMERIK 840C
SINUMERIK 880/880 GA2
PLC 135 WB/WB2/WD

Planning Guide 12.93 Edition

Manufacturer Documentation

SINUMERIK 840C
SINUMERIK 880/880 GA2
PLC 135 WB/WB2/WD

Planning Guide

Manufacturer Documentation

Applies to:

Control	*Software version*
SINUMERIK 840C/CE	1, 2 and 3
SINUMERIK 880 T/M	3, 4, 6
SINUMERIK 880 N	4
SINUMERIK 880 G	2
SINUMERIK 880 GA2 T/M	1

12.93 Edition

SINUMERIK® documentation

Printing history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the ”Remarks” column.

Status code in ”Remarks” *column*:

A . . . New documentation
B . . . Unrevised reprint with new Order No.
C . . . Revised edition with new status.
If factual changes have been made on the page since the last edition, this is indicated by a new edition coding in the header on that page.

Edition	**Order No.**	**Remarks**
03.90	6ZB5 410-0FC02-0BA0	**A**
03.91	6ZB5 410-0FC02-0AA1	**C**
06.92	6ZB5 410-0FC02-0AA2	**C**
11.92	6FC5197-0AA80-1BP0	**A** (replaces 6ZB5 410-0FC02-0AA2)
12.93	6FC5197-3AA80-0BP0	**C**

Preliminary Remarks

Notes for the reader

This manual is intended for the manufacturers of machine tools using SINUMERIK 840C, 880 or 880 GA2.

With every new software version, certain functions, conditions, modules etc. are either no longer possible or are added. Parts of this documentation affected by such changes are marked by the following footnotes.

The Guide describes in detail the program structure and the operation set of the PLC.

The SINUMERIK documentation is organized in four parts:

- General documentation
- User documentation
- Manufacturer documentation and
- Service documentation

The **Manufacturer Documentation** for the **SINUMERIK 840C** and **SINUMERIK 880** controls is divided into the following:

- Loose-leaf Sheets/Instruction Manual
- Interface Guide
  - Part 1: Signals
  - Part 2: Connection Conditions
- Function Macros
- Function Blocks
  - Package 0: Basic Functions
  - Package 1/2: Tool Management
  - Package 4/5: Computer Link
  - Package 6: Loading and Unloading Tools with Code Carriers
  - Package 7: Code Carriers
  - Package 8: PLC-controlled Data Input/Output
- PLC 135 WB/WB2/WD Planning Guide
- S5-HLL High-Level Language Programming

Additional SINUMERIK publications are also available for all SINUMERIK controllers (e.g. publications on the Measuring Cycles, CL800 Cycles language).

Please contact your Siemens regional office for further details.

Technical comments

- ***The PLC 135 WB is used with SINUMERIK 880 and 880 GA2.***
- ***The PLC 135 WB2 is used with SINUMERIK 840C.***
- ***The PLC 135 WD is used with SINUMERIK 840C, SW 3 and higher.***

Contents

1 Introductory Remarks

1.1 Application

The PLC 135 WB/WB2/WD is a powerful interface controller for process automation (controlling, reporting, monitoring). It is integrated in the SINUMERIK 880 / 880 GA2 and SINUMERIK 840C numerical control and controls machine-related functional sequences (auxiliary axes, tool magazines, monitors).

With the SINUMERIK 880 / 880 GA2 up to two PLCs and with the SINUMERIK 840C one PLC (programmable logic controller) can be used per control. The user programs are developed in the STEP 5 programming language or in high-level programming languages.

1.2 Programming languages

1.2.1 STEP 5 programming language

The operations available in STEP 5 enable the user to program functions ranging from simple binary logic to complex digital functions and basic arithmetic operations.

Depending on the programmer (PG) used a STEP 5 program may be written in the form of a control system flowchart (CSF), ladder diagram (LAD) or statement list (STL) (Fig. 1.1), thus enabling the programming method to be adapted to the application. The machine code (MC5) generated by the programmers is identical for all three. Depending on the programmer (PG) used, the user program can be translated from one method of representation to another by conforming to certain programming conventions (see Section RULES OF COMPATIBILITY BETWEEN LAD, CSF AND STL METHODS OF REPRESENTATION).

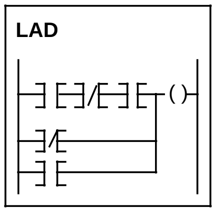
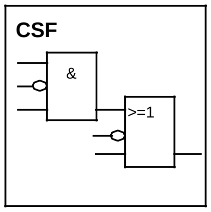
Ladder diagram	**Statement list**	**Control system flowchart**
Programming with symbols similar to those used in schematic circuit diagrams Complies with DIN 19239 (draft)	Programming with mnemonics designating functions Complies with DIN 19239 (draft)	Programming with graphic symbols Complies with IEC 117-15 DIN 40700 DIN 40719 DIN 19239 (draft)
LAD 	**STL** A I AN I A I ON I O I = Q	**CSF** 

Fig. 1.1 Methods of representing the STEP 5 programming language

1.2.2 High-level language programming

The tasks of the PLC in a complex machine tool have grown enormously in the past. A high-level language would be the optimum solution especially when parts of programs contain many branches, jumps and comparisons. It is a quick way of formulating logical expressions which must also be structured so that modules are comprehensible.

Structogram generators, which support structured programming with graphics, are also available for program development using high-level languages. A structogram generator shows the logical program structure in diagrammatical form and generates the source code automatically.

For further information on high-level language programming see the Planning Guide ”PLC 135 WB S5-HLL”!

1.3 Programming

1.3.1 Program structure

The PLC software comprises the system program, the basic program and the user program. The system program contains all statements and declarations for internal system functions. The basic program has a flexible interface to the system program, it consists of technology-specific functions and basic functions (e. g. generation of data blocks, NC-PLC interface initialization, signal interchange with I/O modules). The basic program also contains pretested function blocks written in STEP 5 and assembled to form function macros.

The system program and the basic program are supplied on EPROM submodules (in the case of SINUMERIK 840C SW 3 and higher on hard disk or tape) and must not be modified in any way. The user program is the total of all statements and declarations**/**data programmed by the user.

The structure of the PLC 135 WB **/** WB2 **/** WD makes structured programming essential, i. e. the program must be divided into individual, self-contained sections called blocks. This method offers the following advantages:

- Easy, lucid programming, even of large programs
- Easy standardization of program sections
- Simple program organization
- Fast, easy modification
- Simple program testing
- Easy start-up

A number of block types, each of which is used for different tasks, is available for structuring the user program:

- Organization blocks (OBs)
  The OBs serve as interface between operating system and user program.
- Program blocks (PBs)
  The PBs are used to break the user program down into technologically oriented sections.

- Function blocks (FBs and FXs)
  The FBs are used to program frequently recurring complex functions (such as individual controls, reporting, arithmetic and PID control functions).
- Sequence blocks (SBs)
  SBs are special forms of program blocks used primarily for processing sequencers.
- Data blocks (DBs and DXs)
  DBs are used for storing data or texts, and differ in both function and structure from all other block types.

With the exception of the organization blocks, the maximum number of programmable blocks of each type is 255. The number of organization blocks may not exceed 64; of these, only OB 1-OB 7 are serviced by the operating system (see Section ”Organization blocks”).

The programmer stores all programmed blocks in arbitrary order in program memory (Fig. 1.2).

1.3.2 Program organization

The manner in which the program is organized determines whether and in what order the program, function and sequence blocks are executed. The order in which these blocks are invoked is stipulated by programming the relevant calls (conditional or unconditional) in organization blocks (see Section ”Programming the cyclic program”).

Like the other blocks, the organization blocks are stored in user memory.

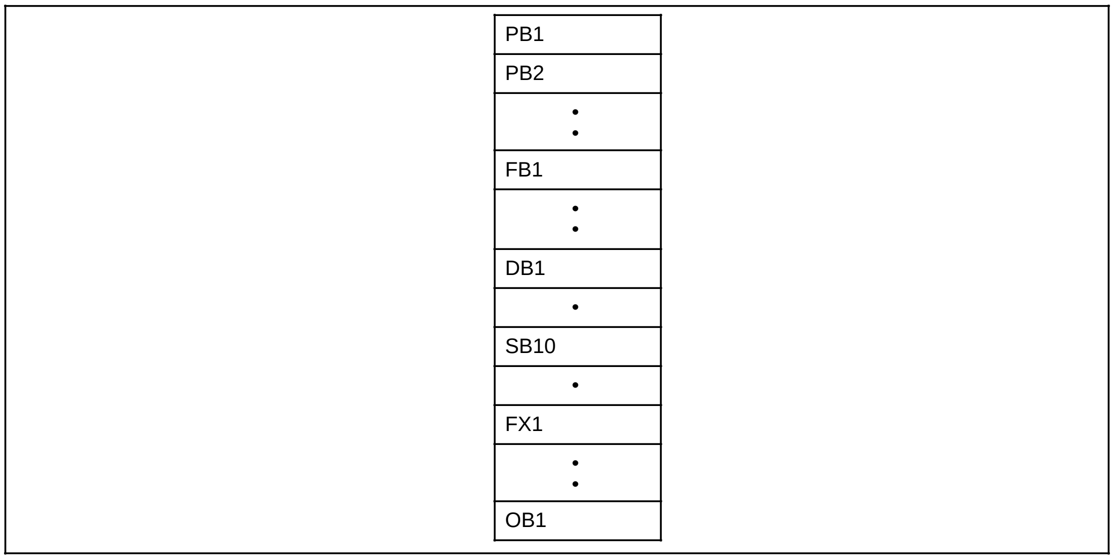


Fig. 1.2 Storing the blocks in arbitrary order in program memory

Different organization blocks are provided for various methods of program execution (see Section ”Organization blocks”).

Organization, program, function and sequence blocks can invoke other program, function and sequence blocks. The user program cannot call organization blocks. The maximum permissible nesting depth is 62 blocks (all levels, Fig. 1.3), not including an accompanying data block, if any.

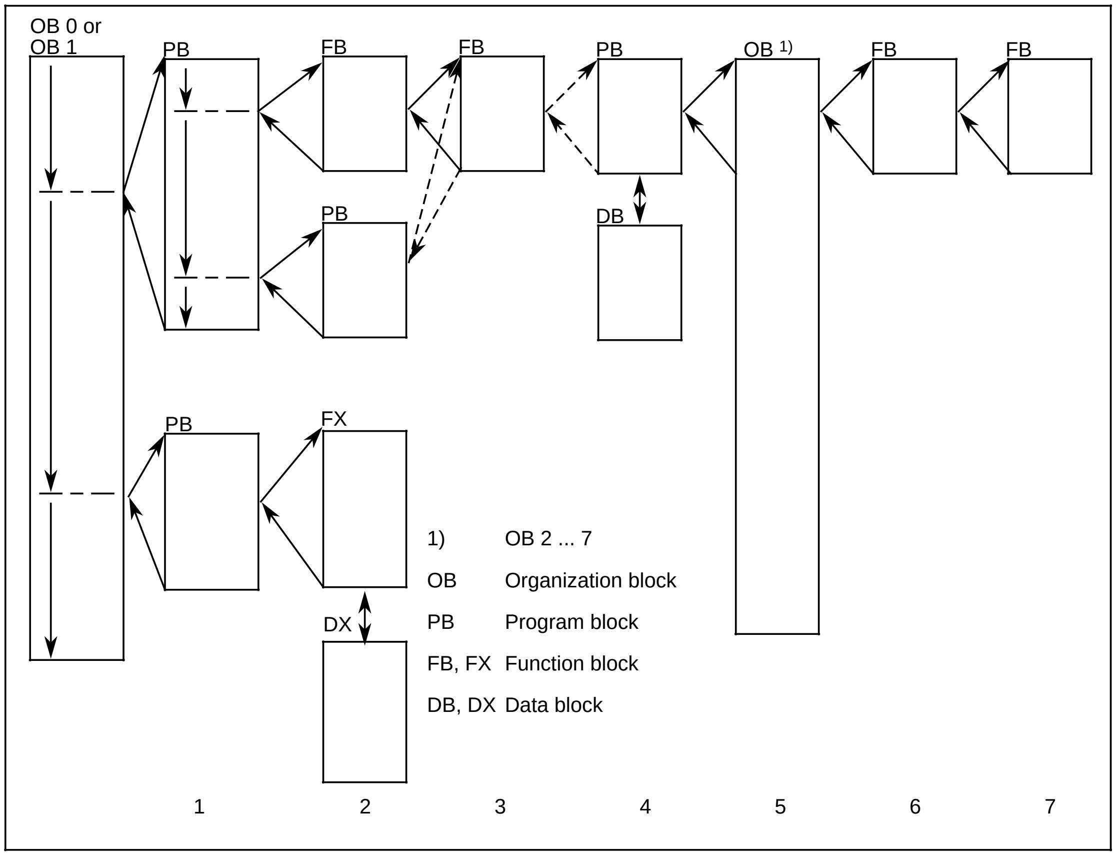


Fig. 1.3 Typical program organization in STEP 5; nesting depth 62

1.3.3 Program processing

The user program can be processed cyclically. Interrupt processing with organization blocks is also possible.

- Cyclic program processing
  For cyclic program processing organization block OB 1 is available. This block is processed cyclically and calls up the blocks programmed in it.
- Interrupt-controlled processing
  Cyclic program processing can be interrupted for interrupt servicing. For this, organization blocks OB 2 to OB 7 are available (Fig. 1.4). OB 2 and OB 5 can only be executed in special mode.

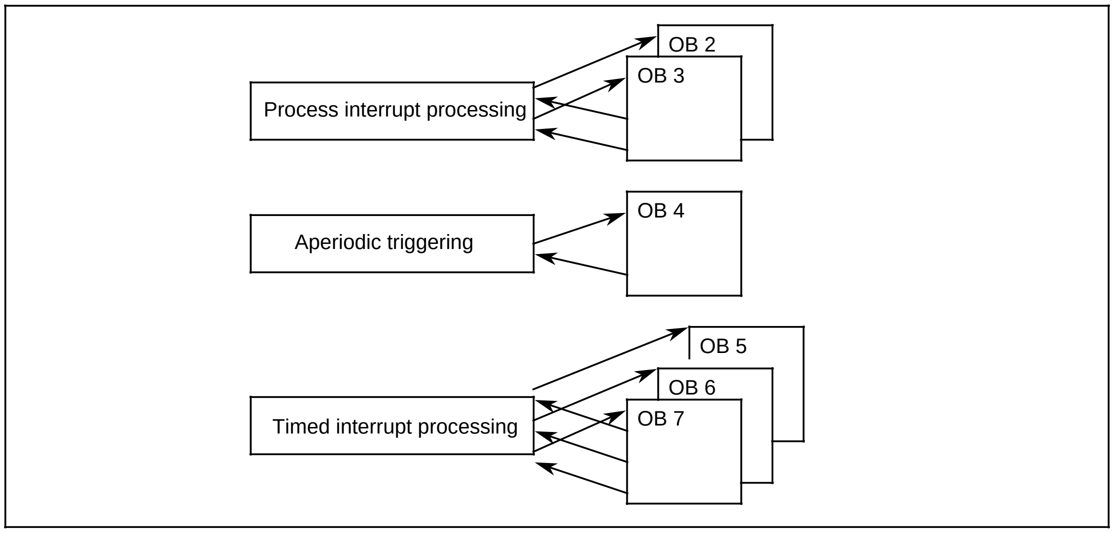


Fig. 1.4 Types of interrupt processing

There are 2 modes available to the user for interrupting cyclic processing:

- Normal mode
  Here interruption of cyclic processing is only possible at the block limits (Fig. 1.5).
  If you are working in normal mode, make sure that non-interruptible programs do not require longer than 10 ms. Otherwise errors can occur:
  - The MC5 times become imprecise.
  - If the 10 ms limit is exceeded by a lot, the mutual NC-PLC monitoring can respond (message "PLC CPU failed").
- Special mode
  In special mode the user program can be interrupted after every MC 5 command.

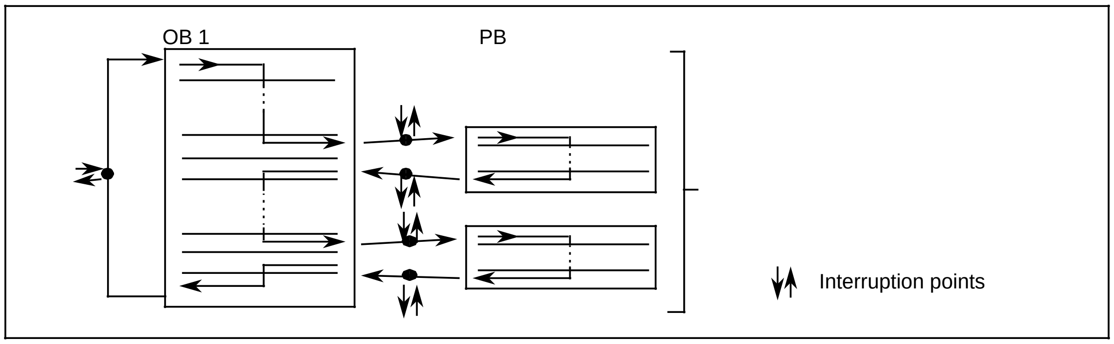


Fig. 1.5 Program processing in normal mode

1.4 Differences between the PLC 135 WB2 and the PLC 135 WD

The table below shows the essential differences:

PLC 135 WB2	PLC 135 WD
Start-up switch (RUN, STOP, GENERAL RESET)	No start-up switch (the functions WARM RESTART, COLD RESTART, BOOTSTRAP and GENERAL RESET can only be executed from the programming unit and/or the operator panel).
EPROM submodules for PLC system program and user program	No EPROM submodules. The PLC system program and user program are booted from the hard disk. The RESTART Eproms are contained in the module.

The PLC user program is to be found on the hard disk in file ANW_PROG. The file can be displayed as follows:

- Select SERVICES operating area
- Press DATA MANAGEMENT softkey
- First select directory PLC in the user branch and then subdirectory PROGRAMS.

2 Program Blocks

2.1 Programming program blocks

The information presented in this Section applies to the programming of organization, program and sequence blocks. These three block types are all programmed in the same way (see Section ”Data blocks” and Section ”Function blocks”). Program, organization and sequence blocks can be programmed in all three STEP 5 modes of representation (STL, LAD, CSF) using the basic operations.

The first step in programming a program block (PB) is the specification of a program block number between 0 and 255 (example: PB 25). This is followed by the actual control program, which is terminated with a "BE" statement.

An S5 block comprises two parts:

- Block header
- S5 operations (block body)

The block header, which the programmer generates automatically, takes up five words in program memory.

A program block should always be a self-contained program. Logical links to other blocks serve no practical purpose.

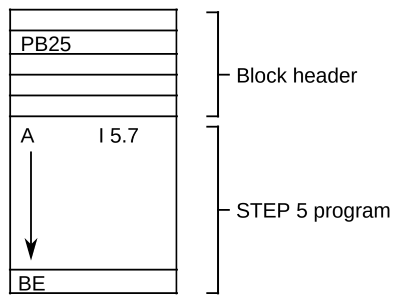


Fig. 2.1 Structure of a program block

2.2 Calling program blocks

Block calls are used to release the blocks for execution (Fig. 2.2). These block calls can be programmed only in organization, sequence, program or function blocks. Organization blocks may not be invoked by the user program, with the exception of OB180. A block call is comparable with a "subroutine branch", and may be both conditional and unconditional.

A "BE" statement is used to return to the block that contained the block call. No further logic operations can be carried out on the RLO in the "new block" following a block call or a "BE". The RLO (result of the logic operation) is passed to the "new block", and can be evaluated there.

Unconditional call: JU xx

The program block is executed without regard to the RLO.

Conditional call: JC xx

The program block is executed in dependence on the RLO.

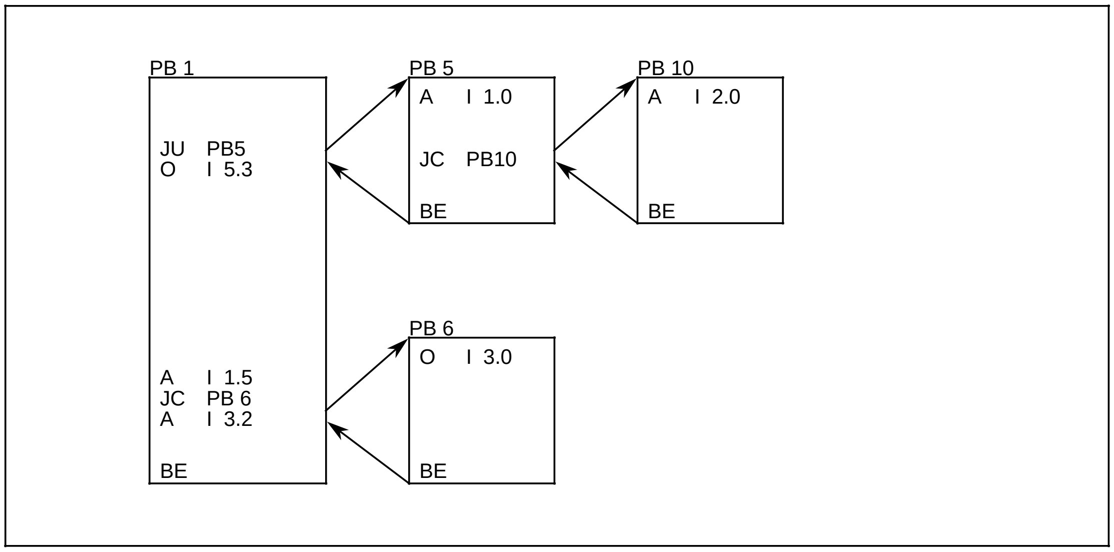


Fig. 2.2 Block calls for enabling execution of a program block

Note:

On the SINUMERIK 880 GA2, SW1, and SINUMERIK 840C, OB 19 is called if a non-existent program block is called.

3 Data Blocks

3.1 Programming data blocks

The data required by the user program is stored in data blocks (DBs and DXs). No STEP 5 operations are programmed in these blocks.

Data can be of the following types:

- Arbitrary bit patterns, e. g. for plant status indications
- Numbers (hexadecimal, binary, decimal), e. g. for times and results of arithmetic operations
- Alphanumeric characters, e. g. for message texts

Generation of a data block on the programmer begins by specifying a data block number between 1 and 255. Each data block (see example: DX 99 in fig. 3.1) may comprise as many as 2043 data words (of 16 bits each). DW 0 to DW 255 can be addressed via load and transfer operations. Data words > 255 can be addressed using O B180. The data must be entered by word, beginning with data word 0; data word 0 (DW 0) should not be used, however, as certain function blocks employ it as a buffer.

One word is reserved in program memory for each data word. The programmer also generates a block header for each data block; the header takes up five words in program memory.

Data blocks DB 2, 3 and 4 are interface blocks between the NC and the PLC 135 WB.The programmer (PG) prevents deletion and modification of these blocks.

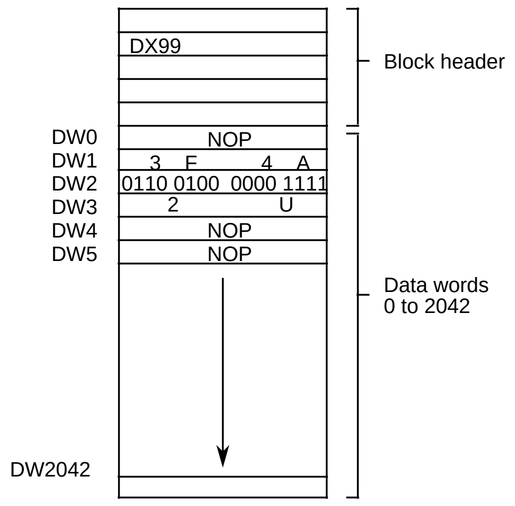


Fig. 3.1 Structure of a data block

3.2 Calling data blocks

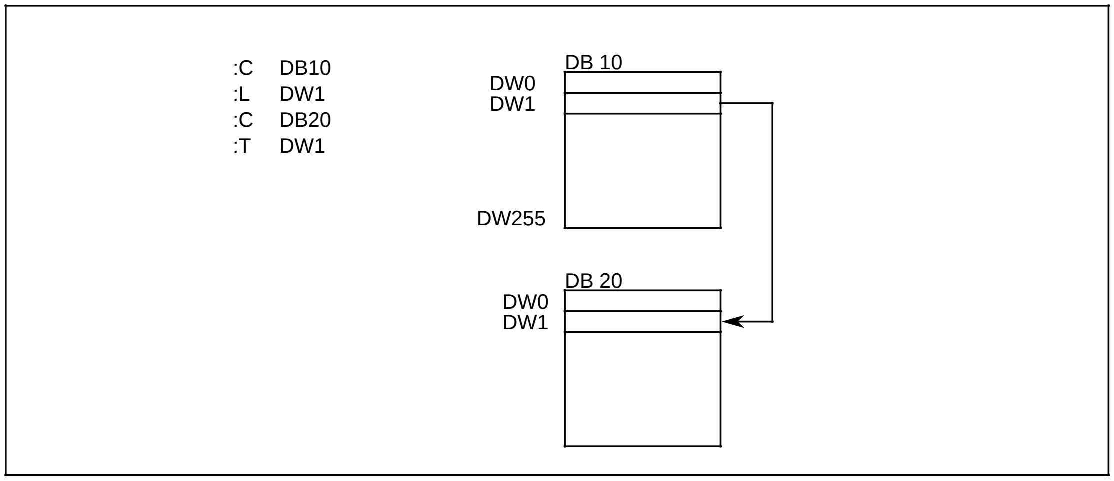
Data blocks can be called unconditionally only. Once called, a data block remains in force until the next is invoked.

User data blocks must not conflict with those required by the system.

A data block call can be programmed in an organization, program, function or sequence block. The "C DB xxx" or "CX DX 200" command calls a data block.

Example 1:

Transferring the contents of data word 1 of data block 10 to data word 1 of data block 20 (Fig. 3.2).



```
:C    DB10
:L    DW1
:C    DB20
:T    DW1
```

Fig. 3.2 Calling a data block

When a program block in which a data block is addressed calls another program block that addresses another data block, the latter is valid only in the program block that was called. The original data block is again valid following return to the calling block
(Fig. 3.3).

Example 2:

Data block 10 is called in program block 7, and the data in this data block are subsequently processed.

Program block 20 is then called and executed. Data block 10 is still valid. Only when data block 11 has been opened is the data area changed. Data block 11 is now valid until program block 20 has terminated.

Data block 10 is once again valid following the return to program block 7.

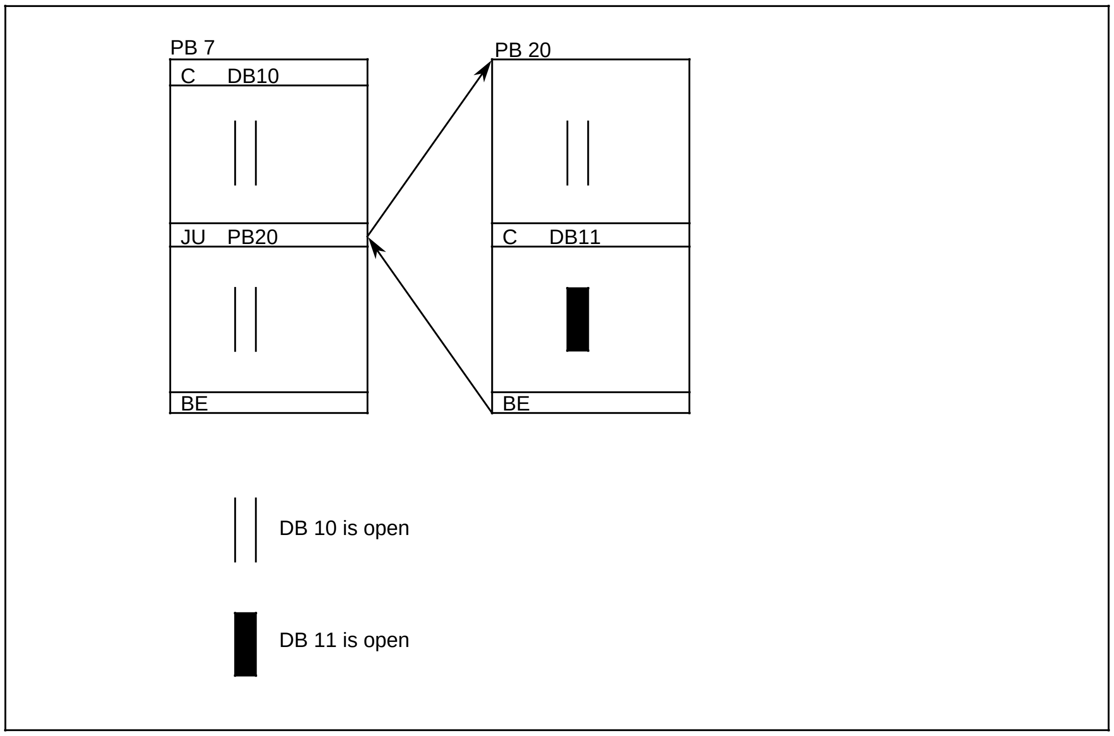


Fig. 3.3 Validity range of a selected data block

Notes:

- If a non-existent data word or a data word of an unopened data block is addressed, the PLC goes into the stop state.
- With SINUMERIK 880 GA2, SW1, and SINUMERIK 840C, OB19 is called if a non-existent data block is called.

3.3 Processing data words greater than data word 255

The size of data blocks was increased from 256 words to 2K words. This gives the user 2043 words as a data field per data block.

The corresponding DBs can be generated using the function macro FB 11 (EINR_DB). The system block OB 180 is provided for addressing the additional data words with STEP 5 statements.

Function description of the OB 180:

OB 180 is used to address data words and data bytes in data blocks over 256 long (up to 2043 words). Only 256 data words (data words 0 to 255) can be addressed using STEP5 statements. In order to address the remaining data words of a longer data block, the initial address of the open data block is shifted in OB 180. The number of data words by which the initial address is shifted must be transferred to ACCU 1 (e.g. with L KF + 12).

The OB 180 can be invoked with the commands JU OB or JC. The data block length is reduced by the corresponding number of words so that a validity check can still be carried out when the data block is written to. If the shift specified is bigger than the data block length or if no data block is open, no shift is performed and the RLO is set to 1. RLO is reset to 0 on a legal shift.

It is possible to switch back to the original initial address by opening the data block again.
If OB 180 is called several times in a row, the shifts are added up until the data block is reopened. It is not possible to shift in the negative direction.

Example:

```
DB222        DW0  :  KH=0000
             DW1  :  KH=1111
             DW2  :  KH=2222
             DW3  :  KH=3333
             DW4  :  KH=4444
             DW5  :  KH=5555
             DW6  :  KH=6666
             DW7  :  KH=7777
             DW8  :  KH=8888
             DW9  :  KH=9999
```

```
Shift by 5 words:                      L     KF +05
                                       JU    OB180
```

After shifting:

```
DB222        DW0  :  KH=5555
             DW1  :  KH=6666
             DW2  :  KH=7777
             DW3  :  KH=8888
             DW4   :   KH=9999
```

4 Function Blocks

4.1 General remarks

Function blocks are used to implement frequently recurring or extremely complex functions. The PLC 135 WB **/** WB2 **/** WD interface control also permits function blocks of type FX to be programmed in addition to the previously used function block type FB. The handling of both ty-pes is identical.

Functions blocks (FBs and FXs) are as much a part of the user program as, for example, program blocks. There are three basic differences between function blocks and organization, program or sequence blocks:

- Function block can be initialized, i. e. a function block's formal parameters can be replaced by the actual operands with which the function block is called.
- In contrast to organization, program and sequence blocks, an extended operation set comprising the STEP 5 supplementary operations can be used to program function blocks, and only function blocks.
- The program in a function block can be generated and logged in statement list form only.

The function blocks in a user program represent complex, self-contained functions. The user can program function blocks in the STEP 5 programming language or purchase them from Siemens as a software product. In addition, a number of pretested, technology-specific function blocks can be assembled to form function macros and linked into the basic program. The user can call these macros as he would a function block, but he cannot modify them. These blocks are written in assembly language and are also referred to as "resident" or "integral function blocks".

4.2 Structure of function blocks

A function block comprises a block header, name and parameter declaration, and the block body (Fig. 4.1).

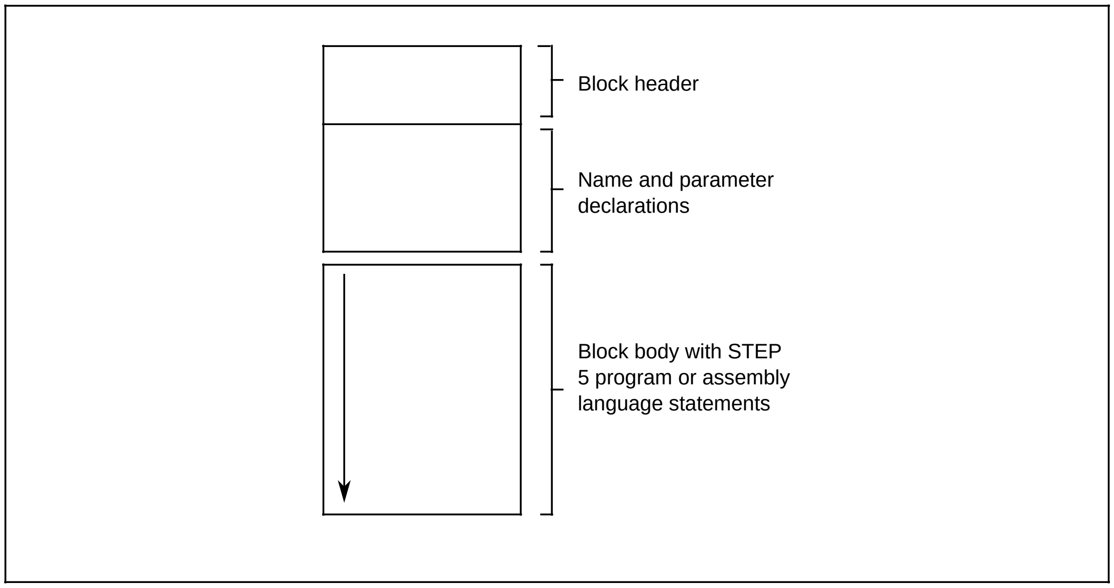

Block header
Name and parameter declarations
Block body with STEP 5 program or assembly language statements

Fig. 4.1 Structure of a function block

4.2.1 Block header

The block header contains all the information which the programmer needs in order to display the function block in graphic form and check the operands when the function block is initialized. The user must enter the header (using the programmer) before programming the function block.

4.2.2 Block body

The block body contains the actual program, i. e. describes the function to be executed in the STEP 5 language. Only the block body is processed when the function block is called.

The programmer echoes the block name and parameter declaration when integral assembly-language function blocks are called.

When the "first executable statement" in the block body is the "ASM" STEP 5 command (switch to assembly code), the processor executes the subsequent assembly language statements immediately.

4.3 Calling and initializing function blocks

Function blocks (FBs, FXs) are present only once in memory. They can be called once or more than once by a block, and different parameters can be used for each call.

Function blocks are programmed or called by specifying a block number (FB 0 to 255 or FX 0 to 255).

A function block call can be programmed in an organization, sequence or program block or in another function block. A call comprises the call statement and the parameter list.

Note:

With SINUMERIK 880 GA2, SW1, and SINUMERIK 840C, OB19 is called if a non-existent data block is called.

4.3.1 Call statement

JU FBn, DO FXn	Unconditional call
JC FBn, DOC FXn	Conditional call

Unconditional call:

The function block is executed without regard to the RLO.

Conditional call:

The function block is executed only if the previous result of logic operation is zero (RLO = 1).

4.3.2 Parameter list

The parameter list immediately follows the call statement (Fig. 4.2), and defines all input variables, output variables and data. The parameter list may contain no more than 40 variables.

The variables from the parameter list replace the formal parameters when the function block is executed. The programmer (PG) monitors the order in which the variables are entered in the parameter list.

The programmer automatically generates, but does not display, the jump statement that follows the FB call.

The FB call reserves two words in program memory, and each parameter one additional word.

The identifiers for the function block's inputs and outputs and the name of the function block are displayed on the programmer when the user programs the function block.
This information is in the function block itself. It is therefore necessary that all required function blocks either be resident as function macros in the PLC's basic program, be transferred to the program diskette, or be entered directly into the programmable controller's program memory before function block programming can begin (for details, refer to the Operator' Guide).

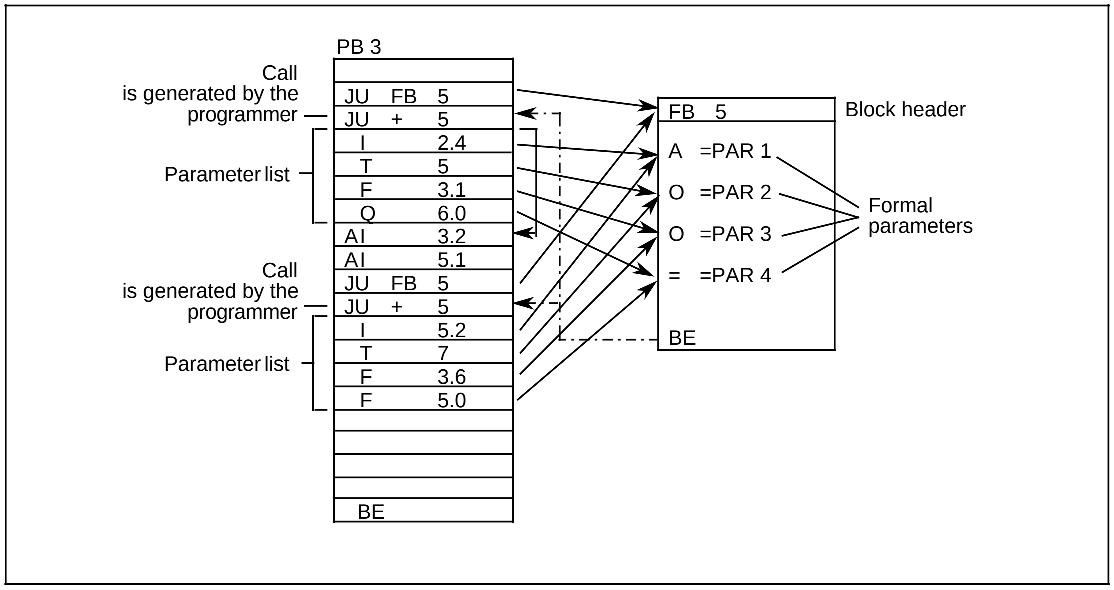


Fig. 4.2 Calling a function block

4.4 Programming function blocks

In keeping with its structure, a function block is generated in two parts:
the block header and the block body.

The block header must be entered before the block body (STEP 5 program). The block header contains

- The library number
- The name of the function block
- The formal operands (the names of the block parameters)
- The block parameters

4.4.1 Library number

The library number can be a number between 0 and 65535. The library number of a function block is unconnected with its symbolic or absolute parameters.

A library number should be assigned only once to permit unique identification of a function block. Standard function blocks have a product number.

4.4.2 Name of the function block

The name that identifies the function block may comprise no more than eight characters, the first of which must be a letter. The name is not identical with the symbolic plant identifier.

4.4.3 Formal operand (block parameter name)

A formal operand may comprise no more than four characters, the first of which must be a letter. A maximum of 40 parameters can be programmed per function block.

STL

```
      :JU     FB 201
NAME  :E-ANTR ────────  Name of the function block
ZU-E  :       DW1
RME   :       I   3.5
ESB   :       F   2.5
UEZ   :       T   2
ZEIT  :       KT010.1
ZU-A  :       DW2
BEA   :       Q   2.3
LSL   :       Q   6.0
```

Formal operands (names of the block parameters)

LAD/CSF

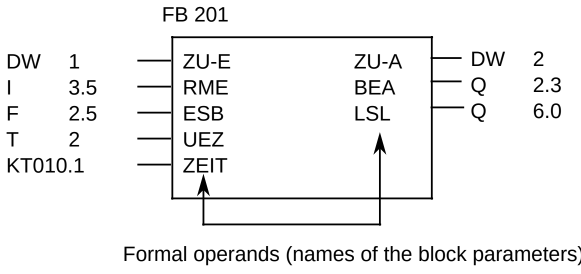


Fig. 4.3 Sample function block call

4.4.4 Block parameter types

A block parameter may be of type "I", "Q", "D", "B", "T", or "C".

I = Input parameter
Q = Output parameter
D = Data
B = Block
T = Timer
C = Counter

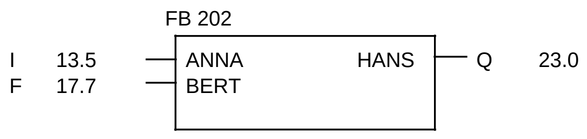
In graphic representation, parameters of type "I", "D", "B" and "C" are shown to the left of the function symbol and those of type "Q" to the right.

Operations to which parameters are to be assigned (substitution operations) are programmed in the function block with formal operands. The formal operands may be addressed at various locations within the function block.

STL

```
        :JU     FB 202
NAME    :EXAMPLE
ANNA    :       I   13.5
BERT    :       F   17.7
HANS    :       Q   23.0
```

LAD/CSF



Program in the function block

```
NAME    :EXAMPLE
ID      :ANNA         I/Q/D/B/T/C:    I   BI/BY/W/D:     BI
ID      :BERT         I/Q/D/B/T/C:    I   BI/BY/W/D:     BI
ID      :HANS         I/Q/D/B/T/C:    Q   BI/BY/W/D:     BI

        :A    =   ANNA
        :A    =   BERT
        :=    =   HANS
```

Formal operand   Parameter type   Parameter type

Program executed

```
        :A      I   13.5
        :A      F   17.7
        :=      Q   23.0
```

Actual operand

Fig. 4.4   Example: Calling a function block

4.4.5 Block data type and permissible actual operand

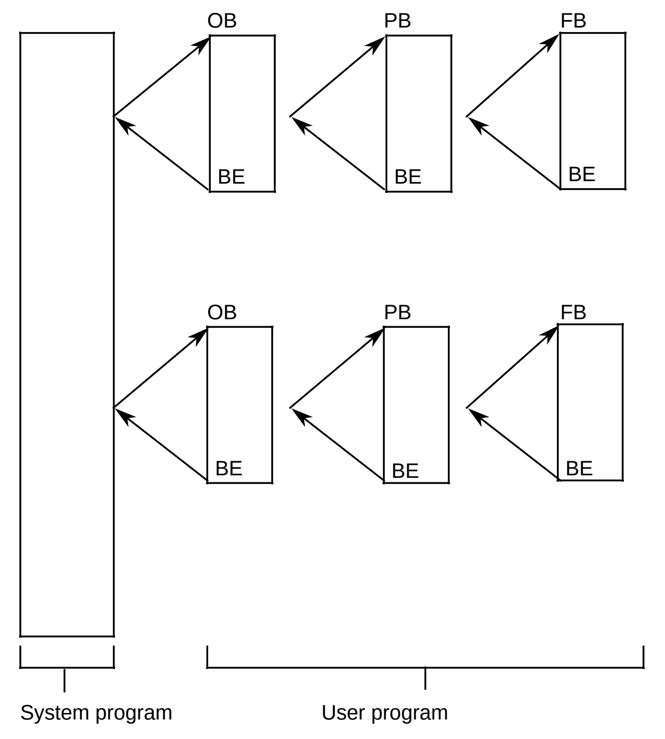
Parameter type	Data type	Permitted actual operands
I, Q	BI for an operand with bit address	I n.m inputs Q n.m output F n.m flags
	BY for an operand with byte address	IB n input bytes QB n output bytes FB n flag bytes DL n data byte left DR n data byte right PB n peripheral bytes
	W for an operand with word address	IW n input words QW n output words FW n flag words DW n data words PW n peripheral words
	D for an operand with double word address	ID n input double words QD n output double words FD n flag double words DD n data double words
D	KM for a binary pattern (16-digit) KY for two absolute numbers bytewise in the range of 0 to 255 KH for a hexadecimal pattern up to four digits KS for a symbol (max. 2 alphanumeric characters) KT for a time value (BCD coded) with time grid 1.0 to 999.3 KC for a count value (BCD coded) 0 to 999 KF for a fixed-point number -32768 to +32767 KG for a floating-point number	Constants
B	data type definitions are not permitted	DB n data blocks, the command B = "Parameter" n is executed FB n function blocks (only permissible without parameters) FX n are called unconditionally with the command JU FB n or BA FX n PB n program blocks are called unconditionally (JU..n) SB n Sequence blocks are called unconditionally (JU..n)
T	data type definitions are not permitted	T timer; the time must be programmed as a data or programmed as a constant in the function block
C	data type definitions are not permitted	C counter; the count value must be programmed as a data or programmed as a constant in the function block

5 Organization Blocks

5.1 General remarks

The organization blocks form the interface between the system program and the user program.

The organization blocks (OBs) are as much a part of the user program as are program blocks, sequence blocks and function blocks, but only the system program can invoke them. A user can only program organization blocks; he cannot invoke them (with the exception of OB 180) (Fig. 5.1).



OB	Organization block
PB	Program block
FB	Function block

Fig. 5.1 PLC program

Appropriate programming of the organization blocks enables the following:

- Cyclic execution
  (see Section PROGRAMMING THE CYCLIC PROGRAM)
- Execution of the interrupt service routine
  (see Section PROGRAMMING THE INTERRUPT SERVICE ROUTINE)
- Aperiodic processing
  (see Section PROGRAMMING APERIODIC PROCESSING)
- Time-controlled processing
  (see Section PROGRAMMING TIMED-INTERRUPT PROCESSING)
- Shift of a DB initial address
  (see Section PROCESSING DATA WORDS GREATER THAN DATA WORD 255)

The organization blocks are programmed in the same manner as program or sequence blocks, and can be programmed and documented in all three methods of representation (statement list STL, control system flowchart CSF, ladder diagram LAD).

5.2 Overview

In addition to OB 1 for cyclic processing and OB 20 for cold restart and warm restart, the following organization blocks can be processed:

OB 1	cyclic processing

OB for process interrupt processing

OB 2 OB 3	I/Os causing interrupts signal state change at interrupt inputs (edge control)

OB for aperiodic processing

OB 4	aperiodic triggering

OB for timed interrupt processing

	time grid with	
OB 5	n · 2.5 ms	n = 1, 2, 3
OB 6	m · 10 ms	m = 1...9
OB 7	p · 100 ms	p = 1...255

OB 19 [1)]	call of non-existent blocks

OB 20	system start (cold restart and warm restart)

OB 180	change of DB-initial address

1) *SINUMERIK 880 GA2, SW 1, and SINUMERIK 840C*

Organization block OB 1 must always be available. Organization blocks OB 2 to OB 7 are not mandatory. If they are missing, the call must be disabled by entering the machine data (see Interface Description Part 1: Signals, ”PLC MD bits for basic program”); otherwise the control goes into the STOP state.

When changing the processing level (execution of an OB is interrupted by interrupt processing or when another OB is called) all MC5 registers (ACCU 1, ACCU 2, RLO etc.) and flag bytes 224 to 255 or MB 200 to 255 1) (if MD bit 6026.3 is set) are saved or reconstituted. Branching to the DB opened in the respective processing level is also saved (as are changes implemen-ted by OB 180).

The "event counter processing timeout in OB 2 to OB 7" registers record how much requests for interrupt processing were lost while an interrupt-controlled program was being executed (see Section "Detailed error code" and Section "Delay during process interrupt processing").

In addition, a machine data bit can be set to cause the control to go into the STOP state when a processing timeout occurs; otherwise a respective bit is set in flag byte 6 (see Interface Description Part 1: Signals).

5.3 Points of interruption

The cyclically processed user program can be interrupted in two different ways for interrupt servicing:

- in normal mode
- in special mode

Normal or special mode can be selected via a machine data bit (see Interface Description Part 1: Signals, "PLC MD bits for basic program").

1) *SINUMERIK 880 GA2, SW 1, and SINUMERIK 840C*

5.3.1 Normal mode

In normal mode the cyclically processed program can only be interrupted at the block limits by interrupt-controlled processing (Fig. 5.2). Only when a change is made from one block to another - either by calling a new block or by returning to a higher-order block after a block end statement - can the system program call up an organization block for interrupt servicing. Organization blocks OB 2 and OB 5 cannot be called up in normal mode. If these blocks are available in the user program and enabled via machine data (see Interface Description Part 1: Signals, "PLC MD bits for basic program"), they are nevertheless not processed.

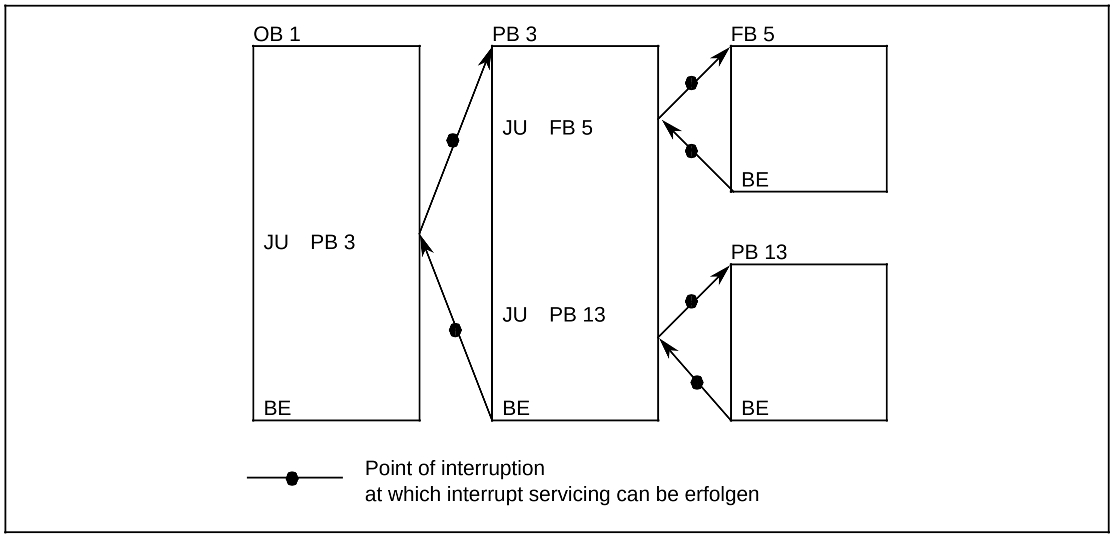


Fig. 5.2 Point of interruption of the cyclically processed program in normal mode

5.3.2 Response time

In normal mode interrupt-controlled processing cannot normally take place while a block is being processed. An interrupt is only serviced on a block change, i.e. when a block is called or terminated. The maximum response time between an interrupt occuring and being serviced is therefore, in the worst case, the processing time of the longest block. If two alarms occur simultaneously, the response time for the interrupt increases for the interrupt with lower priority. First the cyclic program is processed up to the next block change and then the processor processes the "interrupt service routine" with the highest priority. Once the "interrupt service routine" with the higher priority has been completed, the "interrupt service routine" with lower priority is then processed. The response time of this lower-priority program has thus been increased by the processing time of the higher-priority program (see Section "Priority assignments for interrupts").

If several interrupts occur simultaneously, the interrupt with the lowest priority is only serviced when all higher-priority interrupts have been serviced.

The priorities of an interrupt-controlled processing can be shifted if an interrupt occurs while an interrupt-controlled program is being processed. After completion of the servicing of the higher-priority interrupt, the priorities are reassigned, causing the response time for the lowest-priority interrupt to be further increased.

5.3.3 Special mode

In special mode the user program can be interrupted after each MC5 instruction (dependent on PLC MD 6051.0, see Interface Description Part 1: Signals, "PLC MD bits for basic program"). The last command is completed; at the end of the command the timed and process interrupts are enabled. The maximum response time in special mode is therefore reduced to the processing time of a command, if interrupt processing has not yet been activated.

5.3.4 Semaphore technique within the processing levels of a PLC (LIM/SIM)

The commands LIM and SIM give the user a way of protecting sections of programs from interruptions by higher priority processing levels within a PLC. Lock bits, which each apply to an OB, can be first set and then reset in a user program to enclose a section of the program. This section cannot be interrupted by the OB to which a set lock bit applies even if this OB has a higher priority.

Command sequence:

The command LIM loads the OB locks into the ACCU 1 (low byte), which can now be changed by setting and resetting individual bits. Setting a bit disables the OB concerned and resetting the bit enables it. Either all or any combination of OB lock bits can be set or reset. If, after an OB has been disabled, the corresponding timed or process interrupt occurs, the higher priority OB is not executed immediately by the system program but stored in a buffer.

The command SIM supplies the OB lock bits with the content of ACCU 1 (low byte). It also checks whether lock bits were reset. If they were, it checks whether a request was put in the buffer for the corresponding OB. If it was, this OB is invoked, as long as there is no request for a higher-priority OB.

If a time or process interrupt occurs more than once while the lock bit for the corresponding OB is set, requests are lost. In the diagnostics DB (DB 1) a counter for each OB keeps count of how many requests are lost. Flag 6.1 is also set as a group identifier for OB 2 to OB 7.

Assignment of the lock bits (=ACCU 1 low byte):

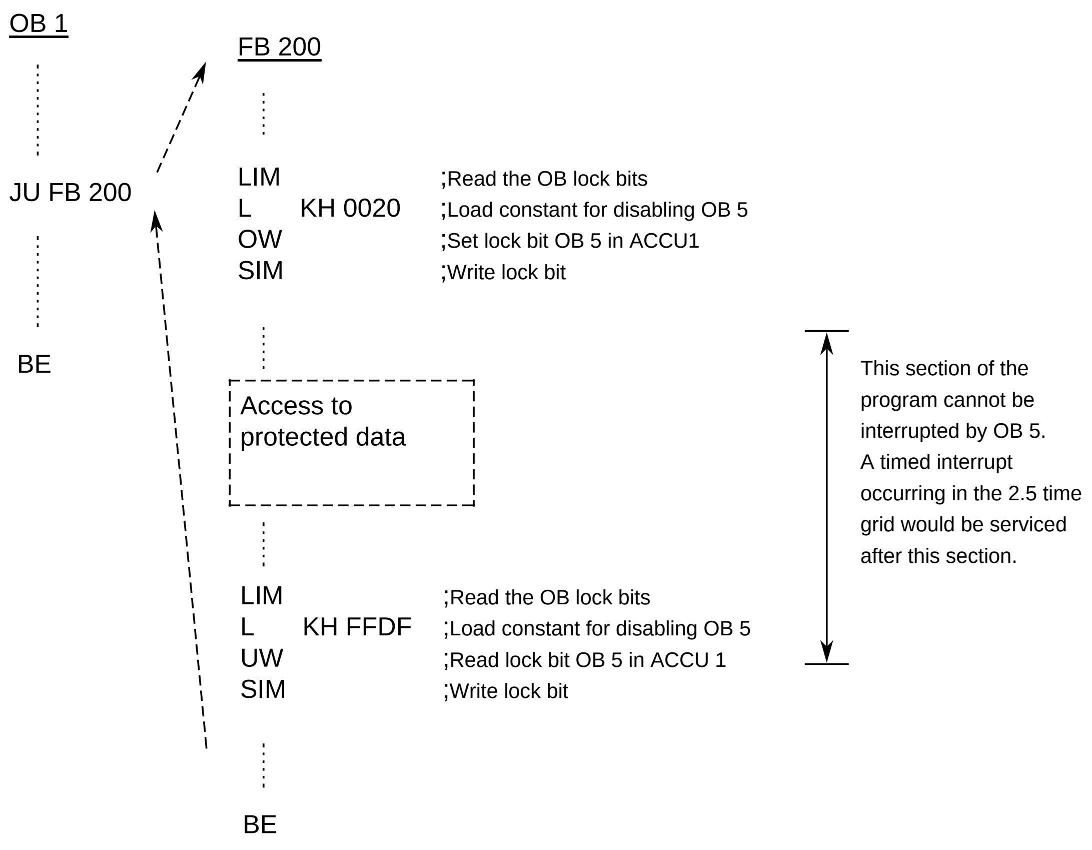
Bit	7	6	5	4	3	2	1	0
	OB 7	OB 6	OB 5	OB 4	OB 3	OB 2	not assigned	not assigned

1 = OB x disabled
0 = OB x enabled

Example:

A user program is processed cyclically, during which the lock bit for the OB5 (time interrupt 2.5 ms - time grid) is set. If a timed interrupt occurs in the 2.5 ms time grid, OB5 is not immediately executed by the system program but the request is put in a buffer. When the lock bit for OB5 is reset in the cyclic user program, the system program recognizes that a request for OB5 was put in the buffer and executes OB5. In this way, the user can make sure that data is not changed by an interrupt before it is stored in the form in which it is required for further processing by the progam.

The use of the commands LIM and SIM can be explained with a programming example:



```
OB 1                FB 200
 :                   :
JU FB 200           LIM                ;Read the OB lock bits
 :                  L      KH 0020     ;Load constant for disabling OB 5
BE                  OW                 ;Set lock bit OB 5 in ACCU1
                    SIM                ;Write lock bit
                     :
                    Access to
                    protected data
                     :
                    LIM                ;Read the OB lock bits
                    L      KH FFDF     ;Load constant for disabling OB 5
                    UW                 ;Read lock bit OB 5 in ACCU 1
                    SIM                ;Write lock bit
                     :
                    BE
```

This section of the program cannot be interrupted by OB 5. A timed interrupt occurring in the 2.5 time grid would be serviced after this section.

Note:

Do not test program sections containing the commands LIM and SIM with the programmer function "status block" because the locked program sections could be interrupted.

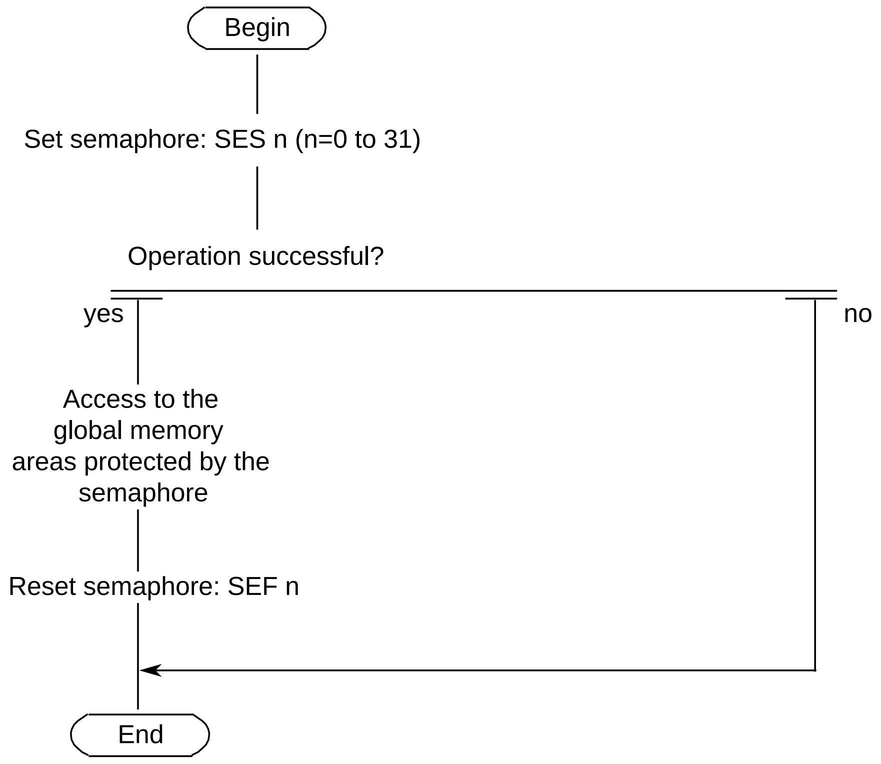
5.3.5 Semaphore technique in multiprocessor mode (SES/SEF) 1)

The user has the use of 32 user semaphores; these are used to protect global memory areas in the communication RAM.

If two or more PLCs use certain global memory areas in the communication RAM in common, there is a danger that the PLCs might overwrite each other's data or that invalid temporary states of the data may be read out. For this reason it is necessary to coordinate access to common memory areas.

The PLCs are coordinated using semaphores and the commands SES and SEF: A PLC can only access the common memory area if it has successfully set the semaphore (SES) for that memory area. The semaphore can only be set by one PLC at a time. If a PLC cannot set the semaphore or if the semaphore has been reset again with the command SEF, the PLC cannot access the memory area.

All the PLCs concerned must contain a block with the following structure:


Begin
Set semaphore: SES n (n=0 to 31)
Operation successful?
yes
no
Access to the global memory areas protected by the semaphore
Reset semaphore: SEF n
End

The commands SES and SEF must be used by all PLCs which need to access synchronisiert a common memory area. The use of SES and SEF guarantees that a piece of information belonging to a PLC can be transferred in or out of a memory area without interruption by another PLC.

1) *SINUMERIK 880 only*

Command description:

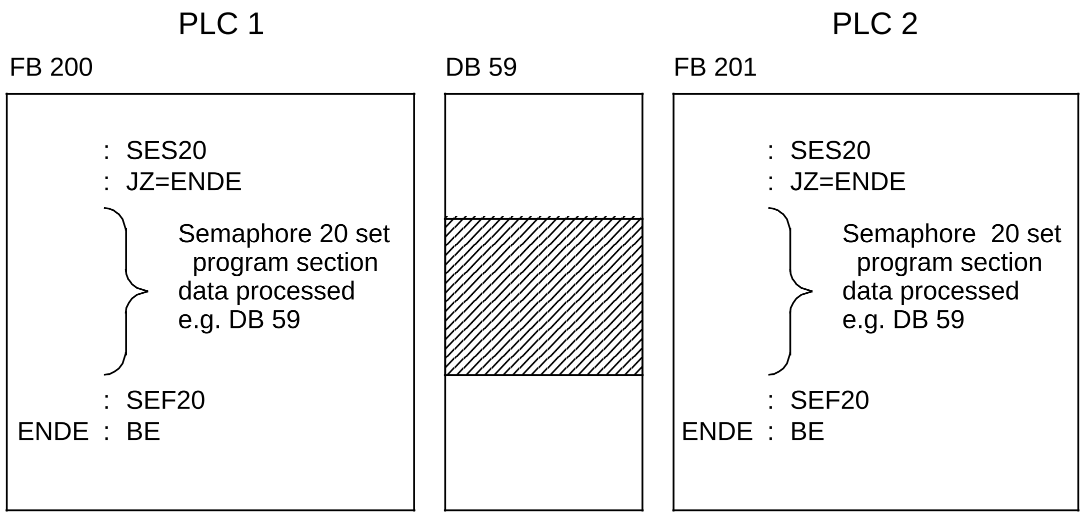
The command "Set semaphore" (SES) sets a byte in the communication RAM assigned to the semaphore for the PLC executing the command (provided it has not already been set by another PLC). As long as this PLC reserves this byte the other PLCs cannot access the area protected by the semaphore.

The command "Reset semaphore" (SEF) resets the byte in the communication RAM assigned to the semaphore. After this, the protected memory area can be read and written by the other PLCs. A semaphore can only be reset by the PLC that set it.

The commands SES and SEF scan the state of the specified semaphore. The flags ANZ0 and ANZ1 are influenced as follows:

ANZ1	ANZ0	Meaning	Evaluation
0	0	**SES:** Semaphore is already set by another PLC and cannot be set. **SEF:** Semaphore is not set or is set by another PLC and cannot be reset	SPZ
1	0	**SES:** Semaphore is set **SEF:** Semaphore is reset	SPN, SPP

Example:


PLC 1
PLC 2
FB 200
DB 59
FB 201
: SES20
: JZ=ENDE
Semaphore 20 set program section data processed e.g. DB 59
: SEF20
ENDE : BE
: SES20
: JZ=ENDE
Semaphore 20 set program section data processed e.g. DB 59
: SEF20
ENDE : BE

5.3.6 Priority assignment for interrupts

If several interrupts occur simultaneously, the interrupts are processed in the following sequence:

Increasing priority ↓

OB 1	cyclic processing
OB 7	time-controlled processing
OB 6	
OB 5	
OB 4	aperiodic processing
OB 3	interrupt-controlled processing
OB 2	

OB 2 and OB 5 are only processed if PLC machine data bit 6051.0 is 0 (Special mode see Installation Guide, "PLC MD bits for basic program").

If the cyclic program is interrupted by an interrupt, all interrupts present are serviced before cyclic program processing is continued. This applies both to interrupts which cause interruption of cyclic operation and all interrupts which occur during interrupt servicing. Hereby after the processing of each interrupt service routine, the interrupt with the next highest priority is found and processed.

5.4 Programming the cyclic program

A programmable controller's program is "normally" scanned cyclically (Fig. 5.4). The processor starts at the beginning of the STEP 5 program, scans the STEP 5 statements sequentially until it reaches the end of the program, and then repeats the entire procedure.

5.4.1 Interface between system program and cyclic program

Organization block OB 1 is the interface between the system program and the cyclic user program. The first STEP 5 statement in OB 1 is also the first statement in the user program, i. e. is equivalent to the beginning of the cyclic program.

The program, sequence and function blocks comprising the cyclic program are called in organization block 1. These blocks may themselves contain block calls, i. e. the blocks can be nested (see Section "Program organization").

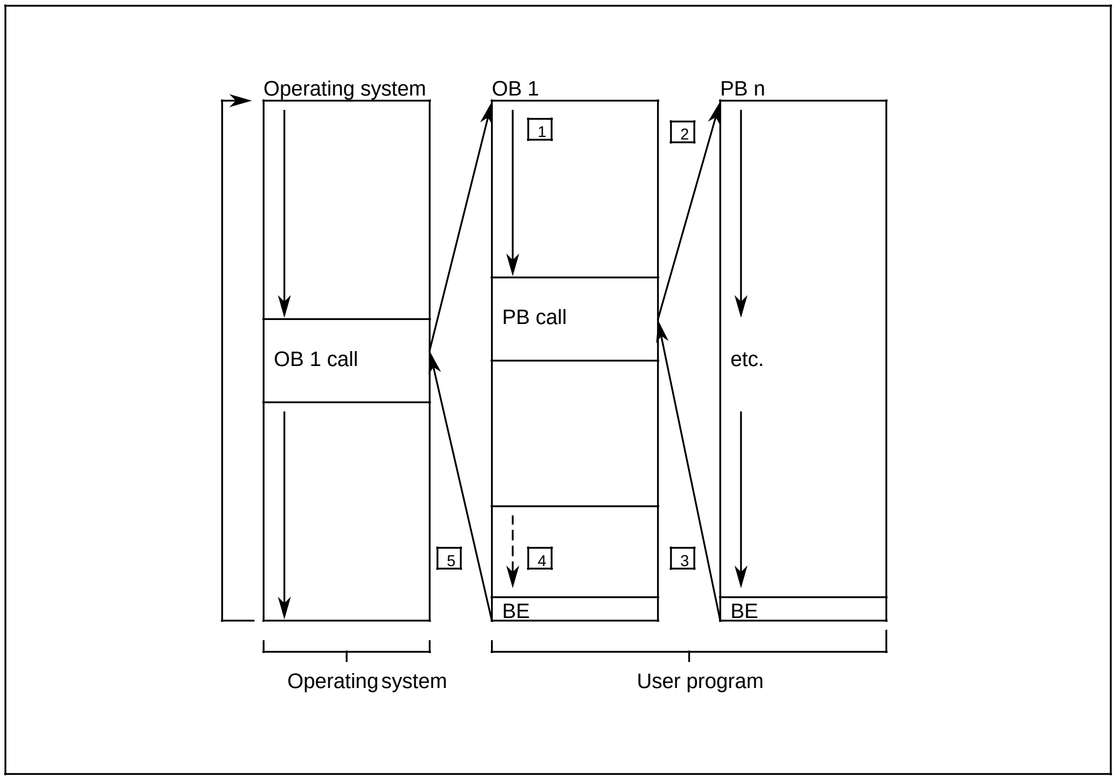


Fig. 5.4 Cyclic program scanning

$\boxed{1}$ First statement in the STEP 5 program.

$\boxed{2}$ First PB call. The block called may contain additional calls (cf. Section 1, "Program organization").

$\boxed{3}$ Return from the last program or function block executed.

$\boxed{4}$ The organization block is terminated with BE.

$\boxed{5}$ Return to operating system.

The user program's runtime is the sum of the runtimes of all blocks called. When a block is called "n" times, its runtime must be added to the total "n" times.

5.4.2 Basic program organization

Organization block OB 1 contains the basic structure of the user program.

A diagram of this block shows the essential program structures at a glance (Fig. 5.5) and emphasizes program-interdependent plant sections (Fig. 5.6).

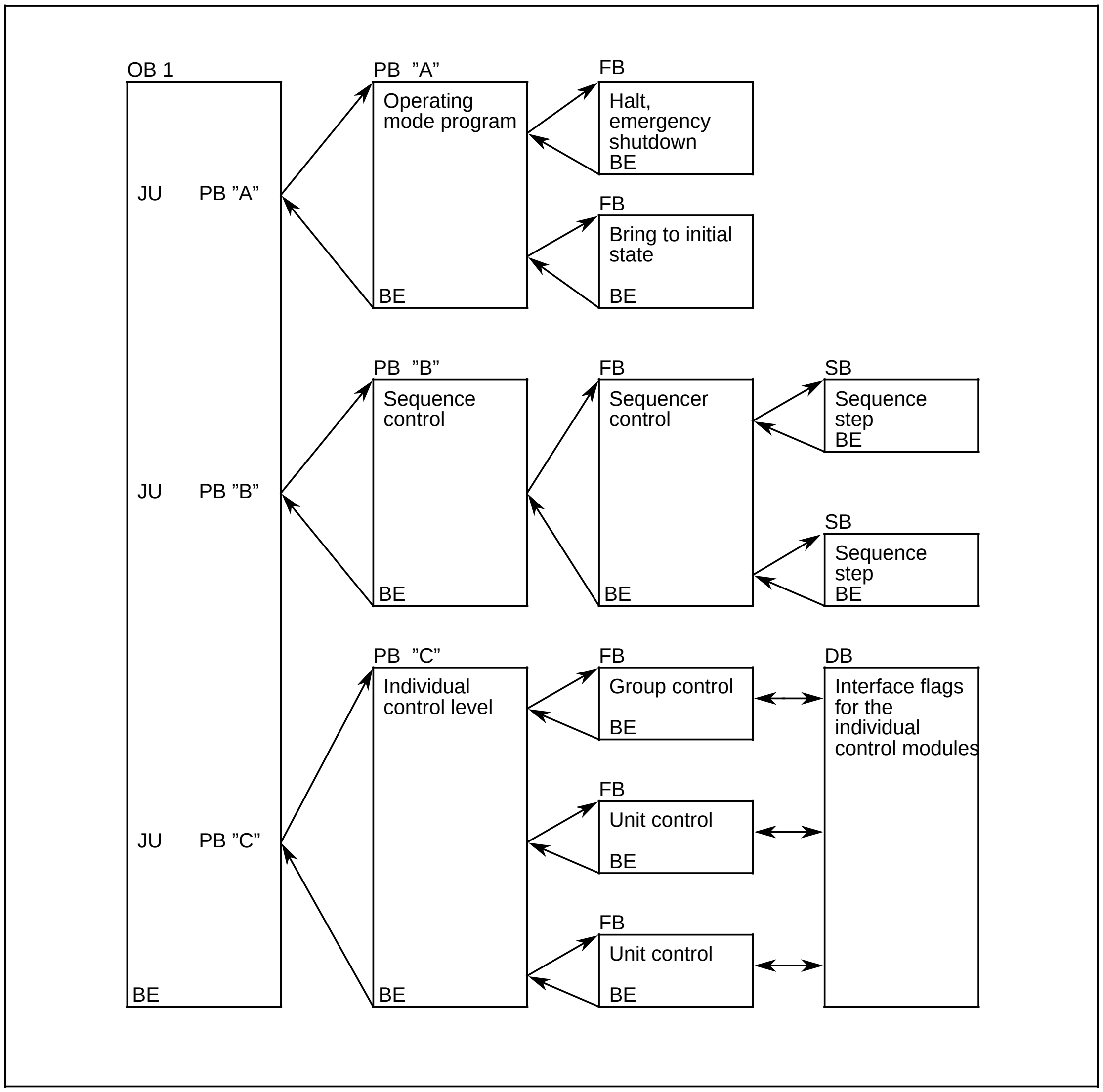


Fig. 5.5 Breakdown of the user program based on the program structure

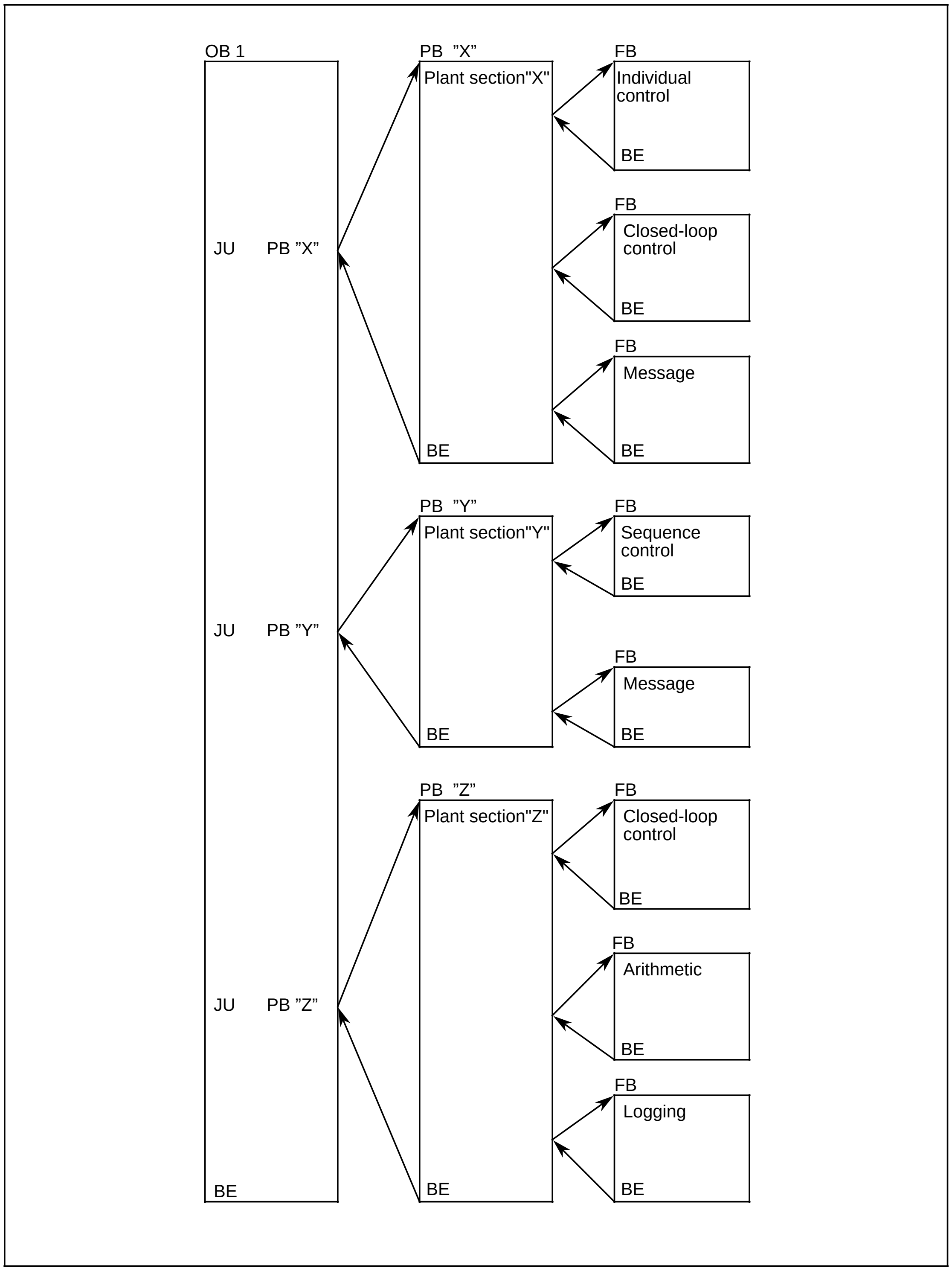


Fig. 5.6 Breakdown of the user program based on the plant structure

5.5 Programming the interrupt service routine

The PLC 135 WB has interrupt-processing capabilities.

In this mode, the cyclic program is interrupted and an interrupt service routine executed. Once the interrupt service routine has terminated, the processor returns to the point of interruption and resumes execution of the cyclic program.

The interrupt service routine is initiated in two different ways:

- I/Os causing interrupts (OB 2)
- signal state change of selected input bits (edge-controlled) (OB 3)

Interrupt service routines on the basis of signal status changes (OB 3) allow the user to react immediately to process signals connected to a maximum of 4 selected input bytes. An edge change in these signals is thus registered before the process image is updated, thereby minimizing the response time to time-critical functions in the process.

5.5.1 Interface between operating system and the interrupt service routine

OB 2 and OB 3 constitute the interface between the operating system and the interrupt service routines.

Organization block OB 2

OB 2, the block with the highest priority, is called by interrupts of the process I/Os causing the interrupts. It can only be processed in special mode. The alarms are retained in flag bytes FY 8 to FY 10 and must be acknowledged (reset) by the user.

Organization block OB 3

OB 3 is always invoked when the signal state of a bit in up to four interrupt input bytes changes. The user may select the input bytes and default it via machine data (PLC MD 124 to 127, see Interface Description Part 1: Signals).

When one of the selected bits changes from "0" to "1" (positive edge) or from "1" to "0" (negative edge), the interrupt service routine is invoked. The system program calls OB 3, which contains the user's interrupt service routine.

The system program checks the interrupt bytes every 10 ms (in dependence on MD 155), and invokes an interrupt service routine when required. OB 3, however, is only processed once per 10 ms scan, even if several signal state changes occur. This means that all the accumulated edges have to be processed in one pass of OB 3 because they will be overwritten the next time OB 3 is called.

The type of signal change (positive or negative edge) is entered as transfer parameter in flag bytes FB 12 to FB 19 (see Interface Description Part 1: Signals). It is not necessary to acknowledge the interrupts in the flag bytes.

OB 3 can be processed in normal or special mode and has a lower priority than OB 2, i.e. OB 3 cannot interrupt OB 2.

5.5.2 Timeout in process interrupt processing

Timeout with OB 3

If an edge change is detected on an interrupt byte while OB3 is running, the call request for OB3 is renewed. After OB3 has terminated it is called again.

If the operating system detects that an interrupt has already been entered, it signals a timeout (FY 6, bit 3). The following then applies:

- On versions up to SINUMERIK 880, SW6, no distinction is made between a positive edge and a negative edge. A timeout arises if, for example, a positive and then a negative edge is detected at a process interrupt input.
- On SINUMERIK 880 GA2, SW1, and SINUMERIK 840C, a timeout is signalled if a positive or negative edge is detected at least twice while OB3 is running.

If the PLC MD bit 6048.3 is set when a timeout occurs, the event counter timeout OB 3 (DB 1, DW 23) is incremented and an OB 3 call is requested again. If the bit is not set, the PLC goes into the stop state.

Notes:

With the SINUMERIK 880 GA2, SW1, and SINUMERIK 840C the following applies:

- If several timeouts occur simultaneously within an interrupt byte, no more than one timeout for the positive edges and one timeout for the negative edges is entered.
- If several edges occur within an interrupt byte only one timeout is signalled (e.g. two positive edges).
- If several positive and several negative edges occur simultaneously within an interrupt byte with timeout, the event counter timeout OB 3 is incremented by two.

Interrupt-forming I/Os (OB 2)

If an edge change is detected at an interrupt input while OB 2 is running, a new request to call OB 2 is set. A timeout occurs, if an edge change is detected while OB 2 is running.

5.6 Programming aperiodic processing

With organization block OB 4, blocks can be triggered aperiodically. By calling a defined function macro FB 68 a delay time is started after which OB 4 is called up by the system program. The function macro is defaulted when called with the desired delay time (0-32767 ms).

5.7 Programming timed interrupt processing

The processor of the interface control also executes time-controlled processing. Time-controlled processing is when a signal coming from the "internal clock" causes the processor in the programmable controller to interrupt "normal" cyclic processing and to process a specific program.

After processing this program the processor returns to the point of interruption in the lower-priority program and continues processing (Figs. 5.7 and 5.8).

Note:

Time-controlled processing is not enabled when the system is started up (OB 20), i.e., OB 20 is not interrupted by OB 5, 6 or 7.

5.7.1 Interface between system program and time-controlled processing

Organization blocks OB 5, 6 and 7 constitute the interface between system program and time-controlled processing. Each of these organization blocks is called by the system program in a defined time grid.

Factors n, m and p can be defined freely by the user within the defined limits and are stored in PLC MD, 2, 3 and 4 (see INSTALLATION INSTRUCTIONS).

Organization block	Time grid	
OB 5	n · 2.5 ms	n = 1, 2, 3
OB 6	m · 10 ms	m = 1...9
OB 7	p · 100 ms	p = 1...255

Organization block OB 5 can only be processed in special mode, OB 6 and 7 can be processed in either normal or special mode.

If several timed interrupts occur simultaneously, interrupt processing is executed with the priorities specified in Section "Priority assignment for interrupts".

Example of processing when several alarms occur:

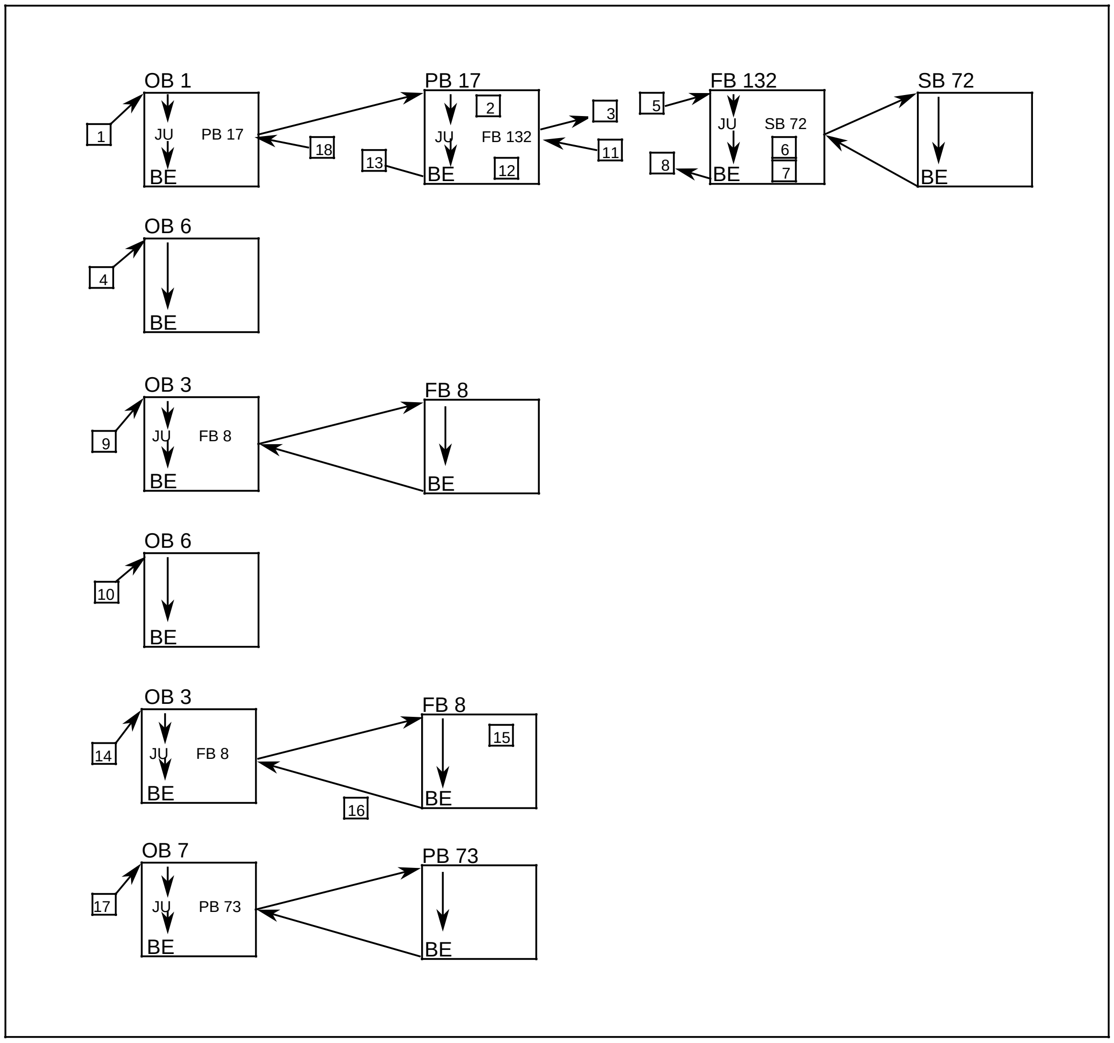


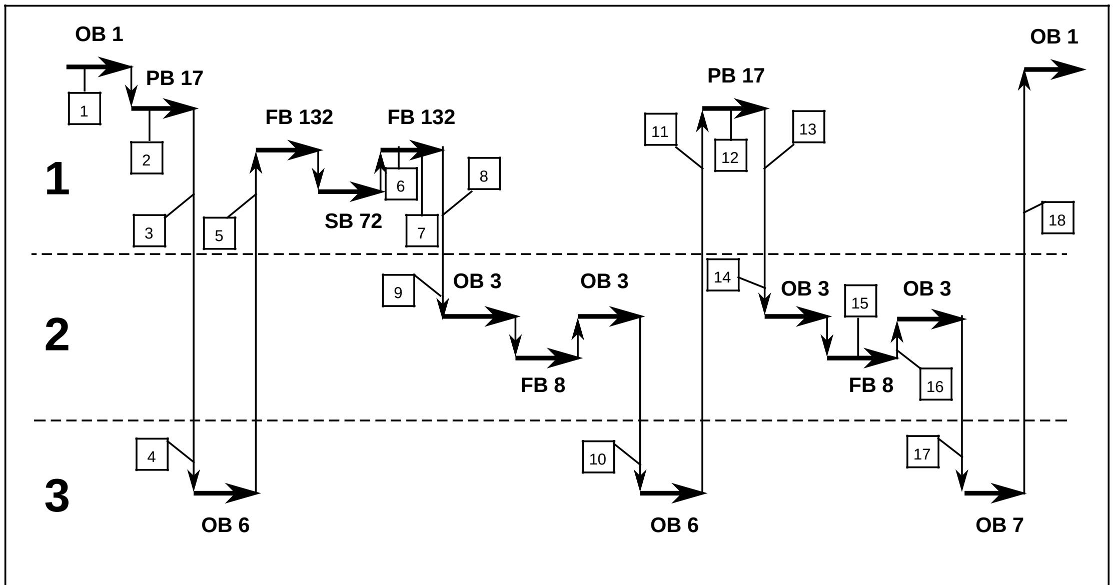
Fig. 5.7 Interrupt-controlled and time-controlled processing (in normal mode) when several interrupts occur

In program block PB 73 is a program part without time-critical response time. It is sufficient if the inputs and outputs programmed are processed every two seconds. (This programming method can reduce the average cycle time of the interface control.)

Programming:

```
OB 7    : JU PB 73
        : BE
```

The call for the program block PB 73 is programmed in organization block OB 7, whereby 20 x 100 ms (p=20) has been selected as time grid for the organization block, so that OB 7 is processed every 2 seconds. The call can be unconditional or conditional depending on the previous operations programmed. The organization block is completed with the BE statement.



1 Cyclic processing
2 Process alarm processing
3 Timed interrupt processing

Fig. 5.8 Schematic representation of the blocks in the previous example in processing sequence

Legend for Figs. 5.7 and 5.8

$\boxed{1}$ Start of cyclic processing. The system program calls up organization block OB 1.

$\boxed{2}$ The timer pulse of the "internal clock" occurs.

$\boxed{3}$ Block change: The request for the time-controlled program is registered. Cyclic program processing is interrupted.

$\boxed{4}$ System program calls up user organization block OB 6 which is to be processed in a 10 ms time frame.

$\boxed{5}$ After processing the time-controlled program, cyclic program processing is continued.

$\boxed{6}$ A timer pulse from the "internal clock" occurs.

$\boxed{7}$ A process interrupt occours: Signals state changes at input 0.4.

$\boxed{8}$ Block change: The request for the time-controlled and interrupt-controlled program processing is registered. Cyclic program processing is interrupted.

$\boxed{9}$ As process interrupt processing of the user program has a higher priority than the time-interrupt processing, process-interrupt processing is carried out first. The system program calls up organization block OB 3.

10 After the request interrupt has been completely processed, cyclic program processing is not resumed, as there is still a time-controlled program processing waiting. The system program calls up organization block OB 6, which is processed in a 10 ms time grid.

11 After processing of the time-controlled program has been completed, cyclic program processing is resumed as there is no further alarm.

12 A process interrupt occurs: Signal state change at input 0.3.

13 Block change: The process interrupt is registered; cyclic program processing is interrupted.

14 The system program calls up organization block OB 3 assigned to input 0.3.

15 A timer pulse of the "internal clock" occurs requesting time-controlled program processing during processing of the process interrupt.

16 Block change: The request for time-controlled program processing is registered. Process interrupt processing is not interrupted.

17 After completion of the processing of the interrupt-controlled program, processing of the time-controlled program is started. The system program calls up OB 7.

18 As there is no further alarm, cyclic program processing is continued at the point of interruption.

5.8 Calling non-existent blocks 1)

Calling a non-existent block

If a user program calls a non-existent (not loaded) block of type PB, SB, FX or OB (OB 180), the system program calls organization block OB 19 (if present) and continues the interrupted program without an error message. If OB 19 is not present the call to the non-existent block is ignored.

The PLC only goes into the stop state with an error message if an non-existent block is called within the level of OB 19.

When OB 19 is called the type and the number of the missing block is loaded into ACCU1-L (see Figure).

1) SINUMERIK 880 GA2, SW 1, and SINUMERIK 840C

Opening a non-existent data block

When you attempt to open a non-existent (not loaded) block of type DB or DX, the system program tries to call the organization block OB 19.

If OB 19 is present, the value "0" is entered in the system data "current DB" and "DB length" (i.e. no data block has been opened). The user program contained in OB 19 is executed and the interrupted program is continued without an error message.

If OB 19 is not present, the PLC goes into the stop state.

Within the level of the OB 19, the PLC always goes into the stop state with an error message when an attempt is made to open a non-existent data block.

When OB 19 is called the type and the number of the missing block is loaded into ACCU1-L (see Figure).

Assignment of the ACCU1-L

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		V			Block type				Block number							

V (validity)	0: Block number is valid
	1: Block number is invalid (the block number could not be calculated)

Block type	0 0 0 0	OB
	0 0 0 1	PB
	0 0 1 0	SB
	0 0 1 1	FB
	0 1 0 0	FX
	1 0 0 0	DB
	1 0 0 1	DX

Notes:

- If an attempt is made to open a non-existent data block in a high-level language block with the function HLL_ADB, OB 19 is also called. The PLC goes into the stop state with the error message 9B. F0004 contains 1.
- If an attempt is made to open a non-existent data block in a high-level language block with the function HLL_HLL or with the function HLL_MACRO, the PLC always goes into the stop state with an error message and OB 19 is not called.

6 Start-up

Starting up of the PLC 135 WB **/** WB2 **/** WD is subordinate to the starting up of the numerical machine tool control (NC). The CPUs of the PLC have subordinate functions.

When the NC branches into the "Start-up mode overall reset" mode a start-up bit is set. In the NC, deletion and loading of NC and PLC machine data or parameter assignment for the system program can be executed, with the aid of which start-up of PLC 135 WB **/** WB2 **/** WD is controlled.

Note:

Start-up of the control and the PLC is described in the INSTALLATION INSTRUCTIONS of the individual controls.

6.1 Self-diagnostics program

After switching on the mains voltage, the interface control runs a self-diagnostics program. This program tests the most important hardware components and initializes the software required for system start-up. If errors in the system are recognized, the LED on the front plate displays the error (Table 6.1 for SINUMERIK 880/880 GA2 and Table 6.2 for SINUMERIK 840C).

LED	Significance (SINUMERIK 880**/**880 GA2)
continuous (green only)	Cyclic operation
continuous (red only)	STOP state
continuous (red and green)	GENERAL RESET required (initial power on or data loss)
flashing once	Error with cross-check sum via the system program
twice	Error in the CPU RAM test
three times	Error in timer 0 (process-internal timer) or watchdog error
4 times	Error in monitoring test for timeout (880 SW4 only)
5 times	Access to link RAM not possible
6 times	Error with test access to link RAM
7 times	Error in system initialization program
8 times	Other PLC failed (if two PLCs are used)
10 times	Error of the internal co-processor (COP) register or in the step address counter (SAZ) creation
11 times	Command delegation to word processor (WOP)
12 times	Processing of binary commands
13 times	Processing of OR, bracket expressions, NOP- or BLD-commands
14 times	Processing of block calls and jump commands
15 times	Processing of timer and counter operations
16 times	Addressing in data memory
17 times	Command execution mode
18 times	Test address comparator machine code (MC5) and interrupt processing via co-processor
19 times	Monitoring test for timeout with access of the co-processor to the user memory
20 times	ACOP2 not present

*Table 6.1 LED displays and their meanings for SINUMERIK 880**/**880 GA2*

LEDs for PLC	Significance (SINUMERIK 840C)
continuous (green only)	Cyclic operation
continuous (red only)	PLC in STOP state
continuous (red and green)	GENERAL RESET required (initial power on or data loss)
flashing (red) once	Error with cross-check sum via the system program
three times	Error timer 0 (process-internal timer) or Watch dog error
4 times	SW 3: Module is a PLC 135 WB (can no longer be used)
5 times	Access to link RAM not possible
6 times	Error with test access to link RAM
7 times	SW 1 and 2: Error in system initialization program (synchronization pattern)
	SW 3: No communication with MMC
9 times	SW 3: Booting error
10 times	ACOP error (group error)
11 times	IF PLC cannot be addressed from the PLC
12 times	IF PLC RAM defective (program memory or CPU RAM)
13 times	Dual Port RAM of IF PLC defective
14 times	ADS link to NC defective
15 times	ADS link to IF PLC defective
16 times	SW 3: ADS link to MMC defective
17 times	ADS link (reserved)
LED for IF PLC	**Significance**
continuous (green only)	IF PLC in cyclic operation (no errors), MPC transmission running
continuous (red only)	IF PLC in STOP state
flashing (red) 12 times	IF PLC RAM defective (CPU RAM)
13 times	Dual Port RAM of IF PLC defective
LED off	No MPC transmission, processor running

Table 6.2 LED displays and their meanings for SINUMERIK 840C

Note:

If the PLC 135 WB2 is used, the LEDs for the PLC and IF PLC are situated on the PLC 135 WB2 or interface PLC module. If the PLC 135 WD is used, all the LEDs are situated on the front panel of this module.

6.2 System initialization program

After the self-diagnostics program has run, the system initialization program is called.

In its first section, the data required for running the organization program are set up. This setting up includes:

- Stack organization,
- Segmentation for word processor and co-processor,
- Entries in the location-dependent CPU interrupt table,
- Task priority lists,
- Setting up task data and
- Initialization of counts and periodic values.

In the second section the system initialization program defines the type of start up after switching on the mains voltage. The following points are checked:

- Whether the switch-on test pattern is missing (i.e. data lost)
- Whether there is a battery interrupt
- If the setting-up bit is set
- Request from the NC "automatic warm restart after setting-up overall reset"
- STOZUS operating status bit set (acquisition of interrupt event or continuation of the STOP state, see Section "Memory Allocation and Organization")
- Cold restart or warm restart attempt aborted.

If the STOZUS identifier is set, the control remains in the STOP state.

If, in the second section, (testing of run-up after switching on the mains voltage) the STOZUS identifier is not set, but one of the other conditions is fullfilled, an automatic cold restart is executed; a warm restart of the control only occurs if none of the mentioned conditions are fulfilled.

Overall reset with subsequent bootstrapping of the user memory (URLOE = 1) is always required

- If first start-up is instigated,
- Data loss occured by removing the PLC CPU from the central controller or with power failure simutaneous battery voltage failure.

If the mains voltage fails during active processing checks, the processing checks are aborted by the programmer. The system initialization program instigates the cold restart.

6.3 User data blocks

On each cold restart, certain data blocks are set up and written in ascending order into the user data segment. A list of these data blocks is contained in the INTERFACE DESCRIPTION PART 1: SIGNALS of the control concerned.

Data blocks DB 30 to DB 71 are defaulted with 0. The channel, spindle and axes data blocks are defaulted with the user data if they are enabled for writing (see INTERFACE DESCRIPTION, PART 1: SIGNALS).

A warm restart has the same initialization as a cold restart except that

- User machine data (DB 62, DB 65) and
- Setting data (DB 68, DB 71)

are not defaulted with 0, but retain their values.

Note:

The decoding blocks for the NC channels (DB 80 and higher) are not set up or initialized by the system.

6.4 Timeout analysis

A write access of the PLC to the communication or local bus is executed independently by the bus interface. Write accesses are acknowledged immediately, and the PLC program can continue (Buffered access to communication**/**local bus). If a timeout occurs during such an access the current state of the registers of the processor and co-processor give no information as to the cause of the timeout.

The user can switch off buffered accesses to the communication and local bus (e.g. to test STEP 5 programs during the installation phase) via machine data (PLC operating system MD bit 6049.0). These accesses are then slower because the processor only receives an acknowledgement when the whole bus cycle has finished.

Machine data 6049.0 must be set in order to be able to determine the exact cause of a timeout.

7 Device Error Analysis

The system program can ascertain faulty operation of the central processor, errors in the system program or the effect of erroneous programming.

If the interpreter ascertains an error in command execution or if another error occurs causing program interruption, a branch is made into the STOP loop.

7.1 Interrupt stack

The programmer's ”Output ISTACK” function can be invoked to help analyze errors. This function displays the control bits.

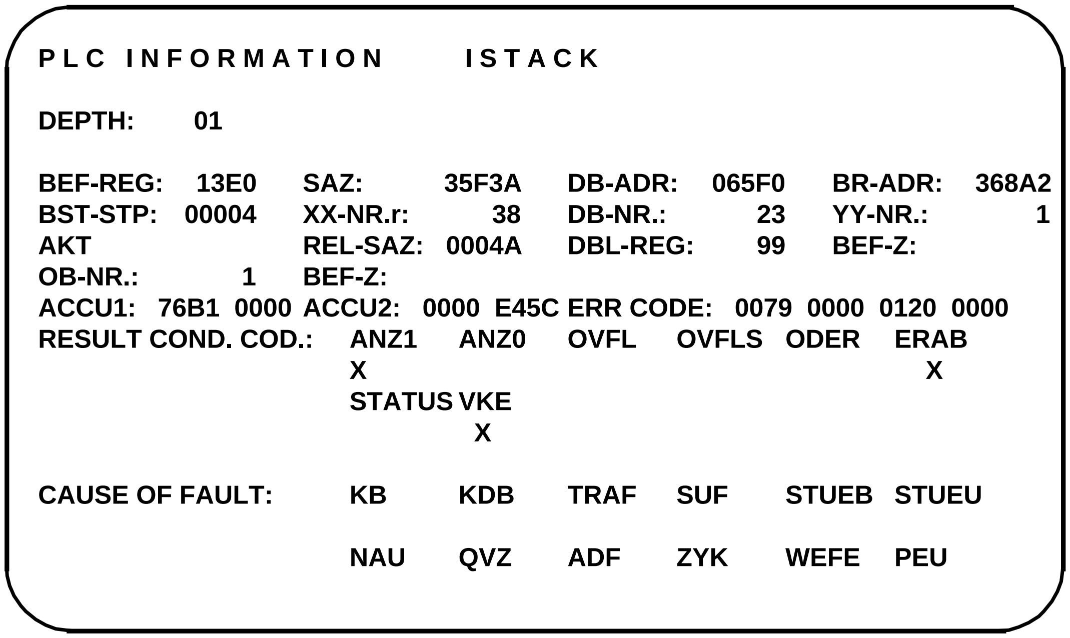
```
C O N T R O L   B I T S

NB      NB      BSTSCH  SCHTAE  ADRBAU  SPABBR  NAUAS   NB
NB      NB      NNN     NB      NB      NB      NB      NB
STOZUS  STOANZ  NEUSTA  WIEDAN  BATPUF  URLAD   BARB    BARBEND
KEINPS  UAFEHL  MAFEHL  EOVH    NB      NB      OBWIED  NB
NB      NB      KOPFNI  NB      NB      PADRFE  ASPLUE  RAMADFE
EAADFE  SYNFEH  NINEU   NIWIED  RUFBST  NB      SUMF    URLOE
NB      NB      STS     STP     NB      NB      NB      NB
NB      NB      NB      NB      NB      NB      NB      NB
```

The control bits in detail:

NB	Not used
BSTSCH	Coordination bit for block shift
SCHTAE	Memory compression facility active
ADRBAU	Successful address table generation following cold/warm restart
SPABBR	Memory compression aborted
NAUAS	Decentralized mains power failure
NNN	Illegal OP-code/invalid command parameter
STOZUS	PLC in STOP mode (may be set/reset by OS only)
STOANZ	STOP status flag
NEUSTA	Cold restart flag
WIEDAN	Warm restart flag
BATPUF	Power supply unit is battery-backed
URLAD	Copying the user program data was aborted

BARB	Processing check test function active
BARBEND	STOP state caused by abortion of processing check test function
KEINPS	No user program memory or user data memory submodule
UAFEHL	STOP status due to interrupt event
MAFEHL	Restart error flag (PLC could not go into normal operation)
EOVH	Input bytes available for process alarms (four successive inputs)
OBWIED	Organization block for warm restart (OB20) active
KOPFNI	Block header in user memory cannot be interpreted
PADFRE	Bad EPROM
ASPLUE	User memory addressing not contiguous
RAMADFE	Bad RAM
EAADFE	Error in I/O area
SYNFEH	Invalid block length in user memory or bad block synchronization word
NINEU	No cold restart possible (bootstrapping required)
NIWIED	No warm restart possible (cold restart required)
RUFBST	Non-existent block called
SUMF	Sumcheck error
URLOE	Overall reset and subsequent bootstrap loading of user memory required
STS	Direct STOP initiated via STS
STP	Interrupt condition code following STP

The following can be called to screen as the next display:



```
P L C  I N F O R M A T I O N       I S T A C K

DEPTH:       01

BEF-REG:    13E0     SAZ:         35F3A     DB-ADR:    065F0     BR-ADR:    368A2
BST-STP:   00004     XX-NR.r:        38     DB-NR.:       23     YY-NR.:        1
AKT                  REL-SAZ:   0004A      DBL-REG:      99     BEF-Z:
OB-NR.:          1   BEF-Z:
ACCU1:  76B1  0000  ACCU2:   0000  E45C ERR CODE:   0079  0000  0120  0000
RESULT COND. COD.:   ANZ1      ANZ0      OVFL      OVFLS   ODER     ERAB
                     X                                                X
                     STATUS VKE
                             X

CAUSE OF FAULT:      KB        KDB       TRAF      SUF      STUEB    STUEU

                     NAU       QVZ       ADF       ZYK      WEFE     PEU
```

DEPTH	The last interrupt event (DEPTH = 01) is always shown for the PLC 135 WB
BEF-REG	OP code and parameters of instruction being content of the step address counter, points to the next instruction
SAZ	Content of the stop address counter points to the address of the next command
DB-ADR	Block start address
BR-ADR	specifies the block return address, i.e. the address at which processing of a block is resumed when the current block has been exited. This address is in the calling block.
BST-STP	Block stack pointer
XX-NR.	Current block whose execution was initiated by the interrupt
DB-NR.	Current data block (DB or DX)
YY-NR.	Block that called the current block (XX)
AKTOB-NR.	Current processing level 9999 is displayed, if no processing level is active
REL-SAZ	Relative step address counter in the current block
DBL-REG	Register containing the data block length (DB or DX)
BEF-Z	Instruction pointer (for high-level language blocks), see description of high-level language blocks
ERR CODE	Error number as in address F0000 (error coding)

See Section EVENT FLAGS OF THE PLC 135 WB/WB2/WD for the meaning of the result condition codes.

KB	Invocation of a non-existent block
KDB	Invocation of a non-existent block
TRAF	Transfer error
SUF	Substitution error
STUEB	Block stack overflow (maximum 64 entries possible)
STUEU	Interrupt stack overflow (maximum 7 levels possible)
NAU	Mains power failure (controller remains in the STOP mode until power is recovered)
QVZ	Timeout
ADF	Adressing error
ZYK	Cycle scan time exceeded
WEFE	Group flag for runtime error of a time-controlled program
PEU	I/Os not ready (in the expansion unit)

7.2 Detailed error code

Using the programmer's info function, the user can display additional information for interrupt analysis by entering pseudo address F0000$_{hex}$.

The following is displayed on the programmer screen:

Up to SINUMERIK 880, SW 6

```
ADR    WERT      ADR    WERT      ADR    WERT      ADR    WERT
F0000  00xx      F0001  xxx1      F0002  xxx2      F0003  xxx3
F0004  zob2      F0005  zob3      F0006  zob4      F0007  zob5
F0008  zob6      F0009  zob7      F000A  0000      F000B  0000
F000C  0000      F000D  0000      F000E  0000      F000F  0000
F0010  0000      F0011  0000      F0012  0000      F0013  * * * *
F0014  * * * *   F0015  * * * *   F0016  * * * *   . . . .
```

SINUMERIK 880 GA2, SW 1, and SINUMERIK 840C

```
ADR    WERT      ADR    WERT      ADR    WERT      ADR    WERT
F0000  00xx      F0001  xxx1      F0002  xxx2      F0003  xxx3
F0004  xxx4      F0005  * * * *   F0006  * * * *   F0007  * * * *
F0008  * * * *   F0009  * * * *   F000A  0000      F000B  0000
F000C  0000      F000D  0000      F000E  0000      F000F  0000
F0010  0000      F0011  0000      F0012  0000      F0013  * * * *
F0014  * * * *   F0015  * * * *   F0016  * * * *   . . . .
```

Addresses F000C to F0012 are described in detail in Section ”Block lists”, the displays ”* * * *” are irrelevant.

The following applies to addresses F0000 to F0009:

xx	**=**	**Internal detailed error code (ERRCODE)**
xxx1	**=**	**Auxiliary error info, word 1**
xxx2	**=**	**Auxiliary error info, word 2**
xxx3	**=**	**Auxiliary error info, word 3**
xxx4	**=**	**Auxiliary error info, word 4** 1)

1) *SINUMERIK 880 GA2, SW 1, and SINUMERIK 840C*

zob2	**=**	**Event counter, processing timeout in OB 2** 2)
zob3	**=**	**Event counter, processing timeout in OB 3** 2)
zob4	**=**	**Event counter, processing timeout in OB 4** 2)
zob5	**=**	**Event counter, processing timeout in OB 5** 2)
zob6	**=**	**Event counter, processing timeout in OB 6** 2)
zob7	**=**	**Event counter, processing timeout in OB 7** 2)

Event counter timeout

The "Event counters, timeout in OB 2 to OB 7" show how many requests for interrupt processing were lost while an interrupt-controlled program was being processed (see the Section "Timeout in process interrupt processing").

The number of the organization block which caused the delay is entered in word 1 (xxx1) of the auxiliary error info.

Note:

On the SINUMERIK 880 GA2, SW1 and higher and SINUMERIK 840C and higher, the "event counters, timeout" were removed from the pseudoaddresses F0004 to F0009 and put into the diagnostics DB (DB 1, DW 22 to DW 27). Flags F0005 to F0009.

Supplementary error data

The auxiliary error info enables more precise analysis of the reason for a timeout or parameter initialization error. All auxiliary error info is deleted on a cold restart.

The opcode of the instruction that generated the timeout is stored in word 1 (xxx1).

Words 2 and 3 (xxx2, xxx3) contain the following:

- Timeout caused by LIR, TIR, TNB or TNW:
  xxx2 Offset address and
  xxx3 Segment number of the non-addressed memory
- Timeout caused by substitution operations:
  xxx2 Substitution operation
- Timeout caused by LPB, LPW, TPB, TPW operations:

xxx2	=	000E	(Timeout during loading of the input modules)
	=	000A	(Timeout during transport to the output modules)
xxx3	=	Byte address (BCD coded) of the operation parameter	

Note:

- On the SINUMERIK 880 GA2, SW1 and higher and SINUMERIK 840C and higher, the detailed error code (F0000) and the supplementary error data (F0001 to F0004) are also stored in the diagnostics DB (DB 1, DW 160 to DW 164).
- The error messages of the detailed error code (ERRCODE) are listed in the INSTALLATION INSTRUCTIONS of the control concerned.

2) Up to and including SINUMERIK 880, SW 6

8 Memory Allocation and Organization

8.1 Segment allocation

The expanded memory area of the PLC 135 WB/WB2/WD interface control can be addressed directly with the new programmer software. The segment switch is no longer required.

The following address areas can be selected:

Segment / Addresses	SYDSEG	AWDSEG	AWPSEG	Segment switch	Error identifier
Programmer initial address/effective address for:					
SINUMERIK 880	00000/00000	06000/0C000	30000/60000	E0000/---	F0000/---
SINUMERIK 880 GA2	00000/00000	06000/0C000	30000/60000	E0000/---	F0000/---
SINUMERIK 840C SW1 and 2	00000/00000	06000/0C000	30000/60000	E0000/---	F0000/---
SINUMERIK 840C SW3	00000/00000	08000/10000	30000/60000	E0000/---	F0000/---

Because the programmer addresses are word-oriented addresses, the programmer address is the effective address divided by two.

The segment switch is now only required when reading in blocks and in the function SPAUS.

Memory allocation

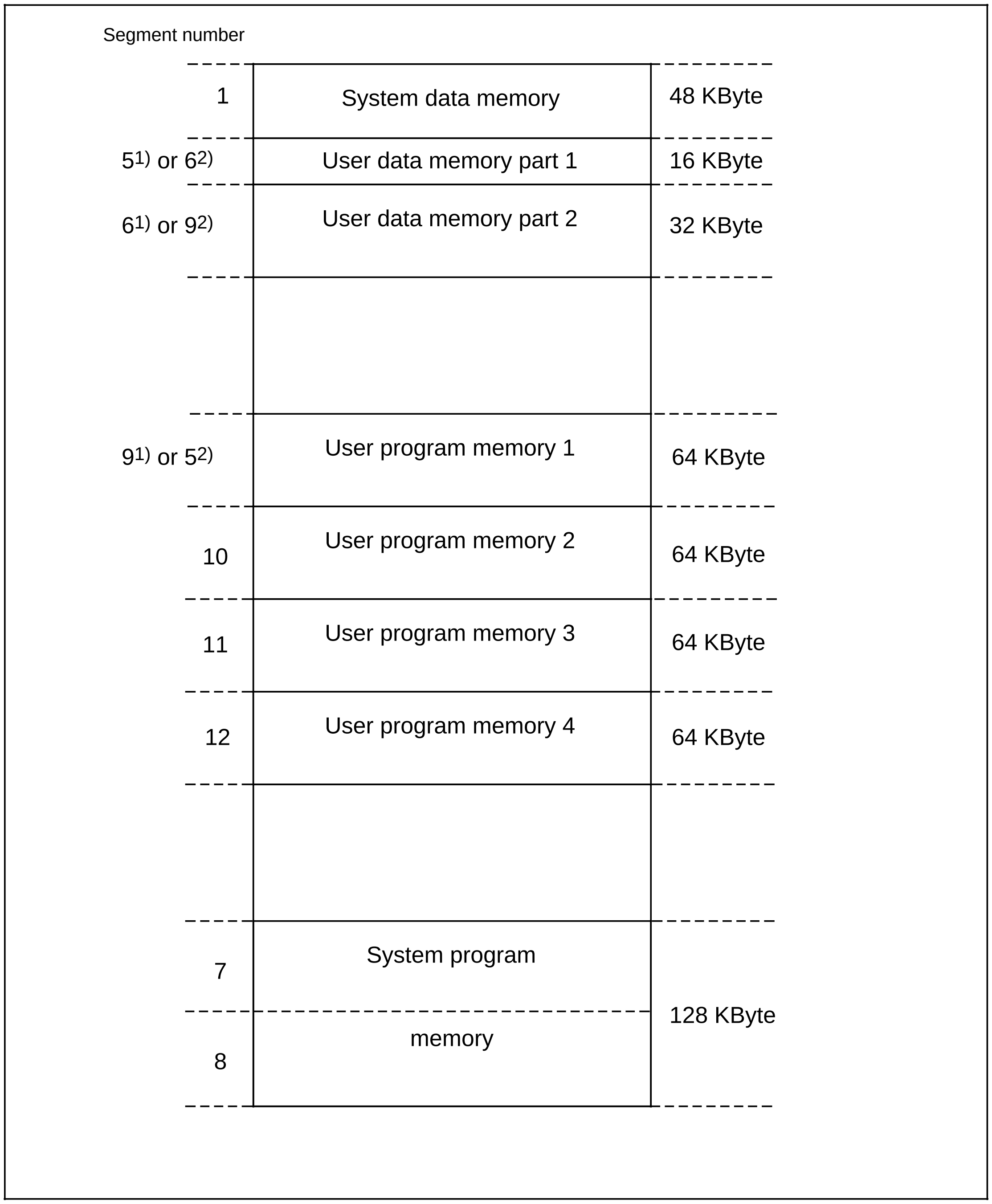


Fig. 8.1 Memory allocation in the PLC 135 WB/WB2 interface control

1) *SINUMERIK 880, SW 6*
2) *Up to and including SINUMERIK 880, SW 4*

Memory allocation

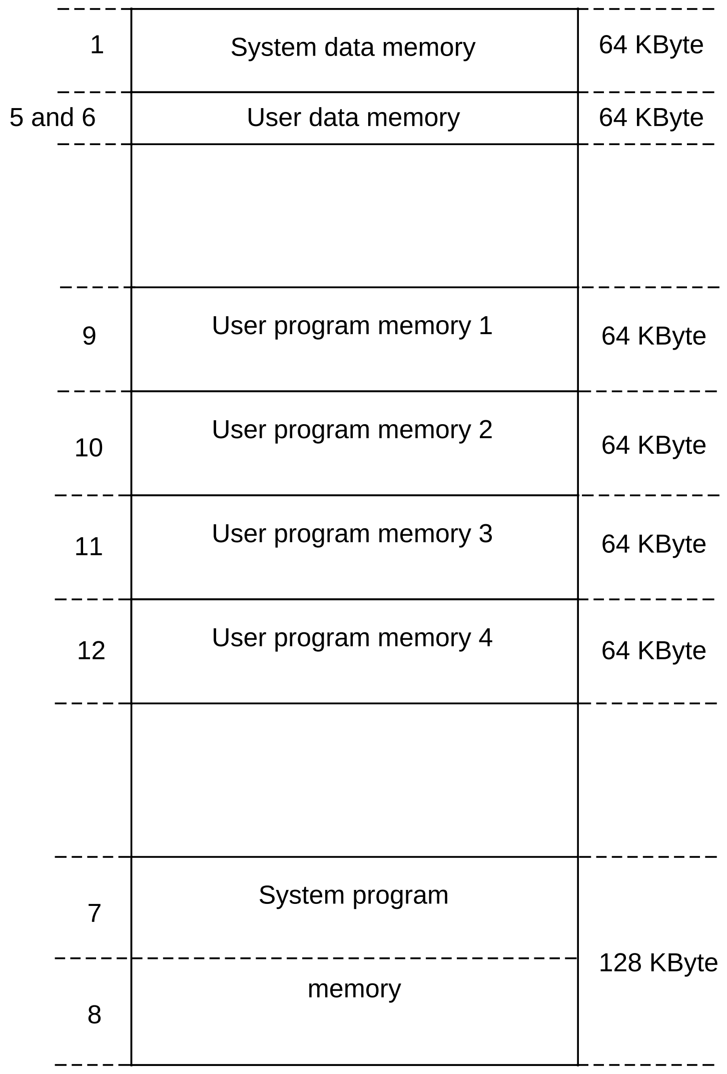


Table 8.2 Memory allocation in the PLC 135 WD interface control

8.2 Segment switch

8.2.1 Changing the segment switch

The following sequence applies to the PG 685 programmer:

- Call the information functions with the F5 key
- Call any memory areas with the F1 key
- Enter the pseudo-address E0000$_{hex}$
- Press the Enter key
- Press the Abort key

The first word shown is the current setting of the segment switch.

- Press the correction key
- Enter the new segment address

8.2.2 Processing data blocks with the segment switch

By setting the segment switch you can choose between the user program memory the target segment of data blocks (DB, DX). If the segment switch is set to AWDSEG, i.e. the content of address E0000$_{hex}$ = 0006, all data blocks inputs from the programmer go to the user data memory.

If the segment switch has another content, i.e. the content of address E0000$_{hex}$ is not equal to 0006 (AWPSEG), all blocks go to the user program memory.

For reasons of compatibility, the value 0006H applies to the user data memory on SINUMERIK 880, software version 6 and higher.

Data blocks, created using the FB11 (EINR-DB) always go to the user data memory.

However, data blocks of FB11 also go to the user program memory on editing if the segment switch is not set to 0006.

Data blocks created using FB11 or edited using the programmer can be edited or erased and can be controlled with a programmer using STEUERN VAR. The initial address of a data block changes when it is read in or compressed.

8.3 Block lists

The start addresses of the block lists can be output by entering pseudo-address F0000$_{hex}$ (see Section ”Detailed error code” for the significance of addresses F0000 to F0009):

F000C	Start address of the OB block list
F000D	Start address of the FB block list
F000E	Start address of the DB block list
F000F	Start address of the FX block list
F0010	Start address of the SB block list
F0011	Start address of the PB block list
F0012	Start address of the DX block list

The following should be observed with regard to output of the block lists via the address list:

- The first address of the address list entry is the offset address; the second one is the segment address.
- High and low bytes are interchanged in the specified address.
- The offset addresses are word-oriented, except that the enteries of the DB list are byte-oriented; these must be divided by two after the high/low swap.

It is important to observe the internal structure of the block list for direct processing of the block list using the instructions LIR and TIR on versions SINUMERIK 880, SW6 and higher and SINUMERIK 840C and higher.

Structure of a block list entry

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Offset address															
	Segment address													D	M/H	

D:	0:	Data block not write-protected	
	1:	Data block write-protected	
M/H:	00:	MC5 block	
	otherwise:	HLL block	

The identification bits 0 to 11 have to be masked for address processing.

9 STEP 5 Operation Set with Programming Examples

9.1 General notes

The STEP 5 operation set is subdivided into basic operations and supplementary operations.

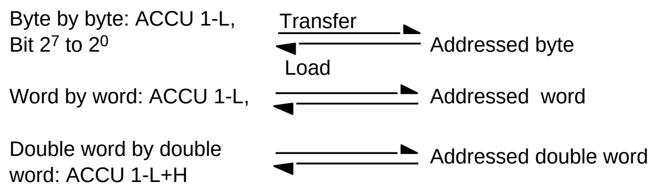
The basic operations are intended for the execution of simple binary functions. As a rule, they can be input**/**output in the three methods of representation (LAD,CSF and STL) of the STEP 5 language on the programmer.

The supplementary operations are intended for complex functions (e.g. control, signalling and logging); they cannot be represented graphically and can only be input/output in the statement list (STL) on the programmer.

Most of the STEP 5 operations use two registers (each 32 bits wide) as the source for the operands and as the destination for the results: Accumulator 1 (ACCU 1) and Accumulator 2 (ACCU 2). Since these registers are not always used or affected in their full width, they are subdivided into smaller units for the following descriptions, as shown below:

Bit significance:	2^{31} ... 2^{16}	2^{15} ... 2^{0}
	ACCU-H	ACCU-L

Load and transfer commands use the contents of ACCU 1 as follows, depending on the addressing (byte, word or double word-oriented):



For load operations, the bit positions of ACCU 1 which are not involved are always filled with zeros. For all load commands, the content of the address is first loaded in ACCU 1. For transfer commands, ACCU 1 and ACCU 2 remain unchanged.

9.1.1 Numeric representation

Numbers in different types of representation are allowed as operands for the STEP 5 commands which operate on or change or compare the contents of ACCU 1 and ACCU 2. Depending on the operation to be executed, the content of ACCU 1 or ACCU 2 is interpreted as one of the following representations:

- **Fixed-point number:** + 0...+ 32 767
  – 1... – 32 768
  The fixed-point number is located in ACCU L and is interpreted as a 16-bit binary number in two's complement representation.

- **Fixed-point double-word:** 0 ... +2 147 483 647
  – 1 ... – 2 147 483 648

  The fixed-point double-word is located in the ACCU and is interpreted as a 32-bit binary number in two's complement representation.

- **Floating-point number:** $m \cdot 2^{exp}$ m = Mantissa
  exp = Exponent
  $\pm 0.1\ 701\ 412 \cdot 10^{39}$
  $\pm 0.1\ 469\ 368 \cdot 10^{-38}$

  The floating-point number is represented as follows in the ACCU (exception: see "Arithmetic operations"):

  Bit significance : $2^{31} ... 2^{24}$ ¦ $2^{23} ... 2^{0}$

  Sign +exp Sign + m

  The exponent is an 8-bit binary number in two's complement representation:

  –128 exp < 127

  The mantissa is 24 bits wide and normalized:

  0.5 positive mantissa < 1
  –1 < negative mantissa –0.5

- **BCD word-coded number with sign and 3 digits:**

  Assignments in ACCU L

  Bit significance: $\underbrace{2^{15}...2^{12}}_{\text{Sign}}$ $\underbrace{2^{11}...2^{8}}_{10^{2}}$ $\underbrace{2^{7}...2^{4}}_{10^{1}}$ $\underbrace{2^{3}...2^{0}}_{10^{0}}$

  The individual numbers are positive 4-bit binary numbers in two's complement representation.

Sign:	0000 if the number is positive
	1111 if the number is negative

- **BCD double word-coded number with sign and 7 digits:**

  Assignments in the ACCU

  Bit significance: $\underbrace{2^{31}...2^{28}}_{\text{Sign}}$ $\underbrace{2^{27}...2^{24}}_{10^{6}}$ $\underbrace{2^{23}...2^{20}}_{10^{5}}$

  $\underbrace{2^{19}...2^{16}}_{10^{4}}$ $\underbrace{2^{15}...2^{12}}_{10^{3}}$ $\underbrace{2^{11}...2^{8}}_{10^{2}}$ $\underbrace{2^{7}...2^{4}}_{10^{1}}$ $\underbrace{2^{3}...2^{0}}_{10^{0}}$

Note:

This internal representation need not comply with the format in which the numbers are entered via the programmer when creating a program. The programmer generates the above representations.

9.1.2 Condition codes of the PLC 135 WB/WB2/WD

There are commands for processing individual bit information, and there are commands for processing word information (8, 16 or 32 bits). In both groups, there are commands which set condition codes and commands which interpret condition codes. For both command groups there are "condition codes for bit operations" and "condition codes for word operations". The CC byte for the PLC 135 is as follows:

2^7 CC for word operations				CC for bit operations 2^0			
CC 1	CC 0	OV	OS	OR		RLO	ER

Condition codes for bit operations:

ER: ER signifies first interrogation. This is the start of a logic operation. ER is set at the end of a logic operation sequence (memory operations).

RLO: Result of logic operation; result of binary operations. Logical value for comparison commands.

OR: This informs the processor that the following AND operations must be handled before an OR operation (AND before OR).

Condition codes for word operations:

OV: OVER; this indicates whether, for the arithmetic operation just terminated, the valid numeric range has been exceeded.

OS: OVER LATCHING; in the course of two or more arithimetic operations, the overflow memory bit serves to indicate whether an overflow error (OVER) has occurred at some time. The bit OS is reset on end-of-block.

CC 1, CC 0 are condition codes whose interpretation can be found in the following table.

Condition codes			Fixed-point calculation: Result	Boo-lean: result	Comparison: Contents of ACCU1 + ACCU 2	Shift: shifted bit	Floating-point calculation: Result
CC 1	CC 0	OVER					
0	0	0	Result = 0	0	ACCU2 = ACCU1	0	Mantissa=0; Exp. allowed
0	1	0	Result < 0	-	ACCU2 < ACCU1	-	Mantissa <0; Exp. allowed
1	0	0	Result > 0	0	ACCU2 > ACCU1	1	Mantissa >0; Exp. allowed
0	0	1	”Over-Null [1)]”	-	-	-	Mantissa=0; Exp.=–128
0	1	1	0 from pos. range	-	-	-	Mantissa <0; Exp.=+127
1	0	1	0 from neg. range	-	-	-	Mantissa >0; Exp.=+127
1	1	1	Division by zero	-	-	-	Division by zero

Jump operations are available for immediate interpretation of the condition codes (see Section "Supplementary operations").

1) *Special case: Greatest negative number added to itself*

9.2 Basic operations

Basic operations are programmable in program, sequence, organization and function blocks. They can be input and output in program, sequence and organization blocks in the three methods of representation (LAD, CSF and STL). In the function blocks, the operations are only displayed in the statement list.

Exceptions:

1. Load, transfer and code operations can only be programmed graphically, indirectly and with limits in conjunction with timing and counting operations.
2. Arithmetic operations and the Stop command (STL) can only be programmed in a statement list.

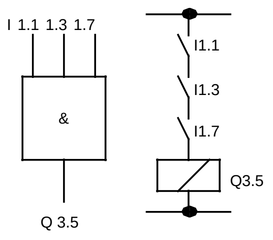
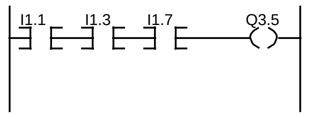
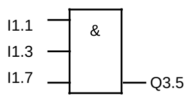
9.2.1 Logic operations, binary

Operation	Parameter	Function
)		Right parenthesis
A(		AND operation with parenthesized expressions
O(		OR operation with parenthesized expressions
O		OR operation on AND functions
A □ □		AND operation with:
O □ □		OR operation with:
I	0.0 to 127.7	Scanning an input for logic 1
Q	0.0 to 127.7	Scanning an output for logic 1
F	0.0 to 255.7	Scanning a flag for logic 1
D	0.0 to 255.15	Scanning data for logic 1
N I	0.0 to 127.7	Scanning an input for logic 0
N Q	0.0 to 127.7	Scanning an output for logic 0
N F	0.0 to 255.7	Scanning a flag for logic 0
N D	0.0 to 255.15	Scanning data for logic 0
T	0 to 255	Scanning a timer for logic 1
N T	0 to 255	Scanning a timer for logic 0
C	0 to 255	Scanning a counter for contents >0
N C	0 to 255	Scanning a counter for contents=0

Binary logic operations produce the "RLO" (result of the logic operation).

At the beginning of a logic sequence, the result depends only on the type of operation (A =AND, AN = AND NOT, O = OR, ON = OR NOT) and the scanned logic level. Within a logic sequence, the RLO is formed from the type of operation, previous RLO and scanned logic level. A logic sequence is terminated by a limited-step command (e.g. storage operations).

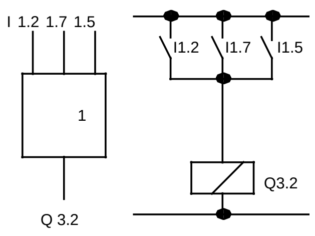
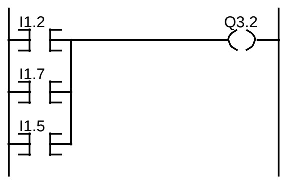
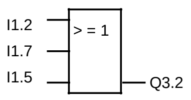
AND operation

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 1.1 A I 1.3 A I 1.7 = Q 3.5		

A logic 1 appears at output Q 3.5 if all inputs are simultaneously at logic 1. A logic 0 appears at output Q 3.5 if at least one of the inputs is at logic 0.

The number of scans and the order of programming are arbitrary.

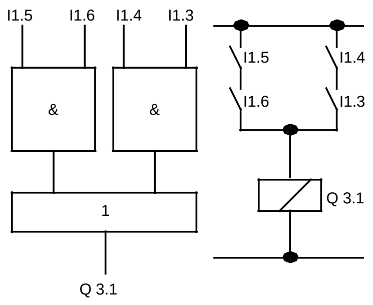
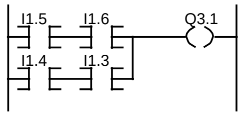
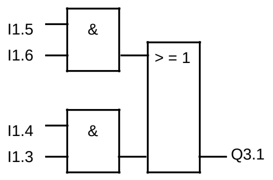
OR operation

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	O I 1.2 O I 1.7 O I 1.5 = Q 3.2		

A logic 1 appears at output Q 3.2 if at least one of the inputs is at logic 1. A logic 0 appears at output Q 3.2 if all inputs are simultaneously at logic 0.

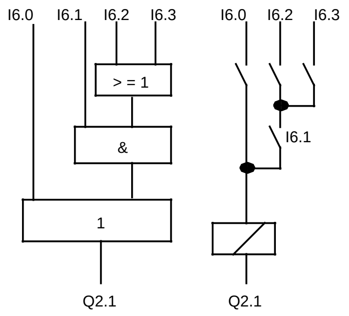
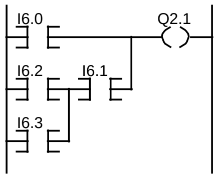
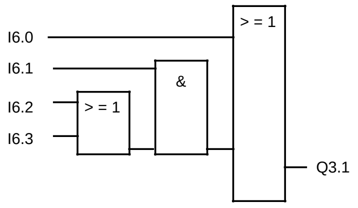
The number of scans and the order of programming are arbitrary.

AND before OR operation

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I1.5 I1.6 I1.4 I1.3 & & 1 Q 3.1 I1.5 I1.4 I1.6 I1.3 Q 3.1	A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q 3.1		

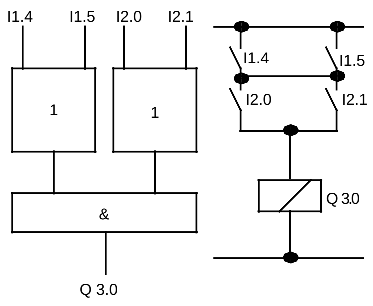
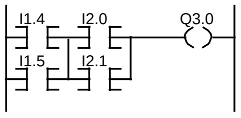
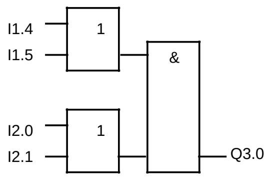
A logic 1 appears at output Q 3.1 if at least one AND condition is fulfilled. A logic 0 appears at output Q 3.1 if **no** AND condition is fulfilled.

OR before AND operation

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I6.0 I6.1 I6.2 I6.3 > = 1 & 1 Q2.1 I6.0 I6.2 I6.3 I6.1 Q2.1	O I 6.0 O A I 6.1 A(O I 6.2 O I 6.3) = Q 2.1		

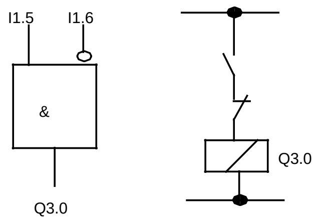
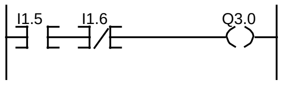
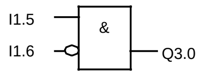
A logic 1 appears at output Q 2.1 if input I 6.0 or input I 6.1 and one of inputs I 6.2 and I 6.3 at logic 1. A logic 0 appears at output Q 2.1 if input I 6.0 is at logic 0 and the AND condition is not fulfilled.

OR before AND operation

Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	A(O I 1.4 O I 1.5) A(O I 2.0 O I 2.1) = Q 3.0		

A logic 1 appears at output Q 3.0 if both OR conditions are fulfilled. A logic 0 appears at output Q 3.0 if at least one OR condition is not fulfilled.

Scanning for logic 0

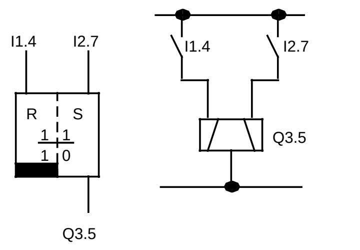
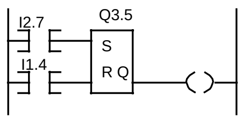
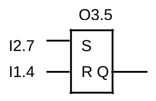
Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	A I 1,5 AN I 1.6 = Q 3.0		

A logic 1 only appears at output Q 3.0 if input I 1.5 is at logic 1 (N/O contact closed) and input I 1.6 is at logic 0 (N/C contact not opened).

9.2.2 Storage operations

Operation		Parameter	Function
S	☐		Set
R	☐		Reset
=	☐		Assign
	▲		
	I	0.0 to 127.7	an input
	Q	0.0 to 127.7	an output
	F	0.0 to 255.7	a flag
	D	0.0 to 255.15	a data word

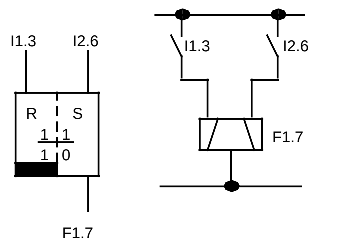
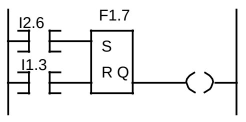
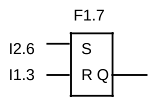
RS flipflop for latching signal output

Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	A I 2.7 S Q 3.5 A I 1.4 R Q 3.5		

A logic 1 at input I 2.7 causes the flipflop to be set. If the logic level at input I 2.7 changes to 0, this state is retained, i.e. the signal is stored. A logic 1 at input 1.4 causes the flipflop to be reset. If the logic level at input I 1.4 changes to 0, this state is retained.

If the set signal (input I 2.7) and reset signal (input I 1.4) are applied simultaneously, the last programmed scan (AI 1.4 in this case) is effective during processing of the rest of the program.

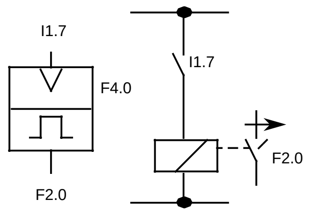
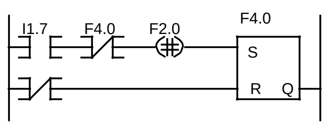
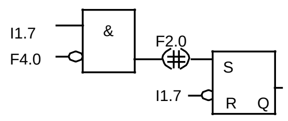
RS flipflop with flags

Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	A I 2.6 S F 1.7 A I 1.3 R F 1.7		

A logic 1 at input I 2.6 causes the flipflop to be set. If the logic level at input I 2.6 changes to 0, this state is retained, i.e. the signal is stored. A logic 1 at input 1.3 causes the flipflop to be reset. If the logic level at input I 1.3 changes to 0, this state is retained.

If the set signal (input I 2.6) and reset signal (input I 1.3) are applied simultaneously, the last programmed scan (AI 1.3 in this case) is effective during processing of the rest of the program.

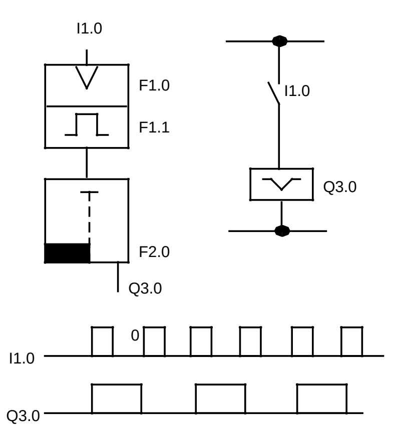
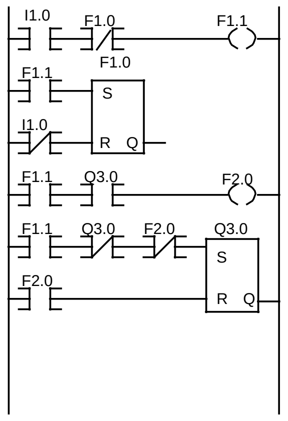
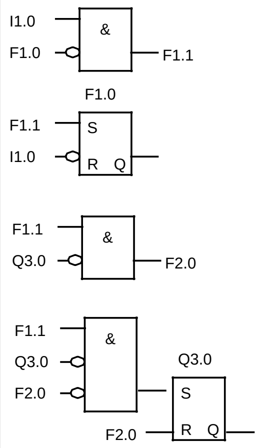
Simulation of a momentary-contact relay

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 1.7 AN F 4.0 = F 2.0 A F 2.0 S F 4.0 AN I 1.7 R F 4.0		

The AND condition (A I 1.7 and AN F 4.0) is fulfilled with each leading edge of input I 1.7; flags F 4.0 ("signal edge flag") and F 2.0 are set when RLO = 1.

With the next processing cycle, the AND condition A I 1.7 and AN F 4.0 are not fulfilled because flag F 4.0 has been set. Flag F 2.0 is reset. Flag F 2.0 is therefore at logic 1 during a single program run.

Binary scaler (trigger circuit)

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I1.0 F1.0 F1.1 F2.0 Q3.0 I1.0 Q3.0 0 I1.0 Q3.0	A I 1.0 AN F 1.0 = F 1.1 A F 1.1 S F 1.0 AN I 1.0 R F 1.0 A F 1.1 A Q 3.0 = F 2.0 A F 1.1 AN Q 3.0 AN F 2.0 S Q 3.0 A F 2.0 R Q 3.0	 I1.0 F1.0 F1.1 F1.1 F1.0 S I1.0 R Q F1.1 Q3.0 F2.0 F1.1 Q3.0 F2.0 Q3.0 S F2.0 R Q	 I1.0 F1.0 & F1.1 F1.0 F1.1 S I1.0 R Q F1.1 Q3.0 & F2.0 F1.1 Q3.0 F2.0 & Q3.0 S F2.0 R Q

The binary scaler (output Q 3.0) changes its logic state with each change of logic level from 0 to 1 (leading edge) of input I 1.0. Half the input frequency therefore appears at the output of the flipflop.

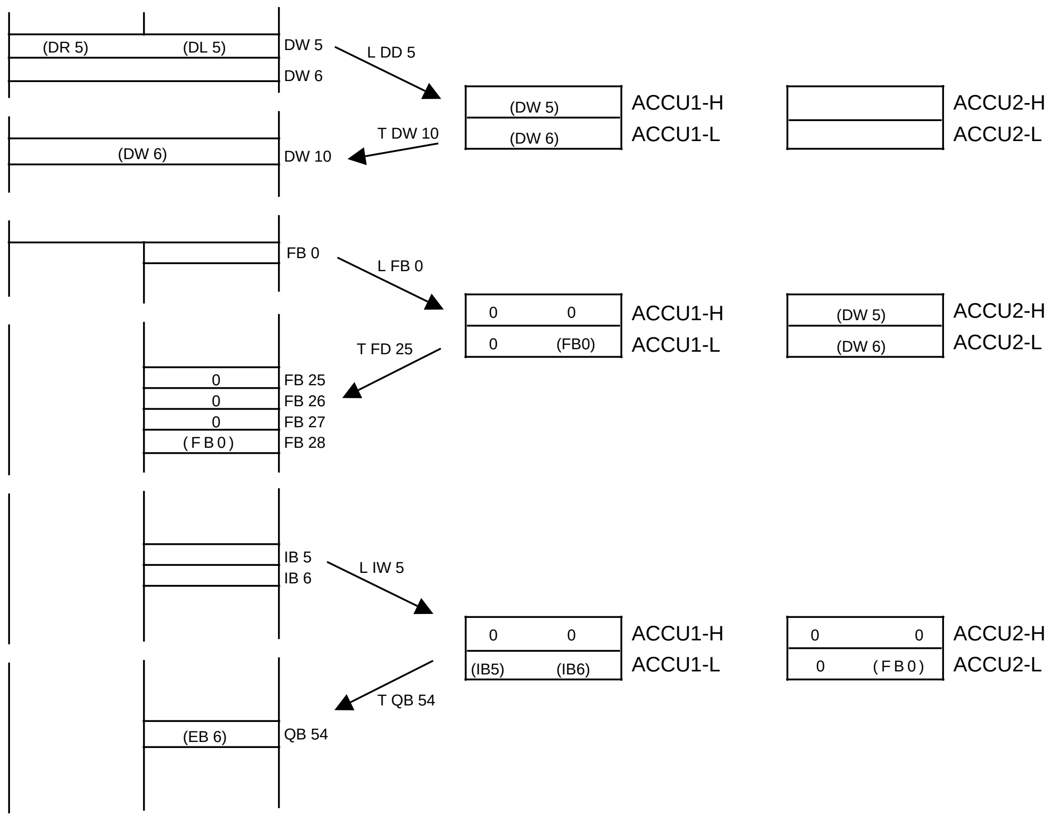
9.2.3 Load and transfer operations

Operation	Parameter	Functions
L □ □		Load
T □ □		Transfer
I B	0 to 127	an input byte from PII 2)
I W	0 to 126	an input word from the PII 2)
I D	0 to 124	an input double word from the PII 2)
Q B	0 to 127	an output byte from the PIQ 3)
Q W	0 to 126	an output word from the PIQ 3)
Q D	0 to 124	an output double word from the PIQ 3)
F B	0 to 225	a flag byte
F W	0 to 254	a flag word
F D	0 to 252	a flag double word
D R	0 to 255	an item of data (right byte)
D L	0 to 255	an item of data (left byte)
D W	0 to 255	a data word
D D	0 to 254	a data double word
T 1)	0 to 255	a time (binary)
C 1)	0 to 255	a count (binary)
P Y	0 to 127	an I/O byte of the digital inputs/outputs with process image
	128 to 255	an I/O byte of the digital or analog inputs/outputs without process image
P W 4)	0 to 126	an I/O word of the digital inputs/outputs with process image
	128 to 254	an I/O word of the digital or analog inputs/outputs without process image
K M 1)	16-bit pattern	a constant as bit pattern
K H 1)	0 to FFFFH	a constant in hex code
K F 1)	–32768 to +32767	a constant as fixed-point number
K Y 1)	0 to 255 for each byte	a constant, 2 bytes
K B 1)	0 to 255	a constant, 1 byte
K S 1)	2 alpha characters	a constant, 2 ASCII characters
K G 1)	$\pm 0.1469368 \cdot 10^{-38}$ to $\pm 0.1701412 \cdot 10^{+39}$	a constant as floating-point number
K T 1)	0.0 to 999.3	a time (constant)
K C 1)	0 to 999	a count (constant)

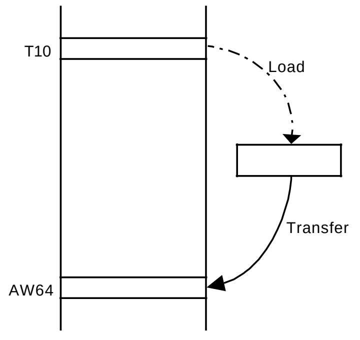
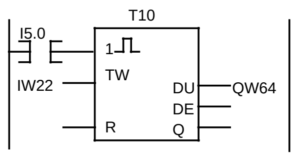
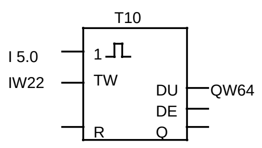
The load and transfer operations are unconditional commands, i.e. they are executed irrespective of the result of the logic operation. The load and transfer operations can only be graphically programmed indirectly in conjunction with time or counting operations, otherwise only in statement lists.

1) Not for transfers
2) PII: Process input image
3) PIQ: Process output image
4) Only even parameters are allowed; error NNP is signalled for odd parameters.

Example: Load and transfer function



Loading and transferring a time (also timing and counting operations)

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 5.0 L IW 22 SP T 10 T QW 64	 T10 I5.0 1 TW IW22 DU DE Q R QW64	 T10 I 5.0 IW22 1 TW DU DE R Q QW64

With graphic input, QW 64 was assigned to output DU of the timer. The programmer then automatically inserts the appropriate load and transfer command in the user program. Thus the contents of the memory location addressed with T 10 are loaded into the accumulator (ACCU 1).

The accumulator contents (ACCU 1) are then transferred to the process image addressed with QW 64.

9.2.4 Timing and counting operations

In order to load a timer or counter with a set command, the value must be loaded in the accumulator beforehand.

The following load operations are expedient: 1)
For timer: L KT, L IW, L QW, L FW, L DW
For counter: L KC, L IW, L QW, L FW, L DW

Operation	Parameter	Function
SP T	0 to 255	Start a timer as a pulse
SE T	0 to 255	Start a timer as an extended pulse
SD T	0 to 255	Start a timer as an ON-delay
SF T	0 to 255	Start a timer as a latching ON-delay
SF T	0 to 255	Start a timer as an OFF-delay
R T	0 to 255	Reset a timer
S C	0 to 255	Set a counter
R C	0 to 255	Reset a counter
CU C	0 to 255	Up counting
CD C	0 to 255	Down counting

Only parameters 0 to 255 are permitted. If a parameter with a higher number is programmed the interpreter outputs error NNP.

Notes:

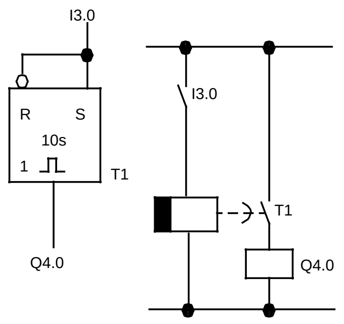
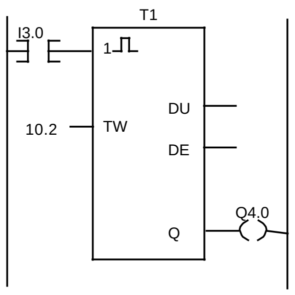
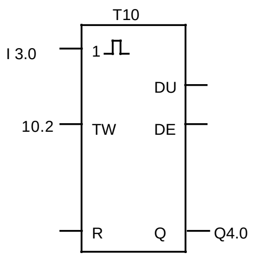
- Since the timer and counter commands are supported by LAD/ACOP, and the latter has no parameter check, it is possible for the signal edge flags to be affected by timers or counters which are not programmed in this command.
- Up to and including SINUMERIK 880, software version 5, up to 128 timers and counters are possible.

Example:

As a result of a DO FW command, command SD T0 (substituted SD T 129) is to be processed. With RLO = 0: Bits ZWG, ZKS, FMS with T 1 are deleted.

1) *Timing or counting operations do not change the contents of ACCU 1.*

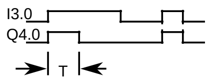
Pulse

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I3.0 R S 10s 1 T1 Q4.0 I3.0 T1 Q4.0	A I 3.0 L KT 10.2 SP T 1 A T 1 = Q 4.0		 T10 I 3.0 1 DU 10.2 TW DE R Q Q4.0

With the first execution, the timer is started if the result of the logic operation is 1. If execution is repeated with RLO = 1, the timer is unchanged. If the RLO = 0 the timer is set to zero (cleared).

Scans A T and O T result in a logic 1 as long as the timer is still running.

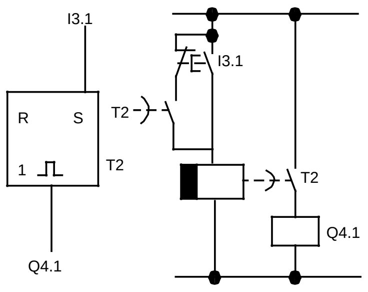
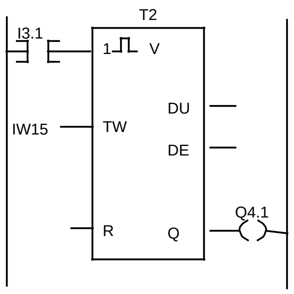
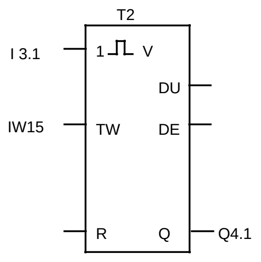
DU and DE are digital outputs of the timer. The time is present with the timebase, binary-coded at output DU and BCD-coded at output DE.



KT 10.2:
The specified value (10) is loaded in the timer. The number to the right of the point specifies the timebase:

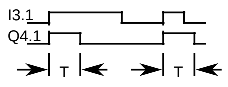
0 = 0.01 s	2 = 1 s
1 = 0.1 s	3 = 10 s

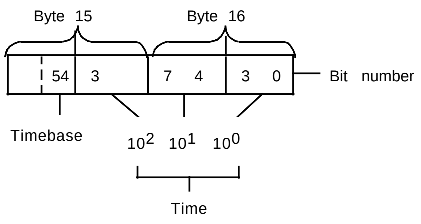
Extended pulse

Given circuit	STEP 5 representation		
	Statement list	Ladder diagramn	Control system flowchart
	A I 3.1 L IW15 SE T 2 A T 2 = Q 4.1		

With first execution, the timer is started if the result of the logic operation is 1. If the RLO = 0 the timer is unchanged.

Scans AT or OT result in a logic 1 as long as the timer is still running.

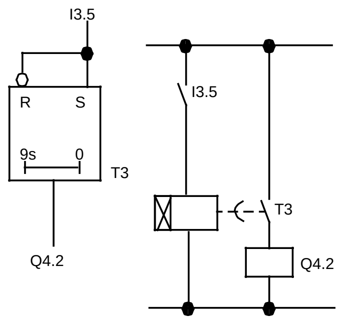
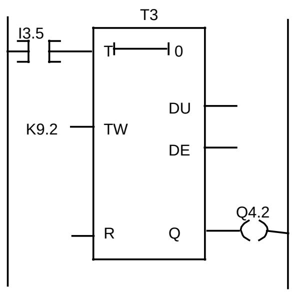
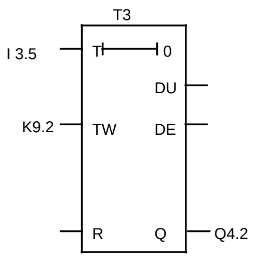




IW 15:

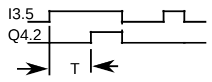
Setting the time with the value of operands I, Q, F or D present in BCD code (input word 15 in the example).

ON-delay

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 3.5 L KT 9.2 SD T 3 A T 3 = Q 4.2		

With the first execution, the timer is started if the result of the logic operation is 1. If execution is repeated and the RLO = 1, the timer is unchanged. If the RLO = 0 the timer is set to zero (cleared).

Scans A T or O T result in a logic 1 if the time has elapsed and the result of the logic operation is still present at the input.

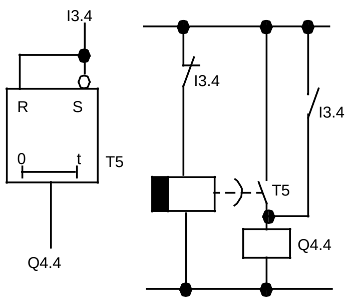
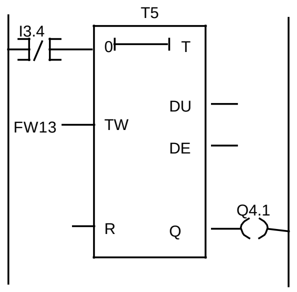
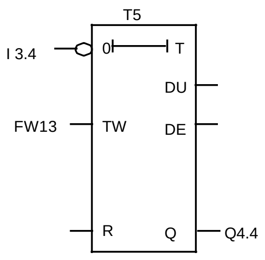


KT 9.2:

The specified value (9) is loaded in the timer.
The number to the right of the point specifies the timebase:

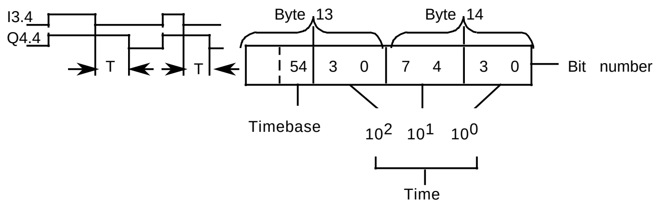
0 = 0.01 s	2 = 1 s
1 = 0.1 s	3 = 10 s

OFF-delay

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	AN I 3.4 L FW13 SF T 5 A T 5 = Q 4.4		

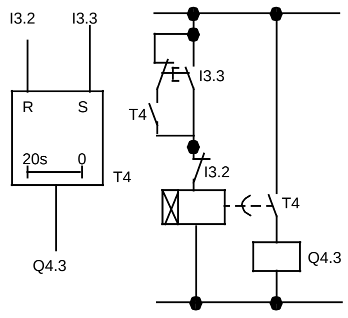
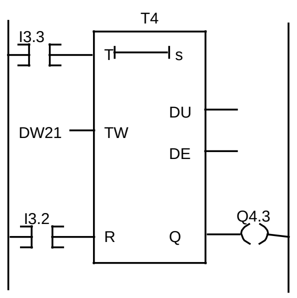
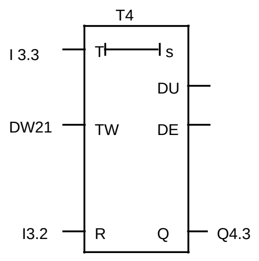
With the first execution, the timer is started if the result of the logic operation is 0. If execution is repeated and the RLO = 0, the timer is unchanged. If the RLO = 1 the timer is set to zero (cleared).

Scans AT and OT result in a logic 1 if the time is still running or the RLO is still present at the input.



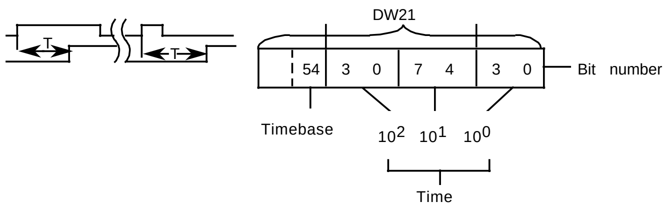
FW 13:
Setting of the time with the value of operands I, Q F or D present in BCD code (flag word 13 in the example).

Latching ON-delay

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I3.2 I3.3 R S 20s 0 T4 Q4.3 I3.3 T4 I3.2 T4 Q4.3	A I 3.3 L DW21 SS T 4 A I 3.2 R T 4 A T 4 = Q 4.3	 T4 I3.3 T s DW21 TW DU DE I3.2 R Q Q4.3	 T4 I 3.3 T s DU DW21 TW DE I3.2 R Q Q4.3

With the first execution, the timer is started if the result of the logic operation is 1. If the RLO = 0, the timer is unchanged. Scans A T and O T result in a logic 1 if the time has elapsed.

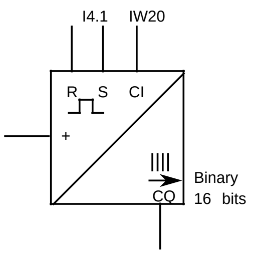
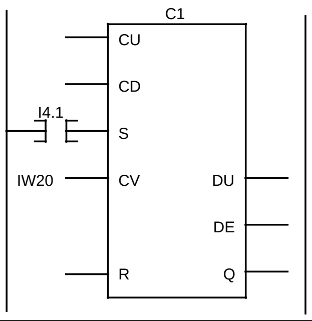
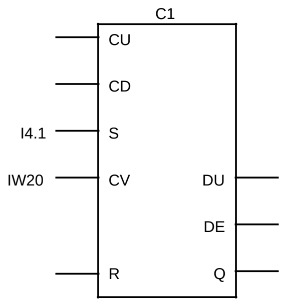
The logic level only goes to 0 when the timer has been reset with function RT.



DW 21:

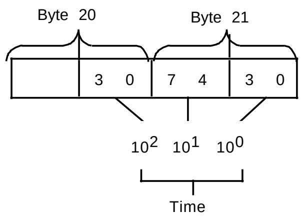
Setting the time with the value of operands I, Q, F or D present in BCD code (data word 21 in the example).

Setting a counter

Given circuit	STEP 5 representation Statement list	 Ladder diagram	 Control system flowchart
	A I 4.1 L IW20 S C 1		

With the first execution, the counter is set if the result of the logic operation is 1. If execution is repeated, the counter is unchanged (irrespectively of whether the RLO is 1 or 0). With repeated first execution with RLO = 1, the counter is set again (signal edge decoding). DU and DE are digital outputs of the counter. The count is present in binary code at output DU, and BCD-coded at output DE.

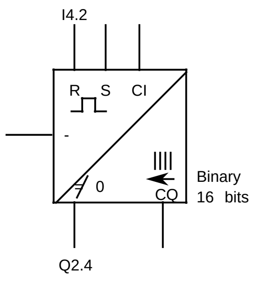
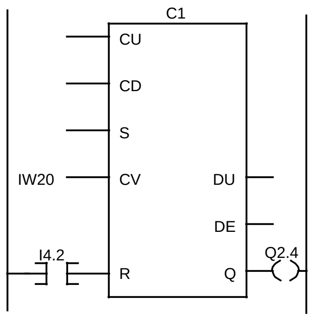
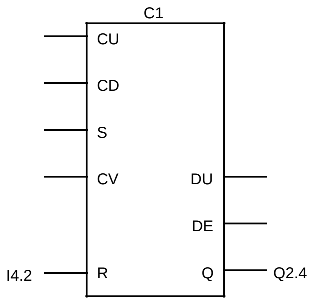
The flag required for signal edge decoding of the set input is also present in the count word.



IW 20:

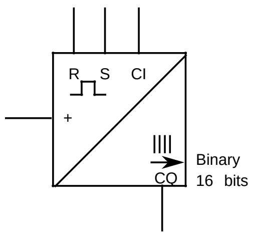
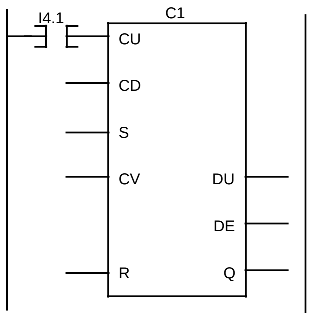
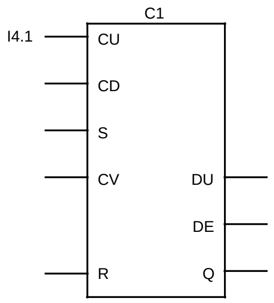
Setting a counter with the value of operands I, Q, F or D present in BCD code (input word 20 in the example).

Resetting a counter

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I4.2 R S CI - 0 CQ Binary 16 bits Q2.4	A I 4.2 R C 1 A C 1 = Q 2.4	 C1 CU CD S IW20 CV DU DE I4.2 R Q Q2.4	 C1 CU CD S CV DU DE I4.2 R Q Q2.4

If the result of the logic operation is 1 the counter is set to zero (cleared).
If the result of the logic operation is 0 the counter is unchanged.

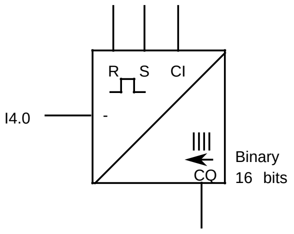
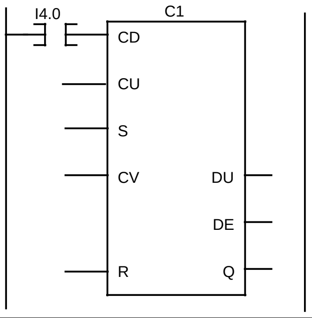
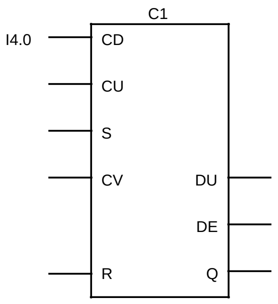
Up counting

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.1 CU C 1		

The value of the addressed counter is incremented by 1. Function CU (count up) is only executed with a positive-going edge (from 0 to 1) of the logic operation programmed before CU. The flags required for signal edge decoding of the count inputs are also contained in the count word.

A counter with two different inputs can be used as an up or down counter by means of the two separate signal edge flags for CU (count up) and CD (count down).

Down counting

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.0 CD C 1		

The value of the addressed counter is decremented by 1. The function only becomes effective with a positive-going edge (from 0 to 1) of the logic operation programmed before CD. The flags required for signal edge decoding of the count inputs are also in the count word.

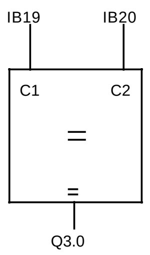
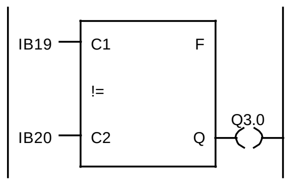
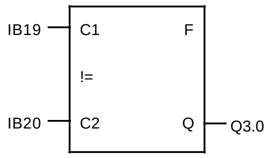
A counter with two different inputs can be used as an up or down counter by means of the two separate signal edge flags for CU (count up) and CD (count down).

9.2.5 Comparison operations

The comparison operations compare the content of Accumulator 1 with the content of Accumulator 2. The values to be compared must therefore first be stored in the accumulators, e.g. with load operations. The accumulator contents remain unchanged during the comparison.

Operation	Parameter	Function
!= □	(none)	Compare for equal
>< □		Compare for not equal
> □		Compare for greater
>= □		Compare for greater than or equal to
< □		Compare for less
<= □		Compare for less than or equal to
↑		
F		Two fixed-point numbers
G		Two floating-point numbers
D		Two fixed-point double-word numbers

Compare for equal

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L IB19 L IB20 != F = Q 3.0		

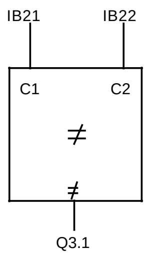
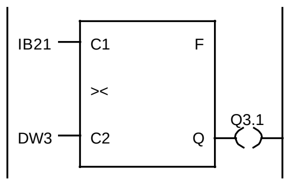
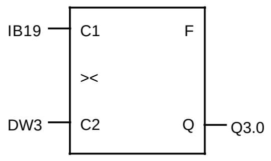
The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.

RLO = 1: Comparison is fulfilled, ACCU 1-L = ACCU 2-L
RLO = 0: Comparison is not fulfilled, ACCU 1-L   ACCU 2-L

ACCU 2-H and ACCU 1-H are not involved in the operation with the fixed point comparison. The numeric representation of the operands (fixed-point calculation) has to be considered in programming the comparison operation.

0	IB19	ACCU 2-L
0	IB19	ACCU 1-L

Compare for not equal

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L IB21 L DW3 >< F = Q 3.1		

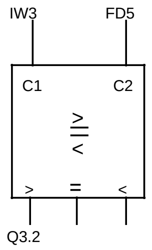
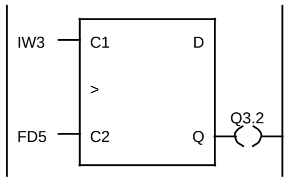
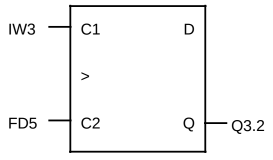
The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.

RLO = 1: Comparison is fulfilled, ACCU 1-L   ACCU 2-L
RLO = 0: Comparison is not fulfilled, ACCU 1-L = ACCU 2-L

ACCU 2-H and ACCU 1-H are not involved in the operation with the fixed point comparison. The numeric representation of the operands (fixed-point calculation in this case) has to be considered in programming the comparison operation.

0	IB21	ACCU 2-L
DW3		ACCU 1-L

Compare for greater

Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	L IW3 L FD5 > D = Q 3.2		

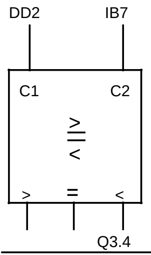
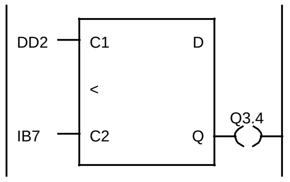
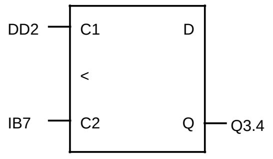
The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.

RLO = 1: Comparison is fulfilled, ACCU 2 > ACCU 1
RLO = 0: Comparison is not fulfilled, ACCU 2   ACCU 1

ACCU2-H	0	0	IB3	IB4	ACCU2-L
ACCU1-H	FB5	FB6	FB7	FB8	ACCU1-L

The numeric representation of the operands has to be considered in programming the comparison operation. The contents of ACCU 1 and ACCU 2 are interpreted as fixed-point numbers with double-word width.

Compare for less

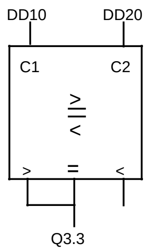
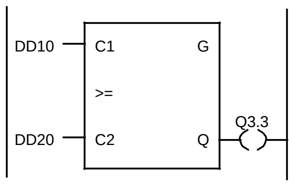
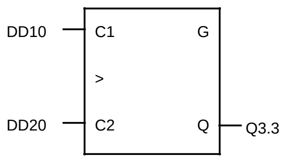
Given circuit	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	L DD2 L IB7 < D = Q 3.4		

The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.
RLO = 1: Comparison is fulfilled, ACCU 2 < ACCU 1
RLO = 0: Comparison is not fulfilled, ACCU 2   ACCU 1

ACCU2-H	DW2		DW3		ACCU2-L
ACCU1-H	0	0	0	IB7	ACCU1-L

The numeric representation of the operands has to be considered in programming the comparison. The contents of ACCU 1 and ACCU 2 are interpreted as fixed-point numbers with double-word width.

Compare for greater than or equal to

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L DD10 L DD20 >=G =		

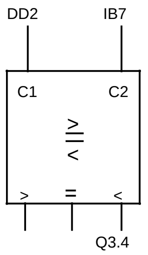
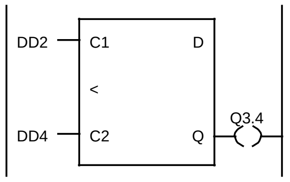
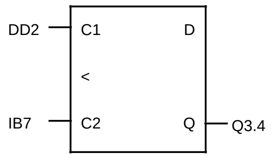
The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.

RLO = 1: Comparison is fulfilled, ACCU 2   ACCU 1
RLO = 0: Comparison is not fulfilled, ACCU 2 < ACCU 1

ACCU2-H	DW10		DW11		ACCU2-L
ACCU1-H	DW20		DW21		ACCU1-L

The numeric representation of the operands has to be considered in programming the comparison operation. The contents of ACCU 1 and ACCU 2 are interpreted as a floating-point number.

Compare for less than or equal to

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L DD2 L DD4 < G = Q 3.4		

The operand first specified is compared with the next operand according to the comparison function. The comparison results in a binary result of the logic operation.

RLO = 1: Comparison is fulfilled, ACCU 2   ACCU 1
RLO = 0: Comparison is not fulfilled, ACCU 2 > ACCU 1

ACCU2-H	DW1		DW2		ACCU2-L
ACCU1-H	DW4		DW5		ACCU1-L

The numeric representation of the operands has to be considered in programming the comparison operation. The contents of ACCU 1 and ACCU 2 are interpreted as a floating-point number.

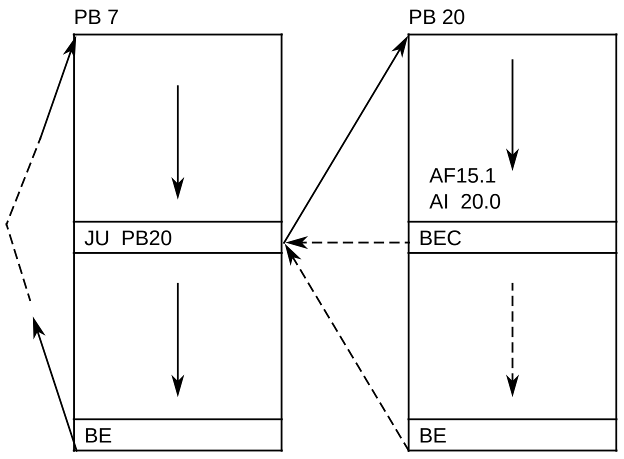
9.2.6 Block calls

Operation	Parameter	Function
JU ☐☐ **JC** ☐☐ ↑ ↑ **P B** **F B** **S B** **O B**	 0 to 255 0 to 255 0 to 255 180	Unconditional jump Conditional jump (depending on the RLO) to a program block to a function block (type FB) to a sequence block to an organization block OB 180
DO ☐☐ **DOC** ☐☐ ↑ ↑ **F X**	 0 to 255	Unconditional jump Conditional jump (depending on the RLO) to a function block (type FX)
C **D B** **CX** **D X**	2 to 255 2 to 255	Data block call DX data block call
BE **BEC** **BEU**		Block end Block end, conditional (depending on RLO) Block end, unconditional

Command C DB and CX DX (data block call) is explained under "Calling data blocks" (see Section ”Data blocks”).

Commands BE (block end) and BEC (block end, conditional) result in a return to the calling block. Command BE must be programmed at the end of each data block (except for DB, DX).

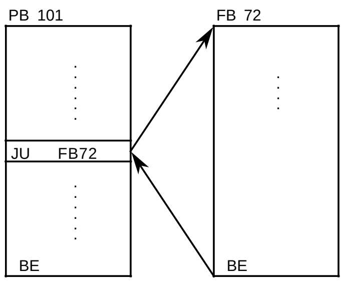
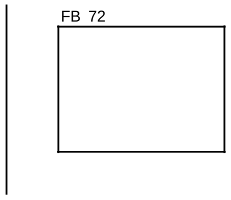
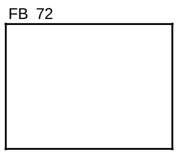
Example:



If the result of the logic operation is 1, the return to PB7 already takes place with processing of the BEC command.

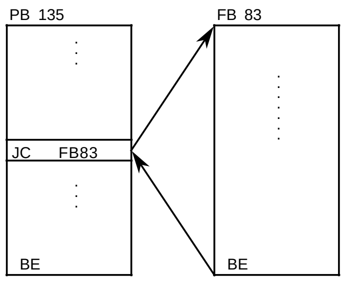
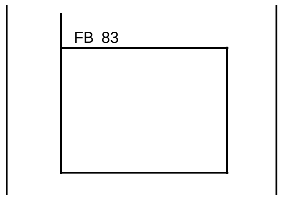
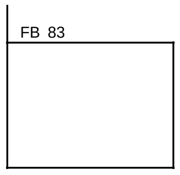
If the result of the logic operation is not equal to 0, processing of PB20 continues up to the BE command, which then initiates the return to PB7 when PB20 has been fully processed.

Unconditional call for a function block

Given circuit	STEP 5 representation		
	Statement-list	Ladder diagram	Control system flowchart
	. . . JU FB 72 . . .		

The unconditional function block call is entered at the desired program point. With graphic methods of representation LAD and CSF, the called block (FB, FX) is represented as a box.

Conditional call for a program block

Given circuit	STEP 5 representation		
	Statement-list	Ladder diagram	Control system flowchart
	. . . . . . JC FB83 . . .		

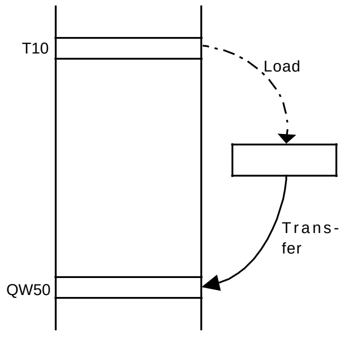
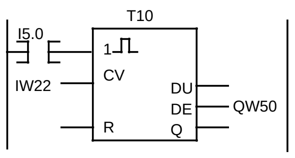
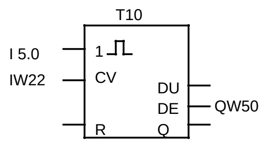
At the desired program point, the conditional block call is entered after the appropriate logic operation. If the result of the logic operation is 1, a jump to the specified block takes place. If the condition is not fulfilled, the jump is not executed. With graphic methods of representation LAD and CSF, the called block is represented as a box.

9.2.7 Code operations

The code operations allow a time or count, which is present in binary from, to be loaded as a code in the accumulator; the corresponding value is still available in BCD form for further processing.

Operation	Parameter	Function
LC □		Load as code
T	0 to 255	times
C	0 to 255	counts

Loading a time (coded)

Given circuit	STEP 5 representation		
	Statement-list	Ladder diagram	Control system flowchart
	A I 5.0 L IW 22 SP T 10 LC T 10 T QW50	 T10 I5.0 1 CV IW22 DU DE R Q QW50	 T10 I 5.0 1 IW22 CV DU DE R Q QW50

The content of the memory location addressed with T 10 is loaded coded into the accumulator.

The subsequent transfer operation transfers the content from the accumulator to the memory location of the process images addressed with QW50. With the graphic methods of representation LAD and CSF a coding operation can only take place indirectly as a result of the assignment of output DE of a timer or counter. With method of representation STL, however, this command can be isolated.

9.2.8 Arithmetic operations

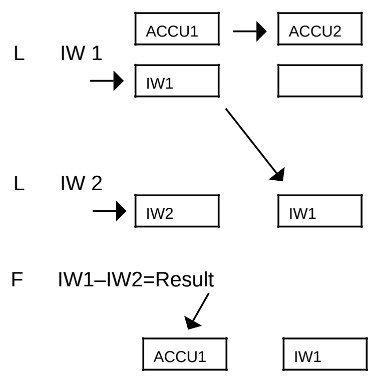
Arithmetic operations can only be represented in the statement list.

They process the contents of Accumulators 1 and 2. Suitable load operations, for example, are required.

Operation	Parameter	Function
+ □	(none)	Addition
- □		Subtraction
x □		Multiplication
: □		Division
↑		
F		of two fixed-point numbers
G		of two floating-point numbers

By means of two load operations, ACCU 1 and ACCU 2 can be loaded according to the operands of the load operations. Arithmetic operations can then be executed with the contents of both accumulators.

Example:



The subsequent transfer operation transfers the result stored in ACCU 1 to the operand issued for the transfer operation. If, during calculation with fixed-point numbers, an overflow occurs (OV = 1) ACCU 1-H is cleared.

In the multiplication and division of floating-point numbers, only a 16-bit mantissa is used for the calculation. The result is reduced precision:

Multiplication:	At least 12 bits of the mantissa are exact
Division:	At least 11 bits of the mantissa are exact

In the subtraction of floating-point numbers, Bit 24 may be incorrect if the difference between the two exponents is greater than 24.

9.2.9 Other operations

The following operations can only be represented in a statement list.

Operation	Parameter	Function
STP		Stop
NOP 0		No operation (all bits cleared)
NOP 1		No operation (all bits set)
BLD	0 to 255	Screen command

The STOP command is used, for example, when the PLC is required to go to the stop state in the event of certain critical states of the system or when a device error occurs.

The no-operations serve, for example, for keeping memory locations free or overwriting them.

The screen command governs the subdivision of progam parts into segments within a block. It is automatically stored in the program by the programmer and is treated as a no-operation by the interface controller.

9.3 Supplementary operations (FBs, FXs only)

Function blocks can be programmed with an operation set which is extended compared to the program blocks. The full operation set for function blocks comprises the basic operations and the supplementary operations.

With the function blocks, the operations are only represented in a statement list. The programs of the function blocks therefore cannot be programmed in graphic form (CSF or LAD).

Described in the following are the supplementary operations. Possibilities of combination of the substitution commands with the actual operands are also given.

9.3.1 Logic operations, binary

Operation	Description
A = ☐	**AND function** to test a formal operand for logic 1
AN= ☐	**AND function** to test a formal operand for logic 0
O = ☐	**OR function** to test a formal operand for logic 1
ON= ☐	**OR function** to test a formal operand for logic 0
	Insert formal operand The actual operands allowed are binary addressed inputs, outputs and flags (parameters: I, O; parameter type DI) as well as timers and counters (parameters T, C).

Example:

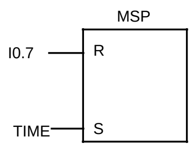
```
: A  =   ON
: AN =   STOP
: AN =   END
: O  =   AMNT
:=   =   RUN
```

9.3.2 Setting operations

Operation	Description
S = ☐	Set (binary) a formal operand
RB= ☐	Reset (binary) a formal operand
== ☐	Assign the result of the logic operation to a formal operand
	Insert formal operand The actual operands allowed are binary addressed inputs, outputs and flags (parameters: I, Q; parameter type DI).

Example:

```
: A  =   I0.7
: RB =   MSP
: A  =   TIME
: S  =   MSP
```



9.3.3 Timing and counting operations

Operation	Description
RD= ☐	Reset (digital) a formal operand (parameters: T,C)
SP= ☐	Start a time, preset as formal operand, with the value stored in the accumulator as a pulse (parameter: T)
SR= ☐	Start a time, preset as formal operand, with the value stored in the accumulator as an on-delay (parameter: T)
SEC= ☐	Start a time, preset as formal operand, with the value stored in the accumulator as an extended pulse; or set a counter, preset as formal operand, with the following, specified count (parameters: T, C)
SSV= ☐	Start a time, preset as formal operand, with the value stored in the accumulator as a latching on delay, or up-counting of a counter specified as formal operand (parameters: T, C)
SFD= ☐	Start a time, preset as formal operand, with the value stored in the accumulator as an off-delay, or down-counting of a counter preset as formal operand (parameters: T, C)
↑	**Insert formal operand** The actual operands allowed are timers and counters; exception: Timers only with SP and SR. The time or count can be specified as follows, as for the basic operations or as a formal operand: Set the time or count with the value present in BCD code of operands IW, QW, FW, DW (parameters: I; parameter type: W) specified as formal operands, or as data (parameter: D; parameter type: KT, KC).

Notes:

The following timers and counters are available to the user:

SINUMERIK 880 SW3 and 4:	128 timers, 128 counters
SINUMERIK 880 GA2, SW1/840C:	255 timers, 255 counters

Examples:

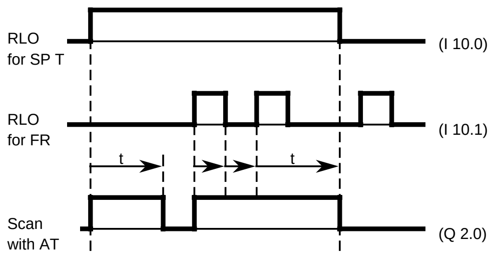
Function block call	Program in function block	Program executed
: JU FB203 NAME : EXAMPLE ANNA : I 10.3 BERT : T 17 JACK : Q 18.4	: A = ANNA : L KT 010.2 : SSU = BERT : A = BERT := = JACK	: A I 10.3 : L KT 010.2 : SS T 17 : A T 17 := Q 18.4
: JU FB204 NAME : EXAMPLE MAXI : I 10.5 IRMA : I 10.6 EVA : I 10.7 DORA : C 15 EMMA : F 58.3	: A = MAXI : SSU = DORA : A = IRMA : SFD = DORA : A = EVA : L KZ100 : SEC = DORA : AN = DORA := = EMMA	: A I 10.5 : CU C 15 : A I 10.6 : CD C 15 : A I 10.7 : L KC100 : S C 15 : AN C 15 := F 58.3
: JU FB205 NAME : EXAMPLE CURT : I 10.4 CARL : T 18 PETE : IW20 CUTE : F 100.7	: A = CURT : L = PETE : SEC = CARL := = CUTE	: A I 10.4 : L IW20 : SE T 18 := F 100.7

9.3.4 Enabling operations for timing and counting operations

Operation	Description
FR T 0 to 255	**Enabling a time for a restart** The operation is only executed with a leading edge of the result of the logic operation. It initiates a restart of the time if the RLO present is 1 for the start operation.
FR C0 to 255	**Enabling a counter** The operation is only executed with a leading edge of the result of the logic operation. It initiates setting, up or down counting of the counter if the RLO present is 1 for the corresponding operation.
FR= ☐	**Enabling a formal operand a restart,** parameters (T, C)

Example:

```
: A     I 10.0
: L     KT 500.0
: SP    T 10
: A     I 10.1
: FR    T 10
: A     T 10
:=      Q 2.0
```



9.3.5 Bit test operations (FB, FX only)

Operation	Parameter	Function
TB ☐		Test the bit for logic 1
TBN ☐		Test the bit for logic 2
SU ☐		Set bit unconditionally
RU ☐		Reset bit unconditionally
I	0.0 to 127.7	an input
Q	0.0 to 127.7	an output
F	0.0 to 255.7	a flag
C	0.0 to 255.15	a count word
T	0.0 to 255.15	a time word
D	0.0 to 255.15	a data word

Operations "P" and "PN" are scans. They test a bit of the operand specified in the following, and then insert the result of the logic operation irrespective of previous scans and the previous status.

Operation	Logic level of the bit in the specified operand	Result of logic operation
TB	0	0
	1	1
TBN	0	1
	1	0

The RLO formed in this way can be subjected to further logic operations. However, a bit test operation must always be positioned at the beginning of a logic operation.

1st example:

The logic level of the 10th bit of data word 205 is ANDed with the logic level of input I 13.7.

```
: C    DB 200
: TB   D 205.10
: A    I 13.7
:=     F 210.3
```

Operations "SU" and "RU" are executed independently of the result of the logic operation. When this operation has been processed, the addressed bit in the specified operand is set to logic 1 (for SU) or logic 0 (for RU).

2nd example:

The third bit is to be set by DW 55 and the 9th bit by DW 103.

```
: SU  D 55.3
: RU  D 103.9
```

9.3.6 Load and transfer operations

Operation	Description
L = ▭	**Load a formal operand** The value of the operand specified as a formal operand will be loaded into the ACCU (parameters: I, Q; parameter type: BY W, D).
LD= ▭	**Load a formal operand as code** The value of the timer or counter specified as a formal operand will be loaded into the ACCU in BCD form (parameters: T, C).
LW= ▭	**Load the bit pattern of a formal operand** The bit pattern of the formal operand will be loaded into the ACCU (parameter : D; parameter type: KF, KH, KM, KY, KS, KT, KC).
LDW= ▭	**Load the bit pattern of a formal operand** The bit pattern of the formal operand will be loaded into the ACCU (parameter: D; parameter type: KG).
T = ▭	**Transfer to a formal operand** The accumulator content will be transferred to the operand specified as formal operand (parameters: I,Q; parameter type: BY, W, D).
↑	**Insert formal operand** The operands corresponding to the basic operations are allowed as actual operands. The data allowed for LW is in the form of a binary pattern, hex pattern, two-byte absolute numbers, characters, fixed-point number, times and counts. For LD, a floating-point number is allowed as data.

Example:

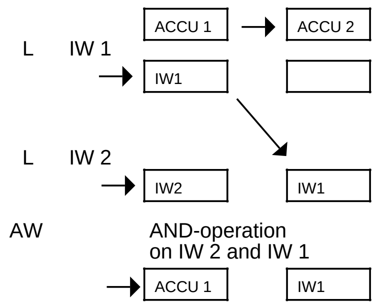
Function block call	Program in function block	Program execution
: JU FB206 NAME : COMP. C1 : KH7F0A C2 : DW20	 : LW =C1 : L =C2 : !=F	 : L KH7F0A : L DW20 : !=F

9.3.7 Logic operations, digital

Operation	Description
AW	AND operation, digital, ACCU 1 and ACCU 2
OW	OR operation, digital, ACCU 1 and ACCU 2
XOW	Exclusive OR operation, digital, ACCU 1 and ACCU 2

ACCU 1 and ACCU 2 can be loaded according to the operands of the load operation, by means of two load operations. The contents of both accumulators can then be subjected to a digital operation.

Example:



The subsequent transfer operation transfers the result stored in ACCU 1 to the operand specified with the transfer operation.

9.3.8 Shift operations

Operation	Description
SLW 0 to 15	Shift left (zeros are shifted in from the right)
SRW 0 to 15	Shift right (zeros are shifted in from the left)
SSW 0 to 15	Shift right with sign (the sign is shifted in from the left)
SLD 0 to 32	Shift left, double word (zeros are shifted in from the right)
SSD 0 to 32	Shift right with sign, double word (the sign is shifted in from the left)

The shift functions are executed independently of conditions. The last bit to be shifted can be scanned with jump functions. JZ can be used for the jump if the bit is 0, and JN or JC if the bit is 1.

Example:

```
STEP 5 program:     Contents of data words

:L     DW52         H = 14AF
:SLW    4
:T     DW53         H = 4AF0
```

9.3.9 Conversion operations

Operation	Meaning
CFW	One's complement of ACCU 1-L
CSW	Two's complement of ACCU 1-L
CSD	Two's complement of ACCU 1
DEF	Conversion of BCD word-coded number to fixed-point number
DUF	Conversion of fixed-point number to BCD word-coded number
DED	Conversion of BCD double-word-coded number to fixed-point double-word
DUD	Conversion of fixed-point double-word to BCD double-word-coded number
FDG	Conversion of fixed-point double-word to floating-point number
GFD	Conversion of floating-point number to fixed-point double-word

Examples:

The contents of data word 64 are to be inverted bit for bit and stored in data word 78.

```
STEP 5 program:          Assignments of data words:

: L    DW64              KM = 0011111001011011
: CFW
: T    DW78              KM = 1100000110100100
```

The contents of data word 207 are to be interpreted as a fixed-point number and stored in data word 51 with the opposite sign.

```
STEP 5 program:          Assignments of data words:

: L    DW207             KF: +51
: CSW
: T    DW51              KF:  – 51
```

9.3.10 Decrementing and incrementing

Operation	Description
D 0 to 255 **I 0 to 255**	Decrementing Incrementing Accumulator contents 1 are decremented/incremented by the number specified in the parameter. Execution of the operation is independent of conditions. It is restricted to the right byte (without carry).

Example:

```
STEP 5 program:          Assignments of data words:

: L    DW7               KH = 1010
: I      16
: T    DW8               KH = 1020
: D      33
: T    DW9               KH = 10FF
```

9.3.11 Jump operations

The jump destination for unconditional and conditional jumps is specified symbolically (maximum of 4 characters): the symbolic parameter of the jump command is identical with the symbolic address of the statement to be jumped to. When programming, ensure that the unconditional jump distance is not more than ± 127 words. If should be noted that a STEP 5 statement must not comprise more than one word. Jumps may only be executed within a block. Jumps extending beyond networks are not allowed.

Operation	Description
JU = ☐	**Jump, unconditional** The unconditional jump will be executed independently of conditions.
JC = ☐	**Jump, conditional** The conditional jump will be executed if the result of the logic operation is 1. If the RLO is 0 the jump will not be executed and the RLO will be set to 1.
JZ = ☐	**Jump if accumulator content is zero** The jump will be executed if the accumulator content is zero. If the accumulator content is not zero the jump will not be executed. The RLO will not be changed.
JN = ☐	**Jump if accumulator content is not zero** The jump will be executed if the accumulator content is not zero. If the accumulator content is zero the jump will not be executed. The RLO will not be changed.
JP = ☐	**Jump if accumulator content is positive** The jump will be executed if the accumulator content is greater than zero. If the accumulator content is zero or less than zero, the jump will not be executed. The RLO will not be changed.
JM = ☐	**Jump if accumulator content is negative** The jump will be executed if the accumulator content is less than zero. If the accumulator content is zero or greater than zero, the jump will not be executed. The RLO will not be changed.
JO = ☐	**Jump if overflow** The jump will be executed in the event of an overflow. If there is no overflow, the jump will not be executed. The RLO will not be changed.

Operation	Description
JOS = $\boxed{}$	**The jump will be executed if "Overflow stored" is set (OS = 1)** Otherwise (OS = 0) the jump will not be executed. "Overflow stored" will be set for arithmetic operations in the event of an overflow, and will remain stored until the arithmetic operation is interrupted. An overflow exists when, with numeric representation, the permissible range is exceeded by an arithmetic operation.
	Insert symbolic address (maximum of 4 characters)

The conditional jump operations (all except for JU) are executed in accordance with the RLO and the indicators in the control unit of the PLC.

Note:

The jump statement and jump destination must be located in a network. Only one symbolic address is allowed for jump destinations per network.

Example:

```
      : JU      =  FORT
      :
FORT  : A       =  STOP
      : A       =  END
      :
ZIEL  : O       Q 7.3
      : O       F 16.6
      :
      : O       =  BETR
      : JC      =  ZIEL
```

Example (comparison operations):

```
      : L          DW 67
      : L          DW 107
      : !=F
      : JC      =  GLCH    (Jump if equal, JZ can also be programmed)
      : JM      =  KLNR    (Jump if less)
      : JP      =  GRSS    (Jump if greater)
      :
KLNR  :
      :
GLCH  :
      :
GRSS  :
```

Example (arithmetic operations):

```
      : L      =  WERT
      : L      =  MESS
      :+F
      : JZ     =  ZERO        (Jump if zero)
      : JP     =  POSI        (Jump if positive)
      :
POSI  :
      :
ZERO  :
```

Example (digital operation):

```
      : L         FW25
      : L         IW10
      : XOW
      : JZ     =  ZERO        (Jump if accu content = KH 0000)

      : JN     =  NONZ        (Jump if accu content   KH 0000)

      :
ZERO  :
      :
NONZ  :
```

Example (shift operations):

```
      : L         QW101
      : SLW       10
      : JZ     =  NULL        (Jump if ACCU 1= KH 0000)
      : JJN    =  EINS        (Jump if ACCU 1  KH 0000)
      :
NULL  :
      :
EINS  :
      :
```

Example (conversion operations):

```
      : L         PW169
      : KZW
      : JO     =  OVFL        (Jump if overflow)
      : JN     =  NONZ        (Jump if accu content = KH 0000)
      :
NONZ  :
      :
OVFL  :
      :
```

9.3.12 Processing operations

Operation	Description
DO=	**Process formal operand** (type of parameter: DO) **Insert formal operand** Only the following operations can be substituted: C DB JU PB JU SB JU FB (See Section "Type of block parameter and allowed actual operand”)
DO DW 0 to 255 (Operation)	**Process data word** The specified operation which follows will be combined with the parameter given in the data word and executed.
DO FW 0 to 25 (Operation)	**Process flag word** The specified operation which follows will be combined with the parameter given in the flag word and executed.

With the operations DO DW and DO FW, two-word commands can also be substituted. The following command sequences, for example, are therefore possible:

```
C    DB 20       C DB 100        L    KH 0064      L    KH 0064
.                .               T    FW 90        T    FW 90
                                 .                 .
.                .               .                 .
DO FW 240        DO DW 21        DO FW 90          DO FW 90
TB  D 0.0        S    C 0.0      C    DB 0         CX  DX 0
```

The address of the bit actually addressed must be stored, as usual, in the corresponding pointer word (FW 240 or DW 21 in the example). The byte/word address must be stored on the right and the bit address on the left. The bits which are beyond the bit address in the high byte of the pointer will be deleted.

When substituting two-word commands (P, SU) in the process image, the following should be noted:

- The distinction between inputs and outputs is not made in the OP code but in the address part of the command i.e. the specification must be made in the pointer word.

Example:

```
L    KH 0104
T    FW 250
.
.
DO FW 250              (or SU Q 0.0)
SU  I 0.0              I 4.1 is set
```

- If Q 4.1 is to be set, the value KH0184 (pointer + KH0000) must be stored in the pointer word.
- In the specification of command F, the two expressions SU I 0.0 or SU Q 0.0 are fully equivalent.
- Any self-programmed bit address in the command (e.g. SU I 4.7) will be ignored.

Example: Process data word

The contents of data words DW 20 to DW 100 are to be deleted. The "index register" for the parameter of the data words is DW 0.

```
      : L       KF 20     Assignment for "index register"
      : T       DW1
M1    : L       KF 0      Reset
      : DO      DW1
      : T       DW0
      : L       DW1       Increment the index register
      : L       KF 1
      :+F
      : T       DW1
      : L       KF        100
      : <=F
      : JC    = M1        Jump if the index is within the range
      .....               Next STEP 5 program
```

The following operations can be combined with DO DW or DO FW:

A, AN, O, ON	Binary operations
S, R, P =	Storage functions
FRT, R T, SF T, SD T, SI T, SS T, SE T	Time functions
FRC, R C, S C, CD C, CU C	Counting functions
L, LD, T	Loading and transfer function
JU, JC, DO, DOC, JZ, JN, JP, JM, JO	Jump functions
SLW, SRW	Shift functions
D, I	Decrementing, incrementing
C DB, CX DX, JU, JC	Block calls

The PG 685 programmer does not verify the validity of the combination. No two or three-word commands and no operations with formal operands may be combined in function blocks.

9.3.13 Operations for page memory processing 1)

Operation	Parameter	Function
ACR		Opening a page (page number is located in ACCU1-L, permissible values 0 to 255)
LB □ □ **TB** □ □ ↑ ↑		Load from the byte-oriented page Transfer into the byte-oriented page
C B	0 to 0FFEH	One byte into/from ACCU1-L (low byte)
C W	0 to 0FFCH	One word into/from ACCU1-L
C D	0 to 0FF8H	One double-word into/from ACCU1-L
LW □ □ **TW** □ □ ↑ ↑		Load from the word-oriented page Transfer into the word-oriented page
C W	0 to 0FFCH	One word into/from ACCU1-L
C D	0 to 0FF8H	One double-word into/from ACCU1-L

The page memory area is used for acquiring and switching I/O signals when the I/O module INT EU/16B is used (linking SIMATIC EUs).

Before the page area can be accessed, one of the 256 pages has to be opened with the command ACR. The number of the page to be opened is transferred in the ACCU1-L.

With the load and transfer operations, an offset in the page memory area is transferred as a parameter.

Note:

- The page number is saved to the ISTACK on a level change (execution of an OB is interrupted by interrupt processing or by a call to another OB), i.e. a separate page number is used on every processing level.
- The page register is cleared before a new processing level is called.
- Accesses by one PLC to a page of another PLC are not possible (SINUMERIK 880).

1) *With SINUMERIK 880 GA2, SW 1 and higher or SINUMERIK 840C and higher*

9.3.14 Other operations

Operation	Description
ADD BF -128 to +127 **ADD KF -32768 to +32767**	Add byte constant (fixed-point) to ACCU 1 Add fixed-point constant (word) to ACCU 1
+ D 2) **– D** 2)	Addition of two fixed-point double-words Subtraction of two fixed-point double-words
STS **TAK**	Stop command Swap contents of ACCU 1 and ACCU 2
SES 0 to 31 **SEF 0 to 31**	Set user semaphore Enable user semaphore
LIR 0 **TIR 2** **TNB 0 to 255** **TNW 0 to 255** **LBS 0 to 255** 1) **TBS 0 to 225** 1) **BBS 0 to 255** 1)	Load register (indirect): with the contents of the memory word addressed via ACCU 1 Transfer register content (indirect): to the memory word addressed via the contents of ACCU 1 Block transfer byte by byte, source address in ACCU 2, destination address in ACCU 1 Block transfer block by block, source address in ACCU 2, destination address in ACCU 1 Load a word from system data area Transfer word to system data area Do command in system data area
LIM **SIM**	Read interrupt mask Set interrupt mask
AFF 2)	Enable addressing error detection

Notes:

- On SINUMERIK 880, SW6 and higher and SINUMERIK 840C and higher, the blocks lie across user program and user data memory boundaries. The double-word commands "+D" and "– D" take both the offset address and any change of segment number into account in calculating addresses that cross segment boundaries. The instructions TNB and TNW permit a transfer of the user program and user data memory across segment boundaries.

1) *The area BS 0 to BS 199 is reserved for the system program.*

2) *With SINUMERIK 880, SW6 and higher and SINUMERIK 840C and higher*

If you want to calculate back within a block, you must use the instructions "L KF+" and "– D".

Example:

```
                      ACCU1
L     MD 240       0006   6002
L     KF +2        0000   0002
–     D            0006   6000
```

- With instructions LIR 0 and TIR 2 the parameters will not be verified, i.e. any value can be programmed. No error message appears.
- In the case of commands with direct memory access (LIR, TIR, TNB, TNW) an offset and a segment address are required. In this case the segment is specified by entering the segment number in the high word of the ACCU 1 and ACCU 2.

The assignments are as follows:

Up to and including SINUMERIK 880, software version 4

Segment no.	Segment
1	System data memory
2	Hardware register
3	Link segment
4	Servo segment (central and distributed I/Os)
5	User program memory 1
6	User data memory 1
7	System program memory 1
8	System program memory 2
9	User data memory 2
10	User program memory 2
11	User program memory 3
12	User program memory 4

On SINUMERIK 880, SW 6 and higher and SINUMERIK 840C, SW 1, 2 und 3

Segment no.	Segment
1	System data memory
2	Hardware register
3	Link segment
4	Servo segment (central and distributed I/Os)
5	User data memory 1
6	User data memory 2
7	System program memory 1
8	System program memory 2
9	User program memory 1
10	User program memory 2
11	User program memory 3
12	User program memory 4

If other numbers are assigned the PLC goes into the stop state and an error message is output. For reasons of safety, the TIR command must not be used in the Servo segment.

The following applies when defaulting the offset address:

LIR, TIR, TBW:	Defaulting word address
TNB:	Defaulting byte address

Example:

```
L KB 6
T FW 250
L KH 0100      Offset address for DWx
T FW 252
L MD 250
LIR 0          Load DWx
```

See Section "Block lists" for the direct processing of the block list with the instructions LIR and TIR.

- If the addressing error code 1) is enabled (AFF), non-existent input and outputs are detected on accesses to the process image and the PLC branches into the STOP state.

  The addressing error code status can be scanned from the PLC user program via interface signal F5.7.

  The instruction AFS deactivates error detection. This is the default setting. The AFF/AFS setting remains after a warm restart.

1) *SINUMERIK 880, SW 6 and higher or SINUMERIK 840C and higher*

10 Rules of Compatibility between the LAD, CSF and STL Methods of Representation

10.1 General

What you can do with each method of representation in the STEP 5 programming language has its limits. It therefore follows that a program block written in an STL cannot be output in an LAD or CSF without restrictions; similarly, the LAD and CSF which are both graphic methods of representation, may not be fully compatible. If the program was entered as an LAD or CSF, it can be translated into an STL.

The purpose of this section is to provide some rules which, when observed, ensure full compatibility between the three methods of representation.

These rules are arranged as follows:

- Rules of compatibility for graphic program input (LAD, CSF)

  With input in a graphic method of representation, the observance of these rules allows output in the other methods of representation.

- Rules of compatibility for program input in a statement list

  With input in the form of a statement list, the observance of these rules ensures output in the other methods of representation.

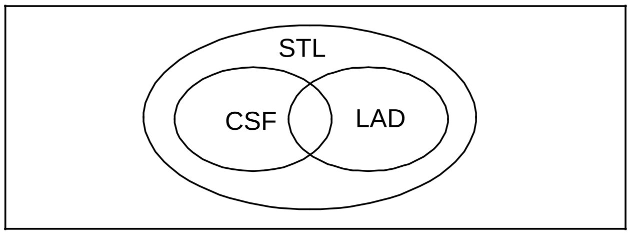


Fig. 10.1 Extent and restrictions of the methods of representation in the STEP 5 programming language

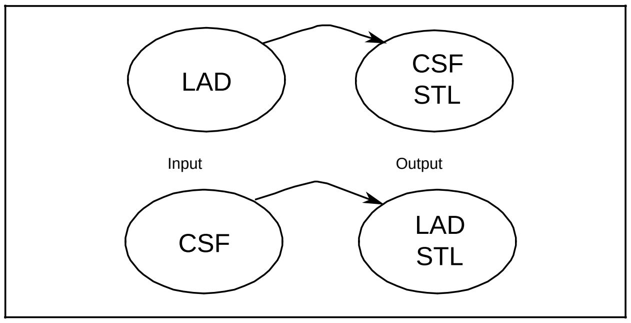


Fig. 10.2 Graphic input

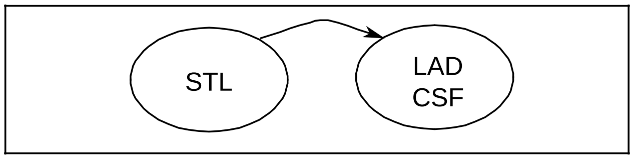


Fig. 10.3 Input in statement list

10.2 Rules of compatibility for graphic program input (LAD, CSF)

Excessively deep nesting can result in the display limits (8 levels) being exceeded in the CSF.

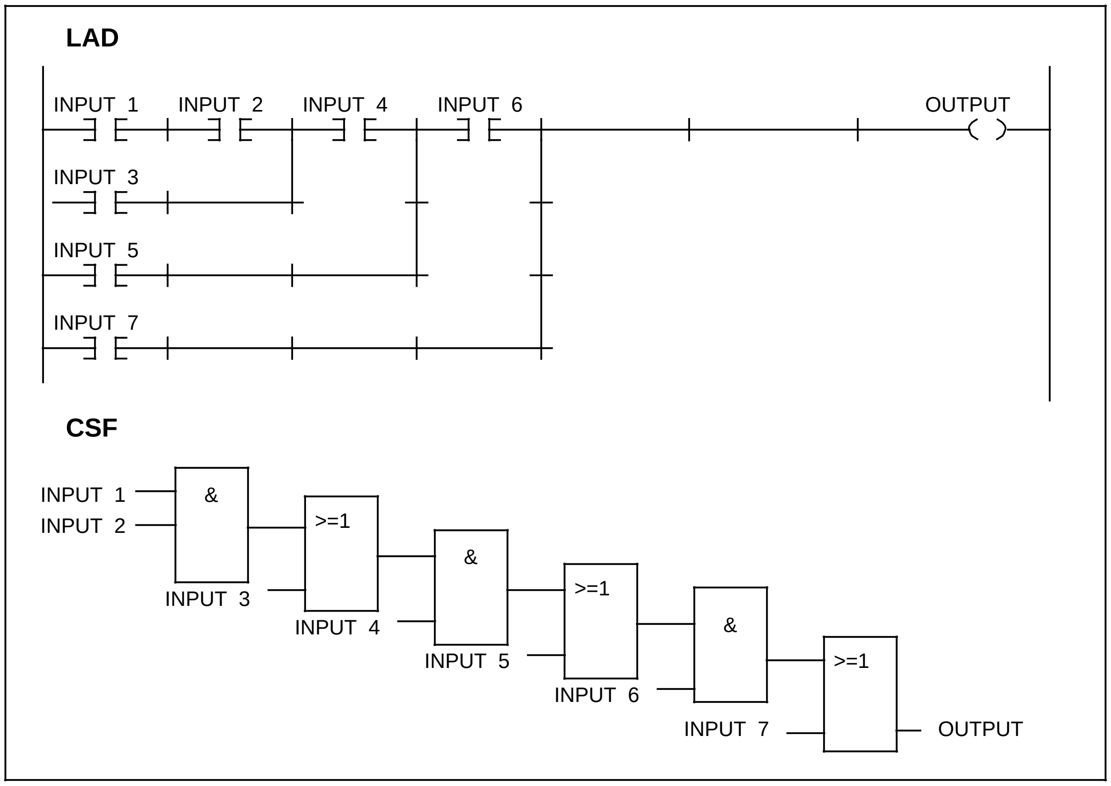


Fig. 10.4 Example of maximum LAD nesting for output in CSF

Input in CSF: Output in LAD and STL

Rule 1: Do not exceed the display limits for LAD:
An excessive number of inputs at a CSF box results in exceeding the LAD display limit.

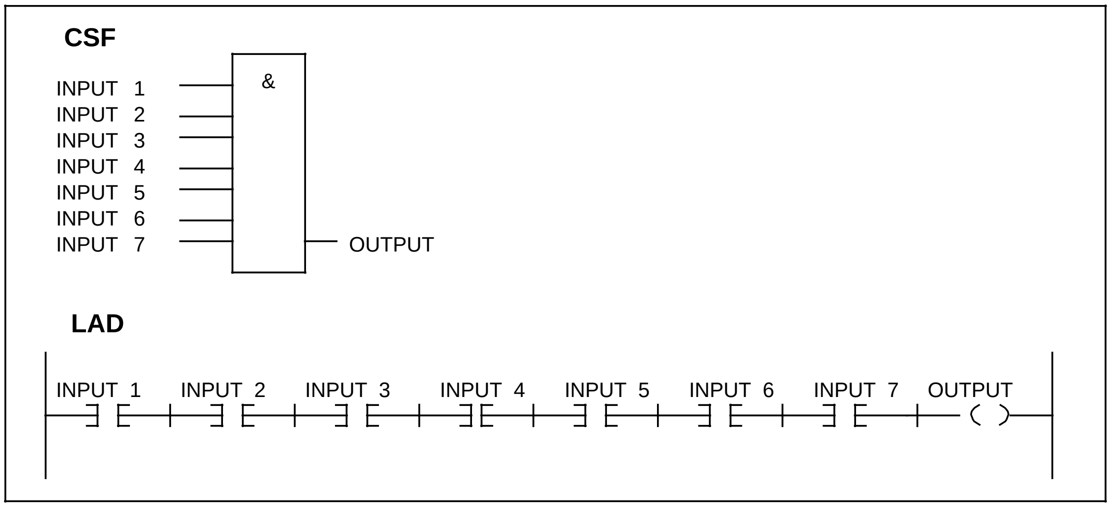


Fig. 10.5 Example of maximum AND-box expansion for output in LAD

Rule 2: The output of a complex element (storage element, comparator, timer or counter) must not be ORed.

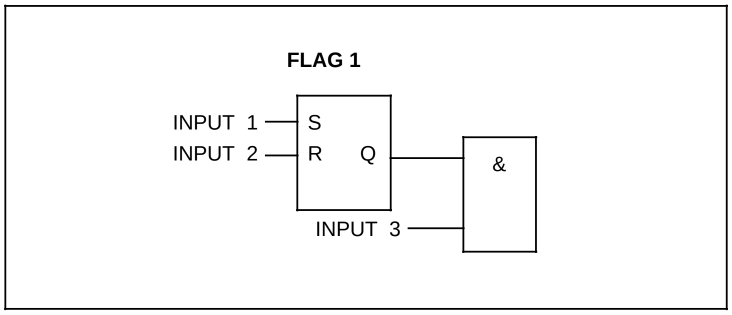


Fig. 10.6 Only AND-boxes are allowed following a complex element

Rule 3: Connectors

- Connectors are always allowed with an OR-box.
- Connectors are only allowed at the first input with an AND-box.

(Connectors are intermediate flags which are used for economy with recurring logic operations).

#	Connector allowed
X	Connector not allowed

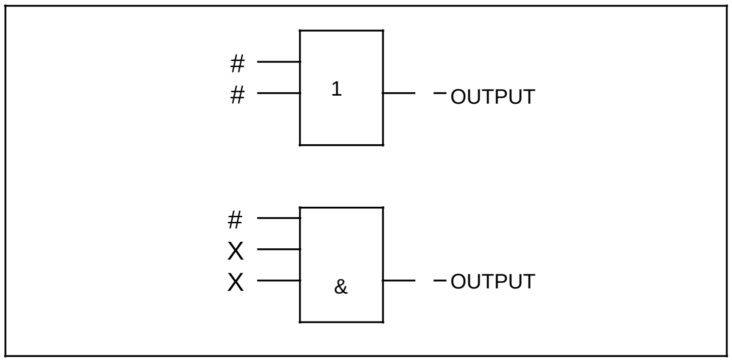


Fig. 10.7 Example showing where connectors are allowed with OR and AND-boxes

10.3 Rules of compatibility for program input in a statement list

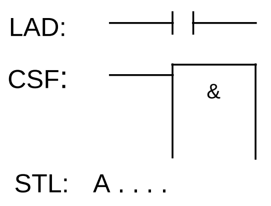
Rule 1: AND operation:

(Test of logic state and AND logic).

LAD: Contact in series

CSF: Input to an AND-box

STL: Statement A . . .



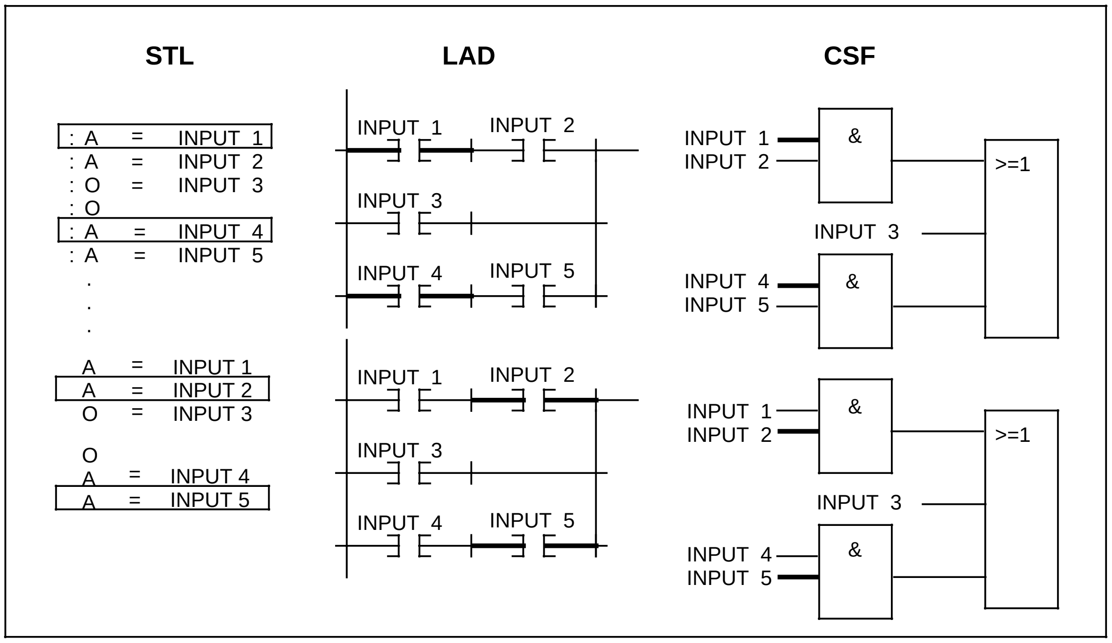


Fig. 10.8 Explanations of the rule for AND operations

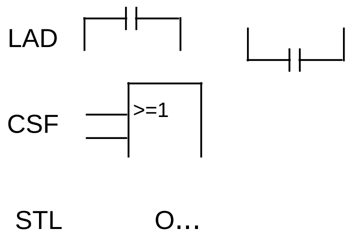
Rule 2: OR operation

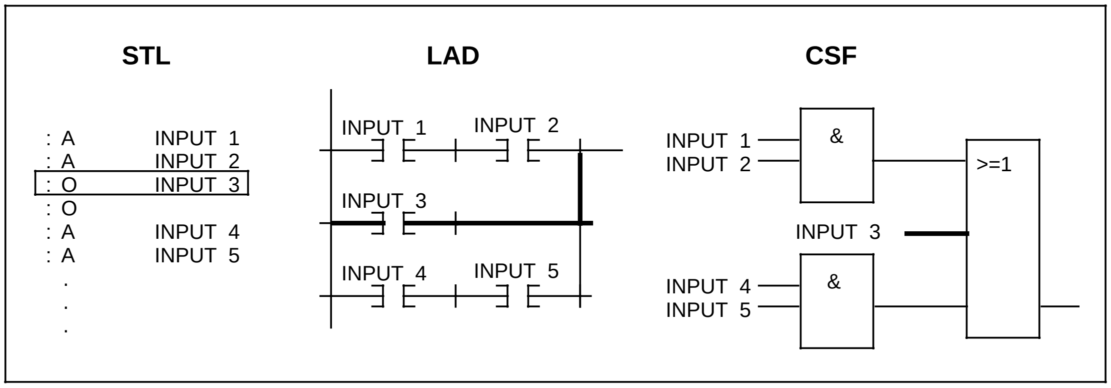
(Scan of logic level and OR logic).

LAD: Only one contact in a parallel branch

CSF: Input to an OR-box

STL: Statement O . . .





```
: A       INPUT 1
: A       INPUT 2
: O       INPUT 3
: O
: A       INPUT 4
: A       INPUT 5
  .
  .
  .
```

Fig. 10.9 Explanations of the rule for OR operations

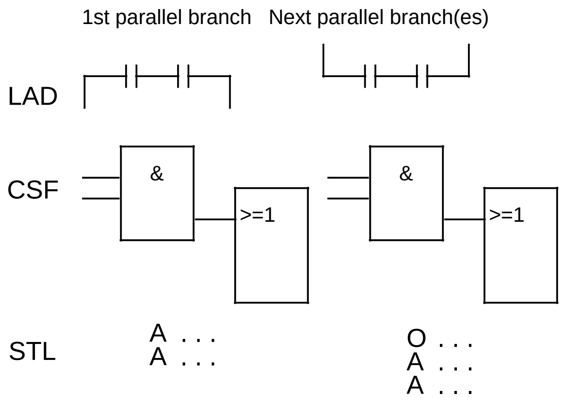
Rule 3: AND before OR operation
(OR operation before AND functions)

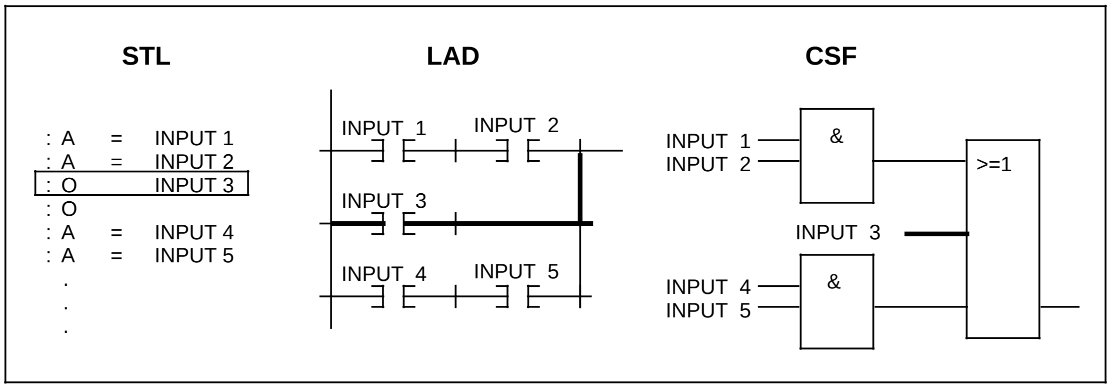
LAD: Two or more contacts in a parallel branch

CSF: AND-box before OR-box

STL: Statements

```
O . . .
A . . .
A . . .
```



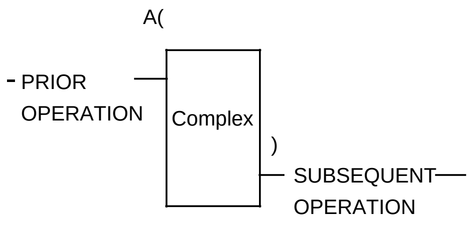


```
: A   =   INPUT 1
: A   =   INPUT 2
: O       INPUT 3
: O
: A   =   INPUT 4
: A   =   INPUT 5
  .
  .
  .
```

Fig. 10.10 Explanations of the rule for AND before OR operation

Rule 4: Parentheses

Covered in this rule are the parenthesized, complex, enclosed binary operations or complex elements with prior or subsequent operations.



a) Complex binary operation

This class of operation includes the OR before AND operations, the rules for which are as follows:

(AND operation before OR functions)

LAD	:	Sequencing of parallel contacts in series
CSF	:	OR-box before AND-box
STL	:	Statements A(OR OPERATION)

The OR before AND operations are a subset of the complex binary operations in which two parallel contacts form the simplest operation.

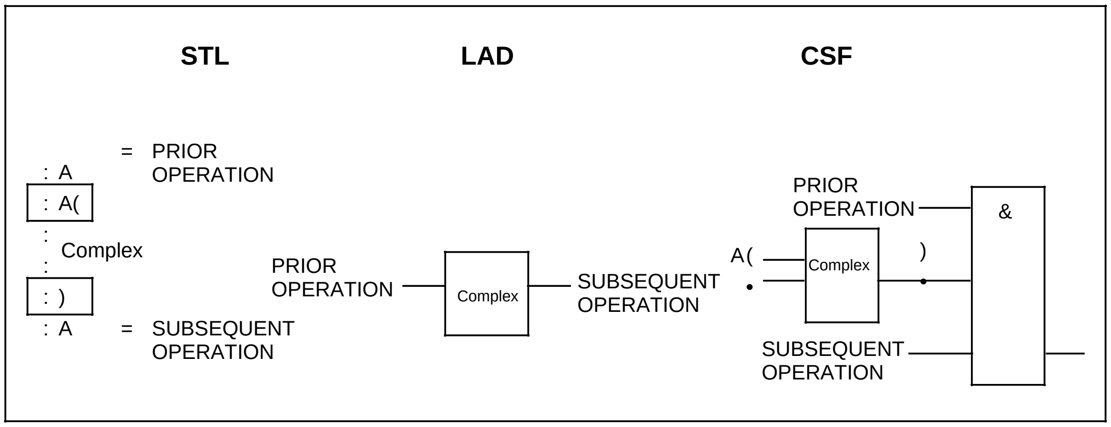


Fig. 10.11 Explanations of parenthesized, complex binary functions

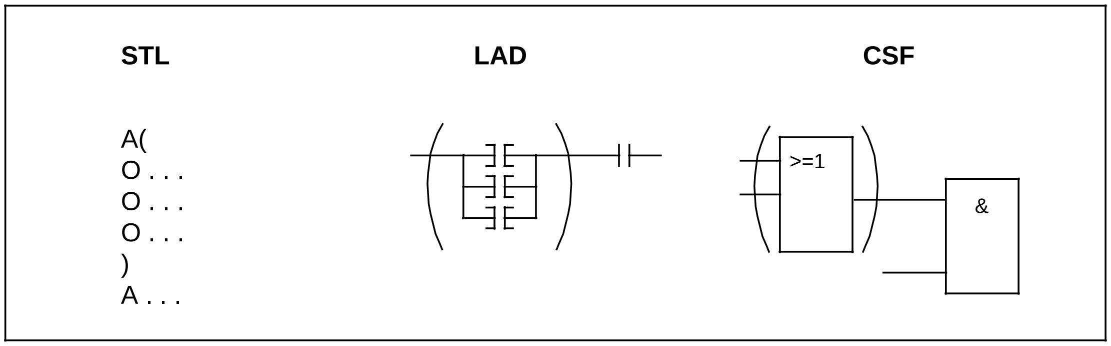


Fig. 10.12 Explanations of the rule for OR before AND operation

b) Complex elements (storage, timing, comparison and counting functions)

The following rules must be observed for complex elements:

- No subsequent operation, no parentheses
- Subsequent operation AND: A (. . .) . . .
- Subsequent operation OR: O (. . .) . . . (only for CSF, not allowed for LAD)
- A complex element cannot have prior operations.

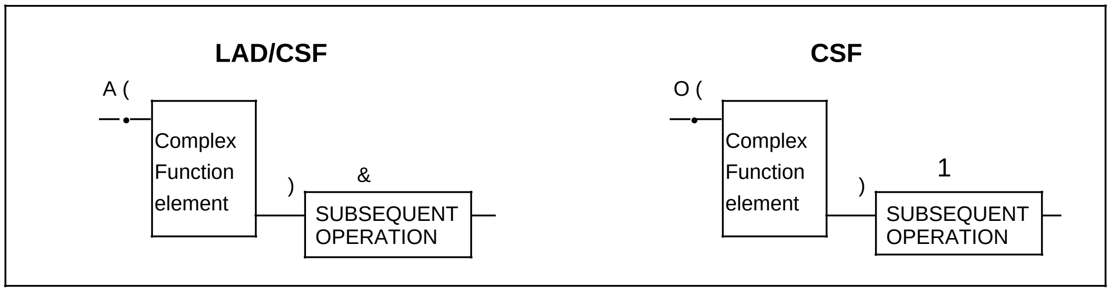
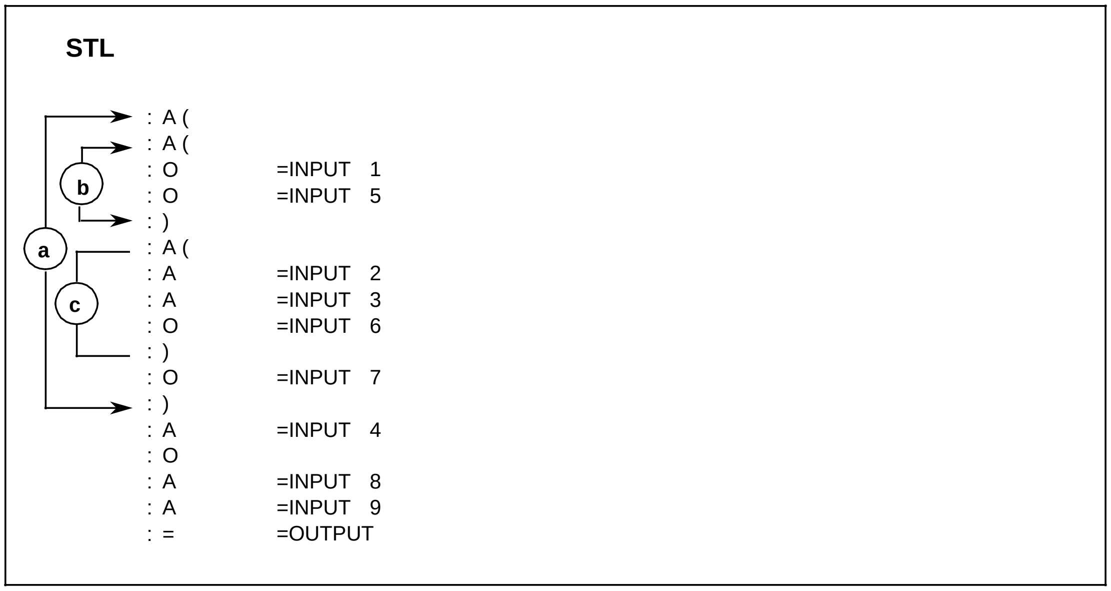


Fig. 10.13 Explanations of parentheses for complex elements

Example 1: LAD/STL

Case 1: AND (contact in series)
Case 2: OR (only 1 contact in a parallel branch)
Case 3: AND before OR (two or more contacts in a parallel branch)
Case 4: OR before AND (parentheses)



```
: A (
: A (
: O         =INPUT   1
: O         =INPUT   5
: )
: A (
: A         =INPUT   2
: A         =INPUT   3
: O         =INPUT   6
: )
: O         =INPUT   7
: )
: A         =INPUT   4
: O
: A         =INPUT   8
: A         =INPUT   9
: =         =OUTPUT
```

Fig. 10.14 Example 1: LAD/STL (continued on next page)

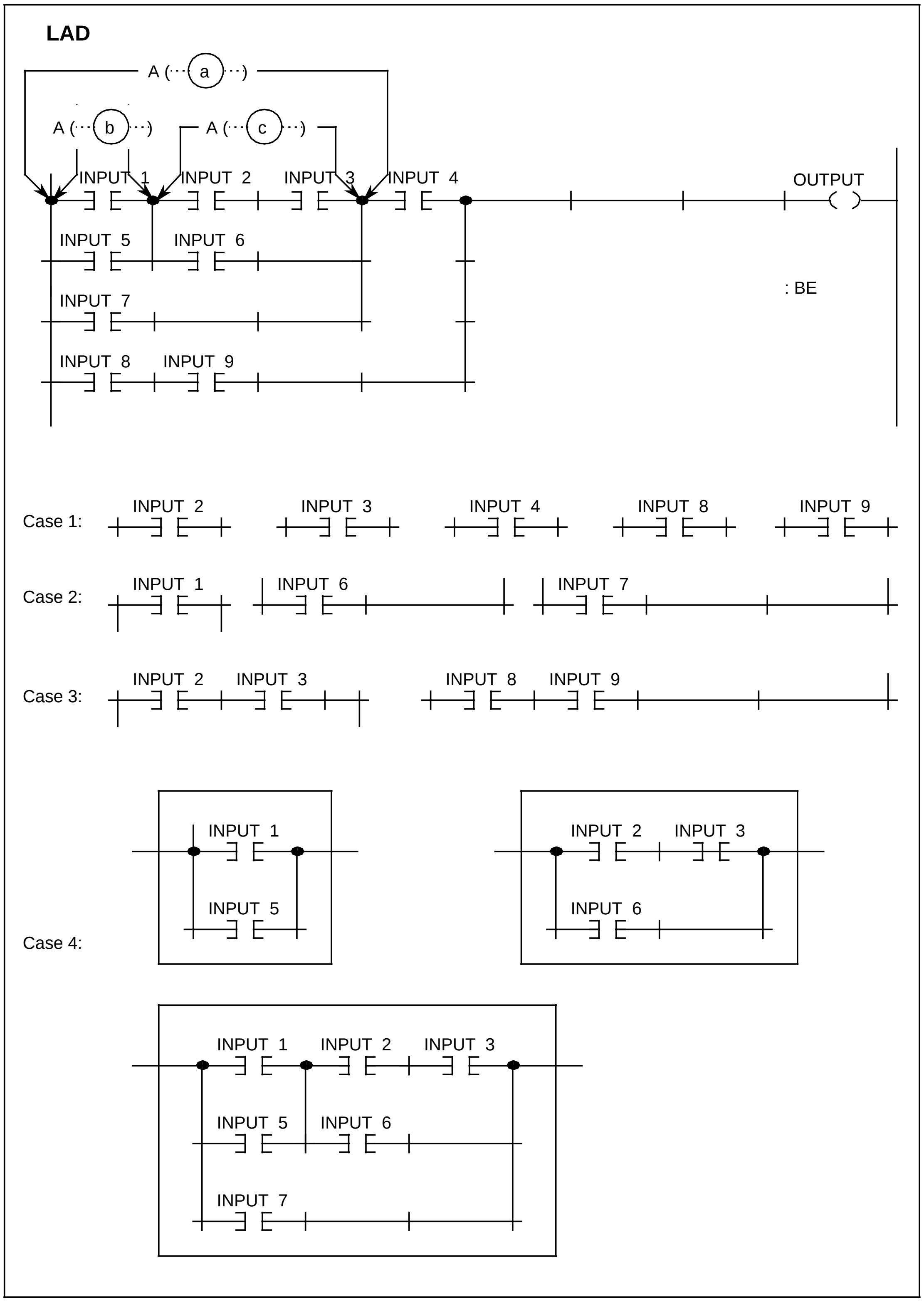


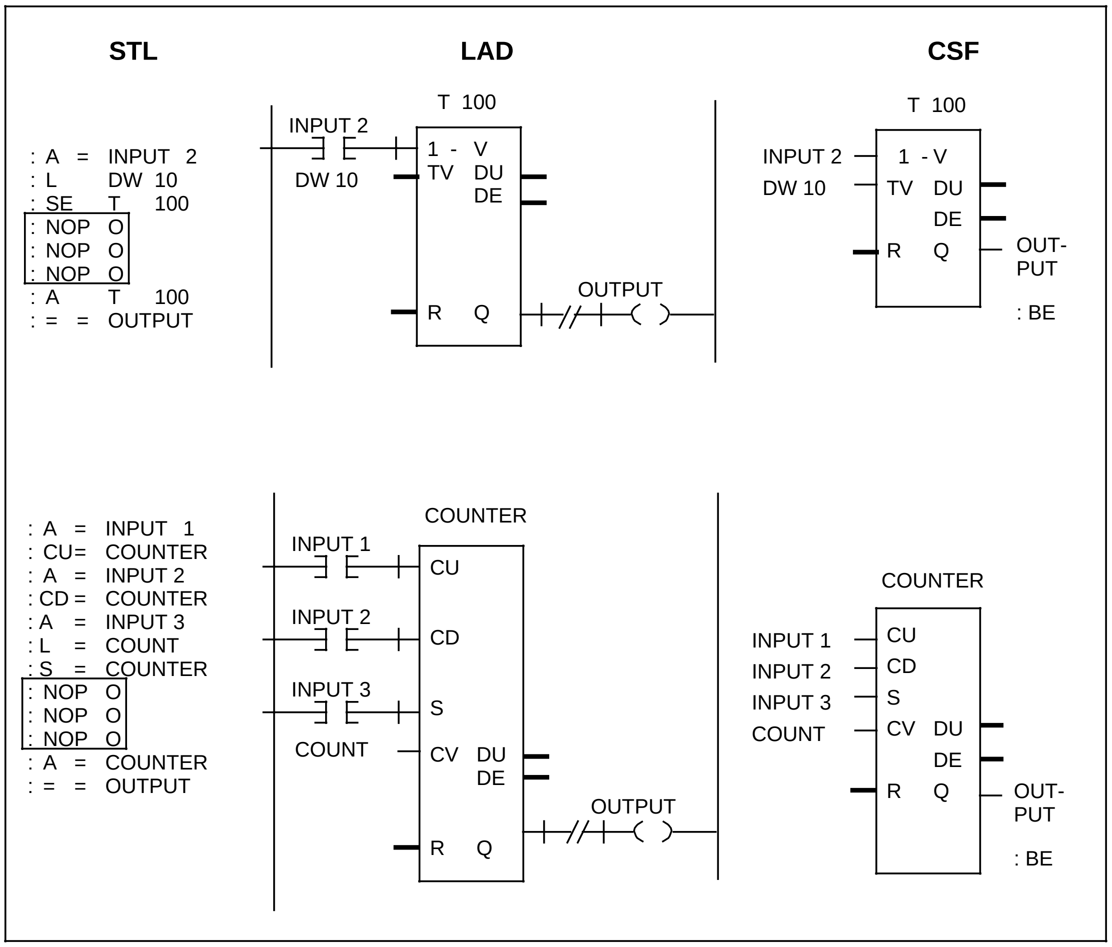
Fig. 10.14 Example 1: LAD/STL (continued)

NOP O must be applied to each unused input or output.

Exception: S and TV for timers, and S and CV for counters must always be used jointly.

For STL programming, the complex elements must be programmed in the same order as the parameter assignment on the screen in the graphic method of representation.

Exception: Time and count; the corresponding value must first be stored in the accumulator by a load command.



```
: A   =   INPUT  2
: L       DW  10
: SE      T     100
: NOP   O
: NOP   O
: NOP   O
: A       T     100
: =   =   OUTPUT
```

```
: A   =   INPUT  1
: CU=     COUNTER
: A   =   INPUT 2
: CD =    COUNTER
: A   =   INPUT 3
: L   =   COUNT
: S   =   COUNTER
: NOP   O
: NOP   O
: NOP   O
: A   =   COUNTER
: =   =   OUTPUT
```

Fig. 10.15 Example of assignments for unused inputs and outputs

Note:

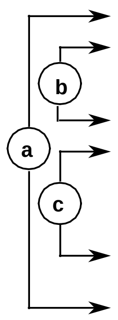
Only one complex function element is allowed per network.

The following examples show the four cases presented in a complex binary operation: In the LAD and STL methods of representation and in the CSF and STL methods of representation.

Example 2: CSF/STL

Case 1: AND (input to an AND-box)
Case 2: OR (input to an OR-box)
Case 3: AND before OR (AND-box before OR-box)
Case 4: OR before AND (OR-box before AND-box)

STL



```
: A (
: A (
: O      = INPUT 1
: O      = INPUT  5
: )
: A (
: A      = INPUT  2
: A      = INPUT  3
: O      = INPUT  6
: )
: O      = INPUT  7
: )
: A      = INPUT  4
: O
: A      = INPUT  8
: A      = INPUT  9
: =      = OUTPUT
```

CSF

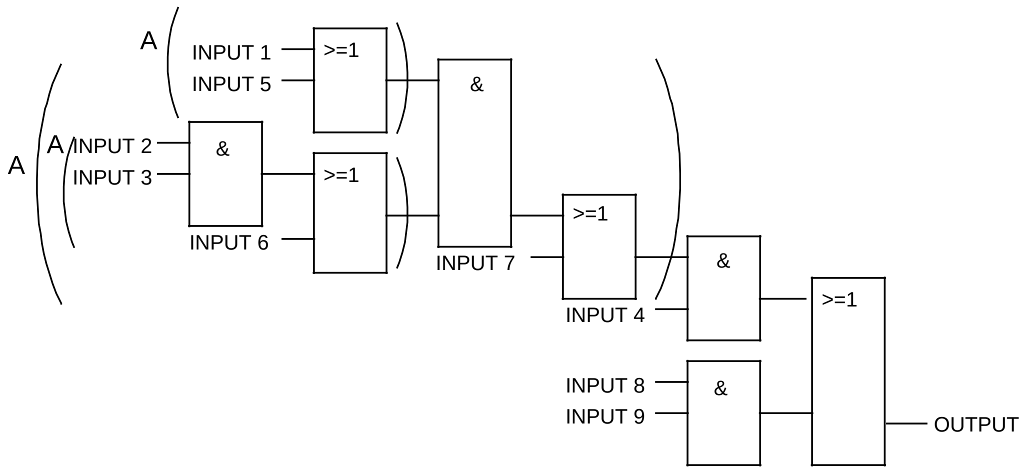


Fig. 10.16 Example 2: CSF/STL (continued on next page)

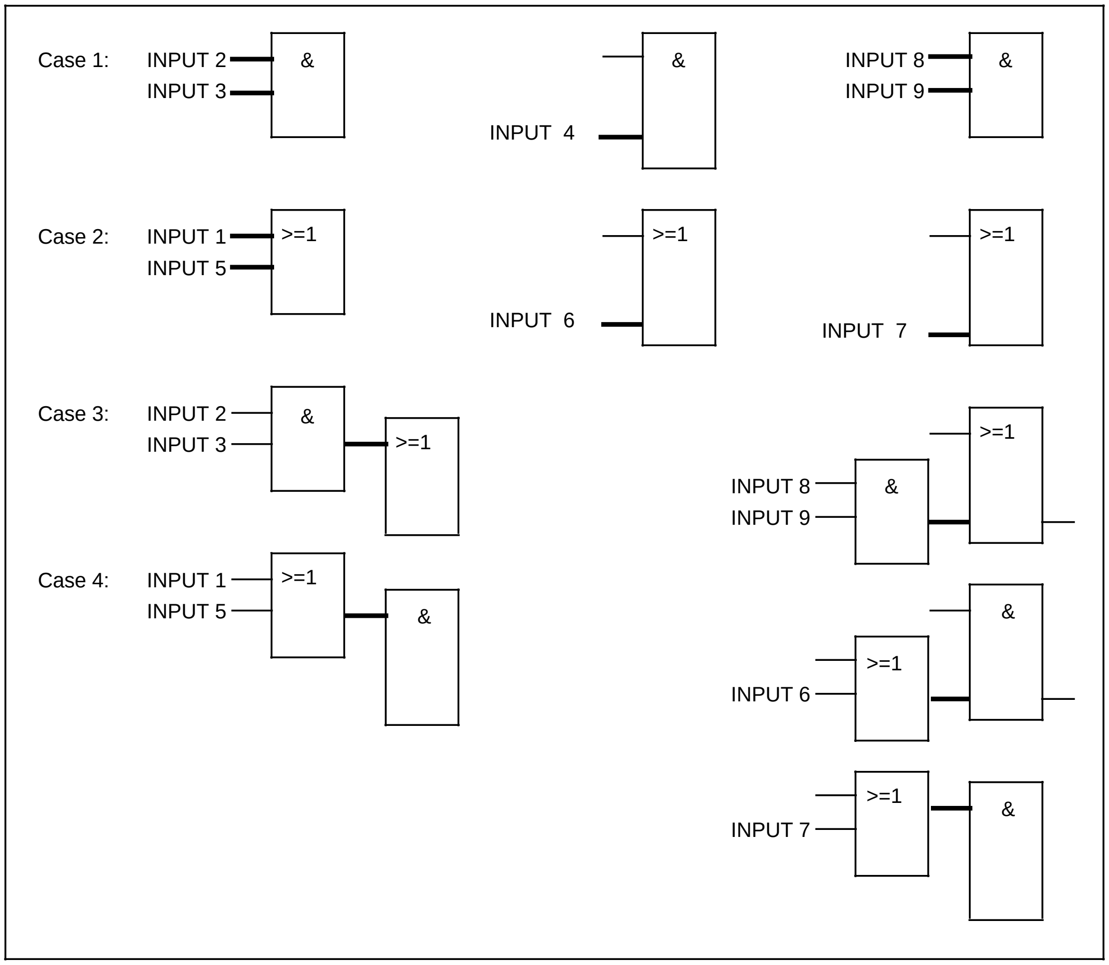
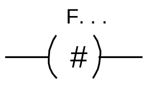


Fig. 10.16 Example 2: CSF/STL (continued)

Rule 5: Connectors

For the sake of clarity, the rules for connectors are listed separately for the LAD and CSF methods of representation. The following example is given for both.

LAD



STL

```
: A  =  F. . .
: =  =  F. . .
```

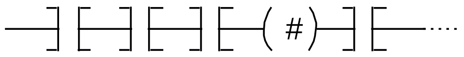
Fig. 10.17 The connector in LAD and STL

a) Connectors with LAD

The result of the logic operations that were programmed on the power rail before the connector is buffered in the connector. The following rules apply:

- Connectors in series (in series with other connectors):
  In this case a connector is treated as a normal contact.
- Connector in a parallel branch:
  Within a parallel branch, a connector is treated as a normal contact. Additionally, the entire parallel branch must be enclosed in parentheses of Type O (. . .).
- A connector may never be located immediately following the circuit (connector as first contact) or immediately after the opening of a circuit (connector as first contact in a parallel branch).

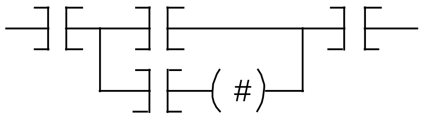
LAD



STL

```
: A
: A (
: A
: =  =  F
: A  =  F
: A
.
.
.
```

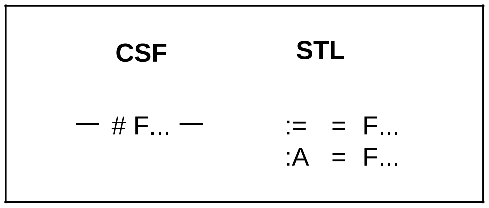
LAD



STL

```
: A . . .
: A (
: A . . .
: A (
: A . . .
: =  =  F
: A  =  F
: )
: )
: A . . .
.
.
.
```

Fig. 10.18 Connector controller for LAD



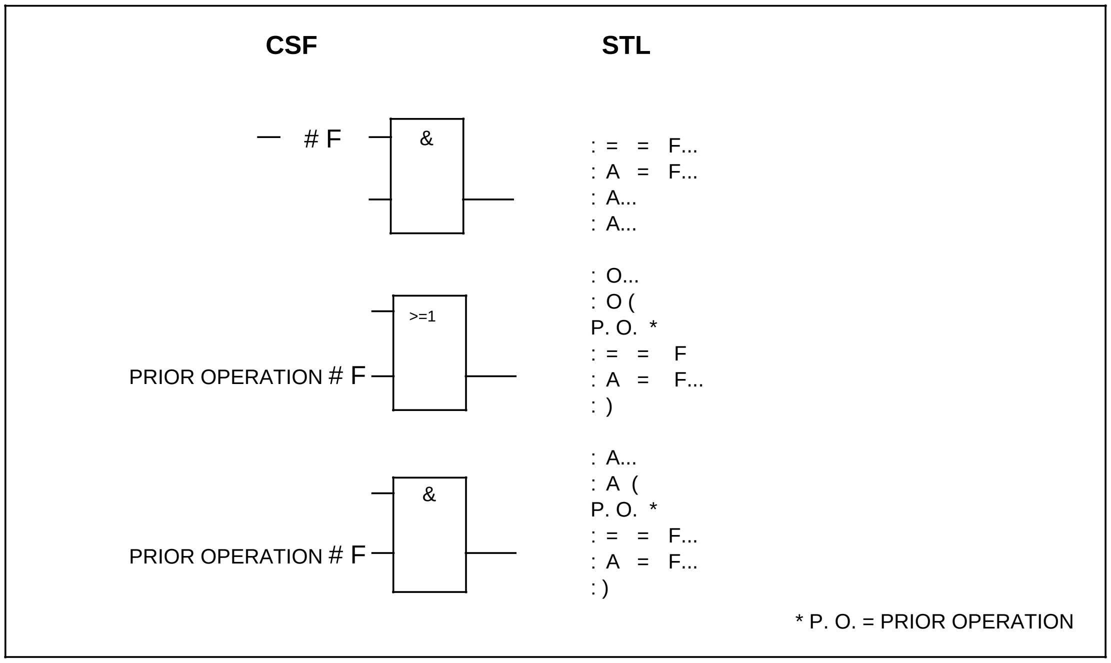
```
:=   =  F...
:A   =  F...
```

Fig. 10.19 The connector in CSF and STL

b) Connectors with CSF

The result of the entire binary logic operation before the connector is buffered in the connector. The following rules apply:

- Connector at the first input of an AND or OR-box:
  The connector is transformed without parentheses.
- Connector not at the first input of an OR-box:
  The entire binary operation preceding the input is enclosed in parentheses of Type O (. . .).
- Connector not at the first input of an AND-box:
  The entire binary operation preceding the input is enclosed in parentheses of Type A (. . .). (Only allowed with CSF; not graphically representable with LAD.)

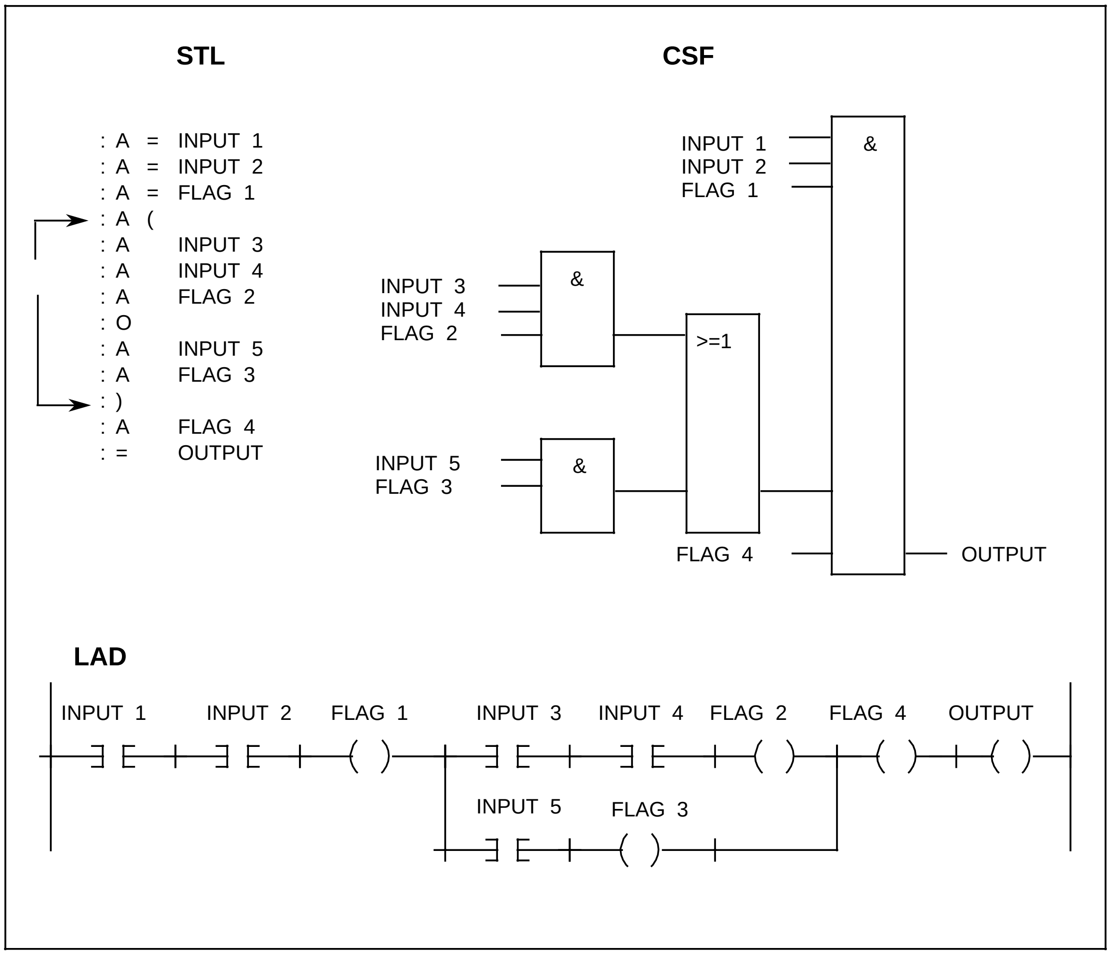


```
: =   =   F...
: A   =   F...
: A...
: A...

: O...
: O (
P. O.  *
: =   =   F
: A   =   F...
: )

: A...
: A  (
P. O.  *
: =   =   F...
: A   =   F...
: )
```

Fig. 10.20 Connector for CSF

Examples for connectors:

Two examples are given: One without and one with connectors.



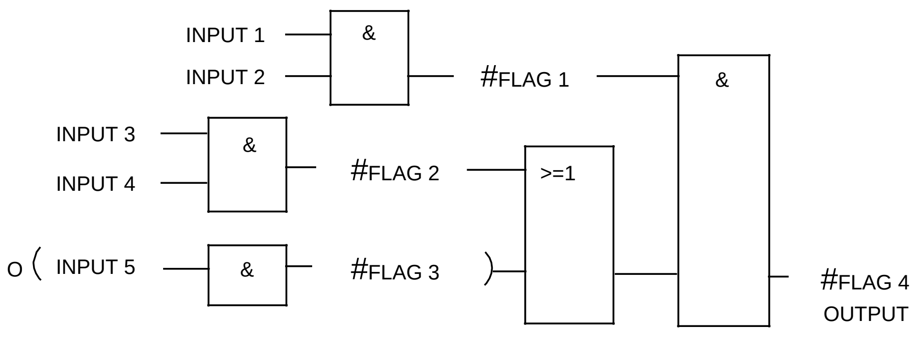
```
: A  =  INPUT  1
: A  =  INPUT  2
: A  =  FLAG  1
: A  (
: A     INPUT  3
: A     INPUT  4
: A     FLAG  2
: O
: A     INPUT  5
: A     FLAG  3
: )
: A     FLAG  4
: =     OUTPUT
```

Fig. 10.21 Example without connectors

STL

```
: A    =    INPUT 1
: A    =    INPUT 2
: =         FLAG 1     }  Connector 1
: A    =    FLAG 1
: A (
: A    =    INPUT 3
: A    =    INPUT 4
: =         FLAG 2     }  Connector 2
: A    =    FLAG 2
: O (
: A    =    INPUT 5
: =         FLAG 3     }  Connector 3
: A    =    FLAG 3
: )
: )
: =         FLAG 4     }  Connector 4
: A    =    FLAG 4
: =    =    OUTPUT
```

CSF



LAD

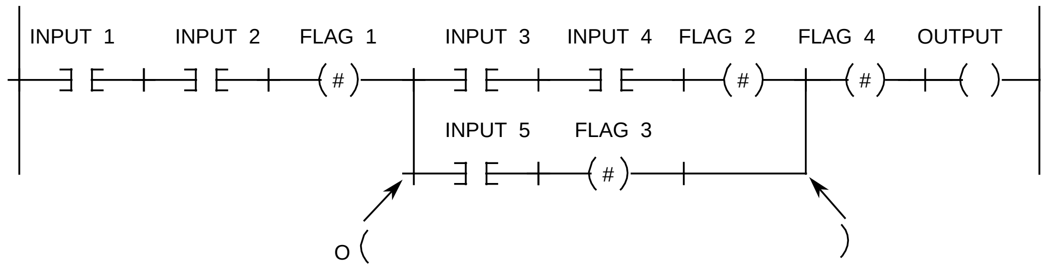


Fig. 10.22 Example with connectors

Connector 1:	Result of logic operation INPUT 1 AND INPUT 2
Connector 2:	Result of logic operation INPUT 3 AND INPUT 4
Connector 3:	Result of logic operation (INPUT 3 AND INPUT 4) OR INPUT 5
Connector 4:	Result of the entire binary operation

11 Hardware

11.1 General notes on the PLC 135 WB/WB2

The interface controller consists of one module. In addition to the CPU it contains the complete user and system memory.

The PLC CPU is a pure "RAM machine", i.e. the system and user programs are copied from the EPROM submodules into the internal CPU RAM on startup and executed there (fast execution without wait states).

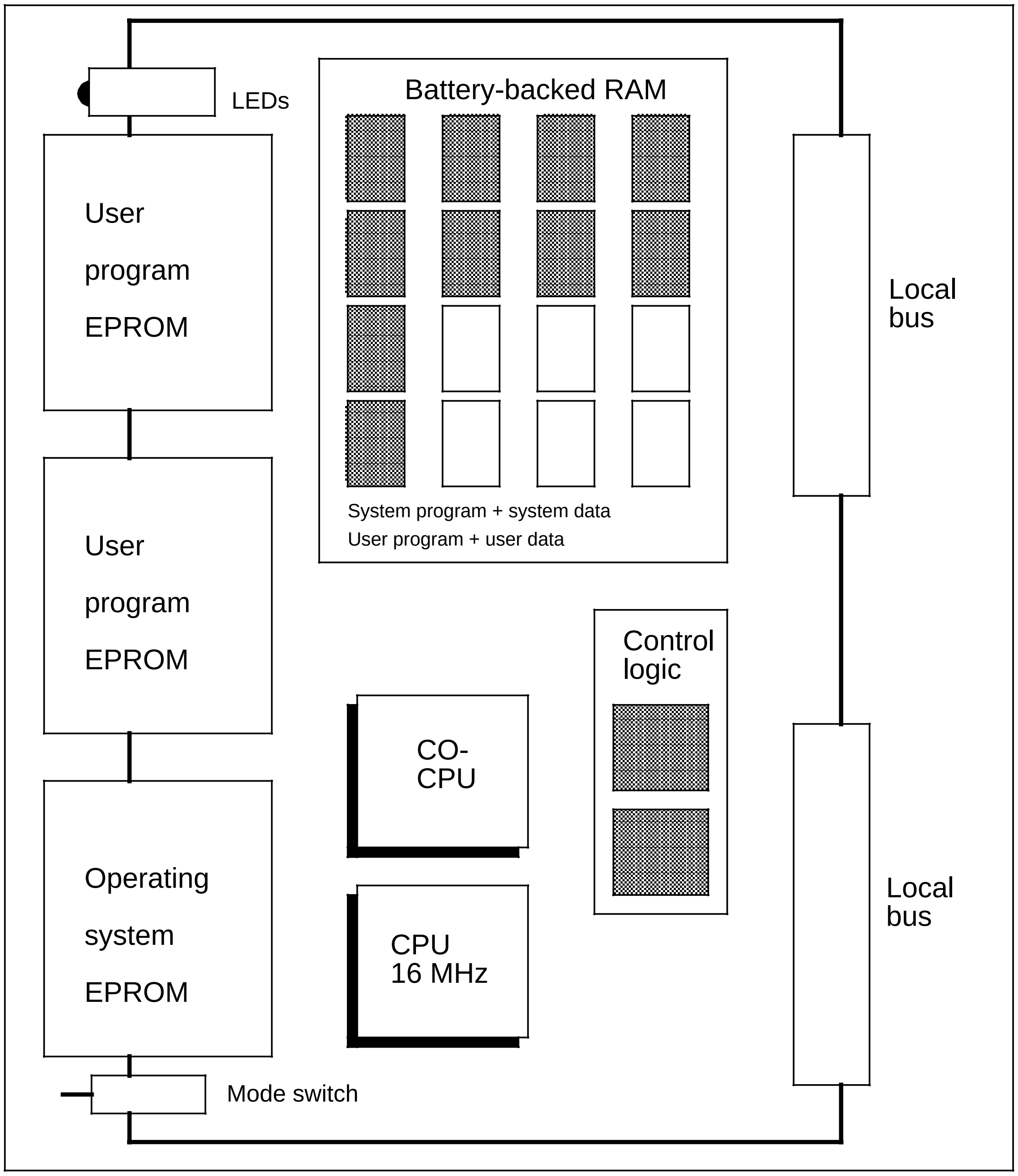


Fig. 11.1 Hardware overview of the PLC 135 WB/WB2

11.1.1 Operating the PLC 135 WB/WB2 using the mode switch

On the front panel of the interface controller is a mode switch with the positions RUN (R), STOP (S) and OVERALL RESET (U). With this switch various modes can be produced.

Operation	**PLC response**	**LED signal state**	**Comment**
RUN STOP RUN	With the change from STOP RUN the PLC performs a warm restart (SINUMERIK hardware reset).	When the green LED is lit the PLC is in the STOP state When the red LED is lit the PLC is in cyclic mode	Warm restart If the STOP state was caused by an error a cold restart is automatically performed.
RUN STOP O. RESET t_v <10 sec. STOP t_v<2 sec. RUN	With the change from STOP RUN a warm restart is performed.	The LEDs blink synchronously at a frequency of 1 sec.	Warm restart
RUN STOP O. RESET t_v <10 sec. STOP t_v>2 sec. RUN	With the change from STOP RUN a cold restart is performed bootstrapping the user program from EPROM without data loss in the data segment.	The LEDs blink synchronously at a frequency of 1 sec.	Bootstrapping of the user program and the data from the EPROM followed by a cold restart. The DBs and DXs in the user data memory (created with FB11 or input on the programmer with a segment switch) are not deleted.

Operation	PLC response	LED signal state	Comment
RUN STOP O. RESET t_v >10 sec. STOP t_v>2 sec. O. RESET STOP RUN	With the change from STOP O. RESET an overall reset is requested. With the second change from STOP O. RESET (confirm reset) an overall reset is performed followed by bootstrapping. With the change from STOP RUN a cold restart is performed.	The LEDs blink synchronously at a frequency of 1 sec. The reset request is indicated by fast asynchronous blinking of both LEDs. After the operation "Confirm reset" and when only the red LED is lit the overall reset has ended.	Overall reset with bootstrapping of the user program and the data from the EPROM. The DBs and DX in the user data memory must be created again. On changing from STOP RUN the reset request is deleted again and a cold restart performed.
RUN STOP RESET t_v >10 sec. STOP RUN	With the change from STOP RUN a cold restart is performed.	The LEDs blink synchronously at a frequency of 1 sec. After 10 sec. have elapsed both LEDs blink rapidly and asynchronously	Cold restart

t_v = dwell time

11.1.2 Eprom submodule for user program

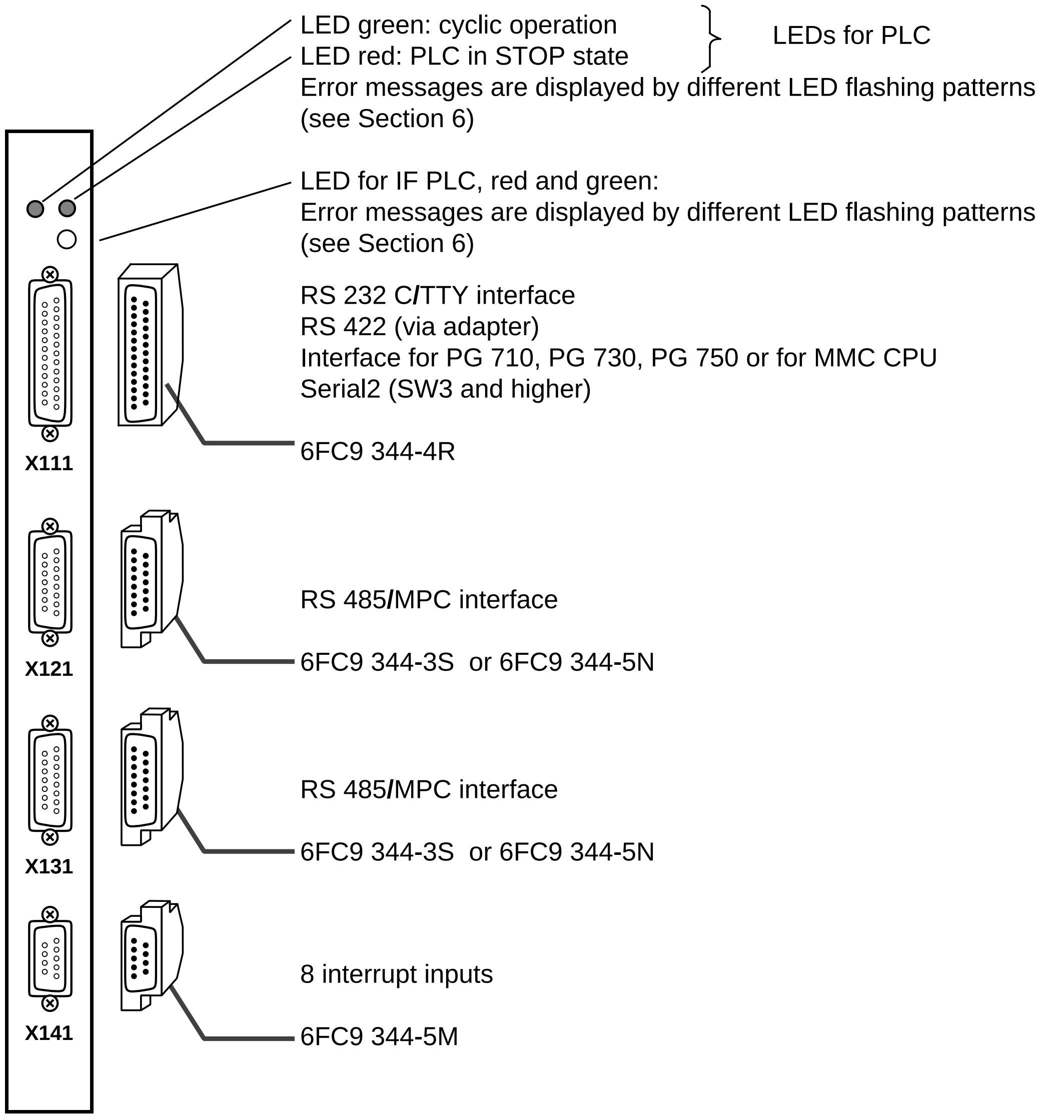
EPROM Order No.	Memory capacity	Programmable with	Program No.
6FX1 130-5BB00	64 KB	PG 685 1) PG 750	162 162
6FX1 145-8BA00 2) 6FC5 130-0CA01-0AA0 3)	256 KB	PG 685 1) PG 750	129

1) *MEP adapter required for programming.*
2) *Order No. valid for SINUMERIK 880 and SINUMERIK 880 GA2*
3) *Order No. valid for SINUMERIK 840C*

11.2 General notes on the PLC 135 WD

The PLC 135 WD module is used with SINUMERIK 840C, SW 3 and higher. The functions of the PLC CPU 135 WB2 and the interface PLC are combined on this module. The PLC system program and user program are stored on the hard disk and are loaded onto the buffered RAM of the PLC from where they are run.

The following interfaces and LEDs are situated on the front panel:


LED green: cyclic operation
LED red: PLC in STOP state
LEDs for PLC
Error messages are displayed by different LED flashing patterns (see Section 6)
LED for IF PLC, red and green:
Error messages are displayed by different LED flashing patterns (see Section 6)
RS 232 C/TTY interface
RS 422 (via adapter)
Interface for PG 710, PG 730, PG 750 or for MMC CPU
Serial2 (SW3 and higher)
6FC9 344-4R
X111
RS 485/MPC interface
6FC9 344-3S or 6FC9 344-5N
X121
RS 485/MPC interface
6FC9 344-3S or 6FC9 344-5N
X131
8 interrupt inputs
6FC9 344-5M
X141

Notes:

- The bootstrap EPROMs are situated on the module.
- See also INTERFACE DESCRIPTION PART 2 - CONNECTION CONDITIONS on how to use the interfaces.

11.2.1 Operating the PLC 135 WD

The PLC 135 WD no longer has a start-up switch. The functions WARM RESTART, COLD RESTART, BOOTSTRAP and GENERAL RESET can be executed from the programming unit and/or the operator panel (see INSTALLATION INSTRUCTIONS).

Notes:

- S5 functions (e.g. programming) can also be executed from the operator panel via the PLC 135 WD, X111   cable 6FC9 344-4R   MMC CPU, serial 2 connection and if the necessary S5 software is installed on the hard disk (Option, SW 3 and higher).
- The PLC system program is only booted from the hard disk to the buffered RAM if
  - no system program is available in the buffered RAM when the control is switched on (e.g. after data loss)
  - the softkey FORCED BOOT is pressed (e.g. to install new software).

12 Programming and Test Functions with the Programmer

12.1 Requirement

To be able to use the full power of the PLC 135 WB/WB2/WD, the following programmers are used with the appropriate software.

Programmer	Package description	Order number
PG 685	S5-DOS ST basic package	6ES5-885-0SA11 1)
PG 750	S5-DOS ST basic package	6ES5-886-0SA11 1)
PG 685 PG 750	S5-DOS ST supplementary package+(GWE)	6FX1-861-5BX-01-1B
PG 685	GRAPH 5 S5-DOS	6ES5-895-1SA01

12.2 Output of information

The following information can be read out from the PLC and displayed using the programmer:

- System parameters
- PLC addresses
- BSTACK, USTACK
- PLC directory and memory configuration

Notes:

- The functions are described in the Product Manual for the programmer.
- System parameters:
  A PLC machine data bit must be set to set the PLC mode for the PLC 135 WB/WB2/WD (see INSTALLATION INSTRUCTIONS).
- PLC addresses:
  The addresses are 20 bits wide (address area: 1 megaword). The system data segment, the user program and user memory are available as readable areas.
  Detailed error coding (ERRCODE) can be scanned at address F0000$_{hex}$ and the position of the segment switch can be scanned at address E0000$_{hex}$.
- Memory configuration:
  The contents of the lines indicating the memory configuration and the available free memory depend on the position of the segment switch. If the segment switch position is 0006 (AWDSEG), the data of the user data memory appears; if the segment switch position is not 0006 (AWDSEG), the values of the user program memory are displayed.

1) Last digits 11: German version
21: English version

12.3 START PLC

The following restart modes can be selected via the programmer screen form START PLC:

NEUSTART (cold restart)	The PLC performs a cold restart.
WIEDERANLAUF (warm restart)	The PLC performs a warm restart. A warm restart is, however, only possible if the PLC has been put into the stop state by the function STOP PLC.
NEUSTART MIT URLADEN (cold restart with bootstrap)	The PLC copies the contents of the user program modules into the internal RAM memory and then performs a cold restart. All blocks are copied into the user program memory. The contents of the user data memory remain unchanged.

It is only possible to start the PLC in the STOP state. If the PLC is not in the STOP state there is no response.

12.4 Block handling using the programmer

Programmer operation	Result in the PLC	Comment
OVERALL RESET	All user program blocks and data blocks are deleted (RAM memory). Then the user EPROMs are automatically bootstrapped.	PLC must be in the STOP state.
COLD RESTART with BOOTSTRAP	The blocks in the user EPROM submodule are copied into the internal user program memory (RAM memory) and keep the EPROM identifier. The user data memory is not changed by this function.	PLC must be in the STOP state.
OUTPUT BLOCK	The function allows blocks to be read out and edited. The blocks are then written back with a RAM identifier.	Note the position of the segment switch when writing back data blocks.
CONTROL	It is only possible to control data blocks if they have a RAM identifier.	
DELETE BLOCK	Only blocks with a RAM identifier can be deleted.	

Notes:

- All user blocks with an EPROM identifier can be edited and input as test blocks with a RAM identifier. If the test block is deleted and a cold restart performed, the EPROM block becomes available again.
- With SINUMERIK 840C, SW 3 and higher, the PLC user program is stored in a file on the hard disk. This file corresponds to the previous EPROM submodule, i.e. to achieve an "empty" PLC on PLC GENERAL RESET, the file must either be deleted or renamed (same as removing the EPROM submodule).