SINUMERIK 810
Basic Version 2
SINUMERIK 820

PLC Programming

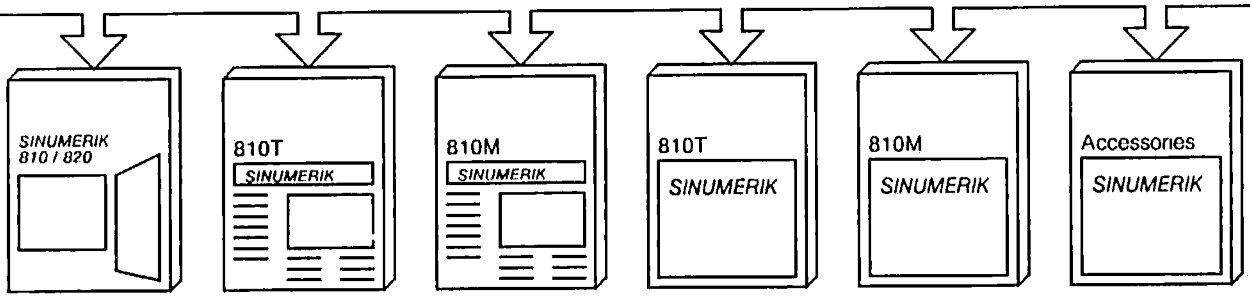
Planning
Guide

SINUMERIK

Manufacturer
Documentation
10.90 Edition

SINUMERIK 810, Basic Version 2 (GA 2)

General Documentation



Sales Brochure

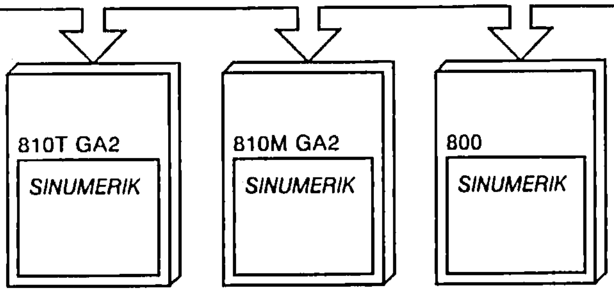
Technical Data

Catalog NC 21

Catalog NC 22

Catalog NC 90

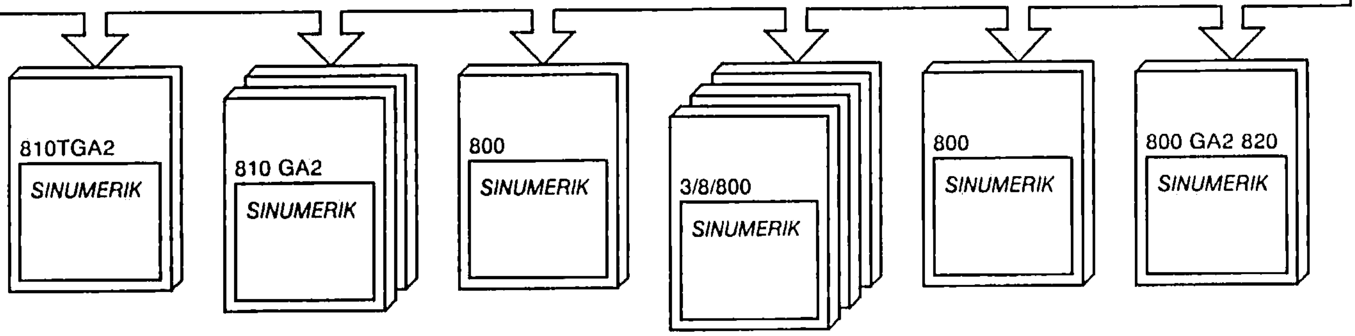
User Documentation



User's Guide
(Operating/Programming)

Contour Calculator
for NC Geometry
Programs

Manufacturer Documentation



Instructions Manual

Interface:
- Signals
- Cables and Connections

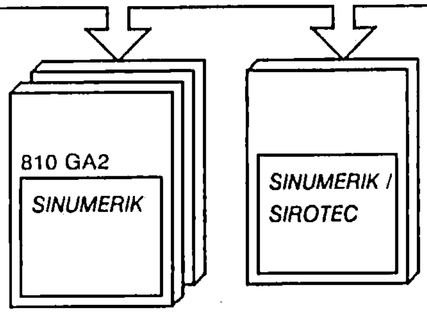
Universal Interface

Measuring Cycles:
- General
- Turning
- Milling

SINUMERIK WS 800
CL 800 Language

PLC Programming

Service Documentation



Installation Guide
- Instructions
- Lists

Spare Parts List

SINUMERIK 820, see last page

SINUMERIK 810
Basic Version 2
SINUMERIK 820

PLC Programming

Manufacturer Documentation

Planning Guide

For:

Control	*Software version*
SINUMERIK 810T/ 810TE (GA 2)	2
SINUMERIK 810M/ 810ME (GA 2)	2
SINUMERIK 810N/ 810NE	1
SINUMERIK 810G/ 810GE	1
SINUMERIK 820T/ 820TE	2
SINUMERIK 820M/ 820ME	2
SINUMERIK 820N/ 820NE	1
SINUMERIK 820G/ 820GE	1

October 1990 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

Edition	**Order No.**	**Remarks**
05.89	6ZB5 410-0BX02-0BA0	**A**
10.90	6ZB5 410-0BX02-0BA1	**C**

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

This publication was produced on the Siemens 5800 Office System.
Subject to change without prior notice.

Preliminary Remarks

Notes to the reader

This manual is intended for the manufacturers of machine tools using SINUMERIK 810 (basic version 2) and SINUMERIK 820.
The Guide describes in detail the program structure and the operation set of the PLC, and explains precisely how basic PLC software and user programs, which comprise data blocks, function blocks and organization blocks, are constructed.

The SINUMERIK documentation is organized in three levels:

- User documentation
- Manufacturer documentation and
- Service documentation

The **Manufacturer Documentation** for **SINUMERIK 810 basic version 2 (GA2) and SINUMERIK 820** is divided into the following:

- Instruction Manual
- Interface
  Part 1: Signals
  Part 2: Cables and hardware
- PLC Programming Guide

Additional SINUMERIK publications are also available for all SINUMERIK controls (e. g. publications on the Universal Interface, Measuring Cycles, CL 800 Cycles language).

Please contact your Siemens regional office for further details.

Technical Information

This documentation applies to:

Control	*Software version*
SINUMERIK 810T/ TE (GA2)	*2*
SINUMERIK 810M/ ME (GA2)	*2*
SINUMERIK 810G/ GE	*1*
SINUMERIK 810N/ NE	*1*
SINUMERIK 820T/ TE	*2*
SINUMERIK 820M/ ME	*2*
SINUMERIK 820G/ GE	*1*
SINUMERIK 820N/ NE	*1*

Contents

Page

1 Introductory Remarks

1.1 Application

The PLC is a powerful interface controller. It has no inherent hardware, and is used as "Software PLC" exclusively in the SINUMERIK 810 basic version 2 (GA 2) and SINUMERIK 820.

The PLC is responsible for control of machine-related functional sequences such as

- Controlling auxiliary axes
- Controlling tool-changers
- Gathering the signals generated by the machine's monitoring devices.

Special features:

On the SINUMERIK 810 basic version 2, all NC and PLC jobs are executed by a common processor (Intel 80 186). A specially developed coprocessor (COP) is responsible for high-speed execution of binary operations in the STEP 5 user program (e. g. A I 3.0, S Q 8.1, etc.). The single-processor structure calls for strict priority scheduling of NC and PLC jobs. NC functions such as position control and block preparation, but also the PLC "Interrupt processing" function (OB 2), have highest priority, while scanning of the cyclic program (OB 1) is of less importance. The amount of processor time available to the PLC has been restricted to a maximum of 20% in order to ensure that the NC can fulfill its primary objectives (positioning accuracy, short block change times) even in extreme situations, e. g. where an exceptionally long STEP 5 program or frequent interrupt servicing is involved. It is also possible to influence the manner in which the cyclic user program scanned via machine data (see Section 1.3.4).

1.2 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), 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 (cf. Section 10).

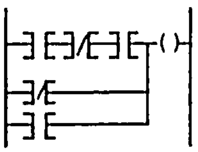
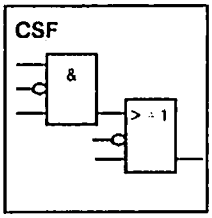
Ladder diagram	Statement list	Control system flowchart
Programming with symbols similar to those used in schematic circuit diagrams Complies with DIN 19239 (draft)	Programming with using mnemonics designating functions Complies with DIN 19239 (draft)	Programming with graphic symbols Complies with IEC 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

1.3 Programming

1.3.1 Program structure

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

The operating system and the basic software are integral components of the PLC; they are supplied on EPROMs, and may not be modified in any way.
The *user program* is the total of all statements and declarations/data programmed by the user.

The PLC structure 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)
  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)
  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 2 and OB 20 are serviced by the operating system (cf. Section 5).
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 involved is stipulated by programming the relevant calls (conditional or unconditional) in organization blocks (see Section 5.4, "Cyclic program").

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

Different organization blocks are provided for various methods of program execution (cf. Section 5.2). 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 for organization blocks is 12 (Fig. 1.3), not including an accompanying data block, if any.

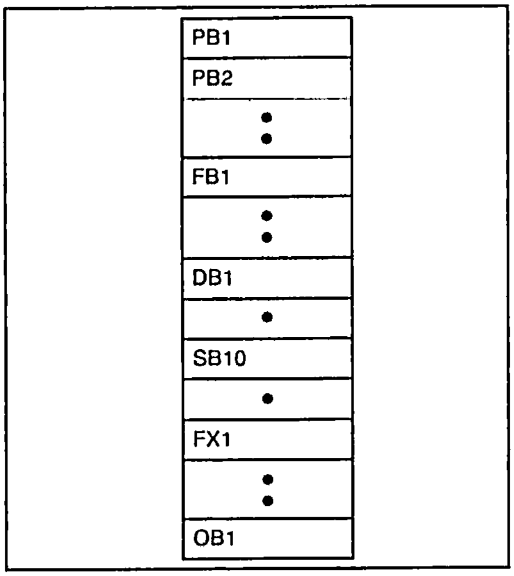


Fig. 1.2 Storing the blocks in arbitrary order in program memory

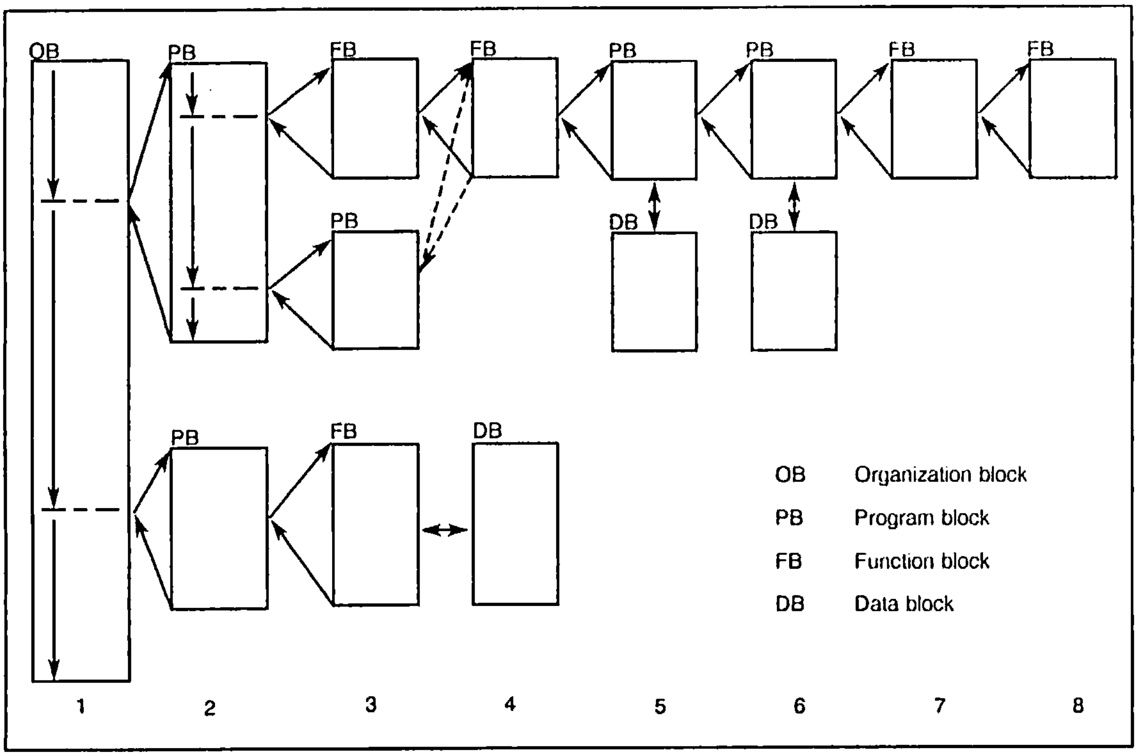


Fig. 1.3 Typical program organization in STEP 5 nesting depth 8 (maximum nesting depth = 12)

1.3.3 Program scanning methods

The user program can be scanned in three ways:

1) One-shot scan in the controller's restart routine (following "Power up" or "RESET"). The statements to be executed must be programmed in OB 20.

2) Cyclic scanning of the statements in OB 1 based on an NC MD 155-dependent timing grid. When the default is in force, the operating system calls OB 1 every 66 ms; the following alternative time grids according to the times shows:

MD 155	Timing grid for OB1
0	60 ms
1	66 ms
2	72 mS
3	78 ms
4	84 ms
5	90 ms

NC MD 155 permits only the standard value 1.

3) Event-driven, one-shot execution of the statements in OB 2, e. g. in response to an interrupt. Prerequisites for event-driven execution are as follows:

- Normal termination of the controller's restart routine
- Interrupt enable via PLC MD 2002, bit 0 = 0
  (no interrupt service)
- A change in the interrupt input byte specified per MD (PLC MD 0)

Cyclic scanning can be interrupted by the high-priority NC functions' interrupt service routine or delayed pending termination of OB 2 following execution of each STEP 5 command.

1.3.4 Influencing factor: User program size

As mentioned in Section 1.1, execution of the STEP 5 program should not take up more then 15 % of the total available CPU time, nor should the interrupt service routine in OB 2 take up more time than necessary because of its high priority, as this would delay execution of important NC functions.

For this purpose, the execution times of both the cyclic user program and the interrupt service routine are monitored at the hardware level. The permissible runtimes can be set in PLC MD 1 (maximum interpreter time OB 1 + OB 2) to 3 (maximum interpreter time PLC-interrupt service), whereby confirmation of the defaults ensures that the "integrated PLC" will not take up more than the admissible amount of CPU time. When these values are exceeded, the PLC goes to the state specified in bit 0 or 1 of PLC MD 2003; when the defaults are used, the "integrated PLC" always enters the Stop mode.

When the user programs exceed the permissible execution times only slightly, program scanning can be upheld by modifying these machine data bits, but this may adversely affect the NC functions.

There are two ways of ensuring the executability of large cyclic user programs (max. 6K words) without placing an excessive load on the CPU:

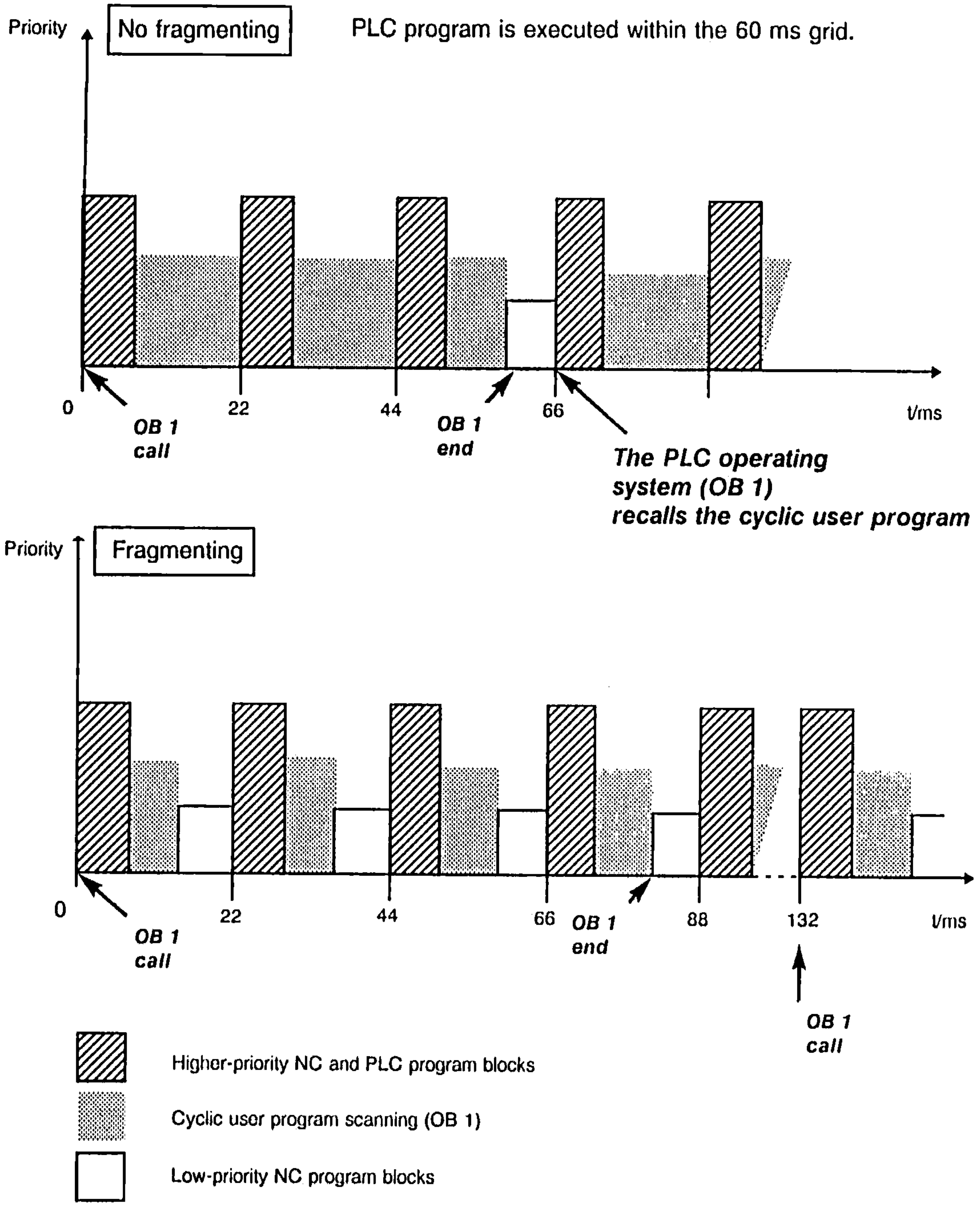
- Fragmenting cyclic execution through autonomous interrupts
- Allowing cyclic execution to be interrupted by higher-priority program blocks only

The mode is set in PLC MD 2003, bit 6 (fragmenting of S5 program execution). The percentage in MD 1 thus assumes the following meaning as regards the scan time:

- No fragmenting (bit 6 of PLC MD 2003 = 0 = default):
  15% of the cyclic program's timing grid (66 ms) results in a permissible runtime of 9.9 ms, when the default is used as basis.

- Fragmenting (bit 6 of PLC MD 2003 = 1)
  Using the default as basis, 15% of 1/3 of the cyclic program's timing grid (66 ms) results in a runtime of 3.3 ms. The cyclic program then interrupts itself autonomously, and resumes execution in the next third.

Graphic overview of the two methods for cyclic program scanning:


Priority
No fragmenting
PLC program is executed within the 60 ms grid.
0
22
44
66
t/ms
OB 1 call
OB 1 end
The PLC operating system (OB 1) recalls the cyclic user program
Priority
Fragmenting
0
22
44
66
88
132
t/ms
OB 1 call
OB 1 end
OB 1 call
Higher-priority NC and PLC program blocks
Cyclic user program scanning (OB 1)
Low-priority NC program blocks

When the OB 1 (cyclical user program) is too large to be executed within a 66 ms grid, fragmenting must be selected.

As already mentioned, the timing grid, based on the default, is 12 · scanning time (12 · 5.5 ms). The admissible execution time of the cyclic program for both variants can also be increased by augmenting PLC MD 1 (CPU load in %).

The admissible runtime for the interrupt service routine (PLC MD 3), regardless of the cyclic execution mode, may not exceed 2000 µs (standard value) in an interval of 4 · scanning time (4 · 5.5 ms = 22 ms).

The entire runtime may either be used for one-shot execution of the statements in OB 2 or, in an extreme situation, for 4-shot execution of OB 2, as the interrupt input byte is scanned for an edge change on the basis of the scanning time. If the default runtime for OB 2 proves insufficient, PLC MD 3 (maximum interpreter time PLC-interrupt service) must either be incremented (to max. 2500 µs) or the PLC set to the Stop mode when OB 2 exceeds the allotted time by modifying bit 1 in MD 2003. The example on the next page emphasizes the difference between the two variants for a cyclic user program with a runtime time of 25 ms.

No fragmentation:

The cyclic user program may be interrupted by higher-priority NC and PLC program blocks only (e. g. position control, interpolation, monitoring functions, programming and test functions, interrupt service routine (OB 2)). It is otherwise continued until it has terminated and all signals it generated have been forwarded to their destinations (NC, I/Os). Only then can the low-priority functions execute.

Depending on the size of the STEP 5 program in OB 1, the remaining interval may be quite short, as the operating system recalls the cyclic program when 66 ms have passed. The consequence is that too little time is available for the low-priority functions, a fact which becomes immediately apparent in a sluggish "operator interface". Display construction is extremely slow, as is the controller's reaction to keyboard entries. In extreme situations, the low-priority functions are allotted almost no time at all to execute.

In the example, the CPU load caused by the STEP 5 program alone would escalate to value x, where

$$x = \frac{25\text{ ms}}{66\text{ ms}} \cdot 100\ \% = 37\ \%,\ \text{assuming that cyclic execution}$$

is terminated within 66 ms. If this is not the case, execution will terminate in the next interval and be restarted immediately if no higher-priority functions are pending. In the example, no regard was paid to the 5% basic load resulting from data transfers prior to and following the OB 1 call.

It is obvious that a CPU load of almost 42% for the controller alone is not acceptable.

Fragmenting

When this variant is used, the cyclic program interrupts itself autonomously. If the default is used, the CPU load, refered to the cyclic program timing grid, would be:

$$x = \frac{3{,}3\text{ ms} \cdot 3}{66\text{ ms}} \cdot 100\ \% = 15\ \%$$

The basic load for the cyclic program, about 5%, must be added to this, so that the cyclic program load on the CPU would always be $\leq$ 20 %. It is less than 20% when cyclic execution is not terminated within one timing period. Extremely long cyclic user programs may cause the cycle time monitor to response (defaulöt: 300 ms).

By selecting the diagnostics function (MD 2003, bit 7 = 1), the user can obtain a general view of the anticipated run time for STEP 5 programs as well as the required cycle time. On the basis of these data, he can optimize the configuration of his MD as regards his job requests and the purpose for which the controller is to be used.

Options for initiating the diagnostics function:

1) One-time-only display via the "STATUS VARIABLE" programmer function

```
STATUS      VARIABLE

DB  1
DW  0       KF = <current cycle time in ms>
DW  1       KF = <interpreter runtime in OB 1 and OB 2 in µs> *
DW  2       KF = <cyclic interpreter runtime in µs> *
DW  3       KF = <interrupt-driven interpreter runtime in µs> *
DW  4       KF = <number of interrupts serviced> *
DW  9       KH = 8000
```

Note: The initiate bit in DW 9 is set once via the programmer, and is reset by the PLC operating system at the start of the cycle when the current values are transferred to DB 1.

* Values relate to the current scan time

2) Periodic display via the "STATUS" programmer function and a corresponding initiation routine in OB 1.

```
OB1

C  DB 1    }
A  F   0.7 }  Initiation routine
           }  in dependence on the flashing frequency (F 0.7)
=  D   9.15}


:          }  Actual
           }  user program
:          }

BE         }
```

Note: This enables a fairly accurate estimation of the fluctuations in the runtimes (cycle time and interpreter runtime are not constant values!), and is the only way to show the interrupt-driven interpreter runtime (OB 2) and the number of interrupts serviced (e. g. by simulating a change in the interrupt input byte at the hardware level).

Representation of the number format must be changed from KF to KH when the interpreter runtime exceeds 32768 µs, as the display would otherwise be incorrect.

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. Section 3 gives information on programming data blocks and section 4 information on programming function blocks. Program, organization and sequence blocks can be programmed in all three STEP 5 modes of representation 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.

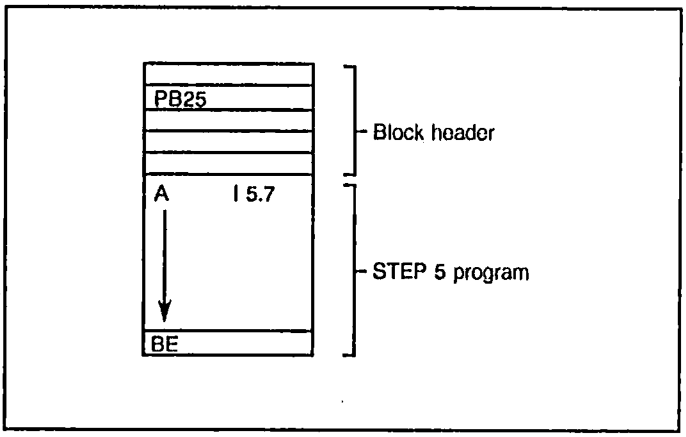


Fig. 2.1 Structure of a program block

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.
(Only organization blocks may not be invoked by the user program).
A block call is comparable to 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 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.

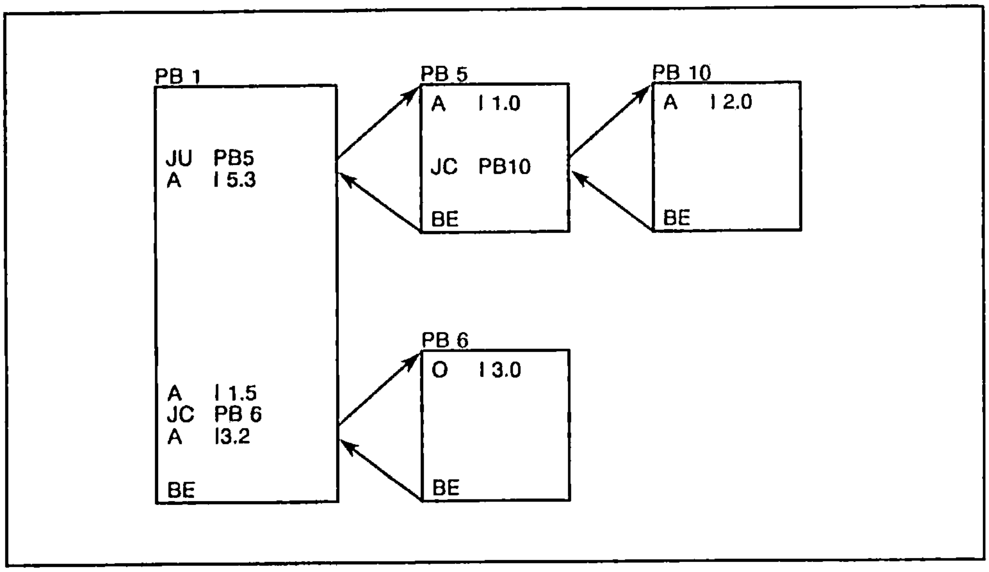


Fig. 2.2 Block calls for enabling execution of a program block

3 Data blocks

3.1 Programming Data Blocks

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

Data may be:
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 (example: DB99) may comprises as many as 256 data words (of 16 bits each) (Fig. 3.1). The data must be entered by word, beginning with data word 0.

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.

The operating system generates data blocks DB 1 (diagnostics DB), DB 36 (read/write job status for NC data) and DB 37 (initialize the two V.24 (RS232C) interfaces) in the controller's restart routine. The programmer (PG) prevents deletion of these blocks (message 70: DB in EPROM).

For test purposes, however, the user can invoke the "Force Variable" programmer function if he wants to view or modify data words.

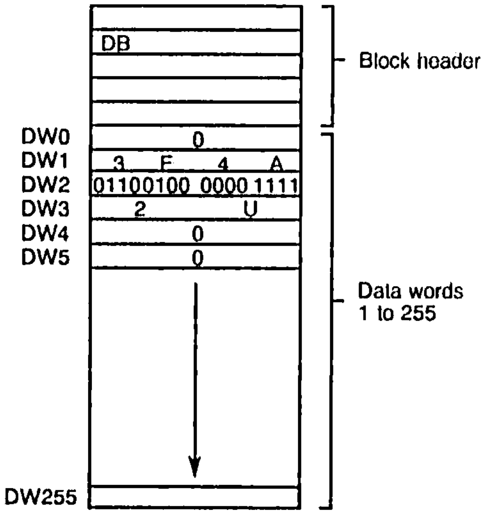


Fig. 3.1 Structure of a data block

3.2 Calling data blocks

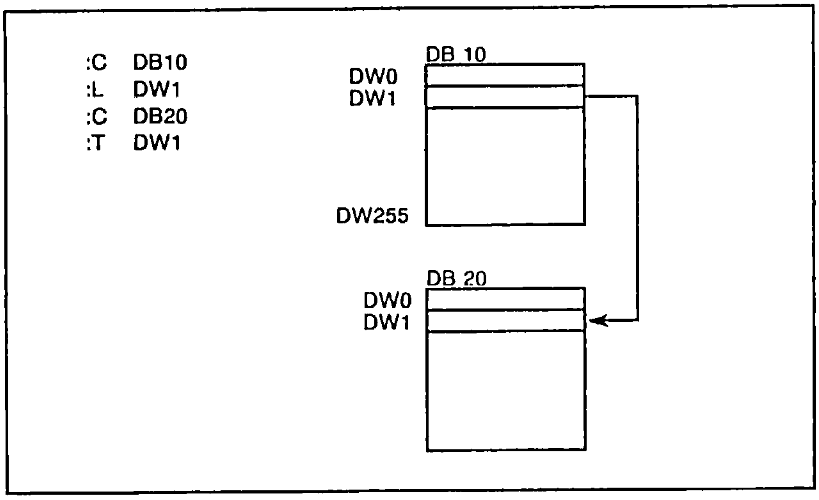
Data blocks can be called unconditionally only. Once called, a data block remains in force until the next is invoked.

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 xx" command calls a data block.

Example 1

Transferring the contents of data word 1, data block 10 to data word 1, 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 (function or sequence block) in which a data block has already been 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 selected or 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.

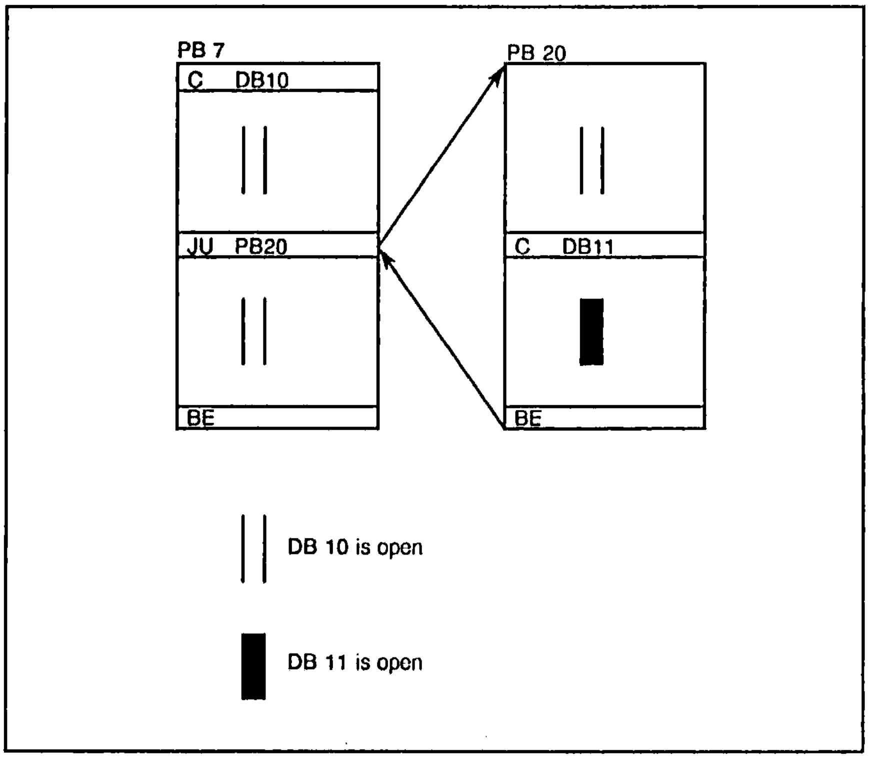


Fig. 3.3 Validity range of a selected data block

4 Function Blocks

4.1 General Remarks

Function blocks are used to implement frequently recurring or extremely complex functions.

Functions blocks (FBs) 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 (see also Section 9.4) 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. A function block programmed in the STEP 5 language can be programmed by the user himself, or may be purchased 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" (cf. Section 6.1).

4.2 Structure of Function Blocks

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

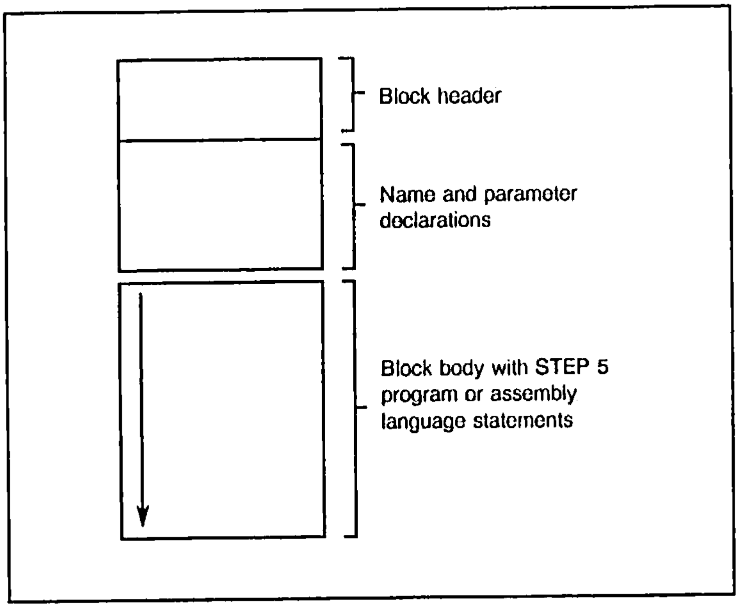

Block header
Namo and paramoter doclarations
Block body with STEP 5 program or assembly language statcments

Fig. 4.1 Structure of a function block

4.2.1 Block header

The block header contains all 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) 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).

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.

4.3.1 Call statement

JU FB n Unconditional call
JC FB n Conditional call

Unconditional call:

The function block is executed without regard to the RLO.

Conditional call:

The function block is executed only when the 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 macro in the PLC's basic software, 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 Operating Instructions).

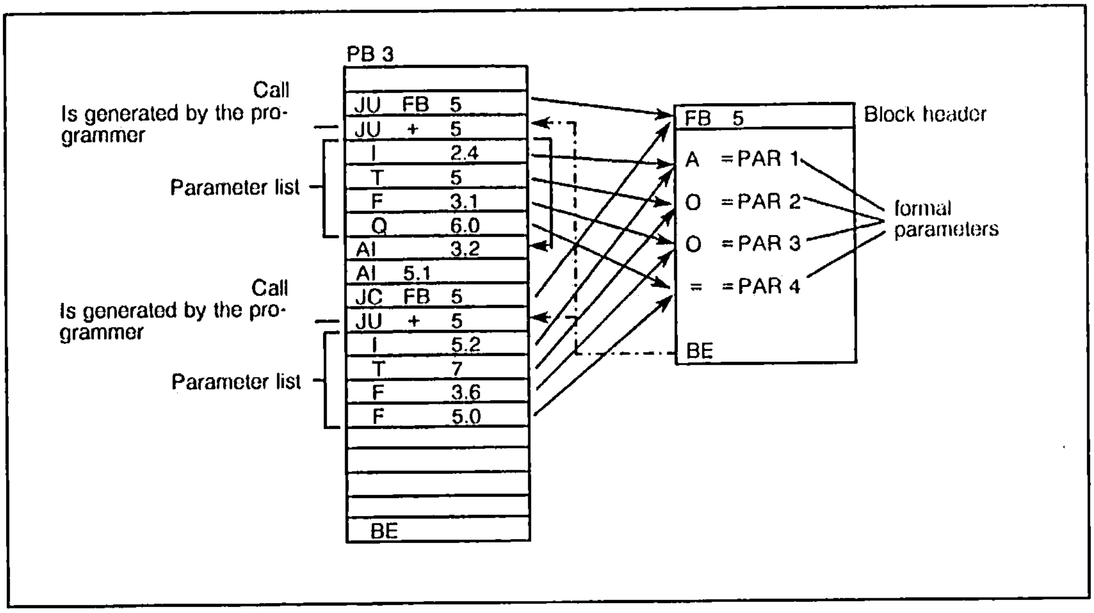


Fig. 4.2 Calling a function block

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 may be a number between 0 and 65535. The function block is assigned this number without regard to 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.

4.4.3 Formal operand (block parameter name)

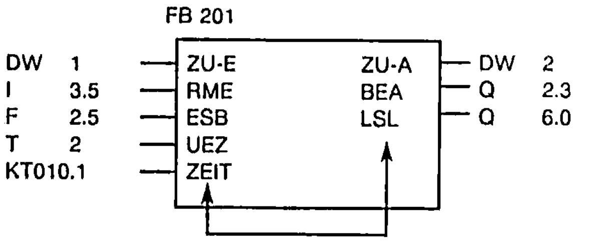
A formal operand may comprise no more than four characters, the first of which must be a letter. A maximum of 40 parameters can be programmed per function block.

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



Formal operands (names of the block parameters)

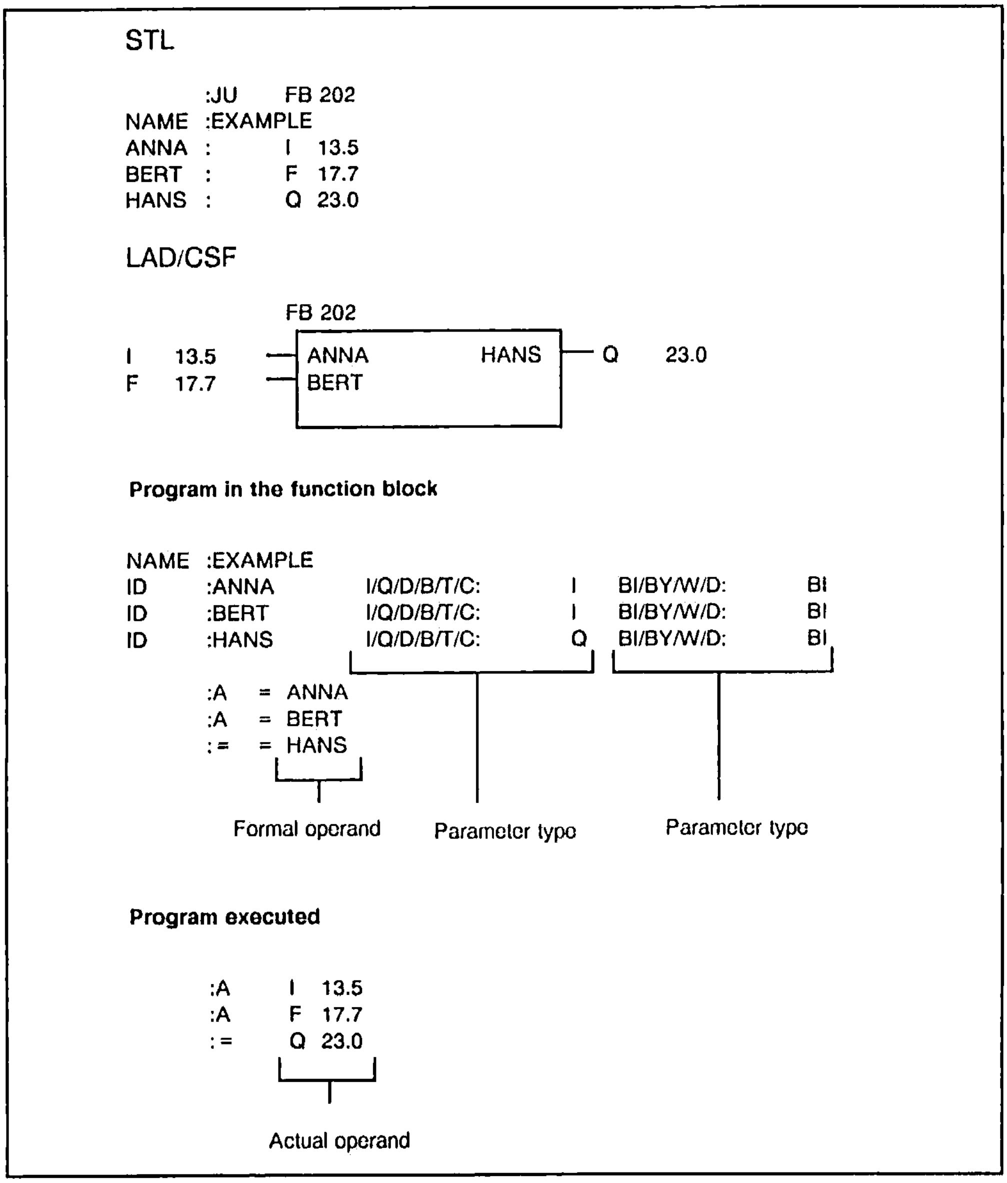
Fig. 4.3 Sample function block call

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 at the left of the function symbol and those of type "Q" at 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
```

Program executed

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

Fig. 4.4 Example: Calling a function block

5 Organization Blocks

5.1 General Remarks

The organization blocks form the interface between the operating system 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 operating system can invoke them. A user can only program organization blocks; he cannot invoke them (Fig. 5.1).

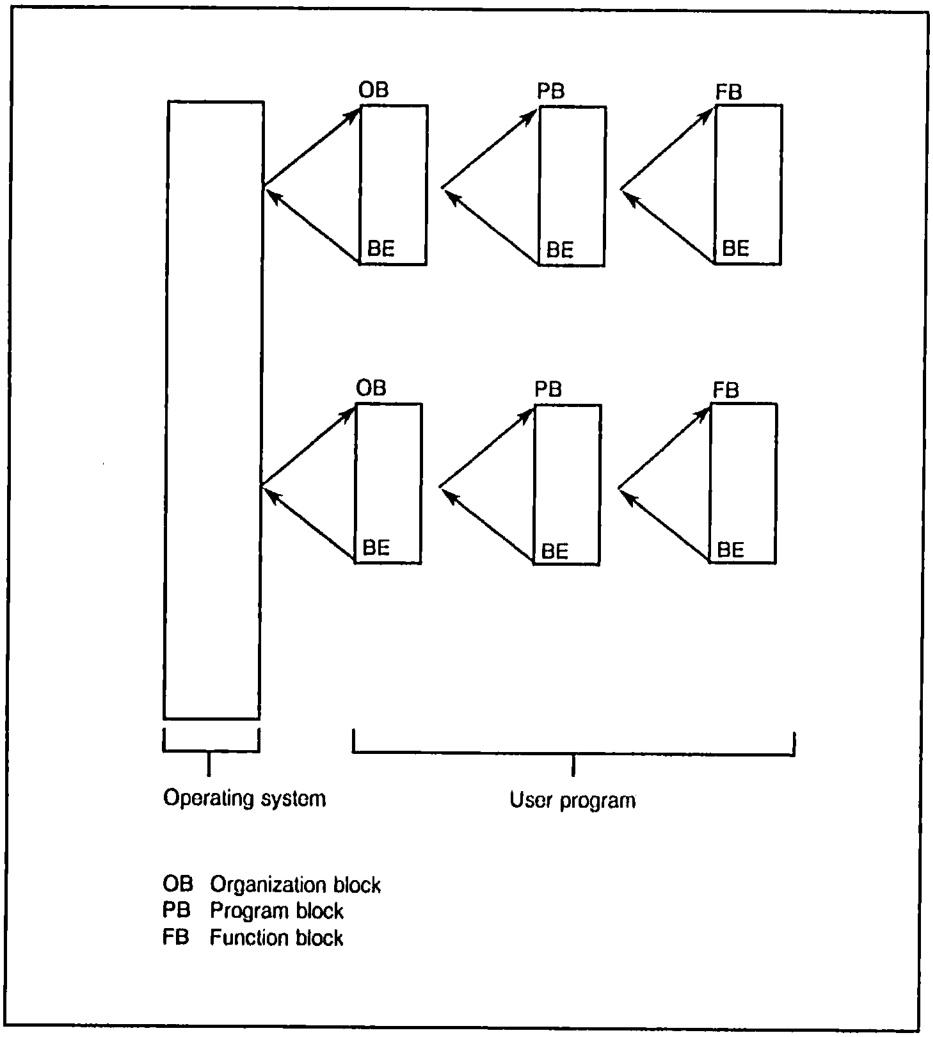


Fig. 5.1 PLC program

Appropriate programming of the organization blocks enables the following:

- One-shot execution following PLC restart (OB20)
- Cyclic execution (OB 1)
  (see "Programming the cyclic program")
- Execution of the interrupt service routine (OB 2)
  (see "Programming the interrupt service routine")

OB 2 must be available when interrupt processing is enabled (MD 2002, bit 0 = 0), otherwise the PLC will stop. OB 20 and OB 1 can be installed in the PLC as required. If one of these two OBs is missing this fact is ignored by the operating system.

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 Cyclic scanning

5.2.1 Interrupt capability

The cyclic user program can be interrupted after each MC 5 instruction. The last instruction is executed in its entirety and pending interrupts (caused by a process interrupt or higher-priority sections of the NC program) are enabled when the instruction been executed.

If the interrupt was caused by detection of an edge change in the interrupt input byte, all MC5 registers, as well as flag bytes 224 to 255, are saved; these registers and flag bytes are recovered following termination of the interrupt service routine. The data block open prior to the interrupt is thus once again valid.

5.2.2 Response time and scan time

a) Response time:

The response time to a process signal or a signal from the NC is defined as follows:
The PLC operating system forwards the signal to the user interface (I, F area), invokes and executes the cyclic user program (OB 1) and transfer the process output image updated by OB 1 to the I/Os and the NC.

The response depends on:

1) Type and length of the cyclic user program in OB 1

2) Execution mode of the cyclic user program (MD 2003, bit 6: fragmentation, no fragmentation)

3) Timing grid for the cyclic user program (MD 155: scanning time)

4) Execution time of the higher-priority NC program blocks and of the interrupt service routine (OB 2) where applicable

5) Instant: Process image updating by the PLC operating system and actual signal status change (I/Os, internal NC-PLC interface)

b) Cycle time:

The cycle time is the period between two updatings of the process input image by the PLC operating system. Because the PLC invokes the cyclic user program in accordance with a specific timing grid (NC MD 155-dependent) rather than restarting it immediately after it has terminated, the cycle time can never be less than 60 ms (default: 66 ms).

Like the response time, the cycle time is dependent on the conditions listed in 1-4 (above). It can be ascertained by invoking the diagnostic function (MD 2003, bit 7 = 1), and is monitored to ensure that it does not exceed a maximum value (PLC MD5 default is 300 ms). If it does, the PLC will stop.

5.2.3 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.

5.2.4 Interface between operating system and cyclic program

Organization block OB 1 is the interface between the operating system 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 1, "Program organization").

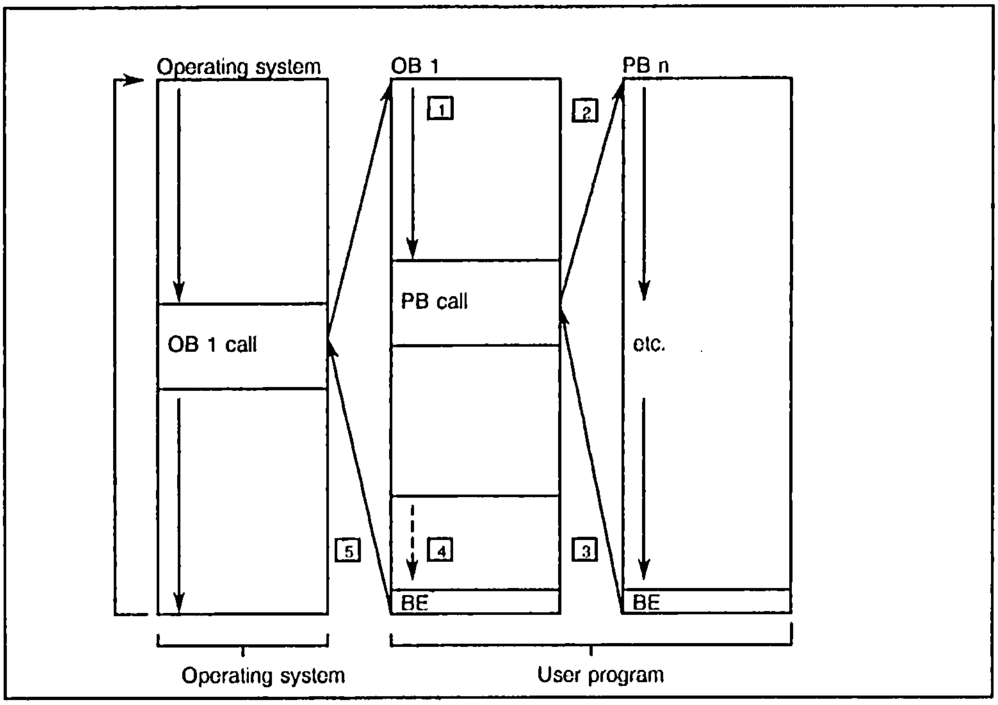


Fig. 5.4 Cyclic program scanning

1 First statement in the STEP 5 program.

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

3 Return from the last program or function block executed.

4 The organization block is terminated with BE.

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.

5.2.5 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).

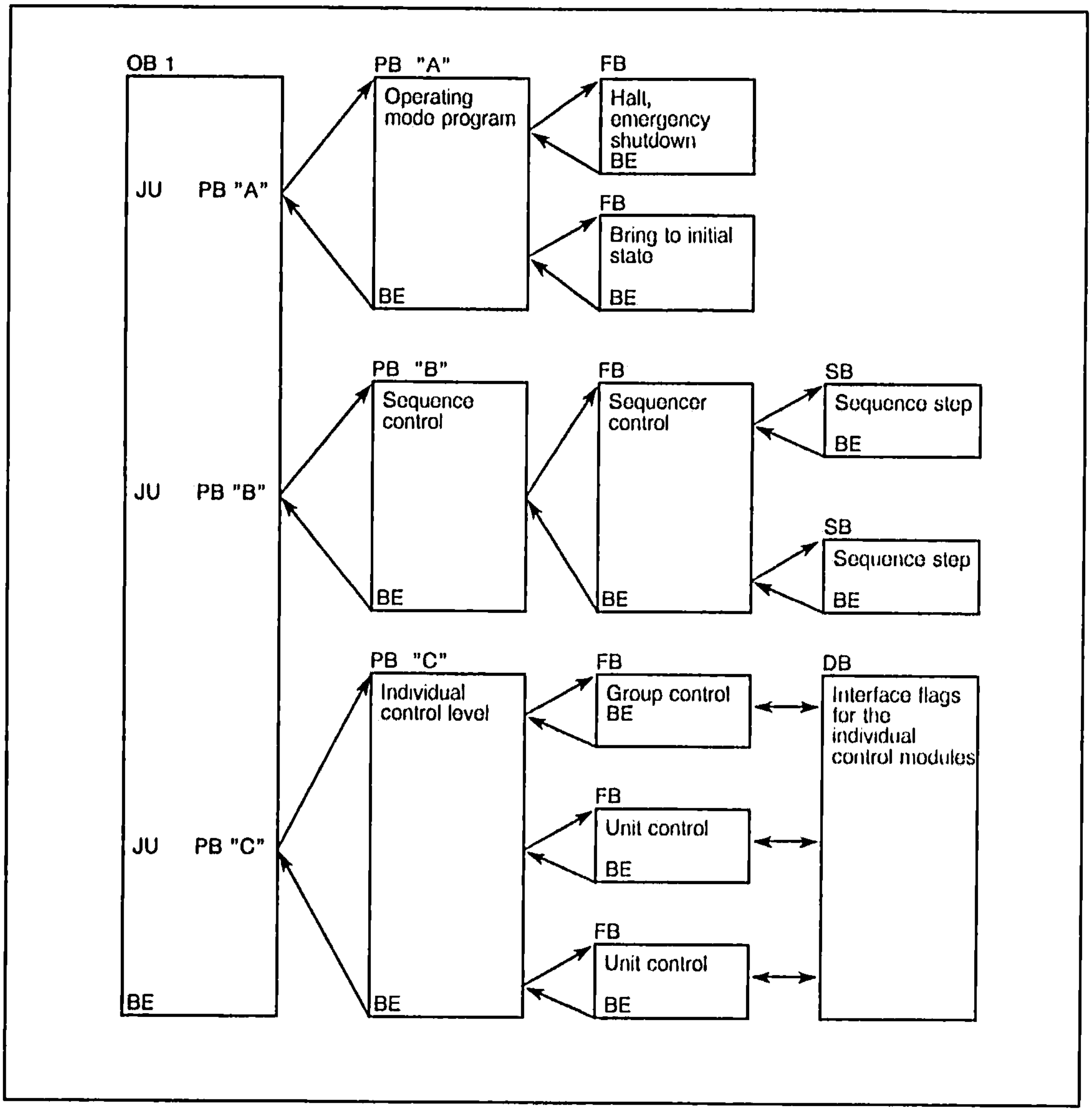


Fig. 5.5 Breakdown of the user program based on the progam structure

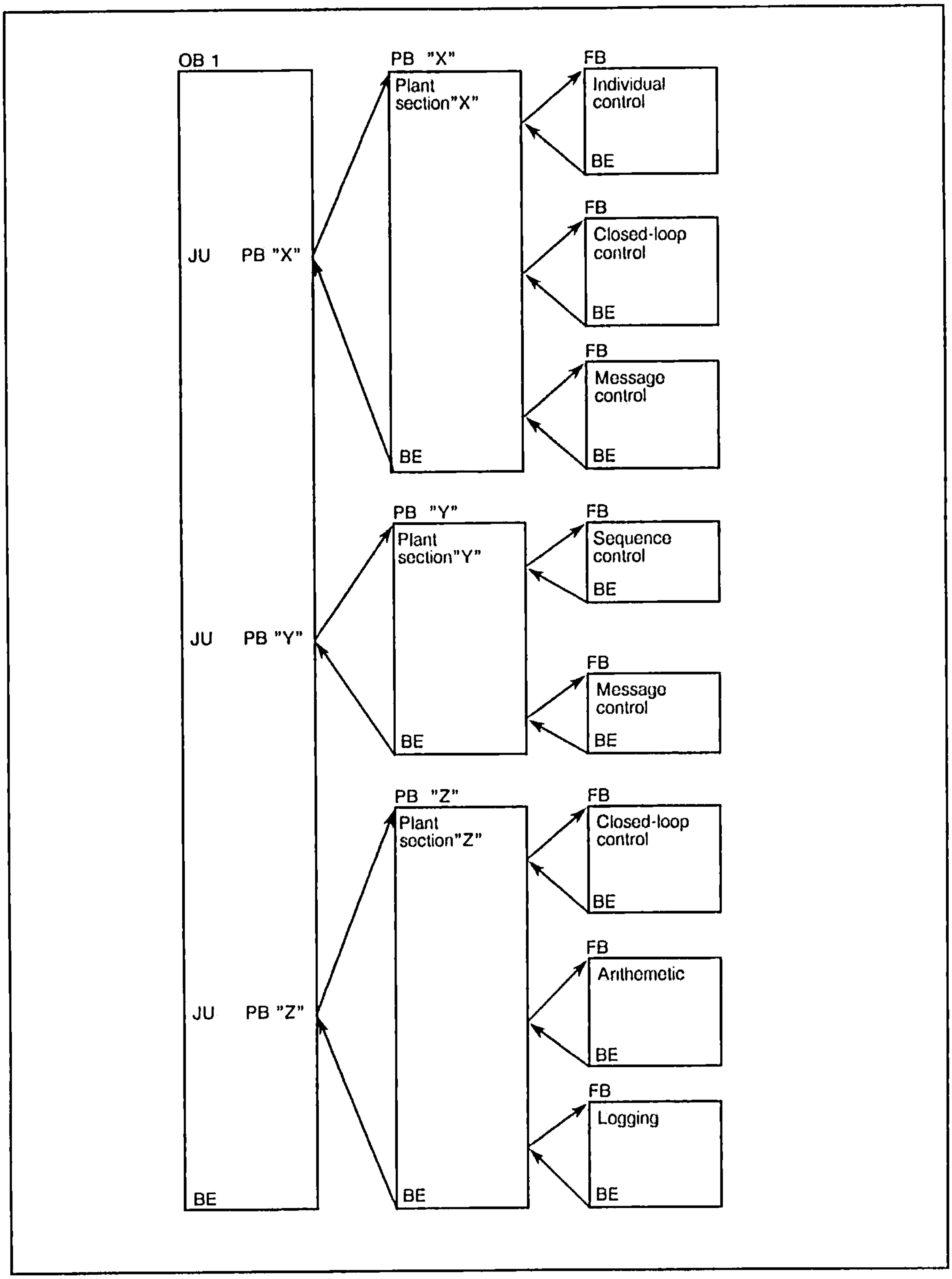


Fig. 5.6 Breakdown of the user program based on the plant structure

5.3 Interrupt Processing

5.3.1 Programming the interrupt service routine

The PLC has interrupt-processing capabilities.

In this mode, the cyclic program is interrupted and an interrupt service routine (OB 2) 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 when the signal state of specific input bits changes (edge-triggered).

Interrupt service routines allow the user to react immediately to process signals. 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.

Because the PLC operating system does not transfer the process image to the I/Os when OB 2 terminates, the user must himself influence the process peripherals with STEP 5 commands T PB/T PW.
Over the interrupt service routine, signals can be forwarded at high speed to the NC in output bytes QB 90 (program override channel 1) and QB 99 (program override channel 2); the NC then transfers these signals in a 22 msgrid (in dependence on NC-MD 155) to the internal NC PLC interface.

Note: No further processing takes place on the NC end up to this point.

5.3.2 Interface between operating system and the interrupt

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

OB 2 is always invoked when the signal state of a bit in the interrupt input byte changes.

The user may select the input bit and it via machine data (PLC MD 0: maximum value 31). MD 2002, bit 0 must also be set to 0 to enable interrupts.

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, i. e. the operating system calls OB 2, which contains the user's interrupt service routine.

The operating system checks the interrupt bytes every 5.5 ms (in dependence on MD 155), and invokes an interrupt service routine when required.

The type of signal change (positive or negative edge) is entered as input parameter in flag bytes FB 12 and FB 16.

The PLC operating system resets flag bytes FB 12 and FB 16 when OB 2 terminates.

5.3.3 Response time

The following factors influence the interrupt service routine's response time to an edge change in the interrupt input byte:

1. Whether the interrupt input byte is a global or local I/O byte
2. The sampling time (NC MD 155)
3. The instant at which the edge change takes place in the interrupt input byte and that at which the PLC operating system scans that byte

Approximate response times:

Global interrupt input byte:
The response time is between 2 ms and 7.5 ms.

Local interrupt input byte:
The response time is between 3.5 ms and 9 ms.

6 Integral Function Blocks

6.1 General Remarks

Function macros are function blocks that are written in assembly language and integrated in the operating system.

The user can invoke and initialize these blocks in the same way as STEP 5 blocks. They are used for time-critical functions, and execute much faster than comparable STEP 5 blocks. The user should therefore give the integral blocks preference over the conventional STEP 5 blocks.

Note the following when linking the blocks into the user program:

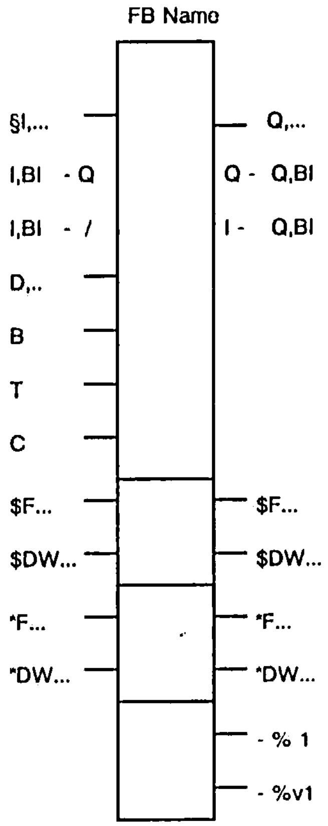
1. These blocks can be called by other blocks when the user program is generated ON-LINE (entered direct in the PLC from the programmer). The corresponding parameter table is displayed on the PG screen, and the user can make all required entries.

2. When programming direct on diskette, the function macros to be called must be disk-resident, i. e. the user must transfer the macros from the PLC to a workdisk. Only the block headers with parameter block are transferred.
   When these FBs are called in the user program, the user need only initialize the parameter table. The block headers must ***not*** be transferred when the user program is transferred to EPROM.

Loading the PLC blocks:

- STEP 5 function blocks that have the same numbers as function macros cannot be transferred to the PLC (PG message: "Block already in EPROM").

6.2 FB Identifiers

FB call	Parameter type	Data type	Permissible actual operands
 FB Name §I,... I,BI -Q I,BI - / D,.. B T C $F... $DW... *F... *DW... Q,... Q - Q,BI I - Q,BI $F... $DW... *F... *DW... - % 1 - %v1	I Input Q Output	BI Operand with bit address	I n.m Input Q n.m Output F n.m Flag
		BY Operand with byte address	IB n Input byte QB n Output byte FB n Flag byte DL n Data byte left DR n Data byte right PB n Peripheral byte
		W Operand with word address	IW n Input word QW n Output word FW n Flag word DW n Data word PW n Peripheral word
	B Block	Not applicable	DB n Data block FB n Function block PB n Program block SB n Sequence block
	T Timer	Not applicable	T n No. of the timer
	C Counter	Not applicable	Z n No. of the counter
	D Data	KM Bit pattern, 16 digits KY Two absolute values (1 per byte) from 0 to 255 KH Hexadecimal number, max. 4 digits KS Two alphanumeric characters KT Time (1.0 to 999.3) KC Count (0 to 999) KF Fixed-point number (-32768 to +32767)	
	I,BI I,BI -Q I,BI - / §I,.. $... *...	Static input signal Input signal acknowledged by FB Input signal whose rising edge is evaluated DW not allowed as parameter Input signal which must be supplied prior to calling the FB Fixed input signal which need not be generated	
	Q,BI Q- Q,BI I- Q,BI $... *... % 1 %v1	Static output signal Output signal which must be acknowledged by the user Output signal for one cycle (pulse) Output signal at defined flag word or data word which can be evaluated immediately following the FB Defined output signal, e. g. NC signal Error code ACCU 2 on system stop (STS); ACCU 1 error code Aux. spec.: No. of the interface byte in ACCU2's HIGH byte	

6.3 Function macros

FB no.	FB ID	FB NAME	Page
11	EINR-DB	Generate data blocks	
60	FB BLOCK-TR	Block transfer	
61	FB NCD-LESE	Read NC data	
62	FB NCD-SCHR	Write NC data	
.			
.			
65	M→STACK	Transfer flags → flag stack	
66	STACK→M	Flag stack → transfer flags	
.			
.			
80			
81			
82			
83			
84			
85			
86			
87			
88			
.			
.			
.			
190	K-LEITPC	Data interchange with master PLC	

FB 11 EINR-DB Generate data blocks

1. Description

The EINR-DB function block is used to generate data blocks for variable data in the PLC's RAM.

EINR-DB generates the specified data block, e.g. in the cold restart routine (OB 20), but only when the DB is not yet in the address table and a valid data word number (DWNR < 255) is specified.
The interface flags (FW 246 and FW 248) are evaluated when parameter AN = 0.

- 255 < DW number < 0
- Data block number > 255
- Data block number = 0
- Not enough RAM available in PLC
- Data block already in PLC and lengths are not identical

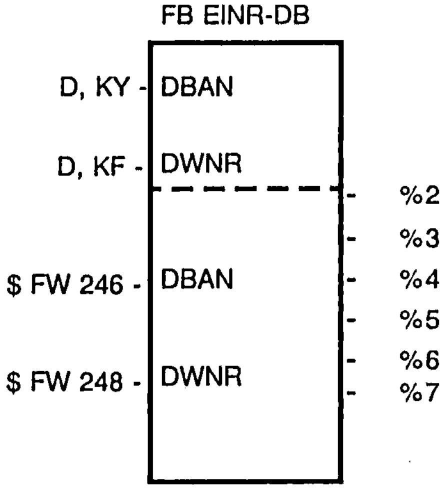
A detailed error identifier is entered in ACCU 2 (ACCU 1 contains the block number). The data block is initialized to zero.

Note: The data blocks for the basic software (interface data blocks) are generated automatically in the cold restart routine.

2. Block data

Lib.-No.	:
FBs to be loaded	: None
DBs to be loaded	: None
Type of FB call	: unconditional or conditional (JU FB11 or JC DB11)
DBs to be entered	: None
Error messages	: %2 DB no. > 255 %3 Spec. DWNR < 0 %4 Length of DBto be generated not identical with length of DB in PLC %5 Not enough RAM available in PLC %6 Spec. DWNR > 255 %7 DB0 cannot be generated

3. Block call



4. Signals

DBAN Number of data blocks to be generated and their numbers.

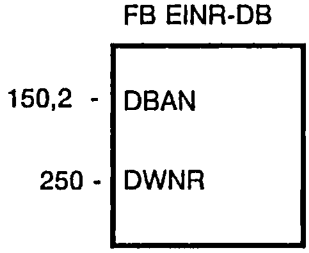
Special case:

When parameter AN is 0, the function block can be initialized over flag words FW 246 and FW 248.

High-Byte : DB number
Low-Byte: Number of data blocks (minimum of 1)

DWNR No. of the last DW

5. Example



Data blocks 150 and 151 are generated from data word 0 to data word 250.

FB 60 BLOCK-TR Block transfer

1. Description

Function block BLOCK-TR transfers the specified number of data words from a source DB located in RAM or EPROM to a destination DB in RAM.

The user may specify both the start of the source DB block to be transferred and the start data word in the destination DB.

Prior to initiating the transfer, the PLC checks to make sure that:

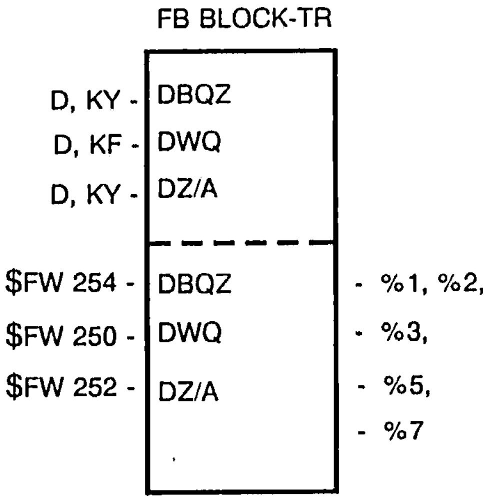
- the source DB in the PLC has the required length
- the destination DB in the PLC has the required length
- the destination DB is located in RAM
- the number of data words to be transferred is > 0 and < 128.

The PLC enters the Stop loop when an error is detected. A detailed error identifier is entered in ACCU 2 (ACCU 1 = block number). Flag words FW 250-254 are scanned when the "DB QZ" parameter is zero.
(Initialization: 0,1 1,0 or 0,0).

2. Block data

Lib. no.	:
FBs to be loaded	: None
DBs to be loaded	: None
Type of FB call	: Absolute or conditional (JU FB 60 or JC FB 60)
DBs to be entered	: None
Error messages	: %1 No. of DWs to be transferred > 127
	%2 No. of DWs to be transferred = 0
	%3 No source DB
	%4 No destination DB
	%5 Destination DB too short
	%6 Destination DB is in EPROM
	%7 Source DB too short

3. Block call



4. Signals

DBQZ Source DB and destination DB numbers

High-Byte : Source DB
Low-Byte : Destination DB

DWQ Start of the data block to be copied in the source DB

DZ/A Start of the data block in the destination DB and number of data words to be transferred

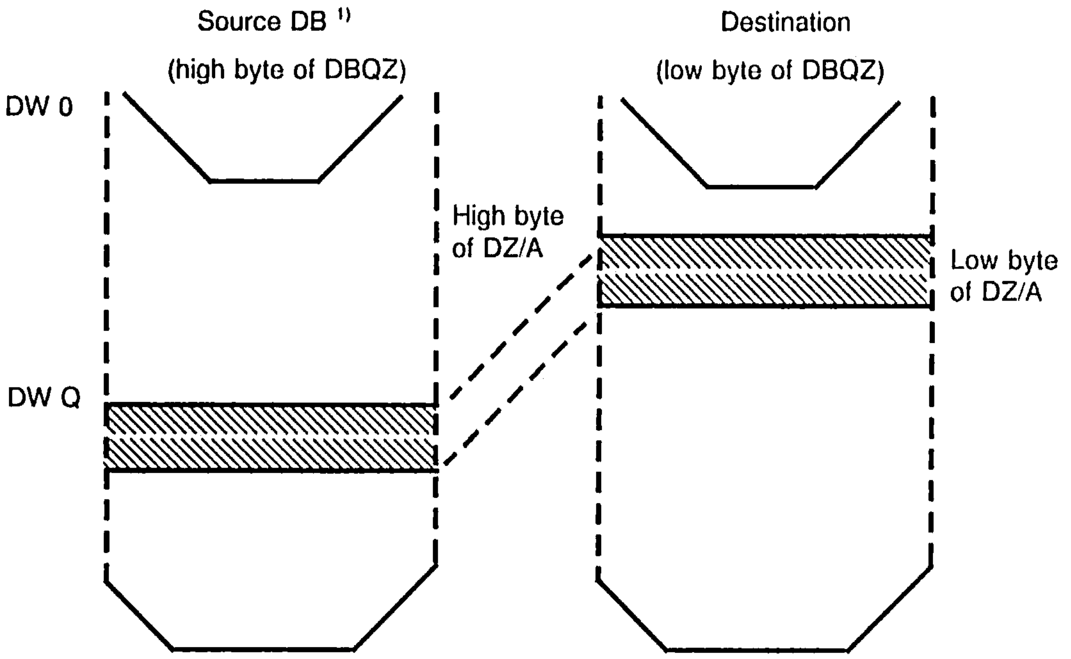
High-Byte : Start of the data block
Low-Byte : Number of data words to be transferred

5. Example

```
             FB  BLOCK-TR
            ┌──────────┐
  10,11  -- │ DBQZ     │
      8  -- │ DWQ      │
  12,05  -- │ DZ/A     │
            │          │
            └──────────┘
```

5 data words, beginning with DW 8, are copied from DB 10 to DB 11; the first destination data word is DW 12.

6. Data transfer diagram



[1)] The "source" data block may be located in either RAM or EPROM

FB 61 NCD-LESE Read NC data
FB 62 NCD-SCHR Write NC data

1. Description

Function blocks FB 61 and FB 62 are used to read NC data from/write NC data to the PLC. Prerequisite is that the job-controlled data interchange between NC and PLC has been enabled by an option bit (error 8 is output when this is not the case).
Parameters must be initialized to inform the function blocks of the data source in the NC or PLC and of the data destination in the PLC or NC. The function blocks must be assigned a byte in the interface data area (DB 36) over the NBSY parameter; this byte informs the function blocks of the current state of the data transfer.
The NC/PLC data transfer function is described in detail in Section 7. NC interrupt 3016 is issued when the NC's data type is unknown.

2. Block data

Lib. no. FB61	:
FB62	:
FBs to be loaded	: None
DBs to be loaded	: None
Type of FB call	: Unconditional or conditional
DBs to be entered	: None
Error messages	: ACCU 1 (FB no.) = 61 or 62 ACCU 2, high byte, = No. of the interface byte, i. e. the number of the job that produced the error (exception: error 8).

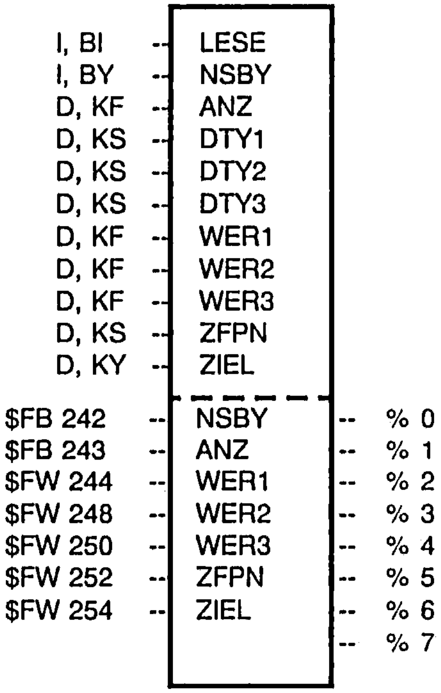
ACCU 2 low byte = detailed error code:

0 : ANZ >1 not allowed
1 : Invalid interface byte
2 : Addressed data word missing / DB missing or DB no. / FW no. invalid
3 : Invalid data type
4 : *ANZ =< 0 or >80
5 : Illegal read / illegal write
6 : Invalid number format
7 : Value 3 (WER3) not 0 (ZOA) or 1 (ZOFA)
8 : Read /write NC data not enabled by option bit

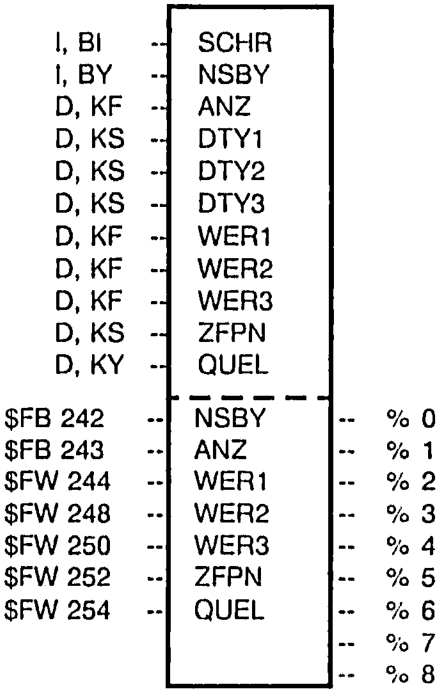
* Refer to the exceptions for the ANZ parameter

3. Block call

FB 61: NCD-LESE



FB 62: NCD-SCHR



4. Signals

LESE Command for data transfer

SCHR The parameters are transferred to the FIFO buffer (job buffer) when the LESE (read) or SCHR (write) signal = 1.
In case of an *unconditional block call*, the user has to reset the LESE or SCHR signal at the end of the data transfer.
On an *unconditional block call*, the signal must be set to "1" (F 0.1) (see also timing diagrams). Data transfer can be activated only when the initialized interface byte's (NSBY's) DATA TRANSFER ENABLED signal is 0.

NSBY The function block must be assigned a byte in the interface data area (DB 36) from which it can ascertain the current state of the data transfer.

Exception: Variable initialization; refer to the overview on the next page.

Note:

When byte DL 32 (= number 65) is used as interface byte, the specified job request is serviced on an "interrupt-driven" basis, i. e. it is "sandwiched" between the job requests that are already in the request buffer. With job number 65 and ANZ > 1, the interrupt job is executed completely.

ANZ Number of data words to be transferred. If more than one word is transferred (ANZ > 1), both source and destination addresses are incremented. If this results, for example, in addressing of a data word that is no longer available, the PLC goes to STOP with % 2.
Note that a different number of data words is required for each value transferred, depending on the ZFPN parameter.

Exceptions:

When ANZ is 0 or 128, the NSBY, ANZ, WER1-WER3, ZFPN and ZIEL/QUEL parameters are initialized over flag words. The information presented in 5. below (Variable initialization) discusses the difference between ANZ = 0 and ANZ = 128.

DTY1 2 to 6 ASCII characters; CL800 mnemonics.

DTY3 The values must be input, beginning with DTY1.

Note:

Unused positions must be padded with blanks. Refer to the table in Section 7.

ZFPN PLC/NC number format
Refer to Section 6 for details on the PLC/NC number format

ZIEL Data destination in the PLC (FB 61)
QUEL Data source in the PLC (FB 62)

High byte	:	DB number (2-255); exception: 0 = flag
Low byte	:	- Word number (for data blocks)
		- Byte number (for flags), (136-255 permitted)

Note:

1. The number of bytes required must accord with the ZFPN parameter.

2. If, for example, a 16-bit word is to be passed to the NC, the value, when transferred, must be written from data words into the initialized word k + 1 when number formats FO-FF are needed.
   The parameterized word k has to be set to 0 (see Section 6).

5. Variable initialization

When the variable initialization option is used, the values must be written into the flag words in the same format as for direct initialization in the FB.

Example:

The following must be programmed to initialize the ZFPN parameter with F1:

```
.
.
.
L   KSF1
T   FW252
.
.
.
```

Exception:

The NSBY parameter can be initialized in one of two ways.

- When ANZ = 0, the number of the interface byte must be entered in FB 242 (refer to Section 1.5, Interface Description).

No. of the interface byte		Byte number
1	≙	DLO
2	≙	DRO
.		.
.		.
.		.
64	≙	DR31
65 (interrupt-driven)	≙	DL32

Example:

The following must be programmed when DL15 is to be used as NSBY:

```
:
L     KB31
T     FB 242
```

- When ANZ = 128, the OP-code[1], with parameters from L DL xx/L DR xy, must be entered in FW 241. This method of initialization is particularly suitable, for example, when FB 61 is called as a subroutine of a function block or when NSBY parameter was declared in "I/BY" data format in the calling FB. The programmer generates the right code in the parameter table when the calling FB is initialized, thus enabling it to be passed to FB 61/FB 62 with the statement sequence.

```
:
LW    = ABCD is a parameter (data type: I/BY) of the calling FB.
T     FW 241
:
```

an den FB 61 / FB 62 übergeben werden kann.

Sample OP-code:

No. of the interface byte	Byte No.	OP-code (hexa)
.		
.		
.		
31	DL15	220F
32	DR15	2A0F

Example:

```
FB 140              SPRM-A                 LIB  =  14050

NETWORK 1
NAME       : STAZ
ID         : DBZW   I/Q/D/B/T/C: B
ID         : K      I/Q/D/B/T/C: I         BI/BY/W/D : BY
ID         : BEAR   I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : ZEAB   I/Q/D/B/T/C: Q         BI/BY/W/D : B
ID         : FEHL   I/Q/D/B/T/C: Q         BI/BY/W/D : B
ID         : T/M    I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : SOSP   I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : AADR   I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : IST    I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : NSBY   I/Q/D/B/T/C: I         BI/BY/W/D : B
ID         : P-NR   I/Q/D/B/T/C: D   KM/KH/KY/KS/KT/KS/KG/ : KF
           :
           :
           :
           : LW     = DBZW !               PARAMETER INITIALIZATION
           : T      FB 254  ZIEL, DB No.   FB 61 TOS COMMAND
                                           (Read tool correction)
           :L       KB40    !
           :T       FB 255  ! ZIEL, DW No.
           :L       KS F5   !
           :T       FW 252  ! ZFPN

           :LW      = NSBY  !              The OP-code of the "NSBY" parameter from
                                           calling FB 140 is stored in FW 241.
           :T       FW 241  ! NSBY
           :L       KB1     !
           :T       FB 243  ! ANZ
           :LW      = P-Nr. !
           :T       FW250   ! WER3
           :L       DR 54   !
           :T       FW 244  ! WER1
           :L       DR 51   !
           :T       FW 248  ! WER2
           :JU      FB 61
NAME       :NCD     LESE
LESE       :        F0.1
NSBY       :        FB242   (all other addresses are valid)
ANZ        :        KF + 128
DTY1       :        KSTO
DTY2       :        KSS
DTY3       :        KS
WER1       :        KF + 0
WER2       :        KF + 0
WER3       :        KF + 0
ZFPN       :        KS00
ZIEL       :        KY0,0
           :
```

6. Overview of NC / PLC number formats

The format of the numbers to be passed to the PLC is specified with the "Number format" parameter. The number formats permissible for all data types are listed in the data type table. The PLC recognizes the following number formats.

Bit pattern

BI

Structure: DWn MWn	DL	DR
	unassigned	1 0 1 1 1 0 1 0

Example:

R100 = 10111010
The "Number format" error is reported when the R parameter contains individual digits.

BCD number with sign and decimal point

BO
:
B9
BG

Structure:

			DL	DR	
DWn/ FWn	unassigned		Sign: 1	8	Sign = 0 Positive Sign = 1 Negative
DWn +1/FWn+2	7	6	5	4	
DWn +2/Fwn +4	E	3	2	1	

Example:

R100 = - 87654.321
ZFPN parameter = B4

Note:

The first letter in the parameter (in this case B) indicates that the data source or data destination in the PLC is in BCD. The second character in the parameter (4 in this case) indicates that the BCD number is stored in the PLC with three decimal places (FB 61).
On a data transfer from PLC to NC (FB 62), the parameter specifies the location of the decimal point in the NC, regardless of where it is located in the PLC.
If PLC data words contain the value 1234.45, the ZFPN parameter is initialized to B2 for a transfer from PLC to NC, and the number 1234.5 entered in R100.

The table below presents an overview of permissible parameters.

ZFPN	Description
B0/F0	Value without decimal point (e.g. 1234)
B1/F1	Value with decimal point (e.g. 1234.)
B2/F2	1 decimal place
B3/F3	2 decimal places
B4/F4	3 decimal places
B5/F5	4 decimal places
B6/F6	5 decimal places
B7/F7	6 decimal places
B8/F8	7 decimal places
B9/F9	8 decimal places
FA	Linear-axis position value *
FB	Rotary-axis position value *
FC	Linear-axis feedrate *
FD	Rotary-axis feedrate *
FE	Rotational feedrate *
FF	Rotational speed value *
BG	Value as stored
BI	Bit pattern

* Depending on the input system (which is set via machine data), these number formats determine only the position of the decimal point within the R parameter value without physical unit.
No checks are made for range violations.

FO 32-bit fixed-point number
:
FF

Structure:

	DL	DR
DWn/FWn	2^{31}...	...2^{16}
DWn + 1/FWn + 2	2^{15}...	...2^{0}

Example:

If ZFPN = F4, R100 = 1234.567 produces a PLC value of 1234567;
i. e. in contrast to BCD, the decimal point is masked (FB 61).
For a data transfer from the PLC to the NC (FB 62), the parameter specifies where the decimal point is to be located in the NC.
PLC data = 121457 and ZFPN=F2 produces NC data item 12145.7.

7. Table for data transfer NC/COM ↔ PLC data words/flags

Functional description	Data type (DTY1-DTY3)	Range	Value (WER1 - WER3)	Number format (ZFPN) FB 61 *1	Number format (ZFPN) FB 62 *1	Max. value of parameter ANZ *2
Machine data						
Machine data NC	MDN <Address>	0...4999	1	B0,F0 (*3)	B0,F0 (*3)	80
Machine data NC bytes	MDNBY <Address>	5000...9999	1	BI	BI	80
Setting data						
Setting data NC	SEN <Address>	0...4999	1	B0,F0 (*3)	B0,F0 (*3)	80
Setting data NC bytes	SENBY <Address>	5000...9999	1	BI	BI	30
Tool offsets						
Tool offset	T0S <range> <D No.> <P No.>	0 1 - 99 0 - 9	1 2 3	BG B0-B9 F0-F9	BG B0-B9 F0-F9	10
Tool offset add.	T0A <D No.> <P No.>	 1 - 99 0 - 9	 1 2		B0-B9 F0-F9 BG	1
Zero offsets						
Settable zero offset (G54 - G57) coarse/fine	Z0A <Group> <Axis No.> <c/f>	1 - 4 1 - 4 0/1	1 2 3	B0-B9,F0-F9	B0-B9,F0-F9 BG	1
Programmable zero offset (G58, G59)	Z0PR <Group> <Axis No.>	1 - 2 1 - 4	1 2	B0-B9,F0-F9	B0-B9,F0-F9 BG	1
Settable zero offset, additive, write only, coarse/fine	Z0FA <Group> <Axis No.> <c/f>	1 - 4 1 - 4 0/1	1 2 3		B0-B9,F0-F9 BG	1
External zero offset from PLC	Z0E <Axis No.>	1 - 4	1	B0-B9,F0-F9	B0-B9,F0-F9 BG	1
PRESET offset	Z0PS <Axis No.>	1 - 4	1	B0-B9, BG,F0-F9	B0-B9,BG,F4	1
Total offset	Z0S <Axis No.>	1 - 4	1	B0-B9,BG,F0		1

* 1 The relevant FB cannot execute a data transfer unless a number format has been specified

* 2 For data types with more than one parameter, the number of parameters is located in the line containing the WERT value parameter that is incremented.
A number > 1 is possible only under the following conditions:
- Data block in the NC is closed
- PLC source or destination address of sufficient length
  Three words per value for B0/B9/BG
  Two words per value for FO-FF
  One word per value for BI

* 3 Input/output is carried out in accordance with the format of the machine/setting data specification.

1) Apply from SW 2 onwards

Functional description	Data type (DTY1-DTY3)	Range	Value (WER1 - WER3)	Number format (ZFPN) FB 61 *1	Number format (ZFPN) FB 62 *1	Max. value of parameter ANZ *2
Actual values						
Axis position Workpiece-related	ACPW < Axis No. >	1 - 4	1	B0-B9/BG, F0		1
Axis position Machine-related	ACPM < Axis No. >	1 - 4	1	B0-B9/BG, F0		1
External setpoints [1]						
External contour feedrate	EXBF < Chan.no. > < lin/rot >	1 - 2 0/1	1 2	B4, F4	B4, F4	1
Program data [1]						
Program pointer for current block	PP < Chan. no. > < Level >	1 - 2 0 - 3	1 2	B0, F0		1 *4
Program selection						
Selection of an NC program	INITMP < Chan.no. >	1 - 2	1		B0, F0	1
Selection of an NC subroutine	INITSP < Chan.no. >	1 - 2	1		B0, F0	1
R parameters						
R parameter, NC channel	RPNC < Chan.no. > < Paramet. >	1 - 2 0 - 499	1 2	B0-B9,F0-FF BG, BI	B0-B9,F0-FF BG, BI	80
R parameter, global	RPNC < Chan.no. > < Paramet. >	0 900 - 999	1 2	B0-B9,F0-FF BG, BI	B0-B9,F0-FF BG, BI	80

* 1 The relevant FB cannot execute a data transfer unless a number format has been specified.

* 2 For data types with more than one WERT (value) parameter, the number of parameters is in the line containing the WERT parameter that is incremented.
A number > 1 is possible only under the following conditions:
- Data block in the NC is closed
- PLC source of destination address of sufficient length:
  Three words per value for B0-B9/BG
  Two words per value for F0-FF
  One word per value for BI

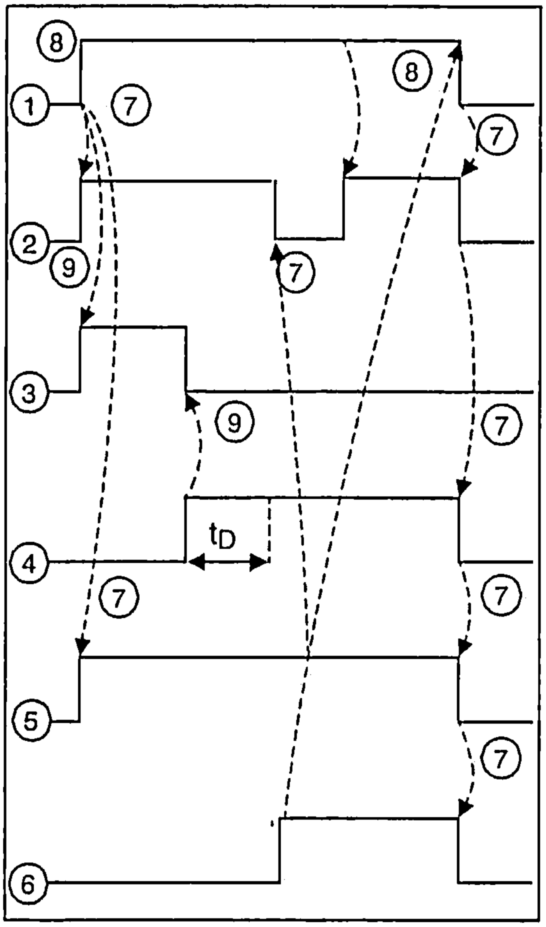
* 3 The value is passed as it is defined internally on the basis of the NC machine data for input resolution.

* 4 Three items of information are provided on reading out the program pointer when ANZ = 1:

- Level 0: Program type	(0 = no program selected, 1 = main program, 2 = subroutine), program number and block number
- Level 1 and up:	Subroutine number, number of passes and block number

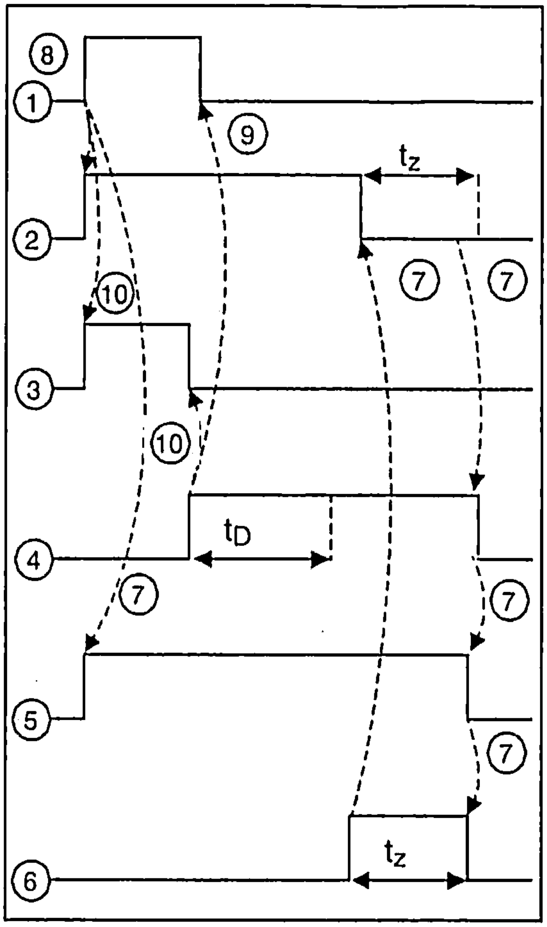
[1] Apply from SW 2 onwards

8a) Timing diagram for the interface signals



1 : READ/WRITE
2 : DATA TRANSFER REQUESTED
3 : FIFO FULL
4 : DATA TRANSFER IN PROGRESS
5 : DATA TRANSFER BLOCK BUSY
6 : DATA TRANSFER TERMINATED
+ ERROR, if any
7 : Signal change initiated by FB
8 : Signal change initiated by user
9 : Signal change initiated by FB,
not applicable if FIFO not yet full
t_D : Data transfer block is using
internal interface

8b) Conditional block call

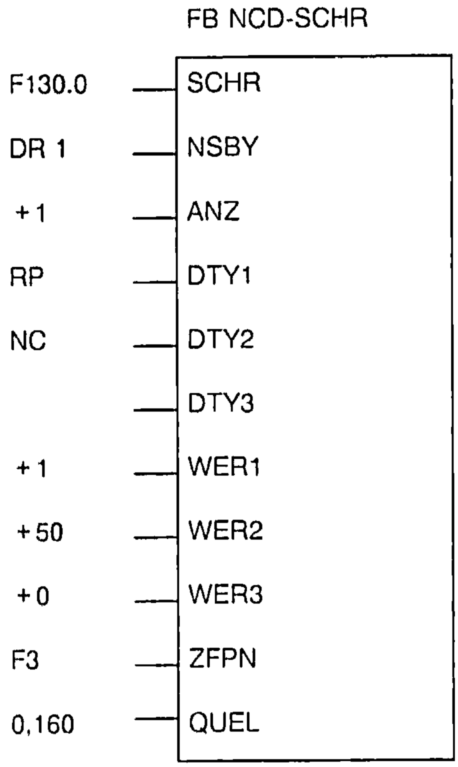


1 : READ/WRITE
2 : DATA TRANSFER REQUESTED
3 : FIFO FULL
4 : DATA TRANSFER IN PROGRESS
5 : DATA TRANSFER BLOCK BUSY
6 : DATA TRANSFER TERMINATED
+ ERROR, if any
7 : Signal change initiated by FB
8 : Signal change initiated by user
9 : User no longer needs block
10 : Signal change initiated by FB,
not applicable if FIFO not yet full
t_z : PLC cycle time
t_D : Data transfer block is using
internal interface

9. Sample parameters for FB 62:

Transferring R parameter 50 from FW 160 and FW 162 to channel 1.

PLC number format: Fixed point
NC number format: 2 decimal places



(Blank page)

FB 65 M→STACK
Transfer flags to stack

1. Description

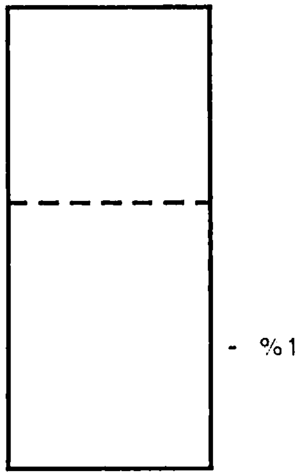
FB 65 is used to save flag area FB 224 - FB 255 in the flag stack to protect intermediate results and transfer flags of the function blocks against overwriting, e. g. when using nested calls (one FB calls another FB which uses the same flag area). FB 65 can be called in conjunction with FB 66 only.

2. Block data

Lib. no.	:
FBs to be loaded	: None
Type of FB call	: Unconditional or conditional
DBs to be input	: None
Error messages	: %1 Stack pointer overflow on flag entry

3. Block call

FB M→STACK



4. Signals

FB 66 STACK→M
Flag stack to transfer flag area

1. Description

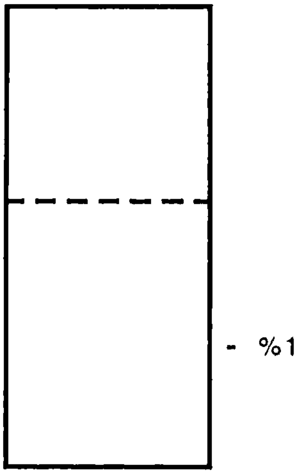
Function block FB 66 writes flag bytes FB 224- FB 255, which were saved in the flag stack by FB 65, back to their original locations, thus ensuring that the function block currently executing has the correct historical values at its disposal.
FB 66 can be called only in conjunction with FB 65.

2. Block data

Lib. no.	:
FBs to be loaded	: None
Type of FB call	: Unconditional or conditional
DBs to be input	: None
Error messages	: %1 Stack pointer underflow on flag removal

3. Block call

FB STACK → M



4. Signals

FB 190 K - LEITPC Signal interchange with master PLC

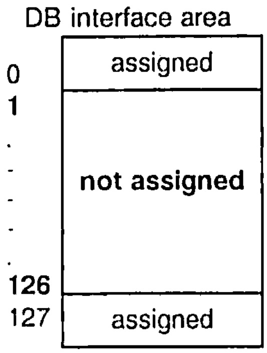
Note: Requires CPU version 6FX 1132-8BA01C or
6FX 1132-8BB01 D

1. Description

Function block FB190 transfers a specifiable number of words from a programmable data block to an interface block to which the master PLC has access.

The direction of signal flow (to the interface block or from the interface block to the 810 basic version 2 or 820 data area). Can be specified by the RI/A parameter.

The error message is written into a selectable output byte. .

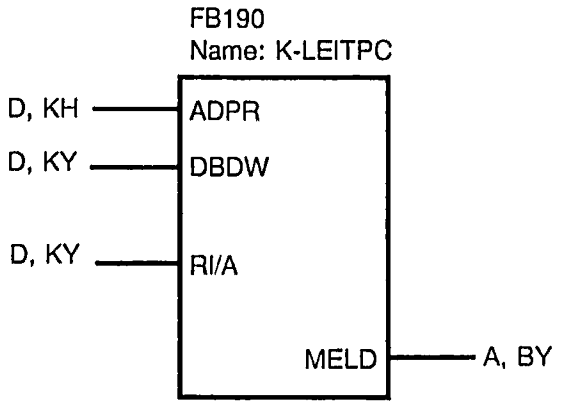


The user interface is a selectable data block on both the SINUMERIK and SIMATIC sides.

2. Block data

Lib. No.:
FBs to be loaded: None
DBs to be loaded: None
Type of FB call: Unconditional or conditional (JU FB190, JC FB190)
DBs to be input: None

3. Block call



4. Signals

ADPR — Starting address in the dualport RAM; is not evaluated (used only for compatibility reasons with FB 190 on the SIMATIC side).

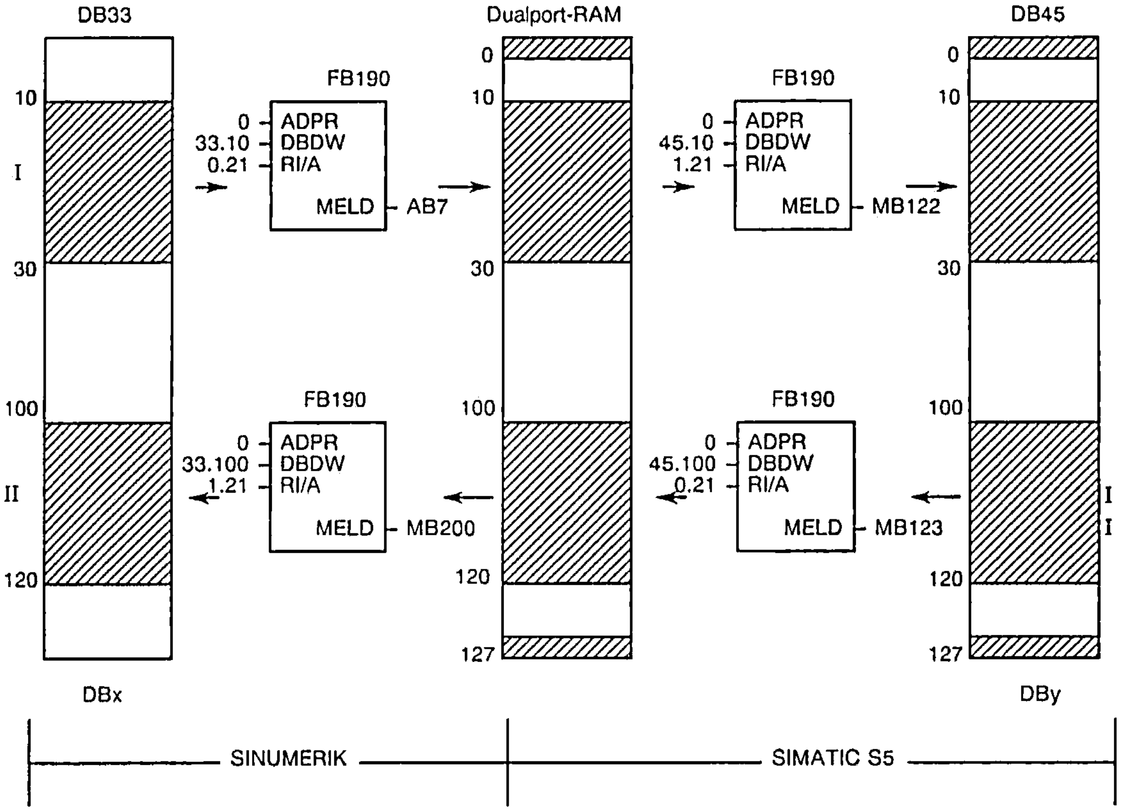
DBDW
High byte: Data block number
Low byte: No. of the first word and, at the same time, starting address in the interface area

RI/A
High byte: Transfer direction
0 ≙ Transfer to interface area
1 ≙ Transfer to data block
Low byte: Number of words to be transferred (< 127)

MELD
Error messages:

1	Interface fault
2	DB does not exist
3	DB too small
4	DB in EPROM
5	Illegal data word ($0 \leq DW \leq 127$)
6	Wrong direction
255	No error

5. Example



As can be seen from the diagram, an FB 190 standard function block on each side transfers data frame marked I from DB 33 on the SINUMERIK side to DB 45 on the SIMATIC side and data frame marked II in the opposite direction.

The desirerd frame length is defined by the RI/A parameter.

The location of the data frame in the interface area is determined by the initial words of the data block. DW 0 and DW 127 must **not** be used. The data frame can be positioned anywhere between these data words.

The user is responsible for writing data to the data blocks. He can fill data frame I of DB 33, using load and transfer statements and define which functions and messages are to be sent to and processed in the SIMATIC S5 side.

The SIMATIC S5 system replies via data frame II.

The internal PLC of the SINUMERIK system should never be replaced by the SIMATIC S5 PLC interfaced to it. In transfer lines, the S5 PLC performs the functions of a master PLC. On system power-up, for example, all SINUMERIKcontrols should be switched from the central control desk to "Reference point approach" mode; the "Automatic" program of the SINUMERIK controls can then be selected and started.

The diagram above illustrates the data flow between the function macro on the SINUMERIK side and the function block on the SIMATIC side via the interface area.

Important:

The number of data words to be transferred (both in the case of the FB190 function macro on the SINUMERIK side and function block FB 190 on the SIMATIC side) must be the same for both FB190 function blocks communicating with each other.

The number of the first data word on the SINUMERIK side must be identical to the number of the first data word on the SIMATIC side.

7 Transfer Parameters and Operating System Machine Data

7.1 Transfer Parameters

Specific bits are set in flag bytes FB0 to FB24 in the restart routine and during program scanning.

FB	*Bit:* 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Comments
0	Flashing frequency 1 Hz				Channel no.	Channel no.	One	Zero	Basic signal
1	OB no. of the current scanning level								
2						OB 2	OB 1		Initial state
3						OB 2	OB 1	OB 0	Cold restart
4									
5									
6						OB 2			Processing delay
7									
8								Fault group bit	Ready
	Negative edge								Alarm input byte
12	En.7	En.6	En.5	En.4	En.3	En.2	En.1	En.0	
	Positive edge								Alarm input byte
16	En.7	En.6	En.5	En.4	En.3	En.2	En.1	En.0	
20									NC Ready
21									
22									
23								V.24 running	
24	Act. of meas. probe 1	Act. of meas. probe 2			NC-BB2 1)	NC-BB1 2)	Battery fault	NC-alarm	Watchdog

1) NC alarm with shutdown
2) NC alarm with cancellation of readin enable
n = alarm input byte (defined via PLC MD 0)

Flashing frequency

The operating system provides a flashing signal with a frequency of 0.5 Hz. Its on-off ratio is 1:1.

Initial state

The initial state signals are set following a cold restart, and are reset when the cyclic user program or interrupt service routine has run a complete cycle.

Cold restart

Cold restart signals F3.1 to F3.7 signal initiation of a cold restart during the initial pass, i. e. they remain set until the relevant OB has executed in its entirety for the first time following a cold restart.
OB20's F3.0 cold restart signal is zero when OB20 is called in the warm restart branch. If an OB20 warm restart is interrupted by a power failure, OB 20 is re-executed in its entirety over a cold restart; F3.0 = 1 in this case.

Processing delay

If the interrupt service routine is reinvoked before it has terminated, the appropriate bit is set in FB6. In this case, the PLC stops and no entry is made in FB6. All bits are reset on a cold or warm restart.

Negative or positive edge

When the signal state of one of the input bytes selected for process interrupt servicing changes, the corresponding bit is set in flag byte FB12 on a positive edge (signal state change from 1 to 0) and in FB16 on a positive edge (signal state change from 0 to 1).

Monitoring

The relevant bits are shown in the following situations:

- Actuation of measuring probe 1 (F24.7)
- Actuation of measuring probe 2 (F24.6)
- NC alarm with shutdown (F24.5)
- Temperature error (F24.2)
- Battery failure (F24.1)
- NC alarm (F24.0)

7.2 PLC Machine Data SINUMERIK 810 basic version 2/820

PLC-VALUES

MD no.	Description	Default value	Max. value	Input resolution
0	No. for interr. input byte	7	31	---
1	Max. interpreter runtime (PLC cyclic)	15 %	20 %	1 %
2	Reserved for Sinumerik 810 basic version 1			
3	Max. interpreter runtime (PLC interr.-driven)	2000	2500	U5
4	Reserved for WD 510 P			
5	Cycle time monitoring	300	320	1 ms
6	No. of the last active MC 5 timer	15	31	---
7				
8[1)]	Interface for DB 37	1	2	---
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				

[1)] Not used with 810G/820G

7.3 PLC Machine Data Bits SINUMERIK 810 basic version 2/ 820

PLC - BITS

MD no.	7	6	5	4	3	2	1	0
2000	PLC MD 1007 BCD	PLC MD 1006 BCD	PLC MD 1005 BCD	PLC MD 1004 BCD	PLC MD 1003 BCD	PLC MD 1002 BCD	PLC MD 1001 BCD	PLC MD 1000 BCD
2001	H No. Channel 1/2 BCD	T No. Channel 1/2 BCD	S No. Channel 1/2 BCD	M No. Channel 1/2 BCD	Reserved 810 basic version 1	Reserved 810 basic version 1	Reserved 810 basic version 1 [1]	Reserved 810 basic version 1 [1]
2002	M decod. with expanded address	Interface Master PLC	Feedrate/ Rapid traverse	Interface distributed I/Os	Transfer machine control panel from process input to process output image	Reserved 810 basic version 1	2nd axis select switch available	No interrupt service routine (OB 2)
2003	Enable diagnostics DB 1	Fragmenting (STEP 5 program)	DEMO mode	Enable STEP 5 system commands		Stop Distributed I/Os faulted	Stop when OB 1 exceeds runtime	Stop when OB 2 exceeds runtime or on OB2 delay

Spindle override switch for S2 available	1st spindle override switch for S1 and S2 active

[1] 810G/820G only

8 Error Analysis

8.1 Types of Error

The PLC operating system can detect internal errors, user programming errors and invalid machine data.

Errors may occur in the restart routine or during cyclic scanning. Errors may result in a PLC stop, or the user receives two kinds of error message:

1.) Over an error display on the NC screen in the form of a PLC interrupt

2.) By the setting of a group bit in the basic signal flag area (F 8.0) and of a detailed error code bit in DB1. The user can then react to the error at the software level. The PLC operating system does not automatically reset these bits on completion of a cycle, the user must reset them himself as required.

DW	Bit No.							
	7	6	5	4	3	2	1	0
DL8					Master PLC failure		OB2 runtime exceeded	OB1 runtime exceeded
DR8					Modification of distributed I/Os	Temperature rise in expansion unit	Distrib. transfer faulted	No reaction from expansion unit

When no higher-priority interrupt is pending (NC, PLC), the PLC fault message, which comprises the alarm number and an accompanying text, is displayed on the uppermost screen line. The alarm can be acknowledged, and the fault message is then erased from the screen. When higher-priority alarms are pending, the user can request information about the fault by pressing the "PLC alarms" menu key.

Alarm number 3 and the text "PLC Stop" are always displayed on the uppermost screen line when the fault results in a PLC stop.

The three methods of detailed analysis while the PLC is in the STOP mode are discussed in Section 8.2 and 8.3.

8.2 Interrupt Stack

The programmer's "Output ISTACK" function can be invoked to help analyze errors. This function displays the control bits. Some PG software releases display a bit identifier in the otherwise unused display locations; this identifier is of no relevance for the PLC105W. On the other hand, identifiers marked with * (for unused) may have a special meaning for the PLC105W.

CONTROL BITS

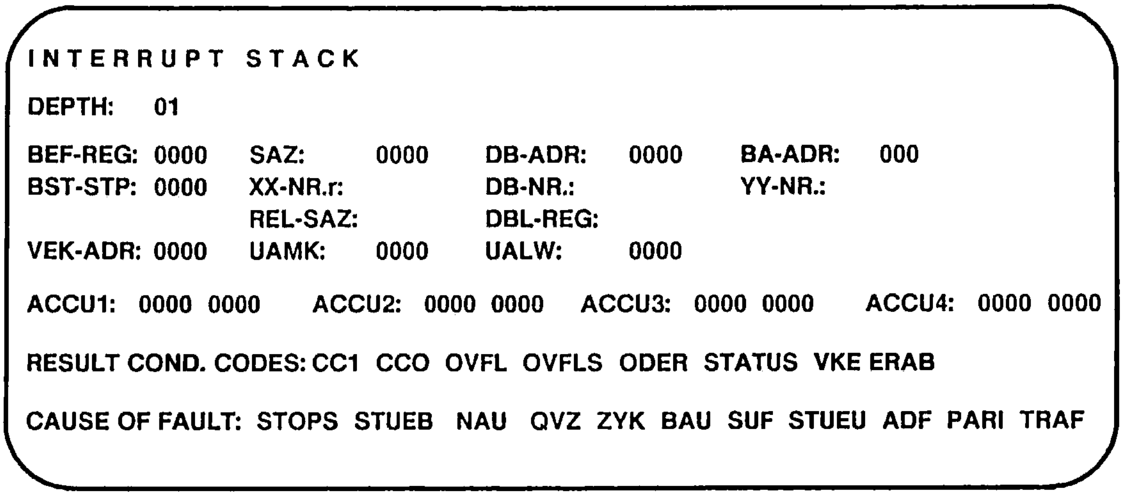
```
NB       NB       BSTSCH   SCHTAE    ADRBAU   SPABBR   NAUAS    NB
NB       NB       NNN*     PERUNKL*  NB       NB       NB       NB
STOZUS   STOANZ   NEUSTA   WIEDAN    BATPUF   NB       NB       NB
KEINPS*  UAFEHL   MAFEHL   EOVH      NB       NB       OBWIED   NB
NB       NB       KOPFNI   NB        WECKFE   PADRFE   ASPLUE   RAMADFE
EAADFE*  SYNFEH   NINEU    NIEWIED   RUFBST   QVZNIN   SUMF     URLAD
NB       NB       STS*     STP*      TBWFEH   LIR/TIR  NB       NB
NB       LPTP*    NB       NB        NB       NB       NB       NB
```

The control bits in detail:

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
PERUNKL	I/Os not ready (in expansion unit)
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
KEINPS	No user program memory or user data memory submodule
UAFEHL	STOP status 15 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
WECKFE	Group flag for user program runtime error (OB1, OB2)
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
QVZNIN	Timeout
SUMF	Sumcheck error
URLAD	Overall reset and subsequent bootstrap loading of user memory required
STS	Direct STOP initiated via STS
STP	Interrupt condition code following STP
TBWFEH	Timeout, block transfer operation
LIRTIR	Timeout, LIR or TIR operation
LPTP	Timeout, direct access to I/Os via user statements

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



```
INTERRUPT STACK

DEPTH:    01

BEF-REG: 0000   SAZ:      0000   DB-ADR:    0000    BA-ADR:   000
BST-STP: 0000   XX-NR.r:         DB-NR.:            YY-NR.:
                REL-SAZ:         DBL-REG:
VEK-ADR: 0000   UAMK:     0000   UALW:      0000

ACCU1:  0000 0000    ACCU2:  0000 0000   ACCU3:  0000 0000    ACCU4:  0000 0000

RESULT COND. CODES: CC1  CCO  OVFL  OVFLS  ODER  STATUS  VKE ERAB

CAUSE OF FAULT:  STOPS  STUEB   NAU   QVZ  ZYK  BAU  SUF  STUEU  ADF  PARI  TRAF
```

Bits ACCU3 and ACCU4 are irrelevant to the PLC, as it is not equipped with these accumulators. Like the control bits, the bits marked with "-" may show an irrelevant bit identifier. Refer to Section 9.2 for details on the result condition codes.

DEPTH	The latest interrupt event (DEPTH = 01) is always shown for the PLC
BEF-REG	Operation representation (hexadecimal)
SAZ	Step address counter The operation representation is ascertained by accessing user program memory via the step address counter (SAZ). Since the step address counter always points to the next operation to be executed, the operation that is interpreted is the one located at the next lower address. The information shown in the BEF-REG is thus incorrect when double-word operations are involved as well as immediately following jump commands.
DB-ADR	Block start address
BST-STP	Block stack pointer
XX-NR.	Current block whose execution was initiated by the interrupt
DB-NR.	Current data block
YY-NR.	Block that called the current block (XX)
REL-SAZ	Relative step address counter in the current block
DBL-REG	Register containing the data block length
VEK-ADR	Vector address for external storage
UAMK	Control register 1 (hexadecimal)*
UALW	Control register 2 (non-existent) * for details, see chapter 8.3.1
STOPS	Irrelevant
STUEB	Block stack overflow (max. 12 entries allowed)
NAU	Mains power failure (controller remains in the STOP mode until power is recovered)
QVZ	Timeout
ZYK	Cycle scan time exceeded
BAU	Battery failure
SUF	Substitution error
STUEU	Interrupt stack overflow (max. 2 levels possible)
ADF	Addressing error
PARI	Parity error
TRAF	Transfer error

8.3 Detailed Error Code

8.3.1 Display on programmer

Using the programmer's info function, the user can display additional information for interrupt analysis by entering pseudo address F000$_{hex}$. The following is displayed on the programmer screen:

```
ADR   VAL     ADR   WERT    ADR   WERT    ADR   WERT
F000  00xx    F001  61yy    F002  xxx1    F003  xxx2
F004  xxx3    F005  zob2    F006  0000    F007  0000
F008  0000    F009  0000    F00A  0000    F00B  0000
F00C  0000    F00D  0000    F00E  0000    F00F  0000
F010  0000    F011  0000    F012  0000    F013  0000
F014  0000    F015  0000    F016  0000    ....
```

Addresses F00C to F011 are described in detail in Section 9.1.2; subsequent displays are irrelevant.

The following applies to addresses F000 to F005:

xx = **Internal detailed error code (ERRCODE for PLC STOP)**

61yy = **NC alarm number 6100 - 6163**
xxx1 = **Auxiliary error info, word 1**
xxx2 = **Auxiliary error info, word 2**
xxx3 = **Auxiliary error info, word 3**
zob2 = **Event counter, processing time-out in OB 2**

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.

1. **PLC-QVZ:**

   The OP-code of the command that caused the timout errors is entered in word 1 (xxx1). xxx2/xxx3 contain the following:

- Timeout caused by L IR, T IR, T NB or T NW:
  xxx2 Offset address
  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) of the operation parameter

2. **Parameter initialization errors**

   xxx3 = Number (BCD) of the invalid operation parameter (peripheral byte no., segment no., block no., timer/counter no.)

Control register UAMK (binary):

Data bit																Function
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
															DB 0	Trigger watchdog - This bit is stored in the register Watchdog is not triggered
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	Retrigger watchdog
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	
														D1		Software clear watchdog (CLR-WD)
X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	X	Reset watchdog (watchdog disabled) - *RDYIN inactive, LED on
X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	Enable watchdog
													D2			Ready internal disable (RDYIN) - No effect on error LED RDYIN circuit dependent on hardware watchdog RDYIN circuit inactive (High)
X	X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	
												D3				Enable system NMI (AUX 0)
X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	NMI-WF effective for own CPU only
X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	NMI-WF signalled to other CPUs as system NMI (AUX 0)
											DB4					Not used
X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	
										DB5						System initialization (CSINI)
X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X	Initialization mode passive (*CSINI = HIGH)
X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	Initialization mode active (*CSINI = LOW)
									DB6							Reset request (RESREQ)
X	X	X	X	X	X	X	X	X	0	X	X	X	X	X	X	No reset request (*RESREQ = LOW)
X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	Reset request (*RESREQ = LOW)
								DB7								MPC mode
X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	MPC byte mode
							DB8									MPC PERI (peripheral error)
X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	PERI input = 0
X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	PERI input = 1
						D9										MPC device number
X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	Device number = EH
X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	Device number = FH
					D10											MPC flag 0
X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	Flag = 0 Command output disable in EU
X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	Flag is controlled by *OUTDS
				D11												MPC flag 1
X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	FLI = 0
X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	FLI = 1 MPC RESET in EU
D15	14	13	12													MPC data length Control DIL3-6

The "Event counter, timeout" shows how often the interrupt service routine (called over OB2) was requested before it could terminate its original run.

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

Internal detailed error code xx differs from NC alarm number 61yy as follows:

xx	- Hexadecimal value between 100 and 163 (64_{hex} - 13_{hex}) - Value ≠ 00 always results in a PLC STOP
61yy	- BCD value between 6100 and 6163 (on NC screen also) - A value ≠ 0000 does not always result in a PLC STOP

The table below provides an overview of the error code allocated to an error event for both the xx and 61yy identifiers:

Error identifier / Error event	XX	61yy
Error detected during scanning of the STEP 5 program	100-115 (64_{hex}-73_{hex})	6100-6115
Error detected during startup	116-143 (74_{hex}-$8F_{hex}$)	6116 -6143
Error detected during scanning of the cyclic program	144-149 (90_{hex}-95_{hex})	6144-6149
Error messages output by the interrupt service routine	150-163 (96_{hex}-$A3_{hex}$)	6150-6163

8.3.2 Display on NC screen

When the PLC stops, alarm number 3 and the text "PLC Stop" are always displayed as highest-priority PLC alarm. The display appears on the alarm line unless a higher-priority alarm is pending (e. g. EMERGENCY SHUTDOWN). In addition to the group fault message, detailed information on the event that caused the fault or error can be obtained by pressing menu keys "Diagnostics" and "PLC alarm". An alarm number between 6100 and 6163 and an accompanying text inform the user of the reason for the nPLC STOP. The display may also include user alarms, which are displayed in the order in which they occur. The operating system interrupt normally appears last. Both alarm 3 and the detailed error interrupt can be delected only over POWER - ON - RESET or by entering the PLC start command from a programmer, but then only when the cause of error has been rectified. A list of operating system interrupts is presented on the next page.

8.4 Alarm List

SYSTEM 810 BASIC VERSION 2/820 ALARM LIST
PLC OPERATING SYSTEM AND USER ALARMS

Alarm no.	Description	Related to block/ sequence	Disabling of:						Called by	
						Machining	NC Start	NC-Ready 2	PLC	PLC-response
6000	User alarms									Message
	"									Message
6063	User alarms									Message
6100	No process interface module									STOP
6101	Invalid MC5 code									STOP
6102	Invalid MC5 parameter									STOP
6103	Transfer to non-existent DB									STOP
6104	Substitution error									STOP
6105	No MC 5 block									STOP
6106	No DB									STOP
6107	Invalid segment LIR/TIR									STOP
6108	Invalid segment Block transfer									STOP
6109	Block stack overflow									STOP
6110	Interrupt stack overflow									STOP
6111	MC5 command STS									STOP
6112	MC5 command STP									STOP
6113	Invalid MC5 timer/counter									STOP
6114	Function macro									STOP
6115	System operations disabled									STOP
6116	MD 0000: Alarm byte no.									STOP
6117	MD 0001: CPU load									STOP
6118	MD 0003: Alarm rout. runtime									STOP
6119	MD 0005: Cycle scan time									STOP
6120										
6121	MD 0006: Last MD5 timer									STOP

SYSTEM 810 BASIC VERSION 2/820 ALARM LIST
PLC OPERATING SYSTEM AND USER ALARMS

Alarm no.	Description	Related to block/sequence	Disabling of:						Called by	
						Machining	NC Start	NC-Ready 2	PLC	PLC response
6122	Wrong jumper setting									Message
6123	Invalid sampling time									STOP
6124	Gap in MC5 memory									STOP
6125	Multiple input definitions									STOP
6126	Multiple output definitions									STOP
6127	No alarm byte									STOP
6128	Incorrect I/O jumpering									STOP
6129										
6130	Sync. error, basic prog.									STOP
6131	Sync. error, MC5 prog.									STOP
6132	Sync. error, MC5 data									STOP
6133	Invalid block, basic prog.									STOP
6134	Invalid block, MC5 prog.									STOP
6135	Invalid block, MC5 data									STOP
6136	Sumcheck error in MC5 block									STOP
6137	Sumcheck error in basic prog.									STOP
6138	No reaction from expansion unit									*
6139	EU transfer error									*
6140										
6141										
6142										
6143	No decoder DB									STOP
6144	Decoder DB not modulo 6									STOP
6145	Wrong number of decoder units									STOP
6146	Decoder DB too short									STOP
6147	Mod. in global I/Os									*

* PLC response depends on MD definition (STOP or message only)

SYSTEM 810 BASIC VERSION 2/820 ALARM LIST
PLC OPERATING SYSTEM AND USER ALARMS

Alarm no.	Description	Related to block/ sequence	Disabling of:						Called by	
						Machining	NC Start	NC Ready 2	PLC	PLC-response
6148	Temperature rise in expansion unit									Message
6149	STOP over PG softkey									STOP
6150	Timeout: User memory									STOP
6151	Timeout: Link memory									STOP
6152	Timeout: LIR / TIR									STOP
6153	Timeout: TNB / TNW									STOP
6154	Timeout: LPB / LPW / TPB / TPW									STOP
6155	Timeout: Substitution command									STOP
6156	Timeout cannot be interpreted									STOP
6157	Timeout: JU FB/JC FB									STOP
6158	Timeout on I/O transfer									STOP
6159	STEP 5 prog. runtime exceeded									*
6160	OB2 runtime exceeded									*
6161	Cycle scan time exceeded									STOP
6162	Delay OB2									*
6163	Master PLC failure									Message
6164										
7000	PLC user status messages									Message
	"									Message
7063	"									Message

* PLC response depends on MD definition (STOP or message only)

9 STEP 5 command set with programming examples

9.1 Memory organization

In order to be able to reach address areas outside the normal range for STEP 5 commands (e.g. process image) with the "Address info" function of a programmer (such as PG675, PG685), the so-called "segment switch" is needed. This is an auxiliary variable with which a particular segment within the 1 Megabyte address area of the processor can be reached.

Each address then comprises the offset address (position in the segment) and the segment address (depending on the position of the segment switch).

The following segments can be selected by the user via a segment number either with the "Address info" diagnostic function of a programmer or with special STEP 5 system commands (c.f. Section 9.4.13):

Segment number	Designation	Segment address	Offset address (word-oriented)	Remarks
1	PSEG	1000 $_{hex}$	0000 $_{hex}$-15FF $_{hex}$	Read and write
2	NCPERSEG	1000 $_{hex}$	0000 $_{hex}$-3000 $_{hex}$	Read and write from 2000 h, else probably PLC-Timeout along with PLC-Stop status
3	BSEG		0000 $_{hex}$-801 $_{hex}$	Read and write
4	VSKSEG		0000 $_{hex}$-230 $_{hex}$	Read and write
5	AWPSEG	3000 $_{hex}$	0000 $_{hex}$-17FF $_{hex}$	Read and write
6	AWPSEG	3300 $_{hex}$	0000 $_{hex}$-C9F $_{hex}$	Read and write
7	SY1SEG		0000 $_{hex}$-7FFF $_{hex}$	Read
8	SY2SEG		0000 $_{hex}$-2000 $_{hex}$	Read
			no Program information function	Read
9	PPWSEG		no Program information function	Read

If another value is defaulted for the segment switch or segment number, it is either ignored (with the programmer Info function) or causes an error message (with STEP 5 system commands LIR, TIR, TNB, TNW).

On cold restart the AWPSEG user program segment is automatically selected with position 5 of the segment switch by the PLC operating system.

Memory organization:

Segment number	Segment contents		Notes
1	PLC system data:	Process images, block lists, memory administration, BSTACK, ISTACK etc. (Σ)	*
2	Hardware components:	I/O interface module, processor register, IPC link memory for global I/Os (Σ)	
3	Internal NC-PLC-Interface:	axis-specific signals, channel-specific signals NC-control panel signals etc.	*
4	Spindle Interface	Spindle-specific signals	*
5	User data memory:	Max. 12 k bytes for Step 5 user program (OB, PB, SB, FB)	* *
6	User data memory:	Max. 6 k bytes for Step 5 user data blocks and 0.3 k bytes for data blocks initialized by the PLC operating system on startup	* *
7	PLC-systemprogram	Programming and testing function, interpreter etc. (Σ 50 kB)	* * *
8	Integral function macros	Fb 11, FB 60 etc. (Σ 16 kB)	* * *
9	Part program memory		
A	Dedicated to WF-option data		

* Internal RAM area is stored on the CPU module
* * Buffered RAM area is stored on machine data card
* * * EPROM area is stored on a separate memory module (Module 2)

9.1.1 Changing the segment switch

The following sequence applies to the PG675 programmer:

- Call the information functions with the F7 key
- Call any memory areas with the F8 key
- Enter the pseudo-address E000$_{hex}$
- Press the Enter key and then
- Press the Abort key immediately

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

- Press the correction key
- Enter the new segment address

9.1.2 Block lists

The start addresses of the block lists can be output by entering pseudo-address F000hex (see Section 8.2 for the significance of addresses F000 to F009):

F00C	Start address of the OB block list
F00D	Start address of the FB block list
F00E	Start address of the DB block list
F010	Start address of the SB block list
F011	Start address of the PB 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.

Exception: The entries of the DB list are byte-oriented; these must be divided by two after the high/ low swap.

If the blocks are output via the directory function of the programmer, blocks which are not located in the user program segment appear with an offset address increased by 8000$_{hex}$ (applies to data blocks and resident function macros).

9.2 General notes

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

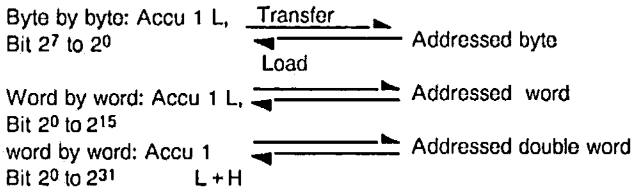
The basic operations are intended for the execution of simple binary functions. As a rule, they can be input/output in the three methods of representation (LAD,CSF and STL) of the STEP 5 language on the programmer.

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.

9.2.1 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:

I) Fixed-point number: +0 to .. 32 767
−1 to .. 32 768
This is located in Accu L and is interpreted as a 16 bit binary number (most significant bit=sign bit).
Negative number: two's-complement representation.

II) Fixed point double-word: + 0 to .. 2147 483 647
− 1 to .. 2 147 483 648
This is located in Accu L and is interpreted as a 16 bit binary number (most significant bit=sign bit).
Negative number: two's-complement representation.

III) Floating-point number: m x 2^{exp}
m = Mantissa ±0.1 701 412 x 10^{39}
exp = Exponent ±0.1 469 368 x 10^{-38}

This is represented as follows in the accumulator (exception: see "Arithmetic operations"):

Bit significance : $2^{31} \dots 2^{24}$ ¦ $2^{23} \dots 2^{0}$
Sign exp ¦ Sign m

The exponent is an 8-bit binary number in two's complement representation:
$-128 \leq exp < 127$

The mantissa is 24 bits wide and normalized::
$0.5 \leq$ positive mantissa < 1;
$-1 <$ negative mantissa ≤ 0.5

IV) BCD word-coded number with sign +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

V) BCD double word-coded number with sign +7 digits:
Assignments in the accumulator

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.

9.2.2 Condition codes of the 135W PLC

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 135 PLC is as follows:

2^7 CC for word operations				CC for bit operations 2^0			
CC 1	CC 0	OV	OS	OR	___	RLO	$\overline{\text{ERAB}}$

Condition codes for bit operations:

$\overline{\text{ERAB}}$: $\overline{\text{ERAB}}$ signifies first interrogation. This is the start of a logic operation. ERAB is set at the end of a logic operation sequence (memory operations).

RLO: Result of logic operation; result of bit-wide 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; the over-bit is stored; in the course of two or more arithmetic operations, this serves to indicate whether an overflow error (OVER) has occurred at some time.

CC1, CC0 are condition codes whose interpretation can be found in the following table.

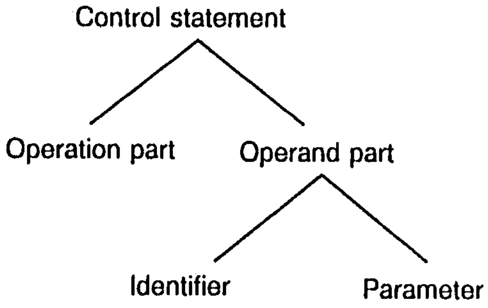
Condition codes for word operations			Fixed-point calculation Result	Boolean result	Comparison Contents of Acc 1 + Accu 2	Shift shifted bit	Floating-point calculation Result
CC 1	CC 2	OVER					
0	0	0	Result = 0	= 0	Accu 2 = Accu	0	Mantissa = 0; Exp. allowed
0	1	0	Result < 0	-	Accu 2 < Accu	-	Mantissa < 0; Exp. allowed
1	0	0	Result > 0	= 0	Accu 2 > Accu	1	Mantissa > 0; Exp. allowed
0	0	1	"Over-Null"*)	-	-	-	Mantissa = 0; Exp. allowed - 128
0	1	1	0 from pos.range	-	-	-	Mantissa < 0; Exp. allowed + 127
1	0	1	0 from neg.range	-	-	-	Mantissa > 0; Exp. allowed + 127
1	1	1	Division by zero	-	-	-	Division by zero

*) Special case: Greatest negative number added to itself

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

9.2.3 STEP 5 command representation

A statement is structured as follows:



The operation part describes the function to be executed. It defines what the processor is to do.

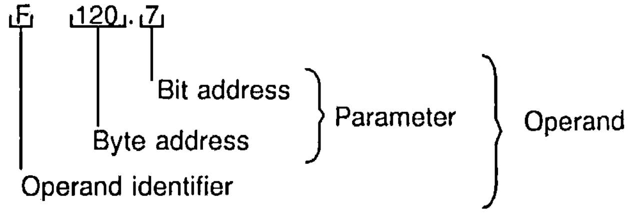
The *operand part* contains the information necessary for the operation to be executed. It specifies what the processor is to do something with. The Step 5 programming language has the following operand areas:

Inputs I	These constitute the interface from the process to the programmable controller (process image),
Outputs Q	These constitute the interface from the programmable controller to the process (process image),
Flags F	These are for storing binary intermediate results,
Data D	These are for storing digital intermediate results,
Timers T	These implement timer functions,
Counters C	These implement counting functions,
Peripherals P, Q	The process I/Os (input/output modules) are addressed directly,
Constants K	Represent a fixed predefined number,
Blocks OB,PB,FB,DB	Serve to structure the program.

The designation of the operand areas is the *(Operand) identifier.* The parameter must be specified to address a certain operand in an operand area.

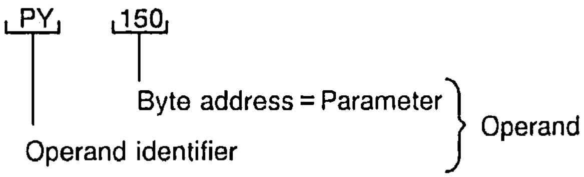
The *parameter* specifies the address of an operand. The operand areas inputs I, outputs Q and flags F are addressed byte-by-byte, i.e. the specified number refers to a specific byte of these operand areas (byte address). These operand areas can also be addressed bit-by-bit. The bit address is separated from the byte address with a fullstop.

Example: I 10 . 3
Q 7 . 0



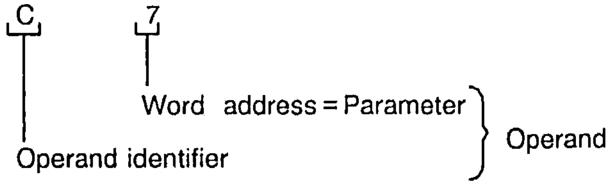
The peripherals P operand area is also addressed byte-by-byte but has no bit address.

Example:



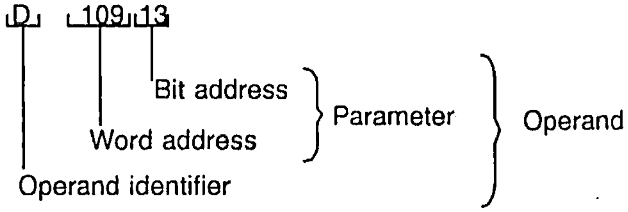
The timers T and counters C operand areas are addressed word-by-word (word address). These operand areas have no bit address.

Example: T 15



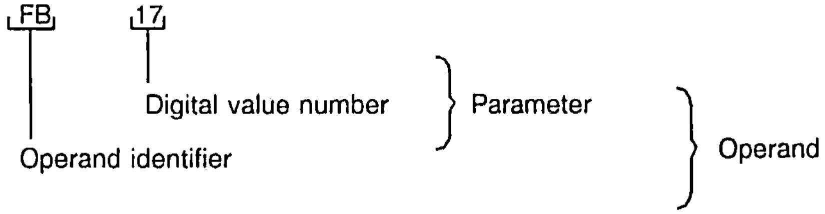
The data D operand area can be addressed word-by-word and bit-by-bit .

Example: DW 127



The parameter of the constant K operand area represents the number which is to be processed as a digital value. The parameters of the OB, PB, FB and DB operand areas specify the number of the operands in this area.

Example: K 1047



9.3 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).

Exceptions:

1. Load, transfer and code operations. These 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.

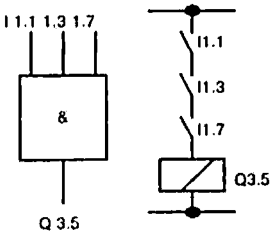
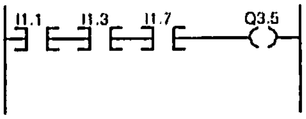
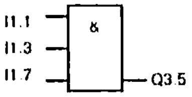
9.3.1 Logic operations, binary

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	1 to 127	Scanning a timer for logic 1
N T	1 to 127	Scanning a timer for logic 0
C	1 to 127	Scanning a counter for contents >0
N C	1 to 127	Scanning a counter for contents = 0

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

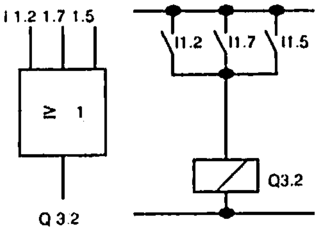
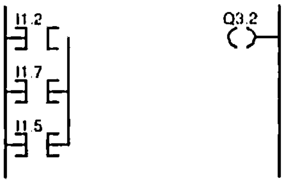
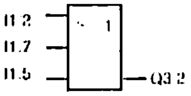
At the beginning of a logic sequence, the result depends only on the type of operation (A ≙ AND, AN ≙ AND NOT, O ≙ OR, ON ≙ OR NOT) and the queried logic level. Within a logic sequence, the RLO is formed from the type of operation, previous RLO and scanned signal state. A logic sequence is terminated 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		

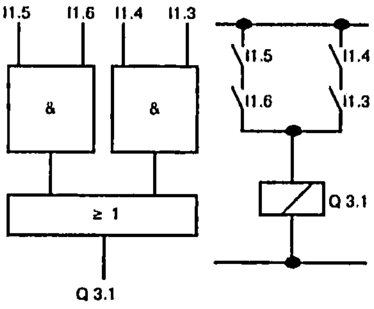
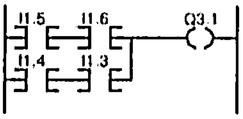
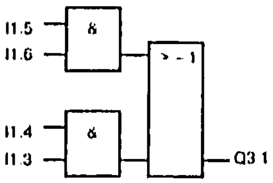
A logic 1 appears at output Q 3.5 if all inputs are simultaneously at logic 1. A logic 0 appears at output Q 3.5 if at least one of the inputs is at logic 0.
The number of scans and the order of programming are arbitrary.

OR operation

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I 1.2 1.7 1.5 ≥ 1 Q 3.2 I1.2 I1.7 I1.5 Q3.2	O I 1.2 O I 1.7 O I 1.5 = Q 3.2		

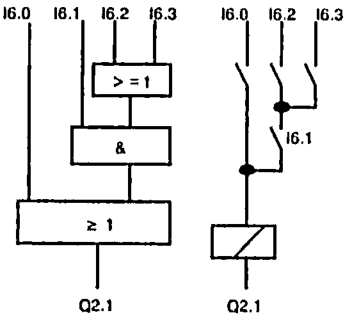
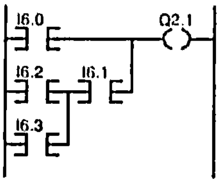
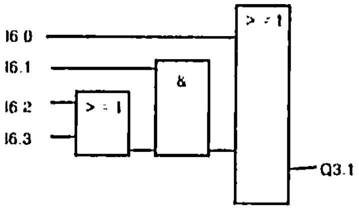
A logic 1 appears at output Q 3.2 if at least one of the inputs is at logic 1. A logic 0 appears at output Q 3.2 if all inputs are simultaneously at logic 0.
The number of scans and the order of programming are arbitrary.

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 I1.6 I1.4 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		

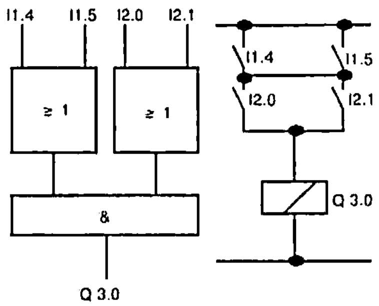
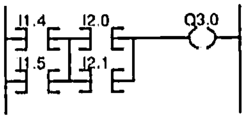
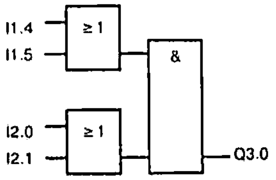
A logic 1 appears at output Q 3.1 if at least one AND condition is fulfilled. A logic 0 appears at output Q 3.1 if no AND condition is fulfilled.

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		

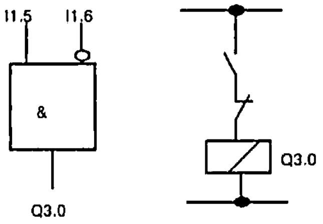
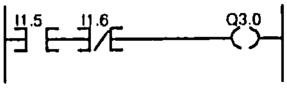
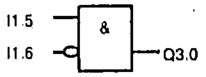
A logic 1 appears at output Q 2.1 if input I 6.0 or input I 6.1 and one of inputs I 6.2 and I 6.3 at logic 1.

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.

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I1.4 I1.5 I2.0 I2.1 ≥ 1 ≥ 1 & Q 3.0 I1.4 I1.5 I2.0 I2.1 Q 3.0	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

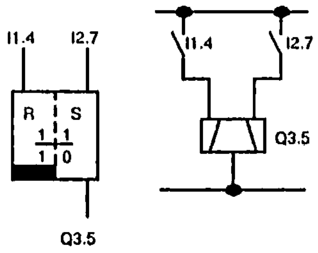
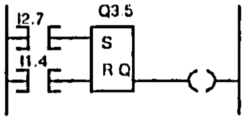
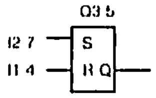
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 1,5 AN I 1.6 = Q 3.0		

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 opened).

9.3.2 Storage operations

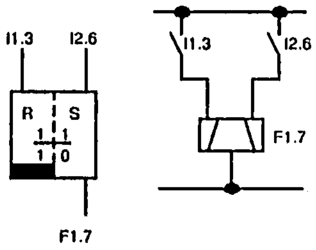
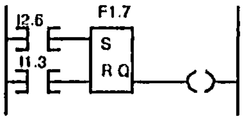
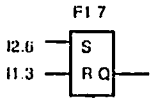
Operation	Parameter	Function
S □		Set
R □		Reset
= □		Assign
↑		
I	0.0 to 127.7	an input
Q	0.0 to 127.7	an output
F	0.0 to 255.7	a flag
D	0.0 to 255.15	a data word

RS Flipflop for latching signal output

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I1.4 I2.7 R S 1 1 1 0 Q3.5 I1.4 I2.7 Q3.5	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.

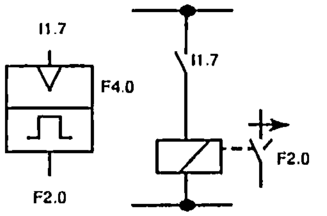
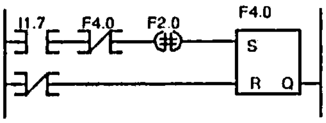
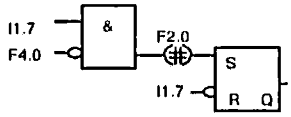
RS Flipflop with flags

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I1.3 I2.6 R S 1 1 1 0 F1.7 I1.3 I2.6 F1.7	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 I1.3 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

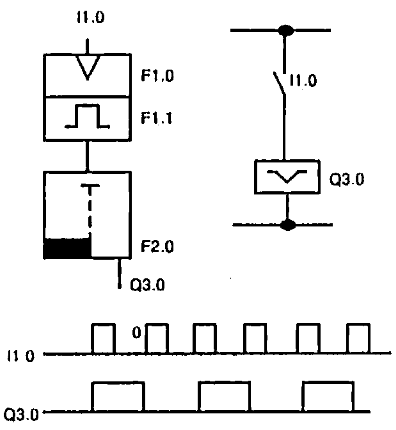
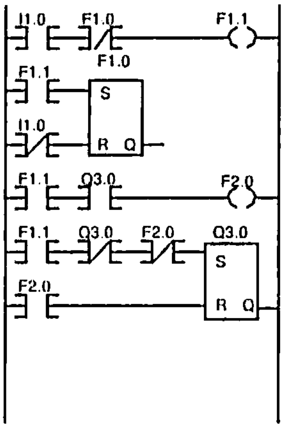
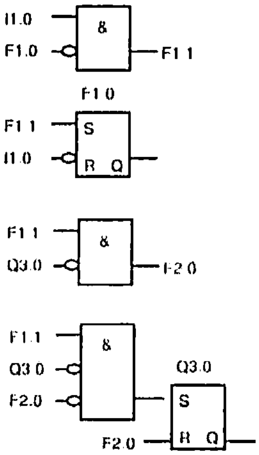
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 1.7 AN F 4.0 = F 2.0 A F 2.0 S F 4.0 AN I 1.7 R F 4.0		

The AND condition (AI 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 AI 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 I1.0 Q3.0 F2.0 Q3.0 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 F1.0 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	 I1.0 F1.0 & F1 1 F1 0 F1 1 I1 0 S 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.

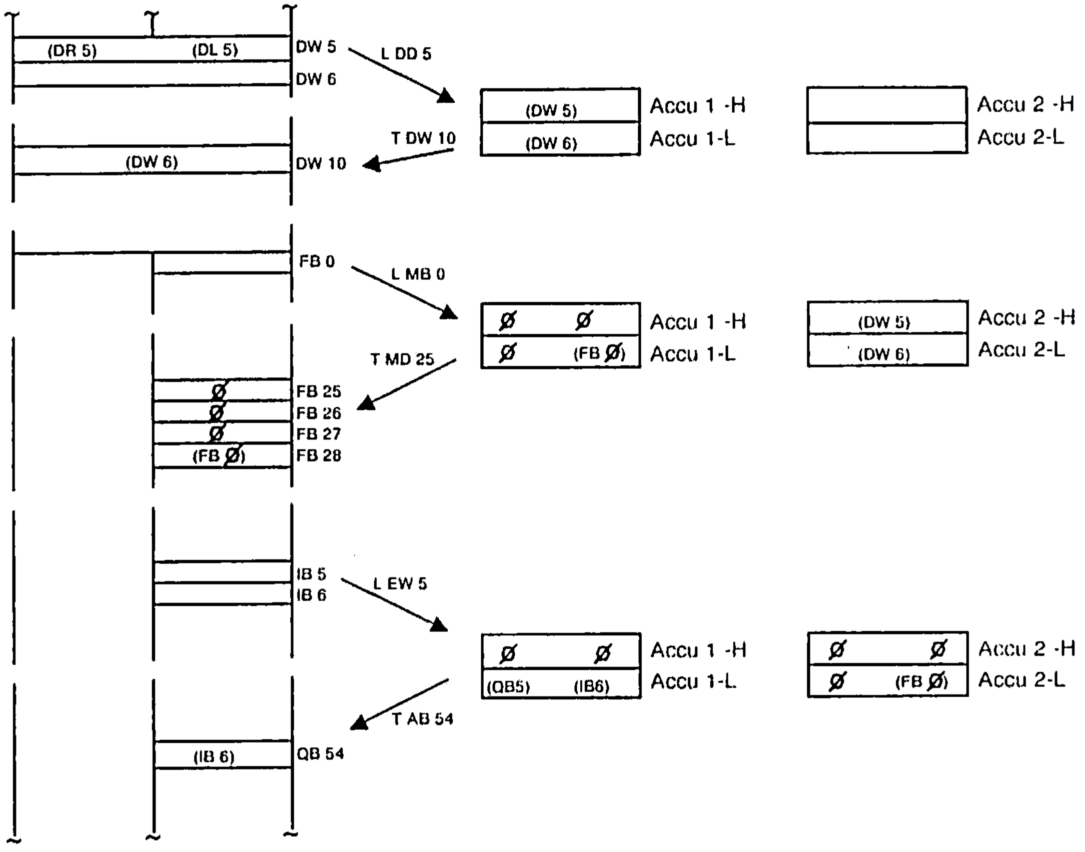
9.3.3 Load and transfer operations

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
I D	0 tos 124	an input double word from the PII
Q B	0 to 127	an output byte from the PIO [3]
Q W	0 to 126	an output word from the PIO
Q D	0 to 124	an output double word from the PIO
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 [2]		0 to 127 a time (binary)
C [2]		0 to 127 a count (binary)
P B	0 to 255	an I/O byte of the digital inputs/outputs
P W [4]	0 to 126	an I/O word of the digital inputs/outputs
K M [1]	16-bit pattern	a constant as bit pattern
K H [1]	0 to $FFFF_{hex}$	a constant in hex code
K F	0 to $+(2^{16}-1)$	a constant as fixed point number
K Y [1]	0 to 255 for each byte	a constant, 2 bytes
K Y [1]	0 to 255	a constant, 1 byte
K S [1]	2 alpha characters	a constant, 2 ASCII characters
K G [1]	$\pm 0.1469368 \times 10^{-38}$ to $\pm 0.1701412 \times 10^{+39}$	a constant as floating point number
K T	0.0 to 999.3	a time (constant)
K C [1]	0 to 999	a count (constant)

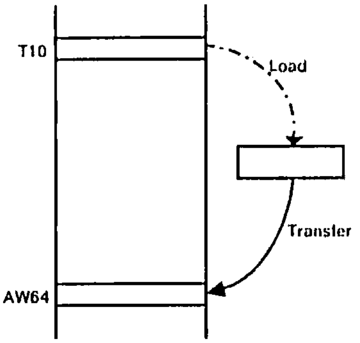
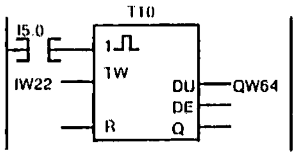
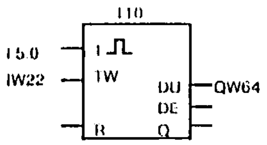
[1] Not for transfers
[2] PII Process input image
[3] PIO Process output image
[4] Only even parameters are allowed; error NNP is signalled for odd parameters.

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.

Example: Load and transfer function



Loading and transferring a time, also timing and counting operations, Page ....).

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		

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.

9.3.4 Timing and counting operations

Timers T 0-T 16 and counters C0-C31 are enabled as standard. If a larger parameter is programmed the interpreter outputs the error message "illegal MC5 type timer/counter".
The number of counters can be increased to a maximum of 31 via PLC MD6. For this purpose MD6 must be used to specify the number of the last "active" MC5 timer (i.e. the one taken into acount by the operating system). This number is efffective from the next cold restart.

The way in which the operating system updates time cells means that programming of the 0.01 time grid is not expedient (e.g. LKT 10.0).
As timer cell updating is also dependent on the "servo-scanning rate", i.e. is dependent on NC MD 155, *timer inaccuracies occur* which are shown in the table below.

Programmed time grid			
MD 155	100 ms	1 s	10 s
0	0%	0%	0%
1	+10%	-1%	+0.1%
2	+20%	-4%	+0.4%
3	+30%	+4%	+0.1%

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

[1)] Timing or counting operations do not change the contents of Accu 1.

	Operation	Parameter	Function
Time operation	S P	T	1 to 127 Start a timer as a pulse
	S E T	1 to 127	Start a timer as an extended pulse
	S D T	1 to 127	Start a timer as an ON-delay
	S F T	1 to 127	Start a timer as a latching ON-delay
	S F T	1 to 127	Start a timer as an OFF-delay
	R T	1 to 127	Reset a timer
Count operation	S C	1 to 127	Set a counter
	R C	1 to 127	Reset a counter
	C U C	1 to 127	Up counting
	C D C	1 to 127	Down counting

[1] Timing or counting operations do not change the contents of Accu 1.

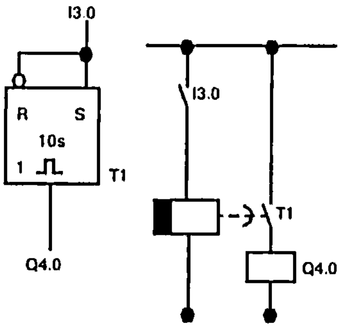
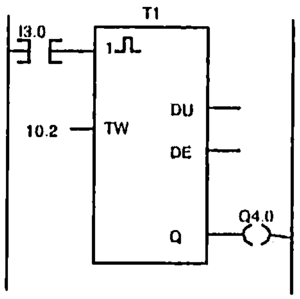
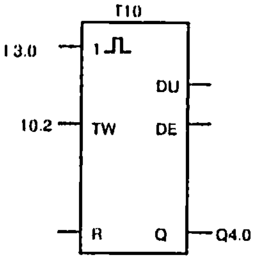
Important note:

Since the timer and counter commands are supported by the COP, 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.

Example:

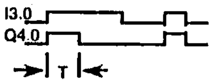
As a result of a DO FW command, command SD T0 (substituted SD T 33) is to be processed, With RLO = 0: Bits ZWG, ZKS, FMS with T 1 are deleted.

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		 T1 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 AT and OT result in a logic 1 as long as the timer is still running.

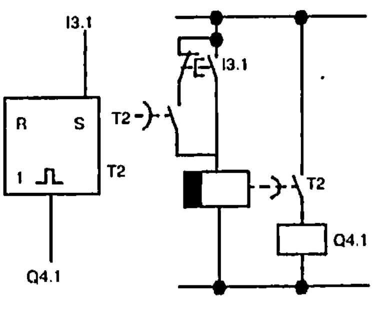
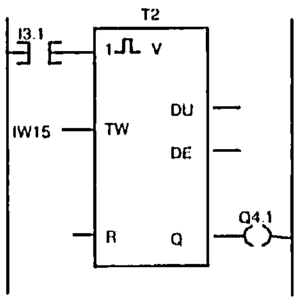
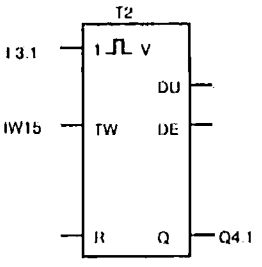
DU and DE are digital outputs of the timer. The time is present with the timebase, binary-coded at output DU and BCD-coded at output DE.



KT10.2:
The specified value (10) is loaded in the timer. The number to the right of the point specifies the timebase:

1 = 0.1 s
2 = 1 s
3 = 10 s

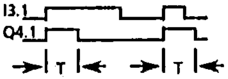
Extended pulse

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		

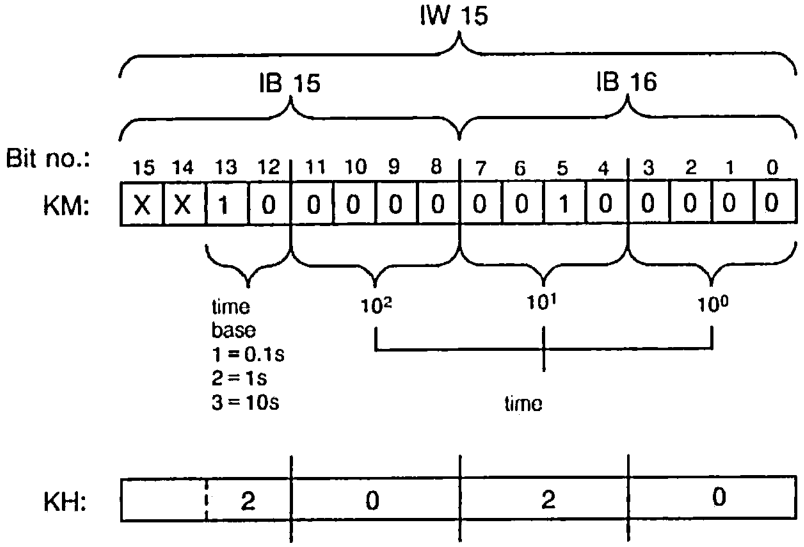
With first execution, the timer is started if the result of the logic operation is 1.

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:
The timer is loaded with the value of operands I, Q, F or D present in BCD code (example IW15). The time reference is determined by bit nos. 12 and 13.

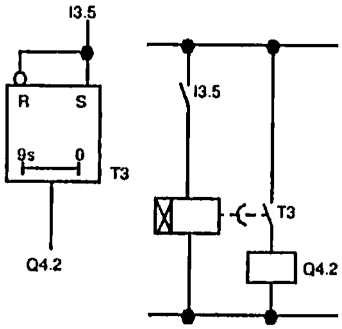
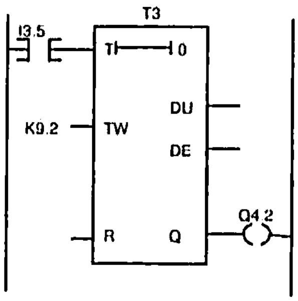
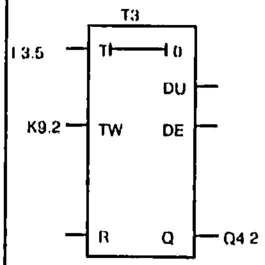


Bit no.:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KM:	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0

KH:		2	0	2	0

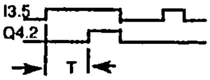
The programmed delay is 2x10x1.0s = 20s
X = any value (0 or 1)

ON-delay

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I3.5 R S 9s 0 T3 Q4.2 I3.5 T3 Q4.2	A I 3.5 L KT 9.2 SD T 3 A T 3 = Q 4.2	 T3 I3.5 T 0 K9.2 TW DU DE R Q Q4.2	 T3 I 3.5 T 0 DU K9.2 TW DE R Q Q4 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).

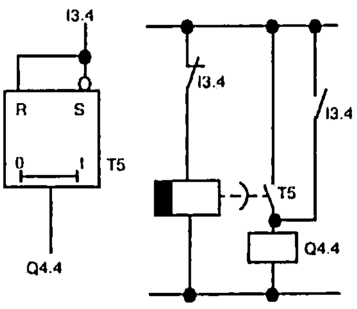
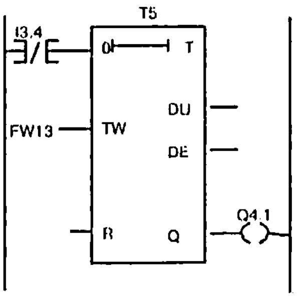
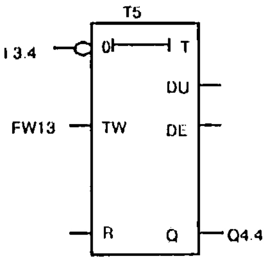
Scans AT or OT result in a logic 1 if the time has elapsed and the result of the logic operation is still present at the input.



KT 9.2:
The specified value (9) is loaded in the timer. The number to the right of the point specifies the timebase:

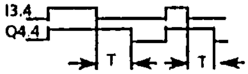
1 = 0.1 s
2 = 1 s
3 = 10 s

OFF-delay

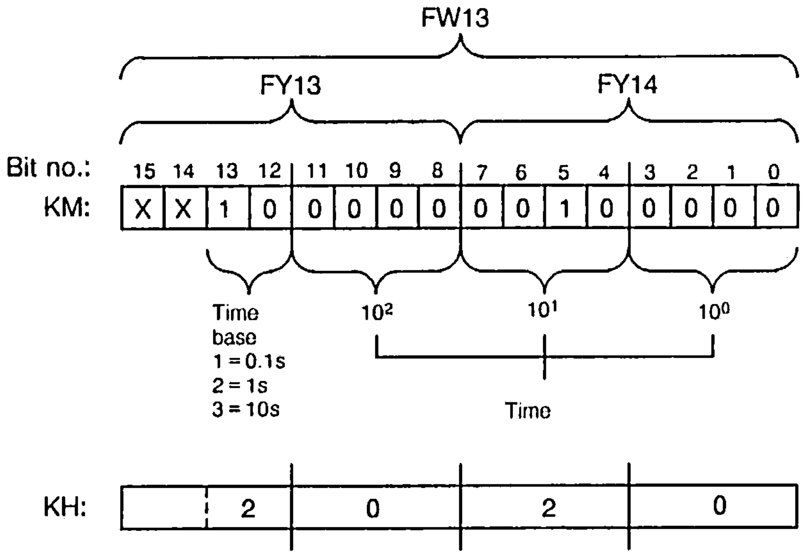
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
 I3.4 R S 0 1 T5 Q4.4 I3.4 I3.4 T5 Q4.4	AN I 3.4 L FW13 SF T 5 A T 5 = Q 4.4	 T5 I3.4 0 1 T FW13 TW DU DE R Q Q4.1	 T5 I 3.4 0 1 T DU FW13 TW DE R Q Q4.4

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). The timebase is determined by bit 12 and bit 13.

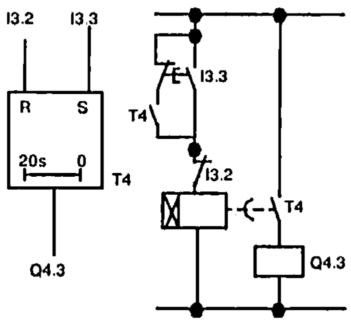
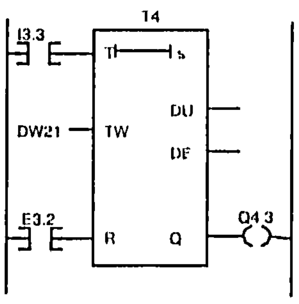
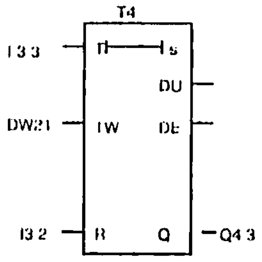


Bit no.:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KM:	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0

KH:	2	0	2	0

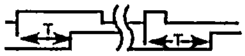
The programmed time equals 2x10x1.0s =20s.
X = optional (0 or 1)

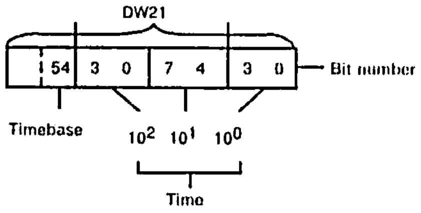
Latching ON-delay

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 TI S DW21 TW DU DE E3.2 R Q Q4.3	 T4 I3.3 TI 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 AT and OT result in a logic 1 if the time has elapsed.

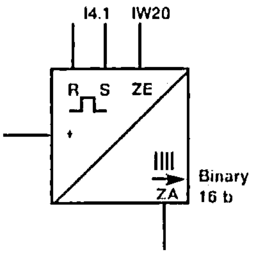
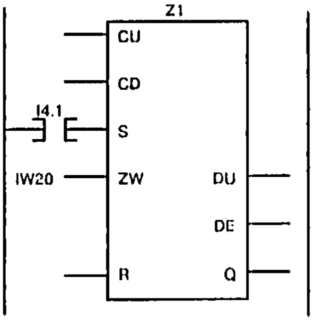
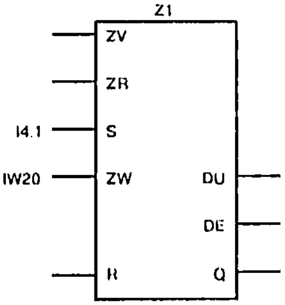
The logic level only goes to 0 when the timer has been reset with function RT.





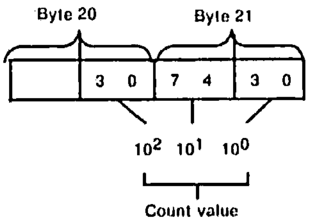
DW 21:
Setting the time with the value of operands I, Q, F or D present in BCD code (data word 21 in the example).

Setting a counter

<table>
<tr><th rowspan="2">Given circuit</th><th colspan="3">STEP 5 representation</th></tr>
<tr><th>Statement list</th><th>Ladder diagram</th><th>Control system flowchart</th></tr>
<tr>
<td>
</td>
<td>A I 4.1
L IW20
S C 1</td>
<td>
</td>
<td>
</td>
</tr>
</table>

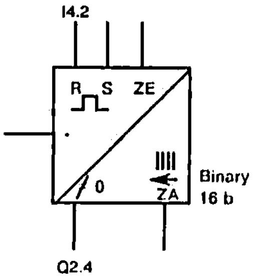
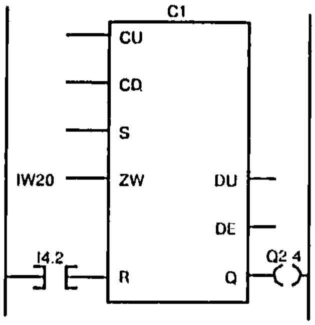
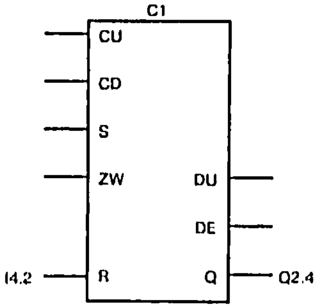
With the first execution, the counter is set if the result of the logic operation is 1. If execution is repeated, the counter is unchanged (irrespectively of whether the RLO is 1 or 0). With repeated first execution with RLO = 1, the counter is set again (signal edge decoding). DU and DE are digital 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.



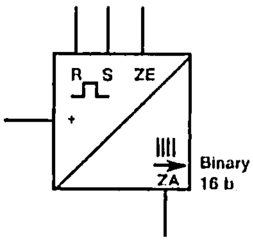
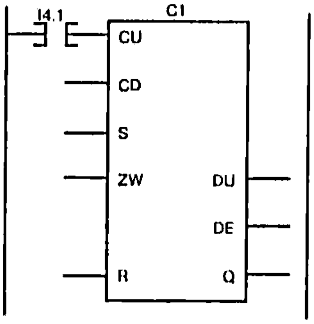
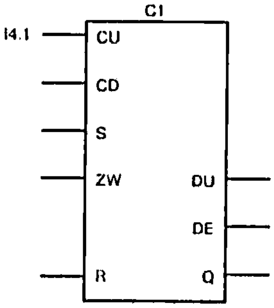
IW 20:
Setting a counter with the value of operands I, Q, F or D present in BCD code (input word 20 in the example).

Resetting a counter

Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	A I 4.2 R C 1 A C 1 = Q 2.4	 C1 CU CD S IW20 ZW DU DE I4.2 R Q Q2 4	 C1 CU CD S ZW DU DE I4.2 R Q Q2.4

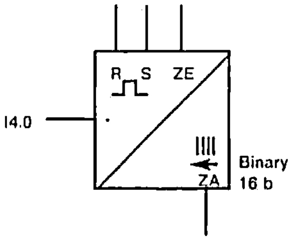
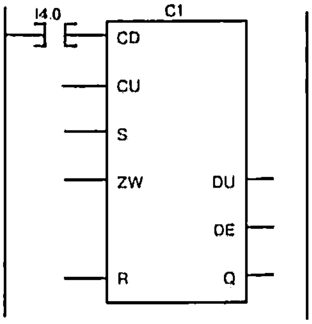
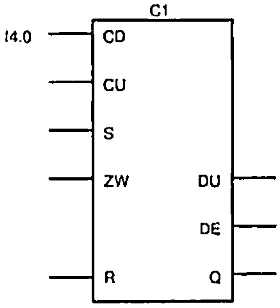
If the result of the logic operation is 1 the counter is set to zero (cleared).
If the result of the logic operation is 0 the counter is unchanged.

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/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		

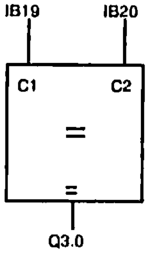
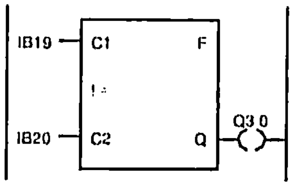
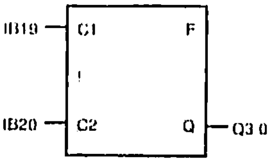
The value of the addressed counter is decremented by 1. The function only becomes effective with a positive-going edge (from 0 to 1) of the logic operation programmed before CD. The flags required for signal edge decoding of the count inputs are also in the count word.
A counter with two different inputs can be used as an up/down counter by means of the two separate signal edge flags for CU (count up) and CD (count down).

9.3.5 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
! = □		Test if equal
>< □		Test if not equal
> □		Test if greater
> = □		Test if greater than or equal to
< □		Test if less
< = □		Test if less than or equal to
↑		
F		Two fixed-point numbers
G		Two floating-point numbers
D		Two fixed-point double-word numbers

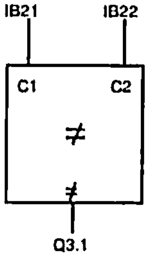
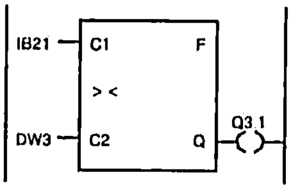
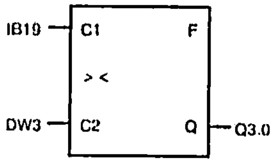
Test if equal

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) is taken into account in the comparison.

0	IB19	Accu 2-L
0	IB20	Accu 1-L

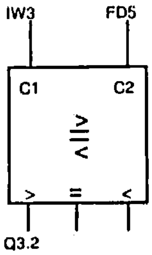
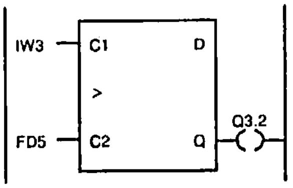
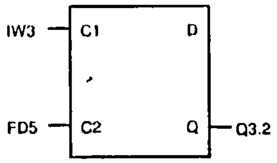
Test if not equal

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) are taken into account in the comparison.

0	IB21	Accu 2-L
DW3		Accu 1-L

Test if greater

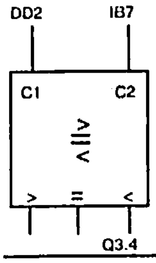
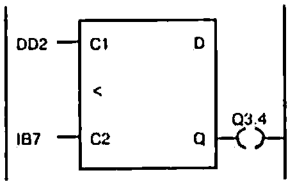
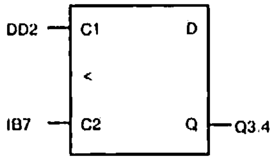
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L IW3 L FD5 > D = Q 3.2		

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

Accu 2-H	0	0	IB3	IB4	Accu 2-L
Accu 1-H	FB5	FB6	FB7	FB8	Accu 1-L

The numeric representation of the operands is taken into account in the comparison, i.e. the contents of Accu 1 and Accu 2 are interpreted as fixed-point numbers with double-word width.

Test if less

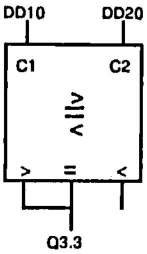
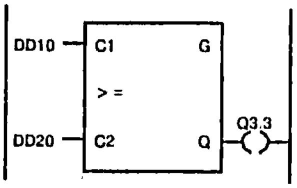
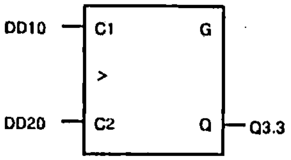
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L DD2 L IB7 < D = Q 3.4		

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

Accu 2-H	DW2		DW3		Accu 2-L
Accu 1-H	0	0	0	IB7	Accu 1-L

The numeric representation of the operands is taken into account in the comparison, i.e. the contents of Accu 1 and Accu 2 are interpreted as fixed-point numbers with double-word width.

Test if greater than or equal to

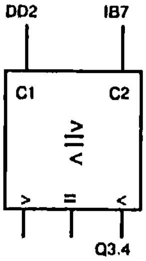
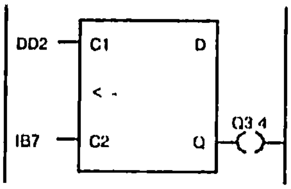
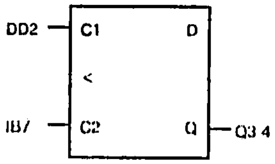
Given circuit	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	L DD10 L DD20 > =G =		

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

Accu 2-H	DW10	DW11	Accu 2-L
Accu 1-H	DW20	DW21	Accu 1-L

The numeric representation of the operands is taken into account in the comparison, i.e. the contents of Accu 1 and Accu 2 are interpreted as a floating-point number.

Test if less than or equal to

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

Accu 2-H	DW2		DW3		Accu 2-L
Accu 1-H	0	0	0	IB7	Accu 1-L

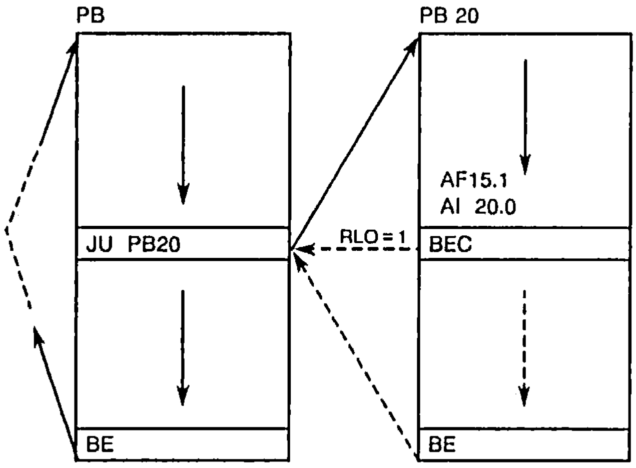
The numeric representation of the operands is taken into account in the comparison, i.e. the contents of Accu 1 and Accu 2 are interpreted as a floating-point number.

9.3.6 Block calls

Operation	Parameter	Function
JU □□ JC □□ ↑↑		Unconditional jump Conditional jump (depending on the RLO)
P B F B S B	1 to 255 1 to 255 1 to 255	to a progam block to a function block (type FB) to a sequence block
C D B	1 to 255	Data block call
BE BEC BEU		Block end Block end, conditional (depending on RLO) Block end, unconditional

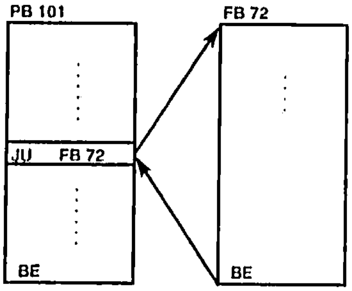
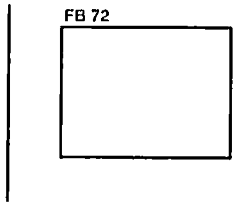
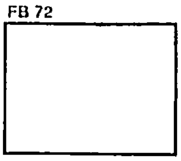
Command C DB (data block call) is explained under "Calling data blocks" (Chapter 3.1).
Commands BE (block end) and BEC (block end, conditional) result in a return to the previously nested block.
Command BE must be programmed at the end of each data block (except for DB, DX).

Example:



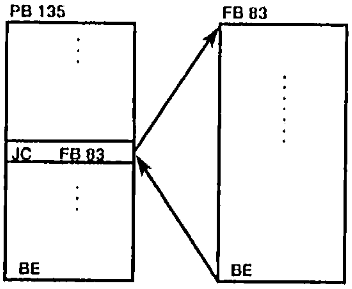
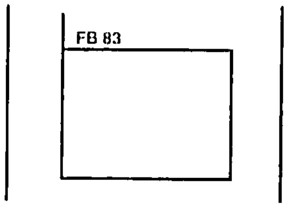
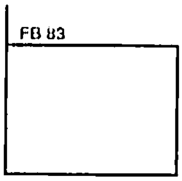
If the result of the logic operation is 1, the return to PB 7 already takes place with processing of the BEC command.
If the result of the logic operation is 0, processing of PB 20 continues up to the BE command, which then initiates the return to PB 7 when PB 20 has been fully processed.

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 FB 83 . . .		

(Previous logic operation given as example only.)

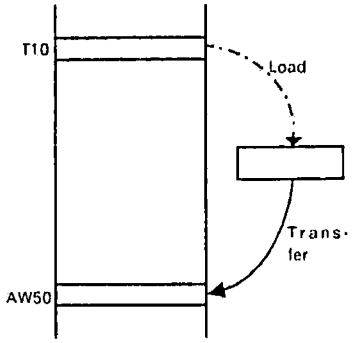
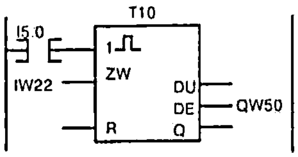
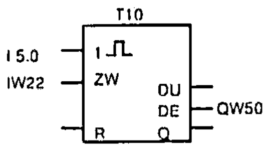
At the desired program point, the conditional block call is entered after the appropriate logic operation. If the result of the logic operation is 1, a jump to the specified block takes place. If the condition is not fulfilled the jump is not executed. With graphic methods of representation LAD and CSF,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

9.3.7 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	1 to 127	times
C	1 to 63	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 ⎍L ZW IW22 DU DE QW50 R Q	 T10 I 5.0 IW22 1 ⎍L ZW DU DE QW50 R Q

The content of the memory location addressed with T 10 is loaded as a code in 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.

9.3.8 Arithmetic operation

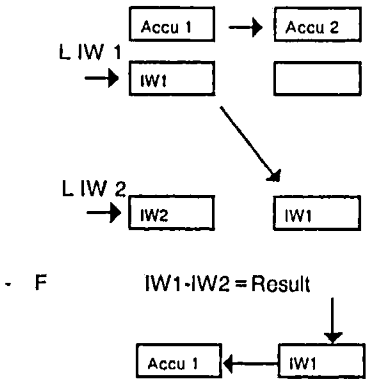
Arithmetic operation can only be represented in the statement list.

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

Operation	Parameter	Function
+ □		Addition
- □		Subtraction
x □		Multiplication
: □		Division
↑		
F		of two fixed-point numbers
G		of two floating-point numbers

By means of two load operations, Accumulators 1 and 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 2-24 may be incorrect if the difference between the two exponents is greater than 24.

9.3.9 Other operations

The following operations can only be representated in the 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.

9.4 Supplementary operation (FB 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 from (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.

9.4.1 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:

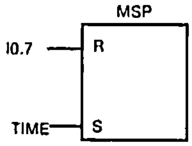
```
: A     -ON
: AN    -STOP
: AN    -END
: O     -AMNT
: =     -RUN
```

9.4.2 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:

```
Q  I  0.7
RB =  MSP
Q  =  TIME
S  =  MSP
```



9.4.3 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 (see Page 8) 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)

Examples (see Chapter 4.4.3):

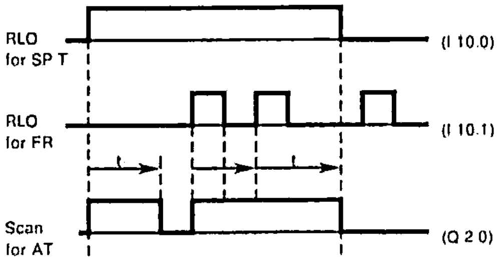
Function block call	Program in function block	Program executed
:JU FB203 NAME :EXAMPLE ANNA : I 10.3 BERT : T 17 HANS : Q 18.4	: A = ANNA : L KT 010.2 : SSU = BERT : A = BERT : = = HANS	: A I 10.3 : L KT 010.2 : A T 17 : A T 17 : = Q 18.4
:JU FB204 NAME :EXAMPLE MAXI : I 10.5 IRMA : I 10.6 EVA : I 10.7 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 KURT : I 10.4 CARL : T 18 EGON : IW20 MAUS : F 100.7	: A = KURT : L = EGON : SEC = CARL : = = MAUS	: A I 10.4 : L IW20 : SE T 18 : = F 100.7

9.4.4 Enabling operations for timing and counting operations

Operation	Description
FR T 0 to 127	**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 127 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
```



9.4.5 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 127.15	a count word
T	0.0 to 127.15	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.

<u>1st example</u>

The logic level of the 10th bit of data word 205 is ANDed with the logic level of input I13.7.

```
: 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).

<u>2nd example</u>

The third bit is to be set by DW55 and the 9th bit by DW103.

```
: SU D 55.3
: RU D 103.11
```

9.4.6 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).
LC = □	**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:

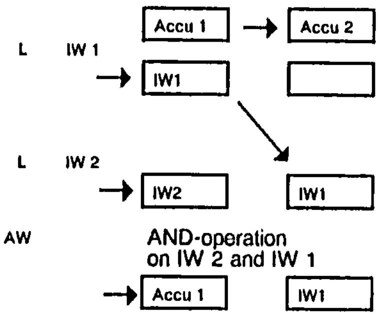
Function block call	Program in function block	Program execution
: JU FB 206		
NAME : COMP.		
C1 : KH 7F0A	: LW =C1	: L KH 7F0A
C2 : DW 20	: L =C2	: L DW 20
	: ! = F	: ! = F

9.4.7 Logic operations, digital

Operation	Description
AW **OW** **XOW**	AND operation, digital, Accu 1 and Accu 2 OR operation, digital, Accu 1 and Accu 2 Exklusive 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.

9.4.8 Shift operations

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

9.4.9 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	Decimal to fixed-point conversion
DUF	Fixed-point to decimal conversion
DED	Decimal to fixed-point, double word conversion
DUD	Fixed-point double word to decimal conversion
FDG	Fixed-point double word to floating point conversion
GFD	Floating point to fixed point, double word conversion

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   DW 64          KM = 0011111001011011
: CFW
: T   DW 78          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   DW 207         KF: +51
: CSW
: T   DW 51          KF: - 51
```

9.4.10 Decrementing/ incrementing

Operation	Description
D 1 to 255	Decrementing
I 1 to 255	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   DW 7         KH = 1010
: I    16
: T   DW 8         KH = 1020
: D    33
: T   DW 9         KH = 10FF
```

9.4.11 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 statement 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 = □	**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.

<u>Note:</u>

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 last shifted bit  =  0)
      : JN      =  EINS      (Jump if last shifted bit  =  1)
      :
NULL  :
      :
EINS  :
      :
```

Example (conversion operations):

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

9.4.12 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 "Type of block parameter and allowed actual operand, Chapter 6.2)
DO DW 0 to 254 (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 254	**Process flag word*** The specified operation which follows will be combined with the parameter given in the flag word and executed.

*Two-word commands can also be substituted. The following command sequences, for example, are therefore possible:

```
C   DB20            C   DB100
.                   .
          or
.                   .
DO FW240            DO DW21
TB  D 0.0           S   C 0.0
```

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. Any bits which are beyond the bit address in the high byte of the pointer will be deleted.

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

- The distinction between inputs and outputs is not made in the opcode 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
 SU I 0.0            (or SU Q 0.0)
->I   4.1            will be set
```

- If Q 4.1 is to be set, the value KH0184 must stored in the flag 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     KF0       Reset
     : DO    DW1
     : T     DW0
     : L     DW1       Increment the index register
     : L     KF1
     : + F
     : T     DW1
     : L     KF100
     : < = F
     : JC    = M1      Jump if the index is within the range
     .....             Other STEP 5 program
```

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

A, AN, O, ON	Binary operations
S, R, = .	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, JZ, JN, JP, JM, JO	Jump functions
SLW, SRW	Shift functions
D, I	Decrementing, incrementing
C DB, JU, JC	Block calls

The 670 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.

Example: **Process data word**

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

```
     : L     KF20      Assignment of "index register"
     : T     DW1
M1   : L     KF0       Reset
     : DO    DW1
     : T     DW0
     : L     DW1       Increment the index register
     : L     KF1
     : + F
     : T     DW1
     : L     KF100
     : < = F
     : JC    = M1      Jump if index is in range
     .....             Other STEP 5 program
```

9.4.13 STEP 5 commands with direct memory access

Operation	Beschreibung
LIR 0	Load ACCU 1: with the contents of the memory word addressed via Accu 1
TIR 2	Transfer ACCU 2: to the memory word addressed via the contents of Accu 1
TNB 0 to 255	Block transfer byte by byte, source address in Accu 2, destination address in Accu 1
TNW 0 to 255	Block transfer block by block, source address in Accu 2, destination address in Accu 1

The LIR, TIR, TNB and TNW commands must first be enabled via machine data (MD 2003, Bit 4 = 1) if thy are to be used in the user program. Otherwise an error message is output and the PLC branches into a stop loop.

Only commands LIR 0 and TIR 2 will be executed. However 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 accumulator.

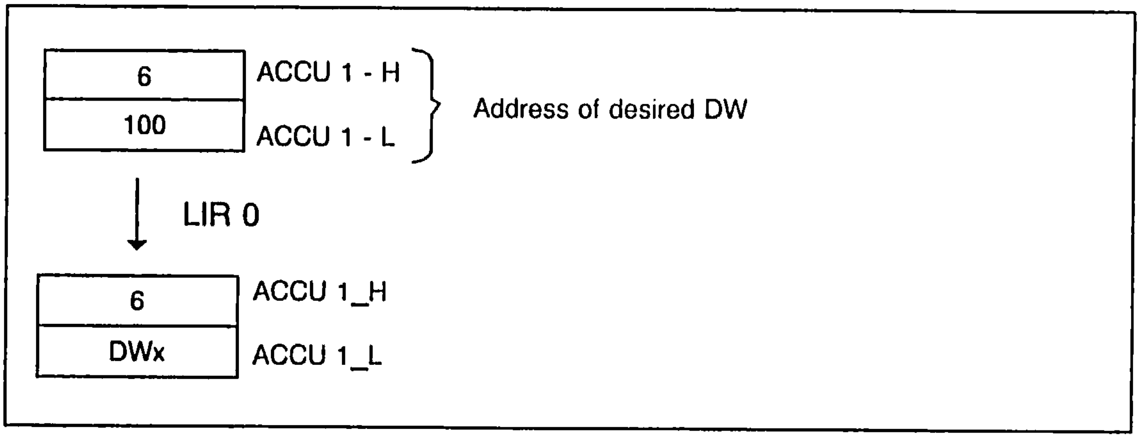
The assignments are as follows:

Segment no.	Segment
1	PLC system data (e.g. process images, block address lists)
2	Hardware (e.g. EIA signal converter, MPC link memory)
3	Internal NC-PLC interface
4	Spindle interface
5	User program memory
6	User data memory
7	PLC system program
8	Resident function blocks (e.g. FB 11)

Segment number 1-10 are permitted for read access (source). Only segments 1-6 are permitted as destination for commands TIR, TNB and TNW. If other numbers are assigned the PLC goes into the stop state and an error message is output.

Example 1: Read access to particular address

```
L KB 6       Segment address for DWx→ ACCU 1_H
TFW 250
L KH 0100    Offset address for DWx→ ACCU 1_L
TFW 252
L FD 250
LIR 0        Load DWx to ACCU 1_L
```



Example 2: Write operation to particular address

```
L   KB 6     Segment address for value x
T FW 250
L KH 0100    Offset address for value x
T FW 252
L   KH FFFF  Load value x
L FD 250
TIR 2        Transfer value x→user data segment
```

9.4.14 Other operations

Operation	Description
ADD BF -127 to +127 **ADD KF -32768 to +32767**	Add byte constant (fixed-point) to Accu 1 Add fixed-point constant (word) to Accu 1
STS **TAK**	Stop command Swap contents of Accu 1 and Accu 2

9.5 Step 5 command execution times

COMMAND	OPCODE	EXECUTION TIME (microseconds)
A AN O ON		0.6 - 1.1
S R =		1.6
O		0.6
NOP 0	0000	0.6
NOP 1	FFFF	0.6
BLD 255	10FF	0.6
FRT	0400	1.1 - 1.6
FRZ	4400	1.1 - 1.6
A (	BA00	0.6
O)	BB00	0.6
)	BF00	0.6
L FB	0A00	11.6
L IB / QB	4A00	14.6
L FW	1200	11.6
L IW / QW	5200	11.6
L FD	1A00	15.2
L ID / QD	5A00	15.2
T FB	0B00	12.8
T IB / QB	4B00	12.8
T FW	1300	14.0
T IW / QW	5300	14.0
T FD	1B00	19.0
T ID / QD	5B00	19.0

COMMAND	OPCODE	EXECUTION TIME (microseconds)
L PB	7200	20.4
L PW	7A00	22.4
T PB	7300	25.8
T PW	7B00	28.8
L KB	2800	8.8
L KH	3000 0000	17.6
L KG	3800 0000 0000	21.0
L C	4200	14.6
L T	0200	14.6
LC C	4C00	28.0 - 76.6
LC T	0C00	30.4 - 79.0
L DR	2A00	17.2
L DL	2200	16.6
L DW	3200	16.6
L DD	3A00	20.0
T DR	2B00	19.2
T DL	2300	17.2
T DW	3300	22.6
T DD	3B00	26.8
LIR	4000	17.0
TIR	4800	18.0
TNB	0300	38.0 - 323.6
TNW	4300	40.6 - 326.4
SR T	2400	1.35 - 36.6
SS T	2C00	1.35 - 36.6
SP T	3400	1.35 - 36.4
SE T	1C00	1.35 - 36.4
SF T	1400	1.35 - 36.4
R T	3C00	1.35 - 12.6

COMMAND	OPCODE	EXECUTION TIME (microseconds)
R C	7C00	1.35 - 12.6
S C	5C00	1.35 - 36.2
CU Z	6C00	1.35 - 23.0
CD C	5400	1.35 - 21.4
F T	0400	1.35
F C	4400	1.35
JU FB	3D00	71.6
JC FB	1D00	0.6 - 71.6
JU PB	7500	60.0
JC PB	5500	0.6 - 60.0
JU SB	7D00	60.0
JC SB	5D00	0.6 - 60.0
BE	6500	16 - 45.6
BEA	6501	16 - 45.6
BEC	0500	0.6 - 45.6
C DB	2000	30.8
JU =	2D00	12.2
JC =	FA00	0.6 - 12.2
JM =	2500	9.6 - 15.6
JN =	3500	9.6 - 15.6
JP =	1500	9.6 - 15.6
JZ =	4500	9.6 - 15.6
JO =	0D00	9.6 - 15.6
SRW	6900	14.2
SLW	6100	14.2
SSD	7100	15.2 - 223.2
SLD	2900	15.2 - 223.2

COMMAND	OPCODE	EXECUTION TIME (microseconds)
>F (GRF)	2120	25.6 - 27.4
<F (KLF)	2140	25.6 - 28.0
><F (UGLF)	2160	25.6 - 28.0
!=F (GLF)	2180	26.2 - 28.8
>=F (GGF)	21A0	26.2 - 28.8
<=F (KGF)	21C0	27.2 - 28.0
>D (GRD)	3920	26.0 - 28.0
<D (KLD)	3940	26.0 - 28.0
><D (UGLD)	3960	26.0 - 28.0
!=D (GLD)	3980	26.0 - 28.0
>=D (GGD)	39A0	26.0 - 28.0
<=D (KGD)	39C0	26.0 - 28.0
>G (GRG)	3120	29.0 - 41.0
<G (KLG)	3140	29.0 - 41.0
><G (UGLG)	3160	29.0 - 41.0
!=G (GLG)	3180	29.0 - 41.0
<=G (GGG)	31A0	29.0 - 41.0
>=G (KGG)	31C0	29.0 - 41.0
+F (ADDF)	7900	14.4 - 19.4
- F (SUBF)	5900	15.6 - 20.6
D	1900	6.8
I	1100	6.8
ADD BN	5000	7.4
ADD KF	58 000 0000	16.6

COMMAND	OPCODE	EXECUTION TIME (microseconds)
*F (NULP)	6004	27.4 - 33.6
:F (DIVF)	6000	22.4 - 33.2
*G (MULG)	6007	35.0 - 55.0
:G (DIVG)	6003	18.8 - 55.0
+G (ADDG)	600F	40.0 - 385.0
- G (SUBG)	600B	40.0 - 400.0
JOS	600C 0000	27.6 - 33.6
CFW	0100	5.6
CSW	0900	13.6 - 15.0
AW	4100	14.2
OW	4900	14.2
XOW	5100	14.2
SSW	6801	24.4
CSD	6807	22.6 - 27.0
BDW	6808	36.0 - 112.4
BDD	680A	29.6 - 330.2
CBW	680C	40.0
DED	680E	22.2 - 90.6
GFD	6002	19.4 - 245.0
FDG	6806	300.0 - 380.0
TAK	7002	13.8
STP	7003	15.8
STS	7000	- - - -
RU C	7015 0000	46.0
SU C	7015 4000	45.4
TBN C	7015 8000	47.6
TB C	7015 C000	47.2
RU T	7025 0000	45.2

COMMAND	OPCODE	EXECUTION TIME (microseconds)
SU T	7025 4000	44.6
TBN T	7025 8000	46.6
TB T	7025 C000	46.6
RU D	7046 0000	51.2
SU D	7046 4000	50.6
TBN D	7046 8000	48.6
TB D	7046 C000	48.6
RU I / Q	7038 0000	47.2
SU I / Q	7038 4000	43.0
TBN I / Q	7038 8000	44.2
TB I / Q	7038 C000	43.8
RU F	7049 0000	47.0
SU F	7049 4000	43.0
TBN F	7049 8000	44.4
TB F	7049 C000	43.6
A D	783F 0000	51.4
O D	783F 1000	51.4
AN D	783F 2000	51.4
ON D	783F 3000	51.4
S D	783F 4000	53.4
R D	783F 5000	54.0
= D	783F 6000	60.4
DO FW	4E00	29.0 - 50.0 + execution time
DO DW	6E00	32.0 - 52.0 + execution time

COMMAND	OPCODE	EXECUTION TIME (microseconds)	
S =	1700	23.2	+ execution time
RB =	3700	23.2	+ execution time
= =	1F00	23.2	+ execution time
A =	0700	26.4	+ execution time
O =	0F00	27.2	+ execution time
AN =	2700	27.2	+ execution time
ON =	2F00	27.2	+ execution time
L =	4600	19.4	+ execution time
T =	6600	20.2	+ execution time
LC =	0E00	23.6	+ execution time
LW =	3F00	20.2	+ execution time
LWD =	5600	25.0	+ execution time
SP =	3600	22.4	+ execution time
SR =	2600	22.4	+ execution time
FR =	0600	24.2	+ execution time
RD =	3E00	23.8	+ execution time
SEC =	1E00	24.4	+ execution time
SSU =	2E00	24.4	+ execution time
SFD =	1600	24.4	+ execution time
DO =	7600	27.0	+ execution time

10 Rules of compatibility between the LAD, CSF and STL methods of representation

10.1 General

Each method of representation in the STEP 5 programming language contains limits. It therefore follows that a program block written in an STL may not 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 a LAD or CSF, it can be retranslated 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.

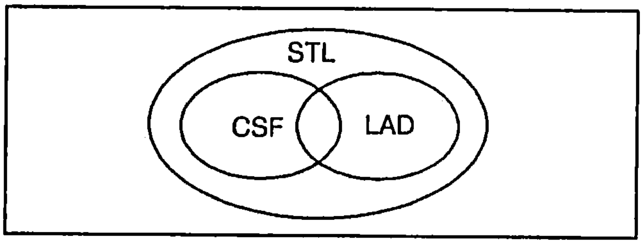


Fig. 10.1 Extent and restrictions of the methods of representation in the STEP 5 programming language

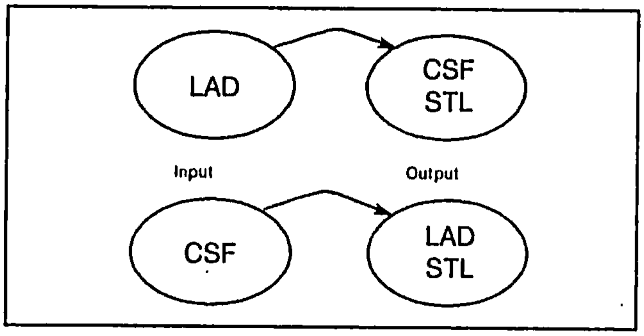


Fig. 10.2 Graphic input

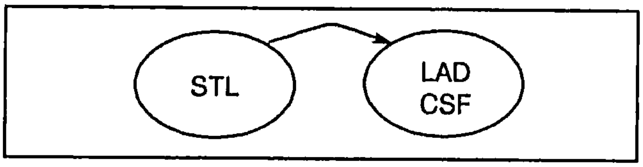


Fig. 10.3 Input in statement list

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.

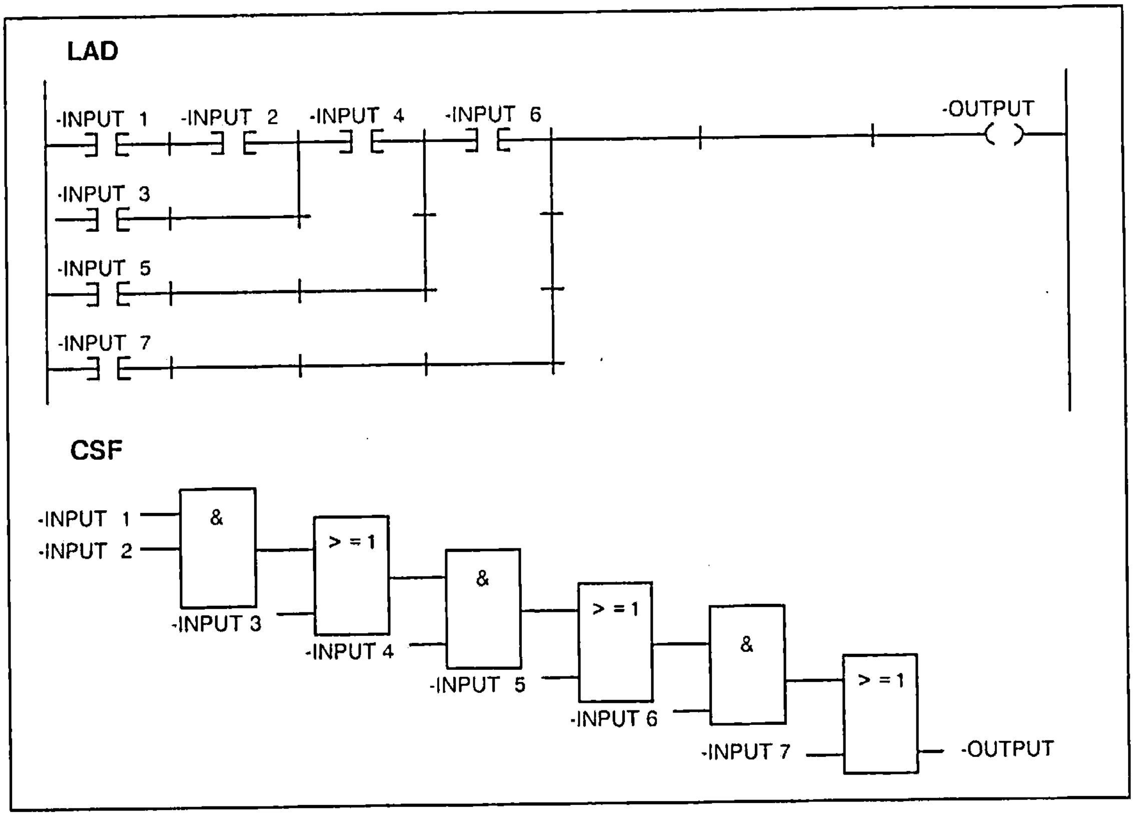


Fig. 10.4 Example of maximum LAD nesting for output in CSF

Input in CSF: Output in LAD and STL

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

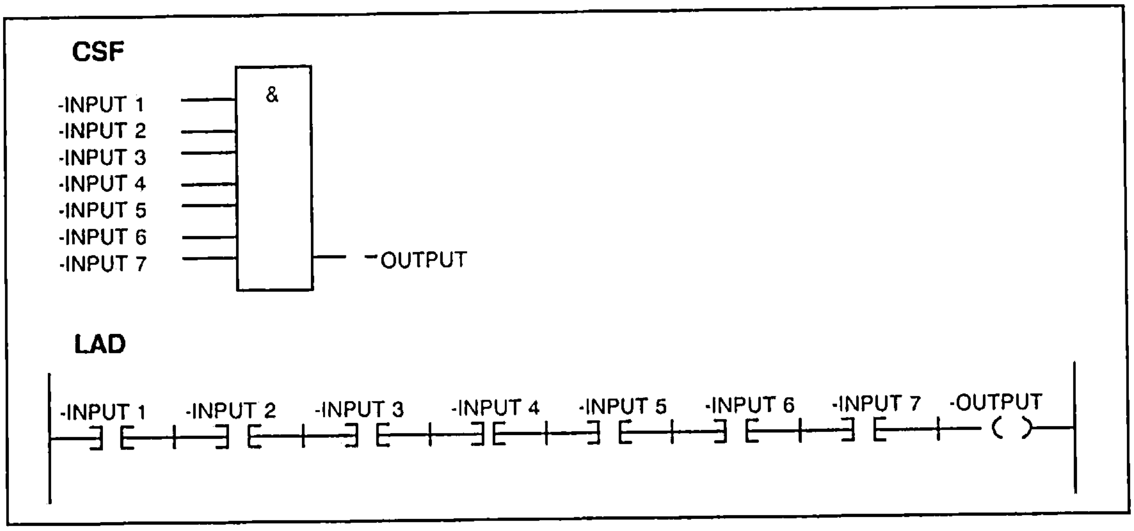


Fig. 10.5 Example of maximum AND-box expansion for output in LAD

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

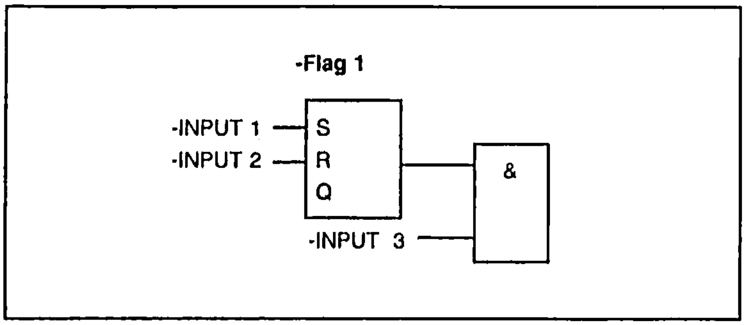


Fig. 10.6 Only AND-boxes are allowed following a complex element

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

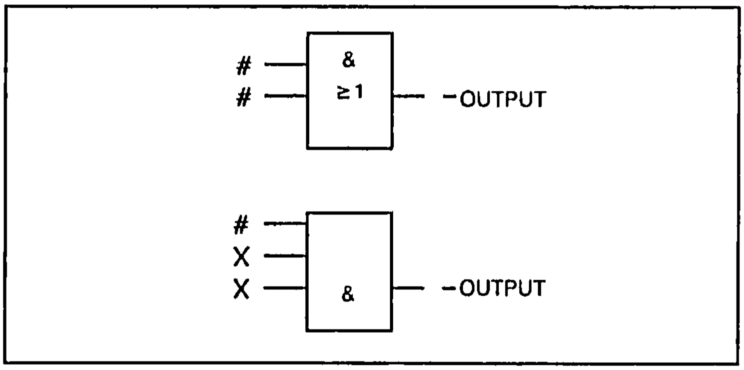


Fig. 10.7 Example showing where connectors are allowed with OR and AND-boxes

10.3 Rules of compatibility for program input in a statement list

Rule 1: AND operation:

(Test of logic state and AND logic).

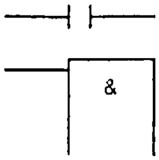
LAD: Contact in series

CSF: Input to an AND-box

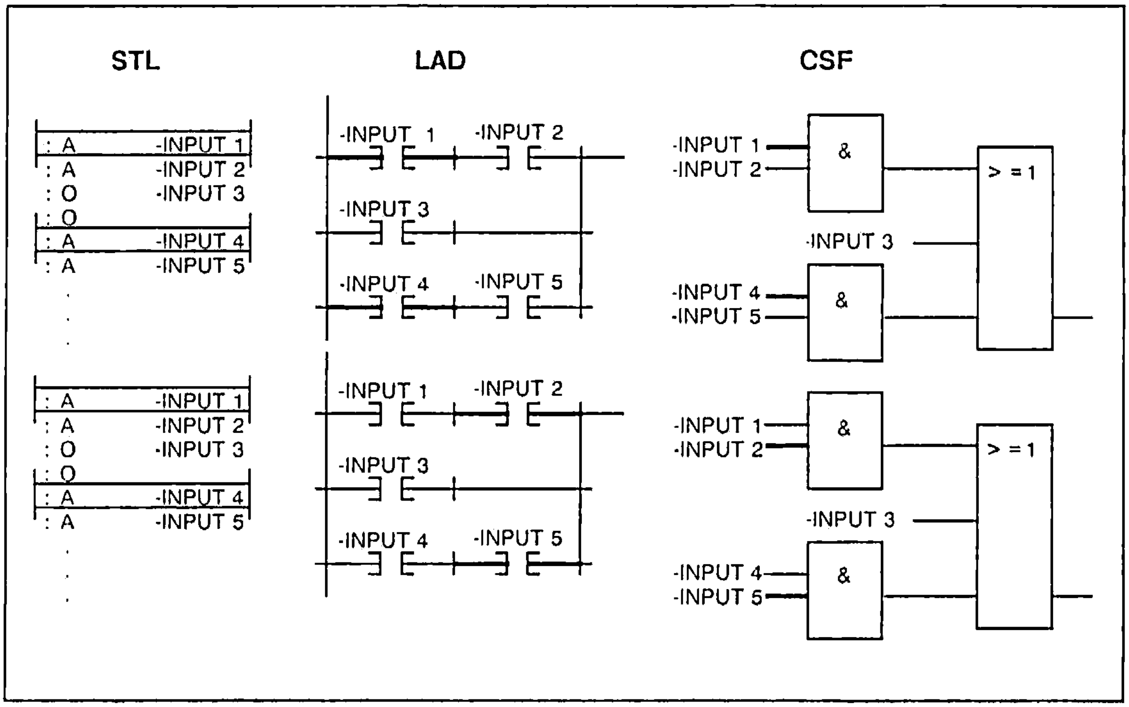
STL: Statement A . . .

LAD:

CSF:



STL: A . . . .



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

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

Fig. 10.8 Explanations of the rule for AND operations

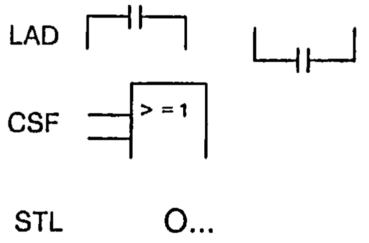
Rule 2: OR operation

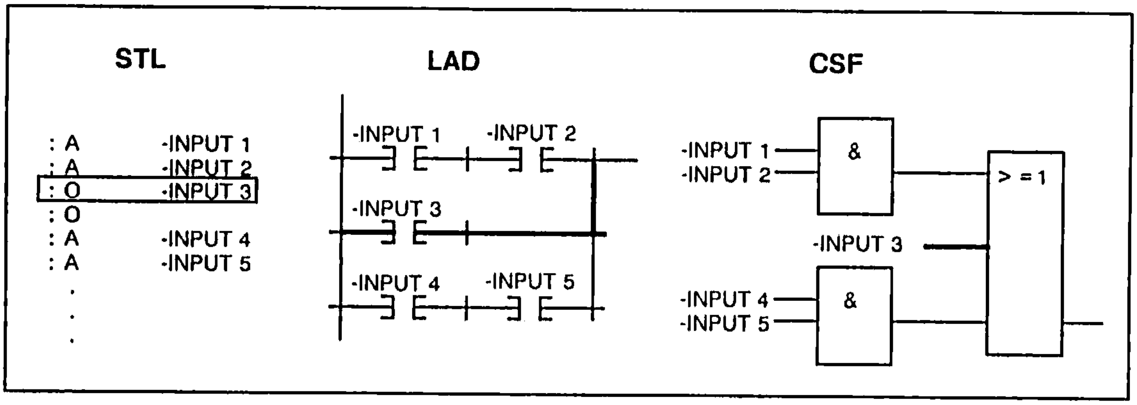
(Test of logic state and OR logic).

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

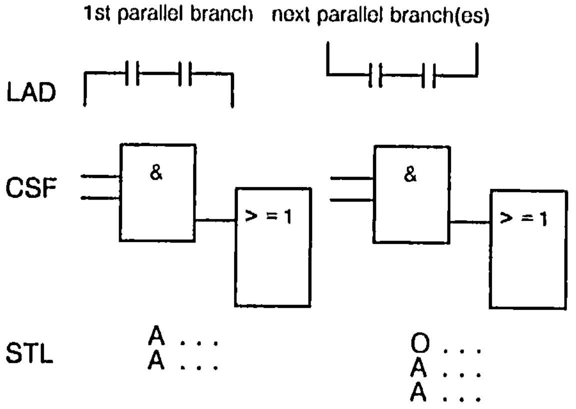
Rule 3: AND before OR operation

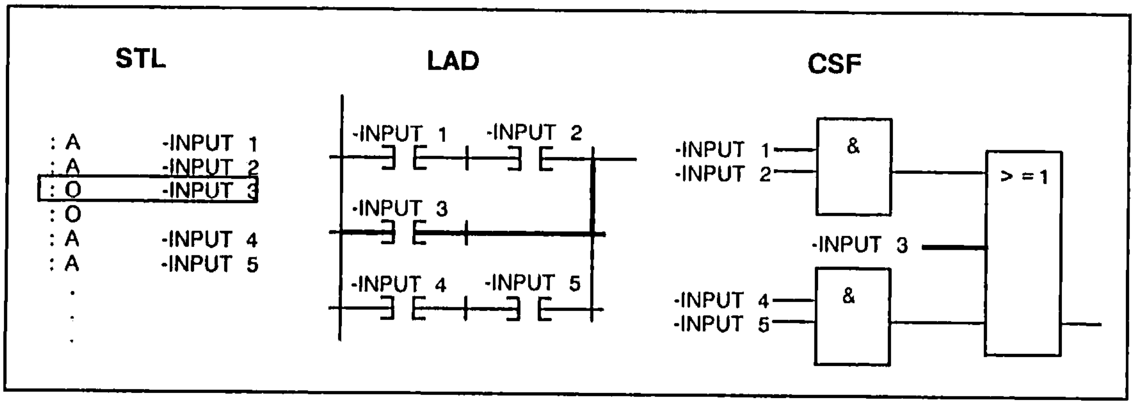
(OR operation before AND functions)

LAD: Two or more contacts in a parallel branch

CSF: AND-box before OR-box

STL:Statements
O . . .
A . . .
A . . .

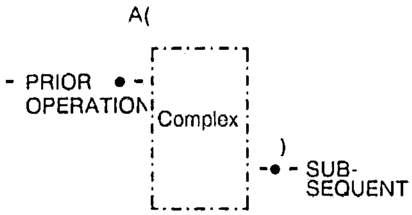




```
: A        -INPUT 1
: A        -INPUT 2
: O        -INPUT 3
: O
: A        -INPUT 4
: A        -INPUT 5
  .
  .
  .
```

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.

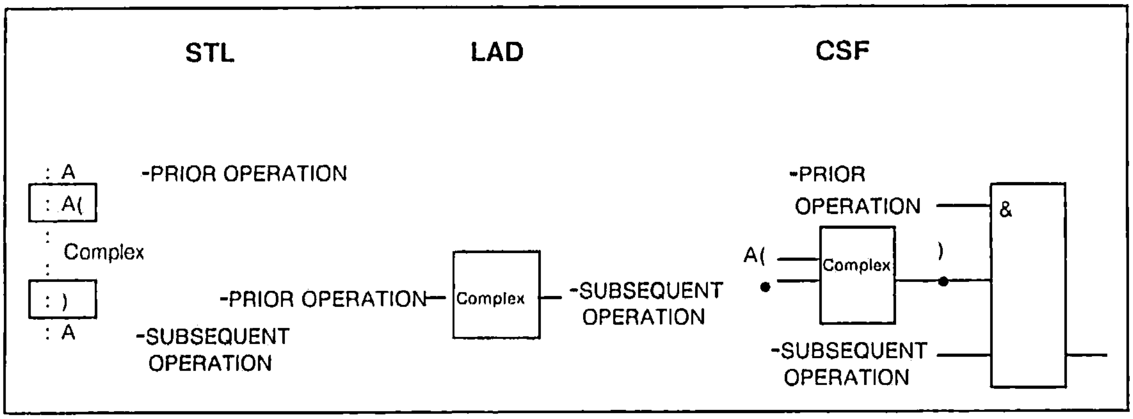


Fig. 10.11 Explanations of parenthesized, complex binary functions

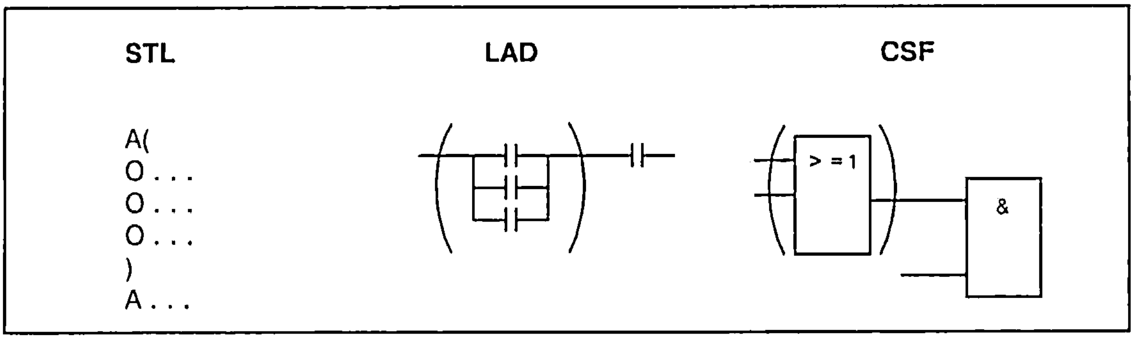


Fig. 10.12 Explanations of the rule for OR before AND operation

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.

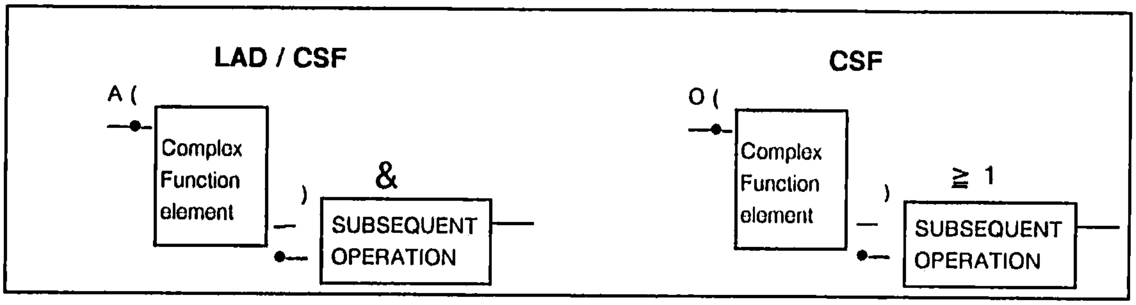
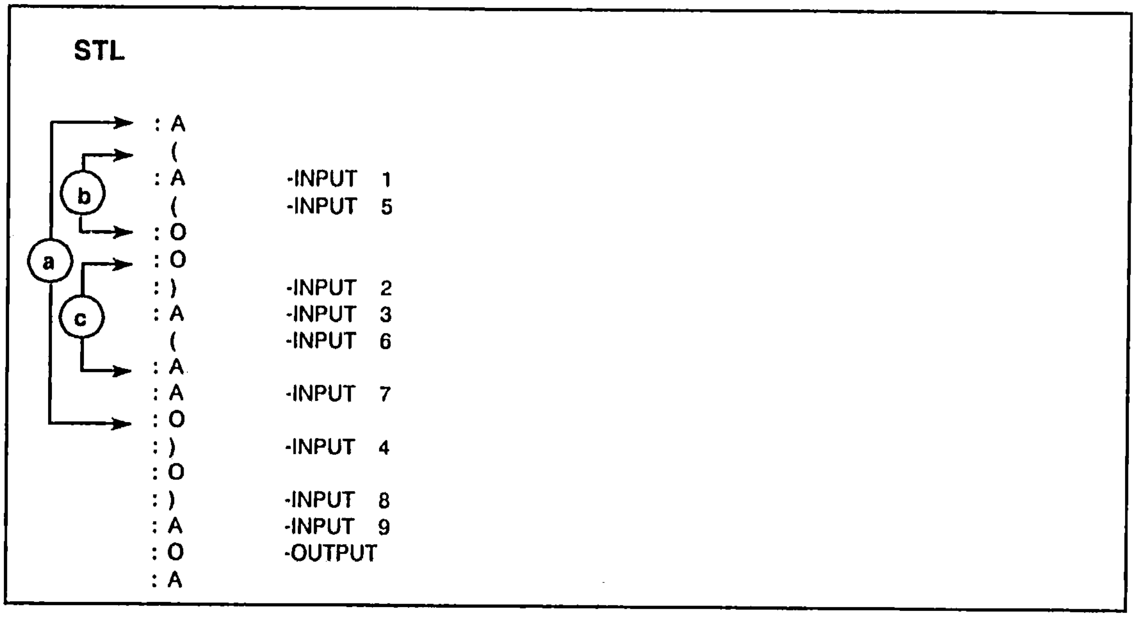


Fig. 10.13 Explanations of parentheses for complex elements

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

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

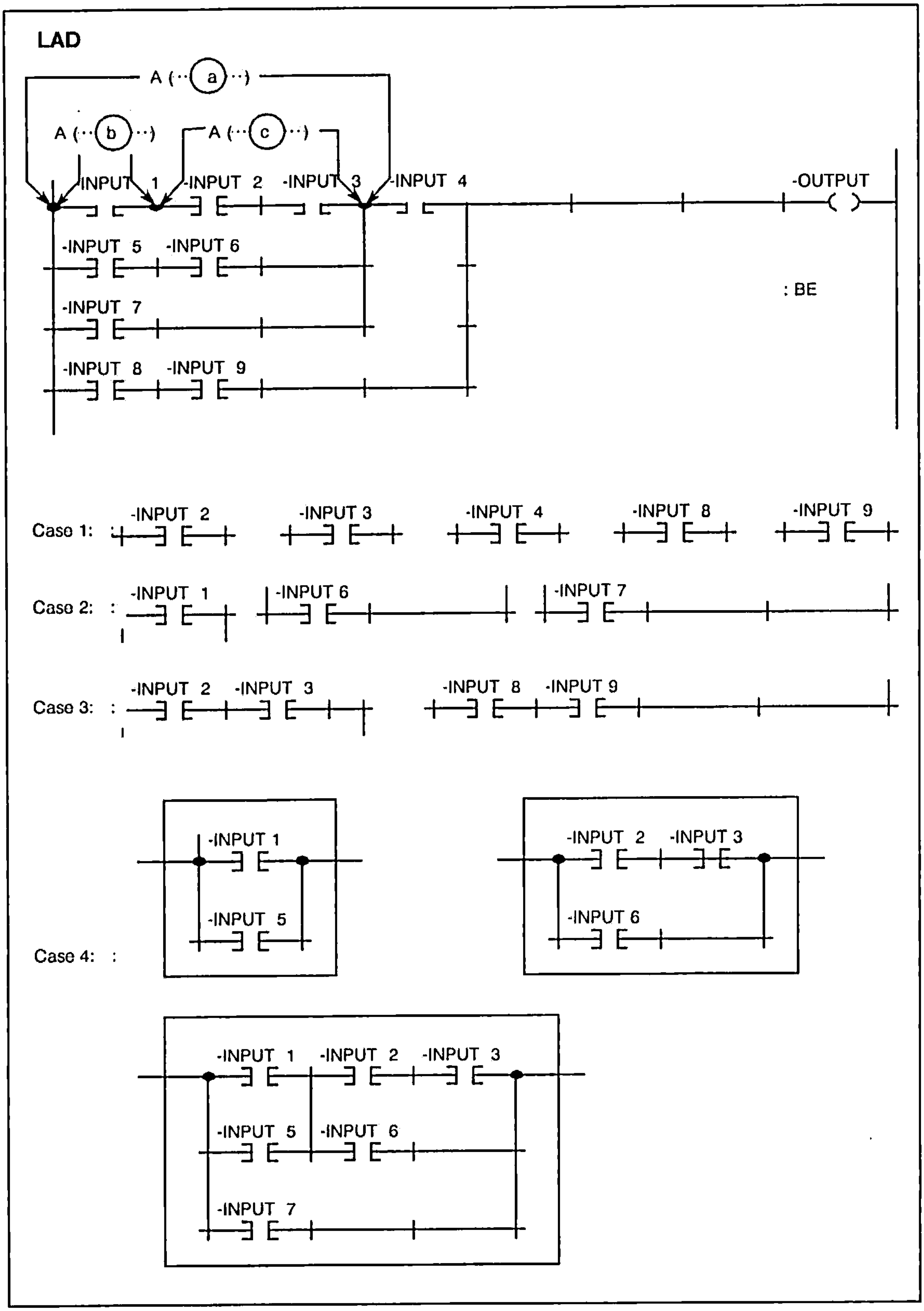


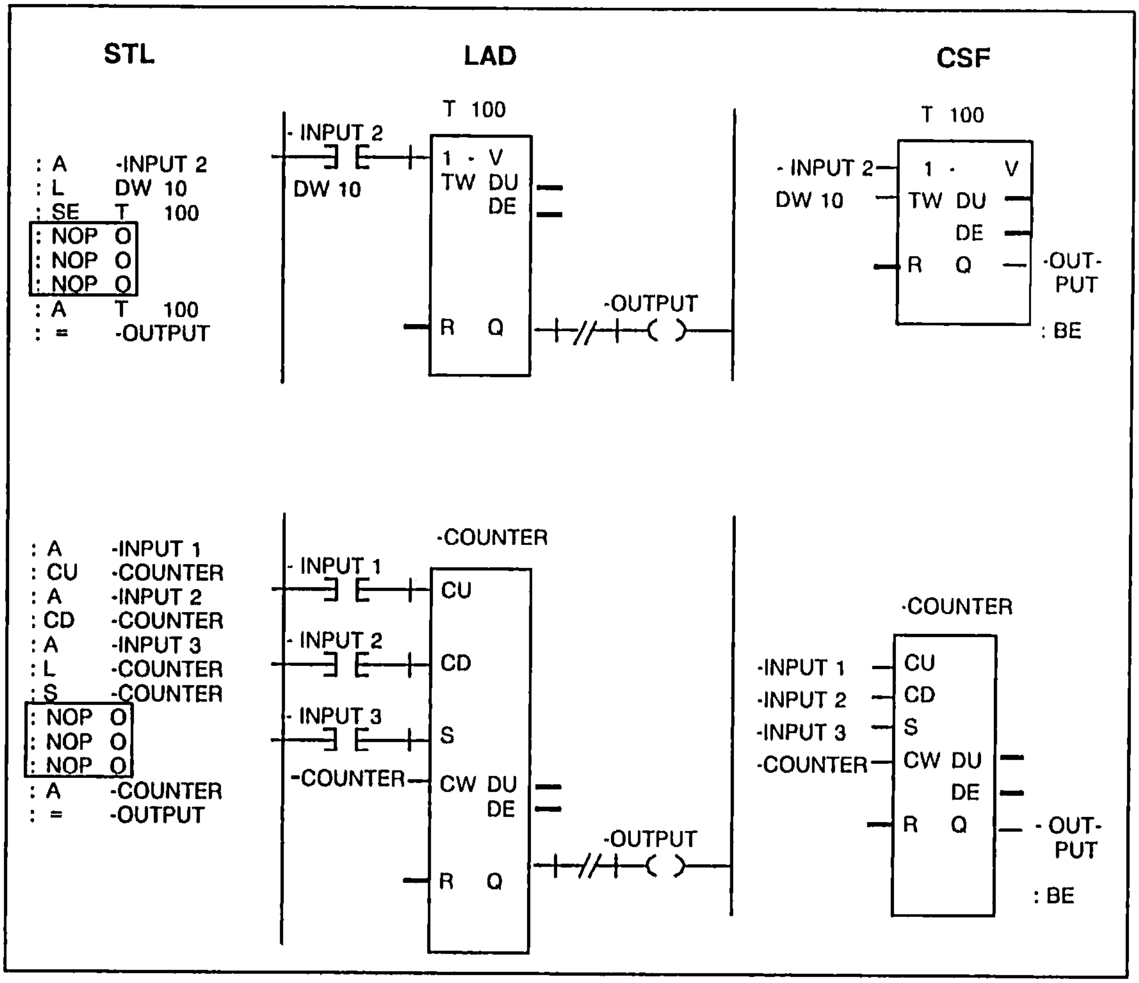
Fig. 10.14 Example 1: LAD/STL (continued)

NOP 0 must be applied to each unused input or output.

Exception: S and TW for timers, and S and CW 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.



STL

```
: 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     -COUNTER
: S     -COUNTER
: NOP  O
: NOP  O
: NOP  O
: A     -COUNTER
: =     -OUTPUT
```

Fig. 10.15 Example of assignments for unused inputs and outputs

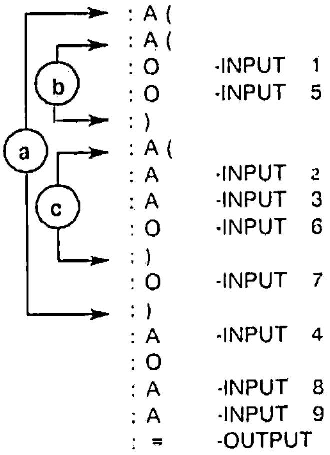
Important note: Only one complex function element is allowed per network.

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.

Examle 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

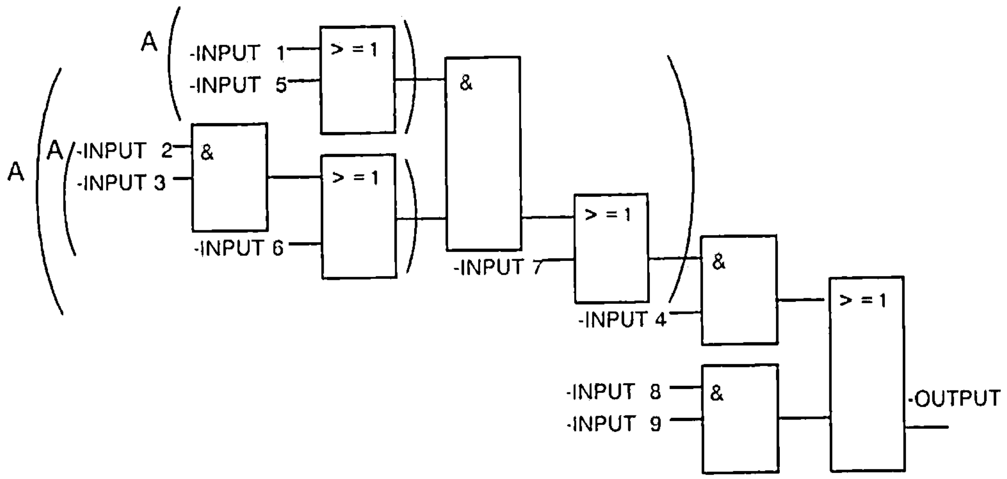


Fig. 10.16 Example 2: CSF/STL (continued on next page)

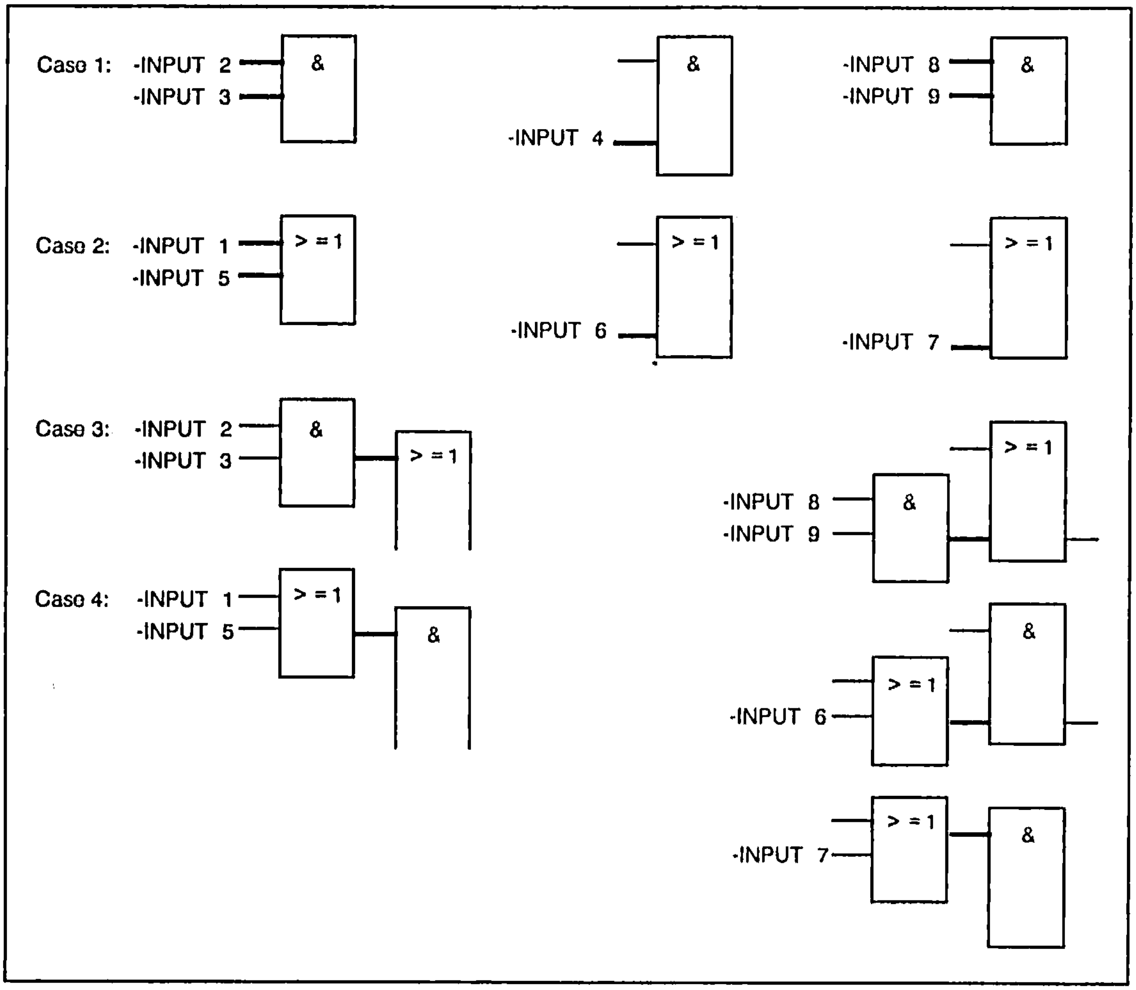
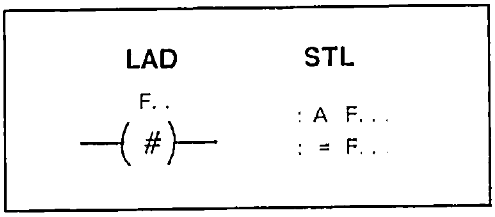


Fig. 10.16 Example 2: CSF/STL (continued)

Rule 5: Connectors

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



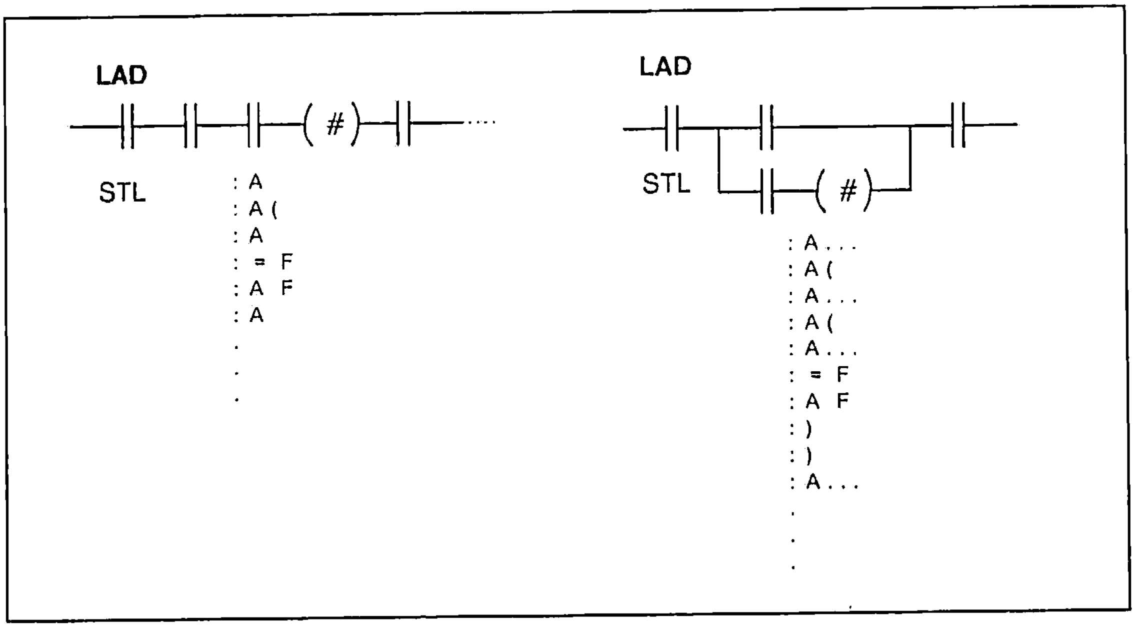
```
: A  F. . .
: =  F. . .
```

Fig. 10.17 The connector in LAD and STL

a) Connectors with LAD

A connector registers, as an intermediate store, the result of the logic operation from the operations programmed before it in its own circuit. 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).



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

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

Fig. 10.18 Connector controller for LAD

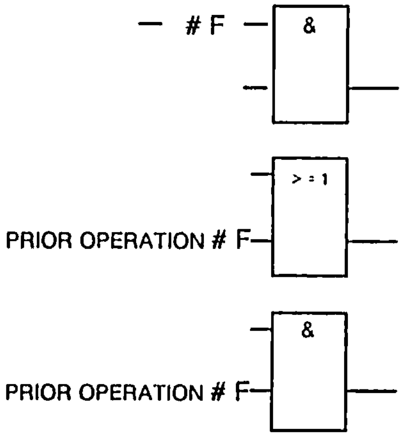
CSF	STL
— # F... —	: = F... :A F...

Fig. 10.19 The connector in CSF and STL

b) Connectors with CSF

A connector registers, as an intermediate store, the result of the entire binary operation preceding this connector. The following rules apply:

- Connector at the first input of an AND or OR-box:
  The connector is processed 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.)



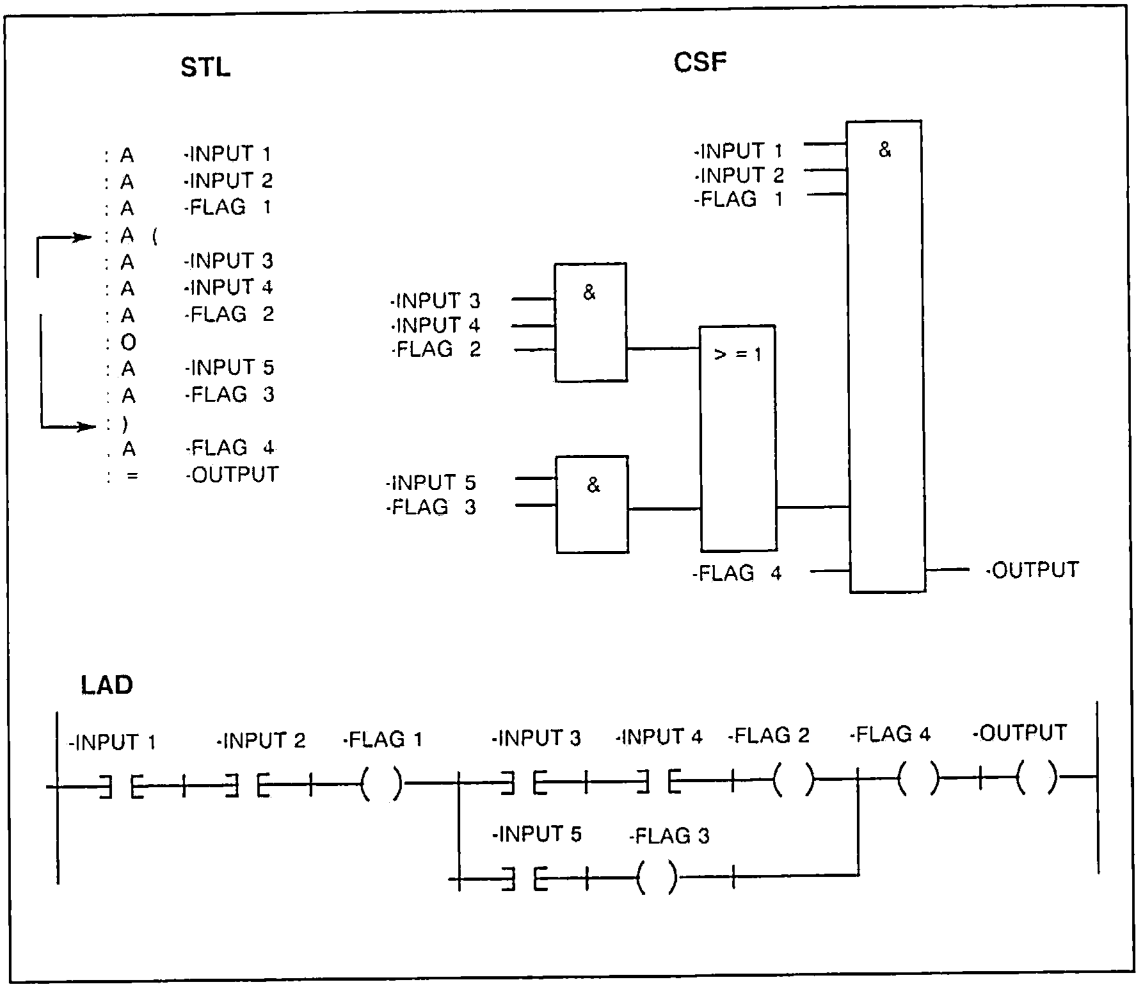
```
: =      F...
: A      F...
: A...
: A...
: O...
: O (
P. O.*
: = F
: A  F...
: )
: A...
: A  (
P. O.*
: = F...
: A  F...
: )
```

*PRIOR OPERATION

Fig. 10.20 Connector controller for CSF

Examples for connectors:

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



STL

```
: 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

Siemens AG

AUT V230
Postfach 48 48
W-8500 Nürnberg 1
Federal Republic of Germany

Suggestions

Corrections

For Publication/Manual:
SINUMERIK 810
Basic Version 2
SINUMERIK 820
PLC Programming
Manufacturer Documentation

Programming Guide

Order No.: 6ZB5 410-0BX02-0BA1
Edition: October 1990

From:

Name ______________________________

Company/Dept. ______________________________

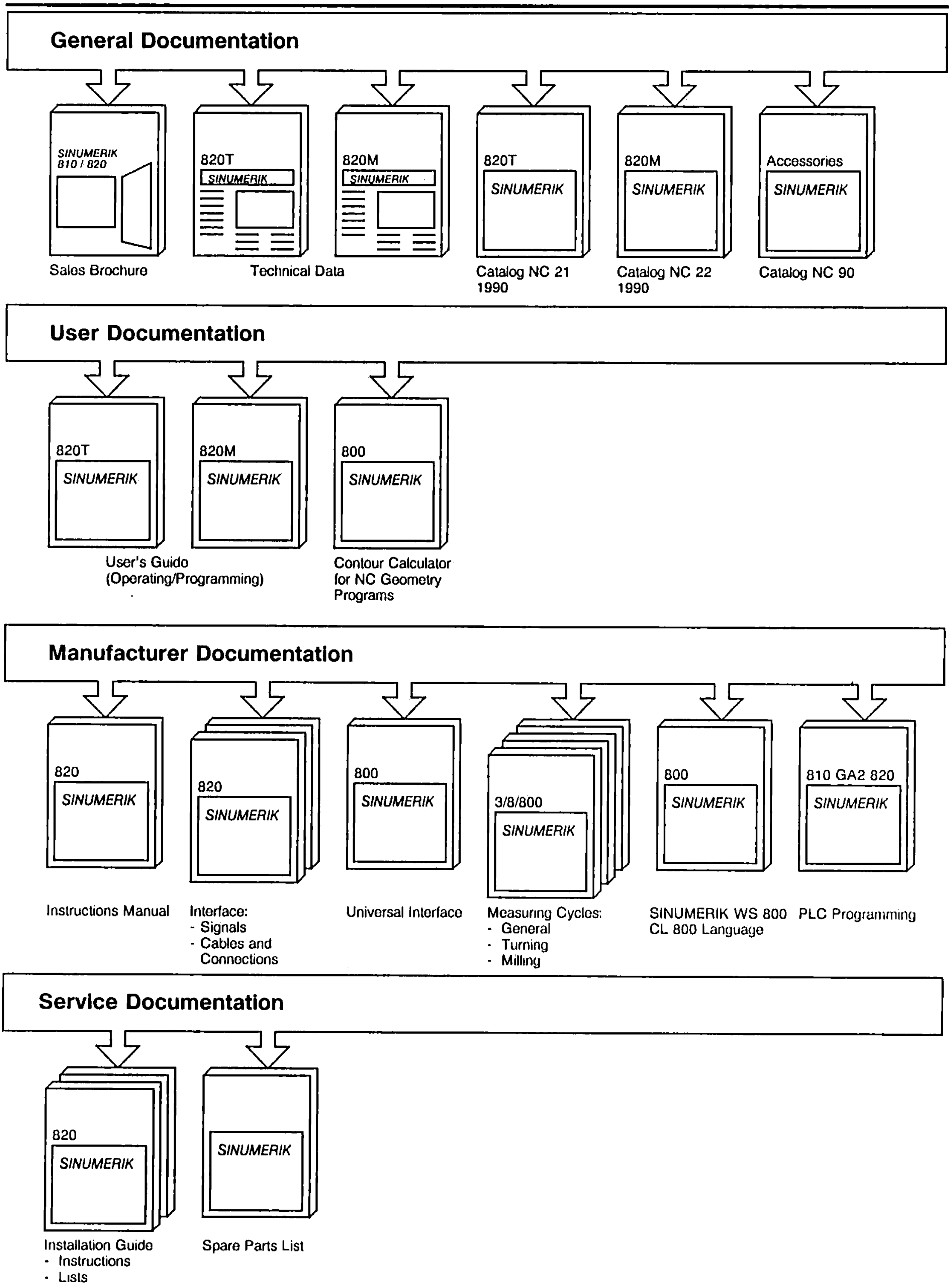
Address ______________________________

Telephone /

Should you come across any printing errors when reading this publication, please notify us on this sheet. Suggestions for improvement are also welcome.

Suggestions and/or corrections

SINUMERIK 820



General Documentation

Sales Brochure

Technical Data

Catalog NC 21
1990

Catalog NC 22
1990

Catalog NC 90

User Documentation

User's Guide
(Operating/Programming)

Contour Calculator
for NC Geometry
Programs

Manufacturer Documentation

Instructions Manual

Interface:
- Signals
- Cables and Connections

Universal Interface

Measuring Cycles:
- General
- Turning
- Milling

SINUMERIK WS 800
CL 800 Language

PLC Programming

Service Documentation

Installation Guide
- Instructions
- Lists

Spare Parts List

Siemens AG
Automation Systems
for Machine Tools, Robots
and Special-Purpose Machines
Postfach 48 48, W-8500 Nuernberg 1
Federal Republic of Germany

Siemens Aktiengesellschaft

Order No.: 6ZB5 410-0BX02-0BA1
Printed in the Federal Republic of Germany
251/211187 906687 PJ 01910.2

progress in automation: Siemens