

Siemens

SIMATIC S5

Programmable Controllers
SIMATIC S5-110S/B

Programming Instructions

SIEMENS

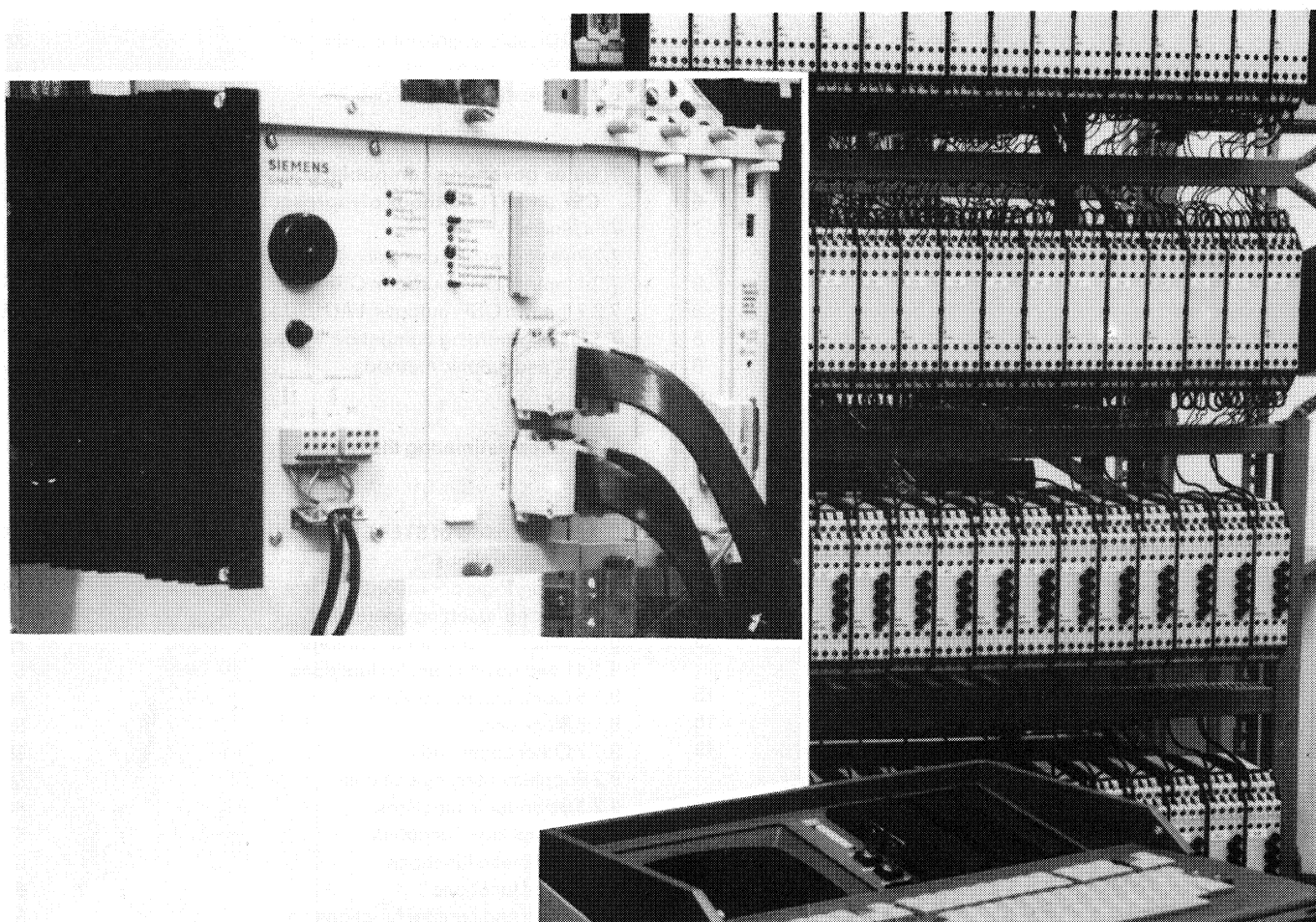
SIMATIC S5-110S/B

Programmable Controllers

6ES5—110

Programming Instructions

Order No.: GWA 4NEB 807 2122-02



S5-110S Programmable controller: CPU (left), peripherals (right) and the 670 programming unit (foreground)

Contents:	Page	Page
1. Introduction	3	
1.1 Application	3	
1.2 STEP 5 programming language	3	
1.3 Programming	3	
1.3.1 Program structure	3	
1.3.2 Program organisation	4	
1.3.3 Program processing	5	
1.4 General notes	5	
2. Program blocks	6	
2.1 Programming program blocks	6	
2.2 Calling program blocks	6	
3. Data blocks	6	
3.1 Programming data blocks	6	
3.2 Calling data blocks	7	
4. Function blocks	8	
4.1 General	8	
4.2 Structure	8	
4.3 Calling function blocks for parameter assignment	8	
4.4 Generation of function blocks	9	
4.5 Standard function blocks	10	
5. Organisational tasks	11	
5.1 General	11	
5.2 Overview	12	
5.3 Programming cyclic processing	12	
5.3.1 Interface between the system program and cyclic processing	12	
5.3.2 Rough organisation of the program	13	
5.4 Programming of interrupt-driven processing (servicing a process interrupt)	15	
5.5 Start-up and restart procedure	18	
5.6 Evaluation of device errors and exception conditions	19	
6. Programming examples	20	
6.1 Basic operations (program and data blocks)	20	
6.1.1 Binary logic functions	20	
6.1.2 Setting/resetting functions	23	
6.1.3 Loading and transfer functions	25	
6.1.4 Timer functions	26	
6.1.5 Counter functions	29	
6.1.6 Comparison functions	31	
6.2 Supplementary operations (function blocks)	34	
6.2.1 Binary logic functions	34	
6.2.2 Digital logic functions	35	
6.2.3 Arithmetic functions		35
6.2.4 Jump functions		36
6.2.5 Timer and counter functions		36
6.2.6 Shift functions		36
6.2.7 Conversion functions		37
6.2.8 Decrementing/incrementing		37
6.2.9 Disable/enable command output		37
6.2.10 Disable/enable interrupts		38
6.2.11 Processing functions		38
6.2.12 Substitution functions		39
7. Rules governing compability between the LAD, CSF and STL methods of representation		42
7.1 General		42
7.2 Rules governing compatibility between the graphic methods		43
7.2.1 Input in LAD, output in CSF (STL)		43
7.2.2 Input in CSF, output in LAD (STL)		43
7.3 Rules governing compatibility between the STL and graphic methods		44
8. Notes on estimating the required memory space		54
9. Total overview of STEP 5 operations		55
9.1 Basic operations		55
9.1.1 Binary logic operations		55
9.1.2 Setting/resetting operations		56
9.1.3 Timer and counter operations		56
9.1.4 Loading and transfer functions		57
9.1.5 Comparison functions		58
9.1.6 Block calls		59
9.1.7 Other commands		59
9.2 Supplementary operations		60
9.2.1 Binary logic functions		60
9.2.2 Digital logic functions		61
9.2.3 Arithmetic functions		61
9.2.4 Jump functions		61
9.2.5 Timer and counter functions		61
9.2.6 Shift functions		62
9.2.7 Conversion functions		62
9.2.8 Decrementing/incrementing		62
9.2.9 Processing function		62
9.2.10 Disable/enable command output		62
9.2.11 Disable/enable interrupts		62
9.2.12 Substitution functions		63

1. Introduction

1.1 Application

The S5–110S is a powerful programmable controller for process automation (logic control, monitoring, listing, signalling). It is suitable both for the simplest control applications with binary signals and for the solution of complicated automation tasks. Its user programs are written in the STEP 5 programming language.

1.2 STEP 5 programming language

The operation set of the STEP 5 programming language makes it possible to program functions ranging from simple binary logic to complex digital processing.

Depending on the programming unit used, the program can be written in three different methods of representation, ladder diagram (LAD), control system flowchart (CSF) and statement list (STL) (Fig. 1), so that the programming method can be adapted to the application. The machine code produced by the programming units is identical for all three methods of representation. If certain programming rules are followed (see "Rules governing compatibility between the LAD, CSF and STL methods of representation", p. 41), the 670/675 programming unit can translate the statement program from one method of representation to another.

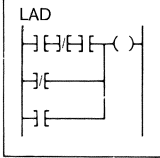
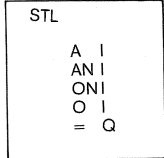
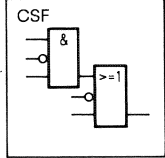
Ladder diagram	Statement list	Control system flowchart
Programming with graphic symbols as in schematic circuit diagram to DIN 19239 (draft)	Programming with mnemonics of the function designations to DIN 19239 (draft)	Programming with graphic symbols to IEC 117-15 DIN 40 700 DIN 40 719 DIN 19 239 (draft)
		

Fig. 1 Methods of representation in the STEP 5 programming language

1.3 Programming

1.3.1 Program structure

The structure of the S5–110S programmable controller compels the user to adopt structured programming techniques, i. e. the program is divided into individual-self-contained program sections (blocks). This method offers the user the following advantages:
 simple and clear programming, even of larger programs,
 standardization of program sections possible,
 simple program organization,
 easy program modification,
 simple program testing,
 simple start up.

Three types of block, each of which has different tasks, can be used to construct the user program:

Program blocks (PB)

These are used to divide the user program into technology-oriented program sections.

Function blocks (FB)

These are used to program complex repetitive functions (e. g. individual open-loop control, alarm and arithmetic functions) and the processing of sequence cascades.

Data blocks (DB)

These are used to store data and text.

The maximum number of programmable blocks is

- 128 program blocks
- 48 function blocks
- 63 data blocks (without DB 0).

No block should exceed 256 statements.

All programmed blocks are stored by the programming unit in the program memory in random order (Fig. 2). An organisation block which is also programmed by the user, determines the sequence in which the blocks are to be processed.

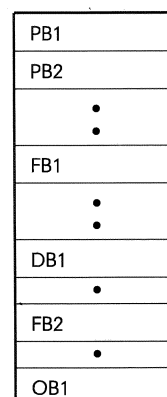


Fig. 2 Location of the blocks in the program memory in random order

1. Introduction

1.3 Programming

1.3.2 Program organisation

1.3.2 Program organisation

The organisation block determines whether and in which sequence the program, function and data blocks are to be processed (Fig. 3). The corresponding calls (conditional or unconditional) for the blocks required are written in the organisation block 1 (OB 1) (see "Organisational tasks", p. 11).

Organisation block 1 is also located in the program memory, as are the other blocks.

The user can program function block 0 to determine his reaction to interrupts (see p. 15).

The program and function blocks can call up further program and function blocks in any desired combination. The maximum permissible nesting depth is 8 blocks, incl. organisation block, but not including any data blocks used (Fig. 4).

OB Organisation block
PB Program block
FB Function block
DB Data block

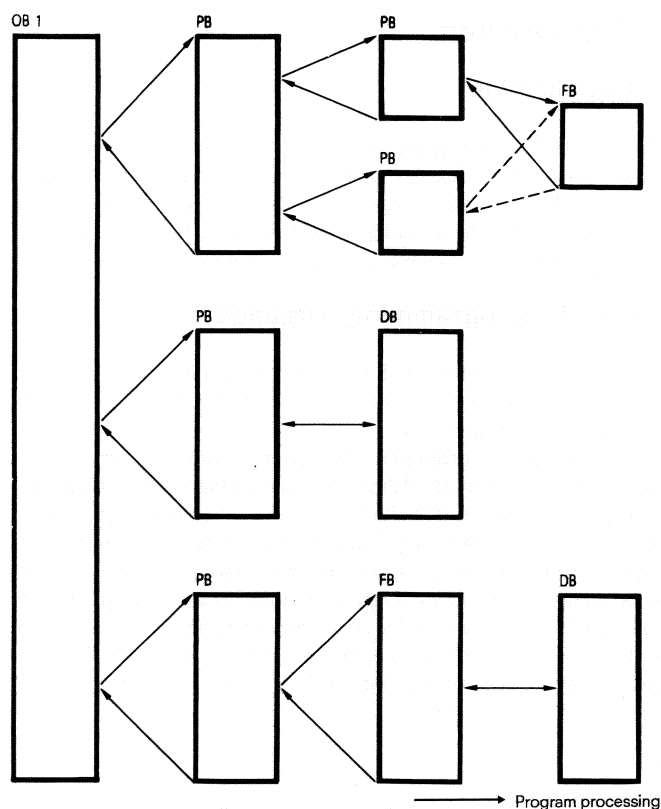


Fig. 3 Program organisation in the STEP 5 programming language

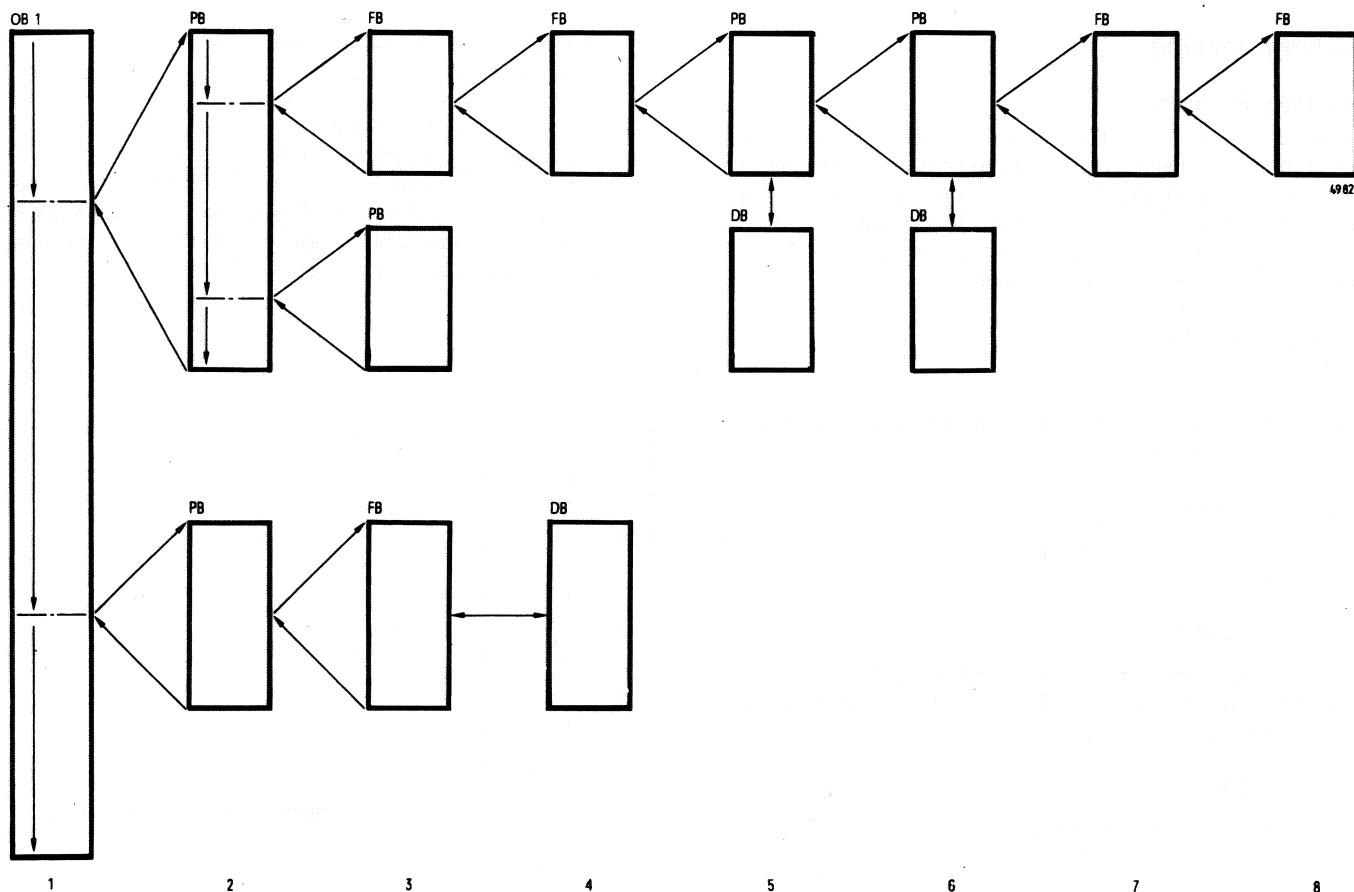


Fig. 4 Example of a program structure taking advantage of the maximum nesting depth

1.3.3 Program processing

The user program can be processed in three different ways (Fig. 5).

Cyclic processing

Organisation block 1 is provided for the cyclic processing of the user program. This block executes cyclically and calls the blocks programmed in the user program. The maximum cycle time is 270 ms. After this, the cycle time monitoring reacts.

Interrupt-driven program processing

With this type of processing, cyclic program processing is interrupted, depending on 32 input signals, at each block change. Function block 0 (FB0) is provided for interrupt processing.

Time-controlled program processing

With this type of program processing, certain program or function blocks are inserted into the cyclic program processing in a freely selectable time grid. The average user program cycle time can be reduced with this programming option.

Time-controlled blocks are not called up automatically, as with the S5 – 150 A/K controller, but are executed as required by the user.

Note:

When programming on the time base of 10 ms, make sure that the processing time of each individual block is < 10 ms since the times are only processed at block boundaries!

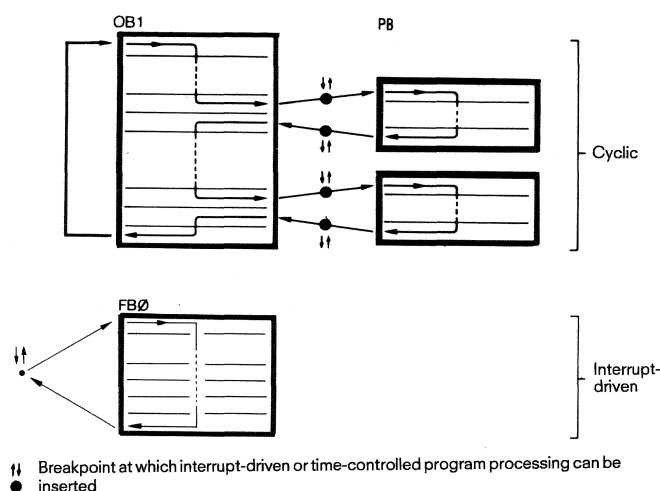
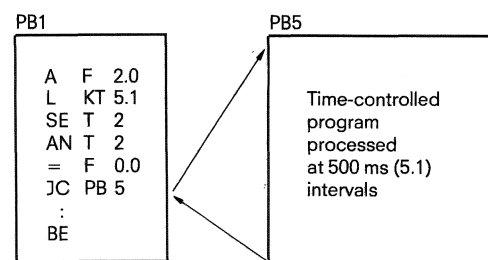


Fig. 5 Types of program processing



Example of time-controlled program processing

1.4 General notes

If standard function blocks are used, the flag bytes 200 to 255 are reserved and are not available to the user.

Timer 0, counter 0 and data block 0 are also reserved.

If the 333 C service unit is used, function block 1 and data block 1 are reserved. Function blocks 2, 3, 4, 5, 6 and data block 2 are reserved for the 512 interface module. Data word 0 of all data blocks cannot be used.

Standard function blocks use block numbers 1 to 12. User function blocks therefore can only use block numbers 13 to 47.

Programming blocks can be programmed in all three methods of representation (STL, LAD, CSF) with the basic operation set of the STEP 5 programming language.

Note:

Programming of data words outside the relevant block can lead to the controller entering an undefined state (Data word No. > data block length).

The following programming errors cause the programmable controller to go into stop state:

1. Programming a STEP 5 operation not in the 110 S controller's operation set.
2. Programming a call for an illegal block (No. too large).
3. Selection of a data word (LDW, TDW) without previous selection of data block (CDB).
4. Exceeding the permissible nesting depth.

2. Program blocks

3. Data blocks

2. Program blocks

2.1 Programming program blocks

The programming of a program block (PB) begins by assigning a block number between 0 and 127 (example: PB 25). This is followed by the actual control program, which ends with the "BE" statement. A program block should not contain more than 256 statements (including "BE") (see Fig. 6). The block header, which the programming unit automatically generates for the program block, takes up five words in the program memory.

A program block should always contain a complete program. Chaining beyond the block limits is not possible.

2.2 Calling program blocks

Processing of a program block is enabled by block calls (Fig. 7) which can be written in organisation, program or function blocks. Block calls can be compared to jumps to a subroutine and can therefore be called either unconditionally (JU PB x) or conditionally, i. e. depending on the result of a logic operation (JC PB x).

After the "BE" statement, control is returned to the block containing the block call. The result of the logic operation can no longer be processed after the block call or after "BE". The result of the logic operation is taken into the "new" block, however, and can be evaluated (see 'Interrupt processing', p. 15).

- Unconditional call:
- The program block addressed is processed independently of the result of the previous logic operation.
- Conditional call:
- The program block addressed is processed dependent on the result of the previous operation.

3. Data blocks

3.1 Programming data blocks

Data blocks (DB) are used to store data required by the user program.

Data may consist of:

- Any desired bit pattern, e.g. for equipment statuses
- Numbers (hexadecimal, binary, decimal), e.g. for timers or calculation results
- Alphanumeric characters, e.g. for message texts.

Data blocks are built up like program blocks. Programming starts by specifying a data block number between 1 and 63 (e.g. DB 25). Each data block can consist of up to 256 data words (16 bits) (Fig. 8). Data must be entered in the ascending order of the data words, beginning with data word 0, whereby data word 0 (DW 0) cannot be used by the user, as it is required as a buffer for certain function blocks.

One memory word is provided per data word in the user memory. A block header occupying a further five words in the user memory is generated by the programming unit for each data block.

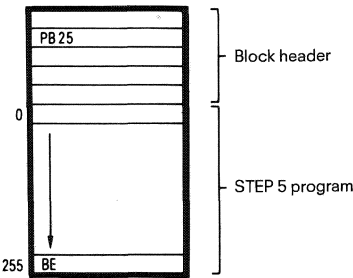


Fig. 6 Structure of a program block

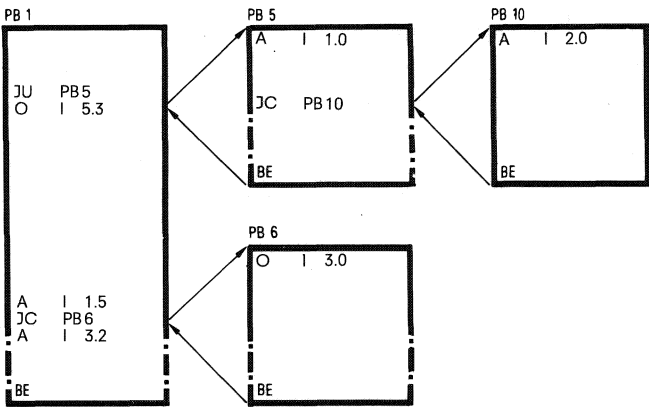


Fig. 7 Block calls which enable program block processing

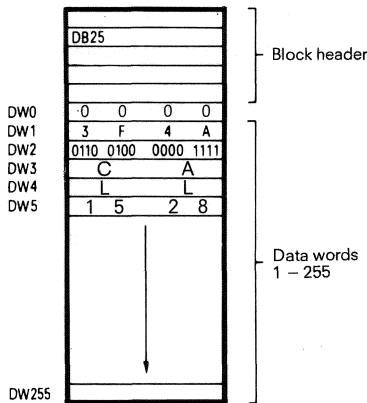


Fig. 8 Structure of a data block

3.2 Calling data blocks

Data blocks (DB) can only be called unconditionally. The call remains valid until a new call is made.

The data block call can be programmed within a program block or function block.

Example:

Transferring the contents of data word 1 of data block DB 10 to data word 1 of data block DB 20 (Fig. 9).

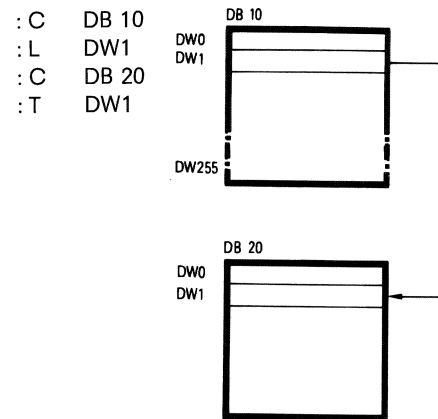


Fig. 9 Calling a data block

If a program block in which a data block has already been addressed calls a further program block and another data block is addressed in this block, this data block is only valid in the program block called. After control is returned to the calling program block, the old data block becomes valid once more (Fig. 10).

Example:

Data block DB 10 is called in program block PB 7. The data in this data block are then processed.

After the call, program block PB 20 is processed. Data block DB 10 is still valid, however. The data area only changes when data block DB 11 is called. Data block DB 11 is now valid up to the end of program block PB 20.

Data block DB 10 becomes valid again when the block changes back to program block PB 7.

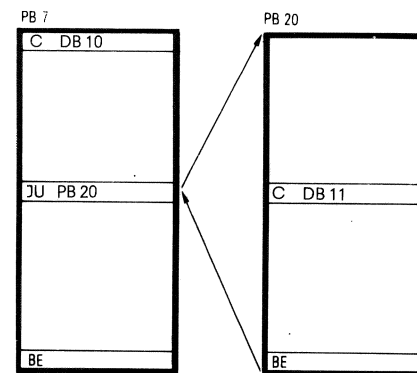


Fig. 10 Calling a data block within another data block

Note:

If a data load or transfer command is programmed with a data word number > block length, this can lead to the controller entering an undefined state.

Example:

Data block 10 is loaded with 10 data words, with the TDW 11 operation a memory word is overwritten **outside the block** (Fig. 11).

The processing of a data load or transfer operation **without previous** data block call causes the controller to stop.

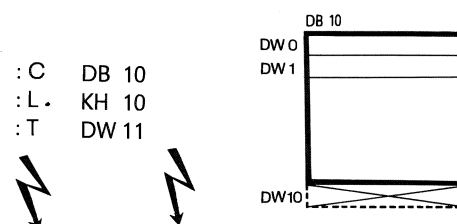


Fig. 11 Fatal programming error

4. Function blocks

5.1 General

4.2 Structure

4.3 Calling function blocks for parameter assignment

4. Function blocks

4.1 General

Like program blocks, function blocks are part of the user program. They differ in four main ways from program blocks:

Function blocks can be assigned parameters, i.e. the actual operands with which a function block is to operate can be specified.

Function blocks can be programmed with an extended operation set compared with program blocks.

The function block program can only be written and documented as a statement list.

A function block call is represented graphically as a black box.

A function block is a complex, complete functional entity within a user program. It can either be obtained from Siemens as a software product ("Standard function blocks" on mini-diskette) or programmed by the user himself. The supplementary operations (see p. 20) can only be programmed in function blocks.

4.2 Structure

A function block consists of a block header and block body (Fig. 12).

Block header

The block header contains all the information required by the programming unit for graphic representation of the function block and for checking the operands when initialising the function block. Before the function block is programmed, this block header is entered by the user (with the aid of the programming unit) (see "Generation of function blocks", p. 9).

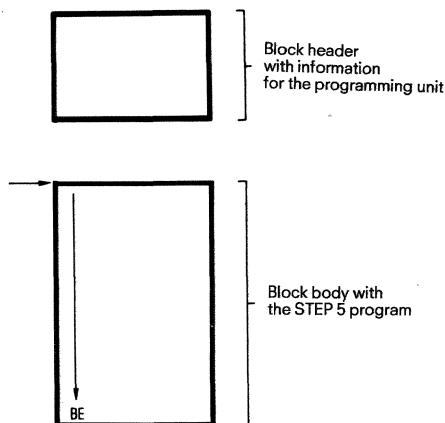


Fig. 12 Structure of a function block

Block body

The block body contains the actual program of the function block. The function to be executed is described by the STEP 5 programming language and deposited in the block body. Only the block body is processed when the function block is called. An extended operation set (compared with the basic operations) is available for programming function blocks (see "Supplementary operations", p. 35).

4.3 Calling function blocks for parameter assignment

Repetitive or very complex functions are implemented by function blocks (FB). They are present only once in the program memory and are called once or several times by a superordinate block. These function blocks can be assigned different parameters each time they are called.

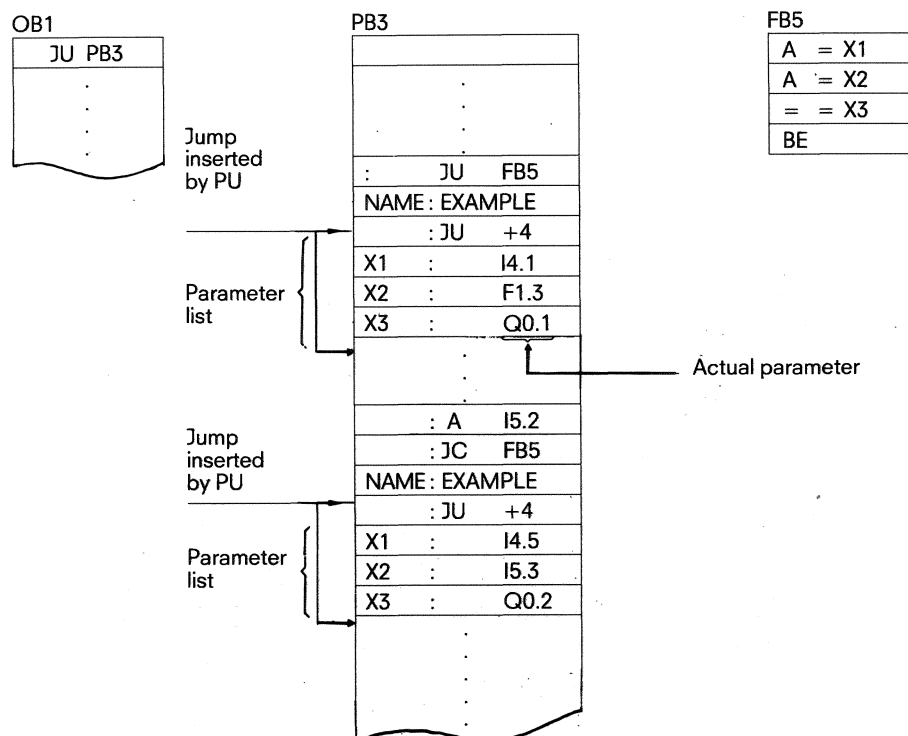


Fig. 13 Parameter assignment of a function block

Like the program and data blocks, the function blocks are located in the program memory under a certain designation (FB 1 to FB 47). User function blocks should be addressed in descending order from FB 47, so as not to collide with the standard function blocks which have the addresses from FB 1 to FB 12.

A function block call can be written within a program block or another function block. The call comprises the call statement and the parameter list.

Call statement

JU FBn unconditional call
JC FBn conditional call

Unconditional call:

The function block addressed is processed regardless of the result of the previous logic operation.

Conditional call:

The function block addressed is only processed if the result of the previous logic operation "RL0" = 1.

Parameter list

The parameter list is situated directly after the call statement (Fig. 13). The input and output variables and data are defined in it (see "Block parameter type"). The parameter list can contain a maximum of **40 variables**.

When processing the function block, the variables from the parameter list are used instead of formal parameters. The sequence of the variables in the parameter list is monitored by the programming unit.

The jump statement after the FB call is automatically inserted by the programming unit, but not displayed when read out.

The FB call occupies two words in the program memory, and each parameter a further memory word.

The memory requirements of the standard function blocks and their runtimes are given in Catalog ST 56.

The qualifiers for inputs and outputs of the function blocks which appear on the programming unit when programming, and the name of the block are stored in the block itself. For this reason, all function blocks required must be transferred to the program mini floppy disk or entered directly into the program memory of the programmable controller before starting to program with the programming unit.

4.4 Generation of function blocks

The function blocks are generated in two parts, corresponding to the structure of a function block:

Entering a) the block header and b) 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

Formal operands (names of the block parameters)

Block parameter type

Block data type.

Library number

A number between 0 and 65535 can be specified. This number is assigned to the function block, regardless of its symbolic or absolute parameters.

A library number should only be specified once to enable a certain function block to be uniquely identified. Standard function blocks have a product number.

Function block name

The name given to the function block can be up to eight characters long. It is different from the system identifier.

Formal operand (name of the block parameter)

The formal operand can be up to four characters long and must start with a letter.

A maximum of 40 parameters can be programmed for each function block.

Block parameter type

"I", "Q", "D", "B", "T" or "C" can be entered as block parameter type.

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

"I, D, B, T" or "C" are parameters which appear on the left of the function symbol in graphic representation. "Q" parameters are shown on the right of the function symbol in graphic representation.

Parameter assignment

Operations (substitution commands) which are to be assigned parameters are programmed in the function block with formal operands. The formal operands can be addressed several times in different parts of the function block.

Example: Program in the function block

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

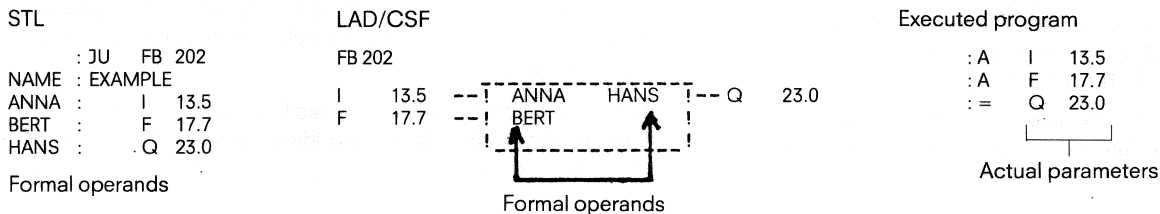
: A = ANNA		
: A = BERT		
: = HANS		

Formal operand	Parameter type	Data type
----------------	----------------	-----------

4. Function blocks

4.5 Standard function blocks

Function block call



Block parameter type and data type with permitted actual parameters

Type of parameter	Type of data	Permitted actual parameters
I, Q	<p>BI for operands with bit addresses</p> <p>BY for operands with byte addresses</p> <p>W for operands with word addresses</p>	<p>I n.m Inputs</p> <p>Q n.m Outputs</p> <p>F n.m Flags</p> <p>IB n Input bytes</p> <p>QB n Output bytes</p> <p>FB n Flag bytes</p> <p>DL n Data bytes left</p> <p>DR n Data bytes right</p> <p>PB n Peripheral bytes</p> <p>IW n Input words</p> <p>AW n Output words</p> <p>FW n Flag words</p> <p>DW n Data words</p> <p>PW n Peripheral words</p>
D	<p>KM for a binary pattern (16 digits)</p> <p>KY for two byte-serial numbers in the range of 0 to 255 each</p> <p>KH for a hexadecimal pattern (max. 4 characters)</p> <p>KS for a symbol (max. 2 alphanumeric characters)</p> <p>KT for a time (in BCD code) with time base 1.0 to 999.3</p> <p>KC for a count (BCD) 0 to 999</p> <p>KF for a fixed-point number in the range from – 32768 to + 32767</p>	Constants
B	No data type permissible	<p>DB n Data blocks; the CDBn command is executed</p> <p>FB n Function blocks (only permitted without parameters) are called unconditionally (JU..n)</p> <p>PB n Program blocks are called unconditionally (JU..n)</p>
T	No data type permissible	T Timer; the time must be assigned parameters as data or must be programmed as a constant in the function block
C	No data type permissible	C Counter; the counter must be assigned parameters as data or must be programmed as a constant in the function block

4.5 Standard function blocks

The following constraints apply when using standard function blocks from the ST56 Catalog with the 110S PC:

The FB30, FB35 and FB36 standard function blocks for the 333C service unit cannot run on the 110S PC. The standard function block specially developed for use with the 110S or 130W PCs must be used.

The standard function blocks for sequence controls (FB70 – FB75) cannot be used on the 110S, as sequence blocks cannot be programmed on the 110S.

For message functions only the function blocks FB50 – FB56 can be used for sending messages to the process peripherals. The FB64 – FB69 function blocks for messages to the standard peripherals cannot be used.

Supplying the standard interface (512C interface module) with function blocks FB120 – FB129 is not possible. The interface modules specially developed for the 110S and 130W PCs must continue to be used.

For closed-loop control with the 110S PC a special software package has been developed.

When using standard function blocks, care must be taken to load blocks with numbers greater than 47 with a block number ≤ 47 , as the 110S has a maximum capacity of only 47 function blocks (FB1 – FB47).

5. Organisational tasks

5.1 General

The full program of a programmable controller consists of the system program and the user program (Fig. 14). The system program is made up of all statements and declarations concerning the internal hardware operating functions (e.g. saving data when the power fails). This program is an integral part of the programmable controller (EPROM) and cannot be changed by the user. The user program consists of all user-programmed statements and declarations for the signal processing affecting the control of the system (process) to be controlled. Organisation block 1 constitutes the interface between the system program and the user program.

Like the program or function blocks, organisation block 1 (OB 1) is also part of the user program. However, organisation block 1 is only called by the system program. A user cannot call organisation block 1. OB 1 controls cyclic processing of the user program.

A programmed reaction of the user to device errors and control of the user program processing mode by further organisation blocks, as is the case with the 150 A/K programmable controller, is not possible.

User program processing mode

- Cyclic processing by programming OB 1 (see p. 12).
- Interrupt-driven processing by programming FB0 (see p. 12 and 15).
- Time-controlled processing programmed directly in the user program (see p. 5).

Cold restart and initial start

Initiated by controls on the central processing module ('Run' switch, reset button).

- Cold restart-manual (see p. 18).
- Cold restart-manual with reset (see p. 18).
- Cold restart-automatic (see p. 18).

Handling device errors

Device errors bring the controller into the stop state.

- Memory errors
- Battery failure on cold restart
- Time-out
- Cycle time exceeded

Exception conditions

Exception conditions bring the controller into the stop state.

- Statement not decodable (p. 19).
- Illegal block (p. 19).
- Data block not available (p. 19).
- Block stack overflow (p. 19).

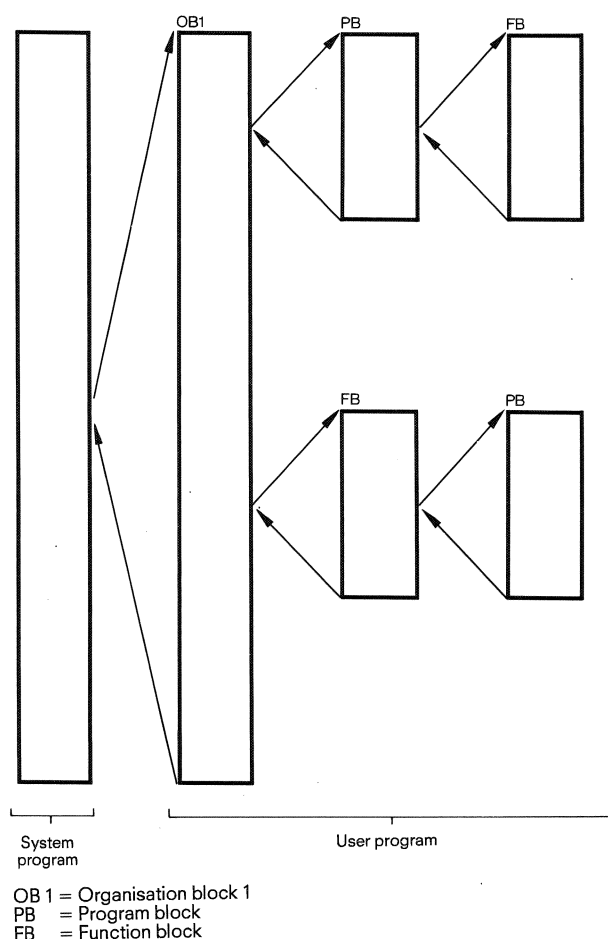


Fig. 14 Entire program of programmable controller

5. Organisational tasks

5.2 Overview

5.3 Programming cyclic processing

5.2 Overview

Processing program	Name or processing initiation	Page
--------------------	-------------------------------	------

OB for cyclic processing

OB 1	Program start	12
------	---------------	----

FB for interrupt-driven processing

FB 0	Signal state change at I0.0 I16.0 I32.0 I48.0 I0.1 I16.1 I32.1 I48.1 I0.2 I16.2 I32.2 I48.2 I0.3 I16.3 I32.3 I48.3 I0.4 I16.4 I32.4 I48.4 I0.5 I16.5 I32.5 I48.5 I0.6 I16.6 I32.6 I48.6 I0.7 I16.7 I32.7 I48.7	15
------	--	----

Time-controlled processing

Defined in user program	select (random) time grid	5
-------------------------	---------------------------	---

Cold restart and initial start

Defined in operating system	Cold restart-manual Cold restart-manual with reset Cold restart-automatic after power failure	18
-----------------------------	---	----

Handling device errors and exception conditions

Processing program	Name or processing initiation	Page
Operating system	Memory error Battery failure on cold restart Time-out Cycle time exceeded Statement not decodable Illegal block Data block not available Block stack overflow	19

5.3 Programming cyclic processing

Cyclic processing is the "normal" processing mode for programmable controllers (Fig. 15). The processor starts program processing at the beginning of the STEP 5 program. It works its way through the STEP 5 statements to the end of the program and starts processing again from the beginning of the program.

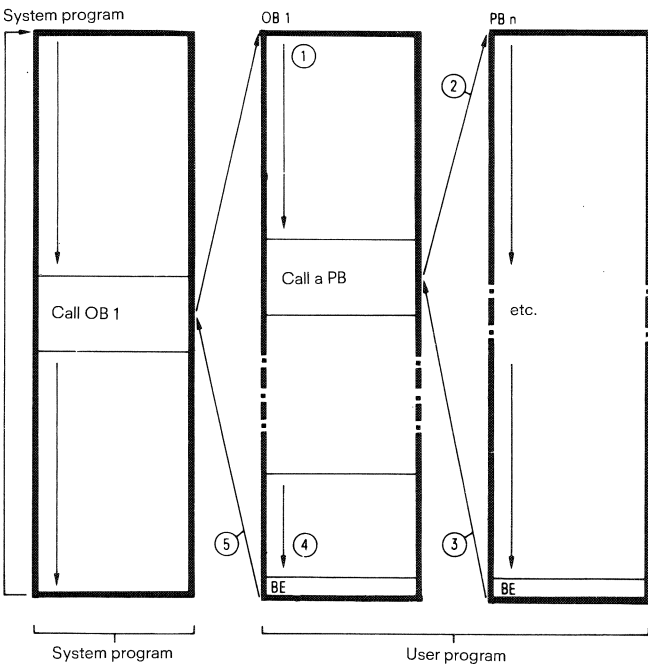
5.3.1 Interface between the system program and cyclic processing

Organisation block 1 is the interface between the system program and the cyclic processing of the user program. The first STEP 5 statement in organisation block 1 is at the same time the first statement of the user program, i.e. it is equivalent to the beginning of the program.

The program and function blocks of the cyclic program are called in organisation block 1. Further block calls can be located in these called-blocks, i.e. the blocks can be nested (see 1.3.2 "Program organisation", p. 4).

The user program runtime is made up of the sum of the runtimes of the blocks called. If a block is called "n" times, its runtime must be taken into account "n" times when working out the total.

The maximum cycle time is 270 ms.



- ① First statement of the STEP 5 program
- ② First program block call. Further calls can be located in this block (see also "Program organisation", p. 4).
- ③ Return from the last program or function block processed.
- ④ The organisation block is terminated with the 'BE' statement.
- ⑤ Return to the system program.

Fig. 15 Cyclic program processing

5.3.2 Rough organisation of the program

Organisation block OB 1 contains a rough arrangement of the user program. The documentation of this block is meant to illustrate the main program structures at a glance (Fig. 16) and their relationship to the process technology (Fig. 17).

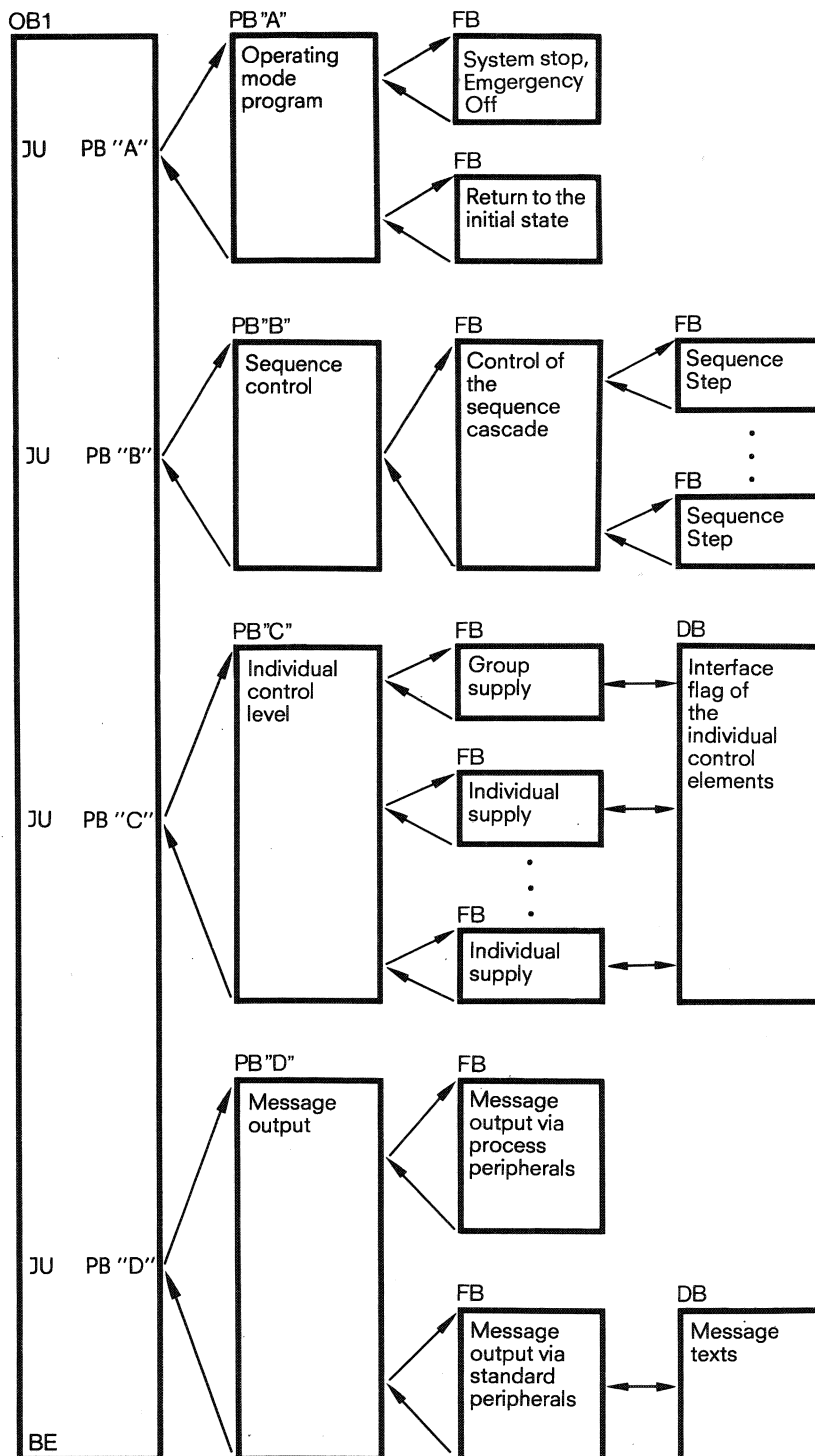


Fig. 16 Organisation of the user program according to the program structure

5. Organisational tasks

5.3 Programming cyclic processing

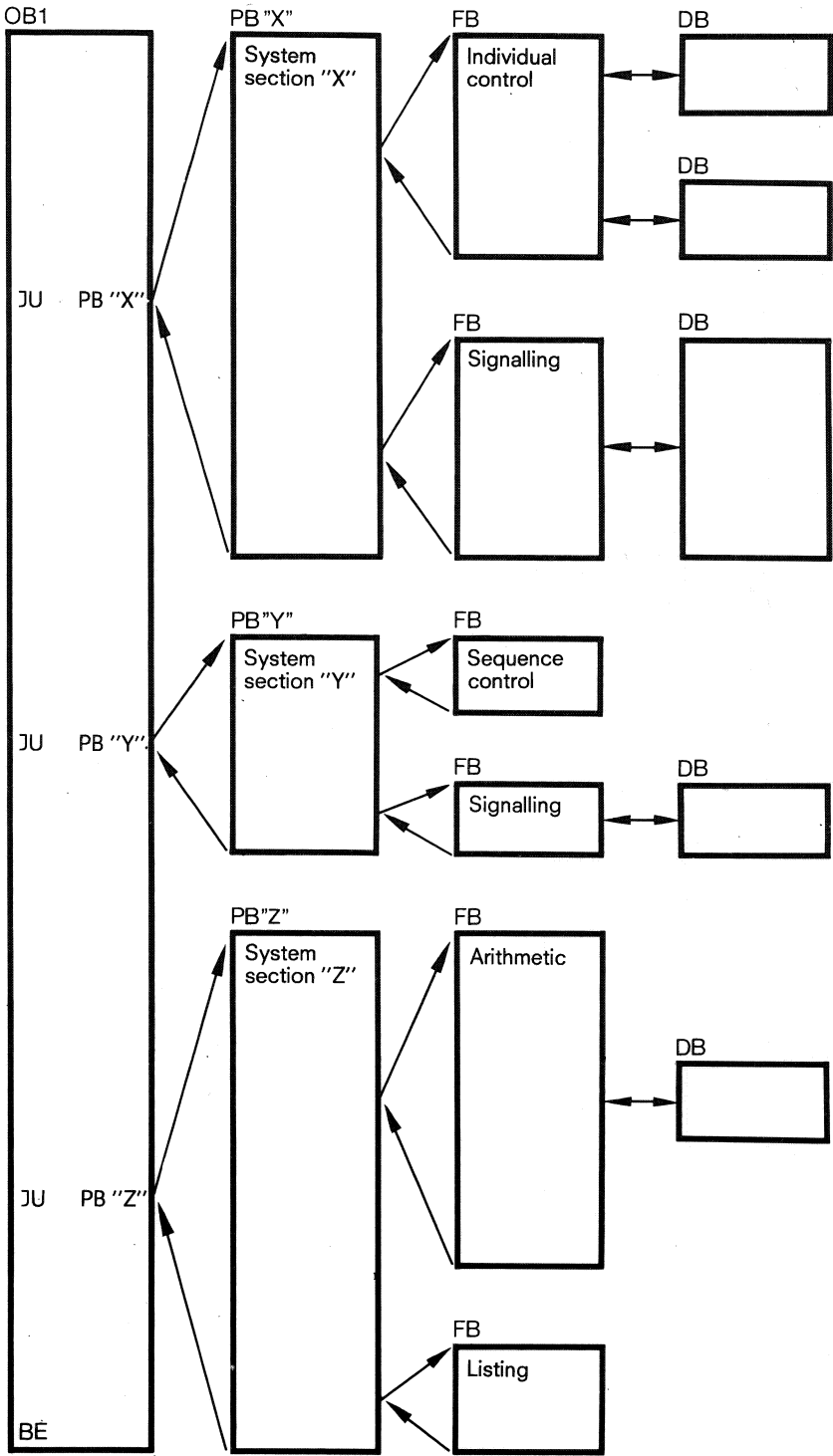


Fig. 17 Organisation of the user program according to process structure

5.4 Programming of interrupt-driven processing (servicing a process interrupt)

Interrupt-driven processing can be implemented with the S5-110S programmable controller. Such processing occurs when a signal from the process causes the processor to interrupt cyclic processing and process a specific program to service the interrupt. After processing this program, the processor returns to the breakpoint in the cyclic program and continues normal processing (see Fig. 19 and 20).

Interface between the system program and interrupt-driven processing

Function block 0 (FB 0) forms the interface between the system program and interrupt-driven processing. With FB 0 the user evaluates the edge change of the interrupt input bytes 0, 16, 32 and 48.

The output bytes 0, 16, 32 and 48 of the 110 peripherals are reserved for reaction to interrupts in the output modules. Minimal reaction time is only assured if these inputs/outputs are used.

User program	Interrupt inputs	Interrupt reaction
FB 0	I 0.0/16.0/32.0/48.0 I 0.1/16.1/32.1/48.1 I 0.2/16.2/32.2/48.2 I 0.3/16.3/32.3/48.3 I 0.4/16.4/32.4/48.4 I 0.5/16.5/32.5/48.5 I 0.6/16.6/32.6/48.6 I 0.7/16.7/32.7/48.7	Q 0.0/16.0/32.0/48.0 Q 0.1/16.1/32.1/48.1 Q 0.2/16.2/32.2/48.2 Q 0.3/16.3/32.3/48.3 Q 0.4/16.4/32.4/48.4 Q 0.5/16.5/32.5/48.5 Q 0.6/16.6/32.6/48.6 Q 0.7/16.7/32.7/48.7

Breakpoints

The cyclic program cannot be interrupted by an interrupt at every point in the processing cycle. This is only possible between blocks (Fig. 18). The system program can only call function block FB 0 for interrupt-driven processing when it is changing from one block to another – e.g. when calling a new block or returning to a superordinate block following a block termination operation.

Assigning priorities to process interrupts

The user stipulates in FB0 the priority of the interrupts if changes in signal states occur simultaneously.

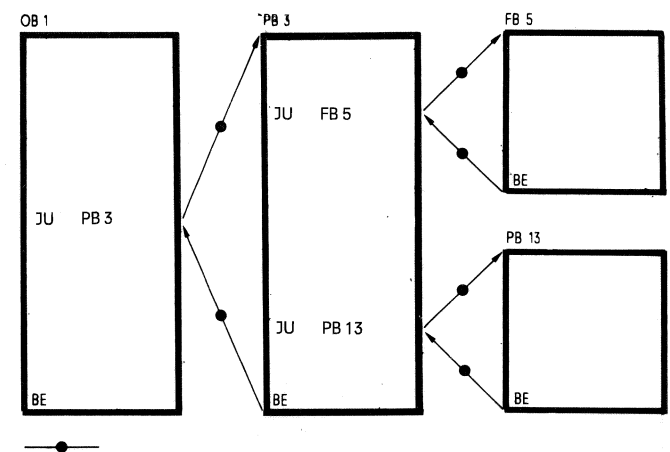
Reaction time

When a module is being processed, no interrupt can be processed, i.e. an interrupt is only processed when a block is called or ended. This means that the maximum reaction time between the occurrence of an interrupt and its being processed corresponds to the processing time of a block, plus interrupt input transfer times and the user program's reaction time to interrupts at the outputs (< 2 ms.)

In order to keep the transfer times of the interrupt inputs and outputs of the 110 peripherals to a minimum, the following must be observed:

1. The interrupt input modules of the 110 peripherals must be plugged into locations 0, 16, 32 or 48.
2. The user references the interrupt outputs 0, 16, 32 or 48 on reacting to interrupts in FB0.
3. Interrupt inputs and outputs are addressed in FB0 with the LPB, LPW, TPB and TPW load and transfer statements.

Further block calls in FB0 delay interrupt reaction time on the interrupt output side.



Breakpoints at which an interrupt can be serviced.

Fig. 18 Breakpoints in the cyclic program

5. Organisational tasks

5.4 Programming of interrupt-driven processing

Example: Interrupt-driven processing

Legend to Figs. 19 and 20

- ① Start of cyclic processing. The system program calls organisation block OB 1.
- ② An interrupt occurs at input I 0.3, the signal state of which changes from "0" to "1".
- ③ Block change. The signal state at input I 0.3 is registered and evaluated. Processing of the cyclic program is interrupted.
- ④ The system program calls function block FB 0. The program of this function block is processed until the BE statement (block end) is reached. The processor then returns to cyclic program processing.
- ⑤ As there is no further interrupt, processing of the cyclic program continues from the point of interruption.
- ⑥ An interrupt occurs at input I 0.6, the signal state of which changes from "0" to "1".
- ⑦ An interrupt occurs at input I 0.0, the signal state of which changes from "0" to "1".
- ⑧ Block change. The change of signal states at inputs I 0.6 and I 0.0 are registered. Processing of the cyclic program is interrupted.
- ⑨ The system program calls function block FB 0. Here the change of signal states at inputs I 0.6 and I 0.0 are evaluated.
- ⑩ An interrupt occurs at input I 0.4, the signal state of which changes from "0" to "1".
- ⑪ The change of signal state at input I 0.4 is registered. Cyclic program processing remains interrupted.
- ⑫ The system program again calls function block FB 0. In FB 0 the change of signal state at input I 0.4 is evaluated.
- ⑬ As there is no further interrupt, processing of the cyclic program continues from the point of interruption.

Disabling interrupt-driven processing

An interrupt-driven program is "inserted" into the cyclic program at a block boundary. The cyclic program is interrupted at this point. This can be disadvantageous if a cyclic program has to be processed within a certain time to achieve, for example, a certain reaction time.

If a program section is not to be interrupted by interrupt-driven processing, the following possibilities are available:

The program contains no block changes, and cannot therefore be interrupted.

The program itself is within an interrupt-driven program. Even on block change it cannot be interrupted by a further interrupt.

The I A "Disable interrupt" operation can be programmed. This can be revoked with the RA ("reset interrupt") (only possible in function blocks: see "Supplementary operations", p. 38). Between the I A and RA operations no interrupt-driven processing is possible (time-controlled processing is not inhibited, however.).

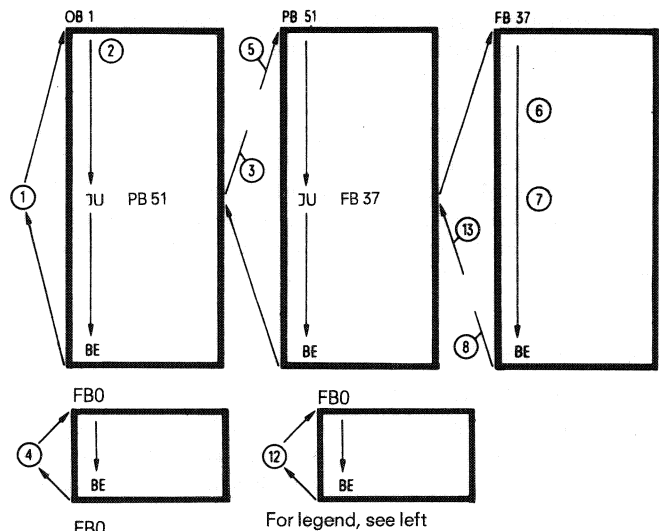


Fig. 19 Interrupt-driven processing with several interrupts

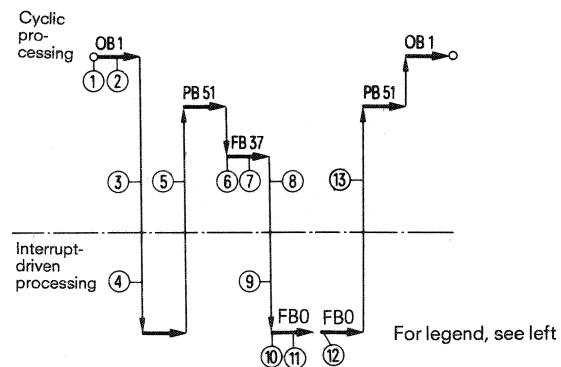


Fig. 20 Processing sequence of the blocks in the above example

Starting interrupt-driven processing

Input modules of the 110 peripherals with group interrupt are used to input interrupt signals. These input modules should be plugged into locations 0 and/or 16/32/48.

Programming function block 0

When an interrupt occurs, function block 0 is called. The user must scan the individual inputs in this block and perform the necessary logic operations. The results of the logic operations at the outputs must be transferred to output modules 0 and/or 16/32/48 (STEP 5 operations TPB/TPW).

If further blocks are called in FB 0, the reaction to the interrupt at outputs 0/16/32/48 is delayed.

In order to be able to detect short interrupt pulses (e.g. counting pulses), the interrupt inputs should be examined at definite intervals by the interrupt service program (FBO) for pulse edge changes. This can be achieved with the following statement sequence, for example (see p. 17):

5. Organisational tasks

5.4 Programming of interrupt-driven processing

Example: Registering interrupt pulses in FB 0 interrupt block

Comments	Statements	Explanation
Scanning of peripheral inputs 0.0 to 0.7 for edge changes. If an input has an edge change, a branch is made to the respective label.	⋮	
	:L PB 0	Load current peripheral byte into accu 1
	:T FB 2	Contents of accu 1 are transferred to flag byte 2 (current PB 0)
	:L FB 1	Flag byte 1 (old PB 0) is loaded into accu 1. The contents of accu 1 (current PB 0) transferred to accu 2
	:XOW	Bit pattern of accu 1 is compared with that of accu 2 and the result stored in accu 1
	:T FB 0	Contents of accu 1 (result) are transferred to flag byte 0
	:L FB 2	Load flag byte 2 (current PB 0) into accu 1
	:T FB 1	Transfer contents of accu 1 (current PB 0) to flag byte 1
	:T IB 0	Transfer contents of accu 1 (current PB 0) to the process image of input byte 0
	:A F 0.0	Scan flag bit 0 of flag byte 0 for log "1"
	:JC =I 00	If flag bit 0.0 is log "1" (RLO = 1), a jump is made to label I 00
	:A F 0.1	Scan flag bit 0.1 of flag byte 0 for log "1"
	:JC =I 01	If flag bit 0.1 is log "1" (RLO = 1), a jump is made to label I 01
	⋮	Processing of flag byte 0
	:JC =I 07	see above
	⋮	Processing of further interrupt-driven statements
	:L PB 0	Load current peripheral byte 0 into accu 1
	:T FB 2	Contents of accu 1 (current PB 0) are transferred to flag byte 2
	:L FB 1	Flag byte 1 (old PB 0) is loaded into accu 1: Contents of accu 1 (current PB 0) shifted to accu 2
Repeat scan of peripheral inputs 0.0 to 0.7 for edge changes.	:XOW	Bit pattern of accu 1 compared with that of accu 2 and the result stored in accu 1.
	:T FB 3	Contents of accu 1 (result) transferred to flag byte 3
	:L FB 2	Load flag byte 2 (current PB 0) into accu 1
	:T FB 1	Transfer contents of accu 1 (current PB 0) into flag byte 1
	⋮	Processing of further interrupt-driven statements
	:L FB 3	Load flag byte 3 (result) into accu 1
	:T FB 0	Contents of accu 1 (result) are transferred to flag byte 0
	:L KF 0	Fixed-point constant 0 is loaded into accu 1: contents of accu 1 (result) shifted into accu 2
	:I=F	Accu 1 compared with accu 1, if equal; RLO = 1 / if not, RLO = 0
	:S F 4.0	If accu are equal (RLO = 1) flag 4.0 is set
If, on repeated scanning of the peripheral inputs, an edge change becomes apparent, a branch is made back to the beginning of FB 0	:AN F 4.0	Scan flag bit 4.0 for log "0"
	:JC FB 0	If flag bit 4.0 is log "0" (RLO = 1), a jump is made to the beginning of function block 0
	:L KF 0	Fixed-point constant 0 is loaded into accu 1
	:T FB 3	Contents of accu 1 (fixed-point constant 0) are transferred to flag byte 3
	⋮	Processing of further interrupt-driven statements
	I 00 :	Processing of statements if flag bit 0.0 is set
	⋮	
	I 07 :	Processing of statements if flag bit 0.7 is set
	:BE	End of program
Processing of edge change		

5. Organisational tasks

5.4 Programming of interrupt-driven processing

5.5 Start-up and restart procedure

Example of interrupt processing

Exact positioning with a limit switch

Functional description:

An output is switched on via an interlock condition I 1.0 or I 2.0 and is switched off again with a short, constant delay when a limit switch I 0.0 has been operated (Fig. 21).

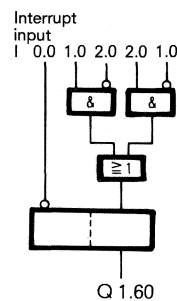
Programming:

The setting condition for the output is programmed in program block 2.

The reset input is assigned to interrupt input I 0.0. FB 0 is called according to the setting on the input module with group interrupt 0 with positive-going or negative-going edge of the signal on input 0.0. Only one input I 0.0 is connected to input byte 0.

Programming FB 0:

In order to scan the status of input 0.0, the input image of byte "0" must first be updated. This is done with the operation LPB and TIB. If there is a negative-going edge at interrupt input I 0.0 and if output Q 16.0 is reset, the output is transferred directly to the output module. This is done by the LQB and TPB operations. When transferring to peripheral bytes 0 to 63, the output process image is automatically updated.



Programming:

FB 0		PB 2	
: L	PB 0	: A	I 1.0
: T	IB 0	: AN	I 2.0
: AN	I 0.0	: O	
: R	Q 16.0	: A	I 2.0
: L	QB 16	: AN	I 1.0
: T	PB 16	: S	Q 16.0
: BE			

Fig. 21 Example of interrupt processing

5.5 Start-up and restart procedure

The system program differentiates between three different start and restart modes for the programmable controller:

- manual restart
- restart with reset
- automatic restart

The type of restart is controlled by the operating system and the user can only influence this by means of the "Reset" button.

Manual restart

A restart is initiated manually by moving the "Stop" switch on the CPU from the "Stop" position to the "Run" position.

The system program then executes the following:

- resets the non-retentive flags (F 128.0 – F 255.7)
- loads the input process image
- erases the output process image
- resets all peripheral outputs
- establishes the block address list

Manual restart with reset

A restart with reset is initiated manually by pressing the "Reset" button and simultaneously moving the stop switch on the CPU from the "Stop" position to the "Run" position.

The system program then executes the following:

- erases all current timer values
- erases all current counter values
- resets all flags
- loads the input process image
- erases the output process image
- resets all peripheral outputs
- establishes the block address list

Automatic restart

The programmable controller tries to execute an automatic restart when the power returns after a power failure. The function of the automatic restart is identical to that of the manual.

If the user does not wish to have an automatic restart on return of power after a power failure he can check a non-retentive flag at the beginning of OB 1 and bring the system into the stop state with the STEP 5 operation "STP".

5.6 Evaluation of device errors and exception conditions

The system program is capable of detecting maloperation of the central processor, errors in the system program or the effects of incorrect programming by the user. With some of these errors, proper operation of the central processor is no longer guaranteed. The programmable controller then stops.

The reaction to device errors and exception conditions is defined in the system. The following events are evaluated:

Memory error

Battery failure (on restart)

Time-out when accessing memory

Cycle time exceeded

Statement not decodable

Illegal block

Non-existent data block

Block stack overflow.

Memory error

In the event of a restart the system program recognises wrongly addressed user memory submodules and stops (erroneous jumper assignment).

In addition, the operating system is checked at each restart. If an error is recognised by the operating system, the PC stops.

If the "Compress memory" function (implemented by the PC for the programming unit) is interrupted by a power failure, for example, the PC stops.

Battery failure

If the battery fails on restart (battery voltage below minimum permissible level), the PC stops.

Time-out

A time-out occurs when a non-existent memory area is addressed. The cause of the time-out can be a fault in the module or the removal of the module during operation of the programmable controller.

Cycle time exceeded

The cycle time consists of the total processing time of a cyclic program. Included in this are the calling and processing of organisation block 1 and the program and function blocks called in this organisation block, with nesting, as well as all the interrupt-driven and timer-driven program sections processed in this cycle.

The cyclic program terminates with a "Block end" statement in organisation block 1 (OB 1). If the processing time exceeds a certain length (the "Cycle time" preset in the processor), the system program recognises a "Cycle time exceeded" error.

This can be caused, for example, by incorrect programming when, under certain process conditions, the processor runs in a program loop or on failure of the system clock.

If the cycle time is exceeded, the system program interrupts the STEP 5 program and stops.

Statement not decodable

If the PC processes a STEP 5 statement which is not part of the set of the SIMATIC S5-110S programmable controller, it stops.

Illegal block

If there is a block call statement in the user program which has a block number higher than the maximum permitted for the PC (127 with PBs, 63 with DBs and 47 with FBs), the controller stops when processing the statement.

Non-existent data block

If a data-word load or transfer statement is processed in the user program without a matching data block having previously been called (address range assignment for DB is missing), the PC stops.

Block stack overflow

If more than 7 blocks are called in a row in the user program without a BE statement being run through, the maximum block nesting depth of 8 is exceeded and the PC stops.

6. Programming examples

6.1 Basic operations (program and data blocks)

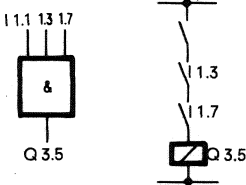
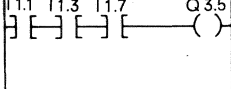
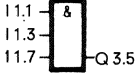
6.1.1 Binary logic functions

6. Programming examples

6.1 Basic operations (Program and data blocks)

6.1.1 Binary logic functions

AND logic

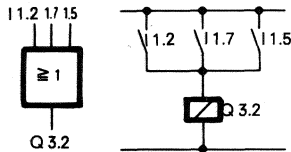
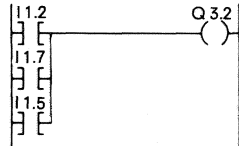
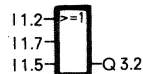
Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 1.1 A I 1.3 A I 1.7 = Q 3.5</pre>		

A "1" signal appears at output Q 3.5 when all inputs have "1" signals simultaneously.

A "0" signal appears at output Q 3.5 if at least one of the inputs has a "0" signal.

There are no restrictions imposed on the number of scans and the programming sequence.

OR logic

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>O I 1.2 O I 1.7 O I 1.5 = Q 3.2</pre>		

A "1" signal appears at output Q 3.2 if at least one of the inputs has a "1" signal.

A "0" signal appears at output Q 3.2 if all the inputs have "0" signals simultaneously.

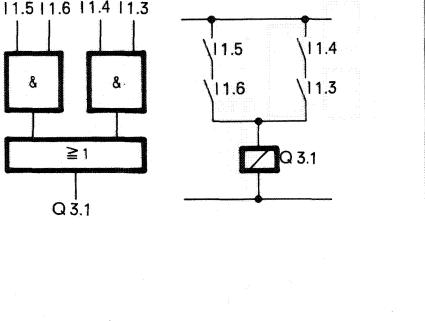
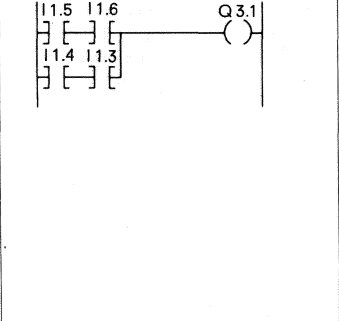
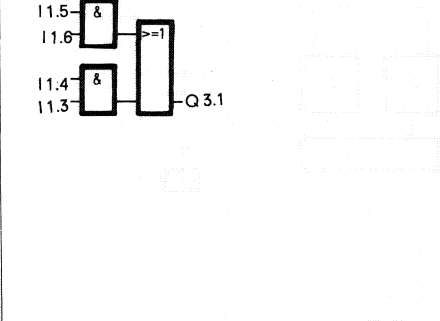
There are no restrictions imposed on the number of scans and the programming sequence.

6. Programming examples

6.1 Basic operations

6.1.1 Binary logic functions

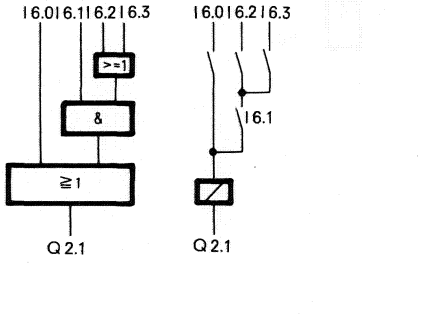
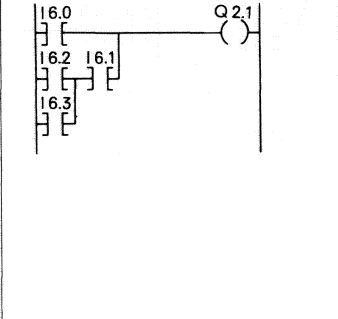
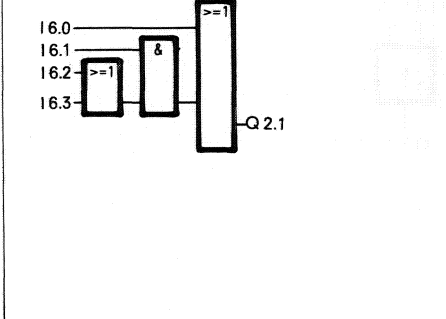
AND before OR logic

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A 1.5 A 1.6 O A 1.4 A 1.3 = Q 3.1 </pre>		

A "1" signal appears at output Q 3.1 when the output of at least one of the AND gates is "1".

A "0" signal appears at output Q 3.1 when neither of the AND gates has a "1" at its output.

OR before AND logic

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> O 6.0 O 6.1 A O 6.2 O 6.3) = Q 2.1 </pre>		

A "1" signal appears at output Q 2.1 when input I 6.0 or input I 6.1 and one of the inputs I 6.2 or I 6.3 have a "1" signal.

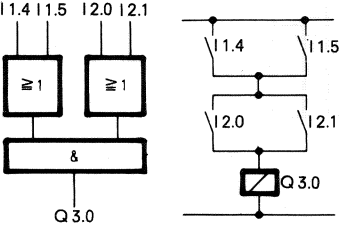
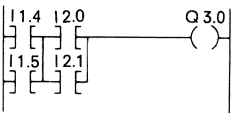
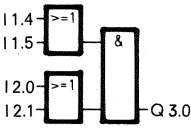
A "0" signal appears at output Q 2.1 when input I 6.0 has a "0" signal and the AND gate has a "0" at its output.

6. Programming examples

6.1 Basic operations

6.1.1 Binary logic functions

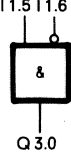
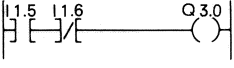
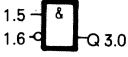
OR before AND logic

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A(O 1.4 O 1.5) A(O 2.0 O 2.1) = Q 3.0</pre>		

A "1" signal appears at output Q 3.0 when both OR gates have "1" signals at their outputs.

A "0" signal appears at output Q 3.0 when at least one of the OR gates has a "0" signal at its output.

Scanning for "0" signal status

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A 1.5 AN 1.6 = Q 3.0</pre>		

A "1" signal only appears at output Q 3.0 when input 1.5 has "1" signal, and the input 1.6 has "0" signal.

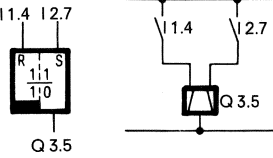
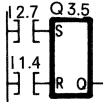
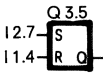
6. Programming examples

6.1 Basic operations

6.1.2 Setting/resetting functions

6.1.2 Setting/resetting functions

RS flip-flop for stored signal output

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 2.7 S Q 3.5 A I 1.4 R Q 3.5 NOP 0 </pre>		

The flip-flop is set when a "1" signal is applied to input I 2.7.

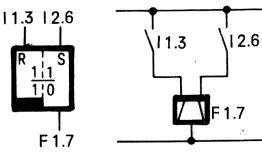
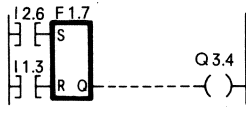
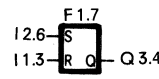
If the signal at input I 2.7 changes to "0", the status remains unchanged, i.e. the signal is stored.

The flip-flop is reset when a "1" signal is applied to I 1.4.

If the signal at input I 1.4 changes to "0" this status is still retained.

The last program scanning operation (in this case A I 4.4) is effective during the processing of the remaining program, if a set signal (input I 2.7), and a reset signal (input I 1.4) are simultaneously applied.

RS flip-flop with flags

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 2.6 S F 1.7 A I 1.3 R F 1.7 A F 1.7 = Q 3.4 </pre>		

The flip-flop is set when a "1" signal is applied to input I 2.6.

If signal at input I 2.6 changes to "0", the status remains unchanged, i.e. the signal is stored.

The flip-flop is reset when a "1" signal is applied to reset input I 1.3.

If the signal at input I 1.3 changes to "0", this status is still retained.

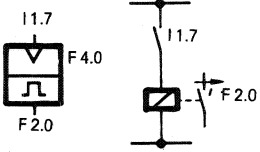
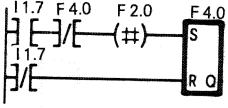
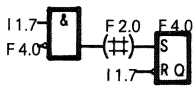
The last program scanning operation (in this case A I 1.3) is effective during the processing of the remaining program, if a set signal (input I 2.6) and a reset signal (I 1.3) are simultaneously applied.

6. Programming examples

6.1 Basic operations

6.1.2 Setting/resetting functions

Implementation of an impulse contact

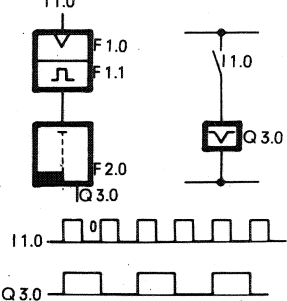
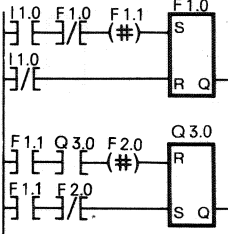
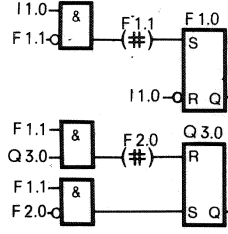
Original	STEP 5 representation		Control system flowchart
Statement list	Ladder diagram		
	<pre> A I 1.7 ANF 4.0 = F 2.0 A F 2.0 S F 4.0 ANI 1.7 R F 4.0 NOP 0 </pre>		

The AND logic (A I 1.7 and AN F 4.0) is fulfilled at each positive-going edge of the signal at input I 1.7 and flags F 4.0 and F 2.0 ("pulse edge flags") are set if the result of the logic operation is "1".

The AND logic A I 1.7 and AN F 4.0 is no longer fulfilled at the next program pass, as flag F 4.0 is set.

Flag F 2.0 is reset. This means that flag F 2.0 only has a "1" signal for one program cycle.

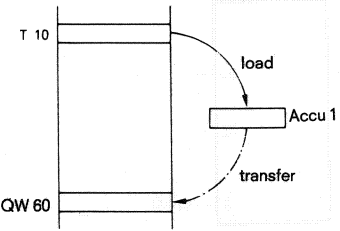
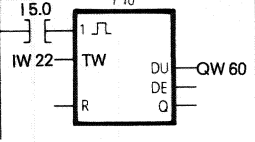
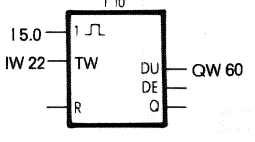
Binary scaler

Original	STEP 5 representation		Control system flowchart
Statement list	Ladder diagram		
	<pre> A I 1.0 ANF 1.0 = F 1.1 A F 1.1 S F 1.0 ANI 1.0 R F 1.0 NOP 0 A F 1.1 A Q 3.0 = F 2.0 A F 2.0 R A 3.0 A F 1.1 ANF 2.0 S Q 3.0 NOP 0 </pre>		

The output of the binary scaler (output Q 3.0) changes its state at each positive-going edge of the signal at input I 1.0, i.e. when input I 1.0 changes from a "0" to a "1". Thus, half the input frequency appears at the binary scaler output.

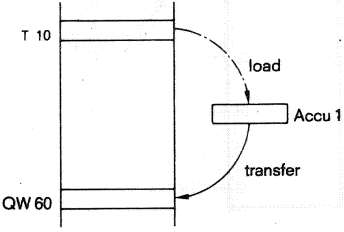
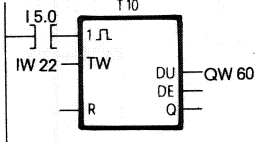
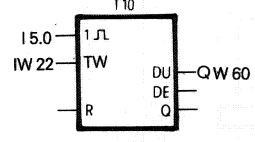
6.1.3 Loading and transfer functions

Loading a timer value

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 5.0 L IW 22 SP T 10 L T 10</pre>		

The contents of the memory location addressed by the T10 statement are loaded into accumulator 1.

Transferring

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 5.0 L IW 22 SP T 10 L T 10 T QW 60</pre>		

The contents of accumulator 1 are transferred into the process image addressed by the QW 60 statement. Transfer of time T 10 to QW 60 is in binary in this example.

6. Programming examples

6.1 Basic operations

6.1.4 Timer functions

6.1.4 Timer functions

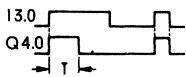
Pulse

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 3.0 L KT 10.2 SP T 1 A T 1 = Q 4.0</pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1". The timer remains unchanged during subsequent processing, resulting in a "1" signal.

The timer is set to zero (reset) when the result of the logic operation is "0".

The AT or OT scans result is a "1" signal as long as the time is running.



$KT\ 10.2 \triangleq 10 \cdot 1s = 10s$

The timer is loaded with the specified value (10). The number to the right of the point indicates the time base:

$0 \triangleq 0.01\ s \quad 2 \triangleq 1\ s$

$1 \triangleq 0.1\ s \quad 3 \triangleq 10\ s$

Outputs DU and DE are digital. The timer value is in binary code at output DU and in BCD code at output DE.

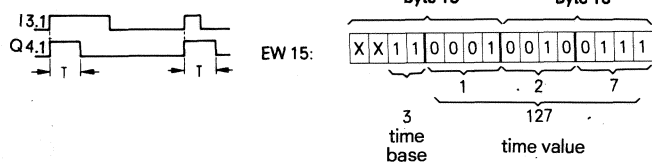
Extended pulse

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 3.1 L IW 15 SE T 2 A T 2 = Q 4.1</pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1".

The timer remains unchanged if the result of the logic operation is "0".

The AT or OT scans result in a "1" signal as long as the time is running.



Set the time value with the BCD-value of the operands I, Q, F or D (input word IW 15 in the above example).

IW 15:

In the above example the time T of the extended pulse is 1270 s, controlled by input word 15 (IW 15).

$EW\ 15 \triangleq 127 \cdot 10\ s = 1270\ s$

Time base

$0 \triangleq 0.01\ s \quad 2 \triangleq 1\ s$

$1 \triangleq 0.1\ s \quad 3 \triangleq 10\ s$

6. Programming examples

6.1 Basic operations

6.1.4 Timer functions

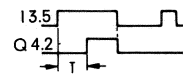
"On" delay timer

Original	STEP 5 representation	Ladder diagram	Control system flowchart
	<pre> Statement list A I 3.5 L KT9.2 SR T 3 A T 3 = Q 4.2 </pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1". The time remains unchanged during subsequent processing if the result of the logic operation remains "1".

The timer is set to zero (reset) if the result of the logic operation is "0".

The AT or OT scans result in a "1" signal when the time has elapsed and the result of the logic operation is still present at the input.



KT 9.2:

Load the timer with the specified value (9).

The number to the right of the point indicates the time base

0 \triangleq 0.01 s 2 \triangleq 1 s

1 \triangleq 0.1 s 3 \triangleq 10 s

"Off" delay

Original	STEP 5 representation	Ladder diagram	Control system flowchart
	<pre> Statement list ANI 3.4 L FW 13 SF T 5 A T 5 = Q 4.4 </pre>		

The timer is started during the first processing cycle if the result of the logic operation is "0". The timer remains unchanged during subsequent processing if the result of the logic operation remains "0".

The timer is set to zero (reset) when the result of the logic operation is "1".

The AT or OT scans result in a "1" signal when the timer is running or the result of the logic operation is still present at the input.

Set the time value with the BCD value of the operand I, Q, F or D (flag word FW 13 in the above example).

FW 13:

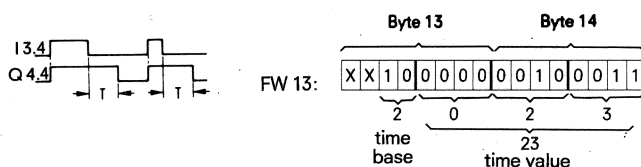
In the above example, the time T of the "off" delay is 23 s, and is determined by flag word 13 (FW 13).

FW 13 \triangleq 23 · 1 s = 23 s

Time base

0 \triangleq 0.01 s 2 \triangleq 1 s

1 \triangleq 0.1 s 3 \triangleq 10 s



6. Programming examples

6.1 Basic operations

6.1.4 Timer functions

Stored "On" delay

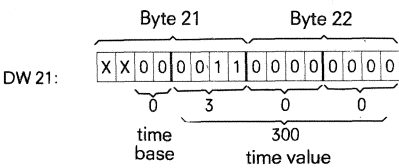
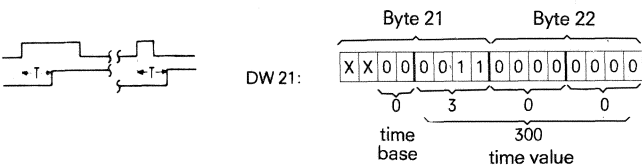
Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>A I 3.3 L DW 21 SS T 4 A I 3.2 R T 4 = Q 4.3</pre>		

The timer is started during the first processing cycle if the result of the logic operation is "1".

The timer remains unchanged if the result of the logic operation "0".

The AT or OT scans result in a "1" signal when the time has elapsed.

The signal status only changes to "0" when the timer is reset by the RT function.



Set the time value with the BCD value of the operands I, Q, F or D (DW 21 in the above example).

In the above example the time T of the "on" delay is 3 s, and is determined by data word 21 (DW 21)

$DW\ 21 \triangleq 300 \cdot 0.01\ s = 3\ s$

Time base

$0 \triangleq 0.01\ s$ $2 \triangleq 1\ s$

$1 \triangleq 0.1\ s$ $3 \triangleq 10\ s$

6. Programming examples

6.1 Basic operations

6.1.5 Counter functions

6.1.5 Counter functions

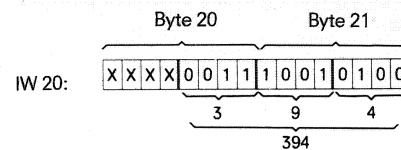
Set counter

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.1 L IW 20 S C 1 </pre>		

The counter is set during the first processing cycle if the result of the logic operation is "1". The counter remains unchanged during subsequent processing (no matter whether the result of the logic operation is "1" or "0"). The counter is set again (pulse edge evaluation) at the next first processing cycle if the result of the logic operation is "1".

The flag necessary for pulse edge evaluation of the set input is included in the counter word. The counter word can be IW, OW, FW, DW.

In these counter words, the counter value of the counter is specified. The counter value is given in 16 bit BCD code, whereby the first 4 bits of the counter are not processed.



In the above example the initial value of the counter is 394. Outputs DU and DE are digital.

The counter value is in binary code at output DU and in BCD code at output DE.

Reset counter

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.2 R C 1 A C 1 = Q 2.4 </pre>		

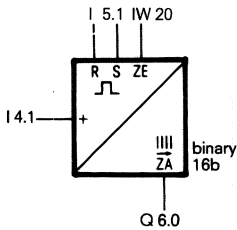
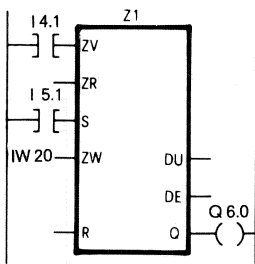
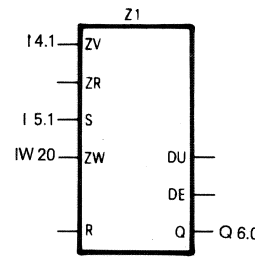
The counter is reset when the result of the logic operation is "1". The counter remains unchanged even if the result of the logic operation becomes "0".

6. Programming examples

6.1 Basic operations

6.1.5 Counter functions

Counting up

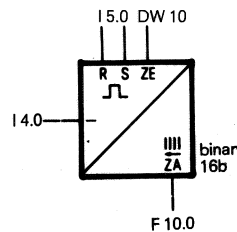
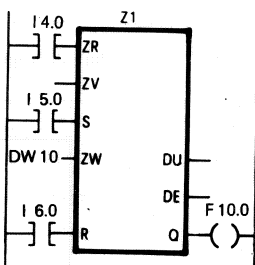
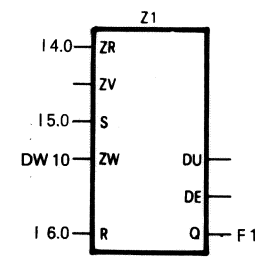
Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.1 CUC 1 NOP 0 A I 5.1 L IW 20 S C 1 NOP 0 NOP 0 NOP 0 A C 1 = Q 6.0 </pre>		

The value of the addressed counter is only incremented by 1 if the CU input of the counter shows an edge-change from "0" to "1". The flags necessary for pulse edge evaluation of the counter inputs are included in the counter word.

A counter with two different inputs can be used as an up/down counter by means of the two separate pulse-edge flags for CU and CD.

The Q output of the counter remains "1" as long as the actual count is > 0.

Counting down

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 4.0 CDC 1 NOP 0 A I 5.0 L DW 10 S C 1 A I 6.0 R C 1 NOP 0 NOP 0 A C 1 = F 10.0 </pre>		

The value of the addressed counter is only decremented by 1 if the CD input of the counter shows an edge-change from "0" to "1". The flags necessary for pulse edge evaluation of the counter inputs are included in the counter word.

A counter with two different inputs can be used as an up/down counter by means of the two separate pulse-edge flags for CU and CD.

The Q output of the counter remains "1" as long as the actual count is > 0.

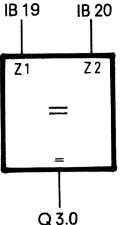
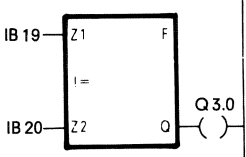
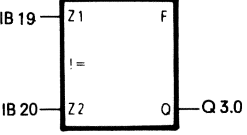
6. Programming examples

6.1 Basic operations

6.1.6 Comparison functions

6.1.6 Comparison functions

Comparing for equal to

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> L IB 19 L IB 20 !=F = Q 3.0 </pre>		

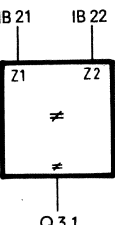
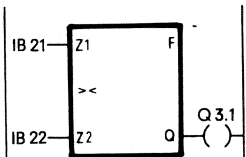
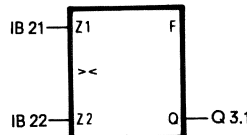
The first operand specified is compared with the subsequent operand according to the comparison function. The result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 19 = IB 20	0	0	1
IB 19 < IB 20	0	1	0
IB 19 > IB 20	1	0	0

The numerical representation of operands (fixed-point arithmetic) is taken into account.

After comparing for equal to, a jump can be made with the JZ = ... function (if the RLO = 1) to a "label" (± 127 words).

Comparing for not equal to

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> L IB 21 L IB 22 ><F = Q 3.1 </pre>		

The first operand specified is compared with the subsequent operand according to the comparison function. The result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 21 = IB 22	0	0	0
IB 21 < IB 22	0	1	1
IB 21 > IB 22	1	0	1

The numerical representation of operands (fixed-point arithmetic) is taken into account.

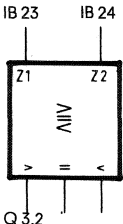
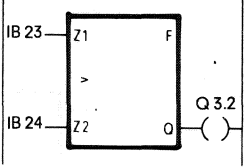
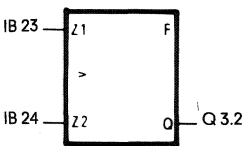
After comparing for not equal to, a jump can be made with the JN = ... function (if the RLO = 1) to a "label" (± 127 words).

6. Programming examples

6.1 Basic operations

6.1.6 Comparison functions

Comparing for greater than

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 23 L IB 24 > F = Q 3.2</pre>		

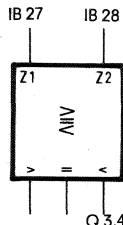
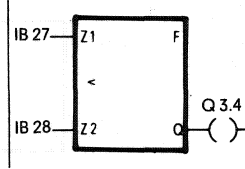
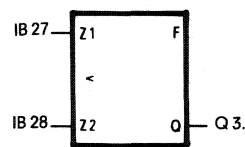
The first operand specified is compared with the subsequent operand according to the comparison function. The result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 23 = IB 24	0	0	0
IB 23 < IB 24	0	1	0
IB 23 > IB 24	1	0	1

The numerical representation of operands (fixed-point arithmetic) is taken into account.

After comparing for greater than, a jump can be made with the JP = ... function (if the RLO = 1) to a "label" (± 127 words).

Comparing for less than

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 27 L IB 28 < F = Q 3.4</pre>		

The first operand specified is compared with the subsequent operand according to the comparison function. The result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 27 = IB 28	0	0	0
IB 27 < IB 28	0	1	1
IB 27 > IB 28	1	0	0

The numerical representation of operands (fixed point arithmetic) is taken into account.

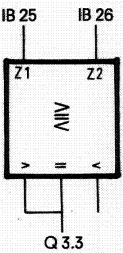
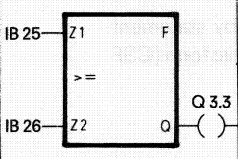
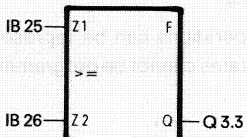
After comparing for less than, a jump can be made with the JM = ... function (if the RLO = 1) to a "label" (± 127 words).

6. Programming examples

6.1 Basic operations

6.1.6 Comparison functions

Comparing for greater than or equal to

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 25 L IB 26 =>F = Q 3.3</pre>		

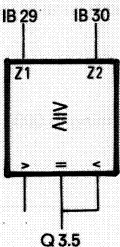
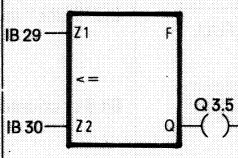
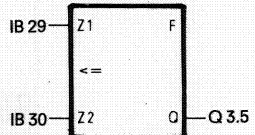
The first operand specified is compared with the subsequent operand according to the comparison function. The result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 25 = IB 26	0	0	1
IB 25 < IB 26	0	1	0
IB 25 > IB 26	1	0	1

The numerical representation of operands (fixed-point arithmetic) is taken into account.

After comparing for greater than or equal to, a jump can be made with the JC = ... function (if the RLO = 1) to a "label" (± 127 words).

Comparing for less than or equal to

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 29 L IB 30 <=F = Q 3.5</pre>		

The first operand specified is compared with the subsequent operand according to the comparison function the result of the comparison is shown by the CC0 and CC1 condition codes.

for	CC1	CC0	RLO
IB 29 = IB 30	0	0	1
IB 29 < IB 30	0	1	1
IB 29 > IB 30	1	0	0

The numerical representation of operands (fixed-point arithmetic) is taken into account.

After comparing for less than or equal to, a jump can be made with the JC = ... function (if the RLO = 1) to a "label" (± 127 words).

6. Programming examples

6.2 Supplementary operations (function blocks)

6.2.1 Binary logic functions

6.2 Supplementary operations (function blocks)

Function blocks can be programmed with an operation set supplementary to the program blocks so that the entire operation set for function blocks consists of the basic operations and the supplementary operations.

Function block operations can be represented only by statement lists, i.e. the programs cannot be programmed in graphic form (CSF or LAD).

The following description refers to operations which are used only with function blocks. The possible combinations of the substitution statements are given with the actual operands.

6.2.1 Binary logic functions

Example	STL	Description
Photoelectric cell equipment connected to input I 2.0 has been installed to count single items. After 100 items, a jump should be made either to function block FB 5 or to FB 6. After a count of 800, the counter 10 is automatically reset and starts counting from the beginning again.	A I 2.0 CU C 10 A I 3.0 L KC 0 S C 10 O I 4.0 O F 5.2 R C 10 LD C 10 T DW 12	The count of the C 10 counter is loaded with the constant 0 by input I 3.0. Each positive edge change at I 2.0 increments the count by 1. The counter is either reset by I 4.0 or flag F 5.2. The current count is stored in the data word as a BCD value.
	TBN DW 12.8 JC FB 5	
	TB DW 12.8 JC FB 6	
	TB DW 12.11 = F 5.2	
Photoelectric cell equipment connected to input I 10.0 counts individual pieces. After every 256, the counter should be reset and start counting again.	:A I 10.0 :CU C 20 :A I 11.0 :L KC 0 :S C 20	Counter C 2.0 is loaded by input I 11.0 with the constant 0. At each positive edge change at I 10.0, the count is incremented by 1. As soon as the count has reached the number 256 \triangleq 100 _H (bit 8 is "1"), a jump is made to the "FULL" label; apart from this, the block is ended.
	TB C 20.8 :JC = FULL ¹⁾ :BEU	
	FULL:RU C 20.8 :BE	

Note:
The times and counts are in the 10 lower-order bits (bits 0 to bit 9) of the time word/count word in hexadecimal code. The time base is stored in bit 12 and 13 of the time word.

¹⁾ See jump functions 6.2.4 (page 36)

6. Programming examples

6.2 Supplementary operations

6.2.2 Digital logic functions

6.2.3 Arithmetic functions

6.2.2 Digital logic functions

Example:	STL	Description
<p>The hexadecimal number 3F84_H is to be ANDed with input word 1 of the inputs (IW 1). The result is to be written into output word 10 of the outputs (QW 10).</p> <pre> 3F84_H IW 1 4793_H ----- Result AW 0784_H </pre>	<pre> L KH 3F84 L IW 1 AW T QW 10 </pre>	<p>The hexadecimal number is loaded into accu 1. The old contents of accu 1 are shifted into accu 2 simultaneously.</p> <p>Input word 1 (IW 1) is loaded into accu 1 and the hexadecimal number shifted into accu 2.</p> <p>The contents of accu 1 are digitally ANDed with those of accu 2 and the result is stored in accu 1.</p> <p>The contents (result) of accu 1 are transferred to output word 10 (QW 10).</p>
<p>The two bit patterns 0101 1110 1000 1011 and 0111 0001 0111 1100 are to be ORed</p> <pre> 0101 1110 1000 1011 0111 0001 0111 1100 ----- Result OW 0111 1111 1111 1111 </pre>	<pre> L KM 01 11 L KM 01 00 OW T FW 13 </pre>	<p>The first bit pattern is loaded into accu 1. The old contents of accu 1 are shifted into accu 2 simultaneously.</p> <p>The second bit pattern is loaded into accu 1 and the first bit pattern shifted into accu 2.</p> <p>The contents of accu 1 are digitally ORed with those of accu 2 and the result is stored in accu 1.</p> <p>The contents (result) of accu 1 are transferred to flag word 13 (FW 13).</p>
<p>Input word 5 is to be compared for equality with data word 12. The dissimilar bits of the words are to be written into output word 6.</p> <pre> DW 12 EA83_H IW 5 68C5_H ----- Result XOW 8246_H </pre>	<pre> L DW 12 L IW 5 XOW T QW 6 </pre>	<p>Data word 12 is loaded into accu 1. The old contents of accu 1 are shifted into accu 2 simultaneously.</p> <p>Input word 5 is loaded into accu 1 und data word 12 is shifted into accu 2.</p> <p>The contents of accu 1 are digitally EXORed with those of accu 2 and the result is stored in accu 1.</p> <p>The contents (result) of accu 1 are transferred to output word 6.</p>

6.2.3 Arithmetic functions

Example:	STL	Description
<p>The right-hand byte of data word 85 is to be subtracted from the number + 127, and the result stored in the left-hand byte of data word 85.</p> <pre> 127 DR 85 74 ----- Result -F 53 </pre>	<pre> L KF +127 L DR 85 -F T DL 85 </pre>	<p>The constant fixed-point number + 127 is loaded into accu 1. The old contents of accu 1 are shifted into accu 2 simultaneously.</p> <p>The right-hand byte of data word 85 is loaded into accu 1 and the fixed-point number + 127 shifted into accu 2.</p> <p>The contents of accu 1 are subtracted from those of accu 2 and the result is stored in accu 1.</p> <p>The contents of accu 1 (result) are transferred to the left-hand byte of data word 85.</p>

Note:

When exceeding the numeral area (−32 768 to +32 767) the result of operation is undefined (OVR = "1").

6. Programming examples
6.2 Supplementary operations

6.2.4 Jump functions
6.2.5 Timer and counter functions
6.2.6 Shift functions

6.2.4 Jump functions

The destination for unconditional and conditional jump statements is specified symbolically (max. 4 characters). The symbolic parameter of the jump statement is identical to the symbolic address of the statement to be jumped to. When programming, it must be remembered that the absolute jump displacement cannot be more than ± 127 words and that a STEP 5 statement cannot consist of more than one word. Jumps can only be executed within a module; jumps across segments are not permitted.

Example:	STL	Explanation
If no input of input word 1 is set, jump is made to the "AN 1" label.	AN0:L IW1	Input word 1 is loaded into accu 1. If the contents of accu 1 = 0, a jump is made to the „AN 1" label, otherwise the next statement (AI 1.0) is processed.
If input word 1 and output word 3 are dissimilar, a jump is made back to the "AN 0" label.	JZ=AN1	
If input word 1 and output word 3 are identical, input word 1 is compared with data word 12.	:A I1.0	
If input word 1 is greater or less than data word 12, a jump is made to the "DES" label.	AN1:L IW1	Comparison of input word 1 and output word 3. If dissimilar, individual bits are set in accu 1.
	:L QW3	
	:XOW	
	JP=AN0	If accu 1 is not zero, a jump is made back to the "AN 0" label, otherwise the next statements are processed.
	:L IW1	Input word 1 is compared with data word 12 for greater than/less than. If larger or smaller, RLO "1" is set.
	:L DW12	
	:><F	
	JC=DES	If RLO = "1", a jump is made to the "destination" label.
	DES: A I12.2	If RLO = "0", the next statement is processed.

6.2.5 Timer and counter functions

Example	STL	Description
A timer T 2 is started as an extended pulse with 50s pulse duration. This timer sets output Q 4.2 for the duration of the pulse.	A I 2.5	A timer T2 is started as an extended pulse. Output 4.2 is set for 50s.
	L KT 5.3	
	SU T 2	
	A T 2	
	= Q 4.2	
	FR T 2	
	BE	
If output Q 3.4 is continually set, the time should be continually restarted.	A Q 3.4	If output 3.4 is set during the time (positive edge change of the RLO) within which input 2.5 is still set, timer T2 is restarted. This means that output 4.2 remains set for the duration of the restarted time or is set again.
		If input 2.5 is not set at edge change of output 3.4, the time is not restarted.

6.2.6 Shift functions

Example	STL	Description
In data word 1 the last four bits of the hexadecimal number 14AFH are to be deleted and the resultant hexadecimal number 014AH stored in data word 3.	L DW 1	Load data word 1 in accu 1.
	SR W 4	Shift contents of accu 1 four bit positions to the right. The bit positions which become vacant on shifting are filled with zeros.
	T DW 3	Transfer the contents of accu 1 to data word 3.

Note:
The shift functions are executed unconditionally. The last bit shifted out can be scanned with jump functions.
A jump can be made with JZ if the bit is "0" and with JN or JP if the bit is "1".

6.2.7 Conversion functions

Example	STL	Description
The contents of data word 64 are to be inverted bit by bit and stored in data word 78	L DW 64 CFW	Load data word 64 into accu 1. One's complement of the contents of accu 1. Result is in accu 1.
DW 64 EA83 _H DW 78 157C _H	T DW 78	Transfer the contents of accu 1 to data-word 78.
The contents of data word 42 are to be interpreted as a fixed-point number and stored with inverted sign in data word 35.	L DW 42 CSW	Load data word 42 into accu 1. Two's complement of the contents of accu 1. Results are in accu 1.
DW 42 +51 DW 35 -51	T DW 35	Transfer the contents of accu 1 to data word 35.

6.2.8 Decrementing/Incrementing

Example	STL	Description
The hexadecimal constant 1010 _H is to be incremented in steps of 16 and stored in data word 8. In addition, the result of incrementing is to be decremented in steps of 33 and stored in data word 9.	L KH 1010 I 16 T DW 8 D 33 T DW 9	Load hexadecimal constant 1010 _H into accu 1. Increment the low byte of accu 1 by 16. The result 1020 _H is in accu 1. Transfer the contents of accu 1 (1020 _H) to data word 8. As the result of incrementing is still in accu 1, the decrement 33 can be formed directly from it. The result would be FFF _H . However, as the high byte of accu 1 was not also decremented, the result in accu 1 is 10FF _H . The contents of accu 1 are transferred to data word 9 (10FF _H).

Note:

Incrementing and decrementing are always decimal, the results are always stored in accu 1 in hexadecimal.

6.2.9 Disable/enable command output

Example	STL	Description
If input 0.5 is set, the commands following are to be disabled.	A I 0.5 BAS	Scanning input 0.5 for "1" Disable command output with input 0.5 set (RLO \triangleq "1")
If input 0.6 is set and input 0.5 reset, the command disable is to be cancelled.	A I 0.6 AN I 0.5 BAF	Scanning input 0.6 for "1" Scanning input 0.5 for "0" Enable command output if input 0.6 is set and 0.5 is not set (RLO \triangleq "1").

Note:

"Disable/enable command output" can be used, for example, to repeat a sequence at a certain step without setting or resetting the steps already run through.

6. Programming examples

6.2 Supplementary operations

6.2.10 Disable/enable interrupts

6.2.11 Processing functions

6.2.10 Disable/enable interrupts

Example	STL	Description
Disabling and then enabling interrupt processing within a certain program section.	<pre> = Q 7.5 IA A I 2.3 JU FB 3 RA </pre>	<p>Disable interrupt</p> <p>If an interrupt occurs, interrupt block FBO is not first jumped to on jumping to FB3, but the program is processed normally.</p> <p>Enable interrupt. If the interrupt is still pending, a jump is first made to the interrupt block FBO at a block boundary.</p>

6.2.11 Processing functions

Example	STL	Description
The actual parameters of the inputs scanned are to be stored in data word 12.	<pre> DO DW12 A I 0.0 </pre>	Process data word 12. If high byte 6 and low byte 43 are in data word 12, input 43.6 is scanned for "1".
The actual parameters of the times it is desired to enable again are to be stored in data word 13.	<pre> DO DW13 FR T0 </pre>	Process data word 13. If high byte 0 low byte 15 are in data word 13, the time 15 is enabled by via input 43.6 for cold restart.
The contents of data words DW 20 to DW 100 are to set to signal state "0". The index register for the parameters of the data words is DW0.	<pre> :L KB20 :T DW1 M1 :L KH0 DO DW1 :T DW0 :L DW1 :L KB1 :+F :T DW1 :L KB100 :<=F :JC= M1 </pre>	<p>Load constant 20 into accu 1. Transfer contents of accu 1 to data word 1. Load hexadecimal constant 0 into accu 1.</p> <p>Process data word 1</p> <p>Transfer contents of accu 1 into the data word the address of which is stored in data word 1.</p> <p>Load data word 1 into accu 1. Load constant 1 into accu 1. Data word 1 is shifted into accu 2. Accu 1 and accu 2 are added together and the result is stored in accu 1 (increasing the data word address). Transfer contents of accu 1 into data word 1 (new data word address). The constant 100 is loaded into accu 1 and the new data word address is shifted into accu 2. Comparison of the accus for less than or equal accu 2 ≤ accu 1 Conditional jump to label M1 as long as accu 2 is ≤ accu 1.</p>

Note:
The 670 programming unit does not check the legality of the combination of the parameters with the operands. The parameter is assigned to the specified operation from the data or flag word. The high byte of the flag or data word is only necessary for inputs/outputs and for flags (between 0 and 7), otherwise it must be 0.

The "AI" operation in combination with "DO DW" and "DO FW" becomes an "AQ" operation if the byte address in the data/flag word is larger than 127.

If a parameter ≠ 0 is specified for the operations which are combined with "DO DW" or "DO FW", no address computation is executed. Both parameters are ORed.

The following operations can be combined with the DO DW/DO FW operation:

A-, AN-, O-, ON- S-, R-, = - FR T, RT, SFT, SRT, SPT, SST, SET FR C, RC, SC, CDC, CUC L-, LD, T- JU, JC, JZ, JN, JP, JM, JO SLW, SRW D, I CDB, JU-, JC-	Binary functions Memory functions Timer functions Counter functions Load and transfer functions Jump functions Shift functions Decrementing, incrementing Block calls
---	---

6.2.12 Substitution functions

When processing the STEP 5 program, the PC executes a "substitution" within a function block if the operand is a formal parameter (e.g.

HANS stands for I 1.5, see page 9). When the function block is called, the formal parameter is replaced (substituted) by a genuine operand.

AND – OR logic with RS flip-flop

Program in function block (FB 30)	Function block call	Executed program
<pre> A =INP 1 AN =INP 2 O =INP 3 S =MOT 5 = =OUT 1 A =VAL 1 A =INP 2 ON =INP 3 RB =MOT 5 = =OUT 2 BE </pre>	<pre> STL :JU FB 30 NAME: LOGIC INP 1: I 2.0 INP 2: I 2.1 INP 3: I 2.2 VAL 1: I 2.3 OUT 1: Q 7.1 OUT 2: Q 7.2 MOT5: Q 7.3 :BE LAD/CSF FB 30 I2.0--- INP 1 OUT 1 --- Q 7.1 I2.1--- INP 2 OUT 2 --- Q 7.2 I2.2--- INP 3 MOT 5 --- Q 7.3 I2.3--- VAL 1 </pre>	<pre> :A I 2.0 :AN I 2.1 :O I 2.2 :S Q7.3 := Q7.1 :A I 2.3 :A I 2.1 :ON I 2.2 :R Q7.3 := Q7.2 :BE </pre>

Operation	Description
A =...	ANDing. Scanning a formal operand for "1" [operands I, Q, F, T, C].
AN =...	ANDing. Scanning a formal operand for "0" [operands I, Q, F, T, C].
O =...	ORing. Scanning a formal operand for "1" [operands I, Q, F, T, C].
ON =...	ORing. Scanning a formal operand for "0" [operands I, Q, F, T, C].
S =...	Binary setting of a formal operand [operands I, Q, F].
RB =...	Binary resetting of a formal operand [operands I, Q, F].
= =...	Assignment of the RLO to a formal operand [operands I, Q, F].

Load and transfer functions

Program in function block (FB 34)	Function block call	Executed program
<pre> :A =I 0 :L =L 1 :S C 6 :A =I 1 :LW =LW 1 :S C 7 :A I 2.2 :CU C 6 :CU C 7 :LD =LC 1 :T =T 1 :A I 2.7 :R C 6 :R C 7 :LW =LW 2 :LD =LC 1 :=F :R C 7 :BE </pre>	<pre> STL :JU FB 34 NAME: LOAD/TRAN I 0 : I 2.0 I 1 : I 2.1 L 1 : FW 10 LW 1: KC 140 LC 1: C 7 T 1: QW4 LW 2: KC 160 :BE LAD/CSF FB 34 I2.0--- I0 T1 --- QW 4 I2.1--- I1 FW 10--- L1 KC 140--- LW 1 C 7 --- LD 1 KC 160--- LW 2 </pre>	<pre> :A I 2.0 :L FW 10 :S C 6 :A I 2.1 :L KC 140 :S C 7 :A I 2.2 :CU C 6 :CU C 7 :LD C 7 :T QW4 :A I 2.7 :R C 6 :R C 7 :L KC 160 :LD C 7 :=F :R C 7 :BE </pre>

Operation	Description
L =...	Loading a formal operand. The value of the operand specified as formal operand is loaded into accu 1 [operands IB, IW, FB, FW, QB, QW, DR, DL, DW, PB, PW].
LD =...	Loading a formal operand as BCD number. The value of the timer or counter location specified as formal operand is loaded into accu 1 as a BCD number [operands T, C].
LW =...	Loading the bit pattern of a formal operand. The bit pattern of the formal operand is loaded into accu 1 [operands KB, KS, KF, KH, KM, KY, KT, KC].
T =...	Transfer to a formal operand. The contents of the accu are transferred to the operand specified as formal operand [operands IB, IW, FB, FW, QB, QW, DR, DL, DW, PB, PW].

6. Programming examples

6.2 Supplementary operations

6.2.12 Substitution functions

Timer functions

Program in function block (FB 32)	Function block call	Executed program
<pre> :AN =I 5 :A =I 6 :L KT 5.2 :SFD =TIM 5 :A =I 5 :AN =I 6 :L KT 5.2 :SSU =TIM 6 :A =TIM 5 :O =TIM 6 := =OUT 6 :A I 2.7 :RD =TIM 5 :RD =TIM 6 :BE </pre>	<pre> STL :JU FB 32 NAME: TIME I 5 : I 2.5 I 6 : I 2.6 TIM 5: T 5 TIM 6: T 6 OUT 6: Q 7.6 :BE LAD/CSF FB 32 I2.5 --- I5 I2.6 --- I6 T5 --- TIM5 T6 --- TIM6 OUT 6 --- Q7.6 </pre>	<pre> :AN I 2.5 :A I 2.6 :L KT 5.2 :SF T 5 :A I 2.5 :AN I 2.6 :L KT 5.2 :SS T 6 :A T 5 :O T 6 := Q 7.6 :A I 2.7 :R T 5 :R T 6 :BE </pre>

Operation	Description
SP =....	Starting a timer specified as formal operand with the previously loaded time as pulse [operand T].
SEC =....	Starting a timer specified as formal operand with the previously loaded time as extended pulse [operands T and C (see counter functions)].
SI =....	Starting a timer specified as formal operand with the previously loaded time as "on" delay [operand T].
SFD =....	Starting a timer specified as formal operand with the previously loaded time as "off" delay [operands T and C (see counter functions)].
SSU =....	Starting a timer specified as formal operand with the previously loaded time as stored "on" delay [operands T and C (see counter functions)].
FR =....	Enable a formal operand for cold restart [operands T and C (see counter functions)].
RD =....	Digital resetting of a formal operand [operands T and C].

Counter functions

Program in function block (FB 33)	Function block call	Executed program
<pre> :A =I 2 :L KC 17 :SEC =COU 5 :A =I 3 :SSU =COU 5 :A =I 4 :SFD =COU 5 :A =COU 5 := =OUT 3 :A I 2.7 :RD =COU 5 :BE </pre>	<pre> STL :JU FB 33 NAME: COUNT I 2 : I 2.2 I 3 : I 2.3 I 4 : I 2.4 COU5: C 5 OUT3: Q 7.3 :BE LAD/CSF FB 33 I2.2 --- I2 I2.3 --- I3 I2.4 --- I4 C5 --- COU5 OUT 3 --- Q7.3 </pre>	<pre> :A I 2.2 :L KC17 :S C 5 :A I 2.3 :CU C 5 :A I 2.4 :CR C 5 :A C 5 := Q 7.3 :A I 2.7 :R C 5 :BE </pre>

Operation	Description
SEC =....	Setting a counter specified as formal operand with the previously loaded count [operands C and T (see timer functions)].
SSU =....	Incrementing a counter specified as formal operand [operands C and T (see timer functions)].
SFD =....	Decrementing a counter specified as formal operand [operands C and T (see timer functions)].
FR =....	Enabling a formal operand for cold restart [operands C and T (see timer functions)].
RD =....	Digital resetting of a formal operand [operands C and T].

Processing functions

Program in function block (FB 35)	Function block call	Executed program
<div><div>:DO =D5</div><div>:L =DW 2</div><div><div>:DO =D 6</div></div><div>:T =DW 1</div><div>:T =Q 4</div><div><div>:DO =F 36</div></div><div>:BE</div></div>	<div>STL</div> <div>:JU FB 35</div> <div>NAME: PROCS.</div> <div>D 5 : DB 5</div> <div>DW2 : DW2</div> <div>D 6 : DB 6</div> <div>DW 1 : DW1</div> <div>Q 4 : QW4</div> <div>F36 : FB 36</div> <div>:BE</div> <div>LAD/CSF</div> <div>FB 35</div> <div><div><div>DB5 --- D5</div><div>DW2 --- DW2</div><div>DB6 --- D6</div><div>FB36 --- F36</div></div><div><div>DW1 --- DW1</div><div>Q 4 --- QW4</div></div></div>	<div>:C DB 5</div> <div>:L DW 2</div> <div>:C DB 6</div> <div>:T DW 1</div> <div>:T AW 4</div> <div>:JU FB 36</div> <div>:BE</div>

Operation	Description
DO =....	Process formal operand Only C DB JU PB JU FB can be substituted.

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.1 General

7. Rules governing compability between the LAD, CSF and STL methods of representation

7.1 General

Each of the methods of representation in the STEP 5 programming language has specific properties and limitations. Consequently, a program block written in STL form cannot simply be output as an LAD or CSF and the graphic methods of representation LAD and CSF may not always be fully compatible. In other words, it is not always possible to translate back from one form to the other. If the program has been entered as a LAD or CSF, it can be always translated back into STL form.

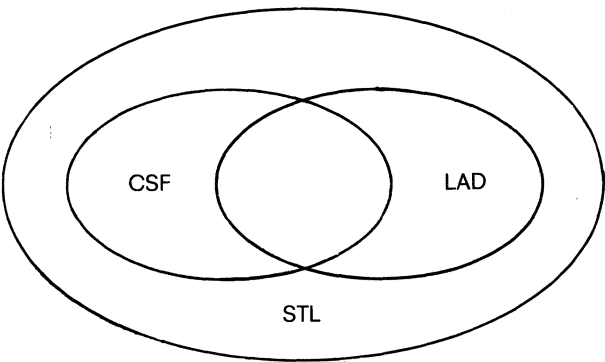


Fig. 22 Range and limitations of the methods of representation of the STEP 5 programming language.

The aim of this section is to establish a number of rules, which, if adhered to, will assure complete compatibility between the three methods of representation.

These rules are classified as follows:

- Rules for compatibility between the graphic methods of representation.

If these rules are followed, input is possible in one graphic form and output in the other form.

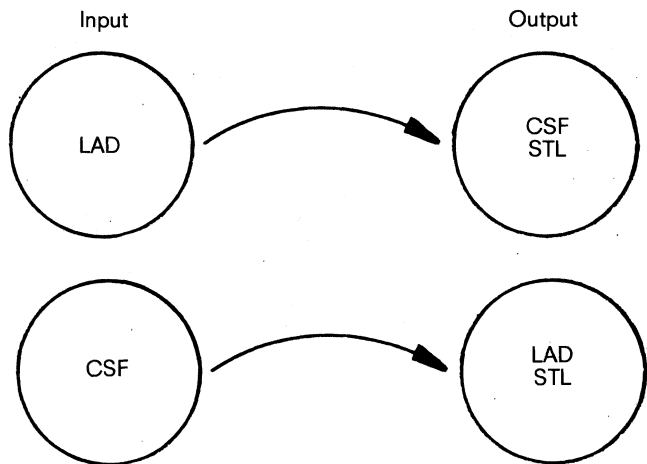


Fig. 23 Graphic input

- Rules for compatibility between statement lists and graphic methods of representation.

If these rules are followed, it is possible to enter a program in any of the methods of representation, graphic or otherwise, and have it output in the other two forms.

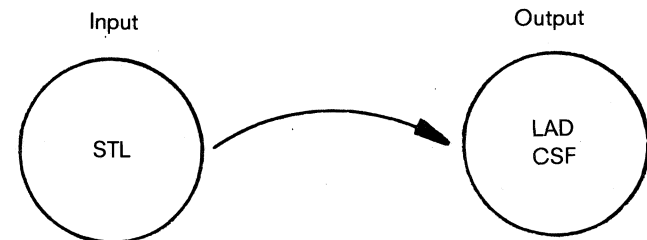


Fig. 24 Input in the form of a statement list

7.2 Rules governing compatibility between the graphic methods

7.2.1 Input in LAD, output in CSF (STL)

Excessive nesting can lead to the exceeding of the image boundaries (8 levels) in the CSF.

[illegible]

b) CSF

The diagram shows a series of seven normally open contacts labeled -EING. 1 through -EING. 7, connected in series to a coil labeled -AUSGANG. The contacts are arranged in a descending staircase pattern, with each contact on a lower horizontal line than the previous one. The first contact is on the left, and the last contact is on the right, connected to the output coil.

Fig. 25 Example of maximum LAD nesting for output in CSF form

Too many inputs on a CSF box cause the LAD display boundary to be exceeded.

```

-EING. 1 ---! & !
-EING. 2 ---! !
-EING. 3 ---! !
-EING. 4 ---! !
-EING. 5 ---! !
-EING. 6 ---! !
-EING. 7 ---! -- -AUSGANG

```

```

I
I-EING. 1  -EING. 2  -EING. 3  -EING. 4  -EING. 5  -EING. 6  -EING. 7  -AUSGANG
+---] [---+---] [---+---] [---+---] [---+---] [---+---] [---+---] [---+---( ,---I
I
I
b) LAD

```

43

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.2 Rules governing compatibility between the graphic methods

7.3 Rules governing compatibility between the STL and graphic methods

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

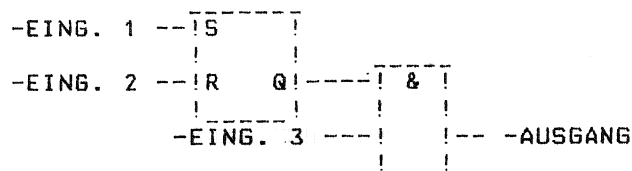


Fig. 27 Only AND boxes are allowed in a CSF after a complex element.

Rule 3: Connectors

- Connectors are always permitted with OR boxes
- Connectors are only permitted at the first input with AND boxes

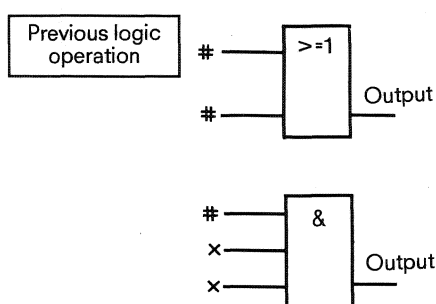


Fig. 28 Examples showing when connectors are permitted with OR and AND boxes (# connectors permitted; x connectors not permitted).

Note:

Connectors are intermediate flags that are used to reduce the number of recurrent operations.

7.3 Rules governing compatibility between the STL and graphic methods

Rule 1: AND operation

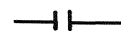
Scanning the signal state and ANDing with the result of the previous logic operation

LAD: Contact in series

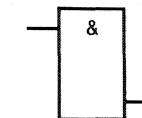
CSF: Input of an AND box

STL: Statement A...

LAD:



CSF:



STL:

A...

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

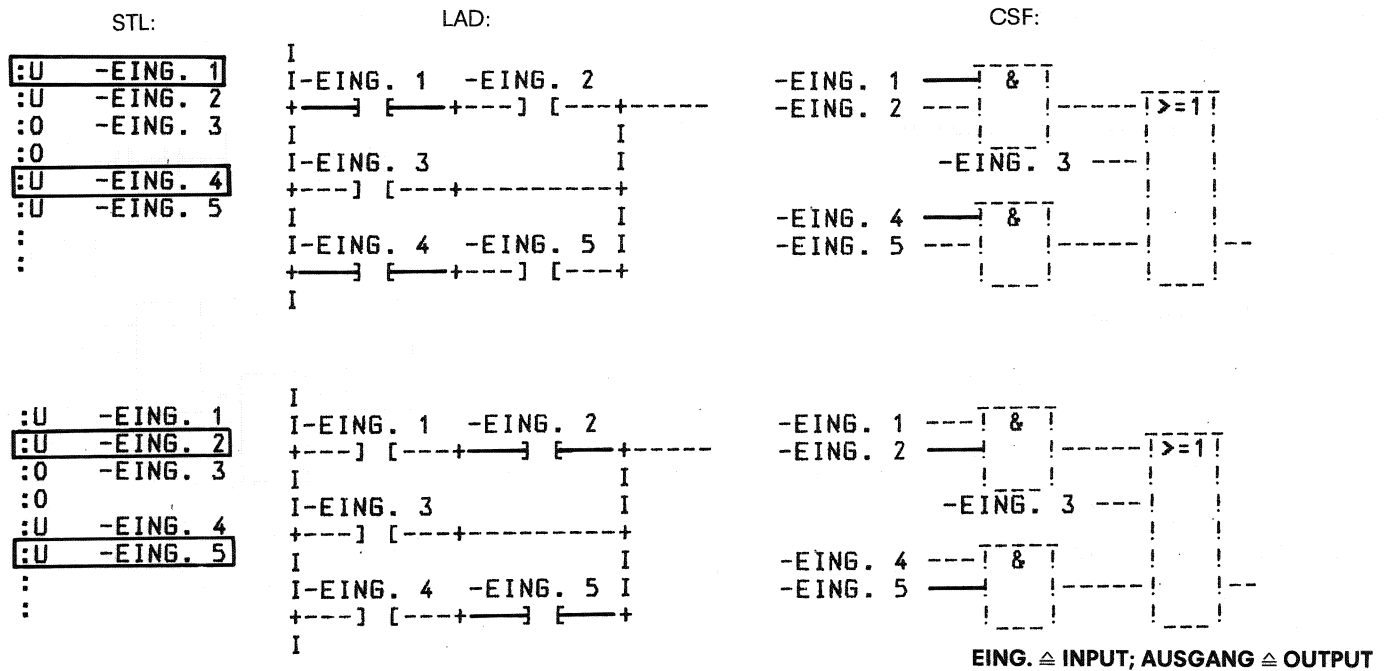


Fig. 29 ANDing

Rule 2: OR operation

Scanning the signal state and operation after ORing with the result of the previous logic operation

LAD: Only one contact in a parallel branch

CSF: Input of an OR box

STL: Statement O...

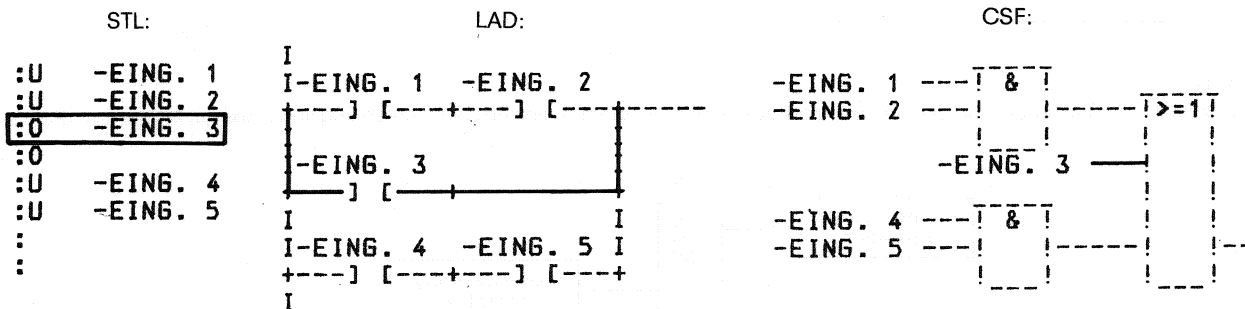
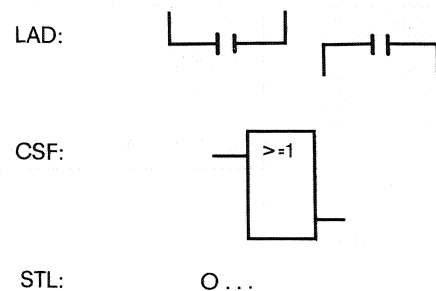


Fig. 30 ORing

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

Rule 3: AND before OR operation
ORing of AND functions

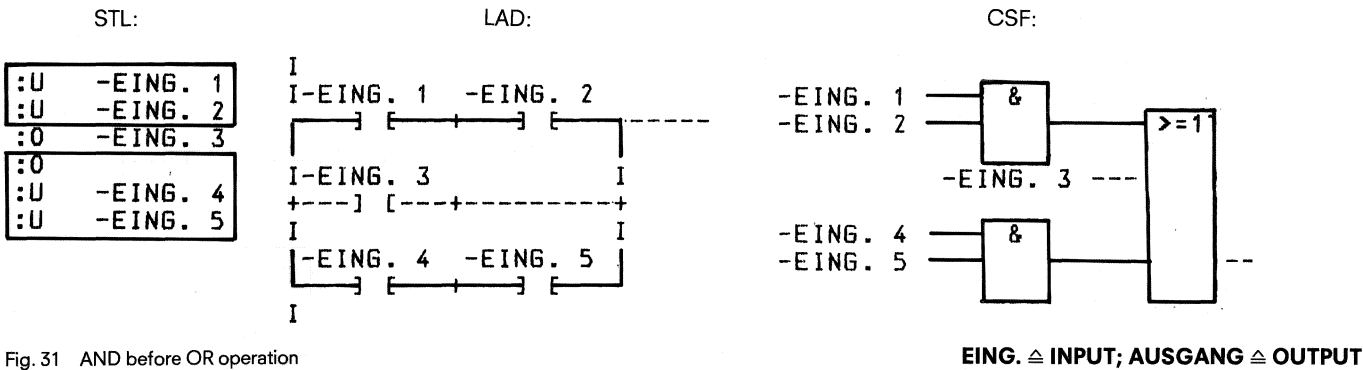
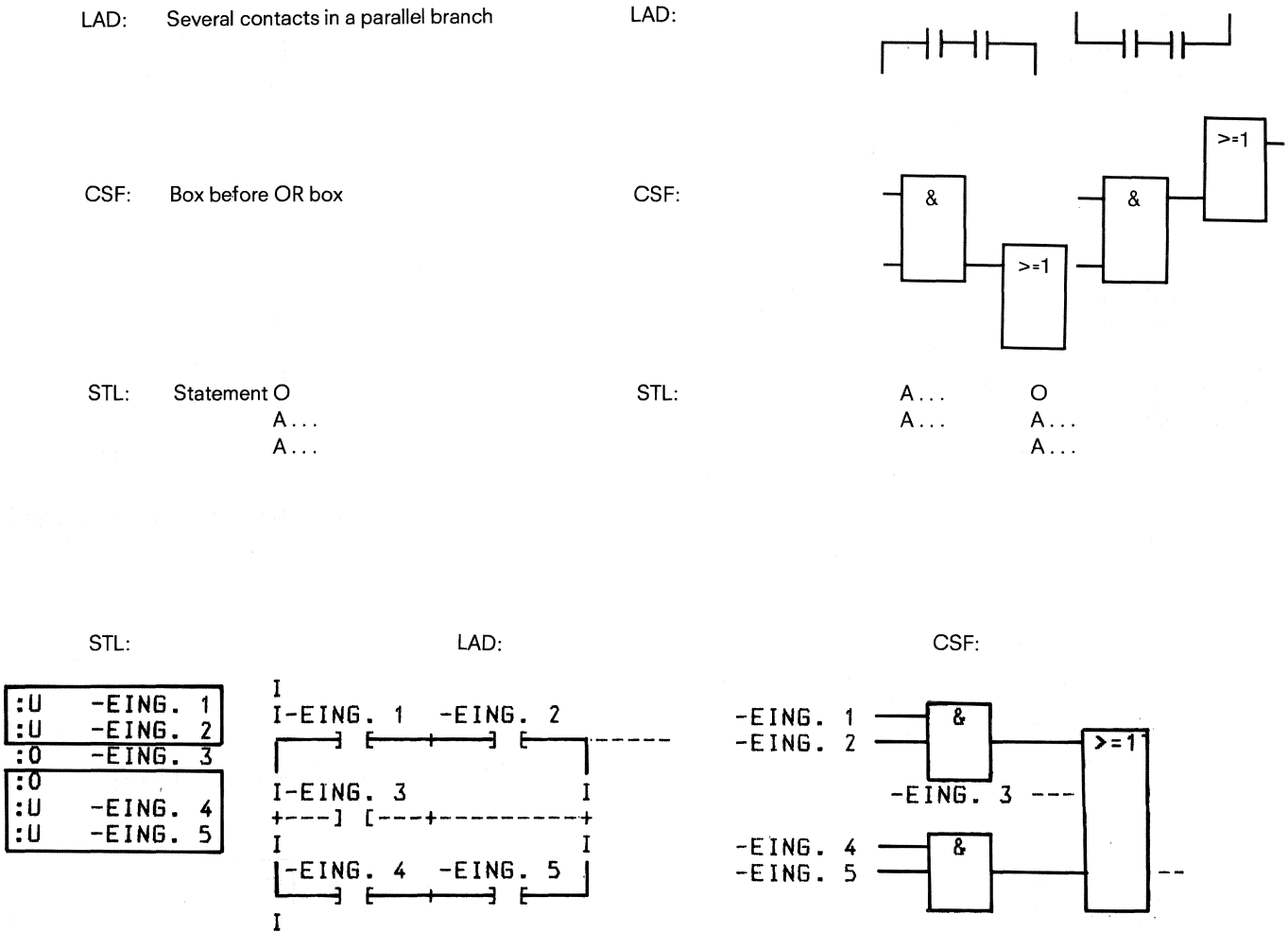


Fig. 31 AND before OR operation

Rule 4: Bracketing
This rule deals with the bracketing of complex binary operations, or complex elements with preceding and subsequent logic operations.

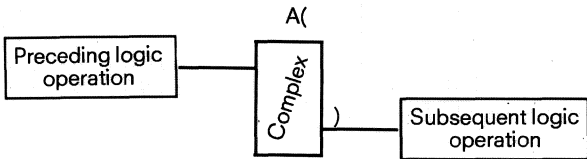


Fig. 32 Bracketing

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

a) Complex binary logic

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

- AND operation before OR functions
LAD: Switch parallel contacts in series
CSF: OR box before AND box
STL: Statements

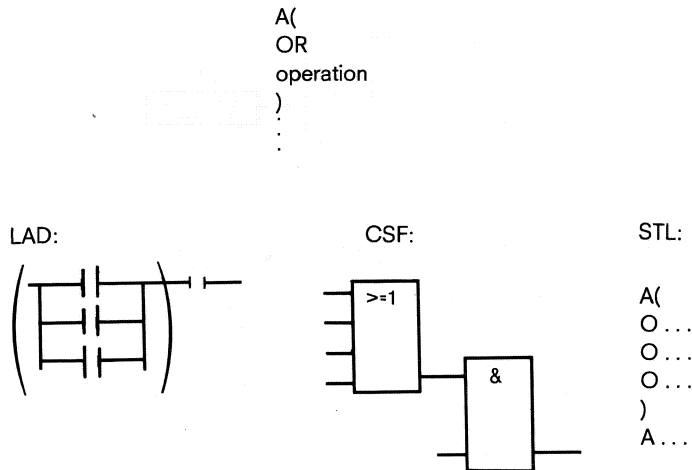


Fig. 33 OR before AND operation

The OR before AND operations are classed as a subset of the complex binary operations, whereby parallel contacts constitute the simplest complex binary logic.

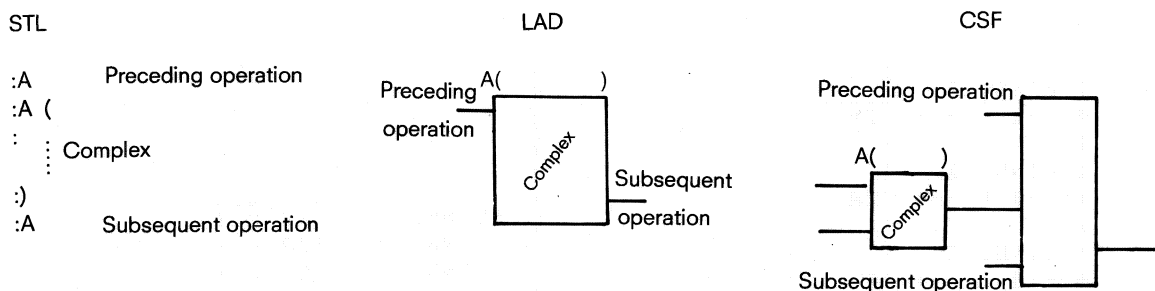


Fig. 34 Bracketing complex binary functions

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

b) Complex elements (memory, time, comparison or counter functions).

The following rules must be adhered to for complex elements:

- No brackets if there is no subsequent operation
- Subsequent operation AND A(...)
- Subsequent operation OR (only for CSF, not permitted for LAD) O(...)
- A complex element must not have a preceding operation.

In addition, each unused input or output must be assigned an NOP 0 operation.

Exception: S, TW with timers and S, CW with counters must always be defined together.

When programming with the STL, the complex elements must be programmed in the same order as they are assigned parameters on the screen in graphic mode.

Exception: Times and counts. The relevant value must previously have been stored in the accumulator with a load statement.

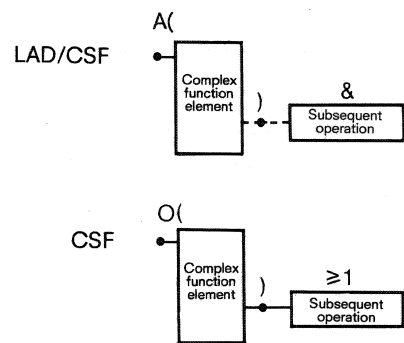


Fig. 35 Bracketing complex elements

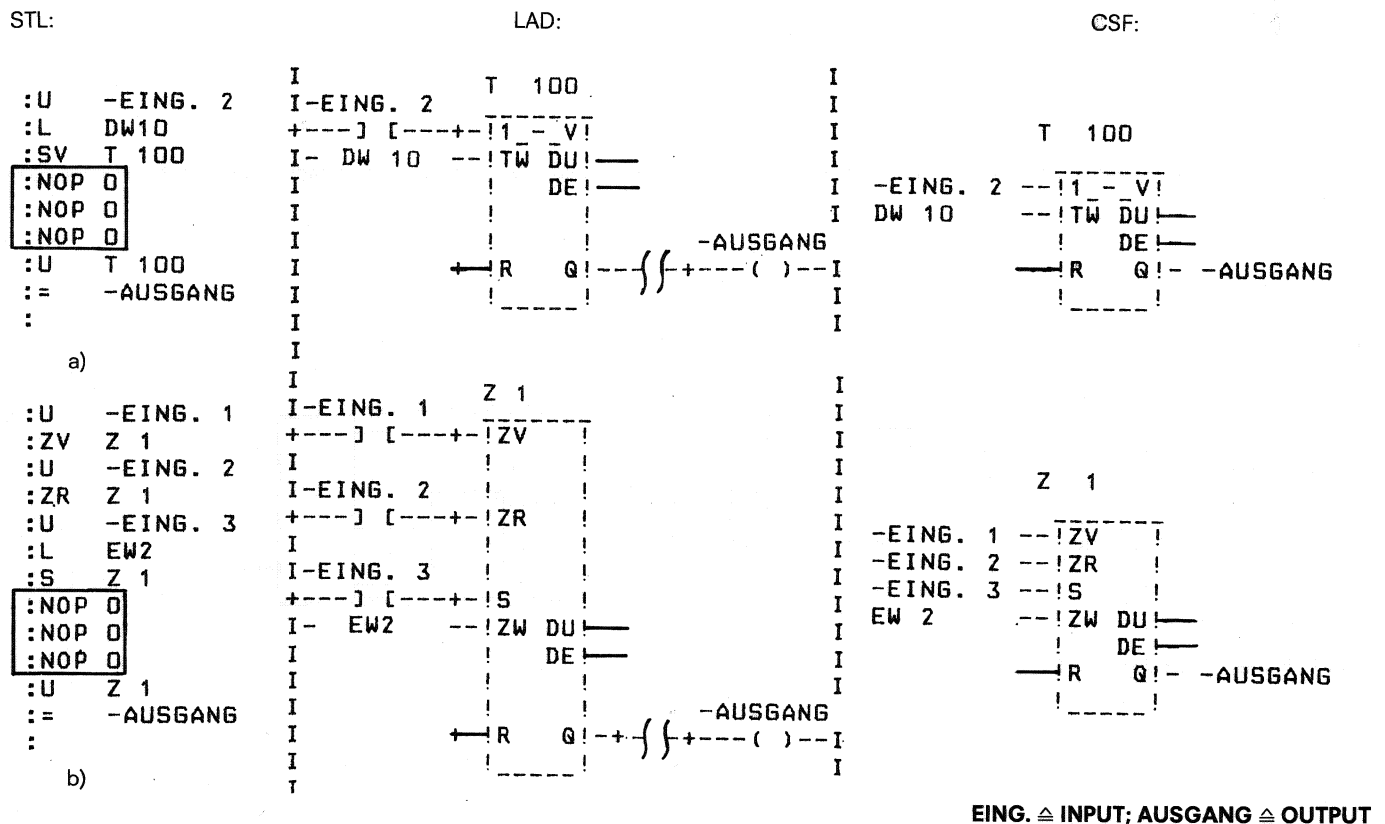


Fig. 36 Parameter assignment to unused inputs and outputs a) in the case of timers b) in the case of counters

Note: Only one complex function element is permitted per segment or rung.

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

The following examples show the four aforementioned cases in a complex binary operation in an STL and LAD (below) and as an STL and CSF (opposite).

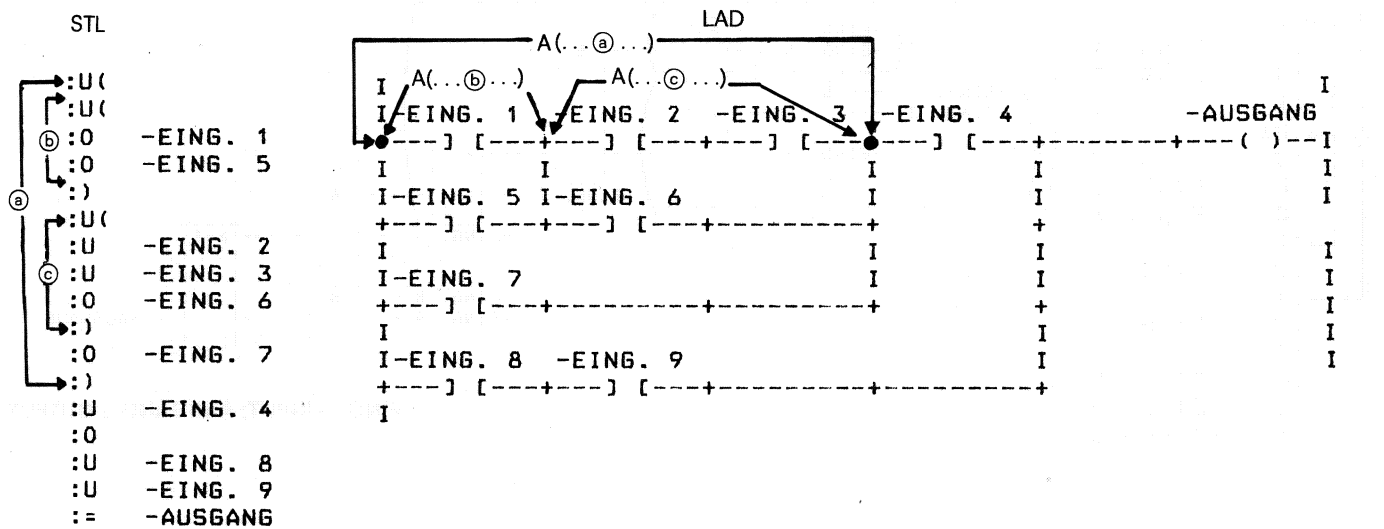
Example 1: STL/LAD

Case 1: AND (contacts in series)

Case 2: OR (only one contact in a parallel branch)

Case 3: AND before OR (several contacts in one parallel branch)

Case 4: OR before AND (bracketing)



EING. \triangleq INPUT; AUSGANG \triangleq OUTPUT

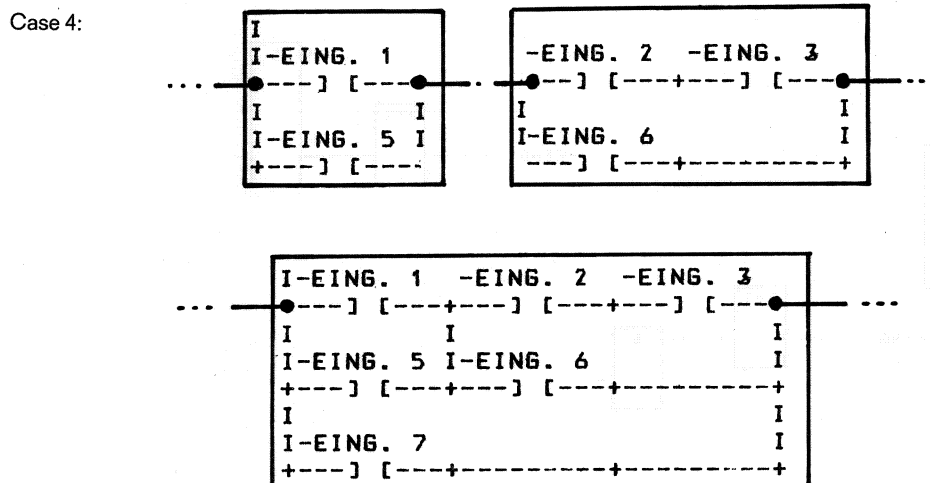
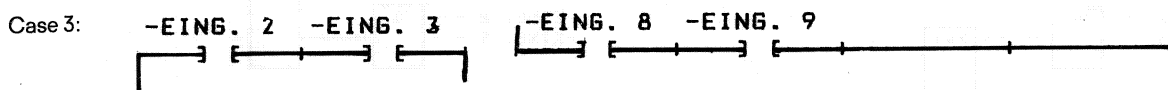
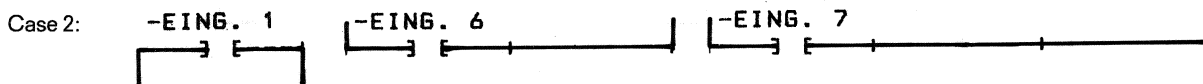
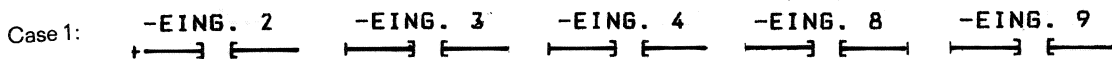


Fig. 37 Example 1: STL/LAD

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

Example 2: STL/CSF

Case 1: AND (input of an AND box)

Case 2: OR (input of an OR box)

Case 3: AND before OR (AND box before OR box)

Case 4: OR before AND (OR box before AND box)

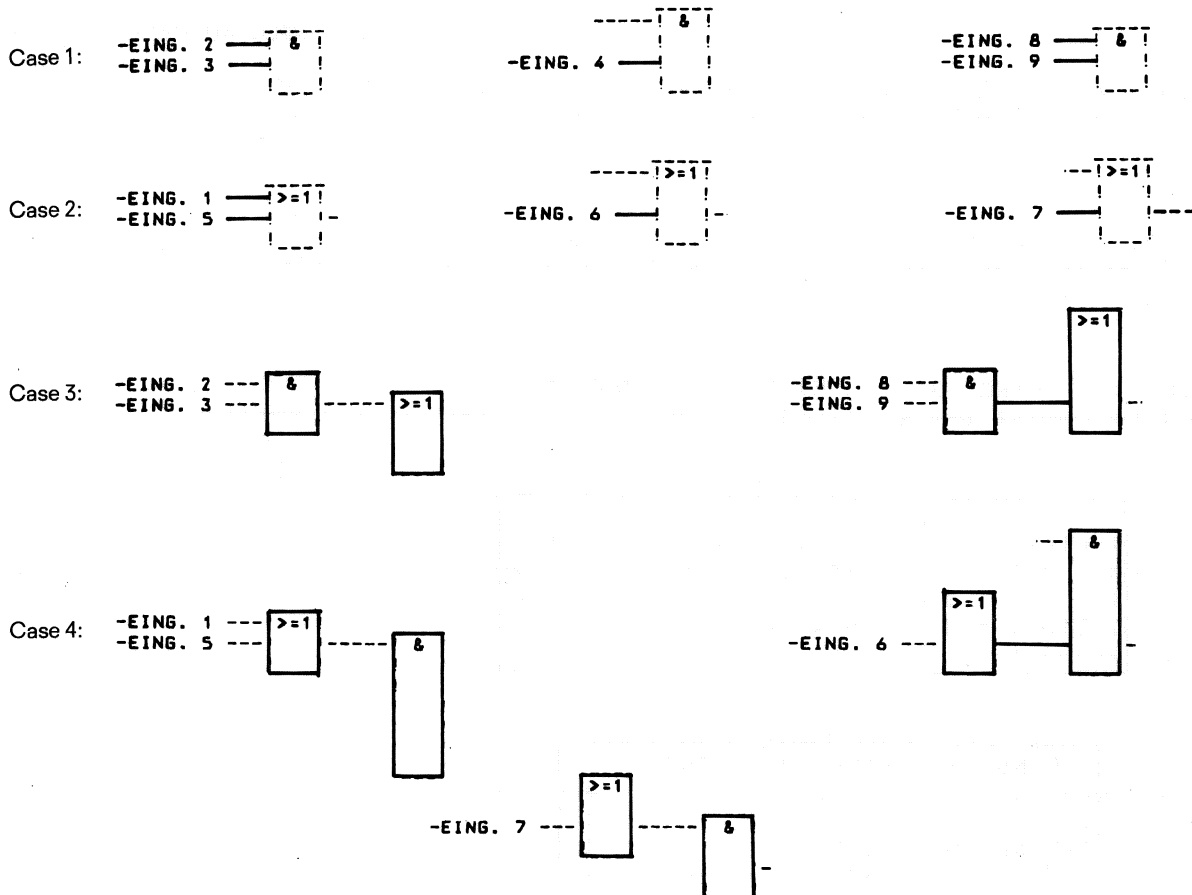
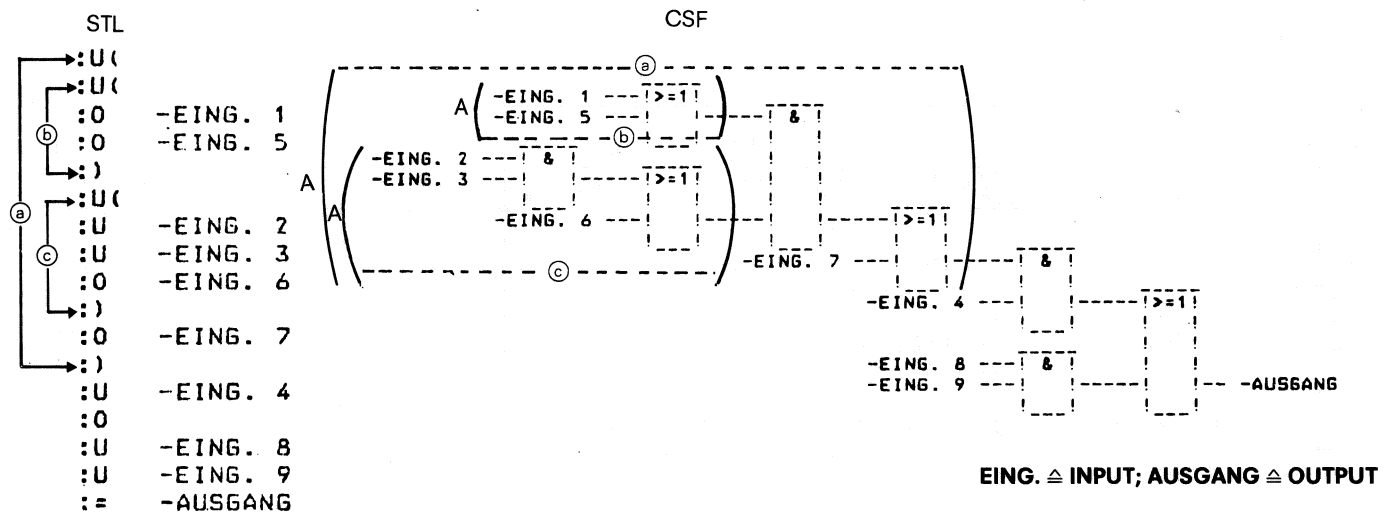


Fig. 38 Example 2: STL/CSF

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

Rule 5: Connectors

For reasons of clarity, the rules for connectors are given separately for the LAD and CSF methods.

a) with LAD

A connector as intermediate memory notes the result of the logic operation programmed before it in its own line.

The following rules apply:

- Connector in series (with other contacts).

A connector is treated in this case as a normal contact.

- Connector in a parallel branch. Within a parallel branch a connector is treated as a normal contact. In addition, the entire parallel branch must be enclosed within brackets of the O type (...).

- A connector must never stand immediately after the line (connector as first contact) or directly after the opening of a line (connector as first contact within a parallel branch).

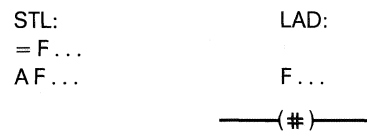
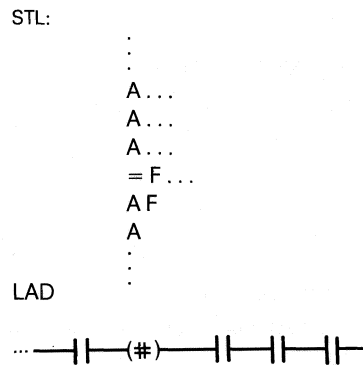


Fig. 39 The connector in the STL and LAD



a)



b)

Fig. 40 Connector rules for LAD a) Connector in series
b) Connector in parallel branch

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

b) with CSF

A connector as intermediate memory notes the result of the entire binary operation before this connector.

The following rules apply:

- Connector at the first input of an AND or OR box.

The connector is not bracketed.

- Connector not at the first input of an OR box.

The entire binary operation before the input is enclosed in parentheses of the O(...) type.

- Connector not at the first input of an AND box.

The entire binary operation before the input is enclosed in parentheses of the A(...) type.

Only permitted with CSF (cannot be represented graphically with LAD).

STL: CSF:
= F ... — # F... —
A F ...

Fig. 41 The connector in STL and CSF

STL: CSF:
= F ... — # F... —
A F ... &
A
A

STL: CSF:
... ...
O
O(Prec. >=1
 Preceding oper. — # F... —
 operation &
= F
A F
) ...
... ...
... ...

STL: CSF:
... ...
A
A(Prec. &
 Preceding oper. — # F... —
 operation &
= F
A F
) ...
... ...
... ...

Fig. 42 Connector rules for CSF

7. Rules governing compatibility between the LAD, CSF and STL methods of representation

7.3 Rules governing compatibility between the STL and graphic methods

Connector examples:

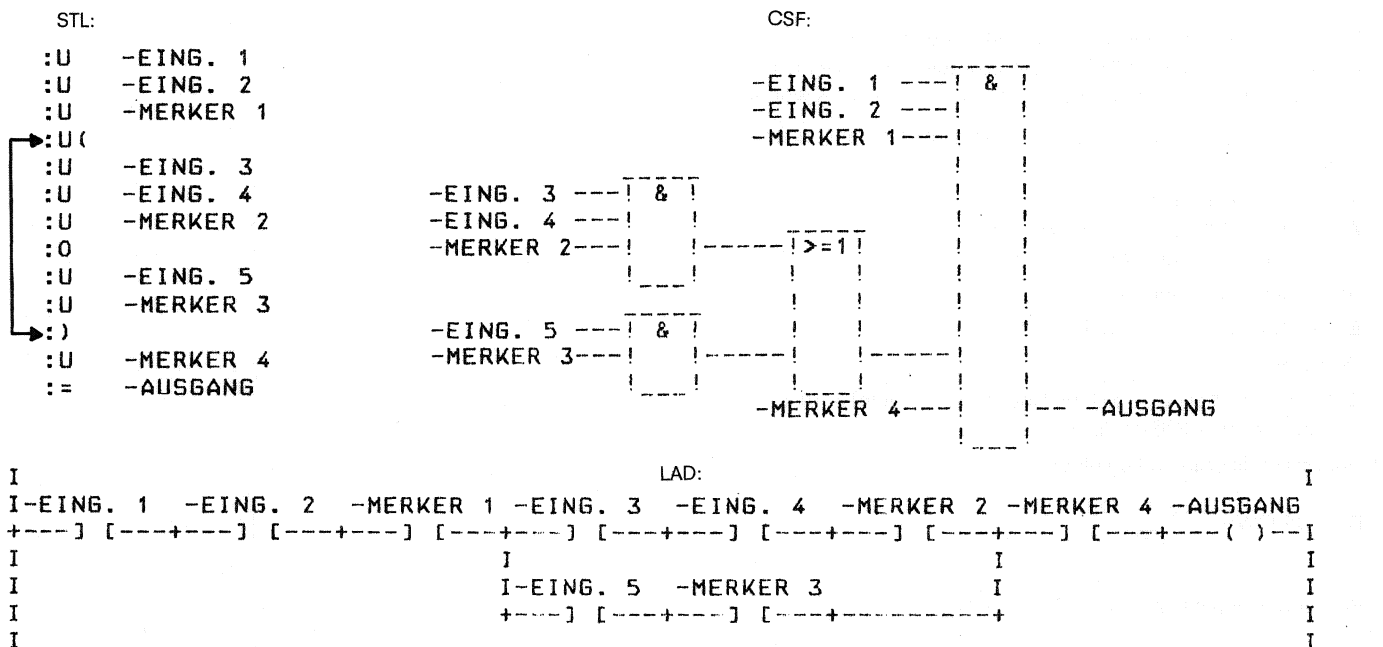


Fig. 43 Example 1: without connectors

EING. \triangle INPUT; AUSGANG \triangle OUTPUT; MERKER \triangle FLAG

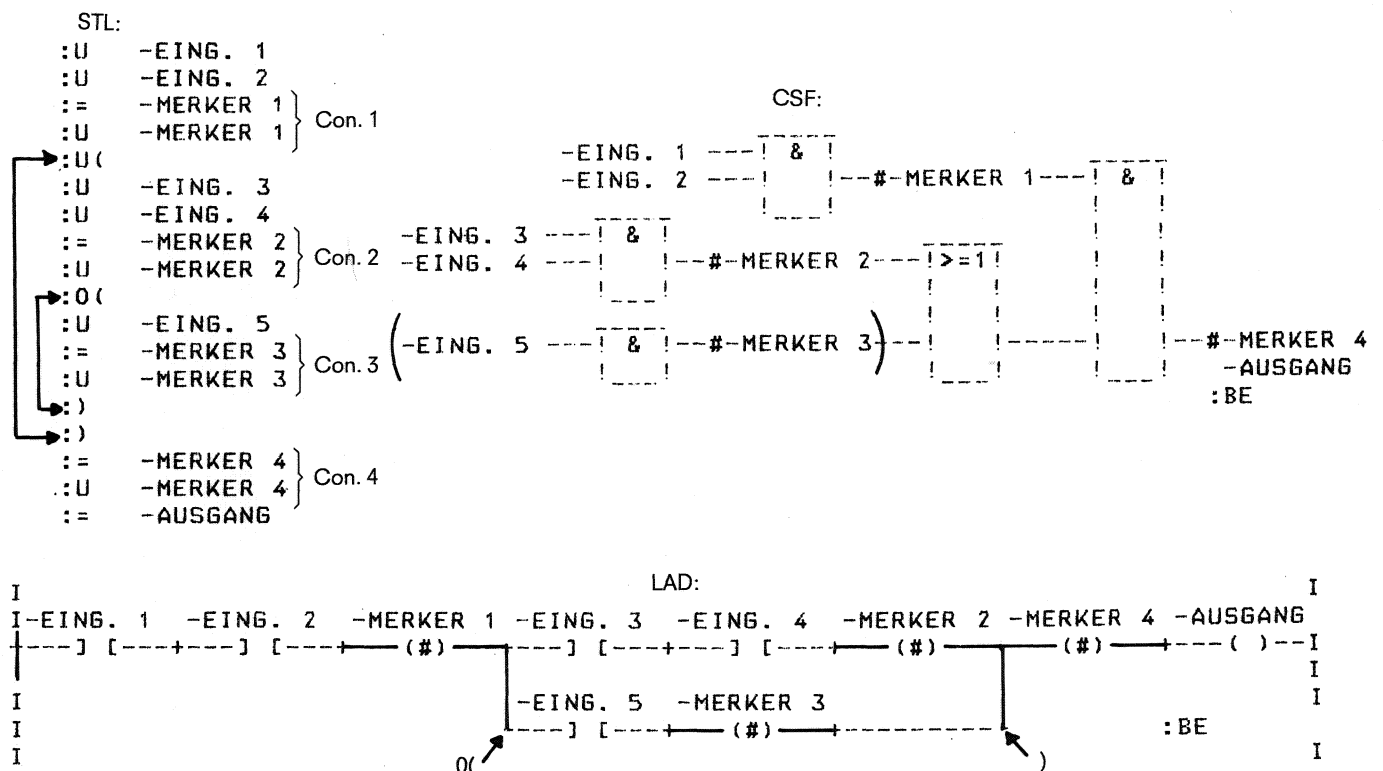


Fig. 44 Example 2: with connectors

- Connector 1: Result of logic operation of AI 1 and AI 2
- Connector 2: Result of logic operation of AI 3 and AI 4
- Connector 3: Result of logic operation of AI 5
- Connector 4: Result of the entire binary operation

8. Notes on estimating the required memory space

8. Notes on estimating the required memory space

The S5-110S programmable controller permits a maximum user memory configuration of 24.5 K words. Regardless of the memories plugged in, the CPU has an internal user memory of 0.5 K words.

The memory space required for a program can be roughly estimated as follows:

Statements in program blocks:

$$\mathbf{A} \quad \Sigma (\text{PB-MQW}) \approx 8 \times \Sigma (I + MQ) + 12$$

(Σ drives + Σ sequence cascades)

Statements in function blocks:

$$\mathbf{B} \quad \Sigma \text{FB-MQW} \approx (\text{number of FB}) \times 150$$

Data words:

$$\mathbf{C} \quad \Sigma \text{DW} \approx 2 \times \Sigma \text{drives} + \Sigma \text{steps (sequence cascade)} + 10 \times \Sigma \text{messages} + 256 \text{ (for listing)}$$

Statements in organization block:

$$\mathbf{D} \quad \Sigma \text{OB-QW: } \frac{8 \times \Sigma (I + Q)}{150}$$

$$\text{Required memory space} \approx A + B + C + D$$

9. Total overview of STEP 5 operations

9.1 Basic operations

9.1.1 Binary logic operations

Operation	Parameter	Cycle time (µs)	Operation code		Condition Codes ²⁾					Function	
			Byte 0	Byte 1	1 depends on	2 FIB	3 CC0	4 CC1	5 OVR		
9.1.1 Binary logic operations											
				Bit addr.	Byte addr.		AND logic				
A	I	0.0 to 127.7 ¹⁾	6.9	C 1100	0 0XXX	0 0XXX	0 XXXX	2	1;2	Scan input for "1"	
A	Q	0.0 to 127.7 ¹⁾	6.9	C 1100	0 0XXX	8 1XXX	0 XXXX	2	1;2	Scan output for "1"	
A	F	0.0 to 255.7	7.1	8 1000	0 0XXX	0 XXXX	0 XXXX	2	1;2	Scan flag for "1"	
AN	I	0.0 to 127.7 ¹⁾	7.4	E 1110	0 0XXX	0 0XXX	0 XXXX	2	1;2	Scan input for "0"	
AN	Q	0.0 to 127.7 ¹⁾	7.4	E 1110	0 0XXX	8 1XXX	0 XXXX	2	1;2	Scan output for "0"	
AN	F	0.0 to 255.7	7.1	A 1010	0 0XXX	0 XXXX	0 XXXX	2	1;2	Scan flag for "0"	
				Bit addr.	Byte addr.		OR logic				
O	I	0.0 to 127.7 ¹⁾	7.2	C 1100	8 1XXX	0 0XXX	0 XXXX	2	1;2	Scan input for "1"	
O	Q	0.0 to 127.7 ¹⁾	7.2	C 1100	8 1XXX	8 1XXX	0 XXXX	2	1;2	Scan output for "1"	
O	F	0.0 to 255.7	7.4	8 1000	8 1XXX	0 XXXX	0 XXXX	2	1;2	Scan flag for "1"	
ON	I	0.0 to 127.7 ¹⁾	7.4	E 1110	8 1XXX	0 0XXX	0 XXXX	2	1;2	Scan input for "0"	
ON	Q	0.0 to 127.7 ¹⁾	7.4	E 1110	8 1XXX	8 1XXX	0 XXXX	2	1;2	Scan output for "0"	
ON	F	0.0 to 255.7	7.7	A 1010	8 1XXX	0 XXXX	0 XXXX	2	1;2	Scan flag for "0"	
				Word address		AND logic					
A	T	1 to 127	7.4	F 1111	8 1000	0 XXXX	0 XXXX	2	1;2	Scan timer for "1"	
AN	T	1 to 127	7.7	F 1111	C 1100	0 XXXX	0 XXXX	2	1;2	Scan timer for "0"	
A	C	1 to 127	7.7	B 1011	8 1000	0 XXXX	0 XXXX	2	1;2	Scan counter for contents > 0	
AN	C	1 to 127	7.7	B 1011	C 1100	0 XXXX	0 XXXX	2	1;2	Scan counter for contents = 0	
				Word address		OR logic					
O	T	1 to 127	7.7	F 1111	9 1001	0 XXXX	0 XXXX	2	1;2	Scan timer for "1"	
ON	T	1 to 127	8.0	F 1111	D 1101	0 XXXX	0 XXXX	2	1;2	Scan timer for "0"	
O	C	1 to 127	8.0	B 1011	9 1001	0 XXXX	0 XXXX	2	1;2	Scan counter for contents > 0	
ON	C	1 to 127	8.2	B 1011	D 1101	0 XXXX	0 XXXX	2	1;2	Scan counter for contents = 0	

1) The input and output bytes (words) 64–127 (64–126) and the peripheral bytes (words) 64–127 (64–126) can be used as additional flag bits, bytes and words, as the maximum peripheral configuration cannot exceed 64 input/output bytes.

2) RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of STEP 5 operations

9.1 Basic operations

9.1.1 Binary logic operations

9.1.2 Setting/resetting operations

9.1.3 Timer and counter operations

Operation	Parameter	Cycle time (μs)	Operation code		Condition Codes ²⁾					Function
					RLO	FIB	CC0	CC1	OVR	
			1	2	3	4	5			
AND/OR logic										
O	_____	4.7	F 1111	B 1011	0 _____	2	1;2	ORing of AND functions		
O(_____	8.5	B 1011	B 1011	0 _____	2	1;2	ORing of bracketed expressions		
A(_____	8.8	B 1011	A 1010	0 _____	2	1;2	ANDing of bracketed expressions		
)	_____	8.5	B 1011	F 1111	0 _____	2	1;2	Right parenthesis		

9.1.2 Setting/resetting operations

			Bit addr.		Byte address					Function
S	I	0.0 to 127.7 ¹⁾	8.0	D 1101	0 0XXX	0 0XXX	0 XXXX	1	2	Set input to "1"
S	Q	0.0 to 127.7 ¹⁾	8.0	D 1101	0 0XXX	8 1XXX	0 XXXX	1	2	Set output to "1"
S	F	0.0 to 255.7	7.1	9 1001	0 0XXX	0 XXXX	0 XXXX	1	2	Set flag to "1"
R	I	0.0 to 127.7 ¹⁾	8.0	F 1111	0 0XXX	0 0XXX	0 XXXX	1	2	Set input to "0"
R	Q	0.0 to 127.7 ¹⁾	8.0	F 1111	0 0XXX	8 1XXX	0 XXXX	1	2	Set output to "0"
R	F	0.0 to 255.7	7.4	B 1011	0 0XXX	0 XXXX	0 XXXX	1	2	Set flag to "0"
=	I	0.0 to 127.7 ¹⁾	7.7	D 1101	8 1XXX	0 0XXX	0 XXXX	1	2	Set input to "1" conditionally
=	Q	0.0 to 127.7 ¹⁾	7.7	D 1101	8 1XXX	8 1XXX	0 XXXX	1	2	Set output to "1" conditionally
=	F	0.0 to 255.7	6.8	9 1001	8 1XXX	0 XXXX	0 XXXX	1	2	Set flag to "1" conditionally

9.1.3 Timer and counter operations

					Word address					Function
SP	T	1 to 127	18.7	3 0011	4 0100	0 XXXX	0 XXXX	1	2	Start timer as pulse
SE	T	1 to 127	17.9	1 0001	C 1100	0 XXXX	0 XXXX	1	2	Start timer as extended pulse
SR	T	1 to 127	18.1	2 0010	4 0100	0 XXXX	0 XXXX	1	2	Start timer as ON delay
SS	T	1 to 127	18.1	2 0010	C 1100	0 XXXX	0 XXXX	1	2	Start timer as stored ON delay
SF	T	1 to 127	17.9	1 0001	4 0100	0 XXXX	0 XXXX	1	2	Start timer as OFF delay
R	T	1 to 127	10.4	3 0011	C 1100	0 XXXX	0 XXXX	1	2	Reset timer
S	C	1 to 127	19	5 0101	C 1100	0 XXXX	0 XXXX	1	2	Set counter
R	C	1 to 127	10.4	7 0111	C 1100	0 XXXX	0 XXXX	1	2	Reset counter
CU	C	1 to 127	17.0	6 0110	C 1100	0 XXXX	0 XXXX	1	2	Count up
CD	C	1 to 127	17.0	5 0101	4 0100	0 XXXX	0 XXXX	1	2	Count down

- 1) The input and output bytes (words) 64–127 (64–126) and the peripheral bytes (words) 64–127 (64–126) can be used as additional flag bits, bytes and words, as the maximum peripheral configuration cannot exceed 64 input/output bytes.
- 2) RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of STEP 5 operations

9.1 Basic operation

9.1.4 Loading and transfer functions

Operation	Parameter	Cycle time (µs)	Operation code Byte 0Byte 1		Condition Codes ²⁾					Function
					RLO	FB	CO0	CC1	OV	
					1 depends on	2	3	4 affects	5	
9.1.4 Loading and transfer functions				Byte/Word address						
L	IB	0 to 127 ¹⁾	7.4	4 0100	A 1010	0 0XXX	0 XXXX	—	—	Load input byte of process input image into accu 1
L	IW	0 to 126 ¹⁾	9.3	5 0101	2 0010	0 0XXX	0 XXXX	—	—	Load input word of process input image into accu 1
L	QB	0 to 127 ¹⁾	7.4	4 0100	A 1010	8 1XXX	0 XXXX	—	—	Load output byte of process output image into accu 1
L	QW	0 to 126 ¹⁾	9.3	5 0101	2 0010	8 1XXX	0 XXXX	—	—	Load output word of process output image into accu 1
L	FB	0 to 255	7.7	0 0000	A 1010	0 XXXX	0 XXXX	—	—	Load flag byte into accu 1
L	FW	0 to 254	9.3	1 0001	2 0010	0 XXXX	0 XXXX	—	—	Load flag word into accu 1
L	DR	1 to 255	12.1	2 0010	A 1010	0 XXXX	0 XXXX	—	—	Load datum (right-hand byte) of current data block into accu 1
L	DL	1 to 255	13.2	2 0010	2 0010	0 XXXX	0 XXXX	—	—	Load datum (left-hand byte) of current data block into accu 1
L	DW	1 to 255	16.5	3 0011	2 0010	0 XXXX	0 XXXX	—	—	Load datum (word) of current actual data block into accu 1
L	T	1 to 127	11	0 0000	2 0010	0 XXXX	0 XXXX	—	—	Load time (binary) of timer into accu 1
L	C	1 to 127	10.4	4 0100	2 0010	0 XXXX	0 XXXX	—	—	Load counter (binary) of counter into accu 1
L	PB	0 to 127 ¹⁾	50	7 0111	2 0010	0 XXXX	0 XXXX	—	—	Load peripheral byte of digital inputs into accu 1, bypassing the process image
L	PW	0 to 126 ¹⁾	52	7 0111	A 1010	0 XXXX	0 XXXX	—	—	Load peripheral word of digital inputs/outputs into accu 1, bypassing the process image
LD	T	1 to 127	23.1	0 0000	C 1100	0 XXXX	0 XXXX	—	—	Load time (BCD) of timer into accu 1
LD	C	1 to 127	22.8	4 0100	C 1100	0 XXXX	0 XXXX	—	—	Load count (BCD) of counter into accu 1
T	IB	0 to 127 ¹⁾	7.1	4 0100	B 1011	0 0XXX	0 XXXX	—	—	Transfer contents of accu 1 into input byte of process input image
T	IW	0 to 126 ¹⁾	7.7	5 0101	3 0011	0 0XXX	0 XXXX	—	—	Transfer contents of accu 1 into input word of process input image
T	QB	0 to 127 ¹⁾	7.1	4 0100	B 1011	8 1XXX	0 XXXX	—	—	Transfer contents of accu 1 into output byte of process output image
T	QW	0 to 126 ¹⁾	7.7	5 0101	3 0011	8 1XXX	0 XXXX	—	—	Transfer contents of accu 1 into output word of process output image
T	FB	0 to 255	7.1	0 0000	B 1011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 into flag byte
T	FW	0 to 254	8.0	1 0001	3 0011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 into flag byte
T	DR	1 to 255	14.3	2 0010	B 1011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 into the word (right-hand byte) of current actual data block
T	DL	1 to 255	13.2	2 0010	3 0011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 into the word (left-hand byte) of current data block
T	DW	1 to 255	13	3 0011	3 0011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 into the word of current data block
T	PB	0 to 127 ¹⁾	53	7 0111	3 0011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 directly into peripheral byte
T	PW	0 to 126 ¹⁾	55	7 0111	B 1011	0 XXXX	0 XXXX	—	—	Transfer contents of accu 1 directly into peripheral word

2) RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of the STEP 5 operations

9.1 Basic operations

9.1.4 Loading and transfer functions

9.1.5 Comparison functions

Operation	Parameter	Cycle time (μs)	Operation code				Condition Codes ²⁾					Function
			Byte 0		Byte 1		RLO depends on	FIB 2	CC0 3	CC1 4	OVR 5	
L	KB	0 to 255	5.0	2 0010	8 1000	0 XXXX	0 XXXX	—	—	—	—	Load constant number (1 byte) into accu 1
L	KS*	2 ASCII-symbols	6.6	3 0011	0 0000	1 0001	0 0000	—	—	—	—	Load constant character into accu 1
L	KF*	−32768 to +32767	6.6	3 0011	0 0000	0 0000	4 0100	—	—	—	—	Load constant fixed-point number into accu 1
L	KH*	0 to FFFF	6.6	3 0011	0 0000	4 0100	0 0000	—	—	—	—	Load constant number (hexadecimal code) into accu 1
L	KM*	0000 ... 00 to 111 ... 11	6.6	3 0011	0 0000	8 1000	0 0000	—	—	—	—	Load constant bit pattern of a word (2 bytes) into accu 1
L	KY*	0 to 255, 0 to 255	6.6	3 0011	0 0000	2 0010	0 0000	—	—	—	—	Load constant number (2 bytes) into accu 1
L	KT*	0.0 to 999.3	6.6	3 0011	0 0000	0 0000	2 0010	—	—	—	—	Load constant number (2 bytes) as time into accu 1
L	KC*	0.0 to 999	6.6	3 0011	0 0000	0 0000	1 0001	—	—	—	—	Load constant number (2 bytes) as counter into accu 1

* These are 4-byte operations. The constant is in byte 2 and in byte 3.

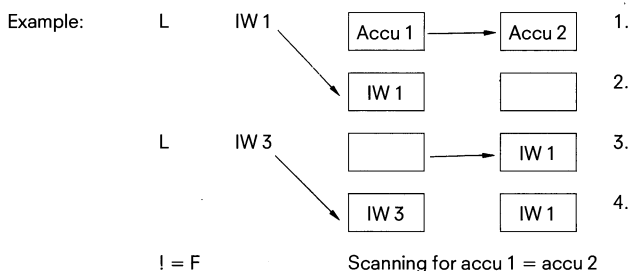
9.1.5 Comparison functions

!= F	—	10.7	2 0010	1 0001	8 1000	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 equal to accu 1. If equal, RLO = "1".
>< F	—	10.7	2 0010	1 0001	6 0110	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 not equal to accu 1. If unequal, RLO = "1".
> F	—	10.7	2 0010	1 0001	2 0010	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 > accu 1. If accu 2 > accu 1, RLO = "1".
< F	—	10.7	2 0010	1 0001	4 0100	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 < accu 1. If accu 2 < accu 1, RLO = "1".
>= F	—	10.7	2 0010	1 0001	A 1010	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 ≥ accu 1. If accu 2 ≥ accu 1, RLO = "1".
<= F	—	10.7	2 0010	1 0001	C 1100	0 —	—	1;2;3;4	Fixed-point comparison for accu 2 ≤ accu 1. If accu 2 ≤ accu 1, RLO = "1".

Note:

The programmable controller has two accumulators for comparison and arithmetic functions and for digital operations.

Loading means that the contents of accu 1 are transferred to accu 2 and that accu 1 is newly loaded according to the operands in the load operation. After two load operations, information on the contents of the accumulators can be obtained with comparison operations.



A transfer operation always transfers the contents of accu 1 to the operands specified in the transfer operation.

2) RLO ≙ status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of STEP 5 operations

9.1 Basic operations

9.1.6 Block calls

9.1.7 Other commands

Operation	Parameter	Cycle time (µs)	Operation code		Condition Codes ²⁾					Function
			Byte 0	Byte 1	RLO	FIB	CC0	CC1	OVR	
					1 depends on	2	3	4	5 affects	

9.1.6 Blocks calls

				Word address						Function
Operation	Parameter	Cycle time (µs)	Byte 0	Byte 1	Word address	RLO	FIB	CC0	CC1	
JU	PB	0 to 127	24.8	7 0111 5 0101	0 0 XXXX XXXX	—	2			Jump unconditionally to program block
JU	FB	0 to 47	25.6	3 0011 D 1101	0 0 XXXX XXXX	—	2			Jump unconditionally to function block
JC	PB	0 to 127	25	5 0101 5 0101	0 0 XXXX XXXX	1	1;2			Jump conditionally to program block
JC	FB	0 to 47	26	1 0001 D 1101	0 0 XXXX XXXX	1	1;2			Jump conditionally to function block
C	DB	1 to 63	8.8	2 0010 0 0000	0 0 XXXX XXXX	—	—			Call data block; the data block is valid until another data block has been called.
BE	—	18.4	6 0110 5 0101	0 0 0000 0000	—	2				End of block
BEU	—	18.4	6 0110 5 0101	0 1 0000 0001	—	2				Unconditional end of block – may be programmed several times within a block
BEC	—	18.6	0 0000 5 0101	0 0 0000 0000	1	1;2				Conditional end of block RLO = "1"

9.1.7 Other commands

NOP 0	—	3.5	0 0000 0 0000	0 0 0000 0000	—	—				No operation (all bits deleted)
NOP 1	—	3.5	F 1111 F 1111	F 1111 F 1111	—	—				No operation (all bits set)
STP		6.0	7 0111 0 0000	0 3 0000 0011	—	—				Programmable stop operation (at the end of the cycle, the programmable controller stops)

2) RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of STEP 5 operations

9.2 Supplementary operations

9.2.1 Binary logic functions

Operation	Parameter	Cycle time (µs)	Operation code				Condition codes ²⁾					Function
			Byte 0	Byte 1	Byte 2	Byte 3	1 RLO	2 FIB	3 CC0	4 CC1	5 OVR	
							Depends on			Affects		

9.2.1 Binary logic function

				Bit address	Word address			
TB	DW ³⁾	1.0 to 255.15	22.7	7 0 0111 0000 C 0 1100 XXXX	4 6 0100 0110 0 0 XXXX XXXX	—	1;2	Test bit of the data word for "1"
TB	C ³⁾	1.0 to 127.15	21.2	7 0 0111 0000 C 0 1100 XXXX	1 5 0001 0101 0 0 XXXX XXXX	—	1;2	Test bit of the counter word for "1"
TB	T ³⁾	1.0 to 127.15	21.2	7 0 0111 0000 C 0 1100 XXXX	2 5 0010 0101 0 0 XXXX XXXX	—	1;2	Test bit of the timer word for "1"

TBN	DW ³⁾	1.0 to 255.15	22.5	7 0 0111 0000 8 0 1000 XXXX	4 6 0100 0110 0 0 XXXX XXXX	—	1;2	Test bit of the data word for "0"
TBN	C ³⁾	1.0 to 127.15	21.0	7 0 0111 0000 8 0 1000 XXXX	1 5 0001 0101 0 0 XXXX XXXX	—	1;2	Test bit of the counter word for "0"
TBN	T ³⁾	1.0 to 127.15	21.0	7 0 0111 0000 8 0 1000 XXXX	2 5 0010 0101 0 0 XXXX XXXX	—	1;2	Test bit of the timer word for "0"

SU	DW ³⁾	1.0 to 255.15	22.7	7 0 0111 0000 4 0 0100 XXXX	4 6 0100 0110 0 0 XXXX XXXX	—	2	Set bit of data word unconditionally to "1"
SU	C ³⁾	1.0 to 127.15	21.2	7 0 0111 0000 4 0 0100 XXXX	1 5 0001 0101 0 0 XXXX XXXX	—	2	Set bit of counter word unconditionally to "1"
SU	T ³⁾	1.0 to 127.15	21.2	7 0 0111 0000 4 0 0100 XXXX	2 5 0010 0101 0 0 XXXX XXXX	—	2	Set bit of timer word unconditionally to "1"

RU	DW ³⁾	1.0 to 255.15	22.5	7 0 0111 0000 0 0 0000 XXXX	4 6 0100 0110 0 0 XXXX XXXX	—	2	Set bit of data word unconditionally to "0"
RU	C ³⁾	1.0 to 127.15	21.0	7 0 0111 0000 0 0 0000 XXXX	1 5 0001 0101 0 0 XXXX XXXX	—	2	Set bit of counter word unconditionally to "0"
RU	T ³⁾	1.0 to 127.15	21.0	7 0 0111 0000 0 0 0000 XXXX	2 5 0010 0101 0 0 XXXX XXXX	—	2	Set bit of timer word unconditionally to "0"

²⁾ RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

³⁾ These are 4-byte-operations.

9. Total overview of STEP 5 commands

9.2 Supplementary operations

9.2.2 Digital logic functions

9.2.3 Arithmetic functions

9.2.4 Jump functions

9.2.5 Timer and counter functions

Operation	Parameter	Cycle time (μs)	Operation code		Condition Codes ²⁾					Function
			Byte 0	Byte 1	1 depends on	2 FIB	3 CC0	4 CC1	5 OVR	

9.2.2 Digital logic functions

AW	_____	6.0	4 0100	1 0001	0 _____	_____	3;4			Digital ANDing of accu 1 and accu 2 (word for word); result stored in accu 1;
OW	_____	8.2	4 0100	9 1001	0 _____	_____	3;4			Digital ORing of accu 1 and accu 2 (word for word); result stored in accu 1;
XOW	_____	6.5	5 0101	1 0001	0 _____	_____	3;4			Digital EXORing of accu 1 and accu 2 (word for word); result stored in accu 1;

9.2.3 Arithmetic functions

+ F	_____	9.9	7 0111	9 1001	0 _____	_____	3;4;5			Add accu 1 to accu 2; result stored in accu 1;
– F	_____	12.1	5 0101	9 1001	0 _____	_____	3;4;5			Subtract accu 1 from accu 2; result stored in accu 1;

9.2.4 Jump functions

				Wordaddress ± 127						
JU =	“Label” (4 ASCII-symbols)	9.9	2 0010	D 1101	0 XXXX	0 XXXX	_____	_____		Jump unconditionally to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JC =	“Label” (4 ASCII-symbols)	10.4	F 1111	A 1010	0 XXXX	0 XXXX	1	1;2		Jump conditionally (if RL0 = “1”) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JP =	“Label” (4 ASCII-symbols)	10.4	1 0001	5 0101	0 XXXX	0 XXXX	3;4	_____		Jump conditionally (if result > zero) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JM =	“Label” (4 ASCII-symbols)	10.2	2 0010	5 0101	0 XXXX	0 XXXX	3;4	_____		Jump conditionally (if result < zero) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JZ =	“Label” (4 ASCII-symbols)	10.4	4 0100	5 0101	0 XXXX	0 XXXX	3;4	_____		Jump conditionally (if result = zero) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JN =	“Label” (4 ASCII-symbols)	10.4	3 0011	5 0101	0 XXXX	0 XXXX	3;4	_____		Jump conditionally (if result ≠ zero) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.
JO =	“Label” (4 ASCII-symbols)	10.4	0 0000	D 1101	0 XXXX	0 XXXX	5	_____		Jump conditionally (if condition code OVR = 1) to label, consisting of 4 ASCII symbols. Jump displacement ≤ ± 127 words.

9.2.5 Timer and counter functions

				Wordaddress						
FRT	0 to 127	8.8	0 0000	4 0100	0 XXXX	0 XXXX	1	2		Enable timer for cold restart (only on positive-going edge of RL0)
FRC	0 to 127	8.8	4 0100	4 0100	0 XXXX	0 XXXX	1	2		Enable counter for cold restart (only on positive-going edge of RL0)

2) RLO ≙ status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

9. Total overview of STEP 5 commands

9.2 Supplementary operations

9.2.6 Shift functions

9.2.7 Conversion functions

9.2.8 Decrementing/incrementing

9.2.9 Process function

9.2.10 Disable/enable command output

9.2.11 Disable/enable interrupts

Operation	Parameter	Cycle time (μs)	Operation code		Condition Codes ²⁾					Function
					Byte 0	Byte 1	1	2	3	
			depends on	affects	RL0	FIB	CC0	CC1	OVR	

9.2.6 Shift functions

						Par.				
SLW	0 to 15	—	6 0110	1 0001	0 0000	0 XXXX	—	—	3,4	Shift contents of accu 1 to the left. The bit positions to the right which become vacant are padded with zeros.
SRW	0 to 15	—	6 0110	9 1001	0 0000	0 XXXX	—	—	3,4	Shift contents of accu 1 to the right. The bit positions to the left which become vacant are padded with zeros.

9.2.7 Conversion functions

CFW	—	4.7	0 0000	1 0001	0 —	0 —	—	—	—	One's complement of accu 1
CSW	—	8.2	0 0000	9 1001	0 —	0 —	—	—	3,4	Two's complement of accu 1;

9.2.8 Decrementing/incrementing

						Dec./Inc. 0 to 255				
D	1 to 255	5.7	1 0001	9 1001	0 XXXX	0 XXXX	—	—	—	Decrement only the low byte of accu 1 by a particular value.
I	1 to 255	4.4	1 0001	1 0001	0 XXXX	0 XXXX	—	—	—	Increment only the low byte of accu 1 by a particular value.

9.2.9 Process functions

						Wordaddress				
DO	FW ⁴⁾	0 to 254	9.9	4 0100	E 1110	0 XXXX	0 XXXX	—	—	Process flag word. The next operation specified is combined with the parameter in the flag word and executed.
DO	DW ⁴⁾	0 to 255	12.1	6 0110	E 1110	0 XXXX	0 XXXX	—	—	Process data word. The next operation specified is combined with the parameter in the data word and executed.

9.2.10 Disable/enable command output

BAS	—	5.5	B 1011	E 1110	0 —	0 —	1	—	—	Disable command output
BAF	—	5.2	F 1111	E 1110	0 —	0 —	1	—	—	Enable command output

9.2.11 Disable/enable interrupts

IA	—	5.0	0 0000	8 1000	0 0000	0 0000	—	—	—	Inhibit interrupt processing
RA	—	5.0	0 0000	8 1000	8 1000	0 0000	—	—	—	Enable interrupt processing

2) RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

4) These are four byte statements; the operation code is in byte 2, the parameters of the Statement to be executed are in byte 3.

9. Total overview of STEP 5 commands

9.2 Supplementary operations

9.2.12 Substitution functions

Operation	Parameter	Cycle time (µs)	Operation code		Condition codes ²⁾					Function
					RLO	FIB	CC0	CC1	OVR	
			Byte 0	Byte 1	1	2	3	4	5	
					Depends on	Affects				

6.2.12 Substitution functions

6.2.12 Substitution functions				Parameter address (hex.)		AND/OR logic functions				
A	=	Formal operand (4 ASCII characters)	X+14.6 ⁵⁾	0 0000	7 0111	0 00XX	0 XXXX	2	1;2	AND function; scan formal operand for "1"
AN	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	2 0010	7 0111	0 00XX	0 XXXX	2	1;2	AND function; scan formal operand for "0"
O	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	0 0000	F 1111	0 00XX	0 XXXX	2	1;2	OR function; scan formal operand for "1"
ON	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	2 0010	F 1111	0 00XX	0 XXXX	2	1;2	OR function; scan formal operand for "0"

Setting functions

S	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	1 0001	7 0111	0 00XX	0 XXXX	1	2	Set (binary) formal operand to "1"
=	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	1 0001	F 1111	0 00XX	0 XXXX	1	2	Set formal operand (restricted to "1")
RB	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	3 0011	7 0111	0 00XX	0 XXXX	1	2	Set (binary) formal operand to "0"
RD	=	Formal operand (4 ASCII characters)	X+14.2 ⁵⁾	3 0011	E 1110	0 00XX	0 XXXX	1	2	Set (digital) formal operand to "0" [operands T and C]

Load and transfer functions

L	=	Formal operand (4 ASCII characters)	X+15.5 ⁵⁾	4 0100	6 0110	0 00XX	0 XXXX	—	—	Load formal operand. The value of the operand specified as formal operand is loaded into accu 1 [operands IB, IW, FB, FW, QB, QW, DR, DL, DW, PB, PW].
LD	=	Formal operand (4 ASCII characters)	X+6.8 ⁵⁾	0 0000	E 1110	0 00XX	0 XXXX	—	—	Load formal operand in BCD code. The value of the timer or counter location specified as formal operand is loaded into accu 1 [operands T, C].
LW	=	Formal operand (4 ASCII characters)	8.5	3 0011	F 1111	0 00XX	0 XXXX	—	—	Load the bit pattern of a formal operand into accu 1 [operands KB, KS, KF, KH, KM, KY, KT, KC].
T	=	Formal operand (4 ASCII characters)	X+14.5 ⁵⁾	6 0110	6 0110	0 00XX	0 XXXX	—	—	Transfer to a formal operand. The contents of accu 1 are transferred to the operand specified as formal operand [operands IB, IW, FB, FW, QB, QW, DR, DL, DW, PB, PW].

²⁾ RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

⁵⁾ X signifies the cycle time of the command to be substituted.

Total overview of STEP 5 commands

9.2 Supplementary operations

9.2.12 Substitution functions

Operation	Parameter	Cycle time (µs)	Operation code		Condition codes ²⁾					Function
			Byte 0	Byte 1	1 RLO	2 FIB	3 CC0	4 CC1	5 OVR	
					Depends on	Affects				
				Parameter address (hex.)		Timer/counter functions				
SP	=	Formal operand (4 ASCII characters)	X+5.5 ⁵⁾	3 0011 6 0110	0 00XX 0 XXXX	1	2	Start timer specified as formal operand with the value stored as pulse [operand T].		
SEC	=	Formal operand (4 ASCII characters)	X+6.7 ⁵⁾	1 0001 E 1110	0 00XX 0 XXXX	1	2	Start timer specified as formal operand with the value stored as extended pulse or set counter specified as formal operand with the count specified [operands C, T].		
SI	=	Formal operand (4 ASCII characters)	X+5.6 ⁵⁾	2 0010 6 0110	0 00XX 0 XXXX	1	2	Start timer specified as formal operand with the value stored as "on" delay [operand T].		
SFD	=	Formal operand (4 ASCII characters)	X+6.8 ⁵⁾	1 0001 6 0110	0 00XX 0 XXXX	1	2	Start timer specified as formal operand with the value stored as stored "off" delay or decrement counter specified as formal operand [operands C, T].		
SSU	=	Formal operand (4 ASCII characters)	X+6.8 ⁵⁾	2 0010 E 1110	0 00XX 0 XXXX	1	2	Start timer specified as formal operand with the value loaded as stored "on" delay or increment counter specified as formal operand [operands C, T].		
FR	=	Formal operand (4 ASCII characters)	X+6.8 ⁵⁾	0 0000 6 0110	0 00XX 0 XXXX	1	2	Enable formal operand for cold restart [operands C, T].		
Processing function										
DO	=	Formal operand (4 ASCII characters)	X+8.5 ⁵⁾	7 0111 6 0110	0 00XX 0 XXXX	—	—	Process formal operand. Only C DB, JU PB, and JU FB can be substituted.		

²⁾ RLO \triangleq status; FIB = 0 means current logic operation; FIB = 1 means first operation scan; CC1 CC0 = 00 result or accu 1 = 0, CC1 CC0 = 01 result or accu 1 less than 0, CC1 CC0 = 10 result or accu 1 greater than 0; OVR (overflow) = 1 means in the case of arithmetic statements that the value is too large for the accu.

⁵⁾ X signifies the cycle time of the command to be substituted.

