

SIEMENS

SIMATIC S5-101U

Programmable Controller

Programming Instructions

Order No.: GWA 4NEB 810 2120-02 a

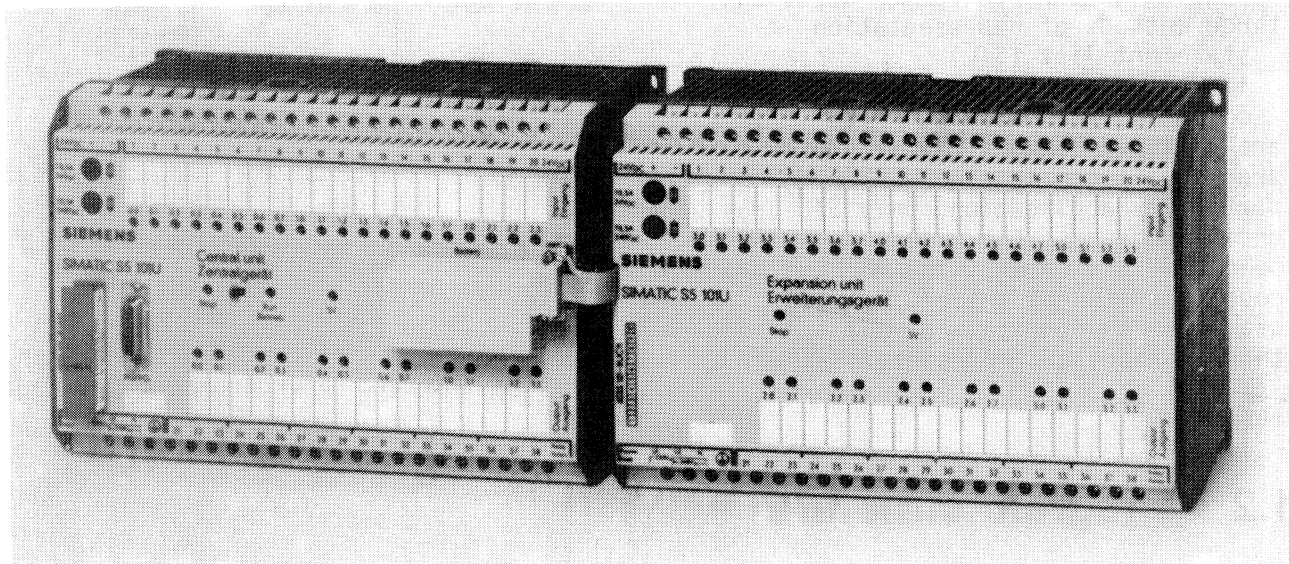


Fig. 1 S5-101U programmable controller

CONTENTS	Page		Page
1. THE PROGRAMMING LANGUAGE		4.2.1 Search function	4.2
1.1 STEP 5 programming language	1.1	4.2.2 Signal status display	4.2
1.2 Program structure	1.1	4.2.3 Forcing of outputs and flags	4.2
2. PRINCIPLE OF OPERATION OF THE PC		4.2.4 Forcing of timers and counters	4.3
2.1 Program processing	2.1	5. PROGRAMMING EXAMPLES	
2.2 "RUN and "STOP" modes	2.2	5.1 Basic operations	
2.3 Memories	2.2	5.1.1 Binary logic operations	5.1
3. NOTES ON PROGRAM DEVELOPMENT		5.1.2 Setting/resetting operations	5.4
3.1 Power-up	3.1	5.1.3 Load and transfer operations	5.6
3.2 Battery monitoring	3.2	5.1.4 Timer functions	5.8
3.3 Retentive/non-retentive flags	3.2	5.1.5 Counter functions	5.12
3.4 Interrupt processing	3.2	5.1.6 Comparison (relational) operations	5.14
3.5 Intercompatibility between LAD, CSF and STL	3.4	5.1.7 Arithmetic operations	5.17
3.6 Operation in the SINEC L1 local area network	3.7	5.1.8 Other functions	5.17
4. PROGRAM START-UP		5.2 Supplementary operations	5.18
4.1 Loading and dumping a program	4.1	5.2.1 Logic operations (word mode)	5.18
4.2 Program test	4.2	5.2.2 Conversion functions	5.19
		5.2.3 Shift operations	5.19
		5.2.4 Jump operations	5.20
		5.2.5 Condition codes	5.22
		6. OPERATION SET	6.1

1. The programming language

1.1 STEP 5 programming language

The user programs are written in the STEP 5 programming language. The statements of this language permit not only the programming of simple binary functions but also the programming of complex digital functions. Depending on the programmer used, all three methods of representation are possible so that the method of programming can be adapted to the particular application. Only STL programming is possible with the hand-held 605U programmer. The machine code generated by the 670/675 programmers is identical for all three methods of representation.


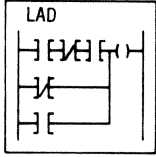
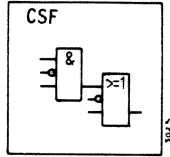
Statement list	Ladder diagram	Control system flowchart
Programming with mnemonics of the function designations	Programming with graphic symbols similar to schematic circuit diagrams	Programming with graphic symbols
to DIN 19 239 (draft)	to DIN 19 239 (draft)	to IEC 117-15 DIN 40 700 DIN 40 719 DIN 19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

Fig. 2: Methods of representation with the STEP 5 programming language

1.2 Program structure

The user program consists of up to 1024 statements and can be written as a program block (PB) or function block (FB). Only PB1 or FB1 can be executed on the S5-101U programmable controller.

Program block

A program block can be programmed and documented in all three methods of representation (STL, LAD and CSF). A program block can be translated from one method of representation into the two other methods with the 670/675 programmers provided certain programming rules are observed (see Section 3.4). For users familiar with contactors and relays, the LAD method is recommended since the ladder diagram has very close similarities with schematic circuit diagrams.

Program blocks are used especially when a CRT-based programmer is available and programming or documentation is to be made in graphic form.

Note: Supplementary operations must not be used in PB1.

Function block

Function blocks can only be written and documented in STL form. Jump operations make it possible to enhance the structuring of the user program and thus also its capabilities. Short, constant response times to interrupts can be implemented with load and transfer operations in conjunction with jump operations (see Section 3.4).

2. Principle of operation

2.1 Program processing

The control functions of the 101U are defined by a user program. In order to be able to scan the user program cyclically statement by statement, the CPU has to perform the following functions:

1. In the case of a cold restart (power switch from "Off" to "On" or mode selector from "Stop" to "Run"), the process output image* is erased, i.e. all outputs are set to zero.
2. The process input image* is updated, i.e. all signal statuses of the inputs are scanned and written into the process input image.
3. The user program (PB1 or FB1) is scanned and processed statement by statement. When scanning the signal statuses of the inputs, the CPU accesses the process input image and not the actual inputs. When latching and unlatching the outputs (coils), only the process output image is overwritten to begin with.
4. Once the user program has been processed, the process output image is transferred to the actual outputs.
5. Points 2, 3 and 4 are handled cyclically.

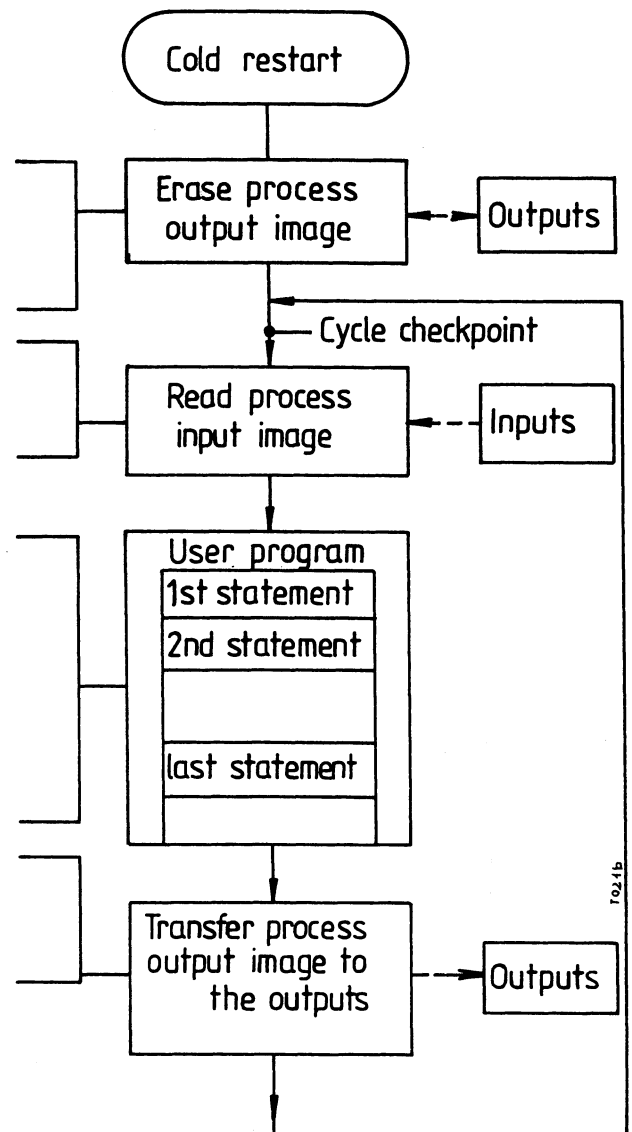


Fig. 3: Principle of operation of the S5-101U

A scanning operation from cycle checkpoint to cycle checkpoint takes approx. 70 ms for 1024 statements (binary). If a scanning cycle is not completed within 300 ms due to program errors or faults, an internal monitor responds, the PC enters the "Stop" status and all outputs (coils) are switched off.

* Process I/O image:
Internal memory area in which the signal status ("0" or "1") of the inputs/outputs is stored.

2.2 The "Run" and "Stop" modes

"RUN" mode

In the "RUN" mode, the program is scanned cyclically from cycle checkpoint to cycle checkpoint. The PC is brought into the "RUN" mode by

- switching the mode selector to "RUN"
- selecting the "PC RUN" function of the programmer (mode selector in "RUN" position)
- and on recovery of the power supply if the mode selector is at "RUN" and was in the "RUN" position prior to the power failure.

"STOP" mode

In the "STOP" mode, the program is not scanned and the outputs (coils) are disabled. While the PC is in the "STOP" state, all timers and counters and the process I/O image retain the values or states they had in the last scanning cycle prior to the PC entering the "STOP" state. If the PC is switched to "RUN", the timers and counters (0..7) are reset. The non-retentive flags and the process I/O image are erased.

The PC is brought into the "STOP" mode by

- switching the mode selector to "STOP"
- selecting the "PC STOP" function on the programmer
- faults or errors in program scanning, e.g. time-out or operations that can not be interpreted by the PC.

The cause for the PC entering the "STOP" state can be traced with the aid of the "DISPLAY ISTACK" function of the programmer (see Section 4.2 of Operating Instructions).

2.3 Memories

The PC has an internal program memory, the data of which can be supported for three years by a backup battery. There are also two different memory submodules (see Fig. 4).

The memory submodules are used for program dumping or for copying the program should only one memory submodule be used for a number of PCs.

On power-up or when the PC is switched to "RUN", the contents of the memory submodule are always copied into the internal memory and processed there.

Memory submodule	EPROM	EEPROM
Program dump	PG 615 (with adapter 984-2UA11) PG 670 (with adapter 984-0UA11) PG 675	PG 615 PG 670 (without 984 adapter) PG 675
Program erasure	Only with special UV lamp (erasure time: 30 min)	Direct in the PC with the above programmers and the PG 605U programmer (Programmer function: PG PC)
Program modification via programmer	Only erasure of entire program possible	possible

Fig. 4: Differences between the EPROM and EEPROM submodules.

3. Notes on program development

3.1 Power up

When the power supply is switched on or on recovery of the power supply after a power failure, the PC assumes the following states without having to take any additional measures in the user program:

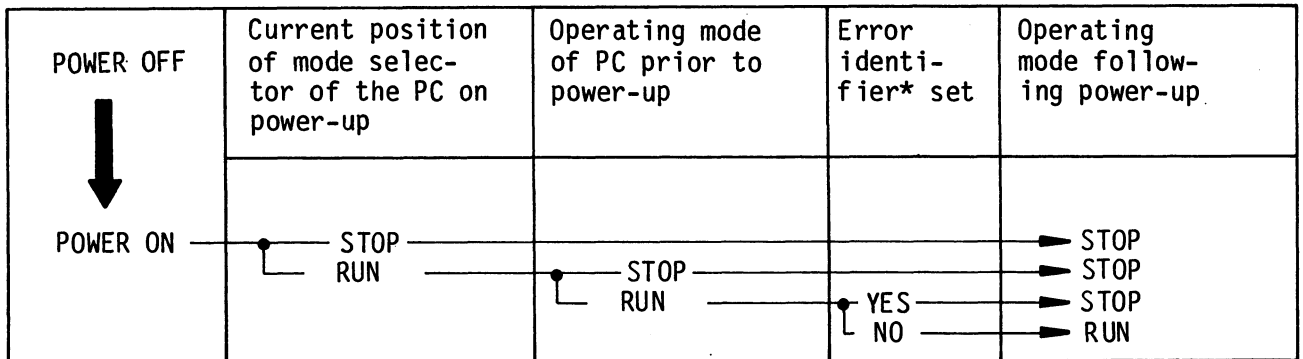


Fig. 5: Automatic mode setting following power-up

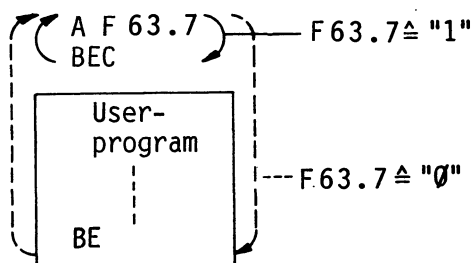
* A fault has occurred in program scanning and the reason for this is stored in the interrupt stack.

Prevention of automatic restart in general

Flag* F 63.7 can be used to prevent automatic restart on power-up. This flag is set by the operating system of the PC on power-up if the "RUN" mode is set and was set prior to power-down. In order to enable manual restart, flag F 63.7 is reset in the "STOP" mode. It can also be reset by the user program (e.g. in conjunction with an input signal).

* Internal relay equivalent

Programming example



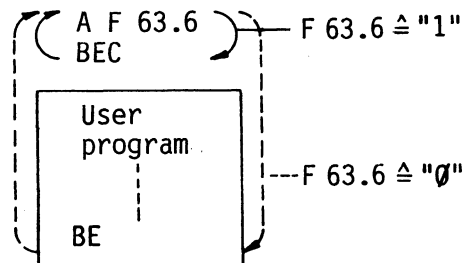
Preventing automatic restart on battery failure

Flag F 63.6 can be used to prevent automatic restart on battery failure (retentive flags reset).

Flag F 63.6 is set by the operating system of the PC on power-up if the backup battery fails or is not connected. The PC must be set to the "RUN" mode in this case.

Flag F 63.6 is reset by the "ERASE PROGRAM" function or by the user program (e.g. in conjunction with an input signal).

Programming example



3.2 Battery monitoring

Flag F 63.6 is used for monitoring the battery.

This flag is set by the operating system of the PC on power recovery and during the normal scanning cycle if failure of the battery backup voltage is detected.

The PC must be in the "RUN" state.

Flag F 63.6 is reset by the "ERASE PROGRAM" function of the programmer or by the user program. The user can therefore determine how the PC is to react to backup battery failure.

3.3 Retentive/non-retentive flags

The S5-101U has a total of 512 flags. The flag area is subdivided as follows:

Retentive flags (F 0.0...F 31.7)

- retain their last state prior to - power-down on power-up (with backup battery only)
- retain their last state when the mode is changed from "STOP" to "RUN" (with and without backup battery)
- are reset like the non-retentive flags on power-up (without backup battery)
- can also be reset by the user program ("ERASE PROGRAM" function).

By using retentive flags, the last status of the plant or machine prior to the PC leaving the "RUN" mode can be stored. On restart, the plant or machine can resume operations at the point at which it was stopped.

Non-retentive flags (F 32.0...F 63.7)

- are reset when the PC mode changes from "STOP" to "RUN" and on power-up.

Flags F 61.0 - F 62.7 are reserved as coordinating flags for operation in the SINEC L1 local area network; flags F 63.0 - F 63.7 are reserved as system flags. Since they are affected by the PC operating system, they must not be used as flags in the normal sense.

3.4 Interrupt processing

When an interrupt signal (e.g. emergency off) from the process is received by the PC, the latter interrupts cyclic scanning of the user program and initiates the processing of a specific interrupt routine.

Interrupt processing with the S5-101U is defined exclusively by the user program so that each input and output can be used for interrupt processing.

In order to achieve minimum response times, the inputs and outputs are referenced direct, i.e. outside cyclic program scanning. The load/transfer operations "LPB" (inputs) and "TPB" (outputs) are available for this purpose.

A more or less constant response time is achieved if the scanning of the inputs programmed by the user as interrupt inputs is uniformly distributed over the entire user program. Fig. 6 shows a user program with interrupt processing.

3.5 Intercompatibility between LAD, CSF and STL

General

Each of the methods of representation in the STEP 5 programming languages has specific properties and limitations.

Consequently, a program block written in STL cannot simply be displayed as an LAD or CSF and the graphic methods of representation, LAD and CSF, may not always be fully compatible. In other words, one form cannot always be translated back into the other form.

If the program has been entered as an LAD or CSF, it can always be translated back into STL form.

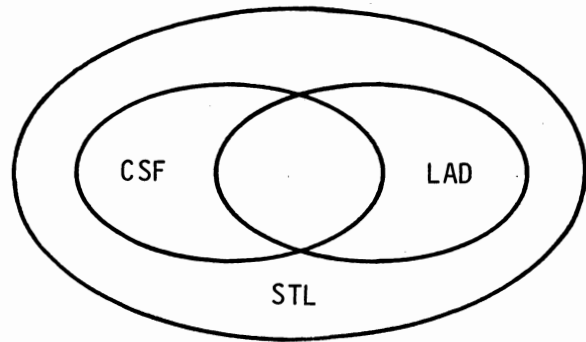


Fig. 7: Range and limitations of the methods of representation in the STEP 5 programming language

The aim of this section is to establish a number of rules, which, if adhered to, will ensure complete compatibility between the three methods of representation. These rules are classified as follows:

- Rules for compatibility between the graphic methods of representation (LAD and CSF).
If these rules are followed, input is possible in one graphic form and display in the others.

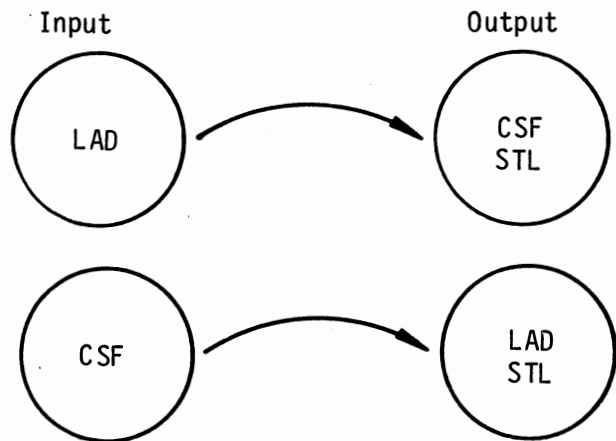


Fig. 8: Graphic input

- Rules for compatibility between the statement list and the graphic methods of representation.
If these rules are observed, it is possible to enter a program in any of the three methods of representation, graphic or not, and to have it displayed in the other two forms.

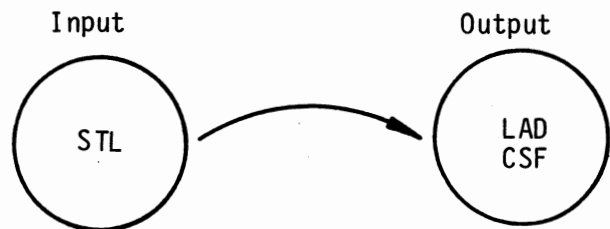


Fig. 9: Input in the form of a statement list

Input as LAD and display as CSF (STL)

Rule: Do not exceed the display boundaries for LAD.
 Excessive nesting may cause the LAD display boundary to be exceeded (8 levels)

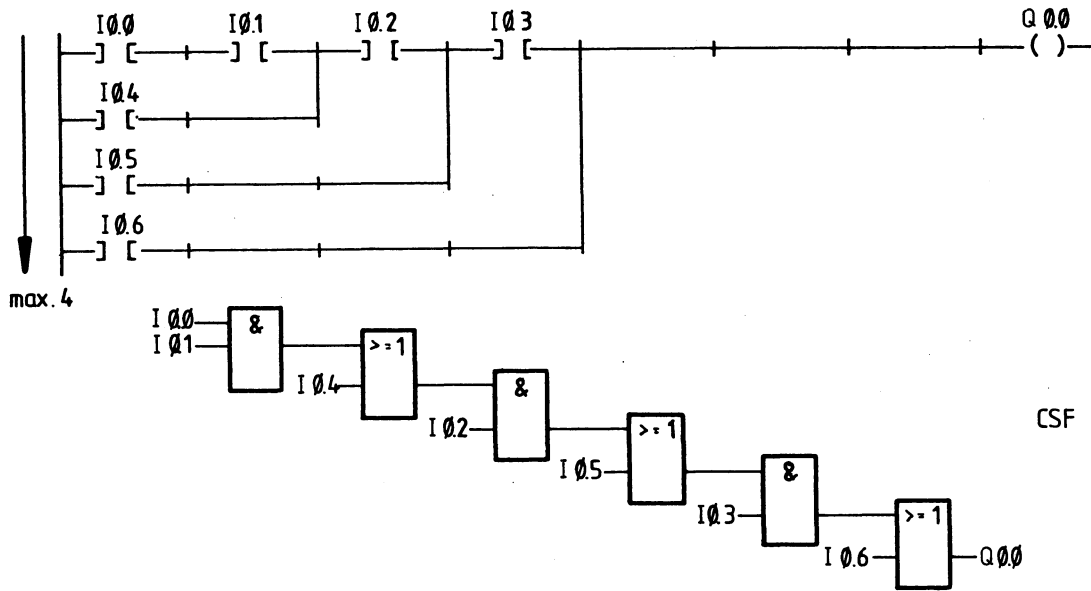


Fig. 10: Example of maximum LAD nesting for display as CSF

Input as CSF and display as LAD (STL)

Rule 1: Do not exceed the display boundaries for LAD.
 Too many inputs on a CSF box cause the ladder diagram display boundary to be exceeded.

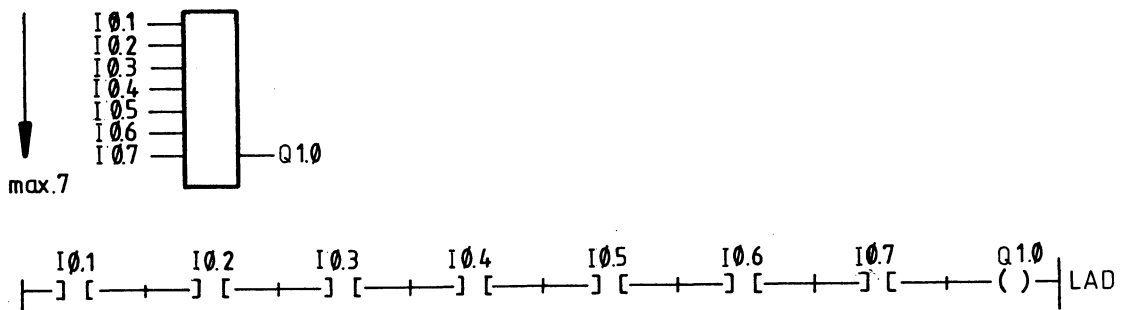


Fig. 11: Example of a maximum AND box in CSF form for display as an LAD

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

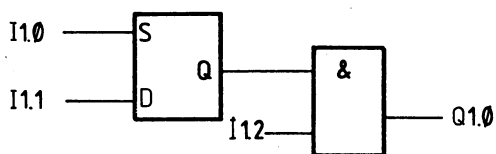


Fig. 12: Only AND boxes are allowed in CSFs after a complex element.

Input as STL and display as LAD or CSF

Rule 1: A complex element must not have a preceding operation.

Rule 2: The inputs and outputs of complex elements must be programmed in the order in which they are assigned parameters on the screen in graphic mode.

Times and counts are exceptions since the relevant value must first be stored in the accumulator with a load operation.

In addition, every unused input or output must be assigned an NOP 0 operation. In the case of timers and counters, the set input and the input for loading the time (TW) or count (ZW) must be disabled together.

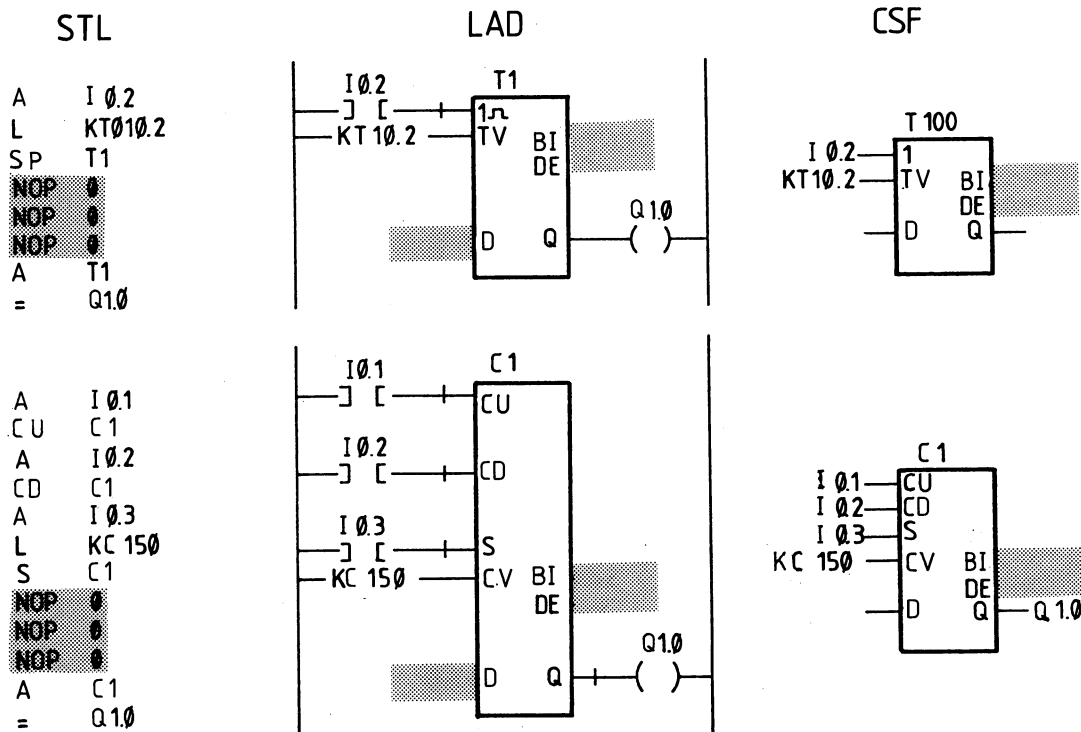


Fig. 13: Example for assigning NOP operations to unused inputs and outputs

3.6 Operation in the SINEC L1 local area network

The SINEC L1 local area network is used for interconnecting programmable controllers of the low-end performance range and operates on the Master-Slave principle.

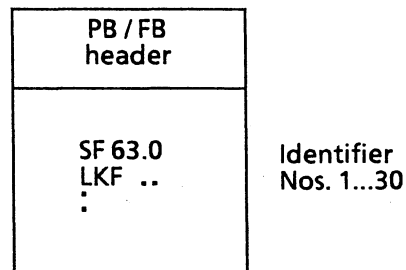
The CP 530 communications processor is always the Master, and the slaves are the CPUs of all small PCs. Each slave is assigned a slave number under which it is referenced. Data can be interchanged between the master and up to 30 slaves, as well as between the individual slaves. In the case of the S5-101U, the slave number, the coordinating flags and the send and receive mailboxes are defined as follows:

In addition to the actual data, control and security information, which the STEP 5 user program can access through a coordinating flag word, is also transmitted.

The actual data are deposited in a receive mailbox and a send mailbox which the user can access with load and transfer operations.

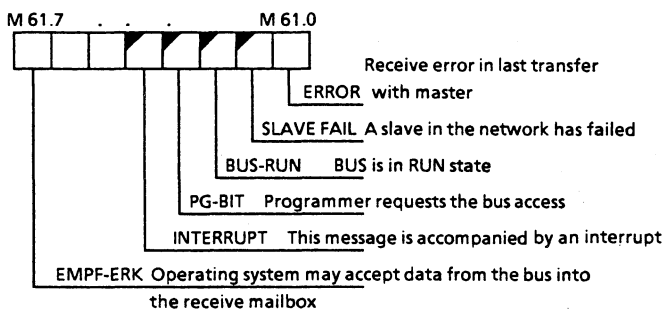
Slave number

The slave number is stored at the beginning of the user program (PB1/FB1) together with an identifier.



RECEIVE coordinating flag byte (KME)

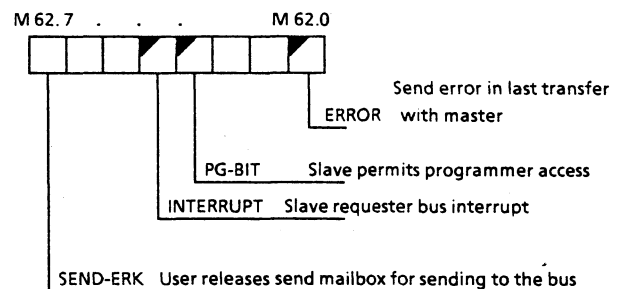
Flag byte FB 61 is used.



Bit from bus master

SEND coordinating flag byte (KMS)

Flag byte FB 62 is used.



Bit for bus master

The coordinating flags are affected by the operating system of the PC and can therefore not be used as flags in the normal sense.

Receive mailbox

The receive mailbox is in data block DB1 (DW 40...DW72) and has the following structure:

DL 40	LENGTH of net (0...64)	DR 40	SOURCE-SLAVE No. (0...30)	DW 40	} Nett data
DL 41	1st item of data	DR 41	2nd item of data	DW 41	
DL 42	3rd item of data	DR 42	4th item of data	DW 42	
.	
.	
.	
.	
DL 71	61st item of data	DR 71	62nd item of data	DW 71	
DL 72	63rd item of data	DR 72	64th item of data	DW 72	

1) Slave No. \emptyset = Master

Send mailbox

The send mailbox is in data block DB 1 (DW88...DW 112) and has the following structure:

DL 80	LENGTH of nett data (0...64)	DR 80	DESTINATION SLAVE No. (0...30)	DW 80	} Nett data
DL 81	1st item of data	DR 81	2nd item of data	DW 81	
DL 82	3rd item of data	DR 82	4th item of data	DW 82	
.	
.	
.	
.	
DL 111	61st item of data	DR 111	62nd item of data	DW 111	
DL 112	63rd item of data	DR 112	64th item of data	DW 112	

1) Slave No. \emptyset = Master

For more detailed information on the SINEC L1 local area network, please refer to the Instructions (4NEB 811-0545) and Programming Instructions (4NEB 811-0546) of the SINEC L1 network.

4. Program Start-up

The hand-held PG 605U/615 programmer and the CRT-based PG 670 and PG 675 programmers can be used for loading and testing programs.

The following settings are necessary on the PG 670 and PG 675 programmers in conjunction with the S5-1101 U programmable controller:

- PG 670: S5-150 AK S5-130 W
- PG 675: S5-150 S NO

4.1 Loading and dumping a program

Before loading the program, the "ERASE PROGRAM" function must be executed.

This deletes

- the internal program memory of the PC
- the binary process I/O image
- all flags
- error identifiers and the causes of interrupts.

When the program has been loaded, it is transferred from the programmer memory to the internal memory of the PC. If an EEPROM submodule is plugged in, the program is automatically dumped.

Once the program has been transferred to the PC memory, it is no longer in the programmer memory and must be brought back into the latter before program corrections can be made* (output FB1/PB1).

Dumping of the program in an EPROM submodule is possible on the PG 615 (with adapter), PG 670 (with 984 adapter) and PG 675 programmers.

To dump the program in an EPROM submodule, proceed as shown in Fig. 14.

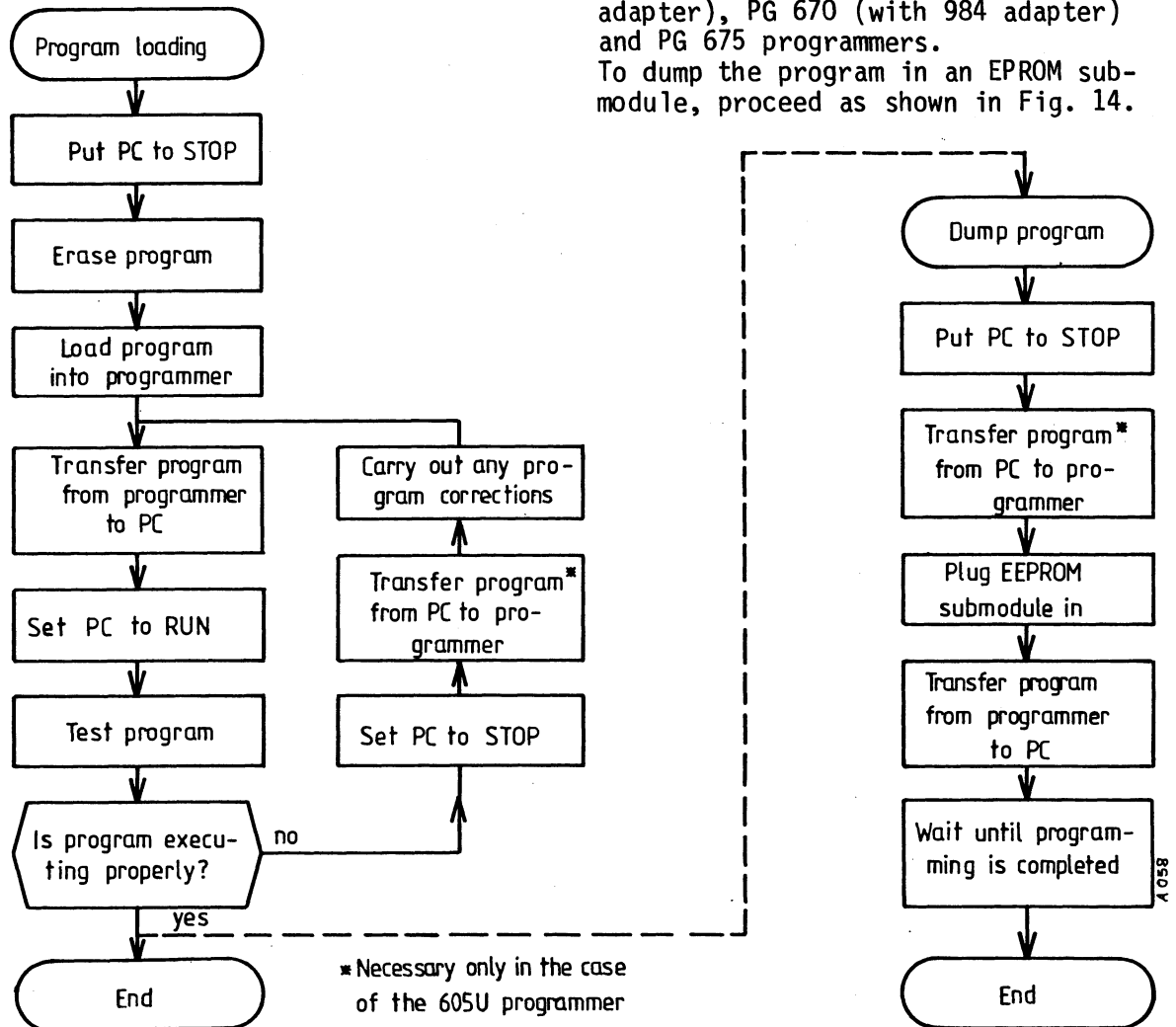


Fig. 14: Schematic of a program loading operation followed by the dumping of the program in an EEPROM submodule

4.2 Program test

Faults causing the PC to enter the "STOP" status can be identified with the aid of the interrupt stack (see Instructions, Section 4.2)

The following debugging functions are available for tracing logic errors in program scanning.

4.2.1 Search function

The programmer "Search" function is available for locating points in the user program.

In the test phase, for instance, all points in a program containing a definite operand can be displayed, e.g. an output not acting as expected.

The following search keys can be used:

- Statements, e.g. AI 1.0
- Operands, e.g. I 1.0
- Labels (FB 1 only)
- Addresses

Search runs are important in conjunction with the following functions:

- Input/correction
- Display
- Program-dependent signal status display

For more details, please refer to the Operating Instructions of the programmers.

4.2.2 Signal status display

The following programmer functions are available for displaying the signal statuses of binary and digital operands:

Direct signal status display

The status of any operands can be observed at the cycle checkpoint (Section 2.1) with the aid of this function.

Program-dependent signal status display

This test function enables the signal status of an operand and the result of the logic operation to be observed when the selected statement is processed.

4.2.3 Forcing of outputs and flags

The "FORCE" function enables definite binary and digital operands to be influenced with the PC in the "RUN" mode.

The desired statuses of the operands are entered from the programmer byte by byte and transferred to the PC.

The following can be forced:

- QB 0... QB 3
- FB 0...FB 63

In this way, it is possible to force definite outputs on system startup without the user program and check the correct wiring of actuators and indicators etc. When forcing while a user program is executing, the operand statuses entered are transferred once to the PC and program scanning resumed with these statuses entered may be modified by the current program.

5. Programming examples

5.1 Basic operations

5.1.1 Binary logic operations

AND logic

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 1.1 A I 1.3 A I 1.7 = Q 1.0 </pre>		

A "1" signal appears at output Q 1.0 when all the inputs have "1" signals simultaneously.

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

A "0" signal appears at output Q 1.0 if at least one of the inputs has a "0" signal.

OR logic

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

A "1" signal appears at output Q 1.2 if at least one of the inputs has a "1" signal.

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

A "0" signal appears at output Q 1.2 when all inputs have "0" signals simultaneously.

AND before OR logic

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 1.5 A I 1.6 O A I 1.4 A I 1.3 = Q 1.1 </pre>		

A "1" signal appears at output Q 1.1 when the output of at least one of the AND gates is "1".

A "0" signal appears at output Q 1.1 when neither of the AND gates has "1" at its output.

OR before AND logic

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> O I 1.0 O A I 1.1 A (O I 1.2 O I 1.3) = Q 1.1 </pre>		

A "1" signal appears at output Q 1.1 if input I 1.0 or I 1.1 and one of the inputs I 1.2 or I 1.3 have a "1" signal.

A "0" signal appears at output Q 1.1 when input I 1.0 has a "0" signal and the AND gate has a "0" at its output.

OR before AND logic

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

A "1" signal appears at output Q 2.0 when both OR gates have "1" signals at their outputs.

A "0" signal appears at output Q 2.0 when at least one of the OR gates has a "0" signal at this output.

Scanning for "0" signal status

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 1.5 ANI 1.6 = Q 2.0 </pre>		

A "1" signal appears at output Q 2.0 only when input I 1.5 has a "1" signal and input I 1.6 a "0" signal.

5.1.2 Setting/resetting operations

RS flip-flop for latched signal outputs

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.7 S Q 1.5 A I 1.4 R Q 1.5 NOP Ø </pre>		

A "1" at signal I 1.7 sets the flip-flop.

If the signal at input I 1.7 changes to "0", this status is maintained, i.e. the signal is latched.

A "1" at input I 1.4 resets the flip-flop.

If the signal at input I 1.4 changes to "0", this status is maintained.

If the set (input I 1.7) and reset (input I 1.4) signals are applied simultaneously, the scan operation last programmed (in this case AI 1.4) remains effective during processing of the remaining program.

***NOP Ø** is only necessary if program is to be represented in LAD or CSF form on the 670/675 programmer. When programming with LAD or CSF, these NOP Ø operations are automatically included.

RS flip-flop with flags

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.3 R F 1.7 A I 1.6 S F 1.7 A F 1.7 = Q 1.4 </pre>		

A "1" at input I 1.6 sets the flip-flop.

If the signal at input I 1.6 changes to "0", this status is maintained, i.e. the signal is latched.

A "1" at input I 1.3 resets the flip-flop.

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

If the set (input I 1.6) and reset (input I 1.3) signals are applied simultaneously, the scanning operation last programmed (in this case AI 1.6) remains effective during processing of the remaining program, i.e. flag F 1.7 is set (setting signal has priority over the resetting signal).

Implementation of a transition-sensitive pulse (pulse edge evaluation)

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 condition (AI 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 (RLO) is "1".

The AND logic condition AI 1.7 and AN F 4.0 is no longer fulfilled at the next program scan since flag F 4.0 has been set.

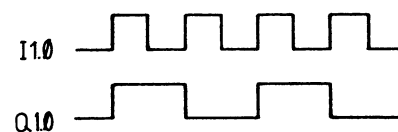
Flag 2.0 is reset, i.e. it is only "1" during a single program pass or scan.

Binary scaler (T or trigger flip-flop)

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

Output Q 1.0 changes its state on a positive-going transition at input 1.0. A negative-going change at the input has no effect on the output.

If a defined frequency is applied to the input, therefore, half the input frequency appears at the output.



5.1.3 Load and transfer operations

Load and transfer

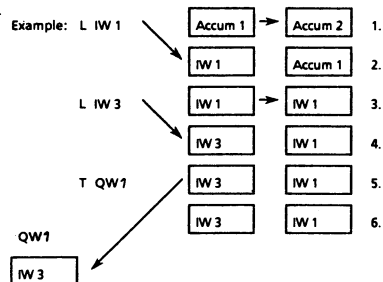
Operation	Parameters	Function
L <input type="checkbox"/> <input type="checkbox"/>		Load
<input type="checkbox"/> <input type="checkbox"/>		
<input type="checkbox"/> <input type="checkbox"/>		
I B	0 to 5	an input byte (from PII ¹⁾)
I W	0 to 4	an input word (from PII)
Q B	0 to 3	an output byte (from PIO ²⁾)
Q W	0 to 2	an output word (from PIO)
F B	0 to 63	a flag byte (from PIO)
F W	0 to 62	a flag word
D R	1 to 255	data (right-hand byte)
D L	1 to 255	data (left-hand byte)
D W	1 to 255	data (word)
P B	0 to 5	a peripheral byte of the digital I/O modules (bypassing the PIO)
T	0 to 15	a time (binary)
CT	0 to 15	(BCD)
C	0 to 15	a count (binary)
CC, KM ³⁾	0 to 15	(BCD)
K H ³⁾	random bit (16 bits)	a constant as bit pattern
K F ³⁾	0 to FFFF	a constant in hexadecimal code
K γ ³⁾	- 32768 to + 32767	a constant as fixed-point number
K γ ³⁾	0 to 255	a constant, 2 bytes
K S ³⁾	for each byte	
K S ³⁾	2 random alpha-numeric characters	a constant, 2 ASCII characters
K T ³⁾	0.0 to 999.3	a time (constant)
K C ³⁾	0 to 999	a count (constant)

Note:

The programmable controller has two accumulators (16 bits) for relational and arithmetic operations and for digital logic.

Loading implies that the contents of accumulator 1 are relocated to accumulator 2 and that accumulator 1 is reloaded in keeping with the operand of the load operation.

After two load operations, therefore, information can be obtained, for example, on the contents of the accumulators in connection with relational or comparison operations.



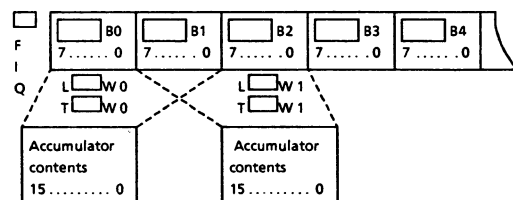
The transfer operation always transfers the contents of accumulator 1 to the operand specified in the transfer operation. The contents are not changed.

Operation	Parameters	Function
T <input type="checkbox"/> <input type="checkbox"/>		Transfer
<input type="checkbox"/> <input type="checkbox"/>		
<input type="checkbox"/> <input type="checkbox"/>		
I B	0 to 5	an input byte (from PII ¹⁾)
I W	0 to 4	an input word (from PII)
Q B	0 to 3	an output byte (from PIO ²⁾)
Q W	0 to 2	an output word (from PIO)
FB	0 to 63	a flag byte
FB	0 to 62	a flag word
D R	1 to 255	data (right-hand byte)
D L	1 to 255	data (left-hand byte)
D W	1 to 255	data (word)

Load and transfer operations are absolute operations, i.e. they are carried out independently of the result of the previous logic operation.

Graphics programming of load and transfer operations is only possible indirectly in connection with timer and counter operation, otherwise only in statement lists.

When loading/transferring FW, IW and QW, the following relationship between the accumulator contents and the byte belonging to a particular word applies:

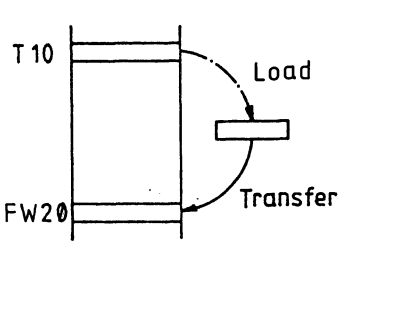
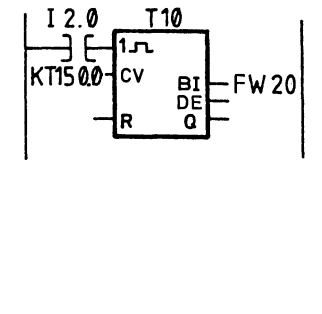
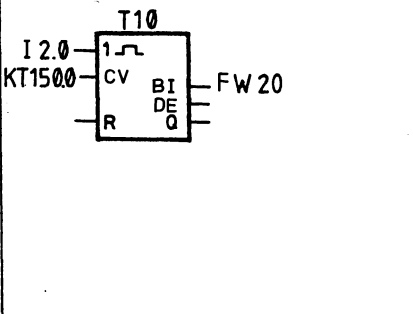


When loading an FB, IB, QB or PB, the byte is always loaded in the low byte of the accumulator. 0 is written into the high byte of the accumulator.

When transferring an FB, IB, QB or PB, it is always the low byte of the accumulator that is transferred.

- 1) PII process image of inputs
- 2) PIO process image of outputs
- 3) Four-byte instruction with the opcode in bytes 0/1 and the constant in bytes 2/3

Loading and transferring a time (see also under timer and counter operations)

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre> AI 2.0 L KT150.0 SPT 10 NOP 0 L T 10 T FW 20 NOP 0 NOP 0 </pre>		

During graphic input, FW 20 was assigned to output BI of the timer.

The programmer automatically stores the corresponding load and transfer operation in the user program. In this way, the contents of the memory location addressed with T 10 are loaded into accumulator 1. The contents of accumulator 1 are then transferred to FW 20.

The time T10 in binary code in this example can be traced at FW 20.

Outputs BI and DI are digital outputs. The time appears in binary code (BCD with time base) at output BI (DE).

5.1.4 Timer functions

Timers are restarted on power recovery following a powerfail condition.

Pulse

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 2.0 L KT10.2 SPT 1 NOP 0 NOP 0 NOP 0 AT 1 = Q 1.0 </pre>		

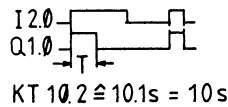
The timer is started during the first scanning cycle if the result of the logic operation is "1". The timer remains unaffected during subsequent scanning resulting in "1" signal.

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

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

The timer is loaded with the specified value (10). The number to the right of the point indicates the time base:
 $0 \hat{=} 0.01 \text{ s}$ $2 \hat{=} 1 \text{ s}$
 $1 \hat{=} 0.1 \text{ s}$ $3 \hat{=} 10 \text{ s}$

BI and DE are digital outputs. The time appears at output BI (DE) in BCD.



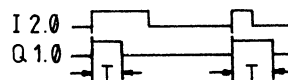
Extended pulse

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 2.0 L KT10.2 SET1 NOP 0 NOP 0 NOP 0 AT 1 = Q 1.0 </pre>		

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

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

The AT or OT scan results in a "1" signal as long as the timer is running.



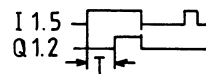
"ON" delay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.5 L KT9.2 SRT 3 NOP 0 NOP 0 NOP 0 AT 3 = Q 1.2 </pre>		

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

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

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



The timer is loaded with the specified value (9). The number to the right of the point indicates the time base:

0 $\hat{=}$ 0.01 s 2 $\hat{=}$ 1 s
1 $\hat{=}$ 0.1 s 3 $\hat{=}$ 10 s

Outputs BI and DE are digital outputs. The time appears at output BI (DE) in BCD.

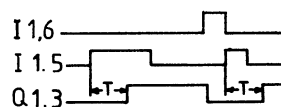
Latching "ON" delay

Original	STEP 5 representation		
	Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.5 L KT9.2 S ST4 A I 1.6 RT 4 NOP 0 NOP 0 AT 4 = Q 1.3 </pre>		

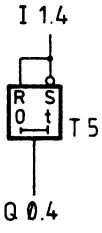
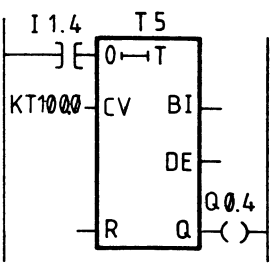
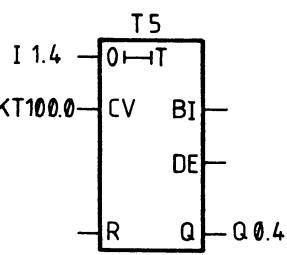
The timer is started during the first scanning cycle if the result of the logic operation is "1".

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

The AT or OT scans result in a "1" signal when the time has elapsed. The signal status only becomes "0" if the timer is reset with the RT operation.



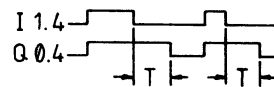
"OFF" delay

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre> A I 1.4 L KT100.0 S FT 5 NOP 0 NOP 0 NOP 0 A T 5 = Q 0.4 </pre>		

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

The timer is set to "0" (reset) if the result of the logic operation is "1".

The A T or OT scan results in a "1" signal if the timer is running or if the result of the logic operation is still present at the input.

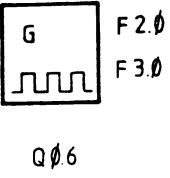
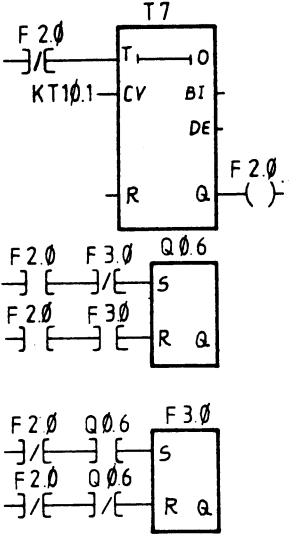
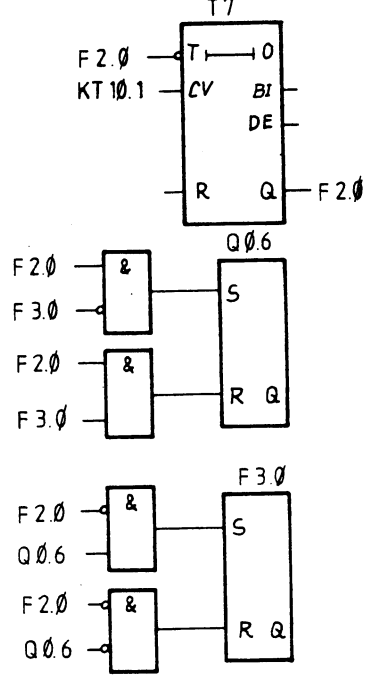


The timer is loaded with the specified value (9). The number to the right of the point indicates the time base:

0 \cong 0.01 s 2 \cong 1 s
1 \cong 0.1 s 3 \cong 10 s

Outputs BI and DE are digital outputs. The time appears at output BI (DE) in BCD.

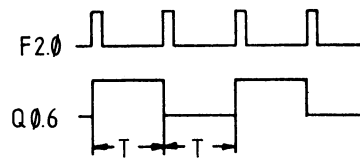
Clock pulse generator

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre> ANF 2.0 LKT 10.1 SRT 7 NOP 0 NOP 0 NOP 0 A T 7 = F 2.0 A F 2.0 ANF 3.0 S Q 0.6 A F 2.0 A F 3.0 R Q 0.6 NOP 0 ANF 2.0 A Q 0.6 S F 3.0 ANF 2.0 ANQ 0.6 R F 3.0 NOP 0 </pre>		

A clock pulse generator can be constructed from a self-clocking timer with a T flip-flop (binary scaler) at its output.

Timer T 7 is restarted with flag F 2.0 each time its time elapses, i.e. flag F 2.0 has a "1" signal for one cycle each time the time elapses.

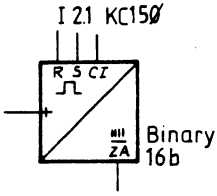
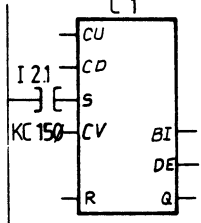
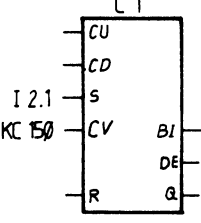
These pulses from flag F 2.0 act on the following T flip-flop with the result that a pulse train with a mark-space ratio of 1 : 1 appears at output Q 0.6. The period duration of this pulse train is twice as great as the time of the self-clocking timer.



5.1.5 Counter functions

Counters 8 to 15 are restarted on power recovery following a powerfail condition

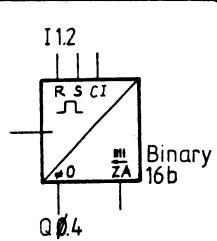
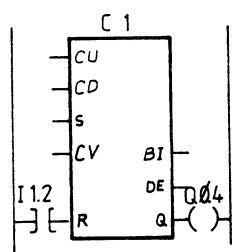
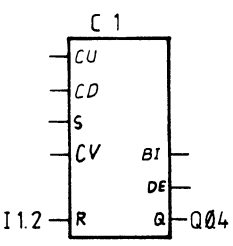
Set counter

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre data-bbox="597 415 705 504">A I 2.1 L KC 150 S C 1</pre>		

The counter is set during the first scanning cycle if the result of the logic operation is "1". The counter remains unchanged during subsequent processing (irrespective of whether the result of the logic operation is "1" or "0"). The counter is set again (pulse edge evaluation) at the next first scanning cycle if the result of the logic operation is "1".

In the above example, the starting value of the counter is 150. BI and DE are digital outputs. The count appears at output BI (DE) in BCD.

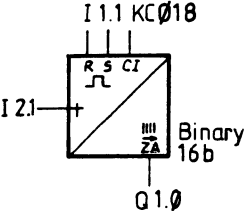
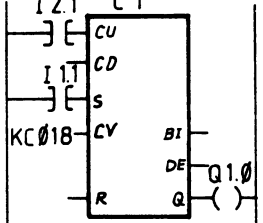
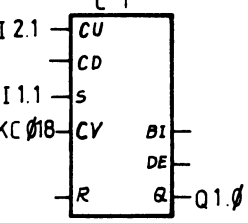
Reset counter

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre data-bbox="597 1457 705 1590">A I 1.2 R C 1 A C 1 = Q 0.4</pre>		

The counter is reset when the result of the logic operation is "1".

The counter remains unchanged if the logic operation becomes "0".

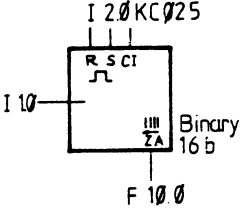
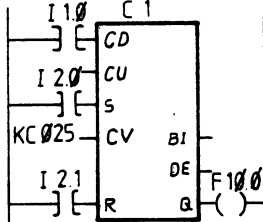
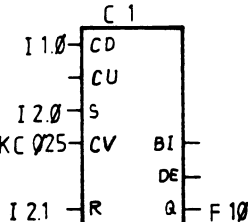
Counting up

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 2.1 CUC 1 NOP 0 A I 1.1 LKC 018 SC 1 NOP 0 NOP 0 NOP 0 AC 1 = Q 1.0 </pre>		

The value of the addressed counter is incremented by 1. The CU function is effective only on a positive-going pulse edge (from "0" to "1") of the result of the logic operation programmed before CU.

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.

Counting down

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre> A I 1.0 CDC 1 NOP 0 A I 2.0 LKC 025 SC 1 A I 2.1 RC 1 NOP 0 NOP 0 AC 1 = F 10.0 </pre>		

The value of the addressed counter is decremented by 1. The CD function is effective only on a positive-going pulse edge (from "0" to "1") of the logic operation programmed before CD.

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.

5.1.6 Comparison (relational) operations

Comparing for equal to

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 0 L IB 1 !=F = Q1.0</pre>		

The operand first specified is compared with the subsequent operation in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

for	CC1	CCO	RLO
IB 0 = IB 1	0	0	1
IB 0 < IB 1	0	1	0
IB 0 > IB 1	1	0	0

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers. After a comparison for equal to, a jump can be made to a label (+ 127 words) with the jump operation JZ = ... (if RLO = 1).

Comparing for not equal to

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre>L IB 0 L IB 1 ><F = Q1.1</pre>		

The operand first specified is compared with the subsequent operand in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

for	CC1	CCO	RLO
IB 0 = IB 1	0	0	0
IB 0 < IB 1	0	1	1
IB 0 > IB 1	1	0	1

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers.

Following a comparison for not equal to, a jump can be made to a label (+ 127 words) with the jump operation JN = ... (if RLO = 1).

Comparing for greater than

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre>L IB 0 L IB 1 > F = Q1.2</pre>		

The operand first specified is compared with the subsequent operand in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

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

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers.

After a comparison for greater than, a jump can be made to a label (+ 127 words) with the jump operation JP = ... (if RLO = 1).

Comparing for less than

Original	STEP 5 representation		Control system flowchart
	Statement list	Ladder diagram	
	<pre>L IB 0 L IB 1 < F = Q1.4</pre>		

The operand first specified is compared with the subsequent operand in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

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

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers.

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

Comparing for greater than or equal to

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre>L IB0 L IB1 >=F = Q1.3</pre>		

The operand first specified is compared with the subsequent operand in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

for	CC1	CCO	RLO
IB 0 = IB 1	0	0	1
IB 0 < IB 1	0	1	0
IB 0 > IB 1	1	0	1

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers.

After comparison for greater than or equal to, a jump can be made to a label (+ 127 words) with the jump operation JC (if RLO = 1).

Comparing for less than or equal to

Original	STEP 5 representation Statement list	Ladder diagram	Control system flowchart
	<pre>L IB0 L IB1 <=F = Q1.5</pre>		

The operand first specified is compared with the subsequent operand in keeping with the comparison function. The result of the comparison is flagged by condition codes CCO and CC1.

for	CC1	CCO	RLO
IB 0 = IB 1	0	0	1
IB 0 < IB 1	0	1	1
IB 0 > IB 1	1	0	0

The numerical representation of the operands is taken into account, i.e. the contents of the accumulators are interpreted as being fixed-point numbers.

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

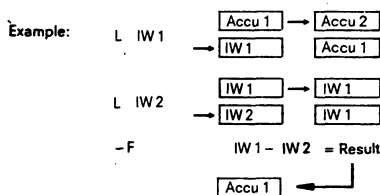
5.1.7 Arithmetic operations

Arithmetic operations can only be represented in statement list form.

They add or subtract the contents of accumulators 1 and 2, using the corresponding load operations.

Operation	Parameters	Function
+ F		Add (accu. 1 + accu.2)
- F		Subtract (accu. 2 - accu. 1)

The two accumulators 1 and 2 can be loaded by two load operations in keeping with the operands of these load operations. The contents of the two accumulators can then be added to each other or one subtracted from the other.



Example	STL	Explanation						
<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> <table style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td style="text-align: right;">127</td> </tr> <tr> <td style="text-align: right;">DR 85</td> <td style="text-align: right;">74</td> </tr> <tr> <td style="text-align: right;">Result F</td> <td style="text-align: right;">53</td> </tr> </table>		127	DR 85	74	Result F	53	<p>L KF +127</p> <p>L DR 85</p> <p style="border: 1px solid black; padding: 2px; display: inline-block;">-F</p> <p>T DL 85</p>	<p>The constant fixed-point number +127 is loaded into accumulator 1; at the same time, the old contents of accumulator 1 are shifted into accumulator 2.</p> <p>The right-hand byte of data word 85 is loaded into accumulator 1 and the fixed-point number +127 shifted into accumulator 2.</p> <p>The contents of accumulator 1 are subtracted from those of accumulator 2 and the result stored in accumulator 1.</p> <p>The contents of accumulator 1 (result) are transferred to the left-hand byte of data word 85.</p>
		127						
	DR 85	74						
Result F	53							

Note:

If the number range (-32768 to + 32767) is exceeded, the result of the operation is undefined (OV = "1")

5.1.8 Other functions

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

Operation	Parameter	Function
S T P		Stop
N O P 0		No operation (all bits reset)
N O P 1		No operation (all bits set)
B L D	0 to 255	Display construction statement

The STOP operation is used if, for example, the programmable controller has to enter the Stop status in response to certain critical plant states or on the occurrence of a hardware fault.

The NOPs are used, for instance, for keeping memory locations free or overwriting them.

The display construction statement defines the subdivision of program sections into segments within a block.

5.2 Supplementary operations

Supplementary operations can only be programmed in FB 1.

5.2.1 Logic operations

Operation	Description
AW	Digital ANDing of accumulators 1 and 2
OW	Digital ORing of accumulators 1 and 2.
XOW	Digital EXORing of accumulators 1 and 2.

Accumulators 1 and 2 can be loaded by two load operations in keeping with the operands of these load operations. The contents of both accumulators can then be digitally gated.

Example	STL	Explanation
<p>The hexadecimal number 3F84_H is to be ANDed with input word 1 of inputs IW 1. The result is to appear in output word zero of the outputs (QW 0).</p> <p style="margin-left: 40px;"> IW 1 3F84_H 4793_H Result AW 0780_H </p>	<p>L KH 3F84</p> <p>L IW 1</p> <p>AW</p> <p>T QW 0</p>	<p>The hexadecimal number 3F84 is loaded into accumulator 1; at the same time, the old contents of accumulator 1 are shifted into accumulator 2.</p> <p>Input word 1 (IW 1) is loaded into accumulator 1 and the hexadecimal number shifted into accumulator 2. The contents accumulator 1 are digitally ANDed with those of accumulator 2 and the result stored in accumulator 1.</p> <p>The contents of accumulator 1 (result) are transferred to output word QW 0.</p>
<p>The two bit patterns 0101 1110 1000 1011</p> <p>and</p> <p>0111 0001 0111 1100</p> <p>are to be ORed with each other.</p> <p style="margin-left: 40px;"> 0101 1110 1000 1011 0111 0001 0111 1100 Result OW 0111 1111 1111 1111 </p>	<p>L KM 01...11</p> <p>L KM 01...00</p> <p>OW</p> <p>T FW 13</p>	<p>The first bit pattern is loaded into accumulator 1; at the same time, the old contents of accumulator 1 are shifted into accumulator 2.</p> <p>The second bit pattern is loaded into accumulator 1 and the first bit pattern shifted into accumulator 2.</p> <p>The contents of accumulator 1 are ORed with those of accumulator 2 and the result stored in accumulator 1.</p> <p>The contents of accumulator 1 (result) are transferred to flagword FW 13.</p>
<p>Input word IW 0 is to be compared with data-word 12 for equality. The non-identical bits of the word are to appear in output word QW 0.</p> <p style="margin-left: 40px;"> DW 12 EA83_H IW 0 68C5_H Result XOW 8246_H </p>	<p>L DW 12</p> <p>L IW 0</p> <p>XOW</p> <p>TQW 0</p>	<p>Data word DW 12 is loaded into accumulator 1; at the same time, the old contents of accumulator 1 are shifted into accumulator 2. Input word IW 0 is loaded into accumulator 1 and data word L DW 12 shifted into accumulator 2.</p> <p>The contents of accumulator 1 are EXORed with those of accumulator 2 and the result stored in accumulator 1.</p> <p>The contents of accumulator 1 (result) are transferred to output word QW 0.</p>

5.2.2 Conversion functions

Operation	Description
CFW	One's complement
CSW	Two's complement
	The value in accumulator 1 is converted. The result can be processed in the accumulator.

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

5.2.3 Shift operations

Operation	Description
SLW 0 to 15	Shift left
SRW 0 to 15	Shift right
	The parameter of this statement specifies the number of bit positions by which the contents of accumulator 1 are shifted to the left (SLW) or to the right (SRW). Any bit positions that become free when shifting are padded with zeros.

Example	STL	Explanation
The last four bits of the hexadecimal number 14AF _H in data word DW 1 are to be truncated and the new hexadecimal number 014A _H resulting stored in data word DW 3.	L DW 1	Load data word 1 into accumulator 1.
	SRW 4	Shift the contents of accumulator 1 four bit positions to the right. The bit positions that become free when shifting are padded with zeros.
	T DW 3	Transfer the contents of accumulator 1 to data word DW 3.

Note:

Shift operations are unconditional operations. The last bit shifted out can be examined by means of a jump operation. A jump can be made with JZ if the bit is "0" and with JN or JP if the bit is "1".

5.2.4 Jump operations

The jump destination for unconditional and conditional jumps is specified as a symbolic address (max. 4 characters). In the case of the PG 605U/ and PG 615U programmers, jump labels MO...M99 can be assigned. The symbolic parameter of the jump statement is identical to the symbolic address of the statement to which the jump is made. When programming, make sure that the absolute jump displacement does not exceed ± 127 words and note that a STEP 5 statement may consist of more than one word.

Jumps over segment boundaries are not permissible.

All jump operations (with the exception of JU) depend on the RLO and the condition codes in the processor of the programmable controller.

Operation	Description
JU = <input type="text"/>	<p>Unconditional jump</p> <p>The unconditional jump is executed independently of any conditions.</p>
JC = <input type="text"/>	<p>Conditional jump</p> <p>The conditional jump is executed if the RLO is "1". If the result of the logic operation is "0", the jump is not executed and the RLO is set to "1".</p>
JZ = <input type="text"/>	<p>Jump if contents of accumulator zero.</p> <p>This jump is executed if the contents of the accumulator are zero. If the contents are not zero, the jump is not executed. The RLO is not changed.</p>
JN = <input type="text"/>	<p>Jump if contents of accumulator not zero.</p> <p>This jump is executed if the contents of the accumulator are not zero. If the contents are zero, the jump is not executed. The RLO is not changed.</p>
JP = <input type="text"/>	<p>Jump if contents of accumulator positive.</p> <p>The jump is executed if the contents of the accumulator are greater than zero. If the contents of the accumulator are zero or less than zero, the jump is not executed. The RLO is not changed.</p>
JM = <input type="text"/>	<p>Jump if contents of accumulator negative.</p> <p>This jump is executed if the contents of the accumulator are less than zero. If the contents are zero or greater than zero, the jump is not executed. The RLO is not changed.</p>
JO = <input type="text"/>	<p>Jump on overflow</p> <p>This jump is executed when an overflow occurs. If there is no overflow, the jump is not executed. The RLO is not changed.</p> <p>Enter symbolic address (max. 4 characters)</p>

Example	STL	Explanation
<p>If none of the inputs of input word IW 1 is set, a jump is made to label "AN 1"</p>	<p>ANO : IW</p>	<p>Input word IW 1 is loaded into accumulator 1. If the contents of accumulator 1 are equal to zero, a jump is executed to label "AN 1", otherwise the next statement (AI 1.0) is executed.</p>
<p>If the input word IW 1 and output word QW 0 are not identical, a jump is made back to label "AN 0". If IW 1 and QW 3 are identical, IW 1 is compared with data word DW 12. If IW 1 is greater or smaller than DW 12, a jump is made to the "Destination" label.</p>	<pre> : JZ=AN1 : A IW ANI :L IW1 :L QW :XOW :JP=ANO :L IW1 :L DW12 : F :JB=Destina- tion . . . Desti-:A I12.2 nation </pre>	<p>Comparison of input word IW 1 and output word QW3. If the two words are not identical, individual bits are set in accumulator 1.</p> <p>If the contents of accumulator 1 are not zero, a jump is made back to the "AN 0" label, otherwise the next statements are executed.</p> <p>Input word IW 1 is compared with data word W 12 for greater or less than. If IW 1 is greater or less than DW 12, the RLO is set to "1".</p> <p>If RLO = "1", a jump is made to the "Destination" label.</p> <p>If RLO = "0", the next statement is executed.</p>

5.2.5 Condition codes

The processor of the SIMATIC S5-101U programmable controller has three condition codes:

- o CC 1 Value positive
- o CC 0 Value negative
- o OV Overflow

The condition codes are affected by comparison operations, arithmetic operations, shift operations and a number of conversion operations.

Generating condition codes in connection with comparison operations

The execution of comparison operations results in the setting of condition codes CC 0 and CC 1. The overflow bit is not affected.

The two operands participating in a comparison are defined as follows:

```
Example :L DW20 (1st operand)
         :L DW21 (2nd operand)
         :=F (comparison operation)
```

In this example, data word DW 20 is the 1st operand and data word DW 21 the 2nd operand.

	Condition codes		Jump operations
	CC 1	CC 0	
Compared with the 1st operand, the 2nd operand is:			The following jump operations are executed:
Equal	0	0	JZ
Less	0	1	JN, JM
Greater	1	0	JN, JP

Note:

Comparison operations affect the result of the logic operation. If the condition is satisfied, the RLO is "1". In this way, the conditional jump operation JB can be written after a comparison operation.

Generating condition codes in connection with arithmetic operations.

Arithmetic operations result in the setting of all condition codes. This depends in turn on the contents of the accumulator (the result of the arithmetic operation).

Generating condition codes in connection with fixed-point arithmetic operation:

	Condition codes			Jump operations
	CC1	CC0	OV	
The result of the arithmetic operation is:				The following jump operations are executed:
< -32768	1	0	1	JN, JP, JO
-32768 to -1	0	1	0	JN, JM
0	0	0	0	JZ
+1 to +32767	1	0	0	JN, JP
> +32767	0	1	1	JN, JM, JO
(-65536*)	0	0	1	JZ, JO

*) Result of the arithmetic operation: -32768 +32768

Generating condition codes in connection with digital logic.

Digital logic operations result in the setting of condition codes CC 0 and CC 1. The overflow bit is not affected. Setting of the condition codes depends on the contents of the accumulator after the operation has been executed.

	Condition codes		Jump operations
	CC1	CC0	
The contents of the accumulator are:			The following jump operations are executed:
Zero (0000)	0	0	JZ
Not zero	1	0	JN, JP

Generating condition codes in connection with shift operations.

The execution of shift operations results in the setting of CC 0 and CC 1. The overflow bit is not affected. Setting of the condition codes depends on the status of the last bit pushed out.

	Condition codes		Jump operations
	CC1	CC0	
The value of the last bit shifted out is:			The following jump operations are executed:
"0"	0	0	JZ
"1"	1	0	JN, JP

Generating condition codes in connection with conversion functions.

Formation of the two's complement (CSW) affects all condition codes. This depends on the result of the conversion operation:

	Condition codes			Jump operations
	CC1	CC0	OV	
The result after the conversion operation is:				The following jump operations are executed:
< -32768	1	0	1	JN, JP, JO
-32768 to -1	0	1	0	JN, JM
+1 to +32767	1	0	0	JN, JP
> +32767	0	1	1	JN, JM, JO
(-65536*)	0	0	1	JZ, JO

*) Result of conversion of KH = 0000

Note:

The jump statement and jump destination must be in the same segment. Only one symbolic address is permissible for branch destinations in each segment.

6. Operation set

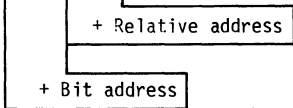
Binary logic operations

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function	
		Word 0		Word 1				CC1	CC0	OV			
		B0	B1	B2	B3								
Scan the following and AND with the result of the previous logic operation:													
A	I	0.0 to 2.3 3.0 to 5.3	C0	00	-	-	N	N	-	-	-	65	Input for "1"
A	Q	0.0 to 1.3 2.0 to 3.3	C0	80	-	-	N	N	-	-	-	65	Output for "1"
A	F	0.0 to 63.7	80	00	-	-	N	N	-	-	-	74	Flag for "1"
A	T	0 to 15	F8	00	-	-	N	N	-	-	-	69	Timer for "1"
A	C	0 to 15	B8	00	-	-	N	N	-	-	-	64	Counter for " > 0"
AN	I	0.0 to 2.3 3.0 to 5.3	E0	00	-	-	N	N	-	-	-	66	Input for "0"
AN	Q	0.0 to 1.3 2.0 to 3.3	E0	80	-	-	N	N	-	-	-	66	Output for "0"
AN	F	0.0 to 63.7	A0	00	-	-	N	N	-	-	-	75	Flag for "0"
AN	T	0 to 15	FC	00	-	-	N	N	-	-	-	70	Timer for "0"
AN	C	0 to 15	BC	00	-	-	N	N	-	-	-	65	Counter for "=0"
Scan the following and OR with the result of the previous logic operation:													
O	I	0.0 to 2.3 3.0 to 5.3	C8	00	-	-	N	N	-	-	-	65	Input for "1"
O	Q	0.0 to 1.3 2.0 to 3.3	C8	80	-	-	N	N	-	-	-	65	Output for "1"
O	F	0.0 to 63.7	88	00	-	-	N	N	-	-	-	74	Flag for "1"
O	T	0 to 15	F9	00	-	-	N	N	-	-	-	71	Timer for "1"
O	C	0 to 15	B9	00	-	-	N	N	-	-	-	64	Counter for " > 0"
ON	I	0.0 to 2.3 3.0 to 5.3	E8	00	-	-	N	N	-	-	-	66	Input for "0"
ON	Q	0.0 to 1.3 2.0 to 3.3	E8	80	-	-	N	N	-	-	-	66	Output for "0"
ON	F	0.0 to 63.7	A8	00	-	-	N	N	-	-	-	76	Flag for "0"
ON	T	0 to 15	FD	00	-	-	N	N	-	-	-	72	Timer for "0"
ON	C	0 to 15	BD	00	-	-	N	N	-	-	-	67	Counter for "=0"
			+ Relative address										
			+ Bit address										
)			BF	00	-	-	N	Y	-	-	-	63	Right paranthesis
A(BA	00	-	-	N	Y	-	-	-	66	ANDing of bracketed expressions
O(BB	00	-	-	N	Y	-	-	-	69	ORing of bracketed expressions
O			FB	00	-	-	N	Y	-	-	-	48	ORing of AND operations

Up to 6 bracketing levels can be programmed.

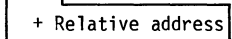
Setting/resetting operations

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
S I	0.0 to 2.3 3.0 to 5.3	D0	00	-	-	Y	Y	-	-	-	70	Set an input to "1" (in the PII)
S Q	0.0 to 1.3 3.0 to 3.3	D0	80	-	-	Y	Y	-	-	-	70	Set an output to "1" (in the PIO)
S F	0.0 to 63.7	90	00	-	-	Y	Y	-	-	-	74	Set a flag
R I	0.0 to 2.3 3.0 to 5.3	F0	00	-	-	Y	Y	-	-	-	70	Reset an input to "0" (in the PII)
R Q	0.0 to 1.3 2.0 to 3.3	F0	80	-	-	Y	Y	-	-	-	70	Reset an output to "0" (in the PIO)
R F	0.0 to 63.7	80	00	-	-	Y	Y	-	-	-	74	Reset a flag to "0"
= I	0.0 to 2.3 3.0 to 5.3	D8	00	-	-	N	Y	-	-	-	66	Assign an input (in the PII)
= Q	0.0 to 1.3 2.0 to 3.3	D8	80	-	-	N	Y	-	-	-	66	Assign an output (in the PIO)
= F	0.0 to 63.7	98	00	-	-	N	Y	-	-	-	72	Assign a flag



Timer and counter functions

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
SP T	0 to 15	34	00	-	-	Y	Y	-	-	-	134	Start timer as pulse
SE T	0 to 15	1C	00	-	-	Y	Y	-	-	-	120	Start timer as extended pulse
SR T	0 to 15	24	00	-	-	Y	Y	-	-	-	136	Start timer as "ON" delay
SS T	0 to 15	2C	00	-	-	Y	Y	-	-	-	120	Start timer as latching "ON" delay
SF T	0 to 15	14	00	-	-	Y	Y	-	-	-	136	Start timer as "OFF" delay
R T	0 to 15	3C	00	-	-	Y	Y	-	-	-	64	Reset timer
S C	0 to 15	5C	00	-	-	Y	Y	-	-	-	114	Set counter
R C	0 to 15	7C	00	-	-	Y	Y	-	-	-	66	Reset counter
CU C	0 to 15	6C	00	-	-	Y	Y	-	-	-	82	Count up
CD C	0 to 15	54	00	-	-	Y	Y	-	-	-	89	Count down



Load and transfer operations

Operation	Parameter	Maschine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
L IB	0 to 5	4A	00	-	-	N	N	-	-	-	61	Load an input byte (from the PII)
L IW	0 to 4	52	00	-	-	N	N	-	-	-	65	Load an input word (from the PIO)
L QB	0 to 3	4A	80	-	-	N	N	-	-	-	61	Load an output byte (from the PIO)
L QW	0 to 2	52	80	-	-	N	N	-	-	-	65	Load an output word (from the PIO)
L FB	0 to 63	0A	00	-	-	N	N	-	-	-	61	Load a flag byte
L FW	0 to 62	12	00	-	-	N	N	-	-	-	67	Load a flag word
L DL	0 to 255	22	00	-	-	N	N	-	-	-	65	Load data (left-hand byte)
L DR	0 to 255	2A	00	-	-	N	N	-	-	-	66	Load data (right-hand byte)
L DW	0 to 255	32	00	-	-	N	N	-	-	-	78	Load data (word)
L T	0 to 15	02	00	-	-	N	N	-	-	-	67	Load a time
L C	0 to 15	42	00	-	-	N	N	-	-	-	68	Load a count
L PB	0 to 5	72	00	-	-	N	N	-	-	-	116	Load a peripheral byte of the inputs
L DT	0 to 15	0C	00	-	-	N	N	-	-	-	124	Load a time (BCD)
L DC	0 to 15	4C	00	-	-	N	N	-	-	-	122	Load a count (BCD)
+ Relative address												
L KC	2 alphanumeric characters	30	10	00	00	N	N	-	-	-	63	2 ASCII characters
L KM	Bit pattern (16 bits)	30	80	00	00	N	N	-	-	-	63	as bit pattern
L KH	0 to FFFF	30	40	00	00	N	N	-	-	-	63	in hexadecimal code
L KF	0 to $(2^{16}-1)$	30	04	00	00	N	N	-	-	-	63	as a fixed-point number
L KY	0 to 255, each byte	30	20	00	00	N	N	-	-	-	63	2 bytes
L KT	0.0 to 999.3	30	02	00	00	N	N	-	-	-	63	as time
L KC	0 to 999	30	01	00	00	N	N	-	-	-	63	as count
+ Constant (1 word)												

Load and transfer operations (cont.)

Operation	Parameter	Maschine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
T IB	0 to 5	4B	00	-	-	N	N	-	-	-	55	Transfer to an input byte (in the PII)
T IW	0 to 4	53	00	-	-	N	N	-	-	-	58	Transfer to an input word (in the PII)
T QB	0 to 3	4B	80	-	-	N	N	-	-	-	55	Transfer to an output (in the PIO)
T QW	0 to 2	53	80	-	-	N	N	-	-	-	58	Transfer to an output word (in the PIO)
T FB	0 to 63	0B	00	-	-	N	N	-	-	-	52	Transfer to a flag byte
T FW	0 to 62	13	00	-	-	N	N	-	-	-	61	Transfer to a flag word
T DR	0 to 255	2B	00	-	-	N	N	-	-	-	57	Transfer to data (right-hand byte)
T DL	0 to 255	23	00	-	-	N	N	-	-	-	54	Transfer to data (left-hand byte)
T DW	0 to 255	33	00	-	-	N	N	-	-	-	61	Transfer to data (word)
T PB	0 to 3	73	00	-	-	N	N	-	-	-	90	Transfer to peripheral byte of outputs with updating of the PIO

+ Relative address

Comparison operations

Operation	Parameter	Maschine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
Compara fixed-point numbers for												
I =F		21	80	-	-	N	N	Y	Y	-	87	Equal to
><F		21	60	-	-	N	N	Y	Y	-	87	Not equal to
>F		21	20	-	-	N	N	Y	Y	-	87	Greater than
>=F		21	A0	-	-	N	N	Y	Y	-	87	Greater than or equal to
<F		21	40	-	-	N	N	Y	Y	-	87	Less than
<=F		21	C0	-	-	N	N	Y	Y	-	87	Less than or equal to

Arithmetic operations

Operation	Parameter	Maschine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
+F		79	00	-	-	N	N	Y	Y	Y	58	Fixed-point addition
-F		59	00	-	-	N	N	Y	Y	Y	54	Fixed-point subtraction

Block calls

Operation	Parameter	Maschine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
BE		65	00			N	Y	-	-	-	42	Block end
BEC		05	00			Y	Y	-	-	-	46	Block end conditional
BEU		65	01			Y	Y	-	-	-	42	Block end unconditional

Other operations

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
NOP 0		00	00			N	N	-	-	-	40	No operation (all bits reset)
NOP 1		FF	FF			N	N	-	-	-	40	No operation (all bits set)
STP		70	03			N	N	-	-	-	44	STOP
												Segment end for programming in statement list
BLD		10	00			N	N	-	-	-	40	Statement for constructing displays in LAD form on the programmer

Logic operations (word mode) (supplementary operations)

AW		41	00			N	N	X	X	-	55	ANDing of accumulators 1 and 2
OW		49	00			N	N	X	X	-	55	ORing of accumulators 1 and 2
XOW		51	00					X	X	-	55	Exclusive-ORing of accumulators 1 and 2

Conversion operations (supplementary operations)

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
CFW		01	00	-	-	N	N	-	-	-	42	Formation of one's complement (fixed point)
CSW		09	00	-	-	N	N	Y	Y	Y	62	Formation of two's complement (fixed point)

Shift operations (supplementary operations)

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function	
		Word 0		Word 1				CC1	CC0	OV			
		B0	B1	B2	B3								
SLW	0 to 15	61	00	-	-	N	N	Y	Y	-	49	Shift left (16 bits)	
SIW	0 to 15	69	00	-	-	N	N	Y	Y	-	47	Shift right (16 bits)	
								+ Number of shifts				+ 10 x Number of shifts	

Jump operations (supplementary operations)

Operation	Parameter	Machine code (hexadecimal)				Depends on RLO	Affects RLO	Condition codes affected			Execution time in μ s (max.)	Function
		Word 0		Word 1				CC1	CC0	OV		
		B0	B1	B2	B3							
JU =	Symbolic address	2D	00	-	-	N	N	-	-	-	55	Unconditional jump
JC =	Symbolic address	FA	00	-	-	Y	Y	-	-	-	62	Conditional jump (Jump condition: RLO)
JZ =	Symbolic address	45	00	-	-	N	N	-	-	-	62	Conditional jump (Jump condition: CC 1, CC 0)
JN =	Symbolic address	35	00	-	-	N	N	-	-	-	62	Conditional jump (Jump condition: CC 1, CC 0)
JP =	Symbolic address	15	00	-	-	N	N	-	-	-	62	Conditional jump (Jump condition: CC 1, CC 0)
JM =	Symbolic address	25	00	-	-	N	N	-	-	-	62	Conditional jump (Jump condition: CC 1, CC 0)
JO =	Symbolic address	0D	00	-	-	N	N	-	-	-	57	Conditional jump (Jump condition: OV)

+ Jump displacement

