Training document for the company-wide automation solution

Totally Integrated Automation (T I A)

Appendix III

Basic programming instructions

LAD/FBD/STL in STEP 7

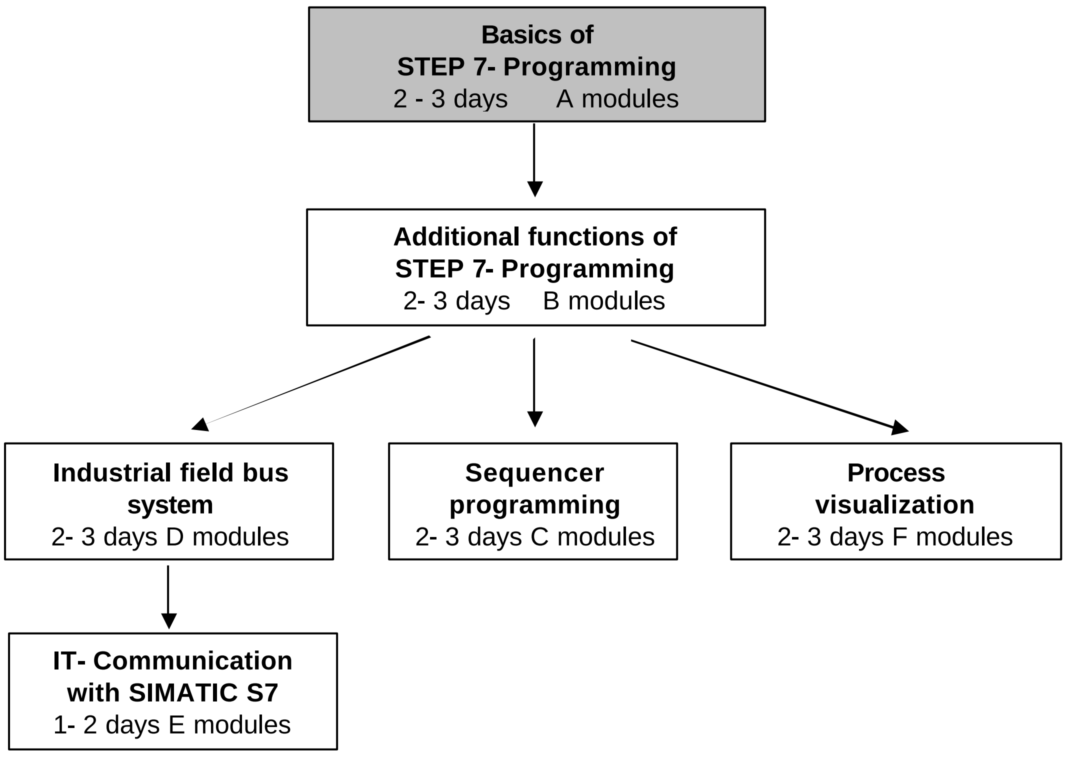
We thank the company Michael Dziallas Engineering and the instructors of vocational schools as well as further persons for the support with the production of the document.

PAGE:

PAGE:

1. FORWARD

Appendix III is needed during processing of all modules.



Learning goal:

The reader receives a collection of the most important programming instructions which are needed for the solution of the tasks of programming in the modules within this appendix.

Requirements:

In order for the instructions and the programming mode to be understood, the following knowledge is assumed:

- PLC- Programming basics (e.g. Appendix I – PLC-Programming basics with SIMATIC S7-300)

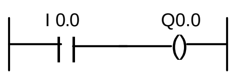
2. BASIC PROGRAMMING INSTRUCTIONS

The following programming instructions are sufficient for the basics of programming. This is however not a complete listing of all instructions. Information for further instructions in LAD/FBD/STL can be found in the manuals or in the **on-line help** under the point of **language description LAD, FBD** and/or **STL**.

2.1 ASSIGNMENT

The assignment (=) copies the logical operation result (RLO) of the preceding operation and assigns it to the following operand.
An operation chain can be locked by an assignment.

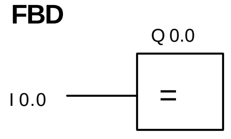
LAD



STL

```
A  I 0.0
=  Q 0.0
```

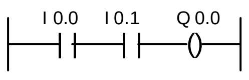
FBD



2.2 AND - OPERATION

The AND -Operation corresponds to a series connection of contacts in the circuit diagram. At the output Q 0.0, the signal status 1 appears if all inputs exhibit a signal status 1 at the same time. If one of the inputs exhibits a signal status 0, the output remains in a signal status 0.

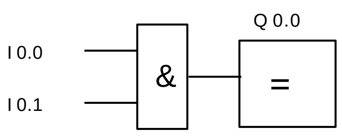
LAD



STL

```
A  I 0.0
A  I 0.1
=  Q 0.0
```

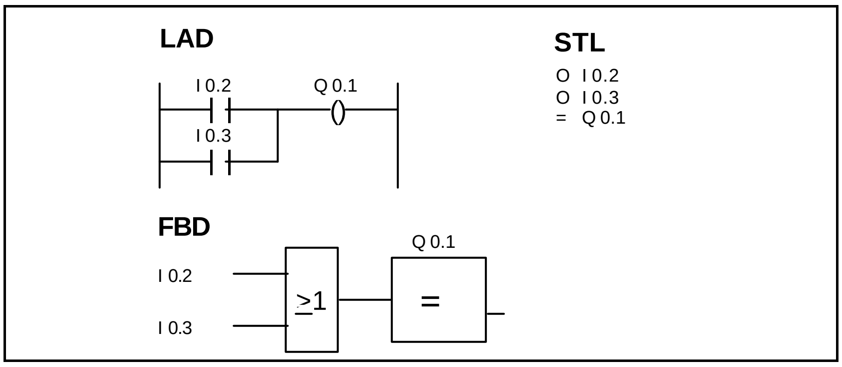
FBD



Forward
Program instructions

2.3 OR - OPERATION

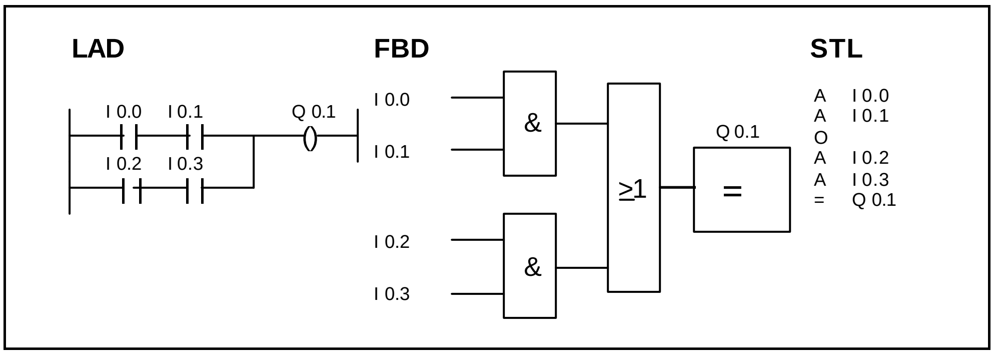
The OR -Operation corresponds to a parallel connection of contacts in the circuit diagram. At the output Q 0.1, a signal status 1 appears if at least one of the inputs exhibits a signal status 1. Only if all inputs exhibit a signal status 0, will the signal status at the output remain on 0.



```
O  I 0.2
O  I 0.3
=  Q 0.1
```

2.4 AND - BEFORE OR - OPERATION

The AND- before -OR -Operation corresponds to a parallel set-up of several contacts in the circuit diagram.
With these branches from rows and parallel circuits aligned together, the output 0.1 is fed the signal status 1, if in at least one branch of all contacts switched in the row are closed (have a signal status 1). The AND before OR- Operations are programmed without parentheses in the STL representation, however the parallel circuit branches must be separated by the input of the character O (OR function). First the AND functions are edited and from their results the result of the OR function is formed. The first AND function (I 0,0, I 0,1) becomes separated by the second AND function (I 0,2, I 0,3) through the single O (OR function).

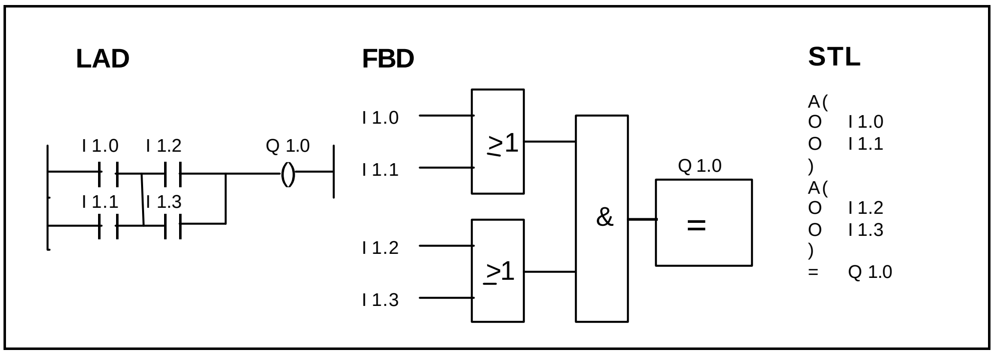


```
A    I 0.0
A    I 0.1
O
A    I 0.2
A    I 0.3
=    Q 0.1
```

The AND- Operations have priority and will always execute before the OR- Operations.

2.5 OR - BEFORE AND - OPERATION

The OR – before -AND operation corresponds to a series connection of several contacts joined in parallel in the circuit diagram.
With these branches from the rows and parallel circuits aligned together, the output 1.0 is fed the signal status 1, if in both branches at least one of the contacts switched in the row is closed (have a signal status 1).



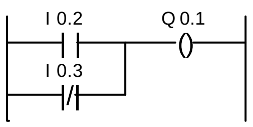
```
A(
O    I 1.0
O    I 1.1
)
A(
O    I 1.2
O    I 1.3
)
=    Q 1.0
```

Parenthesis must be used on the OR- Operations so that they will have a higher priority than the AND-Operations.

2.6 QUERY ON SIGNAL STATE 0

The debugging for the signal status 0 corresponds in a contact-afflicted circuit to an open contact and is realized in the connection AND NOT (AN), OR NOT (ON) and EXCLUSIVE OR NOT (XN).
Example of an OR NOT - Operation:

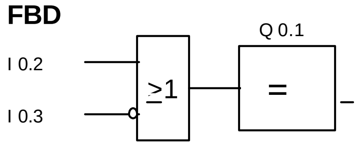
LAD



STL

```
O   I 0.2
ON  I 0.3
=   Q 0.1
```

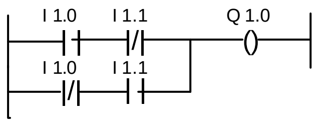
FBD



2.7 EXCLUSIVE - OR - OPERATION

The circuit shows an exclusive-OR operation (X), with which the output 1.0 is switched on (signal status 1) if *only one* of the inputs exhibits a signal status of 1. In an contact-afflicted circuit, this can be realized only with normally open and closed contacts.

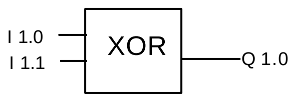
LAD



STL

```
X   I 1.0
X   I 1.1
=   Q 1.0
```

FBD



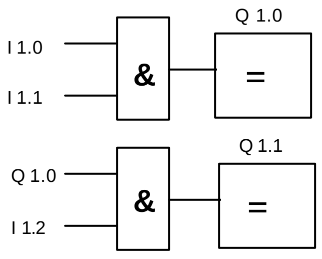
Caution: *The exclusive- OR- Operation should only be used with exactly two inputs.*

Forward | **Program instructions**

2.8 QUERY OF OUTPUTS

For the switching on of the outputs Q 1.0 and Q 1.1, different conditions apply. In these cases a current path and/or an operation symbol must be planned for each output. There the automation equipment can query not only the signal status of inputs, outputs, bit memories, etc. It will also query the outputs Q 1.1 and Q 1.0 from the AND operation.

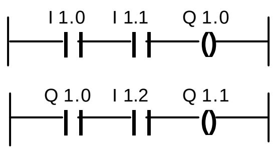
FBD



STL

```
A    I 1.0
A    I 1.1
=    Q 1.0
A    Q 1.0
A    I 1.2
=    Q 1.1
```

LAD

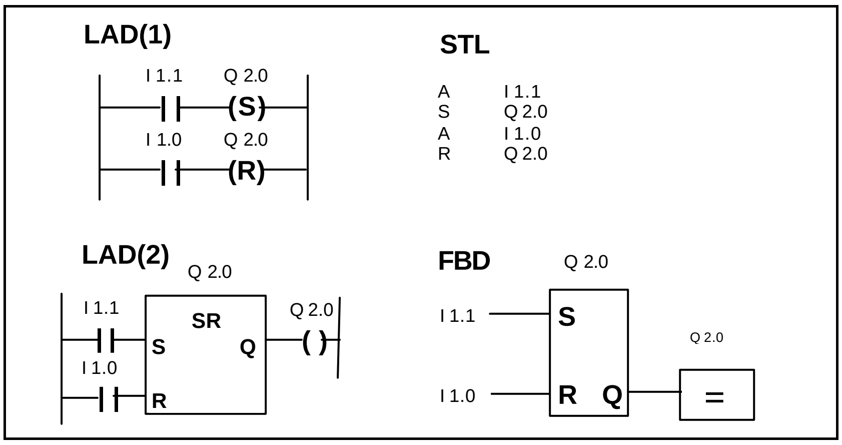


2.9 R - S – STORAGE FUNCTIONS

According to DIN 40900 and DIN 19239, an R-S memory function is represented as a rectangle with the set input S and the reset input R. A signal status 1 at the set input S sets the memory function. A signal status 1 at the reset input R results in the resetting of the memory function. A signal status 0 at the inputs R and S does not change the previously set condition. Should a signal status 1 be applied to both inputs R and S simultaneously, the function will be set or reset. This priority resetting or setting must be considered with programming.

Forward Program instructions

2.9.1 RESET DOMINANT



STL

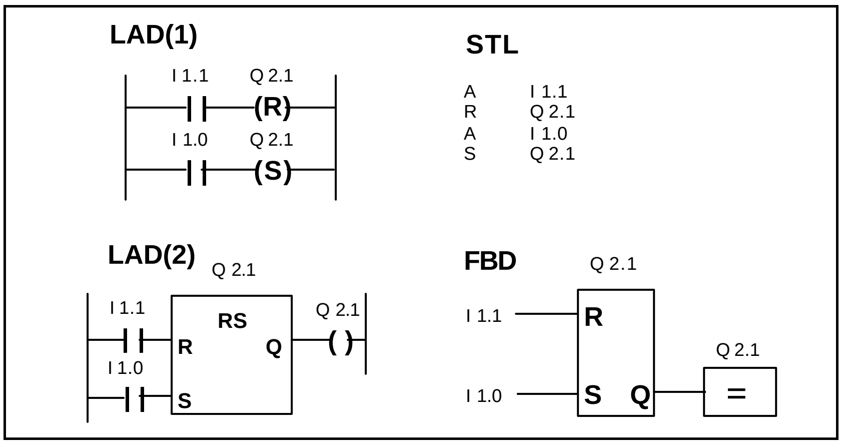
```
A       I 1.1
S       Q 2.0
A       I 1.0
R       Q 2.0
```

The last operations programmed are worked on by the control with priority. In the example the set operation is first implemented; the output Q 2.0 is again reset and remains reset for the remainder of program processing.

This brief setting of the output is accomplished only in the process image. A signal status on the pertinent I/O rack is not affected during program processing.

2.9.2 SET DOMINANT

In accordance with section 4.10.1. the exit Q 2.1 in this example is set with priority.



STL

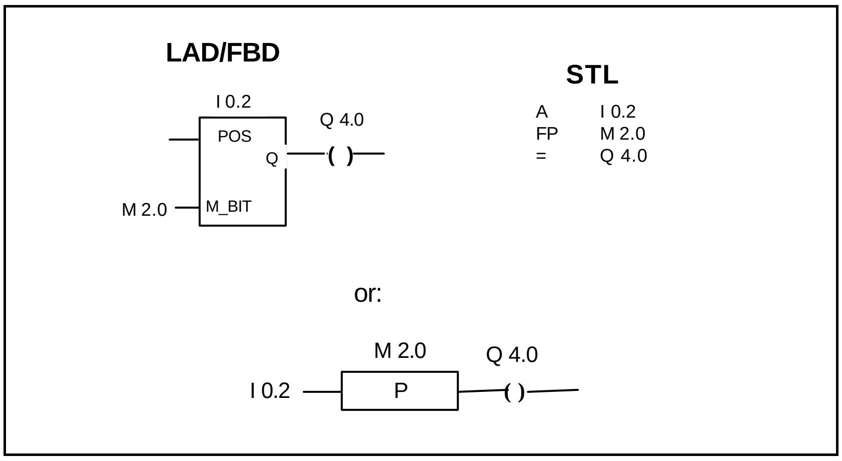
```
A       I 1.1
R       Q 2.1
A       I 1.0
S       Q 2.1
```

2.10 EDGE OPERATIONS

The edge (flank) operations collect in contrary to a static signal status "0" and "1" the *signal change* e.g. of a input. The program of an edge operation corresponds to an edge-recognizing contact in a relay circuit.

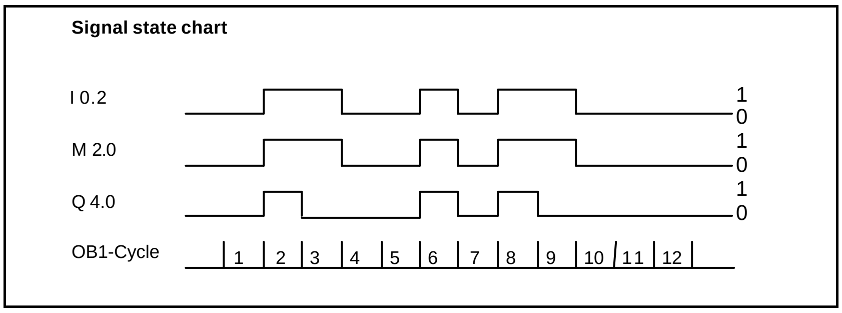
2.10.1 POSITIVE EDGE (FP)

If a rising (positive) edge (change from “0“ to “1“) is recognized by I 0.2, then Q 4.0 for a OB1-Cycle is set to “1“. This output can be again used e.g. to set a memory bit. A rising edge is recognized, as the automation system stores the RLO, which supplied the operation A, in the edge memory bit M 2.0 and compares it with the RLO of the preceding cycle.
The advantage of the second type of representation in LAD/FBD is that logical operations can also be present at the input of the edge operation.



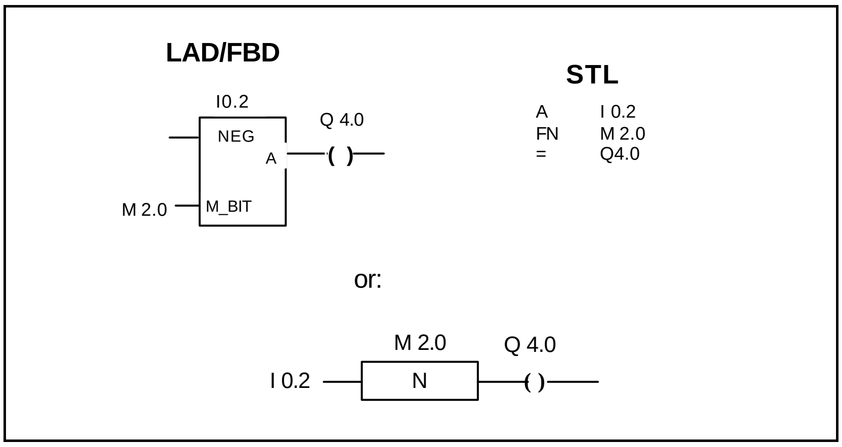
STL

```
A   I 0.2
FP  M 2.0
=   Q 4.0
```



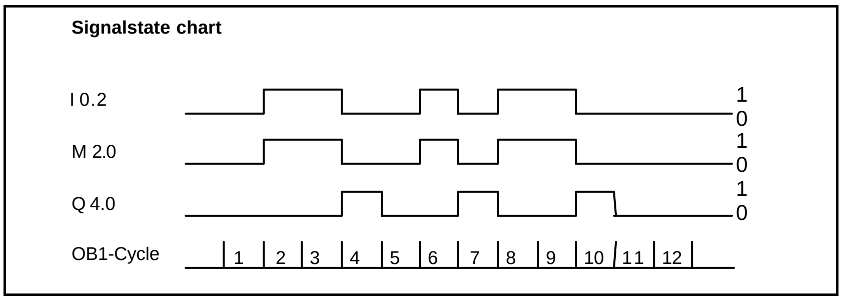
2.10.2 NEGATIVE EDGE (FN)

If a falling (negative) edge (change of “1“ to “0“) is recognized by I 0.2, then Q 4.0 for a OB1-Cycle is set to “1“. This output can be used again e.g. to set a memory bit. A falling edge is recognized, as the automation system stores the RLO, which supplied the operation A in the edge memory bit M 2.0, and compares it with the RLO of the preceding cycle. The advantage of the second type of representation in LAD/FBD is that logic operations can also be present at the input of the edge operation.



STL

```
A     I 0.2
FN    M 2.0
=     Q4.0
```



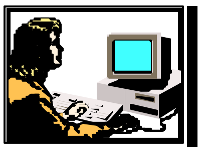
2.11 TIMER FUNCTIONS

For the realization of control tasks, different timer functions must be frequently used. The timer functions are integrated in the CPU of the automation equipment. The setting of the desired running time and the starting of the timer function must be made by the user program. The SIMATIC - Automation devices place a certain number of timer elements (CPU dependent) with different timer functions at one's disposal. A 16-bit-word is assigned to each of the time elements.

The following functions can be programmed with a timer:

2.11.1 RELEASE TIMER (FR) ONLY IN STL

A positive edge change (from “0“ to “1“) in the operation result of the release timer operation (FR) will release a timer.
For starting or for the normal function of a timer, the release is not needed. The release is used only in order to re-trigger a current time i.e. to let it start again. This restart is possible only if the starting operation is edited further with the RLO ‘1‘.



The operation release (FR) exists only in the programming language STL

2.11.2 START TIMER (SI/SE/SD/SS/SF)

With a signal change at the start input (positive edge), the timer is started. In order to start a timer, you must insert three operations in its STL program:

- *Query of a signal status*
- *Load a starting time into ACCU 1*
- *Start operations (alternatively SI, SE, SD, SS or SF)*

```
e.g.:
A     I 0.0
L     S5T#2S
SE    T5
```

Forward Program instructions

2.11.3 TIMER VALUE (TV)

A timer should always execute for a certain time. The length of time value TV can be assigned either as a pre-defined constant in the program or can be given as data to an input word IW, to an output word QW, to a data item DBW/DIW, to a local word LW or to a memory bit word MW. Updating the time decreases the current value in each case by a unit in an interval, which was specified by the time base. You can load a pre-defined current value with the following syntax:

- *L W#16#abcd*
  - with: a = binary coded time base(e.g. time interval or representation unit; see below)
  - bcd = time value in BCD- Format
- *L S5T#aH_bbM_ccS_dddMS*
  - with: a = hours, bb = minutes, cc = seconds and ddd = Milliseconds
  - The time basis is selected automatically

Time base:

The time base defines the interval, in which the time is decreased by a unit. Values with no exact multiple of the time interval are cut off. Values, whose representation unit for the desired range is too large, are rounded off.

Time basis	Binary code	Time length
10ms	00	10MS to 9S_990MS
100ms	01	100MS to 1M_39S_900MS
1s	10	1S to 16M_39S
10s	11	10S to 2H_46M_30S

2.11.4 RESET TIMER (R)

A signal at the reset input terminates the processing of the timer. The current time is deleted and the output Q of the time cell is reset.

2.11.5 LOAD TIMER (L/LC)

A time is stored in a binary coded time word. The value in the word can be loaded as a dual number (DUAL) or as a BCD number (DEC) into the ACCU and be transferred from there into other operands - ranges. With STL programming, you have the choice between L T1 for the query of the dual number and LC T1 for the query of the BCD number.

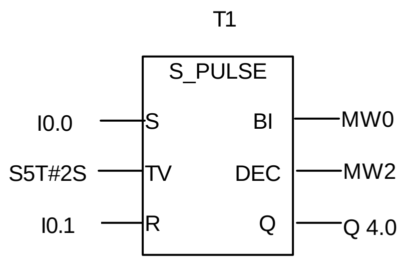
2.11.6 QUERY SIGNAL STATE OF TIMER (Q)

A timer can be queried on its signal status of (“0“ or “1“). Signal statuses can be queried - with A T1, AN T1, ON T1, etc.... and can later be used for further logical operations.
You can select five different timers:

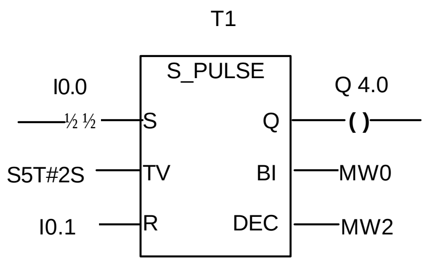
2.11.7 PULSE TIMER (SI)

The output of a timer, which is started as a pulse, is fed a signal status 1 after starting. (1). The output is reset, if the programmed length of time has elapsed (2), if the starting signal is reset to zero (3) or if at the reset input of the timer, a signal status 1 is applied (4). A positive edge change (of “0“ to “1“) in the logical operation result of the operation release (FR), which starts the time again (5). This restart is possible only if the starting operation is edited further with the RLO ‘1‘.

FBD



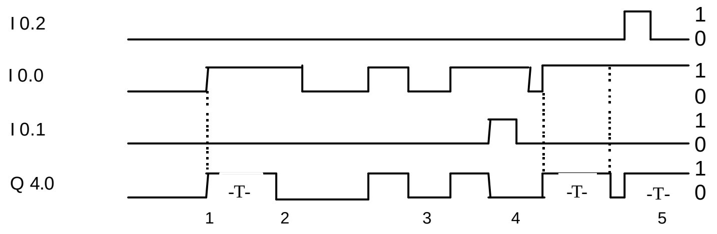
LAD



STL

```
A    I 0.2
FR   T1       Release of the timer T1
              (only in STL)
A    I 0.0
L    S5T#2S   Load start time (2s) in ACCU 1
SI   T1       Start timer T1 as a pulse
A    I 0.1
R    T1       Reset timer T1
L    T1       Load timer T1 DUAL-coded
T    MW0
LC   T1       Load timer T1 BCD-coded
T    MW2
A    T1       Query of the timer T1
=    Q 4.0
```

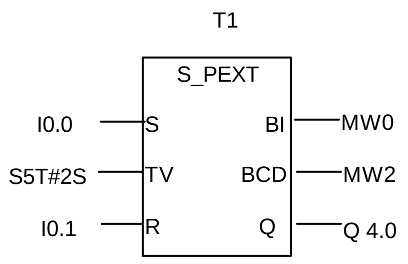
Signal state chart with release (FR)



2.11.8 EXTENDED PULSE TIMER (SE)

The output of a timer, which is started as an extended pulse, is fed a signal status 1 after starting (1). The output is released, if the given length of time has elapsed (2) or if the resetting input of the timer function is switched on (5).
When the time runs, switching the start input off does not cause the output to reset (locking) (3). Step - while the time still runs - a renewed signal changes on 1 at the start input and the timer is again started (re-triggered) (4).

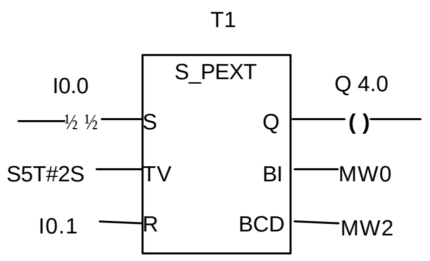
FBD



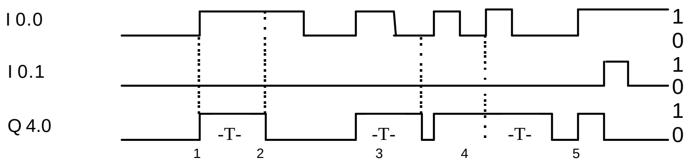
STL

```
A    I 0.0
L    S5T#2S   Load start time (2s) in ACCU 1
SE   T1       Start timer T1 as a pulse
A    I 0.1
R    T1       Reset timer T1
L    T1       Load timer T1 DUAL-coded
T    MW0
LC   T1       Load timer T1 BCD-coded
T    MW2
A    T1       Query of the timer T1
=    Q 4.0
```

LAD



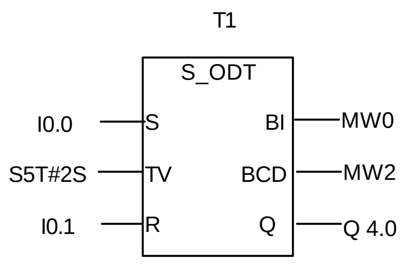
Signalstate diagram



2.11.9 ON –DELAY TIMER (SD)

The output of a timer, which is started as a signal delay, is fed a signal status 1 after starting only if the programmed time has elapsed and the RLO 1 is applied at the start input (1). The switching on of the start input also causes a switching on of the output Q in the given length of time. The output will reset, if the start input is switched off (2) or if a signal status of 1 is applied at the reset input of the timer (3). The output Q is not switched on if during time running, the start input is switched off or a signal status of 1 at the reset input of the timer is closed.

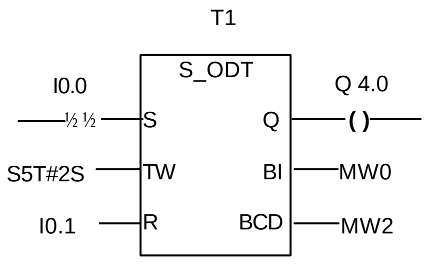
FBD



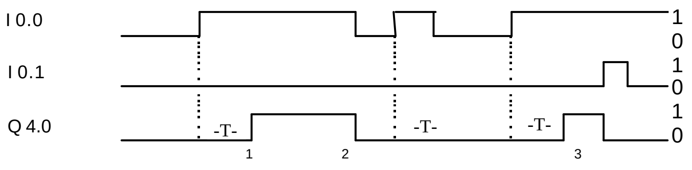
STL

```
A   I 0.0
L   S5T#2S   Load time (2s) in ACCU 1
SD  T1       Start timer T1 as a pulse
A   I 0.1
R   T1       Reset timer T1
L   T1       Load timer T1 DUAL-coded
T   MW0
LC  T1       Load timer T1 BCD-coded
T   MW2
A   T1       Query of the timer T1
=   Q 4.0
```

LAD



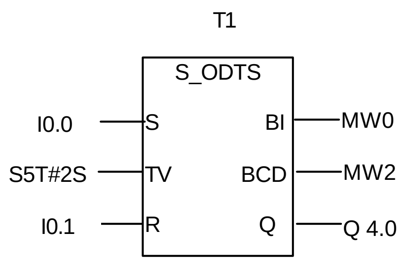
Signal state diagram



2.11.10 RETENTIVE ON-DELAY TIMER (SS)

The output of a timer, which is started as a retentive ON delay, is fed a signal status 1 after starting only if the programmed time has elapsed (1). The function no longer requires an RLO 1 after starting at the start input, thus it cannot be switched off (locking) (3).
The output is reset only if the reset input of the timer function is switched on (2). As long as the time is running, a switching off and renewed switching on of the start input causes the timer function to become once again started (re-triggered) (4).

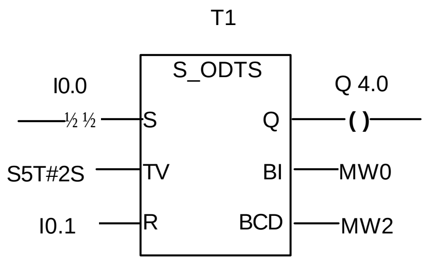
FBD



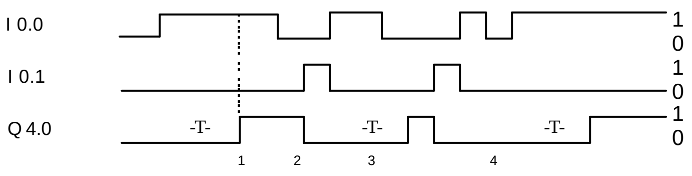
STL

```
A    I 0.0
L    S5T#2S    Load time (2s) in ACCU 1
SS   T1        Start timer T1 as a pulse
A    I 0.1
R    T1        Reset timer T1
L    T1        Load timer T1 DUAL-coded
T    MW0
LC   T1        Load timer T1 BCD-coded
T    MW2
A    T1        Query of the timer T1
=    Q 4.0
```

LAD



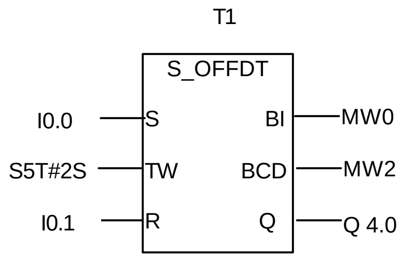
Signalstate diagram



2.11.11 OFF-DELAY TIMER (SF)

With a signal change (positive edge) at the start input of a timer which is started as a switched off delay, the output Q of the timer function is switched on (1). If the start input is switched off, the output is still supplied with the signal status 1 until the programmed time has elapsed (2). Switching the start input (negative edge) around the given length of time, causes the switching of the input off. The output of the timer is also switched off, if at the reset input, the signal status 1 is applied (4). While the time runs, the renewed switching on of the time function causes the execution time to be stopped and then again started only by the next switching off of the start input (3).
If both the start input and the reset input of the timer function are fed a signal status 1, the exit of the timer will only be set if the dominate reset is switched off (5).

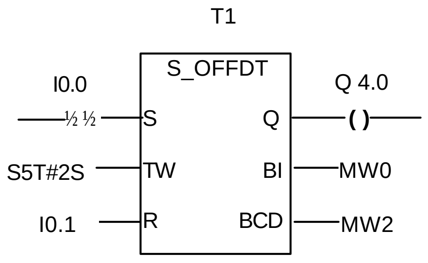
FBD



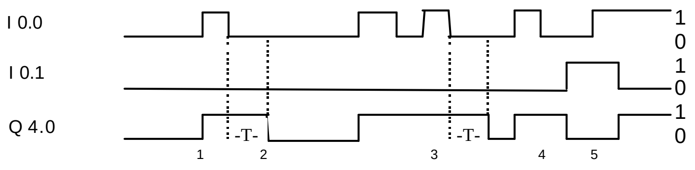
STL

```
A   I 0.0
L   S5T#2S   Load start time (2s) in ACCU 1
SF  T1       Start timer T1 as a pulse
A   I 0.1
R   T1       Reset timer T1
L   T1       Load timer T1 DUAL-coded
T   MW0
LC  T1       Load timer T1 BCD-coded
T   MW2
A   T1       Query of the T1
=   Q 4.0
```

LAD



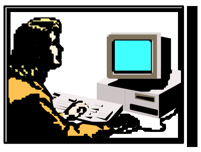
Signal state diagram



Forward | **Program instructions**

2.12 CLOCK PULSE GENERATORS

Clock pulse generators clocks are used for different checking -, monitoring and control tasks. In digital technology, they are called astable trigger circuits. Frequently in operation practice, one needs a flash frequency or fault signals.

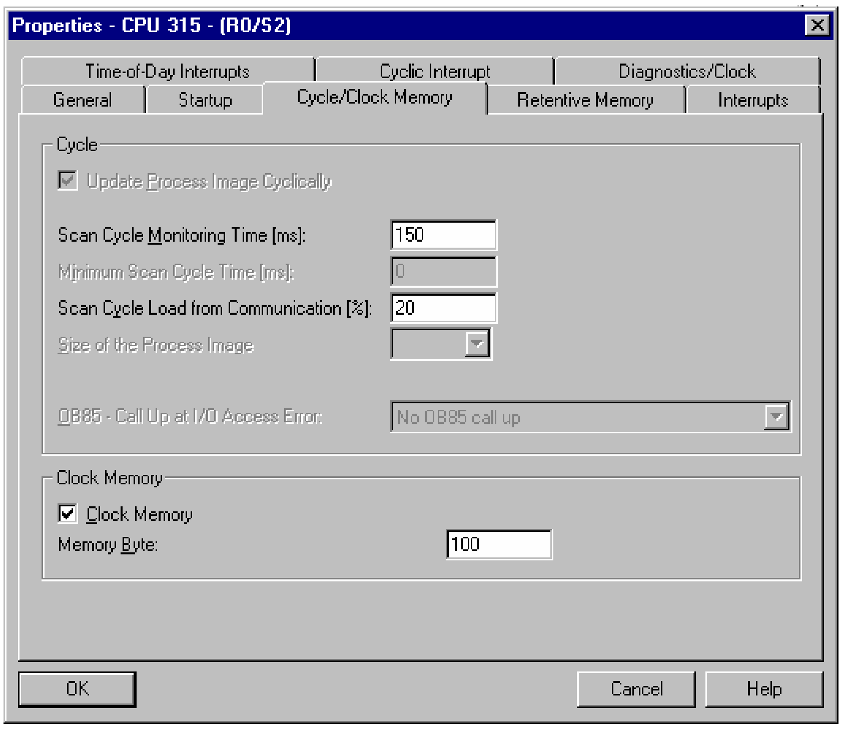


In the CPU S7-300, there exists a parameterized clock memory, that can be set with the Tool S7 Configuration.

Clock memory parameterizing:

Clock memories are memory bits inside of “clock memory bytes“. Any memory bit of the CPU becomes a “clock memory byte“ by parameterizing (Double click on the CPU line in the Tool Configuration!). A clock memory changes its binary value periodically.

If you activate Clock Memory (check is visible in the small control box), then you must also determine the number of the memory byte. The selected memory byte cannot be used for the intermediate storage of data.



Cycle
- [✓] Update Process Image Cyclically
- Scan Cycle Monitoring Time [ms]: 150
- Minimum Scan Cycle Time [ms]: 0
- Scan Cycle Load from Communication [%]: 20
- Size of the Process Image:
- OB85 - Call Up at I/O Access Error: No OB85 call up

Clock Memory
- [✓] Clock Memory
- Memory Byte: 100

Period length from clock:

Each bit of the clock memory byte is assigned a period length/frequency. The following assignments apply:

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Period length (s):	2	1.6	1	0.8	0.5	0.4	0.2	0.1
Frequency (Hz):	0.5	0.625	1	1.25	2	2.5	5	10

Forward	**Program instructions**

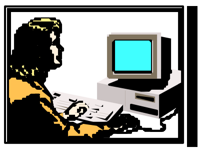
2.13 COUNTER OPERATIONS

In control engineering, counter functions are needed for collecting the number of items or pulses and for the evaluation of times and distances. In the SIMATIC S7, counters are already integrated in the CPU. These counters possess their own reserved storage area. The range of the count value lies between 0 and 999.

The following functions can be programmed with a counter:

2.13.1 RELEASE COUNTER (FR) ONLY IN STL

A positive edge change (of “0“ to “1“) in the logical operation of the operation release (FR) releases a counter.
A counter release is not needed for setting a counter or for normal counting operations. However, if one wants to set a counter without a rising edge before the appropriate counting operation (CU, CD or S), then this can take place with a release. This is however possible only if the RLO bit before the appropriate operation (CU, CD or S) has a signal status “1“.



The operation release (FR) only exists in the programming language STL.

2.13.2 COUNTER UP (CU)

The value of the addressed counter is increased by 1. The function becomes effective only with a positive edge change of the logical operation programmed before CU. If the count value achieves the upper limit of 999, it is no longer increased. (*a carry is not generated!*)

2.13.3 COUNTER DOWN (CD)

The value of the addressed counter is reduced by 1. The function becomes effective only with a positive edge change of the logical operation programmed before CD. If the count value achieves the lower limit 0, it is no longer reduced. *(Only positive counter values!)*

2.13.4 SET COUNTER (S)

In order to set a counter, you must insert three operations into its STL program:

- Query a signal status
- Load a count value
- Set a counter with the loaded count of the function. This function is only edited by a positive edge change of the query.

```
e.g.:
A     I 2.3
L     C#5
S     C1
```

2.13.5 COUNTER VALUE (CV)

If a counter is set, then the contents of ACCU 1 are used as the count . There is a possibility to code the count value either as binary or BCD code. The following operands are possible:

-	*Input word*	*IW ..*
-	*Output word*	*QW ..*
-	*Memory bit word*	*MW ..*
-	*Data word*	*DBW/DIW ..*
-	*Local data word LW ..*	
-	*Constant*	*C#5, 2#...etc.*

2.13.6 RESET COUNTER (R)

The counter is set to zero (to reset) with RLO 1. The counter remains unchanged with RLO 0. Resetting a counter works statically. During a satisfied resetting condition, a counter can be neither set nor counted.

2.13.7 LOAD COUNTER (L/LC)

A count is stored in a counter word binary code. The value in the counter can be loaded as a dual number (DU) or as BCD number (DE) into the ACCU and be transferred from there into other operand ranges. With STL programming, you have the choice between *L C1* for the query of the dual number and *LC C1* for the query of the BCD number.

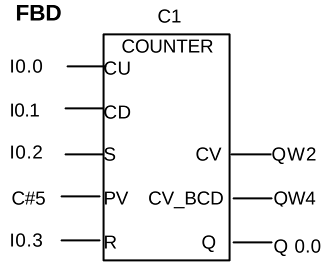
2.13.8 QUERY SIGNAL STATE OF COUNTER (Q)

A counter can be tested for its signal status. The meaning of the signal states are:

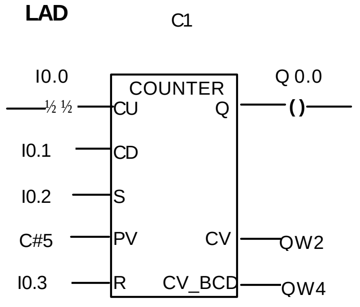
Signal state 0	=	*Counter stays on the value 0;*
Signal state 1	=	*Counter runs, i.e. it is count ready.*

Signal statuses can be queried with A C1, AN C1, ON C1, etc.... and can be used for further logical operations.

FBD



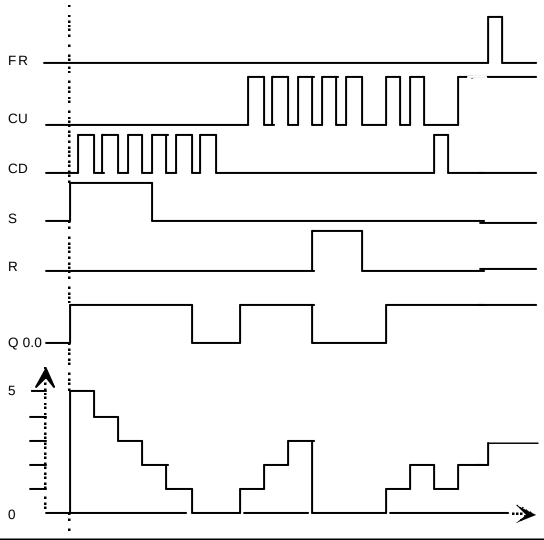
LAD



STL

```
A   I 0.7     Release (only in STL)
FR  C1
A   I 0.0
CU  C1        Count up
A   I 0.1
CD  C1        Count down
A   I 0.2
L   C#5       Load counter with default value
S   C1        Set counter with default value
A   I 0.3
R   C1        Rest counter C1
L   C1        Load counter C1 DUAL-coded
T   QW2
LC  C1        Load counter C1 BCD-coded
T   QW4
A   C1        Query of the counter C1
=   Q 0.0
```

Signal state chart:

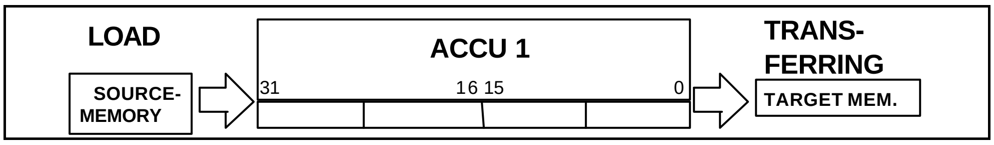


2.14 LOAD-AND TRANSFER OPERATIONS (L/T) ONLY IN STL

In the programming language STEP 7, load and transfer operations make byte -, word -, and/or the double-word orientated exchange of information between input and output modules, the process-image of the input and output, the timer, the counter, and memory bit storage as well as data blocks possible. This information exchange is not made directly, but always by the accumulator 1 (ACCU 1). The ACCU 1 is a register in the processor and serves as a buffer.

The information flow is directed as follows:

LOAD: *from the source memory into the ACCU 1*
TRANSFERRING: *of the ACCU into the target memory*



While the loading contents of the addressed source memory are copied and written into the ACCU 1, The previous ACCU content is transferred into the ACCU 2. When transferring, the contents of ACCU 1 are copied and written into the addressed target memory.
Since the accumulator content was only copied, it is available for further transfer operations.

STL:

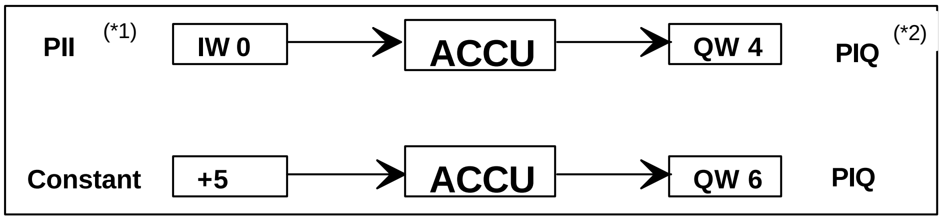
```
: L   IW    0

: T   QW    4

: L   +5

: T   QW    6

: BE
```



*1: Process-image of the input area   *2: Process-image of the output area

Load and transferring are absolute operations, which are implemented independently of the logical operations result in each cyclic circulation.

2.15 COMPARISON FUNCTIONS

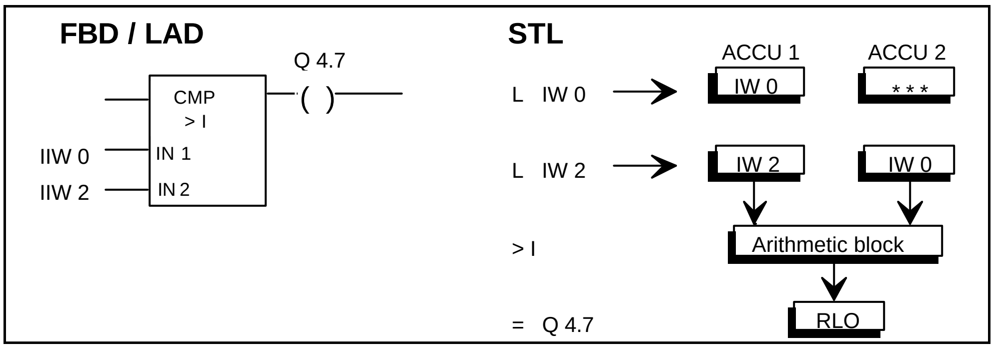
The programming language STEP 7 offers the possibility of comparing two numerical values directly and advancing the result of the comparison (RLO) immediately. A condition for it is that both numbers have the same number format. The following pairs of numerical values can be compared:

· *two integers (16 Bit*	*Symbol: I)*
· *two integers (32 Bit*	*Symbol: D)*
· *two real numbers (Floating point numbers. 32 Bit,*	*Symbol: R)*

There are 6 different comparison operations to choose from:

Is word of counter C1 (in Accu 2)			To word of counter C2 ? (in Accu 1)
	equal	= =	
	unequal	<>	
	greater or equal	> =	
	greater	>	
	smaller or equal	< =	
	smaller	<	

With the comparison functions, two values which lie in the ACCUs 1 and 2 are compared directly with each other. With the first load operation, the first operand (e.g. IW 0) is loaded into ACCU 1. With the second load operation, first the first operand is reloaded by the ACCU 1 into ACCU 2 and then the second operand (e.g. IW 2) is loaded into ACCU 1. Afterwards the numerical values in the arithmetic block in both accumulators are compared with one another bit by bit. The result of the comparison is binary. If the desired comparison is satisfied, the operation result becomes 1. If the desired comparison is not satisfied, then the RLO becomes 0.



Forward **Program instructions**

2.16 PROGRAM ORGANIZATION

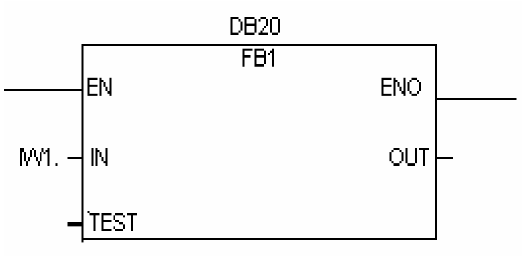
2.16.1 BLOCK CALL (CALL)

With the module call *CALL*, you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). At the same time parameters can be transferred and/or variables described as well as opened in the FB or SFB associated local data blocks (See: *further reference function "Variable declaration in code blocks"*). If no variables are defined in the called block, then this operation corresponds to the operation UC.

STL

CALL	FB1, DB20
IN := IW 1	IN (Formal parameter) assigned to IW 1 (Actual parameter).
OUT :=	OUT (Formal parameter) assigned to no parameter.
TEST :=	TEST (Formal parameter) assigned to no parameter.

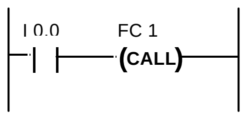
LAD/FBD



2.16.2 CONDITIONAL CALL (CC)

With the block call *CC* you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). However, they cannot transfer any parameters and/or describe variables. The call is implemented only if the logical operation result amounts to a "1".

LAD/FBD



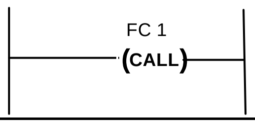
STL

```
A    I 0.0
CC   FC 1
```

2.16.3 UNCONDITIONAL CALL (UC)

With the module call *UC,* you can call functions (FCs) and functional blocks (FBs) as well as system functions (SFCs) and system function blocks (SFBs). They can transfer however no parameters and/or describe variables. The call is implemented independently from the logical operation result.

LAD/FBD



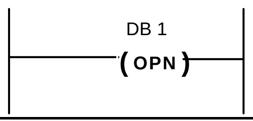
STL

```
UC      FC 1
```

2.16.4 OPEN A DATA BLOCK (OPN)

With the operation open data block (OPN), you can open a data block (DB) or instance -data block (DI), in order to access the contained data (e.g. with load and transfer operations).

LAD/FBD



STL

```
OPN   DB 1
L     DBW 0
T     MW 1
```

2.16.5 BLOCK END CONDITIONAL (BEC) ONLY IN STL

Depending on the logical operation result, this operation terminates the processing of the current block and jumps back into the block that was previously called. This operation occurs only if the logical operation result amounts to "1".

STL

```
A       I 0.0
BEC
```

2.16.6 BLOCK END UNCONDITIONAL (BEU) ONLY IN STL

This operation terminates the processing of the current block and jumps back into the previous block. This operation occurs independently from the logical operation result.

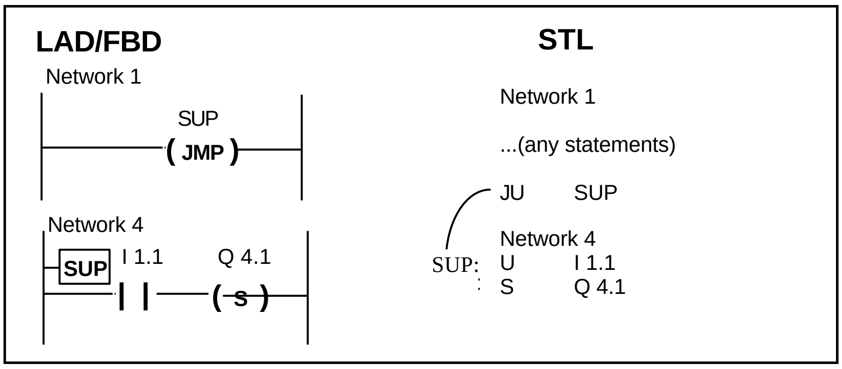
STL

```
...(any statements)
BEU
```

2.17 JUMP OPERATIONS

2.17.1 JUMP UNCONDITIONAL (JU)

The operation JU interrupts the normal execution of the program and jumps to the branch label indicated in the operand. The jump occurs independently from the logical operation result.

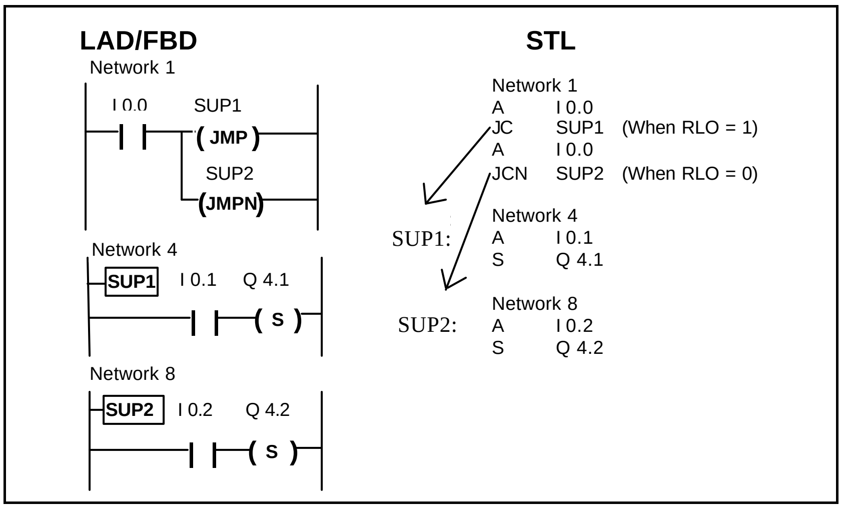


```
Network 1
...(any statements)
      JU      SUP
      Network 4
SUP:  U       I 1.1
 :    S       Q 4.1
```

2.17.2 JUMP IF RLO=1/RLO=0 (JC/JCN)

The conditioned jump operations interrupt the normal execution of the program and initiate a jump to the branch label indicated in the operand. The jump takes place as a function of the logical operation result. The following conditioned jump operations can be implemented:

· *JC :*	*Jump when RLO = 1*
· *JCN :*	*Jump when RLO = 0*

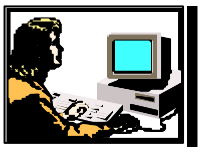


```
       Network 1
       A      I 0.0
       JC     SUP1   (When RLO = 1)
       A      I 0.0
       JCN    SUP2   (When RLO = 0)
       Network 4
SUP1:  A      I 0.1
       S      Q 4.1
       Network 8
SUP2:  A      I 0.2
       S      Q 4.2
```

2.17.3 LOOP (LOOP) ONLY IN STL

With a program loop (LOOP), you can edit a program section several times. In addition you must load a constant into the low order word from the ACCU 1. This number is then decreased by '1' by the operation LOOP. Afterwards the value of this number is examined for < > 0. If it does not amount to '0', then a jump is implemented to the label of the operation LOOP; otherwise the next operation is implemented.

```
      L      5
NEXT: T      MB 10
        |
        |
      L      MB 10
      LOOP   NEXT
```



The program loop (LOOP) exists only in the programming language STL.

2.18 NULL OPERATIONS

2.18.1 NULL OPERATION 0 / 1 (NOP 0/NOP 1) ONLY IN STL

These operations do not implement a function and do not affect contents of the status words. The compiler needs the null operations for re-compilation, e.g. from STL into LAD.

Forward Program instructions

2.19 PROCESSING OF THE RLO

In STEP 7 there are operations with which the logical operation result (RLO) can be altered. Since the RLO is directly affected, these operations do not possess operands.

2.19.1 NEGATE RLO (NOT) ONLY IN STL

You can negate (return) the actual RLO in a program with the operation NOT. If the actual RLO is ‘0’, then the operation NOT changes it into ‘1’.

2.19.2 SET RLO (SET) ONLY IN STL

You can set the RLO-bit unconditionally to ‘1‘ in the program with the operation SET.

2.19.3 RESET RLO (CLR) ONLY IN STL

You can reset the RLO-bit unconditionally to ‘0‘ in the program with the operation CLR.

2.19.4 SAVE RLO (SAVE) ONLY IN STL

You can save the RLO for future use in the status bit (BR) of the status word in the program with the operation SAVE.

The status word contains bits, which you can access in the operand of the bit and word logical operations.

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
e.g.:	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Statement list:	Signal state:	Logical operation result(RLO):
SET		1
= M 1.0	1	
= I 0.0	1	
CLR		0
= M 1.0	0	
= I 0.0	0	
NOT	1	
SAVE	1	save in BR- Bit in the status word

Forward	Program instructions