

# SIEMENS

## Programmable Controller

### SIMATIC S5 - 101R

Programming Instructions

Order No.: GWA 4NEB 810 2039-02

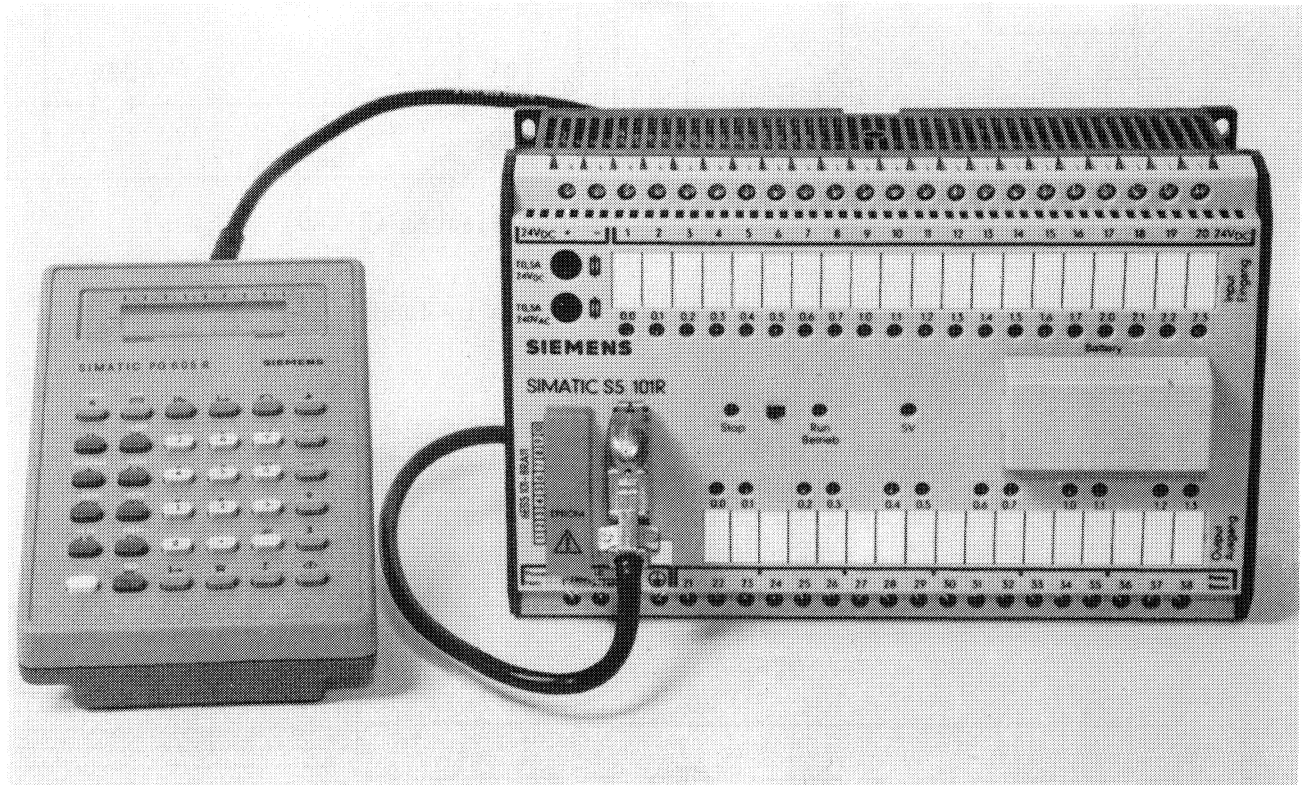


Fig. 1 S5 101R programmable controller with 605R programmer

Contents	Page	Contents	Page
1. The programming language	1.1	3. Program test	3.1
1.1 Formal rules for program generation	1.2	3.1 Search function	3.1
1.2 Binary program elements	1.5	3.2 Signal status display	3.2
1.2.1 NO and NC contacts	1.5	3.3 Forcing	3.2
1.2.2 Outputs (coils) and flags (internal relay equivalents)	1.6	4. Storing the program	4.1
1.2.3 Latching and unlatching	1.7	4.1 Storing the program on an EEPROM submodule	4.1
1.3 Complex program elements	1.8	4.2 Duplicating EEPROM sub-modules	4.2
1.3.1 Timers	1.8	5. Operation set	5.1
1.3.2 Counters	1.12	5.1 Binary operations	5.1
2. Program generation with the 101R PC	2.1	5.2 Complex operations	5.4
2.1 Program input and correction	2.1		
2.2. Example of program generation	2.2		

# 1. The programming language

The 101R PC is programmed in the R-LAD ladder diagram method of representation. The R-LAD is a graphic method of representing the automation problem using circuit diagram symbols. It is both easy to understand and use.

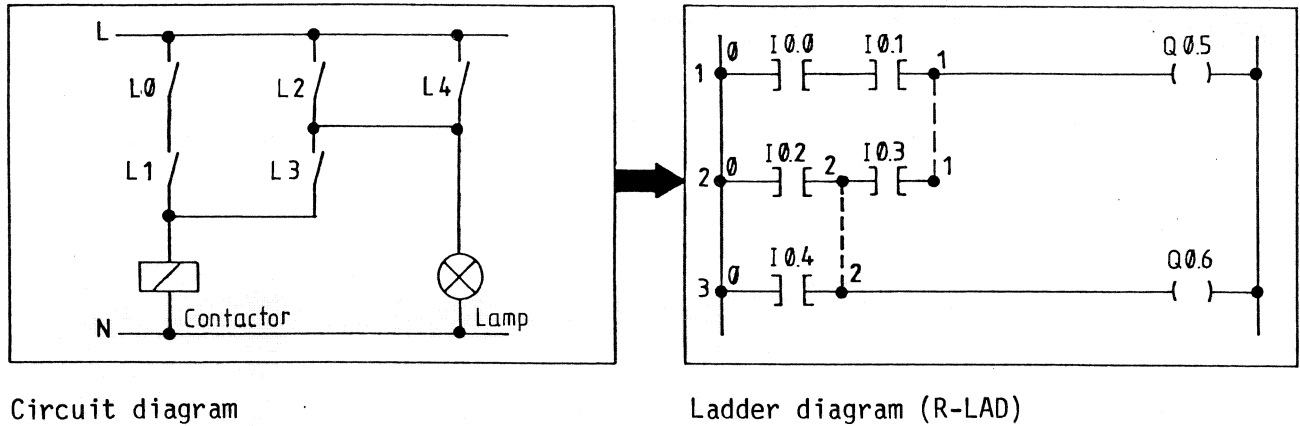


Fig. 2 Conversion of circuit diagram to ladder diagram

The vertical rungs of the circuit diagram are written from left to right in the lines of the programming form. Points with the same potential are described by means of nodes. The ladder diagram in R-LAD representation is entered directly into the programmer.

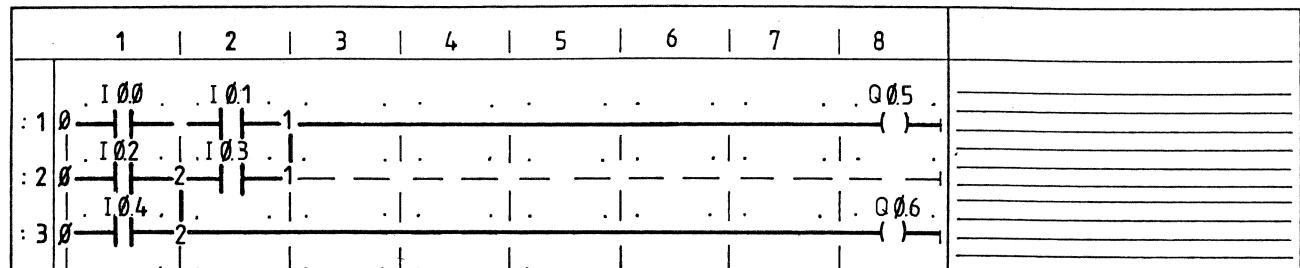


Fig. 3 Representation of the ladder diagram on the programming form

# 1.1 Formal rules for program generation

The program of the 101R PC can consist of a maximum of 16 program blocks (PB). They can be entered in the PC in any order. The PC arranges the program blocks in order and processes them in sequence.

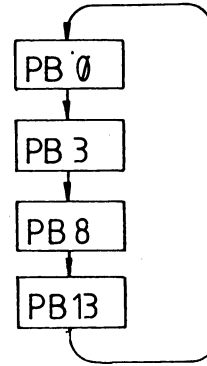


Fig. 4 Processing of PBs in the user program

A PB consists of up to 16 rungs and 24 contacts. 15 different nodes can be assigned for each PB. The nodes are numbered from 0-9 followed by A to E.

A rung can have a maximum of:

- 7 program elements for contacts, timers, counters
- 8 nodes
- 1 program element for flags or outputs

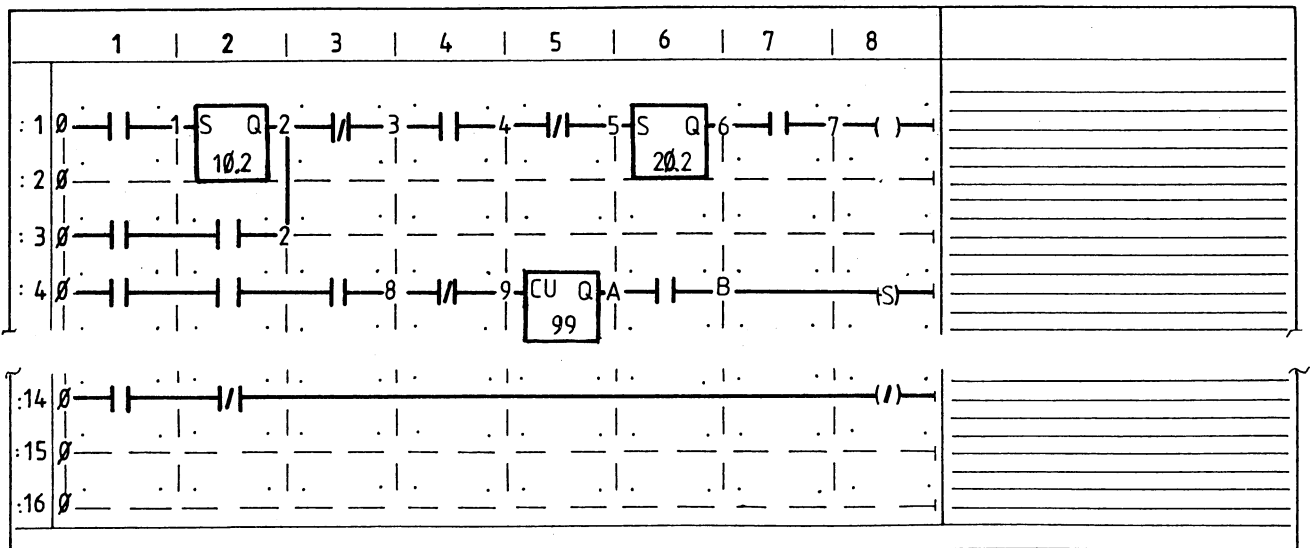
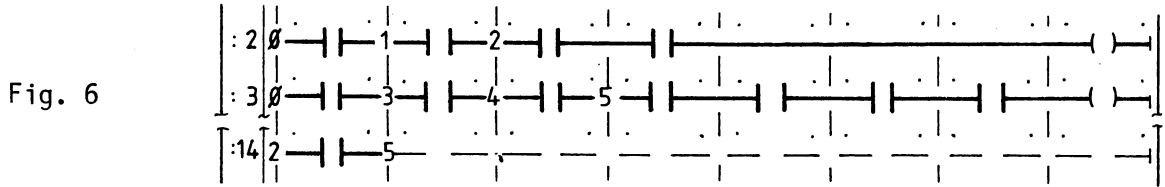


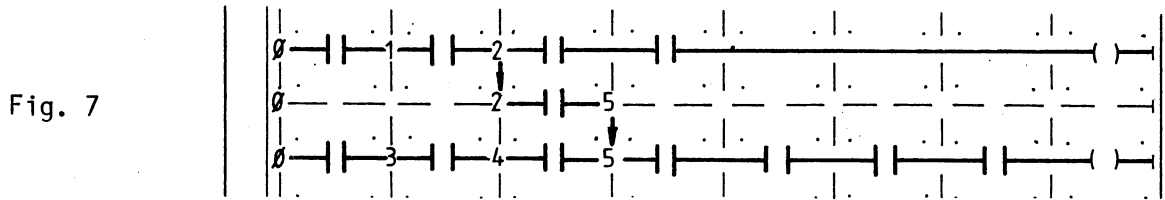
Fig. 5 Organization of a program block

When two rungs are linked by means of nodes, it must be remembered that signal flow is only possible from left to right (diode effect of the program elements). This must be taken particularly into account when adding rungs to PBs.

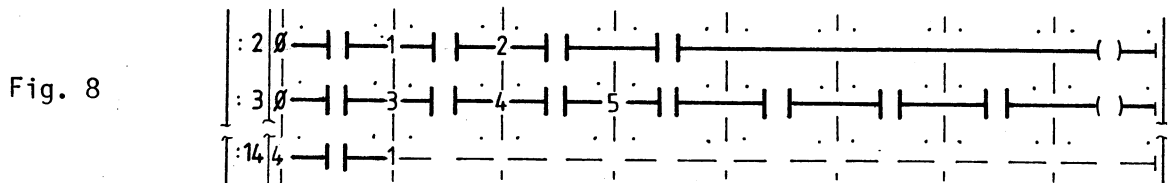
Example: Rung 14 can be added to the existing program section, i.e. rungs 2 and 3.



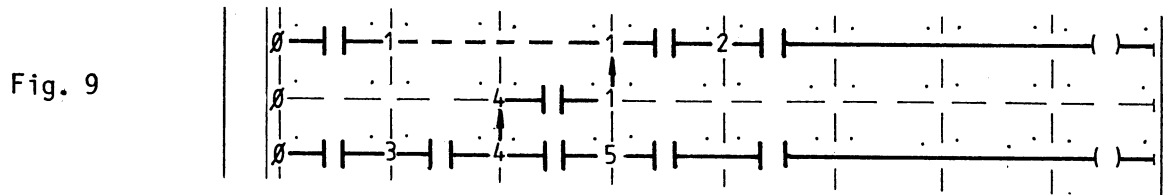
By rearranging the rungs, the correct signal flow from left to right becomes visible.



The apparently illegal insertion of rung 14 is also possible:

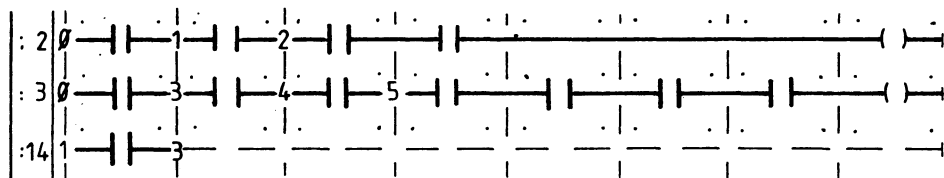


because the following rearrangement is possible by shifting rung 2 from node 1:



It is not possible, however, to add the following node 14:

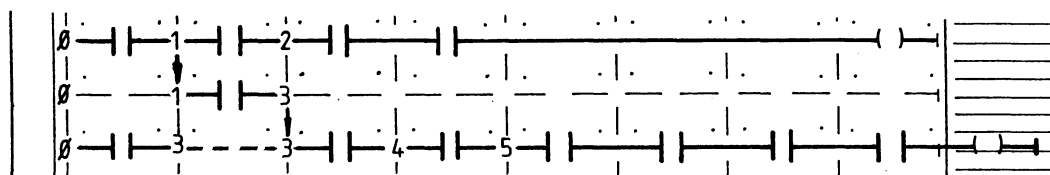
Fig. 10



After pressing the (SORT) key on the programmer, the program entered is checked. Error message E 1 appears, indicating that there are more than 8 program elements in one rung.

When arranging or "sorting" the block, the programmer attempts to maintain the prescribed signal flow from left to right. Rung 3 is therefore shifted to the right from node 3, as in the previous example, to make possible the insertion of rung 14:

Fig. 11



This results in a rung overflow as can be seen from the programming form. This error is recognized by the programmer and an error message appears.

The rungs are only rearranged internally in the programmer. This rearrangement is not displayed on the programmer.

# 1.2 Binary program elements

## 1.2.1 NO and NC contacts

In contrast to real NO and NC contactors in hard wired circuits through which a control current generally flows, the sensor signals are scanned in programmable controller systems for the signal state 0 or 1. The status of the program element is obtained by scanning the signal state of the sensor signal. This makes it possible for the opening or closing function to be simulated by the program.

Example:

Sensor signal from an energized NO contact or from a non-energized NC contact

Input LED bright, signal state of input is "1", i.e. input I0.0 is energized (I0.0 = "1")

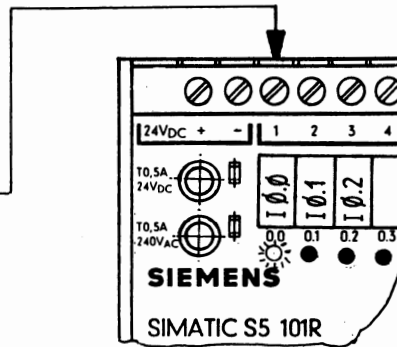


Fig. 12 "1" signal at input I0.0

State of program element if

- scanned for "1" signal: energized, signal flow is possible
- scanned for "0" signal: not energized, no signal flow possible

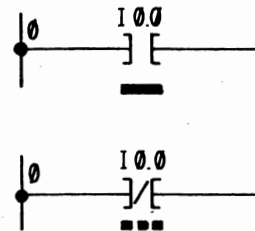


Fig. 13 Representation of the "1" signal on the programmer display

## 1.2.2 Outputs (coils) and flags (internal relay equivalents)

Outputs and flags are always located at the end of a rung if a signal state is assigned to them by the program (->).

They can also be scanned like contacts, e.g.  $\neg E Q0.1$ ,  $\neg E F1.3$ .

The signal states of the outputs are transferred to the output terminals when the PC is in RUN mode.

### Example

Output LED bright  
Signal state of output is "1".

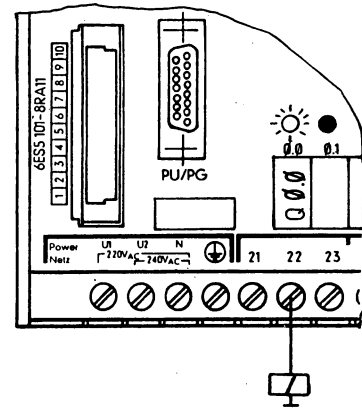


Fig. 14 Output Q0.0 is energized

If there is a signal flow via input I0.0, output Q0.0 is "1".

If there is a signal flow via I0.0, the negated output Q0.0 is "0".

If there is no signal flow via I0.0, the negated output Q0.0 is "1".

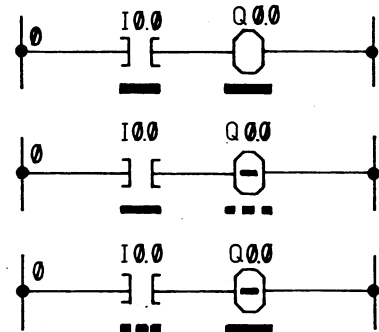


Fig. 15 Display on the 605R programmer

## 1.2.3 Latching and unlatching

Outputs and flags can be "latched" and "unlatched", i.e. the signal state assigned to them is latched until an inverse state is assigned.

### Example

#### Contactor with latching circuit

When the ON button is pressed, Q0.7 is set to "1" and latched.

It can be unlatched by actuating the OFF button, I0.1.

Contact C1 is not required due to the latching feature

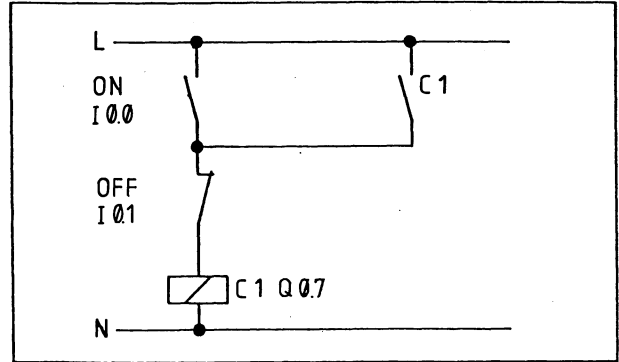


Fig. 16 Circuit diagram representation

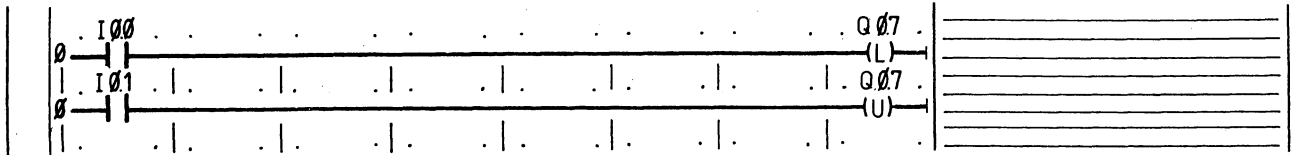


Fig 17 Ladder diagram representation

Scanning unassigned inputs in the unit with 10 inputs and 6 outputs (6ES5 101-8RB11) always results in "0" signal.



## 1.3 Complex program elements

Timers and counters are termed "complex functions" as further data are assigned to them in addition to the signal states "0" and "1".

Selecting one of the above functions on the programmer generates a "box" into which the specific data for the relevant function are "entered" in the next programming steps.

### 1.3.1 Timers

A timer is controlled via the START and HOLD inputs. After the set time has elapsed, output Q of the timer changes from "0" to "1", and output  $\bar{Q}$  from "1" to "0".

Starting the timer:                      Timer starts when there is a signal change from "0" to "1" at the START input.

e.g.

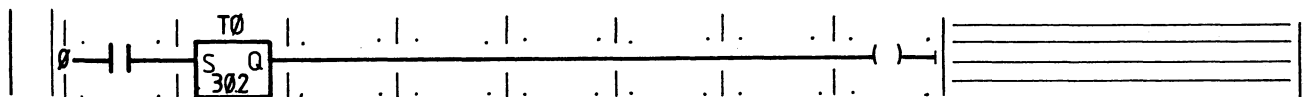


Fig. 18 Starting a timer

Entering the time:                      The time is entered in the "START TIMER" function box.

The desired time can be entered as a constant during programming. It has the form A.B.  
The letter B stands for the time base and the A for a constant multiplier.

A = 1 to 999

B = 0 for time base 10 ms  
1 for time base 100 ms  
2 for time base 1 s  
3 for time base 1 min

Example:                                      If 10.2 is entered, the time set is 10x1s = 10 seconds.

Timer tolerances:                      Each timer has a maximum inaccuracy of the order of the time base selected. It is therefore advisable to use the smallest time base possible.

Example:                                      Run time 8 seconds:

Representation    8.2 max. error    1 second  
                          80.1 max. error   0.1 seconds  
                          800.1 max. error 0.01 seconds

Holding the timer: When the signal changes from "0" to "1" at the HOLD input, the timer is stopped until a "0" signal appears at the HOLD input.

e.g.

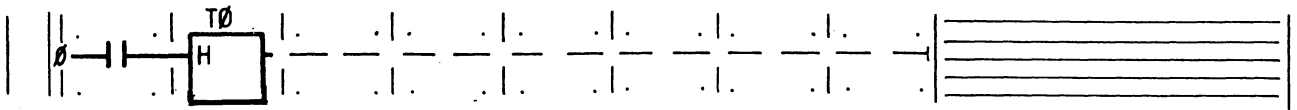


Fig. 19 Holding a timer

Resetting the timer: If the signal at the START input changes from "1" to "0", the timer is reset to the time 0.0 and output Q has a "0" signal and  $\bar{Q}$  a "1" signal.

Scanning the timer: The actual state of the timer can be seen in the "START TIMER" and "HOLD TIMER" function boxes for both Q and  $\bar{Q}$  outputs or can be scanned as a Tx contact element.

Formal rules for using timers in the program.

Timers can be incorporated into a rung like contacts.

Continuation of the output in the same rung is not mandatory; a timer can also be scanned as a contact.

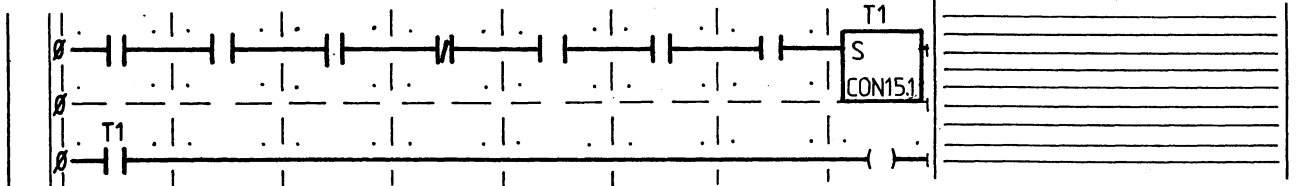


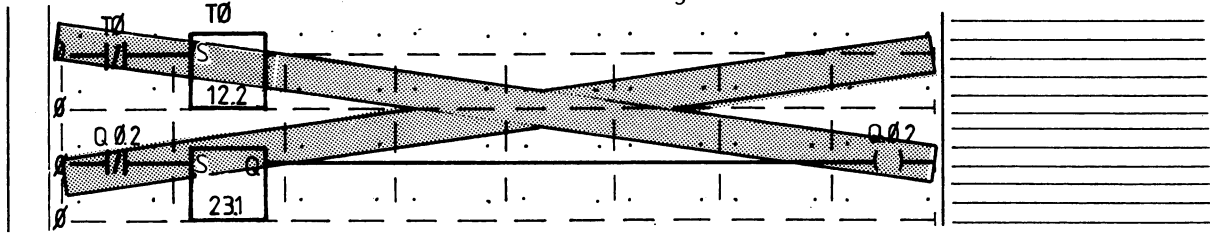
Fig. 20 Scanning a timer

Note:

- The output of a timer must not be brought back to the START input.
- When changing over from RUN to STOP, the timers running are stopped.

- When changing over from STOP to RUN, the stopped timers are reset.

Fig. 21



Typical examples

1. Delay, not latched.

Lamp K2 only lights up when contact B1 has been closed for longer than 1 second.

If there is a "1" signal at input I0.0, timer T1 is started. After 1 second, T1 has elapsed and output Q becomes "1".

If B1 opens ("0" signal at input I0.0), the timer is reset.

If input I0.0 changes from "1" to "0" while the timer is running, output Q stays unchanged at "0", and the timer is reset once more.

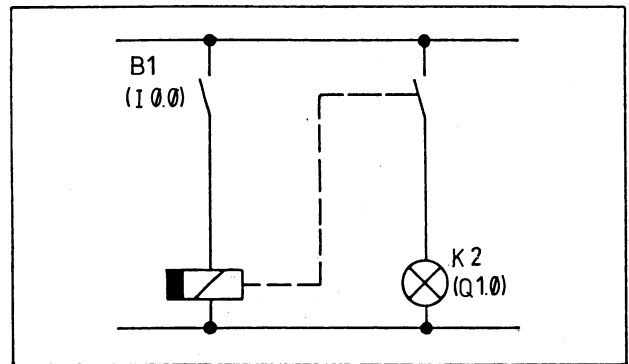


Fig. 22 "ON" delay in circuit diagram representation

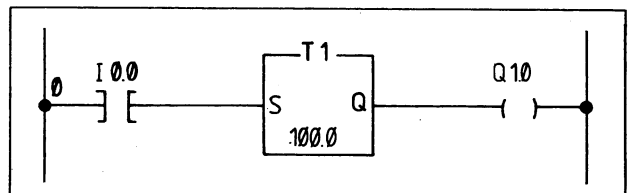


Fig. 23 "ON" delay in LAD representation

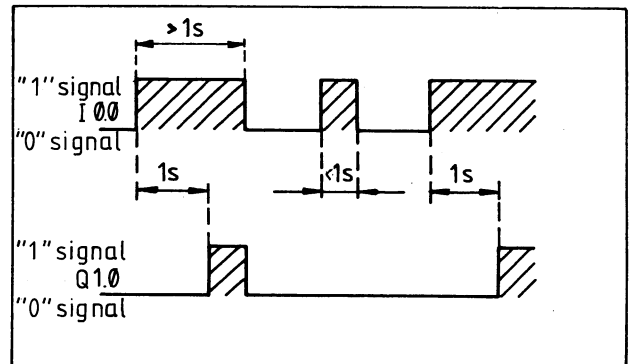


Fig. 24 Timing diagram of "ON" delay

## 2. "OFF" delay:

Time relay K2 remains energized for 10 minutes after contact B1 has opened.

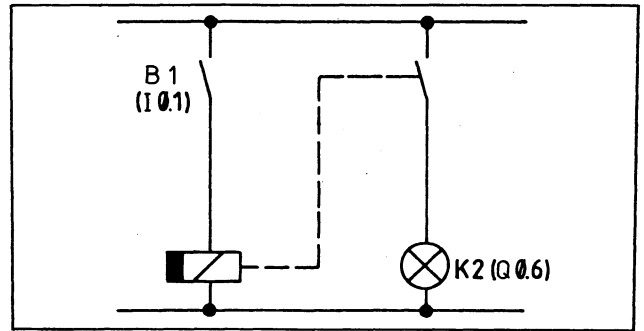


Fig. 25 "OFF" delay

If there is a "1" signal at input  $I0.1$ , negated input  $\overline{I0.1}$  is "0". A "1" signal at input  $I0.1$  latches output  $Q0.6$  to "1". If the status of input  $\overline{I0.1}$  changes from "0" to "1", timer T4 is started with a time of 10 minutes. When the time has elapsed and if there is a "1" signal at input  $\overline{I0.1}$ , output  $Q0.6$  is set to "0".

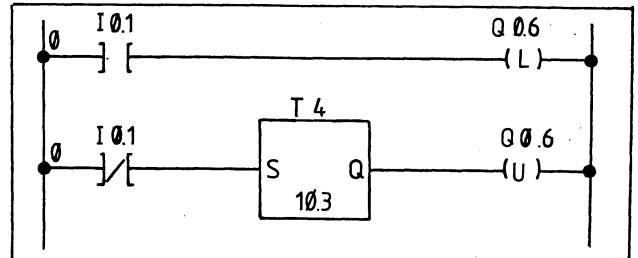


Fig. 26 "OFF" delay in LAD representation

If the signal state of input  $\overline{I0.1}$  changes from "1" to "0" while the timer is running, output  $Q0.6$  does not change its signal status and remains "1".

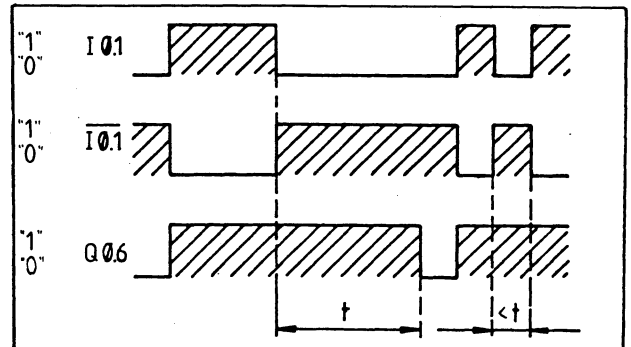


Fig. 27 Timing diagram of "OFF" delay

## 3. Clock pulse generator

After a "1" signal appears at input  $I0.3$ , outputs  $Q0.6$  and  $Q0.7$  are set alternately with selectable time constants.

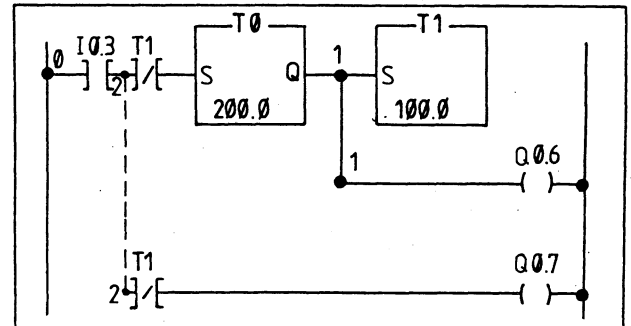


Fig. 28 Clock pulse generator

If input  $I0.3$  has a "1" signal, timer  $T0$  is started. After 2 seconds  $T0$  sets output  $Q0.6$  and simultaneously starts  $T1$ . After 1 second,  $T0$  is reset by  $T1$ . This resets  $Q0.6$  and  $T1$  and then  $T0$  can start again.

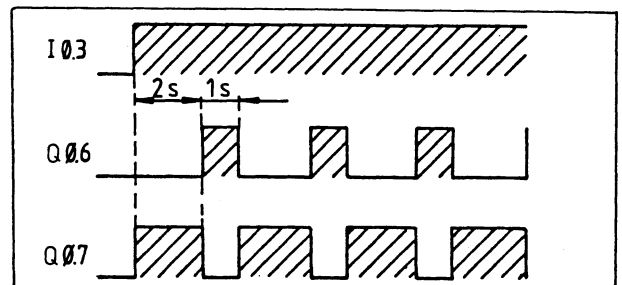


Fig. 29 Timing diagram of clock pulse generator

### 1.3.2 Counters

A counter is driven via the SET(S), COUNT UP(CU) and COUNT DOWN(CD) inputs.

When the relevant final count is reached, the output of the COUNT UP or COUNT DOWN element changes from "0" to "1".

Setting the counter: The counter is enabled and set to the initial count when the signal changes from "0" to "1" at the SET input

e.g.

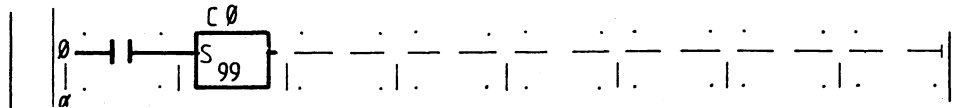


Fig. 30 Setting a counter

Counting up: Each time the signal changes from "0" to "1" at the COUNT UP input, the count is incremented by 1.

e.g.

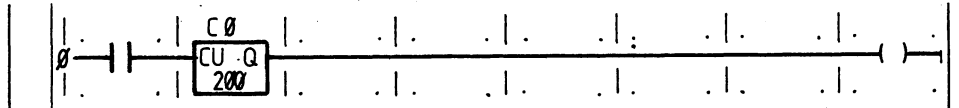


Fig. 31 Counting up

Counting down: Each time the signal changes from "0" to "1" at the COUNT DOWN input, the count is decremented by "1".

e.g.

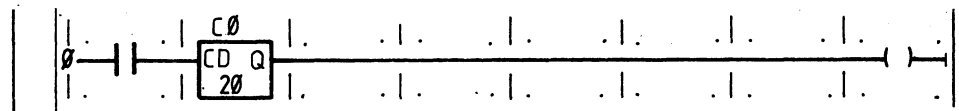


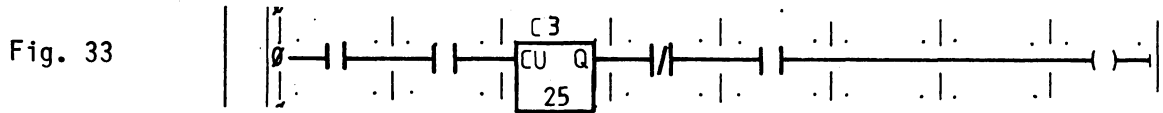
Fig. 32 Counting down

Scanning the counter: The current counter state can be ascertained via the outputs Q or  $\bar{Q}$  of the count up and count down counter elements.

Entering counts: The initial count and limit values of the counter can be input as constants (CON). The counting range is from 0 to 32767.

## Formal rules for using counters in the program

Counters can be incorporated into a rung like contacts.



Scanning or continuation of the output in the same rung is not mandatory; a counter can also be scanned as a contact. It must be noted, however, that the outputs of the COUNT UP and COUNT DOWN counting elements are not identical. The counter cannot be scanned as count up and count down counting element but only as an up counter or down counter.

For this scanning operation the last Q output of the CU or CD element before the scan is used. It is therefore recommended that, if the same counter is used in the program as an up and down counter, the Q or  $\bar{Q}$  outputs of the CU and CD elements be routed to flags and that the flags be then scanned.

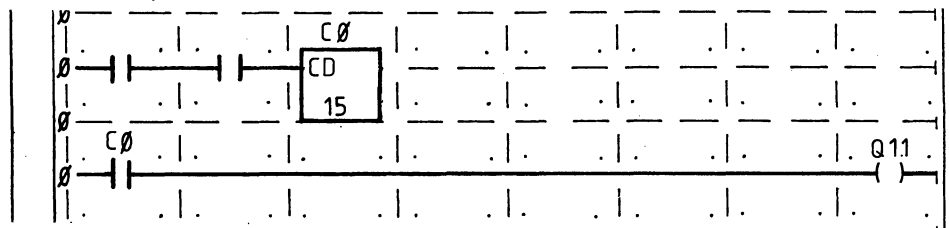


Fig. 34 Scanning a counter

Recommended:

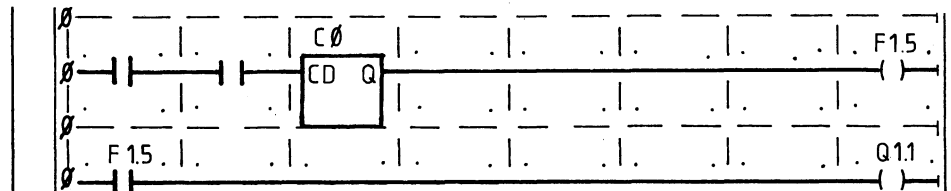


Fig. 35 Scanning a counter via a flag

Output Q1.1. becomes "1" when counter C0 has reached the lower limit.

**Note:**

- The current count is retained when changing over from RUN to STOP.
- When changing from STOP to RUN, the current count and output Q of the counter are set to zero. Only when a rung with START COUNTER is reached is the initial count of the counter entered.

Example

Three-to-one frequency scaler.  
After enabling via I0.2, the lamp Q0.6 lights up at every third pulse at I0.1

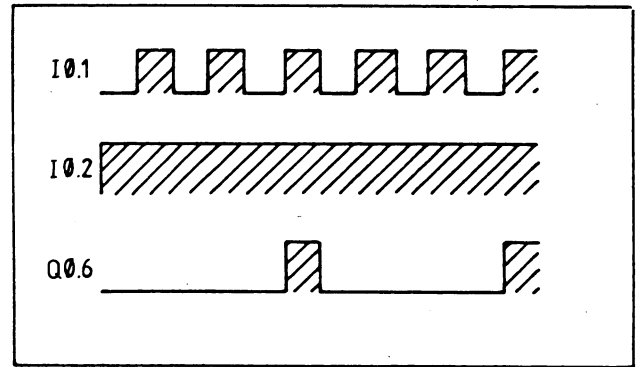


Fig. 36 Timing diagram of the frequency scaler

The scaling ratio is specified by the initial value in the START COUNTER element.

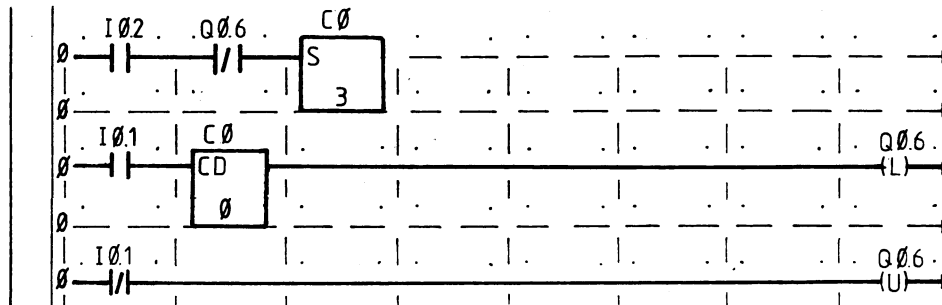


Fig. 37 Frequency scaler in LAD representation

## 2. Program generation with the 101R PC

### 2.1 Program input and correction

#### Preparation

A program can only be entered in the 101R PC if there is no memory submodule plugged in.

When generating a new program, select the ERASE PROGRAM (general reset) programmer function.

This causes

- the internal program memory of the 101R PC to be deleted,
- the process image of the inputs and outputs to be deleted,
- flags, timers and counters to be set to "0" and
- error messages to be deleted.

#### Input

- Select the INPUT/DISPLAY programmer function
- Enter program block number
- Enter program

#### Correction

The following can be deleted:

- The entire program (in the PC)
- Single PBs (in the PC and programmer)
- Single rungs (in the programmer)
- Single program elements (in the programmer)

The following can be inserted:

- Program elements (in the programmer)
- Rungs (in the programmer)

The following can be overwritten:

- Program elements (in the programmer)
- Rungs (in the programmer)
- Program blocks (in the PC)

The following can be assigned new numbers:

- Program blocks (in the PC)
- Program blocks (in the programmer)

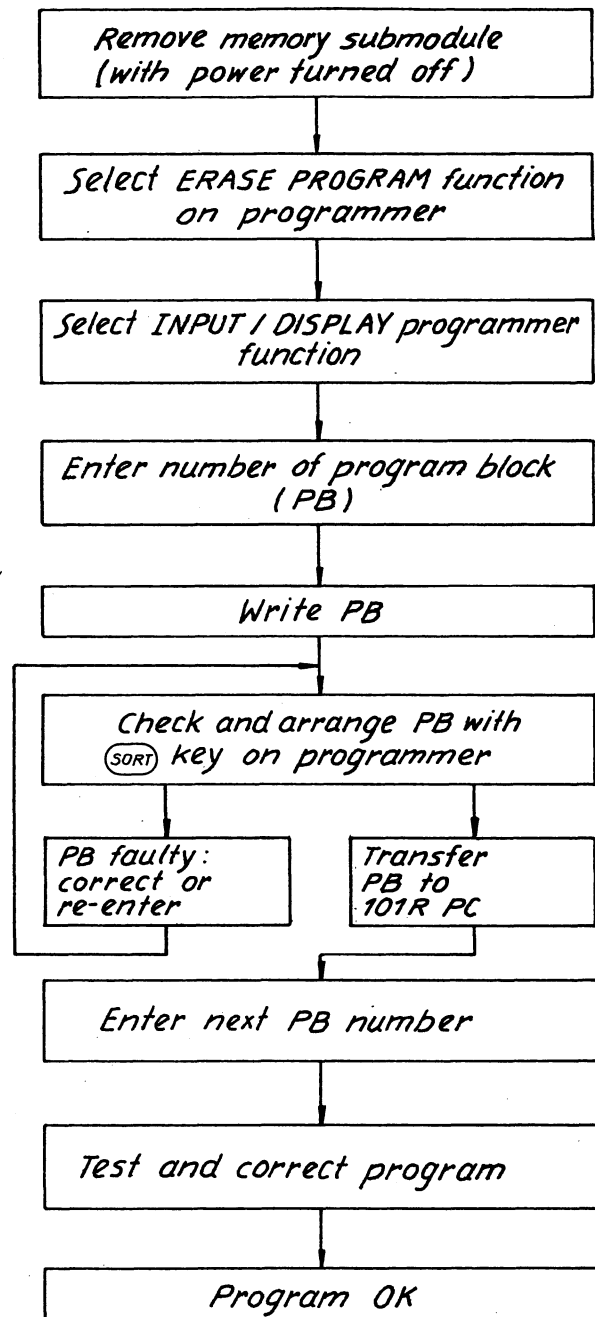


Fig. 38 Flowchart for program generation on the 101R PC



## 2.2 Example of program generation

The task definition of the system is the first item required when writing the program. This task definition includes a process schematic showing the object to be controlled with reference to the technological relationships in the process (points of installation of sensors and actuators, material and material flows, directions of motion, etc.).

Taking the example in Fig. 39 as a basis, the task definition is as follows:

The bulk material is to be transferred from a container via a conveyor belt and loaded into a waggon. The control sequence is enabled with pushbutton S1 (indicator lamp H1 lights up) and disabled with pushbutton S2.

When the control system is enabled, motor contactor K1 switches on the conveyor belt if a waggon is situated in the filling position (limit switch S3). The conveyor is switched off again if the waggon has left the filling position and the next waggon to be filled has not reached the filling position within 20 seconds. Slide valve Y1 is opened if the conveyor motor is switched on and an empty waggon is ready for filling. The slide valve is closed again when the weight set on scale B1 is reached. The commands for opening and closing may only be active until the slide valve has reached the new position. In order for the bulk materials still on the conveyor to be transported to the waggon, latching pawl Y2 is only opened 10 seconds after the "Full" message. The latching pawl is immediately closed again when the filled waggon has left the filling position, i.e. a contact of limit switch S3 has opened again.

When the next waggon reaches the filling position, the described operation is repeated until the control sequence is disabled with pushbutton S2. The next step is to draft the general structure of the task definition for the programmable controller, i.e.

- . Arrangement according to a monitoring scheme (signal statuses of sensors)
- . Modes (idle condition, etc.)
- . Machine functions (stop motor, etc.)

Process schematic

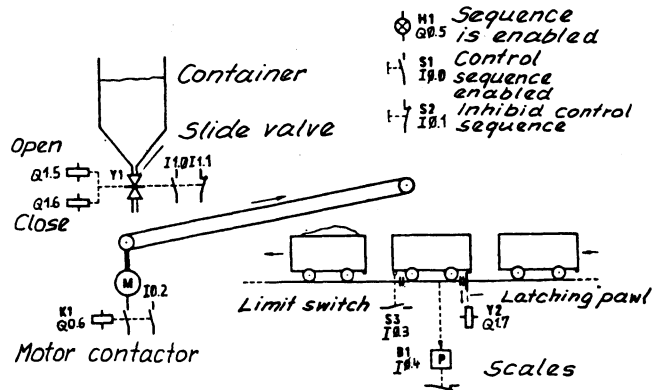


Fig. 39 Filling waggons from a conveyor belt

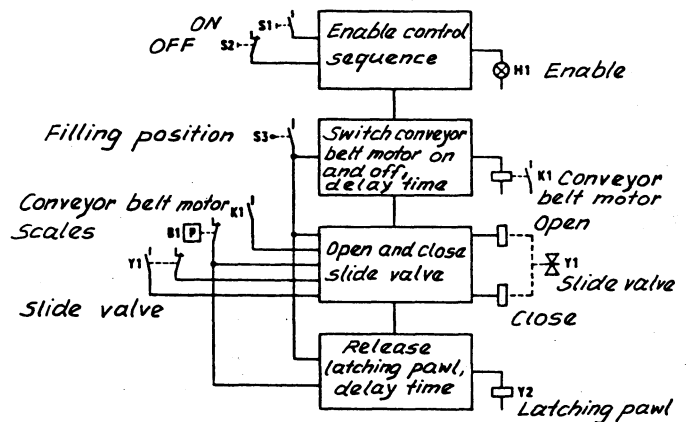


Fig. 40 Structural diagram for waggon filling system

Based on the structural diagram and the process schematic, the assignment list can now be compiled; sensors and actuators are assigned to the terminals of the PC, as is already partly the case in Fig. 39.

## Assignment of inputs and outputs and timers

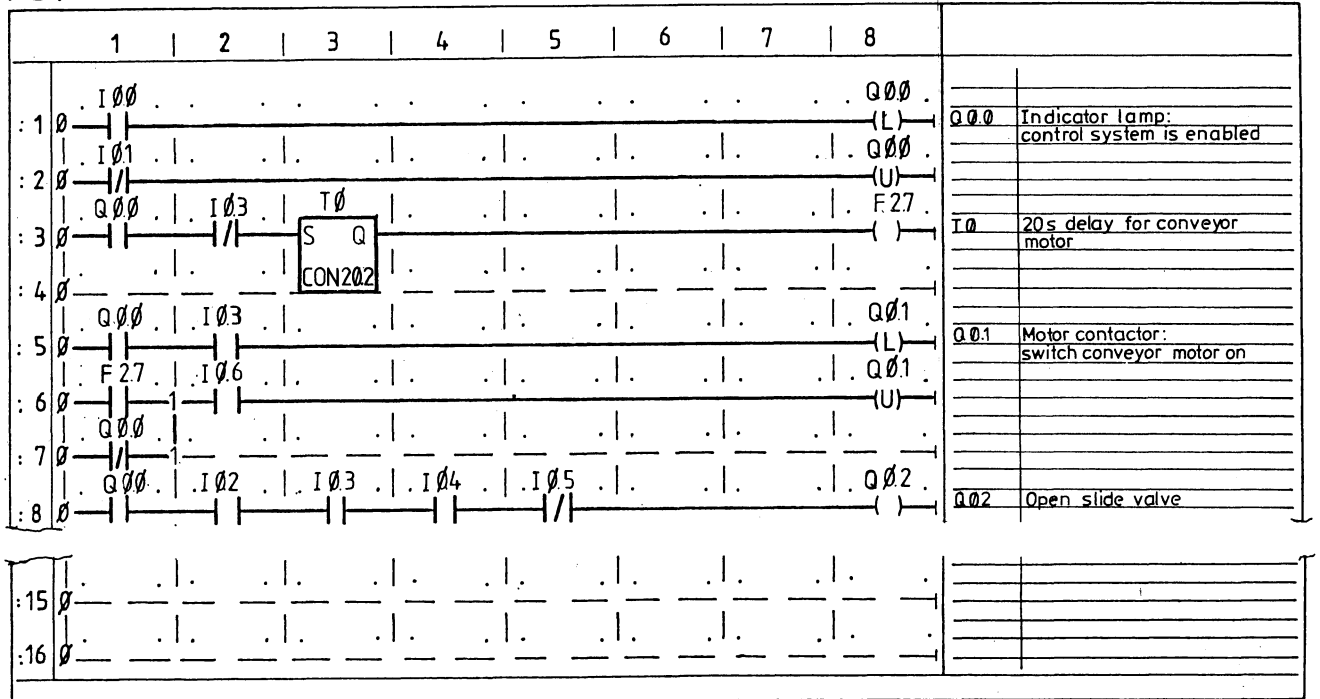
Operand	Device identifier	Functional description
Inputs:		
I 0.0	S 1	ON pushbutton, enable control sequence, idle condition is "0"
I 0.1	S 2	OFF pushbutton, disable control sequence, idle condition is "1"
I 0.2	K 1	Motor contactor, acknowledgement, conveyor motor is ON, idle condition is "1"
I 0.3	S 3	Limit switch, waggon in filling position, idle condition is "0"
I 0.4	B 1	Scales, waggon is full, idle condition is "1"
I 0.5	Y 1	Slide valve, acknowledgement, valve is open, idle condition is "0"
I 0.6	Y 1	Slide valve, acknowledgement valve is closed, idle condition is "1"
Outputs:		
Q 0.0	H 1	Indicator lamp, control system is enabled, idle condition is "0"
Q 1.0	K 1	Motor contactor, switch on conveyor motor 1, idle condition is "0"
Q 0.2	Y 1	"Open" slide valve, idle condition is "0"
Q 0.3	Y 1	"Close" slide valve, idle condition is "0"
Q 0.4	Y 2	Release latching pawl, idle condition is "0"
Timers:		
T 0	-	20 sec. delay for conveyor motor
T 1	-	10 sec. delay for latching pawl

The number of inputs/outputs can be read off the assignment list. The PC can now be installed and the inputs and outputs can be wired.

The program can now be written while the mechanical and electrical part is being installed. The program for the example of the "Waggon filling system" is shown in Fig. 41.

When the program has been written, it is loaded into the PC and tested.

### PB0



### PB1

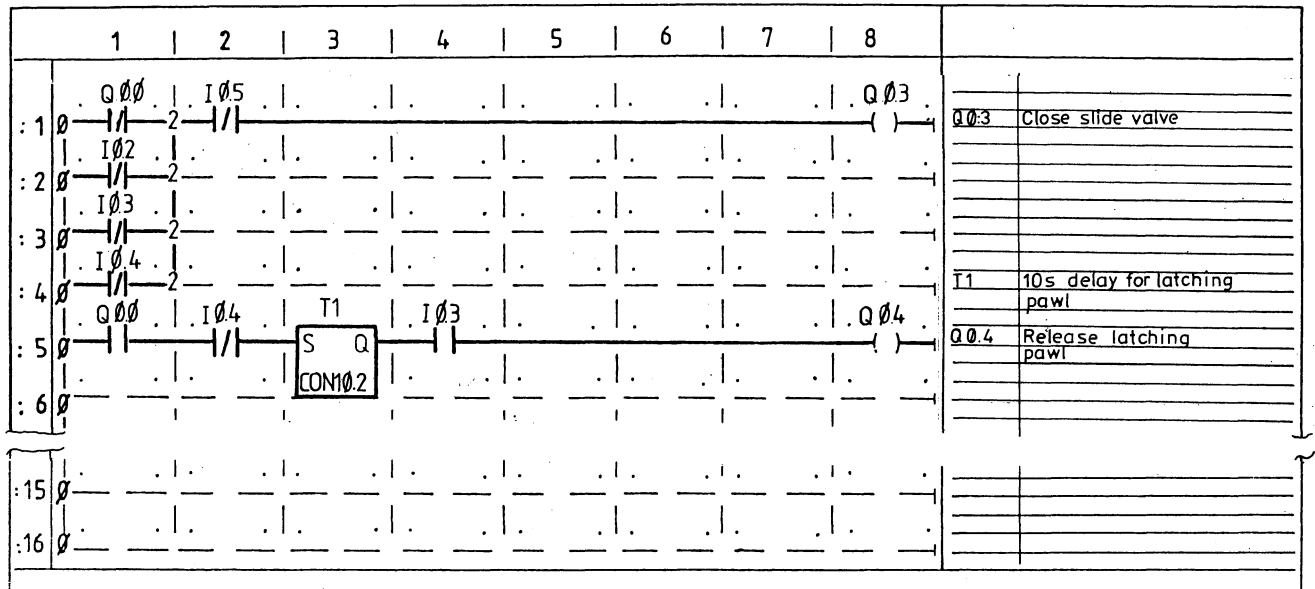


Fig. 41 Program for the waggon filling system

# 3. Program test

## 3.1 Search function

This function searches for the following elements within a program:

- Operands, e.g. I0.0, C5, T4
- Program elements, e.g. ]E I1.0, -(L) Q0.6, S C0

It can be executed in the programmer functions

- INPUT/DISPLAY
- PROG. TEST

### Search function with a program block (PB)

The rungs containing the element searched for are displayed on the programmer. The element searched for is marked with the blinking cursor.

### Search function in the entire program

There is only one program block in the programmer at one time. When this has been checked and the program element not found, the next PB is automatically fetched from the PC and the search continues. This is continued until the program element has been found or there are no PBs left in the PC.

## 3.2 Signal status display

The following can be displayed on the programmer for the program test function:

- Signal states of operands  
e.g. I0.3, F1.7, Q1.1
- Current values and signal states of timers and counters

Two checkpoints can be selected:

- At the end of a PB using the PROGRAM TEST programmer function. The checkpoint of the program block currently in the programmer is the one processed.
- At the end of the program (program checkpoint) by means of the SIGNAL STATUS programmer function

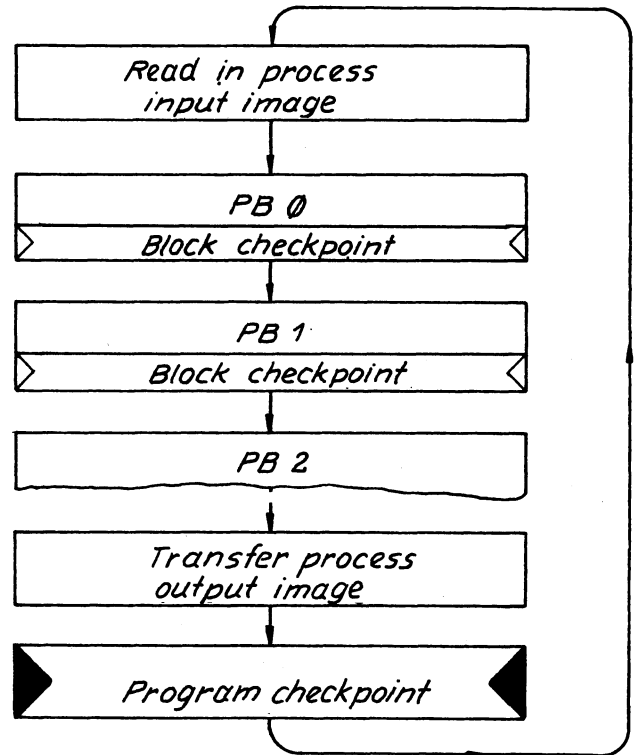


Fig. 42 Processing of program by the 101R PC

## 3.3 Forcing

Forcing is the once-only assignment of a signal state to an operand (e.g. I0.3, F1.2, T2).

This default is only valid until the operand is assigned a new signal status by program processing

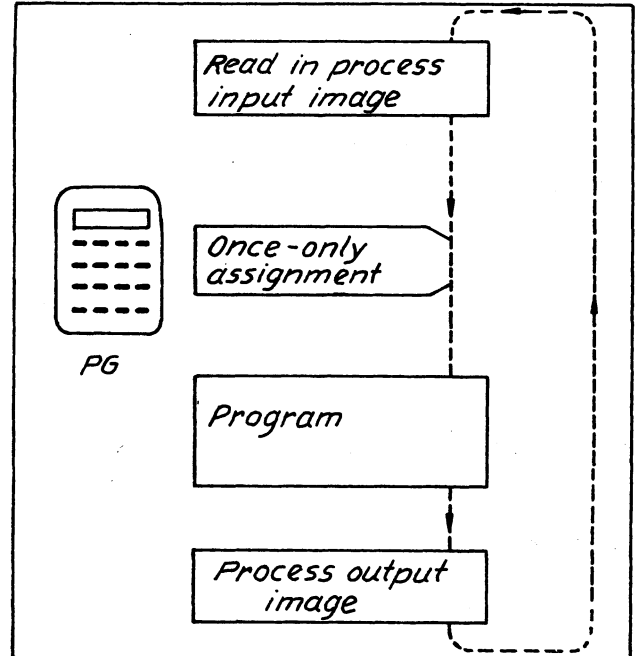


Fig. 43 Once-only assignment of the signal state when forcing

## 4. Storing the program

### 4.1 Storing the program on an EEPROM submodule

A valid program can be transferred from the internal memory of the 101R PC to a plug-in memory submodule for long-term storage.

An EEPROM submodule is plugged into the PC in the off-state. The contents of the 101R program memory (user program, AUTO RESTART and FLAGS RETENTIVE functions) are copied onto the submodule by means of the STORE PROGRAM programmer function.

N.B.

- The memory submodule may only be inserted and removed with the power off!
- Programs already on the EEPROM submodule are overwritten.

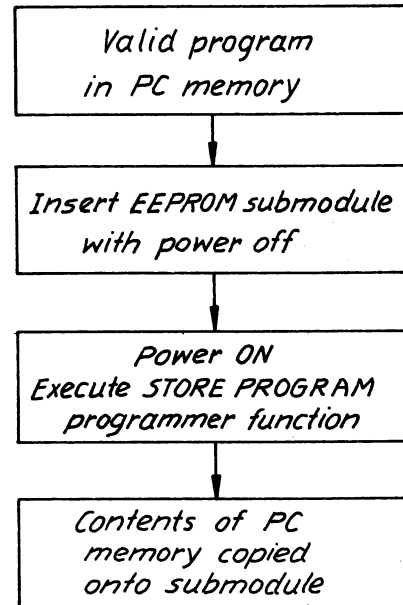


Fig. 44 Sequence of program storage on EEPROM

Note:

It is not possible to store a program on an EPROM submodule on the 101R PC. EPROMs can be programmed using the 655R programmer or the 105R PC. It is, however, possible to copy the contents of the EPROM submodule into the internal program memory of the 101R PC.

## 4.2 Duplicating EEPROM submodules

After erasing the memory of the 101R PC (general reset), a memory submodule with a valid program is inserted.

When the power is switched on, the contents of the submodule are copied into the internal program memory of the 101R PC.

To prepare the STORE PROGRAM function, an entry must be made in the program in the internal memory (see Operating Instructions, Section 3.4 "Using the memory submodules") e.g. read out and re-enter AUTO RESTART = x. See Section 4.1 for further action.

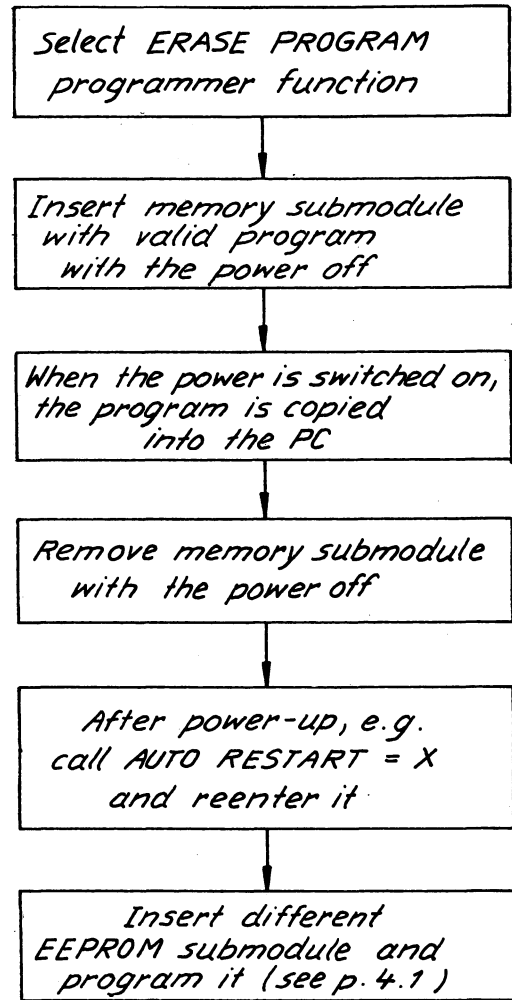
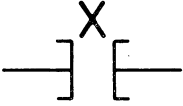


Fig. 45 Duplication of memory submodules

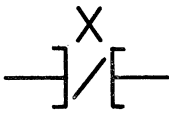
# 5. Operation set

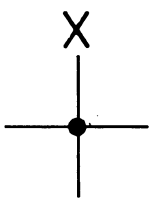
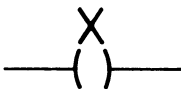
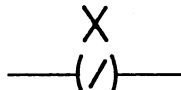
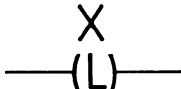
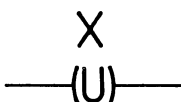
## 5.1 Binary operations

Symbol	Operand	Description
	X=I0.0 to I2.3	Scanning an input for "1".
	X=Q0.0 to Q1.3	Scanning an output for "1".
	X=F0.0 to F3.7 1)	Scanning a flag (internal relay) for "1".
	X=T0 to T7	Scanning a timer for "1".  (The timer has a "1" at the Q output if there is a "1" at the start input of the timer and the time has elapsed.)
	X=C0 to C7	Scanning a counter for "1".  (The up counter has a "1" at the Q output if the specified limit is reached or exceeded. The down counter has a "1" at the Q output if the specified limit is reached or if the count drops below it.)

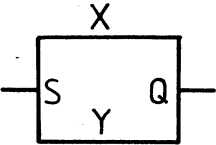
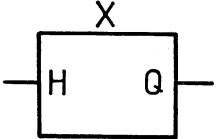
1) Flags 0.0 ... 1.7 can be set as retentive

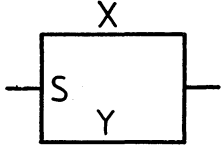
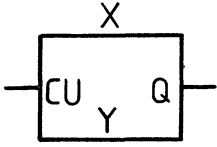
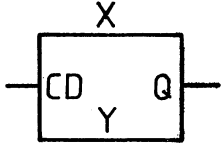


Symbol	Operand	Description
	X=I0.0 to I2.3	Scanning an input for "0".
	X=Q0.0 to Q1.3	Scanning an output for "0".
	X=F0.0 to FM3.7	Scanning a flag for "0".
	X=T0 to T7	Scanning a timer for "0".  (The timer has a "0" at output Q if there is a "0" at the start input of the timer or a "1" at the start input of the timer and the time has not yet elapsed.)
	X=C0 to C7	Scanning a counter for "0".  (The up counter has a "0" at the Q output if the specified limit has been reached or the count has dropped below it and the down counter has a "0" and the Q output if the specified limit has been reached or exceeded.)

Symbol	Operand	Description
	$X=0$ to 14 whereby: $10 \hat{=} A$ $11 \hat{=} B$ $12 \hat{=} C$ $13 \hat{=} D$ $14 \hat{=} E$	The nodes are used for combining rungs. Each rung begins with a node. The connection to the left-hand power rail is always made with node 0.
	$X=Q0.0$ to $Q1.3$	Set output to "1"
	$X=F0.0$ to $F3.7$	Set flag to "1"
	$X=Q0.0$ to $Q1.3$	Set output to "0"
	$X=F0.0$ to $F3.7$	Set flag to "0"
	$X=Q0.0$ to $Q1.3$	Latch output to "1"
	$X=F0.0$ to $F3.7$	Latch flag to "1"
	$X=Q0.0$ to $Q1.3$	Unlatch output
	$X=F0.0$ to $F3.7$	Unlatch flag

## 5.2 Complex operations

Symbol	Operand	Description
<p>Start timer</p>  <p>Y = time <math>\hat{=}</math> time value time base</p> <p>Time value: 1 to 999 Time base: 0 <math>\hat{=}</math> 10 ms 1 <math>\hat{=}</math> 100 ms 2 <math>\hat{=}</math> 1 s 3 <math>\hat{=}</math> 1 min</p>	<p>X=T0 to T7</p> <p>Y=CON 1.0 to CON 999.3</p>	<p>A "1" signal at the S input starts the timer. Output Q is "1" if there is a "1" at the S input and the specified time has elapsed.</p> <p><math>\bar{Q}</math> can also be used as output with the inverted signal of Q.</p> <p>The constant Y, obtained by multiplying a time value with a time base, gives the time (e.g. 8.1 = 800 ms).</p>
<p>Hold timer</p> 	<p>X=T0 to T7</p>	<p>A "1" at the H and S inputs of the same timer stops this timer. Output Q is "1" if there is a "1" at the S input of the same timer and the specified time has elapsed.</p> <p><math>\bar{Q}</math> can also be used as output with the inverted signal of Q.</p>

Symbol	Operand	Description
<p data-bbox="192 309 362 338">Set counter</p> 	<p data-bbox="584 309 738 338"><math>X=C0</math> to <math>C7</math></p> <p data-bbox="584 488 754 544"><math>Y=CON</math> 0 to CON 32767</p>	<p data-bbox="887 309 1303 398">A "1" at the S input causes constant Y to be loaded and enabled.</p> <p data-bbox="887 427 1404 483">Constant Y can have any value between 0 and 32767.</p>
<p data-bbox="192 577 318 607">Count up</p> 	<p data-bbox="584 577 738 607"><math>X=C0</math> to <math>C7</math></p> <p data-bbox="584 696 754 752"><math>Y=CON</math> 0 to CON 32767</p>	<p data-bbox="887 577 1420 667">With every change from "0" to "1" at the CU input, the previous count is incremented by one.</p> <p data-bbox="887 696 1420 819">Constant Y specifies the upper limit of the counter. If this limit is reached or exceeded, the Q output changes to "1".</p> <p data-bbox="887 848 1397 904"><math>\bar{Q}</math> can also be used as output with the inverted signal of Q.</p>
<p data-bbox="192 943 346 972">Count down</p> 	<p data-bbox="584 943 738 972"><math>X=C0</math> to <math>C7</math></p> <p data-bbox="584 1061 754 1120"><math>Y=CON</math> 0 to CON 32767</p>	<p data-bbox="887 943 1420 1032">With every change from "0" to "1" at the CD input, the previous count is decremented by 1.</p> <p data-bbox="887 1061 1404 1218">Constant Y specifies the lower limit of the counter. If this limit is reached or if the count drops below it, the Q output changes to "1".</p> <p data-bbox="887 1218 1393 1274"><math>\bar{Q}</math> can also be used as output with the inverted signal of Q.</p>