

# SIEMENS

## SIMATIC S5 - 105 R

### Programmable Controller

Programming Instructions

Order No. GWA 4NEB 810 0221-02

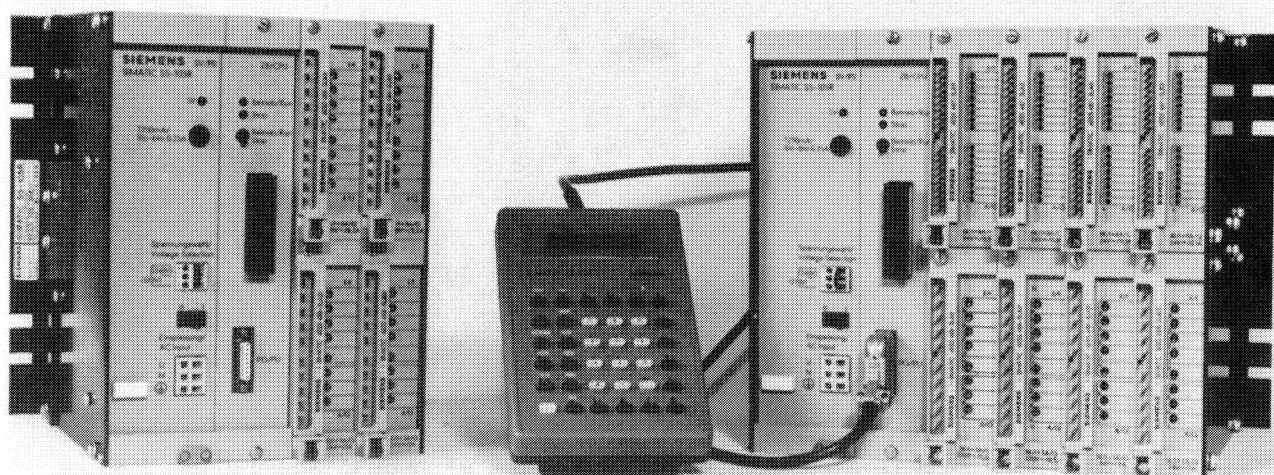


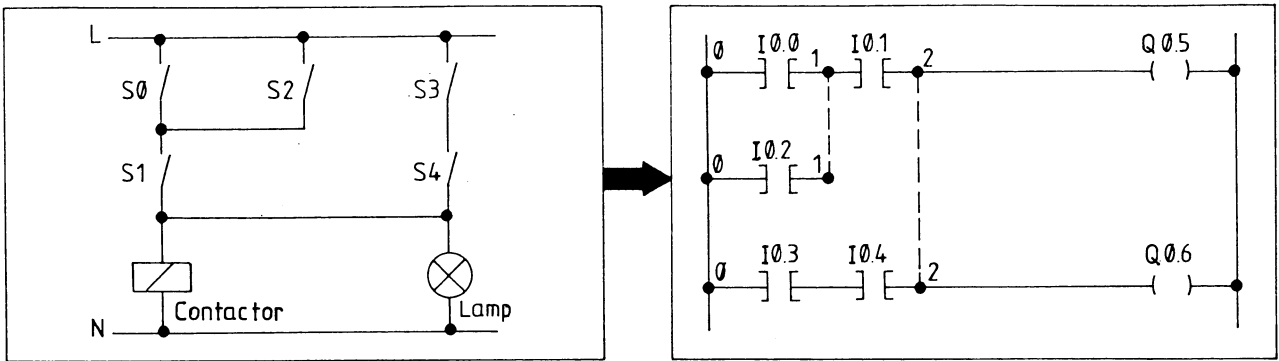
Fig. 1 S5-105R-A/B with 605R programmer

Contents:	Page	Contents:	Page
1. Programming	1.1	3. Program test	3.1
1.1 Formal rules for structuring a program	1.2	3.1 Search function	3.1
1.2 Binary program elements	1.5	3.2 Signal status display	3.1
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	3.4 Permanent forcing	3.2
1.2.3 Latching and unlatching	1.7	3.5 Single scan in HOLD mode	3.3
1.3 The complex program elements	1.8	4. Storing the program	4.1
1.3.1 Timers	1.8	4.1 Storing the program on an EEPROM submodule	4.1
1.3.2 Counters	1.13	4.2 Storing the program on an EPROM submodule	4.1
1.3.3 Impulse relay (transition-sensitive pulse)	1.16	4.3 Duplication of programs	4.2
1.3.4 Jumps	1.19	5. Operation set	5.1
1.3.5 Sequence control systems/sequence cascades (drum sequencers)	1.20	5.1 Binary operations	5.1
2. Program generation with the 105R 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. Programming

The 105R programmable controller uses ladder diagrams (LAD) for programming.

The LAD is a graphic method of representing the automation problem using circuit diagram symbols (American standard).



Circuit diagram

Ladder diagram

The ladder diagram is a symbolic representation of a circuit diagram. Points with the same potential are described by means of nodes. The ladder diagram is entered directly into the programmer.

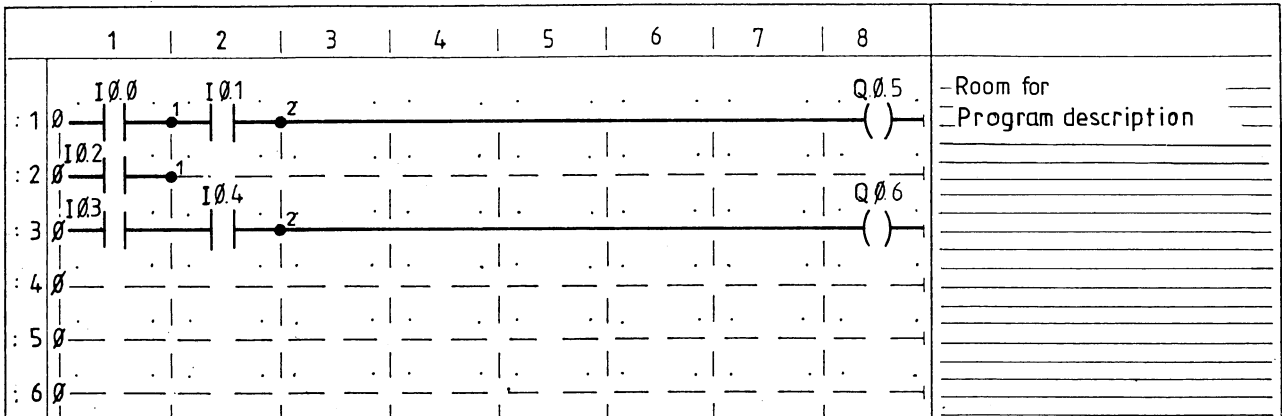


Fig. 2 Representation of the LAD on a programming form for keying directly into the programmer

# 1.1 Formal rules for structuring a program

The program of the 105R PC may consist of a maximum of 64 program blocks (PBs). These are assigned numbers from 0 to 63 and can be entered in any order. They are processed in the 105R PC in the order in which they are stored in the memory.

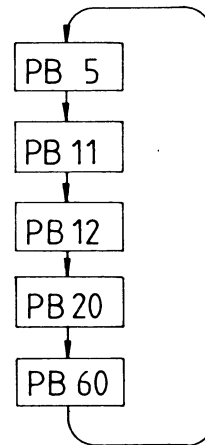


Fig. 3 Processing of the PBs

A program block may consist of up to 16 rungs. Each PB can be assigned 15 different nodes.

- A rung may contain 7 contact elements (scans)
- 8 nodes \* (vertical interconnections)
- 1 output element (assignment, always in column 8)

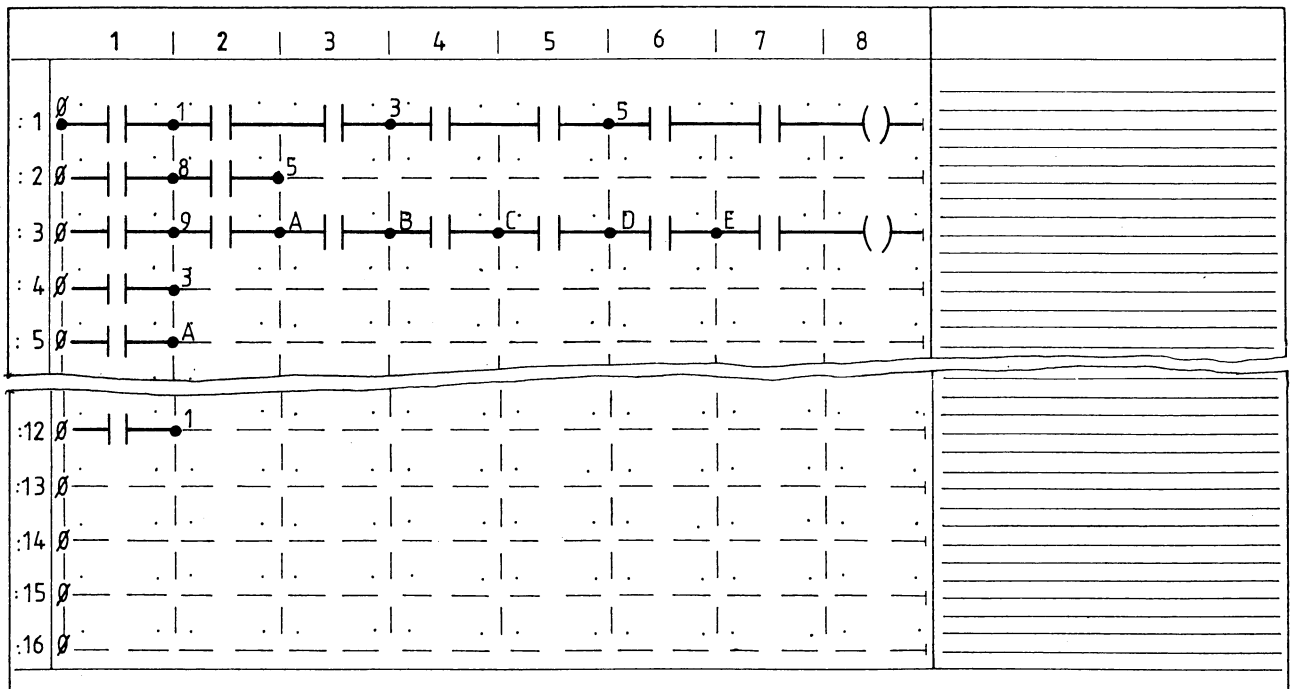
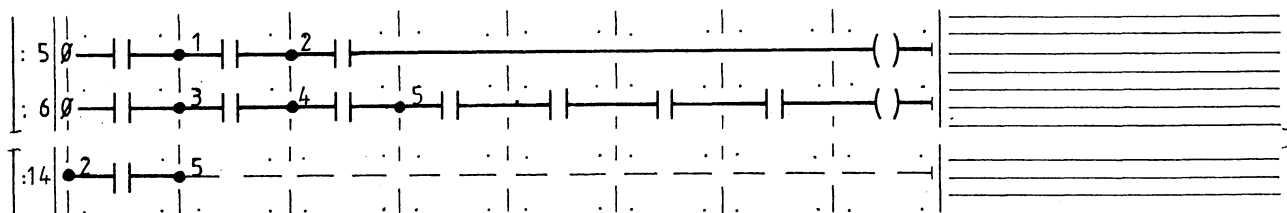


Fig. 4 Structural framework of a program block

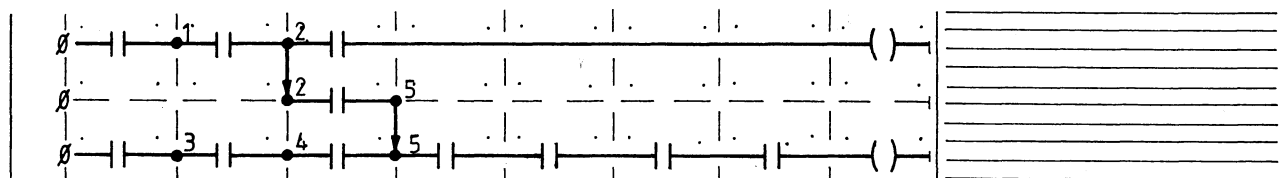
\* Input on programmer: Node 0 ... 14, display on programmer: 0 ... E (hexadecimal representation)

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 contact elements). This must be taken particularly into account when adding rungs to PBs.

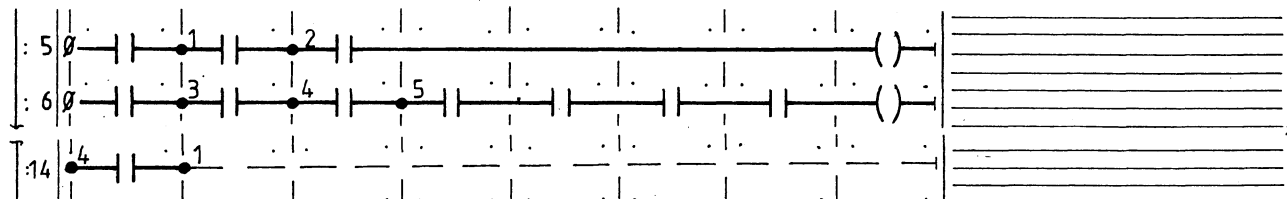
Example: Rung 14 can be added to the given program section, rungs 5 and 6.



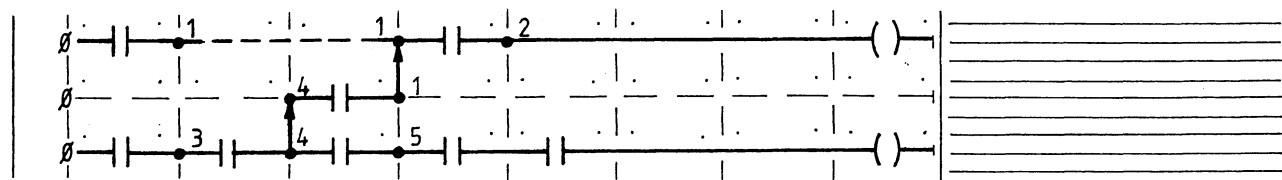
By rearranging the rungs, the correct signal flow from left to right becomes visible.



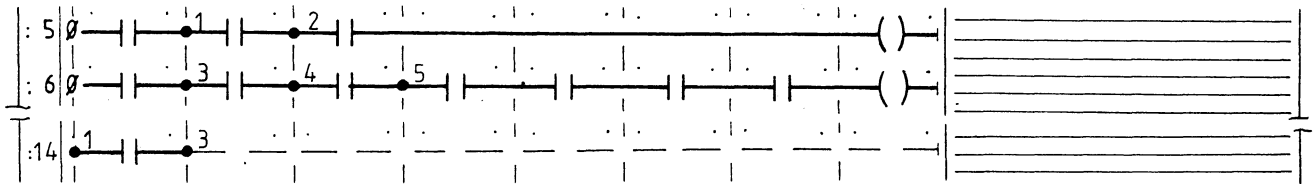
The apparently illegal insertion of rung 14 is also possible,



as the following rearrangement is possible by shifting rung 5 from node 1:

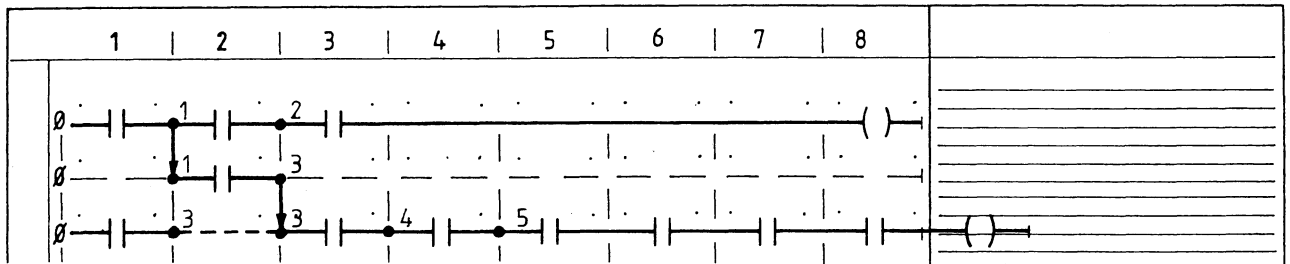


It is not possible, however, to add the following node 14:



After pressing the **SORT** key on the programmer, the program entered is checked. Error message E1 appears, indicating that there are more than 8 program elements in one rung.

When the block is checked internally, the programmer attempts to maintain the prescribed signal flow from left to right. Rung 6 is therefore shifted to the right from node 3, as in the previous example, to make possible the insertion of rung 14:



This results in a rung overflow as can be seen from the programming form. This error is recognised 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!

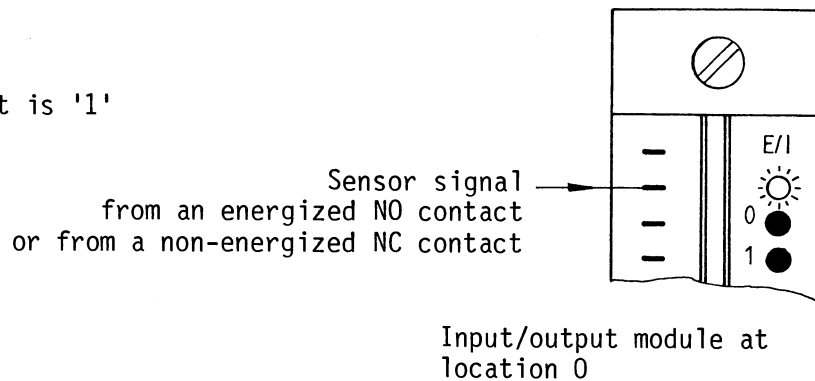
## 1.2 Binary program elements

### 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 their signal state 0 or 1. The switching state of the program element is obtained by scanning the signal state. This makes it possible for the opening or closing function to be simulated by the program.

#### Example

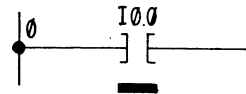
Input LED bright,  
Signal state of input is '1'  
i.e. I0.0 = '1'



State of the program element if

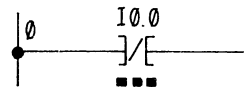
- scanned for '1' signal:

Energized: signal flow  
is possible



- scanned for '0' signal:

Not energized: no signal  
flow possible

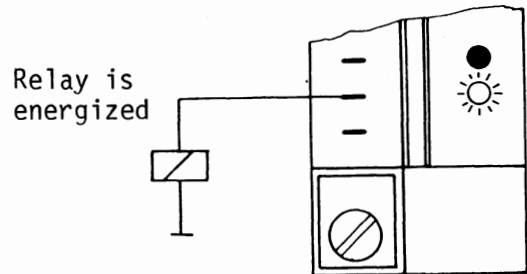


## 1.2.2 Outputs (coils) and flags (internal relay equivalents)

Signal states can be assigned to outputs and flags by the program. They can be scanned like contacts, e.g.  $\neg E$  Q0.6,  $\neg E$  F3.5.  
The signal states of the outputs are transferred to the output modules

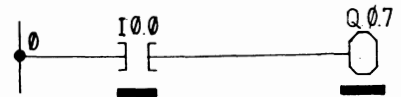
### Example

Output LED bright  
Signal state of output is '1'  
i.e. Q0.0 = '1'

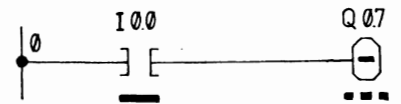


Input/output module  
at location 0

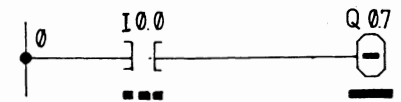
If there is signal flow via I0.0,  
output Q0.7 is '1'



If there is signal flow via I0.0,  
the negated output Q0.7 is '0'



If there is no signal flow via I0.0,  
the negated output Q0.7 is '1'



## 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.

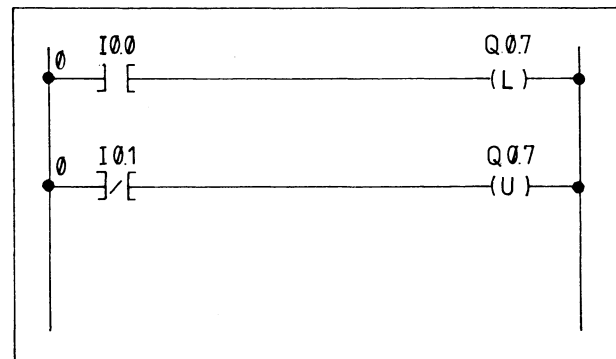
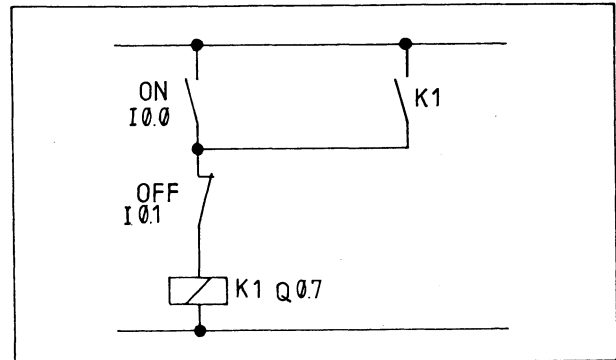
### Example

Contactor with latching circuit

When sensor I0.0 is actuated, Q0.7 is set to '1' and latched.

It can be unlatched by actuating sensor I0.1.

Contact K1 is not required due to the latching feature.



Scanning non-existent \* inputs always results in '0' signal.

Non-existent \* outputs can be used in the program as (non-retentive) flags.

\*Non-existent means

- either the relevant I/O module locations are not occupied
- or the configuration with 5 input/3 output modules does not make use of the full address complement of 64 outputs (see also section 5.1).



## 1.3 The complex program elements

The operations of all program elements are listed in Section 5.

Timers, counters, impulse relays, jump functions and sequence cascades (drum sequencers) 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

The operations of all program elements are listed in Section 5.

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: The timer starts when there is a signal change from '0' to '1' at the START input.
- Holding the timer: When the signal changes from '0' to '1' at the HOLD input, the timer is stopped until a zero signal appears at the HOLD input.
- Resetting the timer: If there is a '0' signal at the START input, the timer is reset to the initial value.
- Scanning the timer: The actual state of the timer can be seen via outputs Q or  $\bar{Q}$  or scanned via a TX contact element.
- Entering the time: The desired time can be entered as  
- a constant (CON)  
- a register time (RT)
- The constant is entered when programming.

<p>The register time can be entered after program generation or it can also be modified in RUN mode. If no value is entered, the maximum time, i.e. 999 minutes, is stored in the register after a general reset of the 105R PC.</p>
--

Entering the time code:

The desired time is entered in the form A.B.

The letter B stands for 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 ms  
3 for time base 1 min

Example

If 10.2 is entered, the time set is  $10 \times 1 \text{ s}$   
= 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.

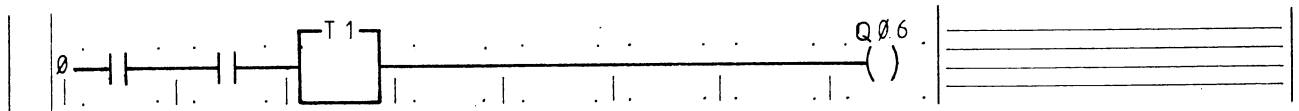
Example:

Run time 8 seconds

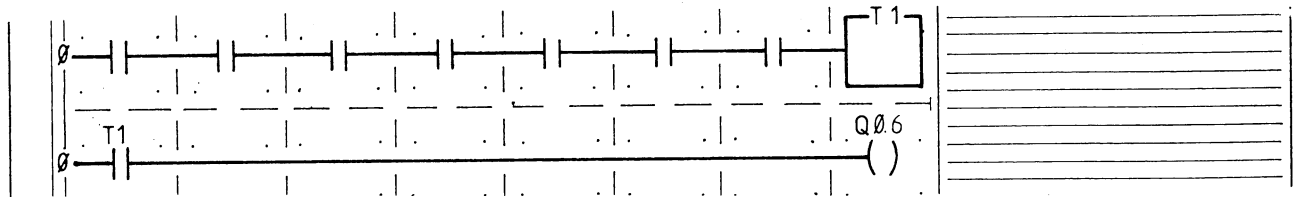
Representation	8.2	max. error	1	seconds
	80.1	max. error	0.1	seconds
	800.0	max. error	0.01	seconds

## Formal rules for using timers in the program

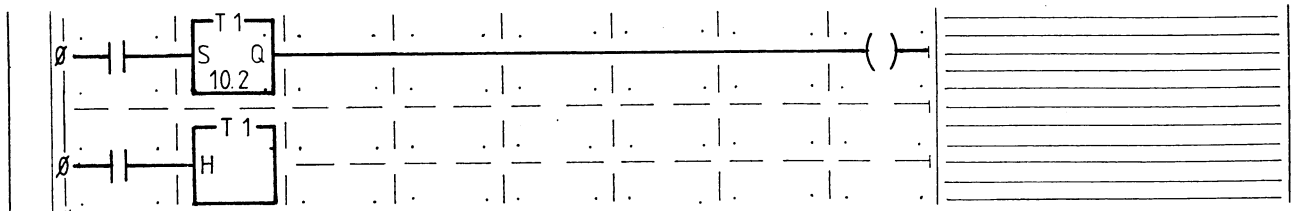
Timers can be incorporated into a rung like contacts.



Scanning or continuation of the output in the same rung is not mandatory; it is therefore permissible to scan a timer as a contact:

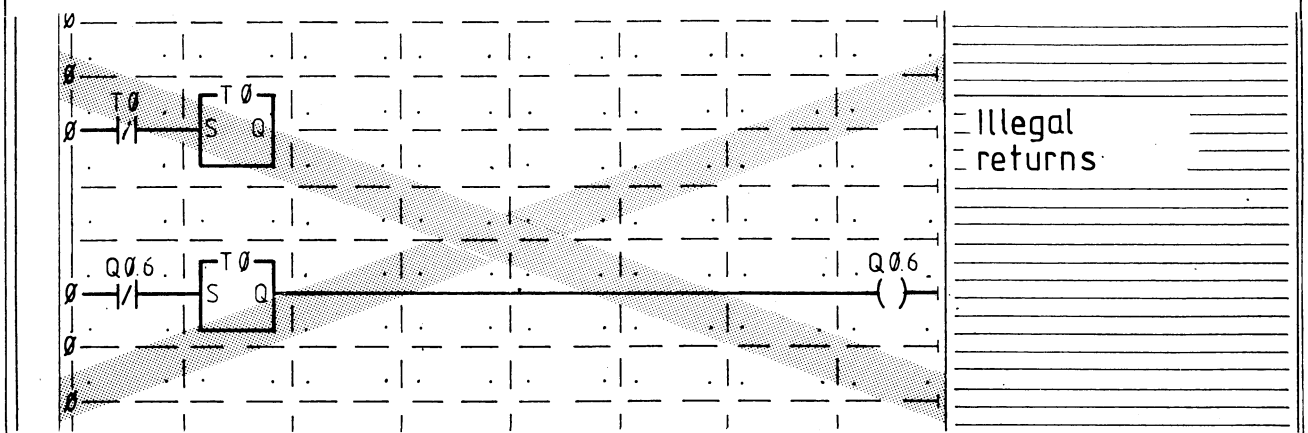


A timer can be addressed via the START (S) and HOLD (H) inputs:



The timer is entered in the "START TIMER 1" function box.  
Output Q or  $\bar{Q}$  of the timer are available for both the START TIMER and HOLD TIMER function boxes.

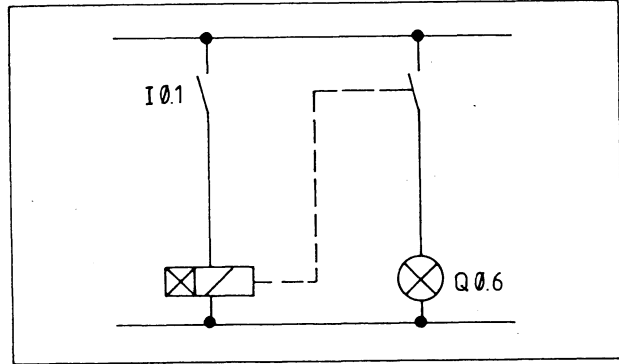
- The output of a timer must not be brought back to the START input.
- When there is a change from RUN to STOP and HOLD, the times are stopped.
- When changing over from STOP to HOLD or RUN the stopped timer is reset.
- When changing from HOLD to RUN (only possible with programmer), the timer continues to run from the point at which it was stopped.



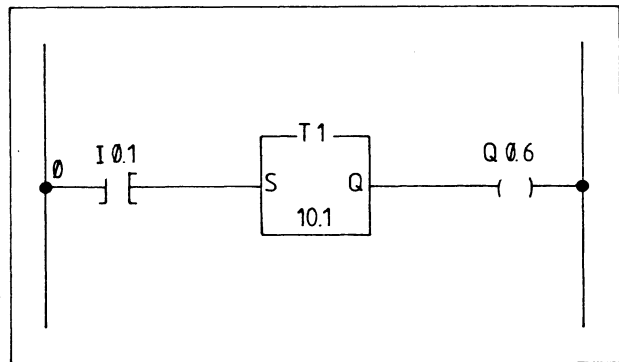
Typical examples

1. "ON" delay, not latched:

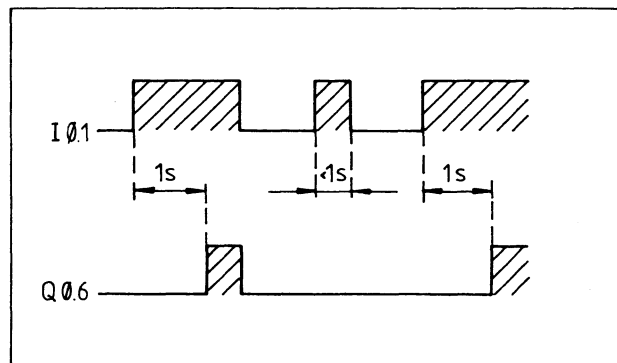
Lamp Q.6 only lights up when contact I.1 has been closed for longer than 1 second.



If there is a '1' at input I.1, timer T1 is started. After 1 second, T1 has elapsed and output Q becomes '1'. A '0' signal at I.1 resets the timer.



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

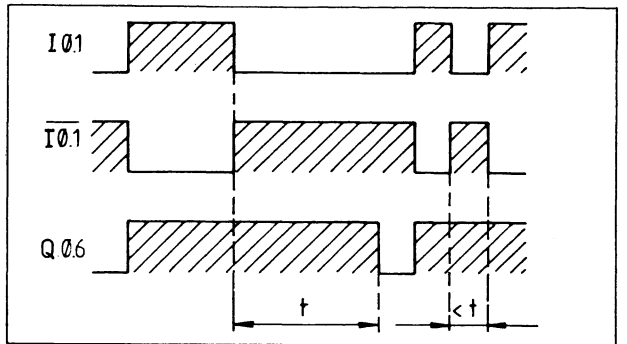
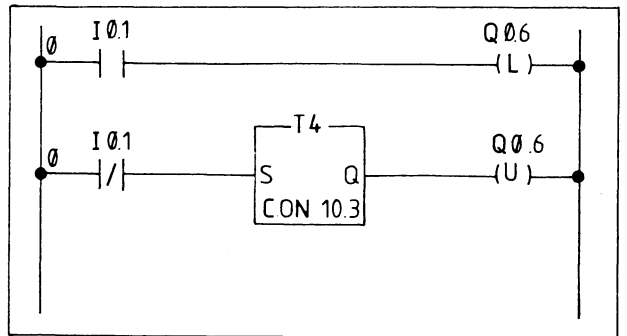
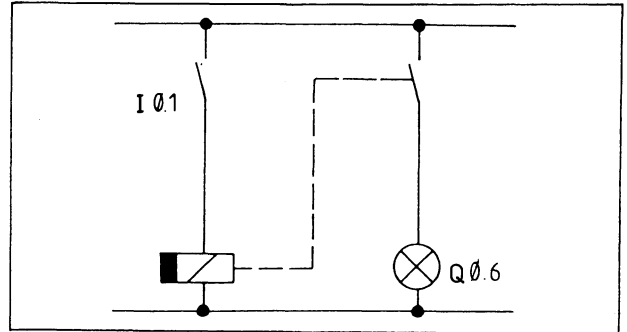


## 2. 'OFF' delay:

Lamp Q0.6 is bright for 10 minutes after contact I0.1 has opened.

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 signal state of input  $\overline{I0.1}$  changes output Q0.6 is latched to '1'. If the signal state of input I0.1 now 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  $\overline{I0.1}$ , output Q0.6 is set to '0'.

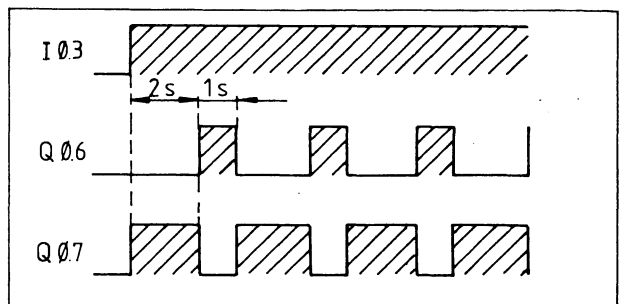
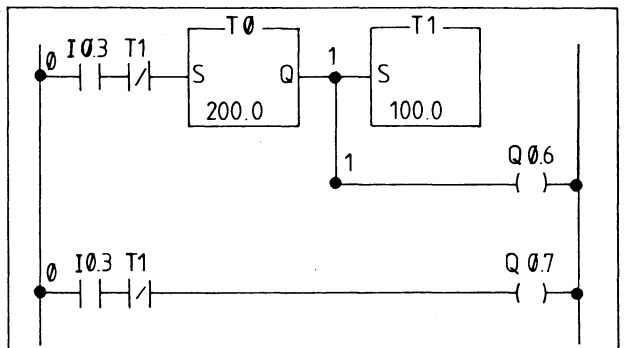
If the signal state of input  $\overline{I0.1}$  changes from '1' to '0' while the timer is still running, output Q0.6 does not change its signal status and remains '1'.



## 3. Clock pulse generator

After contact I0.3 is energized, outputs Q0.6 and Q0.7 are set alternately with selectable time constants.

When contact I0.3 has a '1' signal, T0 is started. After two seconds, T0 sets output Q0.6 and simultaneously starts T1. After one second, T0 is reset by T1. This resets Q0.6 and then T0 can start again.



## 1.3.2 Counters

The operations of all program elements are listed in Section 5.

A counter is driven via the SET, COUNT UP and COUNT DOWN inputs. Output Q of both elements changes from '0' to '1' when the relevant final count has been reached.

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

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

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

Scanning the counter: The current counter state can be ascertained via the outputs Q or  $\bar{Q}$  at the COUNT UP and COUNT DOWN counter elements.

Entering counts: The initial and final counts of the counter can be entered as

- a constant (CON)
- a register value for data (DR)

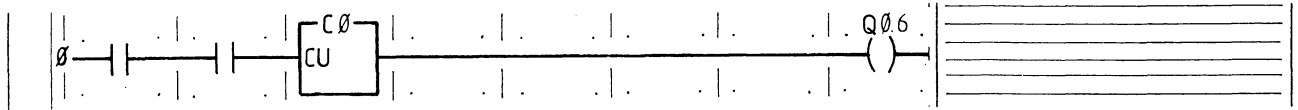
The constants are entered when generating the program.

The register values can be entered after program generation or can also be modified in the RUN mode. If no value has been entered, the highest data value, i.e. 32767, is stored in the register after a general reset of the 105R PC.

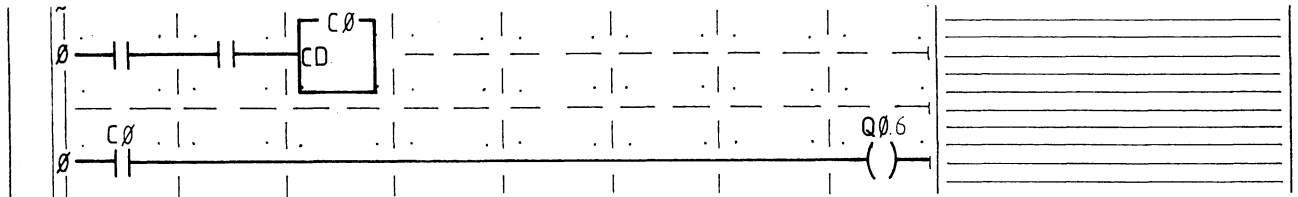
Counting range: 0 to 32767

## Formal rules for using counters in the program

Counters can be incorporated into a rung like contacts.

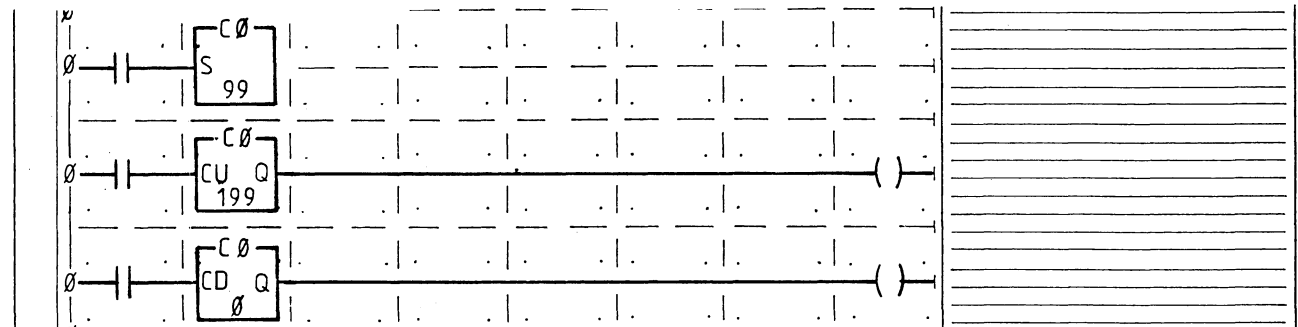


Scanning or continuation of the output in the same run is not mandatory; it is therefore also possible to scan the counter as a contact:



Q0.6 becomes '1', if counter C0 has reached the lower limit.

A counter can be addressed via the SET (S), COUNT UP (CU) and COUNT DOWN (CD) inputs:



The count values are entered in function boxes:  
 SET (S) Initial count (here: 99)  
 COUNT UP (CU) Upper limit (here: 199)  
 COUNT DOWN (CD) Lower limit (here: 0)

Outputs Q and  $\bar{Q}$  of the counter are available on the COUNT UP and COUNT DOWN function boxes.

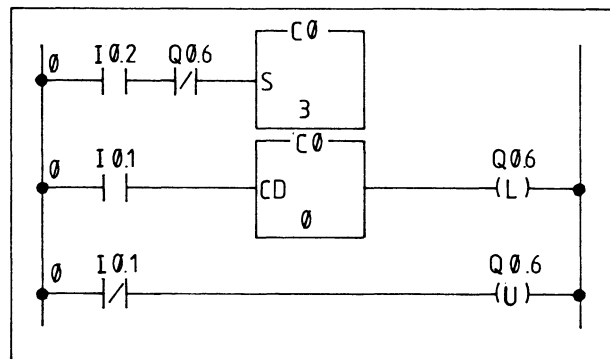
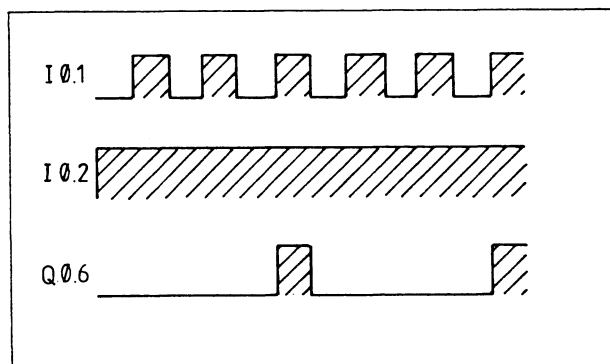
- The current count is retained when changing over from RUN to STOP or HOLD.
- When changing from STOP to HOLD or RUN, the current count and output Q of the counter are set to 0.
- Only when a rung with START COUNTER is reached is the initial count of the counter entered.
- When switching from HOLD to RUN (only possible with programmer), the current count is retained

Example:

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

The scaling ratio is specified by the initial value in the START counting element:

Binary scaler:       CON = 2  
 10:1 scaler:        CON = 10





### 1.3.3 Impulse relay (transition-sensitive pulse)

The operations of all program elements are listed in Section 5.

An impulse relay reacts to a change of signal from '0' to '1' at the START input by producing a pulse at the output.

Starting an impulse relay: When the signal changes from '0' to '1' at the START input, output Q is set to '1'.

Pulse duration: If during cyclic program processing a rung is reached which contains the complex impulse relay function, output Q is set from '1' to '0'. The signal state at the START input is of no consequence.

Scanning the impulse relay: The current state of the impulse relay can be ascertained via outputs Q or  $\bar{Q}$  or scanned via a contact element.

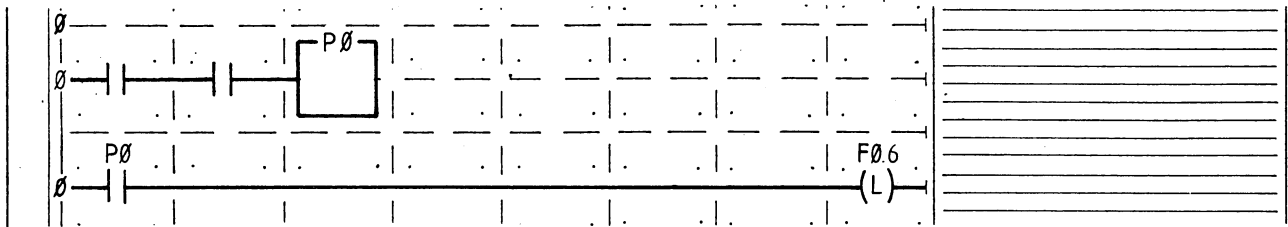
Resetting the impulse relay: When the signal changes from '1' to '0' at the START input, the impulse relay is prepared for a new pulse.

## Formal rules for using impulse relays in the program

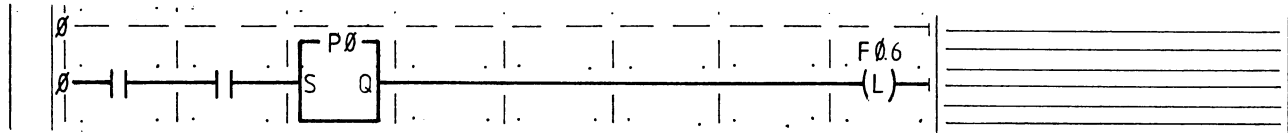
Impulse relays can be incorporated in a rung in the same way as contacts.



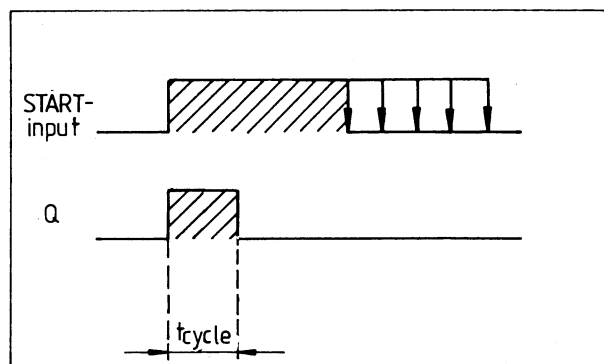
Scanning or continuation of the output in the same rung is not mandatory; it is therefore also possible to scan the impulse relay as a contact.



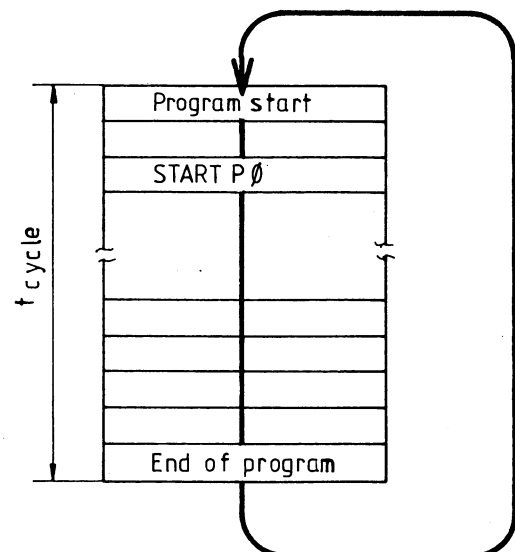
An impulse relay is only controlled via the START input.

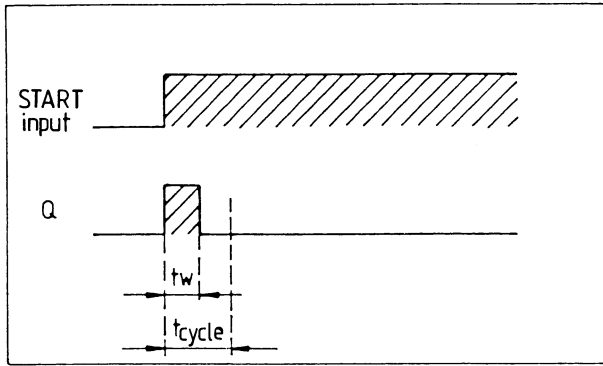


### Pulse duration



If the impulse relay is scanned once only in the program, the output pulse lasts for the length of one cycle.

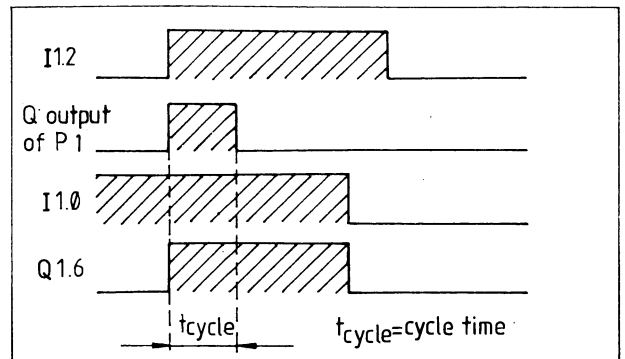
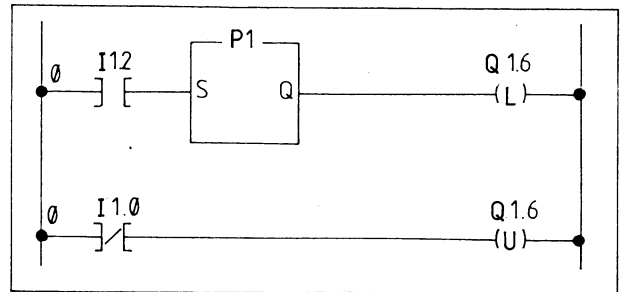
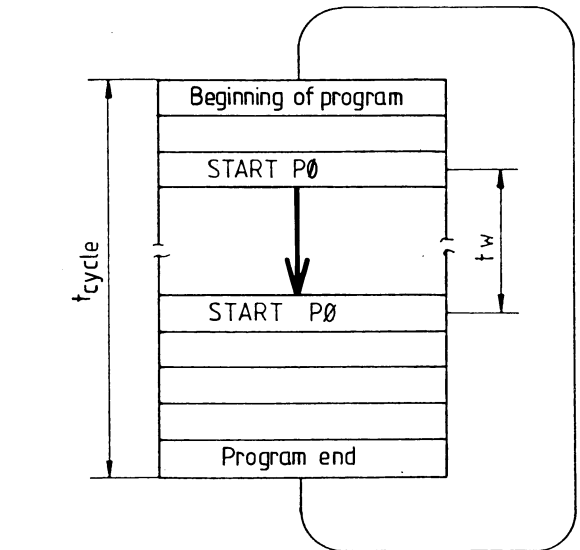




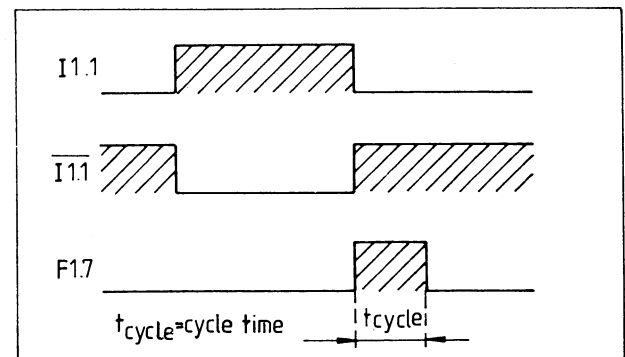
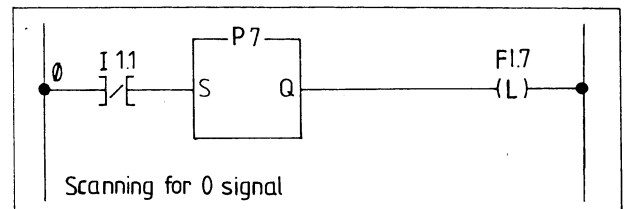
The next time a PØ is scanned, Q becomes '0' again. Signal state '1' applies only for the time between the two PØ operations.

Typical examples:

1. Positive edge evaluation.  
If the signal changes from '0' to '1' at the START input of P1, Q1.6 is set to '1'. If I1.0 changes to '0', Q1.6 is reset.



2. Negative edge evaluation  
When the signal at the START input changes from '1' to '0', flag F1.7 is set.



## 1.3.4 Jumps

The operations of all program elements are listed in Section 5.

The jump function makes it possible to skip one or more program blocks. A jump can be made from anywhere within a program block; the jump destination is always the start of a program block.\*

**Jump condition:** If the input of the jump function has a '1' signal, a jump is executed.  
If the input has a '0' signal, the following program is processed.

**Jump destination:** The destination can be specified as  
- a constant  
- a register stored value for data (DR)  
The constants are entered when generating the program.

The stored values can be entered after program generation or can also be modified in RUN mode.  
If no value is entered, the highest data value, i.e. 32767, is stored in the register after a general reset of the 105R PC.  
The 105R PC does not start because of the error DR X TOO LARGE.

**Jump range:** 0 to 63  
The target program blocks must be available in the program.

**Jump direction:** Jumps are permissible both forwards and backwards.

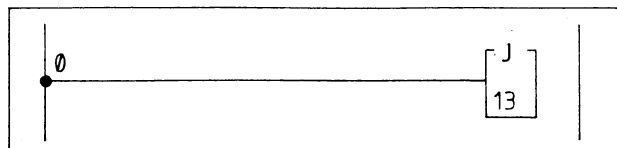
An endless loop must not be created when jumping backwards as the 105R PC otherwise enters the STOP state owing to the scan time being exceeded.

### Formal rules for using jump functions in the program

Jump functions complete a rung. They have no output.

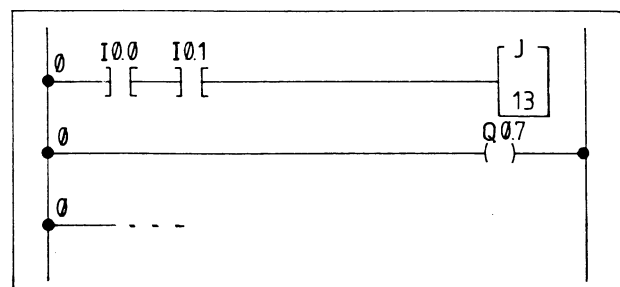
#### Unconditional jump

Whenever the program reaches the rung, a jump takes place to program block 13.



#### Conditional jump

The jump is only executed if I0.0 and I0.1 have a '1' signal. If one of the two inputs has a '0' signal, the following program is processed (in the example: latching the output)

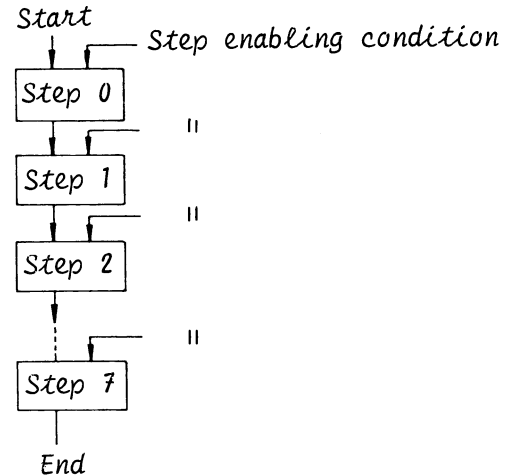


\* see also page 1.23

### 1.3.5 Sequence control systems/sequence cascades (drum sequencers)

The operations of all program elements are listed in Section 5.

A sequence cascade or drum sequencer has a maximum of eight steps. Depending on the step enabling conditions, these are run through one after the other. A step flag is assigned to each step. Only one step is activated at a time.



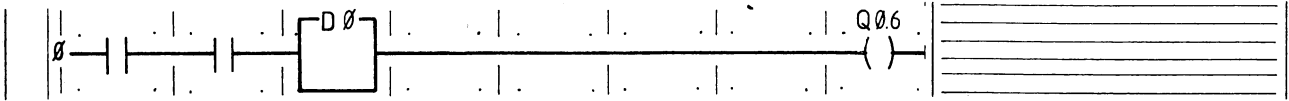
- Starting a sequence cascade: When the signal changes from '0' to '1' at a START input, the sequence cascade is enabled.
  
- Stopping a sequence cascade: When the signal at the HOLD input changes from '0' to '1', the sequence cascade is stopped.
  
- Scanning the sequence cascade: When the last stop has been reached, the sequence cascade output assumes the '1' state.
  
- Resetting: When the signal state at the START input changes from '1' to '0', the sequence cascade is reset.
  
- Entering the number of steps: The number of steps can be entered as
  - a constant (CON)
  - a register value for data (DR)
 The constants are entered when generating the program.

The register values can be entered after generating the program or can be modified in RUN mode. If no value is entered, the highest data value, i.e. 32767, is stored in the register after a general reset of the 105R PC. The 105R PC does not start because of the error `< DR X > TOO LARGE`.

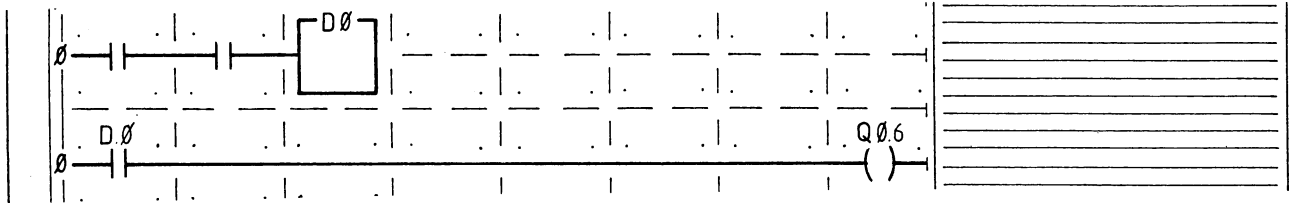
- Step numbers: 0 ... 7  
  
(The maximum number of steps can be increased by connecting sequence cascades in series).

## Formal rules for using sequence cascades in the program

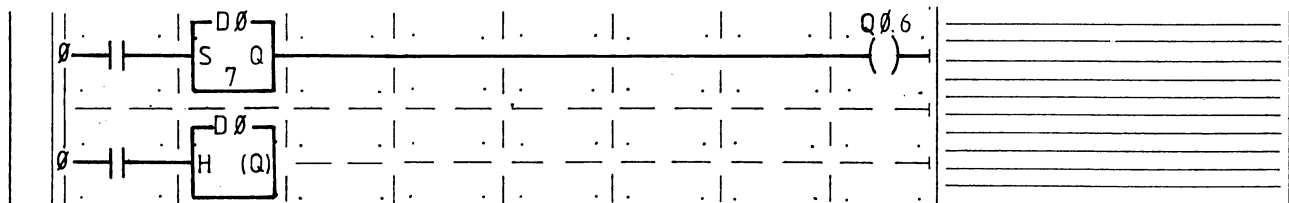
Sequence cascades can be incorporated into a rung like contacts



Scanning or continuation of the output in the same rung is not mandatory; it is therefore possible to scan the sequence cascade as a contact.



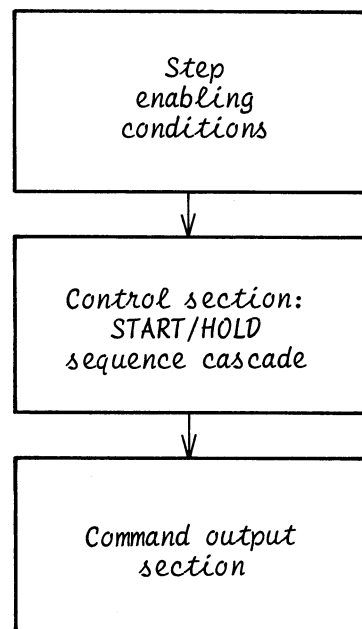
A sequence cascade can be addressed via the START and HOLD inputs.



The highest step number is entered in the 'START' function box of the sequence cascade. Outputs Q and  $\bar{Q}$  are available on both function boxes.

## Programming a sequence cascade

The programming sequence shown in the diagram (right) must be adhered to. It is advisable to divide the program into several program blocks for clarity.



## Step enabling conditions

In the first part of the program the enabling conditions for the step flags are evaluated.

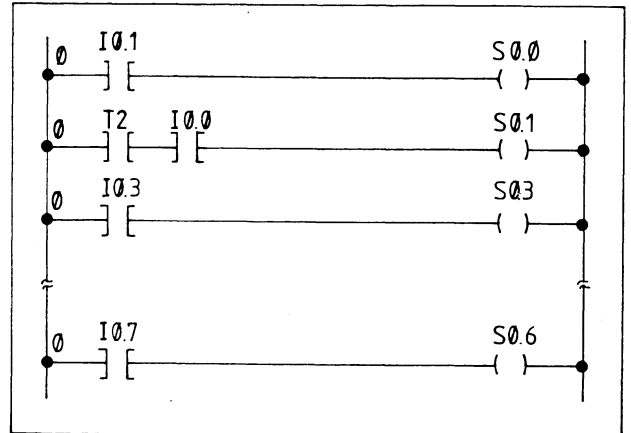
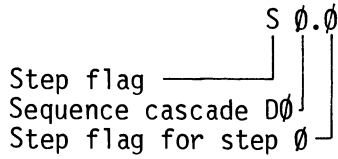


Fig. 5 Enabling conditions for 7 step flags of sequence cascade D0

## Control section

In the control section the sequence cascade is linked into the general program. The sequence cascade is started when contact I0.0 = '1' and condition F0.5 = '1' is fulfilled; it can be stopped when I0.1 = '1'. When the sequence cascade reaches step 6, output Q0.6 is set to '1'.

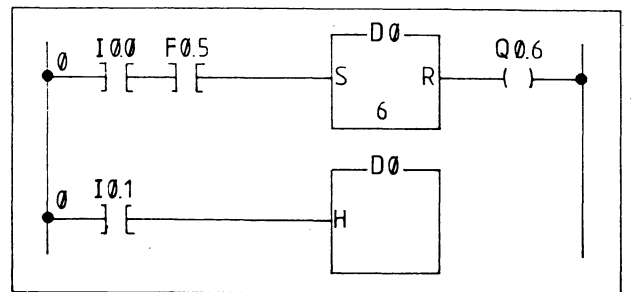


Fig. 6 Calling up a sequence cascade in the program

## Command output section

In the command output section the output commands are assigned to the individual steps.

Only one step flag has a '1', output Q1.0 has been latched in step 0. It therefore remains set during step 1 and is not reset until step 2 is reached.

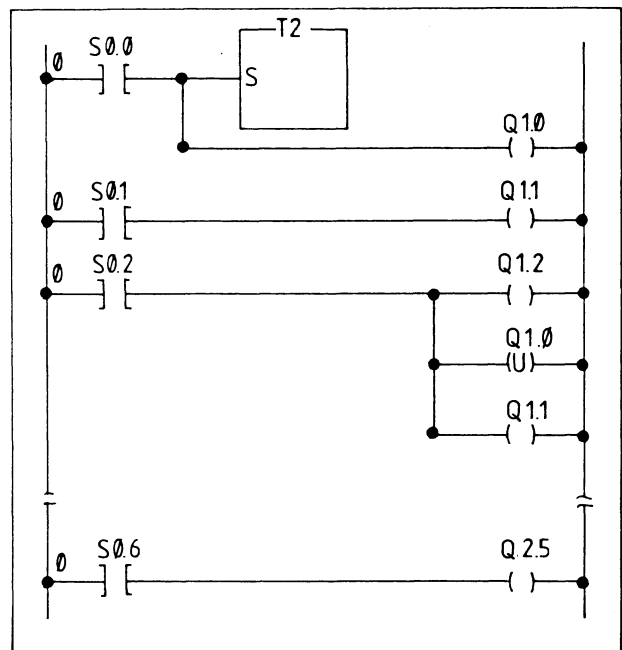
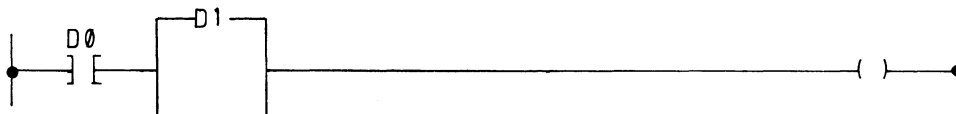


Fig. 7 Command output section of a sequence cascade

- A jump must not be made into or out of a sequence cascade.
- If, for example, a certain waiting period must be observed between steps 5 and 6, a timer must be started with the command output of the 5th step. In the step enabling condition for the 6th step, the timer must be scanned for its signal status.

Each waiting period requires its own timer.

- If more than eight sequence steps are required in a sequence cascade, sequence cascades can be connected in series. This means, for example, that sequence cascade D1 is started when the output Q of D0 is scanned:



- Step flags in sequence cascades not used may be used in the program as (non-retentive) flags. However, they cannot be permanently forced (see Section 3.4, Forcing).
- When changing over from RUN to STOP or HOLD, the current step flag remains set.
- When changing over from STOP to HOLD or RUN the sequence cascade is reset.
- When changing over from HOLD to RUN (only possible with programmer), the sequence cascade is processed from the current step flag on.



The design of a sequence control system based on the example of a control system for a stamping machine is described below. The following steps are to be executed consecutively:

1. When a stamping piece is located in front of the slide arm ( $S5 = I\emptyset.0$ ), the slide arm ( $Y1 = Q\emptyset.5$ ) pushes the stamping piece into the die.
2. When the die is loaded ( $S6 = I\emptyset.1$ ) and the slide arm is in the idle position ( $S7 = I\emptyset.2$ ), the stamping tool ( $Y2 = Q\emptyset.6$ ) presses downwards. After the stamping tool ( $S8 = I\emptyset.3$ ) has been in contact with the stamping piece for two seconds, the stamping tool returns to its idle position ( $S9 = I\emptyset.4$ ).
3. After the stamping operation ( $S9 = I\emptyset.4$ ), the ejector ( $Y3 = Q\emptyset.7$ ) ejects the finished part from the die.
4. A stream of air ( $Y4 = Q1.5$ ) from the air nozzle then blows the stamping piece into the container. A photoelectric cell responds ( $B1 = I1.3$ ) when the stamping piece drops into the container.
5. The next stamping operation can then begin.

All three cylinders are equipped with return springs so that the slide arm, stamping tool and ejector return to their idle positions when the corresponding valves  $Y1$ ,  $Y2$  and  $Y3$  are switched off. The initial condition is: All valves  $Y1$  to  $Y4$  closed and stamping die empty.

Process schematic

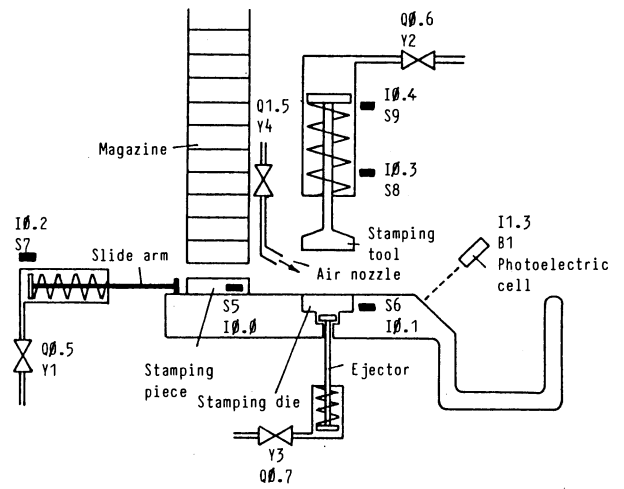


Fig. 8 Example of a sequence control system for a stamping machine

Structure of a sequence control system

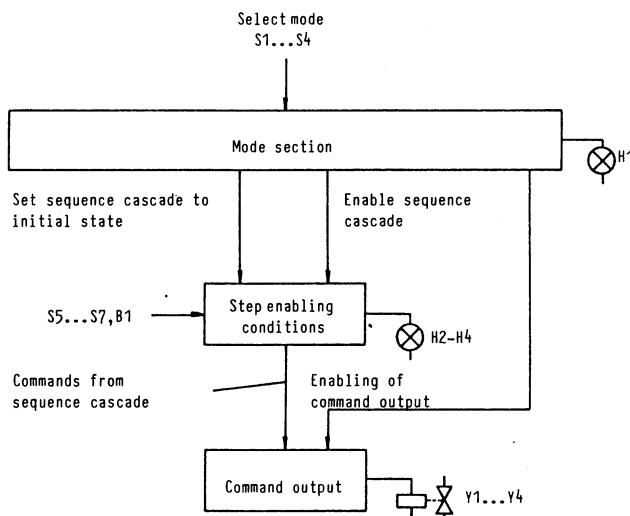
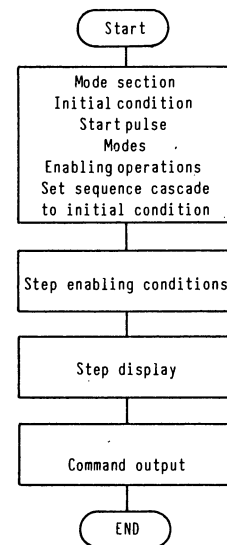


Fig. 9 Basic structure of the control system for the stamping machine

Program structure



The principle of a sequence control system is to break the process down as far as possible into discrete steps. Accordingly, the program itself largely consists of consecutive steps. These are combined to form a sequence cascade. Each step of this cascade is processed individually. The next step is not processed until processing of the previous step has been completed. This greatly simplifies the program, since the interlock conditions can be omitted. To process a step only the signals pertaining to the step concerned need be used; the others are disregarded. A sequence control system consists of three parts.

1. The conditions for individual modes such as start, stop, automatic and single step are processed in the mode section.
2. The actual program of the control system is processed in the step enabling conditions. The individual steps are executed in dependence on the step enabling conditions.
3. The step commands are gated in the command output section with the enabling signal from the mode section and, where applicable, with interlock signal from the machine. As a result, the actuators are switched on and off via the outputs of the programmable controller.

In the case of the stamping machine, the flowchart for the sequence cascade would be as in Fig. 10. The stamping machine must assume its initial condition to permit the sequence cascade to be started. This means that valves Y1 to Y4 are closed (Q0.5, Q0.6, Q0.7, Q1.5 are '0') and sensors S5, S7, S9 (I0.0, I0.2, I0.4) are '1' while sensors S6, S8 (I0.1, I0.3) are '0'. Only when these conditions are fulfilled can the sequence cascade be started with pushbutton I1.0. Pushbutton I1.2 serves to stop the sequence cascade without switching the programmable controller to the stop state. All these signal statuses are scanned in the mode section (see Fig. 12). Measures to implement the initial condition can also be programmed in this section.

The step enabling conditions (see Fig. 11) can easily be read from the arrows entering Fig. 10 from the right.

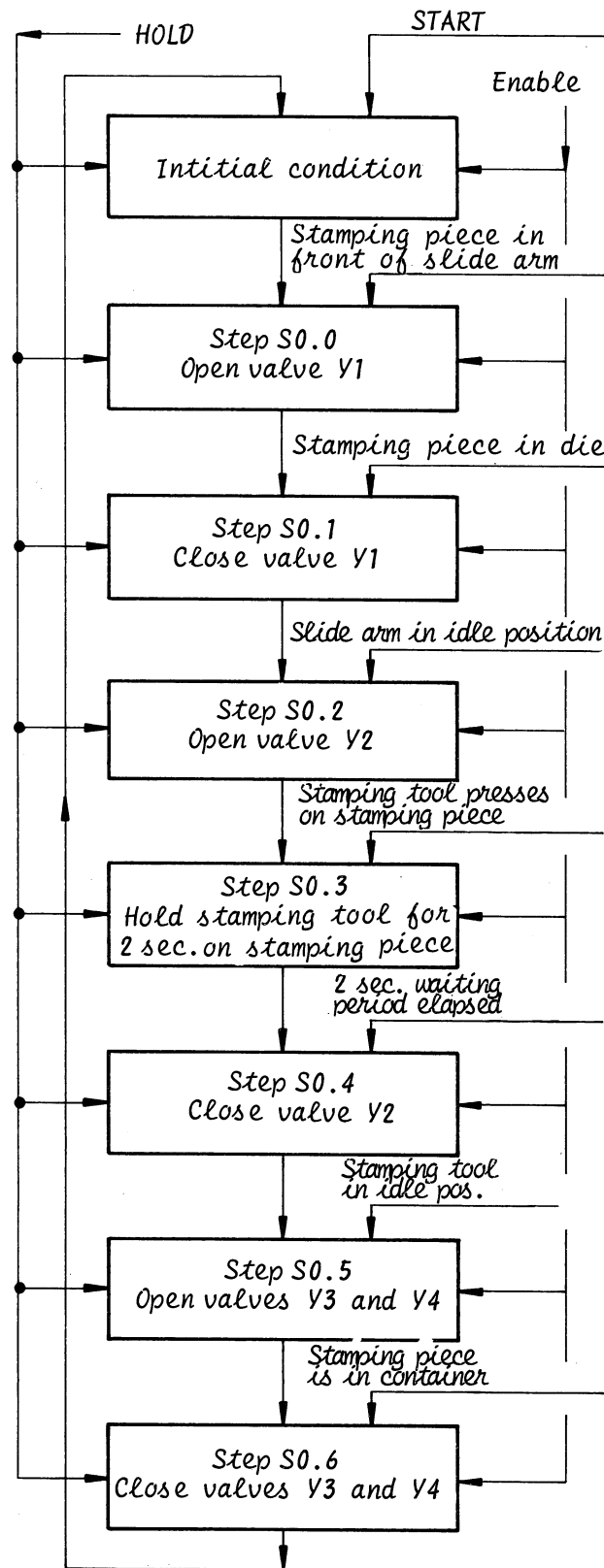


Fig. 10 Flowchart of the sequence cascade for the stamping machine

The command output (see Fig. 13) shows the outputs as a function of the step flags which can also be found easily in Fig. 10.

# Stamping machine program

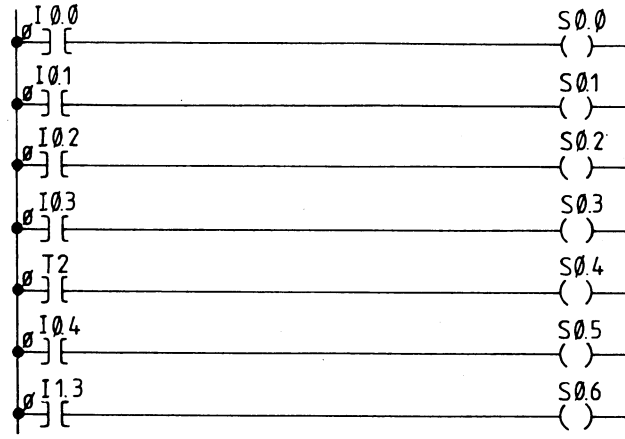


Fig. 11 PB1  
Step enabling  
conditions for the  
stamping machine

*Measures to  
implement  
the initial  
condition for  
the stamping  
machine*

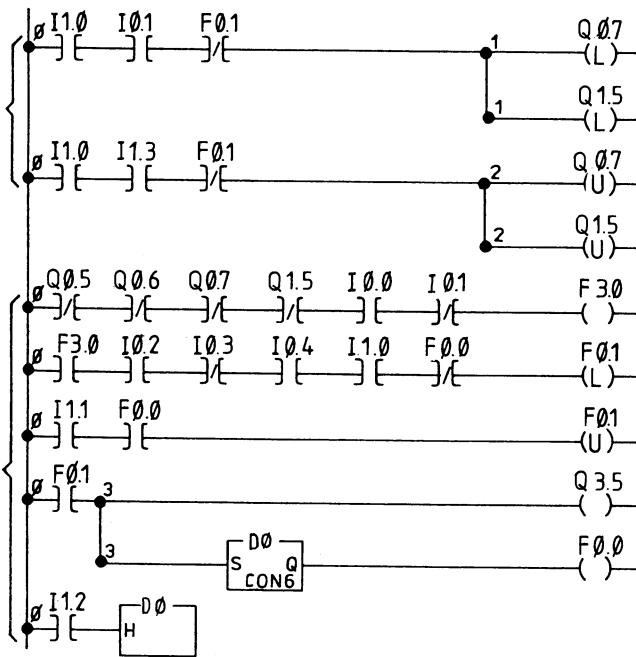


Fig. 12 PB 2  
Control section  
for start, hold

*Scan of  
initial  
condition*

*Control system on  
Control system off  
Start sequence  
cascade  
with control  
system on  
Hold sequence  
cascade*

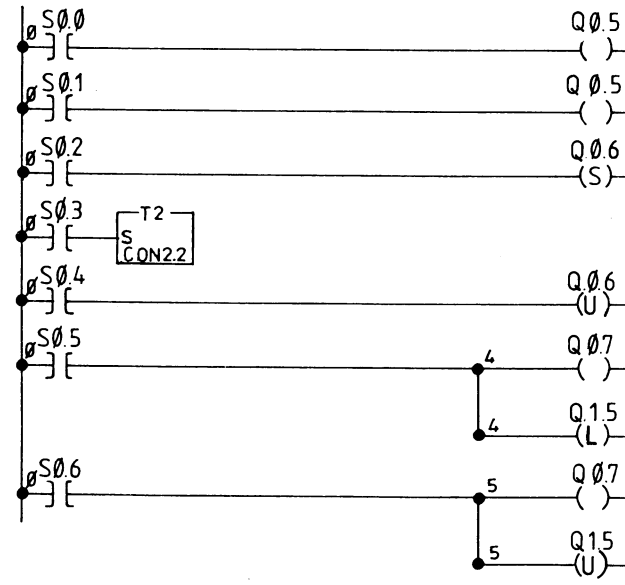


Fig. 13 PB 3  
Command output for  
the stamping machine

## 2. Program generation with the 105R PC

### 2.1 Program input and correction

#### Preparation

A program can only be entered in the 105R PC if there is no memory submodule plugged in.

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

- the internal program memory of the 105R PC to be deleted
- the process image of the inputs and outputs to be deleted
- flags to be set to "0"
- current values of the timers to be deleted and the run time set to 999.3
- current values of the counters to be deleted and the count limits set to 32767.

#### Input

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

#### Correction

The following can be deleted:

- The entire program
- Single PBs
- Single rungs
- Single program elements

The following can be inserted:

- Program elements
- Rungs

The following can be overwritten:

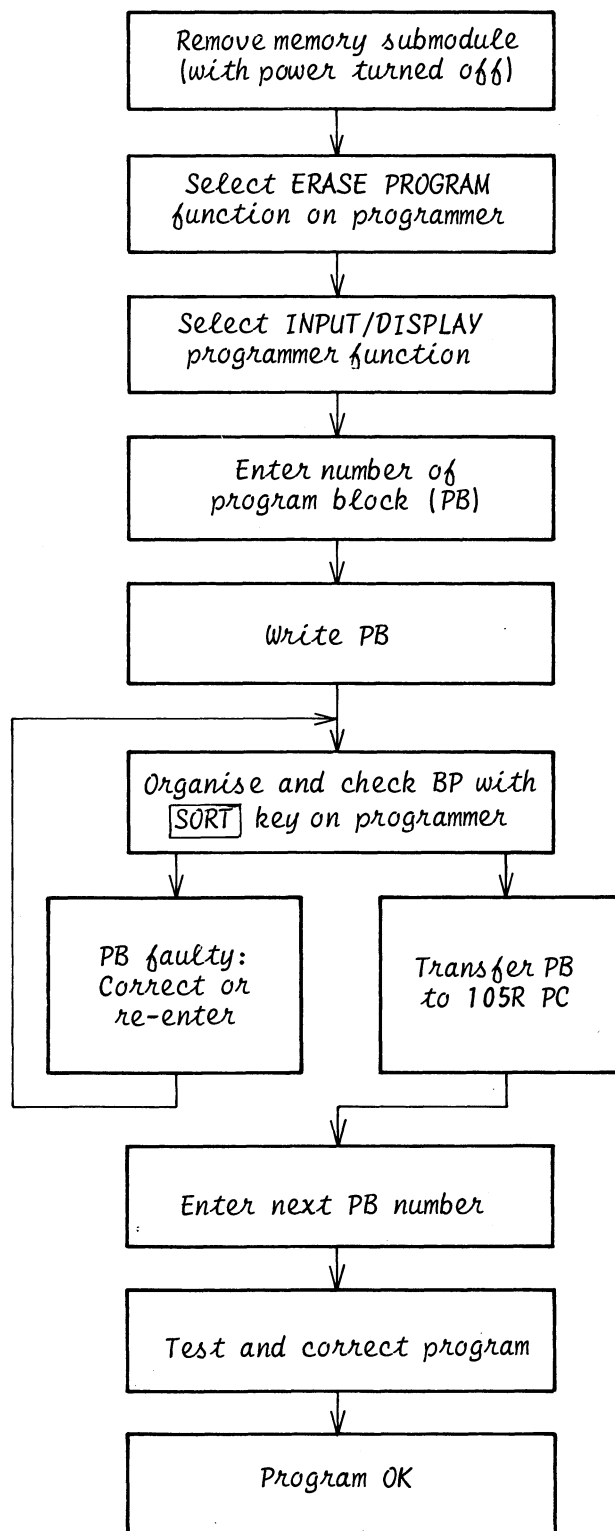
- Program elements
- Rungs
- PBs

The following can be assigned new numbers:

- Program blocks
- The entire program

For further details, see the User Instructions of the 605R and 655R programmers.

2.1



## 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 in the example in Fig. 14 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 wagon. 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 wagon is situated in the filling position (limit switch S3). The conveyor is switched off again if a wagon has left the filling position and the next wagon 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 wagon 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 wagon, latching pawl Y2 is only opened 10 seconds after the "Full" message. The latching pawl is immediately closed again when the filled wagon has left the filling position, i.e. a contact of limits switch S3 has opened again.

When the next wagon 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

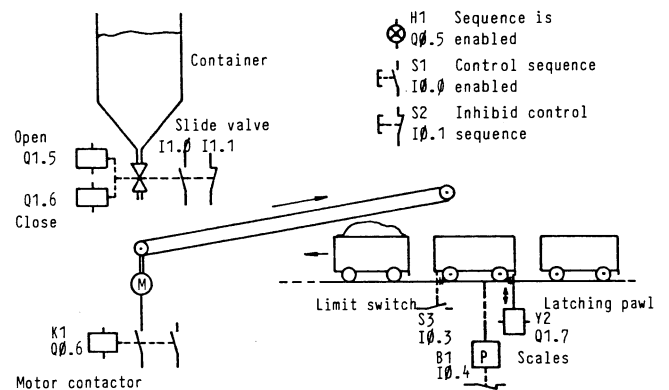


Fig. 14 Filling waggons from a conveyor belt

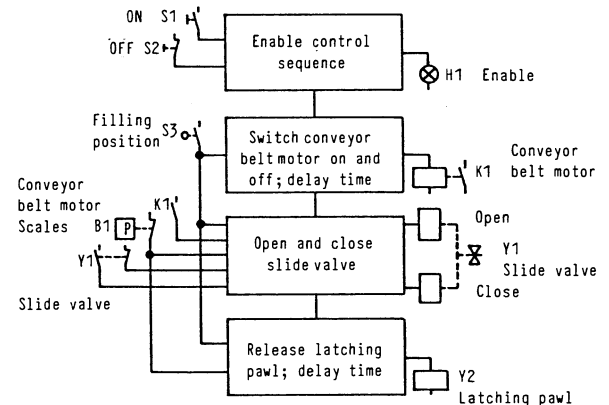


Fig. 15 Structural diagram for wagon filling system

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. 14.

Operand	Device identifier	Functional description
<b>Inputs:</b>		
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 '0'
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 1.0	Y 1	Slide valve, acknowledgement, valve is open, idle condition is '0'
I 1.1	Y 1	Slide valve, acknowledgement, valve is closed, idle condition is '1'
<b>Outputs:</b>		
Q 0.5	H 1	Indicator lamp, control system is enabled, idle condition is '0'
Q 0.6	K 1	Motor contactor, switch conveyor 1, idle condition is '0'
Q 1.5	Y 1	'Open' slide valve, idle condition is '0'
Q 1.6	Y 1	'Close' slide valve, idle condition is '0'
Q 1.7	Y 2	Release latching pawl, idle condition is '0'
<b>Timers:</b>		
T 0	-	20 sec. delay for conveyor motor
T 1	-	10 sec. delay for latching pawl

Fig. 16 Assignment list for waggon filling system

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 for the "Waggon filling system" is shown in Fig. 17. When the program has been written, it is loaded into the PC and tested.

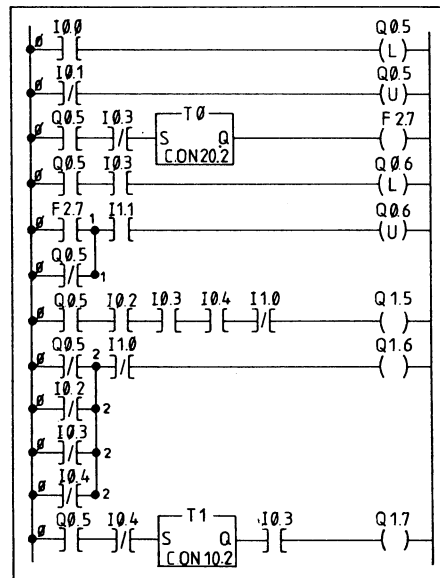


Fig. 17 Program for the waggon filling system

# 3. Program test

## 3.1 Search function

Within a program this function searches for the following:

- Operands e.g. I1.3, F3.0, T1
- Program elements, e.g.  $\neg$ I1.3,  $\neg$ (L) F3.0,  $\neg$ [S] T1

It can be executed in the programmer functions

- INPUT/DISPLAY
- PROG. TEST

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

The rungs containing the element searched for are displayed on the programmer.

### Search function in the entire program

There is only one program block in the programmer at one time. When this has been checked, the search continues in the memory of the 105R PC. The PB numbers containing the term sought are displayed on the programmer. For precise location, the respective PB can be brought into the programmer.

## 3.2 Signal status display

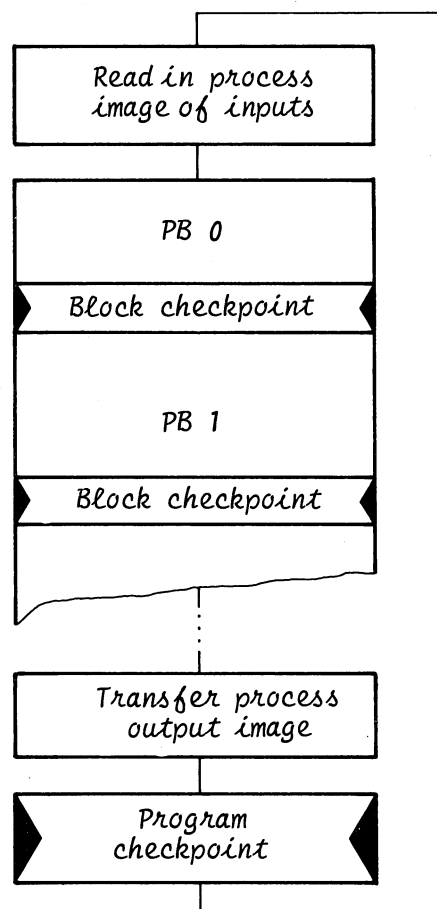
At the end of each processing cycle the following can be observed on the programmer:

- Signal states of operands  
e.g. I0.3, F3.7, Q1.6
- Current values and signal states of timers, counters, impulse relays and sequence cascades.

Two checkpoints can be selected:

- At the end of a PB\*, with the PWRFLOW/FORCE programmer function
- At the end of the program (program checkpoint) by means of the STATUS/SET programmer function

\* It is always the checkpoint of the program block currently in the programmer which is processed.



### 3.3 Forcing

Forcing is the once-only assignment of a signal state to an operand (e.g. I0.3, F3.5, T1)

This default is only valid until the operand is assigned the current signal state by program processing.

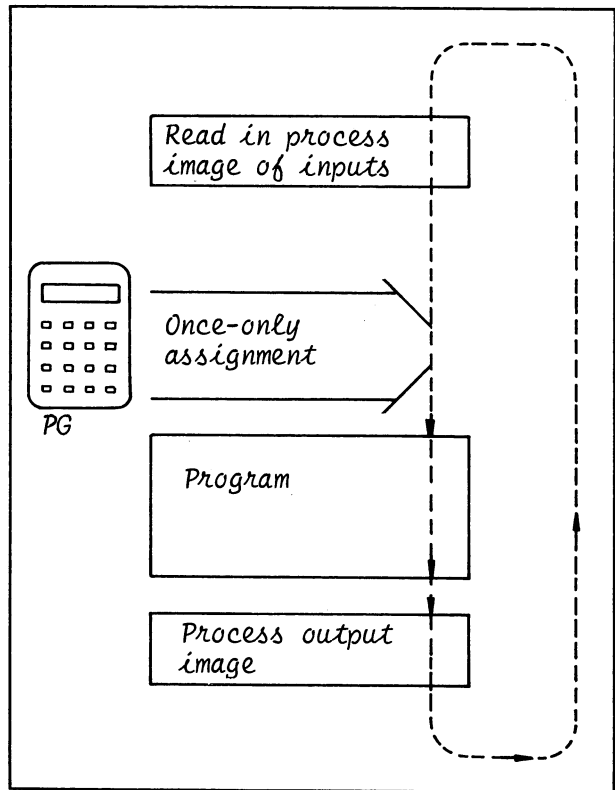


Fig. 18. Once-only assignment of signal state when forcing

### 3.4 Permanent forcing

Permanent forcing is an unmodifiable assignment of a signal state to an operand (e.g. I0.3, F3.5, T1)

A permanently forced operand cannot be changed by assignments resulting from the program processing. All permanently forced elements are enabled by

- DISABLE FORCE (terminate permanent forcing) programmer function
- Operation of the mode selector
- Unplugging the programmer connecting cable

The following cannot be forced:

- Step flags in sequence cascades
- Output  $Q/\overline{Q}$  of sequence cascades
- Output  $Q/\overline{Q}$  of impulse relay



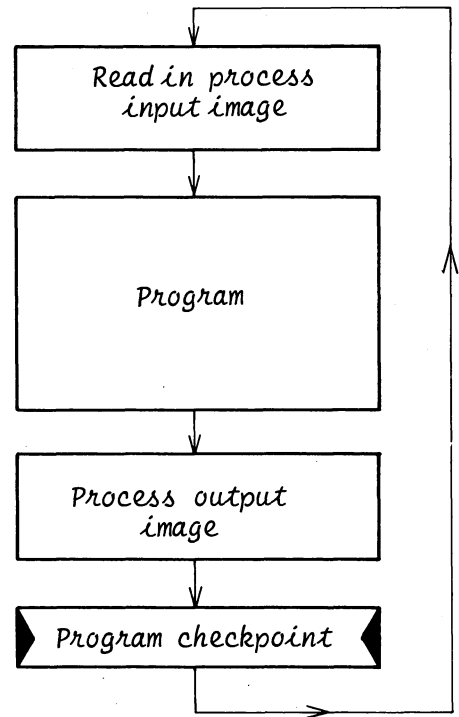
### 3.5 Single scan in HOLD mode

This involves a single, complete scan of the program. The starting and end point is the program checkpoint at which the 105R is waiting in HOLD mode.

A single scan can be triggered by pressing the **SSCN** key in the

- STATUS/SET and
- PWR FLOW/FORCE programmer functions

In conjunction with the forcing and permanent forcing of operands, the single scan is a convenient aid for program testing.



## 4. Storing the program

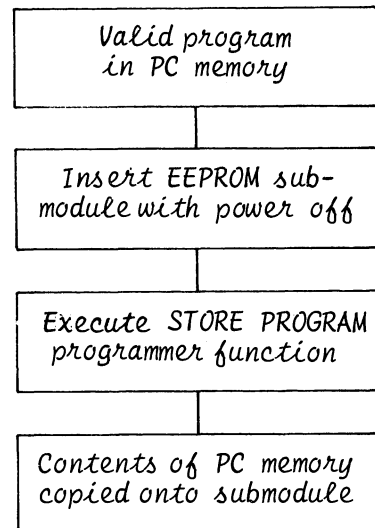
A valid program can be transferred from the internal memory of the 105R PC to a plug-in memory submodule.

### 4.1 Storing the program on an EEPROM submodule

An EEPROM submodule is plugged into the CPU for storing the program. The contents of the 105R PC program memory 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.

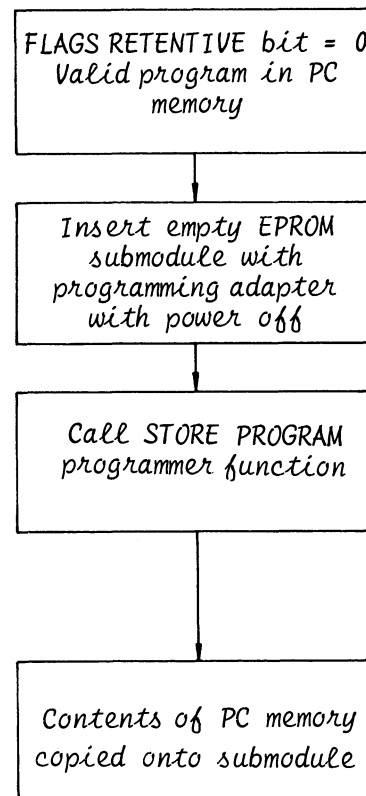


### 4.2 Storing the program on an EPROM submodule

To store a program, an empty EPROM submodule is inserted into the CPU with a programming adapter. The contents of the 105R PC program memory are copied onto the submodule by means of the STORE PROGRAM programmer function.

N.B.

- The memory submodule and program adapter may only be inserted and removed with the PC switched off!
- Programs in which the FLAGS RETENTIVE bit is set to "1" cannot be transferred to EPROM submodules.

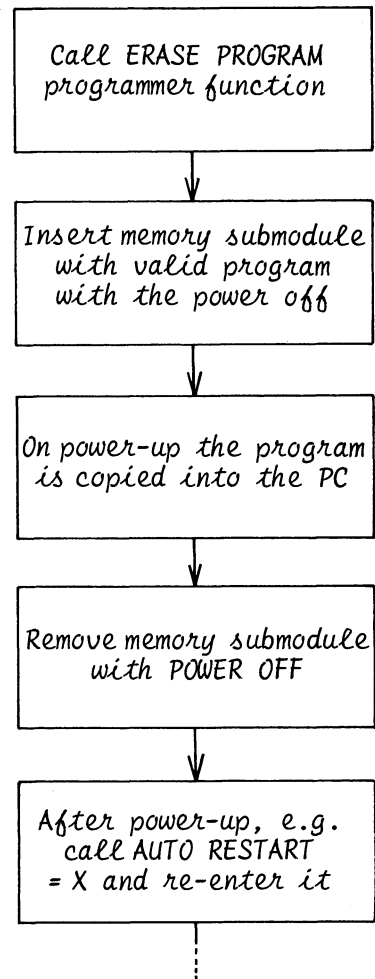


## 4.3 Duplication of programs

After a general reset of the 105R PC, 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 105R PC.

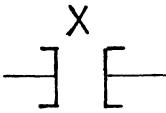
To prepare the STORE PROGRAM function, an entry must be made in the program in the internal memory (see Operating Instructions, Section 3.4 "Using the memory submodule, Note") e.g. read out and re-enter AUTO RESTART = X. See 4.1 or 4.2 for further action.



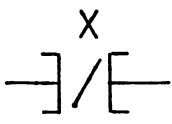
See 4.1 or 4.2

# 5. Operation set

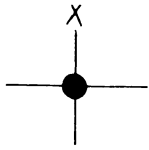
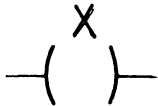
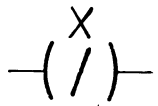
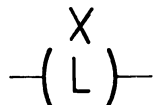
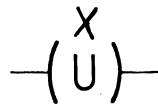
## 5.1 Binary operations

Symbol	Operand	Description
	X=I0.0 to I0.7 =I1.0 to I1.7 : : =I7.0 to I7.7	Scanning an input for "1"
	X=Q0.5 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	Scanning an output for "1"
	X=F0.0 to F5.7 <sup>1)</sup>	Scanning a flag or internal relay for "1".
	X=S0.0 to S3.7 <sup>2)</sup>	Scanning a step flag for "1".
	X=T0 to T31	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 C15	Scanning a counter for "1". (The up-counter has a "1" at the Q output if the specified limit is reached or exceeded, whilst the downcounter has a "1" at the Q output if the specified limit is reached or if the count drops below it).
	X=D0 to D3	Scanning a sequence cascade or drum sequencer for "1". (The sequence cascade has a "1" at the Q output if there is a "1" at the set input of the sequence cascade and the last step of the sequence cascade has been reached).
	X=P0 to P15	Scanning an impulse relay (transition-sensitive pulse) for "1". (The impulse relay has a "1" at the Q output for the duration of one cycle if the signal changes from "0" to "1" at its set input).

1) Flags 0.0...1.7 can be set as retentive in the case of an EEPROM submodule  
 2) Used with sequence cascades.

Symbol	Operand	Description
	X=I0.0 to I0.7 =I1.0 to I1.7 : : =I7.0 to I7.7	Scanning an input for "0".  Scanning an output for "0".
	X=Q0.0 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	
	X=F0.0 to F5.7 <sup>1)</sup>	Scanning a flag for "0".
	X=S0.0 to S3.7 <sup>2)</sup>	Scanning a step flag for "0"
	X=T0 to T31	Scanning a timer for "0". (The timer has a logic "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 C15	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, whilst the down-counter has a "0" at the Q output if the specified limit has been reached or exceeded).
	X=K0 to K3	Scanning a sequence cascade for "0" (The sequence cascade has a "0" at the Q output, if there is a "0" at the set input of the sequence cascade or a "1" at the set input of the sequence cascade and the last step of the sequence cascade has not yet been reached).
	X=P0 to P15	Scanning an impulse relay for "0". (The impulse relay has a "0" at the Q output if there is a "0" at the set input of the impulse relay or a "1" for longer than once cycle).

- 1) Flags 0.0...1.7 can be set as retentive in the case of an EEPROM submodule
- 2) Used with sequence cascades.

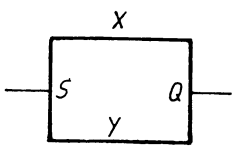
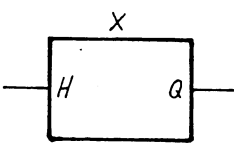
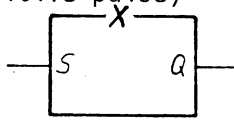
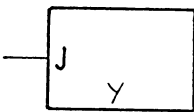
Symbol	Operand	Description
	X=0 to 143)	The nodes are used for combining rungs. Each rung begins with a node. The connection to the left-hand power rail is always made with node 0.
	X=Q0.0 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	Set output to "1"
	X=F0.0 to F5.7 <sup>1)</sup>	Set flag to "1"
	X=S0.0 to S3.7 <sup>2)</sup>	Set step flag to "1"
	X=I0.0 to I0.7 =I1.0 to I1.7 : : =I7.0 to I7.7	Set input to "0" (The "0" is only entered in the process image).
	X=Q0.0 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	Set output to "0"
	X=F0.0 to F5.7 <sup>1)</sup>	Set flag to "0"
	X=Q0.0 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	Latch output to "1"
	X=F0.0 to F5.7 <sup>1)</sup>	Latch flag to "1"
	X=I0.0 to I0.7 =I1.0 to I1.7 : : =I7.0 to I7.7	Set input to "0" (The signal status is only entered in the process image).
	X=Q0.0 to Q0.7 =Q1.0 to Q1.7 : : =Q7.0 to Q7.7	Unlatch output
	X=F0.0 to F1 <sup>1)</sup>	Unlatch flag

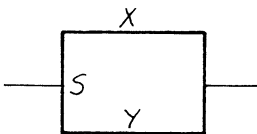
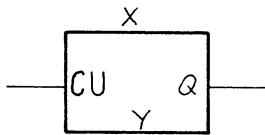
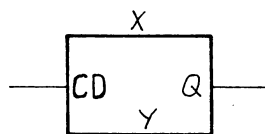
1) Flags 0.0 to 1.7 can be set as retentive in the case of an EEPROM submodule.

2) Used with sequence cascades.

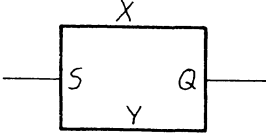
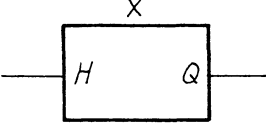
3) Display on 605R programmer in hexadecimal, 0 to E

## 5.2 Complex operations

Symbol	Operand	Description
<p>Start timer</p>  <p>Y = Time = Time value time base</p> <p>Time value: 1 to 999 Time base: 0 <math>\hat{=}</math> 10 ms 1 <math>\hat{=}</math> 100 s 2 <math>\hat{=}</math> 1 ms 3 <math>\hat{=}</math> 1 min</p>	<p>X=T0 to T23</p> <p>Y=CON 1.0 to CON 999.3 or Y=TR0 to TR23</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>The constant Y can also be replaced by a time register TR, in which case it is independent of program input.</p>
<p>Hold timer</p> 	<p>X=T0 to 31</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>
<p>Impulse relay (transition-sensitive pulse)</p> 	<p>X=P0 to P15</p>	<p>With every change from "0" to a "1" at the S input, the Q input changes to "1" for the duration of one scan.</p> <p><math>\bar{Q}</math> can also be used as output, with the inverted signal of Q.</p>
<p>Jump</p> 	<p>Y=CON 0 to CON 63 or Y=DR0 to DR23</p>	<p>A "1" at the J input has the effect of implementing a jump to a program block whose number is specified by constant Y.</p> <p>Constant Y can also be replaced by a data register DR, in which case it is independent of program input</p> <p>Depending on the programming, unconditional jumps can be executed with this operation.</p>

Symbol	Operand	Description
<p>Set counter</p> 	<p>X=C0 to C15</p> <p>Y=CON 0 to CON 32767 or Y=DR0 to DR23</p>	<p>A "1" at the S input causes constant Y to be loaded and enabled.</p> <p>Y can have any value between 0 and 32767.</p> <p>The constant Y can also be replaced by a data register DR, in which case it is independent of program input</p>
<p>Count up</p> 	<p>X=C0 to C15</p> <p>Y=CON 0 to CON 32767 or Y=DR0 to DR23</p>	<p>With every change from "0" to "1" at the CU input, the previous count is incremented by 1.</p> <p>Constant Y specifies the upper limit of the counter. If this limit is reached or exceeded, the Q output changes to "1".</p> <p><math>\bar{Q}</math> can also be used as output, with the inverted signal of Q.</p> <p>Constant Y can also be replaced by a data register DR, in which case it is independent of program input.</p>
<p>Count down</p> 	<p>X=C0 to C15</p> <p>Y=CON 0 to CON 32767 or Y=DR0 to DR23</p>	<p>With every change from "0" to "1" at the CD input, the previous count is decremented by 1.</p> <p>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><math>\bar{Q}</math> can also be used as output, with the inverted signal of Q.</p> <p>Constant Y can also be replaced by a data register DR, in which case it is independent of program input.</p>



Symbol	Operand	Description
<p>Start sequence cascade</p> 	<p>X=C0 to C3  Y=CON 0 to CON 7  or  Y=DR0 to DR23</p>	<p>A "1" at the S input starts and enables the sequence cascade. Constant Y specifies the number of the last sequence step. Constant Y can also be replaced by a data register DR, in which case the constant is independent of program input. The Q output is "1" if there is a "1" at the S input and the last sequence step has been reached. <math>\bar{Q}</math> can also be used as output, with the inverted signal of Q.</p>
<p>Hold sequence cascade</p> 	<p>X=C0 to C3</p>	<p>A "1" at the H input holds the sequence cascade at a particular sequence step. The Q output is "1" if there is a "1" at the S input and the last sequence step has been reached. <math>\bar{Q}</math> can also be used as output, with the inverted signal of Q.</p>