PLUTO

Safety-PLC

Programming language

Table of contents:

1 General

This program manual features the programming language used for programming the PLUTO Safety-PLC. The programming language is related to the programming standard IEC 61131-3. PLUTO Safety-PLC is programmed with the ladder editor Pluto Manger or in textform with a standard text editor. Before downloading to the system the code must be compiled to hex-format. Download of the hex-file to a PLUTO-unit and monitoring is possible by either Pluto Manager or a standard terminal program as Hyper Terminal.

2 Bit-instructions

2.1 Addressing of bit-operands

In PLUTO programming language I/O and memories are addressed as [*I/O-type*][*unit no*].[*I/O no*].
At most 32 PLUTO-units, numbered 0 – 31, can be interconnected via the Bus.

I/O type:	I/O designation Pluto 0	I/O designation Pluto 1	……	I/O designation Pluto 31
Inputs	I0.0 – I0.17	I1.0 – I1.17	……	I31.0 – I31.17
Outputs	Q0.0 – Q0.17	Q1.0 – Q1.17	……	Q31.0 – Q31.17
Memories	M0.0-M0.599	M1.0-M1.599	……	M31.0-M31.599
Global memories	GM0.0-GM0.11	GM1.0-GM1.11	……	GM31.0-GM31.11

<u>Example:</u>

Q10.1 ⇔ Addressing of output 1 on PLUTO no. 10

Following alternatives are also accepted: Q10.01 or Q10.0001 or Q10.1

2.2 Boolean instructions

PLUTO programming language follows the rules for ladder programming of IEC 1131-3 when programming with Pluto Manager.

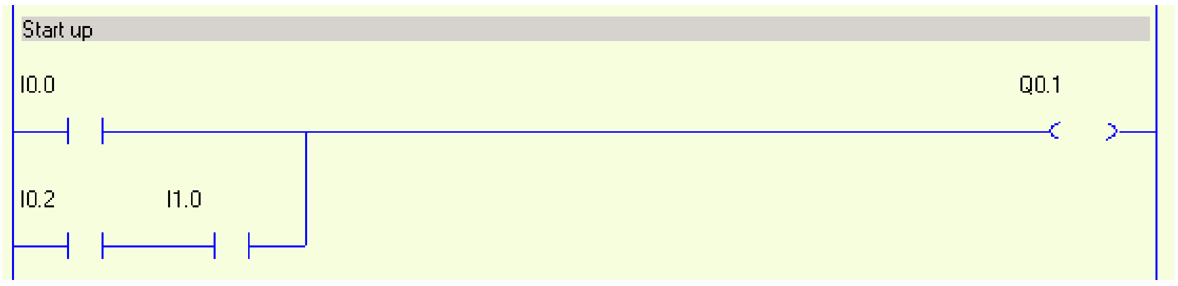
By programming in text form using an text editor the programming language follows the Boolean laws and utilises AND, OR, NOT and EXECUTION -commands.

Program syntax in text form:

Instruction:	Program syntax:
AND	*
OR	+
NOT	/
EXECUTION	=

Example:

In ladder form:



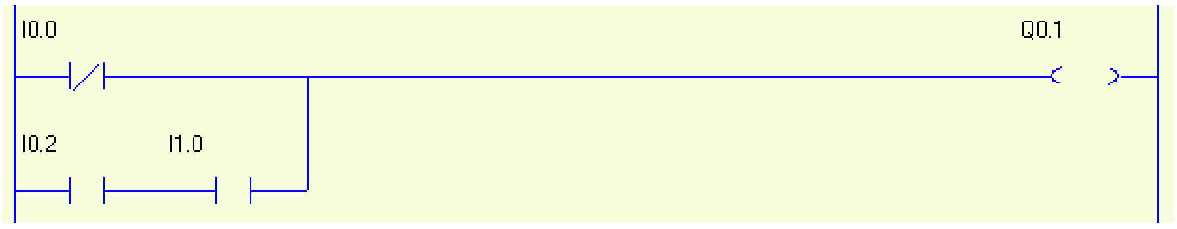
Equivalent text form:

```
Q0.1 = I0.0 + I0.2 * I1.0           ; Start up
```
; (semicolon) defines start of program comments.

Explanation: Output Q0.0 is on when input I0.0 or both of I0.2 and I1.0 is on (‘1’).

Example with negation:

In ladder form:



Equivalent text form:

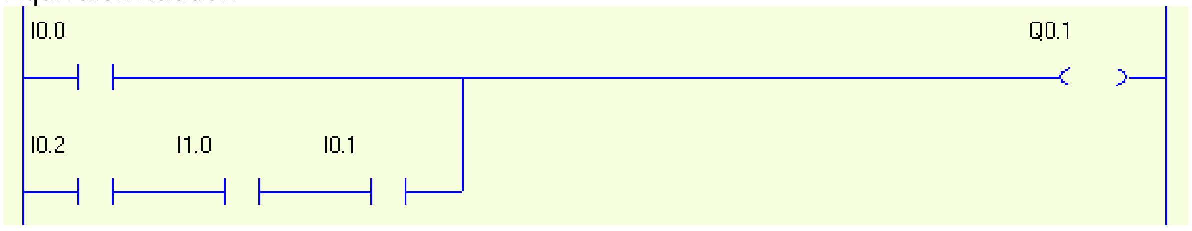
```
Q0.1 = /I0.0 + I0.2*I1.0
```

According to the boolean laws AND-instructions (*) are executed before OR-instructions (+). By using brackets the instruction order can be changed.

Examples:

```
Q0.1 = I0.0 + I0.2*I1.0*I0.1
```

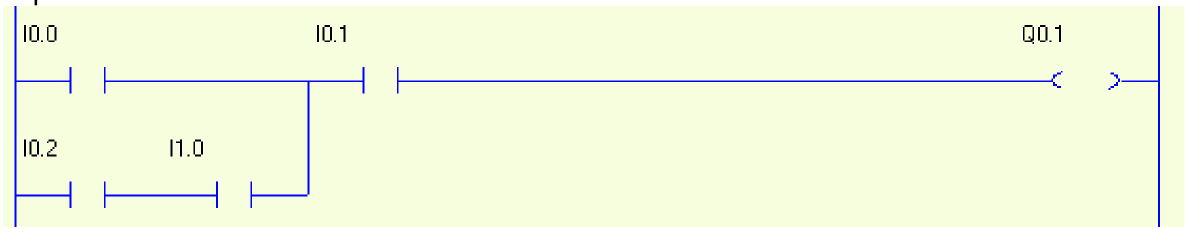
Equivalent ladder:



Example with use of brakets

```
Q0.1 = (I0.0 + I0.2*I1.0)*I0.1
```

Equivalent with:

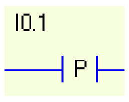


NOTE: In text form the use of spaces have no influence.

2.3 Edge detection

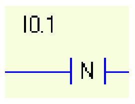
Edge detection can be used on single operands. The EDGE-function enables detection of both positive and negative edges. Relevant program syntax follows in the table below:

Positive edge:



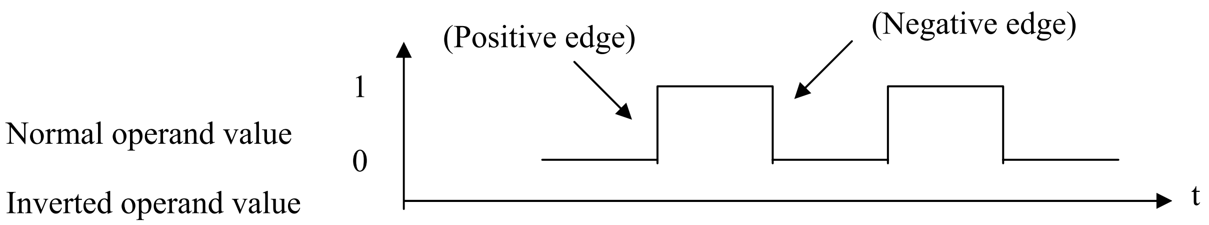
⇔ P(I0.1)

Negative edge:

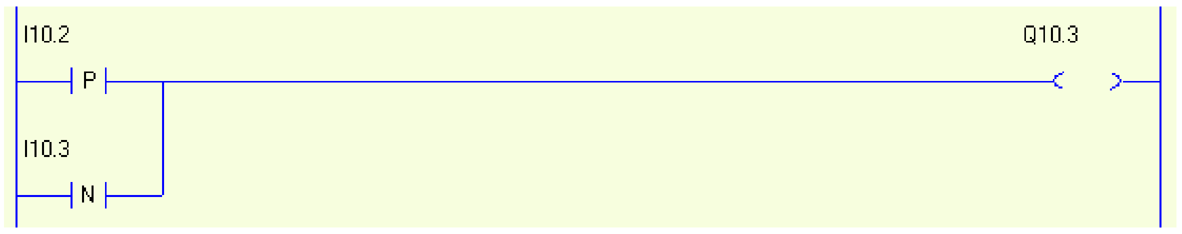


⇔ P(/I0.1)

Function: When an edge is detected a logical “1” is held during a complete program scan cycle.



Example:



`Q10.3 = P(I10.2) * P(/I10.3)` ⇔ Output 3 on PLUTO no. 10 is set HIGH when positive edge is detected on input 2 on PLUTO no. 10

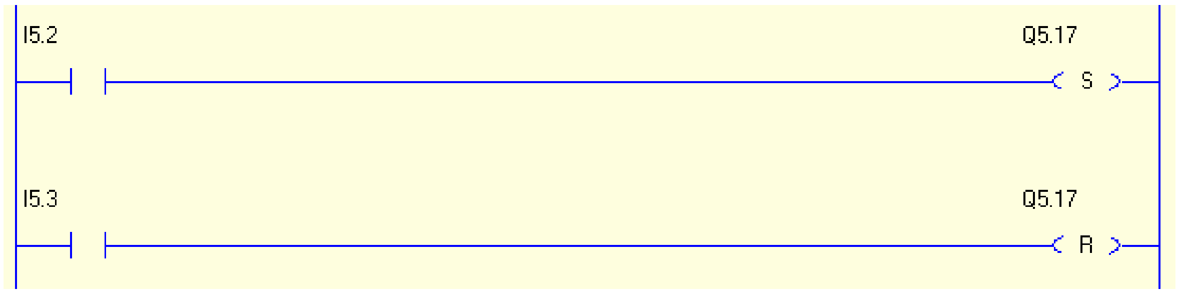
2.4 Latch function

By use of the Latch function an output or a memory-cell is given a self-hold/memory function.

Latch function:	Program syntax:
SET/Latch on	S(Q0.1)
RESET/Latch off	R(Q0.1)

When an output/memory-cell is set HIGH by the SET-instruction, the output/memory-cell will remain HIGH although the previous condition-statement no longer is TRUE. The output/memory-cell can be set LOW by use of the RESET-instruction.

Example:



Equivalent text form:

```
S(Q5.17) = I5.2
R(Q5.17) = I5.3
```

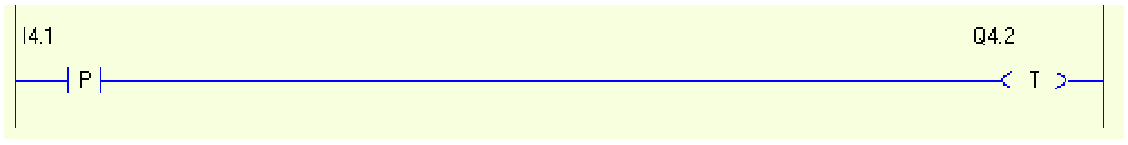
Function: Output 17 on PLUTO no. 5 is set HIGH when input 2 on PLUTO no. 5 is set HIGH. The output remains HIGH until it is RESET by setting input 3 on PLUTO no. 5 HIGH.

2.5 Toggle function

The Toggle function toggles the state of a operand (Q, M or GM).

Toggle function:	Program syntax:
Toggle state	T(Q0.1)

Example:



Equivalent text form:

```
T(Q4.2) = P(I4.1)
```

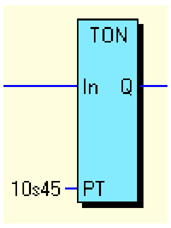
Function: Toggle of output 2 on PLUTO no. 4 changes state from 0 -> 1 or 1 -> 0 on positive edge of input 1 on PLUTO no. 4.

NOTE: In this example edge instruction is used to avoid that Q4.2 toggles more than once. Otherwise the output will toggle ON/OFF every PLC cycle.

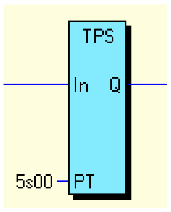
2.6 Timers

PLUTO has 50 timers that all can be used simultaneously in an active sequence steps (see sequences). The timers have a resolution of 10 ms and can be defined in the time-interval 0.01 – 655.35 s.

Timer:	Value	Program syntax:
50	0,01- 655,35 s.	**T**(*nn***S***nn*)



⇔ T(10s45)



⇔ /T(5s0)

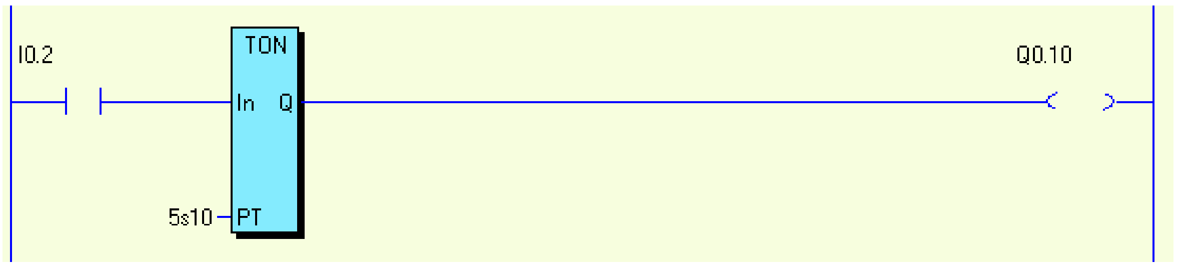
ON-delayed timer (10,45 sec.) Pulse timer (5,0 sec.)

The “s” -symbol corresponds to decimal sign

Function: There are two types of timers: ON-delayed and pulse timers.

ON-delayed timers (TON) starts when the boolean instructions on the left side of the timer instruction is TRUE. When the specified time is elapsed also the timer is TRUE (“1”).
Pulse timers (TPS) are activated in the same way but they are TRUE (“1”) from start and gets FALSE (“0”) when the time has elapsed.

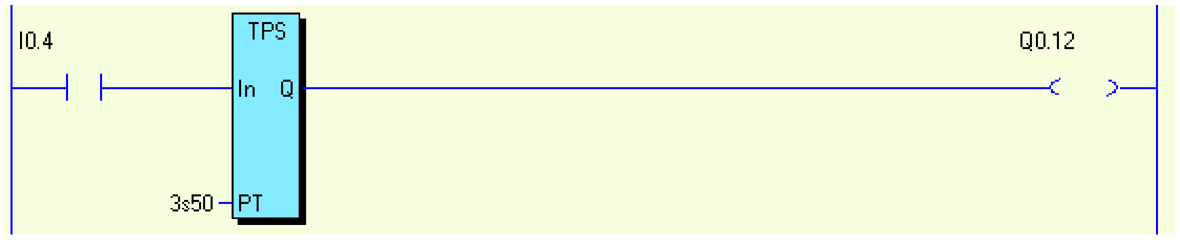
Exemple:



Equivalent text form: `Q0.10 = I0.2 * T(5s10)`

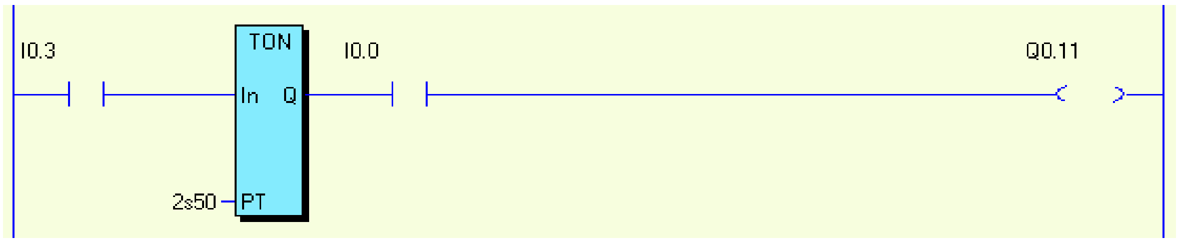
Function: When input I0.2 is set HIGH the timer with time-delay of 5,10s is activated. Output Q0.10 is set HIGH when the time is elapsed.

Example:



Equivalent text form: `Q0.12 = I0.4 * /T(2s5)`

Function: When input I0.4 is set HIGH the timer output and then output Q0.12 is immediately set.
After a delay of 3,5 s the timer switches output Q0.12 off.



Equivalent text form: `Q0.11 = I0.3 * T(2s5) * I0.0`

Function: When input I0.3 is set HIGH the timer is activated.
After a delay of 2,5 s and if input I0.0 is HIGH, output Q0.11 switches on.
Note that the expression after to the right of the timer (I0.0) has no influence on the timer.

3 Memories

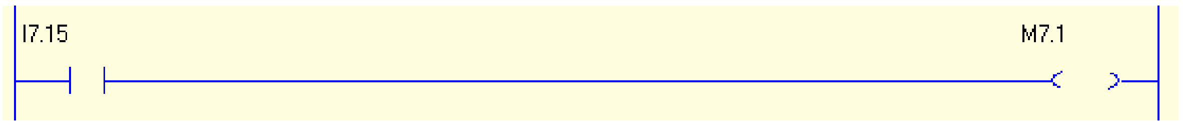
3.1 Local memories (M)

PLUTO has 600 memories free to use in the application program. These memories are local which means that they can just be used in the own Pluto unit. Example memory M0.10 can just be set and read in the application program in Pluto unit no: 0.

The memories are addressed as shown below:

Workmemory-cell:	Program syntax:
600	M_.0 – M_.599

Example:



Equivalent text form: `M7.1 = I7.15`

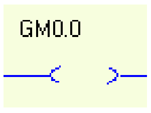
Function: Memory M7.1 is HIGH (1) when input I7.15 is HIGH.

NOTE: Although workmemory-cells are local within one PLUTO PLC, identity of the PLUTO-unit must be set as shown above.

3.2 Global memories (GM)

Global memories can be used in the same way as local memories but with the difference that they are transmitted on the bus and can be read by other Pluto units and used in their application programs as input condition.
One example for use of the global memories is to make it possible to have a memory whitch is the summary of a complex program function. Instead of making the same complex program function in many Pluto:s it can be programmed in just one unit and the result can be stored in a global memory which can be read by all Pluto:s on the bus.

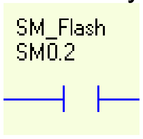
The global memories are addressed as shown below:



Global memory:	Program syntax:
0-11	GM_.0 – GM_.11

3.3 System memories (SM)

A set of system memories with different functions are available in PLUTO.

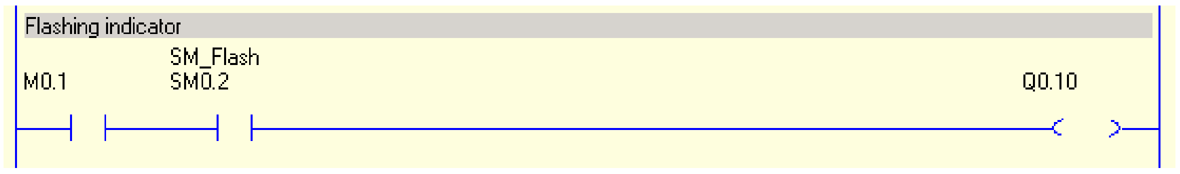


Syntax: SM[*unit*].[*no*]

I/O-address	Symbolic name	Function:	Type:
SM_.0	SM_StepNew	On at first scan in new sequence step.	R
SM_.1	SM_Ditto	Result of last logic operation.	R
SM_.2	SM_Flash	Flash: 0.4 / 0.6 sek. (on/off)	R
SM_.3	SM_1Hz	Pulse 1 Hz	R
SM_.4	SM_10Hz	Pulse 10 Hz	R
SM_.5	SM_FastFlash	Flash: 0.17 / 0,33 sek (on/off)	R
SM_.6	SM_Double Flash	Double flash: 0,11 / 0,2 /0,11 / 0,67 msec	R
SM_.9	SM_SysInit	On at first scan after power on	R
SM_.10	SM_TimerFreeze	Freeze timer, (next timer in program code)	W
SM_.39	SM_Button	Button in front panel	R

(Type: R = Read, W = Write)

<u>Example:</u>



Equivalent text form: `Q0.10 = M0.1 * SM0.2` ;Flashing indicator

Function: System memory SM0.2 is flashing with an on/off rate of 0.4/0.6 seconds.
If M0.1 is set, output Q0.10 flashes with the same rate as SM0.2.

4 Sequences

PLUTO has 9 sequence registers with 255 steps each available for use. The sequences operate in parallel and independent of each other.
In a sequence only the code in one step is executed. The transition from one step to another is conditional via jump-instructions. The result of the previous step is reset when the next step is entered. By start up of the system, sequence step 0 is automatically executed which means that a sequence must contain step 0.

4.1 Addressing

A sequence step starts with an instruction as below declaring sequense number and step number.

Sequence/Step:	Program syntax:
1-9/0-254	S*n*.1_00 – S*n*.9_254 (*n*=Pluto unit no.)

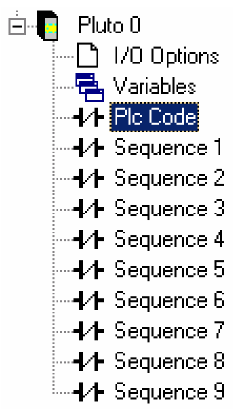
The program syntax in text form is interpreted as follows:

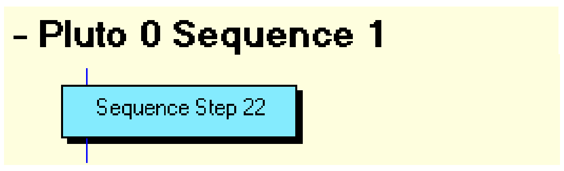
- The first letter concerns sequence register (S)
- The first number sets the identity of the PLUTO-unit where sequence register is to be addressed
- The second number (placed after dot-symbol) sets sequence register to be addressed
- The third number (placed after underscore) sets sequence step to be addressed

Example:

S0.1_22 ⇔ Start of step 22 in sequence 1 on PLUTO no: 0.

Sequence programming in Pluto Manager:

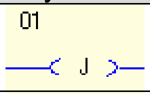
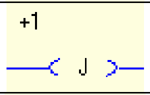
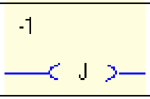
o 



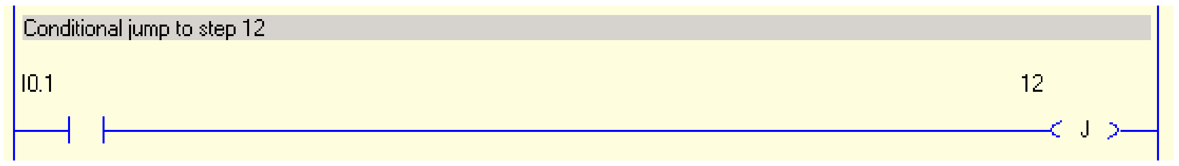
NOTE: By programming in text form Sequence 0 (S*n*.1_0) must be declared. This corresponds to what is named “PLC Code” in ladder programming and is intended to contain the PLC code when sequence programming is not used. Even if it is declared as a sequence it does just contain step 0, the compiler does not accept jumps and declaration of other sequence steps.

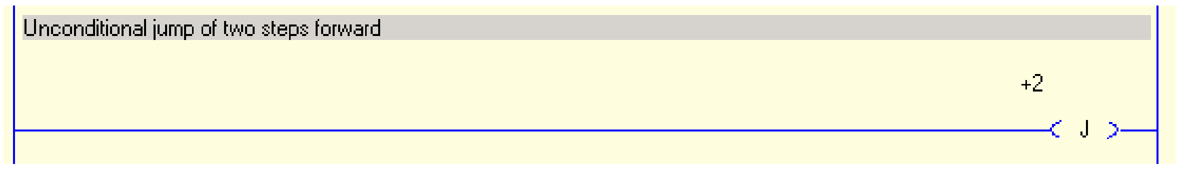
4.2 Jump

The jump instructions are used in sequences in order to jump from one step to another. Jump between sequences steps within a sequence can be performed either absolute or relative to the current active step.

Jump function:	Syntax in text form:	Ladder symbol
Absolute: to step 1	J(01)	
Relative: one step forward	J(+1)	
Relative: one step backward	J(-1)	

The jump can be either condition or unconditional.





<u>Example of a sequence in text form:</u>

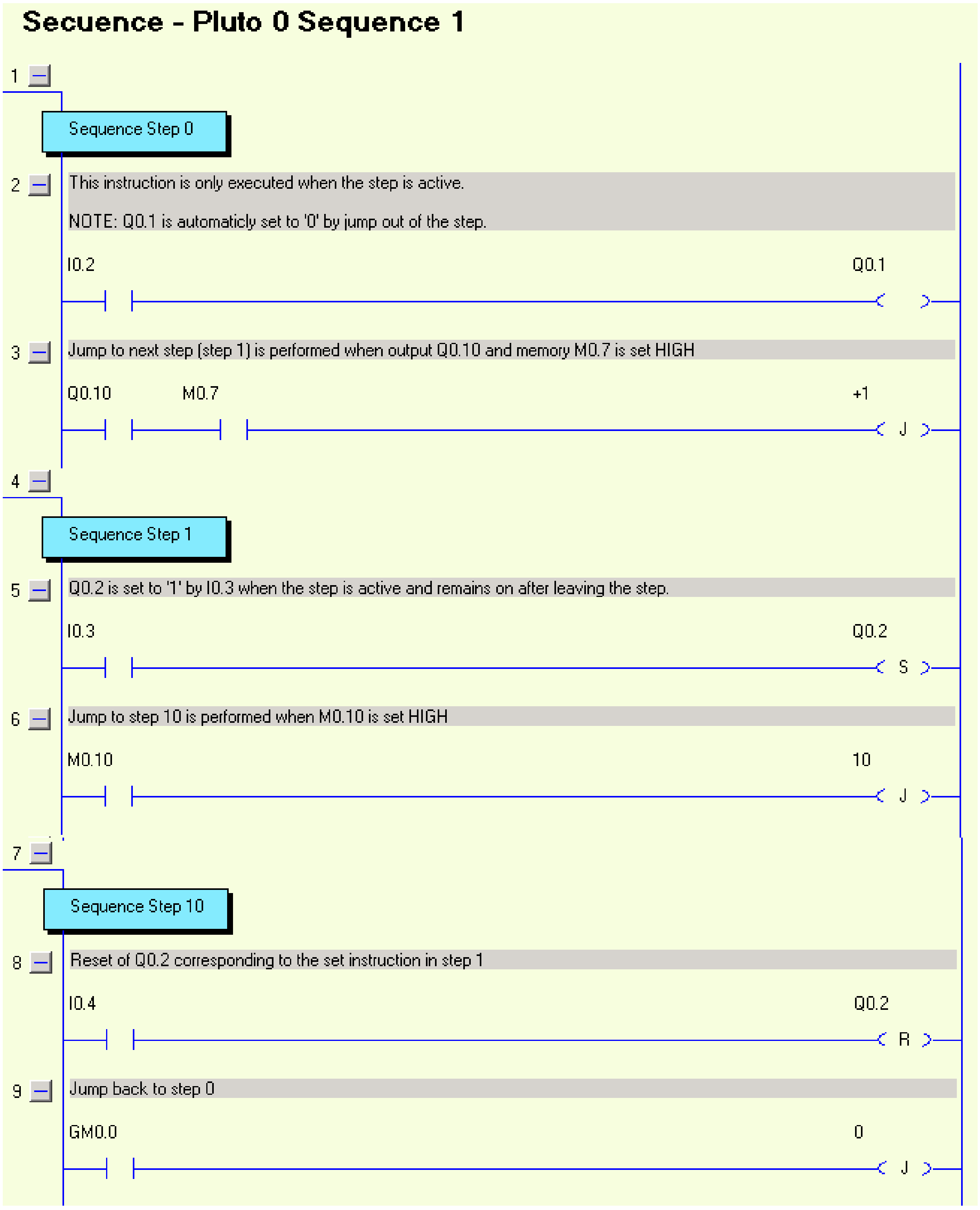
```
S0.1_00
Q0.1 = I0.2
J(+1) = Q0.10*M0.7
```
⇔ Pluto 0, sequence 1, step 0:
Q0.1 is operated by I0.2
Jump to the next step (step 1) when output

Q0.10 and M0.7 is HIGH

```
S0.1_01
S(Q0.2) = I0.3
J(10) = M0.10
```
⇔ Pluto 0, sequence 1, step 1:
Output Q0.2 is set HIGH by I0.3
Jump to step 10 when M0.10 is HIGH.

```
S0.1_10
R(Q0.2) = I0.4
J(0) = GM0.0
```
⇔ Pluto 0, sequence 1, step 10:
Output Q0.2 is set LOW by I0.3
Jump to step 0 when GM0.0 is HIGH.

The equivalence in ladder



Secuence - Pluto 0 Sequence 1

1 Sequence Step 0

2 This instruction is only executed when the step is active.
NOTE: Q0.1 is automaticly set to '0' by jump out of the step.

3 Jump to next step (step 1) is performed when output Q0.10 and memory M0.7 is set HIGH

4 Sequence Step 1

5 Q0.2 is set to '1' by I0.3 when the step is active and remains on after leaving the step.

6 Jump to step 10 is performed when M0.10 is set HIGH

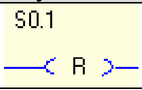
7 Sequence Step 10

8 Reset of Q0.2 corresponding to the set instruction in step 1

9 Jump back to step 0

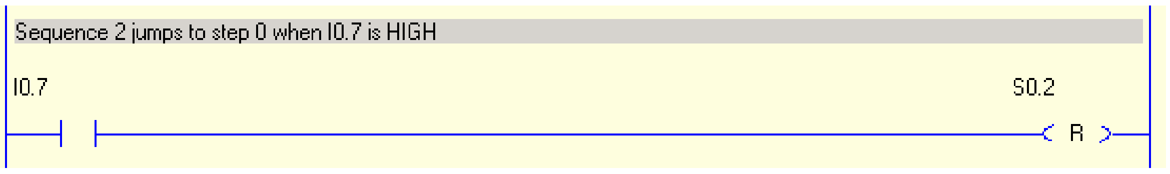
4.3 Reset sequence

It is possible to a sequence with code in another sequences.

Function:	Syntax in text form:	Ladder symbol:
Reset sequence	R(S0.1)	

Function:
Reset forces another sequence to jump to step 0, irrespective of the ordinary jump instructions. The sequence remains in step 0 as long the conditions for the reset instruction is TRUE

Example:



`S0.1_05`
`R(S0.2) = I0.7`
	⇔	In sequence 1 step 5 on PLUTO no: 0. Reset of sequence 2 is demanded when input I0.7 is set HIGH.

NOTE: Reset must be performed from another sequence.

5 Word-operands

5.1 Registers

5.1.1 Addressing

PLUTO has 150 16-bit registers where i.e. calculation results can be stored. The registers have the following number range: -32 768 … +32 767

Register are addressed as shown below:

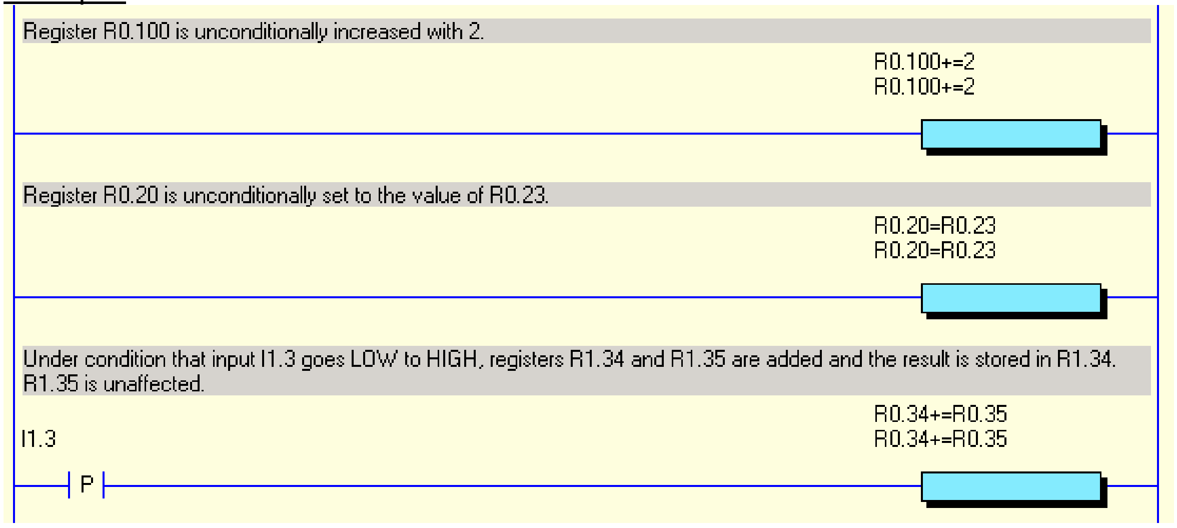
Register:	Syntax:
0-149	R0.0 – R0.149

5.1.2 Operations

Assignment of register

Operation:	Syntax:	
Increment by 1	(R0.100++)	
Decrement by 1	(R0.100--)	
Add constant	(R0.100 += 77)	
Subtract konstant	(R0.100 – = 77)	
Assign with absolute value = 1	(R0.100 = 1)	
Addition with other register (R0.100 = R0.100 + R0.101)	(R0.100 += R0.101)	
Subtract with other register (R0.100 = R0.100 – R0.101)	(R0.100 – = R0.101)	
Assign with other reg. value	(R0.100 = R0.101)	

Example:


Register R0.100 is unconditionally increased with 2.

R0.100+=2
R0.100+=2

Register R0.20 is unconditionally set to the value of R0.23.

R0.20=R0.23
R0.20=R0.23

Under condition that input I1.3 goes LOW to HIGH, registers R1.34 and R1.35 are added and the result is stored in R1.34. R1.35 is unaffected.

R0.34+=R0.35
R0.34+=R0.35

Equivalence in text form:

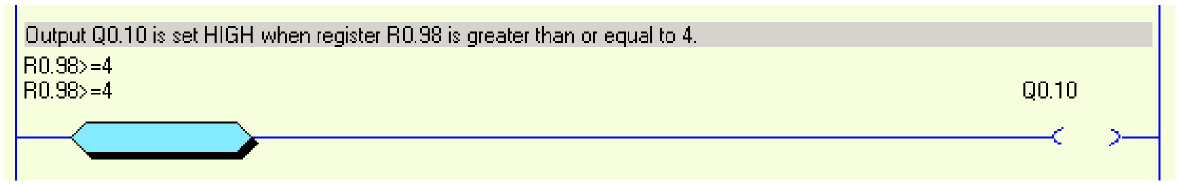
```
(R0.100+=2)
(R0.20=R0.23)
(R1.34+=R1.35) = P(I1.3)
```

Function: By incrimination of register the incrimination stops when the register value reaches the limits (32 767 or -32 768)

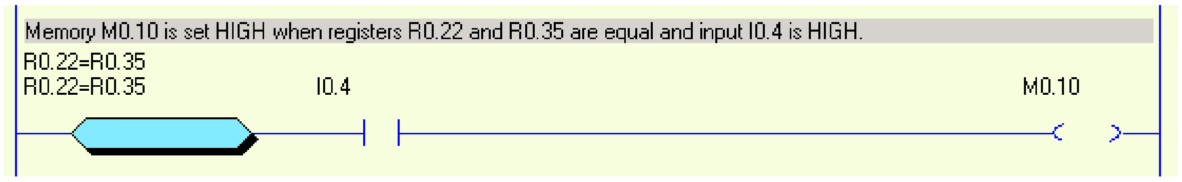
Comparison of register

Comparison	Syntax:
Equal to (constant)	(R0.100=1)
Greater than	(R0.100>1)
Less than	(R0.100<1)
Greater than or Equal to	(R0.100>=1)
Less than or Equal to	(R0.100<=1)
Equal (two registers)	(R0.100=R0.101)
Greater than	(R0.100>R0.101)
Less than	(R0.100< R0.101)
Greater than or Equal to	(R0.100>= R0.101)
Less than or Equal to	(R0.100<= R0.101)

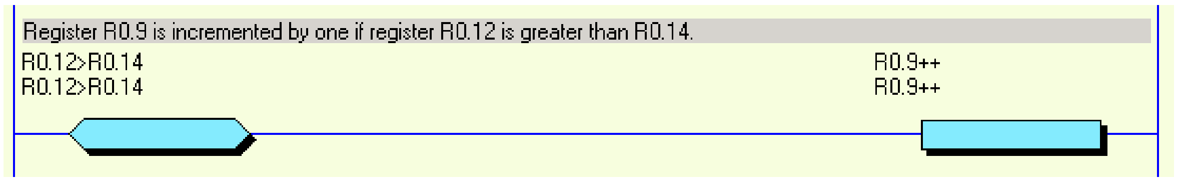
Example:


Output Q0.10 is set HIGH when register R0.98 is greater than or equal to 4.

In text form: `Q0.10 = (R0.98>=4)`


Memory M0.10 is set HIGH when registers R0.22 and R0.35 are equal and input I0.4 is HIGH.

In text form: `M0.10 = (R0.22=R0.35)* I0.4`


Register R0.9 is incremented by one if register R0.12 is greater than R0.14.

In text form: `(R0.9++) = (R0.12>R0.14)`

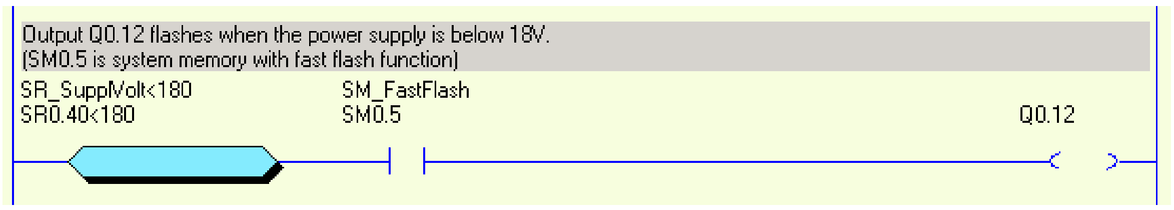
5.1.3 System registers

PLUTO has a set of system registers with different functions.

System registers
Syntax: SR[*unit*].[*no*]

I/O-address	Symbolic name	Function	Type
SR_.10	SR_PlutoDisplay	Display figure	W
SR_.40	SR_SupplVolt	Supply volatge, 10 x Volt	R
SR_.41	*SR_I5_Volt*	*Voltage analogue input I5, 10 x Volt*	*R*
SR_.42	SR_Q16_Current	Current (mA) output no.Q16	R
SR_.43	SR_Q17_Current	Current (mA) output no.Q17	R

<u>Example:</u>


Output Q0.12 flashes when the power supply is below 18V.
(SM0.5 is system memory with fast flash function)

In text form: `Q0.12 = (SR0.40<180) * SM0.5`

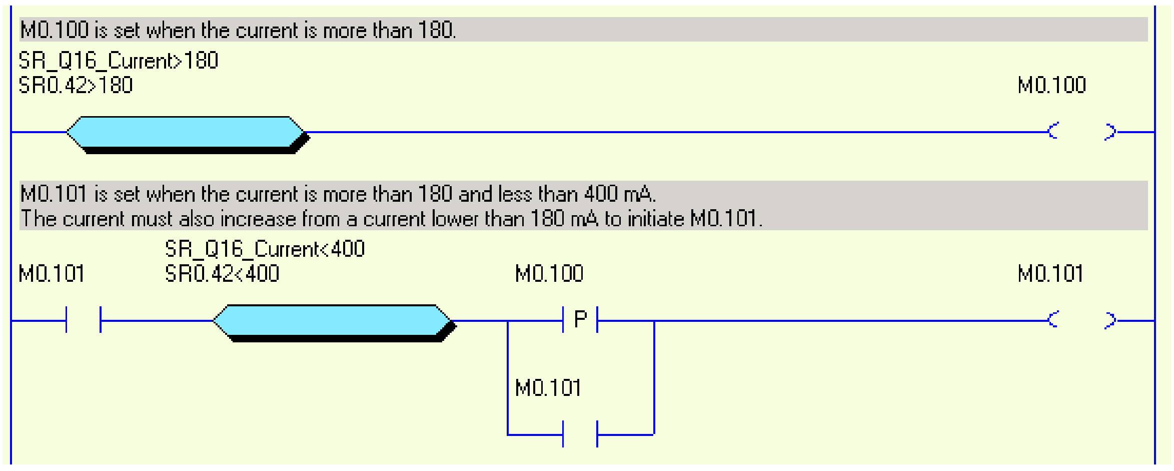
5.2 Use of analogue values

The analogue values are available by reading the system registers SR40…SR43. There are som requirements for the use these functions.

Analogue input I5:
The terminal for input I5 is also an analogue input messuring the voltage at the terminal. In SR41 the value can be read in tenths of volts, (240 = 24.0 volt).
By use in safety applications a 0-value may not be used as safe condition unless it is used in a dynamic monitored way, meaning that the program monitors that the input value changes. This requirement is because value in SR41 will be set to 0 if an internal fault in the system occurs.

Current monitoring of Q16 and Q17:
The output current from Q16 and Q17 is available in SR42 and SR43, the value represents mA. The function is intended for monitoring the current in a muting lamp, but other usage is not excluded. As the hardware for measuring the current is not fully redundant the values must be used in a dynamic way. For example if a current to a muting lamp shall be monitored the program must be written so that the change of current by switching the input on and off is observed.

Example:


M0.100 is set when the current is more than 180.

M0.101 is set when the current is more than 180 and less than 400 mA.
The current must also increase from a current lower than 180 mA to initiate M0.101.

In text form:

```
M0.100 = (SR0.42>180)
M0.101 = M0.100 * (SR0.42<400) * (P(M0.100) + M0.101)
```

6 Program declaration

(Mainly applicable for programming in text form)

In the beginning of the program file different declaration is done. These declarations describe the hardware environment for the Pluto unit.

For more information about the function of the different hardware options see the 'Operating instructions, Hardware'

6.1 Identity, station number

Each unit unit must have a station number 0-31.
It is also possible to connect an external identifier cicuit containing a unique 12 figure hexadecimal number.
These two settings are declared as:

! id_pluto:[*stn.number*]=[*identifier number*]

If identifier is not connected the system will accept this if the identifier numder is declared as 000000000000 (12 zero).

<u>Example:</u>

! id_pluto:00=ffff00007FA3 ⇔ The Pluto-unit is given station number 0 and an identifier with number ffff00007FA3 must be connected to the unit.

! id_pluto:23=000000000000 ⇔ The Pluto-unit is given station number 23
and the unit shall run without identifier.

6.2 Declaration of program code

Since it is possible two have program code for several units stored in one unit it must be declared to which Pluto unit a code part belongs to.

Syntax:

! pgm_pluto:*[station no.]*

6.3 Declaration of I/O

All inputs and the non failsafe output Q10…17 must be declared since they can de used in differen ways. The tables below show the differen options.

InputsSyntax: ! I[no],[pulse type],[switch 1],[switch 2]
Example: ! I0.5,c_pulse,non_inv,no_filt

Inputs	Pulse types (Dynamic sign.)	Switch 1 (optional)	Switch 2 (optional)
I_.0 - I_.17	a_pulse b_pulse c_pulse	non_inv	no_filt
I_.0 - I_.17	static∗)		no_filt

∗) I_.10-I_17,**static** does not fulfill cat. 4 according to EN954-1, as stand alone input

Dynamic outputs Syntax: ! Q[*no],[pulse typ]*
Example: ! Q0.10,a_pulse

Inputs	Pulse types
Q_.10 – Q_.17	a_pulse, b_pulse, c_pulse

Non failsafe-outputs Syntax: ! Q[*no],static*
Exempel: ! Q0.10,static

Inputs	Pulse types
Q_.10 – Q_.17	static

Special function, Illuminated push button
Syntax: ! IQ[*no],[pulse type]*
Example: ! IQ0.12,a_pulse

Inputs	Pulse types
IQ_.10 – IQ_.17	a_pulse, b_pulse, c_pulse

Example:

```
! i0.1,a_pulse             ; Input is suplied with dynamic A signal via inverter.
! i0.2,a_pulse,non_inv     ; Input is suplied with dynamic A signal.
! i0.3,static              ; Input is suplied with +24V.

! q0.10,a_pulse    ; Output generates dynamic A signal for supply of inputs.
! q0.11,static     ; Input is suplied with dynamic A signal.
```

Special function, Cross talk

Syntax: ! I[*no],[pulse type],x_talk*
Exempel: ! I0.12,a_pulse,x_talk

Inputs	Pulse types
I_.10 - I_.17	a_pulse b_pulse c_pulse

6.4 Symbolic names

The variables can also be named with a symbolic name which can make a program easier to understand. In Pluto Manager it is declared on a separate page, see Pluto Manager manual.

By programming in text form it is declared. Where in the code the declaration is made depends on whether it is a global or local variable. Global variables I_._, Q_.0...4 and GM_.0..11 are declared before the program code for the first Pluto since the variable can be used in all Pluto:s. Local variables are named in the beginning of the of the program code for the corresponding Pluto, after the I/O declarations. See example in

Example:

```
! I0.0=MuteSensor1                  ; Symbolic names global variables
! Q0.1=MuteSensor2
! GM0.1=MuteSensor2

! Q0.14=IndReset                    ; Symbolic names local variables
! M0.0=MutingActive
! R0.0=Counter1
```

7 Program example

This program example is the program for the installation example showed in 'Operating instruction, Hardware'

```
$name Example, manual


! id_pluto:00=000034AD4AE1

! pgm_pluto:00


! q0.10,a_pulse                ; Dynamic output A

! i0.00,static                 ; Muting sensor 1
! i0.01,a_pulse,non_inv        ; Muting sensor 2
! i0.02,a_pulse,non_inv        ; Test Contactors
! i0.12,a_pulse                ; Emergency stop PB
! i0.13,a_pulse                ; JSL Lightbeam
! iq0.14,a_pulse               ; Reset with indicator

;*********************************************

s0.0_0                         ; Main sequence start

q0.2 = i0.12 * (i0.13 + m0.0) * ( (p(i0.14) * i0.02) + q0.2)
q0.3 = q0.2
                               ; All safety outputs active when Emergency stop(I0.12)
                               ; and JSL(I.13) or muting(M0.0) are active.
                               ; Reset(I0.14) and Test(I0.02) are also needed in the
                               ; start condition.

q0.14 = /q0.2                  ; Reset indication active when outputs not active


;*********************************************

s0.1_0                         ; Muting Sequence
j(+1)=/i0.00*/i0.01*(SR0.43<100)     ; Start condition: both sensors not active

s0.1_1
q0.17 = i0.00 * i0.01 * i0.13
j(+1) = q0.17 * (SR0.43<100)   ; Muting start when both sensors and JSL active
s0.1_2
m0.0                           ; M0.0, Memory muting active
q0.17                          ; Indicator muting activ
j(0) = /i0.00 + /i0.01         ; Muting stopped by either sensor not active
```