

# System Description

# ABB Procontic b

## Programmable Control System

## Applications

Order number  
GATS 1311 05 R2001

BBC Procontic b4 Anweisungseditor DEMO

Satz	Wort	Bef.	SF	OPK	GN. KM	Symbol	Langtext
0000							
	0000	?	E	00, 01		ENDSL	Endschalter links
	0001	&	E	00, 02		ENDSR	Endschalter rechts
	0002	=	M'	00, 00			
0001	0004	?	E	00, 03		GEBO	Geber oben
	0005	&	E	00, 04		GEBO	Geber B
	0006	=	M'	00, 02			
0002	0008	?	M'	00, 00			
	0010	=S	M	00, 00		MERK0	Merker0
0003	0011	?	M'	00, 02			
	0013	=R	M	00, 00		MERK0	
0004	0014	?	M	00, 00			
	0015	=	M'	00, 01			

## Regulations

### Regulations concerning the setting up of installations

Apart from the basic "Regulations for the setting up of power units" VDE\* 0100 and for "The rating of creepage paths and air gaps" VDE 0110 the regulations "The equipment of power units with electrical components" VDE 0160 in connection with VDE 0660, part 500, have to be taken into due consideration. Further attention has to be paid to VDE 0113 in case of the control of working and processing machines. If operating elements are to be arranged near shock-hazard parts with protection against electrical shock, VDE 0106, part 100, is relevant.

The user has to ensure that the units as well as the associated components have to be installed according to these regulations. Respectively valid safety regulations, e.g. regulation for the prevention of accidents and the law concerning technical working material, are valid for machines and units connected as well.

ABB Procontic units have been built according to VDE regulation 0160. The protection against direct touching as demanded by chapter 5.5.1 of this VDE regulation has to be satisfied by the user, e.g. at installing of switch cabinet.

ABB Procontic units have been designed for operation according to insulation class A of VDE 0110. If considerable pollution is expected during operations, the units have to be installed in housings of the respective kind of protection.

\* VDE stands for "Association of German Electrical Engineers".

Note: Please observe the national regulations for the installation of electrical equipments, which are valid in your country.

ABB Schalt- und Steuerungstechnik GmbH

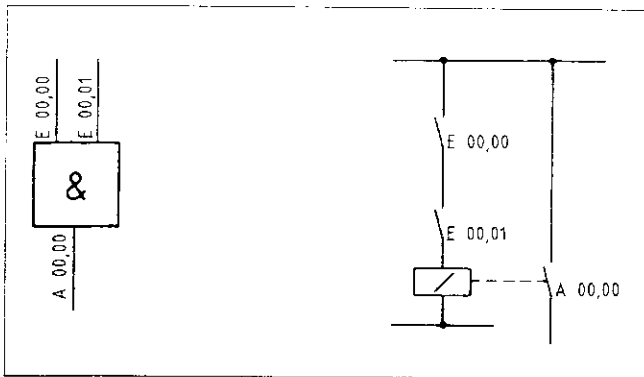
# Table of contents

1	Logic functions	1-1	6.	Registers	6-1
1.1	AND function	1-1	6.1	Shift registers	6-1
1.2	OR function	1-1	6.1.1	Shift registers, general	6-1
1.3	Combination of logic functions	1-1	6.1.2	Shift register, implementation	6-1
1.4	NOT function	1-2	6.2	Stack registers	6-1
1.5	NAND function	1-2	6.2.1	Stack registers, general	6-1
1.6	NOR function	1-2	6.2.2	Stack register, implementation	6-2
1.7	AND before OR function	1-3	7	Code converters	7-1
1.8	OR before AND function	1-3	7.1	Code converter, BCD into 1-out-of-10	7-1
1.9	Exclusive OR function	1-4	7.2	Code converter, 1-out-of-10 into BCD	7-1
1.1	Equivalence gate	1-4	7.3	Code converter, BCD into 1-out-of-8	7-1
1.11	Driving several outputs	1-4	7.4	Code converter, 1-out-of-8 into BCD	7-2
2	Memory functions	2-1	7.5	Code converter, BCD into 7-segment	7-2
2.1	Memory circuit for outputs	2-1	7.6	Code converter, decimal into 7-segment	7-2
2.1.1	– with dominant reset	2-1	8	Comparator	8-1
2.1.2	– with dominant set	2-1	8.1	Comparator for equivalenc	8-1
2.2.	Buffered function flags	2-2	8.2	Greater-less-equal comparator	8-1
2.2.1	– with dominant reset	2-2	9	Signalling controls	9-1
2.2.2	– with dominant set	2-2	9.1	Signalling with continuous light	9-1
2.7	Function flag as flags	2-2	9.2	New value signalling with single flashing light	9-1
3	Edge evaluation	3-1	9.3	New value signalling with double flashing light	9-2
3.1	Edge evaluator, positive going edge	3-1	9.4	First and new value signalling with single acknowledgment	9-3
3.2	Edge evaluator, negative going edge	3-1	9.5	First value signalling with double acknowledgment	9-4
3.3	Wiper relay	3-2	10	Registers	10-1
3.4	Latching relay	3-2	10.1	Register as sequence control	10-1
4	Timer functions	4-1	10.2	Sequential connection of two register chains	10-2
4.1	0-1 on delay	4-1	11	Channel selection	11-1
4.2	0-1 off delay	4-1	12	Programming examples for the fast timer 07 ZG 84	12-1
4.3	Double delay with 1 timer	4-1	12.1	Assignments for the programming examples	12-1
4.4	Universal delay stage	4-2	12.2	Load setpoint into the setpoint register	12-1
4.5	Universal delay stage with premature termination	4-2	12.3	Loading the pre-tip value	12-2
4.6	Oscillator with 1 timer	4-2	12.4	Read actual value and display on output unit	12-2
4.7	Oscillator with 2 timers	4-2	12.5	Polling of the actual value and loading of the setpoint with this actual value	12-3
4.8	Oscillator with frequency divider	4-3	12.6	Polling of the comparator bits and display on the output unit	12-4
5	Counters	5-1	12.7	Input of the signals VK2, FR, SS and R in the program	12-4
5.1	Counters, general	5-1			
5.2	Decimal counter	5-1			
5.2.1	Decimal counter, up	5-1			
5.2.2	Decimal counter, down	5-1			
5.2.3	Decimal counter, up/down 2 decades	5-2			
5.3	Binary counters	5-3			
5.3.1	Binary counters, general	5-3			
5.3.2	Binary counter, up	5-3			
5.3.3	Binary counter, down	5-4			
5.3.4	Binary counter, up/down	5-4			
5.4	BCD counters	5-5			
5.4.1	BCD counters, general	5-5			
5.4.2	BCD counter, up	5-5			
5.4.3	BCD counter, down	5-5			
5.4.4	BCD counter, up/down – 1 decade	5-6			
5.4.5	BCD counter, up/down – 2 decades	5-6			
5.4.6	BCD counter, setting	5-7			



# 1 Logic functions

## 1.1 AND function

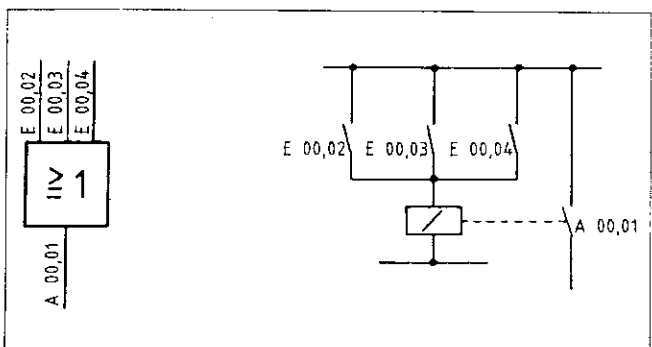


```
! E 00,00 & E 00,01 = A 00,00
Start AND Allocation
Input 1="1"? Input 2="1"? Output="1"
```

In this example, the inputs E 00,00 and E 00,01 are interrogated for the signal state "1" and ANDed together. The result of this interrogation is allocated to the output A 00,00.

E 00,00	E 00,01	A 00,00
0	0	0
0	1	0
1	0	0
1	1	1

## 1.2 OR function

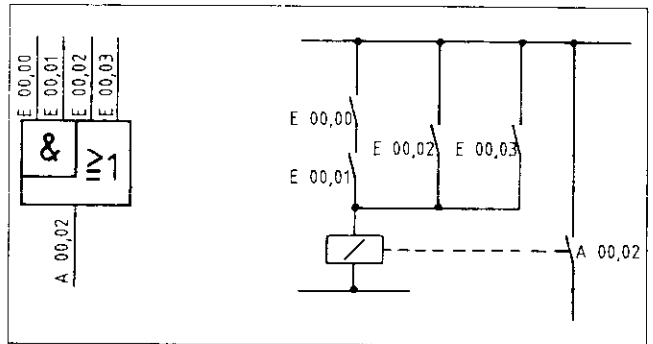


```
! E 00,02 / E 00,03 / E 00,04
Start OR OR
Input 1="1"? Input 2="1"? Input 3="1"
= A 00,01
Allocation
Output 3="1"?
```

In this example, the inputs E 00,02 to E 00,04 are interrogated for signal state "1" and ORed together. The result of this interrogation is allocated to output A 00,01.

E 00,02	E 00,03	E 00,04	A 00,00
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

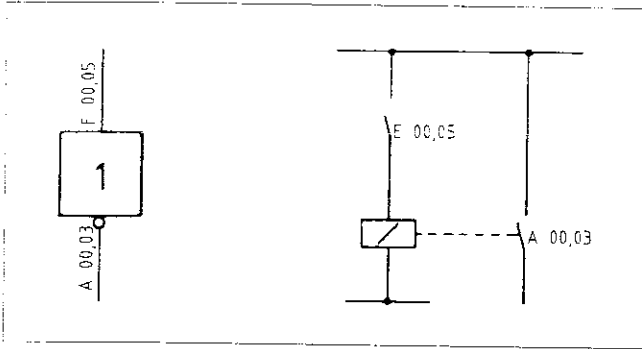
## 1.3 Combination of logic functions



```
! E 00,00 & E 00,01 / E 00,02
Start AND OR
Input 1="1"? Input 2="1"? Input 3="1"?
/ E 00,03 = A 00,02
OR Allocation
Input 4="1"? Output="1"
```

E 00,00	E 00,01	E 00,02	E 00,03	A 00,02
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

## 1.4 NOT function



```
! E 00,05 =N A 00,03
Start      Allocation
Input 1="1"? Output="0"
```

E 00,05	A 00,03
1	0
0	1

All interrogations previously described provided the status "1" for a 1-signal and status "0" for a 0-signal at the inputs. These interrogation operations are called "normally open contacts" in a relay circuit.

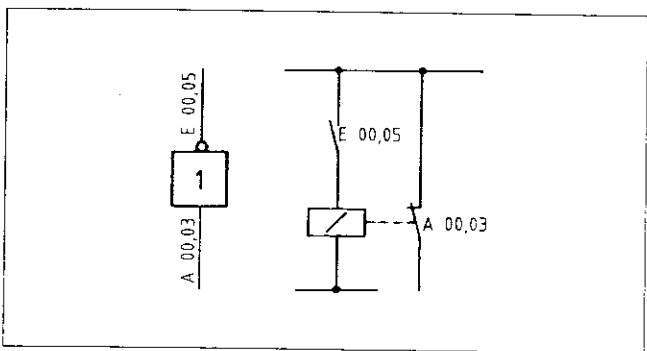
All modern controls, however, require interrogation facilities which supply the status "1" for a 0-signal at the inputs or the status "0" for a 1-signal at the inputs. These interrogation operations correspond to the function of a "normally closed contact" in a relay circuit. In the programming language, the symbol "N" provides this functions.

Instead of

```
! E 00,05 =N A 00,03
```

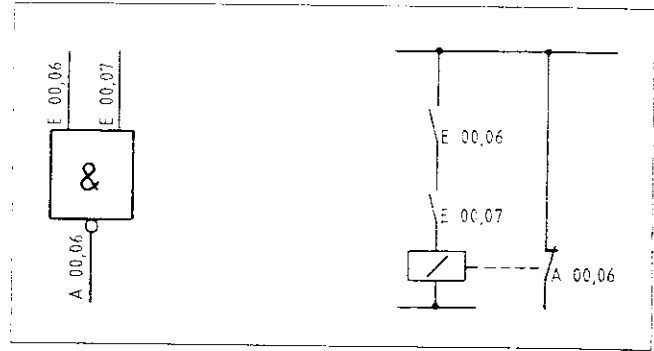
it is possible to write

```
!N E 00,05 = A 00,03
```



without changing the meaning (De Morgan).

## 1.5 NAND function



```
! E 00,06 & E 00,07 =N A 00,06
Start      AND      Allocation
Input 1="1"? Input 2="1"? Output="0"
```

In this example, the inputs E 00,06 and E 00,07 are interrogated for signal state "1" and ANDed together. The result of this interrogation is negated and allocated to output A 00,06.

E 00,06	E 00,07	A 00,06
0	0	1
0	1	1
1	0	1
1	1	0

Instead of

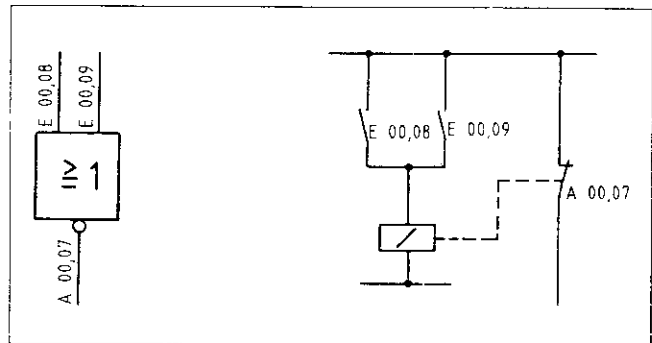
```
! E 00,06 & E 00,07 =N A 00,06
```

it is possible to write

```
!N E 00,06 /N E 00,07 = A 00,06
```

without changing the meaning (De Morgan).

## 1.6 NOR function



```
! E 00,08 / E 00,09 =N A 00,07
Start      OR      Allocation
Input 1="1"? Input 2="1"? Output="0"
```

In this example, the inputs E 00,08 and E 00,09 are interrogated for signal state "1" and ORed together. The result of this interrogation is negated and allocated to A 00,07.

E 00,08	E 00,09	A 00,07
0	0	1
0	1	0
1	0	0
1	1	0

Instead of

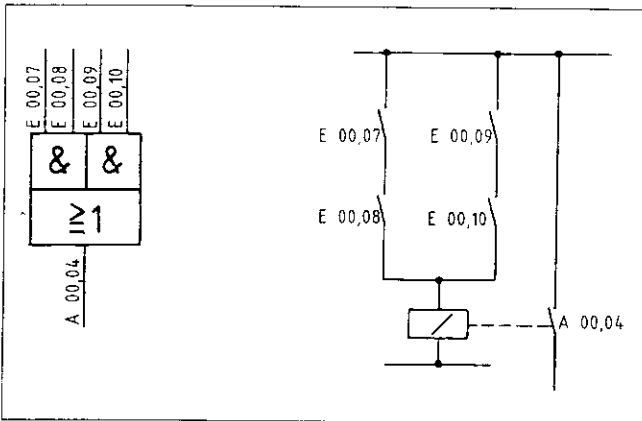
```
! E 00,08 / E 00,09 =N A 00,07
```

it is possible to write

```
!N E 00,08 &N E 00,09 = A 00,07
```

without changing the meaning (De Morgan).

### 1.7 AND before OR function



```
! E 00,07 & E 00,08 / E 00,09
Start AND OR
Input 1="1"? Input 2="1"? Input 3="1"?

& E 00,10 = A 00,04
AND Allocation
Input 4="1"? Output="1"
```

This AND before OR function can be written directly in the programming language (without conversion).

This example is written as follows:

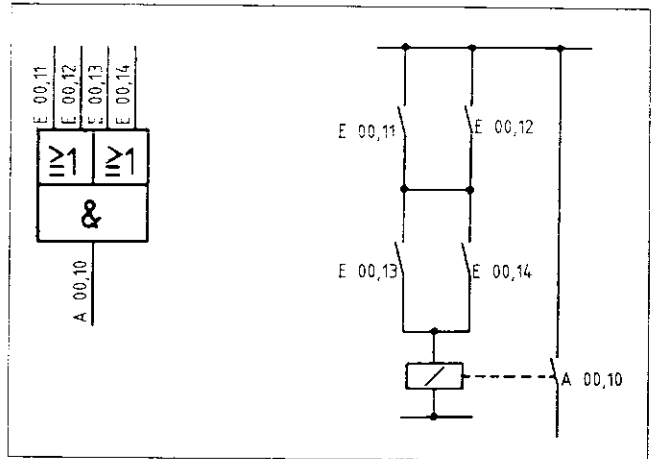
Boolean algebra

```
E 00,07 ^ E 00,08 v E 00,09
^ E 00,10 = A 00,04
```

Programming language

```
! E 00,07 & E 00,08 / E 00,09
& E 00,10 = A 00,04
```

### 1.8 OR before AND function

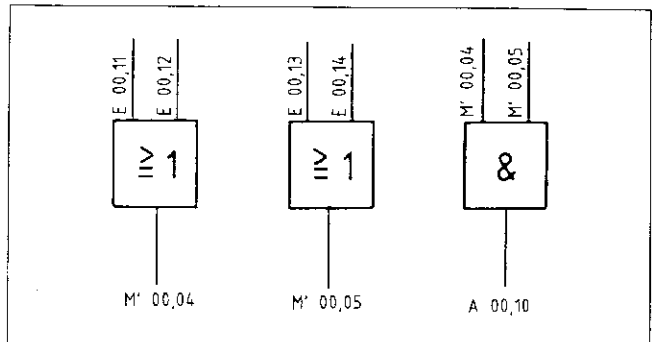


```
! (E 00,11 / E 00,12) & (E 00,13
/ E 00,14) = A 00,10
```

If the "AND before OR" rule is applied, the above logic operations cannot be written without parenthesis. As parenthesis are not provided in the PROCONTIC language, this logic operation must be converted. There are two possibilities for this:

- Conversion into auxiliary flags (clearer - more words)
- Multiplication (only for simple combinations)
- Conversion in accordance with De Morgan

a) Conversion into auxiliary flags (clearer - more words)



Programming

```
! E 00,11 / E 00,12 = M'00,04
! E 00,13 / E 00,14 = M'00,05
! M'00,04 & M'00,05 = A 00,10
```

b) Multiplication (only for simple combinations)

```
! (E 00,11 / E 00,12) & (E 00,13
/ E 00,14) = A 00,10
```

becomes

```
! E 00,11 & E 00,13 / E 00,11
& E 00,14 / E 00,12 & E 00,13
/ E 00,12 & E 00,14 = A 00,10
```

This version can now be programmed directly (but is useful only for small logical explanations).

Output A 00,14 is set to "1" if input E 00,01 is "1" and input E 00,15 is "1" or input E 00,01 is "0" and input E 00,15 is "0".

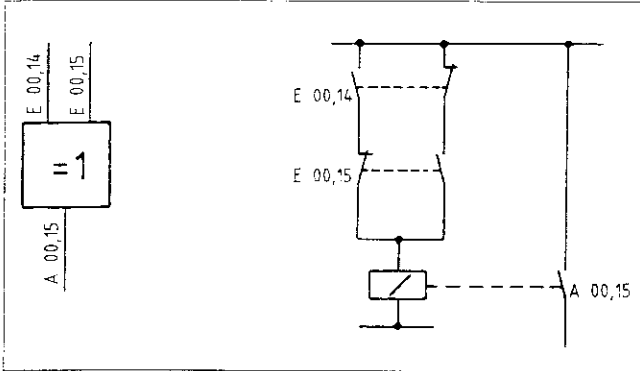
### c) Conversion in accordance with De Morgan

```

!N E 00,11    &N E 00,12    /N E 00,13
&N E 00,14    =N A 00,10
    
```

E 00,01	E 00,15	A 00,14
0	0	1
1	0	0
0	1	0
1	1	1

## 1.9 Exclusive OR function



```

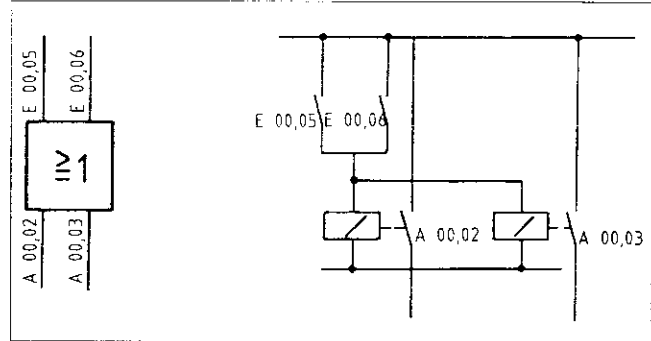
! E 00,14    &N E 00,15    /N E 00,14
Start        AND            OR
Input 1="1"? Input 2="0"?  Input 1="0"?

& E 00,15    = A 00,15
AND          Allocation
Input 2="1"? Output="1"
    
```

The output A 00,15 is set to "1" is either input E 00,14 is "1" and input E 00,15 is "0" or input E 00,14 is "0" and input E 00,15 is "1".

E 00,14	E 00,15	A 00,15
0	0	0
1	0	1
0	1	1
1	1	0

## 1.11 Driving several outputs



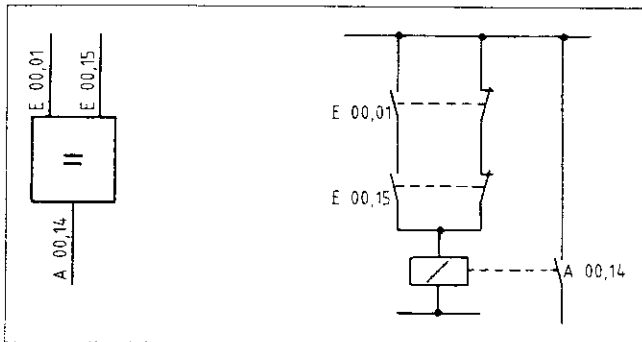
```

! E 00,05    / E 00,06    = A 00,02
Start        OR            Allocation 1
Input 1="1"? Input 2="1"?  Output 1="1"?

= A 00,03
Allocation 2
Output 2="1"
    
```

Any number of allocations can be written, either in buffered or non-buffered form and with or without negation. The allocations always refer to the result of the signal interrogation which preceded the first allocation symbol.

## 1.10 Equivalence gate



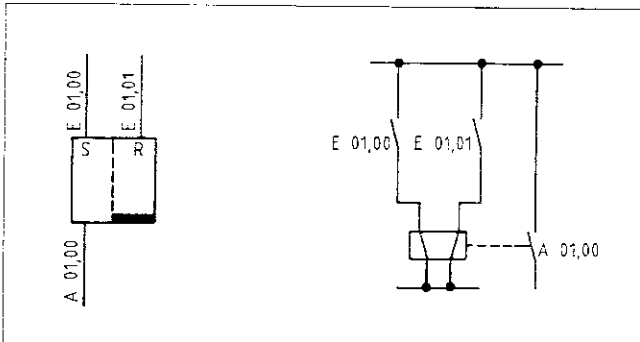
```

! E 00,01    & E 00,15    /N E 00,01
Start        AND            OR
Input 1="1"? Input 2="1"?  Input 1="0"?

&N E 00,15    = A 00,14
AND          Allocation
Input 2="0"?  Output="1"
    
```

## 2 Memory functions

### 2.1 Memory circuit for outputs



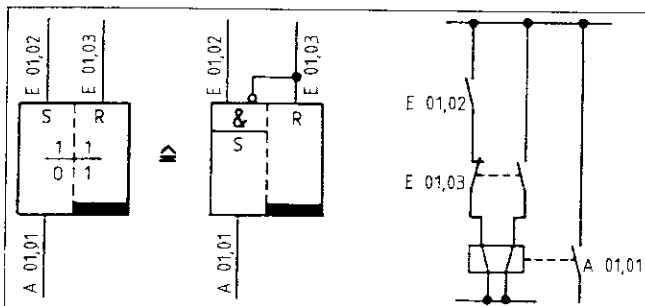
```
! E 01,00 =S A 01,00
Start      Allocation
Input 1="1"? Reset output memory
```

```
! E 01,01 =R A 01,00
Start      Allocation
Input 2="1"? Set output memory
```

Output A 01,00 is set if input E 01,00 is "1". If input E 01,00 then returns to "0" the memory remains set until the input E 01,01 becomes "1". The memory is then reset.

E 01,00	E 01,01	A 01,00
0	0	Initial state
1	0	1
1	1	Depending on memory construction
0	1	0

#### 2.1.1 Memory circuit for outputs – with dominant reset



```
! E 01,02 &N E 01,03 =S A 01,01
Start      AND      Allocation
Input 1="1"? Input 2="0"? Set output memory
```

If output memory is set if input E 01,02 is "1" and input E 01,03 is "0".

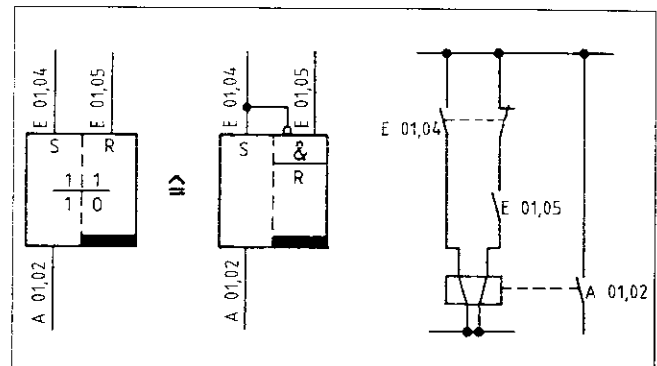
```
! E 01,03 =R A 01,01
Start      Allocation
Input 2="1"? Reset output memory
```

The output memory is reset if input E 01,03 is "1".

If both inputs (E 01,02 and E 01,03) are "1", the memory is reset.

E 01,02	E 01,03	A 01,01
0	0	Initial state
0	1	0
1	0	1
1	1	0

### 2.1.2 Memory circuit for outputs – with dominant set



```
! E 01,04 =S A 01,02
Start      Allocation
Input 1="1"? Reset output memory
```

Output A 01,02 is set if input E 01,04 is "1".

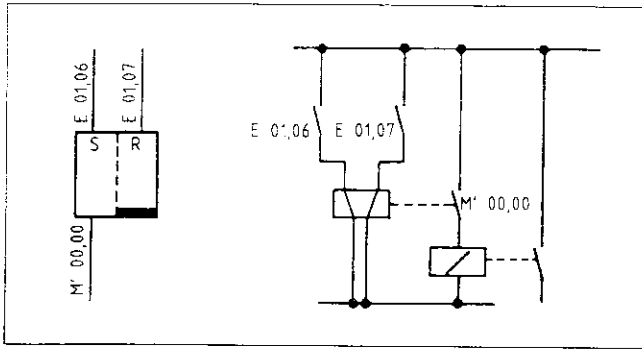
```
! E 01,05 &N E 01,04 =R A 01,02
Start      UND      Allocation
Input 2="1"? Input 1="0"? Set output memory
```

If input E 01,05 is "1" and input E 01,04 is "0", output A 01,02 is reset.

If both inputs (E 01,04 and E 01,05) are "1", the memory is set.

E 01,04	E 01,05	A 01,02
0	0	Initial state
1	0	1
0	1	0
1	1	1

## 2.2 Buffered function flags



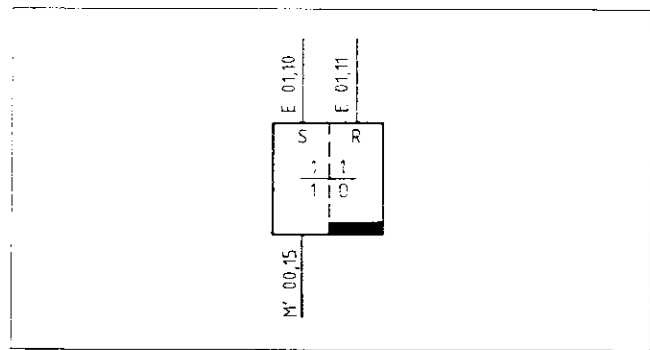
```
! E 01,06 =S M'00,00
Start Allocation
Input 1="1"? Set function flag

! E 01,07 =R M'00,00
Start Allocation
Input 2="1"? Reset function flag
```

If input E 01,06 is "1", the functional flag M'00,00 is set. The function flag is reset with a "1" signal at input E 01,07.

E 01,06	E 01,07	M'00,00
0	0	Initial state
1	0	1
1	1	Depending on memory construction
0	1	0

## 2.2.2 Buffered function flag – with dominant set

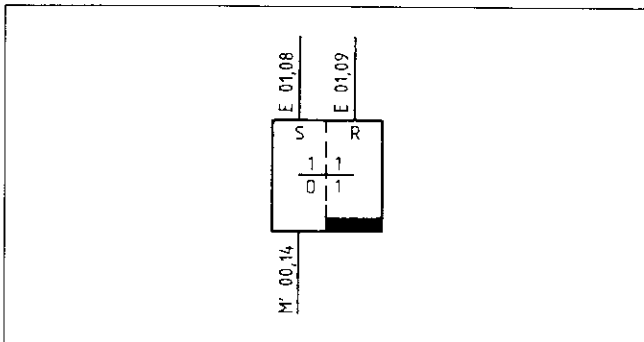


```
! E 01,10 =R M'00,15 with domi-
! E 01,11 =S M'00,15 nant set
```

In the case of a functional flag with dominant set condition, the reset condition must be programmed before the set condition. Reliable dominance is achieved only if the reset and set conditions follow each other directly. If this is not possible, interlocking must be carried out as for the memory outputs.

E 01,10	E 01,11	M'00,15
0	0	Initial state
1	0	1
0	1	0
1	1	1

## 2.2.1 Buffered function flag – with dominant reset

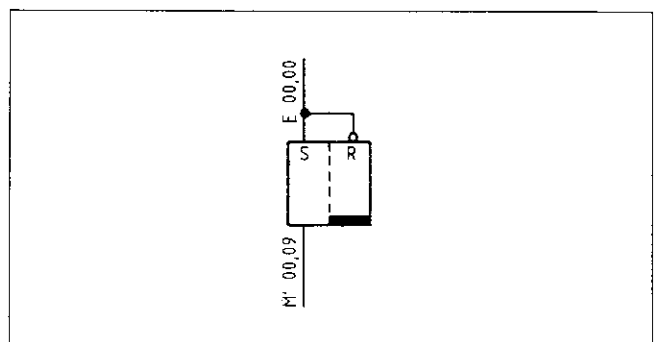


```
! E 01,08 =S M'00,14 with domi-
! E 01,09 =R M'00,14 nant set
```

In the case of a function flag with dominant reset, the set conditions must be written before the reset condition. Reliable dominance is achieved only if the set and reset conditions follows each other directly. If this is not possible, interlocking must be carried out as for memory outputs.

E 01,08	E 01,09	M'00,14
0	0	Initial state
1	0	1
0	1	0
1	1	0

## 2.3 Function flag as flags



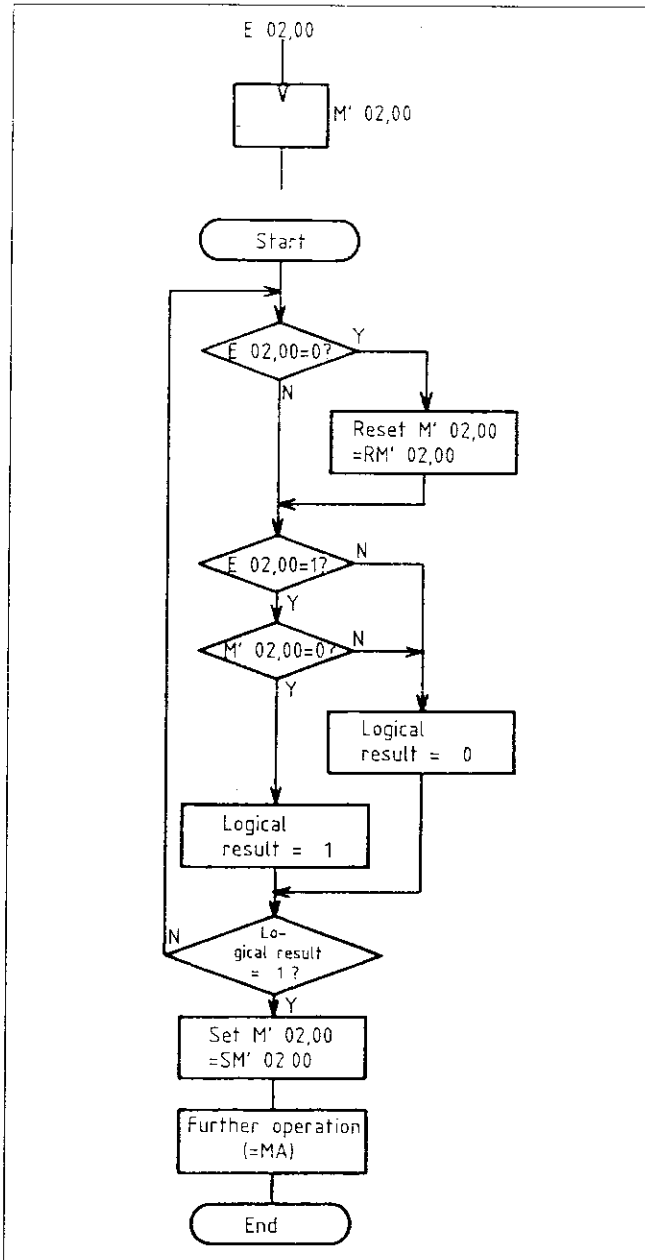
```
! E 00,00 = M'00,09
Start Allocation
Input ="1"? Function flag ="1"
Input ="0"? Function flag ="0"
```

The function flag is in the "1" state as long as a "1" is present at the input. It remains set until a "0" is allocated to the input (if allocation is carried out only once per program, this lasts for at least one program cycle). In the case of multiple allocations in the program, the function flag can change its state several times during a cycle.

E 00,00	M'00,09
0	0
1	1

### 3 Edge evaluation

#### 3.1 Edge evaluator, positive going edge



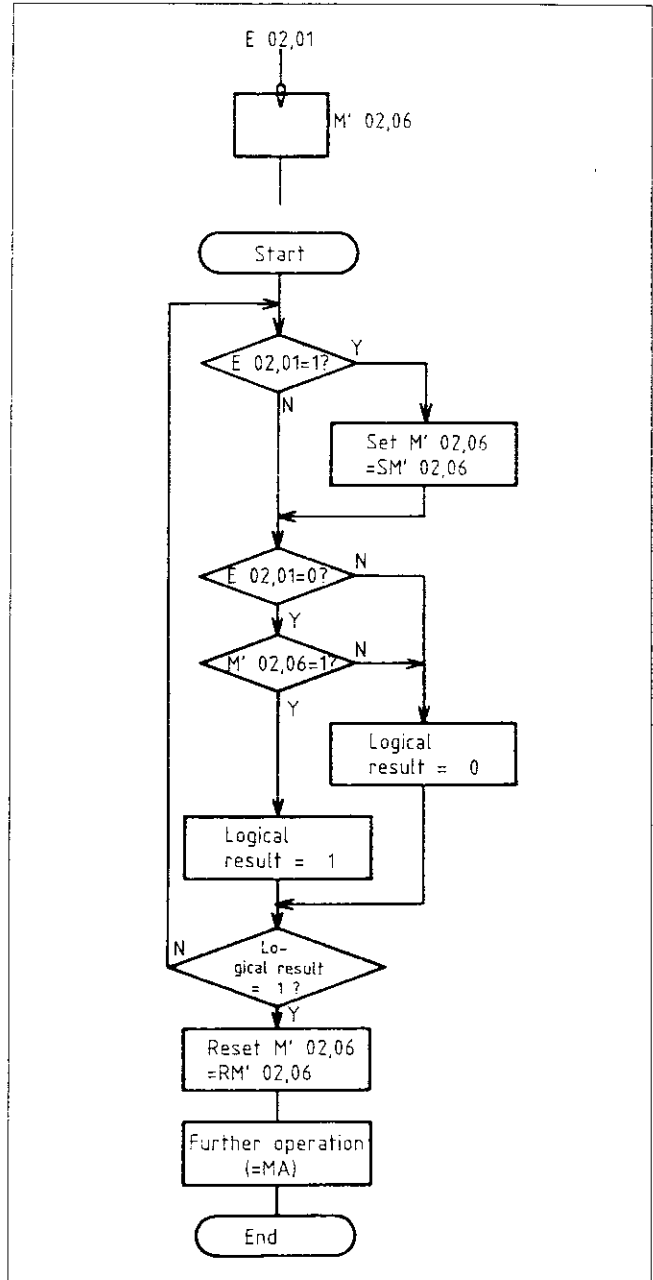
```

!N E 02,00      =R M'02,00
! E 02,00      &N M'02,00   =S M'02,00
(= MA          )
:
:
(! ME          )
    
```

If input E 02,00 is "0", the function flag (edge flag) M'02,00 is reset. If the signal state at the input changes from "0" to "1", then the AND condition (! E 02,00 &N M'02,00) is fulfilled, but flag M'02,00 is still reset. This allocation still sets the function flag M'02,00 and ensures that the AND condition (! E 02,00 &N M'02,00) is no fulfilled in the next cycle.

The allocation (=S M'02,00) is thus "1" for exactly one cycle after the positive-going edge was detected. This signal can be utilized by multiple allocation(= MA) in order to process a subroutine in only one cycle after the signal transition.

#### 3.2 Edge evaluator, negative going edge



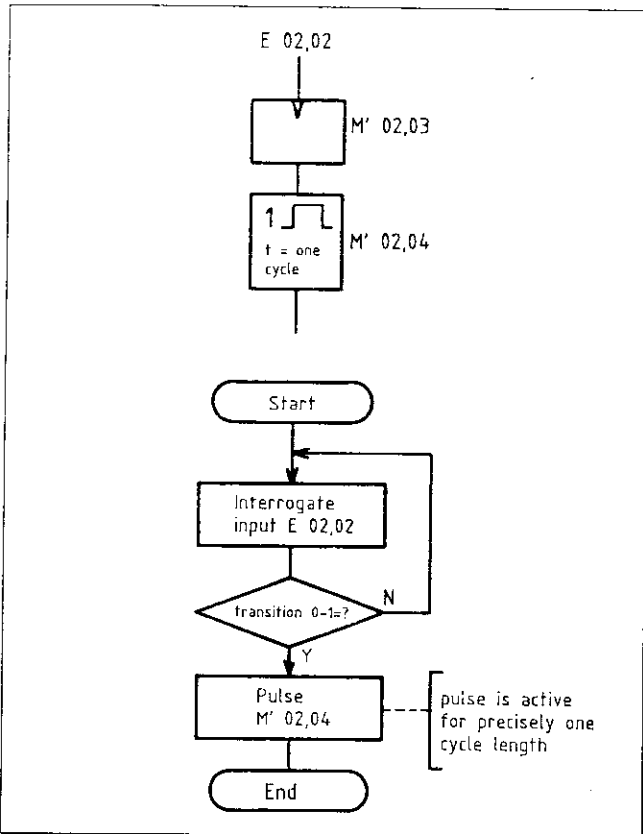
```

: E 02,01      =S M'02,06
!N E 02,01     & M'02,06    =R M'02,06
(= MA          )
:
:
(! ME          )
    
```

If the input is "1", the function flag (edge flag) M'02,06 is set. If the input changes from "1" to "0", the AND condition is fulfilled, but the function flag M'02,06 is still set. This allocation resets the function flag M'02,06 and ensures that the AND condition (!N E 02,01 & M'02,06) cannot be fulfilled in the next cycle.

The allocation (=R M'02,06) is thus "1" for precisely one cycle after detection of the negative-going edge. This signal can be utilized by means of a multiple allocation (= MA) in order to process a subroutine in exactly one cycle after the signal transition.

### 3.3 Wiper relay

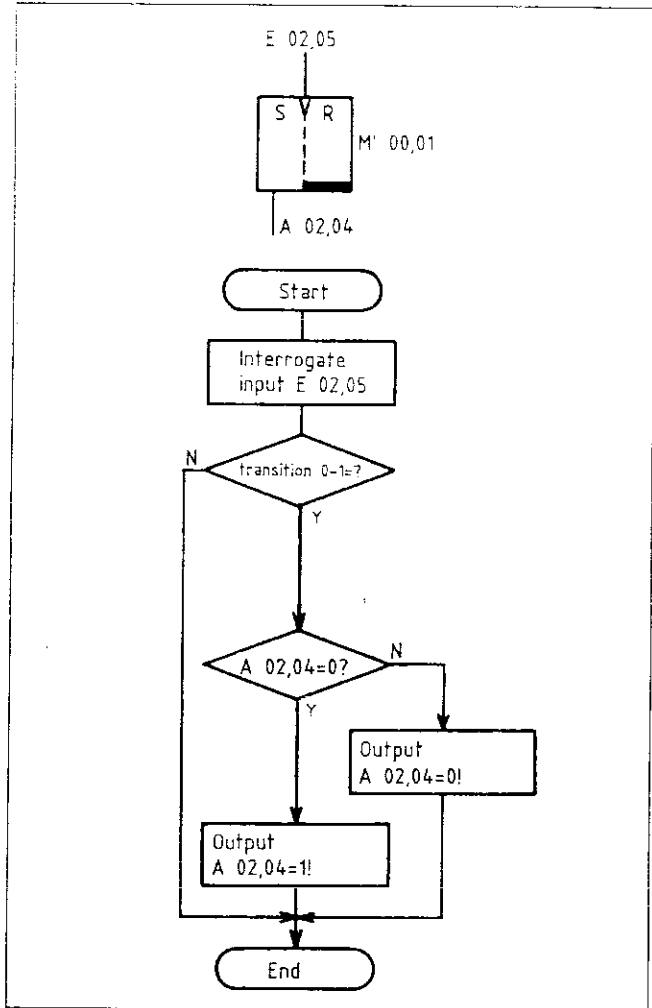


```

!N E 02,02      =R M'02,03
! E 02,02      &N M'02,03      =S M'02,03
= M'02,04
  
```

After the "0-1" transition has been detected (E 02,02), of function flag M'02,04 is set for the length of precisely one cycle.

### 3.4 Latching relay



```

!N E 02,05      =S M'00,01
! E 02,05      & M'00,01      = MA
=R M'00,01
!N A 02,04      = A 02,04
! ME
  
```

After detection of the "0-1" transition, the subroutine is executed precisely once and output A 02,04 is inverted. This status of the output remains stored until the subroutine is processed again with a new "0-1" transition (E 02,05).

Alternative (without subroutine):

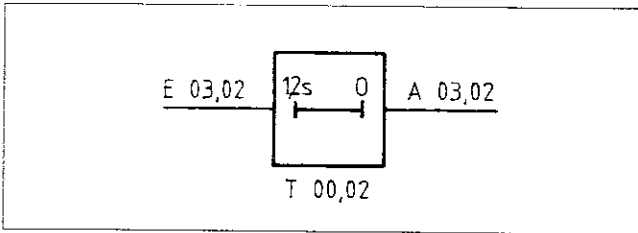
```

! E 02,05      &N M'00,01      =S A 02,04
!N E 02,05      & A 02,04      =S M'00,01
! E 02,05      & M'00,01      =R A 02,04
!N E 02,05      &N A 02,04      =R M'00,01
  
```

One output is set or reset by interlocking with a memory (function flag).

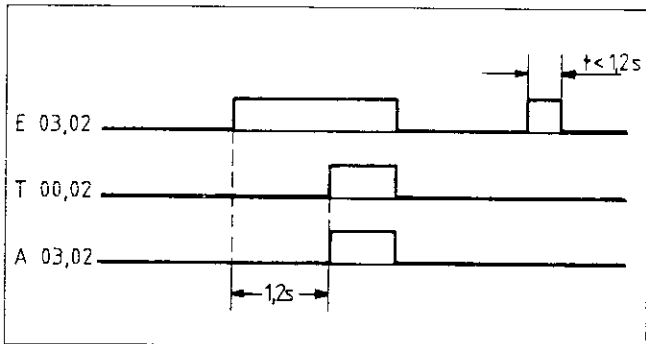
## 4 Timer functions

### 4.1 0→1 on delay



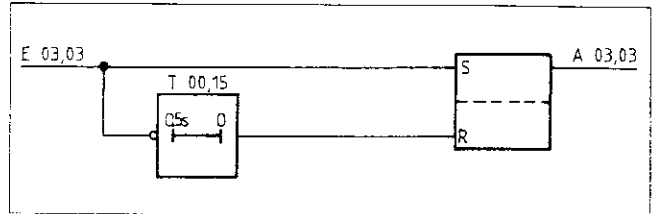
```
! E 03,02 = T 00,02
! T 00,02 = A 03,02
```

In the case of the 0→1 on delay, the time starts running when the input E 03,02 switches to "1". The output A 03,02 is set when the time T 00,02 has elapsed if the input signal is still present. This is done by interrogating the elapsed timer.



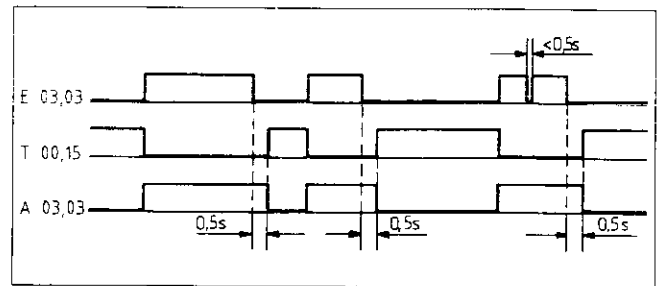
As a safety measure, the output is interrogated in addition to the time interrogation.

### Alternative (with memory)

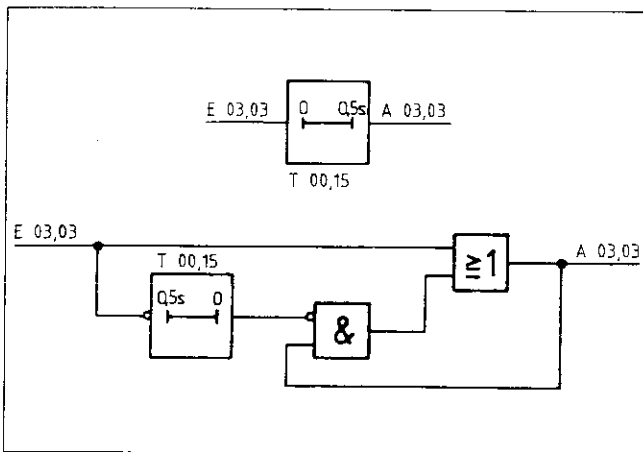


```
! E 03,03 =S A 03,03
!N E 03,03 = T 00,15
! T 00,15 =R A 03,03
```

In this case, the output A 03,03 is set if the input E 03,03 is "1". When the input changes to "0", the timer T 00,15 is started. The output is reset when the timer has elapsed.



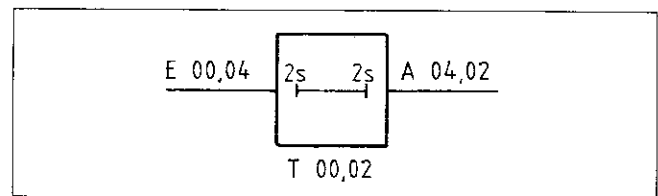
### 4.2 1→0 off delay



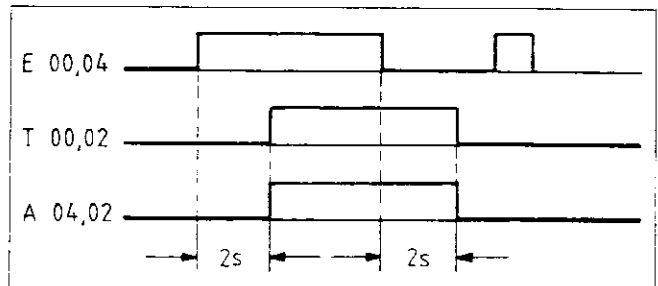
```
!N E 03,03 = T 00,15
! E 03,03 /N T 00,15 & A 03,03
= A 03,03
```

In this version of a 1→0 off delay, the time starts only when the input E 03,03 switches to "0". The output A 03,03 is set only if the input E 03,03 is "1" or if the time T 00,15 is running.

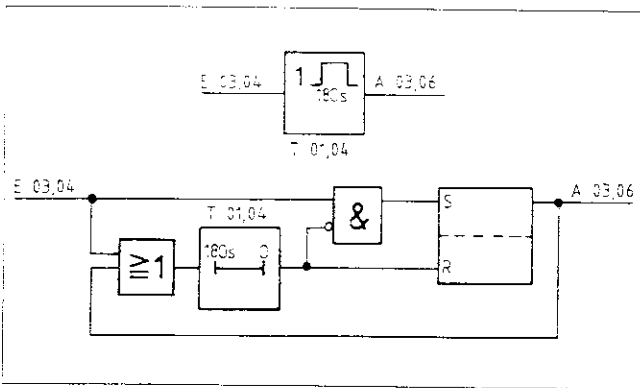
### 4.3 Double delay with 1 timer



```
! E 00,04 &N A 04,02 /N E 00,04
& A 04,02 = T 00,02
! T 00,02 = MA
!N A 04,02 = A 04,02
! ME
```



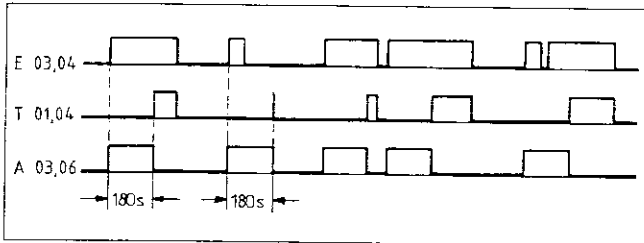
#### 4.4 Universal delay stage



```

! E 03,04   &N T 01,04   =S A 03,06
! A 03,06   / E 03,04   =  T 01,04
! T 01,04   =R A 03,06
    
```

If the input E 03,04 is "1", the output A 03,06 (or function flag) is set. The input signal starts the timer T 01,04. When the timer has elapsed, the output is reset and the set condition is blocked. In the case of a short input signal, the blocker holds itself via the output (the function flag) until the delay stage has elapsed. In the case of a long input signal, the set condition for the output is blocked via the BLOCKING AND circuit.

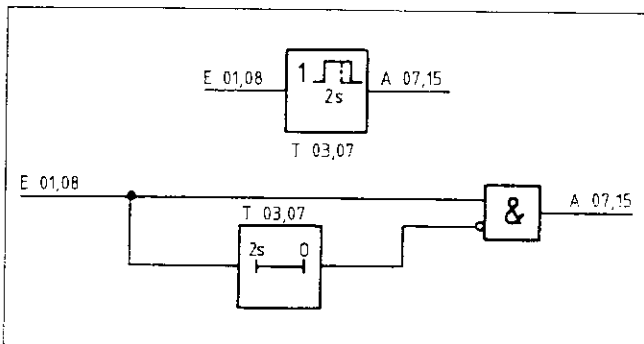


For purely internal use, the delay stage can be simplified as follows:

```

! E 03,04   =S M'00,04
! M'00,04   / E 03,04   =  T 01,04
! T 01,04   =R A 03,06
    
```

#### 4.5 Universal delay stage with premature termination

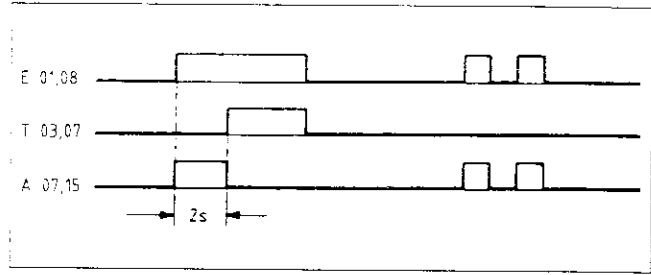


```

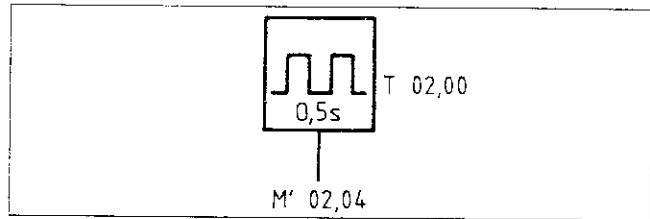
! E 01,08   &N T 03,07   = A 07,15
! E 01,08   =  T 03,07
    
```

If input E 01,08 is "1", a "1" signal is allocated to the output A 07,15. At the same time, the timer T 03,07 is started. After the preset time has elapsed, the timing circuit switches off the output via the BLOCKING AND gate. If the

input signal is terminated during the delay time, a "0" is immediately allocated to the output.



#### 4.6 Oscillator with 1 timer



```

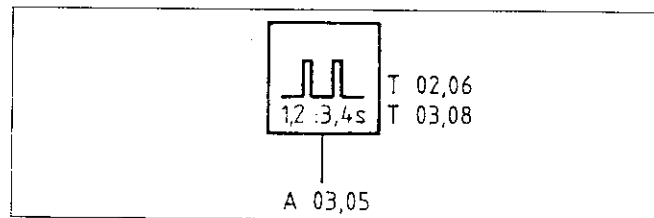
!N T 02,00   =  T 02,00   = M'00,15
!N M'00,15   =  MA
!N M'02,04   = M'02,04
! ME
    
```

If the timer T 02,00 has not yet been started or has already elapsed, the time is started and a function flag M'00,15 is simultaneously set (function flag "Time started"). When the time has elapsed (!N M'00,15), the subroutine is processed. After this, the activation condition for the timing circuit is no longer fulfilled (Allocation of a "0" for T02,00). This resets the output signal from T02,00, and the activation condition (!N T 02,00) is fulfilled in the next cycle.

The subroutine is thus opened once after each time operation and the function flag within the subroutine M'02,04 is inverted.

The function flag M'02,04 can be interrogated as an oscillator.

#### 4.7 Oscillator with 2 timers

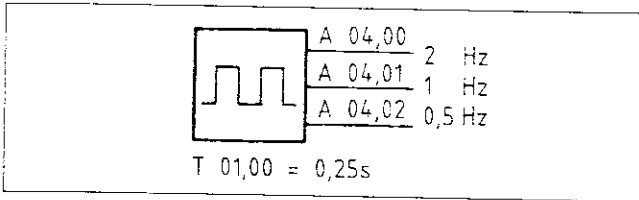


```

!N A 03,05   =  T 02,06
! T 02,06   =S A 03,05
! A 03,05   =  T 03,08
! T 03,08   =R A 03,05
    
```

The first time T 02,06 is started, when the output is not set. When the first time T 02,06 has elapsed, the output A 03,05 is set. When the output is set, the second time T 03,08 is started. When this time has elapsed, the output is reset.

#### 4.8 Oscillator with frequency divider



```

!N T 01.00 = T 01.00
= M'01,00 = MA Pulse generation
!N M'01,00 = MA
!N A 04,00 = A 04,00 2 Hz
= MA
!N A 04,01 = A 04,01 1 Hz
= MA
!N A 04,02 = A 04,02 0,5 Hz
! ME
  
```

By means of nested subroutines, this sequence ensures that the next slower frequency changes its signal state only with a "1" signal from the faster frequency. The oscillator starts with a "1" signal at all frequencies.

**Notice**

The frequency divider can be driven with any required clock signal.



# 5 Counters

## 5.1 Counters, general

Software counters in cyclic controls have a relative low counting frequency. The counting frequency depends on the cycle time and the input delays of the signals.

With a program length of 1 k words, the cycle time of a PROCONTIC b with the central processor unit 07 ZE 82 or 07 ZE 84 (only bit program) is typically 2,5 ms.

Taking into account the input delay (typically 8 ms), the maximum counting frequency is

- for 4 k program max. 60 Hz
- for 8 k program max. 30 Hz.

The counting frequency can be increased by removing the input delay circuits, but this also increases the sensitivity to interference.

For rapid counting operations, it is recommended to use the counter module 07 ZG 84.

Use of counters:

Decimal counters:

For internal counting operations with a large number of required values or interrogations.

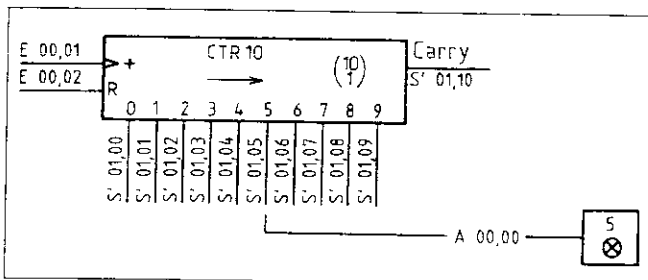
Binary counter:

For internal counting operations with a few required values or interrogations.

BCD counters:

For external set point output or internal actual value display.

### 5.2.1 Decimal counter, up



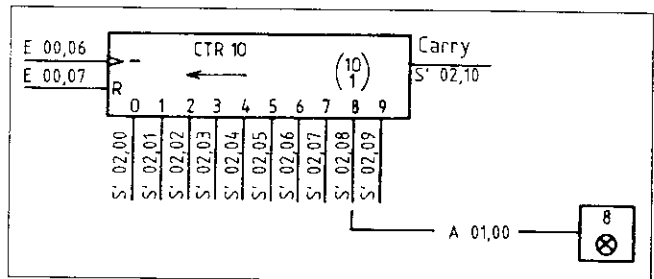
```

!N E 00,01      =R M'00,01
! E 00,01      &N M'00,01      "0→1" edge
= MA           =S M'00,01      evaluation
! S'01,09      = S'01,10      Forward carry
! S'01,08      = S'01,09
! S'01,07      = S'01,08
! S'01,06      = S'01,07
! S'01,05      = S'01,06
! S'01,04      = S'01,05
! S'01,03      = S'01,04
! S'01,02      = S'01,03
! S'01,01      = S'01,02
! S'01,00      = S'01,01
! ME
:
:
:
! E 00,02      = S'01,00      Counter reset
! S'01,05      = A 00,00      Interrogation
    
```

Each time a signal edge is detected, the register chain (starting with S'01,00) is incremented by one (the sequence always being from back to front). The reset signal (E 00,02) resets the register chain.

At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see section 5.2.3 decimal counter, up/down).

### 5.2.2 Decimal counter, down



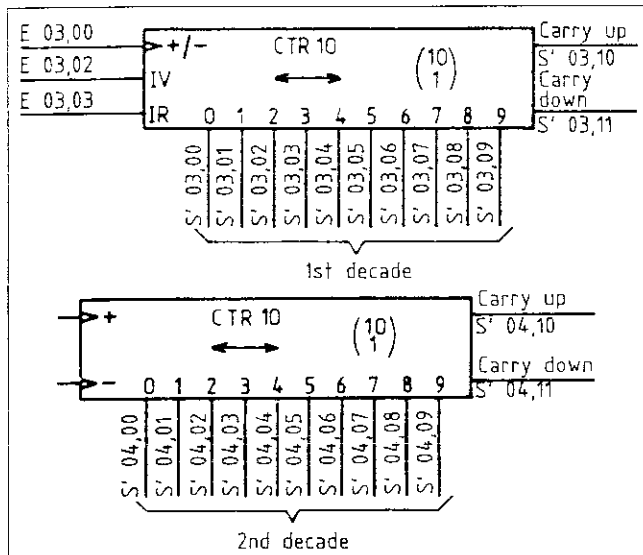
```

!N E 00,06      =R M'00,02
! E 00,06      &N M'00,02      "0→1" edge
= MA           =S M'00,02      evaluation
! S'02,00      = S'02,10      Forward carry
! S'02,01      = S'02,00
! S'02,02      = S'02,01
! S'02,03      = S'02,02
! S'02,04      = S'02,03
! S'02,05      = S'02,04
! S'02,06      = S'02,05
! S'02,07      = S'02,06
! S'02,08      = S'02,07
! S'02,09      = S'02,08
! ME
:
:
:
! E 00,07      = S'02,00      Counter reset
! S'02,08      = A 01,00      Evaluation
! E 00,05      = S'02,05      Setting the
                                counter to 5
    
```

Each time a signal edge is detected, the register chain (starting with S'02,10) is decremented by one (always written from front to back). S'02,10 is used for carry signals. At the start of counting, the register chain is set to S'02,00. The counter can be set to a specific position by allocation without interrogation of the initial state (! E 00,05 = S'02,05). This signal must be omitted at the start of a counting operation.

At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation routine must be modified (see section 5.2.3 decimal counter, up/down).

### 5.2.3 Decimal counter, up/down - 2 decades



```

!N E 03,00 =R M'04,00
! E 03,00 &N M'04,00
& E 03,02 = MA           Edge evaluation,
=S M'04,00                    Flag counting up

! S'03,09 = S'03,10
! S'03,08 = S'03,09
! S'03,07 = S'03,08
! S'03,06 = S'03,07
! S'03,05 = S'03,06      1st decade,
! S'03,04 = S'03,05      counting upwards
! S'03,03 = S'03,04
! S'03,02 = S'03,03
! S'03,01 = S'03,02
! S'03,00 = S'03,01

! S'03,10 = MA
= S'03,00
! S'04,09 = S'04,10
! S'04,08 = S'04,09
! S'04,07 = S'04,08
! S'04,06 = S'04,07
! S'04,05 = S'04,06      2nd decade,
! S'04,04 = S'04,05      counting upwards
! S'04,03 = S'04,04
! S'04,02 = S'04,03
! S'04,01 = S'04,02
! S'04,00 = S'04,01
! ME

```

```

: E 03,00 &N M'04,00
& E 03,02 = MA           Edge evaluation,
=S M'04,00                    Flag counting down

! S'03,00 = S'03,11
! S'03,01 = S'03,00
! S'03,02 = S'03,01
! S'03,03 = S'03,02
! S'03,04 = S'03,03
! S'03,05 = S'03,04      1st decade,
! S'03,06 = S'03,05      counting downwards
! S'03,07 = S'03,06
! S'03,08 = S'03,07
! S'03,09 = S'03,08

! S'03,11 = MA           Setting 1st decade
= S'03,09                    to 9
! S'04,00 = S'04,11
! S'04,01 = S'04,00
! S'04,02 = S'04,01
! S'04,03 = S'04,02
! S'04,04 = S'04,03      2nd decade,
! S'04,05 = S'04,04      counting downwards
! S'04,06 = S'04,05
! S'04,07 = S'04,06
! S'04,08 = S'04,07
! S'04,09 = S'04,08
! ME

```

A positive-going edge is detected and allocated to the appropriate counter section depending on preselection of up (E 03,02) or down (E 03,03).

When a "1" signal is connected to a preselect input, the counter input (E 03,00) must be "0". If this cannot be guaranteed, the edge evaluation must be modified as shown below. Reversal of the direction of counting can also be carried out via one input.

```

: E 03,02 = up
:N E 03,02 = down

```

Edge evaluation for connection of preselection with "1" signal at the counter input.

```

!N E 03,00 =S M'04,00      Edge evalua-
!N E 03,02 =R M'04,00      tion, up
& E 03,00 & M'04,00
= MA =R M'04,00

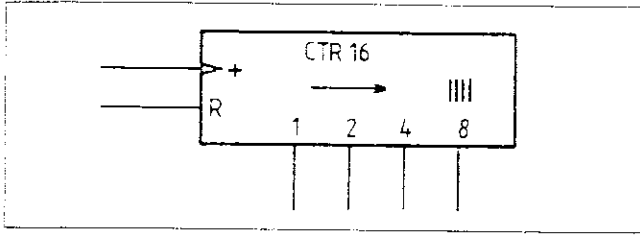
!N E 03,00 =S M'04,01      Edge evalua-
!N E 03,03 =R M'04,01      tion, down
& E 03,00 & M'04,01
= MA =R M'04,01

```

If this type of edge evaluation is used, switching of the counting direction **must** be carried out with **2 inputs**.

### 5.3 Binary counters

#### 5.3.1 Binary counters, general



Code conversion – function diagram

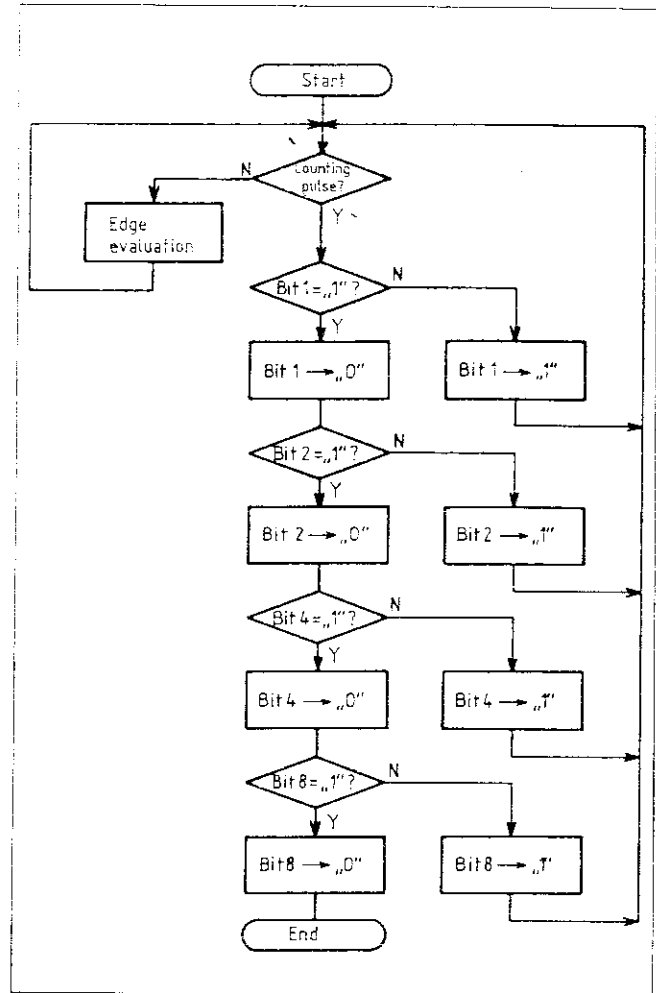
Decimal	Binary Bit values			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

The following rules can be derived from this table:

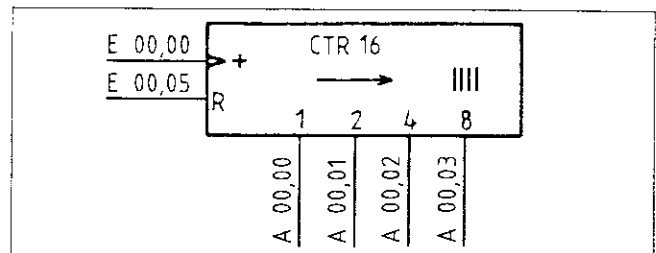
- The lowest bit (1) of the counter changes its signal state ("0 → 1", "1 → 0") with each counting pulse.
- Each time bit position 1 changes from "1 → 0", the state of bit position 2 is changed.
- Each time bit position 2 changes from "1 → 0", bit position 4 is changed.
- Each time bit position 4 changes from "1 → 0", bit position 8 is changed.

The general rule is thus:

Thus change in a bit position from "1 → 0" changes the state of the next higher bit.



#### 5.3.2 Binary counter, up



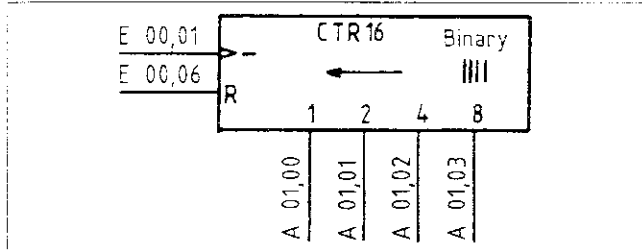
By means of nested subroutines, this routine ensure that all bits – starting from the least significant bit and extending to the most significant bit – change their values sequentially with each counting pulse until a "1" is written into one position. The states of all bits with a higher significance then remain unchanged.

!N E 00,00	=R M'00,00	
! E 00,00	&N M'00,00	Edge evaluation
= MA	=S M'00,00	
! A 00,00	=N A 00,00	Bit 1
= MA		
! A 00,01	=N A 00,01	Bit 2
= MA		
: A 00,02	=N A 00,02	Bit 4
= MA		
! A 00,03	=N A 00,03	Bit 8
: ME		Carry
! E 00,05	=R A 00,00	
=R A 00,01	=R A 00,02	Reset
=R A 00,03		

For purely internal use of the counter, the outputs (A) can be replaced by function flags (M). The binary counter can be extended as required.

At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the evaluation must be modified (see section 5.2.3 decimal counter, up/down).

### 5.3.3 Binary counter, down



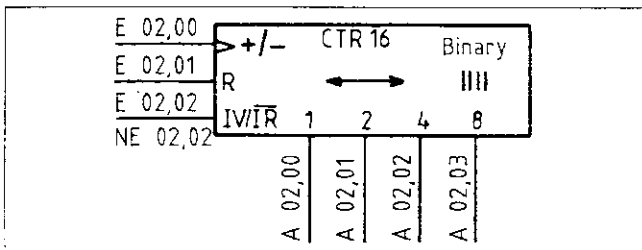
By means of nested subroutines, this routine ensures that all bits – starting from the least significant and extending to the most significant bit – change their signal states sequentially with each counting pulse until a "0" is written into one position. The states of all bits with higher significance then remain unchanged.

```

!N E 00,01 =R M'00,15
! E 00,01 &N M'00,15 Edge evaluation
= MA =S M'00,15
!N A 01,00 = A 01,00 Bit 1
= MA
!N A 01,01 = A 01,01 Bit 2
= MA
!N A 01,02 = A 01,02 Bit 4
= MA
!N A 01,03 = A 01,03 Bit 8
! ME Carry
! E 00,06 =R A 01,00
=R A 01,01 =R A 01,02 Reset
=R A 01,03
  
```

At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see section 5.2.3 decimal counter, up/down).

### 5.3.4 Binary counter, up/down



```

!N E 02,00 =R M'02,04
! E 02,00 &N M'02,04 Edge evaluation,
&N M'02,04 =S M'02,04 up
= MA
! A 02,00 =N A 02,00
= MA
! A 02,01 =N A 02,01 counting upwards
= MA
! A 02,02 =N A 02,02
= MA
! A 02,03 =N A 02,03
! ME
! E 02,00 &N E 02,02
&N M'02,04 =S M'02,04 Edge evaluation,
=S M'02,04 = MA down
!N A 02,00 = A 02,00
= MA
!N A 02,01 = A 02,01 counting downwards
= MA
!N A 02,02 = A 02,02
= MA
!N A 02,03 = A 02,03
! ME
! E 02,01 =R A 02,00
=R A 02,01 =R A 02,02 Reset
=R A 02,03
  
```

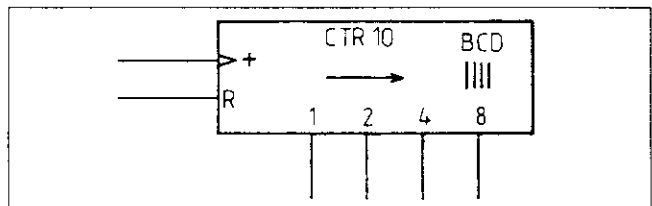
At the start of the program and when the counting direction selection signal is connected, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see decimal counter, up/down). In this case, the following selection of the counting direction with one input is not possible.

```

! E 02,02 = up
!N E 02,02 = down
  
```

## 5.4 BCD counters

### 5.4.1 BCD counters, general



Code conversion – function diagram

Decimal	BCD Bit value			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
0	0	0	0	0

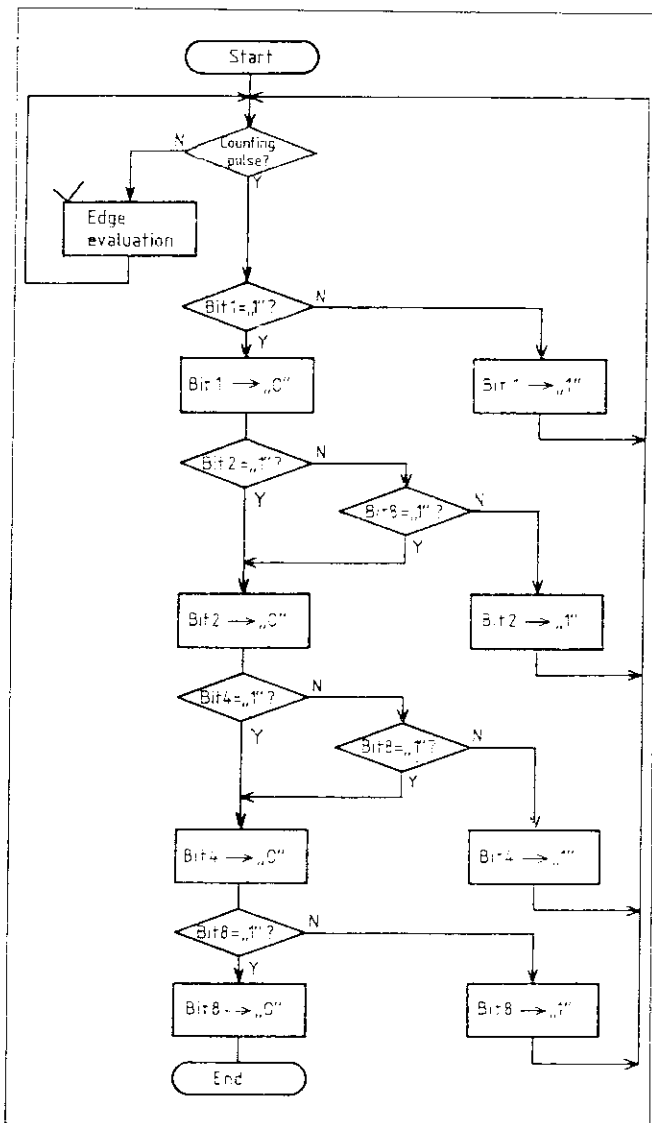
The BCD counter has the same rules as the binary counter.

- The least significant bit (1) of the counter changes its stage ("0 → 1", "1 → 0") with each counting pulse.
- Each time a bit position switches from "1 à 0", the state of the next higher bit is changed.

In addition, the following rule also applies:

- If bits 8 and 1 are "1" (= 9), when a counting pulse arrives, all bit positions are reset.

By means of nested subroutines, this routine ensures that all bits – starting with the least significant and extended to the most significant bit position – change their signal states sequentially with each counting pulse until a "1" is written into one position. The states of all bits with higher significance then remain unchanged. Each time the bit positions 2 and 4 are processed, bit 8 is tested. In the case of a "1" in position 8, a zero is allocated to the following bit positions of the decade, thus setting the counter to zero (10).



```

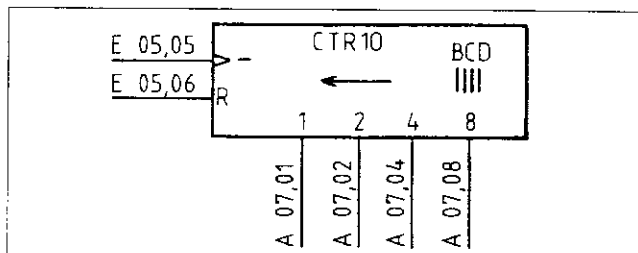
: N E 05,01      =R M'04,01
! E 05,01      &N M'04,01      Edge evaluation
= MA           =S M'04,01
! A 06,01      =N A 06,01      Bit 1
= MA
! A 06,02      / A 06,08      Bit 2
=N A 06,02     = MA
! A 06,04      / A 06,08      Bit 4
=N A 06,04     = MA
! A 06,08      =N A 06,08      Bit 8
! ME           Carry
! E 05,02      =R A 06,01
=R A 06,02     =R A 06,04      Resetting
=R A 06,08     the A flags
  
```

The carry is formed by allocation to N A 06,08 (see section 5.2.3 decimal counter up/down – 2 decades).

For purely internal use of the counter, the outputs (A) can be replaced by function flags (M).

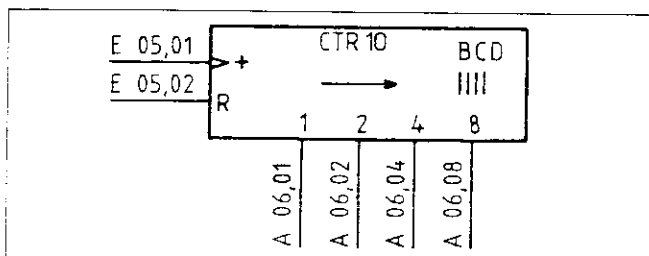
At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see section 5.2.3 decimal counter, up/down).

### 5.4.3 BCD counter, down



By means of nested subroutines, this routine ensures that all bits – starting with the least significant and extending to the most significant bit position – change their signal states sequentially with each counter pulse until a "0" is written into one position. The states of all bits with higher significance remain unchanged.

### 5.4.2 BCD counter, up



In the 1st step, setting of bit position 8 causes bit positions 2 and 4 to be reset (10 → 9).

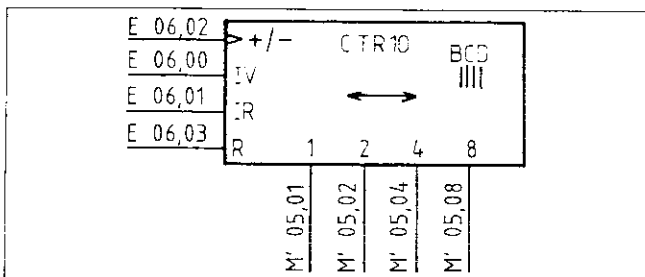
```

!N E 05,05 =R M'04,04
! E 05,05 &N M'04,04 Edge evaluation
= MA =S M'04,04
!N A 07,01 = A 07,01 Bit 1
= MA
!N A 07,02 = A 07,02 Bit 2
= MA
!N A 07,04 = A 07,04 Bit 4
= MA
!N A 07,08 = A 07,08 Bit 8
=N A 07,04 =N A 07,02 Carry
! ME
! E 05,06 =R A 07,01
=R A 07,02 =R A 07,04 Resetting
=R A 07,08 the A flags

```

At the start of the program, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see section 5.2.3 decimal counter, up/down).

#### 5.4.4 BCD counter, up/down - 1 decade



```

!N E 06,02 =R M'05,00
! E 06,02 &N M'05,00 Edge evaluation,
& E 06,00 = MA up
=S M'05,00
! M'05,01 =N M'05,01
= MA
! M'05,02 / M'05,08
=N M'05,02 = MA counting upwards
! M'05,04 / M'05,08
=N M'05,04 = MA
! M'05,08 =N M'05,08
! ME
! E 06,02 &N M'05,00
& E 06,01 = MA Edge evaluation,
=S M'05,00 down
!N M'05,01 = M'05,01
= MA
!N M'05,02 = M'05,02
= MA
!N M'05,04 = M'05,04 counting downwards
= MA
!N M'05,08 = M'05,08
=N M'05,04 =N M'05,02
! ME
! E 06,03 =R M'05,01
=R M'05,02 =R M'05,04 Resetting
=R M'05,08 the counter

```

The direction of counting can be preselected by means of one input.

```

! E 06,00 = up
!N E 06,00 = down

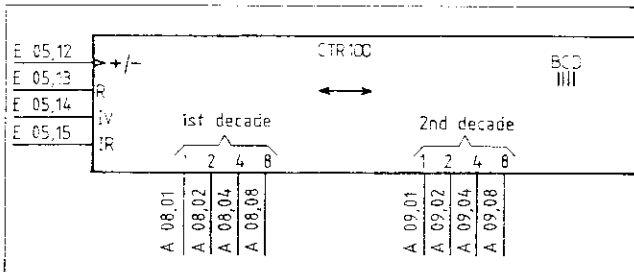
```

When the program is started and when the direction preselection signal is connected, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation

must be modified (see decimal counter, up/down). In this case, preselection of the direction counting with only one input is not possible.

If it is necessary to display the counter steps immediately, an output (A) can be used instead of the function flag (M) for each bit of the counter.

#### 5.4.5 BCD counter, up/down - 2 decades



```

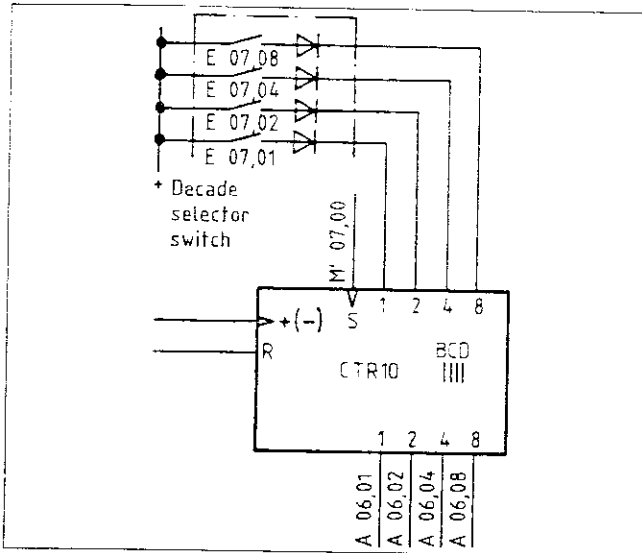
!N E 05,12 =R M'04,15
! E 05,12 &N M'04,15 Edge evaluation,
& E 05,14 = MA up
=S M'04,15
! A 08,01 =N A 08,01
= MA
! A 08,02 / A 08,08
=N A 08,02 = MA
! A 08,04 / A 08,08
=N A 08,04 = MA
! A 08,08 =N A 08,08
= MA counting upwards
! A 09,01 =N A 09,01
= MA
! A 09,02 / A 09,08
=N A 09,02 = MA
! A 09,04 / A 09,08
=N A 09,04 = MA
! A 09,08 =N A 09,08
! ME
! E 05,12 &N M'04,15
& E 05,15 = MA Edge evaluation,
=S M'04,15 down
!N A 08,01 = A 08,01
= MA
!N A 08,02 = A 08,02
= MA
!N A 08,04 = A 08,04
= MA
!N A 08,08 = A 08,08
=N A 08,04 = MA counting downwards
!N A 09,01 = A 09,01
= MA
!N A 09,02 = A 09,02
= MA
!N A 09,04 = A 09,04
= MA
!N A 09,08 = A 09,08
=N A 09,04 =N A 09,02
! ME
! E 05,13 =R A 08,01
=R A 08,02 =R A 08,04 Resetting
=R A 08,08 =R A 09,01 the counter
=R A 09,02 =R A 09,04
=R A 09,08

```

At the start of the program and when a counting direction preselection signal is connected, the counter input signal must be "0". If this cannot be guaranteed, the edge evaluation must be modified (see section 5.2.3 decimal counter, up/down). In this case, preselection of the counting direction with only one input is not possible.

Selection of the operating mode – up/down – can also be carried out with two separate clock signals (inputs).

### 5.4.6 BCD counter – setting



```

!N M'07,00      =R M'07,01
! M'07,00      &N M'07,01      Edge evaluation
= MA          =S M'07,01      Flag "set counter"

! E 07,01      = A 06,01      Bit 1
! E 07,02      = A 06,02      Bit 2
! E 07,04      = A 06,04      Bit 4   Set
! E 07,08      = A 06,08      Bit 8
! ME

```

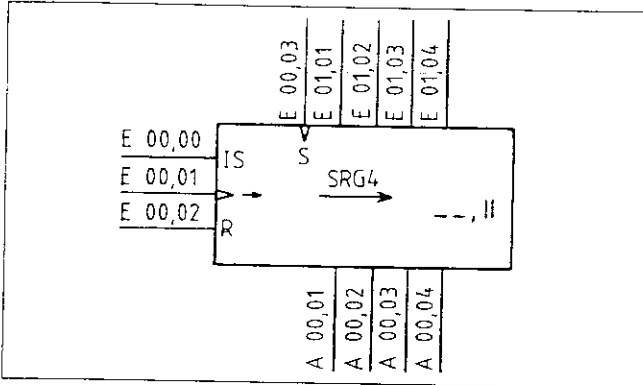
Program for counting  
 – corresponding to BCD counter, up or down



# 6 Registers

## 6.1 Shift registers

### 6.1.1 Shift registers, general



In a shift register, a unit of information (1 bit) is shifted by one position with each clock pulse through the number of available shift positions (4). With each clock pulse, a new bit (E 00,00) is moved into the first position of the register and the bit from the last position of the register is lost. For loading and for correction purposes, the shift register can also be loaded in parallel via separate inputs (E 01,01 – E 01,04) if a pulse is connected to input E 00,03.

		A 00,01	A 00,02	A 00,03	A 00,04
Initial state		0	0	0	0
E 00,03		E 01,01	E 01,02	E 01,03	E 01,04
Parallel loading		↓	↓	↓	↓
E 00,01	E 00,00	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	0
1	1	1	1	1	1
0	0	0	1	1	1
0	0	0	0	1	1
0	0	0	0	0	1
0	0	0	0	0	0

### 6.1.2 Shift register, implementation

By writing in the correct sequence – always from back to front – it is possible to ensure that the information stored in the A flags is shifted back one position with each shift pulse. The information from input E 00,00 is shifted into the first position. The information remains stored until the next shift pulse, parallel input, or reset operation. The set condition for the parallel inputs is located in a separate subroutine.

If the parallel input or reset facilities are not required, they can be omitted.

The shift register can be extended to any required length.

```

! E 00,02      =R A 00,01
=R A 00,02    =R A 00,03      Reset
=R A 00,04    =R M'00,05

!N E 00,01    =R M'00,05      Edge evaluation,
! E 00,01     &N M'00,05      shift
= MA          =S M'00,05

! A 00,03     = A 00,04      Bit 4
! A 00,02     = A 00,03      Bit 3
! A 00,01     = A 00,02      Bit 2 Shift
! A 00,00     = A 00,01      Bit 1
! ME
    
```

### Parallel input, loading

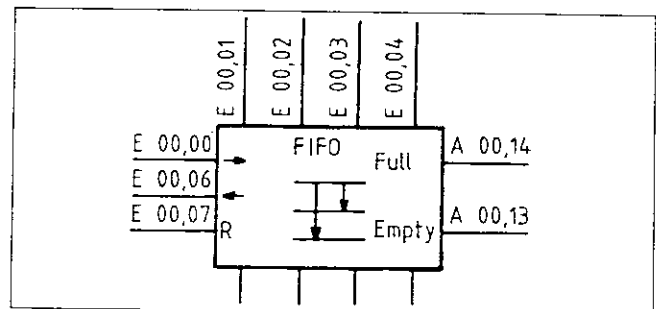
```

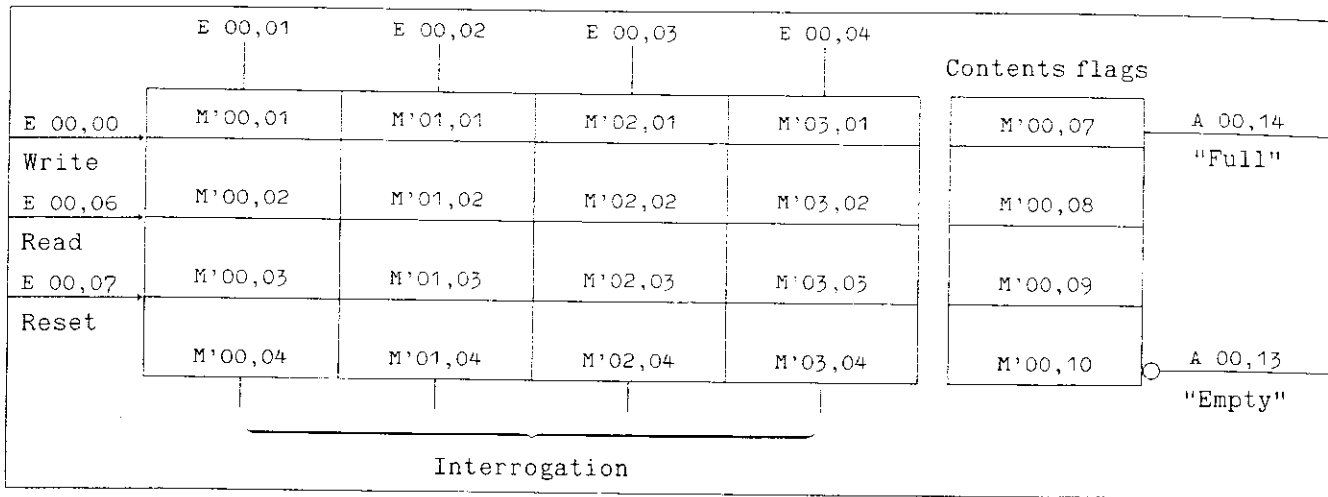
!N E 00,03    =R M'00,06      Edge evaluation
! E 00,03     &N M'00,00      Parallel input
= MA          =S M'00,00

! E 01,01     = A 00,01
! E 01,02     = A 00,02
! E 01,03     = A 00,03      Parallel input
! E 01,04     = A 00,04
! ME
    
```

## 6.2 Stack registers

### 6.2.1 Stack registers, general





In the case of a stack register (FIFO= first in – first out), information is moved in parallel (E 00,01 – E 00,04) into a memory with a write signal (E 00,00). If the register is empty, the information moves automatically to the last line and can be read out at the outputs. If lines are already occupied, the new information moves as far as the last free line.

The read signal (E 00,06) resets the last line and all other lines are shifted by one position. The reset signal clears the complete contents of the register.

If the register is full, i.e. if all lines are occupied, no further information is accepted. The states "Full" and "Empty" of the stack register are displayed.

### 6.2.2 Stack register, implementation

If the register is not already full (N M'00,07), the write pulse (edge detection with E 00,01) shifts the parallel information (E 00,01-E 00,04) into the first line. Acceptance of the data blocks the line (=S M'00,07). If the next line is not blocked (!N M'00,08), the information is moved to the next line, which is then blocked (=S M'00,08), and the block in the preceding line is cleared (=R M'00,07). This operation is continued until the next line is detected as blocked or full. The shift operation is then stopped. If the register was empty, the information moves down to the last line (=S M'00,10).

The read pulse (edge detection with E 00,06) clears the blocking flag of the last line (=R M'00,10). This releases the last line for the shift operation. If the preceding line was full (!M'00,09), the last line is again blocked. If the previous line was not full, the last line is not blocked. One line is processed with each program cycle. If the register is empty (A 00,13), reading is prevented. The information in "Free" lines is also shifted, but without the blocking flag, which means that it has no effect on subsequent writing and reading.

The register can be extended to any required length.

```

! M'00,07 = A 00,14 Register full (only
!N E 00,00 =R M'00,06 for write attempt)
! E 00,00 &N M'00,07
&N M'00,06 = MA Write
! E 00,01 = M'00,01
! E 00,02 = M'01,01
! E 00,03 = M'02,01 Write - 1st line
! E 00,04 = M'03,01
!N M'00,06 =S M'00,07
=S M'00,06 Line 1 blocked
! ME
!N M'00,08 = MA
! M'00,01 = M'00,02
! M'01,01 = M'01,02
! M'02,01 = M'02,02 Shift - 2nd line
! M'03,01 = M'03,02
! M'00,07 =S M'00,08 Line 1 free
=R M'00,07 Line 2 blocked
! ME
!N M'00,09 = MA
! M'00,02 = M'00,03
! M'01,02 = M'01,03
! M'02,02 = M'02,03 Shift - 3rd line
! M'03,02 = M'03,03
! M'00,08 =S M'00,09 Line 2 free
=R M'00,08 Line 3 blocked
! ME
!N M'00,10 = MA
! M'00,03 = M'00,04
! M'01,03 = M'01,04
! M'02,03 = M'02,04 Shift - 4th line
! M'03,03 = M'03,04
! M'00,09 =S M'00,10 Line 3 free
=R M'00,09 Line 4 blocked
! ME
!N M'00,10 = A 00,13 Register empty
! E 00,06 & M'00,10
&N M'00,12 =R M'00,10 Edge evaluation
=S M'00,12 =R M'00,10 Line 4 free
!N E 00,06 =R M'00,12

```

#### Resetting

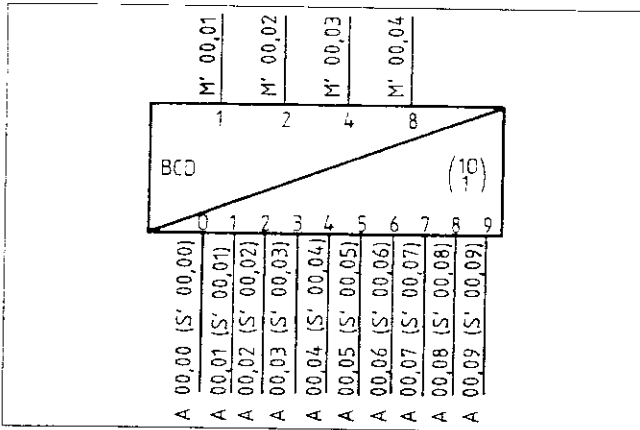
```

!N E 00,07 =R M'00,14
! E 00,07 &N M'00,14
=S M'00,14 = MA
=N M'00,01 =N M'00,02
=N M'00,03 =N M'00,04
=N M'01,01 =N M'01,02
=N M'01,03 =N M'01,04
=N M'02,01 =N M'02,02
=N M'02,03 =N M'02,04
=N M'03,01 =N M'03,02
=N M'03,03 =N M'03,04
=R M'00,07 =R M'00,08
=R M'00,09 =R M'00,10
! ME

```

## 7 Code converters

### 7.1 Code converter, BCD into 1-out-of-10



```

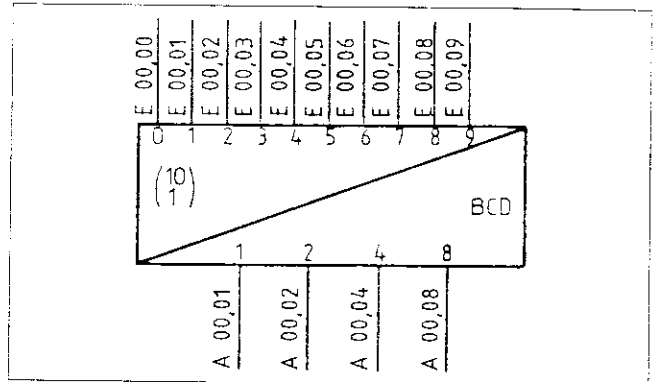
!N M'00,01      &N M'00,02
&N M'00,03      &N M'00,04
= A 00,00
! M'00,01      &N M'00,02
&N M'00,03      &N M'00,04
= A 00,01
!N M'00,01      & M'00,02
&N M'00,03      = A 00,02
! M'00,01      & M'00,02
&N M'00,03      = A 00,03
!N M'00,01      &N M'00,02
& M'00,03      = A 00,04
! M'00,01      &N M'00,02
& M'00,03      = A 00,05
!N M'00,01      & M'00,02
& M'00,03      = A 00,06
! M'00,01      & M'00,02
& M'00,03      = A 00,07
!N M'00,01      & M'00,04
= A 00,08
! M'00,01      & M'00,04
= A 00,09
    
```

Alternative solution for purely internal use with a register chain:

```

!N S'00,00      = S'00,00
! M'00,04      = S'00,08
! M'00,03      = S'00,04
! M'00,02      = S'00,02
! M'00,03      & M'00,02
= S'00,06
! M'00,01      = MA
! S'00,00      = S'00,01
! S'00,02      = S'00,03
! S'00,04      = S'00,05
! S'00,06      = S'00,07
! S'00,08      = S'00,09
! ME
    
```

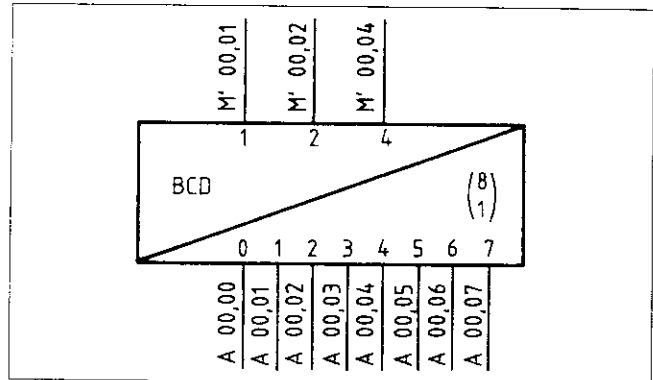
### 7.2 Code converter, 1-out-of-10 into BCD



```

! E 00,01      / E 00,03
/ E 00,05      / E 00,07
/ E 00,09      = A 00,01
! E 00,02      / E 00,03
/ E 00,06      / E 00,07
= A 00,02
! E 00,04      / E 00,05
/ E 00,06      / E 00,07
= A 00,04
! E 00,08      / E 00,09
= A 00,08
    
```

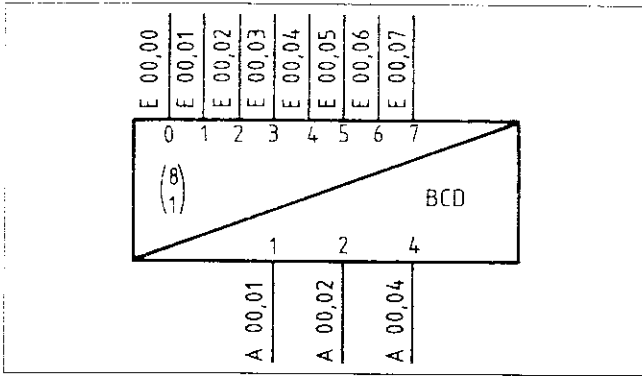
### 7.3 Code converter, BCD into 1-out-of-8



```

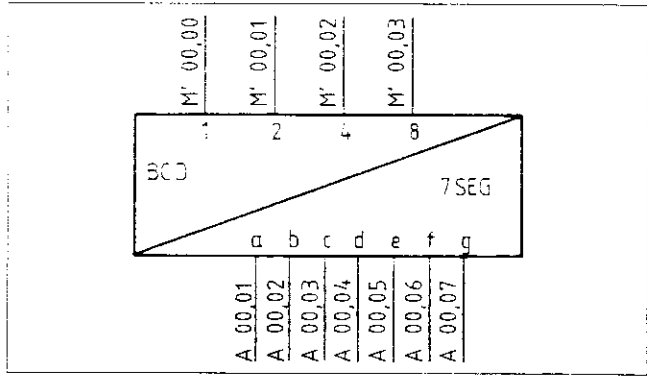
!N S'00,00      = S'00,00
! M'00,04      = S'00,04
! M'00,02      = S'00,02
! M'00,04      & M'00,02      = S'00,06
! M'00,01      = MA
! S'00,00      = S'00,01
! S'00,02      = S'00,03
! S'00,04      = S'00,05
! S'00,06      = S'00,07
! ME
    
```

### 7.4 Code converter, 1-out-of-8 into BCD



```

! E 00,04 / E 00,05
/ E 00,06 / E 00,07
= A 00,04
! E 00,01 / E 00,03
/ E 00,05 / E 00,07
= A 00,01
! E 00,02 / E 00,03
/ E 00,06 / E 00,07 = A 00,02
    
```



```

! M'00,00 &N M'00,01
&N M'00,02 &N M'00,03 Segment a
/N M'00,00 & M'00,02
=N A 00,01

! M'00,02 & M'00,00
&N M'00,01 / M'00,02 Segment b
& M'00,01 &N M'00,00
=N A 00,02

! M'00,01 &N M'00,00
&N M'00,03 =N A 00,03 Segment c

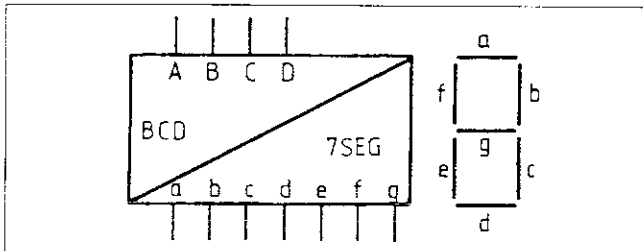
! M'00,00 &N M'00,01
&N M'00,02 / M'00,00 Segment d
& M'00,01 & M'00,02
/N M'00,00 &N M'00,01
& M'00,02 =N A 00,04

! M'00,00 /N M'00,00
&N M'00,01 & M'00,02 Segment e
=N A 00,05

! M'00,00 &N M'00,02
&N M'00,03 / M'00,01 Segment f
&N M'00,02 / M'00,00
& M'00,01 & M'00,02
=N A 00,06

!N M'00,01 &N M'00,02
&N M'00,03 / M'00,00 Segment g
& M'00,01 & M'00,02
=N A 00,07
    
```

### 7.5 Code converter, BCD into 7-segment

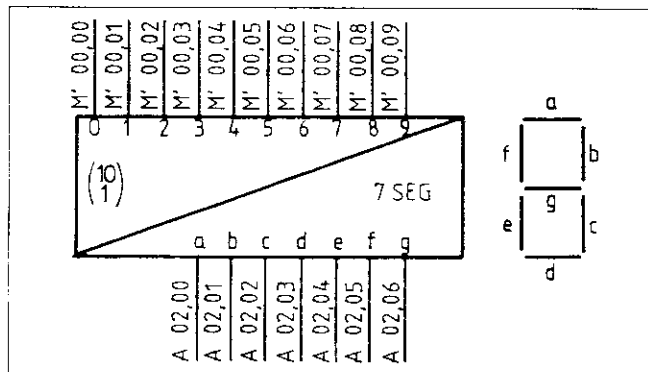


7 segment display	BCD		NAa	NAb	NAc	NAd	NAe	NAf	NAg
	"1"	"0"							
1	A	B, C, D	x			x	x	x	x
2	B	A, C, D		x				x	
3	A, B	C, D					x	x	
4	C	A, B, D	x			x	x		
5	A, C	B, D		x			x		
6	B, C	A, D	x	x					
7	A, B, C	D				x	x	x	x
8	D	A, B, C							
9	A, D	B, C				x	x		
0		A, B, C, D							x

```

! A &N B &N C &N D /N A & C =N Aa
! C & A &N B / C & B &N A =N Ab
! B &N A &N C =N Ac
! A &N B &N C / A & B & C /N A &N B & C
=N Ad
! A /N A &N B & C =N Ae
! A &N C &N D / B &N C / A & B & C =N Af
!N B &N C &N D / A & B & C =N Ag
    
```

### 7.6 Code converter, decimal into 7-segment



	7-segment- display		NAa	NAb	NAc	NAd	NAe	NAf	NAg
	"1"	"0"							
1	b,c	a,d,e, f,g	x			x	x	x	x
2	a,b,d, e,g	c,f			x			x	
3	a,b,c, d,g	e,f					x	x	
4	b,c,f, g	a,d,e	x			x	x		
5	a,c,d, f,g	b,e		x			x		
6	c,d,e, f,g	a,b	x	x					
7	a,b,c	d,e,f, g				x	x	x	x
8	a,b,c,d e,f,g								
9	a,b,c, f,g	d,e				x	x		
0	a,b,c, d,e,f	g							x

```

! 1 / 4 / 6 =N Aa
! 5 / 6 =N Ab
! 2 =N Ac
! 1 / 4 / 7 / 9 =N Ad
! 2 / 6 / 8 / 0 =N Ae
! 1 / 2 / 3 / 7 =N Af
! 1 / 7 / 0 =N Ag

```

```

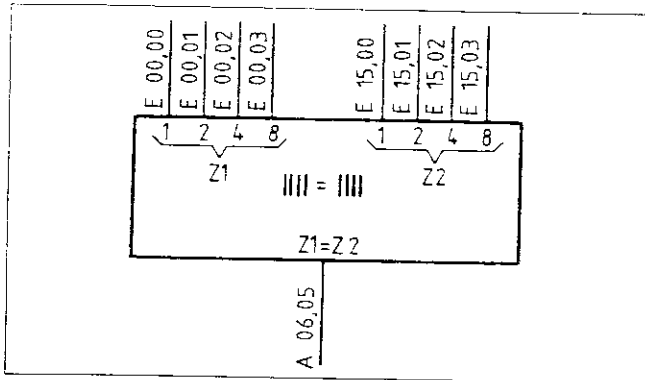
! M'00,01 / M'00,04
/ M'00,06 =N A 02,00 Segment a
! M'00,05 / M'00,06 Segment b
=N A 02,01
! M'00,02 =N A02,02 Segment c
! M'00,01 / M'00,04
/ M'00,07 / M'00,09 Segment d
=N A 02,03
! M'00,02 / M'00,06
/ M'00,08 / M'00,00 Segment e
=N A 02,04
! M'00,01 / M'00,02
/ M'00,03 / M'00,07 Segment f
=N A 02,05
! M'00,01 / M'00,07
/ M'00,00 =N A 02,06 Segment g

```



# 8 Comparator

## 8.2 Greater-less-equal comparator



```

!N S'00,03 = S'00,03 Z1 = Z2
! E 00,00 &N E 15,12
= S'00,01 Z1 > Z2
!N E 00,00 & E 15,12
= S'00,02 Z1 < Z2
! E 00,01 &N E 15,13
= S'00,01
!N E 00,01 & E 15,13
= S'00,02
! E 00,02 &N E 15,14
= S'00,01
!N E 00,02 & E 15,14
= S'00,02
! E 00,03 &N E 15,15
= S'00,01
!N E 00,03 & E 15,15
= S'00,02
! Resetting = S'00,00 (if required)
    
```

Each separate bit position is tested for greater (>) or less (<). If this is not the case, the two values are equal.

Alternative solution without register chain:

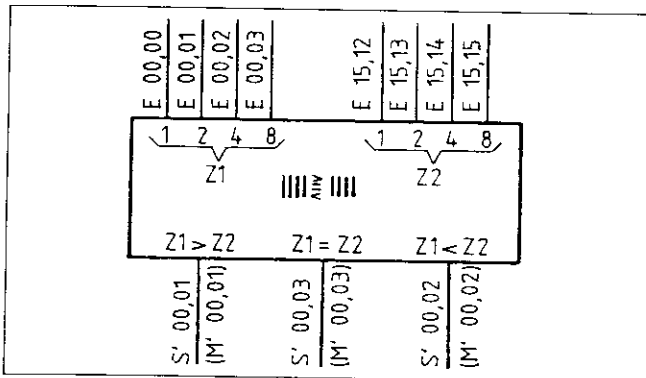
```

!N E 00,00 & E 15,00
/ E 00,00 &N E 15,00
/N E 00,01 & E 15,01
/ E 00,01 &N E 15,01
/N E 00,02 & E 15,02
/ E 00,02 &N E 15,02
/N E 00,03 & E 15,03
/ E 00,03 &N E 15,03
=N A 06,05
    
```

```

!N M'00,03 =R M'00,01
=R M'00,02
! E 00,00 &N E 15,12
=S M'00,01 =R M'00,02
!N E 00,00 & E 15,12
=R M'00,01 =S M'00,02
! E 00,01 &N E 15,13
=S M'00,01 =R M'00,02
!N E 00,01 & E 15,13
=R M'00,01 =S M'00,02
! E 00,02 &N E 15,14
=S M'00,01 =R M'00,02
!N E 00,02 & E 15,14
=R M'00,01 =S M'00,02
! E 00,03 &N E 15,15
=S M'00,01 =R M'00,02
!N E 00,03 & E 15,15
=R M'00,01 =S M'00,02
!N M'00,01 &N M'00,02
=M'00,03
    
```

## 8.1 Comparator for equivalence

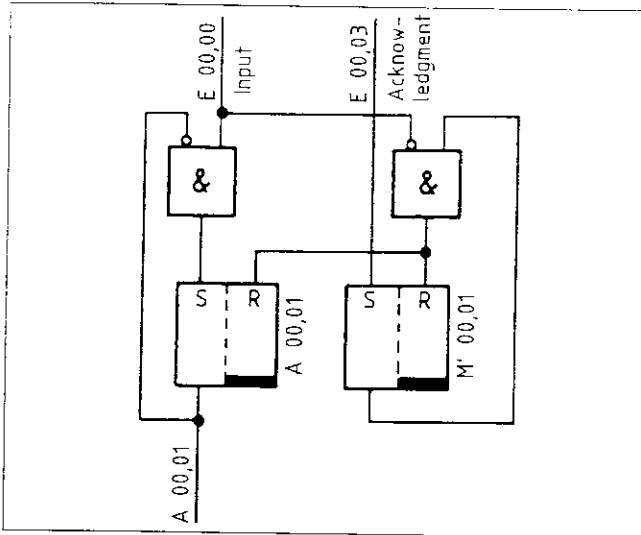


Starting with the least significant bit, each single bit is checked for greater (>) or less (<). If a more significant bit requires correction of a result, the characteristic of the register chain deletes the previous results.



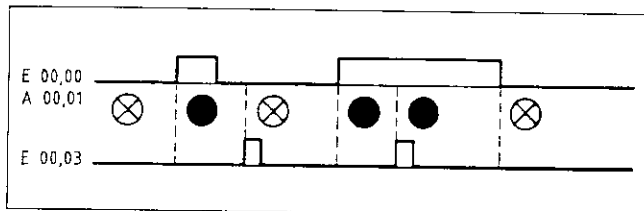
## 9 Signalling controls

### 9.1 Signalling with continuous light



The operational status to be signaled (E 00,00) is indicated by continuous light (A 00,01). If the signal is still present when the acknowledgment arrives (E 00,03), the light remains on until the signal disappears. In all other cases, the lamp is switched off immediately after acknowledgment.

Functional diagram:



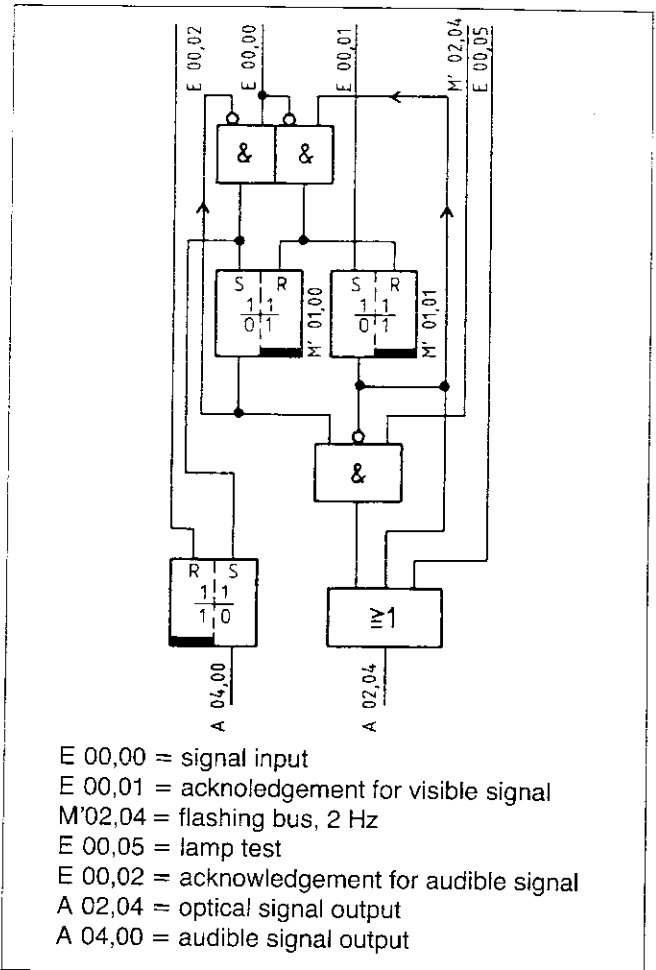
! E 00,00	&N A 00,01	=S A 00,01
! E 00,03	=S M'00,01	
! M'00,01	&N E 00,00	=R A 00,01
=R M'00,01		

### 9.2 New value signalling with single flashing light

Each new signal at the signal input E 00,00 is indicated at input A 02,04 by a 2 Hz flashing signal (N T 02,00 → A 02,04) (see also section 4.6 "Oscillator"). At the same time, an audible signal (A 04,00) is activated.

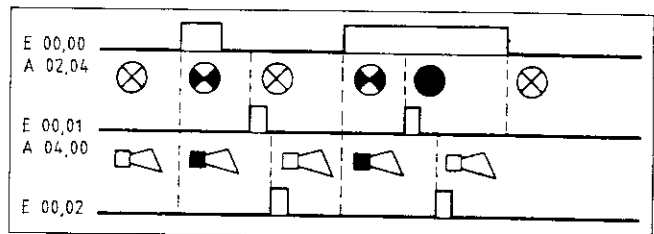
If the acknowledgment key E 00,01 is depressed, then the flashing light changes to a continuous light. When the incoming signal disappears, the optical indication (A 02,04) is switched off.

The audible signal can be switched off only with its own acknowledgment key (E 00,02).



E 00,00 = signal input  
 E 00,01 = acknowledgement for visible signal  
 M'02,04 = flashing bus, 2 Hz  
 E 00,05 = lamp test  
 E 00,02 = acknowledgement for audible signal  
 A 02,04 = optical signal output  
 A 04,00 = audible signal output

Functional diagram:



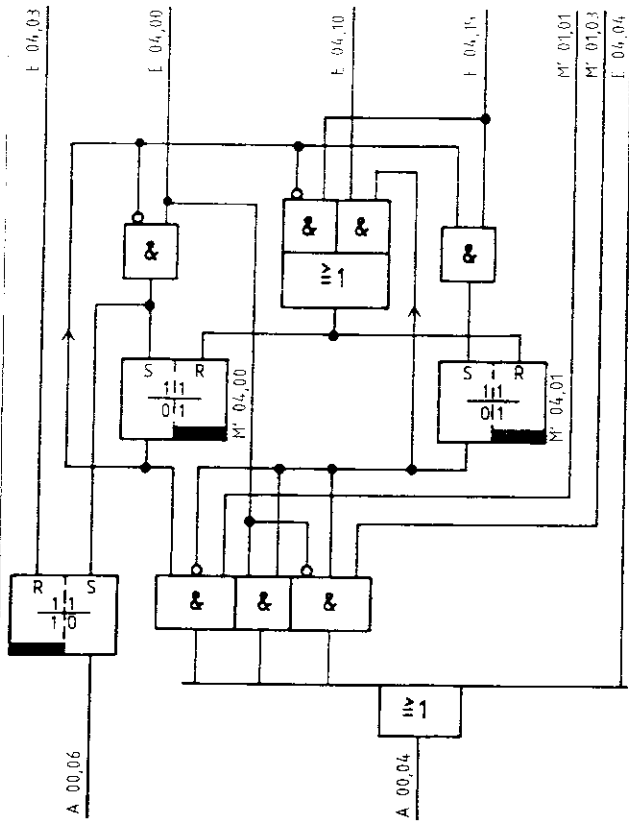
! E 00,00	&N M'01,00	
=S M'01,00	=S A 04,00	
! E 00,01	=S M'01,01	
!N E 00,00	& M'01,01	
=R M'01,00	=R M'01,01	
! M'01,00	&N M'01,01	
& M'02,04	/ M'01,01	
/ E 00,05	= A 02,04	
! E 00,02	=R A 04,00	

Oscillator

:N T 02,00	= T 02,00
= M'00,01	
:N M'00,01	= MA
:N M'02,04	= M'02,04
: ME	

### 9.3 New value signalling with double flashing light

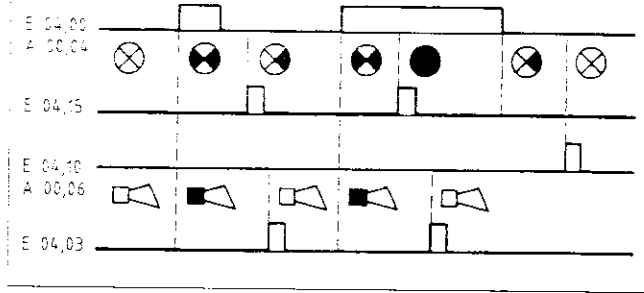
### Functional diagram



- E 04,00 = signal input
- E 04,15 = acknowledgment for visible signal
- E 04,10 = acknowledgment for signal buffer, reset
- M'01,01 = flashing bus, 2 Hz
- M'01,03 = flashing bus, 0.5 Hz
- E 04,04 = lamp test
- E 04,03 = acknowledgment for audible signal
- A 00,04 = optical signal output
- A 00,06 = audible signal output

Each new signal at signal input E 04,00 is indicated at the output A 00,04 via flashing light (2 Hz). At the same time, the audible alarm (A 00,06) is activated. After depression of the acknowledgment key E 04,15, the flashing light switches to a slow flashing light (0,5 Hz) if the input signal is no longer present. If the input signal is still present, then the flashing switches to a continuous light. Slow flashing can be switched off with the acknowledgment key for the signal buffer (E 04,10).

The audible signal (A 00,06) is switched off with its own acknowledgment key (E 04,03). The flashing buses must be programmed separately for this example (see also section 4.8, Oscillator with frequency divider).



```

! E 04,00      &N M'04,00
Signal        Signalbuffer

=S M'04,00    =S A 00,06
Signalbuffer  Horn

! E 04,15     & M'04,00      =S M'04,01
Acknowledg-   Signal        Acknowledged
ment signal  buffer

! E 04,15     &N M'04,00
Acknowledg-   Signalbuffer
ment signal

/ M'04,01     & E 04,10
Acknowledged  Reset

=R M'04,00    =R M'04,01
Signalbuffer  Acknowledged

! M'04,00     &N M'04,01
Signalbuffer  Acknowledged

& M'01,01    / M'04,01
Flash 2 Hz   Acknowledgment

& E 04,00
Signal

/ M'04,01    &N E 04,00
Acknowledged Signal

& M'01,03    / E 04,04
Flash 0,5 Hz Lamp test

= A 00,04
Optical signal

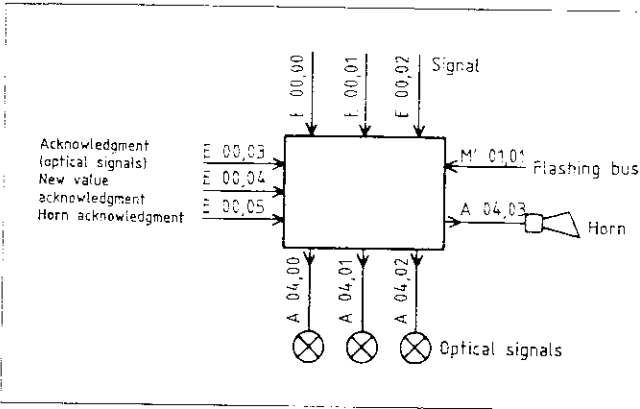
! E 04,03     =R A 00,06
Horn ack-    Horn off
nowledgment
    
```

#### Oscillator with 0,5 and 2 Hz

```

!N T 01,00    = T 01,00 (T01,00 = 0,25 s)
= M'01,00
!N M'01,00    = MA
!N M'01,01    = M'01,01    2 Hz
= MA
!N M'01,02    = M'01,02
= MA
!N M'01,03    = M'01,03    0,5 Hz
! ME
    
```

## 9.4 First and new value signalling with single acknowledgment

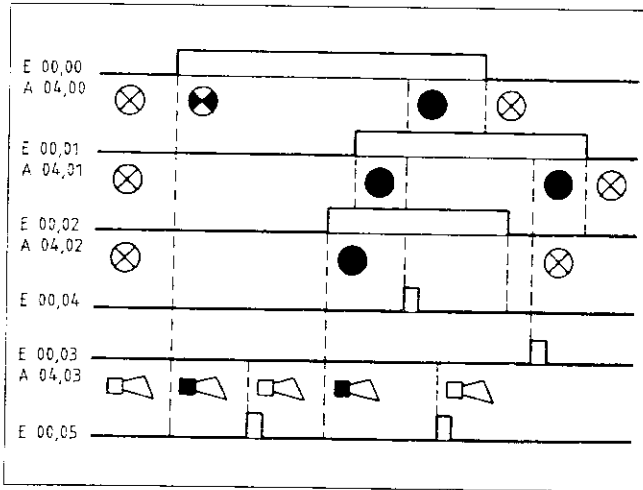


The first value signal identifies that signal in a group (E 00,00 to E 00,02) which first occurred. Only the first state which occurs is signaled with a flashing light (2 Hz); all subsequent states are signaled with continuous light. Acknowledgment is carried out only via a first value acknowledgment key. If the signal is still present after first value acknowledgment, the lamp switches to continuous light until the signal disappears.

Further signals are present as continuous signals until the state to be signaled has disappeared after acknowledgment.

The horn is switched on with each new value which is signaled.

Functional diagram:



```

! E 00,00      &N M'00,00
1st signal    1st signal
buffer

=S M'00,00    =S A 04,03      1st value pulse
1st signal    Horn
buffer

= NA

!N M'00,02    =S M'00,02
1st value     1st value
buffer        buffer

=S M'00,03    1st signal is
1st value     1st value

! ME

! E 00,01    &N M'00,04
2nd signal   2nd signal
buffer

=S M'00,04    =S A 04,03      1st value pulse
2nd signal   Horn
buffer

= MA

!N M'00,02    =S M'00,02
1st value     1st value
buffer        buffer

=S M'00,05    2nd signal
is 1st value

! ME

! E 00,02    &N M'00,06
3rd signal   3rd signal
buffer

=S M'00,06    =S A 04,03      1st value pulse
3rd signal   Horn
buffer

= MA

!N M'00,02    =S M'00,02
1st value     1st value
buffer        buffer

=S M'00,07    3rd signal
is 1st value

! ME

! E 00,03    &N M'00,03
Acknowledgment Acknowledgment
buffer        buffer

/ E 00,04    & M'00,03
1st value     Acknowledgment
acknowledgment buffer

=S M'00,08
Acknowledgment
buffer i
    
```

```

! E 00,04      =R M'00,03
lst value      Acknowledgment
acknowledgment buffer

=R M'00,02
lst value
buffer

! E 00,03      &N M'00,05
Acknowledgment lstvalue ack-
                nowledge ment 2
/ E 00,04      & M'00,05
lst value      lst value ack-
acknowledgment nowledge ment 2

=S M'00,09
Acknowledgment
buffer 2

! E 00,04      =R M'00,05
lst value      lstvalue ack-
acknowledgment nowledge ment 2

! E 00,03      &N M'00,07
Acknowledgment lstvalue ack-
                nowledge ment 3

/ E 00,04      & M'00,07
lst value      lstvalue ack-
acknowledgment nowledge ment 3

=S M'00,10
Acknowledgment
buffer 3

! E 00,04      =R M'00,07
lst value      lstvalue ack-
acknowledgment nowledge ment 3

! M'00,08      &N E 00,00
=R M'00,03      =R M'00,00      Signal 1
=R M'00,08

! M'00,09      &N E 00,01
=R M'00,05      =R M'00,04      Signal 2
=R M'00,09

! M'00,10      &N E 00,02
=R M'00,07      =R M'00,06      Signal 3
=R M'00,10

! E 00,05      =R A 04,03      Horn Acknowledgment

! M'00,03      & M'01,01
/ M'00,00      &N M'00,03      Optical signal 1
= A 04,00

! M'00,05      & M'01,01
/ M'00,04      &N M'00,05      Optical signal 2
= A 04,01

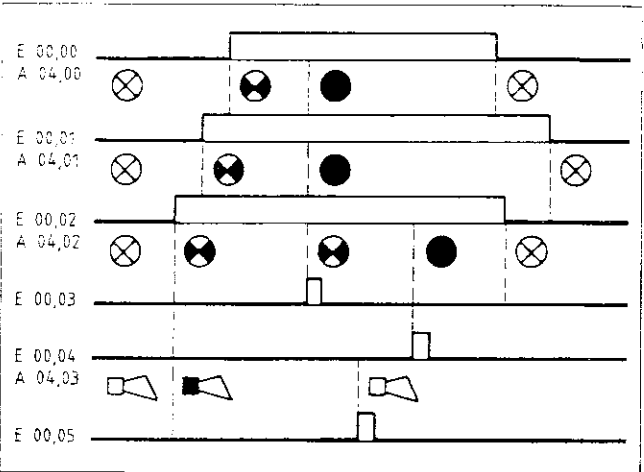
! M'00,07      & M'01,01
/ M'00,06      &N M'00,07      Optical signal 3
= A 04,02

!N T 01,00     = T 01,00
= M'01,00
!N M'01,00     = MA      Oscillator
!N M'01,01     = M'01,01      (T 01,00 = 0,25 s)
! ME

```

by the first value acknowledgment (E 00,04) and then remains active as a continuous light as long as the signal is present.

Functional diagram:



The program is similar to that for first value signalling with single acknowledgment, except that the output module (shown within the box) must be written as follows:

```

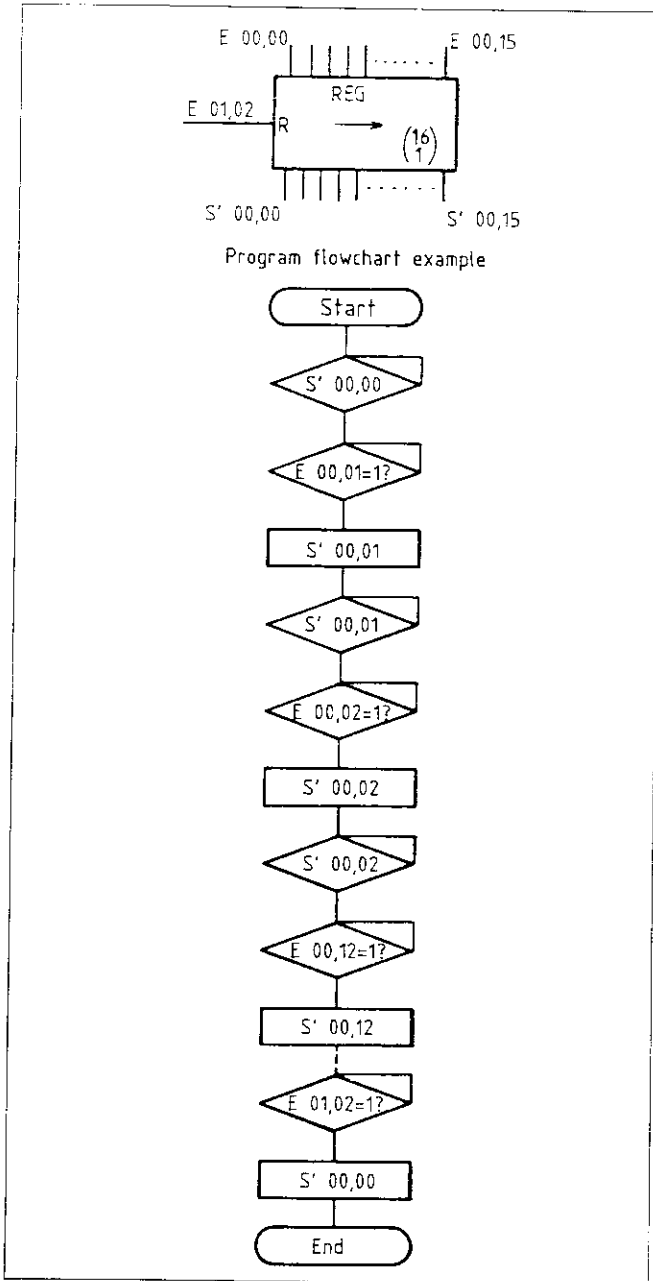
! M'00,00      &N M'00,08
& M'01,01      / M'00,08      Signal 1
= A 04,00
! M'00,04      &N M'00,09
& M'01,01      / M'00,09      Signal 2
= A 04,01
! M'00,06      &N M'00,10
& M'01,01      / M'00,10      Signal 3
= A 04,02

```

### 9.5 First value signalling with double acknowledgment

All operating states which require signalling are displayed as a new value with a flashing light. After acknowledgment of the optical signal (E00.03), only the first value alarm continues to flash. The other signals remain active as continuous light as long as the operating state to be signaled exists. The first value can be acknowledged only

10.1 Register as sequence control



The register chains of the PROCONTIC b are hardware stepping chains with 16 of 1 characteristic. The register steps have a buffer characteristic. Setting of one register step (= S'...,...) resets the register position which was already set. The positions can be set into any sequence. When the voltage is connected, the first step (00) of a register chain is always set. This step should not be used for active switching activities, but only for definition of an idle state.

To set a register step, the previous should always be interrogated in order to achieve the required setting sequence.

! S'00,00	& E 00,01 =	S'00,01	Reset
Interrogate	Stepping	Set 1st regi-	step 00
basic state	condition	ster step	
! S'00,01	& E 00,02 =	S'00,02	Reset
Interrogate	Stepping	Set 2nd regi-	step 01
1st register	condition	ster step	
step			
! S'00,02	& E 00,12 =	S'00,12	Reset
Interrogate	Stepping	Set 12th	step 02
2nd register	condition	register step	Registers
step			03-11 are
			not used
! E 01,02	=	S'00,00	Resetting
Reset con-	Set the		register
dition	basic state		step 12

## 10.2 Sequential connection of two register chains

At the start, the zero state of both register chains must be interrogated:

```
! S'00,00 & S'01,00
& E 00,00 = S'00,01
```

Stepping is carried out within each register chain as for a simple register circuit.

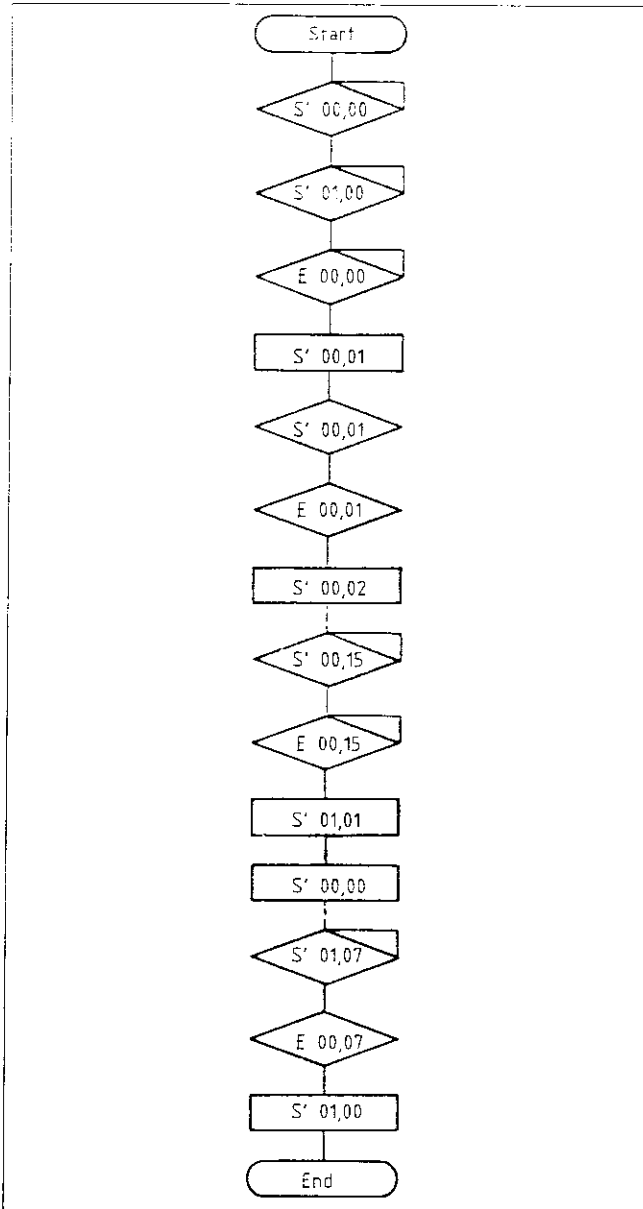
```
! S'00,01 & E 00,01
= S'00,02
! S'00,02 & E 00,02
= S'00,03
etc.
```

Setting the first step of the second register chain does not automatically reset the last step of the first chain. This must be coupled to the setting condition.

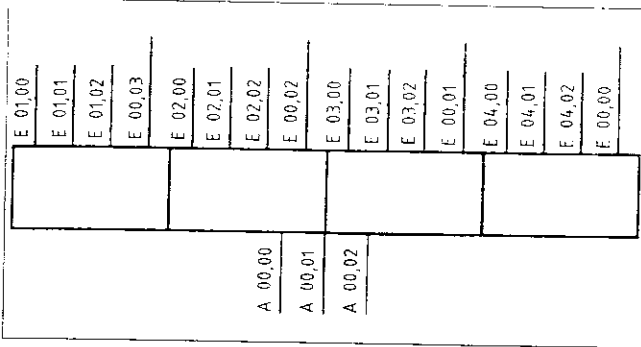
```
! S'00,15 & E 00,15
= S'01,01 = S'00,00
```

The last program step of the second chain must reset the second chain to the idle state in order to release the coupled chains for a subsequent start (E 00,00). The first chain was already reset by the transition condition.

```
! S'01,07 & E 01,07
= S'01,00
```



# 11 Channel selection



The purpose of the channel selection routine (= priority encoder) is to switch the signal states of various data inputs (E 01,00 to E 01,02, E 02,00 to E 02,02, E 03,00 to E 03,02, E 04,00 to E 04,02) to the outputs (A 00,00 to A 00,02) according to the various select inputs (E 00,00 to E 00,03). The select input with the highest channel number has priority (E 00,03 before E 00,02 before E 00,01 before E 00,00).

```

!N E 00,03    & E 00,02
= M'00,01
!N E 00,03    &N E 00,02    Priority
& E 00,01    = M'00,02    definition
!N E 00,03    &N E 00,02
&N E 00,01    & E 00,00
= M'00,03

! E 01,00    & E 00,03
/ E 02,00    & M'00,01
/ E 03,00    & M'00,02
/ E 04,00    & M'00,03
= A 00,00

! E 01,01    & E 00,03
/ E 02,01    & M'00,01
/ E 03,01    & M'00,02
/ E 04,01    & M'00,03
= A 00,01

! E 01,02    & E 00,03
/ E 02,02    & M'00,01
/ E 03,02    & M'00,02
/ E 04,02    & M'00,03
= A 00,01
    
```

PROCONTIC b Applications Issue 02.88

## 12 Programming Examples For The Fast Timer 07 ZG 84

### 12.1 Assignments for the programming examples

For the units listed in the following programming examples these slot assignments are valid:

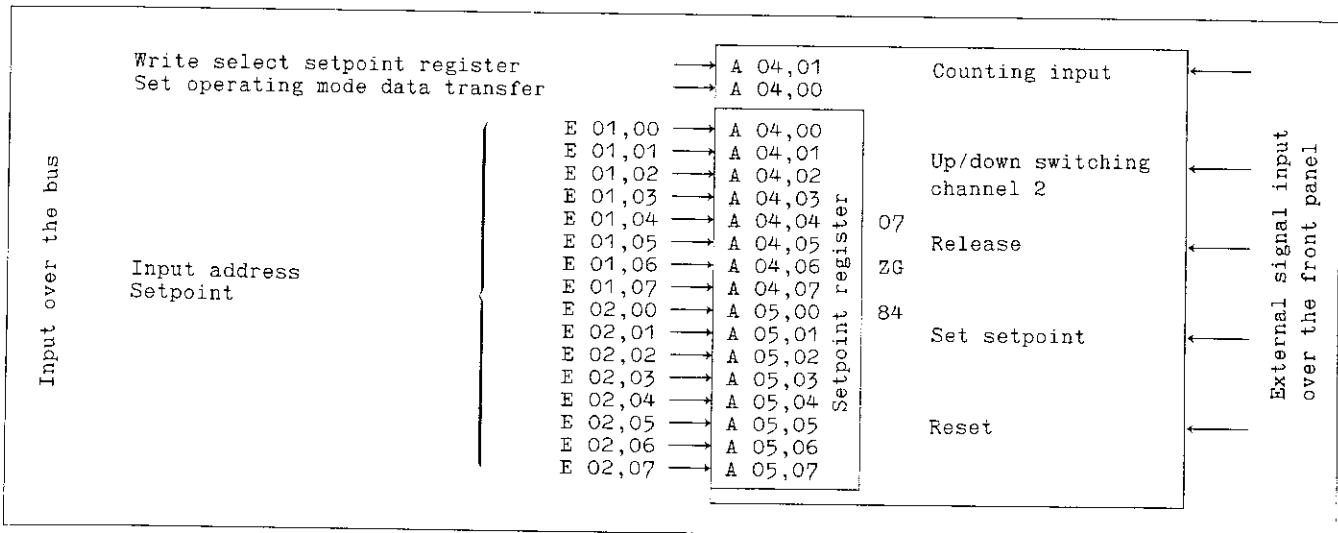
- E 00, ... input unit 1
- E 01, ... input unit 2
- E 02, ... input unit 3
- E 03, ... input unit 4
- E 04, ... counter, input addresses left unit half
- E 05, ... counter, input addresses right unit half
- A 04, ... counter, output addresses left unit half
- A 05, ... counter, output addresses right unit half
- A 06, ... output unit 1
- A 07, ... output unit 2
- A 08, ... output unit 3

The signals VK2 (1-signal = up, 0-signal = down for channel 2), SS (set), FR (release) and R (reset) for the triggering of the counter are entered via the terminals on the front panel of the counter 07 ZG 84 (see section 12.7).

Positions of the coding switch on the left side board of the counter 07 ZG 84 (see also volume 2, section 8.5):

	off	on	
S1		*	- counting direction channel 1 up
S2	*		
S3	*		} pulse counting
S4		*	
S5	*	*	} positive flank channel 1 county
S6		*	
S7	*	*	} positive flank channel 2 county
S8	*		
			- decimal counting

### 12.2 Load setpoint into the setpoint register



Notice:

The sequency of the inputs is obligatory

Program for bit processing:

```

! E 00,00 = MA
= A 04,01 write select
setpoint register
set operating mode
data transfer

= A 04,00

! E 01,00 = A 04,00
! E 01,01 = A 04,01
! E 01,02 = A 04,02
! E 01,03 = A 04,03
! E 01,04 = A 04,04
! E 01,05 = A 04,05
! E 01,06 = A 04,06
! E 01,07 = A 04,07
! E 02,00 = A 05,00 setpoint
! E 02,01 = A 05,01
! E 02,02 = A 05,02
! E 02,03 = A 05,03
! E 02,04 = A 05,04
! E 02,05 = A 05,05
! E 02,06 = A 05,06
! E 02,07 = A 05,07 write bit 15:
setpoint is entered

! E 00,00 /N E 00,00
= A 05,07 set operating mode
control
reset select
setpoint register

=N A 04,01

! ME
    
```

With the word processor 07 WP 84 or the central processor unit 07 ZE 86 by assignment of a constant

```

!N E 00,00 = SPM 000
! E 00,00 = A 04,01
= A 04,00
! 07 00050 = cA 04,00
! E 00,00 /N E 00,00
= A 05,07 =N A 04,01
    
```

With the word processor 07 WP 84 or the central processor unit 07 ZE 86 by assignment of two input units following each other.

```

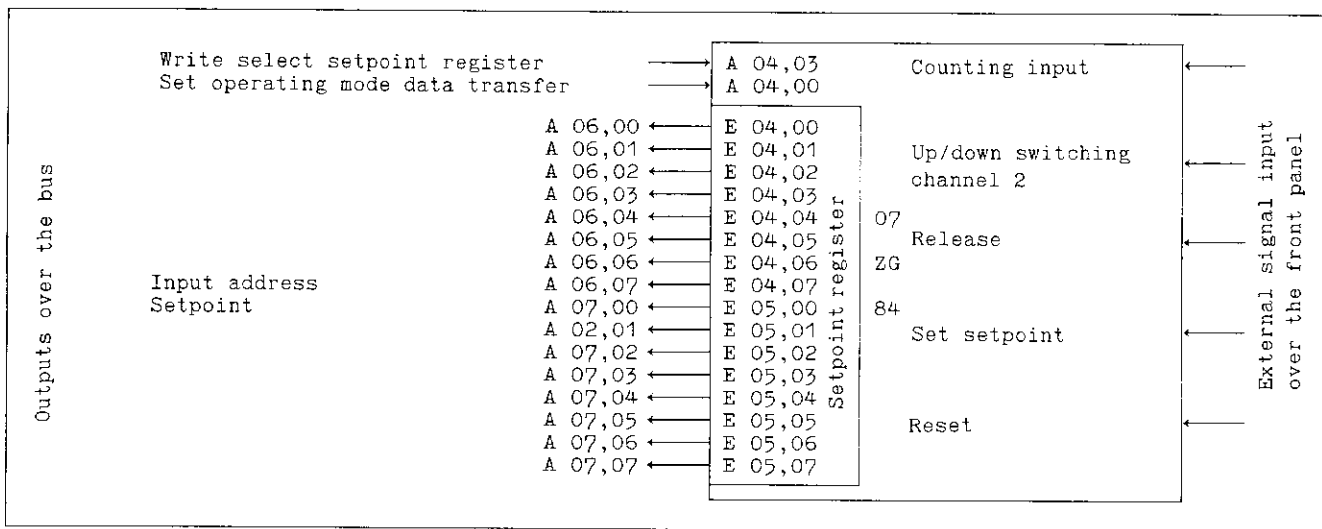
!N E 00,00 = SPM 000
! E 00,00 = A 04,01
= A 04,00
! bE 01,00 = bA 04,00
! E 00,00 /N E 00,00
= A 05,07 =N A 04,01
! MR 000
    
```

### 12.3 Loading the pre-trip value

Loading the pre-trip value is done in the same way as loading the set point, but by using the signal "write select pre-trip value register".

instead of A 04,01 → A 04,02

### 12.4 Read actual value and display on output unit



Notice:  
The sequency of the inputs is obligatory

Program for bit processing:

```

! E 00,00 = MA
= A 04,03      read select actual
                value register
                set operating mode
                data transfer

= A 04,00

! E 04,00 = A 06,00
! E 04,01 = A 06,01
! E 04,02 = A 06,02
! E 04,03 = A 06,03
! E 04,04 = A 06,04
! E 04,05 = A 06,05
! E 04,06 = A 06,06
! E 04,07 = A 06,07      read actual value
                            (16 bits) and
                            display on output
                            unit

! E 05,00 = A 07,00
! E 05,01 = A 07,01
! E 05,02 = A 07,02
! E 05,03 = A 07,03
! E 05,04 = A 07,04
! E 05,05 = A 07,05
! E 05,06 = A 07,06
! E 05,07 = A 07,07
! E 00,01 /N E 00,01
= A 05,07      set operating mode
                control, leave
                operating mode data
                transfer

=N A 04,03     reset read select
                actual value
                register

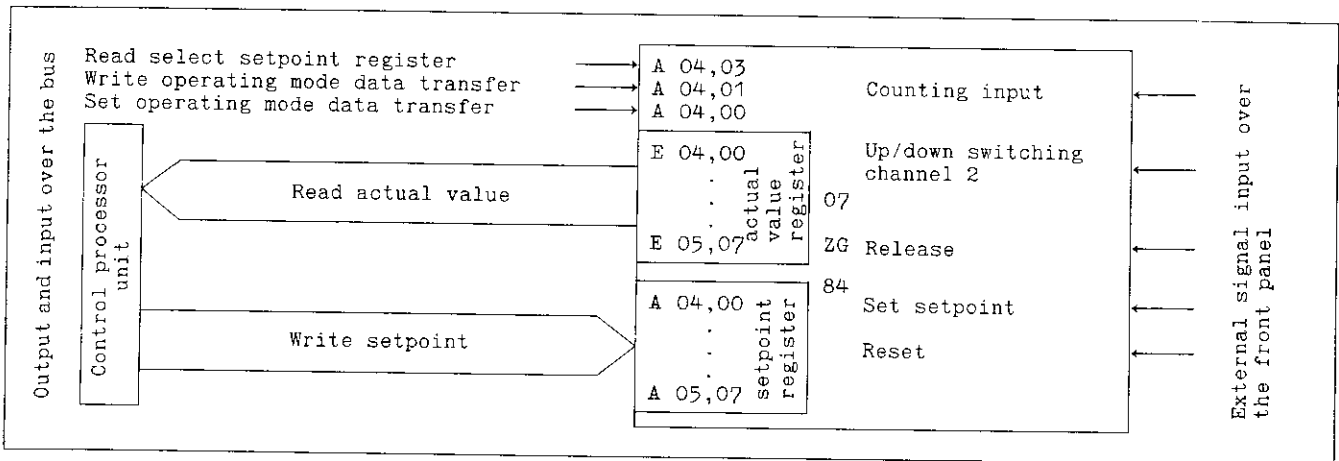
! ME
    
```

Program for word processing

```

!N E 00,02 = SPM 001
! E 00,02 = A 04,03
= A 04,00
! bE 04,00 = bA 06,00
! E 00,02 /N E 00,02
= A 05,07 =N A 04,03
! MR 001
    
```

12.5 Polling of the actual value and loading of the setpoint with this actual value



Notice

The sequency of the inputs is obligatory.

```

! E 00,01 = MA
= A 04,03      read select actual
                value register
                write select
                setpoint register
                set operating mode
                data transfer

= A 04,01

= A 04,00

! E 04,00 = A 04,00
! E 04,01 = A 04,01
! E 04,02 = A 04,02
! E 04,03 = A 04,03      setpoint assignment
                            of the actual value
! E 04,04 = A 04,04
! E 04,05 = A 04,05
! E 04,06 = A 04,06
! E 04,07 = A 04,07
    
```

```

! E 04,07 = A 04,07      setpoint assignment
                            of the actual value
! E 05,00 = A 05,00
! E 05,01 = A 05,01
! E 05,02 = A 05,02
! E 05,03 = A 05,03
! E 05,04 = A 05,04
! E 05,05 = A 05,05
! E 05,06 = A 05,06
! E 05,07 = A 05,07
! E 00,01 /N E 00,01
= A 05,07      set operating mode
                control
=N A 04,03     reset read select
                actual value
                register
=N A 04,01     reset write select
                setpoint register

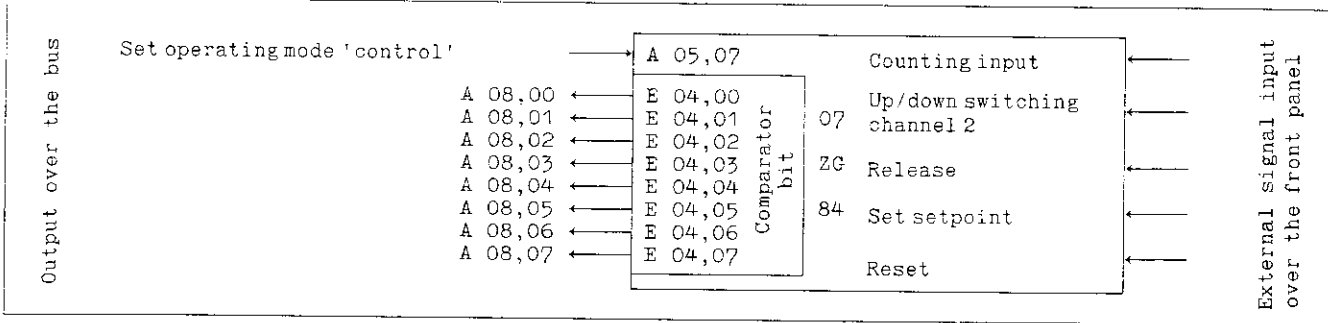
! ME
    
```

Program for word processing:

```

!N E 00,00 = SPM 000
: E 00,00 = A 04,03
= A 04,01 = A 04,00
! bE 04,00 = bA 04,00
: E 00,00 /N E 00,00
= A 05,07 =N A 04,01
=N A 04,03
: MR 000
  
```

### 12.6 Polling of the comparator bits and display on the output unit



Program for bit processing:

```

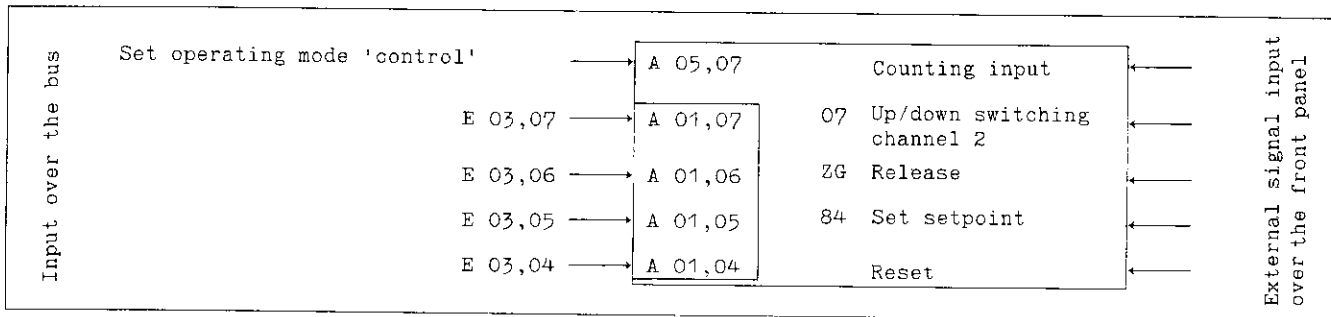
! E 00,03 = MA
= A 05,07 operating mode 'control'
: E 04,00 = A 08,00 free
! E 04,01 = A 08,01 direction of counting
! E 04,02 = A 08,02 actual value=0
! E 04,03 = A 08,03 actual value > 0
! E 04,04 = A 08,04 actual value <
pre-trip value
! E 04,05 = A 08,05 actual value >
pre-trip value
! E 04,06 = A 08,06 actual value <
setpoint
! E 04,07 = A 08,07 actual value =
setpoint
! ME
  
```

Program for word processing:

```

W0108 !N E 00,04 = SPM 002
W0110 : E 00,04 = A 05,07
W0112 ! bE 04,00 = bA 08,00
W0118 ! MR 002
  
```

### 12.7 Input Of The Signals VK2, FR, SS, R In The Program



Program for bit processing

```

! E 00,05 = MA
= A 05,07 set operating mode
control
! E 03,07 = A 04,07 up/down switching
for channel 2
! E 03,06 = A 04,06 release for
counting
! E 03,05 = A 04,05 dynamic setting
(actual value to
setpoint)
! E 03,04 = A 04,04 dynamic erase
! ME
  
```





---

ABB Schalt- und Steuerungstechnik GmbH  
Eppelheimer Straße 82  
D-6900 Heidelberg 1

Telephone (06221) 777-0  
Telefax (06221) 777-111