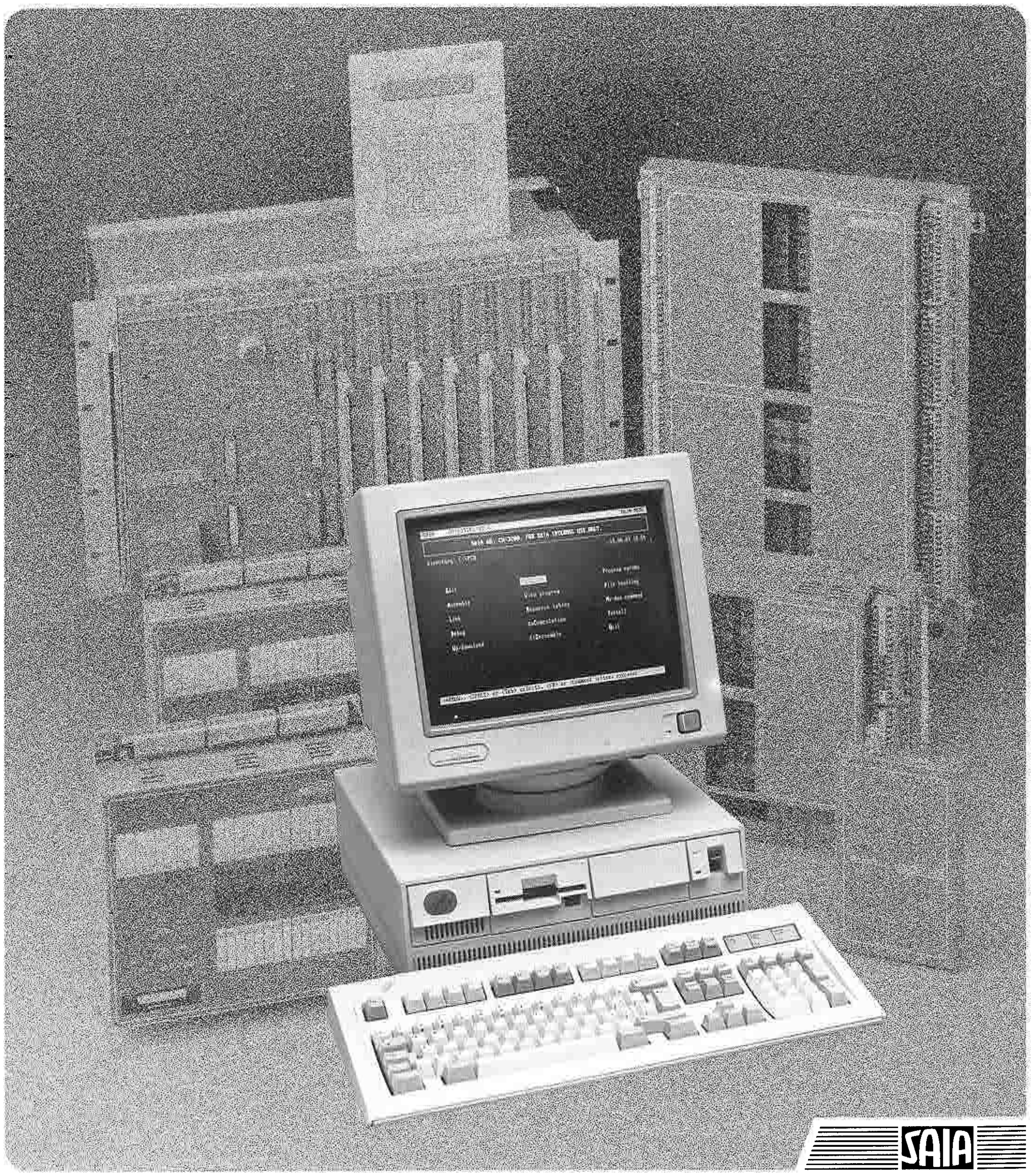
Manual of the software level 1 H

SAIA®PLC
Programmable controllers



English edition 26/731 E1

SOFTWARE LEVEL 1H

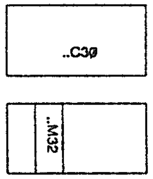
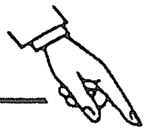
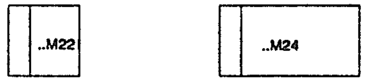
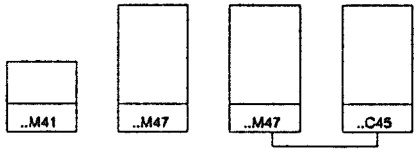
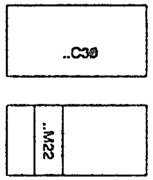
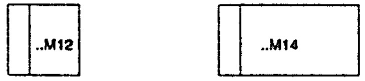
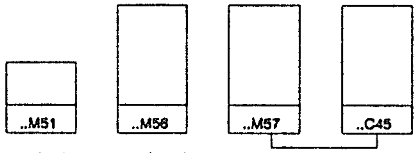
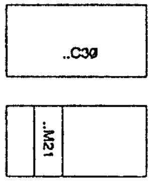
TABLE OF CONTENTS

PART D INTRODUCTION

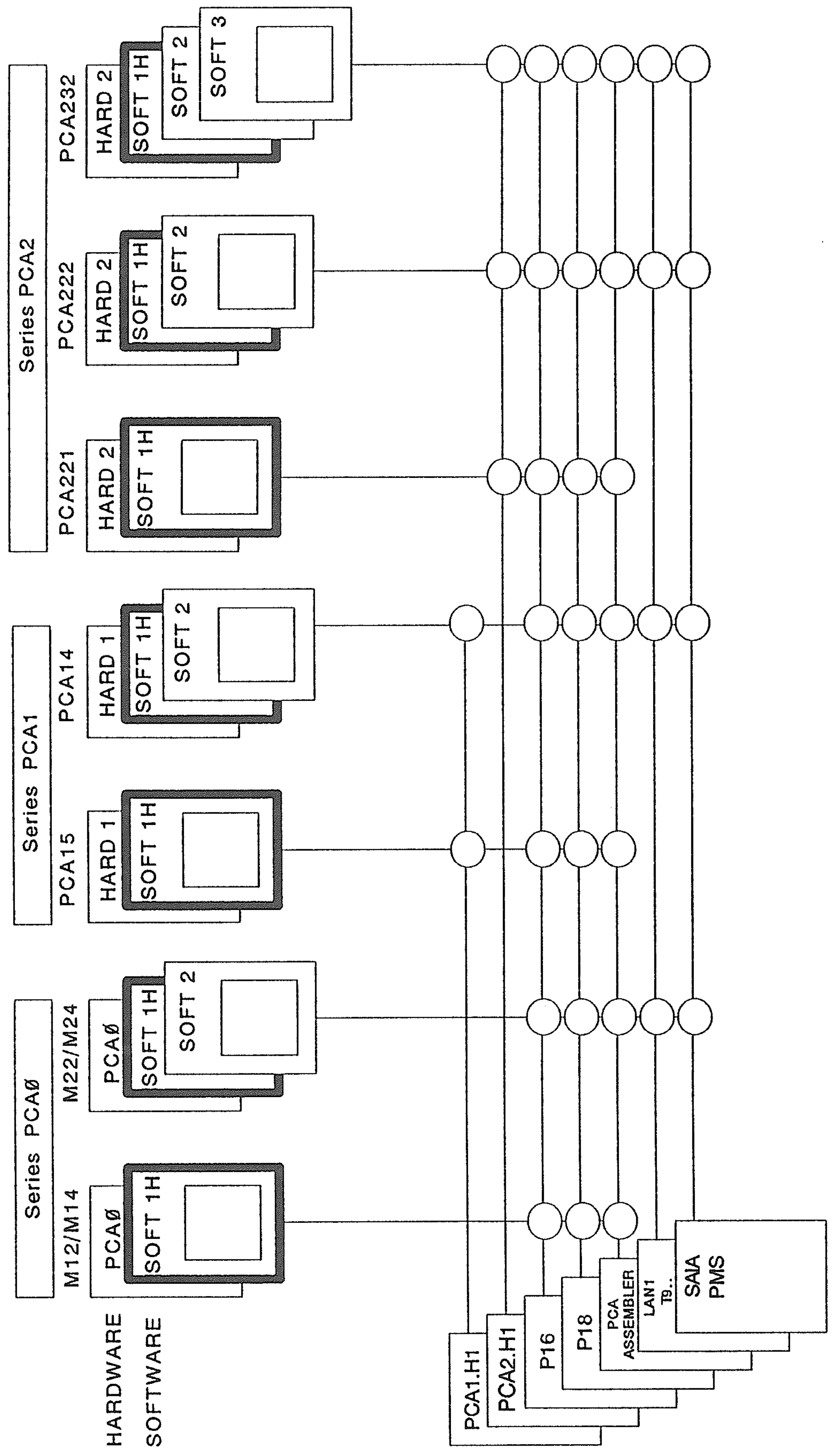
Overview of the SAIA°PLC, system family PCA
Brief overview of the manuals available
Overview of the registers of the PCA family
Basic instructions of the SAIA°PLC, software level 1
Additional instructions of software level 1H

Overview of the SAIA°PLC, system family PCA

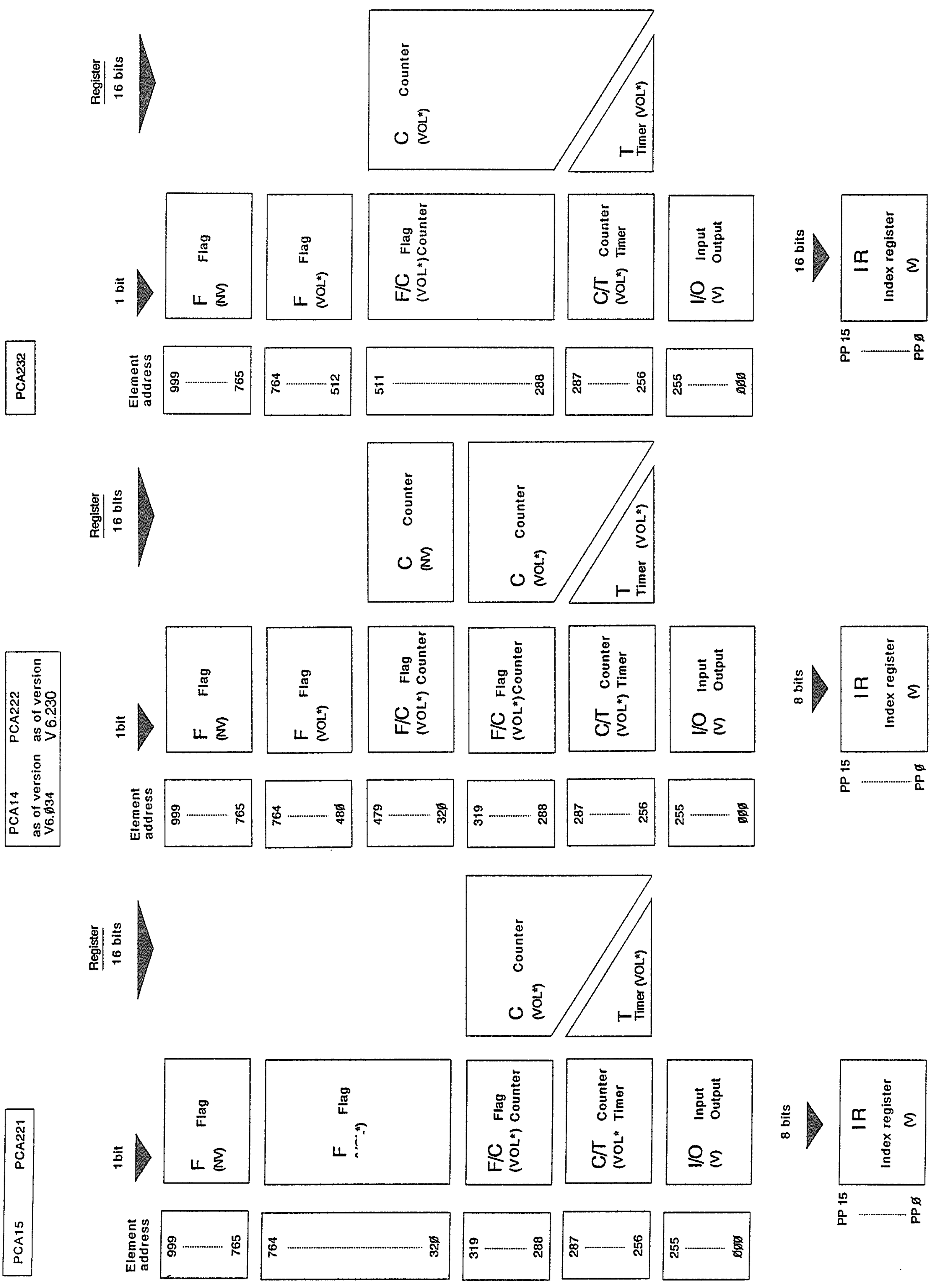
System PCA

	Series PCAØ	**Series PCA1**	**Series PCA2**
Software level 3 Software level 2 + 32 word instructions for - arithmetics, ±9 digits - data transfer - word register			PCA232  256 or 512 I/O User memory 8K program lines + 8K text character + 8K byte data
Software level 2 Software level 1H + serial data interface + date-time + data register + parameter instructions (soft-interrupt, FIFO, PID) 	Standard and OEM-versions PCAØ.M22 PCAØ.M24  max. 32 I/O max. 64 I/O User memory max. 4K program lines max. 4K text character/data	PCA14 PCA141 PCA147 PCA147 + ..C45  32 (56) 64 (112) 128 (224) I/O User memory max. 8K program lines max. 8K text character/data	PCA222  256 or 512 I/O User memory max. 8K program lines max. 8K text character/data
Software level 1H 32 basic instructions for - timer and counter functions - parallel programs and subroutines - indexing etc. 20 additional instructions for - arithmetics - data transfer - check-sum	Standard versions PCAØ.M12 PCAØ.M14  24/32 I/O 48/64 I/O User memory max. 4K program lines	PCA15 PCA151 PCA156 PCA157 + ..C45  32 (56) 64 (112) 128 (224) I/O User memory max. 4K program lines	PCA221  256 or 512 I/O User memory max. 8K program lines

Brief overview of the manuals available



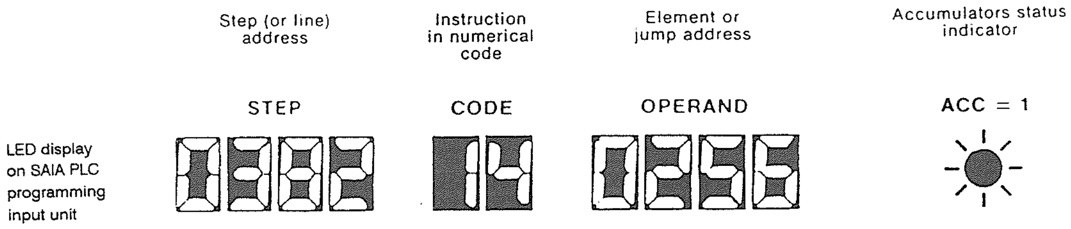
Overview of the registers of the PCA family

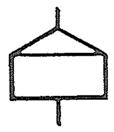
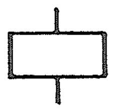
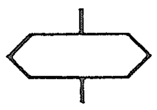
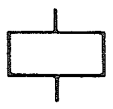


*) Volatile, can be made non-volatile with jumper NVOL

Basic instructions of the SAIA°PLC, software level (1H)

Instructions of software level (1H)


Step (or line) address
Instruction in numerical code
Element or jump address
Accumulators status indicator
STEP
CODE
OPERAND
ACC = 1
LED display on SAIA PLC programming input unit

	Numerical code	Mnemo-code	Instruction	Description
Logic Instructions 	Ø1 Ø2	STH STL	Start High Start Low	Start link with interrogation of the element according to { High / Low
	Ø3 Ø4	ANH ANL	AND High AND Low	AND linkage of accu with following elements interrogated on { High / Low
	Ø5 Ø6	ORH ORL	OR High OR Low	OR linkage of accu with the following element interrogated on { High / Low
	Ø7	XOR	Exclusive OR	Comparison of two elements
	Ø8	NEG	Negate Accu	Negate (invert) status of accu
	Ø9	DYN	Dynamic Control	Dynamic acts so that result of linkage affects accu for first cycle only
Switching Instructions 	1Ø	OUT	Set Output with Status of Accu	Set output or flag with the status of accu
	11 12	SEO REO	Set Output Reset Output	Set (latch) output or flag (if accu = 1) Reset (unlatch) output or flag (if accu = 1)
	13	COO	Complement Output	Interrogate output or flag and set it to the opposite state (if accu = 1)
Timing and Counting Instructions	14*	STR*	Set Timer	Set timer to determined value and start it running (if accu = 1)
	15*	SCR*	Set Counter	Set counter to determined value (if accu = 1)
	17 18	INC DEC	Increment Counter Decrement Counter	Increment / Decrement } content of the counter by 1 (if accu = 1)
Jump Instructions 	2Ø	JMP	Unconditional Jump	Unconditional jump to step address
	21 22	JIO JIZ	Jump if Accu is One Jump if Accu is Zero	Jump if { accu = 1 / accu = Ø } to step address
	23 24	JMS RET	Jump to Subroutine Return from Subrout.	Jump to subroutine (regardless of accu) Return from subroutine (regardless of accu)
Wait Instructions	25 26	WIH WIL	Wait if High Wait if Low	Wait whilst interrogated element is { High / Low
Auxiliary Instructions 	ØØ	NOP	No Operation	No operation
	19	SEA	Set Accu	Set accumulator status to 1
	16	SEI	Set Index	Set index register to operand value
	27 28	INI DEI	Increment Index Decrement Index	Increment / Decrement } content of index register by 1 (until to the operand value)
	29**	PAS**	Program Assignment	Assignment of the parallel programme
	3Ø	DOP	Display Operand	Display content of operand (if accu = Ø)
	31	DTC	Display Timer or Counter	Display timer or counter value (if accu = 1)

* Two line instruction (second line contains determined value)

** Two line instruction (second line contains starting address of parallel programme)

Additional instructions of software level 1H

	Mnemo code	Num. code	Instruction English	Description
Transfer instruc-tions	STR	14	Set Timer	
	SCR	15	Set Counter	
		19	2nd line	Enter 5 x 4 bits BCD
		20		Output 5 x 4 bits BCD
		21		Output 8 bits binary
		22		Output 12 bits binary
		23		Output 16 bits binary
		24		Enter 8 bits binary
		25		Enter 12 bits binary
		26		Enter 16 bits binary
		31		Transfer counter or index register counter counter
Arithmetic instruc-tions	SCR	15	Set Counter	
		27	2nd line	Add +
		28		Subtract -
		29		Multiply x
		30		Divide :
Indexing instruc-tions	SEI	16	Set Index	Set index register to preselected value
	INI	27	Increment-Index	Increment } the index re-gister by 1
	DEI	28	Decrement-Index	Decrement } the index re-gister by 1

	Mnemo code	Num. code	Operand	Description
Special instruc-tions (these are 2-line instructions)	PAS	29	18	Modification of the number of active parallel programs
	PAS	29	30 ... 38	Check sum

As evident from the previous pages, the instruction set of the entire SAIA°PLC family is completely upwards compatible on 4 levels.

Level ①, which includes 32 basic instructions, was reserved for the older series PCA13 and PCA210.

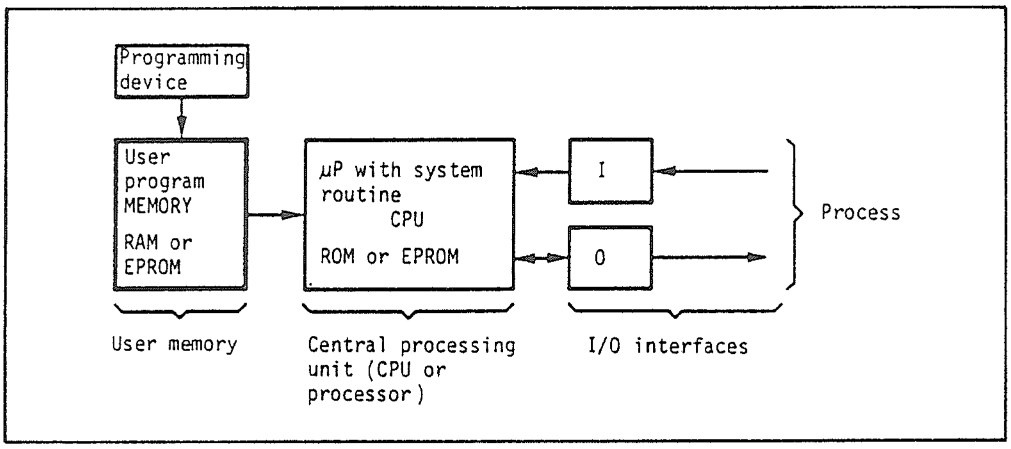
The lowest level today is (1H).
On this level, 20 additional comfortable instructions are made available for series PCA0, PCA15 and PCA221.
This intelligence level is described in this manual.

It is possible to write simple operating programs with just 5 instructions.
In order to make full use of the entire level (1H), however, timer and counter functions, parallel programs and subroutines as well as arithmetic functions can be performed.

The higher intelligence levels ② and ③ of series PCA14, PCA222 and PCA232 comprise all the basic functions, so that programs of level (1H) can still be used, should the requirements grow.

PART D INTRODUCTION

D 1 General



As has already been shown in the hardware introduction, the characteristics of the CPU are determined by the system routine of the µP system. The overall characteristic of the PLC is determined by the system routine. The user has no means of access to the system routine. Individual adaptation to the different process conditions takes place via the user program.

This user program is entered into the user memory in the problem-oriented SAIA°PLC language. It is programmed by means of various available programming devices.

The CPU (also known as the processor) is now capable of "reading" the user program and performing the instructions contained in it, such as e.g. interrogation of input states, their linking and transmission of the result to outputs, thus controlling the process in the desired manner.

According to the task in hand, the program can be prepared as required in accordance with one of the following control descriptions:

- Ladder diagram (circuit diagram)
- Logic diagram (signal flow diagram)
- Flowchart (sequence diagram)
- Function chart in accordance with DIN 44719 or GRAFCET (step control)

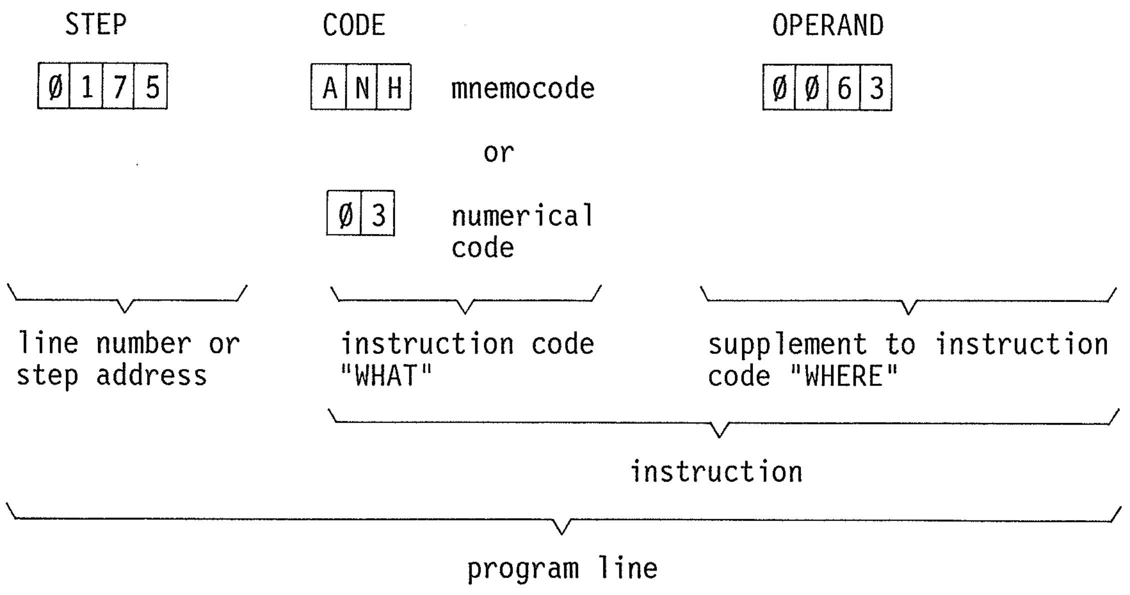
The different programming methods may be combined with one another.

D 2 The program line

Each instruction in the user program comprises 1 program line (in certain cases 2 or 1Ø lines). In addition to the line number or step address (STEP), a line also contains the instruction code (CODE) and the operand (OPERAND). The instruction code states "WHAT" sort of instruction is to be performed, whilst in the operand it is determined "WHERE" this instruction is to take effect.

Each program line comprises 16 bits, so that 1K = 1Ø24 program lines can be stored in a 16 K bit user memory (e.g. EPROM 2716).

Structure of the program or instruction line:



STEP	The location of the instruction in the user memory is defined by the <u>line number</u>. Decimal numbering from Ø...2Ø47 (2K) or 4Ø95 (4K).
CODE	Depending on the programming unit the <u>instruction code</u> can be entered as a <u>3-character mnemocode</u> or as a <u>numerical code from Ø to 31</u>. The mnemocode is an abbreviation of the English term for the corresponding instruction. Therefore, it can be easily remembered and is internationally understood.
OPERAND	Here, by means of jump instructions, the <u>element address</u> (input, output, timer, counter or flag) or the <u>target address</u> (line number) is entered.

Timer and counter instructions comprise 2 program lines. In the second line, the corresponding timer or counter value appears in the operand.

D 3 The operands

As has already been mentioned in the previous description, the operands are the so-called element addresses or step addresses (line number).

D 3.1 Element addresses

All addressable elements (inputs, outputs, timers, counters, non-retentive and retentive flag memories) are numbered decimally from Ø...999.

- The inputs and outputs in the form of interface modules can be plugged into the basic unit of the PLC. They are addressed either from the installation location (PCAØ and PCA1) or by means of a DIL switch (PCA2).

  The address range can be assigned as required, alternatively however, only by means of an input or an output module. (An exception here is the PCA2.C3Ø extension rack, by means of which it is possible to attain 256 I + 256 O = 512 I + O).

  It is only possible to interrogate the signal states of inputs.

  Outputs can be set (switched on) and reset (switched off) and their signal status interrogated (exceptions for certain modules).

- Timers and counters are programmable registers which are located in the CPU module. They can be used alternately as timers or counters.

- Non-retentive and retentive flags are 1 bit storage cells which can be treated like outputs, e.g. they can be set or reset, and interrogated as to their signal state. Accordingly non-retentive and retentive flags are suitable for the storage of any information.

  On the hardware side, these are also located in the CPU module in the form of a separate RAM memory. Because this RAM memory is provided with a back-up battery, the 235 retentive flags retain their information - even in the event of a power failure - for in excess of 1 month. Although the 477 non-retentive flags are likewise provided with a battery back-up, they are reset to the "L" signal state when the PLC is switched on. Therefore they act as zero voltage resetting flags.

  Insertion of hardware bridge NVOL in the CPU module enables all 712 flags to become retentive, as well as all timers and counters.

D 3.2 Step addresses

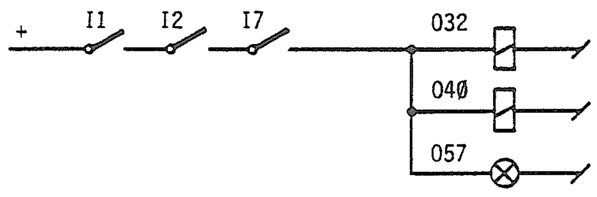
The second type of operands are the step addresses or line numbers. These are required in connection with jump instructions. In this combination it can be determined to which point in the user program the jump is to be made. In order to cover the entire user memory, all step addresses from Ø to 2Ø47 can be used as operands.

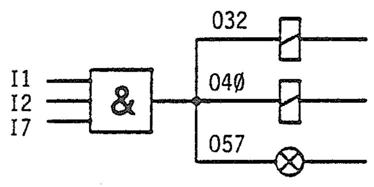
Due to 2-line jump instructions provided for the more efficient CPUs (from intelligence level (1H) on, the step address range is increased to 8191 (8K).

D 4 Further definitions

D 4.1 The linkage line

An example using the ladder or logic diagram:



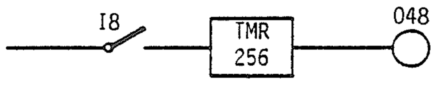


Program

```
        .
        .
     OUT 51  ⎫
     STH  1  ⎪
     ANH  2  ⎪
     ANH  7  ⎬  Linkage line
     OUT 32  ⎪
     OUT 4Ø  ⎪
     OUT 57  ⎪
     STH  9  ⎭
        .
        .
```

A linkage line is a section of a program and consists of several instruction lines. It is a self-contained linkage of elements. It normally commences with a start instruction (STH or STL). The linkage is considered successful when the end result = 1, e.g. when, in our example outputs 032, 04Ø and 057 are activated (switched on).

An example with a timer:



Program

```
        .
        .
     STH   8  ⎫
     STR 256  ⎬  1st linkage line
      ØØ  5Ø  ⎭
     STH 256  ⎫  2nd linkage line
     OUT  48  ⎭
        .
        .
```

As the adjacent program shows, the above function comprises 2 linkage lines. In the first linkage line, input I8 is interrogated and on the basis of its signal state, the timer is set (comparable with a time relay control circuit). In the 2nd linkage line the timer signal state is transferred to output 048 (comparable with a time relay load circuit).

D 4.2 The accumulator = ACCU

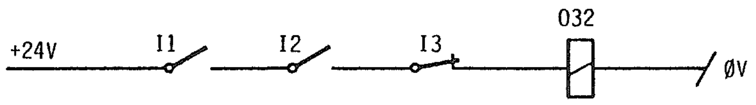
This memory is also located in the CPU and consists of a single storage position which can assume the logical status Ø or 1.

In order to process a complete linkage as far as an action instruction, the existing intermediate result of the linkage must be stored in the ACCU. At the end of a linkage the end result is present in the ACCU (Ø or 1). On the basis of this result the corresponding element (e.g. an output) is either not activated (ACCU = Ø) or activated (ACCU = 1).

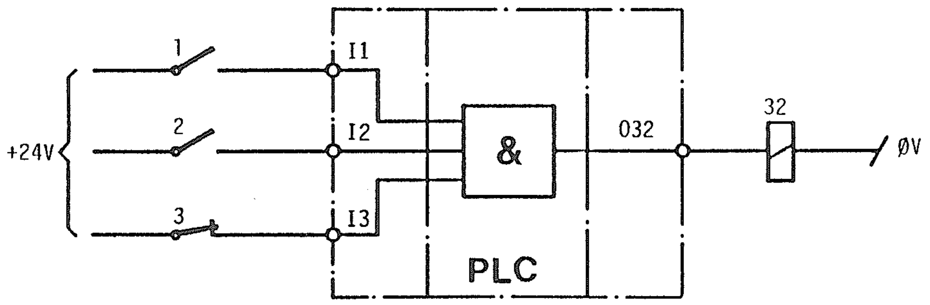
By means of the result stored in the ACCU following instructions can also be skipped (e.g. see jump instructions).

D 4.3 Normally-open/normally-closed contacts or high/low

A contact linkage as shown in the schematic diagram below is wired exactly in accordance with the schematic diagram:

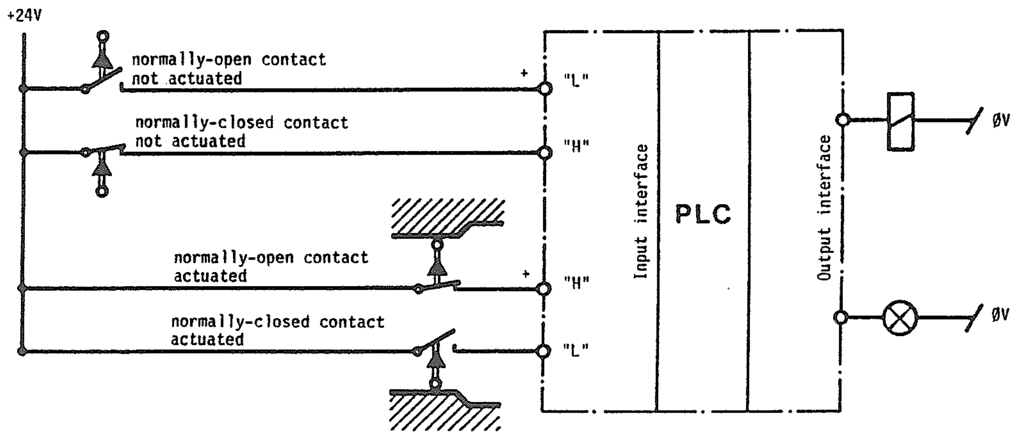


If this problem is solved with a PLC, then each contact is led to an input. Linkage then takes place in the PLC processor and not by means of wiring. The above linkage suitable for a PLC is shown in the figure below:



A PLC cannot determine whether normally-open or normally-closed contacts are connected to its inputs. It does however determine whether a high signal (H) or a low signal (L) is applied to the input. These signal levels are accurately defined in the hardware section for each input interface. For a 24V input in source mode (normal case), it can be stated as a simplification that:

- where +24V is applied to the input ----------> signal state "H" is produced
- where ØV is applied to the input ----------> signal state "L" is produced



+24V
normally-open contact not actuated
"L"
normally-closed contact not actuated
"H"
normally-open contact actuated
"H"
normally-closed contact actuated
"L"
Input interface
PLC
Output interface
ØV
ØV

However the widely used method of representation using contact symbols can be used for programming the PLC as long as the following rules are observed. The following applies to source mode, e.g. with the contact at +24V:

- If a contact in its closed state assists a successful linkage, then the contact concerned must be interrogated if high or linked (normal case).

- If a contact in its open state assists a successful linkage, then the contact concerned must be interrogated if low or linked.

In addition to mechanical contacts these same considerations also apply to proximity switches and light barriers.

For the sake of uniformity it is standard practice to employ only normally open contacts. This is indeed correct for approximately 9Ø% of the cases. With some special applications, however, the use of normally closed contacts is unavoidable for safety reasons. As a simplification it can be stated:

- A procedure must always be started by means of "apply voltage", while it is always stopped by switching off the supply.

In order to take these prior considerations into account, the interrogation and linkage instructions are always in pairs:

STH	STL
ANH	ANL
ORH	ORL
(WIH)	(WIL)
(JIO)	(JIZ)

This means that the input signals can in each case be interrogated if "H" or "L". Where the type of interrogation is in agreement with the actual signal status at the input, this is then processed as a logical "1" in the ACCU.

Signal states do not just apply to inputs but to all addressable elements:

Inputs	"H":	+24V applied (standard, source mode)
	"L":	ØV applied (standard, source mode)
Outputs	"H":	output set = transistor on-state
	"L":	output not set = transistor blocked
Non-retentive/	"H":	set
retentive flags	"L":	not set or reset
Timers	"H":	timer has been set and runs
	"L":	timer has not been set or has run down
Counters	"H":	counter has been set and register content > Ø
	"L":	counter has been not set or register content = Ø

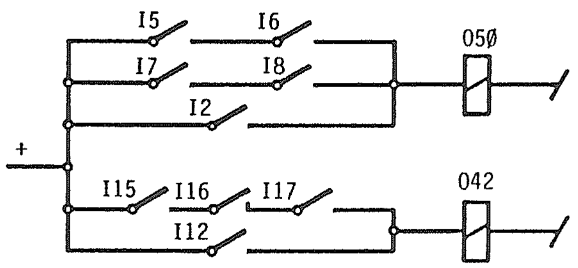
D 5 Programming methods

A programmable control provides a high degree of flexibility concerning the design and adaptation of the program. However as with traditional controls, the PLC does not take the control designer's responsibility of producing a clear description of the control task beforehand. According to type of process and the preference of the designer, this description is produced in the following manner.

Because the SAIA°PLC instructions set is so versatile, the most diverse control descriptions are suitable for use as models for a PLC program.

D 5.1 Programming by ladder diagram

Model:



Program (instruction list):

ADDR	NC	MNC	OPRD
10	01	STH	5
11	03	ANH	6
12	05	ORH	7
13	03	ANH	8
14	05	ORH	2
15	10	OUT	50
16	01	STH	15
17	03	ANH	16
18	03	ANH	17
19	05	ORH	12
20	10	OUT	42
21	20	JMP	10

Characteristics:

----> Instructions required STH, STL, ANH, ANL, ORH, ORL, OUT.

- The linkage is shown by contact symbols. The outputs receive the results of the prior AND/OR linkages.
- All program parts circulate in a cyclic fashion. This is termed a pure circulating program because it contains no wait and no conditional jump instructions.

Advantages:

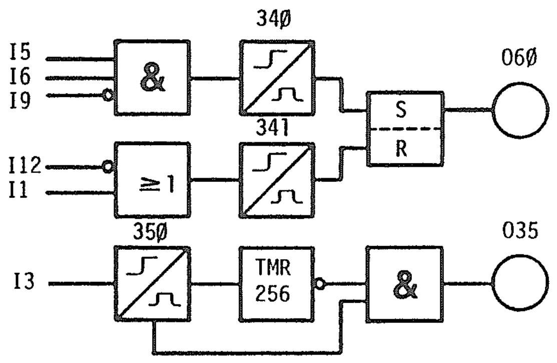
- Simple program preparation with only 7 instructions for users of ladder diagrams.
- Well suited to monitoring tasks.
- Different linkage lines can be strung together almost as desired.

Disadvantages:

- Less suitable for sequential processes (sequence controls), because all parts which are inactive at a given time must be interlocked.
- Mor complex controls become extremely complicated in the ladder diagram form.
- Restricted instruction set.
- In the case of large programs a long reaction time.

D 5.2 Programming by logic diagram

Model:



Program (instruction list):

```
ADDR  NC  MNC  OPRD
  30  01  STH     5 ⎫
  31  03  ANH     6 ⎪
  32  04  ANL     9 ⎪
  33  09  DYN   340 ⎪
  34  11  SEO    60 ⎬ RS
  35  02  STL    12 ⎪
  36  05  ORH     1 ⎪
  37  09  DYN   341 ⎪
  38  12  REO    60 ⎭
  39  01  STH     3 ⎫
  40  09  DYN   350 ⎪
  41  14  STR   256 ⎪
  42  00  00     20 ⎬ Timer
  43  02  STL   256 ⎪
  44  03  ANH   350 ⎪
  45  10  OUT    35 ⎪
  46  20  JMP    30 ⎭
```

Characteristics:

-----> Additional available instructions SEO, REO, COO, STR, SCR, NEG, DYN etc.

- The linkage is shown with function oriented logic symbols enabling the signal flow to be followed up in a similar fashion to the ladder diagram.

- All program parts circulate in a cyclic fashion (pure circulating program).

Advantages:

- The outputs can be set accumulating or non-accumulating.

- Well suited for monitoring tasks and pure logic programs.

- Since a large part of the instructions set can be used, even more complex problems are easily realizable.

- Different linkage lines can be strung together almost as desired.

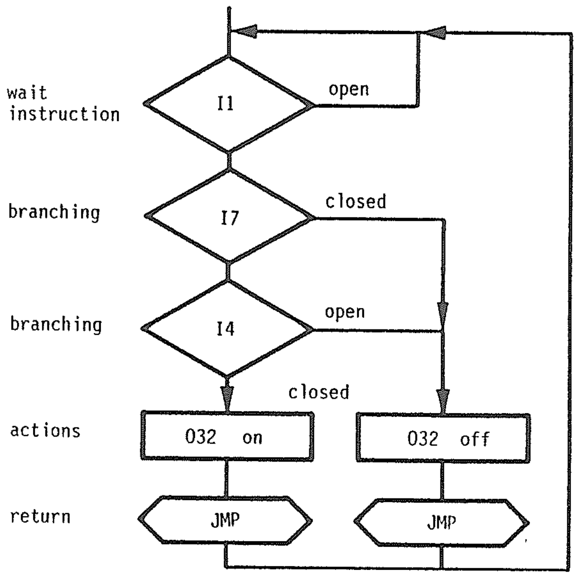
Disadvantages:

- Less suitable for sequential processes (sequence controls), because all parts which are inactive at a given time must be interlocked.

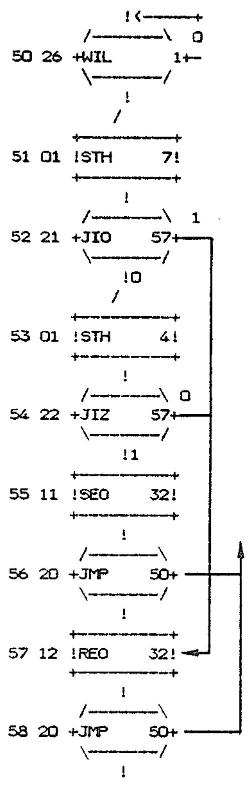
- In the case of large programs a long reaction time.

D 5.3 Programming by flowchart

Model:



Program in flow-chart form:



Program in instruction list form:

ADDR	NC	MNC	OPRD
50	26	WIL	1
51	01	STH	7
52	21	JIO	57
53	01	STH	4
54	22	JIZ	57
55	11	SEO	32
56	20	JMP	50
57	12	REO	32
58	20	JMP	50

Characteristics:

------> Special instructions WIH, WIL, JIO, JIZ

- Wait loops can be formed; these can be retained by the processor until the conditions for continuation are fulfilled.

- Branchings with conditional jump instructions are possible.

- This programming method is suitable for sequential processes (sequence controls).

Advantages:

- Simple programming, in accordance with the functions in the sequential process.

- This programming method is easily understood by the process engineers.

- Due to the wait loops, the program runs only as fast as the process itself. This results in simple commissioning and rapid trouble-shooting.

- Very short reaction times. They are very often determined only by the delay of the input interface.

Disadvantages:

- Monitoring functions must be incorporated in separate parallel programs.

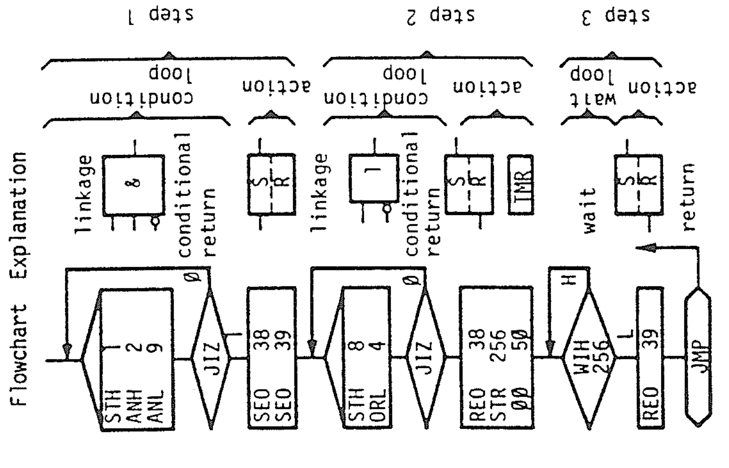
D 5.4 Combined programming by flowchart / logic or in accordance with Grafcet

The universal program language of the SAIA°PLC permits the preparation of a practice related step program by combining instructions.

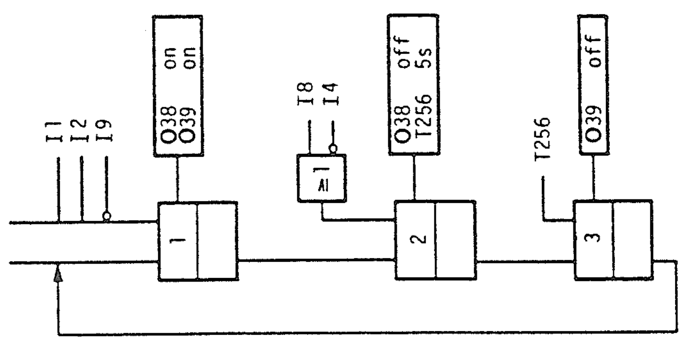
Documentation in accordance with flowchart:

```
                         STEP1
       +-----------+
60 01  !STH      1!<---+
61 03  !ANL      2!    |
62 04  !ANL      9!    |
       +-----------+    |
             !          |
       /-----------\    0
63 22  +JIZ     60+----+
       \-----------/
             !1
       +-----------+
64 11  !SEO     38!
65 11  !SEO     39!
       +-----------+
             \ /
             / \
                         STEP2
       +-----------+
66 01  !STH      8!<---+
67 06  !ORL      4!    |
       +-----------+    |
             !          |
       /-----------\    0
68 22  +JIZ     66+----+
       \-----------/
             !1
       +-----------+
69 12  !REO     38!
70 14  !STR    256!
71 00  ! 00     50!
       +-----------+
             !
                         STEP3
       /<----------\ 1
72 25  +WIH    256+-
       \-----------/
             !
       +-----------+
73 12  !REO     39!
       +-----------+
             !
       /-----------\
74 20  +JMP     60+-->
       \-----------/
```

Model in accordance with flowchart:



Model in accordance with DIN function chart:

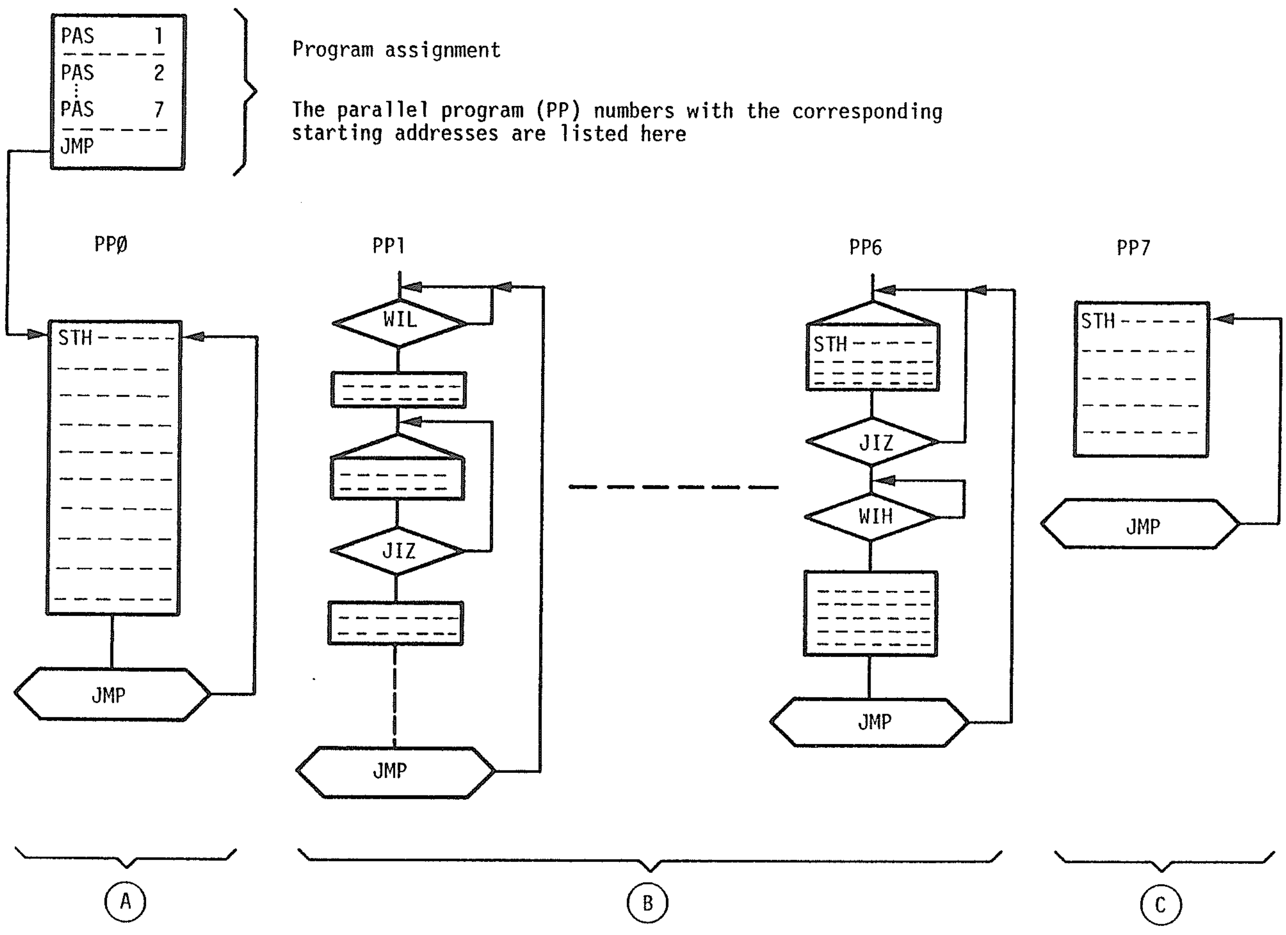


Features: Flexibility is further enhanced owing to the use of logic instructions.

D 5.5 Programming with parallel programs

Optimal conditions are attained with the use of parallel programs. Up to 16 parallel programs can be employed which are independent of one another and which run asynchronously. This means that different sequential functions of a machine (e.g. an automatic assembly machine) can be accomodated in different programs in accordance with the sequence plan and run step by step with the advance of the process.
Monitoring and continuously active functions are accomodated in a parallel circulating program.

Program structure:


PAS 1
PAS 2
PAS 7
JMP
Program assignment
The parallel program (PP) numbers with the corresponding starting addresses are listed here
PPØ
PP1
PP6
PP7
STH
WIL
JIZ
WIH
JMP
A
B
C

Ⓐ: PPØ (parallel program Ø) is a pure circulating program (without wait loops) with monitoring and continuously active functions.

Ⓑ: PP1 to (e.g.) PP6 are parallel sequence programs in the flowchart of different asynchronously circulating sequential functions.

Ⓒ: PP7 is a brief, pure circulating program which contains a few functions requiring extremely short reaction times.

Characteristics:

-----> Instruction PAS for program assignment.
See program structure for other characteristics.

Advantages:

- Sequential runs (with wait loops) and continuously active functions (monitoring, pure logic etc.) can be separately programmed in the most suitable way.

- Only that number of parallel programs are assigned as are required (max. 16).

- By means of appropriate software-interrupt facilities sequential programs can be stopped or started from the beginning at any time (PAS 18 instruction).

Disadvantages:

- In order to maintain rapid reaction times for continuously active functions, it is advantageous to process these in a separate short circulating program.

D 5.6 Programming with subroutines

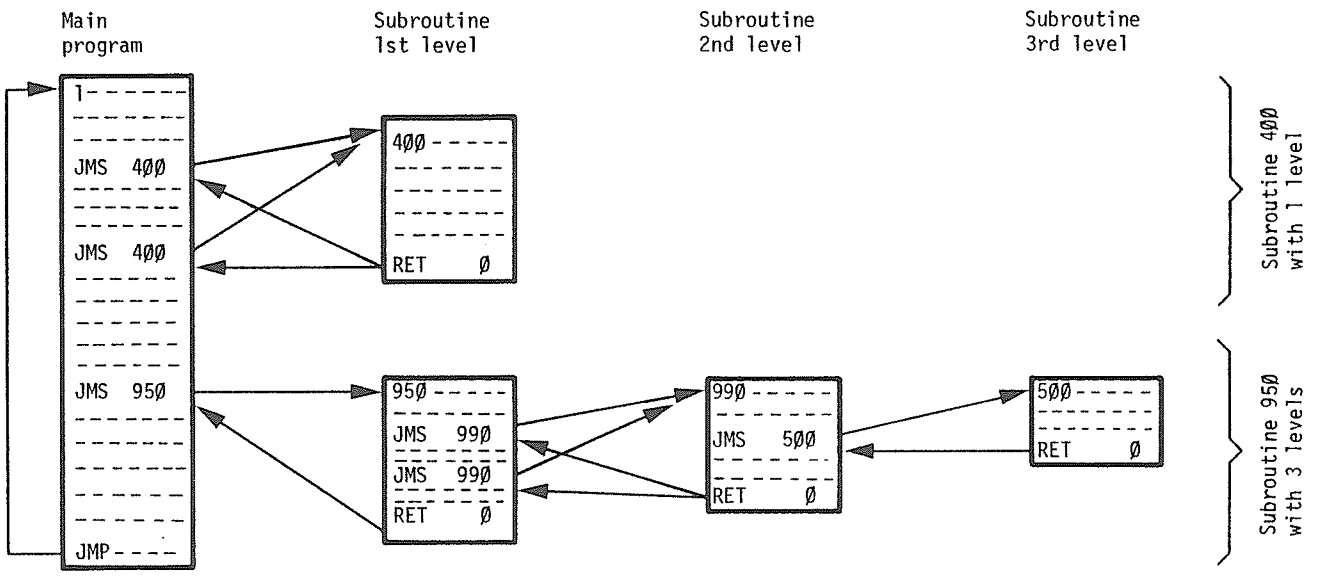
-----> Instructions required JMS and RET.

Frequently repeated parts of programs may be entered as subroutines.

Subroutines can be used in all the previously described program techniques, e.g. ladder diagram, logic diagram, flowchart or when using parallel programs.

Subroutines save memory capacity and programming time. Also, by using subroutines, programs are clearly subdivided into functional blocks (structured programming).

As many subroutines as desired can be activated, each subroutine available in up to three levels. It is only necessary to ensure that a jump is not made simultaneously into the same subroutines from different PPs.



D 5.7 Address indexing (series processing)

-----> Instructions SEI, INI, DEI

Whenever several elements (inputs, outputs, flags or timers) have to be processed in the same fashion (e.g. in monitoring circuits or with shift registers), the result is normally long programs of low linkage depth. If for instance, all 235 retentive flags are to be reset by pressing a pushbutton, <u>236 program lines</u> would be required. By using address indexing the same task can be programmed with <u>only 5 program lines</u>.

Without indexing:

```
┌─►STH     1
│  REO   765
│  REO   766
│  REO   767
│  REO   768
│  REO   769
│  REO   770
│       .
│       .
│  REO   997
│  REO   998
│  REO   999
└─ JMP
```

With indexing:

```
┌──►SEI    0
│ ┌►STH    1
│ │ REO 1765
│ │ INI  234
│ └─JIO
└───JMP
```

The indexing loop will run through 1 + 234 times, the element address (provided with 1000) being increased by 1 each time.

Notes:

NOP No operation

NOP: No Operation ---> no operation

Instruction format:

Instruction code		Operand
Mnemo code	Numerical code	
NOP	ØØ	Ø

Although this instruction is processed by the processor, no functions are initiated. Its purpose is to create spare places for additions to the program and to fill gaps in the program.

Program lines which become superfluous after an alteration can be overwritten (cleared) with NOP. Successive NOPs are obtained by the multiple actuation of the "Enter" pushbutton on the programming unit.

SEA Set ACCU = 1

SEA: Set Accumulator = 1 ---> ACCU is set to 1

Instruction format:

Instruction code		Operand
Mnemo code	Numerical code	
SEA	19	Ø

The ACCU is set unconditionally to 1 with SEA. Therefore SEA is used before instructions which are only performed when ACCU = 1, e.g. before DTC.

Part F
Example 8

E 6 Indexing

SEI
INI
DEI

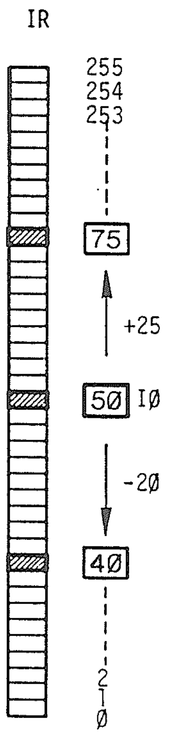
Element addresses - indexing (series processing)

SEI:	Set Index --->	set the index register to the initial value in accordance with operand
INI:	Increment Index --->	increase index register by 1 up to the upper final value in accordance with operand
DEI:	Decrement Index --->	reduce index register by 1 down to the lower final value in accordance with operand

It is frequently necessary for series of inputs, outputs, flags, timers or counters must be treated in the same way (for example resetting of all retentive flags in accordance with D 5.7). In cases like this, long and time-consuming programs can be drastically shortened with the help of address indexing.

This is facilitated by the index register IR, which is a type of counter register having a capacity from 0 to 255*. In three instructions enable the register content to be set or altered to the desired limiting value.

Example:



INI 75 ↑	Increases the index register by 1 until upper final value 75 is reached. The ACCU is then 0.
SEI 50	Sets index register to initial value I0 = 50. The ACCU is then 1.
↓ DEI 40	Reduces index register by 1 until the lower final value 40 is reached. The ACCU is then 0.

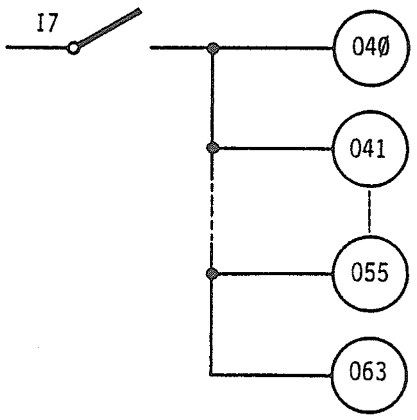
---

*) Each parallel program has its own index register, providing a total of 16. These registers lose their contents in the event of a power failure.

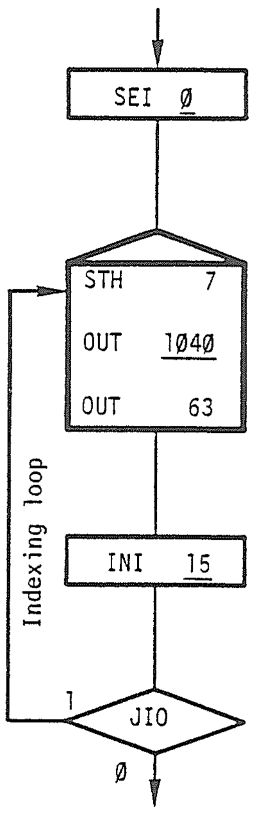
The CPU of PCA232 has index registers with a capacity of 1K (0...1023) (see example d) at the end of this chapter).

All element operands which are shown in the description in Part E with (i) can be indexed. The processor adds the existing reading of the index register to the element address shown in the operand.

An example should clarify this:



By activation of I7, outputs 040 to 055 and additionally output 063 are to be activated.



The index register is set to initial value 0.

Interrogation of input I7.

Indexing is initiated after output 040 by addition of the number 1000. In the first loop run, address 040 is not altered because the index register is first of all 0 (SEI 0). With the following runs the register content is each time increased by 1 by means of the instruction INI, so that outputs 041, 42, 43 to 55 are set one after the other.

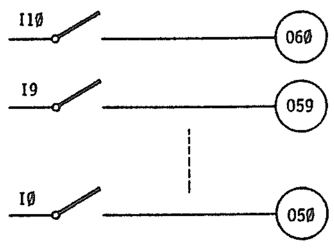
Output 063 is directly addressed (without 1000), so the index register has no effect on this.

INI effects the increase of the register by 1 per run up to the value 15. As long as the register is <15, the ACCU has the value 1 after processing INI 15.

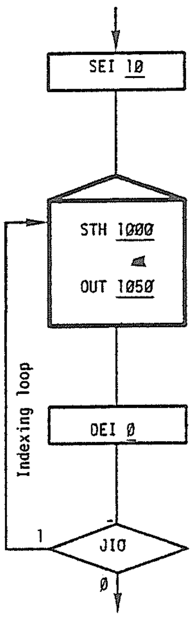
Therefore the conditional jump is performed until the loop is executed 0 to 15 times, e.g. 16 times, and all 16 outputs from 040 to 055 have been processed.

Part F
Examples: 15, 16, 17

Second example:



Inputs I1Ø to IØ are to activate outputs 06Ø to 05Ø. The action is to take place in descending order by means of DEI.



The index register is set to the initial value 1Ø (1Ø - Ø = 1Ø or 6Ø - 5Ø).

Both inputs and outputs are to be indexed; this taking place by the addition of 1ØØØ. The lowest addresses in the series are entered since initially the register content 1Ø is added. Therefore, firstly I1Ø is activated with 06Ø, then I9 with 059 down to IØ with 05Ø.

DEI causes the register value to be reduced by 1 for each run, until the given value Ø is reached.

When register value Ø = DEI Ø the ACCU is likewise Ø, so the indexing loop is now vacated.

Instruction formats:

Instruction code		Operand	
Mnemo code	Numerical code	Description	Range
SEI	16	Initial value (IØ) of the index register	IØ = Ø...255
INI	27	Upper final value for incrementing indexation	1...255 [1]
DEI	28	Lower final value for decrementing indexation	Ø...254 [2]

1) By applying INI, the next index register value after 255 is Ø
Ø (...254, 255, Ø, 1...)

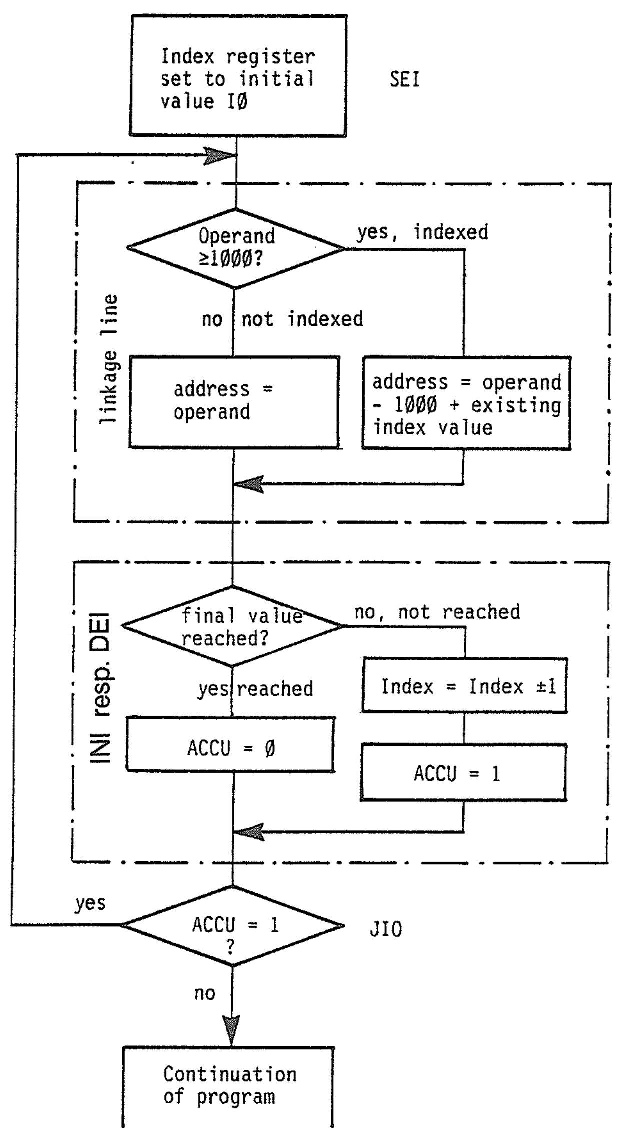
2) By applying DEI, the next index register value after Ø is 255
(...1, Ø, 255, 254).

Truth table:

Instruction	Effect on ACCU
SEI	Instruction sets ACCU = 1
INI	If register value ≠ operand ---> ACCU = 1 If register value = operand ---> ACCU = Ø
DEI	If register value ≠ operand ---> ACCU = 1 If register value = operand ---> ACCU = Ø

Summary of indexing

Indexing scheme:



a) Indexing with INI

- Register generally set to value Ø with SEI.
- The lowest addresses (+1ØØØ) of the elements to be indexed are used.
- The upper final value for INI is obtained from the difference between highest and lowest address which are to be indexed.

b) Indexing with DEI

- Register generally set to value X with SEI.
  X = difference between highest and lowest addresses which are to be indexed.
- The lowest addresses (+1ØØØ) of the elements to be indexed are used.
- Generally Ø is used for the lower final value for DEI.

Further programming facilities offered using indexing

This section only deals with indexing used for series processing. It is also possible however for the index register to be used so that e.g. subroutines act on other elements according to the status of the index register.

Example:

Closing I1 causes 045 to flash, closing I2 causes 05Ø to flash

Main program

```
******************** MAIN PROGRAM
ADDR  NC  MNC  OPRD
700   01  STH     1
701   22  JIZ   704   -->
702   16  SEI     0
703   23  JMS   710   ==>
704   01  STH     2
705   04  ANL     1
706   22  JIZ   700   -->
707   16  SEI     5
708   23  JMS   710   ==>
709   20  JMP   700   -->
```

Subroutine

```
================== SUBROUTINE
ADDR  NC  MNC  OPRD
710   02  STL   256
711   14  STR   256
712   00  00      3
713   13  COO  1045
714   24  RET     0   -->
```

SEI, INI, DEI   Additional functions

SEI(16) iii   Setting the index register

a) iii = Ø...255
The index register is set to value iii of the operand.

b) iii = 256...319* (for PCA232: 256...511)
The index register is loaded by the contents of the addressed T/C = iii.

INI(27) iii   Incrementing index register by 1
DEI(28) iii   Decrementing index register by 1

a) iii = Ø...255
The index register is incremented or decremented as far as the given numerical value iii of the operand.

b) iii = 256...319* (for PCA232: 256...511)
The given value is located in the register of the addressed T/C = iii.

Please note:
- All PLCs have 16 index registers, one for each parallel program.
- The counting capacity of these registers is limited to 255. If counter contents are to be transferred to the index register, it should not exceed 255. The maximum capacity of the IR of PCA232 is 1Ø23 (see example d).

General examples:

a) C267 = 1Ø2
Following instruction SEI(16) 267 the index register contents will be 1Ø2 as well.

b) C256 = 44
Following instruction INI(27) 256 the increment limit value will be 44 as well.

c) C26Ø = 1ØØ, IR = 4
Following instruction SEI(16) 1256 the index register value will be 1ØØ (double indexing).

d) If you need to load values higher than 255, you have to load a counter first.

Example:
```
SCR  28Ø  ⎫ The value 8ØØ is
 ØØ  8ØØ  ⎬ loaded into the
SEI  28Ø  ⎭ index register
```

*) The operand range of counters depends on the hardware and firmware in use (see overview on page III).

E 7 PAS-instructions

PAS Ø ... PAS 15 Assignment of parallel programs (PP)

PAS: Program Assignment ---> assignment of the parallel program

Instruction format (two-line instruction):

Instruction code		Operand		
Mnemo code	Numerical code	Description	Range	
PAS	29	Program number, consecutively from	Ø...15	1st line
---	ØØ	Program start address	Ø...819Ø	2nd line

Where several programs run in parallel (maximum 16), then this information must be passed to the CPU right at the program start point. This is done by the assignment of the start addresses of all parallel programs which are to be processed, with the two-line instruction PAS.

The listing of the parallel programs in the assignment part must be continuous (with no numbers left out) from program number 1 in ascending order. The assignment part may usually only be run through once, immediately after switching on the PLC.

PPs may also be "reassigned" with the instruction PAS Ø...15. It is possible, for example, to reassign PP3 from start address 3ØØ to start address 4ØØ in any programm section.

Parallel program Ø does not have to be assigned. It is activated directly after assigning all parallel programs.

Chapter D 5.5 gives details of the PP design and structure. Processing of the individual PP is performed using the "Time Sharing" method. The processor changes from one PP to the next in accordance with precisely defined conditions.

Conditions causing a PP change

The following instructions cause a PP change:

- WIH, WIL (if the wait condition is fulfilled)
- JMP, JIO, JIZ, JMS, RET
- and every second or third STH or STL instruction

PAS Ø...15 is always executed irrespective of the ACCU status and does not alter the ACCU status.

Part F
Examples: 13, 18

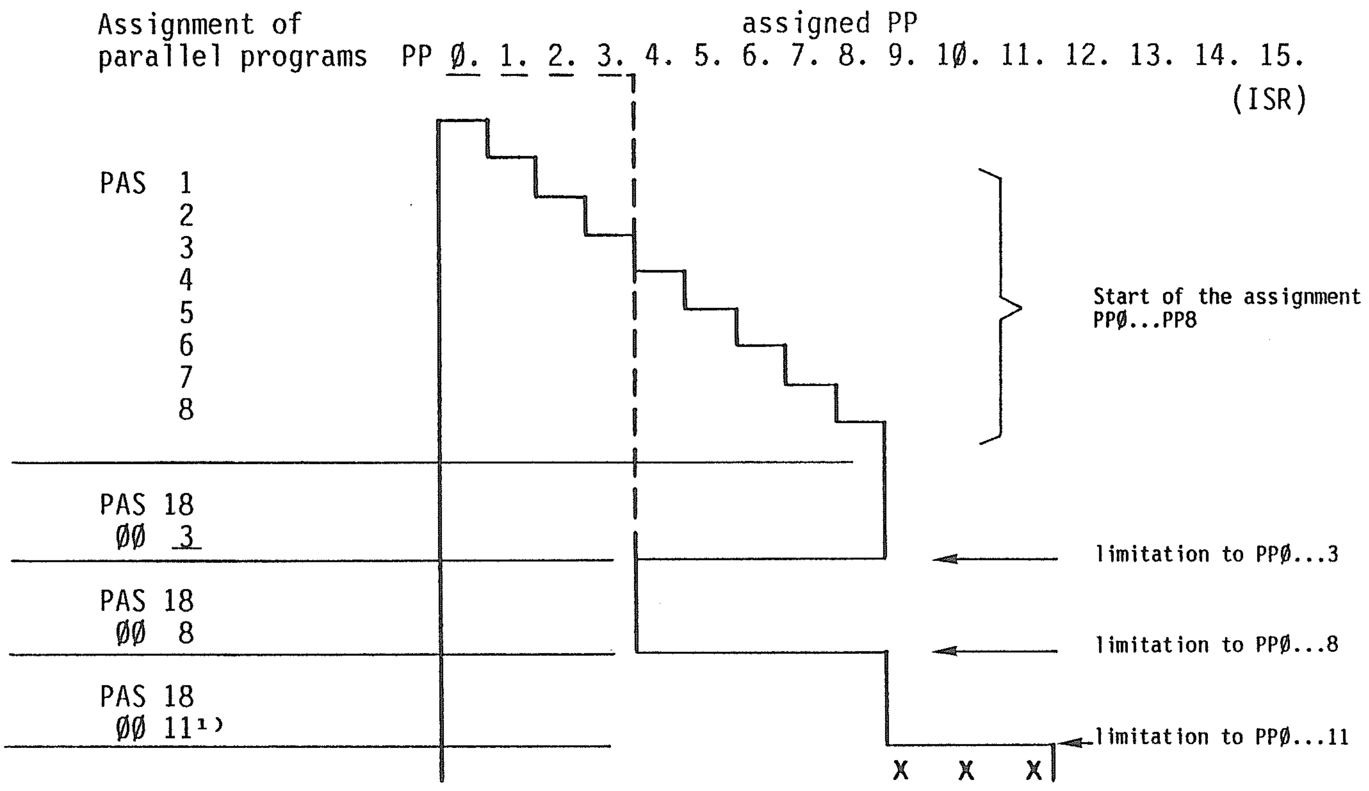
PAS 18 Limitation of the assigned parallel programs

All SAIA°PLC allow assignment of up to 16 parallel programs (PP) and running them in parallel. Up to now it has been necessary to reassign a PP to a dummy program loop if it was no longer required. However, no time could thus be saved during the execution of the remaining PPs.

However, it is now possible to limit a maximum number of active PPs with the PAS 18 instruction. After the PP assignment with PAS Ø...max. 15 a limited number of PPs can be determined from a higher to a lower value in the user program at any place and as many times as desired.

Instruction code		Operand		
Mnemo code	Numerical code	Description	Range	
PAS	29	---	18	1st line
---	ØØ	Upper limit of the PP	PP1...15	2nd line

The PAS 18 instruction is always executed (irrespective of the ACCU status) and does not alter the ACCU status.



1) If a higher number than the PPs originally assigned is entered with PAS 18, no malfunction is caused. The inactive PPs 9...11, however, will require processing time in the system program.

PAS 3Ø PAS 31...38*

"Check sum" of the system and user program

The "Check sum" function serves to establish the sum to check of the memory contents of the system program (PAS 3Ø) or user program (PAS 31...38). Thus, it can be ensured that the contents of the memories have not been changed.

After execution of the instruction:
ACCU = 1 if the reference value complies with the sum to check,
ACCU = Ø if the reference value does not comply with the sum to check.

The instructions PAS 3Ø...38 are always executed irrespective of the ACCU. If a change in memory contents has occurred, the user can take the measures which seem necessary to him: triggering an alarm, resetting the watchdog etc.

Check sum of the system program

Mnemo code	Numerical code	Operand	
PAS	29	Always 3Ø	1st line
---	ØØ	Always Ø	2nd line

Check sum of the user program

Instruction code		Operand		
Mnemo code	Numerical code	Description	Range	
PAS	29	Program part 1.K...8.K*	31...38*	1st line
---	XX	Reference value	xxxx	2nd line

The appropriate reference value for the sum to check of the user program is obtained by executing the respective PAS instruction in the operating mode STEP. The PCA displays this reference value on the programming unit for a few seconds. In the operating mode PROG, the corresponding reference value can then be introduced in the 2nd line.

Attention: execution of these instructions takes quite a long time:
PAS 3Ø ≈ 28.Øms, PAS 31...38 ≈ 8.3ms
(PCA232: PAS 3Ø ≈ 13.6ms, PAS 31...38 ≈ 9.5ms)

Therefore, use "Check sum" only if the sequence to be controlled allows it, e.g. when switching on the PLC, at the end of an operation cycle, etc.

*) The check-sum of the user program is determined individually for every "K" program (see example on the following page).

It is recommended not to introduce this instruction into the user program until it has been completely developed and tested. Each program alteration, irrespective of whether the program was extended or reduced, causes alteration of the check sum and requires modification of the reference value.

Example: A 2K-user program must be executed upon switching on.

```
   2Ø    PAS     31  } "Check sum"
   21     Ø9    825  } 1. K
 ←-22    JIZ     35
   23    PAS     32  } "Check sum"
   24     Ø7   154Ø  } 2. K
 ←-25    JIZ     35


   3Ø  ┌→COO    255  } Main program with
   31  └─JMP     3Ø  } WD-monitoring

 →-35    SEO     15  } Set alarm output
 └-36    JMP     35  } outside the main
                       program
```

Procedure:
- After program input and test, select operating mode STEP
- Key in $\boxed{A}$ 23 $\boxed{+}$
  -> the reference value for the 2nd K (PAS 32) appears for approx. 2Øs
- Input of the reference value in operating mode PROG:
  $\boxed{A}$ 24 ref. value $\boxed{+}$
- Same procedure for PAS 31

E 8 Display instructions

DOP Display of an operand

DOP: Display Operand ---> display of the number in the operand

Instruction format:

Instruction code		Operand	
Mnemo code	Numerical code	Description	Range
DOP	3Ø	Whatever number to be displayed	Ø...2Ø47

DOP is an auxiliary instruction mainly used for commissioning and trouble-shooting. A program can be written in such a way that when a particular process situation or a fault occurs, an identification number is displayed. This display is provided in "RUN" mode and is in the operand display of the programming unit or an operand display.
The display is maintained for 1s. If it is required for the display to be maintained for a longer period, the instruction DOP is to be processed at least once per second (mainly in a circulating program).
DOP is only performed when a linkage was not successful (ACCU = Ø).

Part F
Example 14

DTC Display of a timer or counter reading

DTC: Display Timer or Counter ---> display of timer or counter value

Instruction format:

Instruction code		Operand	
Mnemo code	Numerical code	Description	Range
DTC	31	Timer or counter address	256 ... 319 * (i)

(i) = indexable

DTC is likewise a valuable auxiliary instruction for commissioning and trouble-shooting.
By means of DTC the timing of a timer or reading of a counter can be displayed on the programming unit or the operand display of the PCA in RUN mode (maximum display 9999).
The display is maintained for 1s. If it is required for the display to be maintained for a longer period, the instruction DTC must be processed at least once per second (mainly in a circulating program). DTC is only performed when ACCU = 1.

*) Register structure see page III.

Part F
Examples: 8, 18

PART F PROGRAMMING EXAMPLES

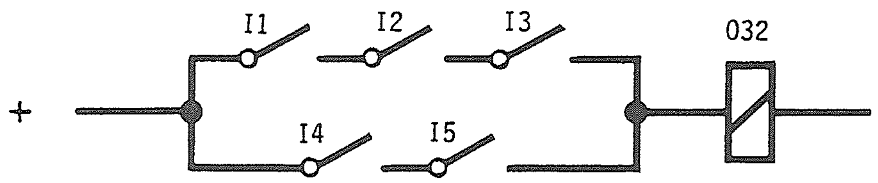
PART F PROGRAMMING EXAMPLES

After defining the instruction set 1H in part E, the following collection of examples shows their interrelation. These are mainly typical examples which explain functions and use of the individual commands. The last example being an overall program of a practical application.
I/O adressing has been selected so that all examples can be reconstructed on a PCA1 or PCA2 with the aid of a PCA2.S1Ø input simulation unit. Accordingly, the address range is as follows:

I = Ø...31
O = 32...63

Example 1: AND/OR linkage with ladder diagram

1) Parallel branchings



Step	Instruction		Acting on	
	Description	In mnemo code (numerical)	element or step	resp. OPERAND
1Ø	Start with interrogate	STH (Ø1)	I1	1
11	AND-linkage	ANH (Ø3)	I2	2
12	AND-linkage	ANH (Ø3)	I3	3
13	OR-linkage	ORH (Ø5)	I4	4
14	AND-linkage	ANH (Ø3)	I5	5
15	Transfer result	OUT (1Ø)	032	32
16	Return to start	JMP (2Ø)	STEP 1Ø	1Ø

STEP | WHAT | WHERE

1	2	3	4

STEP

1	2

CODE

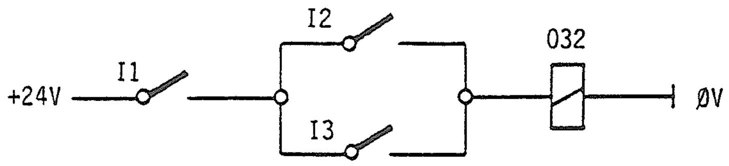
1	2	3	4

OPERAND

2) OR has priority over AND

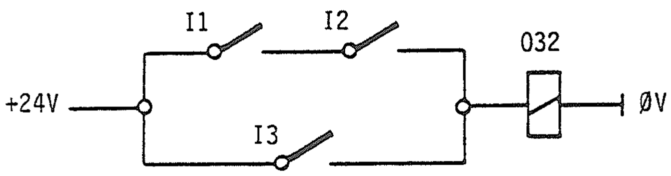
For combinations of OR and AND functions it is necessary to refer to a special feature explained by means of this example.

The following contact configuration must be programmed:



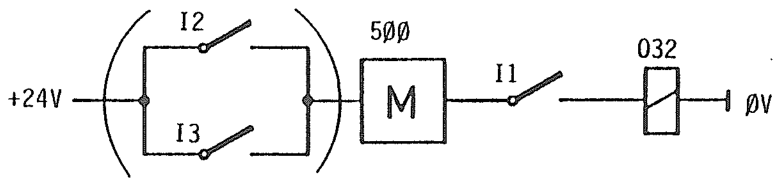
~~STH 1~~
~~ANH 2~~
~~ORH 3~~
~~OUT 32~~

The circuit cannot be programmed in this way because in accordance with the definition, the OR function of a parallel circuit corresponds to the following linkage branch. Otherwise it would correspond to the following schematic diagram:



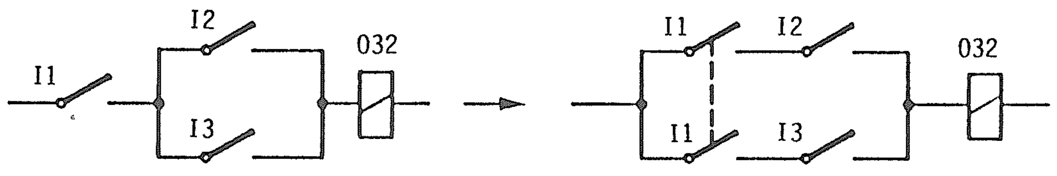
There are two ways of programming the circuit:

- By the intermediate storage of the OR result



```
ADDR NC   MNC   OPRD
  40 01   STH      2   INTERROGATE I2 IF H
  41 05   ORH      3   PARALLEL SWITCHING OF I3; INTERROGATE IF H
  42 10   OUT    500   STORE INTERMEDIATE RESULT IN A FLAG
-------------------
  43 01   STH    500   NEW LINKAGE LINE WITH INTERROGATION OF FLAG
  44 03   ANH      1   AND - LINKAGE WITH I1; INTERROGATED IF H
  45 10   OUT     32   ACCEPTANCE OF RESULT AT 032
  46 20   JMP    40->
```

- By re-drawing the circuit as a directly programmable parallel circuit with continuous parallel branches.



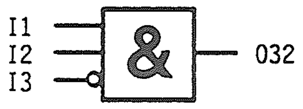
The second I1 contact is only for programming purposes. In reality I1 only exists once, in the program however, it is interrogated twice.

```
ADDR NC   MNC   OPRD
  50 01   STH      1   START OF 1ST LINKAGE BRANCH
  51 03   ANH      2
  52 05   ORH      1   PARALLEL SWITCHING AND START OF 2ND LINKAGE BRANCH
  53 03   ANH      3
  54 10   OUT     32
  55 20   JMP     50->
```

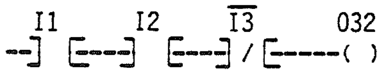
Example 2: AND/OR-linkage with logic diagram

1) AND function

Logic diagram:



Ladder diagram:



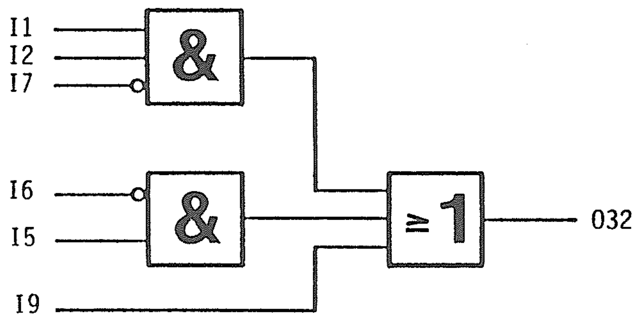
The logic diagram facilitates clear representation of the logical linkages and signal states.

Representation of the following functions in the ladder diagram is problematic. For this reason use is frequently made in practice of the symbolic contacts. Here, the linkage is performed (032 activated) when I1 and I2 = H and I3 = L.

```
ADDR  NC   MNC   OPRD
  60  01   STH      1   INTERROGATE I1 IF H
  61  03   ANH      2   AND - LINKAGE SUCCESSFUL (ACCU = 1) WHEN I2 = H
  62  04   ANL      3   AND - LINKAGE SUCCESSFUL (ACCU = 1) WHEN I3 = L
  63  10   OUT     32
  64  20   JMP     60->
```

When programming with a logic diagram the non-inverted elements are interrogated and linked with STH, ANH, ORH, the inverted elements with STL, ANL, ORL.

2) AND/OR combined

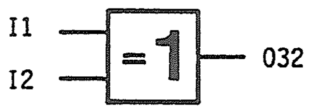


```
ADDR  NC   MNC   OPRD
  70  01   STH      1
  71  03   ANH      2
  72  04   ANL      7
  73  06   ORL      6   NEW, PARALLEL LINKAGE BRANCH
  74  03   ANH      5
  75  05   ORH      9   NEW, PARALLEL LINKAGE BRANCH
  76  10   OUT     32
  77  20   JMP     70->
```

Example 3: EXOR-linkages

1) Comparison for inequality of logical states

Two inputs must be compared. Where both have the same logical state the output must be = L. If they are unequal, the output must be = H (activated).

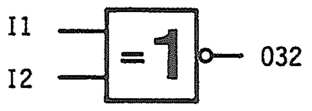


```
ADDR  NC   MNC    OPRD
  80  01   STH       1    INTERROGATE I1
  81  07   XOR       2    EXOR-LINKAGE WITH I2
  82  10   OUT      32
  83  20   JMP      80 ->
```

2) Comparison for equality of logical states

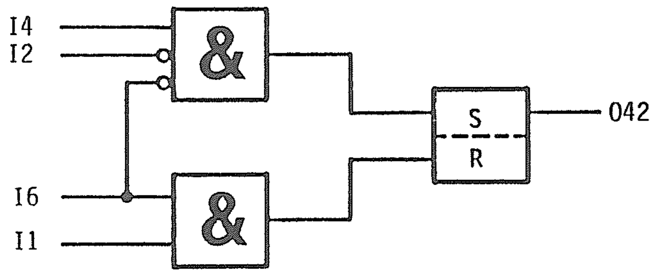
As in example 1), however,

where inputs are equal   ----> output = H (activated)
where inputs are unequal ----> output = L



```
ADDR  NC   MNC    OPRD
  85  01   STH       1
  86  07   XOR       2    EXOR-LINKAGE
  87  08   NEG       0    INVERSION OF ACCU CONTENT
  88  10   OUT      32
  89  20   JMP      85 ->
```

Example 4: Linkages with function chart



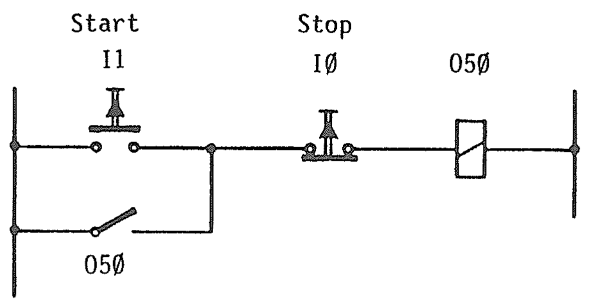
```
ADDR  NC   MNC    OPRD
  90  01   STH       4   INTERROGATE I4
  91  04   ANL       2   AND - LINKAGE, INTERROGATED IF L
  92  04   ANL       6   AND - LINKAGE, IF L
  93  11   SEO      42   IF LINKAGE SUCCESSFUL, SET OUTPUT 42
-----------------------
  94  01   STH       6   NEW LINKAGE, INTERROGATED I6
  95  03   ANH       1   AND - LINKAGE
  96  12   REO      42   IF LINKAGE SUCCESSFUL, RESET OUTPUT 42
  97  20   JMP      90->
```

If both linkages were successful then with each program run, the output would be set at step address 93 and reset at address 96; such a procedure would result in the output 42 oscillating, however, this situation is prevented by an interlock via I6.

Example 5: Start/Stop circuit with self-holding

1) With ladder diagram

The following is a classic protective switching technique:

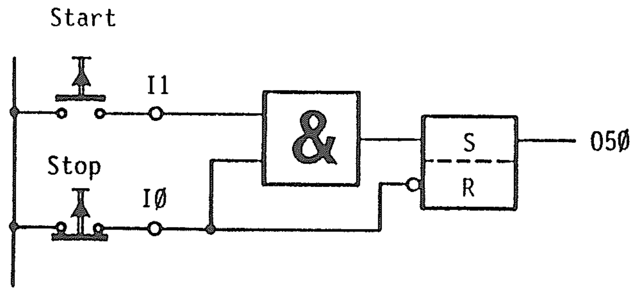


Program:

```
STH      1  }
ORH     5Ø  }  Start
OUT    4ØØ  }
-------------
STH    4ØØ  }
ANH      Ø  }  Stop
OUT     5Ø  }
```

Where the hardware permits (not C3Ø slide-in units or B9Ø interfaces), output 05Ø can be directly included in the linkage. As the program illustrates, opening contact IØ links if "H" because 05Ø can only be activated if IØ is closed.
This programming method also safeguards against a wire breakage. If there is a break, for example in the lines from IØ, I1 or 05Ø, 05Ø is always switched off.

2) With function chart

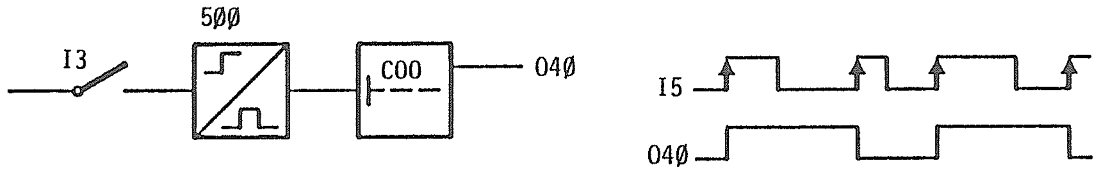


Program:

```
STH      1  }
ANH      Ø  }  Start
SEO     5Ø  }
-------------
STL      Ø  }  Stop
REO     5Ø  }
```

As with example 4, the set instruction is in this case also effective only when IØ = H. If both pushbuttons are pressed, the reset instruction has priority due to the AND-linkage. In the form shown, this method also provides a safeguard against a wire break.

Example 6: Pulse divider (stepping switch)

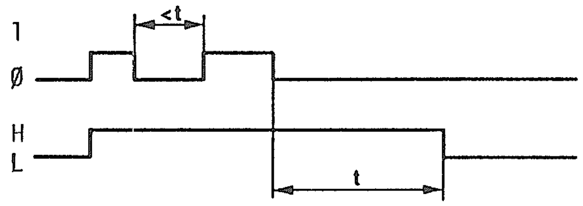


```
ADDR  NC   MNC   OPRD
 100  01   STH     3◄┐ INTERROGATE I 3
 101  09   DYN   500 │ SIGNAL-EDGE TRIGGERING; STORAGE IN FLAG 500
 102  13   COO    40 │ INTERROGATION OF O40 & INVERSION OF ITS STATUS
 103  20   JMP   100─┘
```

If the DYN instruction was omitted in this example, with switch I3 closed, output 4Ø would be complemented on every program run; in this small loop, approximately 3ØØØ times per second.

With DYN, output 4Ø would only be complemented on the first program run with I3 closed. Every subsequent run would have no further effect until the signal state of I3 has altered and another switching signal edge occurs (contact closes).

Example 7: Off delay

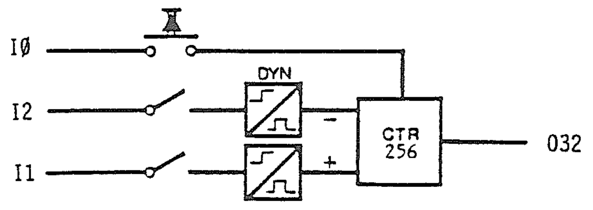


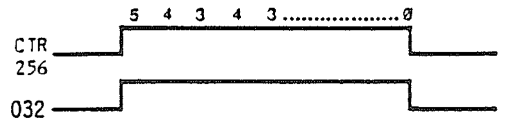
```
ADDR  NC   MNC   OPRD
 104  01   STH      7   INTERROGATE I7
 105  14   STR    256   SET TIMER                 ) 2 - LINE
 106  00    00     75   TIME INPUT IN 1/10TH S    ) INSTRUCTION
------------------------
 107  01   STH    256   INTERROGATE TIMER
 108  10   OUT     52   TRANSFER TO 052
 109  20   JMP    104 ->
```

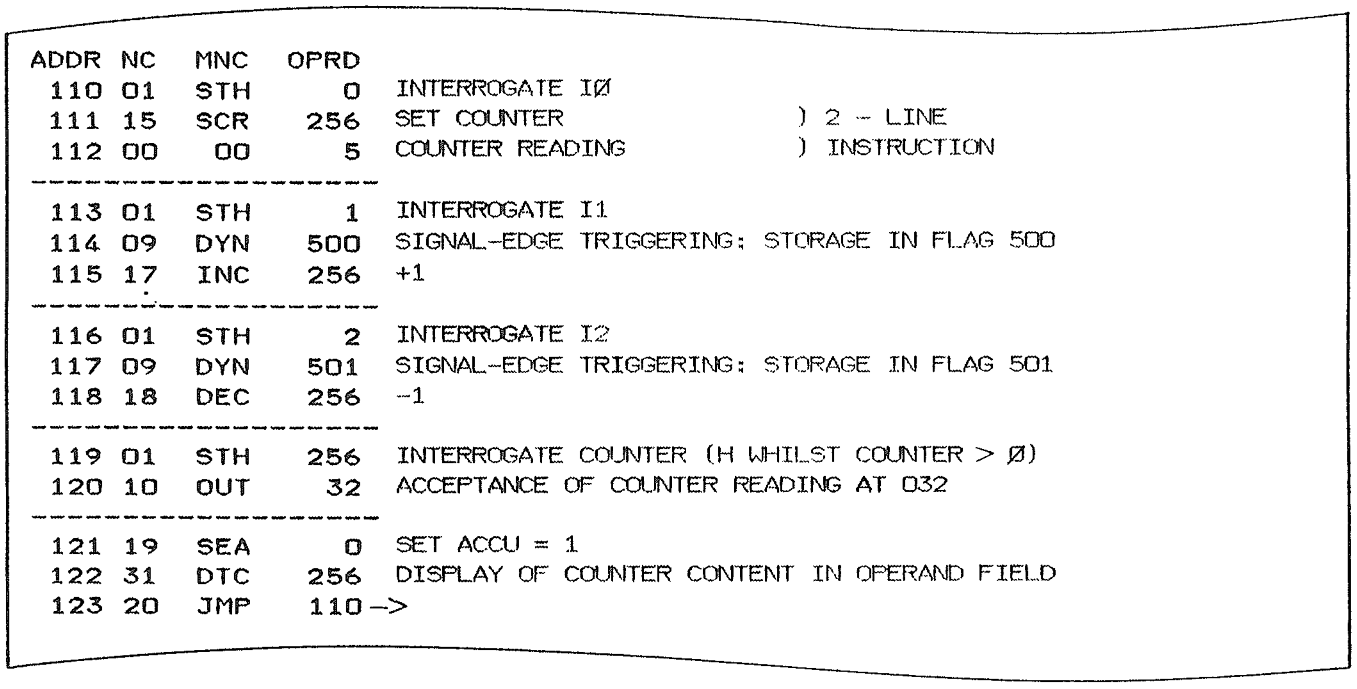
When I7 is closed the timer is set and its logical state becomes H. Timing does not commence until I7 is opened. (In practice timing starts immediately. However with I7 closed the timer is set again in the next program run after a few 1ØØμs and timing commences from the start until the signal is removed from I7, i.e. switch I7 is opened.)

If I7 is closed again during the timing, the timer is reset and restarted.

Example 8: Increment/decrement counter
(with display of counter reading on a programming unit or display module)





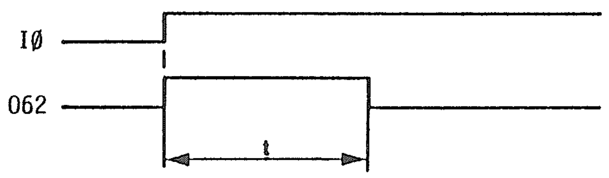


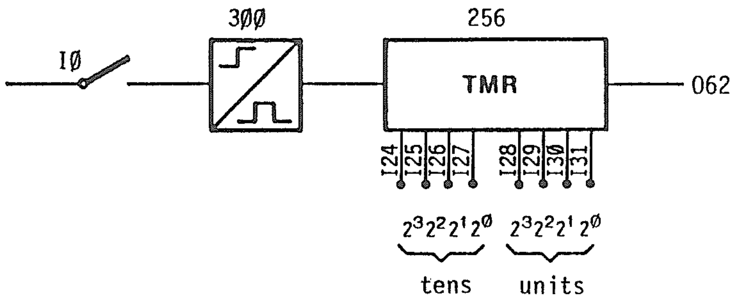
```
ADDR NC   MNC   OPRD
 110 01   STH      0   INTERROGATE IØ
 111 15   SCR    256   SET COUNTER                  ) 2 - LINE
 112 00    00      5   COUNTER READING              ) INSTRUCTION
---------------------
 113 01   STH      1   INTERROGATE I1
 114 09   DYN    500   SIGNAL-EDGE TRIGGERING: STORAGE IN FLAG 500
 115 17   INC    256   +1
---------------------
 116 01   STH      2   INTERROGATE I2
 117 09   DYN    501   SIGNAL-EDGE TRIGGERING: STORAGE IN FLAG 501
 118 18   DEC    256   -1
---------------------
 119 01   STH    256   INTERROGATE COUNTER (H WHILST COUNTER > Ø)
 120 10   OUT     32   ACCEPTANCE OF COUNTER READING AT 032
---------------------
 121 19   SEA      0   SET ACCU = 1
 122 31   DTC    256   DISPLAY OF COUNTER CONTENT IN OPERAND FIELD
 123 20   JMP    110 ->
```

For DTC the ACCU must be = 1 otherwise the counter reading display will be omitted.
Either SEA will be set before DTC (as in example) or the DTC instruction be taken at the start of the circulation loop because ACCU is always 1 after performance of the JMP instruction.

Example 9: Timer fleeting on delay with external timing entry in BCD code

Time range 1...99s adjustable externally via two BCD switches. Inputs I24...31 from BCD switch.





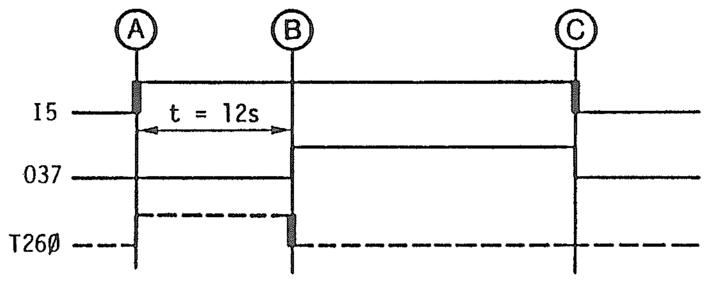
BCD table

Binary signals at 4 inputs (BCD switch)				
I28	I29	I3Ø	I31	Decimal value
$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	
L	L	L	L	Ø
L	L	L	H	1
L	L	H	L	2
L	L	H	H	3
L	H	L	L	4
L	H	L	H	5
L	H	H	L	6
L	H	H	H	7
H	L	L	L	8
H	L	L	H	9

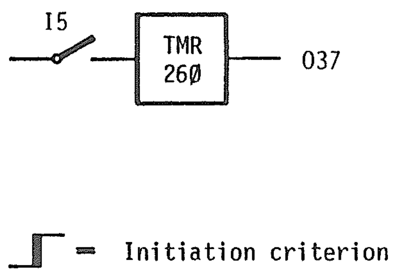
```
ADDR NC   MNC   OPRD
 130 01   STH      0   INTERROGATE IØ
 131 09   DYN    300   SIGNAL-EDGE TRIGGERING; TIMER ONLY SET IN 1ST CYCLE
 132 14   STR    256   SET TIMER
 133 17    17     31   EXTERNAL VALUE * 10 * 1/10THS S; ADDRESS N AT I31
------------------
 134 01   STH    256   INTERROGATE TIMER
 135 10   OUT     62   TRANSFER TO 062
 136 20   JMP    130->
```

Example 1Ø: Switch-on delay with flowchart

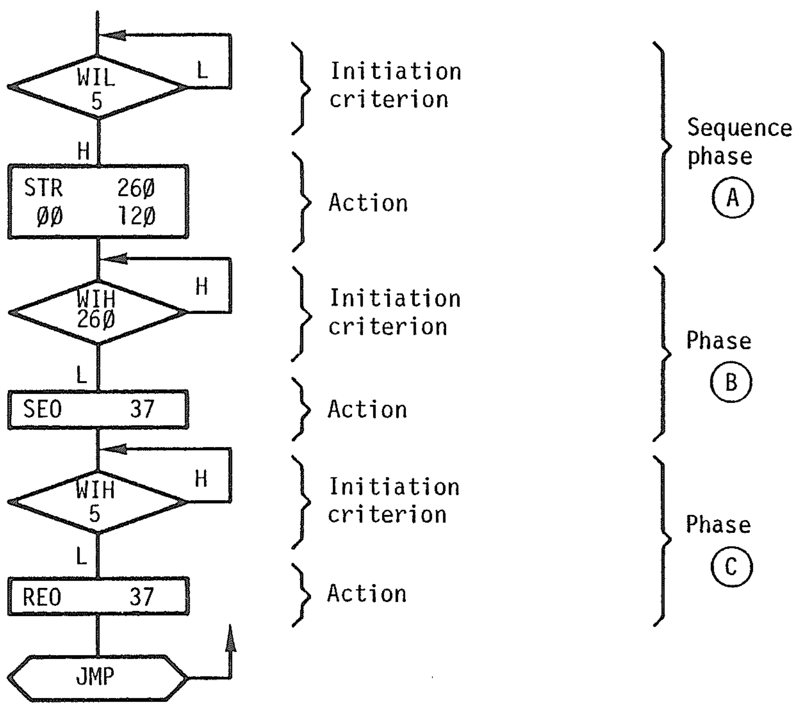
Flowchart:



Ladder diagram:



The sequential time-switch program can be subdivided into different phases. A particular initiation criterion is assigned to each phase. The fulfilment of the appropriate conditions is awaited in a wait loop in order to perform the subsequent function.

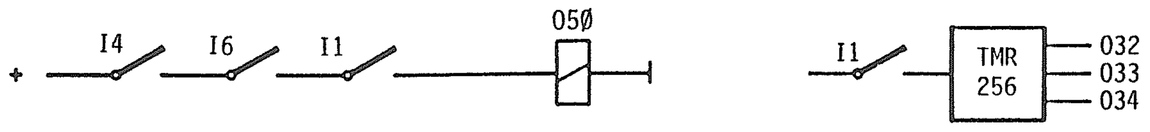


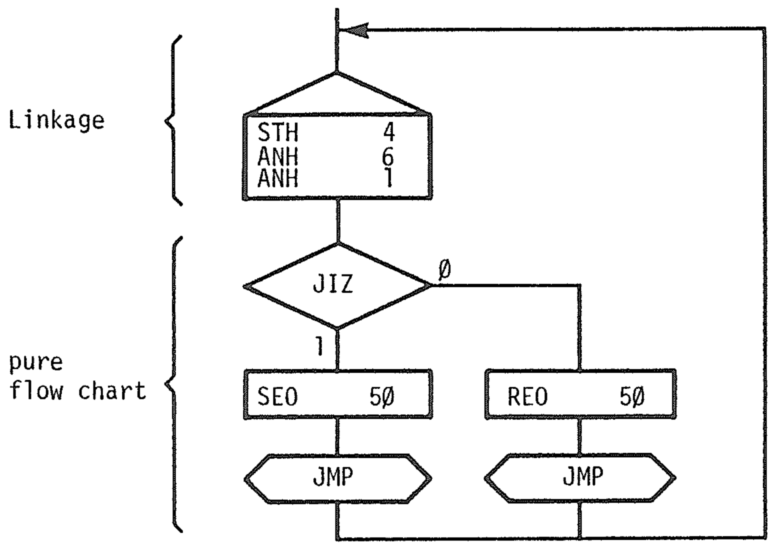
```
ADDR NC  MNC  OPRD
┌►140 26  WIL     5  INTITIATION CRITERION    ) SEQUENCE PHASE
│ 141 14  STR   260  ACTION                   ) A
│ 142 00   00   120
│ 143 25  WIH   260  INITIATION CRITERION     ) PHASE
│ 144 11  SEO    37  ACTION                   ) B
│ 145 25  WIH     5  INITIATION CRITERION     ) PHASE
│ 146 12  REO    37  ACTION                   ) C
└─147 20  JMP   140->
```

Example 11: AND-linkage with flowchart

Combination of linkage and flowchart (without wait loops)

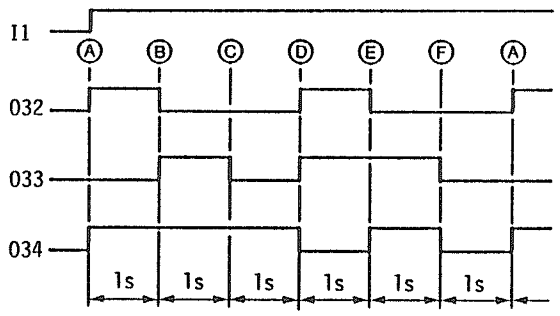
Models in ladder and logic diagram:

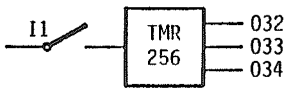


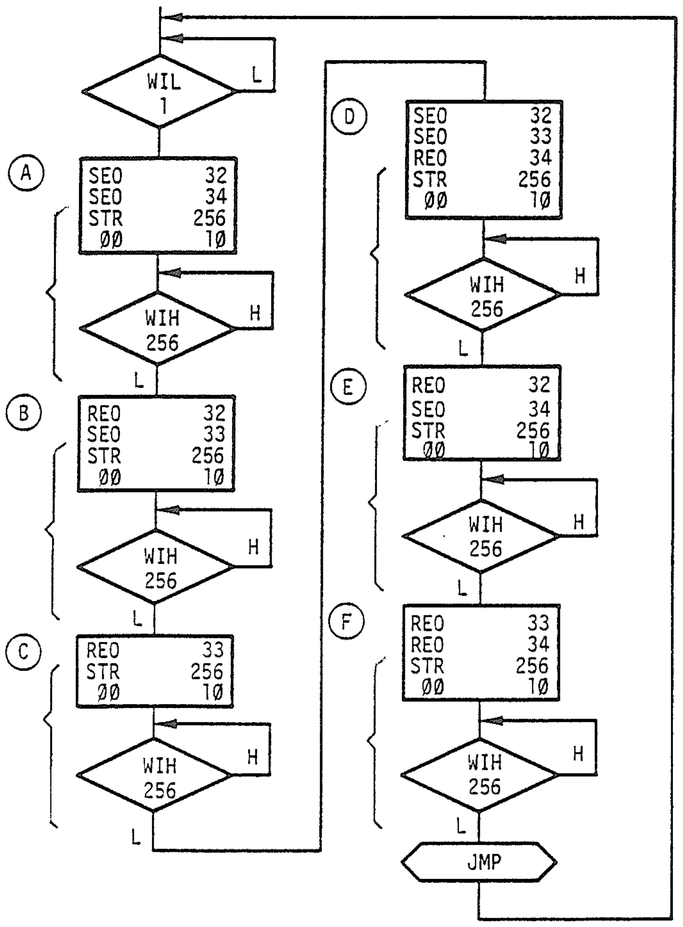


```
 ADDR NC  MNC   OPRD
┌─►150 01  STH      4   INTERROGATE I4
│  151 03  ANH      6   AND-LINKAGE
│  152 03  ANH      1   AND-LINKAGE
│┌─153 22  JIZ    156->  IF.Ø, JUMP TO STEP ADDRESS 156
││ 154 11  SEO     50   IF 1, SET OUTPUT 50
◄┼─155 20  JMP    150->  RETURN TO PROGRAM START
│└►156 12  REO     50   RESET OUTPUT 50
└──157 20  JMP    150->  RETURN TO PROGRAM START
```

Example 12: Sequence control with and without subroutine

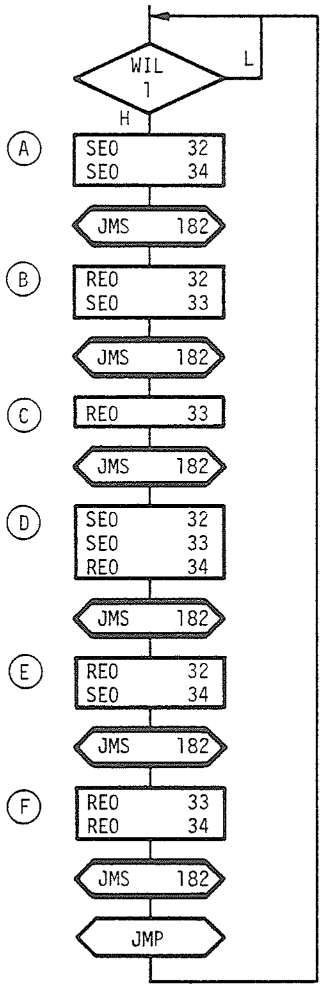






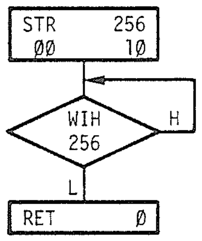
The flowchart illustrates the program sequence without subroutine. The parts of the program in brackets are repeated 6 times and it is advantageous to write these in the form of a subroutine.

Main program



```
******************** MAIN PROGRAM
ADDR NC  MNC  OPRD
 160 26  WIL     1  WAIT, IF I1 OPEN
 161 11  SEO    32
 162 11  SEO    34
 163 23  JMS   182=>  JUMP TO SUBROUTINE 182
 164 12  REO    32
 165 11  SEO    33
 166 23  JMS   182=>  JUMP TO SUBROUTINE 182
 167 12  REO    33
 168 23  JMS   182=>  JUMP TO SUBROUTINE 182
 169 11  SEO    32
 170 11  SEO    33
 171 12  REO    34
 172 23  JMS   182=>  JUMP TO SUBROUTINE 182
 173 12  REO    32
 174 11  SEO    34
 175 23  JMS   182=>  JUMP TO SUBROUTINE 182
 176 12  REO    33
 177 12  REO    34
 178 23  JMS   182=>  JUMP TO SUBROUTINE 182
 179 20  JMP   160->
```

Subroutine "182"



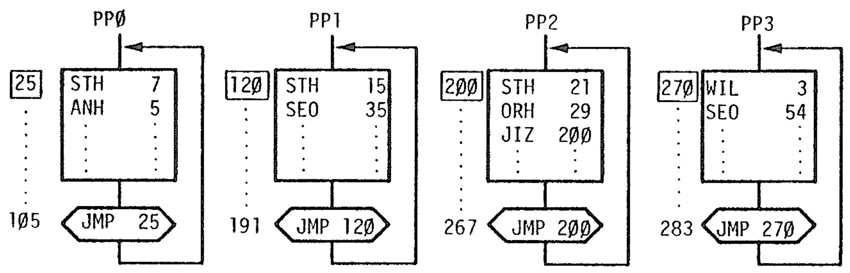
```
=================== SUBROUTINE 182 (WAIT 1S)
 182 14  STR   256
 183 00   00    10
 184 25  WIH   256
 185 24  RET     0->
```

Example 13: Operation with parallel programs

The program for controlling a machine with 2 manipulators and an indexing table is broken down into 4 parallel programs.

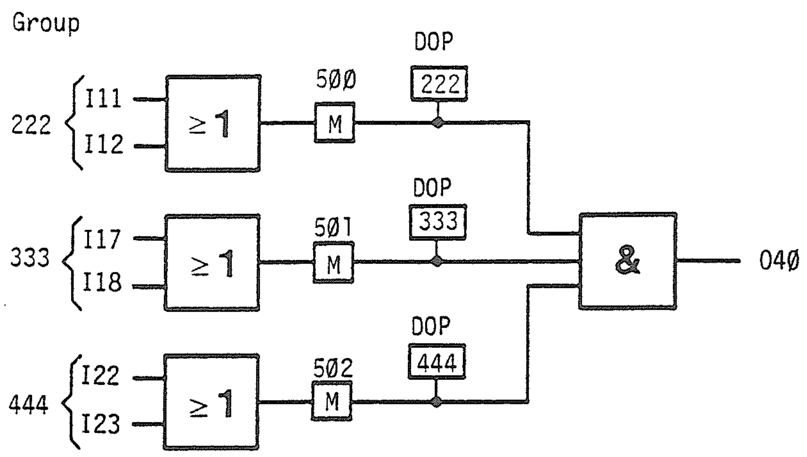
Parallel program Ø	Monitoring	Starting step address 25
Parallel program 1	Manipulator 1	Starting step address 12Ø
Parallel program 2	Manipulator 2	Starting step address 2ØØ
Parallel program 3	Indexing table	Starting step address 27Ø

Parallel program



```
******************  ASSIGNING THE PARALLEL PROGRAMS
ADDR NC  MNC  OPRD
   0 29  PAS     1  ASSIGNMENT PP1
   1 00   00   120
   2 29  PAS     2  ASSIGNMENT PP2
   3 00   00   200
   4 29  PAS     3  ASSIGNMENT PP3
   5 00   00   270
   6 20  JMP    25->  PPØ IS DIRECTLY ACTIVATED

++++++++++++++++++  PARALLEL PROGRAM Ø (MONITORING)
┌► 25 01  STH     7
│   .  .    .     .
│   .  .    .     .
│   .  .    .     .
└ 105 20  JMP    25->

++++++++++++++++++  PARALLEL PROGRAM 1 (MANIPULATOR 1)
┌►120 01  STH    15
│   .  .    .     .
│   .  .    .     .
│   .  .    .     .
└ 191 20  JMP   120->

++++++++++++++++++  PARALLEL PROGRAM 2 (MANIPULATOR 2)
┌►200 01  STH    21
│   .  .    .     .
│   .  .    .     .
│   .  .    .     .
└ 267 20  JMP   200->

++++++++++++++++++  PARALLEL PROGRAM 3 (INDEXING TABLE)
┌►270 01  STH     3
│   .  .    .     .
│   .  .    .     .
│   .  .    .     .
└ 283 20  JMP   270->
```

<u>Example 14: Monitoring circuit with fault display on programming unit</u>



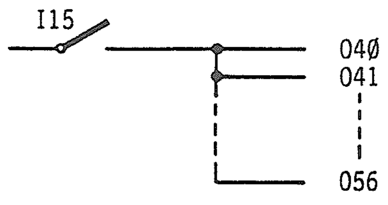
The inputs of groups 222, 333, 444 are monitored. Where a condition is not satisfied the lamp at output 4Ø goes out. The faulty group is then displayed on the programming unit.

```
ADDR. NC   MNC   OPRD
 300  01   STH     11
 301  05   ORH     12
 302  10   OUT    500
 303  30   DOP    222  DISPLAY WHEN LINKAGE NOT COMPLETED (ACCU = 0)
-------------------
 304  01   STH     17
 305  05   ORH     18
 306  10   OUT    501
 307  30   DOP    333  DISPLAY WHEN LINKAGE NOT COMPLETED
-------------------
 308  01   STH     22
 309  05   ORH     23
 310  10   OUT    502
 311  30   DOP    444  DISPLAY WHEN LINKAGE NOT COMPLETED
-------------------
 312  01   STH    500
 313  03   ANH    501
 314  03   ANH    502
 315  10   OUT     40
 316  20   JMP    300->
```

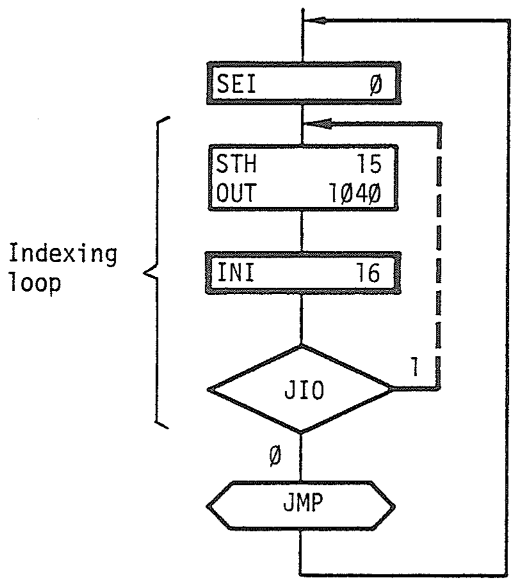
Example 15: Indexing

1st example with address indexing

Outputs 4Ø...56 are switched on or off via input 15.



Number of indexing steps = 56 - 4Ø = 16



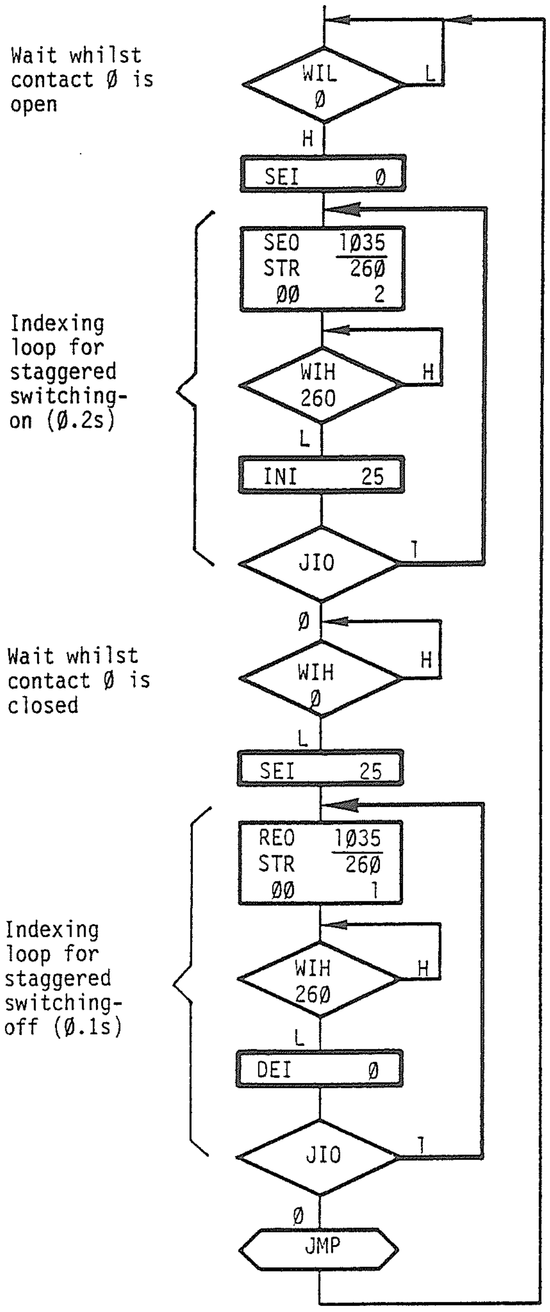
```
ADDR NC   MNC   OPRD
 350 16   SEI      0   SET INDEX REGISTER TO INITIAL VALUE Ø
┌►351 01   STH     15   INTERROGATE INPUT 15
¦ 352 10   OUT   1040   SETTING OF INDEXED OUTPUT (40 + 1000 = 1040)
¦ 353 27   INI     16   INCREMENT INDEX REGISTER UP TO FINAL VALUE 16
└─354 21   JIO    351->  REPEAT INDEXING UNTIL ALL OUTPUTS 04Ø..056 ARE
  355 20   JMP    350->                                         SWITCHED
```

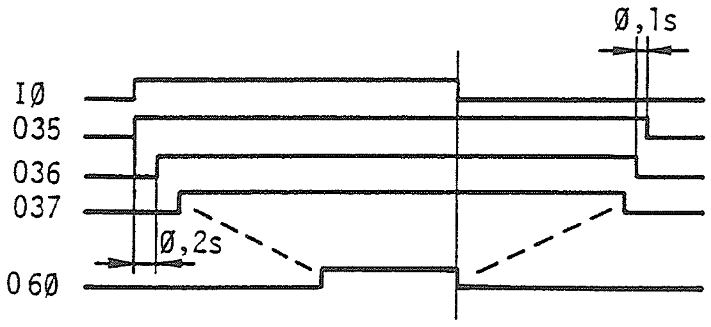
Example 16: Successive switching

2nd example with address indexing.

When input Ø is closed, outputs 35...6Ø are to switch on in succession at Ø.2s intervals.

When input Ø reopens, outputs 6Ø...35 are to switch off in succession at Ø.1s intervals.

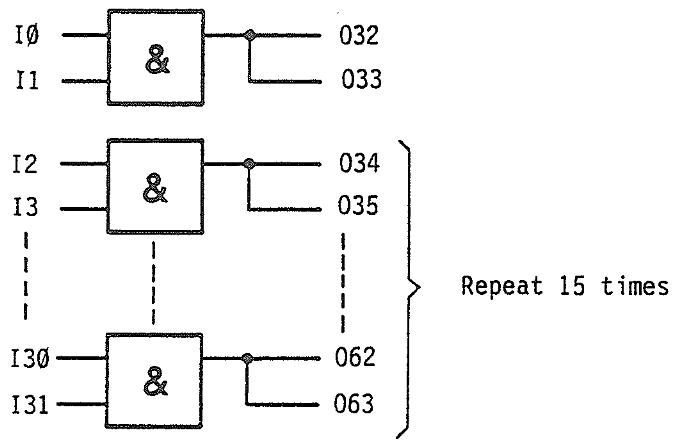

Wait whilst contact Ø is open
WIL Ø
L
H
SEI Ø
SEO 1Ø35
STR 26Ø
ØØ 2
Indexing loop for staggered switching-on (Ø.2s)
WIH 260
H
L
INI 25
JIO
1
Ø
Wait whilst contact Ø is closed
WIH Ø
H
L
SEI 25
REO 1Ø35
STR 26Ø
ØØ 1
Indexing loop for staggered switching-off (Ø.1s)
WIH 26Ø
H
L
DEI Ø
JIO
1
Ø
JMP



```
ADDR NC   MNC   OPRD
--------------------- SWITCHING-ON PHASE
 360 26   WIL      0
 361 16   SEI      0
┌►362 11  SEO   1035  35 + 1000 = 1035
│ 363 14  STR    260
│ 364 00   00      2
│ 365 25  WIH    260
│ 366 27  INI     25  60 -   35 =   25
└─367 21  JIO   362->
--------------------- SWITCHING-OFF PHASE
 368 25   WIH      0
 369 16   SEI     25  60 -   35 =   25
┌►370 12  REO   1035  35 + 1000 = 1035
│ 371 14  STR    260
│ 372 00   00      1
│ 373 25  WIH    260
│ 374 28  DEI      0
└─375 21  JIO   370->
 376 20   JMP   360->
```

Example 17: Small monitoring circuit

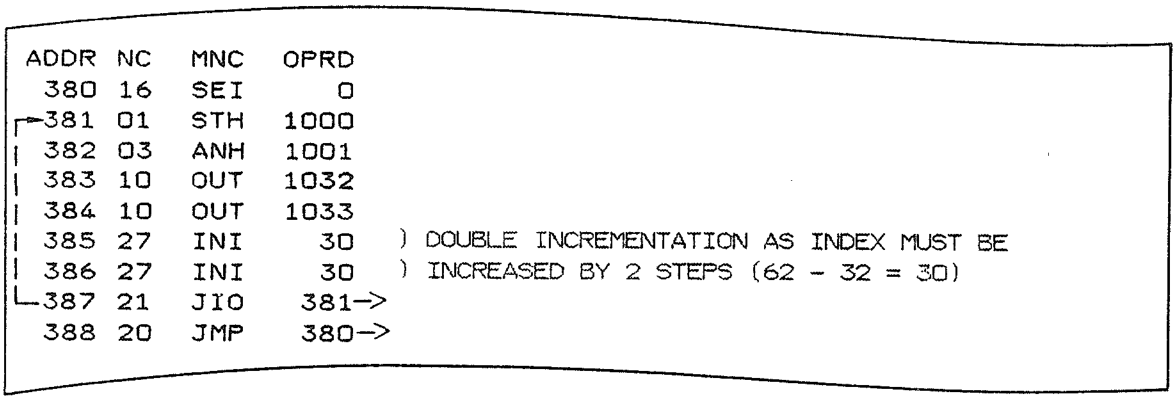
3rd example of address indexing.



Basic program

```
STH      0
ANH      1
OUT     32
OUT     33
```

The whole basic program is indexed in this example.



```
ADDR  NC   MNC   OPRD
 380  16   SEI      0
┌→381  01   STH   1000
│ 382  03   ANH   1001
│ 383  10   OUT   1032
│ 384  10   OUT   1033
│ 385  27   INI     30   ) DOUBLE INCREMENTATION AS INDEX MUST BE
│ 386  27   INI     30   ) INCREASED BY 2 STEPS (62 - 32 = 30)
└─387  21   JIO    381->
  388  20   JMP    380->
```

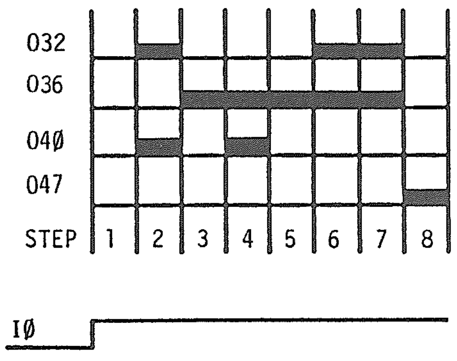
Example 18: Circulating program switch

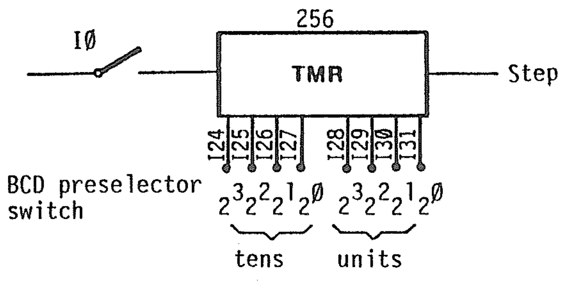
- with variable speed
- with manual/automatic operation
- with step display

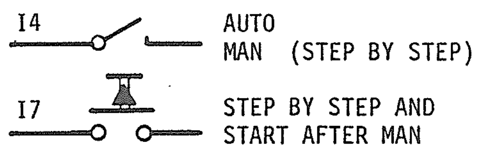
Task: With IØ closed, program switch must run in accordance with diagram. The step time is set on BCD switches I24...31 from Ø.1...9.9s. If switch I4 is closed, step by step operation can be performed via pushbutton I7. Step numbers to be continuously displayed in the operand display.

Solution: Sequence program contained in addresses 4Ø2...425. Periodic interrogation for manual operation and waiting for step time are contained in subroutine 43Ø. Step counting is by means of counter 28Ø. Small parallel program 45Ø assigned in order to provide continuous display.

Diagram:







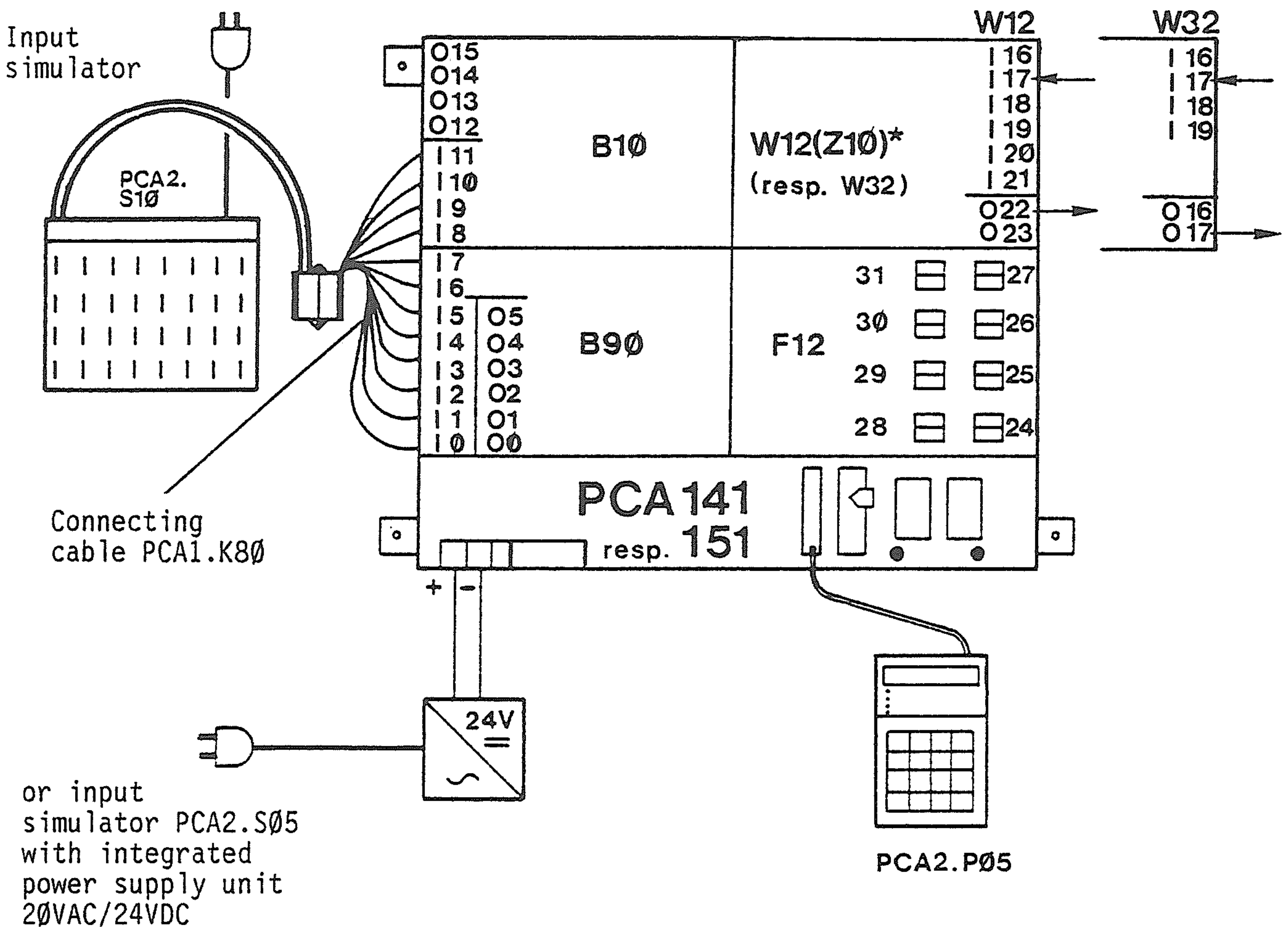
```
******************** ASSIGNMENT OF :
 400 29  PAS     1   PARALLEL PROGRAM 1
 401 00   00   450
******************** MAIN PROGRAM PPO
┌402 26  WIL     0   START PROGRAM RUN
│403 15  SCR   280   STEP COUNTER
│404 00   00     1
-------------------- STEP 1
│405 23  JMS   430=>  JUMP TO SUBROUTINE 430
-------------------- STEP 2
│406 11  SEO    32
│407 11  SEO    40
│408 23  JMS   430=>
-------------------- STEP 3
│409 12  REO    32
│410 12  REO    40
│411 11  SEO    36
│412 23  JMS   430=>
-------------------- STEP 4
│413 11  SEO    40
│414 23  JMS   430=>
-------------------- STEP 5
│415 12  REO    40
│416 23  JMS   430=>
-------------------- STEP 6
│417 11  SEO    32
│418 23  JMS   430=>
-------------------- STEP 7
│419 23  JMS   430=>
-------------------- STEP 8
│420 12  REO    32
│421 12  REO    36
│422 11  SEO    47
│423 23  JMS   430=>
--------------------
│424 12  REO    47
└425 20  JMP   402->  START

==================== SUBROUTINE 430
>430 02  STL     4   MAN / AUTO
┌431 22  JIZ   437->
-------------------- VARIABLE TIMER AUTO
│432 14  STR   256
│433 16   16    31
│434 25  WIH   256
│435 17  INC   280
│436 24  RET     0->
-------------------- MAN-OPERATION
└437 26  WIL     7
 438 25  WIH     7
 439 17  INC   280
 440 24  RET     0->

++++++++++++++++++++ PP1 : DISPLAY
┌450 31  DTC   280
└451 20  JMP   450->
```

Programming examples with analog modules

Addressing of the inputs and outputs was determined in such a way that all examples can be simulated making use of the same configuration of modules and units.

In all examples, programs are listed for the 8-bit module PCA1.W12 as well as for the 12-bit module PCA1.W32, which may be displayed alternately.

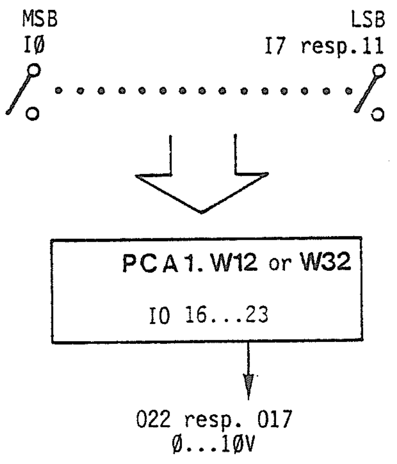
The configuration is the following:



*) Module PCA1.W12 Z1Ø allows input voltages from Ø...1ØV (instead of Ø...5V).

Example 19: Output of analog voltage from 8 to 12 inputs

The binary value formed from 8 or 12 inputs IØ...I7 or I11 must be output via analog output channel O22 or O17.



With module PCA1.W12 (8 bits or 7 bits)

```
4Ø    DTC    3Ø1   Display binary value
      SCR    3Ø1   Inputs IØ...7 to counter 3Ø1
       24      7
     ┌───────────┐
     │SCR    3Ø1 │ Analog output of content of
     │ 21     23 │ counter 3Ø1 via output channel
     │SEO     23 │ O22
     │ORH     22 │
     │REO     23 │
     └───────────┘
  <- JMP     4Ø
```

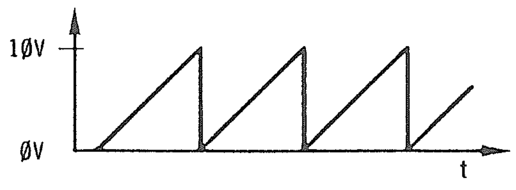
With module PCA1.W32 (12 bits)

```
5Ø    DTC    3Ø1   Display binary value of C3Ø1
      SCR    3Ø1   Transfer from inputs
       25     11   to C3Ø1
  1) JMS    7ØØ    --> SR "Output of
      JMP     5Ø   analog value"
```

1) Subroutines see manual "Hardware series PCA1" module PCA1.W32 or W2..

Example 2Ø: Output of ramp voltage

A ramp voltage must be generated by incrementing a binary value of 8 or 12 bits.



For W12, t is defined by timer 256.
For W32, t (for 4Ø96 runs) depends on the selected subroutine and the number of assigned parallel programs.

With module PCA1.W12 (8 or 7 bits)

```
6Ø  SEI      7  }
    COO   15ØØ  }
    STH   15ØØ  } Increment
    JIO     66  } binary counter
    DEI      Ø  }
    JIO     61  }
    SCR    3Ø1  } Load binary
     24    5Ø7  } value to C3Ø1
    SCR    3Ø1  }
     21     23  } Analog output of
    SEO     23  } C3Ø1 content to
    ORH     22  } output channel 022
    REO     23  }
    STR    256  }
     ØØ      4  } Wait Ø.Ø4s
    WIH    256  }
    JMP     6Ø
```

With module PCA1.W32 (12 bits)

```
8Ø  SEI     11  }
    COO   15ØØ  }
    STH   15ØØ  } Increment
    JIO     86  } binary counter
    DEI      Ø  }
    JIO     81  }
    SCR    3Ø1  } Load binary
     25    511  } value to C3Ø1
1)  JMS    7ØØ    --> SR "Output of analog value"
    JMP     8Ø    (JMS 6ØØ or 65Ø also possible)
```

1) Subroutines see manual "Hardware series PCA1" module PCA1.W3.. or W2..

Example 21: Enter BCD-value of module PCA1.F12 and output it as analog value via modules PCA1.W12 (8 bits) or W32 (12 bits)

The 2-digit BCD-value of 024 (of F12) is to be entered and output every 2s as analog voltage via channel 022 or 017 of the analog module. The corresponding binary value must be indicated on the operand display.

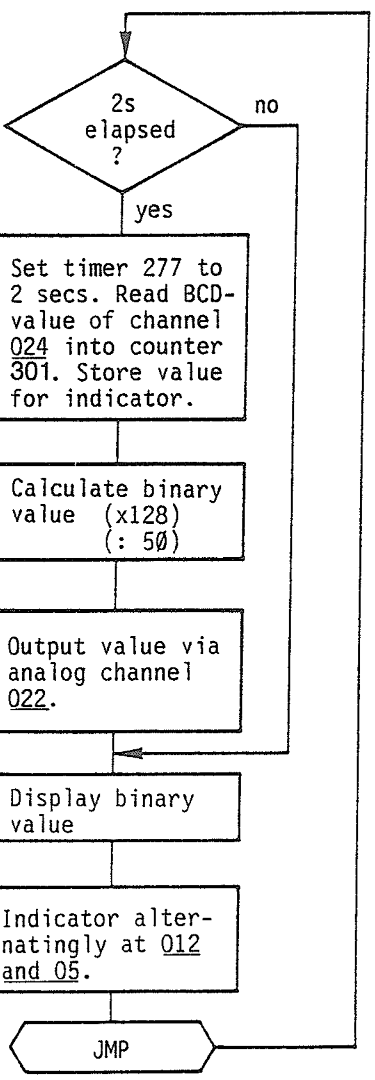
```
              10.ØV ≙ binary value of 128 (or 2Ø48)
| 5 | Ø | ≙    5.ØV ≙ binary value of 256 (or 4Ø96)
```

The outputs 05 and 012 should flash alternatingly according to the adjusted BCD-value (times 1/1Øs).

Program
for PCA1.W12 (8 or 7 bits)

```
┌─►2ØØ  Ø1   STH    277
│ ┌─2Ø1  21   JIO    227
│ │
│ │ 2Ø4  14   STR    277
│ │ 2Ø5  ØØ    ØØ     2Ø
│ │ 2Ø6  11  │SEO     24│
│ │ 2Ø7  15  │SCR    3Ø1│
│ │ 2Ø8  16  │ 16     31│
│ │ 2Ø9  12  │REO     24│
│ │ 21Ø  15   SCR    3Ø2
│ │ 211  31    31    3Ø1
│ │
│ │ 214  15   SCR    3Ø1   ┐
│ │ 215  29    29    128   │
│ │ 216  15   SCR    3Ø1   │
│ │ 217  3Ø    3Ø     5Ø   │
│ │                        ├ *
│ │ 22Ø  15  │SCR    3Ø1│  │
│ │ 221  21  │ 21     23│  │
│ │ 222  11  │SEO     23│  │
│ │ 223  Ø5  │ORH     22│  │
│ │ 224  12  │REO     23│  ┘
│ │
│ └►227  31   DTC    3Ø1
│
│   23Ø  Ø2   STL    278
│   231  14   STR    278
│   232  31    31    3Ø2
│   233  13   COO     12
│   234  Ø2   STL     12
│   235  1Ø   OUT      5
└── 236  2Ø   JMP    2ØØ
```

Flowchart



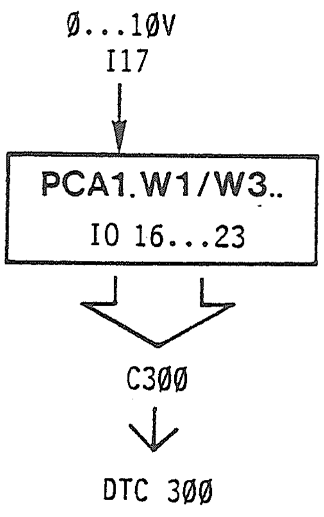
2s elapsed ?
no
yes
Set timer 277 to 2 secs. Read BCD-value of channel 024 into counter 301. Store value for indicator.
Calculate binary value (x128) (: 5Ø)
Output value via analog channel 022.
Display binary value
Indicator alternatingly at 012 and 05.
JMP

---

*) For PCA1.W32 (12 bits)

```
214   15  SCR    3Ø1    BCD-value x 41
215   29   29     41    (1ØV ≙ 1ØØ ≙ 4Ø96 or 41ØØ)
216   23 │JMS    7ØØ│ ──► Subroutine see manual Hardware PCA1 ,
                            chapter Hardwaremodule PCA1.W32 or W2
```

Example 22: Reading an analog voltage into a counter register and display of the binary value with DTC



With module PCA1.W1.. (8 bits)

```
90   OUT    17   Read analog value
     SCR   300   into counter C300
      24    23
     DTC   300   Display binary value
     JMP    90
```

With module PCA1.W3.. (12 bits)

```
100  JMS   500   Read analog value
                 into counter C300 *)
     DTC   300   Display binary value
     JMP   100
```

---

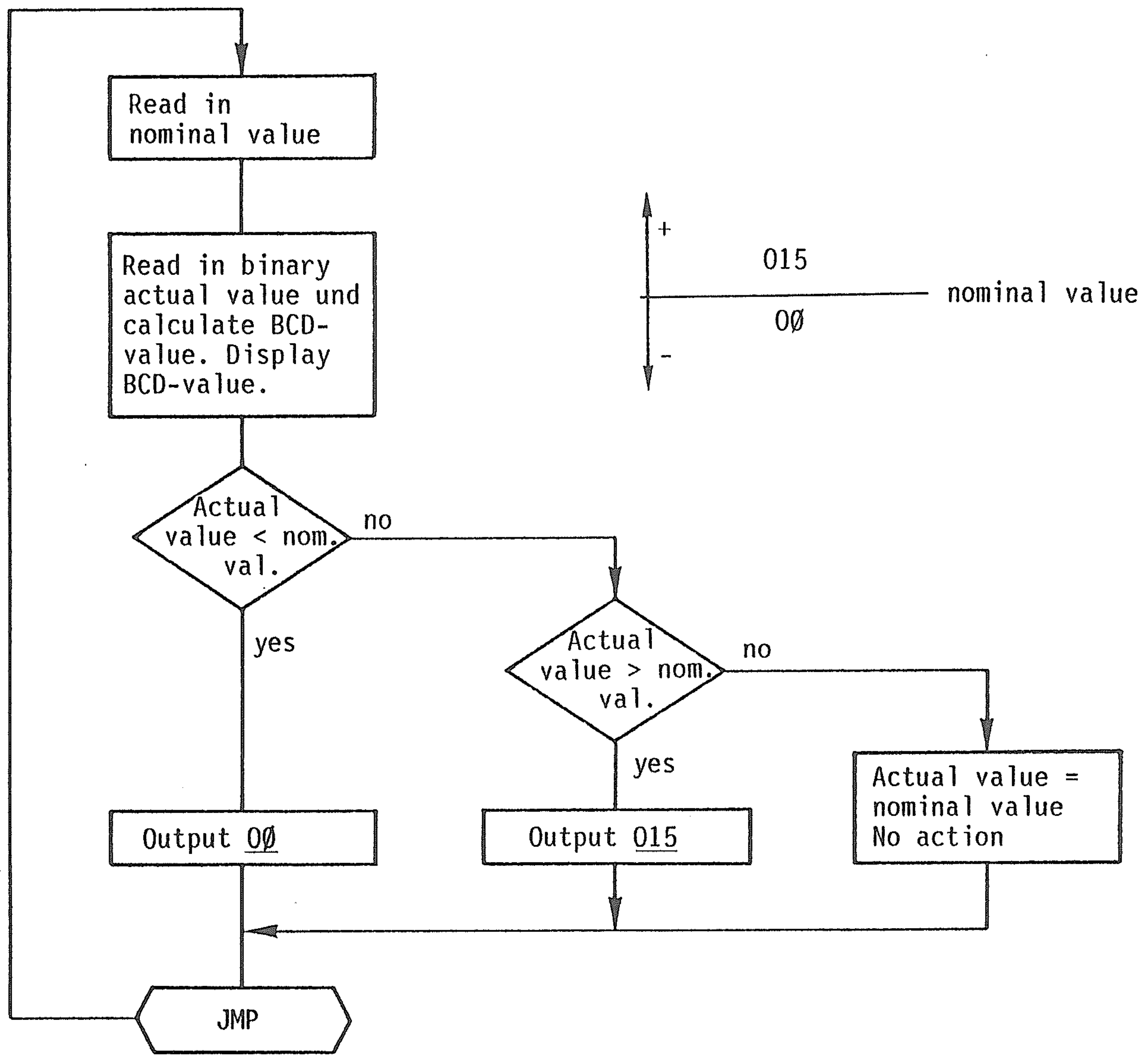
*) For subroutines refer to manual "Hardware series PCA1" (module PCA1.W32)

Example 23: On/off controller and floating controller (with analog module PCA1.W1.. or W3..)

Problem 23a: On/off controller

A nominal value must be predetermined via the two-digit BCD switch of channel 024. The actual value is entered via channel 17 of the analog module. If the actual value is below the selected nominal value, output 0Ø is activated, if it is above the nominal value output 015 is activated. The actual value is to be displayed as BCD-value (ØØ...99 ≙ Ø...9.9V) in the operand display.

Flowchart



Solution 23a:
for PCA1.W1.. (8 bits)

```
 4ØØ   11   SEO      24  }
 4Ø1   15   SCR     31Ø  }  Nominal value as BCD-value
 4Ø2   16    16      31  }  to C31Ø
 4Ø3   12   REO      24  }

 4Ø6   1Ø   OUT      17  }
 4Ø7   15   SCR     312  }  Actual value as binary value to C312
 4Ø8   24    24      23  }

*411   15   SCR     312  }
 412   29    29      5Ø  }
 413   15   SCR     312  }  Convert actual value into BCD:
 414   3Ø    3Ø     128  }  (x5Ø, :128) and display
 415   31   DTC     312  }

 418   15   SCR     31Ø  }  Comparison of actual value
 419   28    28     312  }  and nominal value
 42Ø   22   JIZ     43Ø  ;  Actual value > nominal value
 421   Ø1   STH     31Ø
 422   21   JIO     435  ;  Actual value < nominal value

 425   12   REO       Ø  }
 426   12   REO      15  }  Actual value = nominal value: no action
 427   2Ø   JMP     4ØØ  }

 43Ø   12   REO       Ø  }
 431   11   SEO      15  }  Actual value > nominal value: output 015
 432   2Ø   JMP     4ØØ  }

 435   12   REO      15  }
 436   11   SEO       Ø  }  Actual value < nominal value: output 0Ø
 437   2Ø   JMP     4ØØ  }
```

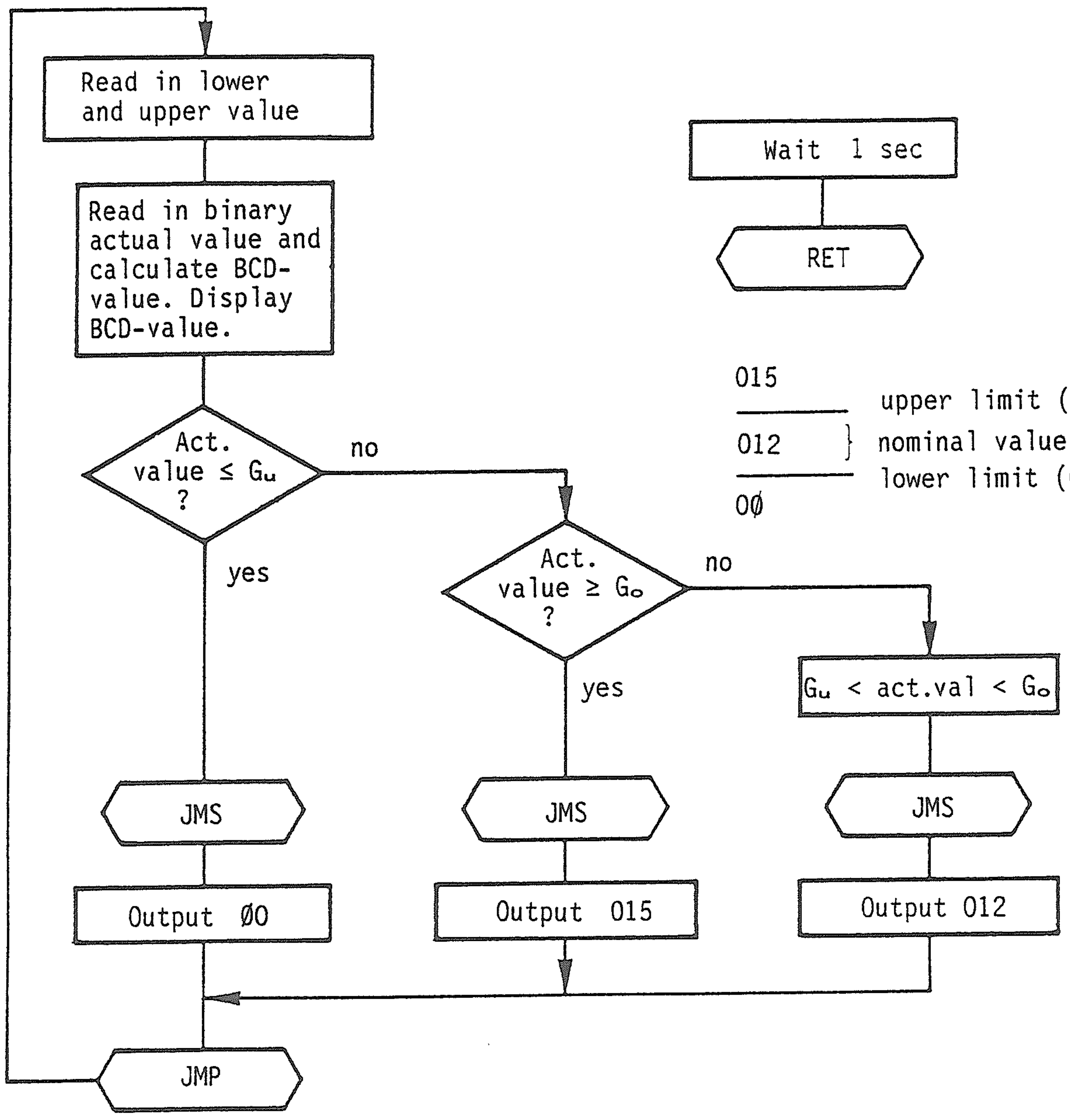
*) For module PCA1.W3..

```
 4Ø6   15   SCR     31Ø  }  Convert BCD-value into binary value
 4Ø7   29    29      41  }  BCD * 41 (1ØV ≙ 1ØØ ≙ 4Ø96 ≙ 41ØØ)

 411   23   JMS     5ØØ  ⇒  Transfer actual value as binary value to C3ØØ
 412   15   SCR     312
 413   31    31     3ØØ     Copy C3ØØ to 312
 414   31   DTC     312     Display C312
```

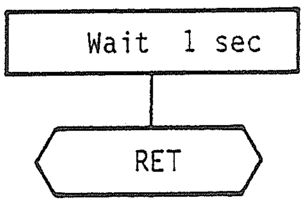
Additional problem 23b: Floating controller with time hysteresis

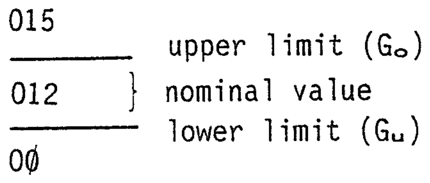
The upper and the lower limiting value of a floating controller must be entered via the 2-digit BCD-switch of 024 and 027. The actual value is entered via channel 17 of the analog module. If it reaches the lower limiting value output 0Ø will be activated. If it reaches the upper limiting value output 015 will be activated as soon as the period of 1s programmed as a hysteresis has elapsed. If the actual value does not exceed neither of both limiting values, output 012 will be active. The actual value is to be displayed as BCD-value in the operand display.

Flowchart



Subroutine for time hyseresis:





Solution 23b

For PCA1.W1.. (8 bits)

```
     3ØØ   11   SEO    24  \
     3Ø1   15   SCR   31Ø   >  Read lower limiting value as
     3Ø2   16    16    31   |  BCD-value into C31Ø
     3Ø3   12   REO    24  /

     3Ø5   11   SEO    27  \
     3Ø6   15   SCR   311   >  Read upper limiting value as
     3Ø7   16    16    31   |  BCD-value into C311
     3Ø8   12   REO    27  /

   / 31Ø   1Ø   OUT    17  \
   | 311   15   SCR   312   >  Read actual value as binary
   | 312   24    24    23  /   value into C312
** <
   | 314   15   SCR   312  \
   | 315   29    29    5Ø   |
   | 316   15   SCR   312   >  Convert actual value:
   | 317   3Ø    3Ø   128   |  (x5Ø; :128) and display
   \ 318   31   DTC   312  /

     321   15   SCR   31Ø  \
     322   28    28   312   >  Actual value ≤ limit value?
     323   22   JIZ   331  /
     324   23   JMS   35Ø  ==>    Time hysteresis
     325   12   REO    15
     326   12   REO    12
     327   111  SEO     Ø   :  Output OØ
     328   2Ø   JMP   3ØØ

     331   15   SCR   312  \
     332   28    28   311   >  Actual value ≥ maximum limit value
     333   22   JIZ   34Ø  /
     334   23   JMS   35Ø  ==>    Time hysteresis
     335   12   REO     Ø
     336   12   REO    12
     337   11   SEO    15
     338   2Ø   JMP   3ØØ   :  Output O15

     34Ø   23   JMS   35Ø  ==>    Time hysteresis
     341   12   REO     Ø
     342   12   REO    15
     343   11   SEO    12   :  Output O12 (min. limit value < nominal value < max. limit value)
     344   2Ø   JMP   3ØØ
```

Subroutine "time hysteresis":

```
     35Ø   14   STR   284
     351   ØØ    ØØ    1Ø
     352   25   WIH   284 *
     353   24   RET     Ø
```

*) If the time hysteresis exceeds one second, the actual value will not be displayed continuously because of the wait command.

**) Use the routine of the previous example also for module PCA1.W3.. The addresses 4Ø6...414 must be adapted to the addresses 31Ø...318.

Example 24: A practical example using an automatic drilling machine

Task

Both the mechanism and step diagram for an automatic drilling machine are specified. Part of the operating console is also specified. The extension, however, is dependent on the facilities offered by the selected PLC.

General functional description (see drawing on next page)

Circular plates are automatically fed, eccentrically drilled and then ejected.

STEP 1: Plates are pushed out of the magazine into the drilling position with piston A and then clamped.
STEP 2: Drill motor switched on and drill lowered.
STEP 3: Drill raised after approximately 4s to remove swarf from hole.
STEP 4: Drill lowered once more until drilling depth reached.
STEP 5: Drill raised.
STEP 6: Drill motor switched off and piston A withdrawn.
STEP 7: Ejector piston C forward.
STEP 8: Ejector piston C back.
Recommence from step 1.

Sensors

All piston end positions are signalled by sensors. Normally closed contacts are used for switching-off functions for wire breakage safeguard reasons. We have selected normally open contacts so that the functions can be more easily simulated.

The minimum magazine contents is also monitored by a sensor. The drill must be checked for breakage by a sensor at the start of each cycle, the machine being stopped where necessary.

Operating console

In addition to the "Start" and "Stop Cycle" functions, there should also be the facility of resetting the program back to its starting point ("Reset Program"). Also, to prevent unintentional resetting it is necessary for the pushbutton "Stop" and "Reset" to be actuated twice.

Emergency stop is effected on the hardware side by a mushroom-type pushbutton acting directly on the main contactor (regulation).

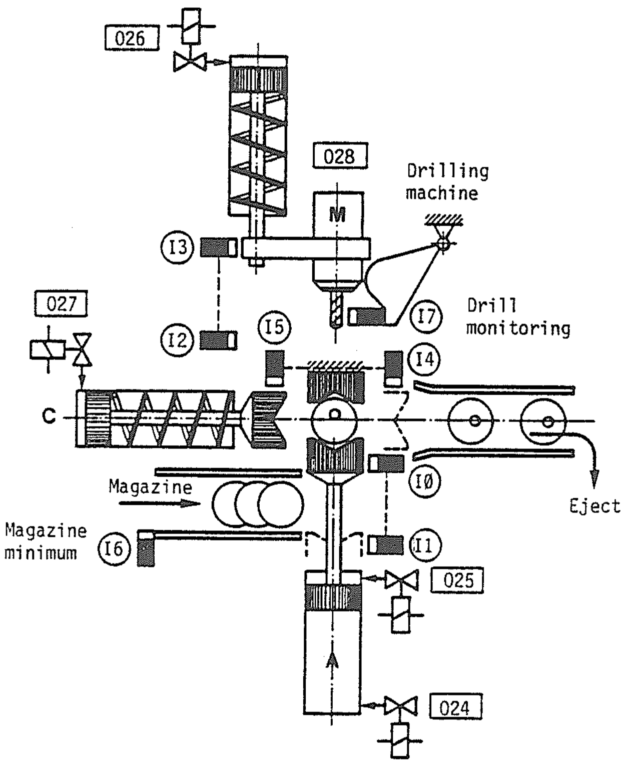
A facility should be available for preselection of the desired number of items per order (1ØØ to 1Ø,ØØØ) with BCD switches. Input or alteration of the pre-selected number of items should be by means of a pulse switch.

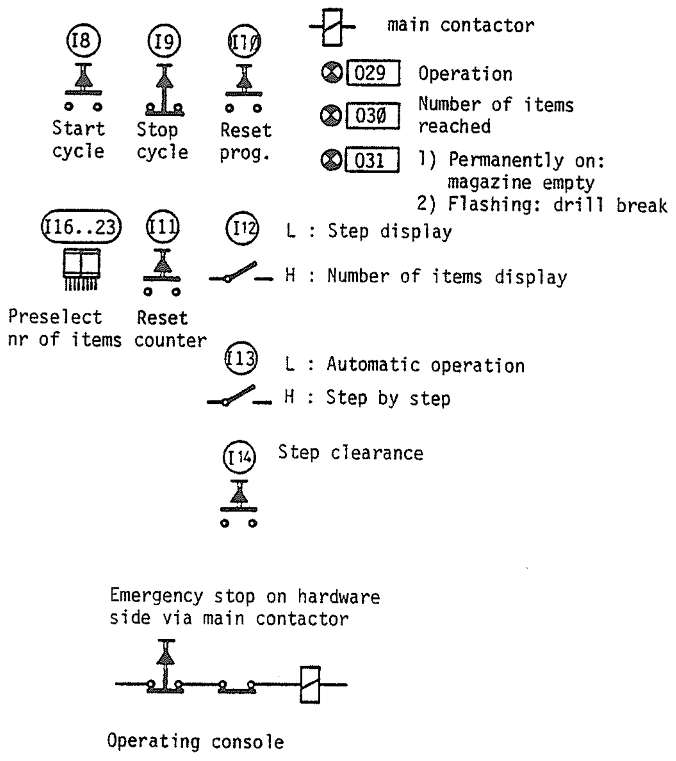
For setting-up and maintenance purposes, a facility should be provided for the entire sequence to be run step by step.

It is desirable to display the number of remaining items and to find out in the event of a fault at which step the machine has stopped or which function is awaited.

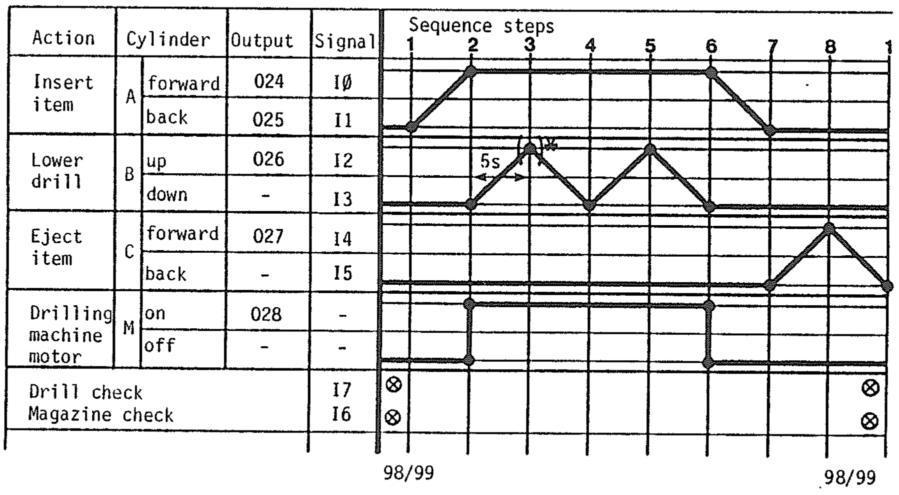
The various functions should be signalled with lamps in accordance with the illustrations.

Drawing shows sequence step 2





Step diagram



Action	Cylinder		Output	Signal
Insert item	A	forward	024	I0
		back	025	I1
Lower drill	B	up	026	I2
		down	-	I3
Eject item	C	forward	027	I4
		back	-	I5
Drilling machine motor	M	on	028	-
		off	-	-
Drill check				I7
Magazine check				I6

*Alternatively drill for 5 sec. or reaching limit position I2.

24.1 Dimensioning the PLC

24.1.1 Functions

In order to satisfy the imposed functions it is necessary to have sequence controls, timing and counting functions in addition to logical linkages. Also required are facilities for counter displays. A task of this scope poses no problems for any SAIA°PLCs.

24.1.2 Number of I/O

Because of the small number of I/O it is possible to use a PCA1. The I/O are listed on the sheet provided for this purpose. It must be noted that the I/O distribution can be in a matrix of 8 or, if required, 4 I/O.

Omitted in this example is the output for indicating magazine or drill monitoring. Solution: Since these displays occur relatively rarely, we use the same output and differentiate by means of continuous or flashing lamp. In this way we can find space on a PCA151.

24.1.3 Type of I/O

We select non-isolated 24VDC I/O modules with transistor outputs, selecting solenoid valves and lamps accordingly.

24.1.4 Memory capacity

As this program is not complex, we can calculate with a linkage depth of approximately 5. If we calculate the 8 inputs of the BCD switch as only 1 input, we obtain:
<u>24 I+O x linkage depth 5 = 12Ø memory locations</u>.
These are easily accomodated in the 1K memory.

We require an external interface with relay for the 22ØVAC drill motor. 24VDC is advantageous on the input side for the use of proximity switches for the transmitters.

24.1.5 Displays

Using module PCA1.D11, the necessary displays can be provided without the loss of I/O.

System: Automatic drilling machine

saia°plc PCA151

Operating console	Magazine empty or drill break (flashes)			31	PCA1.A1Ø
	Set item nr reached			3Ø	
	Operate			29	
	Drill motor			28	
	Piston C forward			27	
	Lower drill			26	
	Piston A back			25	
	Piston A forward			24	
Operating console	Preselection of nr of items via two BCD switches	1ØØØs	$2^Ø$	23	PCA1.E1Ø
			2^1	22	
			2^2	21	
			2^3	2Ø	
		1ØØs	$2^Ø$	19	
			2^1	18	
			2^2	17	
			2^3	16	
Operating console				15	PCA1.E1Ø
	Step clearance			14	
	Auto/step by step			13	
	Display counter			12	
	Reset item counter			11	
	Reset Progr. (E9)			1Ø	
	Stop cycle			9	
	Start cycle			8	
	Drill monitoring			7	PCA1.E1Ø
	Magazine min.			6	
	Piston C back			5	
	Piston C forwards			4	
	Drill up			3	
	Drill down			2	
	Piston A back			1	
	Piston A forward			Ø	

I/O assignment sheet

24.2 Programming

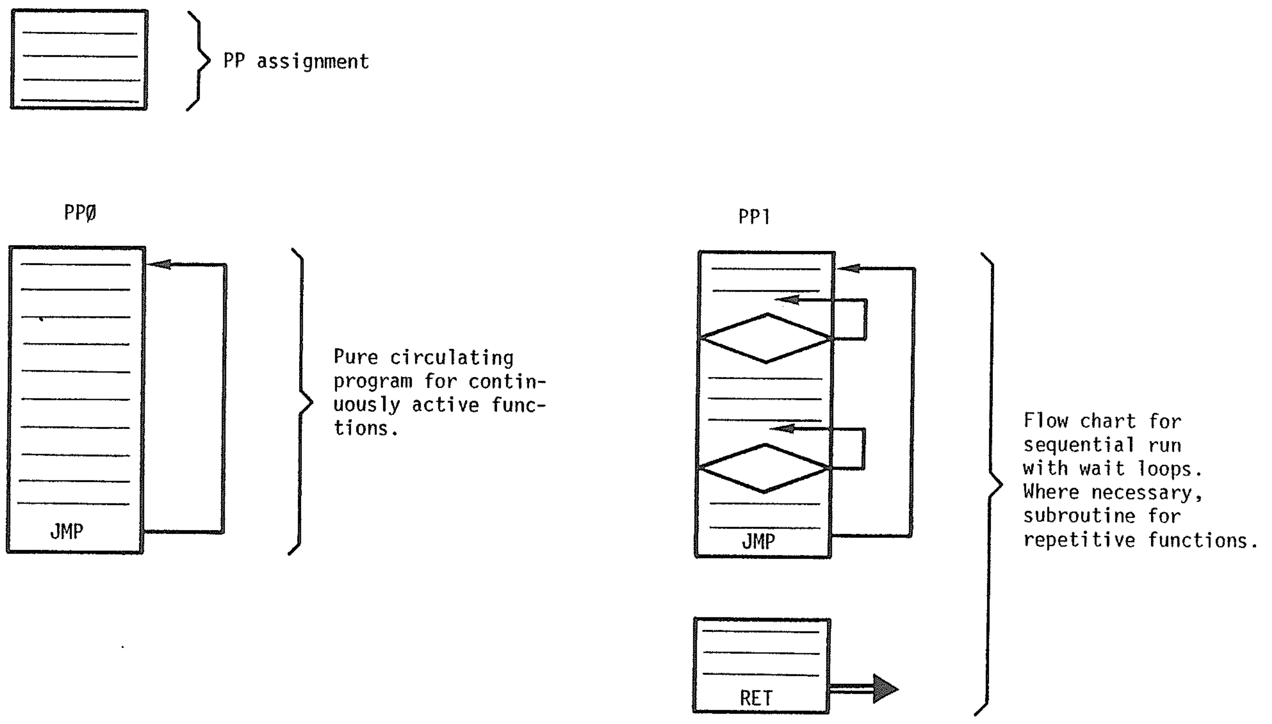
Experienced PLC engineers will find the narrative description together with the illustrations adequate for preparing the program or the flowchart.

The following steps are recommended for beginners:

24.2.1 Program structure

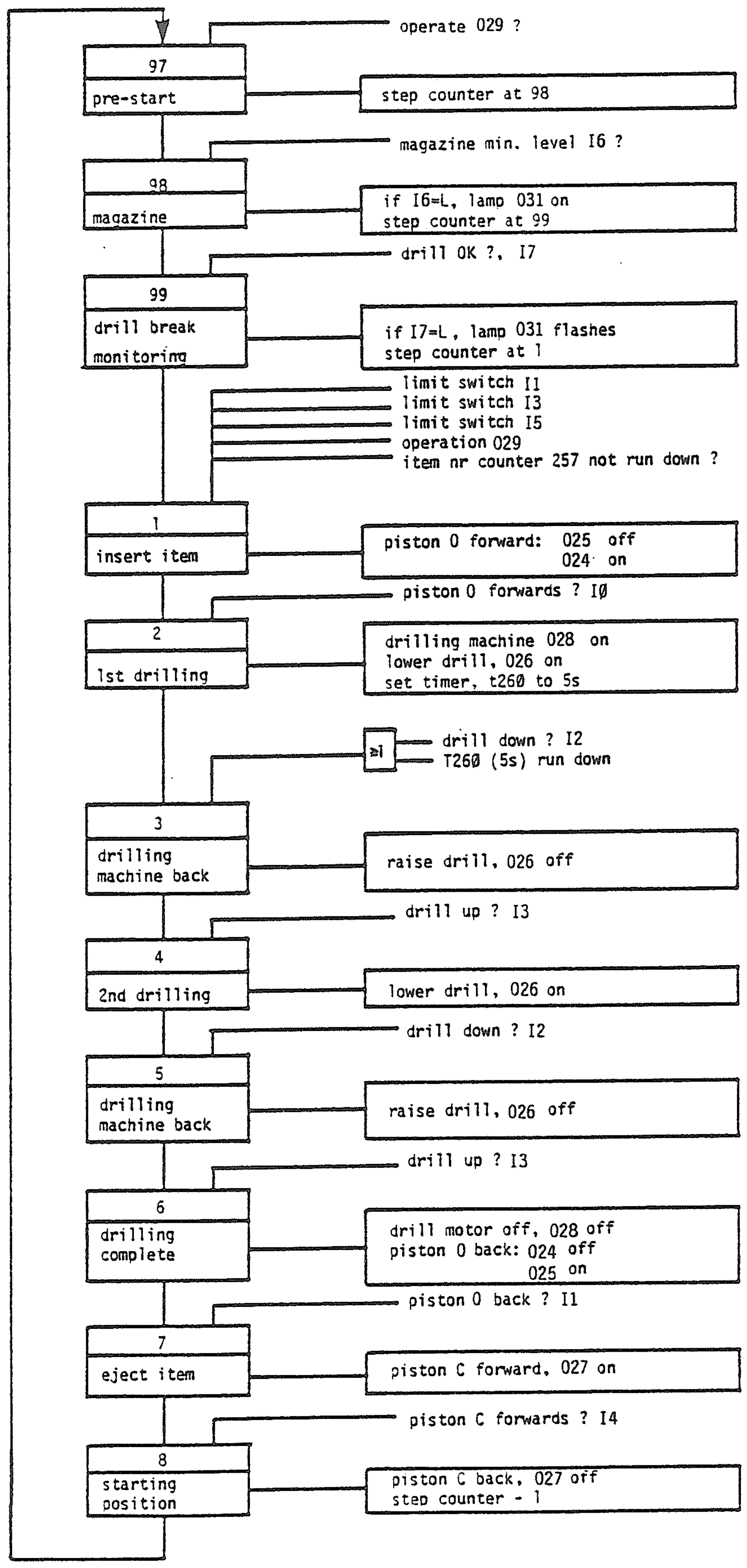
This is a sequential operation which is well suited to programming by means of a
flowchart. For continuously active functions such as start/stop, displays etc. we select a circulating program as a parallel program.

The structure is as follows:


PP assignment
PPØ
JMP
Pure circulating program for continuously active functions.
PP1
JMP
RET
Flow chart for sequential run with wait loops. Where necessary, subroutine for repetitive functions.

24.2.2 Step control plan in accordance with DIN

Where this method has already been used, the narrative description and step diagram can first be represented in this form. It will be noted that prior to step 1, a few additional steps are necessary for checking different functions. Only functions which have to be performed prior to each cycle run are the subject of these preliminary steps. Continuously active functions are in the circulating program PPØ.


operate 029 ?
97
pre-start
step counter at 98
magazine min. level I6 ?
98
magazine
if I6=L, lamp 031 on
step counter at 99
drill OK ?, I7
99
drill break
monitoring
if I7=L, lamp 031 flashes
step counter at 1
limit switch I1
limit switch I3
limit switch I5
operation 029
item nr counter 257 not run down ?
1
insert item
piston 0 forward: 025 off
024 on
piston 0 forwards ? I0
2
1st drilling
drilling machine 028 on
lower drill, 026 on
set timer, t260 to 5s
≥1
drill down ? I2
T260 (5s) run down
3
drilling
machine back
raise drill, 026 off
drill up ? I3
4
2nd drilling
lower drill, 026 on
drill down ? I2
5
drilling
machine back
raise drill, 026 off
drill up ? I3
6
drilling
complete
drill motor off, 028 off
piston 0 back: 024 off
025 on
piston 0 back ? I1
7
eject item
piston C forward, 027 on
piston C forwards ? I4
8
starting
position
piston C back, 027 off
step counter - 1

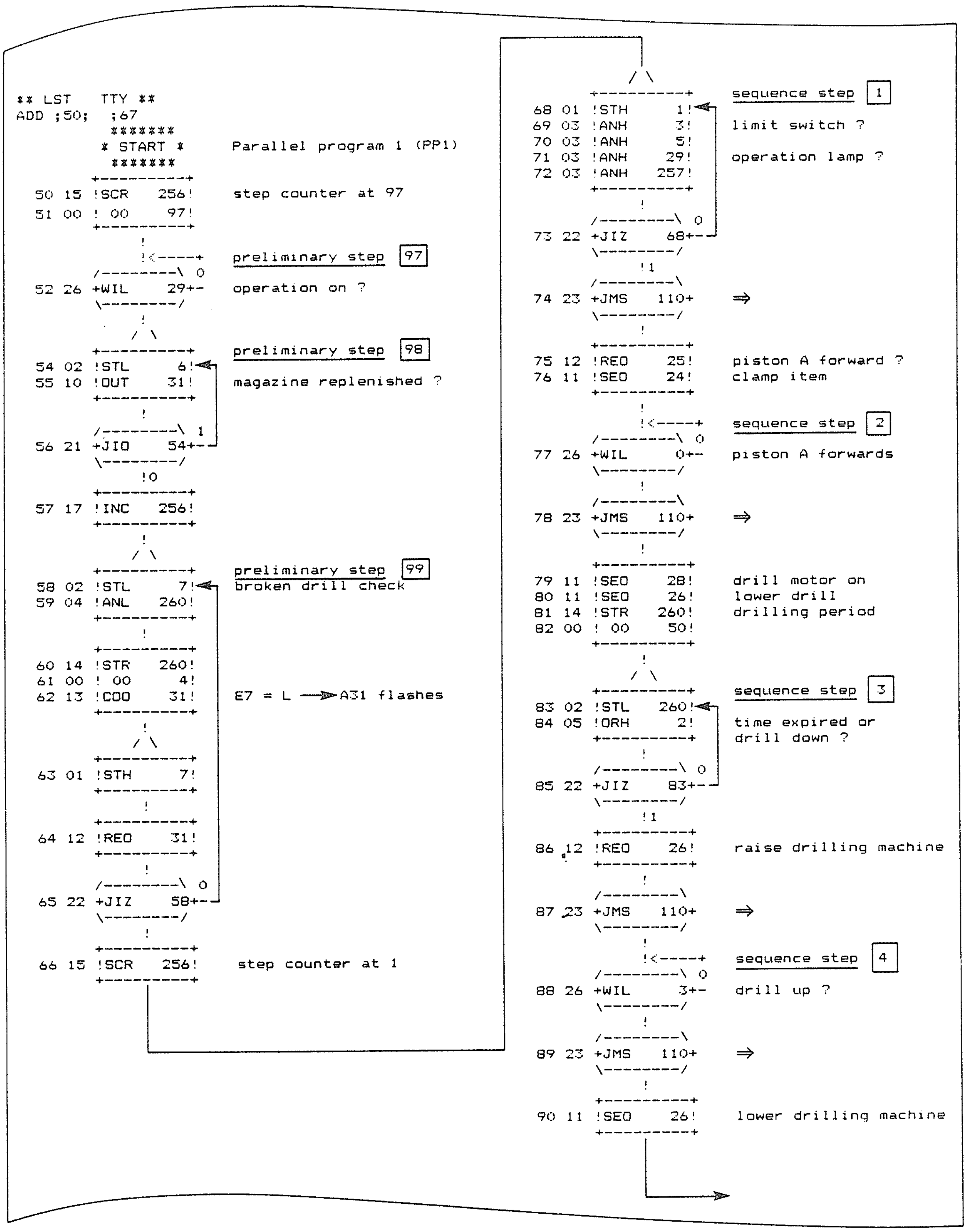
24.2.3 Program preparation

The preparation of the sequence program can now be carried out very simply on the basis of the DIN plan. Because the step by step operation and the program step functions occur in like manner in each sequence step they are included in a subroutine. Here, however, it is important to note that dangerous functions are to be previously reset before waiting for the step controller, i.e. first all REO, then JMS and then SEO.

The circulating program, as already mentioned, includes the continuously active functions. The final section with the program resetting is of significance here. In order to bring the sequence program back to its start, all important outputs are reset indexed and the PP1 is then assigned to its starting address anew.

```
PROGRAM PRINT-OUT WITH COMMENT: THANKS TO PCA-ASSEMBLER

*******************   PP ASSIGNMENT
ADDR NC   MNC   OPRD
   0 00   NOP      0
   1 29   PAS      1
   2 00    00     50
   3 20   JMP    10->
 +++++++++++++++++++  PARALLEL PROGRAM 0 (PP0)
  10 01   STH      8 } START/STOP CYCLE
  11 03   ANH      9 }
  12 11   SEO     29   "OPERATION" LAMP ON
  13 02   STL      9
  14 12   REO     29   "OPERATION" LAMP OFF
 -------------------  SET ITEM COUNTER
  17 01   STH     11
  18 09   DYN    300
  19 15   SCR    257
  20 18    18     23
  21 02   STL    257
  22 10   OUT     30   "SET ITEM NUMBER REACHED" LAMP
 -------------------  DISPLAY PRESELECTION
  25 01   STH     12
  26 31   DTC    257   ITEM NUMBER COUNTER DISPLAY
  27 08   NEG      0
  28 31   DTC    256   ITEM NUMBER COUNTER DISPLAY
 -------------------  RESET PROGRAM TO START (REASSIGNMENT)
  31 01   STH     10 }
  32 04   ANL      9 } RESET ?
  33 09   DYN    301 }
  34 22   JIZ    10->
  35 16   SEI      0
  36 12   REO   1024   RESETTING OF THE OUTPUTS
  37 27   INI      7
  38 21   JIO    36->
  39 29   PAS      1 } REASSIGNMENT OF PP1
  40 00    00     50 } & CONSEQUENT RESETTING
  41 20   JMP    10->
```

Program print-out in flowchart with PCA Assembler



```
** LST    TTY **
ADD ;50;   ;67
         *******
        * START *        Parallel program 1 (PP1)
         *******
      +----------+
 50 15 !SCR   256!       step counter at 97
 51 00 ! 00    97!
      +----------+
           !
           !<----+       preliminary step  [97]
      /---------\ 0
 52 26 +WIL    29+--     operation on ?
      \---------/
           !
          / \
      +----------+       preliminary step  [98]
 54 02 !STL     6!<--
 55 10 !OUT    31!       magazine replenished ?
      +----------+
           !
      /---------\ 1
 56 21 +JIO    54+--
      \---------/
           !0
      +----------+
 57 17 !INC   256!
      +----------+
           !
          / \
      +----------+       preliminary step  [99]
 58 02 !STL     7!<--    broken drill check
 59 04 !ANL   260!
      +----------+
           !
      +----------+
 60 14 !STR   260!
 61 00 ! 00     4!
 62 13 !COO    31!       E7 = L ——> A31 flashes
      +----------+
           !
          / \
      +----------+
 63 01 !STH     7!
      +----------+
           !
      +----------+
 64 12 !REO    31!
      +----------+
           !
      /---------\ 0
 65 22 +JIZ    58+--
      \---------/
           !
      +----------+
 66 15 !SCR   256!       step counter at 1
      +----------+
```

```
              / \
          +----------+     sequence step  [1]
 68 01    !STH     1!<--
 69 03    !ANH     3!      limit switch ?
 70 03    !ANH     5!
 71 03    !ANH    29!      operation lamp ?
 72 03    !ANH   257!
          +----------+
               !
          /---------\ 0
 73 22    +JIZ    68+--
          \---------/
               !1
          /---------\
 74 23    +JMS   110+      ⇒
          \---------/
               !
          +----------+
 75 12    !REO    25!      piston A forward ?
 76 11    !SEO    24!      clamp item
          +----------+
               !
               !<----+     sequence step  [2]
          /---------\ 0
 77 26    +WIL     0+--    piston A forwards
          \---------/
               !
          /---------\
 78 23    +JMS   110+      ⇒
          \---------/
               !
          +----------+
 79 11    !SEO    28!      drill motor on
 80 11    !SEO    26!      lower drill
 81 14    !STR   260!      drilling period
 82 00    ! 00    50!
          +----------+
               !
              / \
          +----------+     sequence step  [3]
 83 02    !STL   260!<--
 84 05    !ORH     2!      time expired or
          +----------+     drill down ?
               !
          /---------\ 0
 85 22    +JIZ    83+--
          \---------/
               !1
          +----------+
 86 12    !REO    26!      raise drilling machine
          +----------+
               !
          /---------\
 87 23    +JMS   110+      ⇒
          \---------/
               !
               !<----+     sequence step  [4]
          /---------\ 0
 88 26    +WIL     3+--    drill up ?
          \---------/
               !
          /---------\
 89 23    +JMS   110+      ⇒
          \---------/
               !
          +----------+
 90 11    !SEO    26!      lower drilling machine
          +----------+
               |
               +——>
               !<-----+     Sequence step [5]
       /--------\ 0
91 26  +WIL      2+-        Drill down?
        \--------/
            !
      +----------+
92 12 !REO     26!          Raise drill
      +----------+
            !
       /--------\
93 23  +JMS    110+         =>
        \--------/
            !
            !<-----+        Sequence step [6]
       /--------\ 0
94 26  +WIL      3+-        Drill up?
        \--------/
            !
      +----------+
95 12 !REO     28!          Drill motor off
      +----------+
            !
       /--------\
96 23  +JMS    110+         =>
        \--------/
            !
      +----------+
97 12 !REO     24!          Piston A back
98 11 !SEO     25!
      +----------+
            !
            !<-----+        Sequence step [7]
       /--------\ 0
99 26  +WIL      1+-        Piston A back?
        \--------/
            !
       /--------\
100 23 +JMS    110+         =>
        \--------/
            !
      +----------+
101 11 !SEO    27!          Eject item
      +----------+
            !
            !<-----+        Sequence step [8]
       /--------\ 0
102 26 +WIL      4+-        Piston C forward?
        \--------/
            !
      +----------+
103 12 !REO    27!          Piston C back
      +----------+
            !
       /--------\
104 23 +JMS    110+         =>
        \--------/
            !                 ^
      +----------+            |
105 18 !DEC   257!          Item counter -1
      +----------+            |
            !                 |
       /--------\             |
106 20 +JMP     50+ ----------+
        \--------/
```

```
ADD :110,  :114
           *******
=>         * START *        Subroutine for step by step
           *******
              /
      +----------+
110 01 !STH     13!         Step by step switch on?
      +----------+
            !
       /-------\ 0
111 22 +JIZ    113+--+      if no, jump
        \-------/    |
            !1       |
            !<-----+ |
       /-------\ 0   |
112 26 +WIL     14+- |      if yes, await step release
        \-------/    |
            !        |
      +----------+   |
113 17 !INC    256!<-+      Step counter +1
      +----------+
            !
       /-------\
114 24 +RET      0+
        \-------/
```

```
CROSS-REFERENCE OF SOME ELEMENTS AND
JUMP ADDRESSES FOR SUBROUTINE 110


                  STEP COUNTER
ADD :0,   :114,  OPERAND :256

   28 31   DTC    256
   50 15   SCR    256
   53 17   INC    256
   57 17   INC    256
   66 15   SCR    256
  113 17   INC    256
                     :

                  STEP COUNTER
ADD :0,   :114,  OPERAND :257

   19 15   SCR    257
   21 02   STL    257
   26 31   DTC    257
   72 03   ANH    257
  105 18   DEC    257
                     :

                  SUBROUTINE 110
ADD :0,   :114,  OPERAND :110

   74 23   JMS    110
   78 23   JMS    110
   87 23   JMS    110
   89 23   JMS    110
   93 23   JMS    110
   96 23   JMS    110
  100 23   JMS    110
  104 23   JMS    110
                     :

                  OPERATION LAMP
ADD :0,   :114,  OPERAND :29

   12 11   SEO     29
   14 12   REO     29
   52 26   WIL     29
   71 03   ANH     29
                     :

                  DRILLING MACHINE UPPER LIMIT SWITCH
ADD :0,   :114,  OPERAND :3

   69 03   ANH      3
   88 26   WIL      3
   94 26   WIL      3
                     :
```

Procedure for solving a control problem by the introduction of a PLC

S T A R T

$\downarrow$

Compile a specification for the overall process (where necessary with sketches, description of sequences, subdivision into process blocks etc.)

Outline design showing which operations are performed mechanically, pneumatically, hydraulically or electrically.

PLC dimensioning

- Determine number of I and O (giving consideration to multiplexing facilities)
- Define I/O types (DC/AC, voltage, current)
- Determine design capacity of PLC (PCA1 ≤ 112 I + O, PCA2 ≥ 96 I + O), possible break down of the process between several PLCs, hierarchy)
- Determining the PLC modules (grids 8/16/32) giving consideration for a 10 - 20% reserve of I + O
- Determine memory capacity in accordance with following rough guidelines

- simple control	memory capacity = approx. 5 x no. of I/O
- average complexity	memory capacity = approx. 10 x no. of I/O
- high complexity	memory capacity = approx. 20 x no. of I/O

- Check whether program can be operated with SAIA°PLC standard functions. Type PCA14 or PCA23 to be implemented where there is a requirement for universal arithmetic functions or comprehensive documentation facilities.
- Definitively determine memory type (Standard = EPROM).

Ordering the PLC

Precise stipulation of PLC equipment including accessories such as cables, watchdog, external interfaces, display module, programming and simulation accessories.

$\downarrow$

T A S K S H A R I N G

$\swarrow \quad \searrow$

T A S K S H A R I N G

Hardware

- Completion of detailed design
- Construction of mechanical components

On delivery of the PLC:
- Installation of the PLC into system
- Wiring the I/O
- Checking the I/O wiring with LED and "MAN" operating mode

Program preparation

- Designation of the I/O
- Defining the program structure and programming methods for correspon-ding part programs (program mo-dules)
- Compiling the program on paper (giving consideration for trouble-shooting using DOP, DTC and watchdog).

When programming location is already available or PLC has been delivered:
- Enter program (enter NOP for reserve and alterations)
- Test out program parts using simulators
- Rectify any faults
- Prepare provisional documentation
- Safeguard the program on EPROM

Commissioning

- If simulated program test and I/O test were successful on "MAN" ---> switch on (where applicable isolate dangerous control elements, do not actuate start pushbutton immediately).
- Where step-by-step mode is provided (do not confuse with STEP mode), operate process in this manner initially.
- Check out automatic mode (where applicable also check dangerous operating circumstances such as wire breakage, simultaneous actuation of several pushbuttons, power failure or process breakdown).
- Where necessary correction of the program on RAM.
- If OK copy program on to EPROM in two sets, 1 set for operation, 1 set as a program safeguard.
- Update program documentation and add comments so that program can be under-stood by third persons.
- Compile trouble-shooting guidelines for maintenance personnel.

E N D

Overview of the instruction set

SAIA AG

Industrial Electronics and Components
3280 Murten/Switzerland

Telephone	037 727111
Telefax	037 714443
Telex	942127

Further representatives

Belgique	Landis & Gyr Belge SA, Dépt. Industrie Avenue des Anciens Combattants 190, B-1140 Bruxelles ☎ 02 2440211, Tx 65930, Fax 02 2428831
Danmark	Skandia-Havemann Vallensbækvej 46, DK-2625 Vallensbæk ☎ 02 643333, Tx 33383, Fax 02 642245
Deutschland	SAIA GmbH Flinschstrasse 67, D-6000 Frankfurt 60 ☎ 069 420993-0, Ttx 6999375, Fax 069 425654
España	Landis & Gyr BC SA Batalla del Salado 25, Apartado 575, 28045 Madrid ☎ 91 4671900, Tx 22976, Fax 91 2394479
France	SAIA Sàrl 10, Blvd. Louise Michel, F-92230 Gennevilliers ☎ 1 40860345, Tx 613189, Fax 1 47914013
Great Britain	A.S.A.P. Ltd. Unit 15D, Compton Place, Surrey Avenue, Camberley, Surrey GU15 3DX ☎ 0276 691580, Fax 0276 691581
Italia	SAIA S.r.l. Via Cadamosto 3 20094 Corsico MI
Nederland	Landis & Gyr BV, Div. Electrowater Kampenringweg 45, Postbus 444, NL-2800 AK-Gouda ☎ 01820 65683, Tx 20657, Fax 01820 32437
Norge	Malthe Winje & Co A/S Cort Adelersgt. 14, Postboks 2440, Solli, N-0202 Oslo 2 ☎ 02 558640, Tx 19629, Fax 02 552211
Österreich **COMECON**	Landis & Gyr Gesellschaft m.b.H Breitenfurterstrasse 148, Postfach 9, A-1230 Wien ☎ 0222 842626-0, Tx 132706, Fax 0222 842626313
Portugal	Infocontrol Electronica e Automatismo LDA. Av. da Igreja No. 68–1° Esq., P-1700 Lisboa ☎ 01 775161-65, Tx 63454, Fax 01 775687
Suomi **Finland**	OY Landis & Gyr AB SF-02430 Masala ☎ 8 029731, Tx 1001153, Fax 8 0297 5531
Sverige	Beving Elektronik AB St. Eriksgatan 113a, Box 21104, S-10031 Stockholm ☎ 08 151780, Tx 10040, Fax 08 336863
USA	After sales services: Maxmar Controls Inc. 99 Castleton Street, Pleasantville, New York 10570-3403 ☎ 914 7473540, Fax 914 7473567
Australia	Landis & Gyr (Australia) Pty Ltd 411 Ferntree Gully Road, P.O. Box 202, Mount Waverley, Vic. 3149 ☎ 3 544-2322, Tx 32244, Fax 3 5437496
Argentina	Electromedidor S.A.I. y C. Defensa 320, RA-1065 Buenos Aires ☎ 1 337125, Tx 23377, Fax 1 3319582

Subject to change without notice

Printed in Switzerland 26/721 E1 11.89 G02

SAIA