PLC

STD 1

PROGRAMMING MANUAL

FOR S&H MOTION CONTROLLER

- **Goya**
- **Picasso2000**
- **Rubens**

MANUAL VERSION : ***1.0***

Refers to software version:	**PLC**	*3.47*

MANUAL CODE : ***MA PLC U STD1 10***

DATA: 23 FEBRUARY 2000

Contents

1.1 Introduction

The I/O handling system, used in the S&H control series (PLC-ORANGE), is a tool for providing control logic for industrial automation systems and machine tools. It can be thought of as the natural partner of the position control (CNC), also present in the S&H controls, to create a full and flexible control system.
The feature of the I/O handling section are such that they cover all the typical functions of the work performed by a PLC, and so in all effects it may be considered to be one.

1.2 Structure of the PLC

ORANGE can be thought of as a virtual PLC, completely managed by software, whose internal structure is made up with:

- 1 bit accumulator that contains the result of operations on digital tags.
- Stack for the conservation of the intermediate results (handled automatically by the PLC).
- Image Memory that contains the copies of the I/O tags, the other tags handled by the program and the 16/32 bit variables.

The program can be subdivided into different SEQUENCES, each of which is an autonomous unit, capable of interacting with the others by means of the image memory.

The program is cyclic. In other words, the sequence of the instructions is followed from the beginning to the end and then repeated until the machine is switched off or a different program is loaded.

A program is typically structured in the following way:

• INPUT Cycle	- group of instructions for acquisition.
• Logic Operations	- execution program.
• OUTPUT Cycle	- group of instructions for writing.

The ***INPUT Cycle*** and the ***OUTPUT Cycle*** are there to transfer the values of the I/O tags into the internal memory of the PLC so that they can be handled more quickly during the execution cycle and, above all, where they create a stable **image** of the state of the field.
N.B.: There is a further acquisition and write phase for the tags and variables that the PLC exchanges with the CNC. This phase is automatically executed at the end of the read-execute-write cycle and exchanges the CNC data in a synchronous manner with the PLC program.

1.3 Organisation of the I/O and IMAGE MEMORY

The states of the **input** and **output** tags are read with the appropriate instructions and transferred into the internal memory where they are organised in a compressed area that represents them.
These real tags form part of the IMAGE MEMORY together with other internal tags. This part of the memory in practice contains a photograph of the situation on the outside world of system ORANGE and is updated in times controlled by the programmer.

1.3.1 Real tags

- There is a maximum of 256 real tags (128 Inputs and 128 Outputs), that corresponds to the external inputs and outputs, and can be handled by ORANGE. The actual number of **input** and **output** tags depends on the physical configuration of the control model.

- **External Inputs**. There are 128. Their state represents the image of the external physical inputs.
- **External Outputs.** There are 128. Their state represents the image of the external physical outputs

1.3.2 Internal tags

The internal tags are those tags whose use is identical to that of the real tags but whose value does not correspond to a digital input or output as they are manipulated entirely within the program itself. Their main purpose is help in the resolution of complex functions that it is more convenient to split up into sub-functions, with the memorisation of intermediate values stored as these internal tags.
There are different types of internal tags, with different quantities and uses. All may be handled directly by the PLC program ORANGE by means of the instructions SET, EQU, CLEAR, AND, ANDNOT, EQUNOT, CHANGE. They are identified with names that depend on the main function, which is associated with them.

- **Sequences**. There are 64 and they have the main purpose of activating or disactivating the sequences that make up the PLC program. From the moment that a program can be not made up of 64 sequences, it is possible to use the remaining tags for a use that is identical to that of the markers.

- **Flags**. There are 64. Their purpose is to receive particular instructions from the CNC handling program. They can only be set to 1 by the CNC by means of the M functions (from M10 to M41) and the T functions (from T00 to T31). The first 32 are dedicated to the M functions and the other 32 to the T functions. The PLC normally sets them to 0 to indicate the execution of a request. Thus these points are the flags that can be raised by the CNC to request a service from the PLC and that the PLC lowers when it has performed that service. The CNC does not continue its own program until the flags are lowered.

- **Counters**. There are 32. Although these are handled in exactly the same way as the other internal tags, their typical use is linked with the same number of down counters. It is possible to assign values to the counters from the program and enable them to count the transitions of a digital input. When the count reaches zero, the internal tag associated with the counter assumes that value 1 to indicate that the count is completed. An external input tag defined with an appropriate instruction is used as a signal that makes the counter decrement.
- **Timers**. There are 32. Although these are handled in exactly the same way as the other internal tags, their typical use is linked with the same number of timers. It is possible to assign values to the timers from the program and enable them to count the time that has passed. When the set time has elapsed, the internal tag associated with the timer assumes the value of 1 to indicate that the time has elapsed.
- **Markers**. There are 256. These are used to store intermediate values. The first 32 can be used to recognise the transitions of input tags (see the DIFUP/DIFDN instructions). Furthermore, half of the markers are not zeroed when the machine is switched on (they keep their state buffered) with the purpose of providing, if necessary, a tool for the hot start of the PLC program. Also, 16 of these tags are reserved as they have predefined meanings (arithmetic flags).
- **Auxiliary Markers**. There are 256 and as Markers they have the purpose of storing the intermediate values and to be used as support tags.
- **CNC Inputs**. There are 32. Their value can be modified by the CNC that with such tags can therefore communicate the result of its executions to the PLC.
- **CNC Outputs**. There are 32. Their value can be modified by the PLC that with such tags can therefore communicate the results of its executions to the CNC. They can also be used to transfer the values of input tags, whose boards are not visible directly, to the CNC.

1.3.3 Layout of the image memory

The tags are arranged in the following order in the image memory:

– 128	External Inputs	(000 - 127)
– 128	External Outputs	(128 - 255)
– 64	Sequences	(256 - 319)
– 64	Flags	(320 - 383)
– 32	Counters	(384 - 415)
– 32	Timers	(416 - 447)
– 256	Markers	(448 - 703)
– 256	Auxiliary Markers	(704 - 959)
– 32	Inputs from CNC	(960 - 991)
– 32	Outputs to CNC	(992 - 1023)

The total number of tags that may be handled is therefore 1024 and each is identifiable by a number between 0 and 1023. The third timer, for example, is identified by number 418.

This is not the only way in which the tags can be identified. They can be identified inside their group with a number followed by an index that identifies the group. So the third timer, previously identified by number 418, can be called 2(T), where the letter T indicates that it is in the timer group (numbered from 0 to 31).

The letters that identify the groups are:

– **I**	for the External Inputs
– **O**	for the External Outputs
– **S**	for the Sequences
– **F**	for the Flags
– **C**	for the Counters
– **T**	for the Timers
– **M**	for the Markers
– **A**	for the Auxiliary Markers
– **D**	for the Inputs from the CNC
– **E**	for the Outputs to the CNC

Evidently, the digital inputs being associated to the starting point of the image memory, their identification with the index I will be the same as their number without index.

1.3.4 Numerical Variables

The system ORANGE possesses the ability to perform arithmetic operations on the registers (subsequently called also words or variables), each made up of 16 or 32 bits. The registers are obtained by regrouping the suitable number of tags in the image memory. To ensure greater flexibility, a further area of memory is available that is reserved for a certain number of these additional registers. It immediately follows that of the image memory of the I/O tags, making thus a single large area (that goes from the tag of Input 0 up to the register with address of 255), in which the registers are identified in groups of 16 or 32 bits.
The registers are treated as 16 or 32 bits according to the instruction that is used to manipulate it. In any case, the addresses with which the variables are identified is that which identifies a 16 bit, and, therefore, the 32 bit variables can only have an even address. Thus the variable number 0 is made up of the first 16 or 32 bit of the External Inputs while number 63 is made up of the last 16 bits of the CNC Outputs tags.

The allocations in the area reserved for the registers are:
- 32 x 32 bit variables (or 64 to 16 bit) for general use by the PLC.
- 8 x 32 bit variables written by the CNC and read by the PLC.
- 8 x 32 bit variables written by the PLC and read by the CNC.
- 8 x 32 bit variables dedicated to special inputs for the PLC.
- 8 x 32 bit variables dedicated to special inputs from the PLC.

As opposed to the logic operations on the I/O tags, which use an accumulator to store the result, the operations on variables place the result directly in the destination variable, contemporaneously changing the state of some MARKERS that can subsequently be read (see JUMP instruction) to make a decision on the result of the operation. Excluded from this logic are the operations of copying from one register to another.
The markers used for this purpose (called “arithmetic markers” and which are not used as normal internal tags) are:

- **247(m)** Marker whose value passes from 0 to 1 (and viceversa) at intervals of one second.
- **248(m)** Value is always equal to 1.
- **249(m)** State of the accumulator.
- **250(m)** Set to one if the result of the last operation is zero or if the comparison between two variables has identified that they are equal.
- **251(m)** Set to one if the result of the last operation is not zero or if the comparison between two variables has identified that they are not equal.
- **252(m)** Set to 1 if the comparison has been made between two variables and the first is greater than the second.

- **253(m)** Set to 1 if the comparison has been made between two variables and the first is smaller than the second.
- **254(m)** Set to one if the result of the last operation performed is larger that the calculation capacity of the PLC.
- **255(m)** Negated state of the accumulator.

1.4 Instruction sets

The operations that may be performed by system ORANGE involve digital I/O. The instructions that may be used are therefore typical of Boolean logic, to which are added those of transfer from and towards the outside world and the handling of timers.
The logic is positive, that is, tags are considered active with the value 1 or TRUE. The AND operation, therefore, gives the result as 1 only if the accumulator and the tag being tested both have the value equal to 1.

In the following examples, numerical values or symbols are used indiscriminately where it is permitted, according to the rules that can be found in the paragraph relative to the use of the PLC program translator (Compiler).

Group Identifiers (operands)				
ID	External Inputs	128	000 : 127	var. 0 : 7
OD	External Outputs	128	128 : 255	var. 8 : 15
SQ	Sequences	64	256 : 319	var. 16 : 19
FL	Flags	64	320 : 383	var. 20 : 23
CN	Counters	32	384 : 415	var. 24 : 25
TM	Timers	32	416 : 447	var. 26 : 27
MK	Markers	256	448 : 703	var. 28 : 43
MA	Auxiliary Markers	256	704 : 959	var. 44 : 59
IC	CNC Inputs	32	960 : 991	var. 60 : 61
OC	CNC Outputs	32	992 : 1023	var. 62 : 63
VC	CNC Variables —> PLC	8		var. 64 : 79
VP	PLC Variables —> CNC	8		var. 80 : 95
VI	Special IN variables	8		var. 96 : 111
VO	Special OUT variables	8		var. 112 : 127
VA	Variables	64		var. 128 : 255

Table A. IMAGE MEMORY Group Identifier Codes.

2. Description of the PLC Instructions

2.1 The "instruction" line

Every ORANGE instruction is made up of at least the operative code and, optionally, of one or more operands and from some auxiliary information for the programmer or for the translator in machine code (Compiler).

The following adopts the convention to close the optional information in square brackets and to indicate one or more character separators with the symbol "_" that can be symbols of spacing, of tabulation or the comma.

The line of program is made up as follows:

[label][_Codop[_Oper1[(Ind)][,Oper2[(Ind)]]]][_][;Comm]

Where:

- **"label"** is a symbol that is used to unequivocally identify the line of program for the control instruction (see JUMP, CALL, etc.). It can also be used to make the program more readable in order to show the logical separation between blocks. It is noted that it must begin at the first column and end with the first character separator.
- **"Codop"** is the operative code of the instruction that cannot begin from the first column.
- **"Oper1"e "Oper2"** are the possible operands
- **"Ind"** is the index that indicates the spacing of the image memory and it can assume one of the values I, M, F, S, T, C, O (see parag.1.3.3 Layout of the image memory)
- **"Comm"** is a phrase of clarification introduced by the programmer to assist the comprehension of the program.

Notes that the following special cases are allowed:

a) Blank line

b) Line that only contains comment

c) Line with only the label

For example, the instruction

```
        AND  3              ; The third bit controls...
```

contains only some of the possible components; it contains the operative code, it does not have the label, it does not have the index of the first operand and does not have a second operand. There is a comment that is separated by spacing characters from the rest of the instruction.

2.2 Addressing by Symbol

Every tag or register is identified by a number that goes, for the tags, from 0 to 1023, and for the registers, from 0 to 255.

The numbers, in the system ORANGE, can also be written to base 16; to distinguish them from those written to base 10, they are followed by the letter 'H' (HEX), for example 12H.
In the same way, to distinguish a number to base 16 from any symbol, they must always begin with a number. It must therefore be written as 0A12H and not A12H.

To help the programmer in laying out the program and, above all, help those who must interpret a program written by another, the system ORANGE makes it possible to identify the objects of the PLC by using symbols that are associated to the numbers. The numbers, however, remain the real identifiers of such objects.

There are some rules to follow when using symbols; these rules, which are valid also for writing labels, are:

1) They can use letters, numbers or special characters provided that the symbol begins with a letter.

E.g. P412 or Inp1 or PIPPO.

2) There is no distinction between upper and lower case letters.

E.g. PIPPO is the same as PiPpo

2.3 Pseudo instructions

An ORANGE program is made up of a succession of instructions that, suitably translated and downloaded into the PLC, are interpreted and executed.

There are a few instructions that do not get translated for the PLC that also form part of the program. These are interpreted directly by the translation program and thus are identified by the term "pseudo instructions". They are used to facilitate the task of the programmer, preventing it from writing the same thing over and over again.

These are:

ASSIGN

INCLUDE.

The pseudo instruction ASSIGN has the purpose of assigning a value to a symbol. The syntax of ASSIGN is:

SYMBOL ASSIGN OTHER

Where:

"*SYMBOL*" is the symbol to define;

"*OTHER*" is the value that becomes associated to the symbol. It can be a number or a previously defined symbol. In both cases it can be accompanied by an index.

E.g.:

```
POSITX     ASSIGN     34
POSITY     ASSIGN     POSITX(M)
```

The pseudo instruction INCLUDE tells the compiler that the reading of the file to translate must continue with a new file, and then return to finish with that that was interrupted. It is especially useful to keep a single file for defining the common symbols for different projects, but it functions even if the file included contains executable instructions. It allows a single level of nesting, which means that it is not permitted to use a pseudo instruction INCLUDE in a file that is already included in another.

The syntax of INCLUDE is the following:

INCLUDE filename

Where:

"filename" is the name of the file to be read, present on the disk as FILENAME.PRG.

Example:

File 'PROGR01.PRG' is made up of:

```
        INCLUDE    BASEDEF

        AND  START
        AND  SAFETY
        SET  MOTOR1
```

File 'BASEDEF.PRG' is made up of:

```
        START          ASSIGN 5
        SAFETY         ASSIGN 12(M)
        MOTOR1         ASSIGN 6(O)
```

For the compiler it is as if the file 'PROGR01.PRG' were made up as follows:

```
START          ASSIGN 5
SAFETY         ASSIGN 12(M)
MOTOR1         ASSIGN 0(O)

        AND  START
        AND  SAFETY
        SET  MOTOR1
```

That is:

```
        AND  5(I)
        AND  12(M)
        SET  0(O)
```

2.4 List of the PLC Instructions (*Programmable Logic Controller*)

The following paragraphs list all the PLC instructions that can be performed by the controller. For every instruction, the syntax and the graphic symbol in Ladder Diagram is illustrated. Furthermore there is a brief description of the instruction, a statement on which operands are allowed and some important notes. Finally, for each instruction, there is an example of a typical way to use it.

2.4.1 INPUT Instruction

Syntax:

INPUT dest, sour

LADDER Symbol

┤INPUT├

Description:

Reading external inputs.
The physical inputs *sour* are read and deposited in their address in the IMAGE MEMORY specified by *dest*. It should be noted that the *sour* can have the value 0(1), to indicate that the inputs of the logic channel 0 (first board), and it can have the value 1(1) to indicate the inputs of the logic channel 1 (second board). The instructions can be placed anywhere in the program. Nevertheless, unless there is a particular reason to suggest otherwise, to create a logical structure for the program, it is advised that the INPUT instructions be placed at the beginning of the PLC program.

Operands:

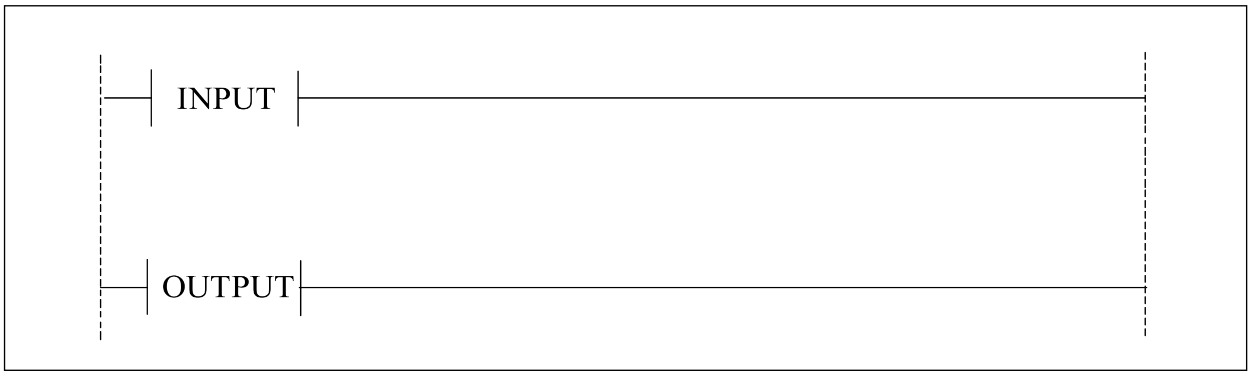
ID (table A page 13).

Notes:

The read instruction for the inputs is used typically for performing the *INPUT Cycle* of the PLC program.

Example:

```
input     0(i),0(1)    ;acquisition of inputs
;
;         Program code
;
output    0(1),0(o)    ;update outputs.
```



2.4.2 OUTPUT Instruction

Syntax:

OUTPUT dest, sour

LADDER Symbol

┤OUTPUT├

Description:

Writing to external outputs.
The zone of the IMAGE MEMORY specified by *sour* is read and used to update the physical output specified by *dest*. It is pointed out that *dest* can have the value 0(1), to indicate outputs of the logic channel 0 (first board), and it can have the value 1(1) to indicate the outputs of the logic channel 1 (second board). The instructions can be placed anywhere in the program. Nevertheless, if there is no particular reason to suggest otherwise, for the logical structure of the program, it is advised that the OUTPUT instructions be placed at the penultimate position of the PLC program. The last is always ENDPRO.

Operands:

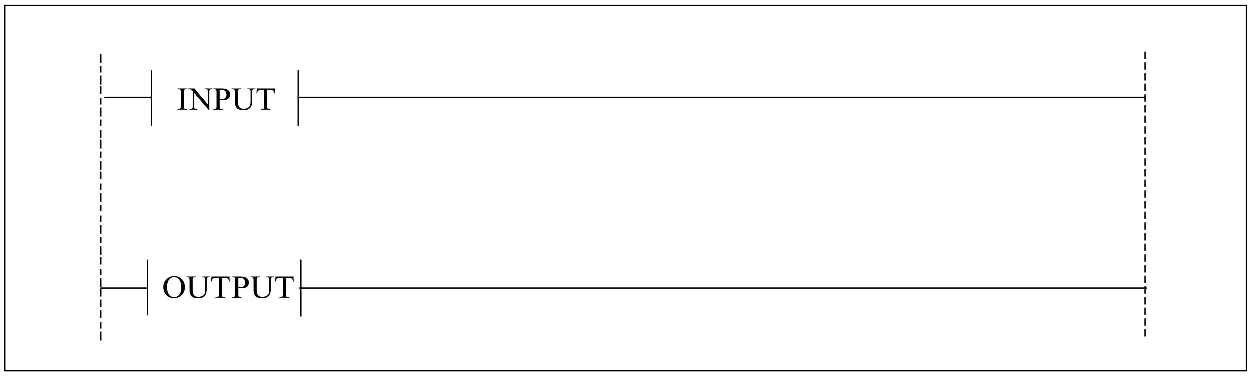
OD (table A page 13)

Notes:

The *write output* instruction is used typically to perform the *OUTPUT Cycle* of the PLC program.

Example:

```
input     0(i),0(1)    ;acquisition of inputs
;
;         Program code
;
output    0(1),0(o)    ;update outputs.
```

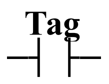


2.4.3 AND Instruction

Syntax:

AND tag

LADDER Symbol



Description:

Acc ⟵ Acc AND tag.
A logic AND operation is performed between the Accumulator and the tag of the IMAGE MEMORY specified. The result of the operation is placed in the accumulator. AND instructions can be put anywhere in a program.

Operands:

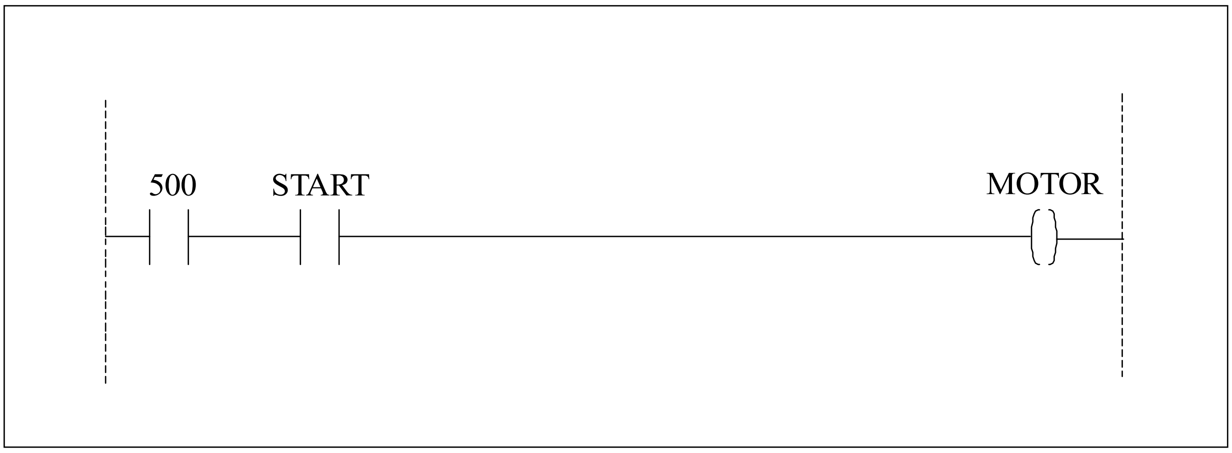
ID IC MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)      ;marker.
MOTOR    assign 1(m)      ;marker.
...
...
and 500                 ;logic AND between the accumulator and the tag
                        ;number 500 of the memory.
and START               ;logic AND between the accumulator and the ;tag
                        assigned to START.
   set MOTOR            ;the tag assigned to MOTOR is placed at logic ;1,
                        only if, both the tag 500 and the tag ;0(m) are
                        equal to 1
endblo
```

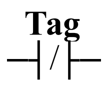


2.4.4 ANDNOT Instruction

Syntax:

ANDNOT tag

LADDER Symbol



Description:

Acc ← Acc ANDNOT tag.
A logic AND is performed between the Accumulator and the negated value of the tag specified in the IMAGE MEMORY. The result of the operation is placed in the Accumulator. ANDNOT instructions can be put anywhere in the program.

Operands:

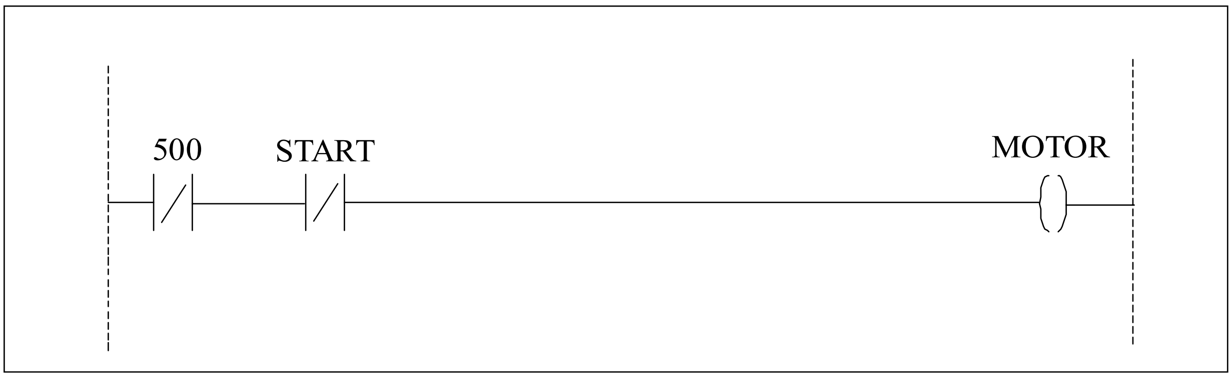
ID IC MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)     ;marker.
MOTORE   assign 1(m)     ;marker.
...
...
andnot 500        ;logic AND between the accumulator and the ;negated
                  value of tag number 500 in the memory.
andnot START      ;logic AND between the accumulator and the ;negated
                  value of the tag assigned to START.
   set MOTORE     ;the tag assigned to MOTOR is set to 1
                  ;only if the accumulator is still 1.
endblo
```



2.4.5 SET Instruction

Syntax:

SET tag

LADDER Symbol

Tag

–( )–

Description:

Tag ← 1.

The *tag* of the IMAGE MEMORY specified is set to the logic value 1. The operation is conditioned by the state of the accumulator. That is, it will be performed only if the accumulator is equal to 1.

SET instructions can be placed anywhere in the program.

Operands:

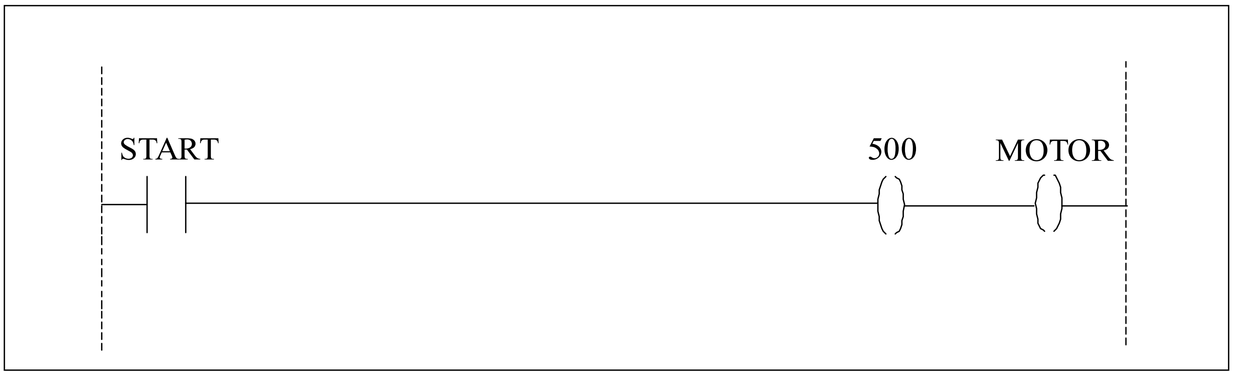
OC OD MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)     ;marker.
MOTORE   assign 1(m)     ;marker.
...
...
andSTART          ;logic AND between the accumulator and the
                  ;value of the tag assigned to START.
   set 500        ;tag number 500 in the memory set to logic 1.
   set MOTOR      ;the tag assigned to MOTOR set to logic 1.
endblo
```



2.4.6 CLEAR Instruction

LADDER Symbol

Tag

–(/)–

Syntax:

CLEAR tag

Description:

Tag ← 0.
The *tag* of the IMAGE MEMORY specified is set to the logic value 0. The operation is conditioned by the state of the accumulator. That is, it will be performed only if the accumulator is equal to 1.
CLEAR instructions can be placed anywhere in the program.

Operands:

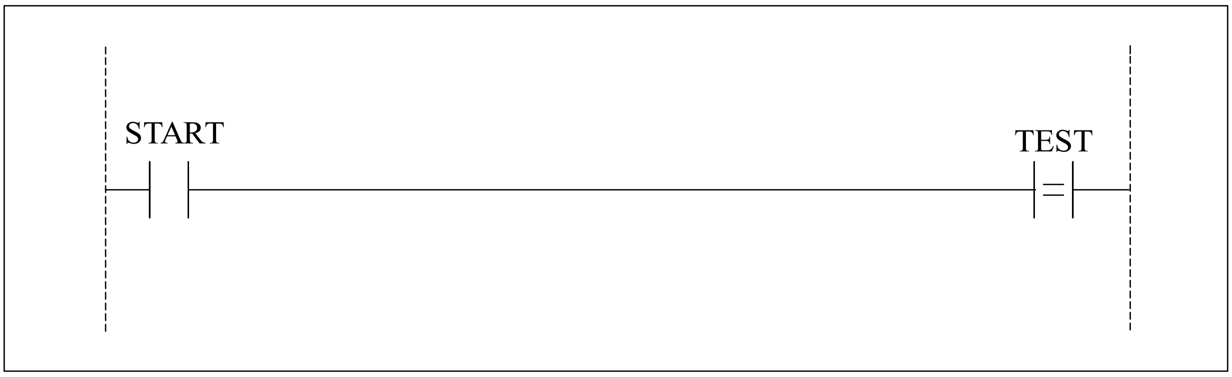
OC OD MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)     ;marker.
MOTORE   assign 1(m)     ;marker.
...
...
andSTART            ;logic AND between the accumulator and the ;value
                    of the tag assigned to START
  clear 500         ;tag number 500 in the memory set to logic 0.
  clear MOTOR       ;the tag assigned to MOTOR set to logic 0.
endblo
```



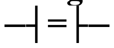
2.4.7 EQU Instruction

Syntax:

EQU tag

LADDER Symbol

Tag



Description:

Tag ← Acc.
The *tag* of the IMAGE MEMORY specified is placed at the logic value of the accumulator.
EQU instructions can be placed anywhere in the program and will be performed regardless of the state of the accumulator.

Operands:

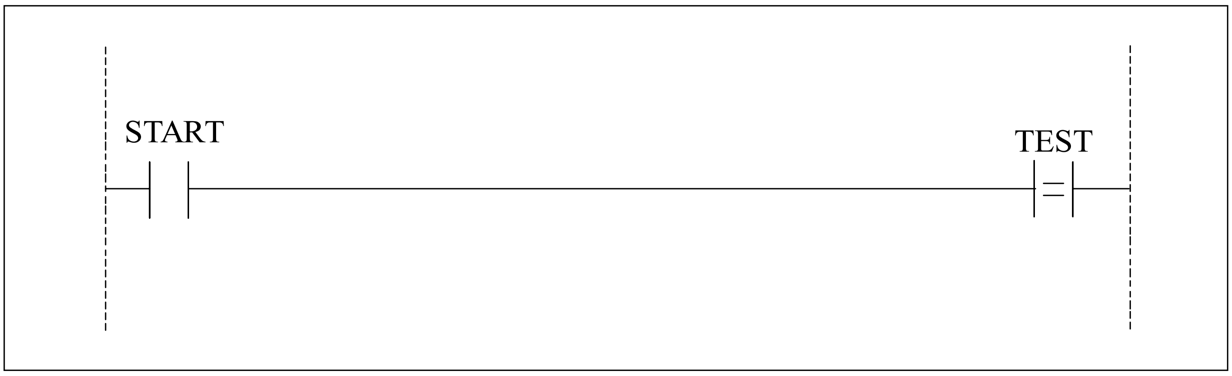
OC OD MK SQ FL MA CN TM (table A page 14)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)      ;marker.
TEST     assign 1(m)      ;marker.
...
...
and START      ;logic AND between the acc. and the tag assigned to
               ;START, the result is placed in the accumulator.
equ TEST       ;the tag assigned to TEST is set to the value of ;the
               accumulator.

endblo
```



2.4.8 EQUNOT Instruction

Syntax:

EQUNOT tag

LADDER Symbol

Tag

─┤<>├─

Description:

Tag ← NOT Acc.

The *tag* of the IMAGE memory specified is set to the *negated* logic value of the accumulator .

EQUNOT instructions can be placed anywhere in the program and are executed regardless of the state of the accumulator.

Operands:

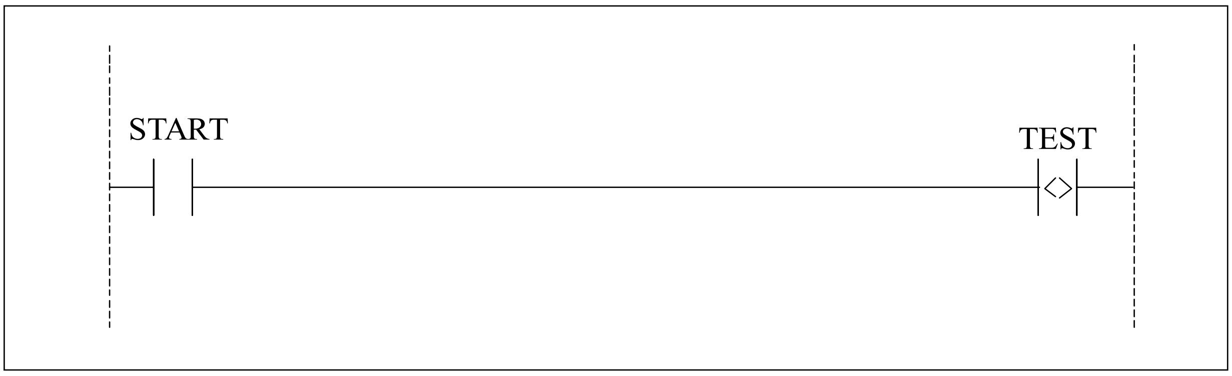
OC OD MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)      ;marker.
TEST     assign 1(m)      ;marker.
...
...
and START      ;logic AND between the acc. and the tag assigned ;to
               START and the result placed in the accumulator.
equnot TEST    ;the tag assigned to TEST is set to the negated ;value of
               the accumulator.
endblo
```

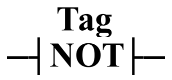


2.4.9 CHANGE Instruction

Syntax:

CHANGE tag

LADDER Symbol



Description:

Tag ← NOT tag.
The *tag* of the IMAGE MEMORY specified is set to the *negated* value of itself.
CHANGE instructions can be placed anywhere in the program and are executed only if the state of the accumulator is equal to 1.

Operands:

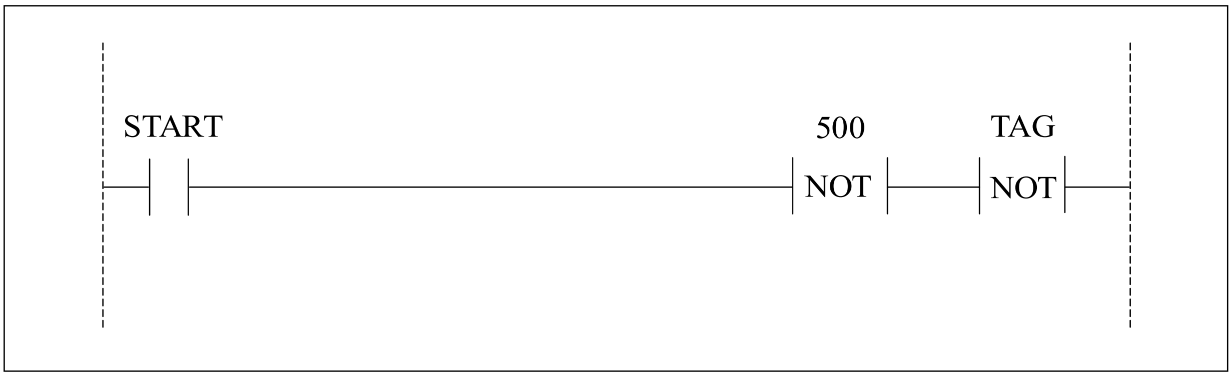
OC OD MK SQ FL MA CN TM (table A page 13)

Notes:

The number of the tag can be any tag in the image memory.

Example:

```
START    assign 0(m)      ;marker.
TAG      assign 0(m)      ;marker.
...
and      START         ;logic AND between the acc. and the tag assigned to
                       ;START, the result is placed in the acc.
change   500           ;the tag number 500 of the memory is set to the
                       ;negated value of itself.
Change   TAG           ;TAG is set to the negated value of itself.

endblo
```



2.4.10 SETTIM Instruction

Syntax:

SETTIM timer, var

LADDER Symbol

Timer

–(TM)–

Description:

The *timer* associated is loaded with the value contained in the variable *var* (1≤value≤65535). The number of timers that can be activated at any one time is 32, and the SETTIM instructions can be placed anywhere in the program and are executed only if the state of the accumulator is equal to 1. There are two different types of timer: fast timers (16) and slow timers (16). The value of the fast timers is stepped every 10ms and that of the slow timers every 100ms. The fast timers are identified by the memory addresses between 416 and 431 [0(t)..15(t)], while the slow ones are between 432 and 447 [16(t)...31(t)].

Operands:

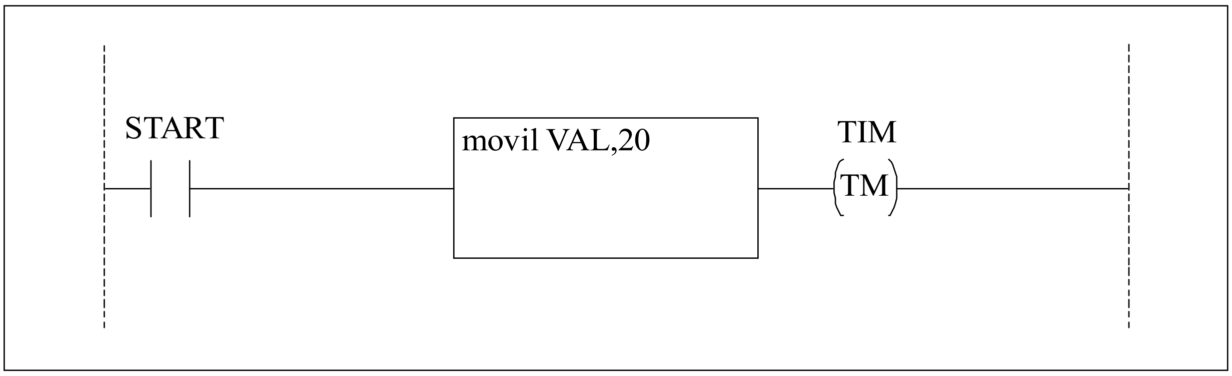
TM (table A page 13)

Notes:

The maximum time that mar be associated with a fast timer is approximately 655 seconds (over 10 minutes) while the slow timers reach 6553 seconds (more that 1h49m). The value of time that may be associated with the fast timers must be expressed in hundredths of a second, and for the slow timers in tenths of a second.

Example:

```
TIMER1    assign 0(t)      ;fast timer (10 ms).
VAL       assign 128       ;intermediate variable.
START     assign 0(m)      ;marker.
...
and       START            ;logic AND between the acc. and    the tag
                           assigned ;to START, the result is  placed in
                           the acc.
movil     VAL,20           ;initialisation of VAL to 20, i.e. 200 ms.
settim    TIMER1,VAL  ;the value of VAL is loaded into    TIMER1.
endblo
```



2.4.11 ENTIM Instruction

Syntax:

ENTIM timer

LADDER Symbol

Timer

─┫TM┣─

Description:

The associated *timer* is enabled for count. When the timer reaches the value 0, the tag associated (timer) is set to 1. The ENTIM instructions can be used anywhere in the program and are executed only if the state of the accumulator is equal to 1.

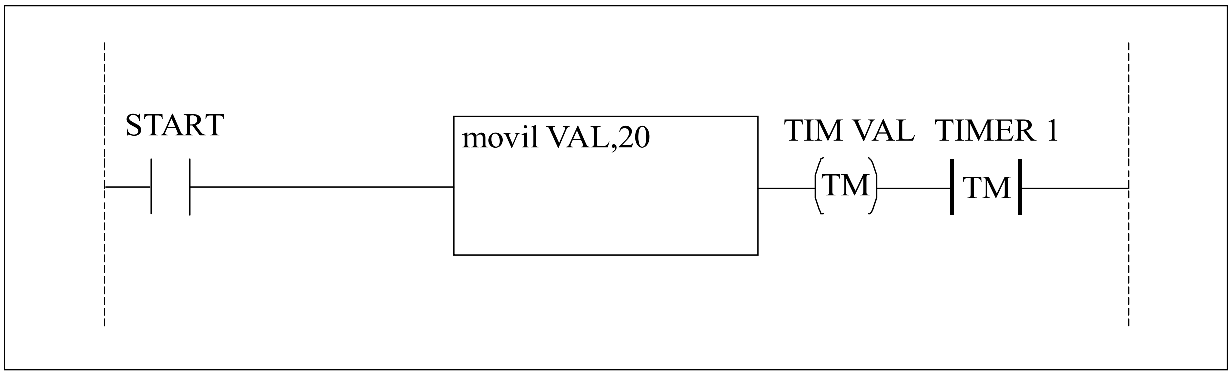
Operands:

TM (table A page 13)

Notes:

Example:

```
TIMER1   assign 0(t)    ;fast timer.
VAL      assign 128     ;intermediate variable.
START    assign 0(m)    ;marker.
...
and       START          ;logic AND between the acc. and the tag
                         assigned ;to START, the result is placed
                         in the acc.
movil    VAL,20         ;initialisation VAL to 20.
settim   TIMER1,VAL     ;the value of VAL is loaded into TIMER1.
entim    TIMER1        ;Count enabled for TIMER1.
...
endblo
```



2.4.12 DEFCNT Instruction

Syntax:

DEFCNT count, tag

LADDER Symbol

Count

—| DCN |—

Description:

An input *tag* is associated to a counter *count*, and becomes the count input of the counter. In this way, it is possible to count the pulses (variations from 0 to 1) of the digital signal associated to any input of the system. There are two types of counter: fast counters (16) and slow counters (16). The first are updated every 2ms and the latter are updated every 10ms. The fast counters are identified with the tags of the image memory between 0(c) and 15(c), and the slow counters go from 16(c) to 31(c).

Operands:

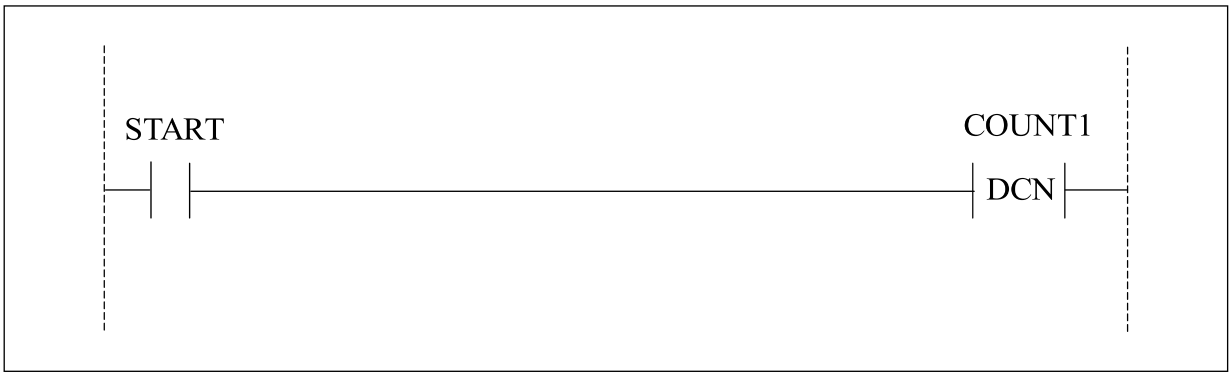
CN (table A page 13)

Notes:

The number of counters that can be activated contemporaneously is 32. The DEFCNT instructions can be put anywhere in the program and executed only if the state of the accumulator is equal to 1.

Example:

```
COUNT1   assign 0(c)     ;fast counter(2 ms).
START    assign 0(m)     ;marker.
...
and   START         ;logic AND between the acc. and the tag associated
                    with START, the result is placed in the acc.
defcnt  COUNT1, 1:0(1)     ;COUNT1 is associated to input 1.
...
endblo
```



2.4.13 SETCNT Instruction

Syntax:

SETCNT count, val

LADDER Symbol

Val

—| SCN |—

Description:

The counter is loaded with the passed value *val* (val < 32767). If the counter is enabled, when it reaches the value 0, the tag associated with the counter (count) is set to 1.

Operands:

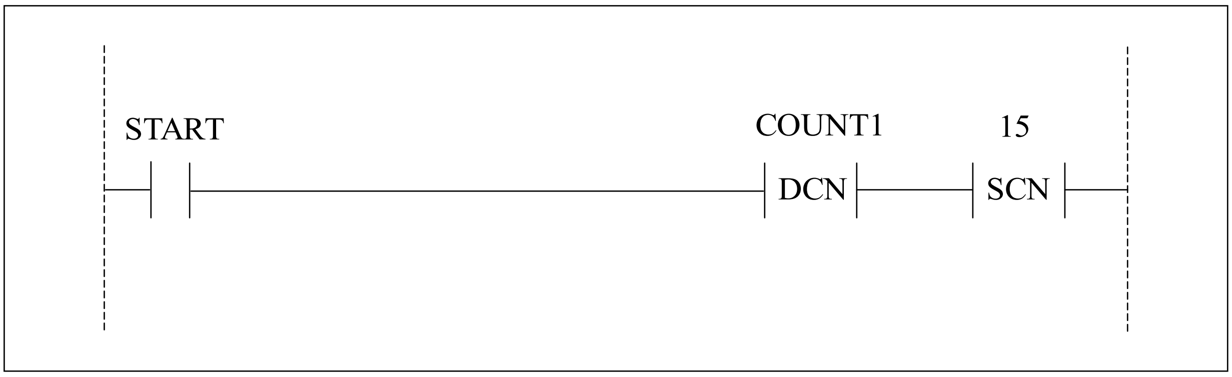
CN (Table A page 13)

Notes:

SETCNT instructions can be used anywhere in the program and are executed only if the state of the accumulator is equal to 1.

Example:

```
COUNT1 assign 0(c) ;fast counter(2 ms).
START  assign 0(m) ;marker.
...
and START   ;logic  AND  between  the  acc.  and  the  tag  assigned
            ;to START, the result is put in the acc.
defcnt COUNT1,1:0(1)    ;input 1 is associated with COUNT1.
setcnt COUNT1, 15       ;COUNT1 is initialised at 15.
...
endblo
```



2.4.14 ENCNT Instruction

Syntax:

ENCNT count

LADDER Symbol

Count
┤ECN├

Description:

The counter associated with the tag (*count*) is enabled to count.

Operands:

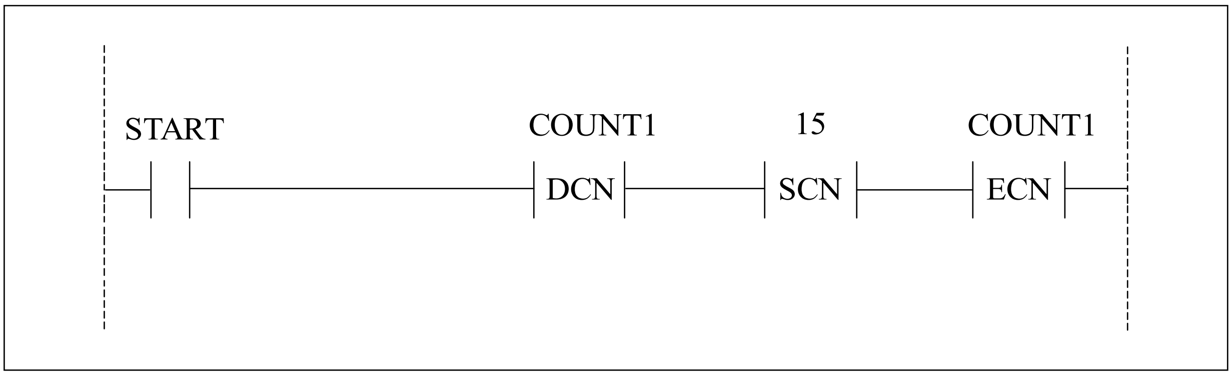
CN (table page 13)

Notes:

ENCNT instructions can be used anywhere in the program and they are executed only if the state of the accumulator is equal to 1.

Example:

```
COUNT1   assign 0(c)      ;fast counter (2 ms).
START    assign 0(m)      ;marker.
...
and START      ;logic AND between the acc. and the tag assigned
               ;to START, the result is placed in the acc.

defcnt   COUNT1, 1:0(1)   ;input 1 is associated with COUNT1.
setcnt   COUNT1, 15       ;COUNT1 is initialised at 15.
encnt    COUNT1           ;CUONT1 is enabled to count.
...
endblo
```

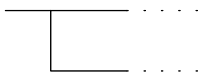


2.4.15 APAR Instruction

Syntax:

APAR

LADDER Symbol



Description :

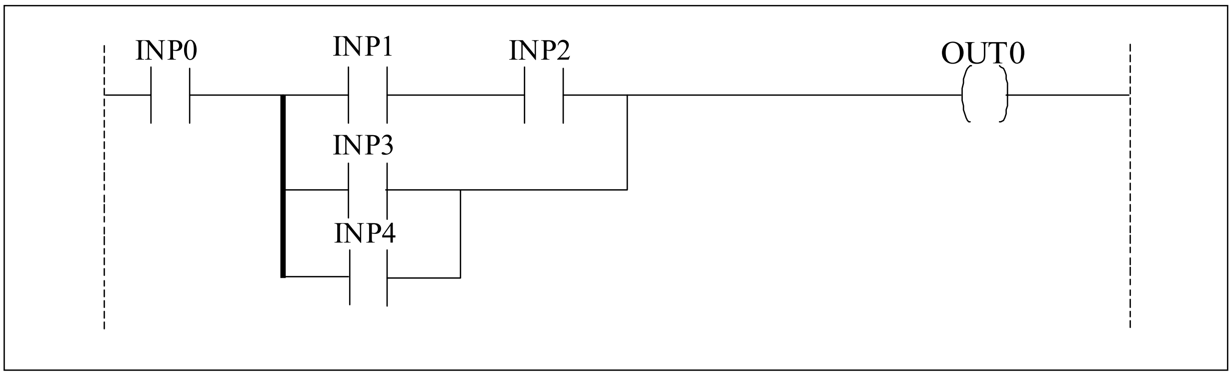
Open brackets (parentheses).

Notes:

The parallel network handling instructions enable the translation of complex networks of the series-parallel type. Brackets can be opened inside others, provided that there are no more than 32.

Example:

```
INP0    assign 0(i)
INP1    assign 1(i)
INP2    assign 2(i)
INP3    assign 3(i)
INP4    assign 4(i)
OUT0    assign 0(o)

andINP0
apar
andINP1
andINP2
chiram
andINP3
chiram
andINP4
chipar
setOUT0
endblo
```

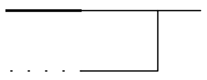


2.4.16 CHIPAR Instruction

Syntax:

CHIPAR

LADDER Symbol



Description:

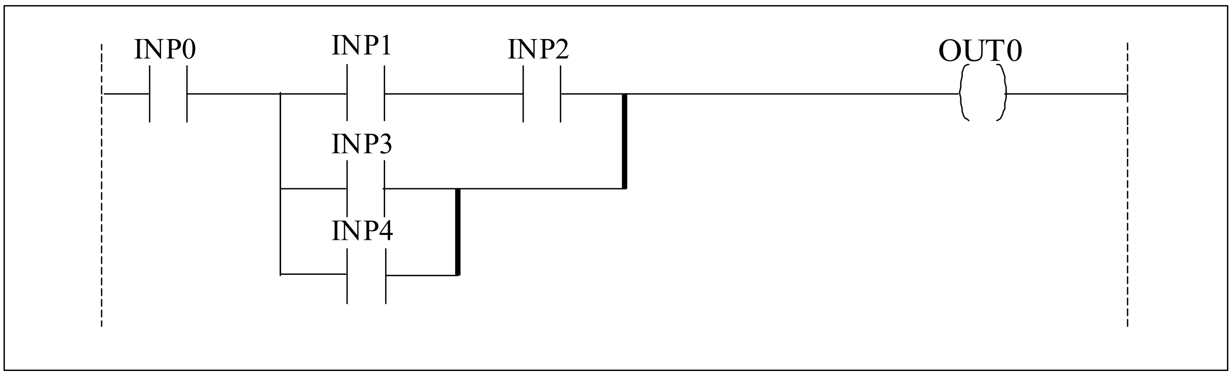
Close brackets (parentheses).

Notes:

The parallel network handling instructions enable the translation of complex networks of the series-parallel type. All the open brackets must be closed before the *endblo* instruction is reached.

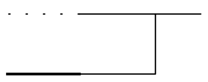
Example:

```
INP0    assign 0(i)
INP1    assign 1(i)
INP2    assign 2(i)
INP3    assign 3(i)
INP4    assign 4(i)
OUT0    assign 0(o)

andINP0
apar
andINP1
andINP2
chiram
andINP3
chiram
andINP4
chipar
setOUT0
endblo
```



2.4.17 CHIRAM Instruction

LADDER Symbol



Syntax:

CHIRAM

Description:

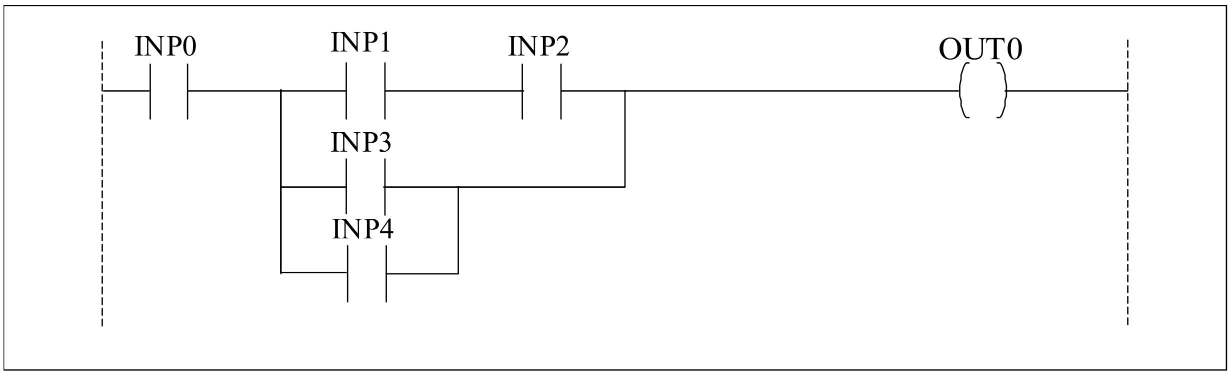
Close branch (maximum 32 branches in parallel).

Notes:

The parallel network handling instructions enable the translation of complex networks of the series-parallel type. Up to 32 parallel branches can be used. The CHIRAM instruction gives the possibility of executing a logic OR between several AND conditions.

Example:

```
INP0    assign 0(i)
INP1    assign 1(i)
INP2    assign 2(i)
INP3    assign 3(i)
INP4    assign 4(i)
OUT0    assign 0(o)

andINP0
apar
andINP1
andINP2
chiram
andINP3
chiram
andINP4
chipar
setOUT0
endblo
```



2.4.18 DIFUP Instruction

Syntax:

DIFUP tag

LADDER Symbol

Tag
─┤DF├─

Description:

The instruction enables checking whether the *tag* has changed from 0 to 1, If the variation has occurred, the state of the accumulator is set to 1. If it has not occurred, it is set to 0. DIFUP instructions can be used anywhere in the program and are executed only if the state of the accumulator is equal to 1.

Operands:

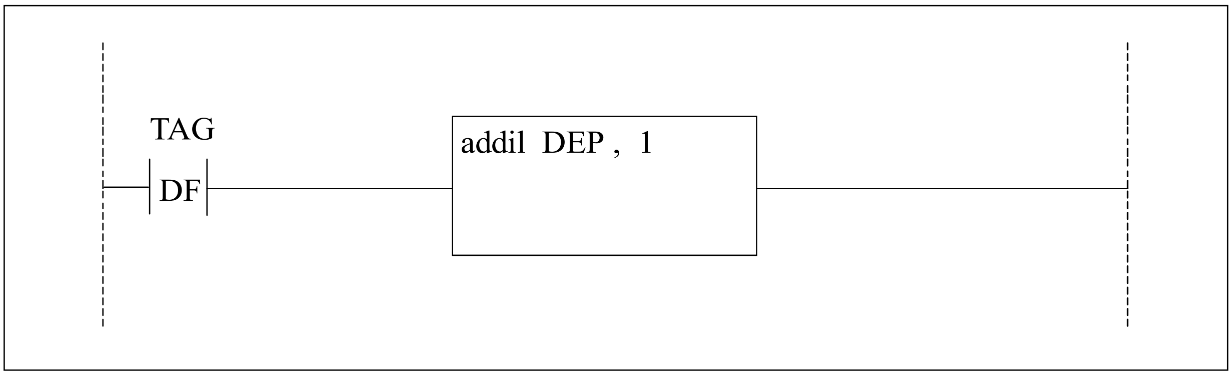
MK (table A page 13)

Notes:

DIFUP recognises whether a *tag* has changed from 0 to 1 with respect to the previous cycle, The tag examined must be one of the first 32 markers.

Example:

```
TAGassign 0(m)     ;marker.
DEP     assign 128 ;intermediate variable.
...
difup TAG          ;if tag has changed from 0 to 1, the
                   ;accumulator is set to 1.
addil DEP,1        ;sum 1 to DEP
endblo
```



2.4.19 DIFDN Instruction

Syntax:

DIFDN tag

LADDER Symbol

Tag
─┤DN├─

Description:

The instruction enables checking whether the *tag* has changed from 1 to 0, If the variation has occurred, the state of the accumulator is set to 1. If it has not occurred, it is set to 0. DIFDN instructions can be used anywhere in the program and are executed only if the state of the accumulator is equal to 1.

Operands:

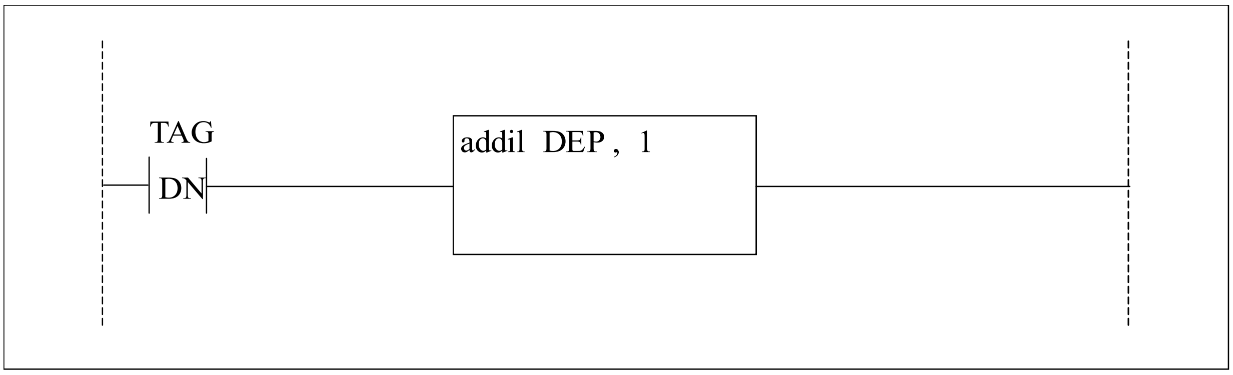
MK (table A page 13)

Notes:

DIFUP recognises whether a *tag* has changed from 0 to 1 with respect to the previous cycle. The tag examined must be one of the first 32 markers.

Example:

```
TAGassign 0(m)      ;marker.
DEP     assign 128 ;intermediate variable.

...
difdn TAG           ;if  the  tag  has  changed  from  1  to  0,  the
                    ;accumulator is set to 1.
addil DEP,1        ;sum 1 to DEP
endblo
```



2.4.20 SEQ Instruction

Syntax:

SEQ number

LADDER Symbol

Number
─┤SEQ├─

Description:

Beginning of a sequence. Identifies a block of instructions terminating at the next SEQ instruction, that is executed only if the tag *number(S)*, in the image memory, equals 1. In this way, it is possible to activate a portion of the PLC program as a function of the evolution of the process, with the advantage of increased program execution speed as the sequence that is disabled is skipped completely. It is not essential to use sequences. If none are employed, the program is a conventional PLC program.

Operands:

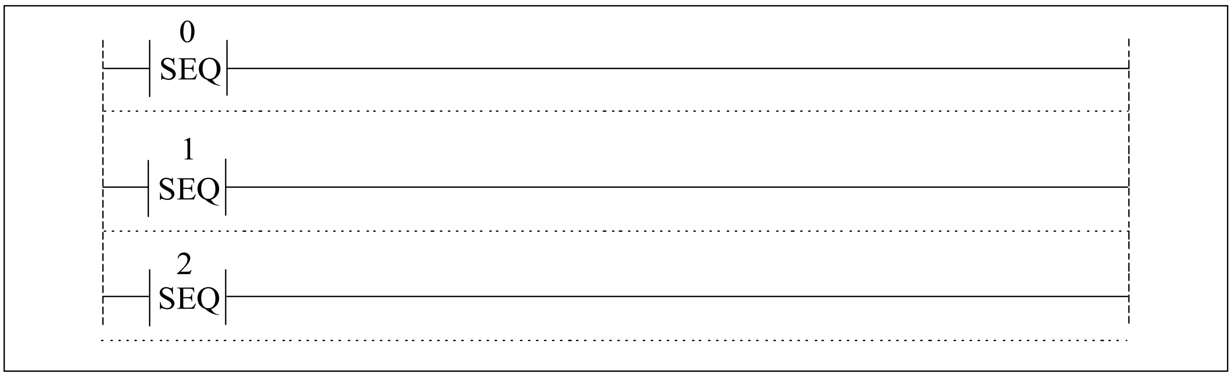
SQ (table A page 13)

Notes:

62 sequences are available. Sequence 0 is activated once only on power up.

Example:

```
seq 0    ;begin sequence number 0.
   set 1(S)   ;enable sequence number 1.
   set 2(S)   ;enable sequence number 2.
...
seq 1         ;begin sequence number 1.
...           ;body of sequence number 1.
...
seq 2         ;begin sequence number 2.
...           ;body of sequence number 2.
```



2.4.21 JUMP Instruction

Syntax:

JUMP tag, label

LADDER Symbol

Label
─┤JUMP├─

Description:

Instruction to skip program. If *tag* equals 1, the program executes the instruction found at *label*. Otherwise it executes the following instruction. *Tag* can be any tag in the image memory. Some event markers with special significance are available to the programmer and these are useful for controlling the flow of the program following operations on the registers (see instructions on registers)

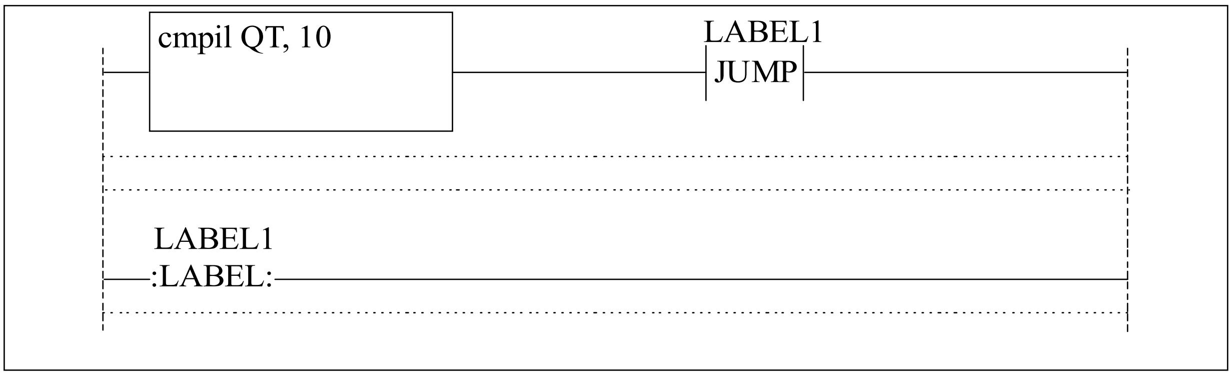
Operands:

Notes:

There is a tag [248(m)] whose value is fixed at 1 that enables the program to be skipped always (unconditioned JUMP).

Example:

```
EQ      assign 250(m)   ;flag of equality.
QT      assign 128 ;quantity variable.
...
cmpil   QT,10      ;if QT=10 then EQ is set to 1.
jump    EQ,LABEL1       ;if EQ is 1 skip to LABEL1.
...
...
...
LABEL1                  ;code to execute if QT=10.
...
...
endblo
```



2.4.22 CALL Instruction

Syntax:

CALL label

LADDER Symbol

Label
─┤CALL├─

Description:

Call a subroutine.
The program executes the instruction found at *label* and returns to the instruction that follows the call when it encounters the ENDSUB instruction. Subroutines must be placed after an ENDPRO instruction. They must be identified by a label and must terminate with the ENDSUB instruction.
CALL instructions can be used anywhere in the program and they are executed only if the state of the accumulator is equal to 1.

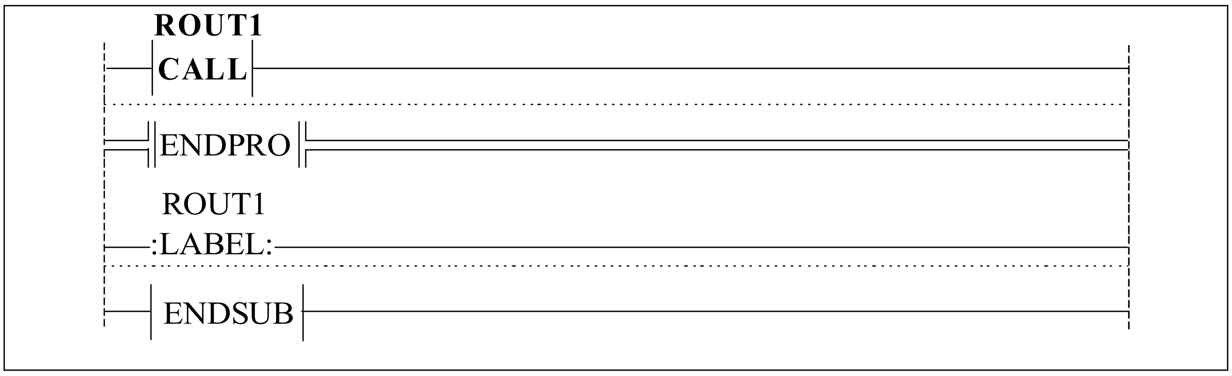
Operands:

Notes:

It is also possible to call a subroutine from another subroutine, provided that this operation is not repeated for more than 8 subroutines one inside the other.

Example:

```
....
call ROUTINE1 ;if acc=1 the subroutine is called.
....
endpro

ROUTINE1
...
...          ;body of the subroutine
...
endsub
```



2.4.23 ENDSUB Instruction

Syntax:

ENDSUB

LADDER Symbol

─┤ **ENDSUB** ├─

Description:

End subroutine.
The program returns to the instruction following the call of the subroutine being executed.

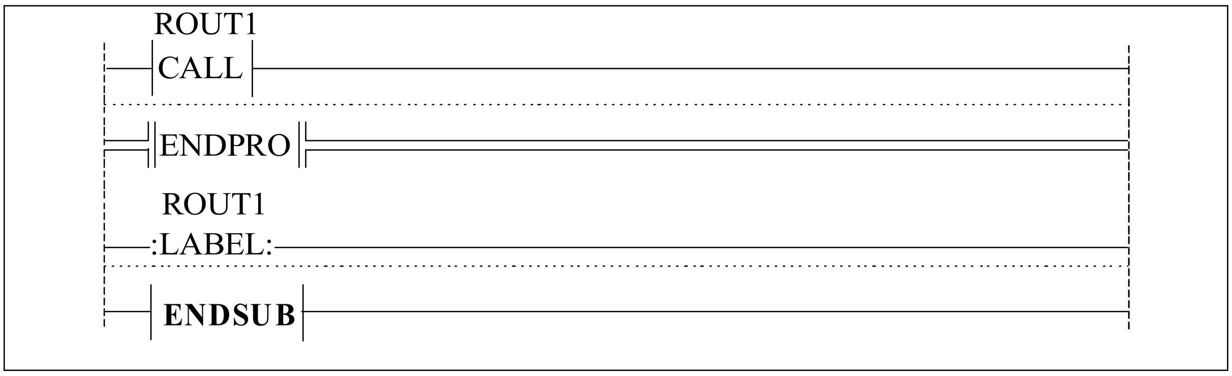
Operands:

Notes:

A subroutine must always terminate with the ENDSUB instruction.

Example:

```
....
....
call ROUTINE1 ;if acc=1 the subroutine is called.
....
....
endpro

ROUTINE1
...
...           ;body of subroutine
...
endsub
```



2.4.24 Definition of a label

Syntax:

label

LADDER Symbol

Label

–: LABEL :–

Description:

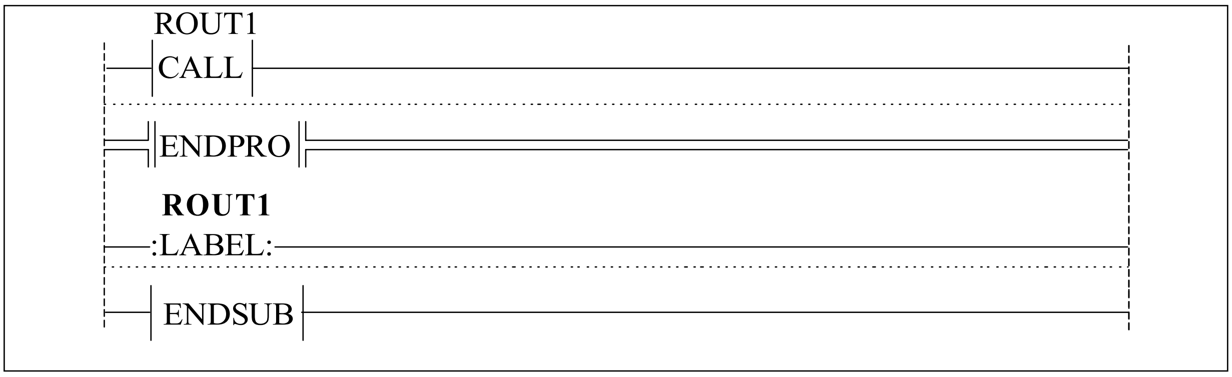
A *label* is defined, to which other instructions, like JUMP or CALL, can make reference.

Operands:

Notes:

Example:

```
....
....
call ROUTINE1 ;if acc=1 the subroutine is called.
....
....
endpro

ROUTINE1      ;label definition of routine1
...
...           ;body of subroutine
...
endsub
```



2.4.25 ENDPRO Instruction

Syntax:

ENDPRO

LADDER Symbol

═╢ENDPRO╟═

Description:

End of program. Closes the PLC program and has the purpose of advising the interpreter to restart the execution of the program from the beginning.

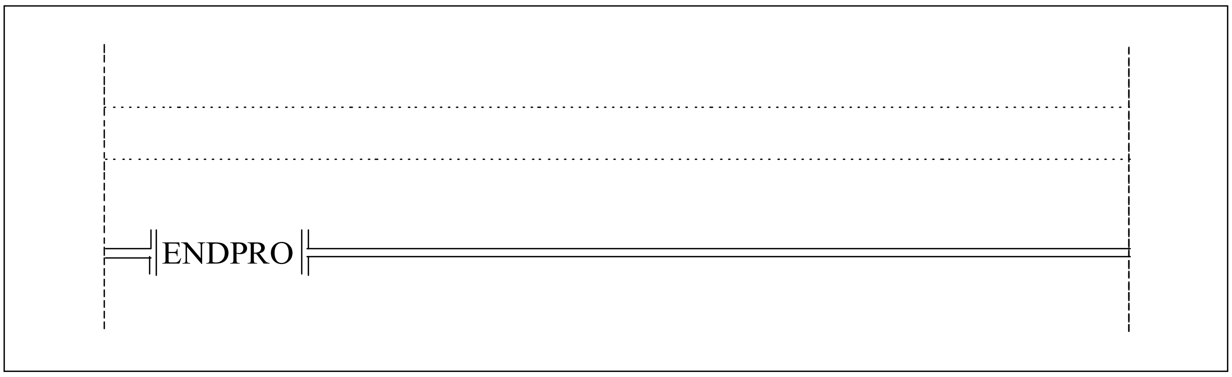
Operands:

Notes:

In order to synchronise the PLC with the CNC, without possibility of error, the execution of this instruction is accompanied by the automatic reading of the tags and the variables that the PLC receives from the CNC.

Example:

```
...
...      ;program body.
...
endpro         ;end program.
```



2.4.26 FAL Instruction

Syntax:

FAL number

LADDER Symbol

Number
─┤FAL├─

Description:

Signalling of anomalies or warnings.
The number must be less than 100, and, in particular, if, se:

- number = 0, then the reset of the fifo message is obtained
- 0 < number < 50, warning
- number ≥ 50, alarm and wait for reset.

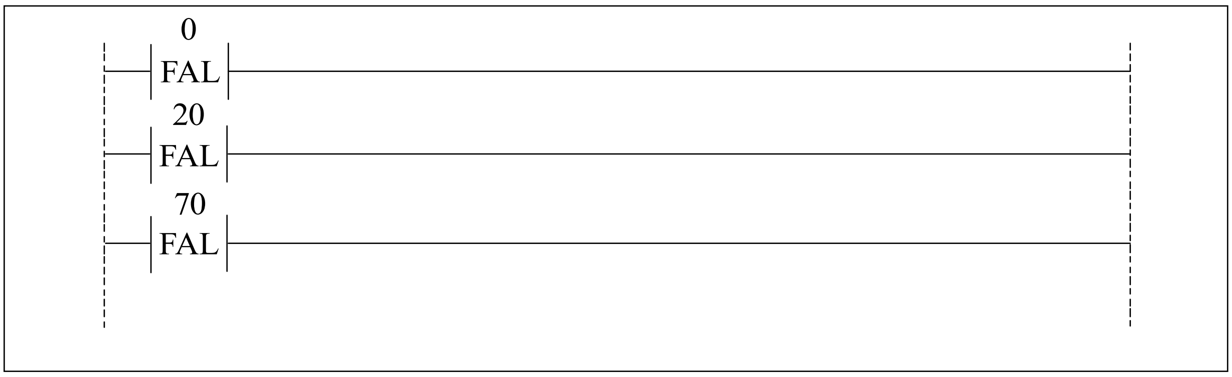
In the event of a warning or anomaly, a BEL is sent on the serial line (can be disabled with the OUTVAR instruction).

Operands:

Notes:

Example:

```
...
...
fal 0         ;fifo messages reset.
fal 20        ;warning.
fal 70        ;alarm, wait for reset.
...
...
```



2.4.27 FALS Instruction

LADDER Symbol

Number
─┤FALS├─

Syntax:

FALS number

Description:

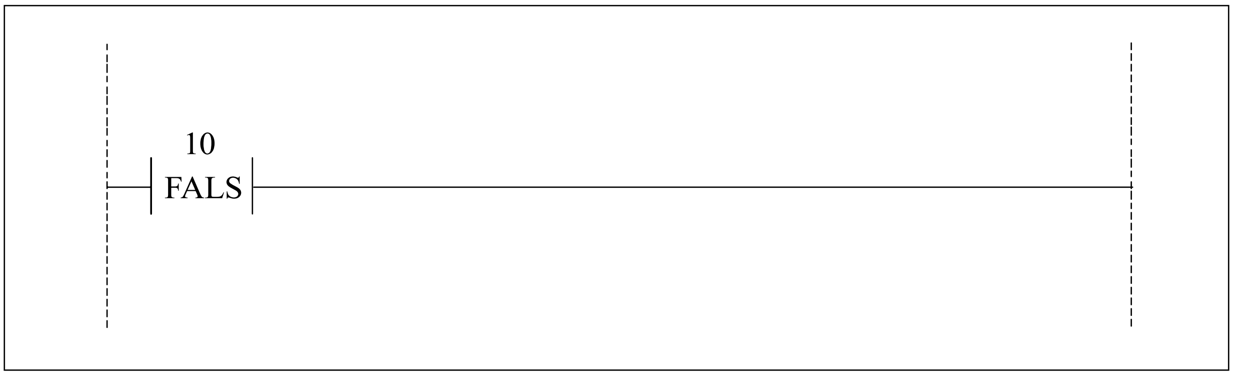
Signalling of a fatal alarm.
The number must be less than 100. The PLC and the CNC are stopped. To exit, it is necessary to switch of the machine. In time, a BEL is sent on the serial line (may be disabled with the OUTVAR instruction).

Operands:

Notes:

Example:

```
...
...
fals 10       ;fatal alarm.
...
...
```



2.4.28 MSG Instruction

Syntax:

MSG "message"

LADDER Symbol

—┤**MSG**├—

Description:

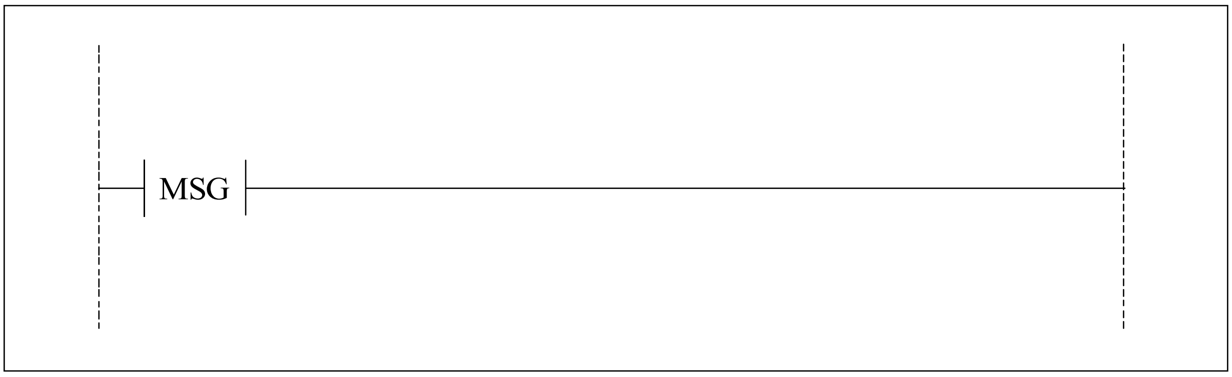
The message (with a maximum of 16 characters) is put in fifo (maximum of 10 messages).
On this request (^R on the serial line), the PLC transmits the message (associating this to a FAL or FALS instruction, it can generate a BEL):

Operands:

Notes:

Example:

```
msg "NO PRESSURE"  ;The string is put into the fifo.
```



2.4.29 INPVAR Instruction

Syntax:

INPVAR dest, sour

LADDER Symbol

—| INPVAR |—

Description:

The contents of *sour* are copied to *dest*. *dest* must be a variable, while *sour* can be an address of a device with the following generic structure:

Par1:Par2 (Disp)

- CNC (Device 2)
  - 0:0(2) read CNC error flag
- Panel (Device 3)
  - 0:0(3) read panel image (variable 14 for the CNC)
- Analogue inputs (Device 4)
  - 0:0(4) reading analogue input from logic channel 0 =>AIN0
  - 1:0(4) reading analogue input from logic channel 0 =>AIN1
  - 2:0(4) reading analogue input from logic channel 0 => AIN2
  - 5:0(4) reading analogue input from logic channel 0 => CURRENT
  - 6:0(4) reading analogue input from logic channel 0 => HAIN

  - 0:1(4) reading analogue input from logic channel 1 =>AIN0
  - 1:1(4) reading analogue input from logic channel 1 =>AIN1
  - 2:1(4) reading analogue input from logic channel 1 => AIN2
  - 5:1(4) reading analogue input from logic channel 1 => CURRENT
  - 6:1(4) reading analogue input from logic channel 1 => HAIN
- Variables from the CNC (Device 5)

– X:1(5)	reading Q1 from CNC	(If X=0 Long, if X=1 Word)
– X:2(5)	reading Q2 from CNC	(If X=0 Long, if X=1 Word)
– ...		
– X:255(5)	reading Q255 from CNC	(If X=0 Long, if X=1 Word)

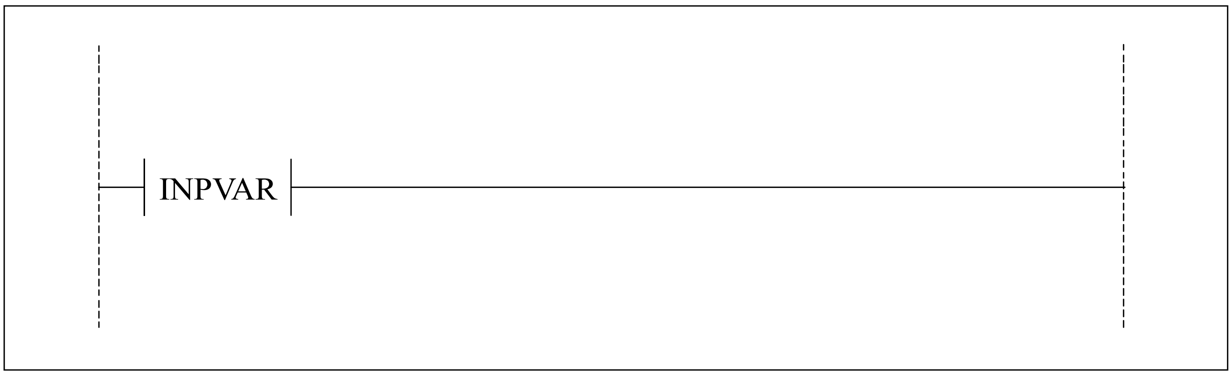
- Timers and Counters (Device 6)
  - 0:0(6) timer 0 reading
  - 0:1(6) timer 1 reading
  - ...
  - 0:31(6) timer 31 reading
  - 1:0(6) counter 0 reading
  - 1:1(6) counter 1 reading
  - ...
  - 1:31(6) counter 31 reading

Operands:

Notes:

Example:

```
QT assign 128 ;declaration of the variable QT.

...
...
inpvar  QT,0:80(5) ;reading of variable Q80 from the CNC.
...
...
```



2.4.30 OUTVAR Instruction

Syntax:

OUTVAR dest, sour

LADDER Symbol

─┤OUTVAR├-

Description:

The contents of *sour* are copied to *dest*. *Sour* must be a variable, while *dest* can be an address with the following generic structure:

Par1:Par2 (1)

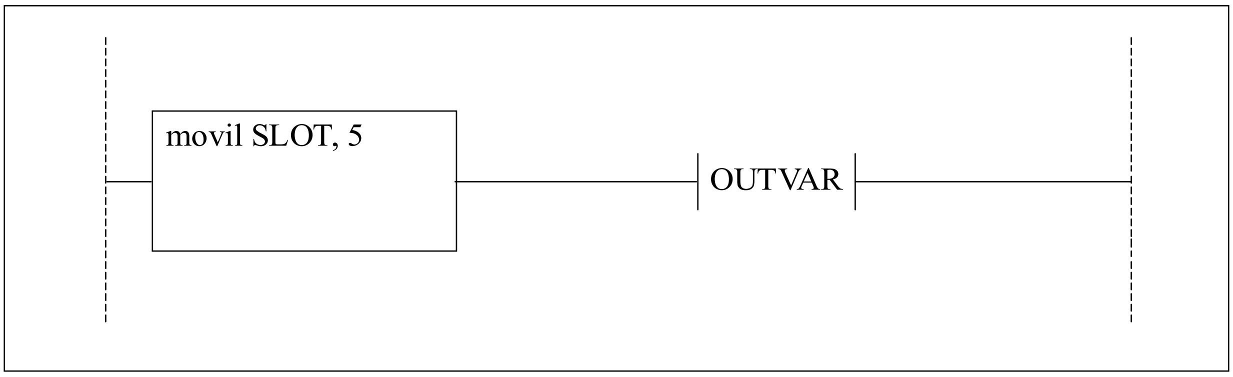
- 0:0(1) enable/disable filter on the inputs (0 disabled)
- 1:0(1) CNC time in 2 ms slot.
- 2:0(1) PLC time in 2 ms slot.
- 3:0(1) time between BEL's in seconds.
- 4:0(1) reset PLC error.
- 5:0(1) writing to analogue output of logic channel 0.
- 5:1(1) writing to analogue output of logic channel 1.
- 6:0(1) instruction equivalent to ENDPRO. Updates the PLC-CNC exchange bits, updates the timer bits and the counter bits, updates the acknowledge bits for the M and T commands.
- 7:0(1) initialisation of record length 0 < lenRec <90 (logic channel operations).

Operands:

Notes:

Example:

```
SLOT     assign 128 ;declaration of the variable SLOT.
...
...
movil    SLOT,5          ;SLOT contains 5.
outvar   1:0(1),SLOT     ;time dedicated to CNC slot 5 = 10 ms.
...
...
```



2.5 Mathematical Block Instructions

LADDER Symbol

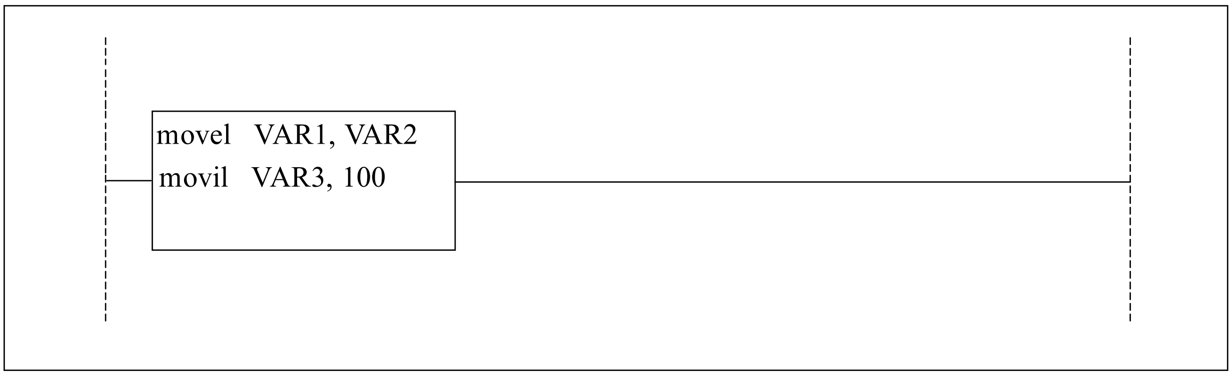
Description:

Inside a mathematical block, it is possible to insert a list of mathematical instructions. The available mathematical instructions are listed, beginning from paragraph 2.5.1.

Notes:

Example:

```
START         assign 0(m)     ;marker.
VAR1          assign 128     ;32 bit variable.
VAR2          assign 130     ;32 bit variable.
VAR3          assign 132     ;32 bit variable.
...
andSTART           ;if START is 1.
        movel VAR1,VAR2 ;VAR2 copied to VAR1.
        movil VAR3,100       ;VAR3 contains 100.
endblo
```



2.5.1 MOVE (MOVE, MOVEL, MOVI, MOVIL) Instruction

Syntax:

MOVE var1,var2

Description:

The contents of *var2* are copied to *var1* only if the Acc. is equal to 1. The arithmetic markers remain unchanged.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	–	–	–	–	–	–

Acc:	unchanged.
Z:	unchanged.
NZ:	unchanged.
LT:	unchanged.
GT:	unchanged.
OVF:	unchanged.
NoAcc:	unchanged.

Notes:

The variants of the MOVE instruction have the following meanings:

• MOVE	var1, var2	var1 and var2 must be 16 bit variables.
• MOVEL	var1, var2	var1 and var2 must be 32 bit variables.
• MOVI	var1, num	var1 16 bit variable, num 16 bit integer.
• MOVIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

ID OD SQ FL CN TM MK MA IC OC VC VP VI VO VA (table A page 13)

Example:

```
START        assign 0(m)     ;marker.
VAR1         assign 128 ;variable 16 bit.
VAR2         assign 129 ;variable 16 bit.
VAR3         assign 130 ;variable 32 bit.
VAR4         assign 132 ;variable 32 bit.
...
andSTART                ;if START is 1.
        move     VAR1,VAR2       ;VAR2 copied to VAR1.
        movelVAR3,VAR4      ;VAR4 copied to VAR3.
        movi     VAR2,100        ;VAR2 contains 100.
        movilVAR4,100000    ;VAR4 contains 100000.
endblo
...
```

2.5.2 ADD (ADD, ADDL, ADDI, ADDIL) Instruction

Syntax:

ADD var1,var2

Description:

The contents of *var2* are summed to the contents of *var1,* only if Acc. is equal to 1, and the result is placed in *var1*. The arithmetic markers are modified in accordance with the outcome of the operation.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	*	*	*	*	*	–

Acc: unchanged.
Z: set to 1 if the result is zero, otherwise set to 0.
NZ: set to 1 if the result is different from zero, otherwise set to 0.
LT: set to 1 if *var1* < *var2*, otherwise set to 0.
GT: set to 1 if *var1* > *var2*, otherwise set to 0.
OVF: set to 1 if the operation has generated an overflow, otherwise set to 0.
NoAcc: unchanged.

Notes:

The variants of the ADD instruction have the following meanings:

- ADD var1, var2 var1 and var2 must be 16 bit variables.
- ADDL var1, var2 var1 and var2 must be 32 bit variables.
- ADDI var1, num var1 16 bit variable, num 16 bit integer.
- ADDIL var1, num var1 variable a 32 bit, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START        assign 0(m)     ;marker.
VAR1         assign 128 ;16 bit variable.
VAR2         assign 129 ;16 bit variable.
VAR3         assign 130 ;32 bit variable.
VAR4         assign 132 ;32 bit variable.
...
andSTART                ;if START é 1.
        add       VAR1,VAR2      ;VAR1 = VAR1 + VAR2.
        addl      VAR3,VAR4      ;VAR3 = VAR3 + VAR4.
        addi      VAR2,100       ;VAR2 = VAR2 + 100.
        addilVAR4,100000    ;VAR4 = VAR4 + 100000.
endblo
...
```

2.5.3 SUB (SUB, SUBL, SUBI, SUBIL) Instruction

Syntax:

SUB var1,var2

Description:

The contents of *var2* are subtracted from the contents of *var1*, only if Acc. is equal to 1, and the result is placed in *var1*. The arithmetic markers are modified according to the outcome of the operation.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	*	*	*	*	*	–

Acc:	unchanged.
Z:	set to 1 if the result is zero, otherwise set to 0.
NZ:	set to 1 if the result is different from zero, otherwise set to 0.
LT:	set to 1 if *var1* < *var2*, otherwise set to 0.
GT:	set to 1 if *var1* > *var2*, otherwise set to 0.
OVF:	set to 1 if the operation has generated an overflow, otherwise set to 0.
NoAcc:	unchanged.

Notes:

The variants of the SUB instruction have the following meanings:

• SUB	var1, var2	var1 and var2 must be 16 bit variables.
• SUBL	var1, var2	var1 and var2 must be 32 bit variables.
• SUBI	var1, num	var1 16 bit variable, num 16 bit integer.
• SUBIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START         assign 0(m)     ;marker.
VAR1          assign 128;16 bit variable.
VAR2          assign 129;16 bit variable.
VAR3          assign 130;32 bit variable.
VAR4          assign 132;32 bit variable.
...
andSTART                ;if START is 1.
        sub       VAR1,VAR2       ;VAR1 = VAR1 - VAR2.
        subl      VAR3,VAR4       ;VAR3 = VAR3 - VAR4.
        subi      VAR2,100        ;VAR2 = VAR2 - 100.
        subilVAR4,100000     ;VAR4 = VAR4 - 100000.
endblo
…
```

2.5.4 MUL (MUL, MULL, MULI, MULIL) Instruction

Syntax:

MUL var1,var2

Description:

The contents of *var1* are multiplied by the contents of *var2*, only if Acc. is equal to 1, and the result is placed in *var1*. The arithmetic markers are modified according to the outcome of the operation.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	*	*	*	*	*	–

Acc:	unchanged.
Z:	set to 1 if the result is zero, otherwise set to 0.
NZ:	set to 1 if the result is different to zero, otherwise set to 0.
LT:	set to 1 if *var1* < *var2*, otherwise set to 0.
GT:	set to 1 if *var1* > *var2*, otherwise set to 0.
OVF:	set to 1 if the operation has generated an overflow, otherwise set to 0.
NoAcc:	unchanged.

Notes:

The variants of the MUL instruction have the following meanings:

• MUL	var1, var2	var1 and var2 must be 16 bit variables.
• MULL	var1, var2	var1 and var2 must be 32 bit variables.
• MULI	var1, num	var1 16 bit variable, num 16 bit integer.
• MULIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table page 13)

Example:

```
START          assign 0(m)     ;marker.
VAR1           assign 128 ;16 bit variable.
VAR2           assign 129 ;16 bit variable.
VAR3           assign 130 ;32 bit variable.
VAR4           assign 132 ;32 bit variable.
...
andSTART                  ;if START is 1.
        mul       VAR1,VAR2       ;VAR1 = VAR1 * VAR2.
        mull      VAR3,VAR4       ;VAR3 = VAR3 * VAR4.
        muli      VAR2,100        ;VAR2 = VAR2 * 100.
        mulilVAR4,100000     ;VAR4 = VAR4 * 100000.
endblo
...
```

2.5.5 DIV (DIV, DIVL, DIVI, DIVIL) Instruction

Syntax:

DIV var1,var2

Description:

The contents of *var1* are divided by the contents of *var2*, only if Acc. is equal to 1, and the result is placed in *var1*. The arithmetic markers are modified according to the outcome of the operation.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	*	*	*	*	*	–

Acc:	unchanged.
Z:	set to 1 if the result is zero, otherwise set to 0.
NZ:	set to 1 if the result is different to zero, otherwise set to 0.
LT:	set to 1 if *var1* < *var2*, otherwise set to 0.
GT:	set to 1 if *var1* > *var2*, otherwise set to 0.
OVF:	set to 1 if the operation has generated an overflow, otherwise set to 0.
NoAcc:	unchanged.

Notes:

The variants of the DIV instruction have the following meaning:

• DIV	var1, var2	var1 and var2 must be 16 bit variables.
• DIVL	var1, var2	var1 and var2 must be 32 bit variables.
• DIVI	var1, num	var1 16 bit variable, num 16 bit integer.
• DIVIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START           assign 0(m)     ;marker.
VAR1            assign 128 ;16 bit variable.
VAR2            assign 129 ;16 bit variable.
VAR3            assign 130 ;32 bit variable.
VAR4            assign 132 ;32 bit variable.
...
andSTART                   ;if START is 1.
        div       VAR1,VAR2       ;VAR1 = VAR1 / VAR2.
        divl      VAR3,VAR4       ;VAR3 = VAR3 / VAR4.
        divi      VAR2,100        ;VAR2 = VAR2 / 100.
        divilVAR4,100000     ;VAR4 = VAR4 / 100000.
endblo
...
```

2.5.6 CMP (CMP, CMPL, CMPI, CMPIL) Instruction

Syntax:

CMP var1,var2

Description:

The contents of *var2* are subtracted from the contents of *var1*, only if Acc is equal to 1, and the arithmetic markers are modified according to the outcome of the operation.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	*	*	*	*	*	–

Acc:	unchanged.
Z:	set to 1 if the result is zero, otherwise set to 0.
NZ:	set to 1 if the result is different to zero, otherwise set to 0.
LT:	set to 1 if *var1* < *var2*, otherwise set to 0.
GT:	set to 1 if *var1* > *var2*, otherwise set to 0.
OVF:	set to 1 if the operation has generated an overflow, otherwise set to 0.
NoAcc:	unchanged.

Notes:

The variants of the CMP instruction have the following meanings:

• CMP	var1, var2	var1 and var2 must be 16 bit variables.
• CMPL	var1, var2	var1 and var2 must be 32 bit variables.
• CMPI	var1, num	var1 16 bit variable, num 16 bit integer.
• CMPIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START         assign 0(m)     ;marker.
VAR1          assign 128;16 bit variable.
VAR2          assign 129;16 bit variable.
VAR3          assign 130;32 bit variable.
VAR4          assign 132;32 bit variable.
...
andSTART                     ;if START is 1.
        cmp       VAR1,VAR2        ;(VAR1 == VAR2).
        cmpi      VAR2,100         ;(VAR2 == 100).
        cmpl      VAR3,VAR4        ;(VAR3 == VAR4).

        cmpilVAR4,100000      ;(VAR4 == 100000).
endblo
...
```

2.5.7

2.5.8 SHIFT (SHIFT, SHIFTL) Instruction

Syntax:

SHIFT var1

Description:

A shift of one bit to the left is made, and the first bit becomes the state of the accumulator. The accumulator is set to the value of the left bit of the variable. The operation is executed regardless of the state of the accumulator. The arithmetic markers remain unchanged.

Acc	Z (EQ)	NZ (NEQ)	LT	GT	OVF	NoAcc
–	–	–	–	–	–	–

Acc:	unchanged.
Z:	unchanged.
NZ:	unchanged.
LT:	unchanged.
GT:	unchanged.
OVF:	unchanged.
NoAcc:	unchanged.

Notes:

The variants of the SHIFT instruction have the following meanings:

- SHIFT var1 var1 must be a 16 bit variable.
- SHIFTL var1 var1 must be a 32 bit variable.

Operands:

VA (table A page 13)

Example:

```
START          assign 0(m)      ;marker.
VAR1           assign 128      ;16 bit variable.
VAR2           assign 129      ;16 bit variable.
VAR3           assign 130      ;32 bit variable.
VAR4           assign 132      ;32 bit variable.
...
and START                      ;if START is 1.
         shift      VAR1       ;shift of one bit to the left of VAR1.
         shiftl     VAR3       ;shift of one bit to the left of VAR3.
endblo
...
```

2.5.9 VAND (VAND, VANDL, VANDI, VANDIL) Instruction

Syntax:

VAND var1,*var2*

Description:

A logic AND operation is performed bit by bit between the variables *var1* e *var2*, and the results are placed in *var1*.

Notes:

The variants of the VAND instruction have the following meanings:

• VAND	var1, var2	var1 and var2 must be 16 bit variables.
• VANDL	var1, var2	var1 and var2 must be 32 bit variables.
• VANDI	var1, num	var1 16 bit variable, num 16 bit integer.
• VANDIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START           assign 0(m)     ;marker.
VAR1            assign 128 ;16 bit variable.
VAR2            assign 129 ;16 bit variable.
VAR3            assign 130 ;32 bit variable.
VAR4            assign 132 ;32 bit variable.
...
andSTART                  ;if START is 1.
        vand      VAR1,VAR2       ;(VAR1 = VAR1 & VAR2).
        vandl VAR3,VAR4      ;(VAR3 = VAR3 & VAR4).
        vandi VAR2,100       ;(VAR2 = VAR2 & 100).
        vandil    VAR4,100000     ;(VAR4 = VAR4 & 100000).
endblo
```

2.5.10 VOR (VOR, VORL, VORI, VORIL) Instruction

Syntax:

VOR var1,*var2*

Description:

A logic OR operation is performed bit by bit between the variables *var1* e *var2*, and the results are placed in *var1*.

Notes:

The variants of the VOR instruction have the following meanings:

• VOR	var1, var2	var1 and var2 must be 16 bit variables.
• VORL	var1, var2	var1 and var2 must be 32 bit variables.
• VORI	var1, num	var1 16 bit variable, num 16 bit integer.
• VORIL	var1, num	var1 32 bit variable, num 32 bit integer.

Operands:

VA (table A page 13)

Example:

```
START        assign 0(m)     ;marker.
VAR1         assign 128     ;16 bit variable.
VAR2         assign 129     ;16 bit variable.
VAR3         assign 130     ;32 bit variable.
VAR4         assign 132     ;32 bit variable.
...
andSTART                 ;if START is 1.
        vor       VAR1,VAR2       ;(VAR1 = VAR1 | VAR2).
        vorl      VAR3,VAR4       ;(VAR3 = VAR3 | VAR4).
        vori      VAR2,100        ;(VAR2 = VAR2 | 100).
        vorilVAR4,100000     ;(VAR4 = VAR4 | 100000).
endblo
```

2.6 Communication Block Instructions

LADDER Symbol

Description:

Inside a communication block, it is possible to insert a list of instructions for communications. These are described below from paragraph 2.6.1, and their functionality is illustrated in the following.

The logic channel handling instructions enable the system to handle the storing and saving of data as well as giving the possibility of interfacing with external devices.

In particular, using these instructions makes the following functions available:

- Positioning inside a zone of memory dedicated to the values of the variables, of the system date and time. The structure of this archive is decided by the programmer, who must ensure that the end of the block is defined by using the FINREC instruction. This enables the archive to be structured inside a record of variable length. A host computer can read the data contained in the archive over a serial connection. The use of a dedicated protocol guarantees the integrity of the transmitted data
- Writing interactive programs with the possibility of interfacing with a terminal connected serially to read and write the values of variables, characters and strings.
- Conversion of formats that enables the binary value of a variable to be transformed into an ASCII string that represents it, and viceversa. The ASCII strings that represent the result of the conversion are held in internal buffers. There are 8 internal buffers (from 1 to 8) and they have a length of 126 bytes.

The functions described reference two entities: the memory and a serial line, towards which to perform read-write operations two logic channels have been described. By means of the concept of the logic channel, it has been possible to uniform the read-write instructions for data as described in the following. Thus an open system is created ready for future developments. Without adding any new instructions, it is possible to define further logic channels.

At present, 3 logic channels are defined:

- LOGIC CHANNEL 0 = front serial port (terminal).
- LOGIC CHANNEL 1 = memory channel (archive). (Not Implemented)
- LOGIC CHANNEL 2 = connection with PC to download archive. (Not Implemented)

For every logic channel, there is a longword (32 bit) that defines its state. Every bit of this longword represents a particular condition in which the channel finds itself.

The meaning of the bits (bit No. 0 is the least significant bit of the longword) can be different for each channel.

State of LOGIC CHANNEL 1:

bit No. 0	Corrupt archive
bit No. 1	Full archive

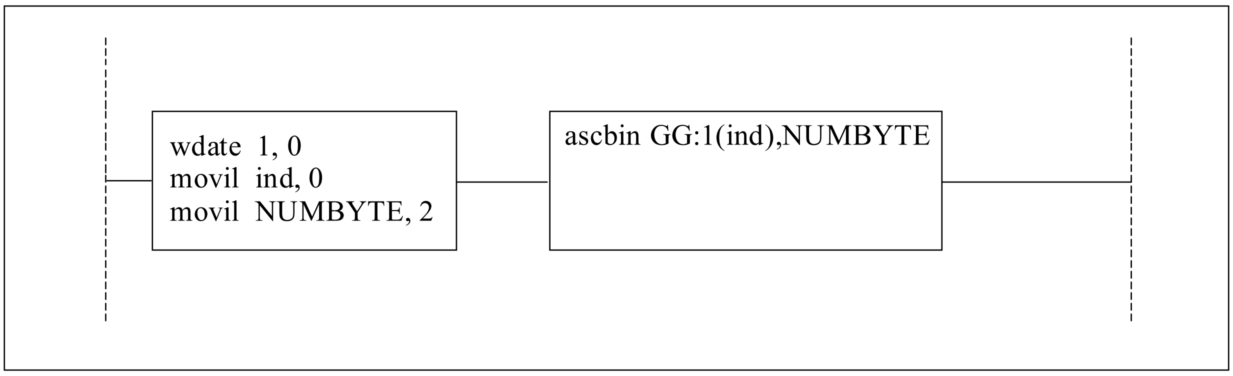
State of LOGIC CHANNEL 0 e 2:

bit No. 0	reception enabled
bit No. 1	transmission enabled
bit No. 2	reception buffer full
bit No. 3	transmission buffer full
bit No. 4	error during data reception
bit No. 5	error during data transmission
bit No. 6	character available in reception
bit No. 7	echo character received enabled

Communications parameters : 9600 - ODD - 8 - 1

Example:

```
GG      assign 128
NUMBYTE assign 130
indassign 132
...
   wdate1,0              ;write data in buffer 1
   movilind,0       ;zero index
   movilNUMBYTE,2        ;NUMBYTE <= 2 (byte to convert)
   ascbin    GG:1(ind),NUMBYTE    ;GG <= number of the day
...
endblo
...
```



2.6.1 INITCN Instruction (Not Implemented)

Syntax:

INITCN ncan

Description:

The instruction execute all the initialisation operations for a logic channel *ncan* (*ncan* can have the values 0, 1 or 2). The operations executed depend on the type of logic channel, in particular:

- for LOGIC CHANNEL 1 the buffered memory area, set aside for the storage of archive records, is cleared and the read and write pointers are repositioned and the channel state is reset to zero.
- for LOGIC CHANNELS 0 and 2, the transmission and reception buffers are reset and the relative read and write pointers are repositioned; the channel state is reset to zero disabling the transmission and reception.

Notes:

Communications parameters : 9600 - ODD - 8 - 1

Example:

```
...
...
initcn 0

endblo
...
```

2.6.2 RCNST Instruction

Syntax:

RCNST ncan, var

Description:

var <= state of the channel.
The state of the logic channel *ncan* is written in the variable *var*. By defining the variable *var* in a suitable place in the image memory and using the instructions to test the points, it is possible to examine the state of the channel.

Notes:

Example:

```
VAR assign 128
...
rcnst 0, VAR

endblo
...
```

2.6.3 WCNST Instruction

Syntax:

WCNST ncan, var

Description:

var ==> state of the channel
The value of the variable *var* is written in the state of the logic channel. Using this instruction, it is possible, for example, to enable the transmission and/or reception of the LOGIC CHANNELS 0 and 2.

Notes:

Example:

```
VAR assign 128
...
movil    VAR,3;VAR contains 3 (0...0011)
wcnst    0, VAR    ;bit No.0 of channel 0 is enabled for
                   ;reception and bit No.1 for transmission

endblo
...
```

2.6.4 RCNx (RCNL, RCNW, RCNB) Instruction

Syntax:

RCNx ncan, var

Description:

var <= logic channel.
The suffix x can take up the following values: L (longword), W (word), or B (byte). The data read by the logic channel *ncan* are written in the variable *var,* and more precisely:

- RCNL modifies the 32 bits of the variable *var;*
- RCNW modifies the 16 least significant bits of the variable *var*;
- RCNB modifies the 8 least significant bits of the variable *var*.

RCNx instructions are of the suspensive type, this means that if characters are not received from the logic channel specified, the channel is suspended expecting the reception. To obtain a reading of the non-suspensive type, it is necessary to use the RCNST instruction to examine the presence or otherwise of data. Furthermore, using the timing instructions it is possible to perform timed readings. The outcome of the read operation is indicated in the state of the logic channel, thus enabling the correctness of the received data to be checked. Typically, therefore, an RCNx instruction will always follow an RCNST one to check the presence of otherwise of errors.

Notes:

Example:

```
VAR     assign 128 ;32 bit variable
STATE        assign 28      ;32 bit variable marker
ERTRA        assign 4(m)    ;data reception error marker
...
rcnl    0, VAR    ;the data read from channel 1 are written
                  ;to VAR.
rcnst   0, STATE  ;state of channel 0 in STATE.
andERTRA;if ERTRA is 1 an error has been detected.
;...
;...error handling
;...
endblo
...
```

2.6.5 WCNx (WCNL, WCNW, WCNB) Instruction

Syntax:

WCNx ncan, var

Description:

var ==> logic channel.
The suffix x can take up the following values: L (longword), W (word), or B (byte).
The value of the variable *var* is written to the logic channel *ncan,* and more precisely:

- WCNL writes 32 bit to the channel;
- WCNW writes the 16 least significant bits of the channel;
- WCNB writes the 8 least significant bits to the channel.

Notes:

Example:

```
VAR      assign 128 ;32 bit variable
STATE         assign 28      ;32bit variable marker
ERTRA         assign 5(m)    ;data transmission error marker
...
wcnl     0, VAR    ;the contents of var are written to channel 0.
rcnst    0, STATE  ;state of channel 0 in STATE.
andERTRA;if ERTRA is 1, an error has been detected.
;...
;...error handling
;...
endblo
...
```

2.6.6 FINREC Instruction (Not Implemented)

Syntax:

FINREC

Description:

End of record ==> logic channel 1.
This instruction is dedicated exclusively to LOGIC CHANNEL 1 and is used to indicate the end of a group of information (record) that, after the execution of the instruction, can be definitively positioned in the archive and sent on request to the host computer.

Notes:

Example:

```
;...
;...series of recordings
;...
finrec
...
...
endblo
...
```

2.6.7 WCNMSG Instruction

Syntax:

WCNMSG ncan, "msg"

Description:

msg ==> logic channel.
The ASCII string ‘msg’ is written to the logic channel *ncan*. The maximum allowed length for ‘msg’ is 40 characters.

Notes:

Example:

```
...
wcnmsg        0, "OK"
...
...
endblo
...
```

2.6.8 WDATE Instruction

Syntax:

WDATE buf, ncan

Description:

date and time ==> logic channel.
Using this instruction, it is possible to write the date and time inside the system on the logic channel *ncan* or in the internal buffer *buf*. If *buf* = 0 the string containing the date and time is written directly to the logic channel *ncan* in the standard format. If *buf* is a value between 1 and 8, the value *ncan* has no significance and the string is written in the internal buffer *buf* beginning from position 0.

The standard format is the following:

DD/MM/YYYY hh:mm:ss

dove;

DD	= 2 digits that indicate the day;
MM	= 2 digits that indicate the month;
YYYY	= 4 digits that indicate the year;
hh	= 2 digits that indicate the hour;
mm	= 2 digits that indicate the minutes;
ss	= 2 digits that indicate the seconds.

The writing of the date and time in one of the available internal buffers instead of directly to a logic channel enables the manipulation of the standard format using the RBUF and WBUF instructions and the subsequent writing with the instruction WCNBUF.
Notes:

Miró does not have an internal clock

Example:

```
...
...
wdate   0, 1       ;standard writing to channel 0
wdate   1, 0       ;standard writing to the internal buffer 1
...
endblo
...
```

2.6.9 BINASC Instruction

Syntax:

BINASC var1:buf(vidx), var2

Description:

var1 ==> internal buffer.
The value of the variable *var1* (32 bit variable) is converted into the ASCII string that represents it. This latter is placed inside the internal buffer *buf* beginning from the position expressed by the variable *vidx*. After the execution of the instruction, the variable *var2* contains the number of digits that make up the converted number, while the variable *vidx* will be updated: *vidx* = *vidx* + *var2*..

Notes:

Example:

```
VAR1    assign 128
VAR2    assign 130
indassign 132
...
   movilind,0
   binasc    VAR1:1(ind),VAR2
...
endblo
...
```

2.6.10 ASCBIN Instruction

Syntax:

ASCBIN var1:buf(vidx), var2

Description:

var1 <= internal buffer.
The ASCII string contained in the internal buffer *buf* beginning at the position specified by the variable *vidx* is converted into the corresponding binary value, which is attributed to the variable *var1* (32 bit variable). The variable *var2* must contain the number of digits (bytes) of the internal buffer that must be taken into consideration for the conversion. After the execution of the instruction, the variable *vdix* will be updated: *vidx* = *vidx* + *var2*.

Notes:

Example:

```
VAR1     assign 128
VAR2     assign 130
indassign 132
...
   movilind,0
   movilVAR2,2
   ascbin    VAR1:1(ind),VAR2
...
endblo
...
```

2.6.11 WCNBUF Instruction

Syntax:

WCNBUF ncan:buf(vidx), var

Description:

logic channel <= internal buffer.
The internal buffer *buf* is written to the channel *ncan*. In particular, the variable *var* contains the number of characters (bytes) that must be written, while the variable *vdix* indicates the position of the initial reading of the characters in the internal buffer. After the execution, the variable *vdix* will be updated: *vidx* = *vidx* + *var*.

Notes:

Example:

```
VARassign 128
indassign 130
...
   movil ind,0
   movil VAR,5
   wcnbuf     0:1(ind),VAR
...
endblo
```

2.6.12 RBUF Instruction

Syntax:

RBUF var:buf(vidx)

Description:

var <= internal buffer.
The character (byte) contained in the internal buffer *buf* in the position *vidx* is written in the least significant 8 bits of the variable *var*. The other bits are not changed. After the execution of the instruction, the variable *vidx* will be updated: *vidx = vidx + 1*.

Notes:

Example:

```
VARassign 128
indassign 130
...
   movilind,0
   rbuf      VAR:1(ind)
...
endblo
```

2.6.13 WBUF Instruction

Syntax:

WBUF var:buf(vidx)

Description:

var ==> internal buffer.
The 8 least significant bits (byte) of the variable *var* are written in the internal buffer *buf* in the position expressed by *vidx*. After the execution of the instruction, the *vdix* will be updated: *vidx = vidx + 1*.

Notes:

Example:

```
VARassign 128
indassign 130
...
   movil ind,0
   movil VAR,10
   wbuf      VAR:1(ind)
...
endblo
```

3. Image Memory Tables

3.1 Introduction

The states of the **input** and **output** tags are read using the appropriate instruction and are transferred to the internal memory, where they are organised in a compacted area that represents them.
Such real tags go towards forming, together with other internal tags, the IMAGE MEMORY, that part of the memory that contained a photograph of the situation outside the Controller system, which is updated at times controlled by the programmer.

3.2 PLC Variables

PLC Variables

addr	LONG WORD 31...........00				
	ODD WORDS 15..........00		EVEN WORDS 15..........00		
000	0031 --------- 0024	0023 --------- 0016	0015 --------- 0008	0007 --------- 0000	**General VARIABLES 32 (0 ---- 31)**
002	0063 --------- 0056	0055 --------- 0048	0047 --------- 0040	0039 --------- 0032	
004	0095 --------- 0088	0087 --------- 0080	0079 --------- 0072	0071 --------- 0064	
006	0127 --------- 0120	0119 --------- 0112	0111 --------- 0104	0103 --------- 0096	
008	0159 --------- 0152	0151 --------- 0144	0143 --------- 0136	0135 --------- 0128	
010	0191 --------- 0184	0183 --------- 0176	0175 --------- 0168	0167 --------- 0160	
012	0223 --------- 0216	0215 --------- 0208	0207 --------- 0200	0199 --------- 0192	
014	0255 --------- 0248	0247 --------- 0240	0239 --------- 0232	0231 --------- 0224	
016	0287 --------- 0280	0279 --------- 0272	0271 --------- 0264	0263 --------- 0256	
018	0319 --------- 0312	0311 --------- 0304	0303 --------- 0296	0295 --------- 0288	
020	0351 --------- 0344	0343 --------- 0336	0335 --------- 0328	0327 --------- 0320	
022	0383 --------- 0376	0375 --------- 0368	0367 --------- 0360	0359 --------- 0352	
024	0415 --------- 0408	0407 --------- 0400	0399 --------- 0392	0391 --------- 0384	
026	0447 --------- 0440	0439 --------- 0432	0431 --------- 0424	0423 --------- 0416	
028	0479 --------- 0472	0471 --------- 0464	0463 --------- 0456	0455 --------- 0448	
030	0511 --------- 0504	0503 --------- 0496	0495 --------- 0488	0487 --------- 0480	
032	0543 --------- 0536	0535 --------- 0528	0527 --------- 0520	0519 --------- 0512	
034	0575 --------- 0568	0567 --------- 0560	0559 --------- 0552	0551 --------- 0544	
036	0607 --------- 0600	0599 --------- 0592	0591 --------- 0584	0583 --------- 0576	
038	0639 --------- 0632	0631 --------- 0624	0623 --------- 0616	0615 --------- 0608	
040	0671 --------- 0664	0663 --------- 0656	0655 --------- 0648	0647 --------- 0640	
042	0703 --------- 0696	0695 --------- 0688	0687 --------- 0680	0679 --------- 0672	
044	0735 --------- 0728	0727 --------- 0720	0719 --------- 0712	0711 --------- 0704	
046	0767 --------- 0760	0759 --------- 0752	0751 --------- 0744	0743 --------- 0736	
048	0799 --------- 0792	0791 --------- 0784	0783 --------- 0776	0775 --------- 0768	
050	0831 --------- 0824	0823 --------- 0816	0815 --------- 0808	0807 --------- 0800	
052	0863 --------- 0856	0855 --------- 0848	0847 --------- 0840	0839 --------- 0832	
054	0895 --------- 0888	0887 --------- 0880	0879 --------- 0872	0871 --------- 0864	
056	0827 --------- 0920	0919 --------- 0912	0911 --------- 0904	0903 --------- 0896	
058	0959 --------- 0952	0951 --------- 0944	0943 --------- 0936	0935 --------- 0928	
060	0991 --------- 0984	0983 --------- 0976	0975 --------- 0968	0967 --------- 0960	
062	1023 --------- 1016	1015 --------- 1008	1007 --------- 1000	0999 --------- 0992	
064	1055 --------- 1048	1047 --------- 1040	1039 --------- 1032	1031 --------- 1024	**CNC --> PLC VARIABLES 8 (0..7)**
066	1087 --------- 1080	1079 --------- 1072	1071 --------- 1064	1063 --------- 1056	
068	1119 --------- 1112	1111 --------- 1104	1103 --------- 1096	1095 --------- 1088	
070	1151 --------- 1144	1143 --------- 1136	1135 --------- 1128	1127 --------- 1120	
072	1183 --------- 1176	1175 --------- 1168	1167 --------- 1160	1159 --------- 1152	
074	1215 --------- 1208	1207 --------- 1200	1199 --------- 1192	1191 --------- 1184	
076	1247 --------- 1240	1239 --------- 1232	1231 --------- 1224	1223 --------- 1216	
078	1279 --------- 1272	1271 --------- 1264	1263 --------- 1256	1255 --------- 1248	
080	1311 --------- 1304	1303 --------- 1296	1295 --------- 1288	1287 --------- 1280	**CNC <-- PLC VARIABLES 8 (0..7)**
082	1343 --------- 1336	1335 --------- 1328	1327 --------- 1320	1319 --------- 1312	
084	1375 --------- 1368	1367 --------- 1360	1359 --------- 1352	1351 --------- 1344	
086	1407 --------- 1400	1399 --------- 1392	1391 --------- 1384	1383 --------- 1376	
088	1439 --------- 1432	1431 --------- 1424	1423 --------- 1416	1415 --------- 1408	
090	1471 --------- 1464	1463 --------- 1456	1455 --------- 1448	1447 --------- 1440	
092	1503 --------- 1496	1495 --------- 1488	1487 --------- 1480	1479 --------- 1472	
094	1535 --------- 1528	1527 --------- 1520	1519 --------- 1512	1511 --------- 1504	
096	1567 --------- 1560	1559 --------- 1552	1551 --------- 1544	1543 --------- 1536	**SPECIAL VARIABLES 8 (0..7)**
098	1599 --------- 1592	1591 --------- 1584	1583 --------- 1576	1575 --------- 1568	
100	1631 --------- 1624	1623 --------- 1616	1615 --------- 1608	1607 --------- 1600	
102	1663 --------- 1656	1655 --------- 1648	1647 --------- 1640	1639 --------- 1632	
104	1695 --------- 1688	1687 --------- 1680	1679 --------- 1672	1671 --------- 1664	
106	1627 --------- 1720	1719 --------- 1712	1711 --------- 1704	1703 --------- 1696	
108	1759 --------- 1752	1751 --------- 1744	1743 --------- 1736	1735 --------- 1728	
110	1791 --------- 1784	1783 --------- 1776	1775 --------- 1768	1767 --------- 1760	
112	1823 --------- 1816	1815 --------- 1808	1807 --------- 1800	1799 --------- 1792	**SPECIAL VARIABLES 8 (0..7)**
114	1855 --------- 1848	1847 --------- 1840	1839 --------- 1832	1831 --------- 1824	
116	1887 --------- 1880	1879 --------- 1872	1871 --------- 1864	1863 --------- 1856	
118	1919 --------- 1912	1911 --------- 1904	1903 --------- 1896	1895 --------- 1888	
120	1951 --------- 1944	1943 --------- 1936	1935 --------- 1928	1927 --------- 1920	
122	1983 --------- 1976	1975 --------- 1968	1967 --------- 1960	1959 --------- 1952	
124	2015 --------- 2008	2007 --------- 2000	1999 --------- 1992	1991 --------- 1984	
126	2047 --------- 2040	2039 --------- 2032	2031 --------- 2024	2023 --------- 2016	

PLC Variables					
addr	LONG WORD 31.....00				
	ODD WORDS 15.....00		EVEN WORDS 15.....00		
128					INTERMEDIATE 64 (0 ---- 63)
130					
132					
134					
136					
138					
140					
142					
144					
146					
148					
150					
152					
154					
156					
158					
160					
162					
164					
166					
168					
170					
172					
174					
176					
178					
180					
182					
184					
186					
188					
190					
192					
194					
196					
198					
200					
202					
204					
206					
208					
210					
212					
214					
216					
218					
220					
222					
224					
226					
228					
230					
232					
234					
236					
238					
240					
242					
244					
246					
248					
250					
252					
254					

3.3 PLC IMAGE Memory

PLC IMAGE Memory					
addr	LONG WORD 31…00				
	ODD WORDS 15…00		EVEN WORDS 15…00		
000	0031 --------- 0024	0023 --------- 0016	0015 --------- 0008	0007 --------- 0000	
002	0063 --------- 0056	0055 --------- 0048	0047 --------- 0040	0039 --------- 0032	**EXTERNAL INPUTS**
004	0095 --------- 0088	0087 --------- 0080	0079 --------- 0072	0071 --------- 0064	**128 (0..127)**
006	0027 --------- 0120	0119 --------- 0112	0111 --------- 0104	0103 --------- 0096	
008	0159 --------- 0152	0151 --------- 0144	0143 --------- 0136	0135 --------- 0128	
010	0191 --------- 0184	0183 --------- 0176	0175 --------- 0168	0167 --------- 0160	**EXTERNAL OUTPUTS**
012	0223 --------- 0216	0215 --------- 0208	0207 --------- 0200	0199 --------- 0192	**128 (0..127)**
014	0255 --------- 0248	0247 --------- 0240	0239 --------- 0232	0231 --------- 0224	
016	0287 --------- 0280	0279 --------- 0272	0271 --------- 0264	0263 --------- 0256	**SEQUENCES 64 (0..63)**
018	0319 --------- 0312	0311 --------- 0304	0303 --------- 0296	0295 --------- 0288	
020	0351 --------- 0344	0343 --------- 0336	0335 --------- 0328	0327 --------- 0320	**FLAGS 64 (0..63)**
022	0383 --------- 0376	0375 --------- 0368	0367 --------- 0360	0359 --------- 0352	
024	0415 --------- 0408	0407 --------- 0400	0399 --------- 0392	0391 --------- 0384	**COUNTERS 32 (0..31)**
026	0447 --------- 0440	0439 --------- 0432	0431 --------- 0424	0423 --------- 0416	**TIMERS 32 (0..31)**
028	0479 --------- 0472	0471 --------- 0464	0463 --------- 0456	0455 --------- 0448	
030	0511 --------- 0504	0503 --------- 0496	0495 --------- 0488	0487 --------- 0480	
032	0543 --------- 0536	0535 --------- 0528	0527 --------- 0520	0519 --------- 0512	
034	0575 --------- 0568	0567 --------- 0560	0559 --------- 0552	0551 --------- 0544	**MARKERS**
036	0607 --------- 0600	0599 --------- 0592	0591 --------- 0584	0583 --------- 0576	**256 (0..255)**
038	0639 --------- 0632	0631 --------- 0624	0623 --------- 0616	0615 --------- 0608	
040	0671 --------- 0664	0663 --------- 0656	0655 --------- 0648	0647 --------- 0640	
042	0703 --------- 0696	0695 --------- 0688	0687 --------- 0680	0679 --------- 0672	
044	0735 --------- 0728	0727 --------- 0720	0719 --------- 0712	0711 --------- 0704	
046	0767 --------- 0760	0759 --------- 0752	0751 --------- 0744	0743 --------- 0736	
048	0799 --------- 0792	0791 --------- 0784	0783 --------- 0776	0775 --------- 0768	
050	0831 --------- 0824	0823 --------- 0816	0815 --------- 0808	0807 --------- 0800	**AUXILIARY MARKERS**
052	0863 --------- 0856	0855 --------- 0848	0847 --------- 0840	0839 --------- 0832	**256 (0..255)**
054	0895 --------- 0888	0887 --------- 0880	0879 --------- 0872	0871 --------- 0864	
056	0927 --------- 0920	0919 --------- 0912	0911 --------- 0904	0903 --------- 0896	
058	0959 --------- 0952	0951 --------- 0944	0943 --------- 0936	0935 --------- 0928	
060	0991 --------- 0984	0983 --------- 0976	0975 --------- 0968	0967 --------- 0960	**INPUTS from CNC 32 (0..31)**
062	1023 --------- 1016	1015 --------- 1008	1007 --------- 1000	0999 --------- 0992	**OUTPUTS to CNC 32 (0..31)**

3.3.1 External Inputs (Long 000)

EXTERNAL INPUTS [128] (Long 000: 31.....0)

wrd	Input	Term.	Code	Description
000	000(I)-0000	terminal		
	001(I)-0001			
	002(I)-0002			
	003(I)-0003			
	004(I)-0004			
	005(I)-0005			
	006(I)-0006			
	007(I)-0007			
	008(I)-0008			
	009(I)-0009			
	010(I)-0010			
	011(I)-0011			
	012(I)-0012			
	013(I)-0013			
	014(I)-0014			
	015(I)-0015			
001	016(I)-0016			
	017(I)-0017			
	018(I)-0018			
	019(I)-0019			
	020(I)-0020			
	021(I)-0021			
	022(I)-0022			
	023(I)-0023			
	024(I)-0024			
	025(I)-0025			
	026(I)-0026			
	027(I)-0027			
	028(I)-0028			
	029(I)-0029			
	030(I)-0030			
	031(I)-0031			

3.3.2 External Inputs (Long 002)

EXTERNAL INPUTS [128] (Long 002: 63.....32)

wrd	Input	Term.	Code	Description
002	032(I)-0032			
	033(I)-0033			
	034(I)-0034			
	035(I)-0035			
	036(I)-0036			
	037(I)-0037			
	038(I)-0038			
	039(I)-0039			
	040(I)-0040			
	041(I)-0041			
	042(I)-0042			
	043(I)-0043			
	044(I)-0044			
	045(I)-0045			
	046(I)-0046			
	047(I)-0047			
003	048(I)-0048			
	049(I)-0049			
	050(I)-0050			
	051(I)-0051			
	052(I)-0052			
	053(I)-0053			
	054(I)-0054			
	055(I)-0055			
	056(I)-0056			
	057(I)-0057			
	058(I)-0058			
	059(I)-0059			
	060(I)-0060			
	061(I)-0061			
	062(I)-0062			
	063(I)-0063			

3.3.3 External Inputs (Long 004)

EXTERNAL INPUTS [128] (Long 004: 95.....64)

wrd	Input	Term.	Code	Description
004	064(I)-0064			
	065(I)-0065			
	066(I)-0066			
	067(I)-0067			
	068(I)-0068			
	069(I)-0069			
	070(I)-0070			
	071(I)-0071			
	072(I)-0072			
	073(I)-0073			
	074(I)-0074			
	075(I)-0075			
	076(I)-0076			
	077(I)-0077			
	078(I)-0078			
	079(I)-0079			
005	080(I)-0080			
	081(I)-0081			
	082(I)-0082			
	083(I)-0083			
	084(I)-0084			
	085(I)-0085			
	086(I)-0086			
	087(I)-0087			
	088(I)-0088			
	089(I)-0089			
	090(I)-0090			
	091(I)-0091			
	092(I)-0092			
	093(I)-0093			
	094(I)-0094			
	095(I)-0095			

3.3.4 External Inputs (Long 006)

EXTERNAL INPUTS [128] (Long 006: 127.....96)

wrd	Input	Term.	Code	Description
006	096(I)-0096			
	097(I)-0097			
	098(I)-0098			
	099(I)-0099			
	100(I)-0100			
	101(I)-0101			
	102(I)-0102			
	103(I)-0103			
	104(I)-0104			
	105(I)-0105			
	106(I)-0106			
	107(I)-0107			
	108(I)-0108			
	109(I)-0109			
	110(I)-0110			
	111(I)-0111			
007	112(I)-0112			
	113(I)-0113			
	114(I)-0114			
	115(I)-0115			
	116(I)-0116			
	117(I)-0117			
	118(I)-0118			
	119(I)-0119			
	120(I)-0120			
	121(I)-0121			
	122(I)-0122			
	123(I)-0123			
	124(I)-0124			
	125(I)-0125			
	126(I)-0126			
	127(I)-0127			

3.3.5 External Outputs (Long 008)

EXTERNAL OUTPUTS [128] (Long 008: 159....128)

wrd	Output	Term.	Code	Description
008	000(O)-0128			
	001(O)-0129			
	002(O)-0130			
	003(O)-0131			
	004(O)-0132			
	005(O)-0133			
	006(O)-0134			
	007(O)-0135			
	008(O)-0136			
	009(O)-0137			
	010(O)-0138			
	011(O)-0139			
	012(O)-0140			
	013(O)-0141			
	014(O)-0142			
	015(O)-0143			
009	016(O)-0144			
	017(O)-0145			
	018(O)-0146			
	019(O)-0147			
	020(O)-0148			
	021(O)-0149			
	022(O)-0150			
	023(O)-0151			
	024(O)-0152			
	025(O)-0153			
	026(O)-0154			
	027(O)-0155			
	028(O)-0156			
	029(O)-0157			
	030(O)-0158			
	031(O)-0159			

3.3.6 External Outputs (Long 010)

EXTERNAL OUTPUTS [128] (Long 010: 191....160)

wrd	Output	Term.	Code	Description
010	032(O)-0160			
	033(O)-0161			
	034(O)-0162			
	035(O)-0163			
	036(O)-0164			
	037(O)-0165			
	038(O)-0166			
	039(O)-0167			
	040(O)-0168			
	041(O)-0169			
	042(O)-0170			
	043(O)-0171			
	044(O)-0172			
	045(O)-0173			
	046(O)-0174			
	047(O)-0175			
011	048(O)-0176			
	049(O)-0177			
	050(O)-0178			
	051(O)-0179			
	052(O)-0180			
	053(O)-0181			
	054(O)-0182			
	055(O)-0183			
	056(O)-0184			
	057(O)-0185			
	058(O)-0186			
	059(O)-0187			
	060(O)-0188			
	061(O)-0189			
	062(O)-0190			
	063(O)-0191			

3.3.7 External Outputs (Long 012)

EXTERNAL OUTPUTS [128] (Long 012: 223....192)

wrd	Output	Term.	Code	Description
012	064(O)-0192			
	065(O)-0193			
	066(O)-0194			
	067(O)-0195			
	068(O)-0196			
	069(O)-0197			
	070(O)-0198			
	071(O)-0199			
	072(O)-0200			
	073(O)-0201			
	074(O)-0202			
	075(O)-0203			
	076(O)-0204			
	077(O)-0205			
	078(O)-0206			
	079(O)-0207			
013	080(O)-0208			
	081(O)-0209			
	082(O)-0210			
	083(O)-0211			
	084(O)-0212			
	085(O)-0213			
	086(O)-0214			
	087(O)-0215			
	088(O)-0216			
	089(O)-0217			
	090(O)-0218			
	091(O)-0219			
	092(O)-0220			
	093(O)-0221			
	094(O)-0222			
	095(O)-0223			

3.3.8 External Outputs (Long 014)

EXTERNAL OUTPUTS [128] (Long 014: 255.....224)

wrd	Output	Term.	Code	Description
014	096(O)-0224			
	097(O)-0225			
	098(O)-0226			
	099(O)-0227			
	100(O)-0228			
	101(O)-0229			
	102(O)-0230			
	103(O)-0231			
	104(O)-0232			
	105(O)-0233			
	106(O)-0234			
	107(O)-0235			
	108(O)-0236			
	109(O)-0237			
	110(O)-0238			
	111(O)-0239			
015	112(O)-0240			
	113(O)-0241			
	114(O)-0242			
	115(O)-0243			
	116(O)-0244			
	117(O)-0245			
	118(O)-0246			
	119(O)-0247			
	120(O)-0248			
	121(O)-0249			
	122(O)-0250			
	123(O)-0251			
	124(O)-0252			
	125(O)-0253			
	126(O)-0254			
	127(O)-0255			

3.3.9 Sequences (Long 016)

SEQUENCES [64] (Long 016: 287.....256)

wrd	Sequence	Term.	Code	Description
016	000(S)-0256			
	001(S)-0257			
	002(S)-0258			
	003(S)-0259			
	004(S)-0260			
	005(S)-0261			
	006(S)-0262			
	007(S)-0263			
	008(S)-0264			
	009(S)-0265			
	010(S)-0266			
	011(S)-0267			
	012(S)-0268			
	013(S)-0269			
	014(S)-0270			
	015(S)-0271			
017	016(S)-0272			
	017(S)-0273			
	018(S)-0274			
	019(S)-0275			
	020(S)-0276			
	021(S)-0277			
	022(S)-0278			
	023(S)-0279			
	024(S)-0280			
	025(S)-0281			
	026(S)-0282			
	027(S)-0283			
	028(S)-0284			
	029(S)-0285			
	030(S)-0286			
	031(S)-0287			

3.3.10 Sequences (Long 018)

SEQUENCES [64] (Long 018: 319.....288)

wrd	Sequence	Term.	Code	Description
018	032(S)-0288			
	033(S)-0289			
	034(S)-0290			
	035(S)-0291			
	036(S)-0292			
	037(S)-0293			
	038(S)-0294			
	039(S)-0295			
	040(S)-0296			
	041(S)-0297			
	042(S)-0298			
	043(S)-0299			
	044(S)-0300			
	045(S)-0301			
	046(S)-0302			
	047(S)-0303			
019	048(S)-0304			
	049(S)-0305			
	050(S)-0306			
	051(S)-0307			
	052(S)-0308			
	053(S)-0309			
	054(S)-0310			
	055(S)-0311			
	056(S)-0312			
	057(S)-0313			
	058(S)-0314			
	059(S)-0315			
	060(S)-0316			
	061(S)-0317			
	062(S)-0318			
	063(S)-0319			

3.3.11 Flags (Long 020)

FLAGS [64] (Long 020: 351.....320)

wrd	Flag	Notes	Code	Description
020	000(F)-0320	CNC->M10		
	001(F)-0321			
	002(F)-0322			
	003(F)-0323			
	004(F)-0324			
	005(F)-0325			
	006(F)-0326			
	007(F)-0327			
	008(F)-0328			
	009(F)-0329			
	010(F)-0330			
	011(F)-0331			
	012(F)-0332			
	013(F)-0333			
	014(F)-0334			
	015(F)-0335			
021	016(F)-0336			
	017(F)-0337			
	018(F)-0338			
	019(F)-0339			
	020(F)-0340			
	021(F)-0341			
	022(F)-0342			
	023(F)-0343			
	024(F)-0344			
	025(F)-0345			
	026(F)-0346			
	027(F)-0347			
	028(F)-0348			
	029(F)-0349			
	030(F)-0350			
	031(F)-0351			

3.3.12 Flags (Long 022)

FLAGS [64] (Long 022: 383.....352)

wrd	Flag	Notes	Code	Description
022	032(F)-0352	CNC->T00		
	033(F)-0353			
	034(F)-0354			
	035(F)-0355			
	036(F)-0356			
	037(F)-0357			
	038(F)-0358			
	039(F)-0359			
	040(F)-0360			
	041(F)-0361			
	042(F)-0362			
	043(F)-0363			
	044(F)-0364			
	045(F)-0365			
	046(F)-0366			
	047(F)-0367			
023	048(F)-0368			
	049(F)-0369			
	050(F)-0370			
	051(F)-0371			
	052(F)-0372			
	053(F)-0373			
	054(F)-0374			
	055(F)-0375			
	056(F)-0376			
	057(F)-0377			
	058(F)-0378			
	059(F)-0379			
	060(F)-0380			
	061(F)-0381			
	062(F)-0382			
	063(F)-0383			

3.3.13 Counters (Long 024)

COUNTERS [32] (Long 024: 415.....384)

wrd	Counter	Notes	Code	Description
024	000(C)-0384	fast (2)		
	001(C)-0385			
	002(C)-0386			
	003(C)-0387			
	004(C)-0388			
	005(C)-0389			
	006(C)-0390			
	007(C)-0391			
	008(C)-0392			
	009(C)-0393			
	010(C)-0394			
	011(C)-0395			
	012(C)-0396			
	013(C)-0397			
	014(C)-0398			
	015(C)-0399			
025	016(C)-0400	slow (10)		
	017(C)-0401			
	018(C)-0402			
	019(C)-0403			
	020(C)-0404			
	021(C)-0405			
	022(C)-0406			
	023(C)-0407			
	024(C)-0408			
	025(C)-0409			
	026(C)-0410			
	027(C)-0411			
	028(C)-0412			
	029(C)-0413			
	030(C)-0414			
	031(C)-0415			

3.3.14 Timers (Long 026)

TIMERS [32] (Long 026: 447.....416)

wrd	Timer	Notes	Code	Description
026	000(T)-0416	fast (10)		
	001(T)-0417			
	002(T)-0418			
	003(T)-0419			
	004(T)-0420			
	005(T)-0421			
	006(T)-0422			
	007(T)-0423			
	008(T)-0424			
	009(T)-0425			
	010(T)-0426			
	011(T)-0427			
	012(T)-0428			
	013(T)-0429			
	014(T)-0430			
	015(T)-0431			
027	016(T)-0432	slow(100)		
	017(T)-0433			
	018(T)-0434			
	019(T)-0435			
	020(T)-0436			
	021(T)-0437			
	022(T)-0438			
	023(T)-0439			
	024(T)-0440			
	025(T)-0441			
	026(T)-0442			
	027(T)-0443			
	028(T)-0444			
	029(T)-0445			
	030(T)-0446			
	031(T)-0447			

3.3.15 Markers (Long 028)

MARKERS [128] (Long 028: 479.....448)

wrd	Marker	Notes	Code	Description
028	000(M)-0448			
	001(M)-0449			
	002(M)-0450			
	003(M)-0451			
	004(M)-0452			
	005(M)-0453			
	006(M)-0454			
	007(M)-0455			
	008(M)-0456			
	009(M)-0457			
	010(M)-0458			
	011(M)-0459			
	012(M)-0460			
	013(M)-0461			
	014(M)-0462			
	015(M)-0463			
029	016(M)-0464			
	017(M)-0465			
	018(M)-0466			
	019(M)-0467			
	020(M)-0468			
	021(M)-0469			
	022(M)-0470			
	023(M)-0471			
	024(M)-0472			
	025(M)-0473			
	026(M)-0474			
	027(M)-0475			
	028(M)-0476			
	029(M)-0477			
	030(M)-0478			
	031(M)-0479			

3.3.16 Markers (Long 030)

MARKERS [128] (Long 030: 511.....480)

wrd	Marker	Notes	Code	Description
030	032(M)-0480			
	033(M)-0481			
	034(M)-0482			
	035(M)-0483			
	036(M)-0484			
	037(M)-0485			
	038(M)-0486			
	039(M)-0487			
	040(M)-0488			
	041(M)-0489			
	042(M)-0490			
	043(M)-0491			
	044(M)-0492			
	045(M)-0493			
	046(M)-0494			
	047(M)-0495			
031	048(M)-0496			
	049(M)-0497			
	050(M)-0498			
	051(M)-0499			
	052(M)-0500			
	053(M)-0501			
	054(M)-0502			
	055(M)-0503			
	056(M)-0504			
	057(M)-0505			
	058(M)-0506			
	059(M)-0507			
	060(M)-0508			
	061(M)-0509			
	062(M)-0510			
	063(M)-0511			

3.3.17 Markers (Long 032)

MARKERS [128] (Long 032: 543.....512)

wrd	Marker	Notes	Code	Description
032	064(M)-0512			
	065(M)-0513			
	066(M)-0514			
	067(M)-0515			
	068(M)-0516			
	069(M)-0517			
	070(M)-0518			
	071(M)-0519			
	072(M)-0520			
	073(M)-0521			
	074(M)-0522			
	075(M)-0523			
	076(M)-0524			
	077(M)-0525			
	078(M)-0526			
	079(M)-0527			
033	080(M)-0528			
	081(M)-0529			
	082(M)-0530			
	083(M)-0531			
	084(M)-0532			
	085(M)-0533			
	086(M)-0534			
	087(M)-0535			
	088(M)-0536			
	089(M)-0537			
	090(M)-0538			
	091(M)-0539			
	092(M)-0540			
	093(M)-0541			
	094(M)-0542			
	095(M)-0543			

3.3.18 Markers (Long 034)

MARKERS [128] (Long 034: 575.....544)

wrd	Marker	Notes	Code	Description
034	096(M)-0544			
	097(M)-0545			
	098(M)-0546			
	099(M)-0547			
	100(M)-0548			
	101(M)-0549			
	102(M)-0550			
	103(M)-0551			
	104(M)-0552			
	105(M)-0553			
	106(M)-0554			
	107(M)-0555			
	108(M)-0556			
	109(M)-0557			
	110(M)-0558			
	111(M)-0559			
035	112(M)-0560			
	113(M)-0561			
	114(M)-0562			
	115(M)-0563			
	116(M)-0564			
	117(M)-0565			
	118(M)-0566			
	119(M)-0567			
	120(M)-0568			
	121(M)-0569			
	122(M)-0570			
	123(M)-0571			
	124(M)-0572			
	125(M)-0573			
	126(M)-0574			
	127(M)-0575			

3.3.19 Buffered Markers (Long 036)

Buffered MARKERS [128] (Long 036: 607.....576)

wrd	Marker(t)	Notes	Code	Description
036	128(M)-0576			
	129(M)-0577			
	130(M)-0578			
	131(M)-0579			
	132(M)-0580			
	133(M)-0581			
	134(M)-0582			
	135(M)-0583			
	136(M)-0584			
	137(M)-0585			
	138(M)-0586			
	139(M)-0587			
	140(M)-0588			
	141(M)-0589			
	142(M)-0590			
	143(M)-0591			
037	144(M)-0592			
	145(M)-0593			
	146(M)-0594			
	147(M)-0595			
	148(M)-0596			
	149(M)-0597			
	150(M)-0598			
	151(M)-0599			
	152(M)-0600			
	153(M)-0601			
	154(M)-0602			
	155(M)-0603			
	156(M)-0604			
	157(M)-0605			
	158(M)-0606			
	159(M)-0607			

3.3.20 Buffered Markers (Long 038)

Buffered MARKERS [128] (Long 038: 639.....608)

wrd	Marker(t)	Notes	Code	Description
038	160(M)-0608			
	161(M)-0609			
	162(M)-0610			
	163(M)-0611			
	164(M)-0612			
	165(M)-0613			
	166(M)-0614			
	167(M)-0615			
	168(M)-0616			
	169(M)-0617			
	170(M)-0618			
	171(M)-0619			
	172(M)-0620			
	173(M)-0621			
	174(M)-0622			
	175(M)-0623			
039	176(M)-0624			
	177(M)-0625			
	178(M)-0626			
	179(M)-0627			
	180(M)-0628			
	181(M)-0629			
	182(M)-0630			
	183(M)-0631			
	184(M)-0632			
	185(M)-0633			
	186(M)-0634			
	187(M)-0635			
	188(M)-0636			
	189(M)-0637			
	190(M)-0638			
	191(M)-0639			

3.3.21 Buffered Markers (Long 040)

Buffered MARKER [128] (Long 040: 671.....640)

wrd	Marker(t)	Notes	Code	Description
040	192(M)-0640			
	193(M)-0641			
	194(M)-0642			
	195(M)-0643			
	196(M)-0644			
	197(M)-0645			
	198(M)-0646			
	199(M)-0647			
	200(M)-0648			
	201(M)-0649			
	202(M)-0650			
	203(M)-0651			
	204(M)-0652			
	205(M)-0653			
	206(M)-0654			
	207(M)-0655			
041	208(M)-0656			
	209(M)-0657			
	210(M)-0658			
	211(M)-0659			
	212(M)-0660			
	213(M)-0661			
	214(M)-0662			
	215(M)-0663			
	216(M)-0664			
	217(M)-0665			
	218(M)-0666			
	219(M)-0667			
	220(M)-0668			
	221(M)-0669			
	222(M)-0670			
	223(M)-0671			

3.3.22 Buffered Markers (Long 042)

Buffered MARKER [128] (Long 042: 703.....672)

wrd	Marker(t)	Notes	Code	Description
042	224(M)-0672			
	225(M)-0673			
	226(M)-0674			
	227(M)-0675			
	228(M)-0676			
	229(M)-0677			
	230(M)-0678			
	231(M)-0679			
	232(M)-0680			
	233(M)-0681			
	234(M)-0682			
	235(M)-0683			
	236(M)-0684			
	237(M)-0685			
	238(M)-0686			
	239(M)-0687			
043	240(M)-0688	reserved		
	241(M)-0689	reserved		
	242(M)-0690	reserved		
	243(M)-0691	reserved		
	244(M)-0692	reserved		
	245(M)-0693	reserved		
	246(M)-0694	reserved		
	247(M)-0695	monost.		
	248(M)-0696	=1		
	249(M)-0697	Acc		
	250(M)-0698	Z EQ		
	251(M)-0699	NZ NEQ		
	252(M)-0700	LT		
	253(M)-0701	GT		
	254(M)-0702	OVF		
	255(M)-0703	NOAcc		

3.3.23 Auxiliary Markers (Long 044)

Auxiliary MARKERS [256] (Long 044: 735....704)

wrd	Marker-A	Notes	Code	Description
044	000(A)-0704			
	001(A)-0705			
	002(A)-0706			
	003(A)-0707			
	004(A)-0708			
	005(A)-0709			
	006(A)-0710			
	007(A)-0711			
	008(A)-0712			
	009(A)-0713			
	010(A)-0714			
	011(A)-0715			
	012(A)-0716			
	013(A)-0717			
	014(A)-0718			
	015(A)-0719			
045	016(A)-0720			
	017(A)-0721			
	018(A)-0722			
	019(A)-0723			
	020(A)-0724			
	021(A)-0725			
	022(A)-0726			
	023(A)-0727			
	024(A)-0728			
	025(A)-0729			
	026(A)-0730			
	027(A)-0731			
	028(A)-0732			
	029(A)-0733			
	030(A)-0734			
	031(A)-0735			

3.3.24 Auxiliary Markers (Long 046)

Auxiliary MARKERS [256] (Long 046: 767....736)

wrd	Marker-A	Notes	Codes	Description
046	032(A)-0736			
	033(A)-0737			
	034(A)-0738			
	035(A)-0739			
	036(A)-0740			
	037(A)-0741			
	038(A)-0742			
	039(A)-0743			
	040(A)-0744			
	041(A)-0745			
	042(A)-0746			
	043(A)-0747			
	044(A)-0748			
	045(A)-0749			
	046(A)-0750			
	047(A)-0751			
047	048(A)-0752			
	049(A)-0753			
	050(A)-0754			
	051(A)-0755			
	052(A)-0756			
	053(A)-0757			
	054(A)-0758			
	055(A)-0759			
	056(A)-0760			
	057(A)-0761			
	058(A)-0762			
	059(A)-0763			
	060(A)-0764			
	061(A)-0765			
	062(A)-0766			
	063(A)-0767			

3.3.25 Auxiliary Markers (Long 048)

Auxiliary MARKERS [256] (Long 048: 799....768)

wrd	Marker-A	Notes	Code	Description
048	064(A)-0768			
	065(A)-0769			
	066(A)-0770			
	067(A)-0771			
	068(A)-0772			
	069(A)-0773			
	070(A)-0774			
	071(A)-0775			
	072(A)-0776			
	073(A)-0777			
	074(A)-0778			
	075(A)-0779			
	076(A)-0780			
	077(A)-0781			
	078(A)-0782			
	079(A)-0783			
049	080(A)-0784			
	081(A)-0785			
	082(A)-0786			
	083(A)-0787			
	084(A)-0788			
	085(A)-0789			
	086(A)-0790			
	087(A)-0791			
	088(A)-0792			
	089(A)-0793			
	090(A)-0794			
	091(A)-0795			
	092(A)-0796			
	093(A)-0797			
	094(A)-0798			
	095(A)-0799			

3.3.26 Auxiliary Markers (Long 050)

Auxiliary MARKERS [256] (Long 050: 831....800)

wrd	Marker-A	Notes	Code	Description
050	096(A)-0800			
	097(A)-0801			
	098(A)-0802			
	099(A)-0803			
	100(A)-0804			
	101(A)-0805			
	102(A)-0806			
	103(A)-0807			
	104(A)-0808			
	105(A)-0809			
	106(A)-0810			
	107(A)-0811			
	108(A)-0812			
	109(A)-0813			
	110(A)-0814			
	111(A)-0815			
051	112(A)-0816			
	113(A)-0817			
	114(A)-0818			
	115(A)-0819			
	116(A)-0820			
	117(A)-0821			
	118(A)-0822			
	119(A)-0823			
	120(A)-0824			
	121(A)-0825			
	122(A)-0826			
	123(A)-0827			
	124(A)-0828			
	125(A)-0829			
	126(A)-0830			
	127(A)-0831			

3.3.27 Auxiliary Markers (Long 052)

Auxiliary MARKERS [256] (Long 052: 863....832)

wrd	Marker-A	Notes	Code	Description
052	128(A)-0832			
	129(A)-0833			
	130(A)-0834			
	131(A)-0835			
	132(A)-0836			
	133(A)-0837			
	134(A)-0838			
	135(A)-0839			
	136(A)-0840			
	137(A)-0841			
	138(A)-0842			
	139(A)-0843			
	140(A)-0844			
	141(A)-0845			
	142(A)-0846			
	143(A)-0847			
053	144(A)-0848			
	145(A)-0849			
	146(A)-0850			
	147(A)-0851			
	148(A)-0852			
	149(A)-0853			
	150(A)-0854			
	151(A)-0855			
	152(A)-0856			
	153(A)-0857			
	154(A)-0858			
	155(A)-0859			
	156(A)-0860			
	157(A)-0861			
	158(A)-0862			
	159(A)-0863			

3.3.28 Auxiliary Markers (Long 054)

Auxiliary MARKERS [256] (Long 054: 895....864)

wrd	Marker-A	Notes	Code	Description
054	160(A)-0864			
	161(A)-0865			
	162(A)-0866			
	163(A)-0867			
	164(A)-0868			
	165(A)-0869			
	166(A)-0870			
	167(A)-0871			
	168(A)-0872			
	169(A)-0873			
	170(A)-0874			
	171(A)-0875			
	172(A)-0876			
	173(A)-0877			
	174(A)-0878			
	175(A)-0879			
055	176(A)-0880			
	177(A)-0881			
	178(A)-0882			
	179(A)-0883			
	180(A)-0884			
	181(A)-0885			
	182(A)-0886			
	183(A)-0887			
	184(A)-0888			
	185(A)-0889			
	186(A)-0890			
	187(A)-0891			
	188(A)-0892			
	189(A)-0893			
	190(A)-0894			
	191(A)-0895			

3.3.29 Auxiliary Markers (Long 056)

Auxiliary MARKERS [256] (Long 056: 927....896)

wrd	Marker-A	Notes	Code	Description
056	192(A)-0896			
	193(A)-0897			
	194(A)-0898			
	195(A)-0899			
	196(A)-0900			
	197(A)-0901			
	198(A)-0902			
	199(A)-0903			
	200(A)-0904			
	201(A)-0905			
	202(A)-0906			
	203(A)-0907			
	204(A)-0908			
	205(A)-0909			
	206(A)-0910			
	207(A)-0911			
057	208(A)-0912			
	209(A)-0913			
	210(A)-0914			
	211(A)-0915			
	212(A)-0916			
	213(A)-0917			
	214(A)-0918			
	215(A)-0919			
	216(A)-0920			
	217(A)-0921			
	218(A)-0922			
	219(A)-0923			
	220(A)-0924			
	221(A)-0925			
	222(A)-0926			
	223(A)-0927			

3.3.30 Auxiliary Markers (Long 058)

Auxiliary MARKERS [256] (Long 058: 959....928)

wrd	Marker-A	Notes	Code	Description
058	224(A)-0928			
	225(A)-0929			
	226(A)-0930			
	227(A)-0931			
	228(A)-0932			
	229(A)-0933			
	230(A)-0934			
	231(A)-0935			
	232(A)-0936			
	233(A)-0937			
	234(A)-0938			
	235(A)-0939			
	236(A)-0940			
	237(A)-0941			
	238(A)-0942			
	239(A)-0943			
059	240(A)-0944			
	241(A)-0945			
	242(A)-0946			
	243(A)-0947			
	244(A)-0948			
	245(A)-0949			
	246(A)-0950			
	247(A)-0951			
	248(A)-0952			
	249(A)-0953			
	250(A)-0954			
	251(A)-0955			
	252(A)-0956			
	253(A)-0957			
	254(A)-0958			
	255(A)-0959			

3.3.31 Input from CNC (Long 060)

Input from CNC [32] (Long 060: 991.....960)

wrd	In.-CNC	Notes	Code	Description
060	000(D)-0960			
	001(D)-0961			
	002(D)-0962			
	003(D)-0963			
	004(D)-0964			
	005(D)-0965			
	006(D)-0966			
	007(D)-0967			
	008(D)-0968			
	009(D)-0969			
	010(D)-0970			
	011(D)-0971			
	012(D)-0972			
	013(D)-0973			
	014(D)-0974			
	015(D)-0975			
061	016(D)-0976			
	017(D)-0977			
	018(D)-0978			
	019(D)-0979			
	020(D)-0980			
	021(D)-0981			
	022(D)-0982			
	023(D)-0983			
	024(D)-0984			
	025(D)-0985			
	026(D)-0986			
	027(D)-0987			
	028(D)-0988			
	029(D)-0989			
	030(D)-0990			
	031(D)-0991			

3.3.32 Output to CNC (Long 062)

Output to CNC [32] (Long 062: 1023.....992)

wrd	Out.-CNC	Notes	Code	Description
062	000(E)-0992			
	001(E)-0993			
	002(E)-0994			
	003(E)-0995			
	004(E)-0996			
	005(E)-0997			
	006(E)-0998			
	007(E)-0999			
	008(E)-1000			
	009(E)-1001			
	010(E)-1002			
	011(E)-1003			
	012(E)-1004			
	013(E)-1005			
	014(E)-1006			
	015(E)-1007			
063	016(E)-1008			
	017(E)-1009			
	018(E)-1010			
	019(E)-1011			
	020(E)-1012			
	021(E)-1013			
	022(E)-1014			
	023(E)-1015			
	024(E)-1016			
	025(E)-1017			
	026(E)-1018			
	027(E)-1019			
	028(E)-1020			
	029(E)-1021			
	030(E)-1022			
	031(E)-1023			

4. Programming Examples

In this chapter, a few examples of programming the PLC part of Controller are illustrated. Each paragraph will show some typical aspects of the control functions, such as the use of timers, counters, data exchange with the CNC section, etc.

4.1 Typical structure of a program

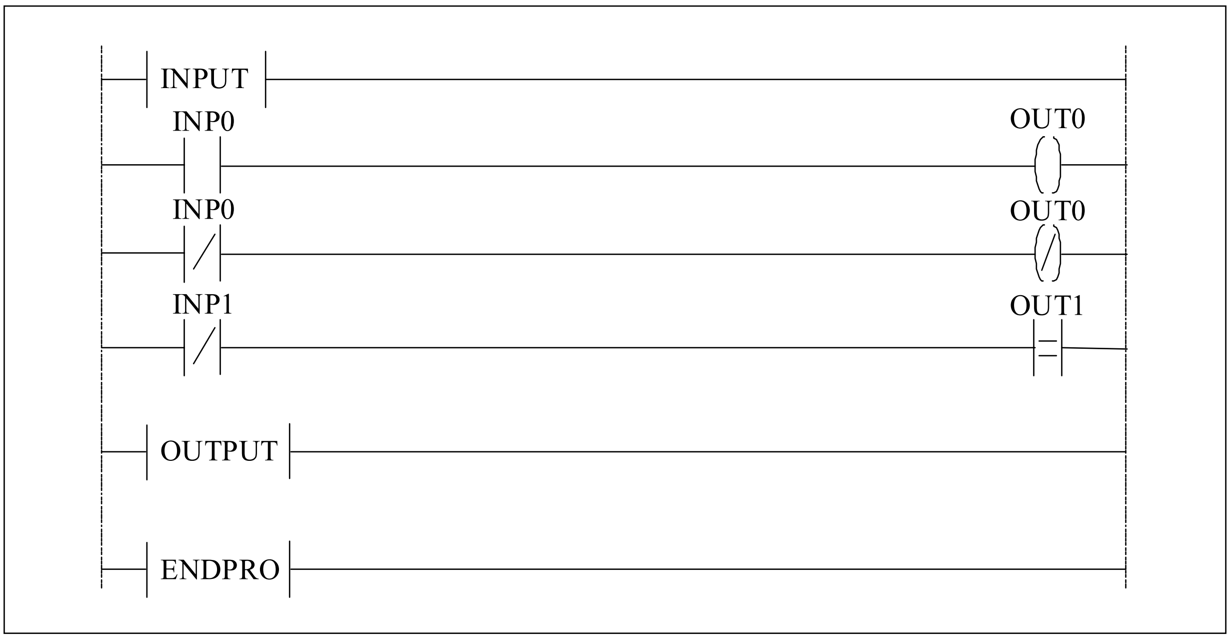
The PLC program is cyclic. The sequence of instructions is repeated continuously from the beginning to the end until a new program is loaded or the machine is switched off. Typically, a program is structured in three main parts: *an INPUT phase* (acquisition of the physical states of the inputs into the internal memory of the PLC), an *execution phase* and an OUTPUT phase (updating the outputs with the states in the image present within the internal memory of the PLC).

The program shown below, has the purpose of illustrating the structure and suggesting guide lines for the programming of the PLC.

```
;DEMO1.PRG
;Demonstration PLC program structure

;The instructions that follow enable the insertion of a program that
;can be common to others or that can contain commonly used
;assignments.

   include demoinc  ;insert the file "demoinc.prg"

;Copies the states of the first 24 inputs into the first 24 ;positions
of the image memory in the PLC
   input 0(i),0(1)
;execution phase.

;The following two blocks copy the state of input 0 to the
;output 0.

  and INP0            ;logic AND between the accumulator and
                      ;INP0<=>0(i).The   result   is   placed   in   the
                      ; accumulator.
       set OUT0       ;OUT0 is set to a logic 1.
  endblo              ;sets the state of the accumulator to 1.

  andnot INP0         ;logic AND between accumulator and the negated
                      ;value of INP0 <=> 0(i). The result is placed in
                      ;the accumulator.
       clear OUT0     ;sets the value of OUT0 to 0.
  endblo              ;sets the state of the accumulator to 1.
```

```
;The following block is logically identical to the two preceding
;blocks. It copies input1 to output 1.

   and INP1         ;logic AND between accumulator and INP1 <=> 1(i).
        equ OUT1    ;sets the value of USC1 to the value of the acc.
   endblo           ;sets the state of the accumulator to 1.

;Copies the state of the image memory to the physical outputs.

   output 0(1),0(o)      ;Update outputs
   endpro                ;End of program
```

Ladder Diagram Version



4.2 Example: using *timers*

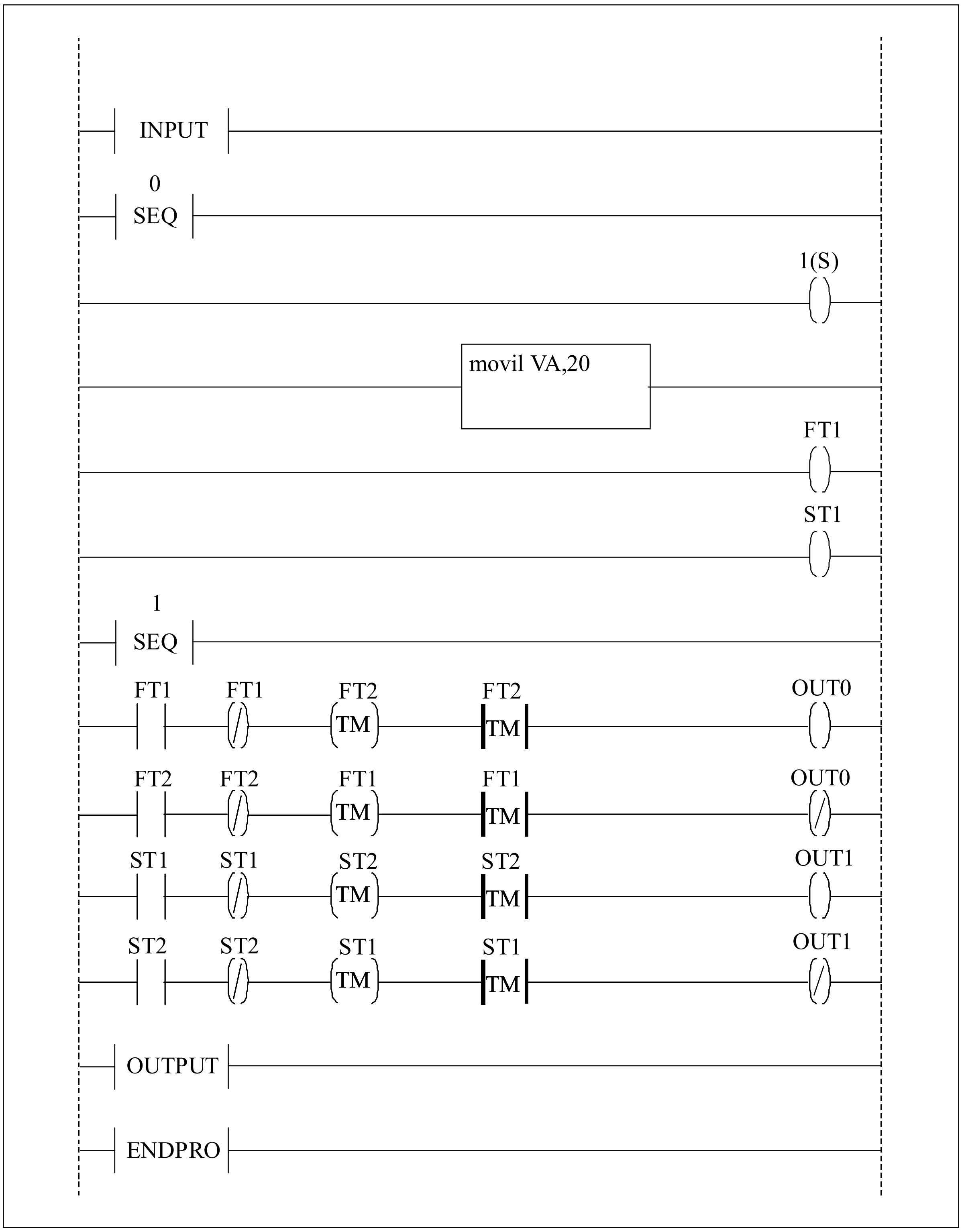
The following program shows how the timers are used, both slow and fast. The timers perform a countdown, starting from the value with which they have been preset. The slow timers decrement every 100ms, so the value is expressed in tenths of a second, while the fast timers decrement every 10ms, so the value is set in hundredths of a second. The timers begin counting only when they are enabled, and when they reach 0, the tag associated with them in the image memory is set to 1.

In particular, the program shows how a square wave of period T can be generated, making use of the timers.
The program also uses *sequences*. Sequence number zero is executed once only on switching on the machine, while the other sequences are executed if the corresponding tag in the image memory is at logic 1. The condition can be summarised in the following statement: sequence number *n* is executed only if the tag *n(S)* in the image memory is at logic 1.

```
;DEMO2.PRG
;Demonstration of using timers

;The following instruction enables the inclusion of a program
;that can be common to others or that can contain commonly used
;assignments.

   include demoinc ;include the file "demoinc.prg"

;The assignments that follow enable the value to be associated
;with a mnemonic name when compiled.

VALTIMER assign 128      ;Variable 128 (32 bit) will count the
                         ;value with which the timer has been
                         ;preset.

FTIMER1 assign 0(t)      ;Fast timer (equivalent to tag 416
                   ;of the image memory)
FTIMER2 assign 1(t)      ; Fast timer(equivalent to tag 417)

STIMER1 assign 16(t)     ;Slow timer(equivalent to tag 432)
STIMER2 assign 17(t)     ;Slow timer (equivalent to tag 433)

;Copies the states of the first 24 inputs into the first 24
;positions in the image memory of the PLC.

   input 0(i),0(1)

;Sequence 0. Is executed once only on start up.
   seq 0
        set 1(s)         ;enables sequence 1

        movil VALTIMER,20    ;presets 20 in VALTIMER

        set FTIMER1  ;sets the tag associated to the timer  to 1.
        Set STIMER1  ;sets the tag associated to the timer  to 1


;Sequence 1. Is enabled by sequence 0
   seq 1

   and FTIMER1                   ;if FTIMER1 has expired
        clear  FTIMER1           ;set the timer FTIMER1 to 0



        settim FTIMER2,VALTIMER  ;FTIMER2 is preset to 20
                                 ;(0,2 sec)
        entim  FTIMER2           ;enables FTIMER2 to count
        set OUT0             ;sets the value of output 0 to 1
   endblo
```

```
and FTIMER2                 ;if FTIMER2 has expired
     clear  FTIMER2         ;sets the timer FTIMER2 to 0
     settim FTIMER1,VALTIMER   ;FTIMER1 is preset to 20
                    ;(0,2 sec)
     entim  FTIMER1         ;enables FTIMER1 to count
     clear  OUT0            ;sets output  0 to 0
endblo

and STIMER1                 ;if STIMER1 has expired
     clear  STIMER1         ;sets timer FTIMER1 to 0
     settim STIMER2,VALTIMER   ;STIMER2 is preset to 20
                    ;(2 sec)
     entim  STIMER2         ;enables STIMER2 to count
     set OUT1               ;sets output 1 to 1
endblo

and STIMER2                 ;if STIMER2 has expired
     clear  STIMER2         ;sets timer FTIMER2 to 0
     settim STIMER1,VALTIMER   ;STIMER1 is preset to 20
                    ;(2 sec)
     entim  STIMER1         ;enables STIMER1 to count
     clear  OUT1            ;sets output 1  to 0.
endblo


output 0(1),0(o)                ;Update outputs

endpro                          ;End of program
```

Ladder Diagram Version

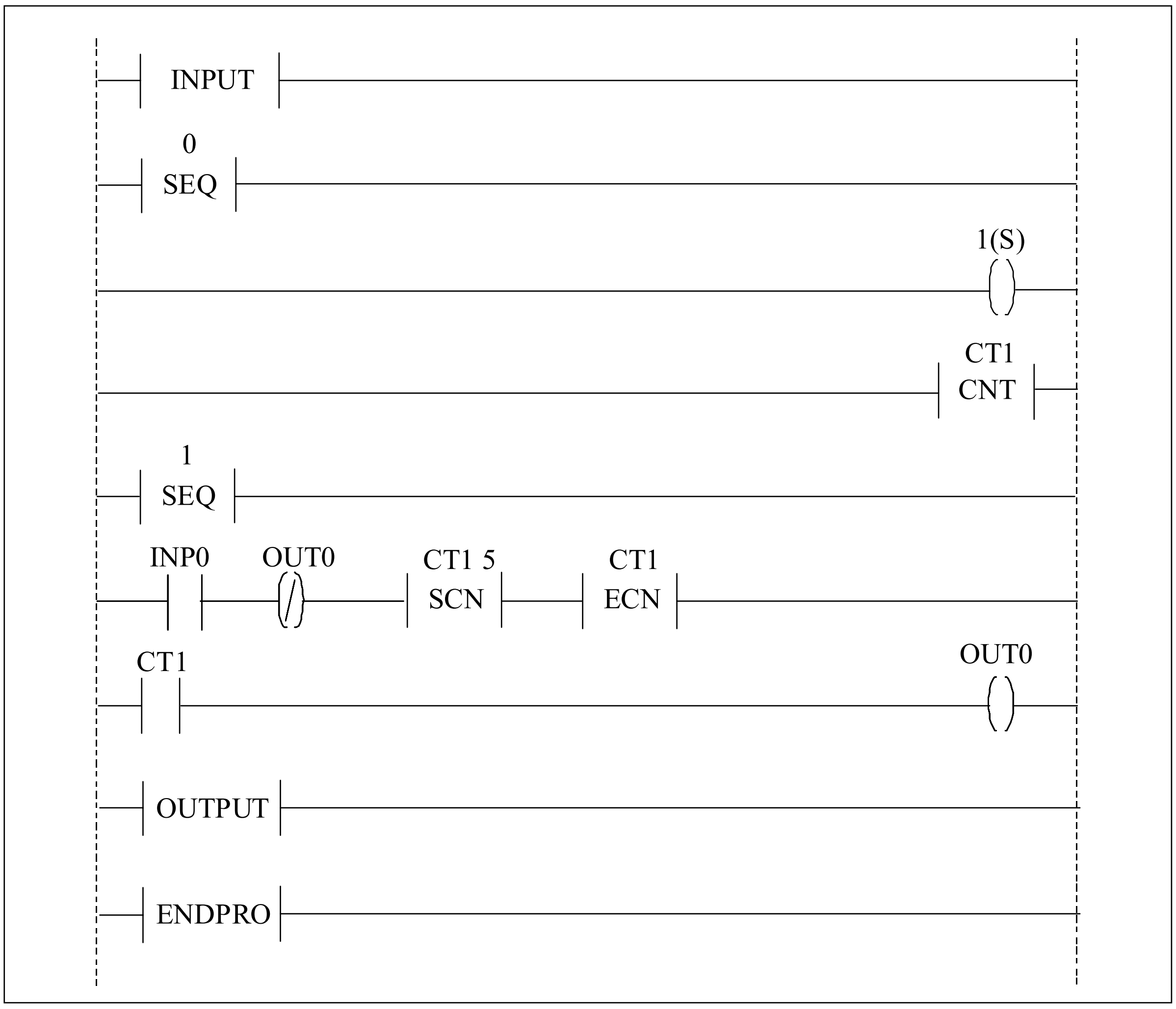


4.3 Example: using counters

The following program shows how counters may be used. Counters count the variations of the logic level at an input. In particular, they count the transitions of the input from a logic 0 to a logic 1. Slow counters are updated every 10ms, while fast counters are updated every 2ms. Counters are initially preset with the number of counts required, after which it counts only if enabled. When it reaches 0, the tag associated with the counter in the image memory is set to 1.

```
;DEMO3.PRG
;Demonstration of using counters

;The following instruction enables a program to be inserted,
;the program can be common to others or can contain commonly
;used assignments.
   include demoinc ;insert the file "demoinc.prg"

;The following assignments enable a mnemonic name to be
;associated to the value when compiled.

FCOUNTER1 assign 0(c)   ;Fast counter(equivalent to position 384
                     ;in the image memory)
FCOUNTER2 assign 1(c)   ;Fast counter(equivalent to position 385)

SCOUNTER1 assign 16(c)  ;Slow counter (position 400)
SCOUNTER2 assign 17(c)  ;Slow counter (position 401)

;Copies the states of the first 24 inputs into the first 24
;positions of the image memory of the PLC.

   input 0(i),0(1)

;Sequence 0. Is executed only once on start up.
   seq 0
        set 1(s)        ;enable sequence 1

        defcnt FCOUNTER1,1:0(1)        ;associates input 1  to the
                                       counter

;Sequence 1. Is enabled by sequence 0
   seq 1

   and INP0                      ;if input 0 is 1
        clear  OUT0              ;set output 0 to zero
        SETCNT FCOUNTER1,5       ;presets the  counter to 5
        ENCNT  FCOUNTER1         ;enables the count
   endblo

   and FCOUNTER1            ;if FCOUNTER1 = 1 has counted to 5
        set    OUT0         ;sets output 0 to 1
   endblo

   output 0(1),0(o)         ;Update outputs
   endpro
```

Ladder Diagram Version



4.4 Example: using mathematical instructions.

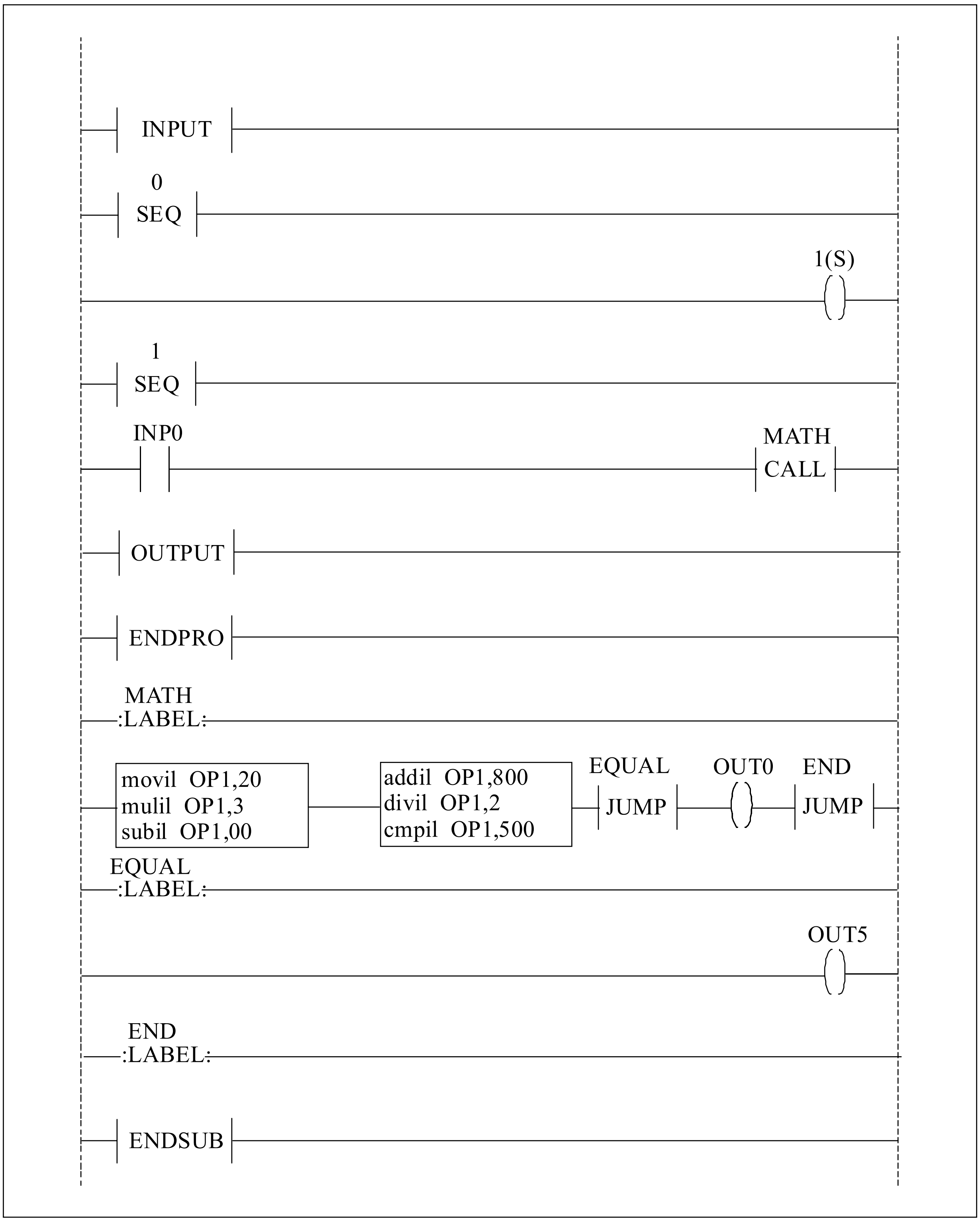
The following program shows how the available mathematical instructions are used. It also illustrates the use of a call for a subroutine and skip to a label.

```
;DEMO4.PRG
;Demonstration of using mathematical instructions, calling a
;subroutine and skipping to a label.

;The following instruction inserts a program that can be common
;to others or that can contain commonly used assignments.

   include demoinc      ;insert the file "demoinc.prg"
   include flagmate          ;insert the file "flagmate.prg"
;The following assignments associate a mnemonic name to the
;value when compiled.
```

```
OP1 assign 128          ;32 bit variable

;Copies the states of the first 24 inputs into the first 24
;positions of the image memory of the PLC.

   input 0(i),0(1)

;Sequence 0. Is executed only once on start up.
   seq 0
        set 1(s)        ;enable sequence 1

;Sequence 1. Is enabled by sequence 0
   seq 1

   and INP0                 ;if input 0 equals 1
        call mathematic ;call mathematical routine
   endblo

;other eventual instructions are executed after the routine

   output 0(1),0(o)         ;Update outputs

   endpro

;**************************************************
;* Definition of routine with label 'mathematic'*
;**************************************************

mathematic

   movil OP1,100   ;OP1=100       OP1 contains 100
   mulil OP1,3     ;OP1=OP1*3 OP1 contains 300
   subil OP1,100   ;OP1=OP1-100   OP1 contains 200
   addil OP1,800   ;OP1=OP1+800   OP1 contains 1000
   divil OP1,2     ;OP1=OP1/2 OP1 contains 500

   cmpil OP1,500   ;OP1 == 500 (updates the mathematical flags)



;if OP1 == 500 the mathematical marker eq(flagmate) is set to 1
;if OP1 <> 500 the mathematical marker eq(flagmate) is set to 0

   jump eq,uguale ;if the flag eq is 1 skip to 'equal'
   set  OUT0      ;if the flag eq is 0 set output 0 to 1

   jump ass,end   ;jump anyway at end ass(flagmate) is always 1)
```

```
;*********************************
;* Label definition ‘equal' *
;*********************************

equal
   set  OUT5 ;set output 5 to 1

;*********************************
;* Label definition 'end'*
;*********************************

end
   endblo

;*********************************
;* End of the routine           *
;*********************************

endsub  ;end of routine 'mathematic'
```

Ladder Diagram Version



4.5 Example: using particular variables and devices.

```
;DEMO5.PRG
;Demonstration: read/write of particular variables
;of different devices

;The following instruction inserts a program that can be common
;to others or that can contain commonly used assignments.

   include demoinc ;include the file "demoinc.prg"

;The following assignments enable the value to be associated
;with a mnemonic name when compiled.

ERRCNC        assign 128
PANNEL        assign 130

INGANA0  assign 132
INGANA1  assign 134

Q1CNC         assign 136
Q2CNC         assign 138
Q255CNC  assign 140

VALTIMER0     assign 142
VALTIMER1     assign 144

VALCOUNTER0   assign 146
VALCOUNTER1   assign 148

SLOT          assign 150
FLAG          assign 152
TIME          assign 154
VANALOG       assign 156

LENREC        assign 158

;Copies the states of the first 24 inputs into the first 24
;positions of the image memory of the PLC.

   input 0(i),0(1)

;Sequence 0. Is executed only once on start up.
   seq 0
        set 1(s)            ;enables sequence 1
```

```
;Sequence 1. Is enabled by sequence 0
   seq 1

   and INP0                      ;if input 0 equals 1
        inpvar ERRCNC,0:0(2) ;acquire CNC error
        inpvar PANNEL,0:0(3) ;acquire CNC panel CNC variable

        inpvar INGANA0,0:0(4)     ;acquire analogue input 0
        inpvar INGANA1,0:1(4)     ;acquire analogue input 1

        inpvar Q1CNC,0:1(5)  ;acquire var. Q1 from CNC
        inpvar Q2CNC,0:2(5)  ;acquire var. Q2 from CNC

        inpvar Q255CNC,0:255(5)   ;acquire var. Q255 from CNC
        inpvar VALTIMER0,0:0(6)   ;acquire value of timer 0
        inpvar VALTIMER1,0:1(6)   ;acquire value of timer 1

        inpvar VALCOUNTER0,1:0(6) ;acquire value of counter 0
        inpvar VALCOUNTER1,1:1(6) ;acquire value of counter 0
   endblo

   and ING1                      ;if input 1 equals 1
        movil  FLAG,0
        outvar 0:0(1),FLAG       ;disables input filter
        movil  TIME,1
        outvar 3:0(1),TIME       ;time between BEL  of 1 second
        outvar 4:0(1),0      ;reset PLC error
        movil  VANALOG,10
        outvar 5:0(1),VANALOG    ;writes to analogue output 0
        movil  LENREC,40
        outvar 7:0(1),LENREC ;initialises record length
                             ;0<lenrec<90
   endblo

   output 0(1),0(o)              ;Update outputs

   endpro
```

4.6 Example: file insertion.

The following two examples represent the format of two programs that define the most commonly used assignments.

```
;DEMOINC
;File of mnemonic name definitions
4.3
INP0  assign 0(i)      ;Input 0  associated with INP0
INP1  assign 1(i)      ;Input 1  associated with INP1
INP2  assign 2(i)      ;Input 2  associated with INP2
INP3  assign 3(i)      ;Input 3  associated with INP3
INP4  assign 4(i)      ;Input 4  associated with INP4
INP5  assign 5(i)      ;Input 5  associated with INP5
INP6  assign 6(i)      ;Input 6  associated with INP6
INP7  assign 7(i)      ;Input 7  associated with INP7
INP8  assign 8(i)      ;Input 8  associated with INP8
INP9  assign 9(i)      ;Input 9  associated with INP9
INP10 assign 10(i) ;Input 10 associated with INP10
INP11 assign 11(i) ;Input 11 associated with INP11
INP12 assign 12(i) ;Input 12 associated with INP12
INP13 assign 13(i) ;Input 13 associated with INP13
INP14 assign 14(i) ;Input 14 associated with INP14
INP15 assign 15(i) ;Input 15 associated with INP15
INP16 assign 16(i) ;Input 16 associated with INP16
INP17 assign 17(i) ;Input 17 associated with INP17
INP18 assign 18(i) ;Input 18 associated with INP18
INP19 assign 19(i) ;Input 19 associated with INP19
INP20 assign 20(i) ;Input 20 associated with INP20
INP21 assign 21(i) ;Input 21 associated with INP21
INP22 assign 22(i) ;Input 22 associated with INP22
INP23 assign 23(i) ;Input 23 associated with INP23

OUT0  assign 0(o)      ;Output 0 associated with OUT0
OUT1  assign 1(o)      ;Output 1 associated with OUT1
OUT2  assign 2(o)      ;Output 2 associated with OUT2
OUT3  assign 3(o)      ;Output 3 associated with OUT3
OUT4  assign 4(o)      ;Output 4 associated with OUT4
OUT5  assign 5(o)      ;Output 5 associated with OUT5
OUT6  assign 6(o)      ;Output 6 associated with OUT6
OUT7  assign 7(o)      ;Output 7 associated with OUT7
OUT8  assign 8(o)      ;Output 8 associated with OUT8
OUT9  assign 9(o)      ;Output 9 associated with OUT9
OUT10 assign 10(o) ;Output 10 associated with OUT10
OUT11 assign 11(o) ;Output 11 associated with OUT11
OUT12 assign 12(o) ;Output 12 associated with OUT12
OUT13 assign 13(o) ;Output 13 associated with OUT13
OUT14 assign 14(o) ;Output 14 associated with OUT14
OUT15 assign 15(o) ;Output 15 associated with OUT15
OUT16 assign 16(o) ;Output 16 associated with OUT16
OUT17 assign 17(o) ;Output 17 associated with OUT17
OUT18 assign 18(o) ;Output 18 associated with OUT18
```

```
OUT19 assign 19(o) ;Output 19 associated with OUT19
OUT20 assign 20(o) ;Output 20 associated with OUT20
OUT21 assign 21(o) ;Output 21 associated with OUT21
OUT22 assign 22(o) ;Output 22 associated with OUT22
OUT23 assign 23(o) ;Output 23 associated with OUT23

;FLAGMATE
;File of mathematical flag definitions (arithmetical markers)

;Tag that always equals logic 1
assassign  248(m)

;Image of the state of the accumulator
accassign  249(m)

;Set to 1 if the comparison
;occurs between two equal variables
eq      assign  250(m)

;set to 1 if the result of the last operation equals zero
z       assign  250(m)

;Set to 1 if the comparison
;occurs between two different variables
neqassign  251(m)

;set to 1 if the result of the last operation is not equal to
;zero
nz      assign  251(m)

;Set to 1 if the comparison between two variables
;where the first is greater than the second.
lt      assign  252(m)

;Set to 1 if the comparison between two variables
;where the first is less than the second.
gt      assign  253(m)

;set to 1 if the result of the last operation is greater
;than the calculation capacity of the PLC
ovfassign  254(m)

;Negated image of the state of the accumulator
noacc   assign  255(m)
```