DUEMMEGI

SISTEMI DI AUTOMAZIONE INDUSTRIALE

CONTATTO

MCP - Programmable Control Module

User's Manual for MCP ver. 4.x and MCP Plus

DUEMMEGI s.r.l. - Via Longhena, 4 - 20139 MILANO
Tel. ++39 2 57300377 - FAX ++39 2 55213686

Rel.: 4.3UK
April 2001

I1- Changes in this manual release 4.3UK in respect to previous release 4.1UK

- This manual refers to MCP Plus too
- MCP address is a number in the range 1 to 255; max number of MCP that can be connected to the same RS485 line is 31
- More details have been added on RAM map of MCP
- PROTOCOL function has been added
- Schematic diagram and description of MCP Plus have been added
- Paragraph on serial ports of MCP Plus has been added
- Updating of technical characteristics
- Overall dimensions of MCP Plus have been added
- The standard (or proprietary) protocol of MCP has been named FXP
- More details on driver implementation have been added
- Appendix B has been added on MODBUS protocol of MCP Plus
- Appendix C has been added with tables for easy localization of virtual point in MCP RAM

I2- Difference among MCP standard (version 4.x) and MCP Plus

MCP Plus is the new programmable control module of the family Contatto DUEMMEGI. The housing is a modular type DIN 9M (as for standard version). The differences in respect of the standard version are the followings:

- The serial display output has been removed
- A serial **RS485 port** has been added to connect more MCP Plus in multi-drop network
- Both RS232 port and RS485 port are **electrically insulated** from other circuits (but no electrical insulation is provided between the two ports); for this feature, no additional external power supply is required
- Beside to MCP standard protocol (named FXP), **MODBUS e JOHNSON CONTROL protocol have been added**, avoiding the connection of external protocol converters

Note: RS232 and RS485 ports are mutually exclusive: the switching from one port to the other occurs through the DSR input signal on RS232 connector.

INDEX

1- INTRODUCTION

MCP is a control module for Contatto bus system. MCP is the monogram of "**M**odulo di **C**ontrollo **P**rogrammabile" (Programmable Control Module), because the handling of the system may be fully defined by the user through some rules, called equations, linking the outputs to the inputs.

To make easy the development of the programs and the setup of the system, a software tool is provided; this tool works on a Personal Computer under WINDOWS® environment and its name is MCPTOOLS. For an explanation about the use of this software tool, please refer to the related documentation.

MCPTOOLS essentially includes:

- a text editor to write the program
- a compiler to allow to translate an ASCII file, containing the operating equations, in a binary file adequate to be transferred in the non volatile memory (FLASH type) of MCP module.
- a simulator to verify the written program, or a part of it, before to transfer it into the MCP memory
- an utility to transfer the program from the PC to MCP (or vice versa)
- a map window to display the status of input and output modules installed in the plant

In the following pages of this manual, the allowable operators and the syntax of the equations will be described; it will be assumed that the user have an adequate knowledge about the basic structure of the Contatto bus system.

***WARNING**: this manual applies to MCP release 4.x and MCP Plus. To verify the release of MCP in your hands, look at the rear of the MCP housing: a yellow label shows a code, e.g. F412599N; the first 3 characters (F41) identify the release number 4.1.*
For a proper use of this release, take in account these few points:

- ***a program written for old MCP releases is still valid and so fully compatible with release 4.x***
- ***a program written for release 4.x may be not fully compatible with old releases***

***IMPORTANT NOTICE**: programs written for MCP release 4.x or MCP Plus must be compiled with the compiler release 4.x integrated in the MCPTOOLS software tool.*

***CAUTION**: this device contains a rechargeable NiCd or NiMH battery: be sure to remove the battery before to throw the device. The battery must be eliminated in a sure way.*

1.1- General Considerations

The Contatto MCP module can process the status and the variations of the inputs to control the outputs or to execute some specific functions, as the pulses counting or the handling of a serial device for the displaying of messages.

1000 virtual outputs are available: instead of the real points, they are not related to any physical field point, but they are only the result of some real or virtual input combinations; this result is stored in the RAM memory of MCP.

The virtual points allow to build some support variables in order to simplify the equations or for special functions. The virtual points may be considered and handled as additional (virtual) inputs, to be used to control real outputs or other virtual points; for this reason, in the following pages, statements as **"virtual input"**, **"virtual output"** and **"virtual point"** will be used as equivalent terms.

The virtual points are mandatory for some specific functions as described in the following paragraphs.

MCP release 4.x allows to define virtual points as function of other virtual points, without nesting limit.

More precisely, **the allowable virtual points are 992**, because the following ones are reserved as here below explained:

- **V1000**: active when MCP detects one or more fault modules (it reflects the status of the MOD.F led on MCP)
- **V999**: active when MCP detects a bus failure condition (it reflects the status of BUS.F led on MCP)
- **V998**: active when MCP ends the initialisation procedure, following a re-programming sequence or a power-on
- **V997**: this points change its status every 0.5 seconds; this point can be used as timebase to implement, e.g., flashing functions, to increment a counter in constant time step, etc.
- **V996**: for future uses
- **V995**: for future uses
- **V994**: when it is active MCP considers, to evaluate the clock functions, the current time (read from the internal timekeeper circuit) + 1 hour; in this way it is possible to change from solar time (V994=0) to summer time (V994=1). When a time request will be made to MCP through the serial port (by MCPTOOLS or other program), the answer will be according to the status of V994
- **V993**: for future uses

These reserved virtual points, with the exception of V994, may be used as "read only" points, so as inputs of equations; typical application is when the presence of a fault module has to be announced through a real output or through the modem to a supervisor system.

***Note**: if V994 virtual point is used to change from solar to summer time, it is mandatory to set the MCP internal clock according to the solar time; in facts, V994 do not change the settings of the internal clock, but it simply adds one hour to the current time when the clock functions are under evaluation, or also when the time is requested to MCP through the RS232 serial port.*

2- EQUATIONS: TYPES AND SYNTAX

The equations can be classified in:

1. Logic equations
   - The output depends by an input or by the combination of more inputs (real or virtual ones), series or parallel connected, eventually complemented
2. Equations with simple operators
   - The basic operators are **RESET** and **SET**, useful for the push-buttons control in START/STOP sequences
3. Equation with special operator
   - The special operator allow the implementation of more complex functions as the pulses counting, the handling of text display, timers, etc.

2.1- Logic equations

Logic equations are the simplest and allow the combinations of inputs to control a real or virtual output.

There is no limits, other than the memory size, to the number of inputs that may be combined each one to other. The Contatto bus system allows a maximum of 1016 real inputs and 1000 virtual points and **all of them can be used to control a real or virtual output**. In addition, **the same input can control more outputs**.

The equations, in their main syntax, may be written as:

```
Ox.y = f ( Ij.k, Vn )

Vn = f( Ij.k, Vn )
```

where **Ox.y** and **Vx** are the controlled output and **f(Ij.k, Vn)** is the combination of real and/or virtual inputs that control the output.

The notation **x.y** and **j.k** (or similar ones used in the following pages) shows the module address (**x** and **j**) and the related point (**y** and **k**). As example, **Ox.y** means the output **y** (in the range 1 to 8) of the module whose address is **x** (in the range 1 to 127). In the same way, **Ij.k** means the input **k** of the module whose address is **j**.

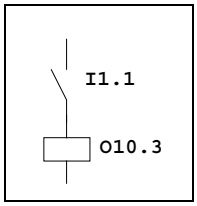
The function defining an output comprehends one or more real and/or virtual inputs, linked together by AND operators (**&** symbol) and by OR operators (**|** symbol); in addition it is possible to complement the logic of an input placing before the symbol **!** (NOT operator).

The **&** operator is equivalent, in the electro-mechanical notation, to the series connection of contacts, while the **|** operator is equivalent to the parallel connection. The function of the NOT operator is equivalent to replace a normally open contact with a normally closed contact.

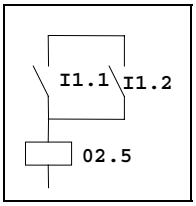
The following examples show the equivalence between a logic combination of MCP and the electro-mechanical diagram.

Examples:

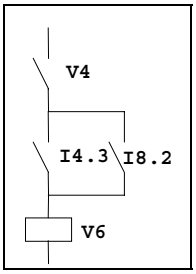
`O10.3 = I1.1` This is the simplest equation, linking an input (1 of module 1) to an output (3 of module 10).



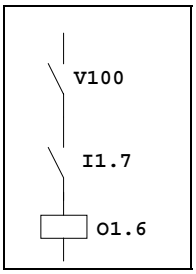
`O2.5 = (I1.1 | I1.2)` This equation represents the parallel connection of two inputs. The brackets, in this case are not mandatory; they may be used for a better interpretation of the equation.



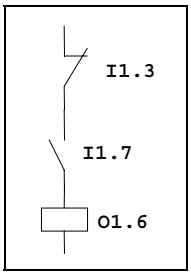
`V6 = (I4.3 | I8.2) & V4` This equation represents the parallel connection of two inputs, both series connected to another input; the output is, as example, a virtual point. In this case the brackets have a well defined meaning and, as described later, they are mandatory.



`O1.6 = V100 & I1.7` This equation represents the series connection of one virtual input and one real input.



`O1.6 = !I1.3 & I1.7` This equation represent the series connection of two inputs, one of them complemented; this means that the normally open contact connected to `I1.3` input, will be instead evaluated as a normally closed contact.

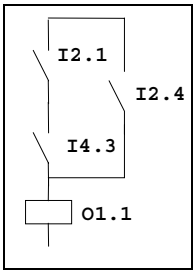


The priority of **&** and **|** operators in the same equation is the following:

1. **&** operator
2. **|** operator

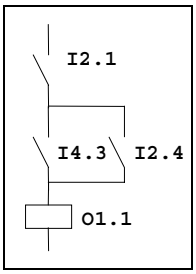
This priority may be modified using the brackets `(` and `)`.

Examples:



`O1.1 = I2.1 & I4.3 | I2.4`

This equation is the parallel connection of two terms:
`I2.1 & I4.3` and `I2.4.`
Note the difference with the following example.



`O1.1 = I2.1 & (I4.3 | I2.4)`

The | is evaluated before the **&** operator. This equation is equivalent to:
`O1.1 = I2.1 & I4.3 | I2.1 & I2.4`
The series term is common to both parallel terms.
Note the difference with the previous example.

During the evaluation of the equation, the current status of the input is considered. As said before, it is possible to complement the input logic, both real and virtual one, using the NOT operator (`!` exclamation mark). In this case the complement of the related input or term will be evaluated.

Examples:

`O1.1 = !I2.4` The output is de-activated when `I2.4` is activated.

`O1.1 = ! (I2.4 & I4.5)` The output is de-activated when both inputs are activated and it is activated when at least one of them is de-activated. This equation is equivalent to: `O1.1 = !I2.4 | !I4.5`.

Note: as shown by last example, the complement of a whole equation, composed by a combination of inputs, is the same as complementing each input and exchanging **&** operator with **|** operators and vice-versa. This rule concerns the boolean algebra and it is always applicable to equations of this type.

2.2- Equations with simple operators

The simple operators are two:

1. `RESET`
2. `SET`

These operators are applicable to real or virtual inputs and they allow to implement a specific function.

The `SET` operator, identified by `S` symbol, and the `RESET` operator, identified by `R` symbol, are useful when a **START-STOP** sequence has to be implemented.
The `SET` operator produces the activation of the output when its related input goes to its active status and the output will be held ON even if the input itself goes back to its steady state (holding the output ON). At the activation of the `RESET` input, the output will be de-activated.

The main equation defining a **START-STOP** sequence is::

```
Ox.y = SIj.k & RIn.m
```

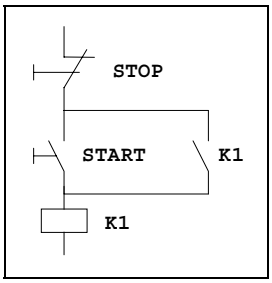
where **Ij.k** is the input producing the activation of the output **Ox.y** and **In.m** is the input causing the switch-off of the same output. Both output and inputs may be also virtual points.

Note: even if in the electro-mechanical diagram the stop push-button (**RESET**) is represented as a normally closed contact, the real contact to be connected to the module input must be a normally open one; for more details, see next examples.

R and **S** operators must be written as prefix to the related input and they may be used together the NOT operator(**!**). In this case **R** and **S** operator must be placed before the NOT symbol.

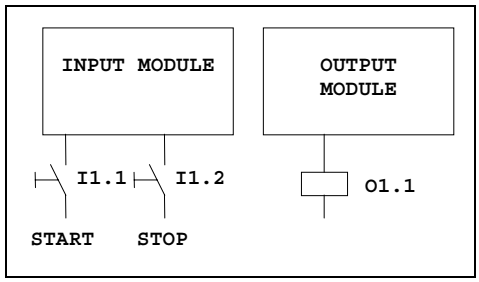
Examples:

Beside figure represents the electro-mechanical equivalent of a START-STOP assembly, made by a relay, a N.O. push-button and a N.C. push-button. The K1 contact keeps the relay ON when the START push-button is released; this condition remains until the STOP push-button is pressed



The equivalent equation for MCP of a START-STOP assembly become:

```
O1.1 = SI1.1 & RI1.2
```



Note, as said before, that both push-buttons connected to the input module have normally open contacts; in this way, the activation of the output occur at the closing of the START push-button, while the de-activation occur at the closing of the STOP push-button. To reverse the logic of the push-buttons, the NOT operator has to be used as in the following example:

```
O1.1 = SI1.1 & R!I1.2
```

In this case the activation of the output occur at the closing of the START push-button and the de-activation occur at the opening of the STOP push-button.

***Note:* & symbol linking SET-RESET operators is mandatory.**

R and **S** operators may be combined to inputs without operator; in this case they act as consents:

```
O1.5 = I2.3 & RI2.1 & SI4.6
```

The input **I2.3** is a consent, because it force OFF the output if it is not activated; note that if the output goes OFF due to the opening of **I2.3**, the output remains de-activated until the closing of **I4.6** together to **I2.3**, exactly as it occurs in the electro-mechanical equivalent.

2.3- Equations with special operators

Special operators allow the execution of some specific functions, both for output control and for the handling of messages through the displays of the family DUEMMEGI DISP. The special operators are:

1. **BINARY CODE GENERATOR**
2. **COUNTER**
3. **SERIAL DEVICE CONTROL (not available for MCP Plus)**
4. **ANALOG THRESHOLD**
5. **TIMER**
6. **PROGRAMMING CLOCK**
7. **ANALOG MODULES CONTROL**
8. **TOGGLE**
9. **16-bit COUNTER MODULE**
10. **16-bit ANALOG INPUT MODULE**
11. **CONFIGURATION**
12. **ADDRESS**
13. **IDENTIFICATION CODE**
14. **EVENTS**
15. **MODEM**
16. **PROTOCOL (MCP Plus only)**

2.3.1- Binary code generator

This operator allows to link 8-bit binary code to the virtual output of the system, and to send it to one or more output module if the related **virtual** point become active. If more than one output are active at the same time, MCP cyclically sends the related binary codes every 2 seconds about (refresh function). If all the virtual points controlling a binary block returns to the steady state, the code 0 (zero) will be sent.

The binary code operator may be used to control a binary input display, connected to an output module (MODPNP) or to directly control a DUEMMEGI DISP-BUS display. Through the virtual points, it is possible to have messages related to a real input or to a combination of real and/or virtual inputs.

Because the code to be sent and the virtual point generating these codes can be specified for each output module, it is possible to handle more than one display, each one with its own messages and independent from the others.

The equation of **BINARY** block, in its main syntax, is the following:

```
BINARY x (  \
      Bn=Vi \
      Bm=Vj \
      ...   \
      )
```

where **Vi** must be specified by proper equations. The "\" symbol, which meaning is that the equation follows on the next line, is mandatory (see chapter about the equation writing). The **BINARY** key word identifies the block start and it is followed by the address **x** of the output module to which the binary code will be transferred. Inside the brackets **(** and **)** are enclosed the terms **Bn=Vi**, defining the binary code **n** to be sent when the related virtual point become active. The binary code can assume the values in the range 1 to 255.

Note: the terms `Vi` on the right side of the equal mark **cannot be combined together other terms or preceded by NOT (!) operator**. Each virtual point can be declared only once in the same **`BINARY`** block (this means that an input cannot produce more than one binary code on the same output module), while a binary code may be repeated more times (many virtual points can produce the same code on the same output module); see next example for more details.

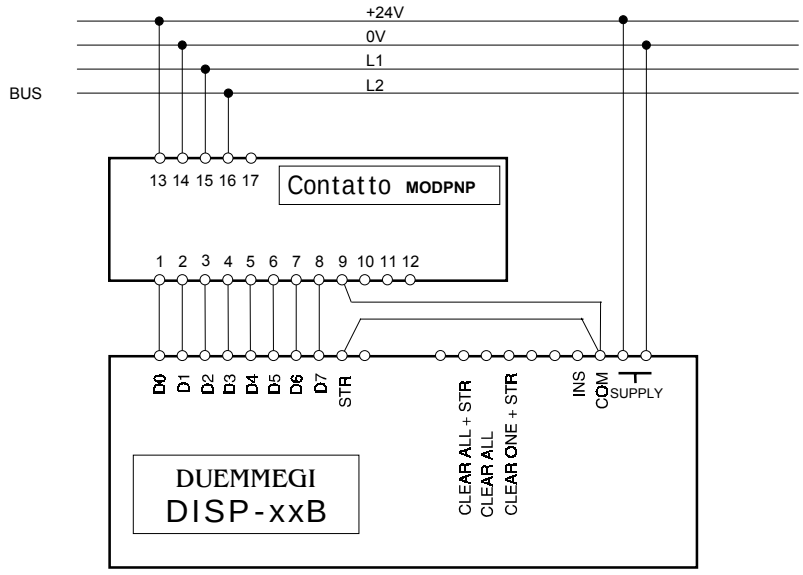
Example:

```
V1 = I1.1
V3 = I1.1 & I1.3
V5 = (I1.5 & I2.3) | I2.1
V8 = RI6.2 & SI4.6
```

`BINARY 4 ( B1=V1 B2=V3 B3=V5 B26=V8 )` 4 virtual points produce 4 distinct codes. Note that the new line symbol (\) is not used because the equation is written on a single line.

`BINARY 12 ( B1=V1 B2=V3 B1=V5 B26=V8 )` Two virtual points produce the same binary code (**`V1`** e **`V5`**).

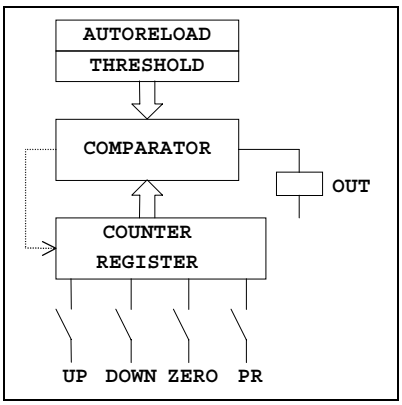
The binary display **DUEMMEGI DISP-XXB** (XX depends on the model, or how many messages have to be displayed) must be connected to the MODPNP output module as shown in the following schematic diagram.



For more information on the bus display **DUEMMEGI DISP-BUS**, refer to related user's manual.

2.3.2- Counter

Some memory locations of MCP can be defined as counters of pulses applied to some inputs of the system. Depending on the counting value and a well defined value, called threshold, an output can be controlled as shown by the following block diagram.



The counter may be controlled, by up to four inputs, each one with a well defined function:

1. up counting input (**U**, up)
2. down counting input (**D**, down)
3. reset input (**Z**, zero)
4. input to load the counter with a specified value (**P**, preset); more than one preset input may be defined with different loading values

The counter, depending on the variation of its inputs, is updated and then compared to the threshold value to control the output.

The output and inputs of a counter may be both real and/or virtual points. The counter function allows three compare functions:

1. Output OFF if counter value < threshold and output ON if counter value >= threshold.
2. Output ON if counter value <= threshold and output OFF if counter value > threshold.
3. Output ON if counter value = threshold and output OFF if counter value <> threshold.

Each counter can be defined as 8 or 16-bit data, allowing a counting in the range 0 to 255 in the first case and 0 to 65535 in the second one. Both 8 and 16-bit counter can be contemporaneously used in the same program, but the total number of them must comply the following rule:

(number of 8-bit counter) + 2 x (number of 16-bit counter) <= 512

The syntax of the equation for a counter function is the following:

```
Oa.b = Cn {op} T,R U[s1]Ic.d & D[s2]Ie.f & ZIg.h & P[p1]Ii.m & Oq
```

where:

Oa.b is the controlled output (may be real or virtual output)

Cn {op} T is the comparing rule between the counter value and the threshold **T**; **n** may assume values from 0 to 511 (see in the following pages for the meaning of this parameter). **{op}** may be one of the following symbols:

- = (equal)
- > (greater or equal)
- < (lower or equal)

T is a numerical value in the range 0 through 255 for the 8-bit counters and in the range 0 through 65535 for 16-bit counters. In this last case the **T** value must be preceded by the **symbol #, expressly declaring that the counter is a 16-bit type**. **T** may be a pointer to another counter register; in other words, the **threshold value may be defined as the contents of another counter** (see in the following pages for details).

`R` is the (optional) value for the autoreset and/or autoreload functions: when the up-counting reach the `R` value, the counter will be automatically reset, or when the down counting goes below zero, the counter will be reloaded with the value `R-1`. The value of `R` may be a pointer to another counter register; in other words, the **autoreset/autoreload value may be defined as the contents of another counter** (see in the following pages for details).
Note: if the autoreset/autoreload value is not specified, the counting will be stopped to 0 (when down-counting) and to the maximum allowed value (when up-counting) avoiding the underflow/overflow of the counter.

`U[s1]Ic.d` defines the input that increases (**U**P) the counter register by an amount `[s1]`; in other words, `s1` **is a number in the range 1 to 32** and it is the step of increment of the counter. **The step is an optional parameter and, if specified, must be surrounded by square brackets**; if step parameter is not specified, it will be assumed equal to 1.

`D[s2]Ie.f` defines the input that decreases (**D**OWN) the counter register by an amount `[s2]`; in other words, `s2` **is a number in the range 1 to 32** and it is the step of decrement of the counter. **The step is an optional parameter and, if specified, must be surrounded by square brackets**; if step parameter is not specified, it will be assumed equal to 1.

`ZIg.h` defines the input that reset the counter (**Z**ERO)

`P[p1]Ii.m` defines the input that loads the counter (**P**RESET) with value `p1`; the preset value `p1` is a number in the range:
1 to 255 for 8-bit counters
1 to 65535 for 16-bit counters
The preset value, if specified, must be surrounded by square brackets and it cannot be equal to zero; it is possible to define more preset inputs (see following examples).

`Oq` defines a **real** output module, whose address is `q`, to which the counter contents will be transferred. If the counter is defined as 16-bit type, only the most significant byte will be transferred to the output module. **It is possible to specify more outputs** where the counter value will be transferred.

The `&` symbol, linking the terms of the equation, is mandatory and it is the only allowed operator in a counter definition.

Up, Down, Zero and Preset parameters may be specified if requested. Preset parameter, as said above, may be specified more than once in the same equation, using distinct inputs, so that to have several loading values.
The counting, up or down, the reset and the preset normally occur on the OFF→ON transition of the input. To use the ON→OFF transition, the NOT (`!`) operator may be placed before the related input.

Output and inputs of a counter may be both r**eal and/or virtual points, with the exception of `Oq` terms that can be real points only**.

The counters of MCP release 4.x (or higher) must be numbered (e.g. `C1, C2, C3,` etc.). In this way the address of the register related to a counter, in the RAM memory of MCP, may be easily found. **When a number is assigned to a counter, take in account that a 16-bit counter get 2 memory cells**; in other words, a 16-bit counter will reserve both the register number that was explicitly assigned to it, and the next one (so this last number cannot be used to define another counter). The following example shows this last concept:

```
V1 = C0 > 10 UI1.1 & DI1.2          // 8-bit counter C0 (1 register,
                                    // called C0 as explicitly declared)
V2 = C1 > #300 U I1.3 & D I1.4      // 16-bit counter C1 (2 registers,
                                    // called C1, as explicitly declared,
                                    // and C2 as hidden meaning)
V3 = C3 > 128 UI1.5 & DI1.6         // 8-bit counter C3 (1 register,
                                    // called C3 as explicitly declared);
                                    // this counter cannot be called C2 because
                                    // this register was already used by the
                                    // counter defined in the previous equation
```

It is not mandatory to number the counter sequentially, and it is not forbidden to leave unused numbers between a counter and another one.

As said above, MCP release 4.x (or higher) **allows to define the threshold (T) and/or the autoreset/autoreload (R) values as the contents of other counters**; in this way it is possible to define **counters with variable threshold** (and/or variable autoreset). The proper threshold can be set by a supervisor (writing in its relevant register), or by an UP and DOWN control of the counter containing the threshold value.
For 16-bit counters (2-byte), the lower identification number register contains the most significant byte, and next register contains the least significant byte. E.g., if C4 was defined as 16-bit counter, then register C4 contains the MSByte and C5 the LSByte; the supervisor, to change the counter value, will write both registers.

Following examples explain the use of counter registers.

Example 1:

```
O1.1 = C1 > C2,C3 UI1.1 & DI1.2
```

The contents of counter C1 is compared with a threshold value that is the contents of counter C2. Counter C1, in addition, increases at any OFF→ON transition of input I1.1 and decreases at any OFF→ON transition of input I1.2. Output O1.1 will be switched ON or OFF depending upon the comparing result. Moreover, when the counter C1 reaches the value contained in register C3, it will be reset; if C1 goes below zero, it will be reloaded with the value contained in register C3 decreased by a unit.
The supervisor (Personal Computer or other device), through the serial port of MCP, may write at RAM addresses of counter C2 and/or C3 to set or change the threshold and the autoreset/autoreload values. Note that if the two parameters are handled by the supervisor only, it is not needed to define C2 and C3 in MCP program.

Example 2:

```
O1.1 = C1 > C2 UI1.1 & DI1.2
V1 = C2 > 1 U[10]I1.3 & D[10]I1.4
```

The contents of counter C1 is compared with a threshold value that is the contents of counter C2; this last one is incremented, by step 10, by I1.3 and decreased, by step 10, by I1.4.
C2 may be however controlled by a supervisor.
V1 and the value 1 assigned to C2 threshold are used, in this example, only to define the counter C2. In other words, V1 may not be used as input of other equations in MCP program and the threshold value may be a "fictitious" value, because any other value in the range 0 to 255 do not modify the program operation (the same thing is true for the comparing rule).

In the following, some other examples show the use of counters.

Examples:

```
O1.1 = C1 < 10 UI1.1 & DI1.2        // O1.1 will be switched ON when the counter
                                    // value become lower or equal to 10. The
                                    // increasing and decreasing step is 1


V34 = C2 > 125 U[25]I1.1 & DI1.2    // V34 will be switched ON when the counter
                                    // value become greater or equal to 125. The
                                    // increasing step is 25 and the decreasing
                                    // step is 1


O78.5 = C8 > #312 UV1 & DV2 & P[300]V998 // the 16-bit counter will be preset
                                         // to 300 when the virtual point V998
                                         // become active, so at the startup of
                                         // MCP (see the meaning of V998 at
                                         // page 4 of this manual

O1.4 = C3 > 10,15 UI1.1 & DI1.2     // The counter activates the output when the
                                    // counter value is greater than 10 and the
                                    // counter itself will be autoreset when the
                                    // value overrides 14. When down-counting,
                                    // the counter will be reloaded with 14 when
                                    // the counter goes below zero.
```

Following example shows as it is possible to implement a semaphore control for a parking, where the total car places may change by the supervisor writing into register `C2-C3` (16-bit).
WARNING: always remember that a 16-bit counter takes 2 RAM registers, so the supervisor, to change the threshold value, has to write both `C2` and `C3` locations.

```
V1 = C2 > #1 P[312]V998                 // preset C2 (16-bit) to 312 (only at
                                        // MCP reset)
V2 = C0 > #C2 UI1.1 & DI1.2 & ZI1.3     // compare C0 (16-bit) and the
                                        // contents of C2
O1.1 = V2                               // switch ON O1.1 if C0>= C2 (red
                                        // light)
O1.2 = !V2                              // switch ON O1.2 if C0 < C2 (green
                                        // light)
```

Following example shows as it is possible to control two dimmers to regulate the lighting power of some lamps. Two pushbuttons (UP and DOWN) increase and decrease the value of a counter (with step 5) and the result will be transferred to the analog output modules `O1` e `O2`, that control two electronic dimmers. By other two pushbuttons connected to `I1.4` and `I1.5`, it is possible to set the lighting power to some well defined preset values (e.g. 30% and 60% of the maximum value). Pressing a pushbutton connected to `I1.6` it is also possible reset the counter and so switch OFF the lamps. Note that `V1` and the value 255 are "fictitious" and they are here used only to define the counter.
WARNING: the preset value cannot be zero, and the step value must be in the range 1 to 32. Note that more than one preset value may be specified in the same equation.

```
V1 = C1 > 255 \
     U[15]I1.1 & D[15]I1.2 & \   // up step 15 by I1.1 and down step 15 by I1.2
     P[85]I1.4 &  \              // preset 85 when I1.4 is activated
     P[170]I1.5 & \              // preset 170 when I1.5 is activated
     ZI1.6 & \                   // reset the counter when I1.6 is activated
     O1 & O2                     // transfer the counter contents to O1 and O2
```

2.3.3- Serial device control

This function is allows the control of a serial display (DUEMMEGI DISP-S) connected to the proper RS232 serial port on MCP; a printer can be connected to the display. Depending on the variation of well defined inputs, it is possible to send proper commands to the display to recall alarm and/or status messages. This function also provides the handling of an output (normally to drive a siren when an alarm occur) and two inputs (acknowledge and reset push-buttons).
WARNING: this function cannot be applied to MCP Plus.
The equation defining this function is the following:

```
SERIAL ( \
      ACK = Vn \
      RST = Vm \
      SIREN = Vp \
      DEFAULT = PRN MEM SIREN \
      Z001 = Vx1 options \
      Z002 = Vx2 options \
      ... \
      )
```

The "\" symbol, which meaning is that the equation follows on the next line, is mandatory (see chapter about the equation writing).

The **SERIAL** keyword defines the start of the information block that must be enclosed between the round brackets **(** and **).** The three parameters for the alarm handling (**ACK**, **RST** and **SIREN**), defining the inputs where the acknowledge and reset push-buttons are connected to, and the output for the siren control are the first ones in the block. This parameters are optional and then can be omitted.

The **DEFAULT** line defines the three options that can be **PRN** or **NOPRN**, **MEM** or **NOMEM** e **SIREN** or **NOSIREN** (upper or lower case characters do not care). These default values defines what options has to be assumed in the following lines, **unless otherwise specified**. If the **DEFAULT** lines is not specified, the default values will be **MEM, PRN** and **SIREN.**

The code list to be transmitted to the display follows (number of the message) together to the related virtual input generating that code (with the proper options if they are different from the default values).

It is mandatory that all the points defined into the SERIAL block be virtual points; this virtual points may be defined before linking them to inputs or a combination of inputs as previously described (with the exception of V999 and/or V1000, see paragraph 1.1); In addition, the **Vi** terms on the right side of the equal mark cannot be combined to other terms, but they can be preceded by the NOT operator (**!**) as shown in the example at the end of this paragraph.

When a virtual point included in the **SERIAL** block become active, the related numerical code will be transmitted to the display and the actions connected to the options defined for that point will be executed:

1. The point will be inserted in the refresh queue and stored.
2. The print command will be transmitted if **PRN** option is specified, otherwise no command will be sent (**NOPRN**).
3. The siren output will be enabled if the **SIREN** option is specified. The option **NOSIREN** will not enable the siren.

The **MEM** option (or **NOMEM**) defines the operation of the sequence when the related virtual point returns to its steady state. If **NOMEM** option was specified, the point will be removed from the refresh queue, else the point remain in the queue and a reset command only can remove it.

MCP cyclically refresh the display with the codes related to the active and/or stored virtual points until the queue is not empty (automatically if **NOMEM** was specified or manually through the reset push-button if **MEM**). In this case the code 000 will be transmitted to the display. The reset sequence is subordinated to the acknowledge of the alarm through the ACK push-button.

Resuming and referring to the above described general syntax, the symbol meanings are the following:

Vn The acknowledge virtual input
Vm The reset virtual input
Vp The siren virtual output
Vx1 The virtual point linked to the message whose code is ESC Z 001 (recall of message 1)
Vx2 The virtual point linked to the message whose code is ESC Z 002 (recall of message 2)
options The option list for the related message (**MEM** or **NOMEM**, **PRN** or **NOPRN**, **SIREN** or **NOSIREN**):
MEM: the message will be stored until a reset occurs
NOMEM: the message will be removed as soon as the related input returns to its steady state
PRN: the message will be sent to the printer too
NOPRN: the message will not be sent to the printer
SIREN: the input will activate the siren
NOSIREN: the input will not activate the siren

DEFAULT The default options to be assumed unless otherwise specified

Notes:
1. The lines defining the ACK, RST, SIREN and DEFAULT are optional.
2. If the default values are not specified, the following options will be assumed: **MEM PRN SIREN**.
3. The virtual points related to ACK and RST inputs, if required, must be specified by proper equations.
4. The virtual point related to the siren may be linked to a real output.
5. The virtual points related to the messages must be specified by proper equations.
6. The acknowledge and the reset of the alarms are executed as specified by the ISA-M alarm sequence; for this reason the alarm, if not previously acknowledged, cannot be reset.
7. If during the reset some alarms are present again, they will be displayed and the siren will be restarted.
8. The NOT (!) operator can be placed before the virtual points. In this case the state of the related input will be complemented.
9. It is possible to recall a message also when a module failure or a bus failure condition occurs, using the virtual points V999 and V1000 (see paragraph 1.1)

Example:

```
V100 = I1.1                    Acknowledge input
V200 = I1.2                    Reset input
O1.1 = V300                    Real output for the first siren (e.g. local siren)
O2.1 = V300                    Real output for the second siren (e.g. remote siren)

V1 = I2.1                      Definition of the input for message 1
V2 = I2.2 | I2.3 | I2.4        Definition of the input for message 2
V3 = I2.5 & I2.6               Definition of the input for message 3
V4 = CLOCK( 12:00, 13:00)      Definition of the input for message 4 (the meaning of CLOCK
                               keyword will be described in a following paragraph)

SERIAL ( \                     Start of SERIAL block
  ACK = V100 \                 Acknowledge command
  RST = V200 \                 Reset command
  SIREN = V300 \               Siren control
  DEFAULT = MEM NOPRN SIREN \  Default options: alarms storing and no siren activation
  Z001 = V1 \                  Alarm 1
  Z002 = !V2 \                 Alarm 2 (complementary logic)
  Z003 = V3 PRN \              Alarm 3 with printing
  Z004 = V4 NOMEM NOSIREN \    Status message (because it is not stored and without siren
                               activation)
)                              End of SERIAL block
```

2.3.4- Threshold

The **THRESHOLD** operator allows to link a digital output point to an analog input module and to control that output as function of the value of the analog input signal and of a configurable threshold. MCP release 4.x (or higher) allows to define variable threshold and hysteresis.
The value of the input signal is compared to the threshold and the result sets the output status. The output may be activated if the input is greater or equal to the threshold or if it is lower or equal to the threshold.
The syntax is the following:

```
Oa.b = Aj {op} T,H
```

where:

Oa.b is the output controlled by the threshold function (may be a virtual point too)

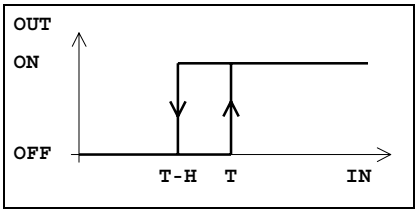
Aj identifies the analog input module whose address is **j**

{op} is the comparing rule: > (greater or equal to) or < (lower or equal to)

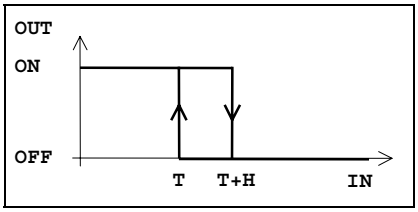
T is the threshold value (0 ÷ 255) or a pointer to a counter register containing the threshold value

H is the hysteresis value (0 ÷ 255) or a pointer to a counter register containing the hysteresis value

The meaning of the hysteresis value depends on the comparing rule (< or >) as here below described:



> the output will be switched on when the input is greater or equal to the threshold (**T**) and it will be switched off when the input is lower or equal to the threshold decreased by the hysteresis (**T - H**)



< the output will be switched on when the input is lower or equal to the threshold (**T**) and it will be switched off when the input is greater or equal to the threshold incremented by the hysteresis (**T + H**)

The threshold is useful, if not mandatory, in all cases where it is necessary to avoid continuous switching of the output when the analog input value is near to the threshold (e.g. thermo regulation through a thermostat, level control, etc.). **If the hysteresis value is omitted, MCP will assume it equal to zero**.

More terms of a threshold function may be linked together through the AND (**&**) and OR (**|**) operators.

T and **H** values must be in the range 0 through 255 (full scale value of the analog input module of the Contatto family).

Examples:

O1.1 = A1 > 240,2	Signaling of over threshold value with hysteresis set to 2; the output will be switched on when the input value is greater or equal to 240 and is switched off when the input value is lower than 238.
V2 = A1 < 40,2 | A2 <30,4	Signaling of under threshold value of two analog inputs. The output (e.g. virtual one) will be switched on when the value returned by the module 1 falls below 40, or when the value returned by the module 2 falls below 30; regarding the hysteresis values, the above explained rules are still valid.
O1.4 = A1 < 128 & A1 > 30	The output will be switched on when the input value is both greater than 30 and lower than 128 (in other words when the input value is in the range 30 through 128).

As said above, **MCP release 4.x (or higher) allows to define, as threshold and/or hysteresis value, the contents of a counter register**; in this way it is possible to define threshold functions with variable parameters. The proper threshold and/or hysteresis can be set by a supervisor (writing in its relevant register), or by an UP and DOWN control of the counter containing the parameter. **The counter register used in a threshold function may be 8-bit type only**. The following examples show the use of threshold functions with variable parameters.

Examples:

In the following example, output **O1.1** is ON when the value returned by the analog input module **A1** is greater than the contents of counter register **C1** and lower than the contents of counter register **C2**. The hysteresis values, in this example, are fixed to 2.

At MCP reset (**V998** change from OFF to ON), the counter **C1** is preset to 64 and **C2** to 128; at this point, a supervisor (PC or other device) may change the threshold values writing to the related register in MCP RAM. **V1**, **V2** and the value 1 in the counter equations are "fictitious" and they are here used only to define the counters to be preset to the specified values.
Remember that counter register in a threshold function may be 8-bit type only.

```
V1 = C1 > 1 P[64]V998
V2 = C2 > 1 P[128]V998
O1.1 = A1 > C1,2 & A1 < C2,2     // O1.1 ON when the analog value returned by A1
                                 // is in the range specified by C1 and C2
                                 // contents
```

In the following example, bot the threshold value and the hysteresis value are stored in two counter registers (**C4** and **C8**). Because in this example the preset function is not requested, and the two registers are exclusively controlled by the supervisor, then it is not necessary to include in the program two equations that define the counters **C4** e **C8**.

```
V100 = A22 < C4,C8
```

In the following example, output **O2.3** is ON when **A10** is greater than the contents of counter register **C20**; this last one is increased by **I1.1** and decreased by **I1.2**.

```
V34 = C20 < 1 UI1.1 & DI1.2
O2.3 = A10 > C20,5
```

2.3.5- Timer

The timer function allows to control a digital output point (both real and virtual one) through a configurable delay in respect to the command signal. This delay can be related to the activation, the de-activation or both ones, and it can be in the range 0 through 6553,5 seconds (1 hour and 50 minutes about) with 0.1 seconds resolution.
Up to 256 independent timers may be defined. The syntax defining a timer function is the following:

```
Oa.b = TIMERn (Ij.k,e,d)
```

or:

```
Oa.b = TIMERn (TIj.k,0,d)
```

or:

```
Oa.b = TIMERn R(TIj.k,0,d)
```

where:

Oa.b is the (real or virtual) output controlled by the timer function
n is the **optional** identification number of the timer (in the range 0 to 255); for more details on this parameter, see in the following pages
Ij.k is the (real or virtual) input controlling the timer function
e is the output activation delay, expressed as hundreds of milliseconds (0.1 seconds); this delay is evaluated by MCP starting from the transition of the input point **Ij.k.** The activation delay may be a **number in the range 0 to 65535 or a pointer to a 16-bit counter register** containing the delay value
d is the output de-activation delay, expressed as hundreds of milliseconds (0.1 seconds); this delay is evaluated by MCP starting from the transition of the input point **Ij.k.** The de-activation delay may be a **number in the range 0 to 65535 or a pointer to a 16-bit counter register** containing the delay value
T is a symbol identifying the **monostable** timer function; when this symbol is placed before the input, the timer acts as a monostable. In other words it generates a pulse on the related output, whose

duration is equal to **d**, triggered by OFF→ON transition of the input point (or by ON→OFF transition if the NOT (**!**) symbol is placed before the input point). In a monostable timer the time **e** will not be considered and it will be assumed equal to zero

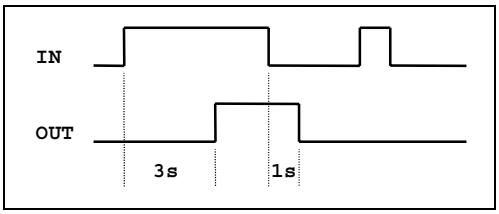
R is a symbol identifying a **retriggerable monostable** timer function; when this symbol is placed before the first bracket of a monostable timer function, the output pulse has a duration specified by **d** parameter starting from the last OFF→ON transition of the input point **Ij.k** (or OFF→ON if the NOT (**!**) symbol is placed before the input point). Note that **R** symbol may be used in monostable timer function only

The real or virtual input point **Ij.k** may be preceded by the negation symbol **!** (NOT). In this way, it is possible to complement the input logic.
If the input point (Ij.k) is preceded by T symbol, the timer acts as a monostable; in other words it generates a pulse on the related output, whose duration is equal to **d** (in this case the time **e** will not be considered and it will be assumed equal to zero).

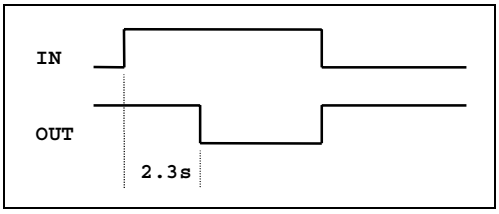
Number n is not an obligatory parameter; it may be used to find in a easy way the timer registers in the RAM memory of MCP when a supervisor control is required (read and write of timer registers).

Examples:

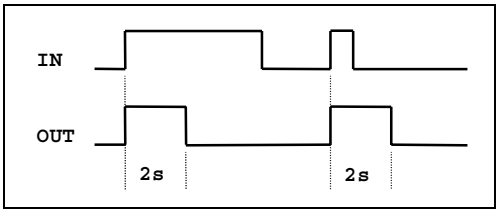
O1.1 = TIMER(I2.5, 30, 10) 3 sec activation delay and 1 sec de-activation delay; note that if the input is active during a time lower than the activation delay, the output is not affected.



V23 = TIMER(!I1.1, 0, 23) Output complementary to the input, no delay at the activation, 2.3 sec de-activation delay.

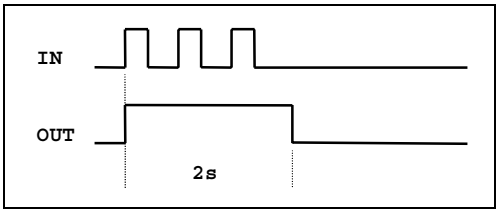


O1.1 = TIMER (TI1.1, 0, 20) 2 sec pulse when **I1.1** switches on; when the input switches off, nothing occurs. Note that the output pulse is generated also if the input remains in the active state during a time lower than the duration of the pulse itself.

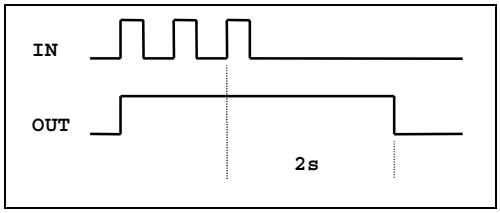


Note: regarding the monostable function, the pulse start input will be ignored during all the duration time of the pulse itself as here beside shown (in other words the monostable is **not retriggerable**). The equation is again:

O1.1 = TIMER (TI1.1, 0, 20)



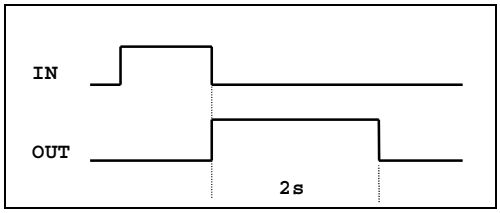
The figure here beside shows the operation of a **retriggerable monostable** timer; the equation, in this case, is:




```
O1.1 = TIMER R (TI1.1, 0, 20)
```

The pulse duration is 2 sec as in previous example, but this duration is evaluated starting from the last OFF→ON transition of the input.

The input negation of a monostable timer, generates a positive pulse triggered on the ON→OFF transition as shown here beside. The equation is:




```
O1.1 = TIMER (T!I1.1, 0, 20)
```

As said before, MCP release 4.x (or higher) **allows defining the contents of a counter register as the value of the activation and/or de-activation delay**; in this way it is possible to implement timer functions having **variable delays** that can be set by a supervisor system (writing the required value in the related RAM register). Delay parameters may be set through UP/DOWN control of the counter containing the delay values. **Counter registers used in timer functions must be exclusively 16-bit type**.

Following examples show the use of timers with variable delays.

Examples:

In the following example, output `O1.1` will be activated after a delay equal to the contents of counter register `C0` and will be de-activated after a delay equal to the contents of counter register `C2`. At MCP reset (V998 active), counter `C0` will be preset to 20 (2 seconds delay) and `C2` to 40 (4 seconds delay); at this point, a supervisor system may change the delay values writing to RAM proper register (`C0-C1` and `C2-C3`). `V1, V2` and value `#1` in the two counter equations are fictitious and they are here used only to define the counters to be preset to the specified values. Remember that counter register in a timer function may be 16-bit type only; this means to place the symbol `#` before the threshold in the counter equation.
WARNING: a 16-bit counter takes two RAM positions.

```
V1 = C0 > #1 P[20]V998
V2 = C2 > #1 P[40]V998
O1.1 = TIMER(I1.1, C0, C2)
```

In the following example, the delay values are stored in two counter registers (respectively `C4` and `C6`). Because in this example the preset function is not requested, and the two registers are exclusively controlled by the supervisor, then it is not necessary to include in the program two equations that define the counters `C4` e `C6`.

```
V100 = TIMER(V8, C4, C6)
```

In the following example, output `O2.3` will be activated after a delay equal to the contents of counter register `C20`; this last one will be increased by `I1.1` and decreased by `I1.2`. The de-activation delay is fixed to 5 seconds. Note that `C20` is 16-bit counter, because it controls a `TIMER` function.

```
V34 = C20 < #1 UI1.1 & DI1.2
O2.3 = TIMER(I2.1, C20, 50)
```

2.3.6- Programming clock

The clock function allows to control an output according to a configurable schedule. The programming can be **both daily and weekly**, and in this last case the week day has to be specified in the equation.
The following syntax may be used:

```
Ox.y = CLOCK( ON,OFF | ...)
```

The controlled output may be both real and virtual point too.
`ON` and `OFF` are respectively the switch on and the switch off schedule, expressed in the following format:

```
GG:HH:MM
```

`GG`: week day; it may assume values in the range 1 through 7: `1=MON, 2=TUE, 3=WED, 4=THU, 5=FRI, 6=SAT, 7=SUN`. **If this parameter is omitted, the schedule will be assumed daily**. Instead of the day number, it is allowed (and recommended) to use the first three characters of the day name (`MON, TUE, WED, THU, FRI, SAT, SUN`).
`HH`: hours (00 ÷ 23); this parameters must be expressed in the 24h notation (e.g. 7 means 7a.m. and 19 means 7p.m.).
`MM`: minutes (0 ÷ 59).
| : OR operator to combine more schedules.

Examples:

`O1.1 = CLOCK(8:15, 17:30)` — `O1.1` output will be switched on everyday since 8:15 to 17:30.

`O1.3 = CLOCK(1:8:00, 5:20:00)` — `O1.3` output will be switched on every Monday at 8:00 and will be switched off every Friday at 20:00.

`O1.3 = CLOCK(MON:8:00, FRI:20:00)` — This equations is fully identical to the previous one.

Note: the time 24:00 will be rejected by the compiler; this time must be specified as 0:00 of the day after; e.g. the following equation:

```
O1.1 = CLOCK(MON:8:00, WED:0:00)
```

will switch on the output every Monday at 8:00 and it will switch off the output every Wednesday at 0:00, that is the midnight of every Tuesday.

MCP release 4.x (or higher) allows defining **variable schedules** (set by a supervisor system) using counter registers. **Two cases are possible**:

1. Fixed weekday (`GG`) and variable time (`HH:MM`); the syntax is the following:

```
Ox.y = CLOCK( Cv, Cw )
```

or:

```
Ox.y = CLOCK( GGa:Cv, GGb:Cw )
```

In this case the counters `Cv` and `Cw` contain a number in the range 0 to 1439, corresponding to the **number of minutes starting from** `0:00` (1439 = `23:59`). In the first equation the schedule is daily, in the second one it is weekly but weekdays are fixed and cannot be changed (`GGa` e `GGb`) by the supervisor.

2. Variable weekday and time (`GG:HH:MM`); la syntax is the following:

```
Ox.y = CLOCK( xxx:Cv, xxx:Cw )
```

or:

```
Ox.y = CLOCK( 8:Cv, 8:Cw )
```

In this case the counters `Cv` and `Cw` contain a number in the range 0 to 10079, corresponding to the **number of minutes starting from** `0:00` of Monday (10079 = `23:59` of Sunday). "**`xxx`**" in the first equation and "**`8`**" in the second one are fixed symbols showing that the following counter register contains the week minutes number.

WARNING: counter registers used in CLOCK function must be exclusively 16-bit type.

Following examples show the use of CLOCK functions with variable schedules.

Examples:

`O1.1 = CLOCK(C0, C2)`	Daily switch on at the time specified by the contents of register C0 and switch off at the time specified by register C2. If C0 contents is 480 and C2 contents is 1020, then the output will be switched on since 8:00 to 17:00
`O1.1 = CLOCK(TUE:C0, THU:C2)`	Weekly switch on Tuesday at the time specified by the contents of register C0 and switch off and switch off Thursday at the time specified by the contents of register C2. If C0 contents is 675 and C2 contents is 1422, then the output will be switched on since Thursday 11:15 to Thursday 23:42.
`O1.1 = CLOCK(xxx:C0, xxx:C2)`	Switch on since the day and time specified by the contents of register C0 to the day and time specified by the contents of register C2. If C0 contents is 675 and C2 contents is 6780, then the output will be switched on since Monday 11:15 to Friday 17:00.
`O1.1 = CLOCK(8:C0, 8:C2)`	This equations is fully identical to the previous one.

As said before, **more schedules may be programmed through the OR (`|`) operator as shown by the following examples (it is the only operator allowed in a `CLOCK` function).**

Examples:

`V4 = CLOCK(TUE:8:00, TUE:12:00 | THU:14:30, SAT:00:00)`	`V4` ON since 8:00 to 12:00 of every Tuesday and since 14:30 of Thursday to midnight of Friday (that is 0:00 of Saturday).
`V2 = CLOCK(MON:C4, TUE:C6 | FRI:14:30, SAT:00:00)`	`V2` ON since Monday at the time specified by C4 to Tuesday at the time specified by C6 and since Friday 14:30 to Saturday 0:00 (or Friday midnight). In this example, the days for switch on and off are fixed.
`V1 = CLOCK(xxx:C0, xxx:C2 | xxx:C4, xxx:C6)`	`V1` ON since the day/time specified by C0 to the day/time specified by C2 and from the day/time specified by C4 to the day/time specified by C6. Counter registers in this example contain both the weekday and the time.

2.3.7- Analog modules control

Even if the function to transfer the value from an analog input module to an analog output module may be implemented by 8 point to point relationships (specifying that each bit of the output be equal to the related bit of the input), the use of the analog operator allows a quickly processing and the possibility to execute simple arithmetical computations on the values returned by the analog input modules.

The simplest syntax of the analog function is the following:

```
Ox = Coeff Ai
```

where `Ox` is the **analog** output module whose address is `x`, `Ai` is the **analog** input module whose address is `i`, and `Coeff` is a scale factor in the range 1 through 256.

The following rules apply to the analog function:

1. the term `Coeff Ai` may be omitted (in this case it will be assumed equal to 1); if `Ai` is omitted, then a constant term equal to `Coeff` will be assumed.

2. More terms `Coeff Ai` may be summed, subtracted, multiplied ad divided each one to others. The shown algebraic computations will be executed in the written order, left-side to right-side. The operators involved in the algebraic computation may be **+** (sum), **-** (subtract), ***** (multiply) and **/** (division).

3. No brackets are allowed.

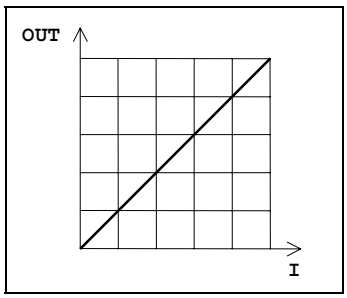
4. Each term is the product of two 8-bit numbers (**Coeff** and input signal value). The result is a 16-bit number. The computations are executed on 16-bit numbers and the results are truncated to 16-bit values; in other words, if a value exceed 65535 (max value that can be expressed by a 16-bit number), the result will be set to 65535. If the result of a subtraction is a negative number, the zero value will be assigned to the result. These limits are applied both to the final and the partial results.

5. The final value will be truncated to a 8-bit number (255 max), so that it can be assigned to the output. In other words, if the result exceed 255, it will be set to 255.

Examples:

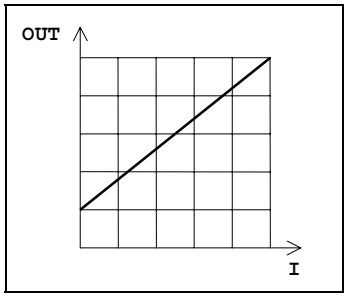
O1 = A2

Simple assignment of the module 2 analog input value to the analog output module 1.



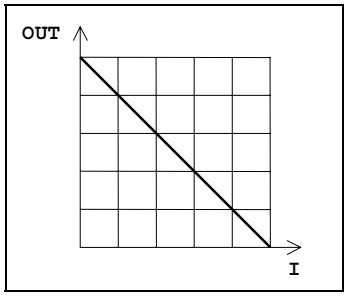
O1 = 4A1 / 5 + 51

This equation compute 4/5 of the input value and add an offset (51). The result will be sent to the analog output module 1.



O1 = 255 - A1

The output is complementary in respect to the input.



O1 = A1 + 3A2

Sum of A1 and 3 times A2. If A1 = 10 and A2 = 5, the output will be 25.

O1 = A1 + 3 * A2

Product of A2 multiplied by the sum of A1 and 3: note the difference against the previous example. In this case MCP executes the sum A1 + 3 and the result will be then multiplied by A2. If A1 = 10 and A2 = 5, then the output will be 65. In this case the equation is made by three terms, instead of two as in the previous example.

2.3.8- Toggle

The toggle operator, identified by **T**, reverse the output status at each OFF→ON variation of the input status. The level of the input (ON or OFF) will be ignored until another variation as above occurs. The common using of this function is the bi-stable relay simulation. In addition, the TOGGLE operator allows the optional definition of an input to force the output switch ON and of an input to force the output reset (asynchronous SET and RESET).

The general format of the toggle function is the following:

Ox.y = TIj.k | RIh.l | SIm.n

where **Ij.k** is the input that control the reversing of the output, **Ih.l** is the reset input and **Im.n** is the set input.

The output and/or inputs in the equation of the TOGGLE function may be virtual points too. It is also possible to **control the output from more inputs using the | (OR) symbol** as explained in the following examples. **In a toggle function the & (AND) symbol is not allowed and each input in the equation must be preceded by one of the three prefix T, S and R .**

To reverse the logic of the toggle, set and reset inputs, the **!** symbol must be placed before the relevant input.

Examples:

O1.1 = TI6.1 Simulation of a **BI-STABLE RELAY**: the output is change at each OFF→ON variation of the input.

O1.1 = T!I6.1 Simulation of a **BI-STABLE RELAY**: in this case the output change at each ON→OFF of the input. The **!** symbol reverse the input logic in respect to the previous example.

V100 = TI1.1 | TV58 In this case the controlled output and an input are virtual points.

O1.1 = TI1.1 | TI1.2 | SI1.3 | SI1.4 | RI1.5 | RI1.6 In this equation there are two inputs for the output reversing, two inputs for the switch-on and two inputs for the switch-off.

2.3.9- 16-bit Counter Module

The 16-bit counter module (code MODCNT) **is an external module** counting the pulses applied at its inputs and stores the total amount in its non-volatile memory. MCP handles this module through a function similar to that used for internal counter. Each counter module may have more than one counting input (4 for MODCNT module), then **it is mandatory to specify, in the equation, the channel number to be considered**. The general syntax is the following:

Ox.y = CIj.k {op} T Zip.q

where:

Ox.y is the (real or virtual) output to be controlled
CIj.k is the **k**-channel (1 to 4) of counter module whose address is **j** (1 to 127)

`ZIp.q` is the (real or virtual) input to reset the counter; this is an optional parameter
`{op}` is the comparing rule between the counter value and the threshold `T.`
`{op}` may be one of the following symbols:
`=` (equal)
`>` (greater or equal)
`<` (lower or equal)
`T` is the threshold and it is a numerical value in the range 0 to 65535. `T` may be a pointer to a counter register; in other words, the **threshold value may be defined as the contents of a counter register** (see in the following pages for details)

As just explained, **it is possible to define, as threshold value `T`, the contents of a counter register**. In this way it is possible to define variable threshold counters; the proper threshold can be set by a supervisor (writing in its relevant register), or by an UP and DOWN control of the counter containing the threshold value.
Counter registers used as threshold for this function must be exclusively 16-bit type.

Examples:

`O1.1 = CI10.1 > 100 ZI1.1` Output `O1.1` is controlled by channel 1 of counter module whose address is 10; output will be activated when the counting will be greater or equal to 100. Input `I1.1`, if active, reset the counter contents.

`V10 = CI1.4 > C2` Virtual point `V10` will be ON when the counting of channel 4 of counter module 1 will be greater or equal to the contents of the counter register `C2` (16-bit type).

2.3.10- 16-bit Analog Input Module

MCP handles this module through a function similar to `Threshold` operator for 8-bit modules. Each 16-bit analog input module may have more than one input, then **it is mandatory to specify, in the equation, the channel number to be considered**. The general syntax is the following:

`Ox.y = Ai.j {op} T,H`

where:

`Ox.y` is the (real or virtual) output to be controlled
`Ai.j` is the `j`-channel of analog input module whose address is `i` (1 to 127)
`{op}` is the comparing rule between the analog value and the threshold `T`.
`{op}` may be one of the following symbols:
`>` (greater or equal)
`<` (lower or equal)
`T` is the threshold and it is a numerical value in the range 0 to 65535. `T` may be a pointer to a counter register; in other words, the **threshold value may be defined as the contents of a counter register** (see in the following pages for details)
`H` is the hysteresis value and it is a numerical value in the range 0 to 65535. `H` may be a pointer to a counter register; in other words, the **hysteresis value may be defined as the contents of a counter register** (see in the following pages for details)

As for 8-bit threshold operator, more terms may be linked together by AND (`&`) and OR (|) operators.

As said above, **MCP release 4.x (or higher) allows to define, as threshold and/or hysteresis value, the contents of a counter register**; in this way it is possible to define threshold functions with variable parameters. The proper threshold and/or hysteresis can be set by a supervisor (writing in its relevant register), or by an UP and DOWN control of the counter containing the parameter. **The counter register used in a threshold function must be 16-bit type only**. The following examples show the use of threshold functions with variable parameters.

Examples:

`O1.1 = A1.1 > 2400,2`	Output ON when the analog signal applied to channel 1 of module 1 is greater or equal to 2400 (steps); hysteresis is fixed to 2 steps.
`V2 = A1.4 < 400,2 | A2.3 < 300,4`	Output `V2` active when channel 4 of module 1 is lower or equal to 400, or when channel 3 of module 2 is lower or equal to 300; hysteresis are 2 and 4 steps respectively.
`V8 = A1.3 > C1,20`	Output `V8` active when channel 3 of module 1 is greater or equal to the contents of counter register `C1` (16-bit); hysteresis is fixed to 20 steps.
`V8 = A1.3 > C1,C4`	Output `V8` active when channel 3 of module 1 is greater or equal to the contents of counter register `C1` (16-bit); hysteresis value is the contents of counter register `C4`.

2.3.11- Configuration

The system configuration, that is how many and what types of modules are connected to the bus of MCP module, is automatically derived from the written equations. In other words, examining the program, MCP builds the map of the involved input and output modules that will be included in the polling cycle.

It is possible, in special applications, that the system have **some modules not necessarily involved in any equation**, e.g. in system where there are input modules for the signal monitoring through a supervisor, or output modules wholly controlled by the computer, etc. **In this case it is however mandatory to declare to MCP module the presence of such modules**, so that they can be polled to read their status and a possible failure.

The equation declaring the presence of some additional modules (not involved in the remaining equations and then not automatically inserted in the configuration map) is the following:

```
CONF ( Ii & Ij & … & Ox & Oy & … & Ck & Cu & … & Av & Aw & … )
```

where `i, j, x, y,` etc. are the module addresses (1÷ 127) and the symbols `I` and `O` show if the following address is related to an input or output module; terms `C` identify the counter modules and terms `A` identify 16-bit analog input modules.

& symbol is mandatory and the equation can be broken in many lines using the symbol "`\`" (see chapter about the equation writing).

CONF operator does not apply to virtual points.

Example:

`CONF ( I1 & I12 & O4 & O65 )` Add to the system map the input modules 1 and 12 and the output modules 4 and 65.

`CONF ( I1 & I2 & O7 & O22 & C50 & C51 & A80 )` Add to the system map the input modules 1 and 2, the output modules 7 and 22, the counter modules 50 and 51 and the 16-bit analog input module 80.

2.3.12- Address

Up to **31 MCP modules** can be connected together through a RS485 line using proper signal converters. In this case, it is mandatory to assign to each MCP a number called address. To do this, the following equation has to be used:

`ADDRESS = n`

where `n` is a number in the range 1÷255 that identifies the MCP module. **The 0 (zero) address is recognized by any MCP**, regardless of its assigned address. The zero address can be therefore used to communicate with any MCP, but to do this, only one module at a time can be connected to the RS485 line in order to avoid bus contention during the reply.

WARNING: even if the address may be a number in the range 1 to 255, the "physical" number of MCP that can be connected on the same RS485 line must be lower or equal to 31.

Example:

`ADDRESS = 4` The address 4 is assigned to MCP; in this way it will answer to the requests containing the value 4 and 0 in the address field (0 is called jolly address, see the Appendix).

2.3.13- Identification code

Each MCP module can be identified through a name that may be very useful to recognize the called unit in network systems where more than one MCP are connected together (using the RS485 line as explained before) or where the handling of the system is made trough a modem.
The identification code is an alphanumeric string containing up to **63 characters** and it assigned using the following equation:

`ID = (Identification code)`

where the desired code specified into the brackets.

Spaces and brackets cannot be used inside the identification code. For best readability, the underscore symbol may be used instead of the spaces.

Example:

`ID = (Control_Station_23_LONDON)` Declaration of the identification code. The underscore symbol is used instead of the not allowed blank symbols.

2.3.14- Events

Mcp release 4.x (or higher) allows to store, in chronological order together to date and time, **up to 1023** events related to the status transition of one or more points that can be **virtual** type only. The general syntax is the following:

```
EVENT ( \
      Vn {ON} {OFF} \
      ... \
      )
```

where:

Vn is the virtual point whose transition has to be stored

ON declares that the storing occurs on 0⟶1 transition of the virtual point

OFF declares that the storing occurs on 1⟶0 transition of the virtual point

When defining the **EVENT** function, MCP automatically reserve a RAM section to store in chronological order the events specified in the block (1023 max). **Each one of these events is stored together the number of the virtual point, the status, the date and time of the event occurrence**.

This event list may be read and reset by a PC or supervisor system connected to the serial port of MCP; the addresses of the events in the RAM are as here below listed:

- events are stored beginning from address 2000h (hex)
- addresses 2000h and 2001h contains the total amount of stored events (2000h is the most significant byte)
- addresses 2002h and 2003h are not used
- information about each event are stored beginning from address 2004h as blocks of 4 bytes per event; each of these blocks contains:

<table>
<tr><td>-</td><td colspan="3">Address n of the virtual point (7 bit)</td></tr>
<tr><td colspan="2">Month (4 bit)</td><td>Status</td><td>Point p (3 bit)</td></tr>
<tr><td colspan="3">Day (5 bit)</td><td>Hour MSB (3 bit)</td></tr>
<tr><td>Hour LSB (2)</td><td colspan="3">Minutes (6 bit)</td></tr>
</table>

The virtual point number related to the event is given by **(n-1)*8 + (p+1)**; *Status* is the logical level of the virtual point, *Month*, *Day*, *Hour* and *Minutes* gives the time during which the event occurred.

The queue is a chronological list and can store up to 1023 blocks. **Further events will be lost**. The supervisor, to reset the event list, has to write zero at addresses 2000h and 2001h, so resetting the amount of stored events.

Notes:

1. The virtual points in an EVENT block have to be defined by proper equations (combining real and/or virtual points). **If ON and OFF parameters are both declared, the storing occur both at 1⟶0 and 0⟶1 transition**
2. A module or bus failure conditions may be stored using the virtual points V999 and V1000 (see related paragraph)
3. The symbol \, that means that the equation follows in the next line, is mandatory (see chapter about the equation writing)

Example:

```
V1 = I2.1                          Definition of event 1
V2 = I2.2 | I2.3 | I2.4            Definition of event 2
V3 = I2.5 & I2.6                   Definition of event 3

EVENT ( \                          Block start
       V1 ON \                     Event 1, at 0→1 transition of V1
       V2 OFF \                    Event 2, at 1→0 transition of V2
       V3 ON OFF \                 Event 3, at both 0→1 and 1→0 transition of V3
)
```

2.3.15- Modem

MCP module can be programmed in such a way to generate a call through a modem when some specified variations of one or more points occur; this point can be **virtual points** only. The equation defining the virtual points to generate a modem call is the following:

```
MODEM ( \
      TEL1 = x \
      TEL2 = y \
      TEL3 = w \
      TEL4 = z \
      Vn {ON} {OFF} \
      ... \
      )
```

where:

`Vn`	is a virtual point that can generate the modem call
`ON`	specifies that the modem call must occur on the 0→1 transition of the virtual point
`OFF`	specifies that the modem call must occur on the 1→0 transition of the virtual point
`x,y,w,z`	are the phone numbers to be called

The "\" symbol, that means that the equation follows in the next line, is mandatory (see chapter about the equation writing).
It is possible to include, in the MODEM block, up to 3 AT command strings to automatically setup the modem at the power up of MCP module (or at the reset); the max length for each string is 17 characters. The setup strings must be placed, inside the MODEM block, before the telephone numbers; the example at the end of this paragraph shows how to use the modem setup strings.

Notes:

1. The virtual points must be specified by proper equations containing combinations of real and/or virtual points. If the ON and OFF parameters are both specified, the calling will be made both on the 1→0 and 0→1 transition
2. It is possible to generate the calling when a module or a bus failure occurs, using the V999 and V1000 virtual points (see related paragraph)
3. It is possible to insert up to 4 phone numbers TEL1, TEL2, TEL3 and TEL4. The calling order will be from TEL1 to TEL4. **It is mandatory to define all four telephone numbers** (repetitions of the same number are allowed)
4. The phone numbers can be up to **15 digit long**
5. It is recommended, even if not mandatory, to define the MODEM setup strings

When one of the specified events occurs, MCP calls, through the modem, the first phone number; if this one is busy or no answer occurs during a 90 seconds time-out, the calling will be repeated two times again, then MCP will go to the next number. If after all the attempts MCP had no answer to its calls, then the sequence will be stopped; instead, if at least a called number was busy, the whole sequence will restart from the first number.

The answering station **must send to MCP an identification request to confirm the occurred acknowledgment of the alarms; on the contrary MCP will call again starting from the first phone number.**

When the modem function is defined, MCP automatically reserves a RAM section to **store in chronological order up to 127 events** among those specified in the MODEM block. Each event will be stored together the number of the virtual points to which it is referred, the date and the time of the occurrence.

WARNING: the event mapping inside the RAM memory of MCP was changed for release 4.x as here below listed:

- events are stored beginning from address 300h (hex)
- address 300h contains the total amount of stored events
- addresses 301h, 302h and 303h are not used
- information about each event are stored beginning from address 304h as blocks of 4 bytes per event; each of these blocks contains:

<table>
<tr><td>-</td><td colspan="3">Address n of the virtual point (7 bit)</td></tr>
<tr><td colspan="2">Month (4 bit)</td><td>Status</td><td>Point p (3 bit)</td></tr>
<tr><td colspan="3">Day (5 bit)</td><td>Hour MSB (3 bit)</td></tr>
<tr><td>Hour LSB (2)</td><td colspan="3">Minutes (6 bit)</td></tr>
</table>

The virtual point number related to the event is given by `(n-1)*8 + (p+1)`; *Status* is the logical level of the virtual point, *Month*, *Day*, *Hour* and *Minutes* gives the time during which the event occurred. The queue is a chronological list and can store up to 127 blocks. **Further events will be lost**. The supervisor, to reset the event list, has to write zero at addresses 300h, so resetting the amount of stored events.

Example:

```
V1 = I2.1                       Definition of alarm 1
V2 = I2.2 | I2.3 | I2.4         Definition of alarm 2
V3 = I2.5 & I2.6                Definition of alarm 3

MODEM ( \                       Block start
   AT&F0E0          \           MODEM setup string 1
   AT-K0X3&K0&D0    \           MODEM setup string 2
   ATT&WZ           \           MODEM setup string 3
   TEL1 = 02112233  \           Phone number 1
   TEL2 = 12345678  \           Phone number 2
   TEL3 = 112       \           Phone number 3
   TEL4 = 113       \           Phone number 4
```

```
   V1 ON          \      Alarm 1, at 0→1 transition of V1
   V2 OFF         \      Alarm 2, at 1→0 transition of V2
   V3 ON OFF      \      Alarm 3, both at 0→1 and 1→0 transition of V3
)                        Block end
```

Note: the modem setup strings in above example are referred to a modem manufactured by DIGICOM. Applying the power supply to MCP module after having switched on the modem and connected via RS232, the modem setup strings will be transferred to the modem itself; it is recommended to always include this AT commands in the MODEM block.
Phone numbers may contains special characters as semicolon symbol to define a pause before to continue in the dialing.

The RS232 serial connection between MCP module and modem is 3-wires type (TX, RX and common).

2.3.16- Protocol

MCP Plus module can communicate with external world by the MODBUS RTU Protocol or by JOHNSON CONTROL VND protocol. These protocols have been integrated into MCP Plus e coexist with the proprietary FXP protocol (see following pages).
MODBUS and JOHNSON protocols are normally disabled; to enable the MODBUS protocol it is necessary to include in the program of MCP Plus the following directive:

```
PROTOCOL = MODBUS
```

In the same manner, to enable the JOHNSON protocol it is necessary to include in the program of MCP Plus the following directive:

```
PROTOCOL = JC
```

The proprietary FXP protocol (that is the normal communication protocol of MCP) is always enabled; in other words, if MCP Plus receives a request in FXP, it will answer according to the same protocol irrespective of the setting of protocol directive.

Notes:

- MODBUS and JOHNSON protocols **cannot** be contemporarily enabled.
- If JOHNSON protocol has been enabled, then MCP Plus will answer to requests in FXP protocol only if containing zero address (jolly address); the answer of MCP Plus will always contain the address defined by the ADDRESS directive.
- If MODBUS protocol has been enables, then MCP Plus MCP Plus will answer to requests in FXP protocol only if containing zero address (jolly address) or if containing the address defined by the ADDRESS directive; the answer of MCP Plus will always contain the address defined by the ADDRESS directive.

3- EQUATION WRITING

The equation writing is the first step of the MCP module programming. The equations must be written according to the syntax described in the previous paragraphs.

To write the equations, the software package **MCPTOOLS** has to be used; this package is provided by **DUEMMEGI** together to MCP module.

MCPTOOLS works on a Personal Computer under WINDOWS® environment and allows an easy writing of the program and system setup. **For more details about the use of this tools, refer to the related documentation.**

MCPTOOLS essentially includes:

- a text editor to write the program
- a compiler to allow the translation of an ASCII file, containing the operating equations, in a binary file adequate to be transferred in the non volatile memory (FLASH type) of MCP module
- a simulator to verify the written program, or a part of it, before to transfer it into MCP memory
- an utility to transfer the program from the PC to MCP (or vice versa)
- a map window to display the status of input and output modules installed in the plant

The file containing the equation is in ASCII format and must have the **.EQU** extension; as example:

***filename*.EQU**

where *filename* is the name of the program file and may be any name allowed by the WINDOWS® syntax.
The **.EQU** extension is mandatory because the sequential steps of MCP programming (compiling and transferring) require that the source file have that extension.

MCP module programming takes place in a 3 sequential steps, through the MCPTOOLS support:

1. building (or editing) of the *filename*.EQU file, containing the operating program
2. compiling of *filename*.EQU, that is the conversion of the ASCII file in the related *filename*.BIN written in a format ready to be transferred into MCP memory
3. uploading of *filename*.BIN into MCP memory

If some syntax errors are detected during the step 2, these ones will be signaled by the compiler, together to some information about the error type and the line number where the error occurs.

3.1- Rule for equations writing

Each equation must be written according to the syntax described in its relevant paragraph (logic, counter, timer, etc. ...).

The following rules have to be observed:

- Spaces and TAB characters have no meaning. They will be ignored by the compiler but **the use of some space characters between the terms of an equation are strongly recommended for a best readability of the program**
- An equation can be broken on several lines using the symbol **** (backslash) at the line end to specify that the equation will continue on the next line
- The equation finishes at the end of the line (if the \ symbol is not specified)
- The // symbol (two slashes) declares that the following words, until the line end, are comments, and so they will be ignored by the compiler. **The comments are very useful for best readability and**

documentation of the program file. The use of the comment is strongly recommended to describe each equation in the program

- Both upper case and lower case characters can be used during the equation writing

Instead of the input and output symbols (**`Ij.k, Ox.y, Vn, Aj`**), it is possible to employ some variable names defined by the user through the **`#define`** directive as here below described:

```
#define     %Pump1%     O1.1  // Output definition
#define     %Command%   I1.1  // Input definition

%Pump1% = %Command%           // Equation
```

The previous equation is fully equivalent to:

```
O1.1 = I1.1
```

The variable names defined through the **`#define`** directive must be enclosed inside two **`%`** symbols and cannot be made by other **`%`** symbols and/or spaces. In addition, the compiler will ignore upper or lower case. The following example shows a possible and simple program:

```
////////////////////////
// Definitions
////////////////////////
#define     %StairLight%      O1.1
#define     %Floor1Button%    I1.1
#define     %Floor2Button%    I1.2
#define     %Floor3Button%    I1.3

// Define a virtual point as OR of each button (parallel connection)
V1 = % Floor1Button% | % Floor2Button% | % Floor3Button%

// Light Output
%StairLight% = TIMER (V1, 0, 450)
```

In the above example there is 3 buttons, one per each floor of a building; the pressing of a button switches on the stair light. This light will remain on during 45 seconds after the button release, then it will be automatically switched off thanks to the TIMER function. The same program may be written without using the definition of variable names as follows:

```
// Command by the buttons
V1 = I1.1 | I1.2 | I1.3

// Light output
O1.1 = TIMER (V1, 0, 450)
```

Note that using the "define" directive, the program has a best and mnemonic readability.

To write and compile a program, it is not necessary that MCP module be connected to the PC.

3.2- Compiling the equations

The compiling is the second step of MCP programming process. **The file containing the written equations (.EQU extension) must be compiled through the proper menu item of MCPTOOLS utility**.

The compiler processes the written equations, check the syntax and the congruence, warns the errors if any and link the data in a binary file which name is the same as the .EQU file but with .BIN extension. The binary file is not in a printable format but it is adequate to be transferred in the MCP memory.

To write and compile a program, it is not necessary that MCP module be connected to the PC.

If during the compiling process one or more errors occur, they will be displayed on the screen of the PC and the program continues to check all the equations but no binary file will be generated.

The compiler may also reports some WARNINGS: this means that no errors have been detected but there are some points to be verified before to upload the program to MCP memory; the binary file will be however created even if one or more warning messages occur.

3.3- Upload of the program into MCP module

Last step of MCP programming process is the **uploading into its flash MEMORY of the binary file** containing the system configuration and the equations code.

The uploading is made by the proper menu item of MCPTOOLS utility trough the RS232 port of PC connected to the MCP serial port.

The uploading of the program requires that MCP module be supplied and connected to PC by means of the proper cable provided with MCP.

*Note: **MCP baud rate factory setting is 19200**; if a slower speed is needed, set the internal jumpers or micro-switches of MCP module as described in next chapter.*

4- SETTING UP

4.1- Connections

Standard MCP is available in 2 housing versions:

- table housing, named **MCP**
- DIN modular housing (9 modules size), named **MCP/MOD**

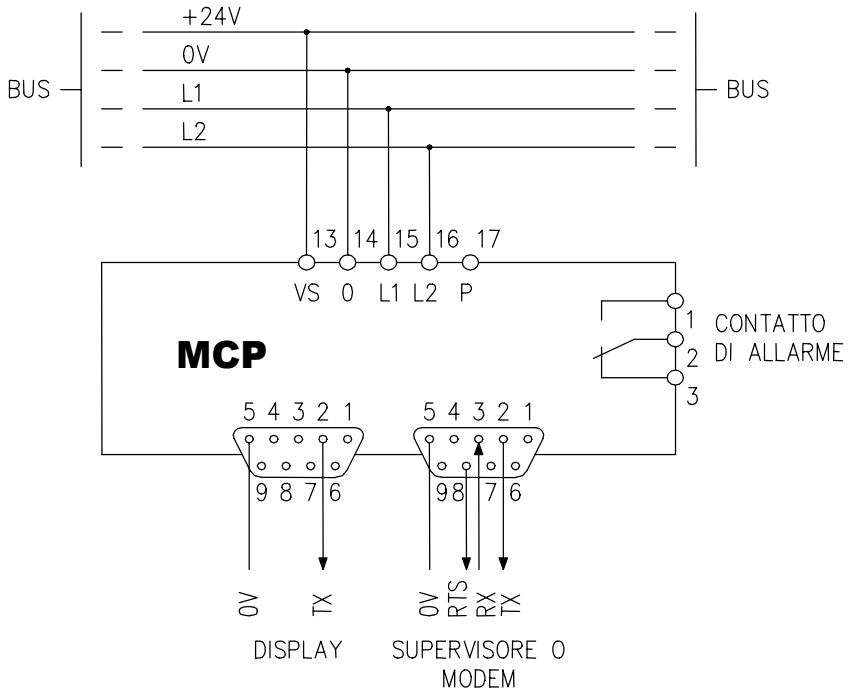
MCP Plus is available in DIN modular housing only (9 modules size).

All versions are provided with a 5 poles removable terminal block for the connection to the system bus and a 3 poles removable terminal block internally connected to a relay for the signaling of system failure (module failure, bus failure, etc.). This relay is **normally energized** and it will be de-energized when a failure occurs; in this way the system anomaly warning will occur also at the failure of MCP module power supply. The restoring of the relay is automatic, because when the anomaly is removed it return to its normal state (energized). Due to the just described operating mode, the optional fault **indicator (flasher, siren or other) has to be connected to the normally closed contact of the relay**; the contact rating is 5A @ 250Vac.

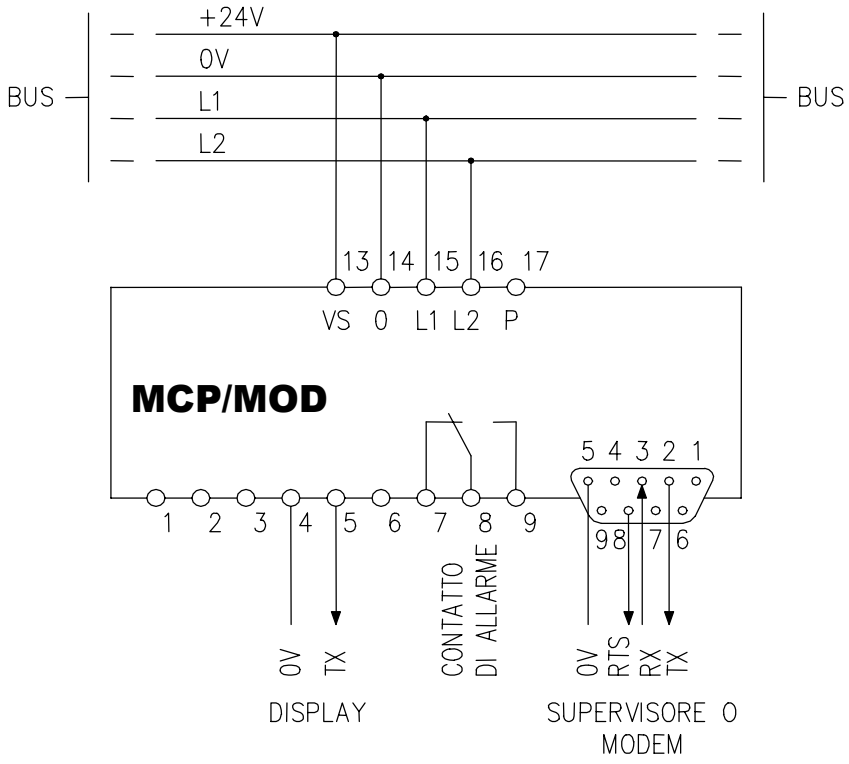
A serial RS232 port allows the connection between MCP module and a Personal Computer (or modem); an additional serial RS232 port is available for standard MCP to connect a serial display (as **DUEMMEGI DISP-S**, optional). MCP Plus, instead of display output, has a RS485 serial port for connection in multi-drop network.

Following figures show the proper connections to be made for available versions.

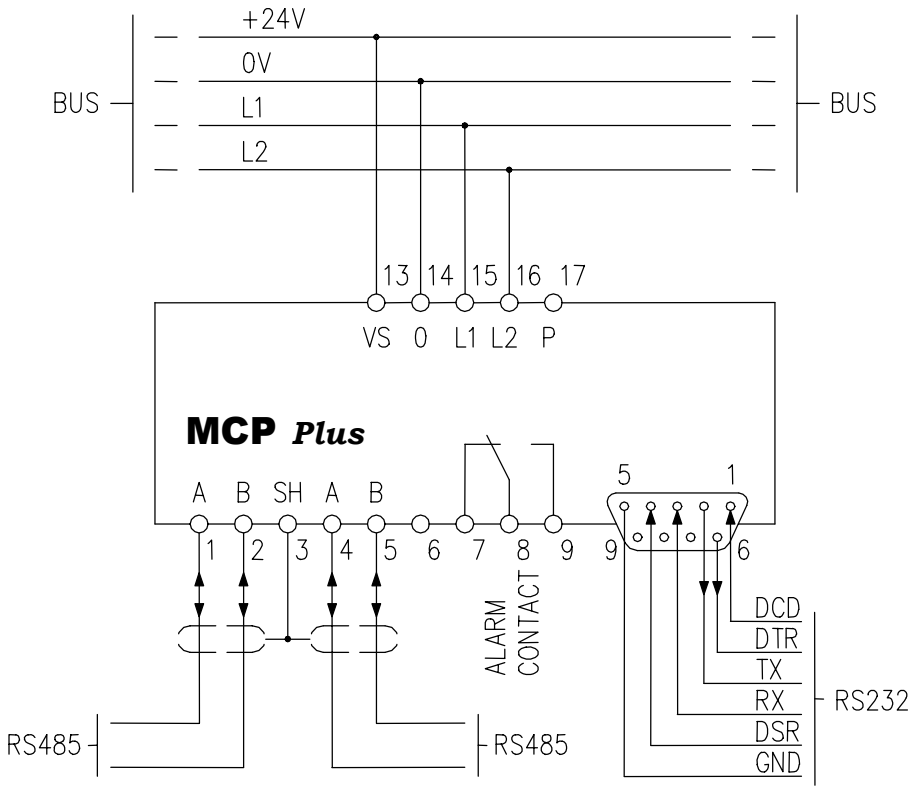
Connections of MCP module



Connections of MCP/MOD module



Connections of MCP Plus module

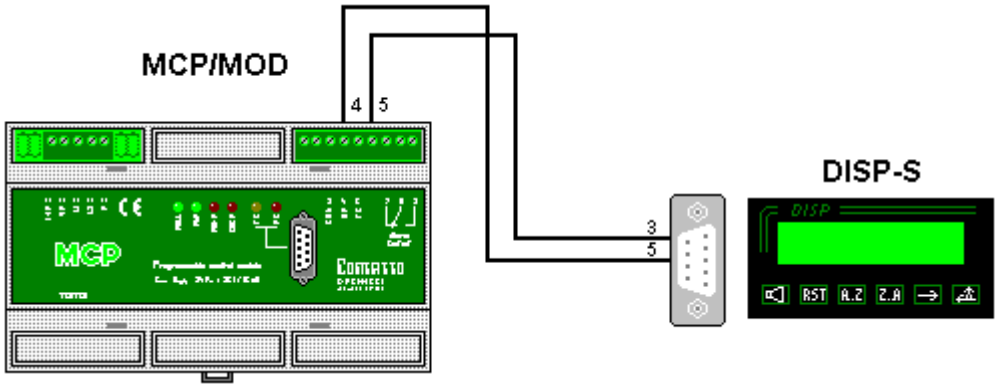


For all versions, the additional 5 poles terminal block, identified by the label "TESTER" on the MCP panel, is reserved to the connection of **Contatto FXPRO** system tester as described in the following chapter.

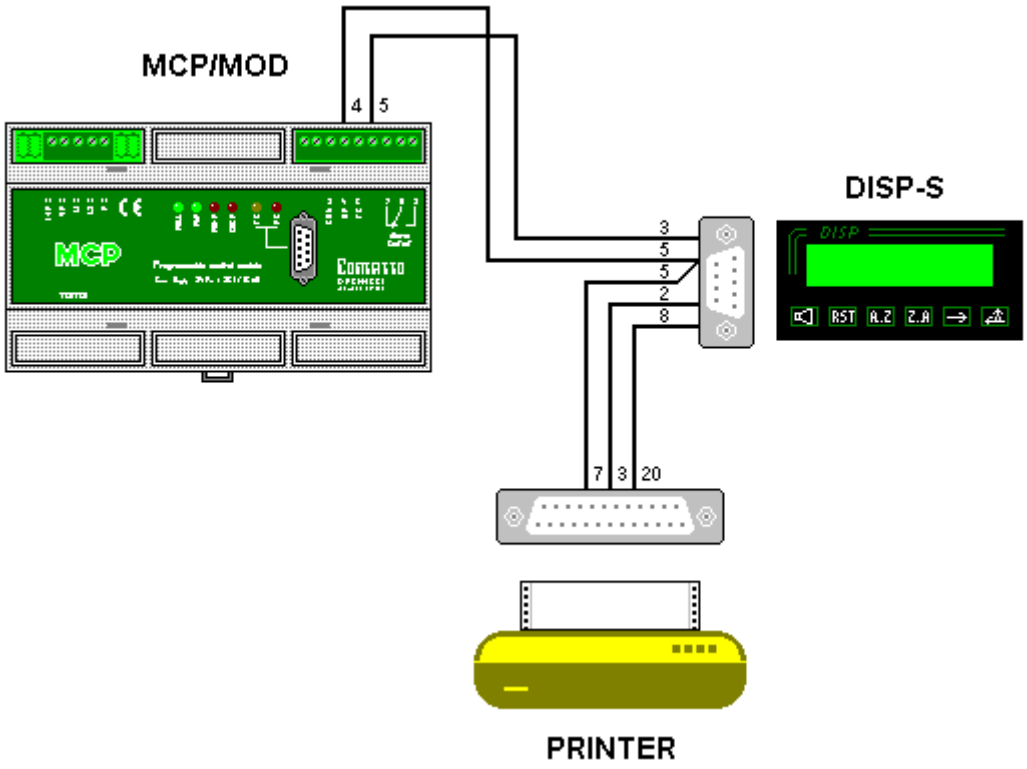
4.2- Connection of the serial display DISP-S

Following figures show how to connect a serial display **DUEMMEGI DISP-S** and (if required) a serial printer to MCP module for both existing versions. All DB connector shown are male connectors.

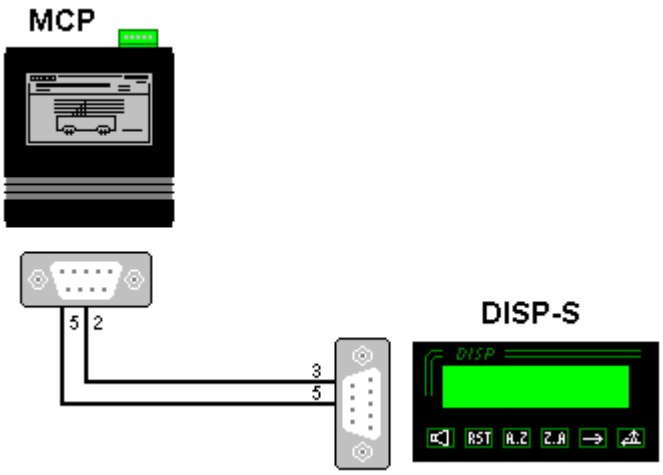
MCP/MOD to DISP-S



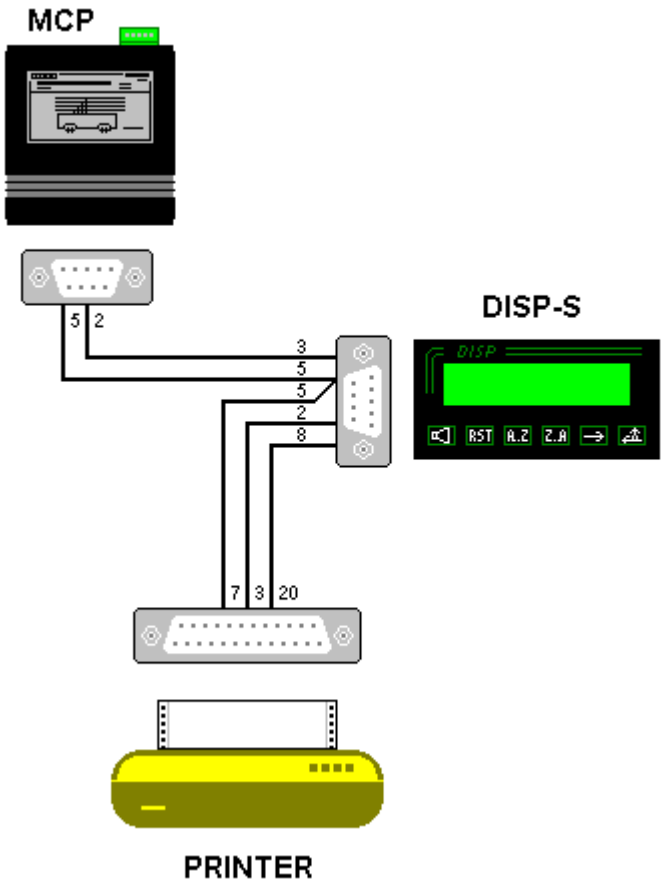
MCP/MOD to DISP-S and serial printer



MCP to DISP-S



MCP to DISP-S and serial printer



4.3- Baud rate selection

Baud rate factory settings for both MCP and MCP Plus is 19200; if for any reasons this speed has to be changed, proceed as here bottom described. Available baud rates are:

- 19200 Baud
- 9600 Baud
- 4800 Baud
- 2400 Baud

1. For table housing (MCP), open the module removing the two screws on the bottom side of the enclosure and move the two jumpers on the component side of the electronic board according to the truth table printed on the board itself.
2. For modular housing (MCP/MOD and MCP Plus), remove the cover between the bus terminal block and the alarm contact terminal block; set the dip-switches to the required baud rate according to the truth table printed on the board itself.

To make operating the new baud rate selection, switch-off and then switch-on again MCP module.

WARNING: the baud rate settings on the serial display output port for MCP and MCP/MOD is always the same selected for the main serial port (Supervisor/Modem).

4.4- RS232 and RS485 serial ports of MCP Plus

MCP Plus provides both RS232 (on front panel) and RS485 (terminals 1 to 5) serial ports. These ports are **electrically insulated from other circuits** by means of some internal opto-couplers and a dc/dc converter (no additional external power supply is required). However, RS232 and RS485 ports **are not insulated each one to the other**.

These ports are **"mutually exclusive"**, because cannot be contemporarily used (they are not independent ports); the switching from one port to the other occurs through the DSR input signal on RS232 connector according to the following logic:

- DSR active (+10V): RS232 port enabled and RS485 port disabled (both trasmitting and receiving)
- DSR not active (-10V or not connected): RS485 port enabled and RS232 port disabled

In other words, having a MCP Plus connected to a RS485 network, simply insert the RS232 connector coming from a PC (or other communication device) to disconnect MCP Plus from the network and to begin the communication with the PC itself (if DSR signal is properly handled as said before); this mode of operation makes easy the re-programming or the check of MCP Plus because these function can be accomplished without "physical" disconnection from the network.

RS485 port of MCP Plus is doubled into 4 terminals (plus another terminal for the shield) in order to make easy the multi-drop connection: in other words, terminals 1 and 4 (signal "A") are internally shorted together; in the same way, terminals 2 and 5 (signal "B").

WARNING: as for all RS485 networks, ***radial connections must be avoided***; in addition, RS485 line ***must be loaded, bot at the beginning and at the end, by a 120 Ohm 1/2W resistor*** between terminals A and B. The maximum number of device that can be connected on RS485 line must be limited to 32.

5- DIAGNOSTICS

5.2- Diagnostics of CONTATTO *system through MCP*

MCP module provides the failure warning through two red leds and a relay contact as described in previous paragraph.

The red leds warns the following alarms:

- **MOD.F** (module failure)
- **BUS.F** (bus failure)

and the internal relay switches at the occurrence of each one of these two failures or at the removing of MCP supply (intrinsic safety).

The search of fault modules may be done in two ways:

1- Through **FXPRO** system tester/programmer.
This is the quickly method to find fault modules. MCP provides a 5 poles terminal connector for the tester connection. Pressing the Verify button of FXPRO, the address of the module that do not answer to the cyclic polling will be shown on the tester display.
Note: During the diagnostic procedure all outputs will be switched off. They will be restored to the proper status at the end of the procedure itself.

2- Through **Personal Computer**.
This is the powerful method. The diagnostic is possible connecting, through the RS232 cable, MCP module to the PC on which is installed the utility MCPTOOLS. Selecting the menu items "Supervisor" "Show Maps", it is possible to display on the screen the module involved in the program uploaded to MCP, allowing the search for possible fault modules (in this case the module is displayed on the screen in red color). For more details, refer to the MCPTOOLS utility program.

If a BUS FAILURE occurs, the bus connections have to be verified. This failure appears when MCP is not able to transmit its signal on the bus (L1 and L2). Due to the several causes of this anomaly, contact DUEMMEGI offices if the connection check did not give results. **Technical cards to guide the operator in the failure searching are available under request.**

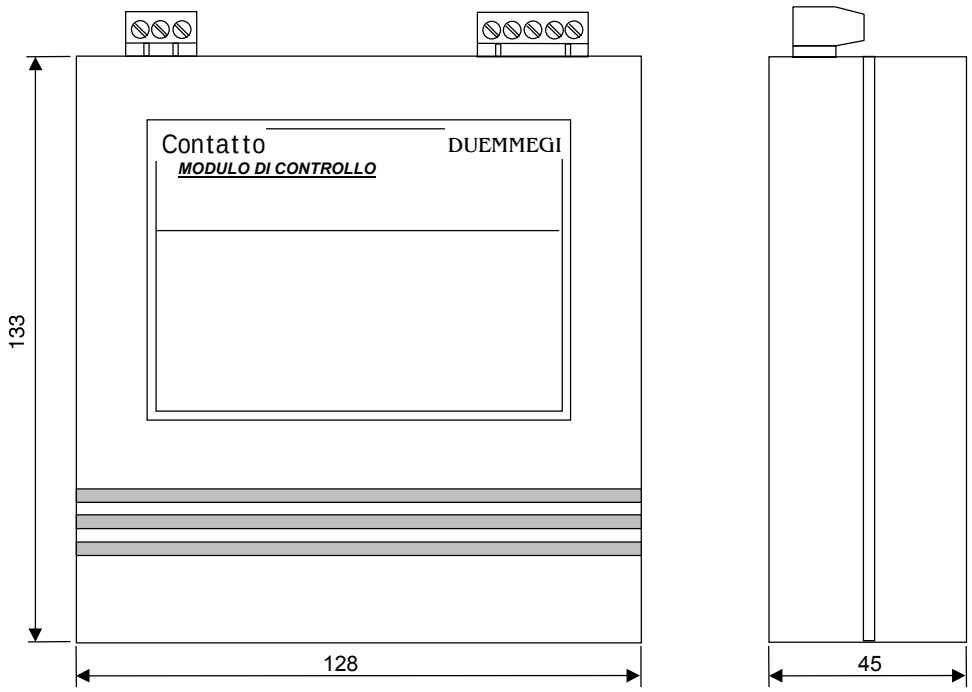
Two green leds on MCP panel signal bus activity: the **POLL** led shows the start of the polling cycle and it flashes at a frequency inversely proportional to the number of configured modules. If the MCP memory is not properly programmed, this led will be continuously lighted.
The **VAR** led shows the occurrence of a status change of one or more input modules. If the VAR led is continuously lighted or it has frequent and long flashes together MOD.F led (even if no status changes occur on any input modules), then two or more modules of the same type (IN or OUT) have the same address; in this case use the menu items "Supervisor" "Show Maps" of MCPTOOLS utility to find the doubled addresses (in this case the doubled modules are displayed on the screen in yellow color).

6- TECHNICAL CHARACTERISTICS

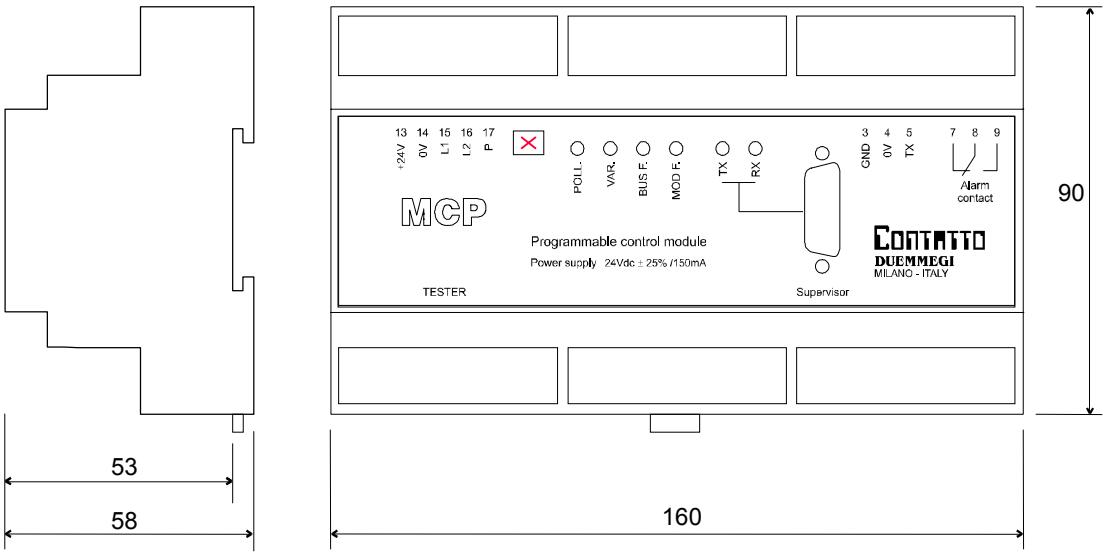
Power supply voltage	24Vdc ± 25%
Max current consumption	150mA
Alarm contact rating	5A@ 250Vac AC1
Number of employed processors	1
Typical input to output reaction time	25msec
User program memory size	FLASH type 128 Kbytes
RAM Memory size	32 Kbytes
Allowable virtual points	1000
Allowable timers	256 with times 0 to 6553 seconds, resolution 0.1 sec.
Allowable counters	512 (8 bit sized) or 256 (16 bit sized)
Programming clock	Weekly (up to 1016 independent points)
Digital input points	1016
Digital output points	1016
Analog input points	Up to 127 (each analog point takes get 8 digital points)
Analog output points	Up to 127 (each analog point takes 8 digital points)
Available serial ports	MCP and MCP/MOD: RS232 (not opto-coupled) MCP Plus: RS232 and RS485 (opto coupled)
Peripheral devices handling	- Modem - Touch screen video terminals - Serial display with alarm handling (not for MCP Plus) - Binary display with alarm handling - Supervision systems on PC
Interfacing to other systems	MODBUS: - MCP Plus: integrated MODBUS protocol - MCP and MCP/MOD: by external external protocol converter, part type **DUEMMEGI CDP** JOHNSON CONTROL - MCP Plus: integrated JOHNSON VND protocol

7- OUTLINE DIMENSIONS

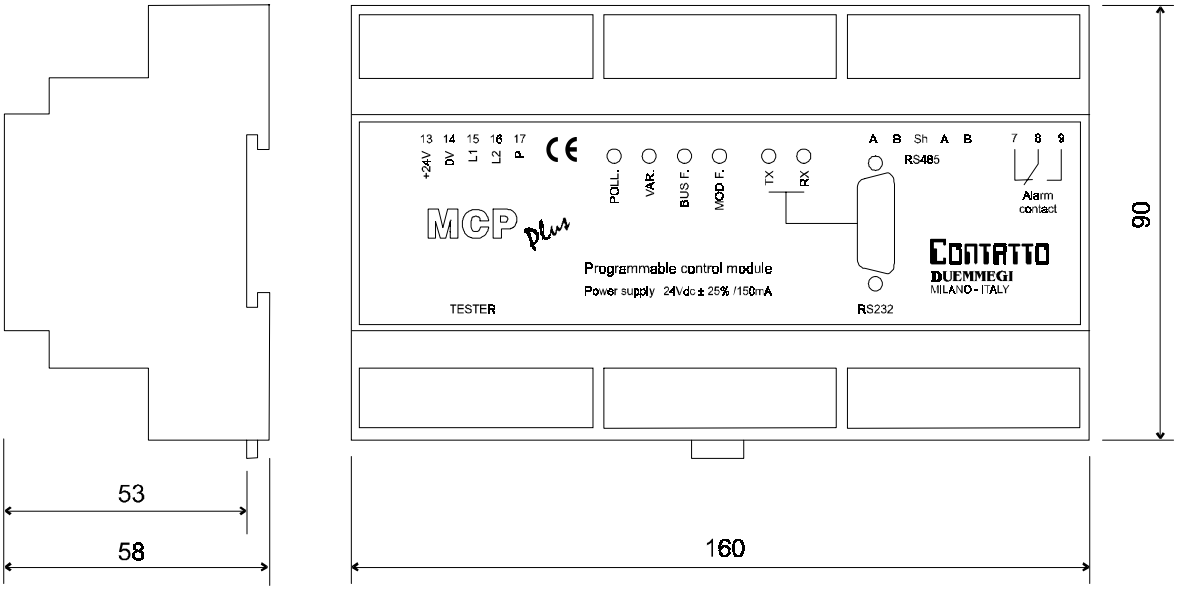
7.1- MCP Dimensions



7.2- MCP/MOD Dimensions



7.3- MCP Plus Dimensions



8- APPENDIX A: FXP SERIAL COMMUNICATION PROTOCOL

8.1- Message format and meanings of FXP protocol

The proprietary protocol implemented into MCP is named FXP protocol; this protocol, has been specifically developed to interface MCP to external world (PC, PLC, etc.) and it is is **NRZ with 1 start bit, 8 data bit, no parity, 1 stop bit**. The baud rate can be selected (at reset) through the jumpers inside MCP as follows: 2400, 4800, 9600, 19200 baud. **MCP acts as a slave unit**, then it answers to the requests of a HOST Computer.
In the following, the numerical data represented with the **0x** notation are intended to be in the hexadecimal format.

The messages between MCP and HOST have the following format:

Address	Code	# Bytes	Data 1	.....	Data N	ChkSum H	ChkSum L

Where:

- **Address**: 1 byte, address of MCP node; the address 00H is valid for any node addresses
- **Code**: 1 byte, message identification
- **# Bytes**: 1 byte, amount of data bytes
- **Data 1 ÷ N** N data bytes
- **ChkSum**: 2 bytes (high, low), equal to the complemented sum of the message bytes, including the address, the code and the number of bytes

The available messages are:

HOST to MCP requests:

Code	# Bytes	Data bytes	Function
0x13	0x03 ÷ 0xFF	First 3 bytes define the starting address. The remaining data bytes (#Byte - 3) are the data to be written in RAM. The address is: 0x00 0xXX 0xXX: external RAM 0x01 0xXX 0xXX: internal RAM (registers)	RAM writing starting from the address specified by first 3 bytes.
0x14	0x04	Starting address (3 bytes), amount of bytes to read (1 byte) (0x00 = 256 byte). 0x00 0xXX 0xXX: external RAM 0x01 0xXX 0xXX: internal RAM (registers)	RAM reading starting from the address specified by first 3 bytes. The amount of bytes is specified by 4th byte.
0x15	0x01	Module address MSB=0 ⇒ input module MSB=1 ⇒ output module	Status request. This function is useful for diagnostic purposes.
0x16	0x03	Address, new status, mask	Real output command. The mask identifies what outputs have to be modified.
0x17	0x02	Address, mask	Real outputs in manual mode. The mask (bit set to 1) identifies what outputs have to be set to manual mode (relevant equations will not be evaluated)
0x18	0x02	Address, mask	Real outputs in automatic mode. The mask (bit set to 1) identifies what outputs have to be set to auto mode (relevant equations will be evaluated). The evaluation of relevant equations is forced after this command.

0x19	0x02	Bit 15: Status Bit 9÷0: Number of virtual (0-1023)	Virtual points writing. This command force the evaluation of relevant output equations depending on selected virtual point.
0x1A	0x02	'I', 'D'	Identification code request.
0x1B	0x00 to 0xFF	ASCII characters.	Copy on serial DISPLAY/PRINTER port.
0x1C	0x06	Hours, Minutes, Week day, Date, Month, Year (BCD).	Clock writing.
0x1D	0x00		Clock reading.

MCP recognizes the following messages from the **MODEM**:

- *OK,*
- *CONNECT XXXX*
- *BUSY*
- *RING*
- *NO CARRIER, NO DIALTONE, NO ANSWER*

preceded and followed by 0x0D-0x0A (CR LF)

MCP to HOST answers:

Code	# Bytes	Data bytes	Function
0x13	0x01	0x00: valid writing. 0xFF: writing error.	Answer to the RAM writing command.
0x14	0xXX	XX bytes read from RAM starting from the address specified by the request. XX number is also specified in the request (XX = 0x00 = 256 bytes).	Answer to the RAM reading command.
0x15	0x03	Flag, Module address, Status. Flag.0 = 1: module not previewed. Flag.1 = 1: module does not answer.	Answer to the status request.
0x16	0x00	None	Answer to the real outputs command
0x17	0x00		ACK to manual mode setting.
0x18	0x00		ACK to automatic mode setting.
0x19	0x00		ACK to virtual points setting.
0x1A	0x40	64 ASCII characters.	Answer to ID code request.
0x1B	0x00		ACK to the serial port writing.
0x1C	0x00		ACK to the clock writing.
0x1D	0x06	Hours, Minutes, Week day, Date, Month, Year (BCD).	Answer to the clock reading.

8.2- RAM Memory Mapping

The following table shows the RAM mapping of MCP module for the commonly used parameters.

8.2.1- External RAM Memory Mapping

Address (Hex)	Description	Comments
0001-007F	Map of input modules status	127 input modules
0081-00FF	Map of output modules status	127 output modules
0101-017D	Map of current status of virtual points	1000 virtual points (digital only)
0181-01FF	Control mode for real outputs	127 output modules If bit = 0 (automatic)→ output controlled by the equation If bit = 1 (manual)→ output controlled by RS232 port (supervisor)
0201-027D	Map of new status of virtual points	1000 virtual points (digital only)
0289	Year in BCD format	Read from internal timekeeper chip. **Note 4**
028A	Day of week in BCD format	Read from internal timekeeper chip; 0=Sunday….6=Saturday. **Note 4**
028B	Month in BCD format	Read from internal timekeeper chip. **Note 4**
028C	Day of month in BCD format	Read from internal timekeeper chip. **Note 4**
028D	Hours in BCD format	Read from internal timekeeper chip. **Note 4**
028E	Minutes in BCD format	Read from internal timekeeper chip. **Note 4**
028F	Seconds in BCD format	Read from internal timekeeper chip. **Note 4**
0300-04FF	Modem alarms	512 bytes reporting the alarms that generated the modem call. **Note 3**
0500-06FF	Counter registers	512 8-bit counters / 256 16-bit counters
0700-077F	Refresh queue for serial device	If bit = 1 then the message related to the virtual point must cycled. **Note 1**
0780-07BF	Changed counters	32 bytes = 512 bits. If bit = 1 the counter contents has changed
1000-13FF	Timers	1024 bytes for 256 timers. **Note 2**
1408-17FF	Map of external 16-bit counter modules	1016 bytes for 127 4-channel 16-bit modules. **Note 5**
1808-1BFF	Map of 16-bit analog input modules	1016 bytes for 127 4-channel 16-bit modules. **Note 5**
2000-2FFF	Event list in chronological order	4096 bytes to store 1023 events together date and time. **Note 3**
3000-3080	Reset of counter modules	

Note 1: When an alarm, generating a message on the serial display, has activated, it will be included in the refresh queue (and related bit set to 1). It will be removed either when de-activated (if set as NOMEM), or when a RESET is performed (if set as MEM). Every 2 seconds, MCP sends to serial display the message to be visualized according to the queue. The refresh will not be executed in chronological order.
Note 2: Each timer takes 4 bytes in the RAM memory: 2 bytes for the timer, 11 bits for the controlled output, 5 flags.
Note 3: When an alarm generating a modem call or an event storing occurs, 4 bytes are included in the list to report the virtual point that generated the event, together to date and time. In this way it is possible to store in chronological order 127 modem alarms or 1023 events. The list has to be reset by the supervisor.
Note 4: MCP Plus only. Cells from 0289 to 028F contain the status of MCP internal timekeeper chip; in addition to reading, these cells may be written and in this case the timekeeper chip will be updated with passed new parameters. This feature allows reading and writing of the timekeeper by RAM reading and writing functions instead of using specific commands; in this way it is possible the reading and setting of the timekeeper through MODBUS protocol. Every times one of these parameters is changed, seconds will be reset to zero.
Note 5: When the register is 16-bit format, the cell having lower address contains the most significant byte and the cell having higher address contains the less significant byte of register itself.

8.2.2- Internal RAM Memory mapping

Address (Hex)	Description	Comments
013F	Higher address	Higher address among the modules included in the loaded program (for FXPRO)
0140-014F	Input modules configuration	128 flags reporting input modules included in the loaded program
0150-015F	Output modules configuration	128 flags reporting output modules included in the loaded program
0160-016F	Fault input modules	128 flags reporting fault input modules included in the loaded program
0170-017F	Fault output modules	128 flags reporting fault output modules included in the loaded program
0180-018F	Doubled input modules	128 flags reporting doubled input modules (with the same address)
0190-019F	Doubled output modules	128 flags reporting doubled output modules (with the same address)
01C0-01CF	16-bit external counter modules configuration	128 flags reporting 16-bit counter modules included in the loaded program
01D0-01DF	16-bit analog input modules configuration	128 flag reporting 16-bit analog input modules included in the loaded program

8.3- Suggestions for the implementation of a communication driver

Preliminary note: in this paragraph the x%8 notation means x module 8 and it is equivalent to the remainder of the division between x and 8; the INT(x) notation means the integer part of x.

To compute x module 8 proceed according to the following steps:

1. Divide x by 8
2. Subtract to result of step 1 the integer part of the result itself
3. Multiply by 8 the result of step 2: the resulting value is module 8 of the starting number; this result is always an integer number in the range 0 to 7

Example to compute 43%8:

1. 43 : 8 = 5.375
2. 5.375 – 5 = 0.375
3. 0.375 x 8 = 3

Input-output status request

The input and output status request can be performed in two ways:

1. Through the message code 0x015: in this way, in addition to the status, MCP answers with a flag byte containing the information regarding the configuration and the functionality of the module.
2. Through the RAM reading request (code 0x14): MCP answers with the module status only. This message can be used to read in a single block the status of whole modules. The map of input module status is in the range 1 to 127 (0x01 to 0x7F), that of output modules is in the range 129 to 255 (0x81 to 0xFF). At address **x** is stored the status of the input module having address **x**, at address **x+128** is stored the status of output module **x**.

Virtual points status request

The virtual points status request can be performed through the RAM reading message (code 0x14). The virtual points status map is in the address range 257 to 381 included (0x101 to 0x17D). The virtual point **x** is the bit **(x - 1)%8** of the RAM location:

$$257 + \mathrm{INT}\left(\frac{x-1}{8}\right)$$

Note that RAM addresses for reading of virtual points are not the same addresses for writing.
Appendix C reports some tables to easily locate the RAM address and bit related to each virtual point.

Output writing

The output can be forced in two ways:

1. Through the message code 0x16: in this case MCP immediately update the selected output status.
2. Through the RAM writing command (code 0x13) at the address identified by the chosen module (129 to 255, that is 0x81 to 0xFF). MCP will update the output through the usual polling cycle (the max delay from the command to the output updating will be equal to the polling cycle time). Output module **x** is at RAM address **x+128**.

Output control

The outputs may get the status derived from the evaluation of relevant equation or instead they can be locked to a given status by a supervisor. The control of the output mode can be done through 0x17 and 0x18 commands, or directly writing in RAM memory the control bits for each output. These bits are located in RAM starting from the address 385 up to address 511 (0x181 to 0x1FF). If the bit is reset to 0, then the output status will be controlled by the related equation, else if the bit is set to 1 the output status will be controlled only by PC (or PLC, etc.). Output module **x** is controlled by RAM cell at address **x+384**.

Virtual points writing

The virtual points may be written in two ways:

1. Through the message code 0x019.

Through the RAM writing command (code 0x13) to related address (513 to 637, that is 0x201 to 0x27D). Virtual point **x** is the bit **(x - 1)%8** of the RAM location:

$$513 + \mathrm{INT}\left(\frac{x-1}{8}\right)$$

Note that RAM addresses for reading of virtual points are not the same addresses for writing.
Appendix C reports some tables to easily locate the RAM address and bit related to each virtual point.

MCP soft-reset

Writing in RAM memory, at addresses 640 to 647 (0x280 to 0x287), 8 null bytes (0x00), MCP will be reset (re-initialized).

Internal Counter registers

The counters are mapped in RAM memory starting from address 1280 (0x500) with offset depending on the number of considered counter (see Chapter 2). The reading and the writing of the counters has to be performed through the RAM reading and writing at the related address. Counter Cn is at RAM location:
(n + 1280).
If a counter is 16-bit type, the addressed cell contains the most significant byte and next cell contains the less significant byte of the counter.

External 4-channel/16-bit Counter Modules

External 4-channel/16-bit counter modules are mapped in RAM starting from address 5128 (0x1408). Reading and writing has to be performed through the RAM reading and writing at the related addresses. Channel Ch (in the range 1 to 4) of external counter module having address Addr is at RAM location:
(Addr - 1) x 8 + 2 x (Ch - 1) + 5128.
The addressed cell contains the most significant byte and next cell contains the less significant byte of the counter.

4-channel 16-bit or 12-bit Analog Input Modules

4-channel 16-bit or 12-bit analog input modules are mapped in RAM starting from address 6152 (0x1808). Reading and writing has to be performed through the RAM reading and writing at the related addresses. Channel Ch (in the range 1 to 4) of module having address Addr is at RAM location:
(Addr - 1) x 8 + 2 x (Ch - 1) + 6152.
The addressed cell contains the most significant byte and next cell contains the less significant byte of the register.

Events

The events are stored in RAM memory starting from address 8192 (0x2000). The total amount of stored events is placed in two bytes at addresses 8192 (MSByte) e 8193 (LSByte) (0x2000 and 0x2001). Information is grouped in 1023 blocks maximum of 4 bytes, and each one of these contains:

-	Address of virtual point (7 bit)		
Month (4 bit)		Status	Point p (3 bit)
Day (5 bit)			Hour MSB (3 bit)
Hour LSB (2)	Minutes (6 bit)		

The virtual point number related to the alarm is given by **(n-1)*8 + (p+1).** *Status* is the virtual point status, *Month, Day, Hour* and *minutes* report the occurrence of the event.
The queue is stored in chronological order and can contain up to 1023 blocks; further events will be lost. To clear the list, the supervisor must reset the amount of occurred events; this number is will be found, as said above, at address 8192 and 8193 (0x2000 ÷ 0x2001).

MODEM alarms

The pending alarms which caused the modem call are mapped in RAM memory starting from address 1924 (0x784). The byte at address 1920 (0x780) contains the amount of stored events. Information is grouped in 31 blocks maximum of 4 bytes, and each one of these contains:

-	Address of virtual point n (7 bits)		
Month (4 bits)		Status	Point p (3 bits)
Day (5 bits)			Hour MSB (3 bits)
Hour LSB (2)	Minutes (6 bits)		

The virtual point number related to the alarm is given by **(n-1)*8 + (p+1).** *Status* is the virtual point status, *Month, day, Hour* and *Minutes* report the occurrence of the event.

The queue is stored in chronological order and can contain up to 31 blocks; further alarms will be lost. The supervisor must reset the alarm amount byte located at address 1920, in order to re-enable MCP to record other events.
Last occurred event may be read in the **RAM inside the microcontroller** at address 138 (0x8A), using the format previously described.
The acknowledgment of the call must be performed through the request of the identification code, otherwise that MCP module will call again.

System configuration

The input module configuration is mapped in the **RAM inside the microcontroller** starting from address 320 (0x140) for 16 consecutive bytes. The configuration of module **x** will be the bit **x%8** of the location:

$$320 + \mathrm{INT}\left(\frac{x}{8}\right)$$

If the bit value is "1", then the module is involved in the loaded program. In a similar way, the output module configuration is mapped in the **RAM inside the microcontroller** starting from address 336 (0x150).

Fault modules

Fault modules are stored in the **RAM inside the microcontroller** in an analogous way as described for the system configuration, starting from address 352 (0x160) for input modules and 368 (0x170) for output modules.

Doubled modules

Doubled modules are stored in the **RAM inside the microcontroller** in an analogous way as described for the system configuration, starting from address 384 (0x180) for input modules and 400 (0x190) for output modules.

9- APPENDIX B: MODBUS COMMUNICATION PROTOCOL

9.1- General characteristics

MCP Plus can interface to external world through MODBUS RTU and JOHNSON CONTROL VND protocols. These protocols are integrated in MCP Plus and coexist, if enabled by the PROTOCOL directive (see related paragraph), together to the FXP proprietary protocol as described prevoiusly; that means:

- MCP Plus will answer according to the MODBUS protocol, if enabled, to any MODBUS requests
- MCP Plus will answer according to the JOHNSON protocol, if enabled, to any JOHNSON requests
- MCP Plus will answer according to the proprietary FXP protocol to any FXP requests

This Appendix will describe some traces about using of MODBUS protocol; for JOHNSON protocol, refer to specific documentation.
The communication parameters for MODBUS protocol implemented into MCP Plus are the followings:

- **1 start bit**
- **8 data bits**
- **no parity**
- **1 or 2 stop bits (automatic detection)**

The baud rate may be set as described in paragraph 4.3 by the micro switches inside MCP Plus to the following values: 2400, 4800, 9600, 19200 baud. **MCP Plus acts as slave (it is a MODBUS peripheral unit)**; this means that it answers to the requests of a MASTER MODBUS DEVICE.
In a MODBUS networks each peripheral device must its own address (station address); the address of MCP Plus has to be set by the ADDRESS function as described in paragraph 2.3.12.
To localize the input and output points (real and/or virtual), registers, etc., refer to the external RAM memory mapping at paragraph 8.2.1.

9.2- Supported MODBUS functions

MCP Plus supports the following MODBUS functions:

Function code	Description
1	Read output table
2	Read input table
3	Read registers (RAM memory)
4	Read analog input
5	Force single digital output point
6	Preset single register
15	Force multiple outputs
16	Preset multiple registers
17	Report device type

9.3- MODBUS function examples

This paragraph describes some examples of frequently used MODBUS functions (request and answer); remember that MCP Plus, in a MODBUS network, is a SLAVE peripheral, that means that it answers to the requests of a MASTER MODBUS DEVICE.

Following examples suggest some MODBUS functions to be used to get and send information to MCP Plus; the MODBUS driver implemented in most common devices (PLC, supervision software for PC, video-terminals, etc.), provides a development platform and a user interface that make easy to use the setting up in respect of that will be described in the following paragraphs. In other words, the setting up of the communication between the MASTER device and MCP Plus will be reduced to the setting up of the communication driver furnished by the manufacturer of the MODBUS device; for these reasons, please refer also to the user's manual of the system.
Following notations, unless otherwise specified, are intended to be in decimal format.

9.3.1- Function 1: Outputs reading

MODBUS function 1 allows to read the output status; it is needed to specify:

- **A Start point**; this value **must be multiple of 8**. Said **i** the address of Contatto module starting from which the outputs have to be read, the value of Start will be: **(i - 1) x 8**. Allowed values: **0 to 1008**; note that this number is not the "physical" address of the output module.
- **How many output points have to be read (Number)**; in other words, how many modules having consecutive address have to be read. To avoid mistakes, this value may be **multiple of 8** and equal to the number of module to be read multiplied by 8. Allowed values: **8 to 1016**.

MCP Plus will return a number of bytes equal to the Number divided by 8.

Example:
Reading of the outputs of module 25, e.g. a MOD8R which has 8 output points. The parameters to be passed to MODBUS driver will be the followings:

Start: 192
Number: 8

MCP Plus will return 1 byte containing the status of output points of module 25, coded according to binary code (1=output ON, 0=output OFF). Less significant bit is related to output point 1, most significant bit is related to output point 8.

9.3.2- Function 2: Inputs reading

MODBUS function 2 allows to read the input status; it is needed to specify:

- **A Start point**; **A Start point**; this value **must be multiple of 8**. Said **i** the address of Contatto module starting from which the inputs have to be read, the value of Start will be: **(i - 1) x 8**. Allowed values: **0 to 1008**; note that this number is not the "physical" address of the input module.
- **How many input points have to be read (Number)**; in other words, how many modules having consecutive address have to be read. To avoid mistakes, this value may be **multiple of 8** and equal to the number of module to be read multiplied by 8. Allowed values: **8 to 1016**.

MCP Plus will return a number of bytes equal to the Number divided by 8.

Example 1:
Reading of the inputs of module 43, e.g. a MOD8I/A which has 8 input points. The parameters to be passed to MODBUS driver will be the followings:

Start: 336
Number: 8

MCP Plus will return 1 byte containing the status of input points of module 43, coded according to binary code (1=input ON, 0=input OFF). Less significant bit is related to input point 1, most significant bit is related to input point 8.

Example 2:
Reading of the input points of modules 57, 58, 59 e 60, e.g. MOD8I/A which have 8 input points each one. The parameters to be passed to MODBUS driver will be the followings:

Start: **448**
Number: **32**

MCP Plus will return 4 bytes, each one containing the status of input points of modules from 57 (first byte) to 60 (last byte).

9.3.3- Function 3: Registers reading (RAM memory)

MODBUS function 3 **is the most frequently used**, because it allows the reading of MCP Plus RAM memory which contains all information about the system.

It is needed to specify:

- **A Start point**; this value is the **address of RAM location** starting from which the information has to be read. Allowed values: **0 to 12416** (hex: 0x0000 to 0x3080) for **external memory** and **33087 to 33247** (hex: 0x813F to 0x81DF) for **microcontroller internal memory** (see paragraph 8.2)
- **How many registers have to be read (Number)**; in other words, how many consecutive RAM cells have to be read. Allowed values: **1 to 125**.

MCP Plus will return a lot of WORDs equal to specified Number (that is **two times the specified Number**).

Each WORD in MCP answer will be made by:

- **The most significant byte equal to zero and the less significant byte containing the data of addressed cell for RAM locations from 0x0000 to 0x04FF (enclosed) and for microcontroller internal RAM**
- **The most significant byte containing the data of addressed cell and the less significant byte containing the data of next cell for RAM locations from 0x0500 to 0x307F (enclosed)**

MODBUS function 3 may be used to read input and output modules status, virtual points status, counter registers, etc.; any data in RAM memory mapping may be read, (see paragraph 8.2), also current time and date of internal timekeeper chip.

Example 1:
Reading of the outputs of module 25, e.g. a MOD8R; alternatively to function 1, it is possible to use function 3. The address of RAM cell containing the status of output module **i** is **i+128**, then for module 25, the parameters to be passed to MODBUS driver will be the followings:

Start: **153**
Number: **1**

MCP Plus will return one WORD where the most significant byte is zero and less significant byte is the binary code of output points of module 25 (1=output ON, 0=output OFF). Less significant bit is related to output point 1, most significant bit is related to output point 8.

Example 2:
Reading of the inputs of module 43, e.g. a MOD8I/A; alternatively to function 2, it is possible to use function 3. The address of RAM cell containing the status of input module **i** is equal to **i** itself, then for module 43, the parameters to be passed to MODBUS driver will be the followings:

Start: **43**
Number: **1**

MCP Plus will return 1 byte containing the status of input points of module 43, coded according to binary code (1=input ON, 0=input OFF). Less significant bit is related to input point 1, most significant bit is related to input point 8.

Example 3:
Reading of the input points of modules 57, 58, 59 e 60, e.g. MOD8I/A, using MODBUS function 3. The parameters to be passed to MODBUS driver will be the followings:

Start: **57**
Number: **4**

MCP Plus will return 4 bytes, each one containing the status of input points of modules from 57 (first byte) to 60 (last byte).

Example 4:
Reading of **virtual point** V328 using MODBUS function 3. Virtual point **Vx** is located at RAM address:

$$257 + \mathrm{INT}\left(\frac{x-1}{8}\right)$$

Because a virtual point takes only one bit, it is needed to specify which is that related to our point; the bit number is given by:

$$(x - 1)\%8$$

(for details on the symbols just used, refer to paragraph 8.3).

Alternatively, Appendix C reports some tables to easily locate the RAM address and bit related to each virtual point.
Note that RAM addresses for reading of virtual points are not the same addresses for writing.

The parameters to be passed to MODBUS driver, for virtual point V328, will be the followings:

Start: **297**
Number: **1**
Bit: **7**

Note: the bit number is always in the range 0 to 7.

MCP Plus will return one WORD where the most significant byte is zero and less significant byte contains the status of virtual points from V321 (less significant bit) to V328 (most significant bit). The virtual points are binary coded (1=ON, 0=OFF).

Example 5:
Reading of the **address of possible fault input modules** using MODBUS function 3. The table of fault input modules is mapped in the microcontroller internal RAM as described in paragraph 8.2.2; to access the internal RAM, the decimal value 32768 (hex: 0x8000) has to be summed to the addresses reported in the a.m. table. To read whole map of fault input modules, the parameters to be passed to MODBUS driver may be the followings:

Start: **33120**
Number: **16**

MCP Plus will return 16 WORDs (32 bytes), each one having the most significant byte equal to zero and the less significant byte containing the input modules condition according to the following rules:

- less significant bit of the 1st WORD (bit 0) is the condition of input module 0 (which does not exist, then this bit may be discarded)
- bit 1 of the 1st WORD is the condition of input module 1
- .
- .
- bit 7 of the 1st WORD is the condition of input module 7
- bit 0 of the 2nd WORD of the first WORD is the condition of input module 8
- .
- .
- bit 7 of the 16th WORD is the condition of input module 127

A bit is set to "1" means that related module has fault (or not connected to the bus).

This example may be applied to all information related to the condition or configuration of the modules, simply changing the address according to the map reported at paragraph 8.2.2.

Example 6:
Reading of **input value of channel 1** of 12-bit analog input module having address 8, e.g. a MOD4AM12; Channel **Ch** (in the range 1 to 4) of module having address **Addr** is at RAM location:
(Addr - 1) x 8 + 2 x (Ch - 1) + 6152

This means that for channel 1 of module 8, the parameters to be passed to MODBUS driver may be the followings:

Start: **6208**
Number: **1**

MCP Plus will return one WORD (16 bit) containing the required alue.

9.3.4- Function 5: Force single digital output point

MODBUS function 5 allows to force the status of a single real output point; it is needed to specify:

- **the Number of the real output point to be forced**; said **i** the address of Contatto real module and said **p** the output point to be changed, then Number will be set to **[(i –1) x 8 + p – 1]**. Allowed values for **i** are 1 to 127, while for **p** these values are 1 to 8 .
- **new output point status** (1=ON, 0=OFF).

Example:
Switch ON point 3 of output module 29; the parameters to be passed to MODBUS driver may be the followings:

Number: **226**
Status: **1**

9.3.5- Function 16: Multiple registers writing (Ram memory)

MODBUS function 16 allows the writing into external RAM memory of MCP Plus (which contains all information about the system). This function, together to function 3, is the much frequently used.
It is needed to specify:

- **A Start point**; this value is the **address of RAM location** starting from which the new value has to be written. Allowed values: **0 to 12416** (hex: 0x0000 to 0x3080) for **external memory**. The microcontroller internal memory MUST NOT BE MODIFIED
- **How many registers have to be written (Number)**; in other words, how many consecutive RAM cells have to be written. Allowed values: **1 to 125**.
- **The values to be written (Data)** into specified cells; each data (how many as specified by Number) must be made by two bytes (one WORD), where the most significant byte is equal to zero and the less significant byte contains the wanted value.

MODBUS drivers allow to force one or more WORD to a new value (useful to change a counter value) or to change only one bit (useful to force a single real output point or virtual point).

MODBUS function 16 may be used to change the status of a whole output (digital or analog) module, the status of a single output point of a module, the status of virtual points, counters, etc.

To change a single bit in a register using function 16, the status of other bits in the same register has to be taken in account because the writing occur on whole WORD; the actual MODBUS drivers, to change a single bit in a WORD, automatically execute the following steps:

1. Reading by function 3 of the WORD containing the bit to be changed
2. Writing by function 16 of the just read WORD, but with the wanted bit changed

MCP will automatically handle the WORD received from MODBUS MASTER:

- **Forcing the most significant byte to zero for RAM locations 0x0000 to 0x04FF (enclosed) and for all microcontroller internal RAM**
- **Leaving the WORD as received from MODBUS MASTER) for RAM locations 0x0500 to 0x307F (enclosed); in this case the most significant byte of the WORD will be written to addressed location, while the less significant byte will be written to the next location**

MODBUS function 16 may be used to set date and time of the timekeeper chip of MCP Plus as described by one of the following examples.

Example 1:
Switch ON point 3 of output module 29. Alternatively to function 5, it is possible to use the function 16. The address of RAM cell containing the status of output module **i** is **i+128**, then for module 29, the parameters to be passed to MODBUS driver will be the followings:

Start: **157**
Number: **1** (normally, in this case, this parameters will not be required by the driver)
Bit: **2**
Value: **1** (or ON, depend on the driver)

Note: point 3 of an output module is bit 2 of the WORD, because the real output points of Contatto system are numbered from 1 to 8, while the MODBUS driver works on bits from 0 to 7.

The execution of this function will be carried out, as before described, according to a specific procedure: MODBUS driver reads RAM cell 157 by function 3, then it changes bit 2 to read value and finally it transfers the new value to RAM cell 157 by function 16. The MODBUS driver normally executes this procedure automatically.

Example 2:
Switch ON all output points of module 29 using function 16. The address of RAM cell containing the status of output module **i** is **i+128**, then for module 29, the parameters to be passed to MODBUS driver will be the followings:

Start: **157**
Number: **1**
Value: **255**

In this case the value 255 will be directly written to RAM cell 157. In addition, the MODBUS drivers allow to execute some logic and mathematical functions between the cell content and a fixed value (e.g. an EXOR between the current value of an output module and the fixed value 255 to invert the status of all output points of that module) and then to write the result to the same cell.

Example 3:
Activation of virtual point V751 by function. The virtual point **Vx** is at Ram location::

$$513 + \mathrm{INT}\left(\frac{x-1}{8}\right)$$

Because a virtual point takes only one bit, it is needed to specify which is that related to our point; the bit number is given by:

(x - 1)%8

(for details on the symbols just used, refer to paragraph 8.3).

Alternatively, Appendix C reports some tables to easily locate the RAM address and bit related to each virtual point.
Note that RAM addresses for reading of virtual points are not the same addresses for writing.

Virtual point V751 is bit 6 of RAM cell 606; the parameters to be passed to MODBUS driver will be the followings:

Start: **606**
Number: **1** (normally, in this case, this parameters will not be required by the driver)
Bit: **6**
Value: **1** (or ON, depend on the driver)

The execution of this function will be carried out, as before described, according to a specific procedure: MODBUS driver reads RAM cell 606 by function 3, then it changes bit 6 to read value and finally it transfers the new value to RAM cell 606 by function 16. The MODBUS driver normally executes this procedure automatically.

Example 4:
Writing of value 157 to counter register C22 (remember that for Contatto system the counters are numbered from 0 to 512). Function 16 will be used. The RAM address of the cell containing the value of counter **Cn** is given by **1280+n**, then, regarding counter C22, the following parameters have to be passed to MODBUS driver:

Start: **1302**
Number: **1**
Value: **157**

In this case the value 157 will be directly written to RAM cell 1302.

Example 5:
Set the minutes of the timekeeper chip of MCP Plus to 36; the information related to time and date of MCP Plus are mapped as follows:

0x0289	Year in BCD format
0x028A	Day of week in BCD format
0x028B	Month in BCD format
0x028C	Day of month in BCD format
0x028D	Hours in BCD format
0x028E	Minutes in BCD format
0x028F	Second in BCD format

The parameters to be passed to MODBUS driver will be the following:

Start: 654
Number: 1
Value: 54

In this case the value 0x36 will be directly written to RAM cell 654 (0x028E). The timekeeper chip will be updated with the new value (remember that the seconds will be automatically reset to zero).

Note that the passed value is 54 (decimal), because the minutes register, as for all timekeeper parameters, requires the BCD format; in facts, 36 in BCD format is equivalent to 54 decimal.

10- APPENDIX C: VIRTUAL POINTS TABLES

10.1- Virtual points reading

	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D	010E	010F
Bit 0	-	V1	V9	V17	V25	V33	V41	V49	V57	V65	V73	V81	V89	V97	V105	V113
Bit 1	-	V2	V10	V18	V26	V34	V42	V50	V58	V66	V74	V82	V90	V98	V106	V114
Bit 2	-	V3	V11	V19	V27	V35	V43	V51	V59	V67	V75	V83	V91	V99	V107	V115
Bit 3	-	V4	V12	V20	V28	V36	V44	V52	V60	V68	V76	V84	V92	V100	V108	V116
Bit 4	-	V5	V13	V21	V29	V37	V45	V53	V61	V69	V77	V85	V93	V101	V109	V117
Bit 5	-	V6	V14	V22	V30	V38	V46	V54	V62	V70	V78	V86	V94	V102	V110	V118
Bit 6	-	V7	V15	V23	V31	V39	V47	V55	V63	V71	V79	V87	V95	V103	V111	V119
Bit 7	-	V8	V16	V24	V32	V40	V48	V56	V64	V72	V80	V88	V96	V104	V112	V120

	0110	0111	0112	0113	0114	0115	0116	0117	0118	0119	011A	011B	011C	011D	011E	011F
Bit 0	V121	V129	V137	V145	V153	V161	V169	V177	V185	V193	V201	V209	V217	V225	V233	V241
Bit 1	V122	V130	V138	V146	V154	V162	V170	V178	V186	V194	V202	V210	V218	V226	V234	V242
Bit 2	V123	V131	V139	V147	V155	V163	V171	V179	V187	V195	V203	V211	V219	V227	V235	V243
Bit 3	V124	V132	V140	V148	V156	V164	V172	V180	V188	V196	V204	V212	V220	V228	V236	V244
Bit 4	V125	V133	V141	V149	V157	V165	V173	V181	V189	V197	V205	V213	V221	V229	V237	V245
Bit 5	V126	V134	V142	V150	V158	V166	V174	V182	V190	V198	V206	V214	V222	V230	V238	V246
Bit 6	V127	V135	V143	V151	V159	V167	V175	V183	V191	V199	V202	V215	V223	V231	V239	V247
Bit 7	V128	V136	V144	V152	V160	V168	V176	V184	V192	V200	V208	V216	V224	V232	V240	V248

	0120	0121	0122	0123	0124	0125	0126	0127	0128	0129	012A	012B	012C	012D	012E	012F
Bit 0	V249	V257	V265	V273	V281	V289	V297	V305	V313	V321	V329	V337	V345	V353	V361	V369
Bit 1	V250	V258	V266	V274	V282	V290	V298	V306	V314	V322	V330	V338	V346	V354	V362	V370
Bit 2	V251	V259	V267	V275	V283	V291	V299	V307	V315	V323	V331	V339	V347	V355	V363	V371
Bit 3	V252	V260	V268	V276	V284	V292	V300	V308	V316	V324	V332	V340	V348	V356	V364	V372
Bit 4	V253	V261	V269	V277	V285	V293	V301	V309	V317	V325	V333	V341	V349	V357	V365	V373
Bit 5	V254	V262	V270	V278	V286	V294	V302	V310	V318	V326	V334	V342	V350	V358	V366	V374
Bit 6	V255	V263	V271	V279	V287	V295	V303	V311	V319	V327	V335	V343	V351	V359	V367	V375
Bit 7	V256	V264	V272	V280	V288	V296	V304	V312	V320	V328	V336	V344	V352	V360	V368	V376

	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	013A	013B	013C	013D	013E	013F
Bit 0	V377	V385	V393	V401	V409	V417	V425	V433	V441	V449	V457	V465	V473	V481	V489	V497
Bit 1	V378	V386	V394	V402	V410	V418	V426	V434	V442	V450	V458	V466	V474	V482	V490	V498
Bit 2	V379	V387	V395	V403	V411	V419	V427	V435	V443	V451	V459	V467	V475	V483	V491	V499
Bit 3	V380	V388	V396	V404	V412	V420	V428	V436	V444	V452	V460	V468	V476	V484	V492	V500
Bit 4	V381	V389	V397	V405	V413	V421	V429	V437	V445	V453	V461	V469	V477	V485	V493	V501
Bit 5	V382	V390	V398	V406	V414	V422	V430	V438	V446	V454	V462	V470	V478	V486	V494	V502
Bit 6	V383	V391	V399	V402	V415	V423	V431	V439	V447	V455	V463	V471	V479	V487	V495	V503
Bit 7	V384	V392	V400	V408	V416	V424	V432	V440	V448	V456	V464	V472	V480	V488	V496	V504

	0140	0141	0142	0143	0144	0145	0146	0147	0148	0149	014A	014B	014C	014D	014E	014F
Bit 0	V505	V513	V521	V529	V537	V545	V553	V561	V569	V577	V585	V593	V601	V609	V617	V625
Bit 1	V506	V514	V522	V530	V538	V546	V554	V562	V570	V578	V586	V594	V602	V610	V618	V626
Bit 2	V507	V515	V523	V531	V539	V547	V555	V563	V571	V579	V587	V595	V603	V611	V619	V627
Bit 3	V508	V516	V524	V532	V540	V548	V556	V564	V572	V580	V588	V596	V604	V612	V620	V628
Bit 4	V509	V517	V525	V533	V541	V549	V557	V565	V573	V581	V589	V597	V605	V613	V621	V629
Bit 5	V510	V518	V526	V534	V542	V550	V558	V566	V574	V582	V590	V598	V606	V614	V622	V630
Bit 6	V511	V519	V527	V535	V543	V551	V559	V567	V575	V583	V591	V599	V602	V615	V623	V631
Bit 7	V512	V520	V528	V536	V544	V552	V560	V568	V576	V584	V592	V600	V608	V616	V624	V632

	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159	015A	015B	015C	015D	015E	015F
Bit 0	V633	V641	V649	V657	V665	V673	V681	V689	V697	V705	V713	V721	V729	V737	V745	V753
Bit 1	V634	V642	V650	V658	V666	V674	V682	V690	V698	V706	V714	V722	V730	V738	V746	V754
Bit 2	V635	V643	V651	V659	V667	V675	V683	V691	V699	V707	V715	V723	V731	V739	V747	V755
Bit 3	V636	V644	V652	V660	V668	V676	V684	V692	V700	V708	V716	V724	V732	V740	V748	V756
Bit 4	V637	V645	V653	V661	V669	V677	V685	V693	V701	V709	V717	V725	V733	V741	V749	V757
Bit 5	V638	V646	V654	V662	V670	V678	V686	V694	V702	V710	V718	V726	V734	V742	V750	V758
Bit 6	V639	V647	V655	V663	V671	V679	V687	V695	V703	V711	V719	V727	V735	V743	V751	V759
Bit 7	V640	V648	V656	V664	V672	V680	V688	V696	V704	V712	V720	V728	V736	V744	V752	V760

	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	016A	016B	016C	016D	016E	016F
Bit 0	V761	V769	V777	V785	V793	V801	V809	V817	V825	V833	V841	V849	V857	V865	V873	V881
Bit 1	V762	V770	V778	V786	V794	V802	V810	V818	V826	V834	V842	V850	V858	V866	V874	V882
Bit 2	V763	V771	V779	V787	V795	V803	V811	V819	V827	V835	V843	V851	V859	V867	V875	V883
Bit 3	V764	V772	V780	V788	V796	V804	V812	V820	V828	V836	V844	V852	V860	V868	V876	V884
Bit 4	V765	V773	V781	V789	V797	V805	V813	V821	V829	V837	V845	V853	V861	V869	V877	V885
Bit 5	V766	V774	V782	V790	V798	V806	V814	V822	V830	V838	V846	V854	V862	V870	V878	V886
Bit 6	V767	V775	V783	V791	V799	V802	V815	V823	V831	V839	V847	V855	V863	V871	V879	V887
Bit 7	V768	V776	V784	V792	V800	V808	V816	V824	V832	V840	V848	V856	V864	V872	V880	V888

	0170	0171	0172	0173	0174	0175	0176	0177	0178	0179	017A	017B	017C	017D	017E	017F
Bit 0	V889	V897	V905	V913	V921	V929	V937	V945	V953	V961	V969	V977	V985	V993	-	-
Bit 1	V890	V898	V906	V914	V922	V930	V938	V946	V954	V962	V970	V978	V986	V994	-	-
Bit 2	V891	V899	V907	V915	V923	V931	V939	V947	V955	V963	V971	V979	V987	V995	-	-
Bit 3	V892	V900	V908	V916	V924	V932	V940	V948	V956	V964	V972	V980	V988	V996	-	-
Bit 4	V893	V901	V909	V917	V925	V933	V941	V949	V957	V965	V973	V981	V989	V997	-	-
Bit 5	V894	V902	V910	V918	V926	V934	V942	V950	V958	V966	V974	V982	V990	V998	-	-
Bit 6	V895	V903	V911	V919	V927	V935	V943	V951	V959	V967	V975	V983	V991	V999	-	-
Bit 7	V896	V904	V912	V920	V928	V936	V944	V952	V960	V968	V976	V984	V992	**V1000**	-	-

10.2- Virtual points writing

	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	020A	020B	020C	020D	020E	020F
Bit 0	-	V1	V9	V17	V25	V33	V41	V49	V57	V65	V73	V81	V89	V97	V105	V113
Bit 1	-	V2	V10	V18	V26	V34	V42	V50	V58	V66	V74	V82	V90	V98	V106	V114
Bit 2	-	V3	V11	V19	V27	V35	V43	V51	V59	V67	V75	V83	V91	V99	V107	V115
Bit 3	-	V4	V12	V20	V28	V36	V44	V52	V60	V68	V76	V84	V92	V100	V108	V116
Bit 4	-	V5	V13	V21	V29	V37	V45	V53	V61	V69	V77	V85	V93	V101	V109	V117
Bit 5	-	V6	V14	V22	V30	V38	V46	V54	V62	V70	V78	V86	V94	V102	V110	V118
Bit 6	-	V7	V15	V23	V31	V39	V47	V55	V63	V71	V79	V87	V95	V103	V111	V119
Bit 7	-	V8	V16	V24	V32	V40	V48	V56	V64	V72	V80	V88	V96	V104	V112	V120

	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	021A	021B	021C	021D	021E	021F
Bit 0	V121	V129	V137	V145	V153	V161	V169	V177	V185	V193	V201	V209	V217	V225	V233	V241
Bit 1	V122	V130	V138	V146	V154	V162	V170	V178	V186	V194	V202	V210	V218	V226	V234	V242
Bit 2	V123	V131	V139	V147	V155	V163	V171	V179	V187	V195	V203	V211	V219	V227	V235	V243
Bit 3	V124	V132	V140	V148	V156	V164	V172	V180	V188	V196	V204	V212	V220	V228	V236	V244
Bit 4	V125	V133	V141	V149	V157	V165	V173	V181	V189	V197	V205	V213	V221	V229	V237	V245
Bit 5	V126	V134	V142	V150	V158	V166	V174	V182	V190	V198	V206	V214	V222	V230	V238	V246
Bit 6	V127	V135	V143	V151	V159	V167	V175	V183	V191	V199	V202	V215	V223	V231	V239	V247
Bit 7	V128	V136	V144	V152	V160	V168	V176	V184	V192	V200	V208	V216	V224	V232	V240	V248

	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	022A	022B	022C	022D	022E	022F
Bit 0	V249	V257	V265	V273	V281	V289	V297	V305	V313	V321	V329	V337	V345	V353	V361	V369
Bit 1	V250	V258	V266	V274	V282	V290	V298	V306	V314	V322	V330	V338	V346	V354	V362	V370
Bit 2	V251	V259	V267	V275	V283	V291	V299	V307	V315	V323	V331	V339	V347	V355	V363	V371
Bit 3	V252	V260	V268	V276	V284	V292	V300	V308	V316	V324	V332	V340	V348	V356	V364	V372
Bit 4	V253	V261	V269	V277	V285	V293	V301	V309	V317	V325	V333	V341	V349	V357	V365	V373
Bit 5	V254	V262	V270	V278	V286	V294	V302	V310	V318	V326	V334	V342	V350	V358	V366	V374
Bit 6	V255	V263	V271	V279	V287	V295	V303	V311	V319	V327	V335	V343	V351	V359	V367	V375
Bit 7	V256	V264	V272	V280	V288	V296	V304	V312	V320	V328	V336	V344	V352	V360	V368	V376

	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239	023A	023B	023C	023D	023E	023F
Bit 0	V377	V385	V393	V401	V409	V417	V425	V433	V441	V449	V457	V465	V473	V481	V489	V497
Bit 1	V378	V386	V394	V402	V410	V418	V426	V434	V442	V450	V458	V466	V474	V482	V490	V498
Bit 2	V379	V387	V395	V403	V411	V419	V427	V435	V443	V451	V459	V467	V475	V483	V491	V499
Bit 3	V380	V388	V396	V404	V412	V420	V428	V436	V444	V452	V460	V468	V476	V484	V492	V500
Bit 4	V381	V389	V397	V405	V413	V421	V429	V437	V445	V453	V461	V469	V477	V485	V493	V501
Bit 5	V382	V390	V398	V406	V414	V422	V430	V438	V446	V454	V462	V470	V478	V486	V494	V502
Bit 6	V383	V391	V399	V402	V415	V423	V431	V439	V447	V455	V463	V471	V479	V487	V495	V503
Bit 7	V384	V392	V400	V408	V416	V424	V432	V440	V448	V456	V464	V472	V480	V488	V496	V504

	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	024A	024B	024C	024D	024E	024F
Bit 0	V505	V513	V521	V529	V537	V545	V553	V561	V569	V577	V585	V593	V601	V609	V617	V625
Bit 1	V506	V514	V522	V530	V538	V546	V554	V562	V570	V578	V586	V594	V602	V610	V618	V626
Bit 2	V507	V515	V523	V531	V539	V547	V555	V563	V571	V579	V587	V595	V603	V611	V619	V627
Bit 3	V508	V516	V524	V532	V540	V548	V556	V564	V572	V580	V588	V596	V604	V612	V620	V628
Bit 4	V509	V517	V525	V533	V541	V549	V557	V565	V573	V581	V589	V597	V605	V613	V621	V629
Bit 5	V510	V518	V526	V534	V542	V550	V558	V566	V574	V582	V590	V598	V606	V614	V622	V630
Bit 6	V511	V519	V527	V535	V543	V551	V559	V567	V575	V583	V591	V599	V602	V615	V623	V631
Bit 7	V512	V520	V528	V536	V544	V552	V560	V568	V576	V584	V592	V600	V608	V616	V624	V632

	0250	0251	0252	0253	0254	0255	0256	0257	0258	0259	025A	025B	025C	025D	025E	025F
Bit 0	V633	V641	V649	V657	V665	V673	V681	V689	V697	V705	V713	V721	V729	V737	V745	V753
Bit 1	V634	V642	V650	V658	V666	V674	V682	V690	V698	V706	V714	V722	V730	V738	V746	V754
Bit 2	V635	V643	V651	V659	V667	V675	V683	V691	V699	V707	V715	V723	V731	V739	V747	V755
Bit 3	V636	V644	V652	V660	V668	V676	V684	V692	V700	V708	V716	V724	V732	V740	V748	V756
Bit 4	V637	V645	V653	V661	V669	V677	V685	V693	V701	V709	V717	V725	V733	V741	V749	V757
Bit 5	V638	V646	V654	V662	V670	V678	V686	V694	V702	V710	V718	V726	V734	V742	V750	V758
Bit 6	V639	V647	V655	V663	V671	V679	V687	V695	V703	V711	V719	V727	V735	V743	V751	V759
Bit 7	V640	V648	V656	V664	V672	V680	V688	V696	V704	V712	V720	V728	V736	V744	V752	V760

	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	026A	026B	026C	026D	026E	026F
Bit 0	V761	V769	V777	V785	V793	V801	V809	V817	V825	V833	V841	V849	V857	V865	V873	V881
Bit 1	V762	V770	V778	V786	V794	V802	V810	V818	V826	V834	V842	V850	V858	V866	V874	V882
Bit 2	V763	V771	V779	V787	V795	V803	V811	V819	V827	V835	V843	V851	V859	V867	V875	V883
Bit 3	V764	V772	V780	V788	V796	V804	V812	V820	V828	V836	V844	V852	V860	V868	V876	V884
Bit 4	V765	V773	V781	V789	V797	V805	V813	V821	V829	V837	V845	V853	V861	V869	V877	V885
Bit 5	V766	V774	V782	V790	V798	V806	V814	V822	V830	V838	V846	V854	V862	V870	V878	V886
Bit 6	V767	V775	V783	V791	V799	V802	V815	V823	V831	V839	V847	V855	V863	V871	V879	V887
Bit 7	V768	V776	V784	V792	V800	V808	V816	V824	V832	V840	V848	V856	V864	V872	V880	V888

	0270	0271	0272	0273	0274	0275	0276	0277	0278	0279	027A	027B	027C	027D	027E	027F
Bit 0	V889	V897	V905	V913	V921	V929	V937	V945	V953	V961	V969	V977	V985	V993	-	-
Bit 1	V890	V898	V906	V914	V922	V930	V938	V946	V954	V962	V970	V978	V986	V994	-	-
Bit 2	V891	V899	V907	V915	V923	V931	V939	V947	V955	V963	V971	V979	V987	V995	-	-
Bit 3	V892	V900	V908	V916	V924	V932	V940	V948	V956	V964	V972	V980	V988	V996	-	-
Bit 4	V893	V901	V909	V917	V925	V933	V941	V949	V957	V965	V973	V981	V989	V997	-	-
Bit 5	V894	V902	V910	V918	V926	V934	V942	V950	V958	V966	V974	V982	V990	V998	-	-
Bit 6	V895	V903	V911	V919	V927	V935	V943	V951	V959	V967	V975	V983	V991	V999	-	-
Bit 7	V896	V904	V912	V920	V928	V936	V944	V952	V960	V968	V976	V984	V992	**V1000**	-	-